Unidad 6

Clases y utilización de objetos

Programación

6

Índice

Introducción

En el mundo real el ser humano tiende a identificar y nombra todas las entidades que existen en el mundo para diferenciarlos del resto, clasificarlos, estructurarlo, etc. En la programación orientada a objetos saber trasladar las entidades del mundo real a su representación en un programa informático es la parte más importante. Hasta ahora se había visto que existían tipos primitivos de datos (int, char, float) ahora veremos un tipo de datos referencia denominado clase.

Al finalizar esta unidad

- Conoceremos que es una clase y su estructura
- Comprenderemos la visibilidad y paso de datos en los elementos de una clase.
- Aprenderemos a la su instanciación y destrucción
- Conoceremos que es una librería y aquellas más usadas.

6.1. Concepto de clase

Una clase es una representación abstracta de una entidad del mundo real en un programa informático que contiene unas propiedades y unas acciones.

Diferentes ejemplos:

- **Una persona.** Sus propiedades pueden ser, nombre, dni, edad, etc y sus acciones, saludar, despedirse, andar, comer...
- **Un vehículo.** Sus propiedades pueden ser, marca, modelo, etc y sus acciones, aparcar, mover, repostar....

Un objeto es una instancia creada en base a una clase. Es decir, una variable de esa clase contendrá las mismas propiedades y acciones definidas en la clase. Cada objeto declarado de una misma clase será independiente de unos de otros, compartirán la misma estructura, pero tendrán diferentes valores.

```
public class Persona {

}
```

Imagen 1. Ejemplo de clase

6.2. Estructura y miembros de una clase

Una clase se puede declarar indicando el nivel de visibilidad, su nombre, el nombre de la clase de la que hereda, si no se indica nada por defecto se hereda de la superclase Object y las interfaces que pueda implementar.

Sintaxis para declarar una clase:

```
[visibilidad] class Nombre [extends Superclase] [implements Interfacel,Interface2,...] {}
```

Los elementos extends declara la herencia e implementa de-clara la interfaz de esa clase, son opcionales. En los siguientes temas se explicará la herencia e interfaces.

Las clases se componen de dos partes:

- Campos, anteriormente propiedades.
- Métodos, anteriormente acciones.

```
public class Persona {

    //campos

    //métodos

}
```

Imagen 2. Ejemplo de clase y estructura

6.2.1. Campos

También denominados atributos, variables o fields. Permiten guarda la información que debe contener la clase. Por cada objeto instanciado de una clase contendrá valores diferentes en sus campos, ahora bien, si queremos que todos los objetos compartan un mismo valor los declararemos como estático, a este tipo de campos se le denomina, campos estáticos, campos de clase o variables de clase. Por ejemplo, una persona no es igual a otra, se puede diferenciar por el DNI, edad, altura, etc. pero todas las personas compartimos características iguales, todos tenemos 2 piernas o 2 brazos.

Se representa dentro del cuerpo de la clase a partir de la siguiente sintaxis:

```
[Visibilidad] [static] tipo campo1;
```

Los campos se declaran indicando un tipo de visibilidad, que normalmente, siempre va a ser privada, para así cumplir con los principios de encapsulación y ocultación de la información. Hay algunos casos en los que los campos podrán ser públicos. Opcionalmente se puede incluir la palabra reservada static para hacer ese campo común entre todos los objetos de esa clase. Posteriormente se especifica el tipo, que puede ser tipo primitivo o un tipo referencia, es decir, otra clase. Y para finalizar el nombre del campo.

```
public class Persona {

    //atributos
    static int numeroBrazos;

    private String nombre;
    private String apellido1;
    private String apellido2;
    private int edad;

    //métodos

}
```

Imagen 3. Ejemplo de clase con campos

6.2.2. Métodos

Los métodos, junto a los campos, son la otra parte que componen a las clases. Su principal funcionalidad es dotar de acciones a esa clase usando los datos de sus campos y se identifican por su nombramiento precedido por un verbo.

Se representa dentro del cuerpo de la clase a partir de la siguiente sintaxis:

```
[visibilidad] [static] tipo nombre ([tipo parám.1]
[,tipo parám.N])
{
// declaraciones locales al método
[return valor]
}
```

Igual que sucedía en los campos en los métodos también se especifica la visibilidad, normalmente siempre serán públicos. Si fuera un método estático, se indicará el tipo de datos que devuelve el método, si no devuelve nada se indica con la palabra reservada void, el nombre del método, por estándar siempre ira precedido por un verbo y por último tantos parámetros que se requiera.

Igual que sucedía con los campos, los métodos también pueden ser estáticos. Esta funcionalidad permite acceder a los métodos sin tener que instanciar objetos, simplemente hay que nombrar a la clase para acceder al método. Cabe destacar que, si un método necesitara usar un campo estático, obligato-riamente el método tiene que ser estático.

Dentro del cuerpo del método podemos definir el algoritmo que tiene que realizar la acción, haciendo uso de los paráme-tros, nuevas variables o usando los propios campos de la clase.

Existen diferentes tipos de métodos en las clases en Java:

- **Constructores:** Inicializan los objetos, pueden ser por defecto o con parámetros.
- **Observadores:** También mencionado como getter. Permiten consultar un campo.
- **Modificadores:** También mencionado como setter. Permiten modificar un campo.
- **Métodos personalizados:** Son métodos definidos según la funcionalidad que necesite la clase.

```
public class Persona {

    //atributos
    static int numeroBrazos;

    private String nombre;
    private String apellido1;
    private String apellido2;
    private int edad;

    //métodos
    //observador o getter
    public String getNombre() {
        return nombre;
    }
    //modificador o setter
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    //método personalizado
    public String saludar() {
        return nombre +" "+apellido1+" "+apellido2+ " te saluda";
    }

}
```

Imagen 4. Ejemplo de clase con métodos

Argumentos y parámetros

Como hemos visto en la sintaxis de los métodos podemos indicar que se solicite tantos parámetros como sea necesario. En su declaración irán precedidos por su tipo y nombre; y vendrán separados por comas entra cada parámetro. Cuando se invoque un método se tendrá que incluir todos los argumentos que hayamos definido como parámetro y que coincidan en el tipo.

En el caso que una variable o parámetro de un método se nombrara igual a un campo de la clase, este quedaría oculto al código del método. Para poder acceder al campo sería necesario usar la palabra reservada this para diferenciarlo del campo de una clase a una variable o parámetro del método.

```
public class Persona {

    //atributos
    static int numeroBrazos;

    private String nombre;

    public void saludarA(String nombre) {

        // Se mostrará el nombre que viene por el parámetro del método
        System.out.println("Hola " + nombre);

        // Se mostrará el nombre del campo de clase.
        System.out.println("Hola " + this.nombre);

    }

}
```

Imagen 5. Ejemplo de método con parámetro y campo nombrado igual.

Cuando invocamos a un método y enviamos unos argumentos, estos pueden sufrir cambios, dependiendo si el tipo de paso es por valor o paso por referencia.

- El paso por valor envía una copia del valor original y por lo tanto no se puede modificar el valor original.
- El paso por referencia solo envía una copia de su dirección de memoria, pero no de su estructura, con lo cual, puede ser modificado.

En Java siempre se aplica el paso por valor para todos los tipos primitivos (int,char,float, etc.), en cambio para los objetos se aplica el paso por referencia, eso quiere decir, que si el método modificar algún campo del objeto se verá modificado en el objeto original.

Valores de retorno

Los métodos pueden operar sin tener que devolver ningún valor o devolviendo algún resultado.

Para ello se indicará void en la declaración del método para que no devuelva nada. O de lo contrario indicaremos el tipo de datos a devolver y luego en el cuerpo del método se tendrá que incluir la instrucción return.

```
// Ejemplo de metodo sin devolver ningun valor
public void saludar() {
    System.out.println(nombre +" "+apellido1+" "+apellido2+ " te saluda");
}

// Ejemplo de metodo devolviendo un valor
public float calcularIMC(float peso, float estatura) {
    float imc = peso / estatura;
    return imc;
}
```

Imagen 6. Ejemplo de métodos con valores de retorno.

Sobrecarga de métodos (Overloading)

La sobrecarga de métodos o overloading es la declaración de un mismo método en una misma clase, pero con diferentes parámetros. El método se declarará con el mismo nombre pero tendrá un tipo de datos o un número de parámetros diferente.

```
public int suma(int a, int b) {
    return a+b;
}

public float suma(float a, float b) {
    return a+b;
}
```

Imagen 7. Ejemplo de sobrecarga de métodos

6.2.3. Constructores e instanciación de objetos

Los constructores son los encargados de inicializar los objetos de las clases. Por defecto, todas las clases aún que no se declare disponen de un constructor sin parámetro, y nosotros podremos definir tantos constructores como necesitemos.

Su sintaxis es la siguiente:

```
[visibilidad] clase ([tipo parám.1] [,tipo parám.N]) { }
```

```
public class Persona {

    //atributos
    static int numeroBrazos;

    private String nombre;
    private String apellido1;
    private String apellido2;
    private int edad;

    //métodos
    //constructores
    public Persona() {

    }

    public Persona(String nombre, String apellido1, String apellido2, int edad) {
        this.nombre = nombre;
        this.apellido1 = apellido1;
        this.apellido2 = apellido2;
        this.edad = edad;
    }
```

Imagen 8. Ejemplo clase con constructores

Para instanciar un objeto solo será necesario indicar el operador new junto al nombre de clase. Ejemplos:

```
Persona persona = new Persona();
Persona persona1 = new Persona("Pepe", "Gines", "Gonzalez", 30);
```

Imagen 9. Ejemplo instanciación objetos

6.2.4. Destructores y liberador de memoria

A diferencia de otros lenguajes como C++ o Delphi en el que hay que definir métodos destructores para liberar memoria en Java no es necesario y no existen ese tipo de métodos.

En java existe un mecanismo denominado recolector de basura (garbage collector) que se encarga de liberar la memoria de aquellos objetos que se han quedado sin referencia. Se puede invocar al recolector de basura mediante el método System.gc(). Este método llamará al método finalize() de la super clase Object por cada objeto que se necesite liberar de la memoria.

6.3. Encapsulación y visibilidad

La encapsulación y visibilidad nos permite definir distintos niveles de protección al acceso de la información de una clase. Las diferentes formas que tenemos para garantizar la protección en los datos es mediante la organización de paquetes y clases; y los modificadores de visibilidad en campos y métodos.

La organización de paquetes se verá en el siguiente punto, ahora nos centraremos en la visibilidad. Los modificadores de visibilidad permiten restringir en mayor o menor medida la visibilidad o acceso a elementos de una clase a partir de otra. Existen cuatro niveles de visibilidad:

- **Public:** Visibilidad menos restrictiva permite acceder a sus elementos desde el exterior y cualquier parte del programa.
- **Protected:** Se tendrá acceso a la clase desde el mismo paquete o subclases que estén en diferente paquete. Una clase externa en otro paquete no tendrá acceso.
- **Package o estándar:** Sin modificador definido. Se tendrá acceso a la clase desde el mismo paquete o subclases. Desde el exterior de otros paquetes u otras clases no se podrá.
- **Private:** La más restrictiva. Solo se puede acceder desde la misma clase no del exterior.

Resumen de acceso según su visibilidad				
MODIFICADOR	Public	Protected	Package	Private
Acceso desde la misma clase	SI	SI	SI	SI
Acceso desde otras clases del mismo paquete	SI	SI	SI	NO
Acceso desde una subclase en el mismo paquete	SI	SI	SI	NO
Acceso desde subclases en otros paquetes	SI	SI	NO	NO
Acceso desde otras clases en otros paquetes	SI	NO	NO	NO

6.4. Librerías y paquetes de clases

Los paquetes nos permiten agrupar según una misma idea varias clases de forma jerárquica. Es muy parecido al sistema de archivos de nuestro directorio. Los beneficios que otorgar organizar nuestro programa en paquetes es la reutilización de código, organizar el proyecto, agrupamiento de clases y mayor seguridad.

Para definir un paquete se usa la palabra reservada package y para importar clases de otro paquete se utiliza import

Una librería es una agrupación de paquetes y clases que comparten una misma funcionalidad, permitiendo su reutilización en otros programas. De esta forma podemos integrar en nuestro programa tantas librerías que necesitemos, pudiendo ser librerías estándar, de terceros o propias.

Las librerías más usadas son:

- **Java.io:** Contiene todas las clases para entrada y salida de programas hechos en java.
- **java.util:** Contiene un conjunto de librerías útiles para el modelado de datos, colecciones, internalización etc.
- **java.lang:** La librería fundamental en java contiene todas las interfaces y clases fundamentales. Esta librería esta importada por default.
- **java.security:** Soporta criptografía, firma digital y diferentes clases que permiten encriptar y desencriptar datos.
- **Apache commons:** Una librería con diferentes utilidades a la hora de escribir en ficheros, tratamiento de cadena de caracteres, etc.
- **jSoup:** Nos permite transformar html de forma fácil.
- **Gson:** Nos permite serializar objetos a formato json y luedo deserializarlos.
- **Jfreechar:** Librería para graficar datos, permite hacer diferentes tipos de gráficos y pasarlos a imagen.
- **SWT:** La librería para componentes gráficos de java con la que fue hecha eclipse.

www.universae.com