UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
DE CAMINOS, CANALES Y PUERTOS

PROGRAMACIÓN EN LENGUAJE
JAVA

Roberto Mínguez y Cristina Solares

CIUDAD REAL, 2006
EDITORIAL UCLM

Estos apuntes has sido realizados con LaTeX, y reproducidos a partir de originales suministrados por Cristina Solares. Nótese que la mayor parte del material ha sido extraída del libro *JAVA TM Un lenguaje de programación en Internet* escrito por E. Castillo, A. Cobo, P. Gómez, y C. Solares, adaptando los contenidos al programa propuesto de la asignatura.

Índice general

Índice de cuadros

Capítulo 1

Introducción

1.1. Internet

Los últimos años han constituido una auténtica explosión y revolución de los ordenadores. La aparición de la red Internet ha dado lugar a que millones de personas que nunca habían utilizado el ordenador se hayan lanzado a una utilización repentina y masiva, que fundamentalmente se basa en navegar electrónicamente a través de bases de datos, buscando información de todo tipo, material bibliográfico, comunicándose con amigos o grupos de amigos, asociándose a grupos de personas afines que se relacionan electrónicamente, etc.

Mediante la red se puede:

- **Acceder a todo tipo de información:** (prensa, bibliotecas, gráfica, etc.).
- **Traer todo tipo de software:** en forma de programas, juegos, aplicaciones, etc.
- **Capturar imágenes de todo tipo:** (fotografías, dibujos, etc.).
- **Traer sonidos:** en varios formatos.
- **Acceder a vídeos.**
- etc.

El uso de Internet tiene ventajas e inconvenientes. Entre las ventajas caben destacar las siguientes:

- **Capacidad ilimitada de almacenamiento:** ya que los servidores se encargan de almacenar por él.
- **Ahorro de espacio de almacenamiento:** sólo hace falta guardar una copia de cada fichero. Se pasa de una copia por usuario a una copia por servidor (unas pocas).
- **Información de los ficheros actualizada independientemente del usuario:** lo que da lugar a que éste pueda usar siempre versiones actualizadas.
- **Se libera al usuario del problema de las copias de seguridad:** limitándose éstos sólo a la seguridad de los ficheros propios.

- **Actualización mucho más fácilmente:** pues sólo hay que actualizar en los ficheros servidores. Además esta actualización puede hacerse por verdaderos especialistas, con lo que se evita al usuario muchos problemas.
- **Software compatible:** independiente de la plataforma, sea ésta Macintosh, PC compatible, UNIX, etc.

En cuanto a las desventajas, algunas son:

- **Necesidad de redes de comunicación muy potentes:** (en cuanto a capacidad y velocidad).
- **Alto coste de acceso a la información:** (modems, líneas telefónicas, pago de cuotas, etc.)
- **Problemas de seguridad:** Hacen falta métodos que garanticen la seguridad de nuestro sistema informático.
- **No existe de momento un lenguaje de programación general y específico para aplicaciones de Internet:** Hoy las aplicaciones se desarrollan en una serie de lenguajes no compatibles (C, C++, Pascal, ADA, etc.).
- **Aplicaciones existentes no son compatibles:** no son portables de un sistema a otro.
- **El desarrollo de aplicaciones verdaderamente interactivas en Internet no es fácil con los medios disponibles**.
- **El trabajo es fuertemente dependiente de los diferentes servidores y de las vías de comunicación**.

1.2. ¿Qué es Java?

Java es un **lenguaje de programación**:

- **Simple:** es casi idéntico al C y C++. Todos los programadores de estos dos lenguajes pueden convertirse de la noche a la mañana, si así lo desean, en programadores de Java.
- **Multiplataforma:** Un sueño de todo programador es escribir un programa en una plataforma determinada y que después funcione en todas. Java es un lenguaje multiplataforma, todo programa se puede ver bajo las plataformas: Windows, Macintosh, Unix, sin necesidad de hacer cambio alguno.
- Se eliminan los tres problemas más importantes de C y C++:

  1. **El uso de punteros:** en Java se han eliminado éstos, con lo que los problemas subyacentes han desaparecido.

2. **La gestión de memoria:** la liberación de memoria en Java se hace de modo automático.
3. **El control de acceso a una matriz:** el lenguaje Java lleva control del tamaño de la matriz y no permite salirse del mismo.

- **Orientado a objeto**, es decir, trabaja con objetos y con métodos que actúan sobre ellos. Las ventajas más importantes de la programación orientada al objeto son:

  1. Los programas son más fáciles de modificar
  2. Los programas son más fáciles de mantener
  3. El código utilizado es mucho más portable y reutilizable
  4. La estructura de los programas se entiende mejor

- **Multitarea**.
- **Prestaciones multimedia:** texto, gráficos, sonido, animaciones, etc.
- **Seguro**
- **Fácilmente accesible:** se puede acceder gratuítamente al sistema Java de SUN a través de Internet (**http://www.sun.com**) y traerse todo el material necesario para trabajar con Java:

  1. Aprender a programar en Java, pues existen cursos interactivos gratuitos y completos que explican, con todo detalle y paso a paso, cómo programar en Java.
  2. Interpretar programas Java, pues hay ya varios programas de acceso a Internet que incorporan un intérprete de Java, como Netscape 3.0, por ejemplo.
  3. Compilar programas Java, pues puede uno traerse, también vía Internet, todo el sistema de programación Java o comprar otros compiladores.

Otra ventaja importante de Java es su utilización en páginas Web, éstas como es sabido contienen muchos elementos tales como:

- Gráficos
- Texto
- Enlaces a otras páginas
- Tablas
- Sonido
- Animaciones
- etc.

Sin embargo, hasta hace poco tiempo no incorporaban aplicaciones. Con el Java, es ya posible incorporar aplicaciones o **APPLETS**, que no son más que programas que residiendo en un servidor remoto son ejecutados a través de la red mediante un intérprete utilizando los recursos del cliente.

1.3. La plataforma Java

La **plataforma Java** tiene dos componentes:

- Java Virtual Machine (**Java VM**)
- Java Application Programming Interface (**Java API**)

Todo intérprete de código Java compilado (JDK, navegador Web,...) es una implementación de la **Java VM**.

La **Java API** es un conjunto de paquetes que pueden ser utilizados por todo programa Java:

- paquete java.applet
- paquete java.awt
- paquete java.awt.event
- paquete java.io
- paquete java.math
- paquete java.lang

1.4. Direcciones de interés en internet

Algunas direcciones de interés son:

- **http://www.sun.com**
  Dirección de la empresa Sun Microsystems, desarrolladora de Java.
- **http://www.javasoft.com**
  Lugar oficial de información sobre el lenguaje Java.
- **http://java.sun.com/docs/books/tutorial/index.html**
  Tutorial interactivo de introducción a la programación en Java.
- **http://www.javaworld.com**
  Revista electrónica sobre Java.

Capítulo 2

PROGRAMACION ORIENTADA A OBJETOS

2.1. Objetos

Los objetos constituyen el concepto fundamental para entender la tecnología que se conoce como tecnología orientada a objetos. Si miramos a nuestro alrededor podemos ver muchos ejemplos de objetos del mundo real: la mesa de estudio, el ordenador, el bolígrafo, etc. Todos estos objetos del mundo real tienen dos características comunes: todos ellos tienen un *estado* y un *comportamiento*. Los objetos de un programa se modelan de manera análoga. Un objeto mantiene su estado en forma de variables miembros e implementa su comportamiento mediante métodos. Los métodos permiten crear un código que puede ser llamado desde fuera del objeto y pueden tomar argumentos y, opcionalmente, devolver un valor.

Definición 2.1 (Objeto). *Un objeto es un conjunto constituido por una o varias variables y, opcionalmente, por métodos.* ■

Todo objeto del mundo real se puede representar en un programa utilizando objetos. Todo aquello que el objeto conoce (estado) se expresa por sus variables, y todo lo que puede hacer (comportamiento) se expresa por sus métodos. Estas variables y métodos se conocen como *variables miembros* y *métodos*. Dentro de un objeto las variables constituyen el núcleo del mismo y los métodos rodean dicho núcleo aislándolo de otros objetos del programa. Este fenómeno de colocar las variables de los objetos bajo la protección de sus métodos se conoce como *encapsulación*.

Ejemplo ilustrativo 2.1 (Objeto automóvil). Un automóvil puede ser considerado un objeto. Así por ejemplo, mi automóvil en un determinado momento puede tener unas características definidas por sus variables, y además, puede hacer las acciones definidas por sus métodos:

1. **Variables:** Definen el estado actual de mi coche:

   a) Modelo: “Ford”.

b) Cilindrada: 1900 c.c.

c) Velocidad 120 km/h.

d) Marcha 5ª.

e) Dirección norte.

2. **Métodos:** Acciones que puede realizar en ese momento:

a) Acelerar.

b) Frenar.

c) Parar.

d) Cambiar de marcha.

e) Girar a la derecha.

f) Girar a la izquierda.

g) Continuar recto.

■

2.2. Clases

En el mundo real, se encuentran muchos objetos del mismo tipo. Por ejemplo, mi ordenador es uno de todos los ordenadores del mundo. Utilizando la terminología orientada a objetos, mi ordenador es un *miembro* o ejemplar de la clase de objetos conocidos como *ordenadores*. Objetos similares se reproducen como objetos del mismo tipo, por lo que se puede crear un prototipo para tales objetos. Estos prototipos son lo que se conoce como *clases*.

Definición 2.2 (Clase). *Una clase es un prototipo que define las variables y los métodos comunes a todos los objetos de un cierto tipo.* ■

Cuando se crea un ejemplar de una clase, las variables declaradas por la clase son almacenadas en memoria. Una vez creadas, se pueden utilizar los métodos miembros de la clase para asignar valores a las variables.

Ejemplo ilustrativo 2.2 (Clase automóvil). Para todos los automóviles que existan puedo crear un prototipo para definir sus variables y métodos comunes:

1. **Variables:**

a) Modelo: “Ford”, “Renault”, “Honda”, “Mercedes”, etc.

b) Cilindrada: 1200 c.c, 1800 c.c, 1900 c.c, 2000 c.c, etc.

c) Velocidad en km/h.

d) Marcha.

e) Dirección.

2. **Métodos:**

   a) Acelerar.

   b) Frenar.

   c) Parar.

   d) Cambiar de marcha.

   e) Girar a la derecha.

   f) Girar a la izquierda.

   g) Continuar recto.

■

2.3. Herencia

Los objetos se definen como miembros de clases. Conociendo la clase de un objeto se puede tener mucha información sobre el mismo. En programación orientada a objetos se pueden definir clases (*subclases*) en términos de otras clases (*superclases*). Toda subclase *hereda* el estado (en la forma de declaración de variables) y los métodos de la superclase. Sin embargo, las subclases no están limitadas al estado y comportamiento de su superclase. Las subclases pueden añadir nuevas variables miembros y nuevos métodos a los heredados de la superclase y dar implementaciones particulares para los métodos heredados anulando la definición original de los métodos dada por la superclase.

Ejemplo ilustrativo 2.3 (Clase camión). En el Ejemplo Ilustrativo 2.2 se define la clase automóvil con las variables y métodos comunes a todos los automóviles. Todas las variables y métodos de la clase automóvil son perfectamente válidas para la definición de un camión, pero hay determinadas características importantes a considerar en un camión que no están incluidas en la clase automóvil, por ejemplo, el peso máximo autorizado (PMA), la tara, o acciones como la carga y la descarga. Así pues es posible crear una subclase **camión** a partir de la clase **automóvil** que herede todas sus características de *estado* y *comportamiento* pero que además incluya otras como:

1. **Variables:**

   a) PMA: peso máximo autorizado.

   b) Tara.

2. **Métodos:**

   a) Cargar.

   b) Descargar.

   c) Se puede redefinir el método frenar, por ejemplo.

■

2.4. Creación de Clases

Una clase es un prototipo que se puede utilizar para crear objetos. La implementación de una clase consta de dos componentes: la declaración de la clase y el cuerpo de la clase.

```
DeclaracionClase {
  CuerpoClase
}
```

En la declaración de una clase debe aparecer, como mínimo, la palabra `class` y el nombre de la clase que se está definiendo.

```
class NombreClase {
        ...
    }
```

Los nombres de las clases, por convenio, comienzan por mayúscula. En la declaración de una clase se puede:

- Declarar cuál es la superclase de la clase.
- Listar las interfases implementadas por la clase.
- Declarar si la clase es pública, abstracta o final.

Comentario 2.1 *Desde el punto de vista práctico, en este curso sólo se trabajará con clases públicas y finales.* ■

En Java, toda clase tiene una superclase. Si no especificamos una superclase para la clase, se supone que es la clase `Object` (declarada en el paquete `java.lang`). Para especificar la superclase explícitamente, se pone la palabra `extends` y el nombre de la superclase entre el nombre de la clase que se está declarando y la llave que abre el cuerpo de la misma:

```
class NombreClase extends NombreSuperClase {
    ...
}
```

Cuando se declara una clase, se pueden especificar también la lista de *interfases* implementadas por la clase. Una interfase declara un conjunto de métodos y constantes sin especificar la implementación para los métodos. Cuando una clase demanda implementar una interfase, está declarando que se da una implementación para todos los métodos declarados en la interfase. Para declarar que la clase implementa una o más interfases se utiliza la palabra clave `implements`. Por convención, la cláusula `implements` sigue a `extends` si ésta existe.

Si se pone antes de `class` la palabra `final`, se declara que la clase es final, es decir, que no pueden crearse subclases de ella:

```
final class NombreClase {
    ...
}
```

El cuerpo de una clase contiene dos secciones diferentes: declaración de variables miembros y definición de los métodos asociados a dicha clase. Las variables definen el estado de un objeto definido por esta clase y los métodos corresponden a los `mensajes` que pueden ser enviados a un objeto. Generalmente, se declaran las variables miembros de la clase primero y luego las declaraciones de los métodos y su implementación:

```
DeclaracionClase {
  declaracionVariablesMiembro
    ...
  declaracionMetodos
    ...
}
```

Como mínimo, en la declaración de una variable miembro aparecen dos componentes: el tipo de dato de la variable y su nombre.

```
tipo variableNombre;
```

También se pueden especificar qué objetos tienen acceso a dicha variable, mediante `public`, `protected`, etc. o si se trata de una constante. La declaración de una variable miembro se realiza en el cuerpo de la clase pero no dentro de un método, y, por convenio, su nombre comienza por minúscula. No se pueden declarar más de una variable miembro con el mismo nombre en la misma clase, sin embargo, puede tener el mismo nombre que un método. Para crear una variable miembro constante se utiliza la palabra `final` en la declaración de la variable. La siguiente clase declara una constante llamada NUM cuyo valor es 2.23.

```
class Const{
  final double NUM=2.23;
}
```

Por convención, los nombres de las constantes se ponen con letras mayúsculas.

De manera similar a como se implementa una clase, la implementación de un método consiste de dos partes: la declaración y el cuerpo del mismo.

```
DeclaracionMetodo{
  CuerpoMetodo;
}
```

Los métodos declarados dentro de una clase tienen acceso completo a todas las variables miembro de dicha clase y pueden acceder a ellas como si estuvieran definidas dentro del método. En la declaración del método además del nombre va una cierta información, como:

- el valor de retorno del método (el que devuelve),
- el número y tipo de argumentos requeridos por dicho método y
- qué otras clases y objetos pueden llamar también al método.

Pero los dos únicos elementos requeridos en la declaración son el nombre y el tipo de dato devuelto. Si un método no devuelve ningún valor, debe declararse como `void` y si devuelve un valor debe utilizar la sentencia `return` para devolverlo.

En Java se permite definir varios métodos con el mismo nombre, estos son diferenciados por el compilador por el número y tipo de argumentos pasados en él. Por ejemplo, en la clase `Math` existen cuatro definiciones del método `abs`, que calcula el valor absoluto de un número. Estas se diferencian en el argumento, que puede ser `double`, `float`, `int` o `long`. Una clase puede redefinir un método de su superclase. El nuevo método debe tener el mismo nombre, tipo de retorno y lista de parámetros que el método que redefine. No se pueden pasar métodos como argumentos en los métodos. El argumento de un método puede tener el mismo nombre que una de las variables miembros de la clase. En este caso se dice que el argumento *oculta* a la variable miembro. Argumentos que ocultan variables miembros son frecuentemente utilizados en constructores para iniciar una clase. Por ejemplo en la clase siguiente:

```
class Persona{
  float peso,estatura;
  Color colorpelo;

  public Persona(float peso,float estatura,Color colorpelo){
    this.peso=peso;
    this.estatura=estatura;
    this.colorpelo=colorpelo;
  }
}
```

los argumentos del constructor ocultan a las variables miembro. Para referirse a la propia clase, se utiliza la palabra clave `this`. Si el método oculta una variable miembro de su superclase el método puede referirse a ella utilizando la palabra clave `super`. Lo mismo sucede si se redefinen métodos de la superclase. Sea la clase:

```
class PracticaSuper{
  int var;
  void metodo(){
  var=1;
 }
}
```

y su subclase que oculta la variable miembro `var` y redefine `metodo()`:

```
class SubPracticaSuper extends PracticaSuper{
  int var;
  void metodo(){
    var=2;
    super.metodo();
    System.out.println(var);
    System.out.println(super.var);
```

```
    }
  }
```

Primero `metodo()` asigna a la variable `var` correspondiente a la subclase el valor 2 y luego asigna a la versión de `var` de la clase superior el valor 1. Entonces `metodo` devuelve los valores de las dos variables con nombre `var`:

```
2
1
```

En el cuerpo de un método se pueden declarar variables para utilizar dentro del método. Estas variables son *variables locales*.

Ejemplo ilustrativo 2.4 (Creación de la clase automóvil). A continuación se muestra el código Java para la creación de la clase automóvil que se ha de guardar en un fichero llamado **Automovil.java**:

```
/**
 * <p>Title: Automovil</p>
 * <p>Description: Mi primera clase tipo automovil</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: </p>
 * @author roberto
 * @version 1.0
 */

public class Automovil {
  String modelo;
  int cilindrada;
  float velocidad;
  int marcha;
  String direccion;

  public Automovil(String m, int c) {
    super();
    modelo=m;
    cilindrada=c;
    velocidad=0;
    marcha=0;
    direccion = "recto";
  }

  public Automovil(String m, int c,String dir) {
    super();
    modelo=m;
    cilindrada=c;
    velocidad=0;
    marcha=0;
    direccion = dir;
  }
```

```
  void Acelera (float v){
    velocidad=velocidad+v;
  }

  void Frena (float v){
    velocidad=velocidad-v;
  }

  void Para (){
    marcha=0;
    velocidad=0;
  }

  void CambiaMarcha (int m){
    marcha=m;
  }

  void girarDerecha (){
    direccion="derecha";
  }

  void girarIzquierda (){
    direccion="izquierda";
  }

  void continuarRecto (){
      direccion="recto";
    }
}
```

■

Ejemplo ilustrativo 2.5 (Creación de la clase camión). A continuación se muestra el código Java para la creación de la clase camión, subclase de la clase automóvil, que se ha de guardar en un fichero llamado **Camion.java**:

```
/**
 * <p>Title: Camion</p>
 * <p>Description: Mi primera clase tipo camion</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: </p>
 * @author roberto
 * @version 1.0
 */

public class Camion extends Automovil {
  float PMA;
  float PesoCarga;

  public Camion(String m,int c,float p) {
   super(m,c);
```

```
   PesoCarga=p;
  }

  void Carga (float c){
    PesoCarga=PesoCarga+c;
  }

  void Descarga (float c){
    PesoCarga=PesoCarga-c;
  }

  void Para (){
   super.Para();
   PesoCarga=0;
 }

}
```

■

2.5. Creación de Objetos

Antes de proceder con los métodos de creación de objetos es importante recalcar lo siguiente:

Comentario 2.2 *Una vez creadas las clases para programar es necesario crear e interaccionar mediante objetos de las clases, se vió que en Java se pueden trabajar o bien con programas independientes o con Applets, en este curso trabajaremos por defecto con programas y para ello todos los objetos se definirán dentro de un método llamado* **main** *dentro de una clase llamada* **Principal** *cuya creación es como sigue a continuación:*

```
/**
 * <p>Title: Principal</p>
 * <p>Description: Clase para la ejecucion de los programas Java</p>
 * <p>Copyright: Copyright (c) 2005</p>
 * <p>Company: </p>
 * @author roberto
 * @version 1.0
 */

public class Principal {

  public Principal() {
  }

  public static void main(String[] args) {
  /*
    Creacion de objetos y manipulacion mediante metodos
  */
  }
```

```
}
```

La estructura de la clase Principal *es la misma para todas las aplicaciones que se emplearán en este curso.* ■

En Java se crea un objeto creando un ejemplar de una clase. Se puede crear un objeto de la forma siguiente:

```
String str = new String();
```

Así se crea un nuevo objeto de la clase String (dicha clase se encuentra en el paquete `java.lang`). Con esta declaración se realizan tres acciones: declaración, creación del objeto e iniciación. Con `String str` se crea una variable de tipo String cuyo nombre es `str`, el operador `new` crea un nuevo objeto de tipo String y `String()` inicia el objeto. La declaración de objetos aparece frecuentemente en la misma línea que la creación pero no tiene por que ser así. Como en la declaración de otras variables, las declaraciones de objetos pueden aparecer solas como:

```
String str;
```

Comentario 2.3 *Los nombres de las clases, por convenio, comienzan por letra mayúscula.* ■

El operador `new` crea un nuevo objeto reservando memoria para él. Este operador requiere un único argumento: una llamada a un método constructor. Los métodos constructores son métodos especiales con los que cuenta cada clase en Java y que son responsables de la iniciación de nuevos objetos de ese tipo. El operador `new` crea el objeto y el constructor lo inicia. Por ejemplo en:

```
new String({'h','o','l','a'});
```

`String( {'h','o','l','a' })` es una llamada al constructor de la clase String. El operador `new` devuelve una referencia al nuevo objeto creado. Esta referencia puede ser asignada a una variable del tipo apropiado.

```
String cadena = new String({'h','o','l','a'});
```

Como ya se ha mencionado las clases contienen métodos constructores para iniciar los objetos. Una clase puede tener varios constructores para realizar diferentes tipos de iniciación de los nuevos objetos. En una clase, los constructores se reconocen porque tienen el mismo nombre que la clase y no tienen tipo de retorno. Un constructor sin argumentos, se conoce como el *constructor por defecto*. Si una clase tiene múltiples constructores todos ellos deben tener el mismo nombre, coincidente con el de la clase, pero deben tener distinto número o tipo de argumentos. Cada constructor inicia un nuevo objeto de diferente modo. El compilador se encarga de identificar cuál de ellos es el llamado. Los constructores sólo pueden ser llamados con el operador `new`. Cuando se declaran los constructores de una clase, se puede especificar qué otros objetos pueden crear ejemplares de esa clase. Los niveles de acceso que se pueden utilizar son:

- privado (`private`): otras clases no pueden crear ejemplares de ésta.
- protegido (`protected`): sólo subclases de dicha clase pueden crear ejemplares de la misma.
- público (`public`): cualquier clase puede crear ejemplares de ésta.
- amistoso (`friendly`): solo clases dentro del mismo paquete que esta clase pueden crear ejemplares de ella.

Nótese que estos niveles se aplican tanto a la creación de clases como al acceso a las variables de clase y a los métodos, el nivel por defecto es el `friendly`.

Comentario 2.4 *En este curso, desde el punto de vista del acceso a objetos, trabajaremos* siempre *en modo* público *o* friendly*, es decir, se pueden crear objetos de esa clase desde cualquier otra clase. Nótese que para determinadas aplicaciones puede ser muy útil la restricción de accesos pero esta característica queda fuera del alcance de este curso.* ■

2.6. Utilización de Objetos

Para acceder a las variables de un objeto, se concatenan los nombres de la variable y del objeto con un punto. Por ejemplo, si se tiene un polígono llamado `pol` en el programa, se puede acceder a su variable miembro `npoints`, el número de puntos con el que se construye dicho polígono, con `pol.npoints`. Las variables miembros `xpoints` e `ypoints` son arrays con las coordenadas x e y de los puntos que se utilizan para construir el polígono, si se quieren dar las coordenadas de uno de estos puntos se consigue mediante la sentencia:

```
pol.xpoints[0]=2;
pol.ypoints[0]=3;
```

Para llamar a los métodos de un objeto, simplemente se concatena el nombre del método con el objeto mediante un punto, dando los argumentos del método entre paréntesis. Si el método no requiere argumentos, se utilizan los paréntesis vacíos. Por ejemplo, en la sección anterior, la variable `cadena` era de tipo `String`, con una llamada al método `length()` de `cadena` se obtiene su longitud.

```
cadena.length();
```

La llamada al método `length()` está directamente dirigida al objeto cadena. Las llamadas a métodos también se conocen como *mensajes*.

Ejemplo ilustrativo 2.6 (Manipulación de las clases automóvil y camión). Una vez creadas las clases *Automovil* y *Camión* se va a proceder a crear objetos y manipularlos. El código para realizar cada una de las siguientes acciones se inserta en el método **main()** de la clase **Principal** mostrada en el Comentario 2.2:

1. Se crea un objeto de la clase automóvil de la marca “Ford” con 2000 c.c. de cilindrada y se escriben sus características en pantalla:

```
Automovil miautomovil=new Automovil("Ford",2000);
    System.out.println("Mi automovil es modelo "+miautomovil.modelo);
    System.out.println("Con una cilindrada de "+miautomovil.cilindrada);
```

2. Se mete la primera marcha y se escribe la marcha actual:

```
miautomovil.CambiaMarcha(1);
    System.out.println("La marcha actual es "+miautomovil.marcha);
```

3. Se gira a la derecha y se escribe en pantalla la dirección actual del coche:

```
miautomovil.girarIzquierda();
    System.out.println("La direccion actual es "+miautomovil.direccion);
```

4. A continuación se crea un objeto de la clase camión, se trata de un "Volvo" de 10000 c.c. de cilindrada y cargado con 25000 kg:

```
Camion micamion;
    micamion = new Camion("Volvo",10000,25000);
```

5. La velocidad inicial es nula, es decir, está parado. A continuación se endereza la dirección y se pone el volante recto escribiendo la dirección actual:

```
micamion.continuarRecto ();
    System.out.println("La direccion del camion es "+micamion.direccion);
```

6. Se acelera a 80 km/h y se escribe la velocidad:

```
micamion.Acelera (80);
    System.out.println("La velocidad es "+micamion.velocidad+" km/h");
```

La salida por pantalla tiene queda de la siguiente manera:

```
Mi automovil es modelo Ford
Con una cilindrada de 2000
La marcha actual es 1
La direccion actual es izquierda
La direccion del camion es recto
La velocidad es 80.0 km/h
```

■

Comentario 2.5 *Para referirse a la propia clase se utiliza la palabra clave* `this` *tanto para las variables como para los métodos.* ■

2.7. Variables de instancia y de clase

Nótese que hasta ahora las variables se definían de la siguiente forma:

```
class A{
  int var;
}
```

que se conoce con el nombre de *variable de instancia*. Cada vez que se crea una instancia de una clase, es decir, un objeto de esa clase, el sistema crea una copia de cada una de las variables de instancia de la clase. Se puede acceder a ellas sólo desde un objeto de la clase.

```
A a=new A();\\ Se crea la variable a de la clase A
A b=new A();\\ Se crea la variable b de la clase A
a.var; \\ Acceso a la variable var del objeto a
b.var; \\ Acceso a la variable var del objeto b
```

Nótese que en las dos últimas sentencias se accede distintas variables.

Alternativamente, existen *variables de clase* que se declaran utilizando la palabra clave **static** de la siguiente manera:

```
class A{
  static int var;
}
```

Estas variables están asociadas a la clase, de forma que todos los ejemplares de la clase comparten la misma copia de las variables de clase. Se puede acceder a ellas a través de un ejemplar o a través de la propia clase.

```
A a=new A();\\ Se crea la variable a de la clase A
A b=new A();\\ Se crea la variable b de la clase A
a.var; \\ Acceso a la variable var a traves del objeto a
b.var; \\ Acceso a la variable var a traves del objeto b
A.var; \\ Acceso a la variable var a traves de la clase
```

Nótese que en las tres últimas sentencias se accede a la misma variable.

2.8. Métodos de instancia y de clase

Análogamente se distinguen *métodos de instancia* y *métodos de clase*, éstos últimos se definen empleando la misma palabra clave **static**.

Los métodos de instancia tienen las siguientes características:

1. Tienen acceso a las variables de instancia y de clase.
2. Sólo se puede acceder a ellos desde un objeto de la clase.

Los métodos de clase por contra:

1. Tienen acceso sólo a las variables de clase, no pueden acceder a las variables de instancia.
2. Pueden llamarse desde la clase, no es necesario un ejemplar.

2.9. Mensajes

Un objeto solo no es útil, por lo que frecuentemente aparece como una componente de un programa que contiene otros muchos objetos. Es a través de la interacción entre estos objetos como se consigue un alto nivel de funcionalidad del programa. Estos objetos interaccionan y se comunican enviándose *mensajes* unos a otros. El objeto que recibe el mensaje necesita la información adecuada para saber qué debe hacer. Esta información se pasa a través del mensaje como un *parámetro*. Un mensaje está formado por tres componentes:

- El objeto al que se envía el mensaje.
- El nombre del método a utilizar.
- Los parámetros que necesita el método.

Un ejemplo es el que sigue.

```
c3.suma(c1,c2);
```

en el que `c3` es el objeto al que se envía el mensaje, `suma` es el método a utilizar, y `c1` y `c2` son los parámetros que necesita el método.

2.10. Liberación de memoria de objetos no utilizados

Java permite crear objetos sin tener que destruirlos posteriormente. Cuando se crea un objeto, el intérprete de Java reserva suficiente espacio en memoria para él. En particular guardará suficiente memoria para sus variables miembros. Cuando un objeto deja de ser accesible, Java lo destruirá automáticamente y devolverá el espacio reservado para el objeto al espacio de memoria libre, que podrá ser utilizada por nuevos objetos. Este proceso se conoce como "recogida de basura"(*garbage collection*).

Antes de que la memoria asociada a un objeto sea liberada, el sistema llamará a su método `finalize()`. Una clase puede definir su propia finalización redefiniendo el método `finalize()` de la clase `java.lang.Object`. Dicho método debe ser declarado de la forma siguiente:

```
protected void finalize() throws throwable{
 ...
}
```

Práctica 2.1 (Clases Punto y Rectángulo). Crear un clase que se llame **Punto** de la que no se puedan generar subclases y que tenga las características siguientes:

1. Pertenece al paquete **geometric**.
2. Se emplea para definir puntos en el espacio dados por sus coordenadas, por ello habrá dos variables $\boldsymbol{x}$ e $\boldsymbol{y}$ asociadas a la clase que pueden tomar cualquier valor real con la máxima precisión posible (tipo `double`).

3. El constructor recibe dos argumentos que se corresponden con las coordenadas $\boldsymbol{x}$ e $\boldsymbol{y}$ que se quieren dar al punto.

4. Tiene un método llamado **distancia** dentro de la clase **Punto** que recibe como argumentos las coordenadas $\boldsymbol{x}_1$ e $\boldsymbol{y}_1$ de forma que calcula y devuelve la distancia $\boldsymbol{d}$ de ese punto con respecto al punto $\boldsymbol{x}$ e $\boldsymbol{y}$ definido por las variables de instancia asociadas a la clase **Punto**. La distancia se calcula con la fórmula:

$$\boldsymbol{d} = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_1)^2 + (\boldsymbol{y} - \boldsymbol{y}_1)^2}, \tag{2.1}$$

   para la cual existen dos métodos tipo `static` de la clase `Math`, uno llamado `sqrt` que recibe como argumento un número tipo `double` y devuelve la raíz cuadrada en una variable también tipo `double`, y el otro `pow` que recibe dos argumentos tipo `double` y devuelve el valor tipo `double` de la potencia del primer argumento elevado al segundo argumento.

5. También tiene otro método **distancia** con distinto tipo de argumentos, en este caso recibe una variable llamada **p** de la clase **Punto**. Dentro de este método se empleará el método **distancia** definido en el punto 4 para el cálculo y devolución de la distancia $\boldsymbol{d}$.

A continuación crear una clase llamada **Rectangulo** con las siguientes características:

1. Pertenece también al paquete **geometric**.

2. Se emplea para definir un rectángulo y tiene como variables de instancia $\boldsymbol{p}_{il}$, $\boldsymbol{p}_{ur}$, **diagonal**, **ancho** y **alto**, donde $\boldsymbol{p}_{il}$ es un objeto de la clase **Punto** que se corresponde con la esquina inferior izquierda del rectángulo, $\boldsymbol{p}_{ur}$ es un objeto de la clase **Punto** que se corresponde con la esquina superior derecha del rectángulo, y las restantes variables son longitudes de la diagonal, del ancho y del alto del rectángulo, respectivamente. Todas las dimensiones son de tipo `double`.

3. El primer constructor recibe como argumentos un ancho $\boldsymbol{a}_1$ y un alto $\boldsymbol{a}_2$ de forma que la esquina inferior izquierda del objeto rectángulo generado con este constructor sea el origen de coordenadas, mientas que las coordenadas del segundo punto se obtienen a partir del origen y de los datos de ancho y alto. La diagonal queda sin definir.

4. El segundo constructor recibe como argumentos dos puntos $\boldsymbol{p}_1$ y $\boldsymbol{p}_2$, objetos de tipo **Punto**, que se emplearán para dar valores a los puntos $\boldsymbol{p}_{il}$ y $\boldsymbol{p}_{ur}$, respectivamente. Las demás dimensiones quedan sin definir.

5. Por último, se crean tres métodos llamados **calculadiag**, **calculancho** y **calculargo** que se encargan de dar valor a las variables **diagonal**, **ancho** y **alto**, respectivamente, a partir de las coordenadas de los puntos $\boldsymbol{p}_{il}$ y $\boldsymbol{p}_{ur}$.

Para poder emplear estas dos clases crear la clase **Principal** definida en el Comentario 2.2, que pertenezca al paquete **geometric**, y que contiene el método `main()`. En ella se realizan las siguientes acciones:

1. Crear una variable $\boldsymbol{p}_1$ de tipo **Punto** con coordenadas $(0, 0)$.
2. Crear una variable $\boldsymbol{p}_2$ de tipo **Punto** con coordenadas $(30, 40)$.
3. Declarar una variable de tipo **Rectangulo** llamada r_1 pero sin inicializarla.
4. Inicializar la variable r_1 empleando los puntos $\boldsymbol{p}_1$ y $\boldsymbol{p}_2$.
5. Escribir la distancia entre los puntos $\boldsymbol{p}_1$ y $\boldsymbol{p}_2$ usando el método `System.out.println()` y el método correspondiente de la clase **Punto**.
6. ¿Podrías escribir los valores de las variables **diagonal**, **ancho** y **alto** del objeto r_1? ¿Por qué?
7. Escribir los valores de esas variables empleando el método que creáis conveniente.

Solución: La solución de todos los apartados se da a continuación. En primer lugar se describe la clase **Punto** que habrá de ubicarse en un fichero llamado "Punto.java":

```
1. package geometric;

   final class Punto {
```

```
2. double x,y;
```

```
3. public Punto(double x,double y) {
           this.x=x;
           this.y=y;
       }
```

```
4. public double distancia(double x1,double y1) {
           double d;
           d = Math.sqrt(Math.pow(x - x1, 2) + Math.pow(y - y1, 2));
           return (d);
       }
```

```
5. public double distancia(Punto p) {
           double d = this.distancia(p.x,p.y);
           return(d);
       }
   }// Final de la definicion de la clase Punto
```

A continuación se describe la clase **Rectangulo** que habrá de ubicarse en un fichero llamado "Rectangulo.java":

```
1. package geometric;

       public class Rectangulo {
```

```
2. Punto pil,pur;
          double diagonal, ancho, alto;
```

```
3. public Rectangulo(double a1,double a2) {
               pil.x = 0.0;
               pil.y = 0.0;
               pur.x = a1;
               pur.y = a2;
               ancho = a1;
               alto = a2;
           }
```

```
4. public Rectangulo(Punto p1,Punto p2) {
               pil = p1;
               pur = p2;
           }
```

```
5. public void calculadiag () {
               diagonal = pil.distancia(pur);
           }

           public void calculancho () {
               ancho = pil.distancia(pur.x,pil.y);
           }

           public void calculalto () {
               alto = pil.distancia(pil.x,pur.y);
           }
       }// Fin de la definicion de la Rectangulo
```

Para finalizar, se crea la clase **Principal** en un fichero llamado "Principal.java" perteneciente al mismo paquete **geometric**:

```
package geometric;

public class Principal {
    public Principal() {
    }
    public static void main(String[] args) {
```

Todo lo que viene a continuación está ubicado en el cuerpo del método `main`:

```
1. Punto p1=new Punto(0,0);
```

2. `Punto p2=new Punto(30,40);`

3. `Rectangulo r1;`

4. `r1 = new Rectangulo(p1,p2);`

5.
```
System.out.println("La distancia entre los punto p1 y p2 es de "
             + p1.distancia(p2));
```

6. No, porque el constructor empleado no define los valores de las variables **diagonal**, **ancho** y **alto** del objeto r_1, y hasta que no estén definidos sus valores no se puede acceder a ellos.

7.
```
r1.calculadiag();
          r1.calculalto();
          r1.calculancho();
          System.out.println("La diagonal del rectangulo r1 mide " + r1.diagonal
              + " unidades");
          System.out.println("El ancho del rectangulo r1 mide " + r1.ancho
              + " unidades");
          System.out.println("El alto del rectangulo r1 mide " + r1.alto
              + " unidades");
```

■

Ejercicios del Capítulo

Ejercicio 1. Diseñar un programa que dibuje varias figuras geométricas (rectángulos, óvalos, triángulos, pentágonos, hexágonos, etc.) cuyo contorno tenga diferentes grosores y puedan rellenarse de diferentes sombreados y colores.

Se pide: Diseñar un conjunto de clases que lo hagan fácilmente implementable y eficaz para dicho propósito.

Ejercicio 2. En una empresa se quiere trabajar con una base de datos para control de la misma que incluya la información sobre:

- El personal, clasificado por categorías.
- Los clientes, con sus nombres, apellidos, direcciones, teléfonos, números de fax, productos más comprados, etc.
- Productos fabricados, con sus precios, cantidades disponibles en almacén, etc.

- Facturas emitidas y su estado.

Se pide: Diseñar un conjunto de clases para dicho tratamiento, incluyendo las variables miembro de cada clase y los métodos correspondientes.

Ejercicio 3. Considera tu escritorio y todos los elementos de trabajo que se encuentran sobre él. Hacer una descripción del mismo mediante clases. Inicialmente se define la clase `Escritorio`, y cada uno de los elementos serían variables miembros de ésta.

Ejercicio 4. Piensa en un deporte que te guste. Describe mediante clases todas las técnicas del mismo y equipos que participan en el mismo. Por ejemplo, si éste deporte es el baloncesto, la clase principal sería la clase `Baloncesto`, cada uno de los equipos serían una clase y serían variables miembros de la clase `Baloncesto` y entre los métodos de la clase principal estarían las distintas estrategias de juego.

Ejercicio 5. Diseñar utilizando programacion orientada a objetos, una aplicación que simule unos grandes almacenes. Dicho lugar constará de tres plantas, la primera de ellas dedicada a alimentación, la segunda dedicada a ropa y la última dedicada a muebles y electrodomésticos. A su vez cada planta tiene sus propias dependencias, así en la planta de alimentación habrá carnicería, pescadería, etc., en la planta de ropa habrá una dependencia para ropa de niño y otra para ropa de adulto, etc., y en la última también se pueden distinguir distintas secciones. Se trata de que una persona que llegue a dichos almacenes pueda visitar cada una de las plantas y secciones, observar los productos y precios, y además un cliente también debe poder preguntar sobre la existencia de un cierto producto o marca. Una vez que un cliente decide comprar un producto, esta compra debe ser almacenada en una base de datos para cuando se acabe el día se pueda hacer un recuento de las ventas.

Ejercicio 6. Hacer una descripción de los coches de una determinada casa automovilística, utilizando programación orientada a objetos. En dicha aplicación aparecerá una clase principal `Coche`, con las características comunes a todos los coches de esa casa, y luego cada marca de coche dentro de esa casa que será una subclase de la clase `Coche`.

Ejercicio 7. Hacer una descripción mediante clases de los animales de un zooloógico. Utilizar subclases para las diferentes subespecies.

Capítulo 3

EL LENGUAJE JAVA

En este capítulo se comienza dando un sencillo ejemplo para ilustrar los elementos más importantes que aparecen en un programa Java. Se trata de que el lector no iniciado tenga una primera visión de la estructura de un programa, sin necesidad de profundizar demasiado en detalles concretos.

Una vez que el lector tenga esta primera impresión, se describirán más en detalle los diferentes elementos, tales como variables, constantes, operadores, matrices, cadenas de caracteres, expresiones, sentencias y bloques, de forma que pueda conocer con más profundidad cómo se tratan estos elementos en el lenguaje Java.

El capítulo termina describiendo las sentencias `if-else`, `else-if`, `switch`, `while`, `for` y `do-while` que sirven para controlar el flujo de los programas.

3.1. Un ejemplo: Los números complejos

En esta sección se da un ejemplo en el que se implementa una clase para trabajar con números complejos. Puesto que se utiliza para ilustrar inicialmente los diferentes elementos del Java, se recomienda al lector que no trate de comprender en detalle todo el programa sino sólo su estructura, ya que únicamente se trata de darle una visión muy general.

Programa 3.1 (Definición de la clase Complejo).

```
package numerocomplejo;

import java.io.*;
import java.lang.*;

// Se define la clase Complejo
public class Complejo extends Object{

// La clase tiene las dos variables miembros siguientes
  double preal;
  double pimag;
// Se define el constructor de la clase
  public Complejo(double partereal,double parteimag){
```

```
    preal=partereal;
    pimag=parteimag;
  }
// Se define el metodo para calcular el complejo opuesto
  public void opuesto(){
    preal=-preal;
    pimag=-pimag;
  }
// Se define el metodo para calcular el complejo conjugado
  public void conjugado(){
    pimag=-pimag;
  }
// Se define el metodo para calcular el modulo
  public double modulo(){
    return Math.pow(preal*preal+pimag*pimag,0.5);
  }
// Se define el metodo para calcular el argumento
  public double argumento(){
    if(preal==0.0 && pimag==0.0){
       return 0.0;
    }
    else
       return Math.atan(pimag/preal);
  }
// Se define el metodo para imprimir el complejo
  public void imprime(){
    System.out.println("(" + preal + "," + pimag + ")");
  }
// Se define el metodo para calcular la suma de dos complejos
  public void suma(Complejo c1,Complejo c2){
    preal=c1.preal+c2.preal;
    pimag=c1.pimag+c2.pimag;
  }
// Se define el metodo para calcular el producto de dos complejos
  public void producto(Complejo c1,Complejo c2){
    preal=c1.preal*c2.preal-c1.pimag*c2.pimag;
    pimag=c1.preal*c2.pimag+c1.pimag*c2.preal;
  }
// Se define el metodo para calcular el cociente de dos complejos
// y se genera una excepcion cuando el divisor es el complejo cero
  public void cociente(Complejo c1,Complejo c2)
    throws ZerodivisorException{
    double aux;
    if(c2.preal==0.0 && c2.pimag==0.0){
       throw new ZerodivisorException("Divide por cero");
    }
    else{
      aux=c2.preal*c2.preal+c2.pimag*c2.pimag;
      preal=(c1.preal*c2.preal+c1.pimag*c2.pimag)/aux;
      pimag=(c1.pimag*c2.preal-c1.preal*c2.pimag)/aux;
```

```
    }
  }
// Se define la excepcion ZerodivisorException
  class ZerodivisorException extends Exception{
    ZerodivisorException(){super();}
    ZerodivisorException(String s){super(s);}
  }
}
```

■

Las dos primeras sentencias `import`, que inician el programa, se utilizan para importar clases ya programadas. De esta forma podemos referirnos a ellas y utilizarlas sin necesidad de programarlas. El lenguaje Java viene con una serie de paquetes de programas como el `io`, `util`, `lang` o `awt`, etc., que se describen al final de este capítulo.

En las sentencias que siguen se define la clase `Complejo`. Toda clase en Java debe ser una extensión de otra clase ya existente. En caso de que no haya alguna que nos interese extender, como en este caso, Java supone que es la clase `Object`, que es la clase que ocupa el lugar preeminente de la jerarquía. La sentencia

```
public class Complejo extends Object
```

da el nombre `Complejo` a la clase, la declara de tipo pública, mediante el adjetivo `public`, con lo que puede ser accesible desde fuera de la clase, y la define como una extensión de la clase `Object`, con lo que hereda todas sus variables y métodos.

Toda clase tiene dos componentes: variables miembros y métodos. En este caso las variables miembros de la clase `Complejo` son su parte real y su parte imaginaria, que se declaran de tipo doble precisión (`double`) y se llaman `preal` y `pimag`, respectivamente, mediante las sentencias:

```
double preal;
double pimag;
```

Seguidamente se definen los métodos, que indican cómo se crean ejemplares de la clase y qué operaciones pueden hacerse con los números complejos y cómo hacerlas. El primero de ellos es el constructor de la clase. Cuando se crea un ejemplar de esta clase, este método lo inicia y le asigna las partes real e imaginaria que le indiquen sus argumentos (`partereal` y `parteimaginaria`) a sus variables miembros (`preal` y `pimag`).

Los métodos `opuesto()` y `conjugado()` cambian las partes imaginarias y/o reales para obtener los complejos opuesto y conjugado, respectivamente.

El método `modulo()` devuelve el módulo del complejo que es

$$|(a,b)| = (a^2 + b^2)^{1/2}$$

Para elevar a la potencia 1/2 se utiliza la función `pow` del paquete `Math`.

El método `argumento()` controla primero si el complejo es nulo, mediante la sentencia `if(preal==0.0 && pimag==0.0)`, que simplemente comprueba si las partes real e imaginaria son nulas, en cuyo caso devuelve el argumento nulo (`return 0.0`). El operador `&&` es

el operador relacional "y". En caso contrario se utiliza la función `atan()` del paquete `Math` que da el arco tangente para calcular el argumento a devolver.

El método `imprime()` imprime el complejo con el formato (a, b) utilizando la clase `System` y la función `println`. Los signos `+` concatenan las diferentes partes del texto. Nótese que hay que abrir y cerrar los paréntesis, poner la coma de separación y escribir las partes real e imaginaria, en su correspondiente orden.

Los métodos `suma()`, `producto()` y `cociente()` calculan la suma, el producto y el cociente de dos números complejos. Para ello, hay que recordar que las fórmulas que dan la suma, el producto y el cociente de complejos son:

$$(a, b) + (c, d) = (a + c, b + d), \tag{3.1}$$

$$(a, b) \times (c, d) = (ac - bd, ad + bc) \tag{3.2}$$

$$(a, b)/(c, d) = \left(\frac{ac + bd}{c^2 + d^2}, \frac{ad - bc}{c^2 + d^2}\right). \tag{3.3}$$

En el cociente, el método genera la excepción `ZerodivisorException`, con el mensaje `‘‘Divide por cero’’`, cuando el complejo divisor es nulo. La variable `aux` es una variable auxiliar de tipo `double` que se utiliza para no tener que calcular dos veces el denominador común de la expressión (3.3).

Finalmente, la clase `ZerodivisorException` define qué hacer cuando se produce dicha excepción. En este caso, solo definimos el constructor, aunque de dos formas diferentes: una sin argumento, que simplemente llama al constructor de la superclase `Exception` y, otra, con un argumento, que también llama al correspondiente constructor de la superclase, es decir al que posee el mismo argumento. Esta posibilidad de definir métodos con el mismo nombre y diferentes argumentos, no admitida en otros lenguajes, caracteriza al lenguaje Java. El compilador Java se encarga de determinar de cuál de ellos se trata en cada caso, simplemente analizando el número de sus argumentos y su tipo.

Para poder utilizar la clase `Complejo` se necesita escribir el método `main()` (si se trata de una aplicación) que cree ejemplares de la clase `Complejo` y utilice sus métodos. Esto se hace en el programa que sigue, si bien antes se ha incluido el resultado que se obtiene en pantalla al ejecutarlo. El lector debe comprobar, paso a paso, cómo este resultado se va generando al ejecutar las sentencias del programa.

Programa 3.2 (Programa para probar la clase complejo).

```
package numerocomplejo;

import numerocomplejo.Complejo.*;

public class Principal {
  public Principal() {
  }
  public static void main(String[] args) {

    Complejo c1,c2,c3;
```

```
    c1=new Complejo(1.0,2.0);
    c2=new Complejo(3.0,4.0);
    c3=new Complejo(0.0,0.0);
    c1.imprime();c2.imprime();
    c3=c2;
    c3.opuesto();c3.imprime();
    c3.conjugado();c3.imprime();
    c3.suma(c1,c2);c3.imprime();
    System.out.println("El modulo de la suma es = " + c3.modulo());
    System.out.println("El argumento de la suma es = " + c3.argumento() +
     " radianes");
    c3.producto(c1,c2);c3.imprime();
    c2.preal=0.0;c2.pimag=0.0;
    System.out.println("El argumento  de c2 es = " + c2.argumento() +
     " radianes");
    try{
      c3.cociente(c1,c2);
      System.out.println("El cociente es : ");
      c3.imprime();
    }catch (ZerodivisorException e){
      System.out.println("Al calcular el cociente se ha producido una excepcion \n"
                + e.getClass() + "\n" + "con el mensaje : " + e.getMessage());
    }
  }
}
```

■

El resultado del programa queda de la siguiente manera:

```
(1.0,2.0)
(3.0,4.0)
(-3.0,-4.0)
(-3.0,4.0)
(-2.0,6.0)
El modulo de la suma es = 6.324555320336759
El argumento de la suma es = -1.2490457723982544 radianes
(-14.0,-22.0)
El argumento  de c2 es = 0.0 radianes
Al calcular el cociente se ha producido una excepcion
class numerocomplejo.Complejo$ZerodivisorException
con el mensaje : Divide por cero
```

La primera sentencia indica que la clase `Principal` pertenece al mismo paquete que la clase `Complejo`, mientras que la segunda importa todas las características de la clase `Complejo`, en la que se han definido los números complejos y las operaciones que pueden realizarse con ellos (calcular su módulo, argumento, su opuesto, su conjugado, imprimirlo, suma, producto y cociente, etc.), imprescindible para que no de un error al arrojar la excepción.

Una de las formas de ejecutar un programa Java es mediante la definición de una clase que contenga el método `main()`. Por ello, la sentencia

```
public class Principal
```

define la clase `Principal`, que contendrá el método main para la generación de objetos y la escritura. El adjetivo `public` indica que va a ser accesible desde el exterior, es decir, que pueden crearse ejemplares de esta clase desde fuera de ella.

Dentro de esta clase se crea el método `main()`, que es el que Java busca entre las clases existentes para comenzar la ejecución del programa. En él se definen, mediante la sentencia `Complejo c1,c2,c3`, tres copias de la clase `Complejo`, que se llaman `c1, c2` y `c3`, y que reciben el nombre de variables, puesto que pueden cambiar sus valores durante la ejecución del programa. Con esta sentencia sólo se indica el tipo de variable (`Complejo`) de que se trata, pero no se reserva espacio en memoria para ellas, lo que sí se hace en las sentencias que siguen

```
c1=new Complejo(1.0,2.0);
c2=new Complejo(3.0,4.0);
c3=new Complejo(0.0,0.0);
```

en las que además se inician los complejos dándoles sus partes reales e imaginarias. La palabra clave `new` sirve para crear un nuevo ejemplar y reservar la memoria necesaria.

Al método `Complejo()` se le llama *constructor*, por encargarse de construir un elemento de la clase, reservando espacio en memoria y haciendo las iniciaciones necesarias. Nótese que tiene el mismo nombre que la clase que construye.

Seguidamente, se imprimen los complejos `c1` y `c2`, llamando al método `imprime()`, que es uno de los métodos de la clase `Complejo`. Nótese que para llamar al método se escribe el nombre de la variable seguido de un punto y del nombre del método. Si éste tiene argumentos, se incluyen entre paréntesis y, si no los tiene, se escriben los paréntesis sin nada en su interior, como ocurre en este caso.

La sentencia `c3=c2` asigna al complejo `c3` el valor que en ese instante tenga el complejo `c2`, por lo que `c3`, que inicialmente era el complejo $(0, 0)$, (fue iniciado a ese valor mediante la sentencia `c3=new Complejo(0.0,0.0)`), pasa a ser el complejo $(3, 4)$.

Las sentencias dobles

```
c3.opuesto();c3.imprime();
c3.conjugado();c3.imprime();
```

convierten a `c3` primero en su opuesto y lo imprime, y luego en su conjugado y lo imprime.

Con las sentencias

```
c3.suma(c1,c2);c3.imprime();
```

se almacena en `c3` la suma de los complejos `c1` y `c2` y se imprime.

Las sentencias

```
System.out.println("El modulo de la suma es = " + c3.modulo());
System.out.println("El argumento de la suma es = " +
c3.argumento() + " radianes");
```

imprimen el módulo de la suma y su argumento, utilizando el método de impresión `println` del sistema.

Posteriormente se calcula el producto de los complejos `c1` y `c2` y se almacena el resultado en el complejo `c3`.

Con objeto de probar el comportamiento de los métodos `argumento` y `cociente` cuando se trata con el complejo nulo, se hace primero `c2` igual al complejo nulo, mediante las sentencias

```
c2.preal=0.0;c2.pimag=0.0;
```

luego se escribe su argumento con la sentencia

```
System.out.println("El argumento  de c2 es
= " + c2.argumento() + " radianes");
```

y seguidamente se calcula el cociente y se imprime mediante

```
try{
  c3.cociente(c1,c2);
  System.out.println("El cociente es : ");
  c3.imprime();
}catch (ZerodivisorException e){
  System.out.println("Al calcular el cociente se ha
    producido una excepcion " + e.getClass()
    + "\ncon el mensaje : " + e.getMessage());
}
```

Nótese que se prevé la posibilidad de obtener un error (en este caso una división por cero), por lo que se escribe el verbo `try` (intentar) antes del cálculo del cociente, y el verbo `catch` (capturar) que indica lo que debe hacerse en dicho caso, es decir, imprimir un mensaje indicando que se ha producido una excepción en la clase correspondiente. Se recomienda al lector en este punto que haga un esfuerzo por entender la estructura de los programas 3.1 y 3.2 anteriores antes de seguir con la lectura de los detalles del lenguaje Java, pues seguidamente se pasa ya a describir los diferentes elementos del lenguaje.

Comentario 3.1 *Desde el punto de vista práctico en este curso no se dará excesiva importancia al tratamiento de las excepciones, para más detalles véase el libro [1].* ■

3.2. Variables

Las variables pueden considerarse como zonas o espacios de memoria en las que se almacena la información. Son los pilares de la programación y los elementos sobre los que se actúa. Para poder utilizar una variable es necesario especificar su *nombre* y su *tipo*. El nombre sirve para saber a qué zona de memoria se está uno refiriendo, y el tipo sirve para saber el tamaño de la memoria necesario para almacenarla y cómo debe interpretarse la información, tanto al guardarla en memoria como al leerla. En realidad, la información, sea

del tipo que sea, se almacena en binario, es decir en forma de bits (ceros y unos), por lo que hay que transformarla tanto para almacenarla como para interpretarla.

El lenguaje JAVA está fuertemente tipificado, lo que significa que toda variable y toda expresión tienen un tipo, que se conoce al compilar, es decir, no hay que esperar a ejecutar el programa para saber el tipo de datos que contiene una variable dada. Esto limita los valores que pueden tomar las variables y producir las expresiones, limita las operaciones que pueden realizarse con las variables y expresiones y determina el significado de las operaciones que las afectan. Todo ello sirve para detectar errores en el momento de la compilación, que de otro modo pasarían desapercibidos.

Por tanto, al declarar una variable deben especificarse su tipo y su nombre. Por ejemplo, las sentencias:

```
int  edad;
double estatura;
boolean terminado;
```

definen tres variables cuyos nombres son "edad", "estatura" y "terminado" y cuyos valores son de tipo "entero" (`int`), "doble precisión" (`double`) y "booleano" (`boolean`), respectivamente.

Algunos ejemplos de definición de variables son:

```
int a,b;
String nombre,domicilio;
long d;
boolean f;
```

El punto y coma al final de toda sentencia en Java indica el final de la misma y no hay que olvidarlo.

3.2.1. Nombre de una variable

El nombre de una variable no es de libre elección, sino que debe cumplir las condiciones siguientes:

1. Debe comenzar siempre con una letra (mayúscula o minúscula) o con un signo de subrayado (`_`).

2. El resto del nombre puede estar formado por combinaciones de letras, números y signos de subrayado. Por ejemplo `miEdad`, `Edad_1`,`A1`

3. No debe incluir espacios ni caracteres como `&` y `*`.

4. Debe ser un identificador legal de Java formado por una serie de caracteres unificados (Unicode). Unicode es un sistema de codificación de caracteres diseñado para soportar texto escrito en diferentes idiomas. Permite hasta un total de 34.168 caracteres diferentes, que incluyen caracteres acentuados de todo tipo, caracteres griegos, hebreos, cirílicos, japoneses, etc.

5. No debe coincidir con una palabra clave ni con `true` o `false`.

   Las palabras clave reservadas por Java son:

`abstract`	`default`	`if`	`private`	`throw`
`boolean`	`do`	`implements`	`protected`	`throws`
`break`	`double`	`import`	`public`	`transient`
`byte`	`else`	`instanceof`	`return`	`try`
`case`	`extends`	`int`	`short`	`void`
`catch`	`final`	`interface`	`static`	`volatile`
`char`	`finally`	`long`	`super`	`while`
`class`	`float`	`native`	`switch`	
`const`	`for`	`new`	`synchronized`	
`continue`	`goto`	`package`	`this`	

6. Tampoco debe coincidir con el nombre de otra variable declarada en su mismo campo de acción, pudiendo hacerlo con variables declaradas en diferente campo.

3.2.2. Tipos de datos

En Java, toda variable tiene siempre asignado un único tipo de dato. Este determina los valores que la variable puede almacenar y las operaciones que se pueden realizar con ella.

Hay tres tipos de variables o expresiones:

- **Básicas** : que contienen la información propiamente dicha. Estas pueden ser:
  - Numéricas : que a su vez pueden ser:
    - Enteras : `byte`, `short`, `int`, `long`, que toman valores enteros.
    - Coma flotante : `float` (simple precisión) y `double` (doble precisión), que toman valores reales.
  - Booleanas : `boolean`, que toman los valores lógicos "`true`" y "`false`".
  - Tipo carácter : `char`, que toman sólo valores de tipo carácter.
- **Referenciales** : que no contienen la información, sino dónde se encuentra ésta, es decir, son referencias a objetos. Estas pueden ser:
  - Tipo clase : `class`, que contienen referencias a clases.
  - Tipo interfase : `interface`, que contienen referencias a una interfase.
  - Tipo matriz : `array`, que contienen referencias a una matriz.
- **Tipo nulo** : que es un tipo especial que no tiene nombre, por lo que no se pueden declarar variables o transformar otras a este tipo. La referencia nula `null` es el único valor que puede tomar una expresión de este tipo. Todas las variables de tipo referencial pueden tomar este valor `null`.

En la Tabla 3.1 se describen, uno a uno, los diferentes tipos de datos o variables básicas, sus tamaños asociados y los valores mínimos y máximos de cada uno de ellos.

Veamos un ejemplo:

Programa 3.3 (Ilustración de los diferentes tipos de variables).

```
class DrawPanel extends Panel{

  static int dimension;
  int x,y;
  double[] a=new double[10];

  void Draw(Graphics g) {
    int RectLength=80;
    g.setColor(Color.blue);
    g.fillRect(x,y,RectLength,RectLength);
    g.setColor(Color.black);
  }
}
```

■

Este programa crea una clase `DrawPanel`, que es una subclase de (extiende) la clase ya existente `Panel`, en la que se definen las variables miembros `dimension`, `x` e `y`, que son de tipo `int` y la variable `a`, que es una matriz, de dimensión 10. Además, se define un método `Draw`, que lleva un argumento `g`, de tipo `Graphics`. Este método dibuja un rectángulo sombreado con su vértice superior izquierdo en el punto (x,y), de anchura `RectLength` y de la misma altura y asigna el color negro al gráfico `g`.

La variable `g` del programa anterior está declarada como un objeto de la clase `Graphics` y es de tipo referencial, es decir, no contiene el gráfico sino sólo la dirección de memoria en la que se encuentra. El nombre `g` hace referencia a la dirección de memoria donde reside la variable. Por el contrario, el nombre de la variable primitiva `RectLength` hace referencia al valor real de la variable.

Tipo	**Tamaño**	**Mínimo**	**Máximo**
`byte`	8 bits	-256	255
`short`	16 bits	-32768	32767
`int`	32 bits	-2147483648	2147483647
`long`	64 bits	-9223372036854775808	922337203685477580
`float`	32 bits	$-2^{24}E-149$	$2^{24}E104$
`double`	64 bits	$-2^{53}E-1045$	$2^{53}E1000$
`boolean`	N/A	`false`	`true`
`char`	16 bits	Carácter Unicode	

Tabla 3.1: Tipos de datos básicos o variables.

Observaciones:

- Si se desea especificar un entero en notación octal, el entero debe estar precedido por un 0. Si se desea notación hexadecimal se precede el entero con un 0 y una `x`.

```
int n=023;
int m=0xF;
```

- El tipo `double` tiene más precisión y puede almacenar más cifras decimales.
- En la mayoría de los ordenadores, los enteros se procesan mucho más rápido que los números almacenados en coma flotante.
- Los números en coma flotante pueden almacenar valores más grandes y más pequeños que los enteros.

3.2.3. Campo de acción de una variable

Un detalle importante a la hora de programar, es el lugar de declaración de una variable, el cual determina su campo de acción.

Se denomina *campo de acción* de una variable al bloque de código en el que dicha variable es accesible. Además este campo determina cuándo se crea la variable y cuándo se destruye.

Atendiendo al campo de acción, se pueden clasificar las variables en las clases siguientes:

- **Variable de clase**. Es una variable de tipo clase que se declara dentro de la definición de una clase o interfase, usando la palabra clave `static`, pudiendo prescindir de ésta en la definición de una interfase. Estas variables son creadas e iniciadas, a valores determinados o por defecto, al construirse la clase, dejando de existir cuando desaparece dicha clase y tras completar el proceso previo a la destrucción del objeto (método `finalize`).
- **Variables miembros de una clase**. Se declaran en la definición de una clase sin usar la palabra clave `static`. Si `a` es una variable miembro de una clase `A`, cada vez que se construye un nuevo objeto de la clase `A` o una subclase de `A` se crea una nueva variable `a` y se inicia dándole un valor concreto o por defecto. La variable deja de existir cuando desaparece la clase a la que pertenece y tras completar el proceso previo a la destrucción del objeto (método `finalize`).
- **Componentes de una matriz**. Son variables sin nombre que se crean e inician a valores concretos o por defecto cuando se crea un objeto nuevo de tipo matriz. Las componentes de la matriz dejan de existir cuando desaparece la misma.
- **Variable local**. Se declara dentro de un método y sólo es accesible dentro del mismo. Por ejemplo, cuando se entra en un bloque o ciclo `for`, se crea un nuevo ejemplar por cada variable local del bloque o ciclo. Sin embargo, las variables locales no se inician hasta que se ejecuta la sentencia de declaración correspondiente. Las variables locales dejan de existir cuando se termina el bloque o ciclo.

- **Parámetro de un método o constructor**. Es el argumento formal de un método o un constructor, mediante el cual se pasan valores a un método. Cada vez que se invoca el método se crea una variable con ese nombre y se le asigna el valor que tenía la correspondiente variable en el método que la llama. Su campo de acción es el método o constructor del cual es argumento.
- **Parámetro de un manipulador de excepciones**. Es similar al parámetro de un método pero en este caso es el argumento de un manipulador de excepciones.

En el ejemplo anterior:

- `RectLength` es una variable local.
- La variable `g` es un parámetro de un método.
- La variable `dimension` es una variable de clase.
- Las variables `x` e `y` son variables miembros de la clase `DrawPanel`.
- `a[]` es una matriz.

3.2.4. Iniciación de variables

Al declarar las variables locales, las de clase y las que son miembros de clases pueden dársele valores iniciales. Por ejemplo, las tres sentencias de definición anteriores pueden sustituirse por las siguientes:

```
int  edad=30;
double estatura=1.78;
boolean Terminado=true;
```

en cuyo caso además de reservar espacio en memoria para ellos, se almacenan en ellos los valores indicados.

Los parámetros de los métodos y los de los manipuladores de excepciones no pueden iniciarse de este modo. El valor de estos parámetros es fijado por el método que los llama.

3.3. Constantes

Las constantes en Java son como las variables, pero se diferencian de ellas en que su valor no se altera durante la ejecución del programa. Para distinguirlas de ellas se las precede del adjetivo `final`. Por convenio, las constantes se escriben todas en mayúsculas. Por ejemplo, las conocidas constantes π y e pueden definirse como sigue:

```
final double PI=3.14159265358979;
final double E=2.71728182;
```

3.4. Comentarios

En un programa pueden introducirse comentarios que aclaren ciertos aspectos del programa siempre que se indique al compilador cuándo comienzan y, en algunos casos, cuándo terminan. En Java se distingen tres clases de comentarios:

- `/* texto */`. El texto comprendido entre `/*` y `*/` es ignorado por el compilador. Se utiliza para comentarios de más de una línea.
- `// texto`. El texto desde `//` hasta el final de línea es ignorado. Se utiliza para comentarios de una sola línea.
- `/** documentación */`. El texto comprendido entre `/**` y `*/` puede ser procesado por otra herramienta para preparar documentación de la siguiente declaración de clase, constructor, interfase, método o campo.

Es una buena práctica de programación utilizar comentarios explicando el significado de las variables y los argumentos de los métodos, así como comentar en detalle qué hacen éstos últimos. De no hacerlo, será muy difícil a otras personas, o incluso a nosotros mismos, entender lo que hace el programa.

3.5. Operadores

Todos los tipos de datos básicos están asociados a ciertos operadores mediante los cuales se construyen expresiones.

Existen operadores *unarios* y *binarios* que requieren uno o dos operandos, respectivamente.

Los operadores unarios pueden aparecer antes o después del operando:

```
x++;
++y;
```

Los operadores binarios siempre aparecen entre los dos operandos:

```
5*6;
a+b;
```

Todo operador devuelve siempre un valor. Dicho valor y su tipo dependen del operador y del tipo de los operandos.

Los operadores pueden clasificarse en las siguientes categorías:

- *aritméticos*
- *relacionales* y *condicionales*
- *bit-a-bit* y *lógicos*
- de *asignación*

3.5.1. Operadores aritméticos

Se distinguirá entre operadores binarios y unarios.

Operadores Binarios

Operadores Binarios		
Operador	**Operación**	**Descripción**
`+`	`a + b`	Adición
`-`	`a - b`	Diferencia
`*`	`a * b`	Producto
`/`	`a / b`	División
`%`	`a % b`	Calcula el resto de dividir a entre b

Tabla 3.2: Operadores binarios

Los operadores aritméticos binarios, salvo el operador `%` que sólo actúa con enteros, manejan indistintamente los tipos enteros y los de coma flotante. El compilador Java se encarga de detectar el tipo correspondiente.

Cuando un operador tiene operandos de diferente tipo, éstos se convierten a un tipo común siguiendo las reglas:

- La categoría de los tipos, de mayor a menor, es la siguiente: `double, float, long, int, short, char`.
- En cualquier operación en la que aparezcan dos operandos de tipos diferentes se eleva la categoría del que la tenga menor.
- En una sentencia de asignación, el resultado final se convierte al tipo de la variable a la que son asignados.

Operadores Unarios

Operadores Unarios		
Operador	**Operación**	**Descripción**
`+` y `-`	`+a` y `-a`	Fijan el signo del operando
`++`	`++x`	Añade 1 a `x` antes de utilizarla
	`x++`	Añade 1 a `x` después de utilizarla
`--`	`--x`	Resta 1 a `x` antes de utilizarla
	`x--`	Resta 1 a `x` después de utilizarla

Tabla 3.3: Lista de operadores unarios

Veamos cómo actúan estos operadores con algunos ejemplos:

```
int i=7;
4 * i++;
i+2;
```

Las dos últimas sentencias dan como resultado `4*7=28` y `8+2=10`, pues en la primera se multiplica 4 por el valor de la variable i, que es 7, y luego se incrementa dicha variable en una unidad, por lo que toma el valor 8. Tras esto se le suma 2 en la sentencia siguiente.

```
int i=7;
4 * ++i;
i+2;
```

En este caso los resultados son `4*8=32` y `8+2=10`, pues primero se incrementa una unidad a la variable i y luego se multiplica por 4.

3.5.2. Operadores relacionales y condicionales

Los operadores relacionales comparan dos valores y determinan la relación que existe entre ellos. Los primeros se muestran en la Tabla 3.4 y los segundos en la Tabla 3.5.

Operadores Relacionales		
Operador	**Empleo**	**Descripción**
`>`	`a > b`	Devuelve `true` si `a` es mayor que `b`, `Devuelve false` en caso contrario.
`>=`	`a >= b`	Devuelve `true` si `a` es mayor o igual que `b`, `Devuelve false` en caso contrario.
`<`	`a < b`	Devuelve `true` si `a` es menor que `b`, `Devuelve false` en caso contrario.
`<=`	`a <= b`	Devuelve `true` si `a` es menor o igual que `b`, `Devuelve false` en caso contrario.
`==`	`a == b`	Devuelve `true` si `a` y `b` son iguales, `Devuelve false` en caso contrario.
`!=`	`a != b`	Devuelve `true` si `a` y `b` son diferentes, `Devuelve false` en caso contrario.

Tabla 3.4: Lista de operadores relacionales.

Normalmente, los operadores relacionales se usan junto a los operadores llamados *condicionales* formando así expresiones más complejas. Existen tres operadores condicionales.

Observar que el operador `&` es equivalente a `&&` si sus dos operandos son de tipo `boolean`. Similarmente ocurre con `|` y con `||`.

3.5.3. Operadores de asignación

El operador `=` asigna a la variable de la izquierda el valor que toma la expresión de la derecha:

Operadores Condicionales		
Operador	**Empleo**	**Descripción**
`&&`	`a && b`	Devuelve `true` si `a` y `b` son `true`, Devuelve `false` en caso contrario.
`||`	`a || b`	Devuelve `false` si `a` y `b` son `false`, Devuelve `true` en caso contrario.
`!`	`!a`	Devuelve `false` si `a` es `true` y Devuelve `true` si `a` es `false`

Tabla 3.5: Lista de operadores condicionales.

```
float i = 3.5;
```

Además de este operador básico existen otros que nos permiten simplificar cualquier operación. Por ejemplo:

```
n = n*5;
```

es equivalente a:

```
n *= 5;
```

La Tabla 3.6 muestra todos los operadores de asignación:

Operadores de Asignación		
Operador	**Empleo**	**Descripción**
`+=`	`a += b`	`a = a + b`
`-=`	`a -= b`	`a = a - b`
`*=`	`a *= b`	`a = a * b`
`/=`	`a /= b`	`a = a / b`
`%=`	`a %= b`	`a = a % b`
Existen otros		

Tabla 3.6: Lista de operadores de asignación.

3.5.4. El operador cast

A veces se necesita almacenar valores de un tipo en variables de tipo diferente. Como Java no permite almacenar a cada variable mas que valores de su tipo, es necesario transformarlos previamente.

El lenguaje Java no permite una conversión tan fácil de tipos como el C. Para realizarla debe recurrirse al operador `cast`, que transforma un tipo de dato en otro compatible, tal como se ve en el siguiente ejemplo:

```
float a;
int b;

b = (int) a;
```

Es decir, a la variable `b`, que es de tipo `int` (entera) se le asigna el valor de la parte entera de la variable `a` que es de tipo `float`.

3.6. Matrices

En el programa pueden necesitarse un gran número de variables, todas ellas del mismo tipo y bajo una característica común. Por ejemplo, considérense los 100 primeros múltiplos de 2. Estos podrían almacenarse así:

```
int m1 = 2;
int m2 = 4;
int m3 = 6;
...
int m100 = 200;
```

lo que resultaría bastante penoso. Por ello, se dispone de las matrices (`arrays`), de manera que la declaración se hace como sigue:

```
int mult[];
mult = new int[100];
```

En la primera sentencia se especifica el tipo de los elementos de la matriz (`int` en este caso) y se da un nombre a la matriz, seguido de un doble corchete `[]` el cual indica al compilador que lo que se declara es una matriz.

```
int mult[]
```

Pero la declaración `int mult[];` no asigna memoria para almacenar los elementos de la matriz, de manera que si se intenta acceder a alguno de ellos o asignarle un valor, el compilador dará un mensaje de error. Para evitar esto y asignar la memoria necesaria se recurre al operador `new`

```
mult = new int[100];
```

A partir de ese instante ya se pueden dar valores a todos los elementos de la matriz:

```
for(int i=0;i<100;i++) {
   mult[i] = 2*(i+1);
}
```

Para acceder al elemento i-ésimo de la matriz, basta con escribir: `mult[i]`, teniendo en cuenta que los índices de la matriz comienzan a numerarse en 0.

Los elementos de una matriz son tratados como cualquier variable y pueden ser de cualquier tipo, incluso de tipo referencial como objetos u otras matrices. Por ejemplo:

```
String a[] = new String[20];
```

Los elementos de esta matriz son tipos de referencia, cada elemento hace referencia a un objeto `String`. Pero en la declaración no se ha asignado suficiente memoria para los objetos `String`, de manera que si se quiere acceder a algún elemento de la matriz `a` se obtiene una excepción `NullPointerException`, que indica que no se ha reservado memoria para dicha variable. Así que se necesitan hacer las siguientes declaraciones:

```
for(int i=0;i<a.length;i++) {
  a[i] = new String(''Elemento '' + i);
}
```

3.7. Cadenas de caracteres

Se denomina *cadena de caracteres* a una secuencia o conjunto ordenado de ellos. Toda cadena de caracteres se implementa mediante la clase `String`.

En un ejemplo anterior se ha visto una forma de manejar estos objetos:

```
String a[] = new String[20];
```

Otra forma consiste en usar cadenas constantes de caracteres, delimitadas por `''` (comillas), como por ejemplo:

```
''Hola'';
```

También se pueden unir dos cadenas de caracteres mediante el operador de concatenación `+`. Por ejemplo:

```
''La ecuacion tiene '' +  n + '' soluciones.''
```

donde `n` es una variable entera definida con anterioridad dentro del programa y convertida a `string` antes de unirse a las dos cadenas. Este recurso se emplea con frecuencia asociado al método `System.out.println()`.

3.8. Expresiones

Toda *expresión* es una combinación de variables, operadores y llamadas a métodos, que calcule un valor.

Las expresiones se utilizan para calcular y asignar valores a variables y para controlar la ejecución de un programa. Realizan las operaciones indicadas por sus elementos y devuelven algún valor.

Existen dos tipos de expresiones:

- **Numéricas**: que toman valores numéricos.
- **Booleanas**: que sólo toman los valores `true` o `false`.

3.8.1. Expresiones numéricas

Cualquier variable que represente un número es una expresión numérica. Veamos algunos ejemplos:

```
int i
i+10
--n;
```

Toda asignación es también una expresión:

```
a = 100
b* = 2
```

3.8.2. Expresiones booleanas

Sólo poseen los valores `true` o `false`. Por ejemplo:

```
n != m
a < b
```

Las dos sentencias:

```
int a = 5
if(a){
  ...
}
```

no son correctas, pues `a` es una expresión numérica y el compilador no puede evaluarla como `true` o `false`.

En una expresión es importante el orden en que se aplican los operadores. No es lo mismo

```
5 * 3 + 4
```

que

```
5 * (3 + 4)
```

Si se omiten los paréntesis el compilador actúa por su cuenta y aplica sus reglas de preferencia de modo que los operadores que figuran delante en el orden de preferencia actúan antes que los que figuran detrás. En este caso, el operador `*` está por delante del operador `+`, de manera que el resultado sería `(5*3) + 4`.

En la Tabla 3.7 se da el orden de preferencia de los operadores: los operadores están ordenados de arriba a abajo según su orden de preferencia; los operadores de una misma línea tienen el mismo orden de preferencia y, dentro de una expresión, se evalúan de izquierda a derecha.

```
. [] ()
++ -- ! ~ instanceof
new (type)
* / %
+ -
<< >> >>>
< >
== !=
&
^
|
&&
||
? :
= += -= *= /= %= ^=
&= |= <<= >>= >>>=
```

Tabla 3.7: Orden de preferencia de los operadores.

3.9. Sentencias y bloques

Un programa no es más que un conjunto de sentencias ordenadas. Una *sentencia* es una instrucción completa para el compilador, que acaba en un punto y coma.

Una expresión como `a = 100` ó `a <b` se convierte en una sentencia cuando va seguida de un punto y coma:

```
a = 100;
a < b;
```

Una sentencia indica una acción. La expresión

```
2+2;
```

indica al compilador que realice esa operación pero no, lo que debe hacer con el resultado. Por el contrario, la sentencia

```
a = 2+2;
```

ordena al compilador que almacene el resultado en la variable `a`. A continuación el ordenador estará preparado para realizar la siguiente tarea.

Veamos cuatro clases de sentencias:

- **Sentencias de declaración**: establecen los nombres y el tipo de variables.
- **Sentencias de asignación**: asignan valores a las variables.
- **Sentencias de función**: llaman a funciones para realizar una determinada tarea.

- **Sentencias estructuradas o de control de flujo**: `while`, `if-else`, `switch`, etc. que se analizan en la sección siguiente.

Un *bloque* es un conjunto de dos o más sentencias agrupadas y encerradas entre llaves, no existiendo punto y coma tras la llave de cierre. Por ejemplo, el programa que sigue contiene un bloque.

```
for(i=0;i<100;i++) {
  mult[i] = 2*(i+1);
  System.out.println(''Elemento i '' + mult[i]);
}
```

3.10. Control de flujo

Una de las cualidades de los lenguajes de programación consiste en que se puede controlar el flujo del programa. Las sentencias se ejecutan ordenadamente, pero un bloque de código puede ejecutarse sólo bajo ciertas condiciones, o bien, hacerlo repetidas veces. Puede determinarse así el orden de ejecución de las sentencias según convenga. Para ello nos servimos de las *sentencias estructuradas* o *de control de flujo*.

3.10.1. Sentencia if-else

Se define primero la sentencia *if*, que tiene el formato:

```
if(expresion)
  {sentencias}
```

Actúa de la manera siguiente: si `expresion` tiene el valor `true`, se ejecuta la sentencia o bloque {`sentencias`}. Si es `false`, no se ejecuta el bloque {`sentencias`}.

Una alternativa más potente es la sentencia `if-else` cuyo formato es:

```
if(expresion)
  {sentencias1}
else
  {sentencias2}
```

Si `expresión` tiene el valor `true`, se ejecuta el bloque {`sentencias1`}; en caso contrario, se pasa a ejecutar la sentencia o bloque {`sentencias2`}.

Veamos un ejemplo:

```
...
if (a < b) {
  a = b * 2;
  System.out.println(''El resultado es '' + a);
  ...
}
```

Si como resultado de sentencias anteriores, `a` vale 3 y `b` vale 5, se ejecuta las sentencia de impresión y da como resultado `a = 10`. Si por el contrario, `a` fuera mayor que `b`, no se entraría a ejecutar el bloque y se pasaría a las siguientes sentencias del programa.

```
...
if (a < b) {
  a = b * 2;
} else  {
  a = b + 5;
  a++;
}
```

En este caso, si `a` es menor que `b`, se ejecuta `a = b * 2`; en caso contrario, es decir, si `a` es mayor o igual que `b`, se pasa a ejecutar el bloque

```
{ a = b + 5;
 a++;}
```

Práctica 3.1 (Cajero Automático). Tratando de diseñar el sistema informático de una entidad bancaria, se está realizando un subprograma para el manejo de los cajeros automáticos. La intención es crear un método llamado `retirar` que tenga las características siguientes:

1. Es un procedimiento de autorización para la retirada de dinero, de forma que lo único que ha de hacer es indicar si autoriza la retirada o no, en función de si el saldo disponible es suficiente. Por ello recibe como argumento local al método la `cantidad`.

2. En caso de autorizar se ha de escribir en pantalla que se autoriza la retirada de la cantidad solicitada, y actualizar el saldo actual del usuario, almacenado en una variable de instancia llamada `saldo`.

3. En caso de no autorizar la operación por falta de fondos se ha de escribir en pantalla que se no autoriza la retirada de la cantidad solicitada por falta de fondos, y escribir el saldo actual.

Nota: El cajero sólo puede dar billetes de 10, 20, y 50 euros.

Solución:

```
// La variable cantidad la defino como entera porque el cajero esta limitado
// a los billetes de 10, 20 y 50

public boolean retirar(int cantidad){
    boolean aut;
// La variable saldo es una variable de instancia accesible desde el metodo
    if (cantidad >= saldo) {
        System.out.println("La operacion esta autorizada");
        saldo += cantidad;
```

```
        aut = true;
        return(aut);
    } else {
        System.out.println("La operacion no se autoriza");
        System.out.println("Su saldo actual es de " + saldo + " euros");
        aut = false;
        return(aut);
    }
}
```

■

3.10.2. Sentencia else-if

La sentencia anterior solo permite elegir entre dos opciones. La sentencia `else-if` amplía el número de posibilidades. Su sintaxis es la siguiente:

```
if(expresion1)
  {sentencias1}
else if(expresion2)
  {sentencias2}
...
else
  {sentenciasN}
```

Las expresiones se evalúan en orden; si cualquier expresión es verdadera, se ejecuta la sentencia o el bloque asociado a ella, y se concluye la ejecución de la sentencia `else-if`. Solo se ejecuta el último `else` en el caso de que todas las expresiones `expresión1`, `expresión2`, ..., `expresión(N-1)` sean falsas. Veamos un ejemplo:

```
...
int b = 10;
int a;
if(a > 1) {
  a = b * 2;
}
else if(a < 2) {
  a = b + 5;
  b++;
}
else if(a != 3) {
  a += b;
}
else {
  a = b ^ 2;
}
```

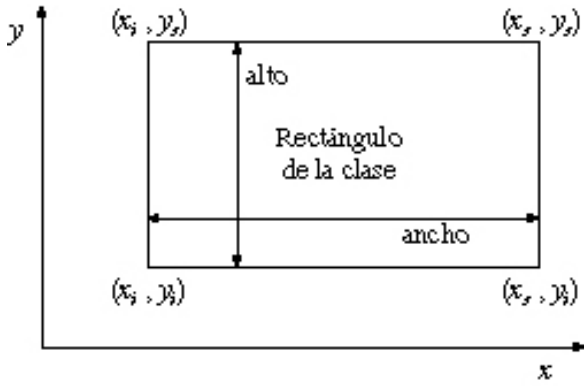
Si `a` cumple alguna de las tres primeras expresiones, se ejecuta la sentencia correspondiente; en caso contrario, se evaluará la última sentencia.

Pudiera ocurrir que fueran ciertas dos o más expresiones (tómese, por ejemplo, `a` igual a 1). En este caso, se ejecutarían sólo las sentencias correspondientes a la primera de ellas y se saldría del bloque `else-if`. (En nuestro caso, serían las sentencias correspondientes a la expresión `(a < 2)`).

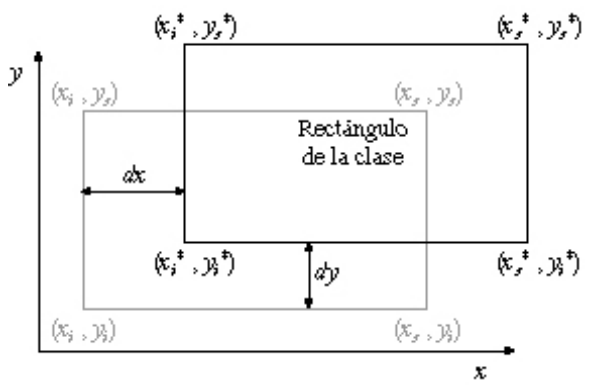
Práctica 3.2 (Manipulación de rectángulos e intersecciones). Para la realización del ejercicio pueden ser de utilidad los métodos de tipo `static` de la clase `Math`, `abs`, `min` y `max` que calculan el valor absoluto de un número, el valor máximo de entre dos números, y el valor mínimo de entre dos números, respectivamente.

Crear un clase que se llame **Rectangulo** y que tenga las características siguientes:

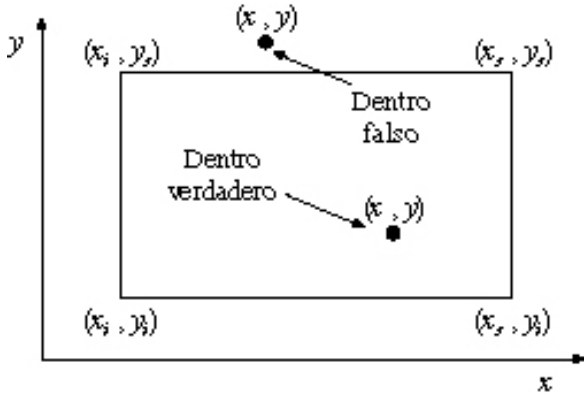
1. Pertenece al paquete **rectangulo**.
2. Se emplea para definir rectángulos paralelos a los ejes de coordenadas, por ello habrá cuatro variables de instancia x_i, x_s, y_i e y_s que se corresponden con las coordenadas $\boldsymbol{x}$ e $\boldsymbol{y}$ de las esquinas del rectángulo tal y como se muestra en la Figura 3.1(a). Pueden tomar cualquier valor real con la máxima precisión posible (tipo `double`).
3. Dispone de dos variables de instancia más para almacenar el ancho y alto del rectángulo.
4. El primer constructor no recibe ningún argumento y sólo reserva espacio en memoria para las variables de instancia. Hacer referencia al método `super()` de la clase de mayor jerarquía `Object`.
5. El segundo constructor recibe cuatro argumentos que se corresponden con las coordenadas de las esquinas, se llamarán `x1`, `x2`, `y1`, e `y2`. Se dan en ese orden y el problema es que tanto para las variables $\boldsymbol{x}$ como para las $\boldsymbol{y}$ no se sabe cual de las dos es mayor, y por tanto cómo asignar los valores a las variables de instancia de la clase. El constructor ha de tenerlo en cuenta. Se inicializarán los valores del `ancho` y el `alto` dentro del constructor.
6. El tercer constructor recibe las coordenadas x e y de la esquina inferior izquierda del rectángulo así como los valores del ancho y alto, éstos últimos sólo pueden tomar valores enteros. Se inicializarán los valores de todas las variables de instancia.
7. Tiene un método llamado **mover** que desplaza el rectángulo una cantidad `dx` a la derecha (si es positivo), y `dy` hacia arriba (si es positivo) (véase la Figura 3.1(b)).
8. Tiene un método llamado **dentro** que permite saber si un punto dado por sus coordenadas `x` e `y` está dentro del rectángulo (véase la Figura 3.1(c)).
9. Tiene un método llamado **menorrec** que dado un rectángulo cualquiera, permite obtener otro rectángulo que tiene una característica especial, es el menor rectángulo que contiene tanto al rectángulo de la clase como al que se ha dado como dato o argumento al método, tal y como se muestra en la Figura 3.1(d).



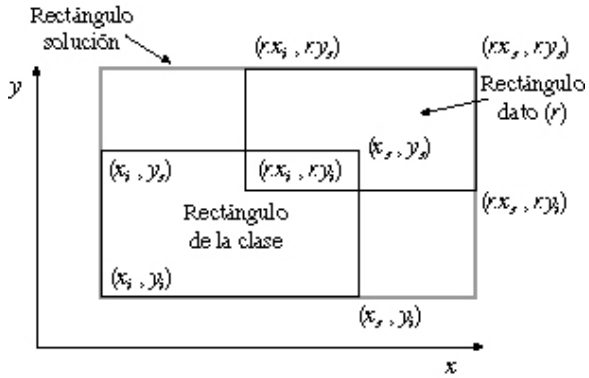
(a) Definición de la clase `Rectangulo`.



(b) Efecto que tiene el método `mover` en el rectángulo asociado a la clase.



(c) Comprobación de si un punto está dentro del rectángulo.



(d) Menor rectángulo circunscrito a dos dados.

Figura 3.1: Interpretación gráfica de la clase `Rectangulo`.

10. Por último contiene un método llamado **intersec**, que dado un rectángulo cualquiera devuelve el rectángulo intersección con el rectángulo asociado a la clase. En este caso la casuística es complicada, para facilitar el control de flujo en la Figura 3.2 se muestran los posibles casos para el cálculo de la intersección. Nótese que son mutuamente excluyentes.

Para poder emplear esta clase crear la clase **Principal** definida en el Comentario 2.2, que pertenezca al paquete **rectangulo**, y que contiene el método `main()`. En ella se realizan las siguientes acciones:

1. Crear una variable `cuadro1` de tipo **Rectangulo** con los datos (20.0,0.0,0.0,30.0).
2. Escribir las coordenadas en pantalla, así como su ancho y su alto.
3. Crear una variable `cuadro2` de tipo **Rectangulo** cuyas coordenadas de la esquina inferior izquierda sean (10.0,10.0) y tengan un ancho y un alto de 10 unidades exactas, respectivamente.
4. Escribir las coordenadas de rectángulo almacenado en la variable `cuadro2`.
5. Desplazar el rectángulo almacenado en la variable `cuadro1` 5 unidades a la derecha y 10 unidades hacia arriba.
6. Escribir las nuevas coordenadas de rectángulo tras su desplazamiento.
7. Comprobar si la esquina inferior izquierda del rectángulo almacenado en la variable `cuadro2` está dentro del rectángulo definido por la variable `cuadro1` y escribir un comentario según el caso.
8. Declarar e inicializar dos objetos nuevos de la clase `Rectangulo` en las variables `cuadro3` y `cuadro4`, en el momento de reservar espacio en memoria para los mismos ninguna de sus variables de instancia recibirá valores.
9. Almacenar en la variable `cuadro3` el menor rectángulo circunscrito a los rectángulos almacenados en las variables `cuadro1` y `cuadro2`.
10. Análogamente, almacenar en la variable `cuadro4` el menor rectángulo circunscrito a los rectángulos almacenados en las variables `cuadro1` y `cuadro2`, pero haciéndolo de forma diferente al apartado anterior.
11. Escribir en pantalla los resultados obtenidos tanto para `cuadro3` como para `cuadro4`. El resultado será correcto si ambos coinciden.
12. Declarar e inicializar dos objetos nuevos de la clase `Rectangulo` en las variables `cuadro5` y `cuadro6`, en el momento de reservar espacio en memoria para los mismos ninguna de sus variables de instancia recibirá valores.
13. Almacenar en la variable `cuadro5` la intersección de los rectángulos almacenados en las variables `cuadro1` y `cuadro2`.

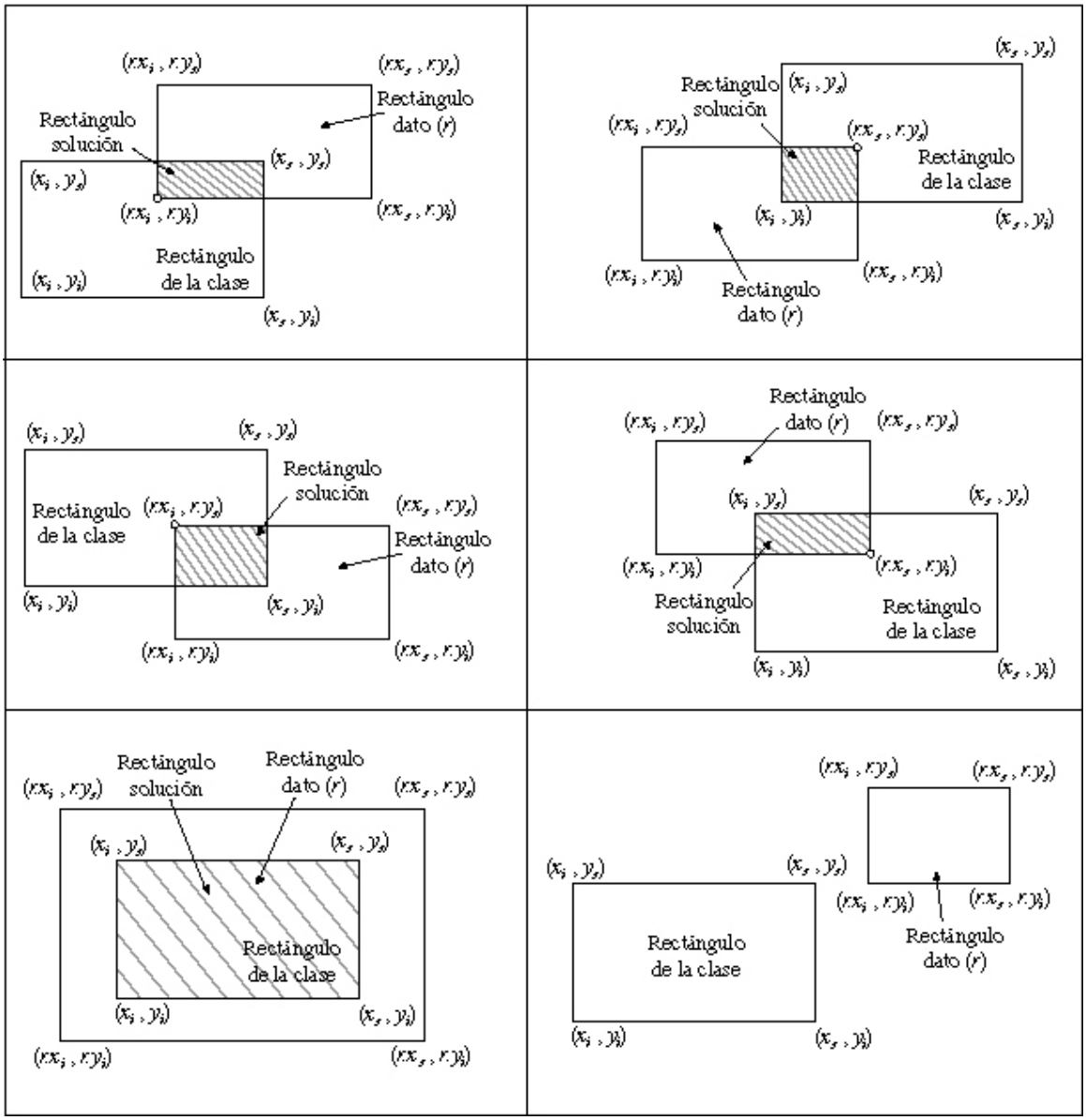


Figura 3.2: Posibles casos para el cálculo de la intersección de dos rectángulos.

14. Análogamente, almacenar en la variable `cuadro6` la intersección de los rectángulos almacenados en las variables `cuadro1` y `cuadro2`, pero haciéndolo de forma diferente al apartado anterior.

15. Escribir en pantalla los resultados obtenidos tanto para `cuadro5` como para `cuadro6`. El resultado será correcto si ambos coinciden.

Solución: La solución de todos los apartados se da a continuación. En primer lugar se describe la clase **Rectangulo** que habrá de ubicarse en un fichero llamado "Rectangulo.java":

```
package rectangulo;

public class Rectangulo {
  double xi,xs,yi,ys;
  double ancho,alto;

  public Rectangulo() {
    super();
  }

  public Rectangulo(double x1, double x2, double y1, double y2) {
    super();
    if(x1>x2){
      xs = x1;
      xi = x2;
    }else{
      xs = x2;
      xi = x1;
    }
    if(y1>y2){
      ys = y1;
      yi = y2;
    }else{
      ys = y2;
      yi = y1;
    }
    ancho=xs-xi;
    alto=ys-yi;
  }

   public Rectangulo(double x1, double y1, int ancho, int alto) {
      super();
      xi = x1;
      yi = y1;
      xs = xi+Math.abs(ancho);
      ys = yi+Math.abs(alto);
      this.ancho=Math.abs(ancho);
      this.alto=Math.abs(alto);
  }
```

```
public void mover(double dx,double dy){
  xi=xi+dx;
  xs+=dx;
  yi=yi+dy;
  ys+=dy;
}

public boolean dentro(double x, double y){
  boolean in=false;
  if((x>=xi)&&(x<=xs)&&(y>=yi)&&(y<=ys)){
    in=true;
  }
  return(in);
}

public Rectangulo menorrec(Rectangulo r) {
    Rectangulo menor=new Rectangulo();
    if(xi<=r.xi){
      menor.xi=xi;
    }else{
      menor.xi=r.xi;
    }
    if(yi<=r.yi){
      menor.yi=yi;
    }else{
      menor.yi=r.yi;
    }
    menor.xs=Math.max(xs,r.xs);
    menor.ys=Math.max(ys,r.ys);
    return(menor);
}

public Rectangulo intersec(Rectangulo r) {
      Rectangulo inter=new Rectangulo();
      if(this.dentro(r.xi, r.yi)){
        inter.xi=r.xi;
        inter.yi=r.yi;
        inter.xs=Math.min(xs,r.xs);
        inter.ys=Math.min(ys,r.ys);
        return(inter);
      }else if(this.dentro(r.xs, r.ys)){
        inter.xs=r.xs;
        inter.ys=r.ys;
        inter.xi=Math.max(xi,r.xi);
        inter.yi=Math.max(yi,r.yi);
        return(inter);
      }else if(this.dentro(r.xi, r.ys)){
        inter.xi=r.xi;
        inter.ys=r.ys;
        inter.xi=Math.min(xs,r.xs);
```

```
          inter.yi=Math.max(yi,r.yi);
          return(inter);
        }else if(this.dentro(r.xs, r.yi)){
          inter.xs=r.xs;
          inter.yi=r.yi;
          inter.xi=Math.max(xi,r.xi);
          inter.yi=Math.min(ys,r.ys);
          return(inter);
        }else if((r.xi<=xi)&&(r.xs>=xs)&&(r.yi<=yi)&&(r.ys>=ys)){
          inter=this;
          return(inter);
        }else{
          System.out.println("No existe interseccion");
          return(inter);
        }
    }
}
```

La clase principal queda de la siguiente manera:

```
package rectangulo;

public class Principal {
  public Principal() {
  }
  public static void main(String[] args) {
    Rectangulo cuadro1=new Rectangulo(0.0,20.0,0.0,30.0);
    System.out.println("Las coordenadas del cuadro1 son (" + cuadro1.xi+", "
           + cuadro1.yi + ") y (" + cuadro1.xs + ", " + cuadro1.ys + ")");
    System.out.println("El ancho del rectangulo es " + cuadro1.ancho);
    System.out.println("El alto del rectangulo es " + cuadro1.alto);
    Rectangulo cuadro2=new Rectangulo(10.0,10.0,10,10);
    System.out.println("Las coordenadas del cuadro2 son (" + cuadro2.xi + ", "
           + cuadro2.yi + ") y (" + cuadro2.xs + ", " + cuadro2.ys + ")");
    cuadro1.mover(5.0,10.0);
    System.out.println("Las nuevas coordenadas del cuadro1 son (" + cuadro1.xi
           + ", " + cuadro1.yi + ") y (" + cuadro1.xs + ", " + cuadro1.ys + ")");
    if(cuadro1.dentro(cuadro2.xi, cuadro2.yi)){
      System.out.println("La esquina inferior izquierda del cuadro 2 "
             + "esta dentro del cuadro1");
    }else{
      System.out.println("La esquina inferior izquierda del cuadro 2 "
             + "esta fuera del cuadro1");
    }
    Rectangulo cuadro3=new Rectangulo();
    Rectangulo cuadro4=new Rectangulo();
    cuadro3=cuadro1.menorrec(cuadro2);
    System.out.println("El cuadro 2 lo constituyen los puntos (" + cuadro2.xi
           + ", " + cuadro2.yi + ") y (" + cuadro2.xs + ", " + cuadro2.ys + ")");
    cuadro4=cuadro2.menorrec(cuadro1);
    System.out.println("El menor rectangulo que contiene a los cuadros 1 y 2 es");
```

```
    System.out.println("El cuadro 3 lo constituyen los puntos (" + cuadro3.xi
           + ", " + cuadro3.yi + ") y (" + cuadro3.xs + ", " + cuadro3.ys + ")");
    System.out.println("El cuadro 4 lo constituyen los puntos (" + cuadro4.xi
           + ", " + cuadro4.yi + ") y (" + cuadro4.xs + ", " + cuadro4.ys + ")");
    Rectangulo cuadro5=new Rectangulo();
    Rectangulo cuadro6=new Rectangulo();
    cuadro5=cuadro1.intersec(cuadro2);
    cuadro6=cuadro2.intersec(cuadro1);
    System.out.println("El rectangulo interseccion de los puntos 1 y 2 es");
    System.out.println("El cuadro 5 lo constituyen los puntos (" + cuadro5.xi
           + ", " + cuadro5.yi + ") y (" + cuadro5.xs + ", " + cuadro5.ys + ")");
    System.out.println("El cuadro 6 lo constituyen los puntos (" + cuadro6.xi
           + ", " + cuadro6.yi + ") y (" + cuadro6.xs + ", " + cuadro6.ys + ")");
  }
}
```

La salida del programa queda como:

```
Las coordenadas del cuadro1 son (0.0, 0.0) y (20.0, 30.0)
El ancho del rectangulo es 20.0
El alto del rectangulo es 30.0
Las coordenadas del cuadro2 son (10.0, 10.0) y (20.0, 20.0)
Las nuevas coordenadas del cuadro1 son (5.0, 10.0) y (25.0, 40.0)
La esquina inferior izquierda del cuadro 2 esta dentro del cuadro1
El cuadro 2 lo constituyen los puntos (10.0, 10.0) y (20.0, 20.0)
El menor rectangulo que contiene a los cuadros 1 y 2 es
El cuadro 3 lo constituyen los puntos (5.0, 10.0) y (25.0, 40.0)
El cuadro 4 lo constituyen los puntos (5.0, 10.0) y (25.0, 40.0)
El rectangulo interseccion de los puntos 1 y 2 es
El cuadro 5 lo constituyen los puntos (10.0, 10.0) y (20.0, 20.0)
El cuadro 6 lo constituyen los puntos (10.0, 10.0) y (20.0, 20.0)
```

■

3.10.3. Sentencia switch

En este caso, se dispone de varias opciones derivadas de la evaluación de una única expresión. La sintaxis es la siguiente:

```
switch(expresion) {
  case {expresion1-constante1}:
    {sentencias1}
  case {expresion2-constante2}:
    {sentencias2}
  ...
  case {expresionN-constanteN}:
    {sentenciasN}
  default {sentencias}
```

Cada `case` lleva una o más constantes enteras o expresiones constantes enteras. Si un `case` coincide con el valor de `expresion`, la ejecución comienza ahí. Todas las expresiones

de los `case` deben ser diferentes. Si ninguna de ellas coincide con `expresion` se ejecutan las sentencias correspondientes a `default`. El `default` es optativo; si no está y ninguno de los casos coincide, no se ejecuta ninguna acción.

Así, el bloque de código:

```
...
int b = 10;
int a;
if (a == 1) {
  a = b * 2;
}
else if (a == 2) {
  a = b + 5;
  b++;
}
else if (a == 3) {
  a += b;
} else {
  a = b ^ 2;
}
```

puede reescribirse de la forma siguiente:

```
...
int b = 10;
int a;

switch (a) {
  case 1:
    a = b * 2;
    break;
  case 2:
    a = b + 5;
    b++;
    break;
  case 3:
    a+ = b;
    break;
  default:
    a = b ^ 2;
    break;
}
```

Sin embargo, la sentencia `switch` es más eficiente.

Un punto a destacar es el uso de la sentencia `break`, la cual provoca una salida inmediata del `switch`. Tras ejecutar las sentencias correspondientes a un `case`, se pasa al siguiente `case` a menos que se fuerce a salir con la ayuda de la sentencia `break`.

El siguiente ejemplo puede ilustrar perfectamente el funcionamiento de las sentencias condicionales `if` y `switch` de Java. Se trata de un programa que recibe un número entero de a lo sumo seis dígitos, es decir en el rango comprendido entre 0 y 999999, y escribe en pantalla la expresión literal de ese número. Por ejemplo, la expresión literal del número 284513 es:

```
doscientos ochenta y cuatro mil quinientos trece
```

Programa 3.4 (Expresión literal de un número).
El programa tiene dos clases, una para la generación de la cadena de caracteres que caracteriza el número y otra principal que contiene el método `main()` en el que se introduce el número deseado y se llama al método correspondiente. Las dos clases pertenecen al paquete `expreliternumero`. La primera clase es `ConvertirLetras`:

```
package expreliternumero;

public class ConvertirLetras {

  public static String Convertir3Digitos(int n) {
    int unidades, decenas, centenas;
    String su, sd, sc, respuesta;
    unidades = n % 10;
    decenas = (n / 10) % 10;
    centenas = n / 100;
    if ( (decenas == 1) && (unidades > 0) && (unidades < 6)) {
      su = new String("");
      switch (unidades) {
        case 1:
          sd = new String("once");
          break;
        case 2:
          sd = new String("doce");
          break;
        case 3:
          sd = new String("trece");
          break;
        case 4:
          sd = new String("catorce");
          break;
        case 5:
          sd = new String("quince");
          break;
        default:
          sd = new String("");
      }
    }
    else {
      switch (unidades) {
        case 1:
          su = new String("uno");
```

```
        break;
      case 2:
        su = new String("dos");
        break;
      case 3:
        su = new String("tres");
        break;
      case 4:
        su = new String("cuatro");
        break;
      case 5:
        su = new String("cinco");
        break;
      case 6:
        su = new String("seis");
        break;
      case 7:
        su = new String("siete");
        break;
      case 8:
        su = new String("ocho");
        break;
      case 9:
        su = new String("nueve");
        break;
      default:
        su = new String("");
    }
    switch (decenas) {
      case 1:
        if (unidades == 0) sd = new String("diez");
        else sd = new String("dieci");
        break;
      case 2:
        if (unidades == 0) sd = new String("veinte");
        else sd = new String("veinti");
        break;
      case 3:
        if (unidades == 0) sd = new String("treinta");
        else sd = new String("treinta y ");
        break;
      case 4:
        if (unidades == 0) sd = new String("cuarenta");
        else sd = new String("cuarenta y ");
        break;
      case 5:
        if (unidades == 0) sd = new String("cincuenta");
        else sd = new String("cincuenta y ");
        break;
      case 6:
```

```
        if (unidades == 0) sd = new String("sesenta");
        else sd = new String("sesenta y ");
        break;
      case 7:
        if (unidades == 0) sd = new String("setenta");
        else sd = new String("setenta y ");
        break;
      case 8:
        if (unidades == 0) sd = new String("ochenta");
        else sd = new String("ochenta y ");
        break;
      case 9:
        if (unidades == 0) sd = new String("noventa");
        else sd = new String("noventa y ");
        break;
      default:
        sd = new String("");
    }
  }
  switch (centenas) {
    case 1:
      if ( (unidades == 0) && (decenas == 0))
        sc = new String("cien");
      else
        sc = new String("ciento ");
      break;
    case 2:
      sc = new String("doscientos ");
      break;
    case 3:
      sc = new String("trescientos ");
      break;
    case 4:
      sc = new String("cuatrocientos ");
      break;
    case 5:
      sc = new String("quinientos ");
      break;
    case 6:
      sc = new String("seiscientos ");
      break;
    case 7:
      sc = new String("setecientos ");
      break;
    case 8:
      sc = new String("ochocientos ");
      break;
    case 9:
      sc = new String("novecientos ");
      break;
```

```
      default:
        sc = new String("");
    }
    respuesta = new String(sc);
    respuesta = respuesta.concat(sd);
    respuesta = respuesta.concat(su);
    return (respuesta);
  }

  public static String ConvertirNumero(int numero) {
    String respuesta = new String();
    int nmiles = numero / 1000;
    if (nmiles > 999) {
      respuesta = "FUERA DE RANGO";
    }
    else if (nmiles == 0) {
      respuesta = Convertir3Digitos(numero);
    }
    else if (nmiles == 1) {
      respuesta = "mil ";
      respuesta = respuesta.concat(Convertir3Digitos(numero % 1000));
    }
    else {
      respuesta = Convertir3Digitos(nmiles);
      respuesta = respuesta.concat(" mil ");
      respuesta = respuesta.concat(Convertir3Digitos(numero % 1000));
    }
    return (respuesta);
  }

}
```

Mientras que la clase principal queda de la siguiente manera:

```
package expreliternumero;

public class Principal {

  public Principal() {
  }

  public static void main(String[] args) {

    int numero= 36879;
    System.out.println(ConvertirLetras.ConvertirNumero(numero));
  }
}
```

■

3.10.4. Ciclo while

La sentencia `while` tiene la estructura:

```
while(expresion)
  {sentencias}
```

Con ella se ejecuta el bloque {`sentencias`} mientras el valor de la expresión es `true`. El ciclo se repite hasta que el valor de `expresion` toma el valor `false`. Entonces, el compilador sale del ciclo.

Hay que tener cuidado de alterar adecuadamente alguno de los elementos que aparecen en `expresion` dentro del bloque {`sentencias`}. De no ser así, el programa no saldría nunca de este ciclo.

```
...
int a = 1;
while ( a != 15 ) {
  a *= 2;
  a = a + 1;
}
...
```

La variable `a` comienza tomando el valor 1, distinto de 15. Por tanto, se ejecutan las sentencias. Luego `a` vale 3, que sigue siendo distinto de 15, por lo que se vuelven a ejecutar las sentencias y se repite el proceso hasta llegar a tener `a = 15`. En este instante, se sale del ciclo `while` y se continúa con las sentencias siguientes del programa.

Práctica 3.3 (Búsqueda de un elemento en una lista). Crear un método estático que realize la búsqueda de un número entero en una matriz ordenada de números utilizando sentencias tipo `while`. Dicha búsqueda se basa en el algoritmo de la bisección, partiendo de los elementos que ocupan las posiciones extremas de la matriz se obtiene el elemento central y se determina en cuál de los dos submatrices debe buscarse el número. Repitiendo el proceso sucesivamente se llega a localizar ese número, si es que se encuentra. La función devuelve la posición en la que se encuentra el número dentro de la matriz; en caso de no encontrarse, devuelve -1.

Nota: La longitud de una lista almacenada en una variables cualquiera `x`, se obtiene mediante la llamada al método `length`, que en este caso se emplea como `x.length`, que devuelve un entero `int` con el valor de su longitud. Por otro lado el método `public static int floor(double a)` de la clase `Math` proporciona el mayor entero menor que el argumento `a`.

Solución:

En primer lugar se crea una clase con un método estático para realizar la búsqueda:

```
package buscar;

public class Buscar {
```

```
  public Buscar() {
  }

  public static int Buscar(int x[], int n) {
    int e1=0;
    int e2=x.length-1;
    int centro;
    if(x[e1]==n)
      return(e1+1);
    else if (x[e2]==n)
      return(e2+1);
    while (e1<e2) {
      centro=(int) Math.ceil((e1+e2)/2);
      if(centro==e1) {
        return(-1);
      } else if(x[centro]==n) {
        return(centro+1);
      } else if (x[centro]<n) {
        e1=centro;
      } else {
        e2=centro;
      }
    }
    return(-1);
  }
}
```

En segundo lugar se genera la clase principal con un ejemplo de búsqueda en una lista determinada:

```
package buscar;

public class Principal {
  public Principal() {
  }
  public static void main(String[] args) {
    int lista[] = {1, 4, 7, 14, 23, 56, 61, 92, 150};
    int numero = 23;
    int posicion;
    posicion = Buscar.Buscar(lista,numero);
    if (posicion == -1) {
      System.out.println("El numero " + numero + " no esta en la lista.");
    } else {
      System.out.println("El numero " + numero + " esta en la posicion " +
         posicion + ".");
    }
  }
}
```

La ejecución del programa genera la siguiente salida:

```
El numero 23 esta en la posicion 5.
```

■

Veamos a continuación otro ejemplo.

Práctica 3.4 (Conjetura de Collatz).

La *conjetura de Collatz* asegura que dada la función

$$F(n) = \begin{cases} \frac{n}{2} & \text{si } n \text{ es par} \\ \\ \frac{3n+1}{2} & \text{si } n \text{ es impar} \end{cases} \qquad \text{con } n \in \mathbf{Z}$$

la sucesión de valores n, $F(n)$, $F(F(n))$, $F(F(F(n)))$,... siempre llega a alcanzar el valor 1, independientemente del valor de partida n. El programa que se incluye a continuación puede servir para comprobar dicha conjetura. La clase `Collatz` tiene definidos dos métodos, el primero de ellos define la función $F(n)$ anterior mediante una sentencia condicional `if`. Por otro lado, se encuentra el método `main()`, se supone que el primer parámetro que reciba es el número n de partida, aunque se define un número por defecto para el caso en que no se den argumentos de entrada en el método main. Ese primer parámetro es almacenado automáticamente en la variable `args[0]`. Los argumentos que recibe el método `main()` de cualquier clase son tratados siempre como cadenas de caracteres y almacenados en una matriz de objetos de la clase `String`. Si se desea tratarlos de otra forma deben ser convertidos explícitamente. En este programa ese primer argumento debe ser convertido a valor entero. Una vez obtenido ese valor entero, comienza un ciclo `while` que finaliza cuando se alcanza el valor 1.

Solución:

```
package collatz;

public class Collatz {

   public static int F(int x) {
      if(x%2==1) {
         return (3*x+1)/2;
      } else {
         return x/2;
      }
   }

   public static void main(String args[]) {
      int n, i=1;
      if (args.length >= 1) {
        Integer argumento = new Integer(args[0]);
        n=argumento.intValue();
      } else {
        n=231;
      }
      while(n!=1) {
```

```
         System.out.println( "Iter. " + i + ":" + n );
         i++;
         n=F(n);
      }
      System.out.println("SE ALCANZA EL 1 TRAS "+ i +" ITERACIONES");
   }
}
```

■

La forma de pasar los argumentos al método `main()` de una clase depende del entorno y del compilador Java en el que se esté trabajando. En algunos casos, cuando se declara el método `main()` con argumentos aparece automáticamente un cuadro de diálogo para introducirlos. También puede utilizarse la utilidad `javai` que forma parte del entorno Java.

Por ejemplo, haciendo la llamada al método `main()` de la clase `Collatz` al mismo tiempo que se pasa el valor inicial $n = 56$, se obtiene un resultado como el que se muestra a continuación:

```
Iter. 1:56
Iter. 2:28
Iter. 3:14
Iter. 4:7
Iter. 5:11
Iter. 6:17
Iter. 7:26
Iter. 8:13
Iter. 9:20
Iter. 10:10
Iter. 11:5
Iter. 12:8
Iter. 13:4
Iter. 14:2
SE ALCANZA EL 1 TRAS 15 ITERACIONES
```

Práctica 3.5 (Comprobador de números primos).
A continuación se presenta un ejemplo muy simple de programa Java que permite reconocer si un número entero es primo o no. El algoritmo es por todos conocido, se trata de comprobar si el número tiene algún divisor distinto de la unidad y del propio número. El único aspecto de la implementación que merece la pena destacar es la forma de pasar el número como argumento del método `main()`.

Solución:

```
package primo;

public class Primo {
```

```
  public static void main(String args[]) {
    long n, i=2;
    boolean fin=false;
    if (args.length>=1) {
      Integer argumento = new Integer(args[0]);
      n = argumento.intValue();
    } else {
      n = 231;
    }
    while ((i<n)&&(!fin)) {
      if(n%i==0) {
        fin = true;
      } else {
        i++;
      }
    }
    if (fin) {
      System.out.println("EL NUMERO " + n + " NO ES PRIMO");
    } else {
      System.out.println("EL NUMERO " + n + " ES PRIMO");
    }
  }
}
```

■

3.10.5. Ciclo do-while

La estructura de la sentencia `do-while` es:

```
do{
  {sentencias}
}while({expresion});
```

Mientras en los ciclos `while` y `for`, que se verá en la próxima sección, las condiciones de término se evalúan al principio de cada iteración, en el ciclo `do-while` se comprueban al final, después de cada repetición del ciclo, por lo que el bloque {`sentencias`} se ejecuta siempre, al menos una vez.

Así, primero se ejecutan las sentencias del bloque `sentencias` y después se evalúa `expresion`. Si es verdadera, se vuelve a ejecutar el bloque `sentencias`, y así sucesivamente. Cuando `expresion` toma el valor `false` se acaba el ciclo.

El ciclo `do-while` es el menos utilizado. A continuación, veamos un ejemplo:

El bloque `while`:

```
int a = 1;
while ( a != 15 ) {
  a = a + 1;
}
```

equivale al bloque con `do-while`:

```
int a = 1;
do {
  a = a + 1;
} while (a != 14);
```

Es decir, se produce el mismo efecto en ambos casos.

Práctica 3.6 (Cuenta atrás). Generar un programa java que cuente hacia atrás desde el número que el usuario introduce en la línea de mensajes.

Solución:

```
public class contar{

  public static void main(String args[]){

     Integer argumento=new Integer(args[0]);
     int count=argumento.intValue();
     do  {
        System.out.println(count + " ");
        count = count - 1; //o  count--;
     } while (count > 0) ;
   }
}
```

■

3.10.6. Ciclo for

Este ciclo actúa hasta un cierto límite impuesto por el programador. Su sintaxis es la siguiente:

```
for({iniciacion} ;{limite} ; {incremento})
  {sentencias}
```

El ciclo `for` es equivalente a:

```
{iniciacion}
while({limite}) {
  {sentencias}
  {incremento}
}
```

Las tres componentes del ciclo `for` son expresiones; `iniciacion` e `incremento` suelen ser asignaciones o llamadas a una función, y `limite`, una expresión relacional.

- La expresión `iniciacion` se ejecuta al comienzo del bucle.

- La expresión `limite` indica cuándo se ejecuta el bucle y se evalúa al comienzo de cada iteración, de manera que cuando su valor es `true` se ejecuta el bloque {`sentencia`} y si es `false`, el ciclo `for` concluye.
- Las sentencias del bloque `sentencias` se ejecutan en cada ciclo.
- Las sentencias del bloque `incremento` se ejecutan al final de cada iteración o ciclo.

Cualquiera de esta expresiones se puede omitir.
El ciclo `for` es frecuente en la iniciación de los elementos de un matriz:

```
int mult[];
mult = new int[100];
for(int i=0;i<100;i++) {
  mult[i] = 2*(i+1);
}
```

Práctica 3.7 (Escritura de números primos de 2 a 1000). Escribir todos los números primos entre 2 y 1000 usando el ciclo for, suponiendo que tenemos un método tipo `static` llamado `check(long a)` de la clase `Primo` que devuelve una variable booleana con valor verdadero si el número es primo y falso si no lo es.

Solución:

```
public class primos {

public static void main(String args[]){
  int STOP = 1000;
  int number = 2;
  boolean isPrime;

  for (int i=number;i<=STOP;i++) {
     isPrime = Primo.check((long) i);
     if (isPrime) {
        System.out.println(i);
     }
  }
}
```

■

Práctica 3.8 (Serie de Fibonacci). Utilizando el ciclo for, construir un programa que genere la sucesión de Fibonacci

$$1, 1, 2, 3, 5, 8, \ldots$$

cuyos términos se obtienen como suma de los dos anteriores.

Solución:

```
public class Fibonacci {
  public static void main(String[] args) {
    int xn, x0 = 0, x1 = 1;
    int limite = 20
    for(int i = 1; i <= limite; i++) {
      xn = x0 + x1;
      System.out.print(xn + " ");
      x0 = x1;
      x1 = xn;
    }
    System.out.println();
  }
}
```

■

Práctica 3.9 (U). tilizando el ciclo for, construir una aplicación que vaya escribiendo en la consola Java los argumentos introducidos en la línea de comando. Y otra aplicación que los escriba en el sentido opuesto al que se escribieron.

Solución:

En primer lugar la escritura normal:

```
public class Escribir {
  public static void main(String[] args) {
    int i = 0;
    for(i =0;i< args.length;i++) {
      System.out.print(args[i] + " ");
      i++;
    }
    System.out.println();
  }
}
```

Y en segundo lugar la inversa:

```
public class Reverse {
  public static void main(String[] args) {
     for(int i = args.length-1; i >= 0; i--){
        for(int j=args[i].length()-1;j>=0;j--){
            System.out.print(args[i].charAt(j));
        }
      System.out.print(" ");
    }
    System.out.println();
  }
}
```

■

Práctica 3.10 (Generación de pirámides mediante de bucles anidados). Como ejemplo de la utilización de los ciclos `for` de Java y de la posibilidad de anidamiento de

unos ciclos dentro de otros, se presenta a continuación un programa muy simple que permite generar pirámides de dígitos como la que puede verse a continuación.

```
1
232
34543
4567654
567898765
67890109876
7890123210987
890123454321098
90123456765432109
0123456789876543210
```

Esta pirámide tiene una altura de 10 filas, sin embargo, pueden construirse pirámides de otras alturas sin más que pasarlas como argumento al método `main()` de la clase `Piramide`, en caso de no hacerlo, la altura por defecto es 10.

Solución:

```
package piramide;

public class Piramide {

  public static void main(String args[]) {
    int i, j, k, m;
    int altura;
    if (args.length >= 1) {
      Integer argumento = new Integer(args[0]);
      altura = argumento.intValue();
    }
    else {
      altura = 10;
    }
    for(i=1,m=1;i<=altura;i++,m+=2) {
      for (j=1;j<=altura-i;j++) {
        System.out.print(" ");
      }
      for (k=i;k<=m;k++) {
        System.out.print(k % 10);
      }
      for (j=m-1;j>=i;j--) {
        System.out.print(j % 10);
      }
      System.out.println("");
    }
  }
}
```

■

Práctica 3.11 (Multiplicación de matrices). Generar un programa Java con las siguientes clases:

- La clase `Matriz` con dos constructores diferentes, un método `Mostrar()` para visualizar la matriz en la pantalla, y un método `Multiplicar` que permita la multiplicación matricial de dos matrices:
$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}.$$
- La clase `Principal` con el método `main()` en el que se definen tres matrices, las dos primeras son
$$A = \begin{pmatrix} 2 & 1 & 0 \\ -1 & 3 & 2 \\ 4 & 1 & -1 \end{pmatrix} \quad \text{y} \quad B = \begin{pmatrix} 2 & 3 \\ 5 & 4 \\ 1 & 2 \end{pmatrix}$$
y la tercera es el resultado de multiplicar las anteriores, es decir:
$$C = \begin{pmatrix} 9 & 10 \\ 15 & 13 \\ 12 & 14 \end{pmatrix}$$

Solución:

La clase matriz que implementa el método mostrar y producto se lista a continuación:

```
package matriz;

public class Matriz {
  int nfilas, ncolumnas;
  float x[][];

  public Matriz(float m[][]) {
    nfilas=m.length;
    ncolumnas=m[0].length;
    x=m;
  }

  public Matriz(int n, int m) {
    nfilas=n;
    ncolumnas=m;
    x=new float[n][m];
  }

  public void Mostrar() {
    int i,j;
    for(i=0;i<nfilas;i++) {
      for(j=0;j<ncolumnas;j++)
        System.out.print(x[i][j]+"  ");
      System.out.println("");
      }
```

```
  }

  public static Matriz Multiplicar (Matriz A, Matriz B) {
    int i,j,k;
    float suma;
    Matriz C= new Matriz (A.nfilas, B.ncolumnas);
    for(i=0;i<A.nfilas;i++)
      for(j=0;j<B.ncolumnas;j++)
      {
        suma=0;
        for(k=0;k<A.ncolumnas;k++)
          suma+=A.x[i][k]*B.x[k][j];
        C.x[i][j]=suma;
      }
    return(C);
  }
}
```

Mientras que la clase `Principal` queda de la siguiente manera:

```
package matriz;

public class Principal {
  public Principal() {
  }

  public static void main(String args[]) {
    float x[][]={{2,1,0},{-1,3,2},{4,1,-1}};
    float y[][]={{2,3},{5,4},{1,2}};
    Matriz A=new Matriz(x);
    Matriz B=new Matriz(y);
    Matriz C;
    System.out.println("Producto:");
    C=Matriz.Multiplicar(A,B);
    C.Mostrar();
  }
}
```

De forma que la salida del programa queda como:

```
Producto:
9.0  10.0
15.0  13.0
12.0  14.0
```

■

Práctica 3.12 (Rotaciones de palabras). El ejemplo que sigue muestra un sencillo programa Java que permite obtener todas las rotaciones de una palabra; por ejemplo, dada la palabra `Bienvenido`, sus rotaciones serían las palabras:

```
ienvenidoB  envenidoBi  nvenidoBie  venidoBien  enidoBienv
nidoBienve  idoBienven  doBienveni  oBienvenid  Bienvenido
```

El programa recibe la palabra a rotar en los argumentos de su método `main()` y obtiene la matriz de caracteres asociada con la llamada al método `toCharArray()` de la clase `String`. A partir de ahí se entra en un ciclo, de manera que en cada iteración el primer caracter es pasado al último lugar y los siguientes adelantan una posición.

Solución:

```
public class Rotaciones {

  public static void main(String args[]) {
    if (args.length>0) {
      int i,j,longitud=args[0].length();
      char cadena[] = args[0].toCharArray();
      char temp;
      for (j=0;j<longitud;j++)
      {
        temp=cadena[0];
        for (i=1;i<longitud;i++) {
          cadena[i-1]=cadena[i];
        }
        cadena[longitud-1]=temp;
        System.out.println(cadena);
      }
    }
  }
}
```

■

Práctica 3.13 (Simulador de pago). El programa que se presenta a continuación utiliza la función `random()` de la clase `Math` para generar aleatoriamente números entre 0 y 1. Si esos números son multiplicados por 50 y convertidos los resultados a tipo `int` se obtienen números enteros aleatorios en el rango comprendido entre 0 y 50.

En una matriz bidimensional se almacenan los valores de las distintas monedas y billetes de curso legal en España junto con números aleatorios entre 0 y 50 que indican las disponibilidades de cada uno de los tipos de monedas y billetes. Tras mostrar en pantalla toda esa información, se entra en un ciclo `while` en el que se generan, también de forma aleatoria, cantidades a pagar entre 0 y 50000 y se indica la forma de hacerlo con las monedas y billetes disponibles. El ciclo termina cuando una cantidad no puede ser abonada íntegramente.

Solución:

```
package dinero;

import java.util.*;

public class Dinero {
  public static void main(String args[]) {
    int i,Total=0, pago, resta, m;
    boolean disponible=true;
```

```
    int x[][] = new int[12][2];
    for (i=0;i<12;i++) {
      x[i][0]=1+5*i;
      x[i][1]=(int)(50*Math.random());
    }
    for(i=0;i<12;i++)
      Total+=x[i][0]*x[i][1];
    System.out.println("Cantidad total disponible: "+Total+" pts");
    for(i=0;i<8;i++)
      System.out.println(x[i][1]+" monedas de "+x[i][0]+" pts");
    for(i=8;i<12;i++)
      System.out.println(x[i][1]+" billetes de "+x[i][0]+" pts");
    while(disponible) {
      pago=(int)(50000*Math.random());
      resta=pago;
      System.out.println("Pago de "+pago+" pts =");
      i=11;
      while ((resta!=0)&&(i>=0)) {
        if(resta>=x[i][0])
        {
          m=Math.min(resta/x[i][0],x[i][1]);
          if (m!=0)
          {
            resta-=m*x[i][0];
            x[i][1]-=m;
            System.out.print(" "+m+"*"+x[i][0]);
          }
        }
        i--;
      }
      if (resta!=0)
      {
        System.out.println(" Faltan "+resta+" pts.");
        disponible=false;
      }
      else
        System.out.println("");
    }
  }
}
```

La ejecución de este programa produciría una salida similar a la siguiente:

```
Cantidad total disponible: 509612 pts
12 monedas de 1 pts
22 monedas de 5 pts
49 monedas de 10 pts
20 monedas de 25 pts
2 monedas de 50 pts
2 monedas de 100 pts
```

```
46 monedas de 200 pts
0 monedas de 500 pts
11 billetes de 1000 pts
44 billetes de 2000 pts
0 billetes de 5000 pts
40 billetes de 10000 pts
Pago de 35950 pts =
 3*10000 2*2000 1*1000 4*200 1*100 1*50
Pago de 29966 pts =
 2*10000 4*2000 1*1000 4*200 1*100 1*50 1*10 1*5 1*1
Pago de 16818 pts =
 1*10000 3*2000 4*200 1*10 1*5 3*1
Pago de 31773 pts =
 3*10000 1*1000 3*200 6*25 2*10 3*1
Pago de 8089 pts =
 4*2000 3*25 1*10 4*1
Pago de 2929 pts =
 1*2000 4*200 5*25 1*1 Faltan 3 pts.
```

■

3.10.7. Sentencias para manejar excepciones

Cuando se produce un error, el método puede lanzar un mensaje de excepción para avisar del mismo. Esto se consigue con la sentencia `throw`. El método también puede usar las sentencias `try`, `catch` y `finally` para manejar la excepción. Las excepciones se tratan en detalle en [1].

3.10.8. Sentencias de bifurcación

La sentencia `break` obliga al programa a pasar al siguiente ciclo. Otra forma de usar `break` es la siguiente.

Supóngase que en el programa se tiene la sentencia siguiente:

```
breakHere: {sentencia}
```

Para pasar directamente a esta línea del programa basta con escribir:

```
break breakHere;
```

Por otra parte, la sentencia `continue` se usa dentro de un ciclo para iniciar la siguiente iteración del ciclo que la contiene. De esta manera se evitan ejecutar las sentencias del ciclo siguientes a `continue`. Por ejemplo, con las sentencias:

```
for (i = 0; i < n; i++) {
  if(a[i] < 0)
  continue;
    ...
}
```

se ignoran los valores negativos.

La sentencia `return` finaliza la ejecución de la función que la contiene y devuelve el control a la sentencia siguiente de la función de llamada. Además, puede devolver un valor. En este último caso, la sintaxis es:

```
return expresion;
```

3.11. El método main

El método `main()` es el método de arranque de toda *aplicación* en lenguaje Java. El nombre se hereda del lenguaje C, en el que la existencia del bloque de código `main()` es forzosa.

De hecho, puede ejecutarse una clase cualquiera sin más que incluir en ella el método `main()`. Cuando se trata de aplicaciones, bastará implementarlo en una de las clases (la que se arranca), y no hay que incluirlo en el resto de las clases definidas en el programa.

El método `main()` es la herramienta de que dispone el lenguaje Java para decirle al ordenador qué hacer tras arrancar una aplicación o clase. Por ello, es llamado automáticamente por el entorno Java al arrancar la clase o aplicación, controla el flujo del programa desde su comienzo, asigna los recursos que se necesiten y se encarga de llamar a cualquier otro método que se requiera.

La forma de declarar el método `main()` es la siguiente:

```
public static void main(String args[]) {}
```

- `public` : puede ser llamado por cualquier objeto.
- `static` : es un método de clase, es decir, está asociado más bien a la clase que a un ejemplar de la clase.
- `void` : no devuelve ningún valor.
- `args[]` : es un objeto de la clase `String` en el que el usuario pasa la información a la clase.

Deben tenerse en cuenta las siguientes observaciones:

- El método `main()` es un método de clase, por lo que si se quieren invocar métodos de la clase, se debe crear un ejemplar de la misma.
- Si se diseña una interfase de usuario, se debe crear un espacio y una ventana donde colocar el `applet` creado.

A continuación se muestra un ejemplo:

Programa 3.5 (Ejemplo del uso del método main).

```
public class Checkboxes extends Applet {

    Checkbox a, b, c;
    Checkbox x, y, z;
    CheckboxGroup group;

    public void init() {
      a = new Checkbox();
      b = new Checkbox(''B'');
      c = new Checkbox(''C'');
      group = new CheckboxGroup();
      x = new Checkbox(''X'', group, true);
      y = new Checkbox(''Y'', group, false);
      z = new Checkbox(''Z'', group, false);

      setLayout(new GridLayout(3,2));
      add(a);
      add(x);
      add(b);
      add(y);
      add(c);
      add(z);
      validate();
    }

    public static void main() {
    // Se crea una ventana
      Frame ventana = new Frame (''Checkboxes'');
    // Se crea un ejemplar de la clase }
      Checkboxes checkboxes = new Checkboxes();
    // Se invoca un m\'etodo de la clase }
      checkboxes.init();
    // Se a\~{n}ade el applet a la ventana }
      ventana.add(''Center'',checkboxes);
      ventana.show();
    }
}
```

■

3.11.1. Llamada al método main()

Cuando el entorno Java ejecuta la clase o aplicación, lo primero que hace es llamar al método `main()`, el cual invoca al resto de métodos necesarios para hacer funcionar la aplicación.

Si la clase a compilar no tiene implementado un método `main()`, el proceso se detiene y se envía el mensaje de error:

```
In class NombreClase : void main(String args[]) is not defined
```

donde `NombreClase` es el nombre de la clase que genera el error.

3.11.2. Argumentos de la línea de comandos

Los programas pueden necesitar información para poder ejecutarse según los deseos del usuario. Una de las herramientas que lo permiten son los argumentos de la línea de comandos. Estos argumentos son caracteres o cadenas de caracteres que el usuario teclea y que permiten modificar el comportamiento de la aplicación sin necesidad de recompilarla.

El método `main()`:

`public static void main(String args[])` acepta como único argumento una matriz de elementos de la clase `String`, que se llaman argumentos de la línea de comandos, puesto que se les puede dar valores al comienzo de la ejecución, mediante un cuadro de diálogo que se muestra al usuario al ejecutar el programa Java o en el comando de arranque de la aplicación (entornos UNIX). Mediante estos argumentos se puede enviar información a la aplicación.

Por tanto, el programa Java debe ser capaz de leer los argumentos y programarlos. El proceso es el siguiente:

Al ejecutar una aplicación, el sistema pasa los argumentos tecleados por el usuario al método `main` de la aplicación, mediante una matriz de objetos de tipo `String`. Cada uno de los argumentos es uno de los elementos de la matriz.

Programa 3.6 (Ejemplo de un programa Java con argumentos).

```
import java.awt.*;

public class Imprimir {
  public static void main(String args[]) {
    for(int i=0; i<args.length; i++) {
      System.out.println(args[i]);
    }
  }
}
```

■

Tras compilar el fichero donde se define esta clase e iniciar la ejecución del programa en algunos entornos como Macintosh, se muestra al usuario un cuadro de diálogo, mediante el cual el usuario introduce los argumentos de la línea de comandos, que pasan a ser los argumentos del método `main`, es decir, los componentes de la matriz `args`.

A continuación se muestran dos ejemplos donde se manipulan los argumentos:

Programa 3.7 (Ejemplo de un programa Java con argumentos).

```
import java.awt.*;

public class Imprimir {
  public void algoritmo1() {
      System.out.println(''Se ejecuta el algoritmo 1'');
  }
```

```
  public void algoritmo2() {
      System.out.println(''Se ejecuta el algoritmo 2'');
  }

  public static void main(String args[]) {
      Imprimir imprimir = new Imprimir();
      for(int i=0; i<args.length; i++) {
        if (args[i].equals(''-alg1'')) {
          imprimir.algoritmo1();
        } else if (args[i].equals(''-alg2'')) {
          imprimir.algoritmo2();
        } else {
        }
      }
  }
}
```

■

Si el usuario introduce como único argumento `-alg1`, el programa ejecuta el método `algoritmo1`.

Si se introduce el argumento `-alg2` se ejecuta el método `algoritmo2`.

En el caso de que se introduzcan ambos argumentos, se ejecutarían los dos algoritmos de la aplicación.

Al introducir cualquier otro argumento, la aplicación no realiza ninguna acción.

Práctica 3.14 (Cambio de la base de numeración). El programa siguiente permite obtener la expresión de un número entero en cualquier base de numeración comprendida entre 2 y 16. El resultado es guardado en una cadena de caracteres que puede estar formada por dígitos y las letras `A`, `B`, `C`, `D`, `E` y `F` dependiendo del número entero y la base elegida.

En el programa se define la clase `CambioBase` con dos métodos:

- El método `digito()` permite obtener el dígito asociado a un número entre 0 y 16 pero tratado como un carácter.
- El método `main()`, en el que se encuentra implementado el algoritmo, recibirá dos argumentos: el número entero y la base de numeración. El algoritmo consta de un bucle `while` en el que se van calculando los sucesivos restos de dividir el número entre la base y obtener de esta forma los diferentes dígitos. Finalmente, es preciso invertir la cadena de caracteres resultante antes de presentarla en el monitor.

La forma de pasar los argumentos al método `main()` de una clase depende del entorno y del compilador Java sobre el que se esté trabajando. En algunos casos, cuando se declara el método `main()` con argumentos, aparece automáticamente un cuadro de diálogo para introducirlos. También puede utilizarse la utilidad `javai` que forma parte del entorno de desarrollo Java. Una vez pasados los argumentos, son tratados como cadenas de caracteres y almacenados en una matriz de objetos de la clase `String`, el primero será `args[0]`,

el segundo `args[1]`, y así sucesivamente. Se puede saber cuántos argumentos han sido pasados al método `main()` por el valor de la expresión `args.length`. Cuando se quieren tratar los argumentos recibidos no como cadenas de caracteres sino de otra forma, deben ser convertidos explícitamente. En este programa, los dos argumentos representan números enteros y son convertidos a ese tipo con el constructor de la clase `Integer` y el método `intValue()` de esa misma clase.

Solución:

```
import Java.io.*;

public class CambioBase {

  public static char digito(int c) {
    char a;
    if(c<10) {
      a=(char)('0'+c);
    } else {
      switch (c) {
        case 10:  a='A'; break;
        case 11:  a='B'; break;
        case 12:  a='C'; break;
        case 13:  a='D'; break;
        case 14:  a='E'; break;
        case 15:  a='F'; break;
        default:  a='X';
      }
    }
    return(a);
  }

  public static void main(String args[]) {
    int num, base, c, i=0, j, len;
    char caux;
    if(args.length>=2) {
      Integer argumento1 = new Integer(args[0]);
      Integer argumento2 = new Integer(args[1]);
      char resultado[] = new char[50];
      num = argumento1.intValue();
      base = argumento2.intValue();
      while (num/base!=0) {
        c=num%base;
        resultado[i++] = digito(c);
        num = num/base;
      }
      resultado[i] = digito(num);
      len = i+1;
      for(i=0,j=len-1;i<len/2;i++,j--) {
        caux=resultado[j];
        resultado[j]=resultado[i];
```

```
        resultado[i]=caux;
      }
      System.out.println( argumento1 + " en base " +
                          argumento2 + " = " + resultado);
    }
    else
      System.out.println(''Datos incompletos'');
  }
}
```

■

Convenios para los argumentos de la línea de comandos

Existen tres tipos de argumentos de la línea de comandos, siguiendo el convenio UNIX:

- **Palabras (opciones)**: `arg1`, `−algoritmo`. Pueden usarse como se ha visto en el ejemplo anterior.

  ```
  if (args[i].equals("alg1"))
     imprimir.algoritmo1();
  ```

- **Argumentos que requieren, a su vez, otros argumentos**. Por ejemplo, el argumento `−resultado` puede facilitar la impresión del resultado final de la aplicación en algún fichero que habrá que especificar. Normalmente, siempre que un argumento necesite más información, ésta viene dada por el siguiente argumento.

- **"Flags"**: son argumentos de un solo carácter, que modifican el comportamiento del programa. Se pueden escribir separadamente en cualquier orden: `−a −b` o concatenados:`−ab`.

Además deben seguirse las reglas siguientes para el uso de estos argumentos:

- El carácter − precede a las opciones, "flags" o series de "flags".
- Los argumentos se pueden dar en cualquier orden, excepto en el caso de que necesiten otros argumentos.
- Los "flags" pueden enumerarse de cualquier manera, con o sin separación.
- Los nombres de ficheros se escriben generalmente al final.
- El programa muestra un mensaje de error si no reconoce algún argumento.

Conviene tener también en cuenta que el sistema falla cuando se tiene un número considerable de argumentos.

3.12. Los Paquetes estándar de Java

Las clases e interfases implementadas en Java están agrupadas en diferentes paquetes, los cuales se listan a continuación:

- **Paquete java.lang**: Se trata del paquete que contiene las clases básicas. Al contrario que el resto de paquetes, para ser utilizado no necesita importarse, ya que Java lo importa automáticamente.
- **Paquete java.io**: Se trata de un paquete que permite la lectura y escritura de datos procedentes de diferentes fuentes, tales como ficheros, cadenas de caracteres, memoria, matrices de octetos, etc., de forma unificada.
- **Paquete java.util**: Es un paquete que contiene clases varias, tales como clases que tratan estructuras de datos genéricas, tratamiento de bits, consulta del tiempo, trabajo con fechas en diferentes formatos, manipulación de cadenas de caracteres, generación de números aleatorios, acceso a las propiedades del sistema, etc.
- **Paquete java.net**: Es el paquete que soporta todo lo relacionado con la red, incluyendo los `URLs`, `TCP` *sockets*, `UDP` *sockets*, direcciones `IPC` y un convertidor de binario a texto y viceversa.
- **Paquete java.awt**: Es el paquete que suministra la interfase de comunicación con el usuario, que se llama (`Abstract Window Toolkit`). De ahí su extensión `awt`. Incluye los elementos básicos, tales como ventanas, cuadros de diálogo, botones, marcas, listas, menús, barras de deslizamiento y campos de texto.
- **Paquete java.awt.image**: Paquete que sirve para tratar los datos de imágenes. Permite fijar el modelo de color, cropping, filtrado del color, asignación de puntos y grabbing snapshots.
- **Paquete java.awt.peer**: Es el paquete que conecta las componentes a sus correspondientes implementaciones independientes de la plataforma.
- **Paquete java.applet**: Paquete que permite la construcción de pequeños programas o `applets`. También suministra información sobre un documento de referencia de `applets`, sobre otros `applets` en dicho documento y permite a un `applet` emitir sonidos.

3.13. Applets y aplicaciones

El lenguaje Java está diseñado para desarrollar tanto aplicaciones como pequeños programas, llamados `applets`. Un `applet` sólo puede ejecutarse mediante un programa auxiliar, ya sea éste un navegador, como Netscape Navigator, o un visor de applets, como AppletViewer. Por el contrario, una aplicación puede ejecutarse independientemente. A continuación se muestran dos ejemplos, uno de applet y otro de aplicación, y se comentan las diferencias entre ambos.

3.13.1. Applets

La implementación de un Applet en una página web se hace mediante el comando <APPLET> siendo su sistaxis mínima:

```
<APPLET CODE=MiApplet.class
        WIDTH=n HEIGHT=m>
</APPLET>
```

Donde:

- El valor del parámetro CODE es el fichero donde se encuentra definido el applet.
- Los parámetros WIDTH y HEIGHT determinan las dimensiones en puntos de la región reservada para el applet en la página Web.

La sintaxis completa del comando es la siguiente:

```
<APPLET CODEBASE=Ejemplos/Applets
        CODE=MiApplet.class
        WIDTH=n HEIGHT=m
        ALT="Texto alternativo"
        NAME=nombre
        ALIGN=TipoAlineamiento
        VSPACE=v HSPACE=h>
<PARAM NAME=NombreParam1 VALUE=valor1>
<PARAM NAME=NombreParam2 VALUE=valor2>
<PARAM NAME=NombreParam3 VALUE=valor3>
.....................................
</APPLET>
```

donde

- CODEBASE indica el URL absoluto o relativo del fichero en el que se define el applet, cuando éste se encuentra en un directorio diferente al que contiene el documento HTML. Es posible también integrar applets definidos en máquinas remotas.
- El comando <PARAM> se utiliza para pasar argumentos a los applets.
- ALT permite indicar un texto alternativo a mostrar si el navegador utilizado no es compatible con Java.
- NAME da un nombre al applet.
- ALIGN indica el tipo de alineamiento con respecto a la página.
- VSPACE y HSPACE indican las distancias en pixels entre la región reservada para el applet y el resto de la página.
- Cualquier texto o código HTML que se encuentre entre los comandos <APPLET> y </APPLET> y que no sea un comando <PARAM> se entiende como código alternativo a interpretar cuando el navegador no soporte Java.

A continuación se muestra un ejemplo de applet. Como puede verse, basta crear una subclase de la clase `Applet` e incorporar en ella el método `init()`, en el que deberá indicarse lo que hace dicho applet (en nuestro ejemplo simplemente se añade un botón).

Programa 3.8 (Ejemplo de applet).

```
import java.applet.Applet;
import java.awt.*;

public class EjemploApplet extends Applet{
  Button boton;
  public void init(){
    boton=new Button(''OK'');
    add(''South'',boton);
  }
}
```

■

Este código debe escribirse primero en un fichero llamado `EjemploApplet.java`, y a continuación compilarse (en el caso del paquete de desarrollo de SUN arrastrando el icono `EjemploApplet.java` sobre el icono Java Compiler). La compilación crea un fichero llamado `EjemploApplet.class`, que contiene la clase compilada. Para poder ejecutarlo hace falta crear también un fichero `EjemploApplet.html`, en el mismo directorio que el de la clase `EjemploApplet.class`, que se encargue de arrancar el applet. Por ejemplo, dicho fichero podría tener el contenido siguiente:

```
<HTML>
      <HEAD>
              <TITLE> </TITLE>
      </HEAD>
      <BODY>
              <APPLET CODE=''EjemploApplet.class'' WIDTH=100 HEIGHT=100></APPLET>
      </BODY>
</HTML>
```

Para ejecutar el applet hay distintas opciones:

1. Arrastrar el fichero HTML sobre el icono Applet Viewer.
2. Dar el URL del fichero HTML creado. Por ejemplo, si el fichero "EjemploApplet.html" estuviera en la carpeta "Ejemplos" del disco "Disco Rigido" bastaría dar el URL:

   `file:/Disco Rigido/Ejemplos/EjemploApplet.html`

Práctica 3.15 (Transformación de aplicación a Applet). Transformar la aplicación de la Práctica 3.2 en un applet. A continuación se muestra el código del Applet que sustituye a la clase `Principal`:

```
package rectangulo;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Applet1 extends Applet {
  private boolean isStandalone = false;

  //Get a parameter value
  public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
      (getParameter(key) != null ? getParameter(key) : def);
  }

  //Construct the applet
  public Applet1() {
  }

  //Initialize the applet
  public void init() {
    try {
      jbInit();
      Rectangulo cuadro1=new Rectangulo(0.0,20.0,0.0,30.0);
      System.out.println("Las coordenadas del cuadro1 son (" + cuadro1.xi+", "
           + cuadro1.yi + ") y (" + cuadro1.xs + ", " + cuadro1.ys + ")");
      System.out.println("El ancho del rectangulo es " + cuadro1.ancho);
      System.out.println("El alto del rectangulo es " + cuadro1.alto);
      Rectangulo cuadro2=new Rectangulo(10.0,10.0,10,10);
      System.out.println("Las coordenadas del cuadro2 son (" + cuadro2.xi + ", "
           + cuadro2.yi + ") y (" + cuadro2.xs + ", " + cuadro2.ys + ")");
      cuadro1.mover(5.0,10.0);
      System.out.println("Las nuevas coordenadas del cuadro1 son (" + cuadro1.xi
           + ", " + cuadro1.yi + ") y (" + cuadro1.xs + ", " + cuadro1.ys + ")");
      if(cuadro1.dentro(cuadro2.xi, cuadro2.yi)){
        System.out.println("La esquina inferior izquierda del cuadro 2 "
             + "esta dentro del cuadro1");
      }else{
        System.out.println("La esquina inferior izquierda del cuadro 2 "
             + "esta fuera del cuadro1");
      }
      Rectangulo cuadro3=new Rectangulo();
      Rectangulo cuadro4=new Rectangulo();
      cuadro3=cuadro1.menorrec(cuadro2);
      System.out.println("El cuadro 2 lo constituyen los puntos (" + cuadro2.xi +
           ", " + cuadro2.yi + ") y (" + cuadro2.xs + ", " + cuadro2.ys + ")");
      cuadro4=cuadro2.menorrec(cuadro1);
      System.out.println("El menor rectangulo que contiene los cuadros 1 y 2 es");
      System.out.println("El cuadro 3 lo constituyen los puntos (" + cuadro3.xi +
           ", " + cuadro3.yi + ") y (" + cuadro3.xs + ", " + cuadro3.ys + ")");
      System.out.println("El cuadro 4 lo constituyen los puntos (" + cuadro4.xi +
           ", " + cuadro4.yi + ") y (" + cuadro4.xs + ", " + cuadro4.ys + ")");
      Rectangulo cuadro5=new Rectangulo();
      Rectangulo cuadro6=new Rectangulo();
      cuadro5=cuadro1.intersec(cuadro2);
      cuadro6=cuadro2.intersec(cuadro1);
      System.out.println("El rectangulo interseccion de los puntos 1 y 2 es");
      System.out.println("El cuadro 5 lo constituyen los puntos (" + cuadro5.xi +
           ", " + cuadro5.yi + ") y (" + cuadro5.xs + ", " + cuadro5.ys + ")");
```

La clase generada se introduce en una página web de la siguiente manera:

```
<html>
<head>
<title>
HTML Test Page
</title>
</head>
<body>
applet.Applet1 will appear below in a Java enabled browser.<br>
<applet
  codebase = "."
  code     = "rectangulo.Applet1.class"
  name     = "TestApplet"
  width    = "400"
  height   = "300"
  hspace   = "0"
  vspace   = "0"
  align    = "middle"
>
</applet>
</body>
</html>
```

■

Restricciones de los applets

Por razones de seguridad los applets tienen varias restricciones. Un applet no puede:

1. Cargar librerías o definir métodos nativos.
2. Leer ni escribir ficheros en el sistema anfitrión que está ejecutándolo.
3. Hacer conexiones de red, excepto al sistema anfitrión del que procede.
4. Ejecutar cualquier programa en el sistema anfitrión que está ejecutándolo.
5. Leer ciertas propiedades del sistema.

Cada navegador tiene un objeto de tipo `SecurityManager` que implementa sus medidas de seguridad. Cuando un objeto de este tipo detecta una violación de tales medidas, envía una excepción de tipo `SecurityException` para que el sistema proceda en consecuencia.

3.13.2. Aplicaciones

A continuación se muestra un ejemplo de aplicación. Para crearla hay que crear un fichero con el nombre `EjemploAplicacion.java`, y entre sus clases debe encontrarse una clase `public class EjemploAplicacion` en la cual hay que definir el método `main()`, que es necesario para arrancar la aplicación.Anteriormente en este capítulo se ha descrito con detalle este método. En dicho método `main()` es necesario crear un ejemplar de dicha clase,

puesto que éste no se crea automáticamente, tal como sucede al ejecutar un applet en un navegador o en un visor de applets. Puesto que, al contrario de lo que sucede en el caso de los applets, en una aplicación, tampoco se crea automáticamente un lugar donde mostrar la interfase, es necesario crear explícitamente una ventana en la que se muestre dicha interfase. Para ello se crea una subclase de la clase `Frame`. Mientras que en el caso de los applets, las dimensiones de la ventana en la que se muestra se dan en el fichero HTML, dando valores a los parámetros WIDTH y HEIGHT, en una aplicación estas dimensiones se dan mediante el método `resize` de la clase `Frame`. Además, para que se muestre la ventana es necesario invocar el método `show()`.

Programa 3.9 (Ejemplo de aplicación).

```
import java.awt.*;

public class EjemploAplicacion extends Frame{
  Button boton;
  public EjemploAplicacion(){
  boton=new Button(''OK'');
  add(''South'',boton);
  }
  public static void main(String[] args){
    EjemploAplicacion ejem=new EjemploAplicacion();
    ejem.resize(100,100);
    ejem.show();
  }
}
```

■

Una vez creado el fichero `EjemploAplicacion.java`, éste se compila arrastrando el icono correspondiente sobre el icono `Java Compiler`, con lo cual se crea el fichero `EjemploAplicacion.class`. Finalmente, pulsando dos veces sobre su icono se arranca la aplicación.

Por tanto, las principales diferencias entre los applets y las aplicaciones pueden resumirse en:

1. Mientras que las aplicaciones arrancan solas, los applets necesitan programas auxiliares (otras aplicaciones) para su arranque.

2. Mientras que las aplicaciones no tienen limitaciones en cuanto a las operaciones a realizar, los applets están sometidos a restricciones por motivos de seguridad.

3. Las aplicaciones arrancan con el método `main` y los applets con el método `init`.

4. Al ejecutar un applet se crea automáticamente un ejemplar de la clase que lo implementa, cosa que no ocurre con las aplicaciones, que deben crear sus ejemplares en el método `main`.

5. Al ejecutar un applet se crea automáticamente una ventana donde mostrar el applet, mientras que en las aplicaciones ésta debe ser creada explícitamente.

6. Las dimensiones de la ventana en la que se muestra un applet se definen en el fichero HTML que lo arranca, mientras que en el caso de las aplicaciones debe hacerse en el programa Java, mediante el método `resize`.

3.14. Más prácticas resueltas

Práctica 3.16 (Eliminación de caracteres). Escribir un programa que elimine caracteres consecutivos iguales de una cadena de entrada. Tendrá las características siguientes:

1. Usad `s.toCharArray()` para convertir una cadena de caracteres `s` en una matriz de caracteres.
2. Usad new `String(t,offset,length)` para convertir una matriz de caracteres t en una cadena de caracteres.
3. Recordad que `t.length` es el tamaño del array.
4. Comprobar que el número de argumentos es uno y que la longitud de la cadena no es cero.

Solución:

```
import java.io.*;

class Eliminar {
   static char t[];

   static public void eliminar() {
    int n = t.length;
    int j = 0;
    for (int i = 0; i < n ; i++) {
       char c = t[j++] = t[i];
       while (i < n && c == t[i] ) i++;
    }
    t = new String(t,0,j).toCharArray();
   }

   public static void main(String args[])  {

     if (args.length != 1) {
      System.out.println("eliminar
          necesita un argumento de tipo string");
     }
     if (args[0].length() <= 0) {
       System.out.println("eliminar
                necesita un argumento no vacio");
     }

     t = args[0].toCharArray();
```

```
    System.out.println(t + " -> ");
    eliminar();
    System.out.println(t + "\n");
  }
```

■

Ejercicios

Ejercicio 1.

Escribir un programa Java que permita convertir horas en notación de 24 horas a su equivalente en notación de 12 horas. Por ejemplo, 23:50 debe ser convertida en 11:50 PM.

Ejercicio 2.

Escribir un pequeño programa Java que permita convertir números romanos a su equivalente en cifras arábigas.

Ejercicio 3.

Escribir un programa que reciba una fecha en formato `dd/mm/aa` (20/08/92) y sea devuelta como texto (20 de Agosto de 1992).

Ejercicio 4.

Desarrollar un pequeño programa que lea la fecha de nacimento de una persona y calcule su edad en años, meses y días.

Ejercicio 5.

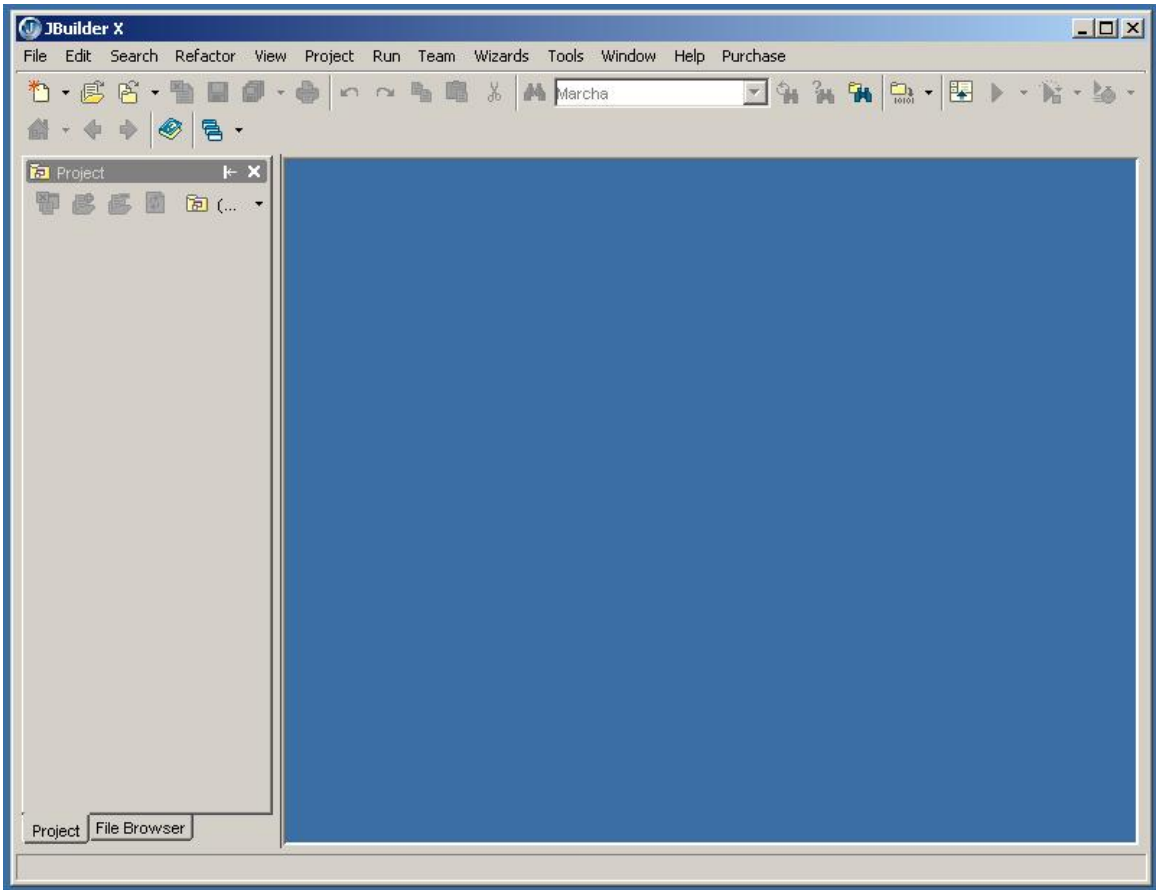
Construir un programa que lea una cadena de texto y devuelva otra cadena formada por todas las letras mayúsculas que aparecen en la primera.

Capítulo 4

Programa JBuilder

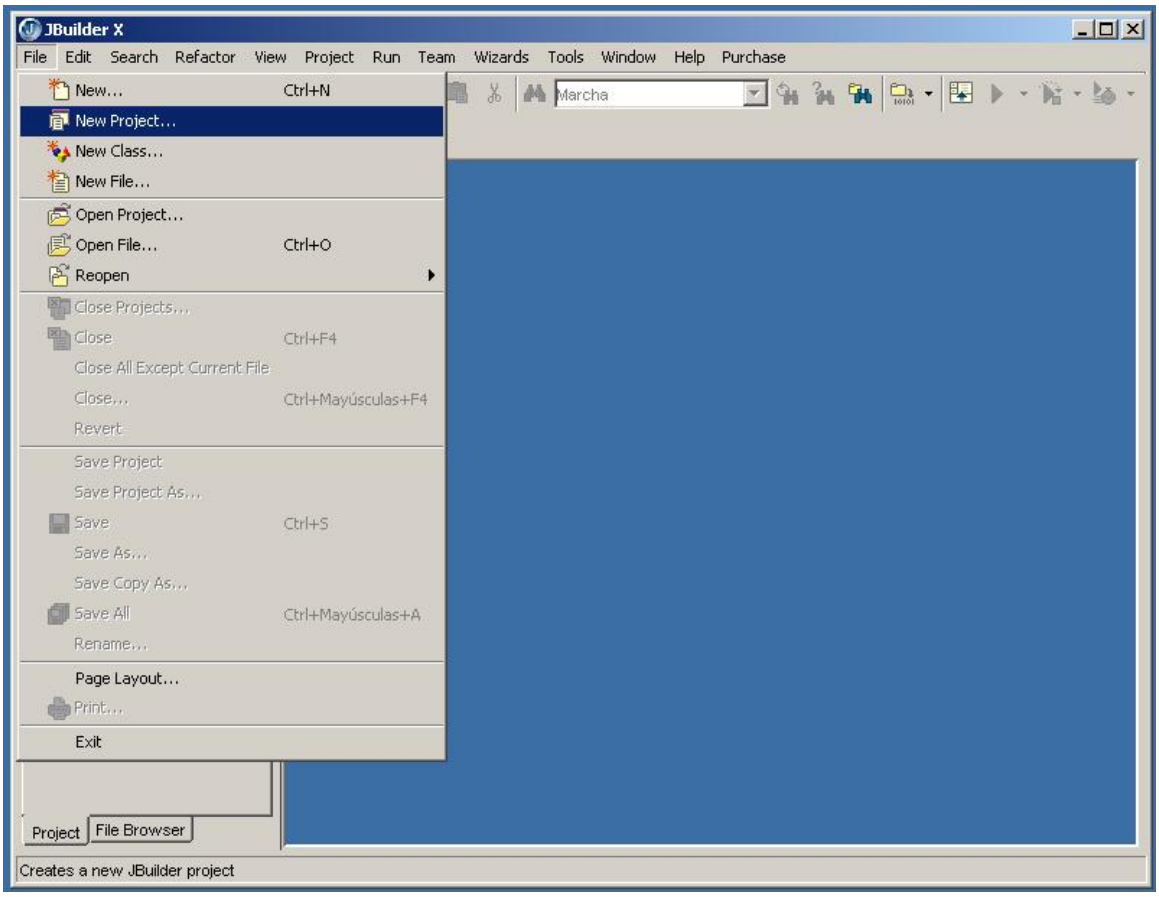
En este curso se va a emplear para la compilación de código Java el programa **JBuilder**, cuya version educacional es gratuita y permita la creación sensilla de paquetes y programas mediante clases. En este capítulo se va a proceder a explicar cómo se crearía un programa que implemente el ejemplo de números complejos de la Sección 3.1.

En primer lugar se procede a abrir el programa **JBuilder** con lo cual aparecerá una pantalla similar a la siguiente:



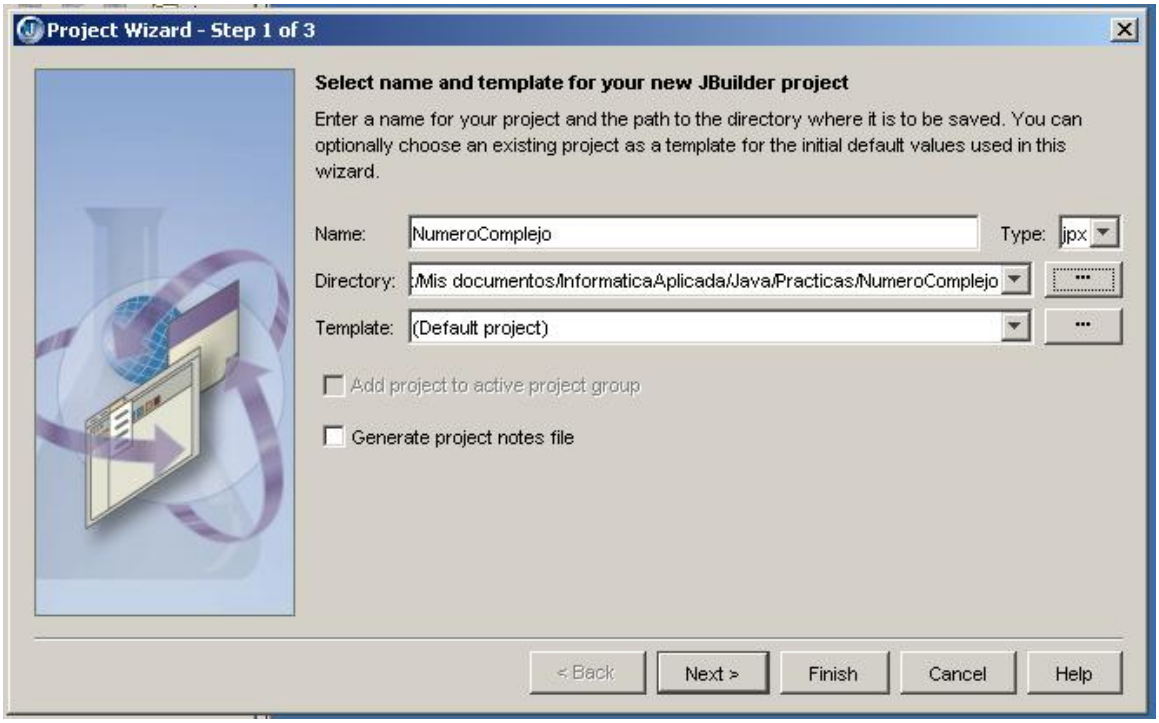
en caso de que no sea así, posiblemente aparezca un proyecto por defecto, si se cierra se llegará a la pantalla anterior.

El siguiente paso es la creación de un proyecto que gestionará todas las clases asociadas al problema y las integrará en un paquete con el mismo nombre que el nombre que le demos al proyecto. El fichero creado tendrá extensión *.jpx, y llevará asociado una serie de carpetas con sus correspondientes ficheros. Así en el menú **file** seleccionamos la opción **New Proyect...** tal y como se muestra en la siguiente ilustración:

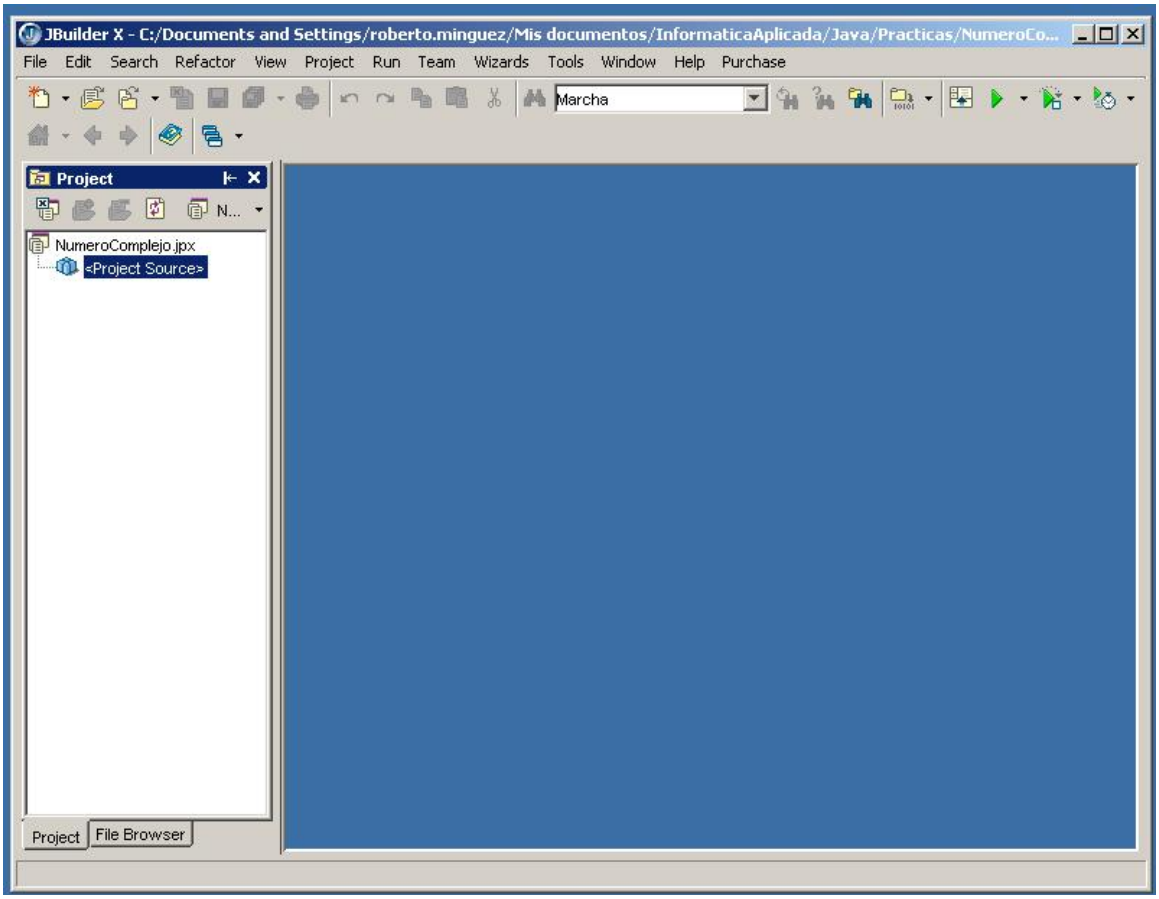


Una vez seleccionada la opción **New Proyect...** se despliega una ventana como la de la gráfica a continuación en la que seleccionamos el nombre del proyecto que en este caso se llamará **NumeroComplejo**, a continuación se verá cómo ese nombre afecta al nombre del paquete en el que se almacenarán todas las clases que creemos. Nótese que en el directorio hay que seleccionar la ubicación dentro del disco duro en la que se desea almacenar la

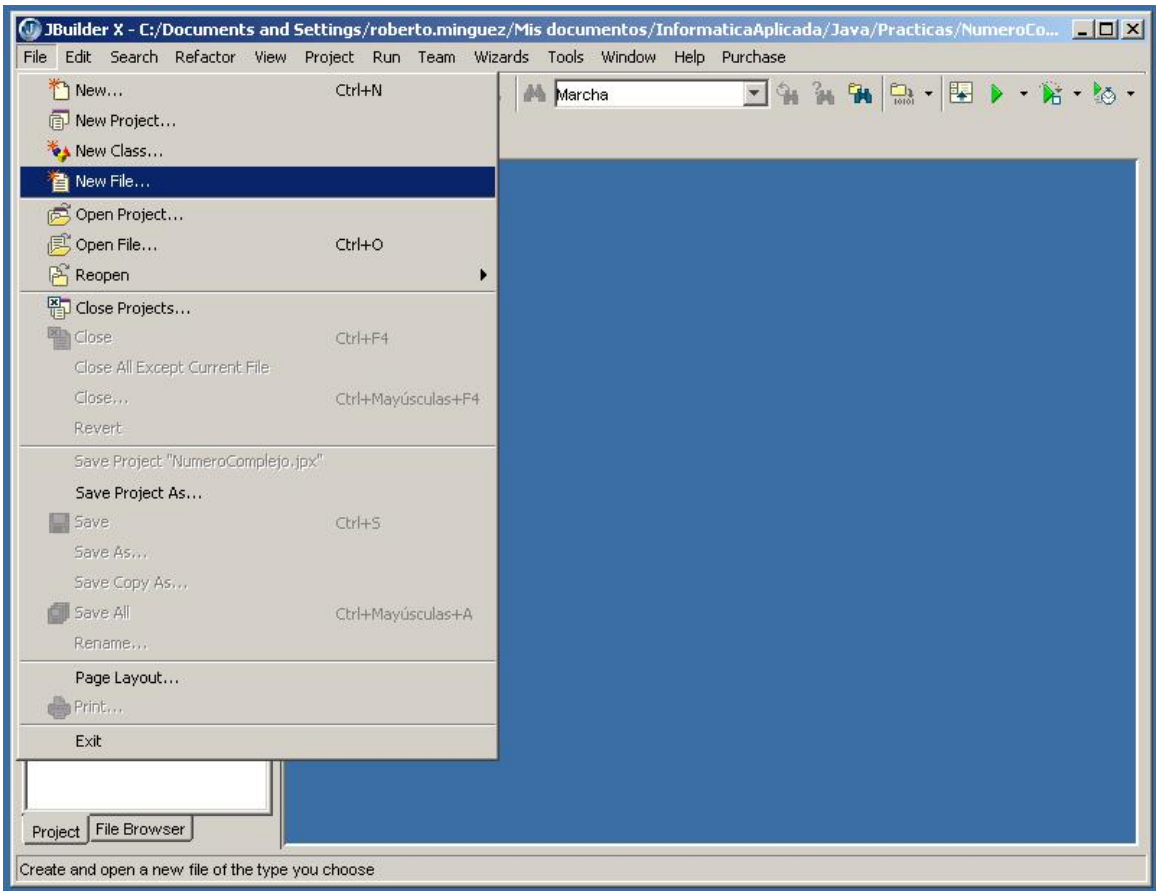
carpeta con todos los ficheros para el proyecto.



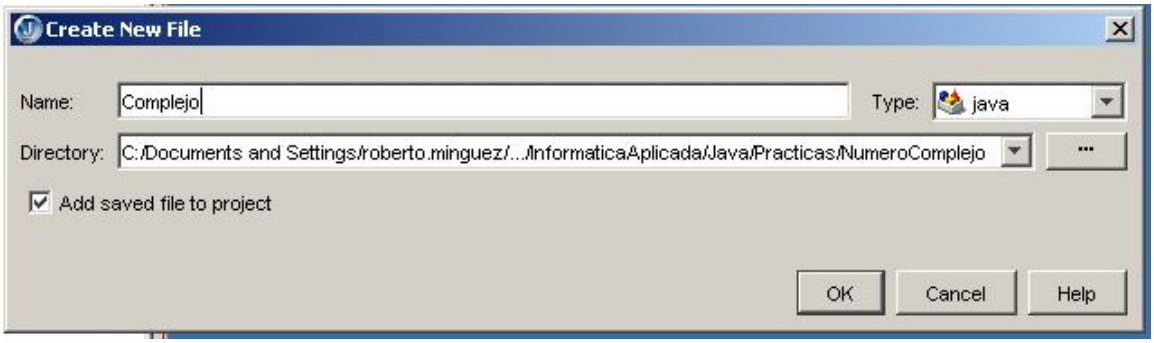
A continuación se pulsa el botón **next** y **end** hasta que aparece la pantalla mostrada en el gráfico a continuación:



Ya hemos creado el proyecto, a continuación se procede a la creación de las clases en ficheros con extensión *.java. Hay dos procedimientos para hacerlo, el primero consiste en utilizar la opción **New File...** del menú **File** tal y como se muestra en la gráfica siguiente.

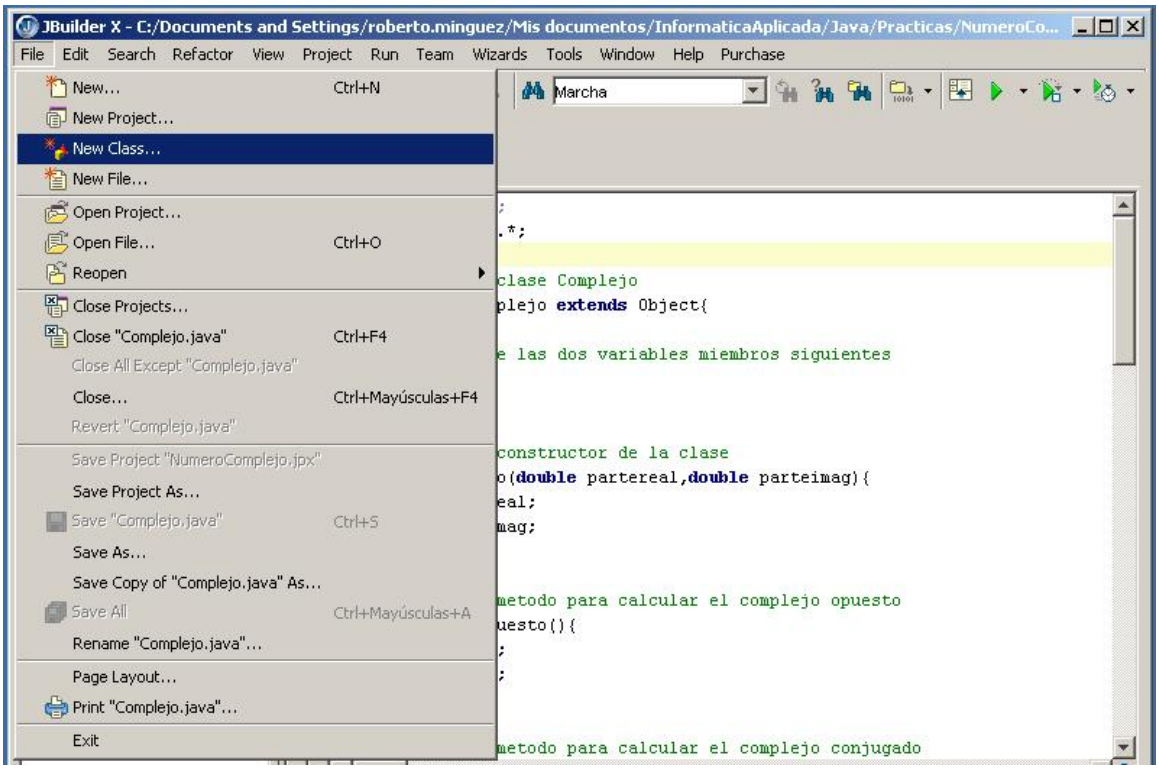


Con lo que aparece el siguiente menú desplegable:

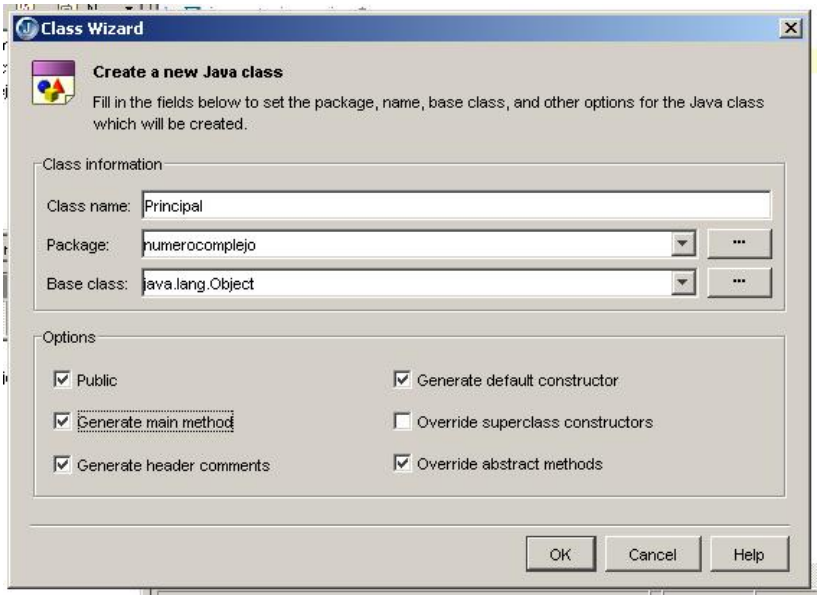


En el que escribimos el nombre de la clase **Complejo**, en este caso, que será un fichero tipo **Java** y que se almacenará en el directorio por defecto, que es el mismo del proyecto.

La segunda opción y más útil desde el punto de vista práctico consiste en emplear la opción **New Class...** del menú **File** tal y como se muestra en la figura siguiente:

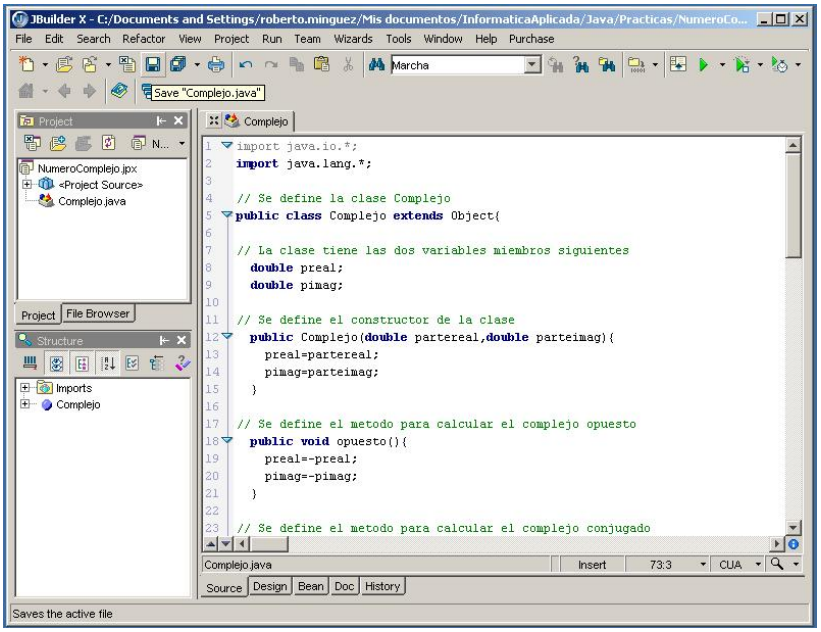


Al pulsar la opción aparece el siguiente menú en el que se le da nombre a la clase, se le dice el paquete al que va a pertenecer, nótese que por defecto el nombre del paquete es el nombre del proyecto pero con minúsculas, y luego da la opción de crear el método `main()`, y dar varias características de las clases.

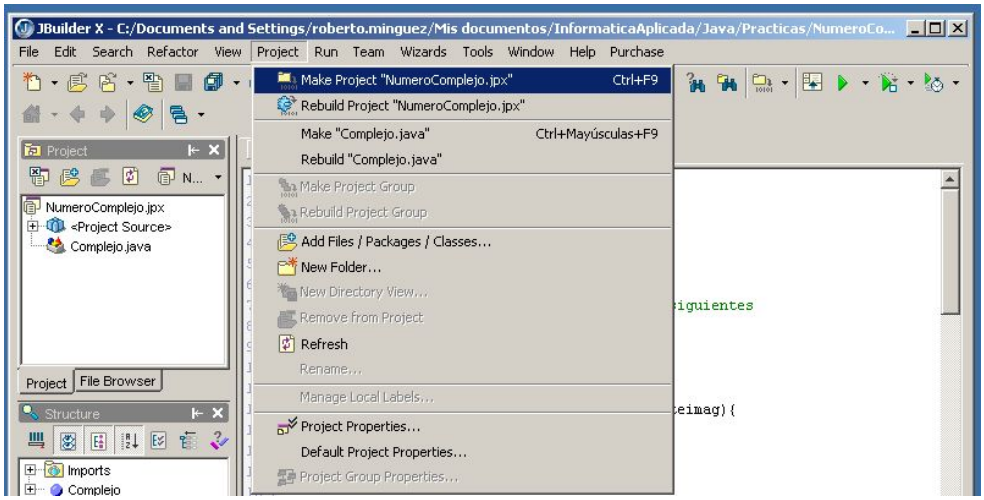


Comentario 4.1 *Recuérdese que en este curso todas las clases se considerarán públicas (*`public`*) o sin adjetivo, lo que implica que son* `friendly`*.* ■

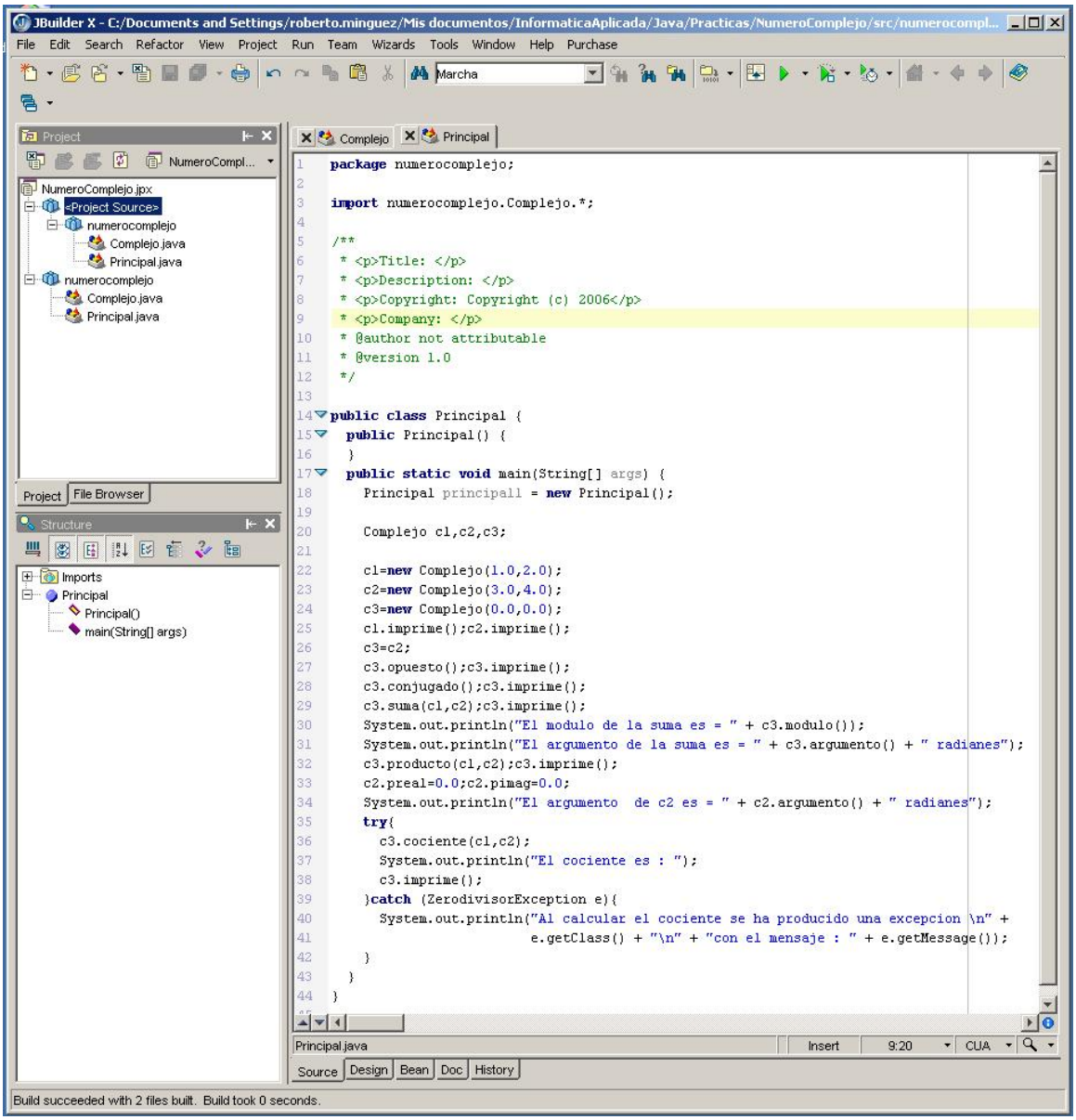
En el fichero generado se pega el código mostrado en el programa 3.1 de la página 24 y se graba el fichero pulsando el botón **grabar**, tal y como se muestra en la gráfica siguiente:



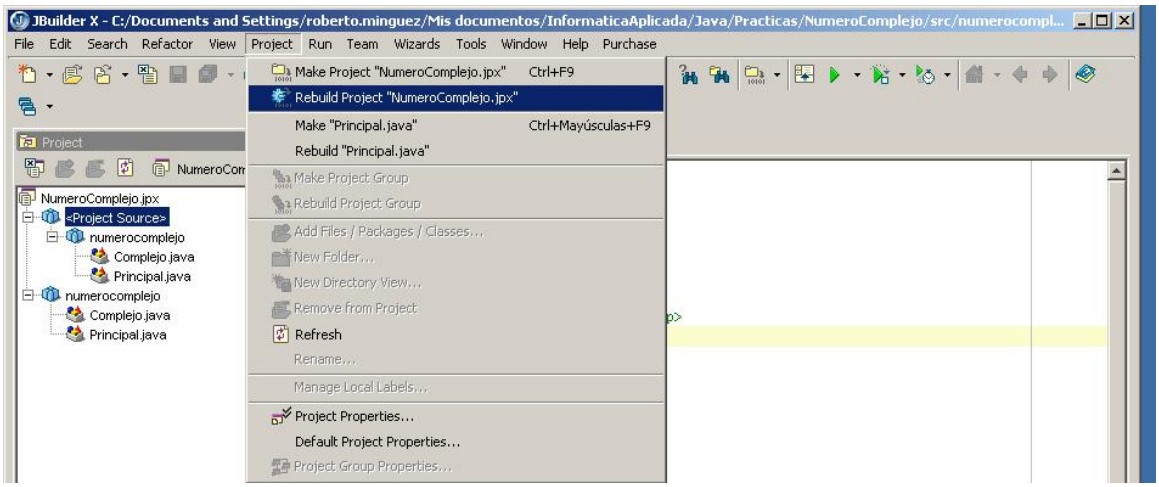
Para compilar las clases dentro del proyecto y generar los códigos compilados, es decir, los ficheros *.class se pulsa la opción **Make Project "NumeroComplejo.jpx"** dentro del menú **Project**:



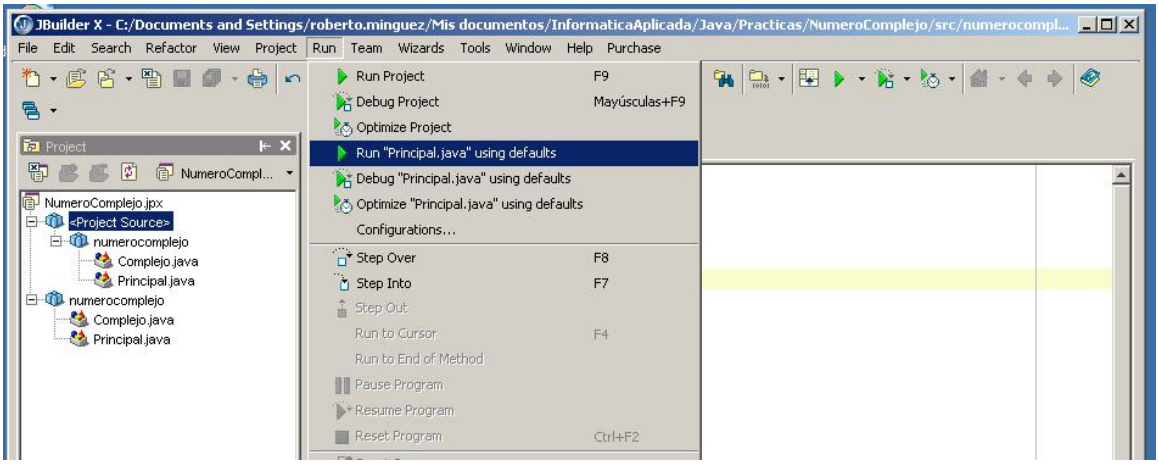
Siguiendo un procedimiento análogo se crea la clase `Principal` que será la que contenga el método `main()` y en la que se operarán con objetos de la clase `Complejo`. Nótese que el código de este fichero será el mismo que el del programa 3.2, de la página 27:


JBuilder X - C:/Documents and Settings/roberto.minguez/Mis documentos/InformaticaAplicada/Java/Practicas/NumeroComplejo/src/numerocompl...
File Edit Search Refactor View Project Run Team Wizards Tools Window Help Purchase
Marcha
Project
NumeroCompl...
NumeroComplejo.jpx
<Project Source>
numerocomplejo
Complejo.java
Principal.java
numerocomplejo
Complejo.java
Principal.java
Project File Browser
Structure
Imports
Principal
Principal()
main(String[] args)
Complejo Principal
1 package numerocomplejo;
3 import numerocomplejo.Complejo.*;
5 /**
6 * Title:
7 * Description:
8 * Copyright: Copyright (c) 2006
9 * Company:
10 * @author not attributable
11 * @version 1.0
12 */
14 public class Principal {
15 public Principal() {
16 }
17 public static void main(String[] args) {
18 Principal principal1 = new Principal();
20 Complejo c1,c2,c3;
22 c1=new Complejo(1.0,2.0);
23 c2=new Complejo(3.0,4.0);
24 c3=new Complejo(0.0,0.0);
25 c1.imprime();c2.imprime();
26 c3=c2;
27 c3.opuesto();c3.imprime();
28 c3.conjugado();c3.imprime();
29 c3.suma(c1,c2);c3.imprime();
30 System.out.println("El modulo de la suma es = " + c3.modulo());
31 System.out.println("El argumento de la suma es = " + c3.argumento() + " radianes");
32 c3.producto(c1,c2);c3.imprime();
33 c2.preal=0.0;c2.pimag=0.0;
34 System.out.println("El argumento de c2 es = " + c2.argumento() + " radianes");
35 try{
36 c3.cociente(c1,c2);
37 System.out.println("El cociente es : ");
38 c3.imprime();
39 }catch (ZerodivisorException e){
40 System.out.println("Al calcular el cociente se ha producido una excepcion \n" +
41 e.getClass() + "\n" + "con el mensaje : " + e.getMessage());
42 }
43 }
44 }
Principal.java
Insert
9:20
CUA
Source Design Bean Doc History
Build succeeded with 2 files built. Build took 0 seconds.

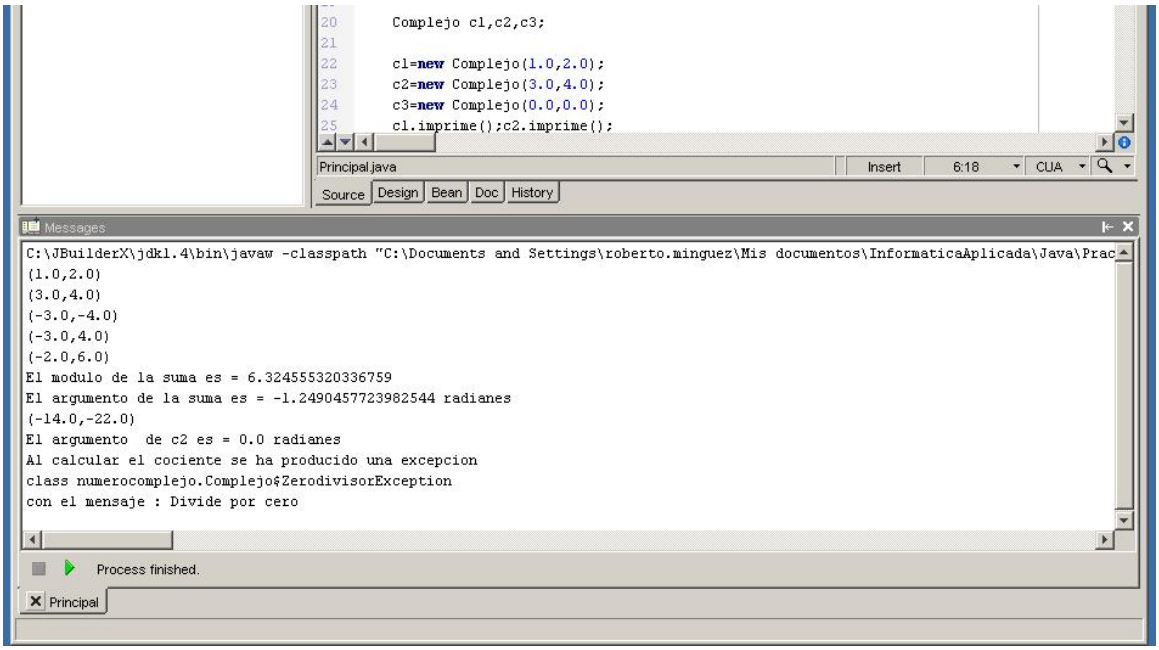
Dado que hemos añadido una nueva clase al paquete, reconstruimos el proyecto y compilamos de nuevo empleando la opción **Rebuild Project "NumeroComplejo.jpx"** dentro del menú **Project**:



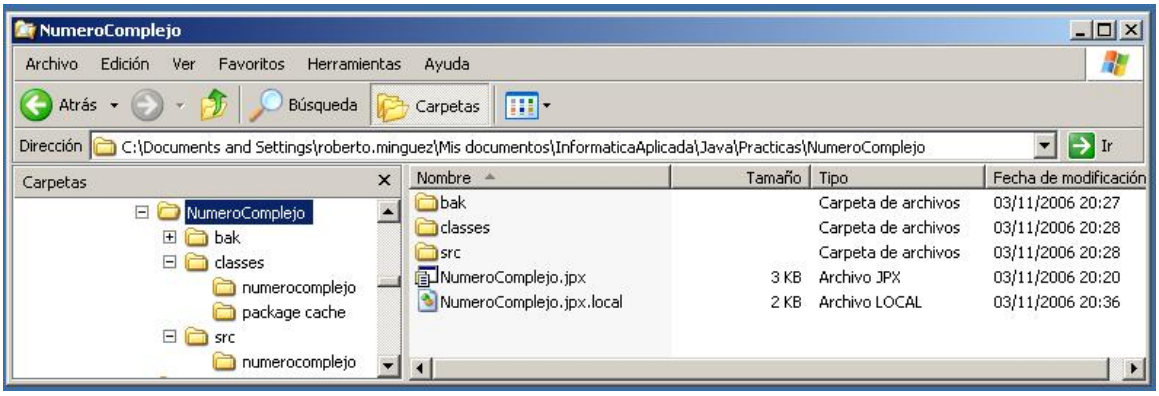
Ya se está en disposición de ejecutar el programa mediante el fichero **Principal.class**, esto se puede hacer con el JBuilder sin más que pulsar la opción **Run "Principal.java" using defaults** dentro del menú **Run**. Nótese que tiene que estar activa la ventana asociada al fichero **Principal.java** para que esa opción esté disponible.



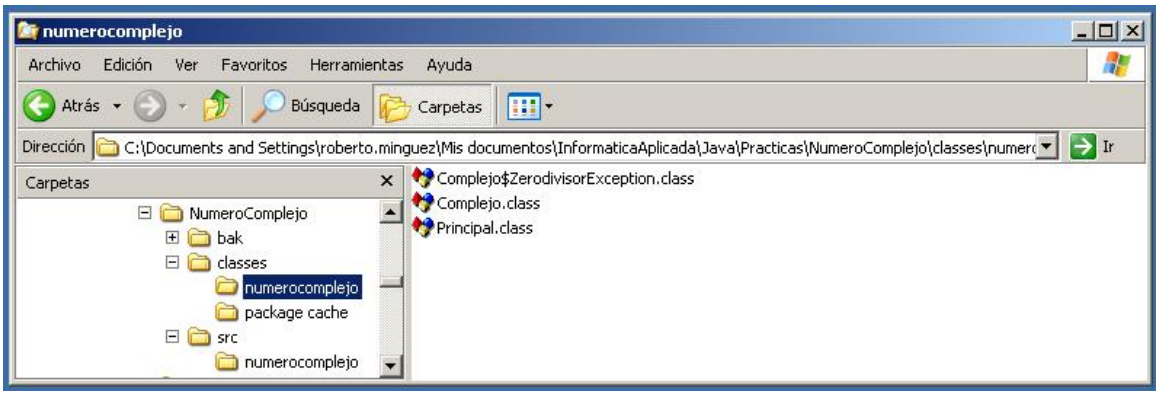
Inmediatamente aparece una ventana de salida en la que se genera la salida escrita generada por el programa, tal y como se muestra en el gráfico siguiente:



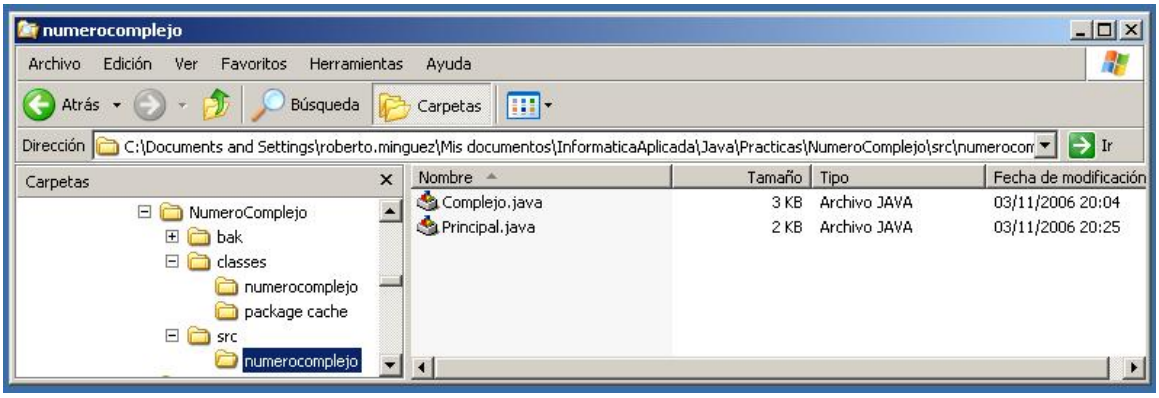
Es interesante saber cómo el programa almacena los ficheros. Inicialmente se crea una carpeta con el mismo nombre que el fichero *.jpx del proyecto en la que se almacena, por un lado el fichero **NumeroComplejo.jpx** y tres carpetas con los nombres **back**, **classes** y **src**:



En la primera se guarda una copia de seguridad, en la segunda se almacenan los ficheros java compilados, es decir, los ficheros con extensión *.class como se muestra a continuación:



Mientras que en la carpeta **src** se almacenan los ficheros fuente con extensión *.java:



Apéndice A

Exámenes Resueltos

Práctica A.1 (Parcial de 2 de Diciembre de 2005). Se va a proceder a elaborar un programa de gestión de alumnos en un instituto. Para ello se generará una clase **Alumno** que viene definida por las siguientes variables y métodos que deberán estar en la definición de la clase. La selección del tipo de variables será decisión vuestra. La clase ha de tener las siguientes características:

1. Las variables principales que caracterizan a un alumno son su número de dni (documento nacional de identidad) (**dni**), su nombre (**nombre**), su primer apellido (**apellido1**) y su segundo apellido (**apellido1**). (2 puntos)

2. Se generará un constructor para permitir la inicialización de un objeto de esta clase sin dar ningún tipo de argumento mediante la siguiente sentencia (2 puntos)

   ```
   super();
   ```

3. También se generará un segundo constructor cuyos argumentos son el dni, nombre, primer apellido y segundo apellido. De esta forma se inicializaran todas las variables principales de la clase alumno definidas en el apartado 1. (8 puntos)

4. El número máximo de asignaturas será de 7 para todos los alumnos y no podrá ser cambiado. Ese valor máximo ha de fijarse dentro de la clase y no podrá modificarse de ninguna forma. (2 puntos)

5. La clase también contendrá dos variables que contengan los nombres de las asignaturas en las que cada alumno está matriculado (**asignaturas**) y las notas (**notas**) de cada asignatura. Se inicializarán al comienzo reservando espacio en memoria para almacenarlas y sólo se asignarán sus valores mediante los métodos **matricular** y **puntuar**. Téngase en cuenta el número máximo de asignaturas para definir las dimensiones de estas variables. (2 puntos)

6. El método **matricular** asignará valores a la variable **asignaturas** y recibirá como argumento un vector que contenga los nombres de las asignaturas en las que se matricula el alumno. Para la asignación de las variables se procederá componente a componente. (7 puntos)

7. El método **puntuar** asignará valor a la nota de alguna de las asignaturas, para ello recibirá como argumentos el nombre de la asignatura para la cual se va a asignar la nota y la puntuación obtenida. A continuación se deberá de buscar la posición adecuada en la que insertar la nota teniendo en cuenta que hay una correspondencia entre el orden del listado de asignaturas que se encuentra almacenado en la variable **asignaturas** y el orden del listado de notas (almacenado en la variable **notas**). En el momento que se haya localizado la asignatura correcta se parará el procedimiento de búsqueda y se asignará la nota. **NOTA:** El método **equals(String arg)** de la clase **String** permite comparar si dos cadenas de caracteres son iguales. Devuelve el valor **true** si son iguales y **false** si son distintos. En este caso **arg** es la variable con la que se quiere comparar. (12 puntos)

8. Se generará un método llamado **aprobado** que me permita saber si un alumno ha aprobado una asignatura o no. Recibirá como argumentos el nombre de la asignatura de la que se quiere saber si aprobó o no. Devolverá **true** o **false**. Al igual que en el método anterior una vez que se sepa si la asignatura se ha aprobado o no, no se continuará buscando en la lista de asignaturas. En caso de que no se localize la asignatura de la que se quiere hacer la consulta se devolverá el valor **false** y se escribirá el siguiente mensaje: (13 puntos)

   "En alumno no está matriculado en la asignatura que está consultando."

9. El último método llamado **media** me permitirá calcular la nota media de todas las asignaturas como

$$x_m = \frac{\sum_{i=1}^{n} x_i}{n},$$

   donde x_m es la media, n es el número de asignaturas y x_i es la nota de la asignatura i-ésima. (12 puntos)

A continuación se generará una segunda clase llamada **Matriculacion** que contendrá el método **main()** en el que realizarán las siguientes acciones:

1. Se generará un objeto de la clase **Alumno** creada anteriormente y lo almacenaréis en una variable que sea vuestro nombre con minúsculas. Por ejemplo si te llamas Roberto el nombre de la variable será **roberto**. Para la generación se empleará el constructor que requiere los datos del dni, nombre, primer apellido y segundo apellido. Utilizad vuestros datos personales (el número de dni os lo podéis inventar). (4 puntos)

2. A continuación crearéis un array de "strings" almacenado en una variable llamada **asignaturas** y reservaréis espacio en memoria para almacenar tantos elementos como número máximo de asignaturas de la clase alumno. Ese número máximo ha de ser relativo a la clase alumno, no vale poner 7 directamente. (4 puntos)

3. Almacenar en ese vector la lista con las siguientes asignaturas "Física", "Matemáticas", "Lenguaje", "Álgebra", "Historia", "Literatura" "Dibujo". (4 puntos)

4. Matricular al objeto de la clase **Alumno** que creasteis en el primer apartado en las asignaturas contenidas en la variable **asignaturas** mediante el método correspondiente. (4 puntos)

5. A continuación se va a proceder a asignar notas a todas las asignaturas. Para ello emplearéis el método **puntuar** creado en la clase **Alumno** anteriormente. La nota numérica se generará de forma aleatoria para cada asignatura utilizando la siguiente expresión: (10 puntos)

   ```
   10*Math.random()
   ```

   El resultado de esta expresión es una variable tipo **double** cuyo valor oscila entre 0 y 10.

6. Escribir para todas las asignaturas del objeto **Alumno** la frase: (10 puntos)

   "La asignatura" nombre de la asignatura *"está aprobada con una nota de"* valor numérico *".*"

   si la asignatura correspondiente está aprobada (nota$\geq$ 5), o

   "La asignatura" nombre de la asignatura *"está suspensa con una nota de"* valor numérico *".*"

   si está suspensa.

7. Por último calcula la nota media mediante el método que generaste en la clase **Alumno** y escríbela. (4 puntos)

Tiempo: 1h 45 min.

Solución:

Se va a proceder a elaborar un programa de gestión de alumnos en un instituto. Para ello se generará una clase **Alumno** que viene definida por las siguientes variables y métodos que deberán estar en la definición de la clase. La selección del tipo de variables será decisión vuestra. La clase ha de tener las siguientes características:

1. Las variables principales que caracterizan a un alumno son su número de dni (documento nacional de identidad) (**dni**), su nombre (**nombre**), su primer apellido (**apellido1**) y su segundo apellido (**apellido1**). (2 puntos)

   ```
   package alumno;

   public class Alumno {
       int dni;
       String nombre,apellido1,apellido2;
   ```

2. Se generará un constructor para permitir la inicialización de un objeto de esta clase sin dar ningún tipo de argumento mediante la siguiente sentencia (2 puntos)

```
    super();


    public Alumno() {
    super();
}
```

3. También se generará un segundo constructor cuyos argumentos son el dni, nombre, primer apellido y segundo apellido. De esta forma se inicializaran todas las variables principales de la clase alumno definidas en el apartado 1. (8 puntos)

```
public Alumno(int dni,String nombre, String apellido1,
    String apellido2) {
super();
this.dni=dni;
this.nombre=nombre;
this.apellido1=apellido1;
this.apellido2=apellido2;
}
```

4. El número máximo de asignaturas será de 7 para todos los alumnos y no podrá ser cambiado. Ese valor máximo ha de fijarse dentro de la clase y no podrá modificarse de ninguna forma. (2 puntos)

```
    static final int numasi=7;
```

5. La clase también contendrá dos variables que contengan los nombres de las asignaturas en las que cada alumno está matriculado (**asignaturas**) y las notas (**notas**) de cada asignatura. Se inicializarán al comienzo reservando espacio en memoria para almacenarlas y sólo se asignarán sus valores mediante los métodos **matricular** y **puntuar**. Téngase en cuenta el número máximo de asignaturas para definir las dimensiones de estas variables. (2 puntos)

```
    String asignaturas[]=new String[numasi];
    float notas[]=new float[numasi];
```

6. El método **matricular** asignará valores a la variable **asignaturas** y recibirá como argumento un vector que contenga los nombres de las asignaturas en las que se matricula el alumno. Para la asignación de las variables se procederá componente a componente. (7 puntos)

```
public void matricular(String asignatura[]){
    asignaturas=asignatura;
    int i;
    for(i=0;i<asignatura.length;i++){
        asignaturas[i]=asignatura[i];
    }
}
```

7. El método **puntuar** asignará valor a la nota de alguna de las asignaturas, para ello recibirá como argumentos el nombre de la asignatura para la cual se va a asignar la nota y la puntuación obtenida. A continuación se deberá de buscar la posición adecuada en la que insertar la nota teniendo en cuenta que hay una correspondencia entre el orden del listado de asignaturas que se encuentra almacenado en la variable **asignaturas** y el orden del listado de notas (almacenado en la variable **notas**). En el momento que se haya localizado la asignatura correcta se parará el procedimiento de búsqueda y se asignará la nota. **NOTA:** El método **equals(String arg)** de la clase **String** permite comparar si dos cadenas de caracteres son iguales. Devuelve el valor **true** si son iguales y **false** si son distintos. En este caso **arg** es la variable con la que se quiere comparar. (12 puntos)

```
public void puntuar(String asigna,float puntuacion){
    boolean encontrado = false;
    short i = 0;
    while (!encontrado & i<asignaturas.length) {
        if (asigna.equals(asignaturas[i])) {
            notas[i] = puntuacion;
            encontrado = true;
        }
        else {
            i++;
        }
    }
    if(!encontrado){
        System.out.println("El alumno no esta matriculado
        de esa asignatura"+asigna);
    }
}
```

8. Se generará un método llamado **aprobado** que me permita saber si un alumno ha aprobado una asignatura o no. Recibirá como argumentos el nombre de la asignatura de la que se quiere saber si aprobó o no. Devolverá **true** o **false**. Al igual que en el método anterior una vez que se sepa si la asignatura se ha aprobado o no, no se continuará buscando en la lista de asignaturas. En caso de que no se localize la asignatura de la que se quiere hacer la consulta se devolverá el valor **false** y se escribirá el siguiente mensaje: (13 puntos)

"En alumno no está matriculado en la asignatura que está consultando."

```
public boolean aprobado(String asigna){
    boolean encontrado = false;
    boolean si=false;
    short i=0;
    while(!encontrado & i<asignaturas.length){
        if(asigna.equals(asignaturas[i])){
            if(notas[i]>=5){
                si=true;
            }
            encontrado=true;
        }else{
            i++;
        }
    }
    if(!encontrado){
        System.out.println("El alumno no esta matriculado de esa
        asignatura"+asigna);
    }
    return(si);
}
```

9. El último método llamado **media** me permitirá calcular la nota media de todas las asignaturas como

$$x_m = \frac{\sum_{i=1}^{n} x_i}{n},$$

donde x_m es la media, n es el número de asignaturas y x_i es la nota de la asignatura i-ésima. (12 puntos)

```
public float media(){
    float med=0;
    int i;
    for(i=0;i<notas.length;i++){
        med+=notas[i];
    }
    return(med/notas.length);
}

}// Fin de la clase Alumno
```

A continuación se generará una segunda clase llamada **Matriculacion** que contendrá el método **main()** en el que realizarán las siguientes acciones:

```
package alumno;

public class Matriculacion {

public static void main(String[] args) {
```

1. Se generará un objeto de la clase **Alumno** creada anteriormente y lo almacenaréis en una variable que sea vuestro nombre con minúsculas. Por ejemplo si te llamas Roberto el nombre de la variable será **roberto**. Para la generación se empleará el constructor que requiere los datos del dni, nombre, primer apellido y segundo apellido. Utilizad vuestros datos personales (el número de dni os lo podéis inventar). (4 puntos)

```
Alumno roberto=new Alumno(20202192,"Roberto","Mínguez","Solana");
System.out.println("Alumno "+roberto.nombre+" "+roberto.apellido1+
" "+roberto.apellido2+" con DNI "+roberto.dni);
```

2. A continuación crearéis un array de "strings" almacenado en una variable llamada **asignaturas** y reservaréis espacio en memoria para almacenar tantos elementos como número máximo de asignaturas de la clase alumno. Ese número máximo ha de ser relativo a la clase alumno, no vale poner 7 directamente. (4 puntos)

```
String asignaturas[]=new String[Alumno.numasi];
```

3. Almacenar en ese vector la lista con las siguientes asignaturas "Física", "Matemáticas", "Lenguaje", "Álgebra", "Historia", "Literatura" "Dibujo". (4 puntos)

```
asignaturas[0]="Fisica";
asignaturas[1]="Matematicas";
asignaturas[2]="Lenguaje";
asignaturas[3]="Algebra";
asignaturas[4]="Historia";
asignaturas[5]="Literatura";
asignaturas[6]="Dibujo";
```

4. Matricular al objeto de la clase **Alumno** que creasteis en el primer apartado en las asignaturas contenidas en la variable **asignaturas** mediante el método correspondiente. (4 puntos)

```
roberto.matricular(asignaturas);
```

5. A continuación se va a proceder a asignar notas a todas las asignaturas. Para ello emplearéis el método **puntuar** creado en la clase **Alumno** anteriormente. La nota numérica se generará de forma aleatoria para cada asignatura utilizando la siguiente expresión: (10 puntos)

```
10*Math.random()
```

   El resultado de esta expresión es una variable tipo **double** cuyo valor oscila entre 0 y 10.

```
int i;
for(i=0;i<asignaturas.length;i++){
    roberto.puntuar(asignaturas[i],10*(float)Math.random());
}
```

6. Escribir para todas las asignaturas del objeto **Alumno** la frase: (10 puntos)

   "La asignatura" nombre de la asignatura *"está aprobada con una nota de"* valor numérico *". "*

   si la asignatura correspondiente está aprobada (nota$\geq$ 5), o

   "La asignatura" nombre de la asignatura *"está suspensa con una nota de"* valor numérico *". "*

   si está suspensa.

```
for(i=0;i<asignaturas.length;i++){
    if(roberto.aprobado(asignaturas[i])){
        System.out.println("La asignatura "+asignaturas[i]+
        " está aprobada con una nota de "+roberto.notas[i]);
    }else{
        System.out.println("La asignatura "+asignaturas[i]+
        " está suspensa con una nota de "+roberto.notas[i]);
    }
}
```

7. Por último calcula la nota media mediante el método que generaste en la clase **Alumno** y escríbela. (4 puntos)

```
        System.out.println("La nota media es de "+roberto.media());
    }// Fin del metodo main
}// Fin de la clase matriculacion
```

Tiempo: 1h 45 min.

```
\\ Metodo de ordenacion de mayor a menor puntuacion
    float notord[]=roberto.notas;
    String asiord[]=roberto.asignaturas;

    int j;
    float auxnot;
```

```
String auxasig;
for(i=0;i<notord.length-1;i++){
  for(j=i+1;j<notord.length;j++){
    if(notord[j]>notord[i]){
      auxnot=notord[j];
      notord[j]=notord[i];
      notord[i]=auxnot;
      auxasig=asiord[j];
      asiord[j]=asiord[i];
      asiord[i]=auxasig;
    }
  }
}

for(i=0;i<notord.length-1;i++){
  System.out.println("Posicion "+(i+1)+": "+asiord[i]+"; nota= "+notord[i]);
}
```

■

Práctica A.2 (Final de 16 de diciembre de 2005). La empresa **Telefónica España** va a proceder a la implantación de un nuevo sistema de gestión telefónica mediante el cual pueda tener almacenada la información de todos sus usuarios, con la posibilidad de dar de alta a nuevos usuarios, consultar números para usuarios dados de alta, etc. Dada la complejidad del problema ha decidido encargaros el trabajo. Partiendo de unas pautas muy precisas dadas por la empresa elabora un programa en JAVA con las siguientes características:

1. Todas las clases necesarias formarán parte de un paquete llamado **telefonica**.

2. La primera de las clases se llamará **Usuario**, las variables principales que caracterizan a un usuario son su nombre (**nombre**), su primer apellido (**apellido1**), su segundo apellido (**apellido1**) y la provincia en donde viven **provincia**. (4 puntos)

3. Se generarán dos constructores. El primero, uno por defecto sin argumentos que permita la inicialización de un objeto de esta clase, y el segundo recibirá como argumentos el nombre, primer apellido, segundo apellido y provincia en la que vive el nuevo usuario. De esta forma se inicializaran todas las variables principales de la clase **Usuario** definidas en el apartado 1. (8 puntos)

4. La segunda de las clases, que también pertenece al paquete **telefonica** se llamará **Datausuarios** y permitirá crear una base de datos para una región en la que se almacenará toda la información relativa a provincias, prefijos de provincia, usuarios, y sus números de teléfono. Por tanto se crearán variables para almacenar el número de provincias de la región (**numprov**), el número máximo de usuarios que soportará el sistema (**numusuarios**), otra variable en la que se almacenen los nombres de las provincias de la región (**provincias**), y otra para los prefijos (**prefijos**) asociados a las provincias. Adicionalmente, se crearán variables para almacenar a los usuarios (**usuarios**) (utilizar la clase **Usuario** creada anteriormente) y sus números de teléfono (**numtelefono**),

respectivamente. **Nota:** Se crearán instancias de las variables pero sin reservar aún espacio en memoria en el caso de utilizar vectores (arrays). (4 puntos)

5. Dado que el sistema podrá actualizarse dando de alta a nuevos usuarios con nuevos números de teléfono, se creará una variable que controle el número de usuarios dados de alta de la clase (**numerousuario**). Nótese que cada vez que se de de alta un usuario nuevo este número tendrá que actualizarse. (2 puntos)

6. Al igual que en el apartado 3 se generarán dos constructores, uno sin argumentos y otro que reciba como argumentos el número de provincias de la región de la cual se va a crear la base de datos, y el número de usuarios que soportará el sistema. Con ellos se inicializarán los valores de las variables de clase pertinentes y se reservará espacio en memoria para aquellos vectores de los cuales se sepa ya la dimensión. Además, se inicializará el valor de la variable creada en el apartado anterior a cero en los dos constructores. (6 puntos)

7. Dentro de la clase **Datausuarios** se generará un método llamado **asignapref** que asigne valores aleatorios a los prefijos de las provincias, para ello se tendrán en cuenta las siguientes consideraciones: (16 puntos)

   a) No recibirá ningún argumento.

   b) Los prefijos serán números enteros comprendidos entre 0 y 999, para crearlos se dispone de la sentencia `(int)(999*Math.random())` que devuelve un número entero aleatorio comprendido entre 0 y 999.

   c) No puede haber dos provincias con el mismo código o prefijo.

8. Para dar valores concretos a los nombres de las provincias se generará otro método que se llamará **generacionprovincias** que reciba como argumentos un vector array que contiene los nombres de las provincias. De esta forma se realizará la asignación de provincias de la clase componente a componente. A su vez, este método procederá a asignar prefijos a cada una de las provincias utilizando el método **asignapref** creado en el apartado anterior. (6 puntos)

9. Para permitir la inclusión de nuevos usuarios se generará un método llamado **altausuario** que recibirá como argumento un objeto de la clase **Usuario** que automáticamente pasará a formar parte de los usuarios asociados a la clase **Datausuarios**. Además, se le asignará un número de teléfono que tendrá las siguientes características: (12 puntos)

   a) Se almacenará también en la base de datos.

   b) Será un número entero comprendido entre 0 y 9999999, para crearlos se dispone de la sentencia `(int)(9999999*Math.random())` que devuelve un número entero aleatorio en ese rango.

   c) No puede haber dos usuarios con el mismo número de teléfono. Para este apartado se dispone del método `public boolean existenumero(int num)` dentro de la clase **Datausuarios** que devuelve que comprueba si el numero **num** que se da

como argumento se corresponde con alguno de los números que hay en la base de datos.

d) Habrá que actualizar la información del número de usuarios disponibles.

10. El sistema permitirá consultar el número de teléfono de un determinado usuario. Para ello se generará un método llamado **consultanumero** que recibirá como argumentos un objeto tipo **Usuario** que busque dentro de los usuarios. Devolverá el número de teléfono de aquel usuario dentro de la base de datos tal que coincida su nombre, primer apellido y segundo apellido con el del usuario dato. **NOTA:** El método **equals(String arg)** de la clase **String** permite comparar si dos cadenas de caracteres son iguales. Devuelve el valor **true** si son iguales y **false** si son distintos. En este caso **arg** es la variable con la que se quiere comparar. (12 puntos)

11. Para probar el programa telefónica os pide que creéis otra clase llamada **Gestion** que contenga el método **main()** y que realice las siguientes acciones:

    a) Crear una base de datos llamada **españa** con dos provincias y un número máximo de usuarios de 500000. (4 puntos)

    b) Crear una variable que contenga las cadenas de caracteres siguientes: "Cantabria" y "Ciudad real". (4 puntos)

    c) Asignar a las provincias de la base de datos **españa** los valores de las provincias almacenadas en la variable creada en el apartado anterior. (4 puntos)

    d) Escribir el siguiente texto: (6 puntos)

    "Los prefijos de las provincias correspondientes a España son: "

    y a continuación

    " Provincia " nombre de la provincia *"; Prefijo ("* su prefijo *")"*

    para cada una de las provincias de la variable **españa**. Se debe haber utilizando algún procedimiento de control de flujo.

    e) Se generará un objeto de la clase **Usuario** creada anteriormente y lo almacenaréis en una variable que sea vuestro nombre con minúsculas. Por ejemplo si te llamas Roberto el nombre de la variable será **roberto**. Para la generación se empleará el constructor que requiere los datos del nombre, primer apellido, segundo apellido y la provincia. Utilizad vuestros datos personales, para la provicia utilizad "Ciudad real". (4 puntos)

    f) Dad de alta al usuario que habéis generado en el apartado anterior para la región **españa** usando alguno de los métodos de los apartados anteriores. (4 puntos)

    g) Escribir el teléfono que el sistema ha asignado a vuestro usuario al darlo de alta. (4 puntos)

Tiempo: 1h 30 min.

Solución:

La empresa **Telefónica España** va a proceder a la implantación de un nuevo sistema de gestión telefónica mediante el cual pueda tener almacenada la información de todos sus usuarios, con la posibilidad de dar de alta a nuevos usuarios, consultar números para usuarios dados de alta, etc. Dada la complejidad del problema ha decidido encargaros el trabajo. Partiendo de unas pautas muy precisas dadas por la empresa elabora un programa en JAVA con las siguientes características:

1. Todas las clases necesarias formarán parte de un paquete llamado **telefonica**.
2. La primera de las clases se llamará **Usuario**, las variables principales que caracterizan a un usuario son su nombre (**nombre**), su primer apellido (**apellido1**), su segundo apellido (**apellido2**) y la provincia en donde viven **provincia**. (4 puntos)

```
package telefonica;

public class Usuario {

   String nombre,apellido1,apellido2,provincia;
```

3. Se generarán dos constructores. El primero, uno por defecto sin argumentos que permita la inicialización de un objeto de esta clase, y el segundo recibirá como argumentos el nombre, primer apellido, segundo apellido y provincia en la que vive el nuevo usuario. De esta forma se inicializaran todas las variables principales de la clase **Usuario** definidas en el apartado 1. (8 puntos)

```
    public Usuario() {
        super();
    }

    public Usuario(String nombre, String apellido1, String apellido2,
            String provincia) {
        super();
        this.nombre=nombre;
        this.apellido1=apellido1;
        this.apellido2=apellido2;
        this.provincia=provincia;
  }

  }// Fin de la clase Usuario
```

4. La segunda de las clases, que también pertenece al paquete **telefonica** se llamará **Datausuarios** y permitirá crear una base de datos para una región en la que se almacenará toda la información relativa a provincias, prefijos de provincia, usuarios, y sus números de teléfono. Por tanto se crearán variables para almacenar el número de provincias de la región (**numprov**), el número máximo de usuarios que soportará el sistema (**numusuarios**), otra variable en la que se almacenen los nombres de las provincias de la

región (**provincias**), y otra para los prefijos (**prefijos**) asociados a las provincias. Adicionalmente, se crearán variables para almacenar a los usuarios (**usuarios**) (utilizar la clase **Usuario** creada anteriormente) y sus números de teléfono (**numtelefono**), respectivamente. **Nota:** Se crearán instancias de las variables pero sin reservar aún espacio en memoria en el caso de utilizar vectores (arrays). (4 puntos)

```
package telefonica;

public class Datausuarios {

    int numprov;
    int numusuarios;

    String provincias [];
    int prefijos [];

    Usuario usuarios [];
    int numtelefono[];
```

5. Dado que el sistema podrá actualizarse dando de alta a nuevos usuarios con nuevos números de teléfono, se creará una variable que controle el número de usuarios dados de alta de la clase (**numerousuario**). Nótese que cada vez que se de de alta un usuario nuevo este número tendrá que actualizarse. (2 puntos)

```
    int numerousuario;
```

6. Al igual que en el apartado 3 se generarán dos constructores, uno sin argumentos y otro que reciba como argumentos el número de provincias de la región de la cual se va a crear la base de datos, y el número de usuarios que soportará el sistema. Con ellos se inicializarán los valores de las variables de clase pertinentes y se reservará espacio en memoria para aquellos vectores de los cuales se sepa ya la dimensión. Además, se inicializará el valor de la variable creada en el apartado anterior a cero en los dos constructores. (6 puntos)

```
    public Datausuarios() {
        super();
        numerousuario=0;
    }

    public Datausuarios(int numprov,int numusuarios) {
        super();
        this.numprov=numprov;
        this.numusuarios=numusuarios;
        this.provincias=new String [numprov];
        this.prefijos=new int [numprov];

        this.usuarios=new Usuario [numusuarios];
        this.numtelefono=new int [numusuarios];

        numerousuario=0;
    }
```

7. Dentro de la clase **Datausuarios** se generará un método llamado **asignapref** que asigne valores aleatorios a los prefijos de las provincias, para ello se tendrán en cuenta las siguientes consideraciones: (16 puntos)

   a) No recibirá ningún argumento.

   b) Los prefijos serán números enteros comprendidos entre 0 y 999, para crearlos se dispone de la sentencia `(int)(999*Math.random())` que devuelve un número entero aleatorio comprendido entre 0 y 999.

   c) No puede haber dos provincias con el mismo código o prefijo.

```
public void asignapref (){
    int aux=0;
    boolean prefok;
    for(int i=0;i<provincias.length;i++){
        prefok=false;
        while(!prefok){
            prefok=true;
            aux=(int)(999*Math.random());
            for(int j=0;j<i;j++){
                if(prefijos[j]==aux){
                    prefok=false;
                    break;
                }
            }
        }
        prefijos[i]=aux;
    }
}
```

8. Para dar valores concretos a los nombres de las provincias se generará otro método que se llamará **generacionprovincias** que reciba como argumentos un vector array que contiene los nombres de las provincias. De esta forma se realizará la asignación de provincias de la clase componente a componente. A su vez, este método procederá a asignar prefijos a cada una de las provincias utilizando el método **asignapref** creado en el apartado anterior. (6 puntos)

```
public void generacionprovincias (String provincia[]){
    for(int i=0;i<provincias.length;i++){
        provincias[i]=provincia[i];
    }
    this.asignapref ();
}
```

9. Para permitir la inclusión de nuevos usuarios se generará un método llamado **altausuario** que recibirá como argumento un objeto de la clase **Usuario** que automáticamente pasará a formar parte de los usuarios asociados a la clase **Datausuarios**. Además, se le asignará un número de teléfono que tendrá las siguientes características: (12 puntos)

a) Se almacenará también en la base de datos.

b) Será un número entero comprendido entre 0 y 9999999, para crearlos se dispone de la sentencia `(int)(9999999*Math.random())` que devuelve un número entero aleatorio en ese rango.

c) No puede haber dos usuarios con el mismo número de teléfono. Para este apartado se dispone del método `public boolean existenumero(int num)` dentro de la clase **Datausuarios** que devuelve que comprueba si el numero **num** que se da como argumento se corresponde con alguno de los números que hay en la base de datos.

d) Habrá que actualizar la información del número de usuarios disponibles.

```
public void altausuario (Usuario nuevo){
    boolean numcorrecto=false;
    while(!numcorrecto){
        int aux=(int)(9999999*Math.random());
        if(!this.existenumero(aux)){
            numtelefono[numerousuario]=aux;
            numcorrecto=true;
        }
    }

    usuarios[numerousuario]=nuevo;
    numerousuario++;
}
```

10. El sistema permitirá consultar el número de teléfono de un determinado usuario. Para ello se generará un método llamado **consultanumero** que recibirá como argumentos un objeto tipo **Usuario** que busque dentro de los usuarios. Devolverá el número de teléfono de aquel usuario dentro de la base de datos tal que coincida su nombre, primer apellido y segundo apellido con el del usuario dato. **NOTA:** El método **equals(String arg)** de la clase **String** permite comparar si dos cadenas de caracteres son iguales. Devuelve el valor **true** si son iguales y **false** si son distintos. En este caso **arg** es la variable con la que se quiere comparar. (12 puntos)

```
public int consultanumero(Usuario dato){
    int num=0;
    if(numerousuario>0){
        for(int i=0;i<numerousuario;i++){
            if(dato.nombre.equals(usuarios[i].nombre) &
                dato.apellido1.equals(usuarios[i].apellido1) &
                dato.apellido2.equals(usuarios[i].apellido2) &
                dato.provincia.equals(usuarios[i].provincia)){

                num=numtelefono[i];
                break;
            }
        }
    }
```

```
        return(num);
    }
}// Fin de la clase Datausuarios
```

11. Para probar el programa telefónica os pide que creéis otra clase llamada **Gestion** que contenga el método **main()** y que realice las siguientes acciones:

    a) Crear una base de datos llamada **españa** con dos provincias y un número máximo de usuarios de 500000. (4 puntos)

    ```
    package telefonica;

    public class Gestion {

        public static void main(String[] args) {

        Datausuarios españa = new Datausuarios(2, 500000);
    ```

    b) Crear una variable que contenga las cadenas de caracteres siguientes: "Cantabria" y "Ciudad Real".

    ```
        String provinciases[]={"Cantabria", "Ciudad Real"};
    ```

    c) Asignar a las provincias de la base de datos **españa** los valores de las provincias almacenadas en la variable creada en el apartado anterior. (4 puntos)

    ```
        españa.generacionprovincias(provinciases);
    ```

    d) Escribir el siguiente texto: (6 puntos)

    "Los prefijos de las provincias correspondientes a España son: "

    y a continuación

    " Provincia " nombre de la provincia *"; Prefijo ("* su prefijo *")"*

    para cada una de las provincias de la variable **españa**. Se debe haber utilizando algún procedimiento de control de flujo.

    ```
        System.out.println("Los prefijos de las provincias
        correspondientes a España son: ");
        for(int i=0;i<españa.provincias.length;i++){
            System.out.println("   Provincia "+españa.provincias[i]
            +"; Prefijo ("+españa.prefijos[i]+")");
        }
    ```

    e) Se generará un objeto de la clase **Usuario** creada anteriormente y lo almacenaréis en una variable que sea vuestro nombre con minúsculas. Por ejemplo si te llamas Roberto el nombre de la variable será **roberto**. Para la generación se empleará el constructor que requiere los datos del nombre, primer apellido, segundo apellido y la provincia. Utilizad vuestros datos personales, para la provicia utilizad "Ciudad real". (4 puntos)

    ```
        Usuario roberto=new Usuario("Roberto","Mínguez","Solana"
        ,"Ciudad Real");
    ```

    f) Dad de alta al usuario que habéis generado en el apartado anterior para la región **españa** usando alguno de los métodos de los apartados anteriores. (4 puntos)

```
españa.altausuario(roberto);
```

g) Escribir el teléfono que el sistema ha asignado a vuestro usuario al darlo de alta. (4 puntos)

```
System.out.println("   El número de teléfono asignado a "
+roberto.nombre+" es "+españa.consultanumero(roberto));
```

■

Práctica A.3 (Extraordinario de 13 de enero de 2006). En ingeniería existen muchos sistemas que están compuestos por "elementos" diferentes, físicamente diferenciables, conectados por sus extremidades o "nudos". Ejemplos de dichos sistemas abundan en ingeniería, por ejemplo, relacionados con las estructuras tenemos los sistemas constituidos por barras, tales como, pórticos, celosías, entramados de edificación, forjados, etc. A todos estos sistemas se les conoce con el nombre de sistemas discretos.

La mayoría de los sistemas discretos pueden analizarse utilizando técnicas de *cálculo matricial* para las cuales el trabajo mediante matrices es imprescindible.

Una matriz consiste en un conjunto ordenado de elementos (números) dispuestos y organizados por filas y por columnas. Suponiendo que los elementos de las matrices que se van a emplear son números reales, se denotará el espacio vectorial de las matrices reales de dimensión $m \times n$ como $\mathbb{R}^{m\times n}$:

$$A \in \mathbb{R}^{m\times n} \Leftrightarrow A = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{mn} \end{bmatrix}. \tag{A.1}$$

donde m es el número de filas y n es el número de columnas.

Para poder tratar de forma eficiente este tipo de elementos se pide crear una clase llamada **Matrix**, que tenga las características siguientes:

1. La clase generada pertenecerá al paquete **matrices (4 puntos)**.

2. Una matriz estará definida por una variable llamada **A** en la que se almacenen los elementos de la matriz y dos variables llamadas **m** y **n** que representan el número de filas y el número de columnas que componen la matriz, respectivamente. En este apartado se creará una instancia para cada una de las variables pero sin reservar espacio en memoria ni asignar valores **(8 puntos)**.

3. El primero de los constructores recibirá como argumentos el número de filas y el número de columnas de la matriz con lo cual se podrán asignar esos valores a las variables de la clase. Adicionalmente se reservará memoria suficiente para almacenar en **A** la matriz de $m \times n$ elementos **(8 puntos)**.

4. El segundo de los constructores recibirá como argumentos el número de filas y el número de columnas de la matriz y un número real almacenado en la variable **s** de

forma que se generará una matriz **A** de $m \times n$ elementos todos iguales al número real **s (8 puntos)**:

$$a_{ij} = s;\ i = 1, \ldots, m;\ j = 1, \ldots, m.$$

5. Se crearán dos métodos públicos que se llamarán **getRowDimension** y **getColumnDimension** que devuelvan el número de filas y el número de columnas de un elemento de la clase, respectivamente **(8 puntos)**.

6. Dispondrá de un método llamado **get** que reciba como argumentos dos numeros indicando una posición de fila i y una posición de columna j y que devuelva el elemento a_{ij} **(8 puntos)**.

7. Para permitir la extracción de submatrices, habrá un método llamada **getMatrix** que permitirá extraer una submatriz. Recibirá como argumentos los indices inicial y final tanto de fila como de columna (i_0, i_1, j_0, j_1) que abarcará la submatriz. Así, por ejemplo, si $(i_0, i_1, j_0, j_1) = (1, 2, 2, 4)$ entonces **(12 puntos)**:

$$\begin{bmatrix} a_{12} & a_{13} & a_{14} \\ a_{22} & a_{23} & a_{24} \end{bmatrix}, \text{ es la submatriz de } \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{4n} \end{bmatrix}. \tag{A.2}$$

8. Otro de los métodos que se llamará **set** permitirá hacer que un elemento de un objeto matriz, definido por su posición (i, j) sea igual a un cierto valor (s) considerado dato: $a_{ij} = s$ **(8 puntos)**.

9. Se dispondrá del método **transpose** que devuelva la matriz traspuesta de la original. Se recuerda que la matriz traspuesta $\mathbf{A}^T$ de una matriz dada **A** es aquella tal que sus elementos $a^T_{ij} = a_{ji}$ **(8 puntos)**.

10. Para poder calcular la norma infinito $||A||_\infty$ de la matriz **A** definida como:

$$||A||_\infty = \underset{i;\ i = 1, \ldots, m}{\text{Máximo}} \sum_{j=1}^{n} |a_{ij}|\ ,$$

donde $|\bullet|$ es el valor absoluto. Es decir, para cada una de las filas de la matriz **A** calculo la suma de los valores absolutos de todos sus elementos y me quedo con el máximo valor de todos. **Nota:** El valor absoluto en JAVA se calcula mediante el método **Math.abs()**. **(12 puntos)**

11. También se dispondrá de un método (**plus**) que dada otra matriz con las mismas dimensiones devuelva una matriz cuyos elementos son la suma de los elementos de las dos matrices. **Nota:** No es necesaria ninguna comprobación sobre el tamaño de las matrices, se supone que la matriz que se da como argumento tiene las mismas dimensiones que la matriz desde la que se llama al método **(8 puntos)**.

12. Por último, se desea crear un método llamado **random** que pueda llamarse a través de la clase sin necesidad de crear un objeto y que devuelva una matriz de dimensiones $m \times n$ que serán dadas como argumento del método. Los elementos de la matriz serán números aleatorios entre 0 y 1. La función **Math.random()** devuelve un número real aleatorio entre 0 y 1. **(8 puntos)**

Tiempo: 1h 30 min.

Solución:

En ingeniería existen muchos sistemas que están compuestos por "elementos" diferentes, físicamente diferenciables, conectados por sus extremidades o "nudos". Ejemplos de dichos sistemas abundan en ingeniería, por ejemplo, relacionados con las estructuras tenemos los sistemas constituidos por barras, tales como, pórticos, celosías, entramados de edificación, forjados, etc. A todos estos sistemas se les conoce con el nombre de sistemas discretos.

La mayoría de los sistemas discretos pueden analizarse utilizando técnicas de *cálculo matricial* para las cuales el trabajo mediante matrices es imprescindible.

Una matriz consiste en un conjunto ordenado de elementos (números) dispuestos y organizados por filas y por columnas. Suponiendo que los elementos de las matrices que se van a emplear son números reales, se denotará el espacio vectorial de las matrices reales de dimensión $m \times n$ como $\mathbb{R}^{m \times n}$:

$$A \in \mathbb{R}^{m \times n} \Leftrightarrow A = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{mn} \end{bmatrix}. \tag{A.3}$$

donde m es el número de filas y n es el número de columnas.

Para poder tratar de forma eficiente este tipo de elementos se pide crear una clase llamada **Matrix**, que tenga las características siguientes:

1. La clase generada pertenecerá al paquete **matrices (4 puntos)**.

```
package matrices;
```

2. Una matriz estará definida por una variable llamada **A** en la que se almacenen los elementos de la matriz y dos variables llamadas **m** y **n** que representan el número de filas y el número de columnas que componen la matriz, respectivamente. En este apartado se creará una instancia para cada una de las variables pero sin reservar espacio en memoria ni asignar valores **(8 puntos)**.

```
public class Matrix  {

    double[][] A;
    int m, n;
```

3. El primero de los constructores recibirá como argumentos el número de filas y el número de columnas de la matriz con lo cual se podrán asignar esos valores a las variables de la clase. Adicionalmente se reservará memoria suficiente para almacenar en **A** la matriz de $m \times n$ elementos **(8 puntos)**.

```
public Matrix (int m, int n) {
    this.m = m;
    this.n = n;
    A = new double[m][n];
}
```

4. El segundo de los constructores recibirá como argumentos el número de filas y el número de columnas de la matriz y un número real almacenado en la variable **s** de forma que se generará una matriz **A** de $m \times n$ elementos todos iguales al número real **s (8 puntos)**:

$$a_{ij} = s;\ i = 1, \ldots, m;\ j = 1, \ldots, m.$$

```
public Matrix (int m, int n, double s) {
    this.m = m;
    this.n = n;
    A = new double[m][n];
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
        A[i][j] = s;
        }
    }
}
```

5. Se crearán dos métodos públicos que se llamarán **getRowDimension** y **getColumnDimension** que devuelvan el número de filas y el número de columnas de un elemento de la clase, respectivamente **(8 puntos)**.

```
public int getRowDimension () {
    return m;
}

public int getColumnDimension () {
    return n;
}
```

6. Dispondrá de un método llamado **get** que reciba como argumentos dos numeros indicando una posición de fila i y una posición de columna j y que devuelva el elemento a_{ij} **(8 puntos)**.

```
public double get (int i, int j) {
    return A[i][j];
}
```

7. Para permitir la extracción de submatrices, habrá un método llamada **getMatrix** que permitirá extraer una submatriz. Recibirá como argumentos los indices inicial y final tanto de fila como de columna (i_0, i_1, j_0, j_1) que abarcará la submatriz. Así, por ejemplo, si $(i_0, i_1, j_0, j_1) = (1, 2, 2, 4)$ entonces **(12 puntos)**:

$$\begin{bmatrix} a_{12} & a_{13} & a_{14} \\ a_{22} & a_{23} & a_{24} \end{bmatrix}, \text{ es la submatriz de } \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{4n} \end{bmatrix}. \tag{A.4}$$

```
public Matrix getMatrix (int i0, int i1, int j0, int j1) {
    Matrix X = new Matrix(i1-i0+1,j1-j0+1);
    for (int i = i0; i <= i1; i++) {
    for (int j = j0; j <= j1; j++) {
        X.A[i-i0][j-j0] = A[i-1][j-1];
    }
    }
    return X;
}
```

8. Otro de los métodos que se llamará **set** permitirá hacer que un elemento de un objeto matriz, definido por su posición (i, j) sea igual a un cierto valor (s) considerado dato: $a_{ij} = s$ **(8 puntos)**.

```
public void set (int i, int j, double s) {
    A[i][j] = s;
}
```

9. Se dispondrá del método **transpose** que devuelva la matriz traspuesta de la original. Se recuerda que la matriz traspuesta $\mathbf{A}^T$ de una matriz dada $\mathbf{A}$ es aquella tal que sus elementos $a^T_{ij} = a_{ji}$ **(8 puntos)**.

```
public Matrix transpose () {
    Matrix X = new Matrix(n,m);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
        X.A[j][i] = A[i][j];
        }
    }
    return X;
}
```

10. Para poder calcular la norma infinito $||A||_\infty$ de la matriz $\mathbf{A}$ definida como:

$$||A||_\infty = \underset{i;\ i=1,\ldots,m}{\text{Máximo}} \sum_{j=1}^{n} |a_{ij}| \ ,$$

donde $|\bullet|$ es el valor absoluto. Es decir, para cada una de las filas de la matriz $\mathbf{A}$ calculo la suma de los valores absolutos de todos sus elementos y me quedo con el máximo valor de todos. **Nota:** El valor absoluto en JAVA se calcula mediante el método **Math.abs()**. **(12 puntos)**

```
public double normInf () {
    double f = 0;
    for (int i = 0; i < m; i++) {
        double s = 0;
        for (int j = 0; j < n; j++) {
            s += Math.abs(A[i][j]);
        }
        f = Math.max(f,s);
```

```
        }
        return f;
    }
```

11. También se dispondrá de un método (**plus**) que dada otra matriz con las mismas dimensiones devuelva una matriz cuyos elementos son la suma de los elementos de las dos matrices. **Nota:** No es necesaria ninguna comprobación sobre el tamaño de las matrices, se supone que la matriz que se da como argumento tiene las mismas dimensiones que la matriz desde la que se llama al método **(8 puntos)**.

```
    public Matrix plus (Matrix B) {
        Matrix X = new Matrix(m,n);
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                X.A[i][j] = A[i][j] + B.A[i][j];
            }
        }
        return X;
    }
```

12. Por último, se desea crear un método llamado **random** que pueda llamarse a través de la clase sin necesidad de crear un objeto y que devuelva una matriz de dimensiones $m \times n$ que serán dadas como argumento del método. Los elementos de la matriz serán números aleatorios entre 0 y 1. La función **Math.random()** devuelve un número real aleatorio entre 0 y 1. **(8 puntos)**

```
    public static Matrix random (int m, int n) {
        Matrix X = new Matrix(m,n);
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                X.A[i][j] = Math.random();
            }
        }
        return X;
    }
}
```

■

Práctica A.4 (Parcial de 1 de Diciembre de 2006). Se va a proceder a elaborar un programa para la resolución y gestión de Su Dokus en una revista. Un Su Doku consiste en un tablero generalmente de 9×9 casillas divididas a su vez en bloques de 3×3 tal y como se muestra en la Figura A.1. El propósito del juego consiste en ubicar 9 símbolos cualesquiera en el tablero de forma que tanto en las 9 filas, como en las 9 columnas, como en los 9 bloques de 3×3 no haya ninguno repetido, respectivamente. Por simplicidad y para facilitar el manejo se emplean como símbolos los números enteros $1, 2, 3, 4, 5, 6, 7, 8, 9$. En la Figura A.1 se puede observar que en todas las filas, columnas y bloques de 3×3 están contenidos los números enteros del 1 al 9 sin repetir.

Obviamente este Su Doku está resuelto y el objetivo del programa es resolver un Su Doku del que sólo se conocen los valores de algunas de las casillas, tal y como se muestra

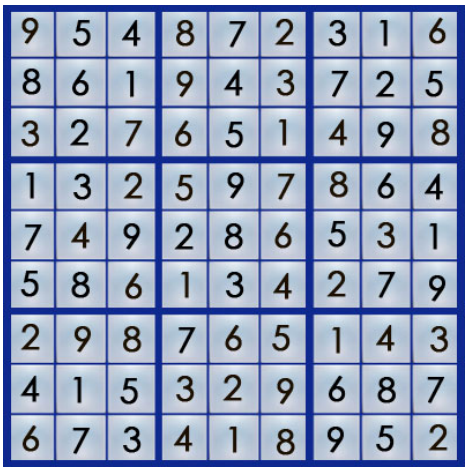


Figura A.1: Representación del problema del Su Doku una vez resuelto.

en la Figura A.2. Nótese que a la hora de resolver el Su Doku en el programa una casilla en la que se desconoce su valor contendrá un cero (0).

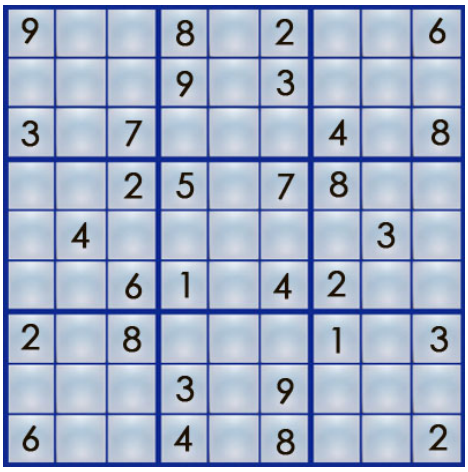


Figura A.2: Ejemplo de datos iniciales para la resolución del Su Doku.

Para la elaboración del programa se comienza creando una clase denominada **Sudoku** que tiene las características siguientes:

1. La clase pertenece al paquete **misudoku**. (2 puntos)

2. Las variables de instancia que caracterizan un objeto de la clase **Sudoku** son el número de filas y el número de columnas, que deben de ser declaradas de forma que no puedan modificarse de forma alguna, y una matriz que almacene el valor de cada una de las casillas del Su Doku, su nombre es **cells**. Inicialmente sólo se declara la variable sin inicializarla. (4 puntos)

3. Se genera un constructor para permitir la inicialización de un objeto de esta clase sin dar ningún tipo de argumento mediante la siguiente sentencia (2 puntos)

```
super();
```

4. El segundo constructor recibe una matriz que se llama **celda** cuyos elementos son del mismo tipo que los elementos de la variable **cells**, y se encarga de: (10 puntos)

   a) Reservar espacio en memoria para la variable **cells**.

   b) Comprobar si las dimensiones de la matriz **celda** (número de filas y de columnas) son iguales a las variables de instancia en las que está almacenada el número de filas y de columnas de los objetos de la clase **Sudoku**, respectivamente. En caso de que sean correctas se continua la ejecución, si no lo son se escribe un mensaje en pantalla que diga:

   ```
   "Las dimensiones de los datos no se ajustan a las dimensiones de Sudoku, compruébelo."
   ```

   c) Si las dimensiones son correctas se procede a asignar valores a la variable **cells** teniendo en cuenta que los valores de los datos están en la variable **celda**. Para la asignación (véase la Figura A.3) se comprueba que cada uno de los datos toma valores entre 0 y 9, en caso de que alguno de ellos no lo cumpla se escribe un mensaje en pantalla que diga:

   ```
   "Alguno de los datos está fuera del rango."
   ```

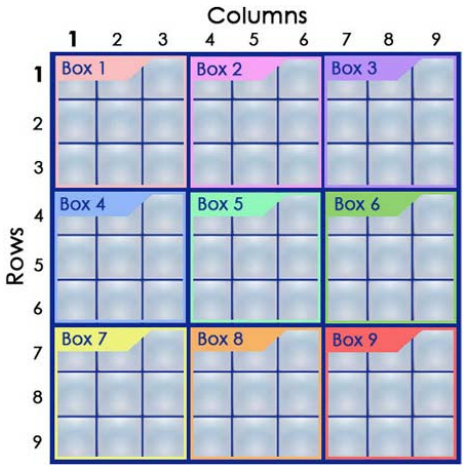


Figura A.3: Distribución por filas, columnas y bloques.

   Y posteriormente se detiene el procedimiento de asignación, ya que no tiene sentido seguir asignando valores si alguno está mal.
   Nota: El comando break sólo sale del ciclo o bucle desde el que se le invoca.

5. El tercer constructor recibe una lista que se llama **cadena** compuesta por elementos `String` y se encarga de: (10 puntos)

   a) Reservar espacio en memoria para la variable **cells**.

b) Comprobar si la dimensión de la lista **cadena** concuerda con el número de elementos total del Su Doku. En caso de que sea correcta se continua la ejecución, si no lo son se escribe un mensaje en pantalla que diga:

```
La dimensión de los datos no se ajusta a las dimensiones de Sudoku, compruébelo.
```

c) Si la dimensión es correcta se procederá a asignar valores a la variable **cells** teniendo en cuenta que se asignan valores comenzando por la primera fila del Su Doku, luego la segunda fila, y así sucesivamente hasta haber asignado todos los elementos de la lista **cadena**. Para la asignación hay que tener en cuenta que existe un método tipo `static` de la clase `Integer` llamado `parseInt(String a)` que devuelve el número representado por "a" pero de tipo `int`. Además se ha de comprobar que cada uno de los datos toma valores entre 0 y 9, en caso de que alguno de ellos no lo cumpla se escribe un mensaje en pantalla que diga:

```
"Alguno de los datos está fuera del rango."
```

Y posteriormente se detiene el procedimiento de asignación, ya que no tiene sentido seguir asignando valores si alguno está mal.
Nota: El comando break sólo sale del ciclo o bucle desde el que se le invoca.

6. La clase contiene un método que se llama **contarblancos** que devuelve el número de elementos sin resolver en el Su Doku correspondiente, es decir, el número de elementos que son cero. (7 puntos)

7. Para ayudar a la resolución se crea un método llamado **elementline** que comprueba si en una fila determinada dada como argumento con el nombre `i` es posible poner el número `num` dado también como argumento. Por ejemplo, tal y como se muestra en la Figura A.4, con esta rutina se podría comprobar si en la fila tercera se puede colocar el número 6 en alguno de sus elementos. En este caso el método debería que contestar que **sí** ya que no existe ningún otro 6 en esa fila. (10 puntos)

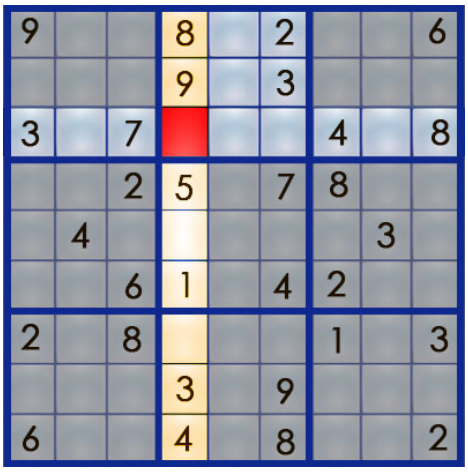


Figura A.4: Ejemplo para mostrar la utilidad de los métodos **elementline** y **elementcolumn**.

8. Análogamente, se crea un método llamado **elementcolumn** que comprueba si en una columna determinada dada como argumento con el nombre `j` es posible poner el número `num` dado también como argumento. Por ejemplo, tal y como se muestra en la Figura A.4, con esta rutina se podría comprobar si en la columna cuarta se puede colocar el número 6 en alguno de sus elementos. En este caso el método debería que contestar que **sí** ya que no existe ningún otro 6 en esa columna. (10 puntos)

9. A continuación vamos a proceder a generar un método llamado **resolver** que supuestamente se encarga de sustituir los ceros del Su Doku por los elementos correctos de forma que se cumpla la condición de no repetición de los números ni por filas, ni por columnas, ni por bloques. Debido a la complejidad de la resolución, lo único que hará el método será escribir el siguiente mensaje cada vez que se invoque:

   ```
   "Problema resuelto."
   ```

   Aún así definiremos el método y se supone que además de la escritura se incluyen las sentencias necesarias para resolverlo, es decir, que pese a que no añadís el código para resolverlo, si invocáis el método se tendrá en la variable correspondiente la solución. (5 puntos)

10. Por último, se crea un método que escribe los valores actuales de las casillas del Su Doku. La rutina ha de escribir cada fila en una línea distinta y para distinguir los bloques se dejará una línea en blanco entre bloques consecutivos, ya sea por filas o por columnas. Dentro de cada bloque de 3×3 y si el elemento es cero, se sustituye por un blanco. A continuación se describe un ejemplo del resultado final deseado:

    ```
    9     8  2     6
          9  3
    3  7  6     4  8

       2  5  7  8
     4              3
       6  1  4  2

    2  8           1  3
          3  9
    6     4  8     2
    ```

    Para este apartado puede ser de utilidad el método `System.out.print("texto")` que escribe lo que haya entre paréntesis sin pasar de línea, es decir, lo que se escriba después se escribe inmediatamente después de `texto` en la misma línea. (10 puntos)

A continuación se genera una segunda clase llamada **Principal** que contiene el método `main`. Todo lo que se pide a continuación se supone dentro del método `main` y no hace falta que generéis el código correspondiente:

1. Se crea una matriz en la que se almacenan los elementos del Su Doku mostrado en la Figura A.2. Recuérdese que las casillas en blanco se corresponden con ceros. (4 puntos)

2. Se crea un objeto de la clase **Sudoku** en una variable que se llame como vuestro nombre. Hay que utilizar uno de los dos constructores de la clase **Sudoku**. (4 puntos)

3. Escribir en pantalla el Su Doku empleando una única sentencia. (4 puntos)

4. Escribir en pantalla el número de elementos blancos que tiene el Su Doku de la siguiente manera:(4 puntos)

```
El número de elementos a resolver es 51.
```

5. Comprobar si se puede colocar el número 6 en la fila tercera, columna cuarta y en caso de que la respuesta sea positiva asignar el valor 6 a la correspondiente fila y columna y escribir el siguiente mensaje: (6 puntos)

```
El número 6 es un posible candidato para ocupar la posición (3,4).
```

   En caso contrario escribir el siguiente mensaje:

```
En número 6 no es válido, ya está repetido en la fila 3 y/o en la columna 4.
```

6. Resolver el Su Doku. (4 puntos)

7. Escribir la solución final en pantalla en un único comando. (4 puntos)

Tiempo: 2h.

Solución:

Se va a proceder a elaborar un programa para la resolución y gestión de Su Dokus en una revista. Un Su Doku consiste en un tablero generalmente de 9×9 casillas divididas a su vez en bloques de 3×3 tal y como se muestra en la Figura A.5. El propósito del juego consiste en ubicar 9 símbolos cualesquiera en el tablero de forma que tanto en las 9 filas, como en las 9 columnas, como en los 9 bloques de 3×3 no haya ninguno repetido, respectivamente. Por simplicidad y para facilitar el manejo se emplean como símbolos los números enteros $1, 2, 3, 4, 5, 6, 7, 8, 9$. En la Figura A.5 se puede observar que en todas las filas, columnas y bloques de 3×3 están contenidos los números enteros del 1 al 9 sin repetir.

Obviamente este Su Doku está resuelto y el objetivo del programa es resolver un Su Doku del que sólo se conocen los valores de algunas de las casillas, tal y como se muestra en la Figura A.6. Nótese que a la hora de resolver el Su Doku en el programa una casilla en la que se desconoce su valor contendrá un cero (0).

Para la elaboración del programa se comienza creando una clase denominada **Sudoku** que tiene las características siguientes:

1. La clase pertenece al paquete **misudoku**. (2 puntos)

```
package misudoku;
```

2. Las variables de instancia que caracterizan un objeto de la clase **Sudoku** son el número de filas y el número de columnas, que deben de ser declaradas de forma que no puedan modificarse de forma alguna, y una matriz que almacene el valor de cada una de las casillas del Su Doku, su nombre es **cells**. Inicialmente sólo se declara la variable sin inicializarla. (4 puntos)

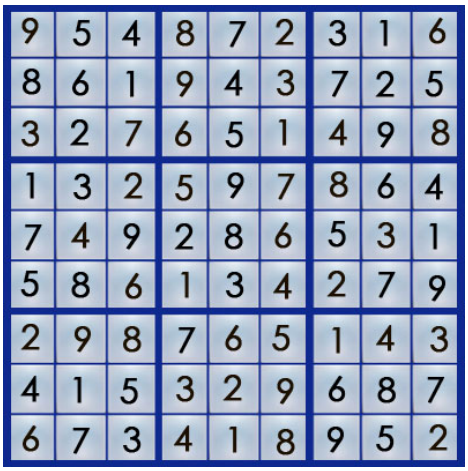


Figura A.5: Representación del problema del Su Doku una vez resuelto.

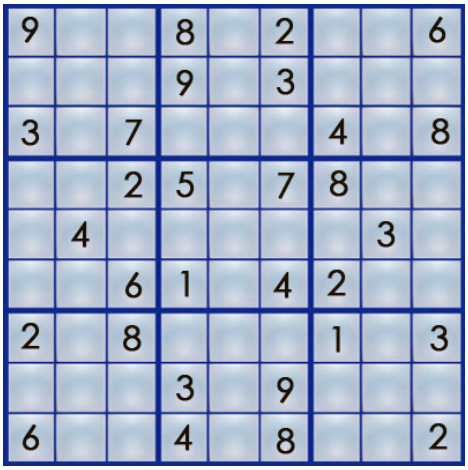


Figura A.6: Ejemplo de datos iniciales para la resolución del Su Doku.

```
public class Sudoku {

  final int nrows = 9;
  final int ncols = 9;

  int cells [][];
```

3. Se genera un constructor para permitir la inicialización de un objeto de esta clase sin dar ningún tipo de argumento mediante la siguiente sentencia (2 puntos)

```
public Sudoku() {
    super();
```

```
}
```

4. El segundo constructor recibe una matriz que se llama **celda** cuyos elementos son del mismo tipo que los elementos de la variable **cells**, y se encarga de: (10 puntos)

   a) Reservar espacio en memoria para la variable **cells**.

   b) Comprobar si las dimensiones de la matriz **celda** (número de filas y de columnas) son iguales a las variables de instancia en las que está almacenada el número de filas y de columnas de los objetos de la clase **Sudoku**, respectivamente. En caso de que sean correctas se continua la ejecución, si no lo son se escribe un mensaje en pantalla que diga:

   ```
   "Las dimensiones de los datos no se ajustan a las dimensiones de Sudoku, compruébelo."
   ```

   c) Si las dimensiones son correctas se procede a asignar valores a la variable **cells** teniendo en cuenta que los valores de los datos están en la variable **celda**. Para la asignación (véase la Figura A.7) se comprueba que cada uno de los datos toma valores entre 0 y 9, en caso de que alguno de ellos no lo cumpla se escribe un mensaje en pantalla que diga:

   ```
   "Alguno de los datos está fuera del rango."
   ```

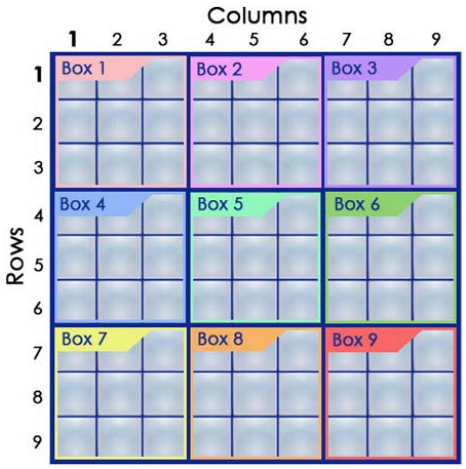


Figura A.7: Distribución por filas, columnas y bloques.

Y posteriormente se detiene el procedimiento de asignación, ya que no tiene sentido seguir asignando valores si alguno está mal.
Nota: El comando break sólo sale del ciclo o bucle desde el que se le invoca.

```
public Sudoku(int[][] celda){
cells = new int[nrows][ncols];
int nr = celda.length;
int nc = celda[0].length;
boolean incorrecto = false;
if(nr != nrows || nc != ncols){
    System.out.println("La dimensión de los datos no se ajusta a las"
```

```
                    + " dimensiones de Sudoku, compruébelo.");
} else {
    for(int i=1;i<=nr;i++){
    for(int j=1;j<=nr;j++){
        if(celda[i-1][j-1]>=0 && celda[i-1][j-1]<=9){
        cells[i-1][j-1]=celda[i-1][j-1];
        }else{
        System.out.println("Alguno de los datos está fuera del rango.");
        incorrecto = true;
        break;
        }
    }
    if (incorrecto) break;
    }
}
}
```

5. El tercer constructor recibe una lista que se llama **cadena** compuesta por elementos `String` y se encarga de: (10 puntos)

   a) Reservar espacio en memoria para la variable **cells**.

   b) Comprobar si la dimensión de la lista **cadena** concuerda con el número de elementos total del Su Doku. En caso de que sea correcta se continua la ejecución, si no lo son se escribe un mensaje en pantalla que diga:

   ```
   La dimensión de los datos no se ajusta a las dimensiones de Sudoku, compruébelo.
   ```

   c) Si la dimensión es correcta se procederá a asignar valores a la variable **cells** teniendo en cuenta que se asignan valores comenzando por la primera fila del Su Doku, luego la segunda fila, y así sucesivamente hasta haber asignado todos los elementos de la lista **cadena**. Para la asignación hay que tener en cuenta que existe un método tipo `static` de la clase `Integer` llamado `parseInt(String a)` que devuelve el número representado por "a" pero de tipo `int`. Además se ha de comprobar que cada uno de los datos toma valores entre 0 y 9, en caso de que alguno de ellos no lo cumpla se escribe un mensaje en pantalla que diga:

   ```
   "Alguno de los datos está fuera del rango."
   ```

   Y posteriormente se detiene el procedimiento de asignación, ya que no tiene sentido seguir asignando valores si alguno está mal.

   Nota: El comando break sólo sale del ciclo o bucle desde el que se le invoca.

```
public Sudoku(String [] cadena){
cells = new int[nrows][ncols];
int n = cadena.length;
int val;
if(n != (nrows*ncols)){
    System.out.println("Las dimensiones de los datos no se ajustan "
                    + " a las dimensiones de Sudoku, compruébelo");
} else {
    for(int i=1;i<=nrows;i++){
    for(int j=1;j<=ncols;j++){
```

```
            val=Integer.parseInt(cadena[(i-1)*nrows+j-1]);
            if(val>=0 && val<=9){
            cells[i-1][j-1]=val;
            }else{
            System.out.println("El dato número "+((i-1)*nrows+j)
                            + " de la cadena de entrada está fuera de rango");
            break;
            }
        }
        }
}
}
```

6. La clase contiene un método que se llama **contarblancos** que devuelve el número de elementos sin resolver en el Su Doku correspondiente, es decir, el número de elementos que son cero. (7 puntos)

```
public int contarblancos (){
    int numb = 0;
    for(int i=1;i<=nrows;i++){
        for(int j=1;j<=ncols;j++){
            if(cells[i-1][j-1]==0){
            numb+=1;
            }
        }
    }
    return(numb);
}
```

7. Para ayudar a la resolución se crea un método llamado **elementline** que comprueba si en una fila determinada dada como argumento con el nombre `i` es posible poner el número `num` dado también como argumento. Por ejemplo, tal y como se muestra en la Figura A.8, con esta rutina se podría comprobar si en la fila tercera se puede colocar el número 6 en alguno de sus elementos. En este caso el método debería que contestar que **sí** ya que no existe ningún otro 6 en esa fila. (10 puntos)

```
public boolean elementline (int i, int num){
    boolean encontrado=false;
    int contador = 1;
    while (!encontrado && contador<=ncols){
        if(cells[i-1][contador-1]==num){
        encontrado = true;
        break;
        }
        contador +=1;
    }
```

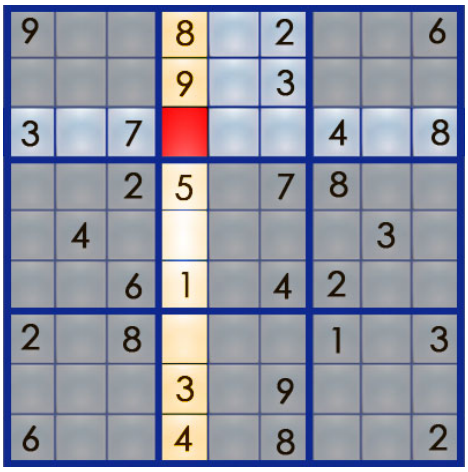


Figura A.8: Ejemplo para mostrar la utilidad de los métodos **elementline** y **elementcolumn**.

```
    return(!encontrado);
}
```

8. Análogamente, se crea un método llamado **elementcolumn** que comprueba si en una columna determinada dada como argumento con el nombre `j` es posible poner el número `num` dado también como argumento. Por ejemplo, tal y como se muestra en la Figura A.8, con esta rutina se podría comprobar si en la columna cuarta se puede colocar el número 6 en alguno de sus elementos. En este caso el método debería que contestar que **sí** ya que no existe ningún otro 6 en esa columna. (10 puntos)

```
public boolean elementcolumn (int j, int num){
    boolean encontrado=false;
    int contador = 1;
    while (!encontrado && contador<=nrows){
      if(cells[contador-1][j-1]==num){
        encontrado = true;
        break;
      }
      contador +=1;
    }
    return(!encontrado);
}
```

9. A continuación vamos a proceder a generar un método llamado **resolver** que supuestamente se encarga de sustituir los ceros del Su Doku por los elementos correctos de forma que se cumpla la condición de no repetición de los números ni por filas, ni por columnas, ni por bloques. Debido a la complejidad de la resolución, lo único que hará el método será escribir el siguiente mensaje cada vez que se invoque:

```
"Problema resuelto."
```

Aún así definiremos el método y se supone que además de la escritura se incluyen las sentencias necesarias para resolverlo, es decir, que pese a que no añadís el código para resolverlo, si invocáis el método se tendrá en la variable correspondiente la solución. (5 puntos)

```
public void resolver (){
  System.out.println("Problema resuelto.");
}
```

10. Por último, se crea un método que escribe los valores actuales de las casillas del Su Doku. La rutina ha de escribir cada fila en una línea distinta y para distinguir los bloques se dejará una línea en blanco entre bloques consecutivos, ya sea por filas o por columnas (). Dentro de cada bloque de 3×3 y si el elemento es cero, se sustituye por un blanco (). A continuación se describe un ejemplo del resultado final deseado:

```
9     8  2     6
      9  3
3  7  6     4  8

   2  5  7  8
 4           3
   6  1  4  2

2  8        1  3
      3  9
6     4  8     2
```

Para este apartado puede ser de utilidad el método `System.out.print("texto")` que escribe lo que haya entre paréntesis sin pasar de línea, es decir, lo que se escriba después se escribe inmediatamente después de `texto` en la misma línea. (10 puntos)

```
public void writeMatrix() {
      for (int i = 0; i < nrows; ++i) {
          if (i % 3 == 0){
            System.out.println();
          }
          for (int j = 0; j < ncols; ++j) {
              if (j % 3 == 0) {
                System.out.print(" ");
              }
              if (cells[i][j]==0){
                System.out.print("  ");
              } else {
                System.out.print(cells[i][j] + " ");
              }
          }
```

```
            System.out.println("");
        }
        System.out.println(" ");
    }

}// Fin la clase Sudoku
```

A continuación se genera una segunda clase llamada **Principal** que contiene el método `main`. Todo lo que se pide a continuación se supone dentro del método `main` y no hace falta que generéis el código correspondiente:

1. Se crea una matriz en la que se almacenan los elementos del Su Doku mostrado en la Figura A.6. Recuérdese que las casillas en blanco se corresponden con ceros. (4 puntos)

```
int a [][]={{9,0,0,8,0,2,0,0,6},
            {0,0,0,9,0,3,0,0,0},
            {3,0,7,0,0,0,4,0,8},
            {0,0,2,5,0,7,8,0,0},
            {0,4,0,0,0,0,0,3,0},
            {0,0,6,1,0,4,2,0,0},
            {2,0,8,0,0,0,1,0,3},
            {0,0,0,3,0,9,0,0,0},
            {6,0,0,4,0,8,0,0,2}};
```

2. Se crea un objeto de la clase **Sudoku** en una variable que se llame como vuestro nombre. Hay que utilizar uno de los dos constructores de la clase **Sudoku**. (4 puntos)

```
Sudoku roberto = new Sudoku(a);
```

3. Escribir en pantalla el Su Doku empleando una única sentencia. (4 puntos)

```
roberto.writeMatrix();
```

4. Escribir en pantalla el número de elementos blancos que tiene el Su Doku de la siguiente manera:(4 puntos)

```
El número de elementos a resolver es ?.
```

```
System.out.println("El número de elementos a resolver es "
                   + roberto.contarblancos()+".");
```

5. Comprobar si se puede colocar el número 6 en la fila tercera, columna cuarta y en caso de que la respuesta sea positiva asignar el valor 6 a la correspondiente fila y columna y escribir el siguiente mensaje: (6 puntos)

```
El número 6 es un posible candidato para ocupar la posición (3,4).
```

En caso contrario escribir el siguiente mensaje:

```
En número 6 no es válido, ya está repetido en la fila 3 y/o en la columna 4.
```

```
if(roberto.elementline (3,6) && roberto.elementcolumn (4,6)){
    System.out.println("El número 6 es un posible candidato para"
                    + " ocupar la posición (3,4).");
    roberto.cells[2][3]=6;
} else {
    System.out.println("En número 6 no es válido, ya está repetido"
                    + " en la fila 3 y/o en la columna 4.");
}
```

6. Resolver el Su Doku. (4 puntos)

```
roberto.resolver ();
```

7. Escribir la solución final en pantalla en un único comando. (4 puntos)

```
System.out.println("La solución del Sudoku actual es:");
roberto.writeMatrix();
```

Tiempo: 2h.

■

Práctica A.5 (Final de 16 de diciembre de 2006). Se va a proceder a generar un programa para la gestión de las quinielas de fútbol. Cada una de las quinielas tiene 14 partidos mas el pleno al quince. Para los primeros 14 partidos hay un total de 8 pronósticos como máximo (no es necesario completar todos), y dentro de cada pronóstico el resultado del partido puede ser '`1`', '`X`' y '`2`', tal y como se muestra en la Figura A.9. Se puede seleccionar para cada partido un único resultado (apuesta simple), dos resultados (apuesta doble) o tres resultados (apuesta triple).

Para la elaboración del programa se comienza creando una clase denominada **Quiniela** que tiene las características siguientes:

1. La clase pertenece al paquete **quiniela**. (2 puntos)

2. Las variables de instancia que caracterizan un objeto de la clase **Quiniela** son el número de partidos sin contar el pleno al quince, y el número máximo de pronósticos, las variables se llamarán **partidos** y **pronosticos**, respectivamente. Tendrán un valor fijo que no se puede modificar y que deberá fijarse en el momento de la declaración de las variables, y serán comunes para todos los elementos de la clase **Quiniela**. (4 puntos)

Figura A.9: Representación de la quiniela, con sus 14 partidos, 8 pronósticos y su pleno al quince.

3. Dado que no es necesario completar todos los pronósticos, se declarará una variable llamada **numeropronosticos** en el que posteriormente se almacenará el número de pronósticos para una quiniela sellada. Además, para almacenar la información de cada quiniela se declarará la variable **apuestas** (véase la Figura A.10) que tendrá tantas filas como número de partidos sin contar el pleno al quince y de columnas el número de pronósticos multiplicado por tres posibles resultados para cada partido. El sistema reconocerá cada apuesta si en la casilla correspondiente hay un 1, en caso contrario habrá un cero, lo que implica que no hay apuesta en esa casilla. Para almacenar el número de partidos en los que se acierta el resultado para cada pronóstico, se declara una variable llamada **aciertos**. Y por último para almacenar el pleno se declara e inicializa la variable **pleno** que contendrá la información de que resultado se selecciona para el partido del pleno al quince. Inicialmente se supone que no se ha seleccionado ningún resultado. (4 puntos)

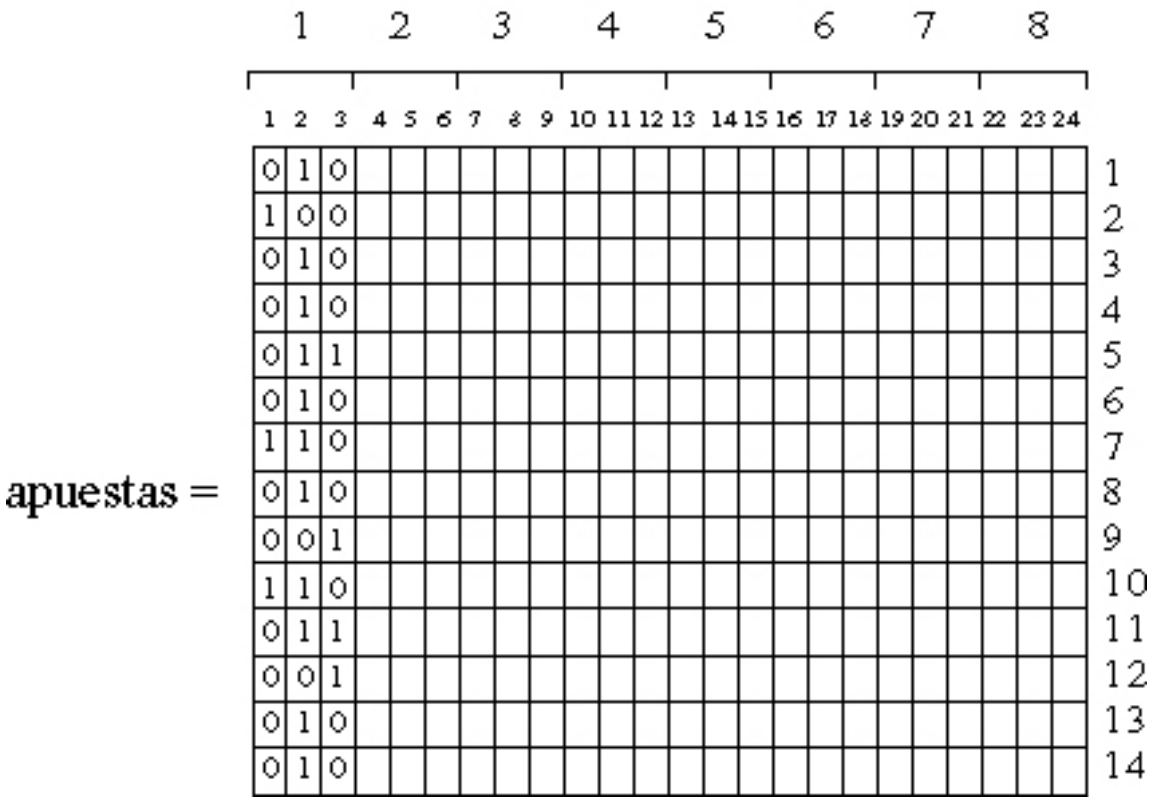


Figura A.10: Representación de la variable **apuestas** para el caso de la quiniela de la Figura A.9.

NOTA: Para la realización del examen puede ser especialmente útil la siguiente expresión

$$(k-1)*3+j,$$

que permite conocer la columna de la matriz almacenada en la variable **apuestas** suponiendo que se está en el bloque número k y que j toma valores 1, 2, o 3, correspondientes, respectivamente, con los valores '1', 'X' y '2'. Así por ejemplo, el resultado asociado a la 'X' del pronóstico 3 está en la columna $(3-1)*3+2 = 8$ (octava columna).

4. Se genera un constructor para permitir la inicialización de un objeto de esta clase sin dar ningún tipo de argumento mediante la sentencia `super()`. (2 puntos)

5. El segundo constructor recibe un número entero que se corresponde con el número de pronósticos que se quieren rellenar, y se encarga de: (10 puntos)

   a) Reservar espacio en memoria para la variable **apuestas**.

   b) En segundo lugar se inicializarán los valores de las apuestas para el número de pronósticos dado como argumento, de forma que se seleccionará para cada partido una única apuesta (apuesta simple). Para escoger el resultado se emplea la

expresión `(int) Math.ceil(3*Math.random())` que devuelve un número aleatorio entre 1 y 3, de forma que si sale 1 significa que se ha seleccionado el `'1'` y ha de ponerse un uno en la primera columna correspondiente al pronóstico, si sale 2 significa que se ha seleccionado la `'X'` y ha de ponerse un uno en la segunda columna correspondiente al pronóstico, y si sale 3 significa que se ha seleccionado el `'2'` y ha de ponerse un uno en la tercera columna correspondiente al pronóstico. Para los resultados no elegidos su valor correspondiente es cero.

c) Por ultimo se selecciona el resultado del pleno al quince empleando la misma expresión `(int) Math.ceil(3*Math.random())`, poniendo un 1 en la posición indicada por ese valor y cero en las otras dos casillas.

6. El tercer constructor recibe como argumentos una lista de listas con las apuestas, y una lista de tres elementos con la información del pleno al quince. (10 puntos)

   a) Reservar espacio en memoria para la variable **apuestas** de la clase.

   b) Comprobar si el número de filas de la variable con la información de las apuestas concuerda con el número de partidos de la clase **Quiniela**. En caso de que sea correcto se continua la ejecución, si no lo es se escribe un mensaje en pantalla que diga:

   ```
   El número de partidos es incorrecto.
   ```

   c) Si la dimensión es correcta se procederá a asignar valores a la variable **apuestas** pero sólo con los datos disponibles.

   d) Asignar valores a la variable **pleno** de la clase **Quiniela**.

7. La clase contiene un método que se llama **aciertospronosticos** que recibe como argumento una lista con 14 cadenas de caracteres en los que está la solución de la quiniela `'1'`, `'X'` o `'2'` para cada uno de los partidos. Con esa información se recorrerán todos los pronósticos y se contarán el número de acietos, almacenándolos en la variable **aciertos**. (8 puntos)

8. Se creará otro método llamado **aciertopleno** que recibe como argumento un string con el resultado `'1'`, `'X'` o `'2'` correcto del pleno al quince, que devuelve "verdadero" si se ha acertado, y "falso" en caso contrario. (8 puntos)

9. Otro de los métodos calcula el número máximo de aciertos almacenado en la variable **aciertos** para los pronósticos en los que hay apuestas. (8 puntos)

10. El siguiente de los métodos llamado **numsimples** calcula el número de apuestas sencillas de la quiniela. (5 puntos)

11. El método **numdobles** calcula el número de apuestas dobles de la quiniela. (5 puntos)

12. El método **numtriples** calcula el número de apuestas triples de la quiniela. (5 puntos)

13. Por último, el método precio calcula el precio de la quiniela teniendo en cuenta que el precio de la apuesta simple es de 0,5 euros, el de la doble 2 euros, y 5 euros el de la triple. Emplear los métodos creados anteriormente. (5 puntos)

A continuación se genera una segunda clase llamada **Apuesta** que contiene el método `main` y que pertenece al mismo paquete **quiniela**:

1. Crear un objeto quiniela con 5 pronósticos en los que los resultados sean aleatorios y almacenarlo en una variable con tu mismo nombre. (4 puntos)

2. Escribir el precio de la quiniela creada en una única línea. (4 puntos)

3. Almacenar en una lista de `Strings` 14 resultados cualesquiera para suponer que son los valores correctos de los partidos de la quiniela. (4 puntos)

4. Calcular los aciertos de cada uno de los pronósticos.(4 puntos)

5. Escribir en pantalla el número máximo de aciertos. (4 puntos)

6. Escribir en pantalla si se ha acertado el pleno o no suponiendo que el valor correcto del partido asociado al pleno es `'1'`. (4 puntos)

Tiempo: 2h.

Solución:

Se va a proceder a generar un programa para la gestión de las quinielas de fútbol. Cada una de las quinielas tiene 14 partidos mas el pleno al quince. Para los primeros 14 partidos hay un total de 8 pronósticos como máximo (no es necesario completar todos), y dentro de cada pronóstico el resultado del partido puede ser '`1`', '`X`' y '`2`', tal y como se muestra en la Figura A.11. Se puede seleccionar para cada partido un único resultado (apuesta simple), dos resultados (apuesta doble) o tres resultados (apuesta triple).

Para la elaboración del programa se comienza creando una clase denominada **Quiniela** que tiene las características siguientes:

1. La clase pertenece al paquete **quiniela**. (2 puntos)

   ```
   package quiniela;
   ```

2. Las variables de instancia que caracterizan un objeto de la clase **Quiniela** son el número de partidos sin contar el pleno al quince, y el número máximo de pronósticos, las variables se llamarán **partidos** y **pronosticos**, respectivamente. Tendrán un valor fijo que no se puede modificar y que deberá fijarse en el momento de la declaración de las variables, y serán comunes para todos los elementos de la clase **Quiniela**. (4 puntos)

   ```
   public class Quiniela {

     final static int partidos = 14;
     final static int pronosticos = 8;
   ```

Figura A.11: Representación de la quiniela, con sus 14 partidos, 8 pronósticos y su pleno al quince.

3. Dado que no es necesario completar todos los pronósticos, se declarará una variable llamada **numeropronosticos** en el que posteriormente se almacenará el número de pronósticos para una quiniela sellada. Además, para almacenar la información de cada quiniela se declarará la variable **apuestas** (véase la Figura A.12) que tendrá tantas filas como número de partidos sin contar el pleno al quince y de columnas el número de pronósticos multiplicado por tres posibles resultados para cada partido. El sistema reconocerá cada apuesta si en la casilla correspondiente hay un 1, en caso contrario habrá un cero, lo que implica que no hay apuesta en esa casilla. Para almacenar el número de partidos en los que se acierta el resultado para cada pronóstico, se declara una variable llamada **aciertos**. Y por último para almacenar el pleno se declara e inicializa la variable **pleno** que contendrá la información de que resultado se selecciona para el partido del pleno al quince. Inicialmente se supone que no se ha seleccionado ningún resultado. (4 puntos)

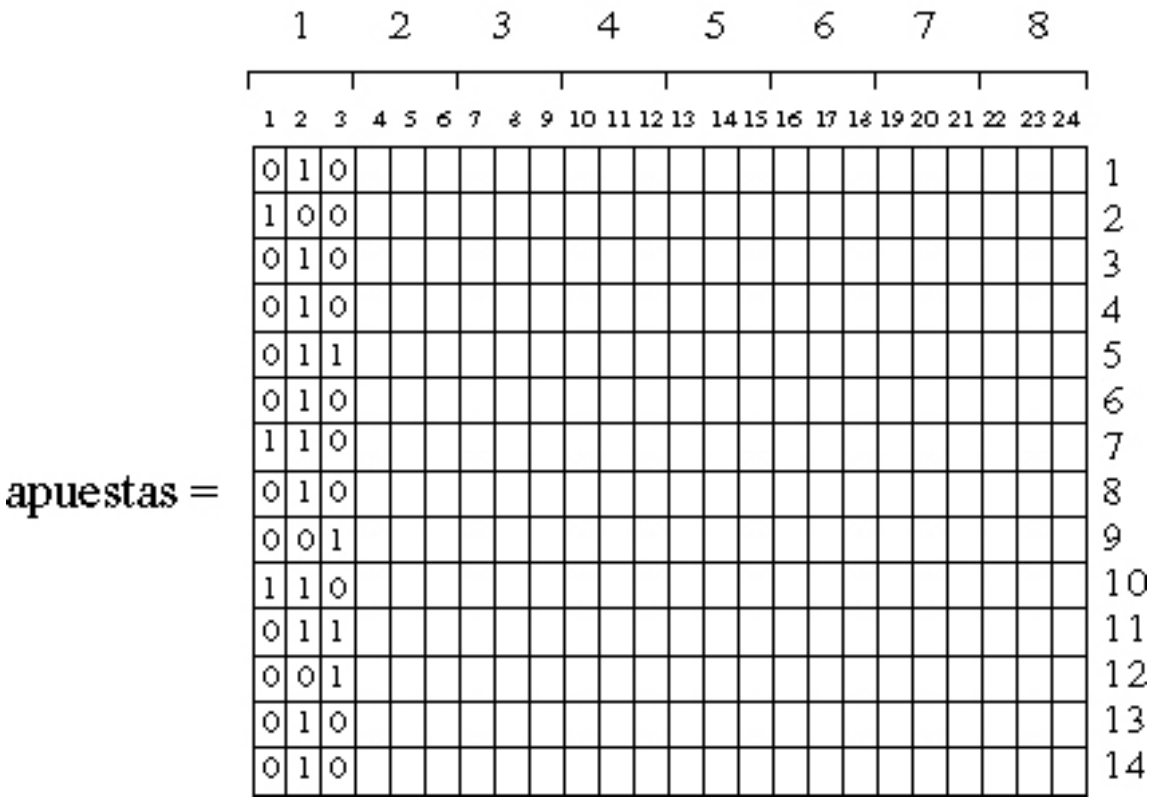


Figura A.12: Representación de la variable **apuestas** para el caso de la quiniela de la Figura A.11.

NOTA: Para la realización del examen puede ser especialmente útil la siguiente expresión

$$(k-1)*3+j,$$

que permite conocer la columna de la matriz almacenada en la variable **apuestas** suponiendo que se está en el bloque número k y que j toma valores 1, 2, o 3, correspondientes, respectivamente, con los valores '1', 'X' y '2'. Así por ejemplo, el resultado asociado a la 'X' del pronóstico 3 está en la columna $(3-1)*3+2 = 8$ (octava columna).

```
int numeropronosticos;
int apuestas [][];
int aciertos [];
int pleno [] = new int[3];
```

4. Se genera un constructor para permitir la inicialización de un objeto de esta clase sin dar ningún tipo de argumento mediante la sentencia `super()`. (2 puntos)

```
public Quiniela() {
```

```
    super();
  }
```

5. El segundo constructor recibe un número entero que se corresponde con el número de pronósticos que se quieren rellenar, y se encarga de: (10 puntos)

   a) Reservar espacio en memoria para la variable **apuestas**.

   b) En segundo lugar se inicializarán los valores de las apuestas para el número de pronósticos dado como argumento, de forma que se seleccionará para cada partido una única apuesta (apuesta simple). Para escoger el resultado se emplea la expresión `(int) Math.ceil(3*Math.random())` que devuelve un número aleatorio entre 1 y 3, de forma que si sale 1 significa que se ha seleccionado el `'1'` y ha de ponerse un uno en la primera columna correspondiente al pronóstico, si sale 2 significa que se ha seleccionado la `'X'` y ha de ponerse un uno en la segunda columna correspondiente al pronóstico, y si sale 3 significa que se ha seleccionado el `'2'` y ha de ponerse un uno en la tercera columna correspondiente al pronóstico. Para los resultados no elegidos su valor correspondiente es cero.

   c) Por ultimo se selecciona el resultado del pleno al quince empleando la misma expresión `(int) Math.ceil(3*Math.random())`, poniendo un 1 en la posición indicada por ese valor y cero en las otras dos casillas.

```
public Quiniela(int numeropronosticos) {
    int posicion;
    this.numeropronosticos = numeropronosticos;
    apuestas = new int [partidos][3*pronosticos];
    for(int j=0;j<numeropronosticos;j++){
      for(int i=0;i<partidos;i++){
        for(int k=1;k<=3;k++){
          posicion = (int) Math.ceil(3*Math.random());
          if(posicion==k){
            apuestas[i][3*j+k-1] = 1;
          }else{
            apuestas[i][3*j+k-1] = 0;
          }
        }
      }
    }
    posicion = (int) Math.ceil(3*Math.random());
    for(int k=1;k<=3;k++){
      if(posicion==k){
        pleno[k-1] = 1;
      }else{
        pleno[k-1] = 0;
      }
    }
  }
```

6. El tercer constructor recibe como argumentos una lista de listas con las apuestas, y una lista de tres elementos con la información del pleno al quince. (10 puntos)

a) Reservar espacio en memoria para la variable **apuestas** de la clase.

b) Comprobar si el número de filas de la variable con la información de las apuestas concuerda con el número de partidos de la clase **Quiniela**. En caso de que sea correcto se continua la ejecución, si no lo es se escribe un mensaje en pantalla que diga:

```
El número de partidos es incorrecto.
```

c) Si la dimensión es correcta se procederá a asignar valores a la variable **apuestas** pero sólo con los datos disponibles.

d) Asignar valores a la variable **pleno** de la clase **Quiniela**.

```
public Quiniela(int apuestas [][], int pleno []) {
  this.apuestas = new int [partidos][3*pronosticos];
  int npar, npro;
  npar = apuestas.length;
  npro = (int) apuestas[0].length / 3;
  numeropronosticos = npro;
  if (npar != partidos) {
    System.out.println("El número de partidos es incorrecto.");
  }
  else {
    for (int j = 0; j < npro; j++) {
      for (int i = 0; i < npar; i++) {
        this.apuestas[i][3 * j] = apuestas[i][3 * j];
        this.apuestas[i][3 * j + 1] = apuestas[i][3 * j + 1];
        this.apuestas[i][3 * j + 2] = apuestas[i][3 * j + 2];
      }
    }
    this.pleno = pleno;
  }
}
```

7. La clase contiene un método que se llama **aciertospronosticos** que recibe como argumento una lista con 14 cadenas de caracteres en los que está la solución de la quiniela ‘1’, ‘X’ o ‘2’ para cada uno de los partidos. Con esa información se recorrerán todos los pronósticos y se contarán el número de acietos, almacenándolos en la variable **aciertos**. (8 puntos)

```
public void aciertospronosticos (String resultados[]){
  aciertos=new int[pronosticos];
  for(int j=0;j<numeropronosticos;j++){
    aciertos[j] =0;
  }
  for(int i=0;i<partidos;i++){
    for(int j=0;j<numeropronosticos;j++){
      if(resultados[i]=="1" && apuestas[i][3*j]==1){
        aciertos[j] += 1;
      } else if (resultados[i]=="X" && apuestas[i][3*j+1]==1){
        aciertos[j] += 1;
      } else if (resultados[i]=="2" && apuestas[i][3*j+2]==1){
```

```
          aciertos[j] += 1;
        }
      }
    }
  }
```

8. Se creará otro método llamado **aciertopleno** que recibe como argumento un string con el resultado '1', 'X' o '2' correcto del pleno al quince, que devuelve "verdadero" si se ha acertado, y "falso" en caso contrario. (8 puntos)

```
public boolean aciertopleno(String plenores){
    boolean plenotrue = false;
    if(plenores=="1" && pleno[0]==1){
      plenotrue = true;
    } else if (plenores=="X" && pleno[1]==1){
      plenotrue = true;
    } else if (plenores=="2" && pleno[2]==1){
      plenotrue = true;
    }
    return(plenotrue);
  }
```

9. Otro de los métodos calcula el número máximo de aciertos almacenado en la variable **aciertos** para los pronósticos en los que hay apuestas. (8 puntos)

```
 public int maxacierto (){
    int maximo = aciertos[0];
    for(int j=1;j<numeropronosticos;j++){
      if(aciertos[j]>maximo){
        maximo = aciertos[j];
      }
    }
    return(maximo);
  }
```

10. El siguiente de los métodos llamado **numsimples** calcula el número de apuestas sencillas de la quiniela. (5 puntos)

```
public int numsimples (){
    int contador = 0;
    int aux;
    for(int i=0;i<partidos;i++){
      for(int j=0;j<numeropronosticos;j++){
        aux = apuestas[i][3*j]+apuestas[i][3*j+1]+apuestas[i][3*j+2];
        if(aux==1){
```

```
            contador++;
          }
        }
      }
      return(contador);
    }
```

11. El método **numdobles** calcula el número de apuestas dobles de la quiniela. (5 puntos)

```
 public int numdobles (){
    int contador = 0;
    int aux;
    for(int i=0;i<partidos;i++){
      for(int j=0;j<numeropronosticos;j++){
        aux = apuestas[i][3*j]+apuestas[i][3*j+1]+apuestas[i][3*j+2];
        if(aux==2){
          contador++;
        }
      }
    }
    return(contador);
  }
```

12. El método **numtriples** calcula el número de apuestas triples de la quiniela. (5 puntos)

```
public int numtriples (){
    int contador = 0;
    int aux;
    for(int i=0;i<partidos;i++){
      for(int j=0;j<numeropronosticos;j++){
        aux = apuestas[i][3*j]+apuestas[i][3*j+1]+apuestas[i][3*j+2];
        if(aux==3){
          contador++;
        }
      }
    }
    return(contador);
  }
```

13. Por último, el método precio calcula el precio de la quiniela teniendo en cuenta que el precio de la apuesta simple es de 0,5 euros, el de la doble 2 euros, y 5 euros el de la triple. Emplear los métodos creados anteriormente. (5 puntos)

```
 public double precio () {
    double coste;
```

```
    coste = this.numsimples()*0.5+this.numdobles()*2+this.numsimples()*5;
    return(coste);
  }

 }// Fin la clase Sudoku
```

A continuación se genera una segunda clase llamada **Apuesta** que contiene el método `main` y que pertenece al mismo paquete **quiniela**:

1. Crear un objeto quiniela con 5 pronósticos en los que los resultados sean aleatorios y almacenarlo en una variable con tu mismo nombre. (4 puntos)

   ```
   Quiniela roberto = new Quiniela(5);
   ```

2. Escribir el precio de la quiniela creada en una única línea. (4 puntos)

   ```
   System.out.println("El precio de la Quiniela es de "+roberto.precio()+" euros.");
   ```

3. Almacenar en una lista de `Strings` 14 resultados cualesquiera para suponer que son los valores correctos de los partidos de la quiniela. (4 puntos)

   ```
   String resultado []= {"X","1","1","2","X","2","1","X","1","1","1","2","X","X"};
   ```

4. Calcular los aciertos de cada uno de los pronósticos:(4 puntos)

   ```
   roberto.aciertospronosticos(resultado);
   ```

5. Escribir en pantalla el número máximo de aciertos. (4 puntos)

   ```
   System.out.println("El número máximo de aciertos es de "+roberto.maxacierto());
   ```

6. Escribir en pantalla si se ha acertado el pleno o no suponiendo que el valor correcto del partido asociado al pleno es `'1'`. (4 puntos)

   ```
   if(roberto.aciertopleno("1")){
        System.out.println("Ha acertado el pleno.");
     } else {
        System.out.println("No ha acertado el pleno.");
     }

    }

   }
   ```

Tiempo: 2h.

■

Figura A.13: Boleto de Banda Verde que edita Loterías y Apuestas del Estado para el sorteo de La Primitiva.

Práctica A.6 (Extraordinario de 9 de enero de 2007). Se va a proceder a generar un programa para la gestión de La Primitiva. El funcionamiento del sorteo es el siguiente. Las Apuestas sólo podrán efectuarse en uno de los 2 boletos que edita Loterías y Apuestas del Estado, que se podrán conseguir en los establecimientos autorizados. El boleto de Banda Verde mostrado en la Figura A.13 sirve para un solo Sorteo de los dos que hay, si se valida de Lunes a Jueves entra a concursar para el Sorteo del Jueves, si se valida de Viernes a Sábado, entra a concursar para el Sorteo del Sábado.

Se puede jugar sin necesidad de utilizar el boleto, siendo el Terminal de Juego, el que por Azar, nos elige la combinación según el importe que se desee gastar.

Utilizando los boletos se puede jugar por varios sistemas, en este caso se opta por el **Método directo sencillo**: Se puede jugar de 1 a 8 Apuestas en un mismo boleto. Para jugar 1 apuesta se marcan las seis cruces en el primer bloque de los 8 que tiene el boleto, eligiendo los números que deseemos. Para jugar 2 Apuestas marcaremos 6 números en el primer bloque y 6 en el segundo, y así sucesivamente. Los bloques marcados serán siempre correlativos de izquierda a derecha, no pudiendo dejar ninguno en blanco.

	Importe en euros	
Apuestas	Un sorteo	Dos sorteos
1	1,00	2,00
2	2,00	4,00
3	3,00	6,00
4	4,00	8,00
5	5,00	10,00
6	6,00	12,00
7	7,00	14,00
8	8,00	16,00

Concluido el Sorteo dará comienzo el escrutinio de todas las apuestas validadas que participan en el Concurso, para asignar los premios que correspondan a cada una por coincidencia entre la Combinación Ganadora y los pronósticos que constan en las apuestas. Solo se podrá percibir un premio por apuesta.

- EL PRIMER premio consiste en acertar los 6 números que salen del bombo, pero dado que la Combinación Ganadora consta de seis números.
- Si acertamos únicamente 5 números de la Combinación Ganadora hay premio.
- Si acertamos únicamente 4 números de la Combinación Ganadora hay premio.
- Si acertamos únicamente 3 números de la Combinación Ganadora hay premio.
- También se establece un premio complementario. Junto a los seis números que compone la Combinación Ganadora se extrae un séptimo denominado Complementario destinado a ofrecer premio a los acertantes de 5 números que además acierten este número, se trata del premio de 5+Complementario.
- También se obtiene premio cuando el número designado como Reintegro en nuestro boleto, coincida con la bola que se extrae en el sorteo destinada a ese premio, en ese caso será abonado el importe de la apuesta jugada en el boleto.

Para la elaboración del programa se comienza creando una clase denominada **Primitiva** que tiene las características siguientes:

1. La clase pertenece al paquete **primitiva**. (2 puntos)
2. Las variables de instancia que caracterizan un objeto de la clase **Primitiva** son el número de números que se seleccionan para cada apuesta (6), el número máximo de apuestas (8), y el número de posibles valores enteros que pueden tomar los números (49). Las variables se llamarán **numeros**, **napuestas** y **numeromaximo**, respectivamente. Tendrán un valor fijo que no se puede modificar y que deberá fijarse en el momento de la declaración de las variables, y serán comunes para todos los elementos de la clase **Primitiva**. (4 puntos)

3. Dado que no es necesario completar todas las apuestas, se declarará una variable llamada **numeroapuestas** en la que posteriormente se almacenará el número de apuestas para una primitiva sellada. Además, para almacenar la información de cada primitiva se declarará la variable **apuestas** que tendrá tantas filas como número de apuestas y de columnas el número de números de cada apuesta (6). En cada una de las filas de la variable **apuestas** se almacenan los seis números seleccionados para cada apuesta. Para almacenar el número de aciertos de cada apuesta se declara una variable llamada **aciertos**. Y por último para almacenar tanto el número complementario como el reintegro se declaran las variables **complementario** y **reintegro**. (4 puntos)

4. Se genera un constructor para permitir la inicialización de un objeto de esta clase sin dar ningún tipo de argumento mediante la sentencia `super()`. (2 puntos)

5. Se creará un método que se llamará **apuestaleatoria**, que no recibe ningún argumento y que devuelve una lista con el número de números (utilizar la información de las variables ya declaradas e inicializadas) necesario para realizar una apuesta, con las siguientes características: (8 puntos)

   a) Los números han de generarse de forma aleatoria para ello se emplea la expresión `(int) Math.ceil(numeromaximo*Math.random())` que devuelve un número entero aleatorio entre 1 y **numeromaximo**.

   b) Ninguno de los números se puede repetir.

6. El segundo constructor recibe un número entero que se corresponde con el número de apuestas que se quieren rellenar, y se encarga de: (8 puntos)

   a) Reservar espacio en memoria para la variable **apuestas**.

   b) Se le asigna valor a la variable **numeroapuestas**.

   c) En tercer lugar se inicializarán los valores de la variable apuestas empleando el método creado en el apartado 5. Nótese que el método devuelve una lista que se corresponde con cada una de las filas de la matriz almacenada en la variable **apuestas**.

   d) A continuación se le da un valor aleatorio entre 1 y **numeromaximo** a la variable **complementario**, en la que se almacena el número complementario.

   e) Análogamente, se le asigna un valor aleatorio entre 1 y **numeromaximo** a la variable **reintegro**, en la que se almacena el número correspondiente al reintegro.

7. El tercer constructor recibe como argumentos una lista de listas con las apuestas, y tienen las siguientes características: (5 puntos)

   a) Asignar valores a la variable **apuestas**.

   b) A continuación se le da un valor aleatorio entre 1 y **numeromaximo** a la variable **complementario**, en la que se almacena el número complementario.

   c) Análogamente, se le asigna un valor aleatorio entre 1 y **numeromaximo** a la variable **reintegro**, en la que se almacena el número correspondiente al reintegro.

8. La clase contiene un método que se llama **aciertosapuestas** que recibe como argumento una lista con 6 números enteros que se corresponden con la combinación ganadora del sorteo. Con esa información se recorrerán todas los apuestas y se contarán el número de aciertos, almacenándolos en la variable **aciertos**. (8 puntos)

9. Se creará otro método llamado **aciertocomplementario** que recibe como argumento un número que se corresponde con el número complementario del sorteo, que devuelve "verdadero" si se ha acertado, y "falso" en caso contrario. (4 puntos)

10. Se creará otro método llamado **aciertoreintegro** que recibe como argumento un número que se corresponde con el número del reintegro del sorteo, que devuelve "verdadero" si se ha acertado, y "falso" en caso contrario. (4 puntos)

11. Otro de los métodos calcula el número máximo de aciertos almacenado en la variable **aciertos** para cada una de las apuestas. (5 puntos)

12. El siguiente de los métodos llamado `cinco_comple` comprueba si al menos en alguna de las apuestas se han obtenido 5 aciertos y además se ha acertado el número complementario, para ello se supone que en todas las variables ya está la información necesaria. (5 puntos)

13. Por último, el método precio calcula el precio de la primitiva teniendo en cuenta que sólo se participa en un sorteo. (5 puntos)

A continuación se genera una segunda clase llamada **Apuesta** que contiene el método `main` y que pertenece al mismo paquete **primitiva**:

1. Crear un objeto primitiva con 5 apuestas en las que los resultados sean aleatorios y almacenarlo en una variable con tu mismo nombre. (4 puntos)

2. Escribe las los números de las apuestas creados anteriormente, escribiendo en cada fila los seis números de cada apuesta. (4 puntos)

3. Escribe el número complementario y el reintegro de tu variable de tipo Primitiva. (4 puntos)

4. Escribir el precio de la quiniela creada en una única línea. (4 puntos)

5. Almacenar en una lista de enteros 6 resultados cualesquiera para suponer que son los valores correctos del sorteo de la primitiva. (4 puntos)

6. Calcular los aciertos de cada una de las apuestas. (4 puntos)

7. Escribir en pantalla el número máximo de aciertos. (4 puntos)

8. Escribir en pantalla si se ha acertado el complementario o no suponiendo que el valor correcto del número complementario es el 17. (4 puntos)

9. Escribir en pantalla si se ha acertado el reintegro o no suponiendo que el valor correcto del reintegro es el 31. (4 puntos)

Tiempo: 2h.

Solución:

Se va a proceder a generar un programa para la gestión de La Primitiva. El funcionamiento del sorteo es el siguiente. Las Apuestas sólo podrán efectuarse en uno de los 2 boletos que edita Loterías y Apuestas del Estado, que se podrán conseguir en los establecimientos autorizados. El boleto de Banda Verde mostrado en la Figura A.14 sirve para un solo Sorteo de los dos que hay, si se valida de Lunes a Jueves entra a concursar para el Sorteo del Jueves, si se valida de Viernes a Sábado, entra a concursar para el Sorteo del Sábado.

Figura A.14: Boleto de Banda Verde que edita Loterías y Apuestas del Estado para el sorteo de La Primitiva.

Se puede jugar sin necesidad de utilizar el boleto, siendo el Terminal de Juego, el que por Azar, nos elige la combinación según el importe que se desee gastar.

Utilizando los boletos se puede jugar por varios sistemas, en este caso se opta por el **Método directo sencillo**: Se puede jugar de 1 a 8 Apuestas en un mismo boleto. Para jugar 1 apuesta se marcan las seis cruces en el primer bloque de los 8 que tiene el boleto, eligiendo los números que deseemos. Para jugar 2 Apuestas marcaremos 6 números en el primer bloque y 6 en el segundo, y así sucesivamente. Los bloques marcados serán siempre correlativos de izquierda a derecha, no pudiendo dejar ninguno en blanco.

	Importe en euros	
Apuestas	Un sorteo	Dos sorteos
1	1,00	2,00
2	2,00	4,00
3	3,00	6,00
4	4,00	8,00
5	5,00	10,00
6	6,00	12,00
7	7,00	14,00
8	8,00	16,00

Concluido el Sorteo dará comienzo el escrutinio de todas las apuestas validadas que participan en el Concurso, para asignar los premios que correspondan a cada una por coincidencia entre la Combinación Ganadora y los pronósticos que constan en las apuestas. Solo se podrá percibir un premio por apuesta.

- EL PRIMER premio consiste en acertar los 6 números que salen del bombo, pero dado que la Combinación Ganadora consta de seis números.
- Si acertamos únicamente 5 números de la Combinación Ganadora hay premio.
- Si acertamos únicamente 4 números de la Combinación Ganadora hay premio.
- Si acertamos únicamente 3 números de la Combinación Ganadora hay premio.
- También se establece un premio complementario. Junto a los seis números que compone la Combinación Ganadora se extrae un séptimo denominado Complementario destinado a ofrecer premio a los acertantes de 5 números que además acierten este número, se trata del premio de 5+Complementario.
- También se obtiene premio cuando el número designado como Reintegro en nuestro boleto, coincida con la bola que se extrae en el sorteo destinada a ese premio, en ese caso será abonado el importe de la apuesta jugada en el boleto.

Para la elaboración del programa se comienza creando una clase denominada **Primitiva** que tiene las características siguientes:

1. La clase pertenece al paquete **primitiva**. (2 puntos)

   ```
   package primitiva;
   ```

2. Las variables de instancia que caracterizan un objeto de la clase **Primitiva** son el número de números que se seleccionan para cada apuesta (6), el número máximo de apuestas (8), y el número de posibles valores enteros que pueden tomar los números (49). Las variables se llamarán **numeros**, **napuestas** y **numeromaximo**, respectivamente. Tendrán un valor fijo que no se puede modificar y que deberá fijarse en el momento de la declaración de las variables, y serán comunes para todos los elementos de la clase **Primitiva**. (4 puntos)

```
public class Primitiva {

  final static int numeros = 6;
  final static int napuestas = 8;
  final static int numeromaximo = 49;
```

3. Dado que no es necesario completar todas las apuestas, se declarará una variable llamada **numeroapuestas** en la que posteriormente se almacenará el número de apuestas para una primitiva sellada. Además, para almacenar la información de cada primitiva se declarará la variable **apuestas** que tendrá tantas filas como número de apuestas y de columnas el número de números de cada apuesta (6). En cada una de las filas de la variable **apuestas** se almacenan los seis números seleccionados para cada apuesta. Para almacenar el número de aciertos de cada apuesta se declara una variable llamada **aciertos**. Y por último para almacenar tanto el número complementario como el reintegro se declaran las variables **complementario** y **reintegro**. (4 puntos)

```
  int apuestas [][];
  int aciertos [];

  int numeroapuestas,complementario, reintegro;
```

4. Se genera un constructor para permitir la inicialización de un objeto de esta clase sin dar ningún tipo de argumento mediante la sentencia `super()`. (2 puntos)

```
 public Quiniela() {
    super();
 }
```

5. Se creará un método que se llamará **apuestaleatoria**, que no recibe ningún argumento y que devuelve una lista con el número de números (utilizar la información de las variables ya declaradas e inicializadas) necesario para realizar una apuesta, con las siguientes características: (8 puntos)

   a) Los números han de generarse de forma aleatoria para ello se emplea la expresión `(int) Math.ceil(numeromaximo*Math.random())` que devuelve un número entero aleatorio entre 1 y **numeromaximo**.

   b) Ninguno de los números se puede repetir.

```
 public int[] apuestaleatoria (){
  int lista[] = new int[numeros];
  int aux=0;
  boolean repetido;
  for(int i=1;i<=numeros;i++){
    repetido = true;
    while(repetido){
```

```
        aux = (int) Math.ceil(numeromaximo*Math.random());
        repetido = false;
        for(int j=1;j<i;j++){
          if(aux==lista[j-1]){
            repetido = true;
            break;
          }
        }
      }
      lista[i-1]=aux;
    }
    return(lista);
  }
```

6. El segundo constructor recibe un número entero que se corresponde con el número de apuestas que se quieren rellenar, y se encarga de: (8 puntos)

   a) Reservar espacio en memoria para la variable **apuestas**.

   b) Se le asigna valor a la variable **numeroapuestas**.

   c) En tercer lugar se inicializarán los valores de la variable apuestas empleando el método creado en el apartado 5. Nótese que el método devuelve una lista que se corresponde con cada una de las filas de la matriz almacenada en la variable **apuestas**.

   d) A continuación se le da un valor aleatorio entre 1 y **numeromaximo** a la variable **complementario**, en la que se almacena el número complementario.

   e) Análogamente, se le asigna un valor aleatorio entre 1 y **numeromaximo** a la variable **reintegro**, en la que se almacena el número correspondiente al reintegro.

```
  public Primitiva(int numeroapuestas) {
     this.numeroapuestas = numeroapuestas;
     apuestas = new int[numeroapuestas][numeros];
     int lista[];
     for(int j=1;j<=numeroapuestas;j++){
       lista = this.apuestaleatoria();
       for(int i=1;i<=numeros;i++){
         apuestas[j-1][i-1]=lista[i-1];
       }
     }
     complementario = (int) Math.ceil(numeromaximo*Math.random());
     reintegro = (int) Math.ceil(numeromaximo*Math.random());
  }
```

7. El tercer constructor recibe como argumentos una lista de listas con las apuestas, y tienen las siguientes características: (5 puntos)

a) Asignar valores a la variable **apuestas**.

b) A continuación se le da un valor aleatorio entre 1 y **numeromaximo** a la variable **complementario**, en la que se almacena el número complementario.

c) Análogamente, se le asigna un valor aleatorio entre 1 y **numeromaximo** a la variable **reintegro**, en la que se almacena el número correspondiente al reintegro.

```
public Primitiva(int apuestas [][]) {
 this.apuestas = apuestas;
 complementario = (int) Math.ceil(numeromaximo*Math.random());
 reintegro = (int) Math.ceil(numeromaximo*Math.random());
}
```

8. La clase contiene un método que se llama **aciertosapuestas** que recibe como argumento una lista con 6 números enteros que se corresponden con la combinación ganadora del sorteo. Con esa información se recorrerán todas los apuestas y se contarán el número de aciertos, almacenándolos en la variable **aciertos**. (8 puntos)

```
public void aciertosapuestas (int resultados[]){
 aciertos=new int[numeroapuestas];
 for(int i=1;i<=numeroapuestas;i++){
   aciertos[i-1] =0;
 }
 int k;
 boolean encontrado;
 for(int i=1;i<=numeroapuestas;i++){
   for(int j=1;j<=numeros;j++){
     k=1;
     encontrado = false;
     while(!encontrado && k<=numeros){
       if(apuestas[i-1][j-1]==resultados[k-1]){
         encontrado = true;
         aciertos[i-1]+=1;
       }
       k++;
     }
   }
 }
}
```

9. Se creará otro método llamado **aciertocomplementario** que recibe como argumento un número que se corresponde con el número complementario del sorteo, que devuelve "verdadero" si se ha acertado, y "falso" en caso contrario. (4 puntos)

```
public boolean aciertocomplementario(int compleres){
   boolean completrue = false;
   if(complementario==compleres){
     completrue = true;
```

```
    }
    return(completrue);
  }
```

10. Se creará otro método llamado **aciertoreintegro** que recibe como argumento un número que se corresponde con el número del reintegro del sorteo, que devuelve "verdadero" si se ha acertado, y "falso" en caso contrario. (4 puntos)

```
  public boolean aciertoreintegro(int reinres){
    boolean reintrue = false;
    if(reintegro==reinres){
      reintrue = true;
    }
    return(reintrue);
  }
```

11. Otro de los métodos calcula el número máximo de aciertos almacenado en la variable **aciertos** para cada una de las apuestas. (5 puntos)

```
 public int maxacierto (){
    int maximo = aciertos[0];
    for(int j=1;j<numeroapuestas;j++){
      if(aciertos[j]>maximo){
        maximo = aciertos[j];
      }
    }
    return(maximo);
  }
```

12. El siguiente de los métodos llamado `cinco_comple` comprueba si al menos en alguna de las apuestas se han obtenido 5 aciertos y además se ha acertado el número complementario, para ello se supone que en todas las variables ya está la información necesaria. (5 puntos)

```
public boolean cinco_comple (int compleres){
    boolean acierto = false;
    for(int i=1;i<=numeroapuestas;i++){
      if(aciertos[i-1]==5 && this.aciertocomplementario(compleres)){
        acierto = true;
        break;
      }
    }
    return(acierto);
  }
```

13. Por último, el método precio calcula el precio de la primitiva teniendo en cuenta que sólo se participa en un sorteo. (5 puntos)

```
public double precio () {
   double coste;
   coste = numeroapuestas*1;
   return(coste);
 }

}// Fin la clase Primitiva
```

A continuación se genera una segunda clase llamada **Apuesta** que contiene el método `main` y que pertenece al mismo paquete **primitiva**:

1. Crear un objeto primitiva con 5 apuestas en las que los resultados sean aleatorios y almacenarlo en una variable con tu mismo nombre. (4 puntos)

```
Primitiva roberto = new Primitiva(5);
```

2. Escribe las los números de las apuestas creados anteriormente, escribiendo en cada fila los seis números de cada apuesta. (4 puntos)

```
System.out.println("Las apuestas por filas son las siguientes:");
 for(int i=1;i<=roberto.numeroapuestas;i++){
   for(int j=1;j<=Primitiva.numeros;j++){
     System.out.print(roberto.apuestas[i-1][j-1]+" ");
   }
   System.out.println(" ");
 }
```

3. Escribe el número complementario y el reintegro de tu variable de tipo Primitiva. (4 puntos)

```
System.out.println("El número complementario es el "+roberto.complementario);

System.out.println("El número de reintegro es "+roberto.reintegro);
```

4. Escribir el precio de la quiniela creada en una única línea. (4 puntos)

```
System.out.println("El precio de la Quiniela es de "+roberto.precio()+" euros.");
```

5. Almacenar en una lista de enteros 6 resultados cualesquiera para suponer que son los valores correctos del sorteo de la primitiva. (4 puntos)

```
int resultado []= {12,24,2,49,36,27};
```

6. Calcular los aciertos de cada una de las apuestas. (4 puntos)

```
roberto.aciertosapuestas(resultado);
```

7. Escribir en pantalla el número máximo de aciertos. (4 puntos)

```
System.out.println("El número máximo de aciertos es de "+roberto.maxacierto());
```

8. Escribir en pantalla si se ha acertado el complementario o no suponiendo que el valor correcto del número complementario es el 17. (4 puntos)

```
if(roberto.aciertocomplementario(17)){
  System.out.println("Ha acertado el complementario.");
} else {
  System.out.println("No ha acertado el complementario.");
}
```

9. Escribir en pantalla si se ha acertado el reintegro o no suponiendo que el valor correcto del reintegro es el 31. (4 puntos)

```
if(roberto.aciertocomplementario(17)){
  System.out.println("Ha acertado el complementario.");
} else {
  System.out.println("No ha acertado el complementario.");
}
```

Tiempo: 2h.

Bibliografía

[1] E. Castillo, A. Cobo, P. Gómez, y C. Solares, *JAVA TM Un lenguaje de programación en Internet.* Madrid, España: Paraninfo, 2000. Paraninfo, 1997. Libro que por sus contenidos puede ser considerado como básico para los alumnos. En este libro se desarrollan con gran detalle los temas del programa correspondientes a la parte de Java, excepto el tema de control de sucesos. Los distintos conceptos se explican de una forma clara y con gran número de ejemplos, lo cual facilita su estudio. En cada capítulo se plantea una colección de problemas relacionados con los conceptos vistos.

[2] D. Flanagan. *Java Examples in a Nutshell.* o'Reilly, 1997. Se trata de un libro muy interesante del cual pueden ser extraidos gran cantidad de ejemplos y ejercicios que se corresponden con los contenidos de los Temas 1, 2, 3, 4, 5 y 6 de lenguaje Java.

[3] Cay S. Horstmann and Gary Cornell. *Core Java. Volume 1.* Mc Graw-Hill, 1999. Libro de nivel avanzado, en este libro se tratan con rigor cada uno de los temas del programa de Lenguaje Java. Además muestra ejemplos muy interesantes de utilización de dicho lenguaje. Cabe destacar el tratamiento que hace de los Temas 4 y 5 del programa.

[4] C. Thomas Wu. *An Introduction to Object-Oriented Programming with Java.* Sun Microsystems, 1997. Libro muy interesante en el cual se exponen de forma muy clara y concisa los temas de lenguaje orientado a objeto, lenguaje java y control de sucesos. Al principio de cada tema hace un esquema con los objetivos del mismo, la exposición se hace de una forma muy clara con esquemas gráficos que facilitan su comprensión y con gran número de ejemplos. Al final de cada tema se presenta una colección my completa de ejercicios relativos al mismo.

[5] W. Savitch. *Java. An Introduction to Computer Science and Programming.* Prentice Hall, 1999. Se trata de un libro muy completo en el cual se tratan todos los temas del programa de Java salvo los Temas 7 y 8 de procesos ligeros y desarrollo de animaciones en Java. La exposición es muy clara y detallada, los conceptos y comandos nuevos de cada tema aparecen destacados respecto al resto del texto, lo mismo ocurre con los distintos programas ejemplo que se muestran. Todos los conceptos y comandos aparecen ilustrados con programas de ejemplo, y al final de cada tema se presenta una colección de ejercicios relativos al mismo.