Programación Concurrente y de Tiempo Real

Guión de prácticas 2: Introducción a la Orientación a Objetos

Natalia Partera Jaime
Alumna colaboradora de la asignatura

Índice

1. Clases y Programación Orientada a Objetos

Como ya vimos anteriormente, Java es un lenguaje orientado a objetos. Todos los elementos de un programa en Java están organizados en clases (`class`), que proporcionan abstracción y encapsulación. La abstracción y la encapsulación son dos de los principios en los que se sustenta la orientación a objetos.

La *abstracción* consiste en considerar sólo aquellos aspectos de un problema que son importantes desde un cierto punto de vista y despreciar el resto. La *encapsulación* significa reunir en una cierta estructura a todos los elementos que a un cierto nivel de abstracción se pueden considerar pertenecientes a una misma entidad.

Por tanto, al programar en Java creamos clases que engloban datos y funciones muy relacionadas entre sí (*encapsulación*). Además, al usar las clases nos fijamos sólo en *qué* realiza la clase, mientras que cuando las implementamos nos centramos en *cómo* lo realiza (*abstracción*).

A rasgos generales, la idea fundamental del paradigma de la orientación a objetos es organizar el sistema en torno a los objetos que intervienen en él. Cada objeto pertenecerá a alguna clase; en el caso concreto de Java, cada objeto pertenece a una única clase. Y cada clase define el comportamiento externo de sus objetos, sirviendo de interfaz entre el objeto y sus clientes. Debido a otro principio del paradigma de la orientación a objetos, cada atributo y cada método de una clase puede ser usado, o no, desde el exterior de la clase según su visibilidad. La visibilidad de los elementos de la clase es asignada en la creación de la clase.

En resumen, en la programación orientada a objetos, las clases son consideradas como una herramienta que definen o modelan realidades concretas mediante la abstracción y encapsulación. Cada objeto de una clase se corresponde con un objeto concreto de la vida real, y la clase es la idea o el concepto general de lo que define a ese tipo de objeto. Sus características y funcionamiento son los atributos y métodos, respectivamente.

Pongamos un ejemplo práctico: piense en un objeto de la vida real, por ejemplo, un coche. Pues una clase definirá a cualquier coche, con sus características físicas y funcionamiento. Sin embargo, un objeto de esa clase será un coche en concreto. ¿Cuáles son las características que distinguen a un coche de otro? Por ejemplo: la marca, el modelo, el número de puertas, la clindrada, la potencia, el año de fabricación, ... Estas características son los atributos, serán las variables que habrá que declarar en la definición de la clase. Estas variables tomarán un valor cuando creemos un objeto concreto de esa clase, porque estaremos hablando de un objeto en particular en lugar de una clase en general. ¿Y cuál es el comportamiento que queremos tener en cuenta del coche? Pues aparte de conocer los datos concretos que ya hemos mencionado antes, podemos calcular el consumo de gasolina de un coche concreto por cada 100 kilómetros, por ejemplo. De cálculos como estos, y de informar sobre los datos del coche que guardamos anteriormente para protegerlos de cambios no deseados, se ocupan los métodos. Todos los métodos funcionan igual, sea cual sea el objeto de la clase, pero tomará las variables del objeto concreto desde el que se llama, por lo que el resultado que obtengamos será distinto para cada objeto.

1.1. Clases, objetos y ciclo de vida

Una clase actúa como interfaz entre sus objetos y el código en el que se utilicen dichos objetos, describiendo la estructura que tienen estos objetos y sirviendo como base para construirlos. Mientras que un objeto concreto es una instancia de una clase, que contiene datos y operaciones que se ocultan al exterior y son accesibles o no según lo indique su clase.

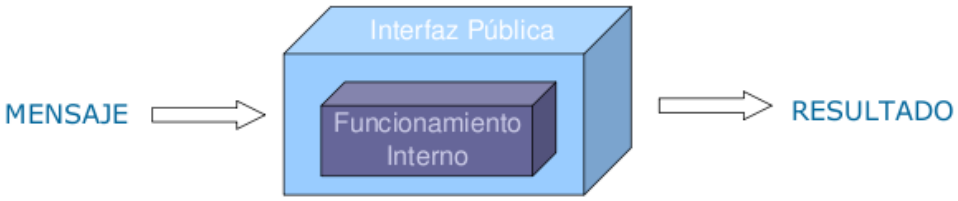

Figura 1: Esquema de la estructura de un objeto.

Ya vimos que toda clase tiene un nombre y define a los miembros que pertenecen a ella, ya sean atributos (*datos*) o métodos (*funciones*). Y que una vez que la clase es declarada, ésta funciona como un nuevo tipo de datos. Al igual que los tipos de datos, una clase ya declarada puede ser utilizada para declarar o para crear nuevos objetos de dicha clase.

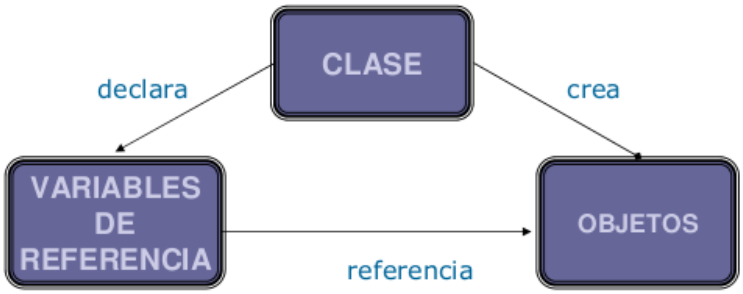

Figura 2: Funciones de una clase.

Recordemos la estructura general de una clase:

```
/**
 * Estructura de una clase en Java
 */
public class NombreDeLaClase
{
  //Cuerpo de la clase: atributos y métodos.

  //Declaración de los atributos de la clase

  //Declaración de los métodos de la clase
}
```

Los miembros de la clase pueden ser públicos, protegidos, privados o de paquete, según su visibilidad. Explicaremos la visibilidad algo más detenidamente en el apartado 1.3.

Para poder usar objetos de una clase, éstos deben ser declarados, creados e inicializados. La declaración de un objeto indica la intención de usar un objeto de cierto tipo con un cierto nombre y obliga al sistema a crear una variable que represente a ese objeto. La creación, construcción o definición de un objeto es

la reserva del espacio necesario en la memoria para el objeto. Y la inicialización consiste en poner en los atributos los valores adecuados para poder usar el objeto por primera vez.

Por otra parte, estos objetos dejarán de existir cuando se llame a su destructor. Normalmente el destructor es llamado al llegar al final de la función donde el objeto fue declarado. Nosotros no llamaremos al destructor, ya que Java se ocupa de ello con su mecanismo de recolección de basura.

En Java, la declaración de objetos se realiza de forma análoga a como se realiza la declaración de tipos simples: se escribe el nombre de la clase seguido del nombre que se le dará al objeto que se cree. La construcción de un objeto se realiza asignándole a la variable el valor devuelto en la llamada a `new`. El operador `new` devuelve una referencia al espacio de memoria reservado para el objeto. Este espacio de memoria es inicializado mediante la llamada al constructor que sigue a la llamada a `new`. El objeto debe de ser siempre inicializado con la llamada a su constructor. En el siguiente apartado veremos cómo se realizan estos pasos en Java. Recuerde que en otros lenguajes, pueden llevarse a cabo de distinta manera. El siguiente diagrama resume el ciclo de vida de los objetos en Java:

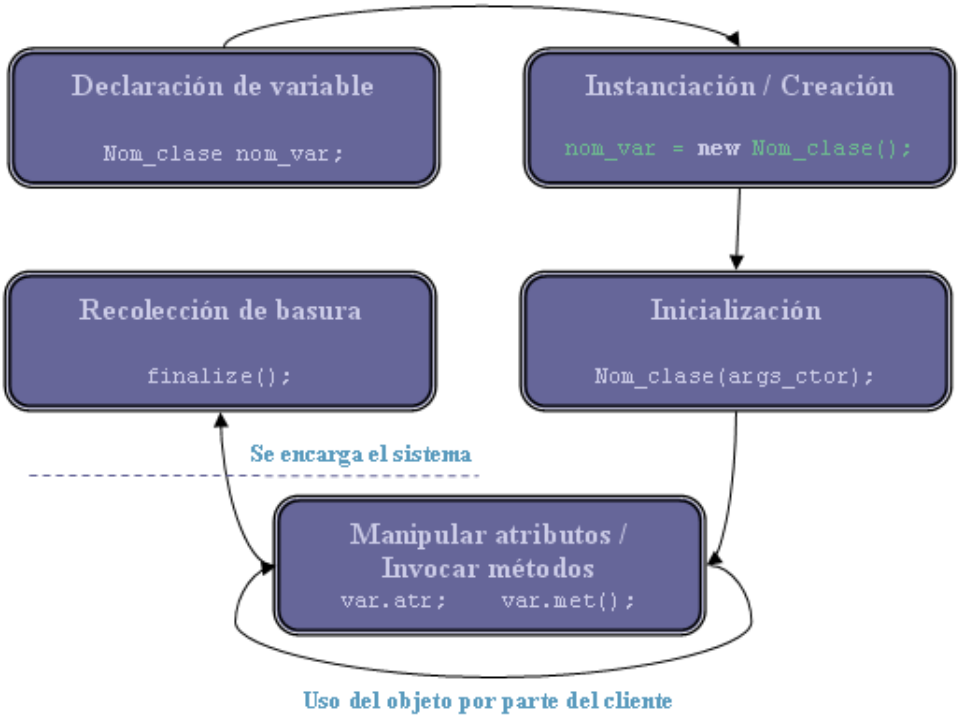

Figura 3: Ciclo de vida de los objetos en Java.

Antes indicamos que el operador `new` devuelve una referencia. El concepto de referencia es un concepto nuevo de la programación orientada a objetos. Una referencia es, como su propio nombre indica, una referencia a un objeto. Aunque la referencia guarda la dirección de memoria en la que está alojado el objeto, trabajar con referencias es como trabajar con el objeto. El programador no tiene que preocuparse de acceder al contenido de la dirección a la que apunta la referencia, ni trabajar con direcciones de memoria. Las referencias simplemente actúan como nombres de objetos. El nombre de un objeto es una referencia o variable de referencia.

Desde una variable de referencia se puede llamar a los atributos o métodos del objeto, utilizando un punto (`.`). La sintaxis utilizada para ello es:

```
referencia.atributo;
referencia.método(argumentos);
```

En Java es posible cambiar el valor de una referencia para que señale a otro objeto o a `null` con el operador de asignación (`=`). Si dos referencias apuntan a un mismo objeto, un cambio en el objeto producido desde una referencia, también será visible desde la otra referencia. A continuación, implementamos una clase que modela una cuenta bancaria:

```
/**
 * Clase que modela una cuenta bancaria.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class CuentaBancaria {
  //Atributos privados
  //Titular de la cuenta
  private String titular;
  //Código identificativo de la cuenta
  private long codigo;
  //Saldo actual de la cuenta
  private double saldo;

  //Constructor nulo
  public CuentaBancaria() {}
  //Constructor
  public CuentaBancaria(String titu, long cod, double sald) {
    titular = titu;
    codigo = cod;
    saldo = sald;
  }

  //Métodos observadores
  public String Titular() {
    return titular;
  }
  public long Codigo() {
    return codigo;
  }
  public double Saldo() {
    return saldo;
  }
  //Métodos modificadores
  public void Titular (String titu) {
    titular = titu;
  }
  public void Saldo (double sald) {
```

```
    saldo = sald;
  }
}
```

Veamos un programa de ejemplo con esta clase, donde dos referencias apuntan al mismo objeto:

```
/**
 * Programa que usa cuentas bancarias.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class UsaCuentaBancaria {
  public static void main (String args[]) {
    //Declaración de variables
    CuentaBancaria c1, c2;
    //Creación e inicialización de c1
    c1 = new CuentaBancaria("Pepito Grillo", 1234567890, 1500.85);
    //Observamos los valores de c1
    System.out.println("La cuenta c1 es de " + c1.Titular() + ".");
    System.out.println("El código de la cuenta c1 es " + c1.Codigo() +
      " y su saldo es de " + c1.Saldo() + " euros.");
    //Hacemos que c2 y c1 apunten al mismo objeto
    c2 = c1;
    //Observamos los valores de c2
    System.out.println("La cuenta c2 es de " + c2.Titular() + ".");
    System.out.println("El código de la cuenta c2 es " + c2.Codigo() +
      " y su saldo es de " + c2.Saldo() + " euros.");
    //Modificamos los valores de c2
    c2.Titular("Pinocho");
    c2.Saldo(c2.Saldo() - 350.65);
    //Observamos los valores actuales de c2
    System.out.println("La cuenta c2 ha sufrido cambios.");
    System.out.println("Ahora la cuenta c2 es de " + c2.Titular() + ".");
    System.out.println("El código de la cuenta c2 es " + c2.Codigo() +
      " y su saldo ahora es de " + c2.Saldo() + " euros.");
    //Observamos los valores actuales de c1
    System.out.println("Comprobamos la cuenta c1.");
    System.out.println("La cuenta c1 es de " + c1.Titular() + ".");
    System.out.println("El código de la cuenta c1 es " + c1.Codigo() +
      " y su saldo es de " + c1.Saldo() + " euros.");
  }
}
```

Ejercicio 1 Observe el código anterior de las clases `CuentaBancaria` y `UsaCuentaBancaria`. ¿Cuál cree que será la salida del programa? A continuación, descárguelo y compílelo para comprobar su solución.

Las clases y los objetos también se pueden representar gráficamente. Supongamos que queremos crear una clase llamada `Conversor` y queremos respresentarla gráficamente antes de implementarla. Nuestro conversor modelará el funcionamiento de un conversor numérico entre 2 unidades de medida relacionadas por un factor constante: 2 unidades monetarias, 2 unidades de medidas de longitud, 2 unidades de medidas de temperatura, etc.

1. Lo primero que debemos hacer es pensar cuáles son las características de un conversor de la vida real que nos interesa que tenga nuestra clase. En este caso, nos interesa guardar el valor del factor de conversión.

2. También debemos pensar qué queremos que haga nuestro conversor. Si el factor de conversión cambia porque se trate de 2 unidades monetarias, necesitaremos actualizar su valor. También nos puede interesar consultar el valor de conversión, para obtener la relación entre las 2 unidades de medida. Y por supuesto, nos interesa calcular a cuánto equivale una cierta cantidad.

Para representar nuestra clase `Conversor` gráficamente dibujaremos un rectángulo para la clase y uno por cada objeto. En el rectángulo de la clase tendremos varios apartados donde aparezcan: el nombre de la clase, los atributos y los métodos. En cada objeto aparecerá el nombre y del objeto y la clase a la que pertenece, y sus atributos con sus valores. A continuación puede ver un ejemplo.

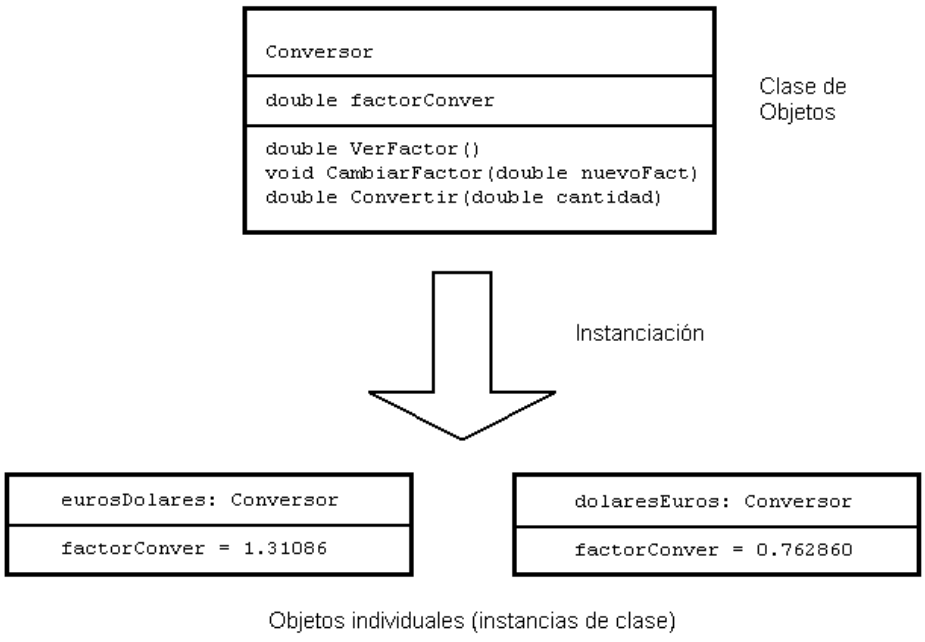

Figura 4: Representación gráfica de una clase y dos objetos.

El gráfico anterior representa la clase dada por el siguiente código:

```
/**
 * Clase que realiza conversiones de dólares a euros.
 *
 * @author Natalia Partera
```

```
 * @version 1.0
 */

public class Conversor
{
  //Atributo privado: factor de conversión
  private double factorConver;

  //Constructor
  public Conversor(double factor) {
    factorConver = factor;
  }

  //Método observador
  public double VerFactor() {
    return factorConver;
  }

  //Método modificador
  public void CambiarFactor(double nuevoFact) {
    factorConver = nuevoFact;
  }

  //Método observador
  public double Convertir(double cantidad) {
    return cantidad*factorConver;
  }
}
```

1.2. Atributos y métodos

Los *atributos* son los campos de los objetos donde se guardan datos o información. Un objeto puede tener atributos de cualquier clase que haya sido definida previamente. Es conveniente que desde el exterior de la clase sólo se pueda acceder a los atributos a través de los métodos que proporciona su clase. Para ello se usarán los métodos observadores y modificadores y la protección de datos mediante distintos valores de su visibilidad.

Los *métodos* son las funciones u operaciones que se pueden llevar a cabo con dicho objeto. Los métodos siempre pueden acceder a los atributos del objeto al que pertenecen. La definición de un método especifica el número y tipo de los argumentos requeridos (parámetros formales). Por eso, dentro de una clase un método no se identifica sólo por su nombre, sino también por la lista de parámetros formales.

Hay varios tipos de métodos:

- **Métodos Constructores:** permiten crear instancias de una clase.
- **Métodos Observadores:** permiten consultar la información almacenada en los atributos.
- **Métodos Modificadores:** permiten cambiar la información almacenada en los atributos.
- **Métodos Destructores:** destruyen objetos innecesarios y liberan recursos.

Ademas, los métodos pueden estar *sobrecargados*. Dos métodos están sobrecargados si se llaman igual pero tienen listas de parámetros diferentes. Por ejemplo, tenemos una clase con un atributo y queremos implementar dos métodos que utilicen ese atributo: uno para modificarlo y el otro para observar el valor del atributo. Pues podemos llamar a ambos métodos igual. Según cómo sea la llamada al método, el compilador sabrá si nos estamos refiriendo al método observador o al método modificador.

```
public class Clase
{
  private int atrib;

  //Constructor
  public Clase(int at) {
    atrib = at;
  }

  //Método observador
  public int Atributo() {
    return atrib;
  }

  //Método modificador
  public void Atributo(int at) {
    atrib = at;
  }
}
```

Todos los métodos de una clase pueden estar sobrecargados, excepto el destructor. De todos modos, en Java no necesitamos implementar destructores de las clases que creemos porque Java se ocupa de destruir los objetos con el recolector de basura, como ya mencionamos anteriormente. Un ejemplo de método sobrecargado es `System.out.println`, que permite enviar a la salida estándar texto, enteros, etc.

Ejercicio 2 Descargue el código anterior de la clase `Conversor` y compílelo para comprobar que no hay errores en el código. A continuación, sobrecargue los métodos que considere adecuados. Vuelva a compilar el código para comprobar errores de sintaxis. Puede comprobar su solución con el código que aparece en 6.1.

Ya mencionamos anteriormente que para usar poder usar objetos hay que declararlos, crearlos e inicializarlos.

1. El primer paso es declarar los objetos. Para ello debemos declarar la clase seguida del nombre de los objetos que queremos crear. El nombre de la clase actúa como un tipo de dato, y los objetos se crean y se referencian con variables del tipo. Estas variables son de referencia, y por tanto contienen la dirección de memoria de un objeto de la clase o el valor `null` para una referencia no válida.

2. A continuación hay que crearlos. En Java, este paso lo realizamos llamando al operador `new`, que asigna espacio de memoria dinámicamente. En Java, todos los objetos son creados en tiempo de ejecución mediante `new`.

3. Por último, los inicializamos. Para eso utilizamos un método especial que recibe el mismo nombre de la clase. A este método se le llama constructor. En cada clase hay un constructor que se encarga

de inicializar los campos del objeto creado. Estos campos pueden inicializarse con un valor concreto. Si no, se inicializan con `null` para evitar que almacenen "basura".

Veamos un ejemplo de la creación de objetos, usando la clase `Clase` del ejemplo anterior de la sobrecarga. Podemos declarar objetos de las siguientes formas:

```
Clase obj1;
Clase obj2 = new Clase(85);
```

La primera declaración no inicializa el objeto `obj1`, y la referencia por tanto sería `null`. La segunda declara el objeto `obj2`, lo crea mediante el operador `new`, y lo inicia mediante la llamada al constructor `Clase(85)`. Para iniciar el objeto `obj1` basta con incluir la siguiente sentencia:

```
obj1 = new Clase(2012);
```

El único propósito de un constructor es inicializar el objeto cuando éste ya ha sido creado. Por tanto, el constructor nunca devuelve un tipo de dato. El resultado de utilizar el constructor es una referencia (dirección de memoria) al objeto creado, que se guarda en una variable del mismo tipo. A partir de entonces, el objeto es referenciado a través de dicha variable. Es posible sobrecargar el constructor, siempre que cada versión tenga una lista de argumentos distinta de las demás versiones. A la versión del constructor con lista de argumentos vacía se le llama *constructor nulo*. Todas las clases poseen por defecto un constructor nulo. Si un programador define otro constructor distino del nulo, el constructor nulo por defecto queda sustituido por el constructor o los constructores definidos por el programador. Los constructores pueden ser públicos, privados, etc., lo cuál permite otorgar diferentes privilegios de creación de objetos.

```
public Clase()                      //Constructor nulo
{}
public Clase(int at)                //Constructor
{ atrib = at; }
```

Una vez que un objeto ha sido creado, podemos comunicarnos con él enviándole mensajes, a través de la interfaz que su conjunto de métodos nos define. Es decir, podremos comunicarnos con el objeto según lo que especifique su clase. Un mensaje será por tanto una llamada a un método del objeto, junto con los parámetros que en su caso sean necesarios.

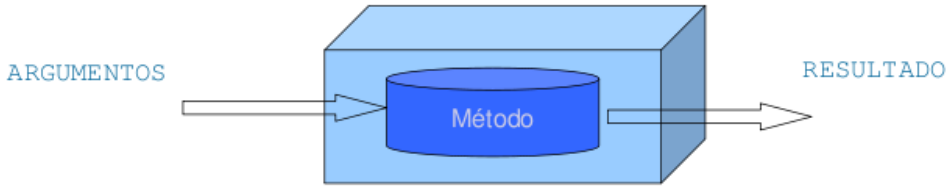

Figura 5: Representación gráfica de la llamada a un método de un objeto.

La notación general de acceso a un atributo o a un método de un objeto ya creado es:

```
objeto.atributo;
objeto.metodo(lista_de_parámetros);
```

Como veremos a continuación, desde el exterior de la clase sólo son accesibles los atributos o métodos que han sido declarados como `public`. En cambio, desde el interior de la clase son accesibles todos los atributos y métodos.

Un ejemplo de cómo crear y llamar a métodos de una clase es el siguiente código que utiliza la clase `Conversor` del apartado 1.1:

```
/**
 * Programa que utiliza la clase Conversor.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class ConversorLongitud
{
  public static void main (String[] args)
  {
    Conversor mInch;

    mInch = new Conversor(39.3700787);

    System.out.print("Conversor de metros a pulgadas");
    System.out.println("Factor de conversión: 1 metro = " + mInch.VerFactor()
      + " pulgadas");
    double metros = 0.25;
    System.out.println(metros + " metros son " + mInch.Convertir(metros)
      + " pulgadas");
    mInch.CambiarFactor(mInch.VerFactor()/100);
    double cm = 25;
    System.out.println(cm + " cm son " + mInch.Convertir(cm) + " pulgadas");
  }
}
```

Ejercicio 3 Escriba un programa en el que utilice la versión de la clase `Conversor` obtenida en el ejercicio 1 y realice conversiones entre euros y dólares. Su programa deberá crear dos objetos de la clase `Conversor` y utilizar cada método de la clase al menos una vez con cada uno de los objetos. Puede comparar su programa con el que se encuentra en el apartado 6.2.

En Java, los argumentos de tipos primitivos se pasan siempre por valor, mientras que los objetos (incluyendo los arrays que veremos en el apartado 2) se pasan siempre por referencia. Esto no necesita ser indicado por el programador, sino que es el propio compilador, examinando cada argumento, quien decide cómo debe pasarse. Esto no es estrictamente cierto: al pasar un objeto, lo que se pasa es la referencia al mismo por valor, teniendo esto el mismo efecto que un paso por referencia.

1.3. Protección de miembros de la clase

El principio de ocultación es el responsable de que sea posible o no el acceso a algunos miembros desde diferentes ámbitos. En Java los niveles de visibilidad de los miembros de una clase son:

- **Miembros públicos:** no tienen ningún tipo de protección. Son accesibles desde cualquier código que haya importado a estos miembros.
- **Miembros protegidos:** se puede acceder a ellos desde la propia clase, desde el resto de clases del paquete y desde las clases descendientes (entenderá mejor esto cuando conozca el mecanismo de *herencia*).
- **Miembros de paquete:** son elementos accesibles desde la propia clase y el resto de clases del paquete.
- **Miembros privados:** sólo son accesibles desde la propia clase.

Veamos un ejemplo. Tenemos la siguiente clase:

```
/**
 * Clase que ilustra el funcionamiento de los modificadores de acceso.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class Permisos {
  public int publico;
  protected int protegido;
  int sinModificador;
  private int privado;

  public Permisos(int i) {
    publico = i;
    protegido = i;
    sinModificador = i;
    privado = i;
  }

  protected Permisos(int i, int j) {
    publico = i;
    protegido = j;
    sinModificador = i;
    privado = i;
  }

  Permisos(int i, int j, int k, int l) {
    publico = i;
    protegido = j;
    sinModificador = k;
    privado = l;
  }

  private Permisos() {}

  public void MetPubl (int i) {
    MetProt(i+13);
  }
```

```
    protected void MetProt (int i) {
      protegido = i;
      MetPriv(i/3);
    }

    void MetSinM (int i) {
      sinModificador = i;
    }

    private void MetPriv (int i) {
      privado = i;
      MetSinM(i+6);
    }
  }
```

Y en el mismo directorio, tenemos el siguiente programa de prueba. Ambos pertenecen al mismo paquete.

```
  /**
   * Programa que ilustra el acceso según la visibilidad de los miembros.
   *
   * @author Natalia Partera
   * @version 1.0
   */

  public class UsaPermisos {
    public static void main (String[] args) {
      Permisos p1, p2, p3, p4;

      p1 = new Permisos(1);
      System.out.println("p1.publico = " + p1.publico);
      System.out.println("p1.protegido = " + p1.protegido);
      System.out.println("p1.sinModificador = " + p1.sinModificador);
//    System.out.println("p1.privado = " + p1.privado);    //Acceso no permitido
      p2 = new Permisos(1, 2);
      System.out.println("p2.publico = " + p2.publico);
      System.out.println("p2.protegido = " + p2.protegido);
      System.out.println("p2.sinModificador = " + p2.sinModificador);
//    System.out.println("p2.privado = " + p2.privado);    //Acceso no permitido
      p3 = new Permisos(1, 2, 3, 4); //Acceso no permitido
      System.out.println("p3.publico = " + p3.publico);
      System.out.println("p3.protegido = " + p3.protegido);
      System.out.println("p3.sinModificador = " + p3.sinModificador);
//    System.out.println("p3.privado = " + p3.privado);    //Acceso no permitido
//    p4 = new Permisos(); //Acceso no permitido

      p1.MetPubl(2);
      p2.MetProt(3);
      p3.MetSinM(4);
//    p1.MetPriv(5); //Acceso no permitido
```

```
    System.out.println("p1.publico = " + p1.publico);
    System.out.println("p1.protegido = " + p1.protegido);
    System.out.println("p1.sinModificador = " + p1.sinModificador);
    System.out.println("p2.publico = " + p2.publico);
    System.out.println("p2.protegido = " + p2.protegido);
    System.out.println("p2.sinModificador = " + p2.sinModificador);
    System.out.println("p3.publico = " + p3.publico);
    System.out.println("p3.protegido = " + p3.protegido);
    System.out.println("p3.sinModificador = " + p3.sinModificador);
  }
}
```

En el código del programa de prueba aparecen comentadas las sentencias que darían fallo al compilar por no tener acceso.

Quizá el siguiente cuadro resumen le ayude a recordar los permisos según la visibilidad de los elementos.

Modificador	**¿Accesible desde ...?**			
	Clase	Paquete	Subclases	El exterior
`public`	Sí	Sí	Sí	Sí
`protected`	Sí	Sí	Sí	No
Sin modificador	Sí	Sí	No	No
`private`	Sí	No	No	No

Cuadro 1: Visibilidad y acceso a miembros de la clase.

Ejercicio 4 Diseñe una clase que guarde información relativa a una asignatura: código, nombre, departamento, número de créditos, tipo de asignatura, curso y cuatrimestre en el que se imparte. Ningún atributo será accesible desde fuera de la propia clase, así que deberá construir también métodos observadores para cada atributo. Añada además algunos métodos modificadores. Cree también un programa en el que se pruebe la clase creada y donde compruebe cómo pueden ser modificados los atributos. Puede ver una posible solución en 6.3.

1.4. Errores y manejo de excepciones

Una excepción es un error producido en tiempo de ejecución. A veces ocurren errores en tiempo de ejecución y es necesario capturarlos y actuar en consecuencia según el error. En Java, estos errores se manejan con mecanismos que capturan la excepción y la tratan. Hay diferentes tipos de errores, y cada uno se representa con un objeto de excepción. Normalmente para capturar y tratar las excepciones se utiliza un bloque `try-catch` de prueba:

```
try {
  código que podría generar la excepción
}
catch (tipo_excepción_1 e) {
  instrucciones que tratan la excepción 1
```

```
}
catch (tipo_excepción_2 e) {
  instrucciones que tratan la excepción 2
}
finally {
  instrucciones en otro caso
}
```

Las cláusulas de captura hacen coincidir secuencialmente las excepciones causadas por cualquier instrucción de la parte `try` con los tipos de excepción, tratándolas de acuerdo con las instrucciones correspondientes.

```
/**
 * Programa en Java que transforma cadenas que almacenen números en números enteros.
 *
 * @author Natalia Partera
 * @version 1.0
 */

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

public class CadenasEnteros
{
  public static void main (String[] args)
  {
    String s1 = "2012";
    String s2 = "67.89";
    String s3 = "a213";

    try {
      System.out.print(s1 + " es ");
      int ent1 = new Integer(s1);
      System.out.println(ent1);

      System.out.print(s2 + " es ");
      int ent2 = Integer.parseInt(s2);
      System.out.println(ent2);

      System.out.print(s3 + " es ");
      int ent3 = Integer.valueOf(s3);
      System.out.println(ent3);
    } catch(NumberFormatException e) {
      System.err.println("Error: la cadena no contiene un entero parseable.");
      e.printStackTrace();
      System.exit(-1);
    }

  }
}
```

Ejercicio 5 Observe el código anterior. ¿Cree usted que lanzará alguna excepción? Razone su respuesta. Copie y compile el código. ¿El resultado era el esperado por usted? Corrija el código para que no lanzase excepciones si fuera necesario. Puede ver las respuestas en 6.4.

Otras veces, como al comprobar los argumentos recibidos por el método `main`, se puede realizar una comprobación simple en lugar del bloque de prueba. En el caso de los argumentos de `main`, si se produce un error, se puede usar un objeto `System.err` para enviar el mensaje de error a la pantalla y a continuación terminar el método llamando a `System.exit(estado)`, siendo `estado` un número. Para saber más sobre los argumentos que recibe el método `main` y ver algún ejemplo de cómo capturar su excepción, consulte el apartado 3.

2. Arrays

Un array es un objeto en el que se almacenan datos del mismo tipo en varias ubicaciones contiguas de memoria. Un array puede almacenar cualquier tipo de dato, ya sean datos primitivos u objetos de una clase que creemos.

2.1. Array de objetos

Este es un tipo general de array. Un array de objetos es creado con `new` y sus posiciones son indexadas, desde cero hasta la longitud total menos uno. La creación del array nos devuelve un array de referencias de la longitud deseada. Cada referencia vale inicialmente `null`, y será posteriormente sustituida por una referencia al tipo indicado al crear el array. Cada elemento del array debe ser construido con su constructor para ser insertado en el array.

```
int [] b = new int[10];
```

Se crea un array de 10 enteros, referenciado por `b` y accesibles por `b[0], b[1], ..., b[9]`. También es posible declarar un array y crear el objeto más tarde al asignarle memoria:

```
int [] arr;
...
arr = new int[30];
```

El método `length()` devuelve siempre la longitud el array. Así pues, `arr.length` devuelve `30` y `b.length` devuelve `10`. Veamos un ejemplo:

```
/**
 * Programa que utiliza la clase Asignatura.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class UsaAsignatura2
```

```
{
  static void MuestraAsignaturas (Asignatura [] asignaturas)
  {
    int i;
    System.out.println("Asignaturas");
    for(i = 0; i < asignaturas.length; ++i) {
      System.out.println("Asignatura: " + asignaturas[i].NombreAsignatura());
      System.out.println("Código: " + asignaturas[i].Codigo());
      System.out.println("Departamento: " + asignaturas[i].Departamento());
      System.out.println("Tipo de asignatura: " + asignaturas[i].TipoAsignatura());
      System.out.println("N° de créditos: " + asignaturas[i].CreditosAsignatura());
      System.out.println("Curso: " + asignaturas[i].Curso());
      System.out.println("Cuatrimestre: " + asignaturas[i].Cuatrimestre());
      System.out.println();
    }
  }

  public static void main (String[] args)
  {
    Asignatura [] segundo = new Asignatura[3];

    segundo[0] = new Asignatura(21714019, "Arquitectura de Computadores",
        "Ingeniería de Sistemas y Automática, Tecnología Electrónica", 6,
        "Obligatoria", 2, 1);
    segundo[1] = new Asignatura(21714020, "Programación Concurrente y de Tiempo "
        + "Real", "Lenguajes y Sistemas Informáticos", 6, "Obligatoria", 2, 1);
    segundo[2] = new Asignatura(21714017, "Programación Orientada a Objetos",
        "Lenguajes y Sistemas Informáticos", 6, "Obligatoria", 2, 2);

    MuestraAsignaturas(segundo);
  }
}
```

Ejercicio 6 Realice un programa que almacene la temperatura de los últimos 7 días en un array y luego la muestre por pantalla. Puede comprobar su programa con el que se encuentra en 6.5.

2.2. Array de cadena de caracteres

Los arrays de cadenas de caracteres son un tipo especial. Una cadena es una secuencia finita de caracteres. Java proporciona la clase `String` para manejarlas. Un objeto de la clase `String` referencia a una cadena. Inicializar una cadena de caracteres es muy sencillo:

```
String mensaje = "En un lugar de la Mancha...";
```

La clase `String` tiene muchos métodos de utilidad que sus objetos pueden invocar. Para saber la longitud de la cadena, podemos usar el método `length()`, mientras que para acceder a un caracter en concreto de la cadena podemos usar `charAt()`. También es posible concatenar cadenas usando el operador `+`:

```
int longitud = mensaje.lenth();
char caract = mensaje.charAt(0);
System.out.println("Don" + " " +"Quijote");
```

Ejercicio 7 Añada al ejercicio anterior un array de cadenas de caracteres dónde guarde el día del año al que se refiere cada dato. El programa de salida también debe mostrar el día. Cuando lo termine, compárelo con el código de 6.6.

---

3. Argumentos de la línea de comandos

Mencionamos anteriormente que cada programa en Java tiene un método `main` en alguna de las clases que participan en el programa. El método `main` es un método especial puesto que marca el inicio de la ejecución por parte del intérprete del lenguaje. Un método `main` también recibe argumentos, especificados en la línea de comandos cuando se ejecuta un programa Java desde la consola del sistema. La declaración habitual del método `main` en cualquier clase es de la forma:

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

Si nos fijamos, podemos ver que el método `main` recibe un argumento llamado `args` que es un array de cadenas. Este array es un argumento que se le pasa a `main` desde la línea de comandos. Si se desean pasar argumentos de otros tipos, primero deben pasarse como cadenas dentro de `args`, y luego deben ser convertidos a los tipos deseados.

Si la línea de comandos es `$java prueba arg1 arg2`, entonces el método `main` de la clase `prueba` recibe dos argumentos, situados en `args[0]` y `args[1]`. Para este caso, `args.length` vale 2. Veamos un ejemplo con la clase `Conversor`:

```
/**
 * Programa que utiliza la clase Conversor.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class UsaConversor2
{
  public static void main (String[] args)
  {
    if(args.length != 2) {
      System.err.println("Sintaxis incorrecta.");
      System.err.println("Sintaxis: java UsaConversor2 factor_de_conversion "
        + "cantidad_a_convertir");
      System.exit(-1);
    }
```

```
    double conver = Double.valueOf(args[0]).doubleValue();
    double cant = Double.valueOf(args[1]).doubleValue();
    Conversor eurosDolares;

    eurosDolares = new Conversor(conver);

    System.out.println("Factor de conversión: 1 euro = " +
      eurosDolares.FactorConversion() + " dólares");
    System.out.println(cant + " euros son " + eurosDolares.Convertir(cant)
      + " dólares");
  }
}
```

Ejercicio 8 Implemente una clase que contenga 4 cadenas. Esta clase inicializará 3 de sus atributos de los valores que reciba por la línea de comandos. La cuarta cadena será el resultado de concatenar las 3 anteriores. Muestre por pantalla el resultado de la concatenación. No olvide manejar las excepciones necesarias. Puede ver una versión de este programa en 6.7.

---

4. Flujos de datos

Los programas no sólo necesitan comunicarse con las clases que lo forman. También necesitan comunicarse con los usuarios, con el registro de error o con otros programas en máquinas remotas. Para ello necesitamos los flujos de datos.

Un flujo de datos es un canal de comunicación por el que se transmite una secuencia de datos. Estos datos pueden tener diversas características, por lo que existen distintos tipos de flujos de datos. Aquí veremos algunos de los principales.

Lo habitual al trabajar con flujos de datos es seguir los siguientes pasos:

1. Crear un objeto del flujo de datos deseado.
2. Utilizar el objeto.
3. Cerrar el flujo de datos.

Los flujos de datos definidos en Java se encuentran en el paquete `java.io`.

4.1. Flujo de E/S estándar

Vimos en la práctica anterior que podíamos usar los objetos `System.in` y `System.out` para manejar, respectivamente, la entrada y salida de datos estándar. También mencionamos anteriormente el objeto `System.err`, que se utiliza para mandar mensajes de error a la salida de error estándar. Estos objetos son instancias de clases concretas definidas en la biblioteca de flujo de E/S de Java. A continuación puede ver un programa de ejemplo en el que se usan estos tres objetos:

```
/**
 * Programa en Java que suma dos números obtenidos desde la entrada estándar.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;


public class SumaNumeros
{
  public static void main (String[] args)
  {
    double x, y;

    System.out.println("Suma de dos números");
    try {
      InputStream in = System.in;
      InputStreamReader isr = new InputStreamReader(in);
      BufferedReader br = new BufferedReader (isr);

      System.out.print("Primer sumando: ");
      String dato = br.readLine();
      if (!dato.isEmpty())
        x = Double.parseDouble(dato);
      else
        x = 0.0;

      System.out.print("Segundo sumando: ");
      dato = br.readLine();
      if (!dato.isEmpty())
        y = Double.parseDouble(dato);
      else
        y = 0.0;

      System.out.println(x + " + " + y + " = " + (x+y));

    }
    catch(IOException exc) {
      System.err.println("¡Ups! Ha tenido lugar un error de E/S");
      exc.printStackTrace();
      System.exit(-1);
    }

  }
}
```

Ejercicio 9 Basándose en el ejemplo anterior, realice una versión del programa `Hola_mundo`. El programa debe pedir por teclado el país de residencia usando `System.in` y devolverá un saludo a ese país.

Puede comparar su solución con el programa del apartado 6.8.

4.2. Flujo de E/S de archivos

Para leer y escribir en un archivo, existen en Java distintos tipos de flujos de E/S. Las distintas clases de la biblioteca `java.io` nos permite tratar con flujos de E/S de archivos de entrada, de salida, con bytes, con caracteres, etc.

4.2.1. Flujo de E/S secuencial de archivos

Supongamos que queremos leer datos desde un fichero. Para ello, es necesario conectar los objetos de E/S con los archivos a los que se desea acceder. Podemos usar un objeto de la clase `FileInputStream`, que nos permite leer bytes del archivo. Si quisiéramos escribir bytes en un archivo, usaremos un objeto de la clase `FileOutputStream`. Podemos usar estos objetos cuando queremos trabajar con cualquier tipo de archivo, como una imagen. Veamos cómo enlazaríamos estos objetos con los archivos con los que trabajar:

```
FileInputStream miEntrada = new FileInputStream("datos");
FileOutputStream miSalida = new FileOutputStream("resultados");
```

De este modo hemos establecido que el objeto `miEntrada` lea del archivo `datos` y que el objeto `miSalida` escriba en el archivo `resultados`. Como se puede observar, el constructor recibe como argumento el nombre del archivo. El constructor de `FileOutputStream` comprueba si el nombre de archivo que recibe ya existe. Si no existe, se creará; mientras que si existe, se sobreescribirá.

Una vez creados los objetos del flujo de datos, nos interesa leer o escribir en el archivo. Para ello utilizaremos las funciones:

```
public int read() throws IOException
public void write(int c) throws IOException
```

Estas funciones nos permiten leer y escribir un byte. Si durante la lectura o escritura se produjese un error, estas funciones lanzarían una excepción del tipo `IOException`. Por último, cerraremos los flujos abiertos llamando al método `close()` o, en su defecto, se cerrarán automáticamente al finalizar el programa.

Veamos un ejemplo completo:

```
/**
 * Programa en Java que recibe un archivo y filtra las vocales.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;

public class FiltraVocales {
  static void FiltraV (FileInputStream entrada, FileOutputStream salida) {
    int p;
    char c;
```

```
    try {
      while ((p = entrada.read()) >= 0) {
        c = Character.toLowerCase((char) p);
        if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
          salida.write(p);
      }
      entrada.close();
      salida.close();
    } catch (IOException e) {
      System.err.println("Problemas de E/S");
      System.err.println(e.getMessage());
      System.exit(1);
    }
  }

  public static void main (String[] args) {
    if(args.length != 2) {
      System.err.println("Debe incluir dos nombres de fichero de argumento");
      System.exit(1);
    }
    try {
      FileInputStream entra = new FileInputStream (args[0]);
      FileOutputStream sale = new FileOutputStream (args[1]);
      FiltraV(entra, sale);
    } catch(FileNotFoundException e) {
      System.err.println("Fichero no encontrado...");
      System.err.println(e.getMessage());
      System.exit(1);
    }
  }
}
```

Si desea una lectura/escritura de caracteres puede conseguirlo de manera más cómoda con otras clases de la biblioteca `java.io`. Sin embargo, para implementar la concurrencia usaremos normalmente flujos con bytes. Para más información de flujos de E/S de archivo con caracteres, puede echar un vistazo a las clases `FileReader` y `FileWriter` de la biblioteca `java.io`.

Ejercicio 10 Siguiendo el ejemplo anterior, realice un programa que reciba como argumento el nombre de un archivo y muestre por pantalla el número de palabras que contiene. Puede considerar que una palabra termina cuando detecta un espacio. Puede ver una solución a este ejercicio en 6.9.

4.2.2. Flujo de E/S de archivos de acceso aleatorio

La clase `RandomAccessFile` permite abrir un archivo como de lectura, o de lectura y escritura simultáneamente. Si se utiliza en modo lectura (modo `"r"`), dispone de métodos para leer elementos de cualquier tipo primitivo. Del mismo modo, en modo de lectura y escritura (modo `"rw"`) existen métodos para la escritura de tipos de datos primitivos.

Sin embargo, la mayor particularidad de estos archivos es que podemos acceder a un lugar concreto dentro del archivo y conocer el punto en el que se va a realizar la operación de lectura y/o escritura. Algunos de los métodos más interesantes para el acceso aleatorio son los siguientes:

- `getFilePointer()`: Devuelve la posición en bytes donde se encuentra actualmente el cursor del archivo y se va a realizar la operación de lectura o escritura.
- `length()`: Devuelve el tamaño actual del archivo en bytes.
- `seek()`: Sitúa el cursor para la próxima operación de lectura o escritura en el byte especificado.

```
/**
 * Programa que ilustra cómo trabajar con archivos de acceso aleatorio.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;

public class ArchivoAleatorio {

  public static void main(String arg[]) {
    char c;
    boolean finArchivo = false;
    RandomAccessFile archivo = null;

    try {
      archivo = new RandomAccessFile("prueba.txt", "rw");
      System.out.println("El archivo ha sido abierto en modo de lectura y "
        + "escritura.");
      System.out.println("El tamaño es: " + archivo.length());
      do {
        try {
          c = (char) archivo.readByte();
          archivo.seek(archivo.getFilePointer() - 1);
          archivo.writeByte(Character.toUpperCase(c));
        } catch (EOFException e) {
          finArchivo = true;
          archivo.close();
          System.out.println("El contenido del archivo ahora está en mayúsculas.");
        }
      } while (!finArchivo);
    } catch (FileNotFoundException e) {
      System.out.println("No se encontró el archivo.");
    } catch (IOException e) {
      System.out.println("Problemas con el archivo.");
    }
  }
}
```

Ejercicio 11 Cree un programa que dado un archivo de texto, lo copie en otro pasando a mayúsculas la primera letra de cada palabra. Cuando lo termine, puede comprobar su solución con el programa que se encuentra en 6.10.

4.2.3. Flujo de E/S de objetos del sistema de archivos

La biblioteca de Java también permite obtener la representación de cualquier objeto del sistema de archivos, ya sea un archivo, un directorio o incluso una ruta completa. La clase que usamos para ello es la clase `File`. Esta clase no sirve para leer ni escribir en un archivo, sino que permite trabajar con las propiedades de un archivo o de un directorio, crear directorio u obtener la ruta, por ejemplo. Algunas de las funciones más importantes son:

- `getTotalSpace()`: obtiene el tamaño de un archivo en bytes.
- `getPath()`: obtiene el nombre completo de un archivo, incluida la ruta.
- `getParent()`: obtiene el nombre del padre.
- `renameTo()`: cambia el nombre.
- `delete()`: elimina el nombre.
- `isDirectory()`, `isFile()`: indica si es un directorio o un archivo.
- `list()`: si es un directorio, obtiene la lista de los archivos y directorios que contiene.
- `mkdir()`: crea un directorio.

```
/**
 * Programa que muestra el contenido de un directorio.
 *
 * @author Natalia Partera
 * @version 1.0
 */

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


public class ListarDirectorio {
  public static void main (String arg[]) {
    String directorio;
    if(arg.length > 0)
      directorio = arg[0];
    else
      directorio = ".";

    File actual = new File(directorio);
    System.out.print("El directorio es: ");
    try {
      System.out.println(actual.getCanonicalPath());
```

```
    } catch (IOException e) {
      System.err.println("¡Error! El programa finalizará.");
      System.exit(-1);
    }
    System.out.println("Su contenido es:");

    String[] archivos = actual.list();
    for(int i = 0; i < archivos.length; ++i)
      System.out.println(archivos[i]);
  }
}
```

Ejercicio 12 Cree una clase llamada `OperacionesArchivos` sin atributos, y que tenga un método llamado `copiarArchivo` que reciba dos parámetros: la ruta del archivo de origen y la ruta de destino. Cree también un programa de prueba para comprobar el correcto funcionamiento de la clase `OperacionesArchivos`. Cuando lo termine, puede contrastar su código con el código del apartado 6.11.

4.3. Flujo de E/S con objetos

Para que un objeto pueda ser escrito en un fichero, o sea enviado por un socket de red, debe ser serializado. La **serialización** consiste en la conversión de un objeto a un flujo de bytes. Un flujo de bytes puede ser transmitido sin que se altere la información que contiene.

En Java, todas las clases del API estándar son serializables. Pero para serializar una clase de usuario hay que implementar la clase como *serializable*. Ésto se consigue añadiendo `implements Serializable` al nombre de la clase. Una vez que la clase de usuario ha sido implementada como serializable, sólo queda utilizar las clases y métodos correspondientes de lectura o escritura controlando las excepciones que puedan lanzar.

Pasos para leer o escribir objetos desde un fichero o a un fichero:

1. Convertir la clase en serializable (`implements Serializable`).
2. Utilizar las clases de lectura o escritura de Java y el método correspondiente.
   a) Para leer, utilizar las clases `ObjectInputStream` y `FileInputStream` y el método `readObject(objeto)`.
   b) Para escribir, utilizar las clases `ObjectOutputStream` y `FileOutputStream` y el método `writeObject(objeto)`.
3. Controlar todas las excepciones que sean necesarias.

Veamos un ejemplo de cómo sería una clase serializada y de cómo habría que trabajar con ella.

```
/**
 * Clase que modela escuetamente una asignatura.
 *
 * @author Natalia Partera
 * @version 1.0
```

```
 */

import java.io.*;

public class Asig implements Serializable {
  private int codigo;
  private String nombre;

  public Asig() {}

  public Asig(int cod, String nom) {
    codigo = cod;
    nombre = nom;
  }

  public int Codigo() {
    return codigo;
  }

  public void Codigo(int cod) {
    codigo = cod;
  }

  public String Nombre() {
    return nombre;
  }

  public void Nombre(String nom) {
    nombre = nom;
  }
}
```

El siguiente código muestra cómo guardar los objetos en un fichero y cómo recuperarlos luego.

```
/**
 * Programa que utiliza la clase Asig.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;

public class UsaAsig {

  public static void main (String[] args)
  {
    Asig asig1 = new Asig(21714020, "Programación Concurrente y de Tiempo Real");
    Asig asig2 = new Asig(21714017, "Programación Orientada a Objetos");

    try {
      FileOutputStream ficheroSal = new FileOutputStream("datos.dat");
```

```
      ObjectOutputStream sal = new ObjectOutputStream(ficheroSal);
      sal.writeObject(asig1);
      sal.writeObject(asig2);
      sal.close();
    } catch(FileNotFoundException e) {
      System.err.println("Error de creación de fichero...");
      System.exit(1);
    } catch(IOException e) {
      System.err.println("Error de E/S.");
      System.exit(1);
    }
    System.out.println("Dos asignaturas han sido guardadas correctamente.");
    System.out.println("Ahora las recuperamos.");
    try {
      FileInputStream ficheroEnt = new FileInputStream("datos.dat");
      ObjectInputStream ent = new ObjectInputStream(ficheroEnt);
      Asig asig3 = (Asig) ent.readObject();
      Asig asig4 = (Asig) ent.readObject();
      System.out.println("Las asignaturas recuperadas son:");
      System.out.println(asig3.Codigo() + " - " + asig3.Nombre());
      System.out.println(asig4.Codigo() + " - " + asig4.Nombre());
    } catch(FileNotFoundException e) {
      System.err.println("Error de creación de fichero...");
      System.exit(1);
    } catch(IOException e) {
      System.err.println("Error de E/S.");
      System.exit(1);
    } catch(ClassNotFoundException e) {
      System.err.println("Otros errores de E/S.");
      System.exit(1);
    }
  }
}
```

Ejercicio 13 Modifique la clase `Asignatura` que creó en el ejercicio 3 para que sea serializable. Cree un programa en el que se creen 3 objetos de la clase, se guarden en un fichero, sean rescatados y se muestren sus atributos. Puede comparar su solución, cuando la haya terminado, con la que se encuentra en 6.12.

---

5. Ejercicios

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

Ejercicio 14 Escriba una clase que modele los datos de una persona. Deberá incluir su nombre, apellidos, edad, número de teléfono, dirección de email, ciudad de residencia, nacionalidad y profesión. Proteja los datos almacenados de forma en que sólo se pueda acceder a ellos desde las llamadas a los métodos de su clase. Cree también un programa en el que se introduzcan los datos de, al menos, una persona y los muestre. 6.13.

Ejercicio 15 Escriba un programa que reciba un parámetro entero positivo. En el programa, se ejecutará una función que devolverá un array con tantas posiciones como indique el parámetro recibido. El array será rellenado con potencias de 2, empezando por 2^0. A continuación, otra función recibirá el array y devolverá el sumatorio del array. Puede comparar su programa con la solución del apartado 6.14.

Ejercicio 16 Escriba un método que reciba por parámetro un array de cadenas y devuelva un array de enteros con los tamaños de las cadenas contenidas en el array. Escriba un programa de prueba para comprobar el resultado. Al finalizar puede compararlo con el programa del apartado 6.15.

Ejercicio 17 Escriba un programa que introduzca los 100 primeros números naturales en un archivo dado como argumento. Puede comparar su programa con el que se encuentra en 6.16.

Ejercicio 18 Escriba un método que reciba como parámetro un array de valores enteros, correspondientes a referencias a artículos, y un array de valores reales, correspondientes a los precios de los artículos anteriores. El método recibirá también el nombre de un archivo en el que escribirá cada referencia de artículo seguida de su precio. Realice también un programa de prueba. Cuando lo termine, puede comparar su solución con la ofrecida en 6.17.

Ejercicio 19 Añada a la clase `OperacionesArchivos` un método que reciba como parámetro la ruta del archivo de origen y la del archivo de destino y mueva el archivo a la ruta dada. Modifique el programa de prueba creado en el ejercicio 11 para que también compruebe el funcionamiento de esta función. Cuando finalice, puede comparar su código con el indicado en 6.18.

Ejercicio 20 Añada a la clase `OperacionesArchivos` un método que reciba como parámetro la ruta del directorio de origen y la del directorio de destino y copie el directorio desde el origen al destino. Compruebe su correcto funcionamiento modificando el programa de prueba. Puede comparar su código con el que se encuentra en 6.19.

Ejercicio 21 Modifique la clase `Persona` creada en el ejercicio 14 para que sea serializable. Implemente un programa en el que se inicialicen 10 personas y se guarden en un fichero. Posteriormente, los datos de las personas deben ser rescatados de ese fichero y copiados en otro fichero distinto según se traten de menores de edad, adultos o mayores de 65 años. Compruebe su solución con la ofrecida en el apartado 6.20.

---

6. Soluciones de los ejercicios

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

6.1. Ejercicio 2

Los métodos que hay que sobrecargar son `VerFactor()` y `CambiarFactor()`, ya que devuelven y modifican, respectivamente, el valor del atributo `factorConver`. No tiene sentido sobrecargar el método `Convertir()` con ninguno de los anteriores.

```
/**
 * Clase que realiza conversiones de dólares a euros.
 *
 * @author Natalia Partera
 * @version 2.0
 */

public class Conversor
{
  //Atributo privado: factor de conversión
  private double factorConver;

  //Constructor
  public Conversor(double factor) {
    factorConver = factor;
  }

  //Método observador
  public double FactorConversion() {
    return factorConver;
  }

  //Método modificador
  public void FactorConversion(double nuevoFact) {
    factorConver = nuevoFact;
  }

  //Método observador
  public double Convertir(double cantidad) {
    return cantidad*factorConver;
  }
}
```

6.2. Ejercicio 3

Esta es una posible solución para el ejercicio 3.

```
/**
 * Programa que utiliza la clase Conversor.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class UsaConversor
{
  public static void main (String[] args)
  {
    Scanner teclado = new Scanner(System.in);
    Conversor eurosDolares, dolaresEuros;

    eurosDolares = new Conversor(1.31086);
    dolaresEuros = new Conversor(0.762860);

    System.out.println("Recuerde que para introducir números decimales debe usar "
      + "',' ");
    System.out.print("Introduzca la cantidad que quiere convertir de euros a "
      + "dólares: ");
    double euros = teclado.nextDouble();
    System.out.println("Factor de conversión: 1 euro = " +
      eurosDolares.FactorConversion() + " dólares");
    System.out.println(euros + " euros son " + eurosDolares.Convertir(euros)
      + " dólares");
    System.out.println("Actualice el factor de conversión.");
    System.out.print("Ahora 1 euro vale en dólares: ");
    double nuevoFact = teclado.nextDouble();
    eurosDolares.FactorConversion(nuevoFact);
    System.out.println("Factor de conversión actualizado.");
    System.out.println("Ahora 1 euro = " + eurosDolares.FactorConversion()
      + " dólares");

    System.out.print("Introduzca la cantidad que quiere convertir de dólares a "
      + "euros: ");
    double dolares = teclado.nextDouble();
    System.out.println("Factor de conversión: 1 euro = " +
      dolaresEuros.FactorConversion() + " dólares");
    System.out.println(dolares + " dólares son " + dolaresEuros.Convertir(dolares)
      + " euros");
    System.out.println("Actualice el factor de conversión.");
    System.out.print("Ahora 1 dólar vale en euros: ");
    nuevoFact = teclado.nextDouble();
    dolaresEuros.FactorConversion(nuevoFact);
    System.out.println("Factor de conversión actualizado.");
    System.out.println("Ahora 1 dólar = " + dolaresEuros.FactorConversion()
```

```
      + " euros");
  }
}
```

6.3. Ejercicio 4

Esta es una posible implementación de la clase `Asignatura`:

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

public class Asignatura
{
  //Atributos privados
  private int codigo;
  private String nombreAsig;
  private String departamento;
  private float creditosAsig;
  private String tipoAsig;
  private int curso;
  private int cuatrimestre;

  //Constructor nulo
  public Asignatura() {}

  //Constructor
  public Asignatura(int cod, String nombre, String dpto, float cred, String tipo,
  int curs, int cuatr) {
    codigo = cod;
    nombreAsig = nombre;
    departamento = dpto;
    creditosAsig = cred;
    tipoAsig = tipo;
    curso = curs;
    cuatrimestre = cuatr;
  }

  //Métodos observadores
  public int Codigo() {
    return codigo;
  }

  public String NombreAsignatura() {
    return nombreAsig;
  }

  public String Departamento() {
```

```
    return departamento;
  }

  public float CreditosAsignatura() {
    return creditosAsig;
  }

  public String TipoAsignatura() {
    return tipoAsig;
  }

  public int Curso() {
    return curso;
  }

  public int Cuatrimestre() {
    return cuatrimestre;
  }

  //Métodos modificadores
  public void CambiarDepartamento(String dpto) {
    departamento = dpto;
  }

  public void CambiarCreditos(float cred) {
    creditosAsig = cred;
  }

  public void CambiarTipo(String tipo) {
    tipoAsig = tipo;
  }

  public void CambiarCurso(int curs) {
    curso = curs;
  }

  public void CambiarCuatrimestre(int cuatr) {
    cuatrimestre = cuatr;
  }

}
```

Y este es un programa de prueba para la clase anterior:

```
/**
 * Programa que utiliza la clase Asignatura.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;
```

```
public class UsaAsignatura
{
  public static void main (String[] args)
  {
    Scanner teclado = new Scanner(System.in);
    Asignatura pctr, poo;

    pctr = new Asignatura(21714020, "Programación Concurrente y de Tiempo Real",
        "Lenguajes y Sistemas Informáticos", 6, "Obligatoria", 2, 1);
    poo = new Asignatura(21714017, "Programación Orientada a Objetos", "Lenguajes "
        + "y Sistemas Informáticos", 6, "Obligatoria", 2, 2);

    System.out.println("Asignatura: " + pctr.NombreAsignatura());
    System.out.println("Código: " + pctr.Codigo());
    System.out.println("Departamento: " + pctr.Departamento());
    System.out.println("Tipo de asignatura: " + pctr.TipoAsignatura());
    System.out.println("Nº de créditos: " + pctr.CreditosAsignatura());
    System.out.println("Curso: " + pctr.Curso());
    System.out.println("Cuatrimestre: " + pctr.Cuatrimestre());
    System.out.println();
    System.out.println("Asignatura: " + poo.NombreAsignatura());
    System.out.println("Código: " + poo.Codigo());
    System.out.println("Departamento: " + poo.Departamento());
    System.out.println("Tipo de asignatura: " + poo.TipoAsignatura());
    System.out.println("Nº de créditos: " + poo.CreditosAsignatura());
    System.out.println("Curso: " + poo.Curso());
    System.out.println("Cuatrimestre: " + poo.Cuatrimestre());
    System.out.println();
    System.out.println("Cambiamos la planificación de la titulación.");
    pctr.CambiarCurso(3);
    System.out.println();
    System.out.println("Nueva planificación:");
    System.out.println("Asignatura: " + pctr.NombreAsignatura());
    System.out.println("Código: " + pctr.Codigo());
    System.out.println("Departamento: " + pctr.Departamento());
    System.out.println("Tipo de asignatura: " + pctr.TipoAsignatura());
    System.out.println("Nº de créditos: " + pctr.CreditosAsignatura());
    System.out.println("Curso: " + pctr.Curso());
    System.out.println("Cuatrimestre: " + pctr.Cuatrimestre());
    System.out.println();
    System.out.println("Asignatura: " + poo.NombreAsignatura());
    System.out.println("Código: " + poo.Codigo());
    System.out.println("Departamento: " + poo.Departamento());
    System.out.println("Tipo de asignatura: " + poo.TipoAsignatura());
    System.out.println("Nº de créditos: " + poo.CreditosAsignatura());
    System.out.println("Curso: " + poo.Curso());
    System.out.println("Cuatrimestre: " + poo.Cuatrimestre());
  }
}
```

6.4. Ejercicio 5

Ese código lanza una excepción en la línea `25` y luego cancela la ejecución del programa al intentar parsear un número decimal como número entero. Se puede corregir cambiando el valor inicial de `s2` por un número entero. Una vez corregido este error, lanzará otra excepción en la línea `29` al intentar parsear una cadena que no representa un número al contener letras. Se puede corregir eliminando la letra del valor inicial de `s3`. A continuación, el código corregido que no lanza excepciones.

```
/**
 * Programa en Java que transforma cadenas que almacenen números en números enteros.
 *
 * @author Natalia Partera
 * @version 1.0
 */

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

public class CadenasEnteros
{
  public static void main (String[] args)
  {
    String s1 = "2012";
    String s2 = "67";
    String s3 = "213";

    try {
      System.out.print(s1 + " es ");
      int ent1 = new Integer(s1);
      System.out.println(ent1);

      System.out.print(s2 + " es ");
      int ent2 = Integer.parseInt(s2);
      System.out.println(ent2);

      System.out.print(s3 + " es ");
      int ent3 = Integer.valueOf(s3);
      System.out.println(ent3);
    } catch(NumberFormatException e) {
      System.err.println("Error: la cadena no contiene un entero parseable.");
      e.printStackTrace();
      System.exit(-1);
    }

  }
}
```

6.5. Ejercicio 6

Este es un programa que almacena la temperatura de los últimos 7 días y la muestra:

```
/**
 * Programa que guarda las temperaturas máximas de los últimos 7 días.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class TempMax
{
  static void MuestraTemperaturas (double [] temperaturas)
  {
    int i;
    System.out.println("Temperaturas");
    for(i = 0; i < temperaturas.length; ++i) {
      System.out.println("Dia " + (i+1) + ": " + temperaturas[i]
        + " grados de máxima.");
    }
  }

  public static void main (String[] args)
  {
    double [] tMaximas = new double[7];

    tMaximas[0] = 22.5;
    tMaximas[1] = 21.0;
    tMaximas[2] = 19.5;
    tMaximas[3] = 20.0;
    tMaximas[4] = 22.0;
    tMaximas[5] = 21.5;
    tMaximas[6] = 21.0;

    MuestraTemperaturas(tMaximas);
  }
}
```

6.6. Ejercicio 7

Posible solución para el ejercicio 7.

```
/**
 * Programa que guarda las temperaturas máximas de los últimos 7 días.
 *
 * @author Natalia Partera
 * @version 2.0
 */
```

```
import java.util.*;

public class TempMax
{
  static void MuestraTemperaturas (double [] temperaturas, String [] dias)
  {
    int i;
    System.out.println("Temperaturas");
    for(i = 0; i < temperaturas.length; ++i) {
      System.out.println("El " + dias[i] + " hizo " + temperaturas[i]
        + " grados de máxima.");
    }
  }

  public static void main (String[] args)
  {
    double [] tMaximas = new double[7];
    String [] dias = new String[7];

    tMaximas[0] = 22.5;
    tMaximas[1] = 21.0;
    tMaximas[2] = 19.5;
    tMaximas[3] = 20.0;
    tMaximas[4] = 22.0;
    tMaximas[5] = 21.5;
    tMaximas[6] = 21.0;

    dias[0] = "martes 6 de marzo de 2012";
    dias[1] = "miércoles 7 de marzo de 2012";
    dias[2] = "jueves 8 de marzo de 2012";
    dias[3] = "viernes 9 de marzo de 2012";
    dias[4] = "sábado 10 de marzo de 2012";
    dias[5] = "domingo 11 de marzo de 2012";
    dias[6] = "lunes 12 de marzo de 2012";

    MuestraTemperaturas(tMaximas, dias);
  }
}
```

6.7. Ejercicio 8

Posible solución del ejercicio 8.

```
/**
 * Programa que recibe 3 cadenas como argumentos y las concatena.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;
```

```
public class ConcatenaTresCadenas
{
  private String cad1, cad2, cad3, cadFinal;

  public ConcatenaTresCadenas() {
    cad1 = new String();
    cad2 = new String();
    cad3 = new String();
    cadFinal = new String();
  }

  public void Cadena1(String aux) {
    cad1 = aux;
  }

  public void Cadena2(String aux) {
    cad2 = aux;
  }

  public void Cadena3(String aux) {
    cad3 = aux;
  }

  public void CadenaFinal(String aux) {
    cadFinal = aux;
  }

  public String Cadena1() {
    return cad1;
  }

  public String Cadena2() {
    return cad2;
  }

  public String Cadena3() {
    return cad3;
  }

  public String CadenaFinal() {
    return cadFinal;
  }

  public static void main (String[] args)
  {
    ConcatenaTresCadenas conc = new ConcatenaTresCadenas();

    if(args.length != 3) {
      System.err.println("Sintaxis incorrecta.");
      System.err.println("Sintaxis: java ConcatenaTresCadenas cadena1 cadena2 " +
        "cadena3");
```

```
      System.exit(-1);
    }

    conc.Cadena1(args[0]);
    conc.Cadena2(args[1]);
    conc.Cadena3(args[2]);

    conc.CadenaFinal(conc.Cadena1() + conc.Cadena2() + conc.Cadena3());

    System.out.println("Las cadenas " + conc.Cadena1() + ", " + conc.Cadena2() +
      " y " + conc.Cadena3() + " forman la cadena:");
    System.out.println(conc.CadenaFinal());
  }
}
```

6.8. Ejercicio 9

Versión del programa `Hola_pais`.

```
/**
 * Programa en Java que saluda por la pantalla.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;


public class Hola_pais
{
  public static void main (String[] args)
  {
    System.out.print("¿En qué país se encuentra? ");
    try {
      InputStreamReader isr = new InputStreamReader(System.in);
      BufferedReader br = new BufferedReader (isr);
      String pais = br.readLine();

      System.out.println("Hola " + pais);
    } catch(IOException exc) {
      System.err.println("¡Ups! Ha tenido lugar un error de E/S");
      exc.printStackTrace();
      System.exit(-1);
    }

  }
}
```

6.9. Ejercicio 10

A continuación, una posible solución para el ejercicio 10.

```
/**
 * Programa en Java que recibe un archivo y cuenta las palabras que contiene.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;

public class CuentaPalabras {

  static int NumPalabras(FileInputStream entrada) {
    int cont = 0, p;
    char c;

    try {
      while ((p = entrada.read()) >= 0) {
        c = Character.toLowerCase((char) p);
        if (c == ' ')
          ++cont;
      }
      entrada.close();
    } catch (IOException e) {
      System.err.println("Problemas de E/S");
      System.err.println(e.getMessage());
      System.exit(1);
    }
    return cont;
  }

  public static void main (String args[]) {
    int palabras;

    if (args.length != 1) {
      System.err.println("Debe incluir el nombre del fichero de lectura en el "
        + "argumento.");
      System.exit(1);
    }
    try {
      FileInputStream origen = new FileInputStream(args[0]);
      palabras = NumPalabras(origen);
      System.out.println("El archivo consta de " + palabras + " palabras.");
    } catch (FileNotFoundException e) {
      System.err.println("Fichero no encontrado...");
      System.err.println(e.getMessage());
      System.exit(1);
    }
  }
}
```

6.10. Ejercicio 11

El siguiente código copia el contenido de un archivo a otro poniendo en mayúsculas la primera letra de cada palabra:

```
/**
 * Programa que pone en mayúsculas la primera letra de cada palabra de un archivo
 * dado.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;

public class CapitalizarInicio {

  public static void main(String arg[]) {
    char c, ant = '0';
    boolean finArchivo = false;
    long pointer;
    RandomAccessFile archivoLect = null;
    RandomAccessFile archivoEscr = null;

    try {
      archivoLect = new RandomAccessFile("origen.txt", "r");
      archivoEscr = new RandomAccessFile("texto.txt", "rw");
      System.out.println("Los archivos han sido abiertos con éxito.");
      c = (char) archivoLect.readByte();
      do {
        try {
          if(Character.isLetter(c) && ((archivoLect.getFilePointer() - 1) == 0 ||
          ant == ' '))
            archivoEscr.writeByte(Character.toUpperCase(c));
          else
            archivoEscr.writeByte(c);
          ant = c;
          c = (char) archivoLect.readByte();
        } catch (EOFException e) {
          finArchivo = true;
          archivoLect.close();
          archivoEscr.close();
          System.out.println("Fin de procesamiento de los archivos.");
        }
      } while (!finArchivo);
    } catch (FileNotFoundException e) {
      System.out.println("No se encontró el archivo.");
    } catch (IOException e) {
      System.out.println("Problemas con el archivo.");
    }
  }
}
```

6.11. Ejercicio 12

Este es el código de la clase `OperacionesArchivos`:

```
/**
 * Clase que inmplementa operaciones de copia sobre archivos y directorios.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;

public class OperacionesArchivos {
  public OperacionesArchivos() {}

  public void copiarArchivo (String rutaOrigen, String rutaCopia) {
    if(rutaOrigen == null || rutaCopia == null) {
      System.err.println ("Argumentos no válidos.");
      System.exit(1);
    }
    File origen = new File(rutaOrigen);
    File copia = new File(rutaCopia);
    FileInputStream ent;
    FileOutputStream sal;
    if(copia.isDirectory()) {
      copia = new File(copia.getAbsolutePath() + "/" + origen.getName());
    }
    try {
      //System.out.println("Origen: " + origen.getAbsolutePath());
      ent = new FileInputStream(origen.getAbsolutePath());
      //System.out.println("Destino: " + copia.getAbsolutePath());
      sal = new FileOutputStream(copia.getAbsolutePath());
      int leido;
      while((leido = ent.read()) != -1) {
        sal.write(leido);
      }
      sal.close();
      ent.close();
    } catch (FileNotFoundException e) {
      System.err.println("Error al copiar: No se pudo abrir algún archivo.");
      System.exit(1);
    } catch (IOException e) {
      System.err.println("Error al copiar: Error de lectura/escritura.");
    }
  }
}
```

Y este es un posible programa de prueba:

```
/**
 * Programa que recibe dos rutas (origen y destino) y realiza copias entre ellas.
 *
```

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

public class UsaOperacionesArchivos {
  public static void main (String args[]) {
    if(args.length != 2) {
      System.err.println("Error de sintaxis: $java UsaOperacionesArchivos "
        + "ruta_origen ruta_destino");
      System.exit(1);
    }
    OperacionesArchivos operarch = new OperacionesArchivos();
    operarch.copiarArchivo(args[0], args[1]);
    System.out.println("Archivo copiado con éxito.");
  }
}
```

6.12. Ejercicio 13

Esta es la versión serializable de la clase `Asignatura`.

```
/**
 * Clase que modela una asignatura serializable.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;

public class AsignaSerial implements Serializable
{
  //Atributos privados
  private int codigo;
  private String nombreAsig;
  private String departamento;
  private float creditosAsig;
  private String tipoAsig;
  private int curso;
  private int cuatrimestre;

  //Constructor nulo
  public AsignaSerial() {}

  //Constructor
  public AsignaSerial(int cod, String nombre, String dpto, float cred, String tipo,
  int curs, int cuatr) {
    codigo = cod;
    nombreAsig = nombre;
    departamento = dpto;
    creditosAsig = cred;
```

```
  tipoAsig = tipo;
  curso = curs;
  cuatrimestre = cuatr;
}

//Métodos observadores
public int Codigo() {
  return codigo;
}

public String NombreAsignatura() {
  return nombreAsig;
}

public String Departamento() {
  return departamento;
}

public float CreditosAsignatura() {
  return creditosAsig;
}

public String TipoAsignatura() {
  return tipoAsig;
}

public int Curso() {
  return curso;
}

public int Cuatrimestre() {
  return cuatrimestre;
}

//Métodos modificadores
public void CambiarDepartamento(String dpto) {
  departamento = dpto;
}

public void CambiarCreditos(float cred) {
  creditosAsig = cred;
}

public void CambiarTipo(String tipo) {
  tipoAsig = tipo;
}

public void CambiarCurso(int curs) {
  curso = curs;
}

public void CambiarCuatrimestre(int cuatr) {
```

```
    cuatrimestre = cuatr;
  }
}
```

Y el programa de prueba es el que sigue:

```
/**
 * Programa que utiliza la clase AsignaSerial.
 *
 * @author Natalia Partera
 * @version 1.0
 */

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

public class UsaAsignaSerial
{
  static void MuestraAsignaturas (AsignaSerial [] asignaturas)
  {
    int i;
    System.out.println("Asignaturas");
    for(i = 0; i < asignaturas.length; ++i) {
      System.out.println("Asignatura: " + asignaturas[i].NombreAsignatura());
      System.out.println("Código: " + asignaturas[i].Codigo());
      System.out.println("Departamento: " + asignaturas[i].Departamento());
      System.out.println("Tipo de asignatura: " + asignaturas[i].TipoAsignatura());
      System.out.println("N° de créditos: " + asignaturas[i].CreditosAsignatura());
      System.out.println("Curso: " + asignaturas[i].Curso());
      System.out.println("Cuatrimestre: " + asignaturas[i].Cuatrimestre());
      System.out.println();
    }
  }

  public static void main (String[] args)
  {
    AsignaSerial [] segundo = new AsignaSerial[3];

    segundo[0] = new AsignaSerial(21714019, "Arquitectura de Computadores",
        "Ingeniería de Sistemas y Automática, Tecnología Electrónica", 6,
        "Obligatoria", 2, 1);
    segundo[1] = new AsignaSerial(21714020, "Programación Concurrente y de Tiempo "
        + "Real", "Lenguajes y Sistemas Informáticos", 6, "Obligatoria", 2, 1);
    segundo[2] = new AsignaSerial(21714017, "Programación Orientada a Objetos",
        "Lenguajes y Sistemas Informáticos", 6, "Obligatoria", 2, 2);

    try {
      FileOutputStream ficheroSal = new FileOutputStream("datos.dat");
      ObjectOutputStream sal = new ObjectOutputStream(ficheroSal);
      for(int i = 0; i < 3; ++i)
        sal.writeObject(segundo[i]);
      sal.close();
```

```
    } catch(FileNotFoundException e) {
      System.err.println("Error de creación de fichero...");
      System.exit(1);
    } catch(IOException e) {
      System.err.println("Error de E/S.");
      System.exit(1);
    }
    System.out.println("Las asignaturas han sido guardadas correctamente.");
    System.out.println("Ahora las recuperamos.");
    try {
      FileInputStream ficheroEnt = new FileInputStream("datos.dat");
      ObjectInputStream ent = new ObjectInputStream(ficheroEnt);
      AsignaSerial [] recuperadas = new AsignaSerial[3];
      for(int i = 0; i < 3; ++i)
        recuperadas[i] = (AsignaSerial) ent.readObject();
      System.out.println("Las asignaturas recuperadas son:");
      MuestraAsignaturas(recuperadas);
    } catch(FileNotFoundException e) {
      System.err.println("Error de creación de fichero...");
      System.exit(1);
    } catch(IOException e) {
      System.err.println("Error de E/S.");
      System.exit(1);
    } catch(ClassNotFoundException e) {
      System.err.println("Otros errores de E/S.");
      System.exit(1);
    }
  }
}
```

6.13. Ejercicio 14

Este es el código de la clase `Persona`.

```
/**
 * Clase que guarda información sobre los datos de una persona.
 *
 * @author Natalia Partera
 * @version 1.0
 */


public class Persona {
  private String nombre, apellidos, email, ciudad, nacionalidad, profesion;
  private int edad, tlfn;

  public Persona() {}

  public Persona(String nomb, String ape, int ed, int tel, String mail, String ciud,
  String nacio, String prof) {
    nombre = nomb;
```

```
  apellidos = ape;
  edad = ed;
  tlfn = tel;
  email = mail;
  ciudad = ciud;
  nacionalidad = nacio;
  profesion = prof;
}

//Métodos observadores
public String Nombre() {
  return nombre;
}
public String Apellidos() {
  return apellidos;
}
public int Edad() {
  return edad;
}
public int Telefono() {
  return tlfn;
}
public String Email() {
  return email;
}
public String Ciudad() {
  return ciudad;
}
public String Nacionalidad() {
  return nacionalidad;
}
public String Profesion() {
  return profesion;
}
//Métodos modificadores
public void Nombre(String nomb) {
  nombre = nomb;
}
public void Apellidos(String ape) {
  apellidos = ape;
}
public void Edad(int ed) {
  edad = ed;
}
public void Telefono(int tel) {
  tlfn = tel;
}
public void Email(String mail) {
  email = mail;
}
public void Ciudad(String ciud) {
  ciudad = ciud;
```

```
  }
  public void Nacionalidad(String nacio) {
    nacionalidad = nacio;
  }
  public void Profesion(String prof) {
    profesion = prof;
  }
}
```

Y este es el código del programa que usa la clase anterior:

```
/**
 * Programa que maneja información sobre los datos de una persona.
 *
 * @author Natalia Partera
 * @version 1.0
 */


public class UsaPersona {
  public static void main(String arg[]) {
    Persona p1 = new Persona("Marta", "Sánchez Sánchez", 16, 663777888,
      "marta@gmail.es", "Lugo", "española", "estudiante");

    System.out.println(p1.Nombre() + " " + p1.Apellidos() + " tiene " + p1.Edad() +
      " años.");
    System.out.println("Vive en " + p1.Ciudad() + " y tiene nacionalidad " +
      p1.Nacionalidad() + ".");
    System.out.println("Es " + p1.Profesion() + ". Puede llamarle al " +
      p1.Telefono() + " o mandarle un email a " + p1.Email());

    p1.Edad(32);
    p1.Telefono(632111222);
    p1.Ciudad("Nueva York");
    p1.Nacionalidad(p1.Nacionalidad() + " y estadounidense");
    p1.Profesion("economista");

    System.out.println("Algunos años después...");
    System.out.println(p1.Nombre() + " " + p1.Apellidos() + " tiene " + p1.Edad() +
      " años.");
    System.out.println("Vive en " + p1.Ciudad() + " y tiene nacionalidad " +
      p1.Nacionalidad() + ".");
    System.out.println("Es " + p1.Profesion() + ". Puede llamarle al " +
      p1.Telefono() + " o mandarle un email a " + p1.Email());
  }
}
```

6.14. Ejercicio 15

Una posible solución al ejercicio es la que sigue:

```
/**
 * Programa en Java que almacena potencias de dos en un array y calcula su suma.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class Potencias {
  public int[] ArrayPotencias(int num) {
    int[] pot = new int[num];
    int p = 1;
    for(int i = 0; i < num; ++i) {
      pot[i] = p;
      p = p * 2;
    }
    return pot;
  }

  public int SumaArray(int[] array) {
    int suma = 0;
    for(int i = 0; i < array.length; ++i) {
      suma = suma + array[i];
    }
    return suma;
  }

  public static void main (String[] args) {
    if(args.length != 1) {
      System.err.println("Sintaxis incorrecta.");
      System.err.println("Sintaxis: java Potencias entero");
      System.exit(-1);
    }
    int num = Integer.valueOf(args[0]).intValue();
    if(num < 0) {
      System.err.println("Sintaxis incorrecta.");
      System.err.println("El argumento debe ser mayor o igual que 0.");
      System.exit(-1);
    }
    Potencias pot = new Potencias();

    int[] potencias = pot.ArrayPotencias(num);

    int suma = pot.SumaArray(potencias);
    System.out.println("La suma de las " + num + " primeras potencias de 2 es " +
      suma + ".");
  }
}
```

6.15. Ejercicio 16

A continuación una posible solución para el ejercicio 16:

```
/**
 * Programa en Java que dado un array de cadenas, calcula un array con las
 * longitudes de dichas cadenas.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.util.*;

public class LongCadenas {
  public int[] Longitudes(String[] cadenas) {
    if(cadenas == null) {
      System.err.println("Array de cadenas vacío.");
      System.exit(-1);
    }

    int[] longit = new int[cadenas.length];
    int tam = 0;
    for(int i = 0; i < cadenas.length; ++i) {
      if(cadenas[i] != null)
        longit[i] = cadenas[i].length();
      else
        longit[i] = 0;
    }
    return longit;
  }

  public static void main (String[] args) {
    String[] cadenas = new String[5];
    cadenas[0] = new String("Amarillo");
    cadenas[1] = new String("Verde");
    cadenas[2] = new String("Azul");
    cadenas[3] = new String("Rojo");
    cadenas[4] = new String("Púrpura");

    LongCadenas lc = new LongCadenas();
    int[] longitudes = lc.Longitudes(cadenas);

    System.out.println("Longitudes de las cadenas: ");
    for(int i = 0; i < cadenas.length; ++i) {
      System.out.println(longitudes[i]);
    }
  }
}
```

6.16. Ejercicio 17

A continuación, un ejemplo de un programa que escribe los 100 primeros números naturales en un fichero.

```
/**
 * Programa en Java que escribe en el archivo indicado los 100 primeros números
 * naturales.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;

public class EscribeNumeros {
  public static void main (String[] args) {
    if(args.length != 1) {
      System.err.println("Debe indicar el fichero en el que escribir los números.");
      System.exit(1);
    }
    try {
      FileOutputStream archivoNumeros = new FileOutputStream (args[0]);
      String s;
      for (int i = 0; i < 100; ++i) {
        archivoNumeros.write(i);
        archivoNumeros.write(' ');
      }
      archivoNumeros.close();
    } catch(IOException e) {
      System.err.println("Error de escritura: " + e.toString());
      System.exit(1);
    }
  }
}
```

6.17. Ejercicio 18

Este es el programa que relaciona las referencias y los precios en un archivo. Además, los muestra.

```
/**
 * Programa que relaciona las referencias de los artículos con sus precios,
 * guardándolos en un fichero dado.
 *
 * @author Natalia Partera
 * @version 1.0
 */

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

public class Tienda {
```

```
public void ArticulosPrecios(int[] referencias, double[] precios,
String nombreArchivo) {
  if(referencias.length != precios.length) {
    System.out.println("¡Cuidado! No hay el mismo número de referencias y " +
      "precios.");
  }
  try {
    RandomAccessFile archivo = new RandomAccessFile(nombreArchivo, "rw");
    for(int i = 0; i < referencias.length && i < precios.length; ++i) {
      archivo.writeInt(referencias[i]);
      archivo.writeDouble(precios[i]);
    }
    archivo.close();
  } catch(FileNotFoundException e) {
    System.err.println("Error de creación de fichero...");
    System.exit(1);
  } catch(IOException e) {
    System.err.println("Error de E/S.");
    System.exit(1);
  }
}

public void VerInventario(String nombreArchivo) {
  try {
    boolean finArchivo = false;
    int referencia;
    double precio;
    RandomAccessFile archivo = new RandomAccessFile(nombreArchivo, "r");
    do {
      try {
        referencia = archivo.readInt();
        precio = archivo.readDouble();
        System.out.println("El artículo con referencia " + referencia + " vale " +
          precio + " euros.");
      } catch(EOFException e) {
        finArchivo = true;
        archivo.close();
      }
    } while(!finArchivo);
  } catch(FileNotFoundException e) {
    System.err.println("Error de creación de fichero...");
    System.exit(1);
  } catch(IOException e) {
    System.err.println("Error de E/S.");
    System.exit(1);
  }
}

public static void main(String[] args) {
  Tienda t = new Tienda();
  String archivo = "inventario.txt";
  int[] referencias = {123324, 561298, 112908, 127786, 432285};
```

```
      double[] precios = {14.95, 9.95, 24.95, 6.95, 45.95};
      t.ArticulosPrecios(referencias, precios, archivo);
      t.VerInventario(archivo);
    }
  }
```

6.18. Ejercicio 19

La función añadida a la clase `OperacionesArchivo` es la siguiente:

```
  public void moverArchivo (String rutaOrigen, String rutaDestino) {
    if(rutaOrigen == null || rutaDestino == null) {
      System.err.println("Argumentos no válidos.");
      System.exit(1);
    }
    File origen = new File(rutaOrigen);
    File destino = new File(rutaDestino);
    try {
      copiarArchivo(origen.getAbsolutePath(), destino.getAbsolutePath());
      origen.delete();
    } catch (Exception e) {
      System.err.println("Error al mover.");
      System.exit(1);
    }
  }
```

Y así quedaría el programa de prueba:

```
  /**
   * Programa que recibe dos rutas (origen y destino) y realiza copias entre ellas.
   *
   * @author Natalia Partera
   * @version 1.0
   */

  public class UsaOperacionesArchivos {
    public static void main (String args[]) {
      if(args.length != 2) {
        System.err.println("Error de sintaxis: $java UsaOperacionesArchivos "
          + "ruta_origen ruta_destino");
        System.exit(1);
      }
      OperacionesArchivos operarch = new OperacionesArchivos();
//    operarch.copiarArchivo(args[0], args[1]);
//    System.out.println("Archivo copiado con éxito.");
      operarch.moverArchivo(args[0], args[1]);
      System.out.println("Archivo movido con éxito.");
    }
  }
```

6.19. Ejercicio 20

Este es el código de la función `copiarDirectorio`:

```
public void copiarDirectorio (String rutaOrigen, String rutaDestino) {
  if(rutaOrigen == null || rutaDestino == null) {
    System.err.println("Argumentos no válidos.");
    System.exit(1);
  }
  File origen = new File(rutaOrigen);
  File destino = new File(rutaDestino);
  if(!origen.isDirectory()) {
    System.err.println("No se pudo realizar la copia. El directorio origen no "
      + "existe.");
    System.exit(1);
  }
  if(destino.isDirectory()) {
    System.err.println("No se pudo realizar la copia. El directorio destino ya "
      + "existe.");
    System.exit(1);
  }

  String[] listaOrigen = origen.list();
  destino.mkdirs();
  for(int i = 0; i < listaOrigen.length; ++i) {
    File f = new File(origen.getAbsolutePath() + "/" + listaOrigen[i]);
    if(f.isFile()) {
      copiarArchivo(f.getAbsolutePath(), destino.getAbsolutePath());
    }
    else {
      File d = new File(destino.getAbsolutePath() + "/" + f.getName());
      copiarDirectorio(f.getAbsolutePath(), d.getAbsolutePath());
    }
  }
}
```

El siguiente código corresponde al programa de prueba:

```
/**
 * Programa que recibe dos rutas (origen y destino) y realiza copias entre ellas.
 *
 * @author Natalia Partera
 * @version 1.0
 */

public class UsaOperacionesArchivos {
  public static void main (String args[]) {
    if(args.length != 2) {
      System.err.println("Error de sintaxis: $java UsaOperacionesArchivos "
        + "ruta_origen ruta_destino");
      System.exit(1);
    }
    OperacionesArchivos operarch = new OperacionesArchivos();
```

```
//    operarch.copiarArchivo(args[0], args[1]);
//    System.out.println("Archivo copiado con éxito.");
//    operarch.moverArchivo(args[0], args[1]);
//    System.out.println("Archivo movido con éxito.");
      operarch.copiarDirectorio(args[0], args[1]);
      System.out.println("Directorio movido con éxito.");
    }
  }
```

6.20. Ejercicio 21

Se muestra la clase `Persona` modificada:

```
/**
 * Clase que guarda información sobre los datos de una persona.
 *
 * @author Natalia Partera
 * @version 2.0
 */

import java.io.*;

public class Persona implements Serializable {
  private String nombre, apellidos, email, ciudad, nacionalidad, profesion;
  private int edad, tlfn;

  public Persona() {}

  public Persona(String nomb, String ape, int ed, int tel, String mail, String ciud,
  String nacio, String prof) {
    nombre = nomb;
    apellidos = ape;
    edad = ed;
    tlfn = tel;
    email = mail;
    ciudad = ciud;
    nacionalidad = nacio;
    profesion = prof;
  }

  //Métodos observadores
  public String Nombre() {
    return nombre;
  }
  public String Apellidos() {
    return apellidos;
  }
  public int Edad() {
    return edad;
  }
  public int Telefono() {
```

```
    return tlfn;
  }
  public String Email() {
    return email;
  }
  public String Ciudad() {
    return ciudad;
  }
  public String Nacionalidad() {
    return nacionalidad;
  }
  public String Profesion() {
    return profesion;
  }
  //Métodos modificadores
  public void Nombre(String nomb) {
    nombre = nomb;
  }
  public void Apellidos(String ape) {
    apellidos = ape;
  }
  public void Edad(int ed) {
    edad = ed;
  }
  public void Telefono(int tel) {
    tlfn = tel;
  }
  public void Email(String mail) {
    email = mail;
  }
  public void Ciudad(String ciud) {
    ciudad = ciud;
  }
  public void Nacionalidad(String nacio) {
    nacionalidad = nacio;
  }
  public void Profesion(String prof) {
    profesion = prof;
  }
}
```

Este es el programa que almacena los datos en un fichero y luego los recupera y muestra separándolos por edades:

```
/**
 * Programa que maneja información sobre los datos de personas.
 *
 * @author Natalia Partera
 * @version 1.0
 */

import java.io.*;
```

```
import java.util.*;

public class UsaPersonaSerializable {
  public Persona[] inicializarListaPersonas() {
    Persona[] lista = new Persona[10];
    lista[0] =  new Persona("Marta", "Sánchez Sánchez", 16, 663777888,
      "marta@gmail.es", "Lugo", "española", "estudiante");
    lista[1] = new Persona("Mario", "Pérez López", 22, 699722011,
      "mariopl@gmail.es", "Málaga", "español", "fontanero");
    lista[2] = new Persona("Carmen", "Gómez García", 26, 611222333,
      "carmengg@gmail.es", "Badajoz", "española", "abogada");
    lista[3] = new Persona("Marco", "Polo Sur", 33, 633455644,
      "exploradores.sin.fronteras@gmail.es", "Venecia", "italiano", "explorador");
    lista[4] = new Persona("Inga", "Schmidt", 14, 677887778, "ingaschmidt@gmail.de",
      "Hamburgo", "alemana", "estudiante");
    lista[5] = new Persona("Juan", "Ruíz Méndez", 40, 699876543, "juanito@gmail.es",
      "Barcelona", "español", "profesor");
    lista[6] = new Persona("Ana", "Paez Gómez", 45, 623465432, "anita@gmail.es",
      "Bogotá", "colombiana", "arquitecta");
    lista[7] = new Persona("Sergio", "Rey Núñez", 58, 678123456,
      "el_rey_sergio@msn.es", "Palencia", "español", "albañil");
    lista[8] = new Persona("Aurelia", "Santos Inocente", 67, 612345678,
      "yaya_aurelia@gmail.es", "Guadalajara", "española", "jubilada");
    lista[9] = new Persona("John", "Smith", 78, 699123456, "john_smith@gmail.es",
      "Londres", "británico", "jubilado");
    return lista;
  }


  public void guardarPersonas(Persona[] personas) {
    try {
      FileOutputStream ficheroSal = new FileOutputStream("personas.dat");
      ObjectOutputStream fichPersonas = new ObjectOutputStream(ficheroSal);
      for(int i = 0; i < personas.length; ++i) {
        fichPersonas.writeObject(personas[i]);
      }
      fichPersonas.close();
    } catch(FileNotFoundException e) {
      System.err.println("Error de creación de fichero...");
      System.exit(1);
    } catch(IOException e) {
      System.err.println("Error de E/S.");
      System.exit(1);
    }
    System.out.println("Todas las personas han sido guardadas en un fichero.");
  }


  public int[] separarPersonas() {
    int[] longitudes = new int[3];
    try {
      FileInputStream personasEnt = new FileInputStream("personas.dat");
```

```
    FileOutputStream menoresSal = new FileOutputStream("menores.dat");
    FileOutputStream adultosSal = new FileOutputStream("adultos.dat");
    FileOutputStream mayoresSal = new FileOutputStream("mayores.dat");
    ObjectInputStream personasObj = new ObjectInputStream(personasEnt);
    ObjectOutputStream fichMenores = new ObjectOutputStream(menoresSal);
    ObjectOutputStream fichAdultos = new ObjectOutputStream(adultosSal);
    ObjectOutputStream fichMayores = new ObjectOutputStream(mayoresSal);
    Persona p = new Persona();
    int men = 0, adu = 0, may = 0;
    for(int i = 0; i < 10; ++i) {
      p = (Persona) personasObj.readObject();
      if(p.Edad() < 18) {
        fichMenores.writeObject(p);
        ++men;
      }
      else if (p.Edad() >= 65) {
        fichMayores.writeObject(p);
        ++may;
      }
      else {
        fichAdultos.writeObject(p);
        ++adu;
      }
    }
    longitudes[0] = men;
    longitudes[1] = adu;
    longitudes[2] = may;
    personasObj.close();
    fichMenores.close();
    fichAdultos.close();
    fichMayores.close();
  } catch(FileNotFoundException e) {
    System.err.println("Error de creación de fichero...");
    System.exit(1);
  } catch(IOException e) {
    System.err.println("Error de E/S.");
    System.exit(1);
  } catch(ClassNotFoundException e) {
    System.err.println("Otros errores de E/S.");
    System.exit(1);
  }
  System.out.println("Las personas han sido separadas según su edad.");
  return longitudes;
}


public void muestraPersonas(Persona[] p) {
  for(int i = 0; i < p.length; ++i) {
    System.out.println(p[i].Nombre() + " " + p[i].Apellidos() + " tiene " +
      p[i].Edad() + " años.");
    System.out.println("Vive en " + p[i].Ciudad() + ", es " +
      p[i].Nacionalidad() + " de nacimiento.");
```

```
      System.out.println("Es " + p[i].Profesion());
      System.out.println("Puede llamarle al " + p[i].Telefono() +
        " o mandarle un email a " + p[i].Email());
    }
  }


  public Persona[] muestraFichero(String nomFich, int num) {
    Persona[] personas = new Persona[num];
    try {
      FileInputStream ficheroEnt = new FileInputStream(nomFich);
      ObjectInputStream ent = new ObjectInputStream(ficheroEnt);
      Persona p = new Persona();
      for(int i = 0; i < personas.length; ++i) {
        p = (Persona) ent.readObject();
        personas[i] = p;
      }
      ent.close();
    } catch(FileNotFoundException e) {
      System.err.println("Error de creación de fichero...");
      System.exit(1);
    } catch(IOException e) {
      System.err.println("Error de E/S.");
      System.exit(1);
    } catch(ClassNotFoundException e) {
      System.err.println("Otros errores de E/S.");
      System.exit(1);
    }
    return personas;
  }


  public static void main(String arg[]) {
    UsaPersonaSerializable p = new UsaPersonaSerializable();
    Persona[] personas = p.inicializarListaPersonas();
    p.guardarPersonas(personas);
    int[] longitudes = p.separarPersonas();
    Persona[] menores = p.muestraFichero("menores.dat", longitudes[0]);
    Persona[] adultos = p.muestraFichero("adultos.dat", longitudes[1]);
    Persona[] mayores = p.muestraFichero("mayores.dat", longitudes[2]);
    System.out.println("Los menores de edad son:");
    p.muestraPersonas(menores);
    System.out.println("Los adultos son:");
    p.muestraPersonas(adultos);
    System.out.println("Los mayores de 65 años son:");
    p.muestraPersonas(mayores);
  }
}
```