POO programación dirigida a objetos

Introducción amable: http://www.ciberaula.com/articulo/tecnologia_orientada_objetos/

Concepto de Clase:

Una clase es una agrupación de ***datos*** (llamados atributos) y de ***funciones*** (llamados métodos) que operan sobre esos datos.

La decisión de QUÉ atributos y métodos tiene una clase se hace como ABSTRACCIÓN de la realidad. Se intenta plasmar los atributos que define QUE DATOS tiene que tener esa clase para el problema que estamos usando y que ACCIONES ha de llevar a cabo.
La definición de una clase se realiza en la siguiente forma:
[public] **class NOMBRECLASE {**
// definición de variables y métodos
...
}
donde la palabra ***public*** es opcional: si no se pone, la clase tiene la visibilidad por defecto, esto es, sólo es visible para las demás clases del ***package.***

POO programación dirigida a objetos

Objetos frente a clases.

Un ***objeto (en inglés, instance) es un ejemplar concreto de una clase.***

Las ***clases*** son como **tipos** de variables, mientras que los ***objetos*** son como **variables** concretas de un tipo determinado***.***

A continuación se muestra un ejemplo de clase en JAVA y su uso en un programa de consola.

En JAVA los programas de consola empiezan por el método main que se encuentra en una clase.
En este método creamos los objetos de la clase persona.

POO programación dirigida a objetos

```
public class principal
{
    public static void  main ( String [] args ) {
        System.out.printf("\nCreamos dos objetos de la clase persona");
        persona p1 = new persona("Pepe",1);
        persona p2 = new persona();
        p2.setId(2);
        p2.setNombre("juan");
        System.out.printf("\nUsamos el método mostrar para mostrar por pantalla los datos");
        p1.mostrar();
        p2.mostrar();

     }
}
```

persona p1 = new persona();

Implica dos acciones:

1. declarar p1 como objeto de la clase persona
   persona p1;
2. Crear realmente el objeto, reservando memoria
   p1= new persona();

POO programación dirigida a objetos

```
public class persona
{
    private String nombre;
    private int idpersona;

    /** Constructores   */
    public persona() {}

    public persona ( String  p_nombre , int p_id){
        nombre=p_nombre;
        idpersona=p_id;
    }
    /*Métodos get y set*/
    public int getId()
    {
        return idpersona;
    }

    public void setId(int p_id) {
        idpersona=p_id;
    }

    public String getNombre (){
        return nombre;
    }
    public void setNombre( String p_nombre){
        nombre=p_nombre;
    }
    /*Métodos*/
    public void mostrar( ){
        System.out.printf("\nDatos de la persona:Id= %d Nombre= %s",this.idpersona, this.nombre );
    }
}
```

POO programación dirigida a objetos

Programación Orientada a Objetos ***(OOP - Object Oriented Programming)***.

Algunos de los conceptos más importantes de la POO son los siguientes:

1. ***Encapsulación.*** Consiste en no permitir el acceso a métodos y atributos cuando ésto no sea necesario. Una clase es como una caja negra , que oculta su estructura interna y tiene unos métodos con los que interactuar con el objeto. Java tiene unos especificadores que permiten limitar el acceso a clases y miembros de la clase.
   Especificadores de clase: public y package (por defecto)
   Especificadores de miembro: public, private, protected y package

   Con ellos se puede controlar el acceso y evitar un uso inadecuado. (lo veremos en próximas diapositivas)

2. ***Herencia.*** *Una clase puede derivar de otra* **(extends***), y en ese caso hereda todas sus* atributos y métodos. Una clase derivada puede **añadir** nuevas variables y métodos y/o **redefinir** los métodos heredados.

3. ***Polimorfismo.*** Ejemplo: Existen dos clases, A y B. La B es subclase de la A. Ambas tienen un método llamado f1 pero con códigos diferentes. Existe polimorfismo, ya que una llamada a f1 puede llevar a ejecutar el código de f1 en A o el de f1 en B en función de cual es el objeto usado para llamar a f1. (A.f1 o B.f1)
Se dice que f1 de b es una redefinición de f1 en A.

POO programación dirigida a objetos

1.PERMISOS DE ACCESO EN JAVA

La *encapsulación* consiste básicamente en ocultar la información que no es pertinente o necesaria para realizar una determinada tarea. Es una de las características importantes en POO.
Los permisos de acceso son una de las herramientas para conseguir la encapsulación.

Accesibilidad de clases o interfaces

Una clase o interfaz con especificador public es accesible para todas las clases, sean o no del mismo package.
Una clase son especificación package o sin especificador (por defecto) solo es visible desde las clases de ese package.

POO programación dirigida a objetos

Accesibilidad de las variables y métodos miembros de una clase

Desde dentro de la propia clase:

1. Todos los miembros de una clase son directamente accesibles desde dentro de la propia clase (sin cualificar con ningún nombre o cualificando con la referencia this). Los métodos de una clase no necesitan que las variables miembro sean pasadas como argumento.

2. Los miembros private de una clase sólo son accesibles para la propia clase**.**

Desde una *sub-clase: (*1)*

1. Las *sub-clases heredan los miembros private de su super-clase, pero sólo pueden acceder a* ellos a través de métodos *public, protected o package de la super-clase.*

POO programación dirigida a objetos

Desde otras clases del *package:*

1. Desde una clase de un package se tiene acceso a todos los miembros que no sean private de las demás clases del package.

Desde otras clases fuera del package:

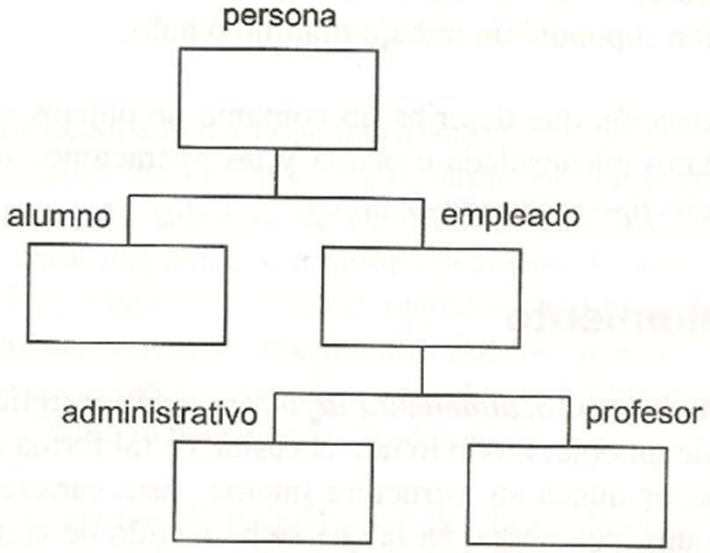
1. Los métodos y variables de una clase A son accesibles si la clase A es public y el miembro (método o variables) es public.
2. También son accesibles si la clase que accede es una sub-clase y el miembro es protected.

Visibilidad	public	protected	private	default
Desde la propia clase	Sí	Sí	Sí	Sí
Desde otra clase en el propio package	Sí	Sí	No	Sí
Desde otra clase fuera del package	Sí	No	No	No
Desde una sub-clase en el propio package	Sí	Sí	No	Sí
Desde una sub-clase fuera del propio package	Sí	Sí	No	No

POO programación dirigida a objetos

2. Herencia.

Un ejemplo de clases que heredan unas de otras sería la que se representa en la imagen. La clase persona tendría como atributos nombre y dirección y como métodos cambiarNombre, cambiarDireccion y listarDatos .



La clase alumno y empleado (tipos de persona) heredan de la clase persona y , por tanto, sin necesidad de código, tienen los atributos y métodos de la clase persona. (recordar ***(*1))***
Además la clase alumno añade como atributo el nombre de su estudio y el curso y como métodos cambiarEstudios, cambiarCurso y listarDatos (ejemplo de polimorfismo) .
Empleado hereda métodos y atributos de persona y añade atributo sueldo y método bajarSueldo (falta implementar subirSueldo) .
A su vez, administrativo y profesor heredan de empleado y añade métodos y atributos propios de la clase que representan.

¿Un objeto de la clase profesor hereda atributos de la clase persona?

POO programación dirigida a objetos

3. Polimorfismo

Binding o vinculación es *la relación que se establece entre la llamada a un método y* el código que efectivamente se asocia con dicha llamada.

La vinculación puede ser:

- **temprana** (en tiempo de compilación)
- **tardía** (en tiempo de ejecución).

El problema de vinculación aparece con el polimorfismo y la sobrecarga.

Con funciones sobrecargadas se utiliza vinculación temprana (es posible y es lo más eficiente).
Con funciones redefinidas en Java se utiliza siempre vinculación tardía, excepto si el método es final (static). El tipo del objeto al que apunta una referencia sólo puede conocerse en tiempo de ejecución, y por eso el polimorfismo necesita evaluación tardía.

El polimorfismo puede hacerse con referencias de super-clases abstract, super-clases normales e interfaces.

POO programación dirigida a objetos

Diferencia entre sobrecarga y polimorfismo.

•La sobrecarga se da siempre dentro de una sola clase, mientras que el polimorfismo se da entre clases distintas.

•Un método está sobrecargado si dentro de una clase existen dos o más declaraciones de dicho método con el mismo nombre pero con parámetros distintos, por lo que no hay que confundirlo con polimorfismo.

•En definitiva: La sobrecarga se resuelve en tiempo de compilación utilizando los nombres de los métodos y los tipos de sus parámetros; el polimorfismo se resuelve en tiempo de ejecución del programa, esto es, mientras se ejecuta, en función de la clase a la que pertenece el objeto.

Un ejemplo clásico de sobrecarga son los constructores, ya que se aplica un código u otro en función de los parámetros.

POO programación dirigida a objetos

Ejemplo con herencia.

```
Creamos dos objetos de la clase persona
Usamos el método mostrar para mostrar por pantalla los datos
Datos de la persona:Id= 1 Nombre= Pepe
Datos de la persona:Id= 2 Nombre= juan
Datos de la persona:Id= 3 Nombre= David
Tiene media : 8,300000
Datos de la persona:Id= 4 Nombre= Carlos
Tiene media : 7,600000
```

```java
public class principal
{
    public static void  main ( String [] args ) {
        System.out.printf("\nCreamos dos objetos de la clase persona");
        persona p1 = new persona("Pepe",1);
        persona p2 = new persona();
        p2.setId(2);
        p2.setNombre("juan");
        System.out.printf("\nUsamos el método mostrar para mostrar por pantalla los datos");
        p1.mostrar();
        p2.mostrar();
        estudiante est1= new estudiante();
        est1.setId(3);
        est1.setNombre("David");
        est1.setMedia((float)8.3);
        est1.mostrar();

        estudiante est2=new estudiante((float)7.6 , 4, "Carlos" );
        est2.mostrar();


    }
}
```

POO programación dirigida a objetos

```java
public class persona
{
    private String nombre;
    private int idpersona;

    /** Constructores   */
    public persona() {}

    public persona ( String  p_nombre , int p_id){
        nombre=p_nombre;
        idpersona=p_id;
    }
    /*Métodos get y set*/
    public int getId()
    {
        return idpersona;
    }

    public void setId(int p_id) {
        idpersona=p_id;
    }

    public String getNombre (){
        return nombre;
    }
    public void setNombre( String p_nombre){
        nombre=p_nombre;
    }
    /*Métodos*/
    public void mostrar( ){
        System.out.printf("\nDatos de la persona:Id= %d Nombre= %s",this.idpersona, this.nombre );
    }
}
```

POO programación dirigida a objetos

```
public class estudiante extends persona
{
    private float media;

    public estudiante(){}
    public estudiante( float p_media ){
        media=p_media;
     }
     public estudiante(float p_media, int p_id, String p_nombre){
          super(p_nombre,p_id);
          media=p_media;
     }
    public void setMedia( float p_media ){
        media=p_media;
     }
    public float getMedia(){
        return media;
     }
     public void mostrar(){
         super.mostrar();
         System.out.printf("\nTiene media : %f",media);
     }

}
```

POO programación dirigida a objetos

El código en JAVA se escribe en ficheros con extensión .java

Un fichero .java puede contener varias clases , pero solo una puede ser pública.

El nombre del fichero debe coincidir con el de la clases pública que contiene.

Los ficheros compilados tienen extensión .class

Las clases de Java se agrupan en packages, que son librerías de clases.

Para que una clase pase a formar parte de un package, en el fichero que contiene a la clase hay que poner:

package utilidades;

Donde utilidades es el nombre del paquete. Esta línea ha de ser la primera línea de código del fichero.

POO programación dirigida a objetos

Los nombres de los packages se suelen escribir con minúsculas, para distinguirlos de las clases, que empiezan por mayúscula.

El nombre de un package puede constar de varios nombres unidos por puntos (los propios packages de Java siguen esta norma, como por ejemplo java.awt.event).

Todas las clases que forman parte de un package deben estar en el mismo directorio. Los nombres compuestos de los packages están relacionados con la jerarquía de directorios en que se guardan las clases. Es recomendable que los nombres de las clases de Java sean únicos en Internet.
Es el nombre del package lo que permite obtener esta característica de unicidad. Una forma de conseguirlo es incluir el nombre del dominio (quitando quizás el país), como por ejemplo en el package siguiente:
es.ceit.jgjalon.infor2.ordenar
Las clases de un package se almacenan en un directorio con el mismo nombre largo (path) que el package. Por ejemplo, la clase, es.ceit.jgjalon.infor2.ordenar.QuickSort.class debería estar en el directorio, CLASSPATH\es\ceit\jgjalon\infor2\ordenar\QuickSort.class donde CLASSPATH es una variable de entorno del PC que establece la posición absoluta de los directorios en los que hay clases de Java (clases del sistema o de usuario), en este caso la posición del directorio es en los discos locales del ordenador.

POO programación dirigida a objetos

Los packages se utilizan con las finalidades siguientes:

1. Para agrupar clases relacionadas.
2. Para evitar conflictos de nombres (se recuerda que el dominio de nombres de Java es la Internet). En caso de conflicto de nombres entre clases importadas, el compilador obliga a cualificar en el código los nombres de dichas clases con el nombre del package.
3. Para ayudar en el control de la accesibilidad de clases y miembros.

Con la sentencia ***import packname;*** se importa un package***,*** esto es , importar todas las clases del package, pero no las clases de subpackage.
Por ejemplo:

Import ***java.awt;*** no importa java.awt.event.

POO programación dirigida a objetos

Se quiere gestionar las personas vinculadas con la facultad, que se pueden clasificar en: estudiantes, profesores y personal de servicio. Por cada persona se debe conocer su nombre y apellidos, su DNI y su estado civil. Si son empleados de la facultad se debe saber su año de incorporación y el número de despacho que tienen asignado. Los profesores sólo pueden pertenecer a un departamento determinado y el personal de servicio a una sección concreta. Por último, se tiene que almacenar a que curso pertenecen los estudiantes.

POO programación dirigida a objetos

Un inmobiliaria vende dos tipos de **Inmuebles**: **Pisos** y **Locales**. Para cualquier tipo de inmueble, se conoce su dirección y el número de metros cuadrados. Además, para los pisos, habrá que conocer el piso concreto en el que se encuentra la vivienda, mientras que para los locales importará el número de ventanas que tenga. Además, cualquiera de estos inmuebles puede ser nuevo o de segunda mano. El precio de cada inmueble se calcula a partir de un precio base y una serie de características: para cualquier inmueble, si tiene menos de 15 años, su precio se rebaja un 1 %, mientras que si tiene más se reduce un 2 %. En el caso de los pisos, si es un tercero o más, su precio se incrementa un 3 %. Para los locales, si tienen más de 50 metros cuadrados el precio se incrementa un 1 %, si tienen 1 o ningún ventanal, su precio se reduce un 2 % y si tienen más de 4 ventanales se añade un 2 %.

POO programación dirigida a objetos

Ejemplo 3:
Un centro cultural se dedica al préstamo de dos tipos de materiales: discos y libros. Para los dos se guarda información general, como su código identificativo, el título y el autor.
En el caso de los libros, almacenamos también su número de páginas, y para los discos el nombre de la discográfica.
Crea una clase principal para probar el programa.

Ejemplo 4:

Se hará una librería de clases que represente figuras tridimensionales y bidimensionales, y su respectiva jerarquía de clases. Las clases deben ser capaces de tener funcionamiento bastante básico, como obtener áreas, volúmenes y perímetros de la figura correspondiente.

Diagrama de classes