DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROGRAMACIÓN

TEMA 5

PROGRAMACIÓN ORIENTADA A OBJETOS

CONTENIDOS

5.1. INTRODUCCIÓN.

La Programación Orientada a Objetos (POO) es un paradigma (modelo) de programación totalmente diferente al método clásico (Programación Estructurada o Procedimental). En POO se utilizan objetos y su comportamiento para resolver problemas y generar programas y aplicaciones informáticas.

Con la POO se aumenta la modularidad de los programas y la reutilización de los mismos. Además, la POO se diferencia de la programación clásica porque utiliza técnicas nuevas como la encapsulación, la abstracción, la herencia, el polimorfismo, etc.

Generalmente, los lenguajes de programación de última generación permiten la programación orientada a objetos, así como también la programación clásica (lenguajes multiparadigma), con lo cual puede entenderse la POO como una evolución de la programación clásica.

5.2. CONCEPTO DE OBJETO.

Los programas realizados mediante el paradigma de la POO manejan objetos. Cada objeto del programa pertenece a una clase.

Por ejemplo, el objeto “pepe” pertenecería a la clase “Cliente”. Todos los objetos de la clase “Cliente” se identificarán, entre otros atributos, con un nombre, en este caso “pepe”, con un DNI, una fecha de nacimiento,… o cualquier otro atributo que el programador decida.

Cuando se escribe un programa o aplicación OO, lo que se hace es definir las clases de objetos dotándolas de estado y comportamiento y cuando se ejecute el programa se crearán los objetos necesarios (Java permite la creación de objetos de forma dinámica mediante la instrucción *new*).

Un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso, frente a objetos de una misma clase, al poder tomar valores bien diferenciados en sus atributos.

5.2. CONCEPTO DE OBJETO.

Las clases son los “moldes” de los cuales se generan los objetos. Los objetos se instancian y se generan (instancia y objeto son sinónimos). Por ejemplo, “pepe” será un objeto concreto de la clase “Cliente”.

Cuando se programa, las clases se escriben en ficheros ASCII con el mismo nombre que la clase y extensión *.java*. Las clases tienen una estructura como la siguiente:

```
[…] class nombre_de_la_clase […]
{
    [Atributos]
    [Métodos]
}
```

Los elementos entre corchetes son opcionales. Dentro de una clase podemos encontrar cero o muchos, atributos y/o métodos.

5.2. CONCEPTO DE OBJETO.

Un objeto tiene una serie de características como son las siguientes:

Identidad. Cada objeto es único y diferente a otro objeto. El objeto/cliente “pepe” es diferente a otros clientes aunque tenga el mismo nombre, dni, fecha de nacimiento,…

Estado. El estado será el conjunto de valores de los atributos del objeto.

Comportamiento. El comportamiento serían los métodos que realiza dicho objeto. Dependiendo del tipo o clase de objeto, estos realizarán unas operaciones u otras: comprar, vender, alquilar, pagar,…

Los programas OO están compuestos por objetos, los cuales interactúan unos con otros a través de paso de mensajes. Cuando un objeto recibe un mensaje lo que hace es ejecutar el método asociado.

Los métodos son los procedimientos/funciones que ejecuta el objeto cuando recibe un mensaje vinculado a ese método concreto. En ocasiones este método envía mensajes a otros objetos, solicitando acciones o información.

5.2. CONCEPTO DE OBJETO.

Los objetos del mundo real tienen dos características principales: estado y comportamiento. Al programar una aplicación debemos de modelar estos estados y comportamientos en clases y objetos.

La POO proporciona los siguientes beneficios:

Modularidad. El código fuente de un objeto puede mantenerse y reescribirse sin que ello implique la reprogramación del código de otros objetos de la aplicación.

Reutilización de código. Es muy sencillo utilizar clases y objetos de terceras personas. La ventaja de esto es que no tenemos que conocer los detalles de su implementación interna sino solamente su interfaz.

Facilidad de testeo y reprogramación. Si tenemos un objeto que está dando problemas en una aplicación, no tenemos que reescribir el código de toda la aplicación, sino que tenemos que reemplazar el objeto por otro similar, o bien, reprogramarlo.

Ocultación de información. En POO se ocultan los detalles de implementación y lo que se publica es la interfaz.

5.3. CARACTERÍSTICAS DE LA POO.

Características fundamentales de la POO:

Abstracción. “Separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción”. Cuando se programa OO lo que se hace es abstraer las características de los objetos que van a tomar parte del programa y crear las clases con sus atributos y sus métodos.

Encapsulamiento. Propiedad fundamental de la OO. Cuando se programa OO, los objetos se ven según su comportamiento externo. Por ejemplo, a la clase “Cliente” se le puede enviar un mensaje para establecer su nombre y el objeto “cliente” ejecutará su método establecer nombre. Lo más interesante de todo esto es que el programador no tiene por qué saber cómo funciona internamente el método de establecer nombre, simplemente lo ejecuta.

> *Dicho de otro modo, en la POO las clases son vistas como una caja negra. Los programadores no tienen por qué saber nada de los datos que almacenan y el interior de los métodos que permiten manipularlos, solamente tienen que conocer su interfaz.*

5.3. CARACTERÍSTICAS DE LA POO.

Características fundamentales de la POO:

Herencia. Todas las clases se estructuran formando jerarquías de clases. Las clases pueden tener superclases (por ejemplo, la clase “Cliente” puede tener la superclase “Persona”) y subclases “ClienteOro” o “ClientePreferente”. También existe la posibilidad de que una clase herede de varias superclases. Cuando una clase hereda de una superclase obtiene los métodos y las propiedades de dicha superclase. Además, la funcionalidad propia de la misma clase se combinará con la heredada de la superclase.

> *En Java, una clase solo puede tener una superclase, herencia simple. En C++ se permite la herencia múltiple. Java puede simular la herencia múltiple utilizando interfaces.*

Polimorfismo. Permite crear varias formas del mismo método, de tal manera que un mismo método (con el mismo nombre) ofrezca comportamientos diferentes.

5.4. PROPIEDADES Y MÉTODOS.

En una clase se agrupan datos (variables) y métodos (funciones). Todas las variables o funciones creadas en Java deben pertenecer a una clase, con lo cual no existen variables o funciones globales como en otros lenguajes de programación.

En una clase Java, los atributos (o propiedades) pueden ser tipos primitivos (*char, int, boolean, etc.*) o bien pueden ser objetos de otra clase (lo que se conoce como "Composición"). Por ejemplo, un objeto de la clase "Coche" puede tener un objeto de la clase "Motor".

5.5. PARÁMETROS Y VALORES DEVUELTOS.

Los métodos pueden permitir que se les llame especificando una serie de valores. A estos valores se les llama parámetros. Los parámetros pueden tener un tipo básico (*char, int, boolean, etc.*) o bien ser un objeto.

Además, los métodos (salvo el constructor) pueden retornar un valor o no. Si retornan un valor hay que especificar su tipo delante del nombre del método, si no retornan ningún valor se utiliza la palabra reservada *void*.

Ejemplos:

```
public void setColor (char c) { color=c; }
public char getColor () { return color; }
```

5.6. CONSTRUCTORES Y DESTRUCTORES.

En Java existen unos métodos especiales que son los constructores y destructores del objeto. Estos métodos son opcionales, es decir, no es obligatorio programarlos salvo que se necesiten.

El constructor del objeto es un procedimiento llamado automáticamente cuando se crea un objeto de esa clase. Si el programador no lo declara, Java generará uno por defecto. La función del constructor es inicializar el objeto.

El destructor, por lo contrario, se ejecutará automáticamente siempre que se destruye un objeto de dicha clase. Se utilizan, generalmente, para liberar recursos y cerrar flujos abiertos. Los destructores no reciben parámetros, al contrario que los constructores, la sobrecarga no está permitida.

> *En C++, el destructor se denomina igual que el constructor pero con el símbolo ~ delante del nombre. En Java NO hay destructores como en C++, la destrucción de objetos sigue otra filosofía distinta a la de otros lenguajes de programación.*

5.7. CONCEPTO DE CLASE.

En la POO las clases permiten a los programadores abstraer el problema a resolver ocultando los datos y la manera en la que estos se manejan para llegar a la solución (se oculta la implementación).

En un programa OO es impensable que desde el mismo programa se acceda directamente a las variables internas de una clase si no es a través de métodos *getters* y *setters*.

Por lo tanto, en la definición de nuestras clases hay que tener en cuenta:

> *No se deberá tener acceso directo a la estructura interna de las clases. El acceso a los atributos será a través de getters y setters.*
>
> *En el supuesto que haya que modificar el código sin modificar el interfaz con otras clases o programas, esto debería poder hacerse sin tener ninguna repercusión con otras clases o programas. Se busca que las clases tengan un alto grado de cohesión (independencia).*

5.8. NIVELES DE ACCESO A UNA CLASE.

En Java hay varios niveles de acceso a los miembros de una clase:

public (acceso público).
protected (acceso protegido).
private (acceso privado).
"no especificado" (acceso en su paquete).

Cuando especificamos el nivel de acceso a un miembro de una clase, lo que estamos especificando es el nivel de accesibilidad que va a tener un atributo o método que puede ir desde el acceso más restrictivo (*private*) al menos restrictivo (*public*).

Cuando hablamos de miembros de una clases nos referimos a atributos y/o métodos de la clase.

5.8. NIVELES DE ACCESO A UNA CLASE.

Dependiendo de la finalidad de la clase, utilizaremos un tipo de acceso u otro:

Acceso público. Un miembro público puede ser accedido desde cualquier otra clase que necesite utilizarlo. Una interfaz de una clase estará compuesta por todos los miembros públicos de la misma.

Acceso privado. Un miembro privado puede ser accedido solamente desde los métodos internos de su propia clase. Otro acceso será denegado.

Objeto THIS:

Java, al igual que C++, proporciona una referencia al objeto con el que se está trabajando. Esta referencia se denomina “this”, que no es ni más ni menos que el objeto que está ejecutando el método.

En muchos casos se obvia esta referencia puesto que se sobreentiende que el objeto está invocando al método, en otras ocasiones nos va a servir para resolver ambigüedades o para devolver referencias al propio objeto.

5.9. EJEMPLO.

```java
public class Cliente {

    /* Atributos ***************************************************/
    private String dni;
    private String nombre;

    /* Constructor *************************************************/
    public Cliente() {
        dni="";
        nombre="";
    }

    /* Métodos getters and setters *********************************/
    public void setDNI(String dni) { this.dni=dni; }
    public String getDNI() { return dni; }
    public void setNombre(String nombre) { this.nombre=nombre; }
    public String getNombre() { return nombre; }
}
```

5.9. EJEMPLO.

```java
public class Producto {

    /* Atributos ***************************************************/
    private int codigo;
    private String nombre;
    private double precio;

    /* Constructor *************************************************/
    public Producto() {
        codigo=0;
        nombre="";
        precio=0.0;
    }

    /* Métodos getters and setters *********************************/
    public void setCodigo(int codigo) { this.codigo=codigo; }
    public int getCodigo() { return codigo; }
    public void setNombre(String nombre) { this.nombre=nombre; }
    public String getNombre() { return nombre; }
    public void setPrecio(double precio) { this.precio=precio; }
    public double getPrecio() { return precio; }
}
```

5.9. EJEMPLO.

```java
public class Factura {

    /* Atributos ***********************************************************/
    private String id;
    private String fecha;
    private String dniCliente;
    private int codigoProducto;
    private int cantidad;
    private double total;

    /* Constructor *********************************************************/
    public Factura() {
        id="";
        fecha="";
        dniCliente="";
        codigoProducto=0;
        cantidad=0;
        total=0.0;
    }

    /* Métodos getters and setters *****************************************/
    public void setId(String id) { this.id=id; }
    public String getId() { return id; }
    public void setFecha(String fecha) { this.fecha=fecha; }
    public String getFecha() { return fecha; }
    public void setDNICliente(String dniCliente) { this.dniCliente=dniCliente; }
    public String getDNICliente() { return dniCliente; }
    public void setCodigoProducto(int codigoProducto) { this.codigoProducto=codigoProducto; }
    public int getCodigoProducto() { return codigoProducto; }
    public void setCantidad(int cantidad) { this.cantidad=cantidad; }
    public int getCantidad() { return cantidad; }
    public void setTotal(double total) { this.total=total; }
    public double getTotal() { return total; }
}
```

5.9. EJEMPLO.

```java
import java.util.Scanner;

public class Test01_poo {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        System.out.println("");

        Cliente c=new Cliente();
        System.out.print("Introduzca el dni del cliente: ");
        c.setDNI(sc.next());
        System.out.print("Introduzca el nombre del cliente: ");
        c.setNombre(sc.next());

        Producto p=new Producto();
        System.out.print("Introduzca el codigo del producto: ");
        p.setCodigo(sc.nextInt());
        System.out.print("Introduzca el nombre del producto: ");
        p.setNombre(sc.next());
        System.out.print("Introduzca la precio del producto: ");
        p.setPrecio(sc.nextDouble());
```

5.9. EJEMPLO.

```
        Factura f=new Factura();
        System.out.print("Introduzca el id de la factura: ");
        f.setId(sc.next());
        System.out.print("Introduzca la fecha de la factura: ");
        f.setFecha(sc.next());
        System.out.print("Introduzca la cantidad del producto: ");
        f.setCantidad(sc.nextInt());
        f.setDNICliente(c.getDNI());
        f.setCodigoProducto(p.getCodigo());
        f.setTotal(f.getCantidad()*p.getPrecio());

        System.out.println("");
        System.out.printf("FACTURA:\t %s %s\n",f.getId(),f.getFecha());
        System.out.printf("CLIENTE:\t %s %s\n",f.getDNICliente(),c.getNombre());
        System.out.printf("PRODUCTO:\t %d %s %d %.2f euros\n",
                    f.getCodigoProducto(),p.getNombre(),f.getCantidad(),p.getPrecio());
        System.out.printf("TOTAL:\t\t %.2f euros\n",f.getTotal());
        System.out.println("");
    }
}
```

5.10. USO DE ATRIBUTOS O MÉTODOS ESTÁTICOS Y DINÁMICOS.

Cuando un atributo o método se define como estático (*static)* quiere decir que se va a crear para esa clase solo una instancia de ese atributo o método. Es decir, se reserva la misma zona de memoria para el miembro estático para todos los objetos de esa clase.

Ejemplo:

```
public class Cliente2 {

    /* Atributos ***************************************************/
    private String dni;
    private String nombre;
    private static int numClientes=0;

    /* Constructor *************************************************/
    public Cliente2() {
        dni="";
        nombre="";
    }
```

5.10. USO DE ATRIBUTOS O MÉTODOS ESTÁTICOS Y DINÁMICOS.

```
    /* Métodos getters and setters ****************************************/
    public void setDNI(String dni) { this.dni=dni; }
    public String getDNI() { return dni; }
    public void setNombre(String nombre) { this.nombre=nombre; }
    public String getNombre() { return nombre; }
    public int getNumClientes() { return numClientes; }
    public void setNumClientes(int numClientes) { this.numClientes=numClientes; }

    /* Método Principal ***************************************************/
    public static void main(String[] args) {

        Cliente2 c=new Cliente2();
        c.setNumClientes(1);

        // Uso más adecuado pero necesitaría que el atributo fuera public
        // En este caso funciona porque estoy en la misma clase!!
        Cliente2.numClientes=1;
    }
}
```

5.11. LIBRERÍAS DE OBJETOS (PAQUETES).

Una librería o paquete es un conjunto de clases relacionadas entre sí, las cuales están ordenadas de forma arbitraria.

Las clases que forman parte de un paquete no derivan todas ellas de una misma superclase. Por ejemplo, el paquete “java.io” agrupa las clases que permiten a un programa realizar entrada y salida de información. Un paquete también puede contener a otros paquetes.

Con el uso de paquetes se evitan conflictos, como llamar a dos clases con el mismo nombre (si existen, estarán cada uno en paquetes diferentes). Al estar en el mismo paquete, las clases de dichos paquetes tendrán un acceso privilegiado a los miembros de otras clases del mismo paquete.

Gracias a los paquetes es posible organizar las clases en grupos. Las clases de un mismo paquete están relacionadas entre sí.

5.11. LIBRERÍAS DE OBJETOS (PAQUETES).

Librerías Java (API 8):

Paquete o librería	Descripción
java.io	Librería de Entrada/Salida. Permite la comunicación del programa con ficheros y periféricos.
java.lang	Paquete con clases esenciales de Java. No hace falta ejecutar la sentencia import para utilizar sus clases. Librería por defecto.
java.util	Librería con clases de utilidad general para el programador.
java.applet	Librería para desarrollar *applets*.
java.awt	Librerías con componentes para el desarrollo de interfaces de usuario.
java.swing	Librerías con componentes para el desarrollo de interfaces de usuario. Similar al paquete awt.
java.net	En combinación con la librería java.io, va a permitir crear aplicaciones que realicen comunicaciones con la red local e Internet.
java.math	Librería con todo tipo de utilidades matemáticas.
java.sql	Librería especializada en el manejo y comunicación con bases de datos.
java.security	Librería que implementa mecanismos de seguridad.
java.rmi	Paquete que permite el acceso a objetos situados en otros equipos (objetos remotos).
java.beans	Librería que permite la creación y manejo de componentes *javabeans*.

5.11. LIBRERÍAS DE OBJETOS (PAQUETES).

La sentencia *import*:

Podemos importar una clase individual contenida en un paquete:

import java.lang.System;

O bien podemos importar todas las clases del paquete:

import java.awt.;*

También es posible utilizar una clase sin utilizar la sentencia *import*:

java.awt.Frame fr=new java.awt.Frame("Panel Ejemplo");

Se pueden importar métodos y atributos estáticos de un paquete y acceder a ellos como si hubieran sido declarados en la propia clase:

import static java.lang.Math.floor;

import static java.lang.Math.PI;

5.12. CLASE STRING.

La clase *String* pertenece al paquete *java.lang* y proporciona todo tipo de operaciones con cadenas de caracteres. Esta clase ofrece métodos de conversión a cadenas de números, conversión a mayúsculas, minúsculas, reemplazamiento, concatenación, comparación, etc.

Métodos:

String (String dato) // Constructor
int length()
String concat (String s)
String toString ()
int compareTo (String s)
boolean equals (String s)
String valueOf (…) // cualquier tipo primitivo
…

5.13. CLASE DATE.

El paquete *java.util* dispone de varias clases para manejar fechas y tiempos: *Date, Calendar, GregorianCalendar,…* Estas clases dan diversos problemas, de hecho, muchos métodos han sido "deprecated".

La clase *Date* permite representar un instante dado con precisión de milisegundos. La fecha y la hora se almacenan en un entero tipo *Long* que registra los milisegundos transcurridos desde el 1/1/1970 GMT (tiempo del meridiano de *Greenwich*) a las 00:00:00.

```
Date d=new Date();
System.out.println(d);

long hoy=System.currentTimeMillis();
Date d=new Date();
d.setTime(hoy);
System.out.println(d.toString());
```

5.13. CLASE DATE.

```
Calendar c=Calendar.getInstance();
System.out.println(c.getTime().toString());
System.out.println("Dia: "+c.get(Calendar.DAY_OF_MONTH));
System.out.println("Mes: "+(c.get(Calendar.MONTH)+1));
System.out.println("Año: "+c.get(Calendar.YEAR));
System.out.println("Hora: "+c.get(Calendar.HOUR));
System.out.println("Minuto: "+c.get(Calendar.MINUTE));
System.out.println("Segundo: "+c.get(Calendar.SECOND));

GregorianCalendar g=new GregorianCalendar();
//GregorianCalendar g=new GregorianCalendar(2015,11,25);
//g.set(2015,0,1); // otra forma de asignar una fecha
//g.set(2015,0,1,1,2,3); // otra forma: año, mes, dia, hora, minuto, segundo
System.out.println(g.getTime().toString());
System.out.println("Dia: "+g.get(Calendar.DAY_OF_MONTH));
System.out.println("Mes: "+(g.get(Calendar.MONTH)+1)); // Ojo se suma 1 !!
System.out.println("Año: "+g.get(Calendar.YEAR));
System.out.println("Hora: "+g.get(Calendar.HOUR)); // Tb existe HOUR_OF_DAY
System.out.println("Minuto: "+g.get(Calendar.MINUTE));
System.out.println("Segundo: "+g.get(Calendar.SECOND));
```

5.13. CLASE DATE.

A partir de la versión Java 8, se ha creado una nueva API *java.time* para el manejo de fechas y tiempos. Este paquete es una extensión de las clases *Date* y *Calendar*. Están basadas en el sistema de calendario ISO, el cual es el calendario mundial de facto que sigue las reglas del calendario *Gregoriano*.

```
DayOfWeek lunes=DayOfWeek.MONDAY;
System.out.println(lunes);
Month mes=Month.JANUARY;
System.out.println(mes);

Locale l=new Locale("es","ES");
System.out.println(lunes.getDisplayName(TextStyle.FULL,l));
System.out.println(lunes.getDisplayName(TextStyle.SHORT,l));
System.out.println(lunes.getDisplayName(TextStyle.NARROW,l));
System.out.println(mes.getDisplayName(TextStyle.FULL,l));
System.out.println(mes.getDisplayName(TextStyle.SHORT,l));
System.out.println(mes.getDisplayName(TextStyle.NARROW,l));
```

5.13. CLASE DATE.

```
LocalDate d=LocalDate.now();
//LocalDate d=LocalDate.of(2015,11,1);
System.out.println(d);
System.out.println(d.getDayOfMonth()+"/"+d.getMonthValue()+"/"+d.getYear());
System.out.printf("%02d/%02d/%04d\n",d.getDayOfMonth(),d.getMonthValue(),d.getYear());
System.out.println(d.format(DateTimeFormatter.ofPattern("dd/MM/uuuu")));

LocalTime t=LocalTime.now();
//LocalTime t=LocalTime.of(1,2,3);
System.out.println(t);
System.out.println(t.getHour()+":"+t.getMinute()+":"+t.getSecond());
System.out.printf("%02d:%02d:%02d\n",t.getHour(),t.getMinute(),t.getSecond());
System.out.println(t.format(DateTimeFormatter.ofPattern("hh:mm:ss"))); // "kk:mm:ss"

// Tb podemos utilizar la clase LocalDateTime
```

5.14. OTRAS CLASES.

Para cada tipo primitivo, Java nos ofrece una clase envolvente o *wrapper* que proporciona una serie de métodos estáticos muy utilizados. Ejemplos: *Boolean, Character, Byte, Short, Integer, Long, Float, Double*.

Métodos:

Clase Integer:

static String toString (int i)

static int parseInt (String s)

static Integer valueOf (int i)

static Integer valueof (String s)

Clase Double:

static String toString (double d)

static double parseDouble (String s)

static Double valueOf (double d)

static Double valueOf (String s)

FIN