Centro Asociado Palma de Mallorca

Introducción Práctica de Programación Java

Antonio Rivero Cuesta

Sesión I

Introducción

Java Development Kit

La herramienta básica para empezar a desarrollar aplicaciones en Java es el JDK (*Java Development Kit*).

Consiste en un compilador y un intérprete (JVM) para la línea de comandos.

No dispone de un ***entorno de desarrollo integrado*** (IDE), pero es suficiente para aprender el lenguaje y desarrollar pequeñas aplicaciones.

IDE

- Bloc de notas.
- JCreator.
- NetBeans.
- BlueJ.
- Eclipse.

El **IDE** para este curso será el NetBeans.

Se puede descargar de:

https://netbeans.org/downloads/index.html

Utilizaré también JCreator.

Se puede descargar de:

http://www.jcreator.com/

Bibliografía

Programación orientada a objetos con Java. Una introducción práctica usando BlueJ 6ª ed.

David J. Barnes, Michael Kölling.

Editorial Pearson

Java 2: Iniciación y Referencia (2ª ed.)

ISBN: 9788448198169

Editorial: S.A. MCGRAW-HILL

Clases y Objetos

En ***Programación Orientada a Objetos*** hay que distinguir entre dos conceptos íntimamente ligados.

La ***clase*** y el ***objeto***.

Una ***clase*** es una plantilla donde vamos a definir unos ***atributos*** y unos ***métodos***.

Una clase es la ***implementación de un tipo de objeto***.

Considerando los objetos como ***instancias*** de las clases.

Cuando se crea un objeto, ***se instancia una clase,*** mediante el operador **new.**

```
Alumno alumno1 = new Alumno();
```

Se ha de especificar de qué clase es el objeto instanciado, para que el compilador comprenda las características del objeto.

Cuando se instancia un objeto el compilador crea en la memoria dinámica un espacio para tantas variables como atributos tenga la clase a la que pertenece el objeto.

Desde el punto de vista de la programación estructurada:

- Una ***clase*** se asemejaría a un ***módulo***.
- Los ***atributos*** a las ***variables globales***.
- Los ***métodos*** a las ***funciones*** del módulo.

Estructura General

La parte interna de la clase es el lugar en el que definimos los:

- Campos.
- Constructores.
- Métodos.

```java
package Persona;

import java.util.Scanner;

public class Alumno {

    // Campos
    private String nombre;

    // Constructor de Clase
    public Alumno(String nombre) {
        this.nombre = nombre;
    }

    // Métodos
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return (this.nombre);
    }

} // Fin Clase Alumno
```

Las características principales son:

- Los ***campos*** almacenan datos para que cada objeto los use.
- Los ***constructores*** permiten que cada objeto se prepare adecuadamente cuando es creado.
- Los ***métodos*** implementan el comportamiento de los objetos.

Orden en las Clases

```
public class NombreDeClase{

    Campos
    Constructores
    Métodos

}
```

Campos

Almacenan datos de manera persistente dentro de un objeto y son accesibles para todos los métodos del objeto.

Son conocidos como ***variables de instancia***.

Como los campos pueden almacenar valores que pueden variar a lo largo del tiempo, podemos llamarlos ***variables***.

Todos los objetos una vez creados dispondrán de un espacio para cada campo declarado en su clase.

Se definen fuera de los constructores y de los métodos.

Mantienen el estado actual de un objeto.

Tienen un tiempo de vida que finaliza cuando termina el objeto.

Su accesibilidad se extiende a toda la clase, así que pueden usarse dentro de cualquier constructor o método de clase en la que estén definidos.

Como son definidos privados, **`private`**, no pueden ser accedidos desde el exterior de la clase.

El uso de campos públicos es una práctica que probablemente crea un gran ***acoplamiento*** entre las clases.

Para definir una ***variable de instancia*** dentro de una clase seguiremos el siguiente patrón:

- Normalmente, comienzan con la palabra clave **`private`**.
- Incluyen un nombre de tipo, **`int`**, **`String`**, etc.
- Incluyen un nombre elegido por el usuario.
- Terminan en punto y coma.

El tipo de un campo especifica la naturaleza del valor que puede almacenarse en dicho campo.

Si el tipo es una clase, el campo puede contener objetos de esa clase. Esto se conoce como composición.

```
public class Alumno {
    private String nombre;
    private String apellidos;
    private int telefonoContacto;
    private String direccion;
...
}
```

Hay diferentes tipos de variables que requieren distintas cantidades de memoria para guardar datos.

Todas las variables han de declararse antes de usarlas.

La declaración consiste en una sentencia en la que figura el tipo de dato y el nombre que asignamos a la variable.

Una vez declarada se le podrá asignar valores.

Tipos de Variables Java

- De Instancia. Campos o atributos.
- Locales.
- De Clase.
- Constantes.

Las variables son uno de los elementos básicos de un programa, y se deben:

- Declarar
- Inicializar
- Usar

Para declarar una variable hay que seguir una serie de normas:

1. El primer carácter del nombre debe ser una letra mayúscula o minúscula, “_” o “$”.
2. No pueden utilizarse como nombres de variables las palabras reservadas de JAVA.
3. Los nombres deben ser continuos, es decir, sin espacios en blanco.
4. Los identificadores de variables distinguen las mayúsculas y las minúsculas.

Variables de Instancia

Se usan para guardar los atributos de un objeto particular.

Las ***variables de instancia*** también son conocidas como ***campos***.

Se definen siempre fuera de los métodos y constructores.

Variables Locales

Se utilizan dentro de los métodos.

En el siguiente ejemplo *area* es una variable local al método *calcularArea* en la que se guarda el valor del área de un objeto de la clase *Circulo*.

Una variable local existe desde el momento de su definición hasta el final del bloque en el que se encuentra.

```
public class Circulo{

//...

  public double calcularArea(){
    double area = PI*radio*radio;
    return area;
  }

}
```

Las variables locales se declaran en el momento en el que son necesarias.

Es una buena costumbre inicializar las variables en el momento en el que son declaradas.

Delante del nombre de cada variable se ha de especificar el ***tipo*** de la variable.

Variables de Clase

Son variables cuyos valores son los mismos para la clase y para todas sus instancias.

Se utiliza la palabra clave **`static`** en la declaración de las variables:

```
static tipoVariable nombreVariable;
```

Ocupa un único lugar en memoria.

A las variables de clase se les llama variables estáticas.

Si no se usa **`static`**, el sistema crea un lugar nuevo para esa variable con cada instancia.

Un atributo de clase lo tenemos que declarar obligatoriamente en la cabecera de clase.

Junto a los campos, debajo de la signatura de clase.

Constantes

Utilizamos la palabra reservada **`final`**.

Así indicamos que una variable es de tipo constante.

No admitirá cambios después de su declaración y asignación de valor.

`final` determina que un atributo no puede ser sobreescrito o redefinido.

En el caso de una constante no tiene sentido crear un nuevo lugar de memoria por cada objeto de una clase que se cree.

Cuando usamos **`static final`** se dice que creamos una constante de clase.

Toda constante declarada con **`final`** ha de ser inicializada en el mismo momento de declararla.

`final` también se usa como palabra clave en otro contexto.

Una clase **`final`** es aquella que no puede tener clases que la hereden.

Por estilo y para una mayor claridad en el código usaremos letras mayúsculas en los identificadores.

```
private static final double PI=3.1416;
```

`final` puede ser usado dentro de métodos y también dentro de un método **`main`**.

```
public class Constantes{

   public static final float PI = 3.141592f;
   public static final float E = 2.728281f;

   public static void main(String[] args){
     System.out.println("PI = " + Constantes.PI);
     System.out.println("E = " + Constantes.E);
   }
}
```

Constructores

Permiten que cada objeto sea preparado adecuadamente cuando es creado.

Esta operación se denomina ***inicialización***.

El constructor inicializa el objeto en un estado razonable.

Tienen el mismo nombre que la clase en la que son definidos.

El nombre del constructor sigue inmediatamente a la palabra **`public`**.

Los campos del objeto se inicializan en el constructor, bien con valores fijos, o bien con parámetros del propio constructor.

En Java todos los campos son inicializados automáticamente con un valor por defecto, si es que no están inicializados explícitamente.

El valor por defecto para los campos enteros es 0.

Sin embargo, es preferible escribir explícitamente las asignaciones.

No hay ninguna desventaja en hacer esto y sirve para documentar lo que está ocurriendo realmente.

```
public class Profesor {
    private String nombre;
    private String apellidos;
    private int dni;

public Profesor( String nombre,
                 String apellidos,
                 int dni){
    this.nombre = nombre;
    this.apellidos = apellidos;
    this.dni = dni;
}
```

La forma de llamar al constructor es con el operador **new**.

```
Profesor Miguel = new Profesor("Miguel","García",1234);
```

Parámetros

La manera en que los constructores y los métodos reciben valores es mediante sus ***Parámetros***.

Son otro tipo de variable, igual que los campos y permiten almacenar valores.

Se utilizan para almacenar datos.

Se definen en el encabezado de un constructor o un método.

Transportan datos que tienen su origen fuera del constructor o método.

Hacen que esos datos estén disponibles en el interior del constructor o método.

Métodos

Se componen de dos partes:

- Una cabecera.
- Un cuerpo.

Es importante distinguir entre la ***cabecera*** del método y ***declaración de campos*** porque son muy parecidos.

Podemos decir que algo es un método y no un campo porque está seguido de un par de paréntesis: ().

La signatura de un método incluye, en este orden:

- Un modificador de acceso, **`public`**, **`private`**.
- El tipo de retorno del método.
- El nombre del método.
- Una lista de parámetros.

La interfaz de un método proporciona todos los elementos necesarios para saber cómo usarlo.

```java
/**
* Método que imprime la información sobre un Alumno.
*
*/

public void mostrarInfoAlumno(){
   System.out.println("Nombre: " + getNombre());
   System.out.println("Apellidos: " + getApellidos());
   System.out.println("DNI:" +getDni()
   System.out.println();
}
```

No hay un punto y coma al final de la signatura.

El cuerpo del método es la parte restante del método, que aparece a continuación de la cabecera.

Está siempre encerrado entre llaves: {…}.

Los cuerpos de los métodos contienen las ***declaraciones*** y las ***instrucciones*** que definen qué ocurre dentro de un objeto cuando es invocado ese método.

Las declaraciones se utilizan para crear espacio adicional de ***variables temporales***.

Las instrucciones describen las acciones del método.

Cualquier conjunto de declaraciones y sentencias, ubicado entre un par de llaves, es conocido como un ***bloque***.

Por lo que el cuerpo de una clase y los cuerpos de todos los métodos de las clases son bloques.

Existen dos diferencias significativas entre las cabeceras de los constructores de una clase y de los demás métodos:

Por un lado los ***constructores*** tienen el mismo nombre que la clase en la que están definidos.

Los métodos siempre tienen un ***tipo de retorno***, aunque sea **`void`.**

Es el responsable de devolver un valor que coincida con el tipo de retorno de la signatura del método.

El constructor no tiene tipo de retorno.

El tipo de retorno se escribe exactamente antes del nombre del método.

Por otro lado tanto los constructores como los métodos pueden tener cualquier número de parámetros formales, inclusive pueden no tener ninguno.

Cuando un método contiene una sentencia **`return`**, siempre es la última sentencia que se ejecuta del mismo.

Una vez que se ejecutó esta sentencia no se ejecutarán más sentencias en el método.

Los tipos de retorno y las instrucciones de retorno funcionan conjuntamente.

Podemos decir que una llamada a un método es una especie de pregunta que se la hace el objeto.

El valor de retorno proporcionado por el método es la respuesta que el objeto da a esa pregunta.

Métodos `get`

Los **métodos** **`get`** devuelven información sobre el estado del objeto.

Contienen una sentencia **`return`** para devolver información de un valor en particular.

```java
/**
* Devuelve el nombre del alumno.
* @return nombre del alumno.
*/
public String getNombre(){
   return nombre;
}
```

Métodos `set`

Son métodos que modifican el estado de su objeto.

La forma básica admite un único parámetro.

Este valor se utiliza para sobreescribir directamente lo que haya almacenado en uno de los campos del objeto.

La cabecera tiene:

- Un tipo de retorno **`void`.**
- Un solo parámetro formal, el nuevo valor del campo a modificar.

Un tipo de retorno **`void`** significa que el método no devuelve ningún valor cuando es llamado.

No hay ninguna sentencia **`return`**.

Siempre tienen al menos una sentencia de asignación.

```java
/**
* Modifica el nombre de un alumno.
* @param nombre.
*/
public void setNombre(String nombre){
   this.nombre = nombre;
}
```

El método `main`

Las clases son las únicas cosas que tenemos inicialmente.

El primer método que será invocado debe ser un ***método de clase***.

El usuario especifica la clase que será iniciada y el sistema Java luego invocará un método denominado **`main`** ubicado dentro de dicha clase.

Este método debe tener una signatura específica.

Si no existe tal método en esa clase se informa un error.

Hola Mundo

```java
/**
* @author Antonio Rivero
* @version 1.00 2014/4/9
*/
public class HolaMundo {

   public static void main(String[] args){

      System.out.println("Hola mundo");
   }
}
```

Sobrecarga

La única limitación en la elección del nombre de un método o constructor es que, en una clase, todos los métodos o constructores deben tener diferente signatura, es decir, distinto nombre y parámetros.

Esto permite que existan varios métodos o constructores con el mismo nombre pero con diferentes parámetros.

```
public class Arboles{
    public Arboles(){
    }
    public Arboles(String tipo){
    }
    public Arboles(int altura){
    }
    public Arboles(int altura,String tipo){
    }
    public Arboles(int altura,String tipo,int lado){
    }
}
```

Dependiendo de los valores de los argumentos con que se llame al constructor **`Arboles`**, se ejecutaría uno u otro de los definidos.

Especificadores de Acceso

Son opcionales y determinan el tipo de acceso al método.

Un especificador de acceso o alcance es una palabra reservada que nos permite controlar nivel de alcance que tiene cada variable, método o clase.

Tenemos:

- **`public`.**
- **`private`.**
- **`protected`.**
- **`default`.**

Especificadores de Acceso

Visibilidad	public	protected	default	private
UML	+	#	~	–
Misma Clase	SI	SI	SI	SI
Cualquier clase del mismo paquete	SI	SI	SI	NO
Subclase mismo Paquete	SI	SI	SI	NO
Subclase diferente Paquete	SI	SI Herencia	NO	NO
Cualquier clase fuera del paquete	SI	NO	NO	NO

public

Establece un nivel de acceso total.

Una clase **`public`** puede ser usada por otras clases del mismo paquete y de otros paquetes no relacionados.

De igual forma para los métodos y variables.

private

Establece un nivel restringido de acceso.

Los miembros declarados como tal solo son accesibles dentro de la clase.

Establecer miembros **`private`** establece el mayor nivel de seguridad.

protected

Establece un nivel de acceso intermedio entre **`public`** y **`private`**.

La declaración de un campo o un método como **`protected`** permite su acceso directo desde las subclases.

Lo utilizaremos cuando usemos Herencia.

Tipos Enumerados

Un enumerado o `enum` es una clase “especial” que limita la creación de objetos a los especificados explícitamente en la implementación de la clase.

La única limitación que tienen los enumerados respecto a una clase normal es que si tiene constructor, este debe de ser privado para que no se puedan crear nuevos objetos.

Imprimir desde Métodos

La sentencia:

```
System.out.println("Hola");
```

Imprime **Hola** en la terminal de texto.

Una sentencia como:

```
System.out.println("Hola");
```

Imprime literalmente la cadena que aparece entre el par de comillas dobles.

Todas estas sentencias de impresión son invocaciones al método **`println`** del objeto **`System.out`** que está construido dentro del lenguaje Java.

Cuando se usa el símbolo «+» entre una cadena y cualquier otra cosa, este símbolo es un operador de concatenación de cadenas.

El método **`println`** se puede llamar sin contener ningún parámetro de tipo cadena.

Esto está permitido y el resultado de la llamada será dejar una línea en blanco entre esta salida y cualquier otra que le siga.

printf

Imprime un mensaje por pantalla utilizando una cadena de formato que incluye las instrucciones para mezclar múltiples cadenas en la cadena final a mostrar por pantalla.

Es una función especial porque recibe un número variable de parámetros.

El primer parámetro es fijo y es la cadena de formato.

En ella se incluye texto a imprimir literalmente y *marcas* a reemplazar por texto que se obtiene de los parámetros adicionales.

Se llama con tantos parámetros como marcas haya en la cadena de formato más uno, la propia cadena de formato.

El siguiente ejemplo muestra cómo se imprime el valor de la variable **`contador`**.

```
printf("El valor es %d.\n", contador);
```

El símbolo **`%`** denota el comienzo de la marca de formato.

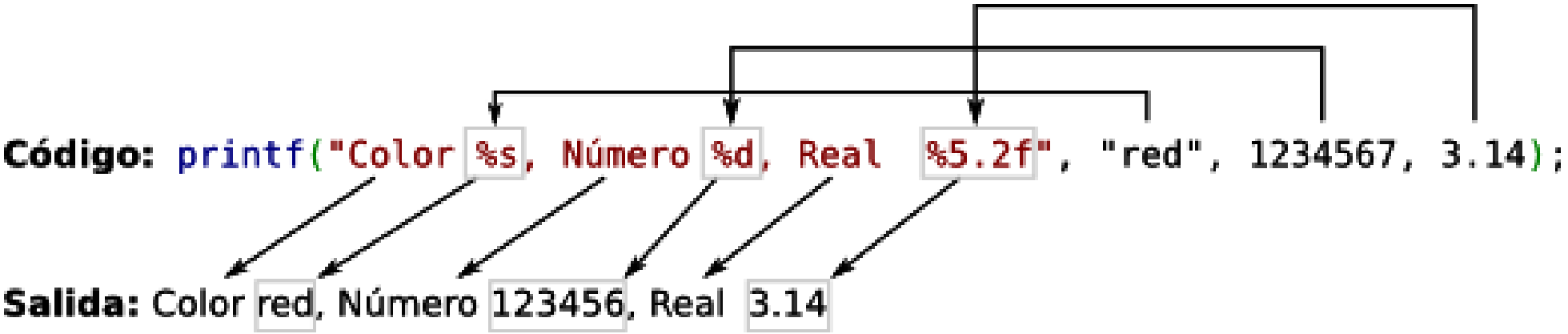
La marca **`%d`** se reemplaza por el valor de la variable **`contador`** y se imprime la cadena resultante.

El símbolo **\n** representa un salto de línea.

La salida, por defecto, se justifica a la derecha del ancho total que le hayamos dado al campo, que por defecto tiene como longitud la longitud de la cadena.

Si en la cadena de formato aparecen varias marcas, los valores a incluir se toman en el mismo orden en el que aparecen.

La siguiente figura muestra un ejemplo en el que la cadena de formato tiene tres marcas, **`%s`, `%d`** y **`%5.2f`**, que se procesan utilizando respectivamente la cadena “red”, el entero **`1234567`** y el número real **`3.14`**.



Las marcas en la cadena de formato deben tener la siguiente estructura.

Los campos entre corchetes son optativos.

```
%[parameter][flags][width][.precision][length]type
```

Toda marca comienza por el símbolo **%** y termina con su tipo.

Cada uno de los nombres:

parameter

flags

width

precision

length

type

Representa un conjunto de valores posibles.