UT5: Utilización de objetos

RESULTADO DE APRENDIZAJE
2. Escribe y prueba programas sencillos, reconociendo y aplicando los fundamentos de la programación orientada a objetos.

Contenido

1. PROGRAMACIÓN ORIENTADA A OBJETOS

La POO fue una revolución en los años 80, aportando una nueva visión del mundo de la programación al aportar un enfoque distinto a los lenguajes tradicionales imperativos procedurales. La programación orientada a objetos introdujo una nueva forma de organizar el código de un programa, agrupándolo por objetos, que en sí son elementos individuales que contienen funciones e información.

Hoy en día no se entiende la programación de apps para móviles o el desarrollo web sin el uso de un lenguaje POO.

Los elementos básicos de la programación orientada a objetos son los siguientes:

- Atributos o propiedades: en POO cada objeto dispone de una serie de atributos que definen sus características individuales y le permiten diferenciarse de otros (apariencia, estado, etc).
- Método: es una subrutina que puede pertenecer a una clase u objeto, y son una serie de sentencias para llevar a cabo una acción.
- Clase: las clases son un pilar fundamental de la POO y representan un conjunto de variables y métodos para operar con datos.
- Objeto: en el paradigma de programación orientada a objetos, son elementos de un programa que tienen un estado y un comportamiento, conteniendo datos almacenados y tareas realizables durante su ejecución.
- Herencia: la herencia facilita la creación de objetos a partir de otros ya existentes o hace que una subclase obtenga el comportamiento de su clase principal o superclase.
- Polimorfismo: está estrechamente ligado a la herencia y es la capacidad que tienen los objetos de una misma clase de responder al mismo mensaje o evento, en función de los parámetros que se usen.

Ventajas

- Modificabilidad: en la POO es sencillo añadir, modificar o eliminar nuevos objeto o funciones que nos permiten actualizar programas fácilmente.
- Gestión de los errores: cuando se trabaja con un lenguaje POO se sabe exactamente dónde mirar cuando se produce un error, ventaja del trabajo modular de los lenguajes POO. Al poder dividir los problemas en partes más pequeñas se pueden probar de manera independiente y aislar los errores que puedan producirse en el futuro.
- Trabajo en grupo: es más fácil trabajar en grupo gracias al encapsulamiento, que permite minimizar la posibilidad de duplicar funciones cuando varias personas trabajan sobre un mismo objeto al mismo tiempo.
- Herencia: crear una única clase y compartir muchas de sus propiedades con múltiples subclases, permite ahorrar mucho trabajo. Al hacer un cambio en la clase, automáticamente todas las subclases adoptarán el mismo.
- Reducción de costes de programación: especialmente en proyectos grandes la POO reduce los costos de programación ya que se los programadores pueden usar el trabajo de los

otros, ahorrando horas de desarrollo. Crear librerías y compartirlas o reutilizar librerías de otros proyectos es algo habitual en la programación orientada a objetos.

2. CLASES Y OBJETOS

Una clase es una especie de plantilla con una serie de atributos que representa la información que se guardará en un objeto, así como su estado y un conjunto de métodos que permiten al objeto recibir mensajes que determinarán su comportamiento.

Una clase sirve tanto de módulo, como de tipo, es decir, es un módulo porque es una unidad de descomposición del software y es un tipo porque representa la descripción de un conjunto de objetos.

Como se ha expuesto en el apartado anterior, el encapsulamiento consiste en unir en la Clase las características y comportamientos, esto es, las variables y métodos. Es tener todo esto es una sola entidad. En los lenguajes estructurados esto era imposible. No podemos hablar de encapsulamiento sin tener en cuenta antes:

La abstracción

Es un principio que consiste en captar las características generales de un objeto y su comportamiento aislando la información que no resulta relevante ¿qué características tiene un coche? ¿cuáles son comunes a todos? P.e. matrícula, marca, modelo, pero… también ¿qué podría hacer un coche? Acelerar, frenar, calcular gastos, etc. La abstracción es clave para el diseño de un software de calidad.

Ocultación de datos

El usuario de una clase en particular no necesita saber cómo están estructurados los datos dentro de ese objeto, es decir, un usuario no necesita conocer la implementación Al evitar que el usuario modifique los atributos directamente y forzándolo a utilizar funciones definidas para modificarlos (llamadas **interfaces**), se garantiza la integridad de los datos (por ejemplo, uno puede asegurarse de que el tipo de datos suministrados cumple con nuestras expectativas bien que los se encuentran dentro del periodo de tiempo esperado).

La encapsulación define los niveles de acceso para elementos de esa clase. Estos niveles de acceso definen los derechos de acceso para los datos, permitiéndonos el acceso a datos a través de un método de esa clase en particular, desde una clase heredada o incluso desde cualquier otra clase. Existen tres niveles de acceso:

- **público**: funciones de toda clase pueden acceder a los datos o métodos de una clase que se define con el nivel de acceso *público*. Este es el nivel de protección de datos más bajo
- **protegido**: el acceso a los datos está restringido a las funciones de clases heredadas, es decir, las funciones miembro de esa clase y todas las subclases
- **privado**: el acceso a los datos está restringido a los métodos de esa clase en particular. Este es nivel más alto de protección de datos

Por otro lado, un objeto se puede definir como una unidad atómica que encapsula estado y comportamiento. Es una instancia de una clase y todos los objetos de una misma clase comporten ciertas características: sus atributos y el comportamiento que exhiben.

Todo objeto tiene:

- Identidad, puede distinguirse de otros objetos
- Estado, datos asociados a él
- Comportamiento qué puede hacer

Al programar, definimos una clase para especificar cómo se comportan y mantienen su estado los objetos de esas clases. Una vez definida la clase, se crearán tantos objetos como sean necesarios.

Si Vehículo es la clase, los objetos serían Vehículo c1, Vehículo c2, etc.

Objeto = Identidad + Estado + Comportamiento

Identidad: Lo identifica unívocamente, es independiente de su estado y no cambia durante la vida del objeto

Estado: Viene dado por los valores de sus atributos.

Comportamiento: Las acciones realizadas por un objeto son consecuencia directa de un estímulo externo (mensaje enviado desde otro objeto) y dependen del estado del objeto.

A continuación, se muestra una clase Java de nombre Persona con dos atributos: nombre y apellido, un constructor y 5 métodos que representan lo que un objeto de la clase persona puede hacer.

```java
public class Persona {

    private String nombre;
    private String apellido;

    public Persona(String nombre, String apellido) {
        this.nombre = nombre;
        this.apellido = apellido;
    }

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    @Override
    public String toString() {
        return "Persona->{nombre=" + nombre + ";apellido="+apellido+"}";
    }

}
```

La creación de un objeto de esta clase sería:

Persona p=new Persona("Juan","Pérez");

3. HERENCIA

La herencia es el mecanismo de implementación mediante el cual elementos más específicos incorporan la estructura y comportamiento de elementos más generales (Rumbaugh 99).

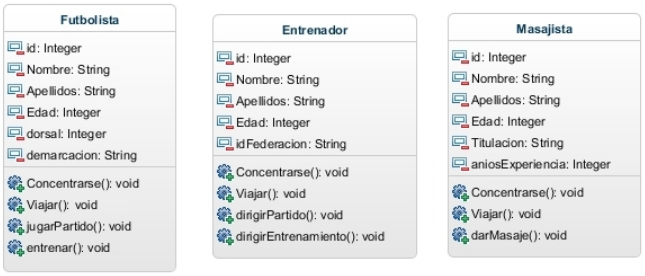
Gracias a la herencia es posible especializar o extender la funcionalidad de una clase, derivando de ella nuevas clases. Esto es muy útil ya que existen clases que comparten gran parte de sus características.

La herencia es siempre transitiva: una clase puede heredar características de superclases que se encuentran muchos niveles más arriba en la jerarquía de herencia. Ejemplo: si la clase Perro es una subclase de la clase Mamífero, y la clase Mamífero es una subclase de la clase Animal, entonces el Perro heredará atributos tanto de Mamífero como de Animal.

La clase A se debe relacionar mediante herencia con la clase B si "A ES-UN B". Por ejemplo, un pájaro es un animal o un coche es un vehículo

En el siguiente ejemplo se muestran 3 clases que modelan el comportamiento que tendrían los diferentes integrantes de la selección española de futbol; tanto los futbolistas como el cuerpo técnico.

De cada uno de ellos vamos a necesitar algunos datos que reflejaremos en los atributos y una serie de acciones que reflejaremos en sus métodos. Estos atributos y métodos los mostramos en el siguiente diagrama de clases:



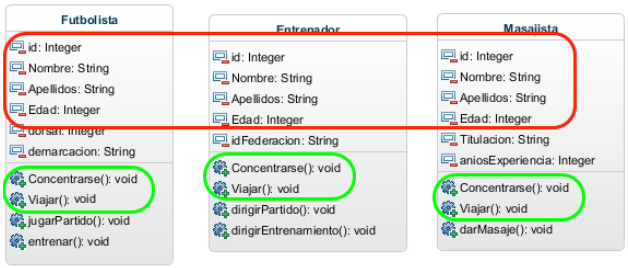
Como se puede observar, en las tres clases tenemos atributos y métodos que son iguales ya que los tres tienen los atributos *id, Nombre, Apellidos y Edad*; y los tres tienen los métodos de *Viajar* y *Concentrarse*

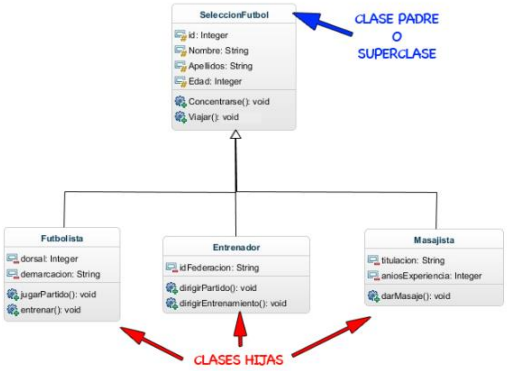
La herencia es un mecanismo que permite definir unas clases a partir de otras heredando todas sus propiedades. Permite reutilizar clases ya existentes.

El resultado es una nueva clase que extiende la funcionalidad de una clase existente sin tener que reescribir el código asociado a esta última.

La nueva clase, a la que se conoce como **subclase**, puede poseer atributos y métodos que no existan en la clase original

Los objetos de la nueva clase heredan los atributos y los métodos de la clase original, que se denomina **superclase**

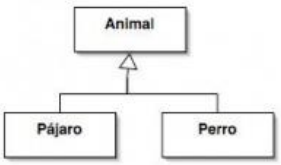




Una mejor solución a este problema será crear una clase con el código que es común a las tres clases que será la Clase Padre o SuperClase y como Clases Hijas o subclases definiremos aquellas que contenga con el código que es especifico.

TIPOS DE HERENCIA

- Herencia Simple: Indica que se pueden definir nuevas clases solamente a partir de una clase inicial

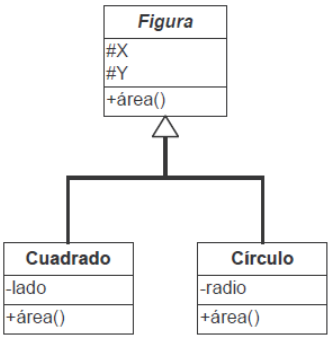


- Herencia de implementación: La implementación de los métodos es heredada. Puede sobreescribirse en las clases derivadas.
- Herencia de interfaz: Sólo se hereda la interfaz, no hay implementación a nivel de clase base.

CLASES ABSTRACTAS

Una clase abstracta es una clase que actúa como un depósito de métodos y atributos compartidos para las subclases de nivel inferior y que no tienen instancias directamente. Se usan para agrupar otras clases y para capturar información que es común al grupo.

Por ejemplo, Figura es una clase abstracta porque no tiene sentido calcular su área, pero sí la de un cuadrado o un círculo.



No existirán objetos de la clase Figura pero si habrá objetos de la clase cuadrado y de la clase círculo.

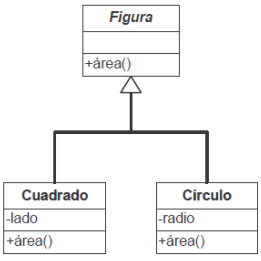
INTERFACES

Un interfaz es una lista de métodos (solamente cabeceras de métodos, sin implementación) que define un comportamiento específico para un conjunto de objetos. Cualquier clase que declare que implementa un determinado interfaz, debe comprometerse a implementar todos y cada uno de los métodos que ese interfaz define. Esa clase, además, pasará a pertenecer a un nuevo tipo de datos extra que es el tipo del interfaz que implementa.

Los interfaces actúan, por tanto, como tipos de clases. Se pueden declarar variables que pertenezcan al tipo del interfaz, se pueden declarar métodos cuyos argumentos sean del tipo del interface, asimismo se pueden declarar métodos cuyo retorno sea el tipo de un interface.

Una clase puede implementar tantos interfaces como desee, pudiendo, por tanto, pertenecer a tantos tipos de datos diferentes como interfaces implemente. En este sentido, los interfaces suplen la carencia de herencia múltiple de Java (y además corrigen o evitan todos los inconvenientes que del uso de ésta se derivan).

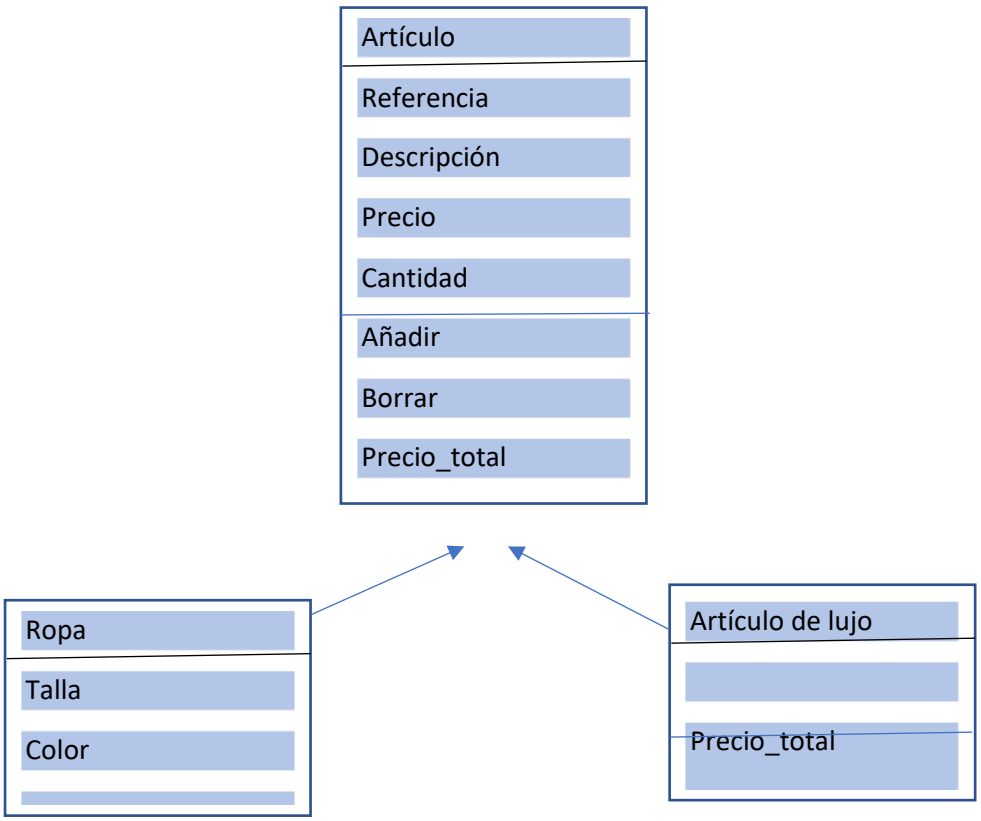
En el ejemplo anterior, si no estuviésemos interesados en conocer la posición de una `Figura`, podríamos eliminar por completo su implementación y convertir `Figura` en una interfaz:




```
public interface Figura
{
  public double area ();
}
```

ENRIQUECIMIENTO Y SUSTITUCIÓN

Los atributos y los métodos definidos en la superclase, son heredados por la subclases, pero éstas también pueden añadir atributos y modificar métodos. Esto se conoce como enriquecimiento y sustitución respectivamente.



Enriquecimiento

(añade nuevos atributos)

Sustitución

(modifica el método precio_total)

4. POLIMORFISMO

En programación orientada a objetos el polimorfismo se refiere a la posibilidad de definir clases diferentes que tienen métodos o atributos denominados de forma idéntica, pero que se comportan de manera distinta.

TIPOS DE POLIMORFISMO

Este es el concepto que se utiliza habitualmente en los lenguajes OO cuando se habla de polimorfismo. Y es el que se considera aquí.

Significa, al menos,

1) Sobrecarga de operadores: el significado de un operador depende del tipo de los operandos. Por ejemplo, agregar el operador + y hacer que se comporte de manera distinta cuando está haciendo

referencia a una operación entre dos números enteros (suma) o bien cuando se encuentra entre dos cadenas de caracteres (concatenación).

2) Sobrecarga de funciones: dos funciones se distinguen por el número y tipo de sus argumentos.

Por lo tanto, podemos por ejemplo, definir varios métodos homónimos de addition() efectuando una suma de valores.

*El método int addition (int,int) devolvería la suma de dos números enteros.

*El método float addition (float, float) devolvería la suma de dos flotantes.

3) Tipos genéricos o parametrización de tipos: un tipo genérico podrá instanciarse para diferentes tipos.

Por ejemplo, podemos crear una estructura de datos de tipo lista basada en arrays gracias a la clase ArrayList de Java. Es posible parametrizar el ArrayList para definir qué tipos de objetos contendrá esta estructura.

```
ArrayList<String> alist=new ArrayList<String>();
ArrayList<Integer> list=new ArrayList<Integer>();
```

4) Polimorfismo de mensajes o dinámico característico de los LOO.

Sea un clase padre P con un método m, con subclases S1 y S2. Las clases S1 y S2 pueden sobreescribir el método m de la clase padre. Se puede usar un objeto genérico de la clase P llamado por ejemplo *padre* y luego asignarle un objeto de las subclases s1 o s2. Al invocar al método m de ese objeto *padre* , se ejecutarán o bien el método m de s1 o el de s2 dependiendo del tipo de objeto en el momento de la ejecución.

5. BIBLIOGRAFÍA

- https://openwebinars.net
- Ramos Martín, A: “Entornos de desarrollo” Garceta 1ª Edición, 2014
- Muller, P: “Modelado de objetos con UML” Ediciones Gestión 2000
- Manualjava.com