Programación Concurrente y de Tiempo Real

Guión de Prácticas 3: Reutilización de Clases: Modelo de Herencia en Java

Natalia Partera Jaime
Alumna colaboradora de la asignatura

Índice

1. Herencia en la Programación Orientada a Objetos

Anteriormente hemos visto algunos de los principios de la orientación a objetos. Otro de sus principios básicos es el concepto de reutilización de clases. La reutilización evita que tengamos que codificar o implementar varias veces el mismo fragmento de código. En el caso concreto de la reutilización de clases, evita que tengamos que implementar aspectos comunes o compartidos por varias clases.

En la orientación a objetos, la reutilización de clases se modela mediante la herencia. La herencia de clases nos permite crear una clase a partir de otra, haciendo que la nueva clase tenga los mismos atributos y los mismos métodos que la clase de la que hereda. La clase de la que se hereda puede recibir distintos nombres: superclase, clase madre o clase base, por ejemplo. La clase heredada también puede ser nombrada de distintas formas: subclase, clase hija o clase extendida.

En la orientación a objetos, la herencia de clases puede ser simple o múltiple. Es decir, una subclase puede heredar de una o más superclases. Al heredar, una subclase crea tantos atributos como su superclase, pero no toma valores de éstos, y hereda también todos los métodos de su superclase. Todo esto siempre que sea posible según la visibilidad de sus elementos. Además de los campos heredados, una subclase puede definir más atributos o más métodos. Una subclase puede incluso sobrecargar métodos heredados de su superclase para que se comporten de otro modo.

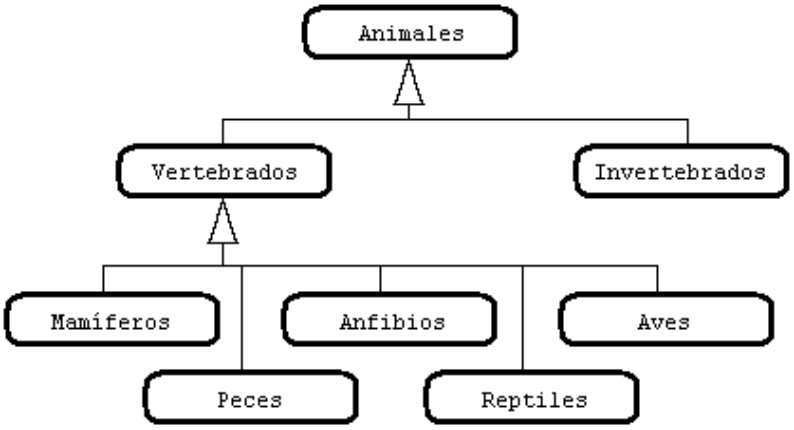


Figura 1: Esquema de herencia simple.

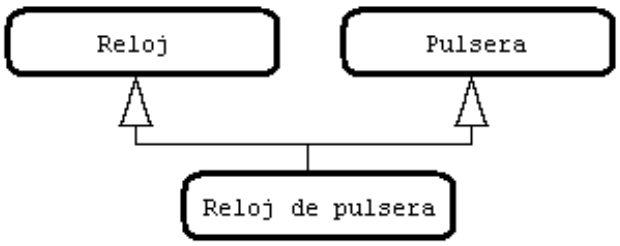


Figura 2: Esquema de herencia multiple.

Hay que pensar en la subclase como en un tipo especial de superclase, que además tiene características o comportamientos propios que no tiene la superclase o que funciona de manera distinta a como ocurre en la superclase. La relación que existe entre ambas es del tipo *es un/a*, es decir un objeto de la subclase *es un* objeto de la superclase también (ver Figura 1).

Por ejemplo, a continuación (en la Figura 3) podemos ver un diagrama de clases en el que se modela la relación entre una clase `Persona` y una clase `UsuarioWeb`. En los recuadros modelamos las clases tal y como se explicaron en el guión anterior. La clase `Persona` tiene 3 atributos que guardan el nombre, el país de residencia y el año de nacimiento. También tiene 3 métodos observadores y un método modificador. La clase `UsuarioWeb` que hereda a la clase `Persona` tiene 5 atributos: 3 atributos heredados de la clase `Persona` (en negro) y 2 atributos propios (en rojo). De la misma manera, se puede observar que tiene 7 métodos: los 4 métodos que incorporaba la clase `Persona` (en negro) y 3 métodos nuevos (en rojo).

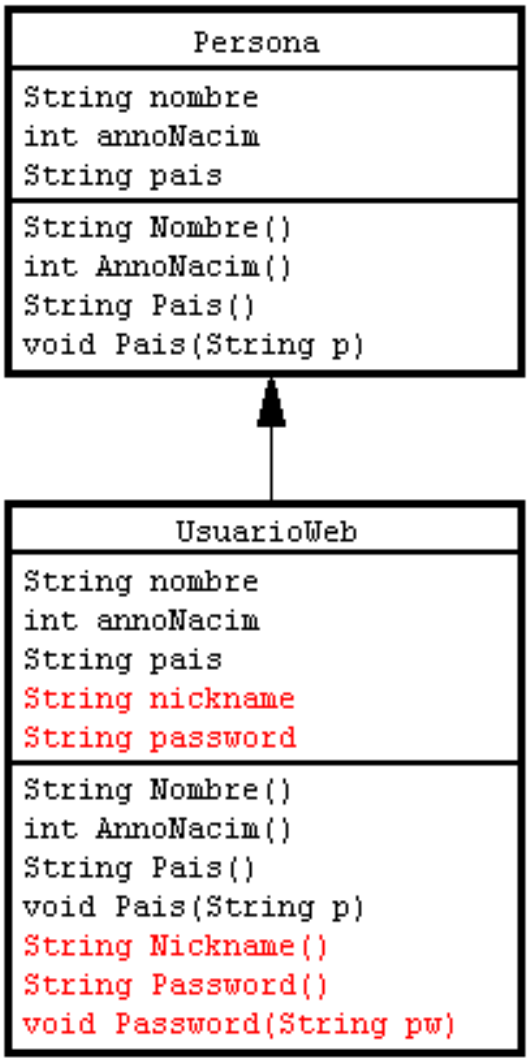


Figura 3: Diagrama que muestra la herencia entre dos clases.

Ejercicio 1 Imagine una superclase que modele figuras geométricas y subclases que modelen círulos, rectángulos y polígonos. Diseñe su diagrama de clases indicando los atributos y métodos que tendría cada clase. Compruebe su solución con la que se encuentra en el apartado 8.1.

1.1. Herencia en Java

En Java la herencia se modela mediante el mecanismo de *extensión de clases*. Esto se consigue utlizando la palabra reservada `extends` en la declaración de la subclase. Tras el nombre de la subclase a declarar, escribimos la palabra `extends` seguida de la superclase de la que hereda.

```
class NombreSubclase extends NombreSuperclase
{
  //Cuerpo de la subclase
}
```

Una subclase puede ser a la vez superclase de otra que la extiende. Veamos un ejemplo en el que se crean 3 clases. La clase `UsuarioWeb` extiende a la clase `Persona` y es a su vez superclase de `Webmaster`:

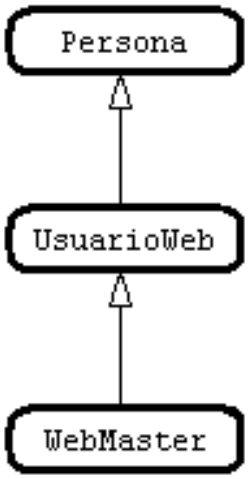


Figura 4: Ejemplo: una clase extendida puede ser subclase de otra clase.

```
class Persona {
  ...
}

class UsuarioWeb extends Persona {
  ...
}

class WebMaster extends UsuarioWeb {
  ...
}
```

La subclase hereda sólo los campos que sean visibles para ella según la protección de los datos. Si recuerda el cuadro resumen del guión anterior acerca de la visibilidad y acceso a los miembros de la clase según la protección de datos, la subclase sólo puede acceder a elementos `public` y `protected`. En la sección 2.3 se explicará este aspecto en más profundidad. Recuérdelo a la hora de implementar su superclase. Un ejemplo de una superclase con elementos heredables es la siguiente:

```
public class Superclase {
  protected int dato1, dato2;
  private int dato3;
  public int dato4;

  Superclase(int d) {
    dato1 = d + 1;
    dato2 = d + 2;
    dato3 = d + 3;
    dato4 = d + 4;
  }

  public void VerDatos() {
    System.out.println("dato1 = " + dato1 + "; dato2 = " + dato2 + "; dato3 = " +
      dato3 + "; dato4 = " + dato4);
  }

  protected void OperacionDatos() {}
}
```

La subclase puede poseer además otros atributos y otros métodos distintos de los de la superclase. También puede sobrecargar los métodos de la superclase, para que se comporten de una forma distinta con los objetos de la subclase. En cuanto a los atributos, recuerde que la subclase sólo hereda el número de atributos y su tipo, no los valores de éstos. Observe el siguiente ejemplo de subclase:

```
public class Subclase extends Superclase {
  private int dato5;

  public Subclase(int d1, int d5) {
    super(d1);
    dato5 = d5;
  }

  public void VerTodosDatos() {
    VerDatos();
    System.out.println("dato5 = " + dato5);
  }

  public void OperacionDatos() {
    int suma;
    suma = dato1 + dato2 + dato4 + dato5;
    System.out.println("La suma de dato1, dato2, dato4 y dato5 es:");
    System.out.println(dato1 + " + " + dato2 + " + " + dato4 + " + " + dato5 +
      " = " + suma);
  }
}
```

Para comprobar el funcionamiento de estas clases, tenemos el siguiente programa:

```
public class UsaClases {
  public static void main (String[] args) {
    Superclase super1;
    Subclase sub1;

    super1 = new Superclase(3);
    super1.VerDatos();

    sub1 = new Subclase(5, 1);
    sub1.VerDatos();
    sub1.OperacionDatos();
  }
}
```

Ejercicio 2 ¿Qué cree que hará el programa de prueba del código anterior? Compile los códigos y compruebe si se corresponde con lo que usted esperaba.

Casi todas las clases en Java pueden ser superclases de otras, es decir, pueden extenderse. Las únicas clases que no pueden ser extendidas son aquellas declaradas como `final`.

2. Funcionamiento de la herencia en Java

Como ya hemos mencionado, al crear una subclase, ésta contiene tantos parámetros y métodos como su superclase. No es necesario volver a definirlos, aunque pueden sobrecargarse. Y además pueden crearse otros atributos u otros métodos nuevos. Veamos con más detalle cómo funciona la herencia en Java.

2.1. Creación de una subclase: atributos y métodos

Al construir un objeto de la subclase, debemos construir un objeto de su superclase y darle valores a los atributos propios de la subclase. La construcción de un objeto de la superclase se hace en el cuerpo del constructor de la subclase llamando al método `super()`. A este método hay que pasarle tantos parámetros como tenga el constructor de la superclase y no devuelve nada. Se encarga de inicializar los parámetros que la subclase ha heredado de su superclase con su lista de parámetros. En el cuerpo del constructor de la subclase, tras la llamada a `super()`, inicializamos los atributos propios de la subclase (si los hay).

Los métodos de la superclase se heredan automáticamente. Si en el cuerpo de la clase extendida añadimos la cabecera de un método de la superclase y definimos entre llaves su comportamiento, estaremos sobreescribiendo el método. A partir de ese momento, ese método funcionará para la subclase tal y como lo acabamos de definir, no como estaba definido en su superclase. Los métodos que se creen por primera vez en la subclase, sólo existirán para ésta, nunca para la superclase.

Veamos un ejemplo de cómo implementar una superclase:

```
/**
 * Clase que modela las figuras geométricas.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class Figura {
  //Atributos
  protected double perimetro = 0.0;
  protected double area = 0.0;

  //Constructor nulo
  public Figura() {}

  //Métodos observadores
  public double Area() {return area;}

  public double Perimetro() {return perimetro;}

  //Métodos modificadores
  protected void CalcularArea() {}

  protected void CalcularPerimetro() {}
}
```

Ahora implementamos una subclase:

```
/**
 * Clase que modela rectángulos.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class Rectangulo extends Figura {
  //Atributos
  private double base, altura;

  //Constructor nulo
  public Rectangulo() {}
  //Constructor
  public Rectangulo(double b, double a) {
    super();
    base = b;
    altura = a;
  }

  //Métodos observadores
```

```
  public double Base() {
    return base;
  }

  public double Altura() {
    return altura;
  }

  //Métodos modificadores
  public void CalcularArea() {
    area = base * altura;
  }

  public void CalcularPerimetro() {
    perimetro = 2 * base + 2 * altura;
  }
}
```

Ejercicio 3 Añada a la superclase `Figura` dos subclases llamadas `Circulo` y `Poligono` que modelen círculos y polígonos regulares, respectivamente. Cree otra subclase llamada `Cuadrado`, esta vez que extienda a la clase `Rectangulo` y que modele cuadrados. Compruebe sus códigos con las soluciones que aparecen en 8.2.

2.2. Utilizando una subclase: resolución de llamadas a métodos

Una vez creado un objeto de la subclase, se pueden realizar desde él llamadas a métodos tanto de la subclase como de la superclase. Al invocar un método, el compilador busca si el método está definido en la subclase. Si encuentra en la subclase un método que coincida con la llamada, lo ejecuta. Si no, busca un método coincidente en su superclase para ejecutarlo.

En caso de que la superclase haya heredado de otra, se seguiría buscando un método coincidente en la siguiente superclase, hasta encontrar el método o llegar a la clase base de todas y si no se ha encontrado, lanzará un error. A continuación podemos ver un ejemplo del uso de las subclases:

```
/**
 * Programa que hace uso de figuras geométricas.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class UsaFiguras
{
  public static void main (String args[])
  {
    Rectangulo[] rectangulos;
```

```
    Scanner teclado = new Scanner(System.in);

    System.out.println("Introduzca el número de rectángulos que desea: ");
    int numRect = teclado.nextInt();
    rectangulos = new Rectangulo[numRect];
    for(int i = 0; i < numRect; ++i) {
      System.out.print("Introduzca la base del rectángulo:");
      double base = teclado.nextDouble();
      System.out.print("Introduzca la altura del rectángulo:");
      double altura = teclado.nextDouble();
      rectangulos[i] = new Rectangulo(base, altura);
    }
    System.out.println("Ahora calculamos y mostramos el área y el perímetro de " +
      "cada rectángulo.");
    for(int i = 0; i < numRect; ++i) {
      rectangulos[i].CalcularPerimetro();
      rectangulos[i].CalcularArea();
      System.out.println("Rectangulo " + (i+1) + ": perímetro = " +
        rectangulos[i].Perimetro() + "; área = " + rectangulos[i].Area());
    }
  }
}
```

Ejercicio 4 Modifique el programa de prueba anterior para que también trabaje con las clases `Circulo`, `Poligono` y `Cuadrado` que implementó en el ejercicio anterior. Cuando termine, eche un vistazo al programa de prueba del apartado 8.3.

2.3. Protección de datos mediante permisos

La protección y ocultación de datos también está presente en la herencia. Una subclase sólo heredará los campos públicos (`public`) y protegidos (`protected`) de su superclase. Ya sean atributos o métodos.

De este modo, es posible definir atributos y métodos privados (`private`) en la superclase que sólo pueden ser accedidos desde la propia superclase, aunque esto no es lo más frecuente en la herencia. Normalmente, pueden llegar a definirse algunos atributos o métodos como privados, pero no la mayoría.

Lo habitual en una superclase es que contenga campos protegidos y públicos. Los atributos y métodos protegidos serán heredados por las subclases, pero no podrán ser accedidos desde el exterior. De este modo, se protege a estos campos para que no puedan ser accedidos desde cualquier clase, mientras que se permite que las subclases lo hereden.

Pero la ocultación de datos también es extensible a los objetos: un objeto protegido, por ejemplo, sólo puede ser accedido por código de paquete y por subclases que lo tengan como miembro heredado.

Ejercicio 5 Implemente una clase persona que almacene los datos relevantes de una persona: nombre, apellidos, año de nacimiento, DNI, ciudad de residencia, país de residencia y estudios. Cree también una subclase usuario que almacene los datos de registro de un usuario en una página web: el alias del usuario, contraseña, nombre real, año de nacimiento y el país de residencia. Implemente estas clases de modo que cada una de ella pueda acceder sólo a los datos especificados. Realice un programa de prueba para comprobar su resultado. Cuando lo termine, compárelo con los códigos del apartado 8.4.

---

3. Compatibilidad de tipos

La clase base (o superclase) siempre es compatible con todos sus tipos derivados (sus subclases), pero no al revés. Esto es porque las subclases tienen todos los atributos de la superclase, pero la superclase no tiene los atributos específicos de sus subclases.

El que la superclase sea compatible con todas sus subclases permite que una subclase pueda ser referenciada con un objeto que ha sido declarado de la superclase. Esto permite, por ejemplo, crear un *array* de objetos de la superclase y que luego referencie a distintos objetos de las subclases.

```
Figura f = new Figura();
Rectangulo r1 = f;        //Esto es legítimo
Rectangulo r2 = new Rectangulo(8, 4);
r2 = f;                   //Esto es ilegal
Figura f2 = r2;           //Esto es válido
```

Los objetos se pueden referenciar mediante diferentes clases de referencias, siempre que sean compatibles bajo herencia. También se puede forzar la compatibilidad al revés, que un objeto de la subclase referencie a uno de la superclase, mediante *casting*, pero esto no siempre será legal.

```
RefSubclase = (TipoSubClase) RefSuperClase
```

Ejercicio 6 Modifique código obtenido en el ejercicio 4 para que las figuras que se crean se almacenen en un array de figuras. Luego se recorre el array mostrando el perímetro y el área de las figuras. Compare su programa de prueba con el que puede observar en el apartado 8.5.

---

4. Sobrescritura

Ya hemos mencionado anteriormente que la herencia permite la sobrescritura. La sobrescritura es la modificación parcial del comportamiento de un fragmento de código, en este caso, de una superclase, para adaptarla a un nuevo comportamiento en la subclase.

Con la sobrescritura no perdemos el comportamiento de la función de la superclase que deseamos sobrescribir. Sin embargo este ya no estará disponible en la subclase a menos que se vuelva a especificar. Sobrescribir no implica añadir comportamiento a un método de la superclase, sino cambiar su comportamiento. Esto permite versatilizar y personalizar el comportamiento de funciones que, aunque reciben y devuelven la misma cantidad y tipo de parámetros, actúan de manera distinta en alguna subclase. El que una función se sobrescriba en una subclase, no afecta a las demás subclases. En las demás, dicha función sigue comportándose de la misma manera que lo hacía en la superclase.

Para llevar a cabo la sobrescritura de un método, éste mantiene su cabecera y se le cambia su cuerpo. Tenga en cuenta las siguientes pautas:

- Asegúrese de que el método a sobrescribir no entre en conflicto con métodos estáticos.
- La signatura del método original y del sobrescrito deben ser iguales.
- El método sobrescrito puede tener cláusula `throws`.
- El método sobrescrito sólo podrá lanzar excepciones declaradas en el método original.
- Los métodos declarados `final` no pueden ser sobrescritos.

Veamos un esquema de la sobrescritura:

```
class Superclase {
  ...
  tipo_dev metodo1(tipo_arg1 arg1, ...) {
    //Cuerpo del metodo1: puede estar vacío o no
  }
}

...

class Subclase extends Superclase {
  ...
  tipo_dev metodo1(tipo_arg1 arg1, ...) {
    //Cuerpo del metodo1:
    //esto será lo que se ejecute en la llamada a objSubclase.metodo1(args)
  }
}
```

Un ejemplo concreto de sobrescritura lo podemos encontrar en el apartado 2.1 al definir los métodos `CalcularArea()` y `CalcularPerimetro()` en la clase `Figura` e implementarlos posteriormente en la clase `Rectangulo`.

Ejercicio 7 Modifique el programa de prueba del ejercicio 6 de modo que al recorrer el array sólo tenga que llamar a una función para mostrar por pantalla todos los datos pertenecientes a la figura en cuestión. Para ello, añada un método en la clase `Figura` que muestre los datos disponibles en dicha clase, y sobrescríbalo en las demás clases para que muestre todos los datos posibles en cada una de ellas. En el apartado 8.6 se encuentran todas las clases que han sufrido cambios. Compare sus códigos con los que aparecen allí.

---

5. Interfaces

Una interfaz es un mecanismo que permite, entre otras cosas, realizar herencia múltiple en Java, ya que una clase puede implementar varias interfaces a la vez. Las interfaces son clases abstractas, o sin implementación definida. Son clases compuestas sólo de signaturas de métodos, por defecto públicos. Las interfaces no pueden tener variables y sus métodos están vacíos, por lo que no pueden construirse objetos de ese tipo. La palabra reservada del lenguaje que las define es `interface`.

```
interface Interfaz {
  tipo_dev metodo1 (tipo_arg1 arg1, tipo_arg2 arg2, ...);
  tipo_dev metodo2 (tipo_arg1 arg1, tipo_arg2 arg2, ...);
}
```

Mientras que las superclases se extendían (`class Subclase extends Superclase`), las interfaces se implementan (`class Clase implements Interfaz`). Al implementar una interfaz, la clase que la implementa describe un nuevo comportamiento para los métodos de la interfaz. Una interfaz puede ser definida en cualquier fichero.

Veamos un ejemplo de una interfaz. Crearemos una interfaz con los métodos de la clase `Figura` que calculaban el perímetro y el área de una figura dada. Declarando estos métodos en esta interfaz, podemos eliminarlos de la clase `Figura`.

```
interface CalculosFigura {
  void CalcularArea();
  void CalcularPerimetro();
}
```

Una misma interfaz puede ser implementada en diferentes clases. Cada una de estas clases le dará un comportamiento distinto a los métodos de la interfaz. Por tanto, el comportamiento exacto de las funciones de la interfaz anterior seguirá estando definido en las subclases que hereden de `Figura` y que implementen esta interfaz. Veamos cómo quedaría, a grandes rasgos, la clase `Rectangulo`:

```
public class Rectangulo extends Figura implements CalculosFigura {
  //Atributos
  ...

  //Constructores
  ...

  //Métodos observadores
  ...

  //Métodos de la interfaz
  public void CalcularArea() {
    area = base * altura;
  }
  public void CalcularPerimetro() {
    perimetro = 2 * base + 2 * altura;
  }
}
```

Como recordará, una subclase sólo puede extender a una superclase a la vez, pero puede implementar varias interfaces a la vez.

Ejercicio 8 Modifique las clases `Figura`, `Rectangulo`, `Circulo`, `Poligono` y `Cuadrado` usadas en los ejercicios anteriores. Añada a interfaz del ejemplo en alguno de los ficheros, por ejemplo, en `Figura.java`, y elimine estas funciones de la clase `Figura`. Haga las modificaciones necesarias para que las clases restantes implementen la interfaz. Por último, modifique el programa de prueba del ejercicio anterior en donde sea necesario. Tenga en cuenta a qué clase pertenecen ahora los métodos `CalcularArea()` y `CalcularPerimetro()`. Compruebe que todo se ejecute correctamente y compare sus cambios con los que aparecen en los códigos del apartado 8.7.

---

6. Biblioteca de contenedores

En esta sección veremos algunas clases definididas en la biblioteca de Java que nos puede ayudar a almacenar nuestras clases de una forma más cómoda de lo que ya conocemos. Estas son clases de la biblioteca de contenedores, que están parametrizadas para poder almacenar cualquier tipo de clase. Las clases que veremos son `ArrayList`, `LinkedList` y `Stack`.

6.1. Clase `ArrayList`

La clase `ArrayList` que encontraremos en el paquete `java.util` implementa una lista de cualquier tipo de clase. La lista que creamos con ella permite el acceso aleatorio a cualquier miembro de la lista. Otra cualidad interesante es que permite el almacenamiento multiobjeto, es decir, en un mismo objeto de la clase `ArrayList` podemos almacenar objetos de distintas clases. Aunque en las últimas versiones de Java se aconseja que los objetos que creemos de la clase `ArrayList` tan sólo almacenen un tipo concreto de objetos, y que lo tipifiquemos en su declaración:

```
ArrayList<Tipo>
```

A continuación veremos algunos de los métodos más importantes de esta clase.

- `ArrayList(int capacidad)`: constructor de la clase. Permite crear un objeto `ArrayList` de la capacidad especificada.
- `boolean add(Objet dato)`: método modificador de la clase. Permite añadir al final de la lista el objeto pasado como parámetro. Devolverá `true` si el objeto ha sido insertado en la lista.
- `void add(int i, Object dato)`: método modificador de la clase. Permite añadir en la posición indicada el objeto pasado como parámetro.
- `void clear()`: método modificador de la clase. Borra todos los elementos de la lista y la deja vacía. Lista para volver a añadir elementos si se desea.
- `boolean isEmpty()`: método observador de la clase. Indica si la lista está vacía. Si lo está, devuelve `true`, en otro caso, devuelve `false`.
- `Object set(int indice, Object dato)`: método modificador de la clase. Reemplaza el objeto que se encuentra en la posición indicada por el otro objeto que se le pasa. Devuelve el objeto que ha sido reemplazado.
- `Object[] toArray()`: método observador de la clase. Devuelve un array que contiene todos los objetos almacenados en la lista en el mismo orden que se encuentran en la lista.

Para conocer más métodos de la clase `ArrayList` o para obtener más información sobre la clase, consulte la documentación online.

A continuación puede ver un ejemplo de un programa que usa `ArrayList` para almacenar rectángulos.

```
/**
 * Programa que hace uso de figuras geométricas.
 * Utiliza la versión 2 de las clases Figura y Rectangulo.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class UsaFigurasArrayList
{
  public static void main (String args[])
  {
    ArrayList<Rectangulo> figuras;
    Scanner teclado = new Scanner(System.in);

    System.out.print("Introduzca el número de rectángulos que desea: ");
    int numFig = teclado.nextInt();
    figuras = new ArrayList(numFig);

    for(int i = 0; i < numFig; ++i) {
      System.out.println("Nuevo rectángulo.");
      System.out.print("Introduzca la base del rectángulo:");
      double base = teclado.nextDouble();
      System.out.print("Introduzca la altura del rectángulo:");
      double altura = teclado.nextDouble();
      Rectangulo r = new Rectangulo(base, altura);
      r.CalcularPerimetro();
      r.CalcularArea();
      if(!figuras.add(r)) {
        System.err.println("Error al introducir elemento en el ArrayList");
      }
    }

    System.out.println("Ahora mostramos el área y el perímetro de " +
      "cada figura.");
    for(int i = 0; i < numFig; ++i) {
      System.out.println("Figura " + (i+1) + ": perímetro = " +
        figuras.get(i).Perimetro() + "; área = " + figuras.get(i).Area());
    }
  }
}
```

Ejercicio 9 Añada las clases `Circulo`, `Poligono` y `Cuadrado` al programa de prueba del ejemplo anterior sobre la utilización de `ArrayList`. Añada objetos de estas clases en un objeto de la clase `ArrayList` y realice operaciones con ellos. Puede consultar su programa con el que se encuentra en el apartado 8.8.

6.2. Clase `LinkedList`

La clase `LinkedList` también pertenece al paquete `java.util`. Esta clase implementa una lista enlazada para almacenar objetos de cualquier clase. La lista es modelada de la forma habitual, pero proporcionando una API transparente al programador. Al igual que ocurría con la clase anterior, es recomendable tipificar los objetos que se creen pertenecientes a esta clase.

```
LinkedList<Tipo>
```

Esta clase implementa los métodos habituales en las listas. Para aprender sobre los métodos que implementa, consulte la documentación.

Un ejemplo de cómo utilizar `LinkedList` es el que sigue:

```
/**
 * Programa que hace uso de figuras geométricas.
 * Utiliza la versión 2 de las clases Figura y Rectangulo.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class UsaFigurasLinkedList
{
  public static void main (String args[])
  {
    LinkedList<Rectangulo> figuras;
    figuras = new LinkedList();
    Scanner teclado = new Scanner(System.in);
    String opc;
    do {
      System.out.println("Nuevo rectángulo.");
      System.out.print("Introduzca la base del rectángulo:");
      double base = teclado.nextDouble();
      System.out.print("Introduzca la altura del rectángulo:");
      double altura = teclado.nextDouble();
      Rectangulo r = new Rectangulo(base, altura);
      r.CalcularPerimetro();
      r.CalcularArea();
      if(!figuras.add(r)) {
        System.err.println("Error al introducir elemento en el ArrayList");
      }
```

```
      System.out.print("¿Desea introducir otro rectángulo? (S/N) ");
      opc = teclado.next();
    } while (opc.compareToIgnoreCase("s") == 0 ||
            opc.compareToIgnoreCase("si") == 0);

    System.out.println("Ahora mostramos el área y el perímetro de " +
      "cada figura.");
    for(int i = 0; i < figuras.size(); ++i) {
      System.out.println("Figura " + (i+1) + ": perímetro = " +
        figuras.get(i).Perimetro() + "; área = " + figuras.get(i).Area());
    }
  }
}
```

Ejercicio 10 Añada las clases `Circulo`, `Poligono` y `Cuadrado` al programa de prueba del ejemplo anterior sobre la utilización de `LinkedList`. Añada objetos de estas clases en un objeto de la clase `LinkedList` y realice operaciones con ellos. Puede consultar su programa con el que se encuentra en el apartado 8.9.

6.3. Clase `Stack`

La clase `Stack`, contenida también en `java.util`, implementa una pila de elementos de cualquier clase. Es una subclase de la clase `Vector` del mismo paquete. Al igual que las clases anteriores, es recomendable tipificar la clase con el tipo de datos que contendrá al declararla:

```
Stack<Tipo>
```

Esta clase incorpora un API específica de un tipo abstracto de datos que modele una pila (TAD pila). Los métodos propios de relevantes de este API son:

- `boolean empty()`: método observador de la clase. Indica si la pila está vacía.
- `Object peek()`: método observador de la clase. Devuelve el objeto que se encuentra en la cima de la pila, pero sin modificarla.
- `Object pop()`: método modificador de la clase. Devuelve el objeto que se encuentra en la cima de la pila y lo elimina de la pila.
- `Object push(Object dato)`: método modificador de la clase. Inserta un nuevo elemento en la cima de la pila.
- `int search(Object o)`: método observador de la clase. Devuelve la posición en la que se encuentra el objeto que se le pasa como parámetro.

Para más información sobre estos métodos, consulte la documentación online. Además, como esta clase hereda de la clase `Vector`, incorpora todos los métodos de `Vector`. Es posible usar estos métodos, pero así se pervierte el funcionamiento del TAD pila que modela la clase `Stack`.

El siguiente código de ejemplo ilustra la utilización de la clase `Stack`:

```
/**
 * Programa que hace uso de figuras geométricas.
 * Utiliza la versión 2 de las clases Figura y Rectangulo.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class UsaFigurasStack
{
  public static void main (String args[])
  {
    Stack<Rectangulo> figuras;
    figuras = new Stack();
    Scanner teclado = new Scanner(System.in);
    String opc;
    do {
      System.out.println("Nuevo rectángulo.");
      System.out.print("Introduzca la base del rectángulo:");
      double base = teclado.nextDouble();
      System.out.print("Introduzca la altura del rectángulo:");
      double altura = teclado.nextDouble();
      Rectangulo r = new Rectangulo(base, altura);
      r.CalcularPerimetro();
      r.CalcularArea();
      figuras.push(r);
      System.out.print("¿Desea introducir otro rectángulo? (S/N) ");
      opc = teclado.next();
    } while (opc.compareToIgnoreCase("s") == 0 ||
             opc.compareToIgnoreCase("si") == 0);

    System.out.println("Ahora mostramos el área y el perímetro de cada figura.");
    while(!figuras.empty()) {
      System.out.println("Sacamos los elementos de la pila y mostramos " +
        "su perímetro y su área.");
      Figura f = figuras.pop();
      System.out.println("Figura: perímetro = " + f.Perimetro() + "; área = " +
        f.Area());
    }
  }
}
```

Ejercicio 11 Añada las clases `Circulo`, `Poligono` y `Cuadrado` al programa de prueba del ejemplo anterior sobre la utilización de `Stack`. Añada objetos de estas clases en un objeto de la clase `Stack` y realice operaciones con ellos. Puede consultar su programa con el que se encuentra en el apartado 8.10.

---

7. Ejercicios

En esta sección encontrará ejercicios adicionales con los que afianzar los conocimientos adquiridos en este guión.

Ejercicio 12 Realice un programa que almacene y permita añadir, borrar y editar usuarios de un servicio. Puede usar las clases `Persona` y `UsuarioWeb` del ejercicio 5. Almacene los usuarios en el contenedor que considere más oportuno. Puede comprobar su programa con el código del apartado 8.11.

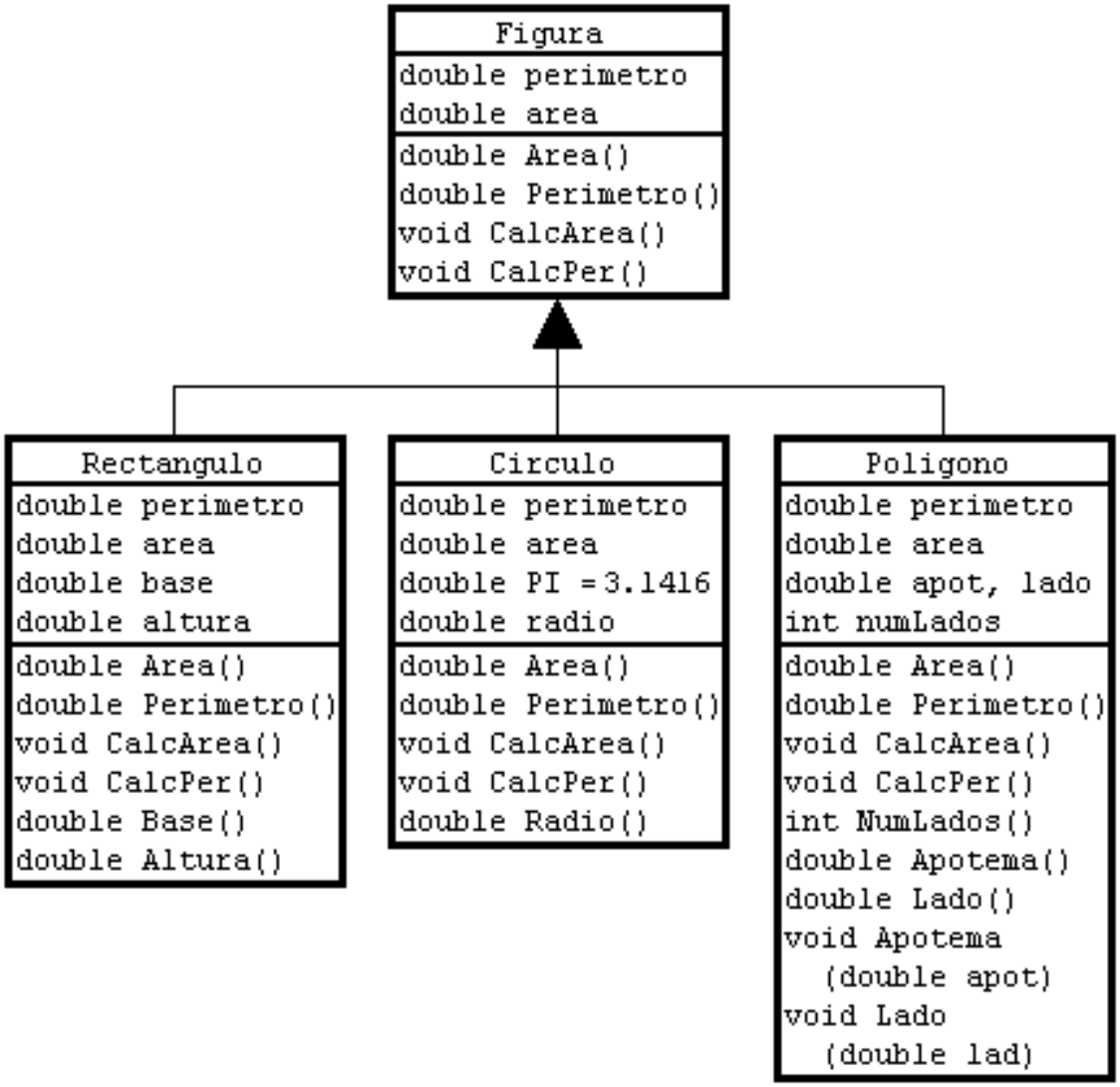
---

8. Soluciones de los ejercicios

En esta sección encontrará las soluciones a los ejercicios propuestos a lo largo del guión.

8.1. Ejercicio 1

El siguiente diagrama modela las entidades `Figura`, `Rectangulo`, `Circulo` y `Poligono`:



8.2. Ejercicio 3

Subclase `Circulo` que modela círculos:

```
/**
 * Clase que modela círculos.
 *
```

```
 * @author Natalia Partera
 * @version 1.0
 */

public class Circulo extends Figura {
  //Atributos
  private double PI = 3.1416;
  private double radio;

  //Constructor nulo
  public Circulo() {}
  //Constructor
  public Circulo(double rad) {
    super();
    radio = rad;
  }

  //Métodos observadores
  public double Radio() {
    return radio;
  }

  //Métodos modificadores
  public void CalcularArea() {
    area = PI * radio * radio;
  }

  public void CalcularPerimetro() {
    perimetro = 2 * PI * radio;
  }
}
```

El siguiente código es la clase `Poligono` que modela polígonos regulares:

```
/**
 * Clase que modela polígonos regulares.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class Poligono extends Figura {
  //Atributos
  private double apotema, lado;
  private int numLados;

  //Constructor nulo
  public Poligono() {}
  //Constructores
  public Poligono(int nLados) {
    super();
    numLados = nLados;
```

```
  }

  public Poligono(int nLados, double lad) {
    super();
    numLados = nLados;
    lado = lad;
  }

  public Poligono(int nLados, double lad, double apot) {
    super();
    numLados = nLados;
    lado = lad;
    apotema = apot;
  }

  //Métodos observadores
  public int NumLados() {
    return numLados;
  }

  public double Apotema() {
    return apotema;
  }

  public double Lado() {
    return lado;
  }

  //Métodos modificadores
  public void Apotema(double apot) {
    apotema = apot;
  }

  public void Lado (double lad) {
    lado = lad;
  }

  public void CalcularArea() {
    if(Perimetro() == 0)
      CalcularPerimetro();
    if(apotema == 0)
      System.out.println("No se ha podido calcular el área: es necesario " +
        "inicializar el valor de la apotema.");
    else
      area = (Perimetro() * apotema) / 2;
  }

  public void CalcularPerimetro() {
    perimetro = numLados * lado;
  }
}
```

Por último, la clase `Cuadrado` que modela cuadrados extendiendo la clase `Rectangulo`:

```
/**
 * Clase que modela cuadrados.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class Cuadrado extends Rectangulo {
  //Constructor nulo
  public Cuadrado() {}
  //Constructor
  public Cuadrado(double lad) {
    super(lad, lad);
  }

  //Métodos observadores
  public double Lado() {
    return Base();
  }
}
```

8.3. Ejercicio 4

A continuación se muestra el programa de prueba para las clases del ejercicio anterior.

```
/**
 * Programa que hace uso de figuras geométricas.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class UsaSubFiguras
{
  public static void main (String args[])
  {
    Scanner teclado = new Scanner(System.in);
    Rectangulo rectangulo;
    Circulo circulo;
    Poligono hexagono;
    Cuadrado cuadrado;

    System.out.println("Inicializando rectangulo ...");
    System.out.print("Introduzca la base del rectángulo:");
    double base = teclado.nextDouble();
    System.out.print("Introduzca la altura del rectángulo:");
    double altura = teclado.nextDouble();
    rectangulo = new Rectangulo(base, altura);
```

```
    System.out.println("Inicializando círculo ...");
    System.out.print("Introduzca el radio del círculo:");
    double radio = teclado.nextDouble();
    circulo = new Circulo(radio);

    System.out.println("Inicializando hexágono ...");
    System.out.print("Introduzca la longitud de un lado del hexágono:");
    double lado = teclado.nextDouble();
    System.out.print("Introduzca el valor de la apotema del hexágono:");
    double apotema = teclado.nextDouble();
    hexagono = new Poligono(6, lado, apotema);

    System.out.println("Inicializando cuadrado ...");
    System.out.print("Introduzca el lado del cuadrado:");
    double ladoC = teclado.nextDouble();
    cuadrado = new Cuadrado(ladoC);

    System.out.println("Ahora calculamos y mostramos el área y el perímetro de "
      "cada figura.");
    rectangulo.CalcularPerimetro();
    rectangulo.CalcularArea();
    circulo.CalcularPerimetro();
    circulo.CalcularArea();
    hexagono.CalcularPerimetro();
    hexagono.CalcularArea();
    cuadrado.CalcularPerimetro();
    cuadrado.CalcularArea();

    System.out.println("Rectangulo: perímetro = " + rectangulo.Perimetro() + "; "
      "área = " + rectangulo.Area());
    System.out.println("Círculo: perímetro = " + circulo.Perimetro() + "; "
      "área = " + circulo.Area());
    System.out.println("Hexágono: perímetro = " + hexagono.Perimetro() + "; "
      "área = " + hexagono.Area());
    System.out.println("Cuadrado: perímetro = " + cuadrado.Perimetro() + "; "
      "área = " + cuadrado.Area());
  }
}
```

8.4. Ejercicio 5

El siguiente código corresponde a la implementación de la clase `Persona`:

```
/**
 * Clase que modela personas.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class Persona {
  protected String nombre, apellidos;
```

```
protected int añoNacimiento;
private int dni;
private String ciudadR;
protected String paisR;
private String estudios;


public Persona() {}

protected Persona(String nom, String ape, int año, String pais) {
  nombre = nom;
  apellidos = ape;
  añoNacimiento = año;
  paisR = pais;
}

public Persona(String nom, String ape, int año, int dni, String ciudad,
  String pais, String est) {
  nombre = nom;
  apellidos = ape;
  añoNacimiento = año;
  this.dni = dni;
  ciudadR = ciudad;
  paisR = pais;
  estudios = est;
}

protected String Nombre() {
  return nombre;
}

protected String Apellidos() {
  return apellidos;
}

protected int AñoNacimiento() {
  return añoNacimiento;
}

private int Dni() {
  return dni;
}

private String Ciudad() {
  return ciudadR;
}

protected String Pais() {
  return paisR;
}

private String Estudios() {
```

```
    return estudios;
  }

  public void Ciudad(String ciudad) {
    ciudadR = ciudad;
  }

  public void Pais(String pais) {
    paisR = pais;
  }

  public void Estudios(String est) {
    estudios = est;
  }

  public void VerDatos() {
    System.out.println(Nombre() + " " + Apellidos() + ", con DNI: " + Dni() +
      " nació en " + AñoNacimiento());
    System.out.println("Vive en " + Ciudad() + ", " + Pais() + ". Tiene estudios " +
      "de " + Estudios());
  }
}
```

El siguiente código corresponde a la subclase `UsuarioWeb`:

```
/**
 * Clase que modela los datos almacenados de los usuarios de una web.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class UsuarioWeb extends Persona {
  private String nombreUsuario, contraseña;

  public UsuarioWeb() {}

  public UsuarioWeb(String alias, String contras, String nom, String ape,
    int añoNac, String pais) {
    super(nom, ape, añoNac, pais);
    nombreUsuario = alias;
    contraseña = contras;
  }

  public String Alias() {
    return nombreUsuario;
  }

  public String Contraseña() {
    return contraseña;
  }
```

```
    public void Contraseña(String contras) {
      contraseña = contras;
    }

    public boolean ValidarContraseña(String contras) {
      if(contraseña == contras)
        return true;
      else
        return false;
    }
  }
```

Por último, el programa de prueba para ambas clases es el que sigue:

```
  /**
   * Programa que simula el guardado de datos de los usuarios de una web.
   *
   * @author Natalia Partera
   * @version 1.0
   */

  import java.util.*;

  public class UsaUsuarioWeb
  {
    public static void main (String args[])
    {
      Persona p1;
      UsuarioWeb user2;

      p1 = new Persona("Pepito", "de los Palotes", 1923, 12345678, "Salamanca",
        "España", "Master en decir tonterías");
      user2 = new UsuarioWeb("mBombo", "palillos11", "Manolo", "el del Bombo", 1900,
        "España");

      System.out.println("Persona p1:");
      System.out.println("Nombre: " + p1.Nombre());    //protected
      System.out.println("Apellidos: " + p1.Apellidos());    //protected
      System.out.println("Año de nacimiento: " + p1.AñoNacimiento());    //protected
//    System.out.println("DNI: " + p1.Dni());    //private
//    System.out.println("Ciudad: " + p1.Ciudad());    //private
      System.out.println("Pais: " + p1.Pais());    //protected
//    System.out.println("Estudios: " + p1.Estudios());    //private
      p1.Ciudad("Lisboa");    //public
      p1.Pais("Portugal");    //public
      p1.Estudios("Master en tocar los palillos");    //public
      p1.VerDatos();    //public

      System.out.println("User user2:");
      System.out.println("Alias: " + user2.Alias());    //public
      System.out.println("Contraseña: " + user2.Contraseña());    //public
      System.out.println("Nombre: " + user2.Nombre());    //protected
```

```
      System.out.println("Apellidos: " + user2.Apellidos());    //protected
      System.out.println("Año de nacimiento: " + user2.AñoNacimiento());  //protected
//    System.out.println("Ciudad: " + user2.Ciudad());    //private
      System.out.println("Pais: " + user2.Pais());    //protected
    }
  }
```

8.5. Ejercicio 6

A continuación, el programa de prueba donde todas las figuras se insertan en un array:

```
/**
 * Programa que hace uso de figuras geométricas.
 *
 * @author Natalia Partera
 * @version 2.0
 */

import java.util.*;

public class UsaFiguras
{
  public static void main (String args[])
  {
    Figura[] figuras;
    Scanner teclado = new Scanner(System.in);

    System.out.print("Introduzca el número de figuras que desea: ");
    int numFig = teclado.nextInt();
    figuras = new Figura[numFig];
    for(int i = 0; i < numFig; ++i) {
      switch(i % 4) {
        case 0: //Circulo
          System.out.print("Introduzca el radio del círculo:");
          double radio = teclado.nextDouble();
          figuras[i] = new Circulo(radio);
          break;
        case 1: //Cuadrado
          System.out.print("Introduzca el lado del cuadrado:");
          double ladoC = teclado.nextDouble();
          figuras[i] = new Cuadrado(ladoC);
          break;
        case 2: //Rectángulo
          System.out.print("Introduzca la base del rectángulo:");
          double base = teclado.nextDouble();
          System.out.print("Introduzca la altura del rectángulo:");
          double altura = teclado.nextDouble();
          figuras[i] = new Rectangulo(base, altura);
          break;
        case 3: //Polígono
          System.out.print("Introduzca el número de lados del polígono:");
          int numLados = teclado.nextInt();
```

```
          System.out.print("Introduzca la longitud de un lado del polígono:");
          double ladoP = teclado.nextDouble();
          System.out.print("Introduzca el valor de la apotema del polígono:");
          double apotema = teclado.nextDouble();
          figuras[i] = new Poligono(numLados, ladoP, apotema);
          break;
      }
    }
    System.out.println("Ahora calculamos y mostramos el área y el perímetro de " +
      "cada figura.");
    for(int i = 0; i < numFig; ++i) {
      figuras[i].CalcularPerimetro();
      figuras[i].CalcularArea();
      System.out.println("Figura " + (i+1) + ": perímetro = " +
        figuras[i].Perimetro() + "; área = " + figuras[i].Area());
    }
  }
}
```

8.6. Ejercicio 7

Tras añadir la nueva función, la clase `Figura` queda así:

```
/**
 * Clase que modela las figuras geométricas.
 *
 * @author Natalia Partera
 * @version 2.0
 */

public class Figura {
  //Atributos
  protected double perimetro = 0.0;
  protected double area = 0.0;

  //Constructor nulo
  public Figura() {}

  //Métodos observadores
  public double Area() {return area;}

  public double Perimetro() {return perimetro;}

  public void VerDatos() {
    System.out.println("La figura mide " + perimetro + " metros de perímetro.");
    System.out.println("La figura tiene un área de " + area + " m^2.");
  }

  //Métodos modificadores
  protected void CalcularArea() {}

  protected void CalcularPerimetro() {}
```

```
}
```

En la clase `Rectangulo` se sobrescribe el método `VerDatos`:

```
/**
 * Clase que modela rectángulos.
 *
 * @author Natalia Partera
 * @version 2.0
 */

public class Rectangulo extends Figura {
  //Atributos
  private double base, altura;

  //Constructor nulo
  public Rectangulo() {}
  //Constructor
  public Rectangulo(double b, double a) {
    super();
    base = b;
    altura = a;
  }

  //Métodos observadores
  public double Base() {
    return base;
  }

  public double Altura() {
    return altura;
  }

  public void VerDatos() {
    System.out.println("La base del rectangulo mide " + base +
      " metros y la altura, " + altura + " metros.");
    System.out.println("El rectángulo mide " + perimetro + " metros de perímetro.");
    System.out.println("Y tiene un área de " + area + " m^2.");
  }

  //Métodos modificadores
  public void CalcularArea() {
    area = base * altura;
  }

  public void CalcularPerimetro() {
    perimetro = 2 * base + 2 * altura;
  }
}
```

La clase `Circulo` queda así:

```
/**
 * Clase que modela círculos.
 *
 * @author Natalia Partera
 * @version 2.0
 */

public class Circulo extends Figura {
  //Atributos
  private double PI = 3.1416;
  private double radio;

  //Constructor nulo
  public Circulo() {}
  //Constructor
  public Circulo(double rad) {
    super();
    radio = rad;
  }

  //Métodos observadores
  public double Radio() {
    return radio;
  }

  public void VerDatos() {
    System.out.println("El radio del círculo mide " + radio + " metros.");
    System.out.println("El círculo mide " + perimetro + " metros de perímetro.");
    System.out.println("Y tiene un área de " + area + " m^2.");
  }

  //Métodos modificadores
  public void CalcularArea() {
    area = PI * radio * radio;
  }

  public void CalcularPerimetro() {
    perimetro = 2 * PI * radio;
  }
}
```

El polígono también añade el nuevo método:

```
/**
 * Clase que modela polígonos regulares.
 *
 * @author Natalia Partera
 * @version 2.0
 */

public class Poligono extends Figura {
  //Atributos
```

```
private double apotema, lado;
private int numLados;

//Constructor nulo
public Poligono() {}
//Constructores
public Poligono(int nLados) {
  super();
  numLados = nLados;
}

public Poligono(int nLados, double lad) {
 super();
  numLados = nLados;
  lado = lad;
}

public Poligono(int nLados, double lad, double apot) {
  super();
  numLados = nLados;
  lado = lad;
  apotema = apot;
}

//Métodos observadores
public int NumLados() {
  return numLados;
}

public double Apotema() {
  return apotema;
}

public double Lado() {
  return lado;
}

public void VerDatos() {
  System.out.println("El polígono tiene " + numLados + ". Cada lado mide " + lado
    + " metros y su apotema mide " + apotema + " metros.");
  System.out.println("El polígono mide " + perimetro + " metros de perímetro.");
  System.out.println("Y tiene un área de " + area + " m^2.");
}

//Métodos modificadores
public void Apotema(double apot) {
  apotema = apot;
}

public void Lado (double lad) {
  lado = lad;
}
```

```
  public void CalcularArea() {
    if(Perimetro() == 0)
      CalcularPerimetro();
    if(apotema == 0)
      System.out.println("No se ha podido calcular el área: es necesario " +
        "inicializar el valor de la apotema.");
    else
      area = (Perimetro() * apotema) / 2;
  }

  public void CalcularPerimetro() {
    perimetro = numLados * lado;
  }
}
```

Así queda la clase `Cuadrado` tras sobrescribir la función:

```
/**
 * Clase que modela cuadrados.
 *
 * @author Natalia Partera
 * @version 2.0
 */

public class Cuadrado extends Rectangulo {
  //Constructor nulo
  public Cuadrado() {}
  //Constructor
  public Cuadrado(double lad) {
    super(lad, lad);
  }

  //Métodos observadores
  public double Lado() {
    return Base();
  }

  public void VerDatos() {
    System.out.println("Un lado cualquiera del cuadrado mide " + Lado() +
      " metros.");
    System.out.println("El cuadrado mide " + perimetro + " metros de perímetro.");
    System.out.println("Y tiene un área de " + area + " m^2.");
  }
}
```

A continuación puede comprobar los cambios hechos en el programa de prueba del ejercicio 6:

```
/**
 * Programa que hace uso de figuras geométricas.
 *
 * @author Natalia Partera
```

```
 * @version 3.0
 */

import java.util.*;

public class UsaFiguras
{
  public static void main (String args[])
  {
    Figura[] figuras;
    Scanner teclado = new Scanner(System.in);

    System.out.print("Introduzca el número de figuras que desea: ");
    int numFig = teclado.nextInt();
    figuras = new Figura[numFig];
    for(int i = 0; i < numFig; ++i) {
      switch(i%4) {
        case 0: //Circulo
          System.out.print("Introduzca el radio del círculo (en metros):");
          double radio = teclado.nextDouble();
          figuras[i] = new Circulo(radio);
          break;
        case 1: //Cuadrado
          System.out.print("Introduzca el lado del cuadrado (en metros): ");
          double ladoC = teclado.nextDouble();
          figuras[i] = new Cuadrado(ladoC);
          break;
        case 2: //Rectángulo
          System.out.print("Introduzca la base del rectángulo (en metros): ");
          double base = teclado.nextDouble();
          System.out.print("Introduzca la altura del rectángulo (en metros): ");
          double altura = teclado.nextDouble();
          figuras[i] = new Rectangulo(base, altura);
          break;
        case 3: //Polígono
          System.out.print("Introduzca el número de lados del polígono: ");
          int numLados = teclado.nextInt();
          System.out.print("Introduzca la longitud de un lado del polígono " +
            "(en metros): ");
          double ladoP = teclado.nextDouble();
          System.out.print("Introduzca el valor de la apotema del polígono " +
            "(en metros): ");
          double apotema = teclado.nextDouble();
          figuras[i] = new Poligono(numLados, ladoP, apotema);
          break;
      }
    }
    System.out.println("Ahora calculamos y mostramos el área y el perímetro de " +
      "cada figura.");
    for(int i = 0; i < numFig; ++i) {
      figuras[i].CalcularPerimetro();
      figuras[i].CalcularArea();
```

```
      figuras[i].VerDatos();
    }
  }
}
```

8.7. Ejercicio 8

Siguiendo las instrucciones del enunciado, el fichero `Figuras.java` debería quedarle así:

```
/**
 * Clase que modela las figuras geométricas.
 *
 * @author Natalia Partera
 * @version 3.0
 */

interface CalculosFigura {
  void CalcularArea();
  void CalcularPerimetro();
}

public class Figura {
  //Atributos
  protected double perimetro = 0.0;
  protected double area = 0.0;

  //Constructor nulo
  public Figura() {}

  //Métodos observadores
  public double Area() {return area;}
  public double Perimetro() {return perimetro;}
  public void VerDatos() {
    System.out.println("La figura mide " + perimetro + " metros de perímetro.");
    System.out.println("La figura tiene un área de " + area + " m^2.");
  }
}
```

De la misma manera, de acuerdo con el enunciado, el fichero `Rectangulo.java` debe quedar:

```
/**
 * Clase que modela rectángulos.
 *
 * @author Natalia Partera
 * @version 3.0
 */

public class Rectangulo extends Figura implements CalculosFigura {
  //Atributos
  private double base, altura;

  //Constructor nulo
```

```
  public Rectangulo() {}
  //Constructor
  public Rectangulo(double b, double a) {
    super();
    base = b;
    altura = a;
  }

  //Métodos observadores
  public double Base() {
    return base;
  }

  public double Altura() {
    return altura;
  }

  public void VerDatos() {
    System.out.println("La base del rectangulo mide " + base +
      " metros y la altura, " + altura + " metros.");
    System.out.println("El rectángulo mide " + perimetro + " metros de perímetro.");
    System.out.println("Y tiene un área de " + area + " m^2.");
  }

  //Métodos modificadores
  public void CalcularArea() {
    area = base * altura;
  }

  public void CalcularPerimetro() {
    perimetro = 2 * base + 2 * altura;
  }
}
```

El fichero `Circulo.java` debe parecerse a este código tras las modificaciones:

```
/**
 * Clase que modela círculos.
 *
 * @author Natalia Partera
 * @version 3.0
 */

public class Circulo extends Figura implements CalculosFigura {
  //Atributos
  private double PI = 3.1416;
  private double radio;

  //Constructor nulo
  public Circulo() {}
  //Constructor
  public Circulo(double rad) {
```

```
    super();
    radio = rad;
  }

  //Métodos observadores
  public double Radio() {
    return radio;
  }

  public void VerDatos() {
    System.out.println("El radio del círculo mide " + radio + " metros.");
    System.out.println("El círculo mide " + perimetro + " metros de perímetro.");
    System.out.println("Y tiene un área de " + area + " m^2.");
  }

  //Métodos modificadores
  public void CalcularArea() {
    area = PI * radio * radio;
  }

  public void CalcularPerimetro() {
    perimetro = 2 * PI * radio;
  }
}
```

Su fichero `Poligono.java` debería parecerse al siguiente:

```
/**
 * Clase que modela polígonos regulares.
 *
 * @author Natalia Partera
 * @version 3.0
 */

public class Poligono extends Figura implements CalculosFigura{
  //Atributos
  private double apotema, lado;
  private int numLados;

  //Constructor nulo
  public Poligono() {}
  //Constructores
  public Poligono(int nLados) {
    super();
    numLados = nLados;
  }

  public Poligono(int nLados, double lad) {
    super();
    numLados = nLados;
    lado = lad;
  }
```

```
  public Poligono(int nLados, double lad, double apot) {
    super();
    numLados = nLados;
    lado = lad;
    apotema = apot;
  }

  //Métodos observadores
  public int NumLados() {
    return numLados;
  }

  public double Apotema() {
    return apotema;
  }

  public double Lado() {
    return lado;
  }

  public void VerDatos() {
    System.out.println("El polígono tiene " + numLados + ". Cada lado mide " + lado
      + " metros y su apotema mide " + apotema + " metros.");
    System.out.println("El polígono mide " + perimetro + " metros de perímetro.");
    System.out.println("Y tiene un área de " + area + " m^2.");
  }

  //Métodos modificadores
  public void Apotema(double apot) {
    apotema = apot;
  }

  public void Lado (double lad) {
    lado = lad;
  }

  public void CalcularArea() {
    if(Perimetro() == 0)
      CalcularPerimetro();
    if(apotema == 0)
      System.out.println("No se ha podido calcular el área: es necesario " +
        "inicializar el valor de la apotema.");
    else
      area = (Perimetro() * apotema) / 2;
  }

  public void CalcularPerimetro() {
    perimetro = numLados * lado;
  }
}
```

El fichero `Cuadrado.java` no sufre cambios.

Como ahora los métodos `CalcularArea()` y `CalcularPerimetro()` pertenecen a las subclases de la clase `Figura`, hay que invocarlos desde objetos de las subclases. Modificaremos el programa para que se ejecuten estos métodos antes de insertar las figuras en el array:

```
/**
 * Programa que hace uso de figuras geométricas.
 *
 * @author Natalia Partera
 * @version 4.0
 */

import java.util.*;

public class UsaFiguras
{
  public static void main (String args[])
  {
    Figura[] figuras;
    Scanner teclado = new Scanner(System.in);

    System.out.print("Introduzca el número de figuras que desea: ");
    int numFig = teclado.nextInt();
    figuras = new Figura[numFig];
    for(int i = 0; i < numFig; ++i) {
      switch(i%4) {
        case 0: //Circulo
          System.out.print("Introduzca el radio del círculo (en metros):");
          double radio = teclado.nextDouble();
          Circulo circ = new Circulo(radio);
          circ.CalcularPerimetro();
          circ.CalcularArea();
          figuras[i] = circ;
          break;
        case 1: //Cuadrado
          System.out.print("Introduzca el lado del cuadrado (en metros): ");
          double ladoC = teclado.nextDouble();
          Cuadrado cuad = new Cuadrado(ladoC);
          cuad.CalcularPerimetro();
          cuad.CalcularArea();
          figuras[i] = cuad;
          break;
        case 2: //Rectángulo
          System.out.print("Introduzca la base del rectángulo (en metros): ");
          double base = teclado.nextDouble();
          System.out.print("Introduzca la altura del rectángulo (en metros): ");
          double altura = teclado.nextDouble();
          Rectangulo r = new Rectangulo(base, altura);
          r.CalcularPerimetro();
          r.CalcularArea();
          figuras[i] = r;
```

```
          break;
        case 3: //Polígono
          System.out.print("Introduzca el número de lados del polígono: ");
          int numLados = teclado.nextInt();
          System.out.print("Introduzca la longitud de un lado del polígono " +
            "(en metros): ");
          double ladoP = teclado.nextDouble();
          System.out.print("Introduzca el valor de la apotema del polígono " +
            "(en metros): ");
          double apotema = teclado.nextDouble();
          Poligono p = new Poligono(numLados, ladoP, apotema);
          p.CalcularPerimetro();
          p.CalcularArea();
          figuras[i] = p;
          break;
      }
    }
    System.out.println("Ahora mostramos el área y el perímetro de cada figura.");
    for(int i = 0; i < numFig; ++i)
      figuras[i].VerDatos();
  }
}
```

8.8. Ejercicio 9

El siguiente programa almacena figuras en un objeto de la clase `ArrayList`:

```
/**
 * Programa que hace uso de figuras geométricas.
 *
 * @author Natalia Partera
 * @version 2.0
 */

import java.util.*;

public class UsaFigurasArrayList
{
  public static void main (String args[])
  {
    ArrayList<Figura> figuras;
    Scanner teclado = new Scanner(System.in);

    System.out.print("Introduzca el número de figuras que desea: ");
    int numFig = teclado.nextInt();
    figuras = new ArrayList(numFig);

    for(int i = 0; i < numFig; ++i) {
      System.out.println("Nueva figura.");
      switch(i % 4) {
        case 0: //Circulo
          System.out.print("Introduzca el radio del círculo (en metros):");
```

```
          double radio = teclado.nextDouble();
          Circulo circ = new Circulo(radio);
          circ.CalcularPerimetro();
          circ.CalcularArea();
          if(!figuras.add(circ))
            System.err.println("Error al introducir elemento en el ArrayList");
          break;
        case 1: //Cuadrado
          System.out.print("Introduzca el lado del cuadrado (en metros): ");
          double ladoC = teclado.nextDouble();
          Cuadrado cuad = new Cuadrado(ladoC);
          cuad.CalcularPerimetro();
          cuad.CalcularArea();
          if(!figuras.add(cuad))
            System.err.println("Error al introducir elemento en el ArrayList");
          break;
        case 2: //Rectángulo
          System.out.print("Introduzca la base del rectángulo (en metros): ");
          double base = teclado.nextDouble();
          System.out.print("Introduzca la altura del rectángulo (en metros): ");
          double altura = teclado.nextDouble();
          Rectangulo r = new Rectangulo(base, altura);
          r.CalcularPerimetro();
          r.CalcularArea();
          if(!figuras.add(r))
            System.err.println("Error al introducir elemento en el ArrayList");
          break;
        case 3: //Polígono
          System.out.print("Introduzca el número de lados del polígono: ");
          int numLados = teclado.nextInt();
          System.out.print("Introduzca la longitud de un lado del polígono " +
            "(en metros): ");
          double ladoP = teclado.nextDouble();
          System.out.print("Introduzca el valor de la apotema del polígono " +
            "(en metros): ");
          double apotema = teclado.nextDouble();
          Poligono p = new Poligono(numLados, ladoP, apotema);
          p.CalcularPerimetro();
          p.CalcularArea();
          if(!figuras.add(p))
            System.err.println("Error al introducir elemento en el ArrayList");
          break;
      }
    }

    System.out.println("Ahora mostramos el área y el perímetro de cada figura.");
    for(int i = 0; i < numFig; ++i) {
      System.out.println("Figura " + (i+1) + ": perímetro = " +
        figuras.get(i).Perimetro() + "; área = " + figuras.get(i).Area());
    }
  }
}
```

8.9. Ejercicio 10

El siguiente programa almacena figuras en un objeto de la clase `LinkedList`:

```
/**
 * Programa que hace uso de figuras geométricas.
 *
 * @author Natalia Partera
 * @version 2.0
 */

import java.util.*;

public class UsaFigurasLinkedList
{
  public static void main (String args[])
  {
    LinkedList<Figura> figuras;
    figuras = new LinkedList();
    Scanner teclado = new Scanner(System.in);
    String opc;
    int i = 0;
    do {
      System.out.println("Nueva figura.");
      switch(i % 4) {
        case 0: //Circulo
          System.out.print("Introduzca el radio del círculo (en metros):");
          double radio = teclado.nextDouble();
          Circulo circ = new Circulo(radio);
          circ.CalcularPerimetro();
          circ.CalcularArea();
          if(!figuras.add(circ))
            System.err.println("Error al introducir elemento en el ArrayList");
          break;
        case 1: //Cuadrado
          System.out.print("Introduzca el lado del cuadrado (en metros): ");
          double ladoC = teclado.nextDouble();
          Cuadrado cuad = new Cuadrado(ladoC);
          cuad.CalcularPerimetro();
          cuad.CalcularArea();
          if(!figuras.add(cuad))
            System.err.println("Error al introducir elemento en el ArrayList");
          break;
        case 2: //Rectángulo
          System.out.print("Introduzca la base del rectángulo (en metros): ");
          double base = teclado.nextDouble();
          System.out.print("Introduzca la altura del rectángulo (en metros): ");
          double altura = teclado.nextDouble();
          Rectangulo r = new Rectangulo(base, altura);
          r.CalcularPerimetro();
          r.CalcularArea();
          if(!figuras.add(r))
            System.err.println("Error al introducir elemento en el ArrayList");
```

```
          break;
        case 3: //Polígono
          System.out.print("Introduzca el número de lados del polígono: ");
          int numLados = teclado.nextInt();
          System.out.print("Introduzca la longitud de un lado del polígono " +
            "(en metros): ");
          double ladoP = teclado.nextDouble();
          System.out.print("Introduzca el valor de la apotema del polígono " +
            "(en metros): ");
          double apotema = teclado.nextDouble();
          Poligono p = new Poligono(numLados, ladoP, apotema);
          p.CalcularPerimetro();
          p.CalcularArea();
          if(!figuras.add(p))
            System.err.println("Error al introducir elemento en el ArrayList");
          break;
      }
      System.out.print("¿Desea introducir otra figura? (S/N) ");
      opc = teclado.next();
      ++i;
    } while (opc.compareToIgnoreCase("s") == 0 ||
             opc.compareToIgnoreCase("si") == 0);

    System.out.println("Ahora mostramos el área y el perímetro de cada figura.");
    for(int j = 0; j < figuras.size(); ++j) {
      System.out.println("Figura " + (j+1) + ": perímetro = " +
        figuras.get(j).Perimetro() + "; área = " + figuras.get(j).Area());
    }
  }
}
```

8.10. Ejercicio 11

El siguiente programa almacena figuras en un objeto de la clase `Stack`:

```
/**
 * Programa que hace uso de figuras geométricas.
 *
 * @author Natalia Partera
 * @version 2.0
 */

import java.util.*;

public class UsaFigurasStack
{
  public static void main (String args[])
  {
    Stack<Figura> figuras;
    figuras = new Stack();
    Scanner teclado = new Scanner(System.in);
    String opc;
```

```
int i = 0;
do {
  System.out.println("Nueva figura.");
  switch(i % 4) {
    case 0: //Circulo
      System.out.print("Introduzca el radio del círculo (en metros):");
      double radio = teclado.nextDouble();
      Circulo circ = new Circulo(radio);
      circ.CalcularPerimetro();
      circ.CalcularArea();
      figuras.push(circ);
      break;
    case 1: //Cuadrado
      System.out.print("Introduzca el lado del cuadrado (en metros): ");
      double ladoC = teclado.nextDouble();
      Cuadrado cuad = new Cuadrado(ladoC);
      cuad.CalcularPerimetro();
      cuad.CalcularArea();
      figuras.push(cuad);
      break;
    case 2: //Rectángulo
      System.out.print("Introduzca la base del rectángulo (en metros): ");
      double base = teclado.nextDouble();
      System.out.print("Introduzca la altura del rectángulo (en metros): ");
      double altura = teclado.nextDouble();
      Rectangulo r = new Rectangulo(base, altura);
      r.CalcularPerimetro();
      r.CalcularArea();
      figuras.push(r);
      break;
    case 3: //Polígono
      System.out.print("Introduzca el número de lados del polígono: ");
      int numLados = teclado.nextInt();
      System.out.print("Introduzca la longitud de un lado del polígono " +
        "(en metros): ");
      double ladoP = teclado.nextDouble();
      System.out.print("Introduzca el valor de la apotema del polígono " +
        "(en metros): ");
      double apotema = teclado.nextDouble();
      Poligono p = new Poligono(numLados, ladoP, apotema);
      p.CalcularPerimetro();
      p.CalcularArea();
      figuras.push(p);
      break;
  }
  System.out.print("¿Desea introducir otra figura? (S/N) ");
  opc = teclado.next();
  ++i;
} while (opc.compareToIgnoreCase("s") == 0 ||
         opc.compareToIgnoreCase("si") == 0);

System.out.println("Ahora mostramos el área y el perímetro de cada figura.");
```

```
    while(!figuras.empty()) {
      System.out.println("Sacamos los elementos de la pila y mostramos " +
        "su perímetro y su área.");
      Figura f = figuras.pop();
      System.out.println("Figura: perímetro = " + f.Perimetro() + "; área = " +
        f.Area());
    }
  }
}
```

8.11. Ejercicio 12

En este programa se permite añadir, borrar, editar y listar los usuarios existentes. Para ello, hemos considerado más adecuado usar un contenedor `LinkedList`. Nos hemos basado en las clases `Persona` y `UsuarioWeb`, adaptándolas un poco. A continuación se muestra la adaptación de la clase `Persona`:

```
/**
 * Clase que modela personas.
 *
 * @author Natalia Partera
 * @version 2.0
 */

public class Persona {
  protected String nombre, apellidos;
  protected int añoNacimiento;
  private int dni;
  private String ciudadR;
  protected String paisR;

  public Persona() {}

  public Persona(String nom, String ape, int año, String pais) {
    nombre = nom;
    apellidos = ape;
    añoNacimiento = año;
    paisR = pais;
  }

  public Persona(String nom, String ape, int año, int dni, String ciudad,
  String pais) {
    nombre = nom;
    apellidos = ape;
    añoNacimiento = año;
    this.dni = dni;
    ciudadR = ciudad;
    paisR = pais;
  }

  protected String Nombre() {
    return nombre;
  }
```

```
  protected String Apellidos() {
    return apellidos;
  }

  protected int AñoNacimiento() {
    return añoNacimiento;
  }

  private int Dni() {
    return dni;
  }

  private String Ciudad() {
    return ciudadR;
  }

  protected String Pais() {
    return paisR;
  }

  protected void Ciudad(String ciudad) {
    ciudadR = ciudad;
  }

  protected void Pais(String pais) {
    paisR = pais;
  }
}
```

Aquí puede ver la adaptación de la clase `UsuarioWeb`:

```
/**
 * Clase que modela los datos almacenados de los usuarios de una web.
 *
 * @author Natalia Partera
 * @version 2.0
 */

public class UsuarioWeb extends Persona {
  private String nombreUsuario, contraseña;

  public UsuarioWeb() {}

  public UsuarioWeb(String alias, String contras, String nom, String ape,
  int añoNac, String pais) {
    super(nom, ape, añoNac, pais);
    nombreUsuario = alias;
    contraseña = contras;
  }

  public UsuarioWeb(String alias, String contras, String nom, String ape,
```

```
  int añoNac, int dni, String ciudad, String pais) {
    super(nom, ape, añoNac, dni, ciudad, pais);
    nombreUsuario = alias;
    contraseña = contras;
  }

  public String Alias() {
    return nombreUsuario;
  }

  protected String Contraseña() {
    return contraseña;
  }

  protected void Contraseña(String contras) {
    contraseña = contras;
  }

  public boolean ValidarContraseña(String contras) {
    if(contraseña == contras)
      return true;
    else
      return false;
  }
}
```

Por último, aquí está el programa de prueba:

```
/**
 * Programa que simula el guardado de datos de los usuarios de una web.
 *
 * @author Natalia Partera
 * @version 2.0
 */

import java.io.*;
import java.util.*;

public class UsaUsuarioWeb
{
  public static void main (String args[])
  {
    UsaUsuarioWeb web = new UsaUsuarioWeb();
    LinkedList<UsuarioWeb> usuarios = new LinkedList();
    boolean salir = false;
    int uId, año;
    UsuarioWeb user = new UsuarioWeb();
    String alias, contraseña, nombre, apellidos, pais;
    BufferedReader br = new BufferedReader (new InputStreamReader(System.in));

    try {
      do {
```

```
int opc = web.Menu();
switch (opc) {
  case 1:
    System.out.println("Introduzca su nombre de usuario: ");
    alias = br.readLine();
    System.out.println("Introduzca su contraseña: ");
    contraseña = br.readLine();
    System.out.println("Introduzca su nombre real: ");
    nombre = br.readLine();
    System.out.println("Introduzca sus apellidos reales: ");
    apellidos = br.readLine();
    System.out.println("Introduzca el año en que nació: ");
    año = Integer.parseInt(br.readLine());
    System.out.println("Introduzca el país en el que reside: ");
    pais = br.readLine();
    user = new UsuarioWeb(alias, contraseña, nombre, apellidos, año, pais);
    usuarios.add(user);
  break;
  case 2:
    if(usuarios.size() > 0) {
      for (int i = 0; i < usuarios.size(); ++i) {
        user = usuarios.get(i);
        System.out.println((i + 1) + ") " + user.Alias());
      }
      do {
        System.out.print("Indique el número del usuario que desea borrar:");
        uId = Integer.parseInt(br.readLine());
      } while (uId < 1 || uId > (usuarios.size() + 1));
      --uId;
      usuarios.remove(uId);
    }
    else {
      System.out.println("No hay ningún usuario.");
    }
  break;
  case 3:
    if(usuarios.size() > 0) {
      for (int i = 0; i < usuarios.size(); ++i) {
        user = usuarios.get(i);
        System.out.println((i + 1) + ") " + user.Alias());
      }
      do {
        System.out.print("Indique el número del usuario que desea editar:");
        uId = Integer.parseInt(br.readLine());
      } while (uId < 1 || uId > (usuarios.size() + 1));
      --uId;
      user = usuarios.get(uId);
      System.out.println("El usuario " + user.Alias() + " con contraseña " +
        user.Contraseña() + " se llama en realidad " + user.Nombre() + " " +
        user.Apellidos() + ". Nació en " + user.AñoNacimiento() +
        " y reside en " + user.Pais() + ".");
      System.out.println("Editando país. Introduzca país actual: ");
```

```
              pais = br.readLine();
              user.Pais(pais);
              System.out.println("Editando contraseña. Introduzca nueva " +
                "contraseña: ");
              contraseña = br.readLine();
              user.Contraseña(contraseña);
              usuarios.set(uId, user);
            }
            else {
              System.out.println("No hay ningún usuario.");
            }
          break;
          case 4:
            for (int i = 0; i < usuarios.size(); ++i) {
              user = usuarios.get(i);
              System.out.println((i + 1) + ") " + user.Alias());
            }
          break;
          case 5:
            salir = true;
          break;
        }
      } while(!salir);
    }
    catch(IOException exc) {
      System.err.println("¡Ups! Ha tenido lugar un error de E/S");
      exc.printStackTrace();
      System.exit(-1);
    }
  }


  int Menu() {
    Scanner teclado = new Scanner(System.in);
    int opc;

    System.out.println("Lista de opciones:");
    System.out.println("1) Añadir nuevo usuario.");
    System.out.println("2) Borrar usuario.");
    System.out.println("3) Editar usuario.");
    System.out.println("4) Ver todos los usuarios.");
    System.out.println("5) Salir.");
    do {
      System.out.print("¿Qué acción desea realizar? ");
      opc = teclado.nextInt();
    } while(opc < 1 || opc > 5);

    return opc;
  }
}
```