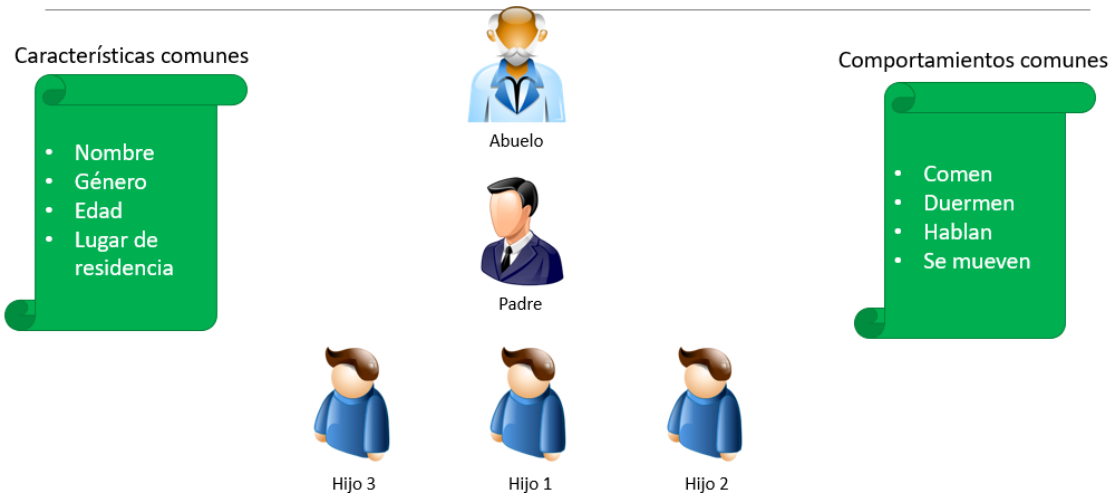
Programación Orientada a Objetos (POO IV)

Herencia

La herencia es una característica fundamental de la Programación Orientada a Objetos que consiste en la creación de nuevas clases a partir de otras ya existentes. Este término ha sido prestado de la vida real donde las personas heredamos los rasgos físicos de nuestros padres e incluso en algunas ocasiones sus bienes.

La herencia es la característica fundamental que distingue un lenguaje orientado a objetos, como C++ o Java, de otro convencional como C, BASIC, etc. Java permite heredar a las clases características y conductas de una o varias clases denominadas **base**. Las clases que heredan de clases base se denominan **derivadas**, estas a su vez pueden ser clases bases para otras clases derivadas. También se utiliza la nomenclatura de "**clases padre**" y "**clases hijas**" o "**superclases**" y "**subclases**". Se establece así una clasificación jerárquica, similar a la existente en Biología con las personas animales y las plantas.



- En la ilustración anterior se observa una herencia típica entre una familia de personas que podemos extrapolar a la herencia en POO. Todas las personas representadas tienen características y comportamientos comunes. Pero además los hijos heredarán algunas características del padre que a su vez podrá heredar características del abuelo. Este comportamiento también se da en la herencia de clases en Java. Cada persona equivaldría a una clase en un programa Java donde ocurriría el fenómeno de la herencia.

La principal ventaja que ofrece la herencia, es **la reutilización del código**. Una vez que una clase ha sido depurada y probada, el código fuente de dicha clase no necesita modificarse. Su funcionalidad se puede cambiar derivando una nueva clase que herede la funcionalidad de la clase padre y le añada otros comportamientos propios. Reutilizando el código existente, el programador ahorra tiempo y dinero, ya que solamente tiene que verificar el funcionamiento que proporciona la clase hija.

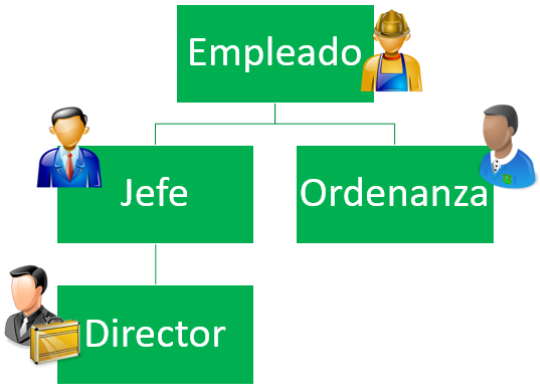
La programación en entornos gráficos, por ejemplo Windows con el lenguaje C++, es un ejemplo ilustrativo. Los compiladores de diversos fabricantes (Microsoft, Borland etc) proporcionan librerías cuyas clases describen el aspecto y comportamiento de las ventanas, controles, menús, etc. Una de estas clases denominada *TWindow* describe el aspecto y la conducta de una ventana, tiene una función miembro denominada *Paint*, que no dibuja nada en el área de trabajo de la misma. Definiendo una clase que herede de *TWindow*, podemos redefinir en ella la función *Paint* para que dibuje una figura. Aprovechamos de este modo la gran cantidad y complejidad del código necesario para crear una ventana en un entorno gráfico. Únicamente tendremos que añadir en la clase hija el código necesario para dibujar un rectángulo, una elipse, etc.

¿Cuándo crearemos una clase padre (base o superclase)? Principalmente bajo dos circunstancias:

1. Cuando caigamos en la cuenta de que diversos tipos (objetos) tienen algo en común. Pongamos por ejemplo el juego del ajedrez: peones, alfiles, rey, reina, caballos y torres, son piezas del juego. Crearíamos, por tanto, una clase padre y derivamos cada pieza individual a partir de dicha clase base.
2. Cuando necesitemos ampliar la funcionalidad de un programa sin tener que modificar el código existente.

La POO nos permite crear tipos (objetos) a partir de una clase, la herencia nos permite extender esos tipos sin necesidad de rescribir todo el código, únicamente escribimos las mejoras y lo demás se hereda de la clase padre.

Una regla que nos puede ayudar en el diseño de la jerarquía de herencia de clases en nuestros programas de Java es la regla de "**Es-un**". Si un objeto de la subclase "**es-un**" objeto de la superclase, entonces el diseño sería correcto. Aclaremos esto con una ilustración:



Vemos aquí una jerarquía de herencia. ¿Cómo sabemos que Empleado es clase padre o superclase, mientras que Jefe y Ordenanza son clases hijas o subclases? ¿Cómo saber que no es por ejemplo al revés? Nos puede ayudar a resolver esto la regla "**es-un**". Un Jefe siempre "**es-un**" Empleado. Sin embargo, un Empleado no siempre "**es-un**" Jefe. De esta forma sabemos quela clase Empleado es superclase y Jefe subclase. Lo mismo ocurriría con la clase Ordenanza. Observa también como la jerarquía de herencia puede extenderse y ramificarse hacia abajo, igual que en la vida real. Una clase Director heredaría de Jefe ya que un Director siempre "**es-un**" Jefe mientras que al revés no. La clase Director sería la más completa de todas ya que además de sus propias características y comportamiento, estaría heredando el de las clases Jefe y Empleado (de la clase Empleado heredaría indirectamente a través de la clase Jefe).

Una vez que tenemos claro el diseño de la herencia, debemos seguir tres simples consejos para que esta sea correcta:

- Poner las operaciones y campos comunes en la superclase.
- Procurar no utilizar campos protegidos.
- No abusar en el uso de la herencia. Usarla solo cuando tenga sentido

La clase Object

En el lenguaje Java, todas las clases heredan implícitamente de la clase base ***Object***, por lo que también poseen las funciones miembro definidas en dicha clase. Las clases derivadas (hijas o subclases) pueden redefinir algunas de estas funciones miembro como toString y definir otras nuevas.

Debido a esto, la clase **Object** es la clase con jerarquía más alta en Java. Todas las clases que podamos encontrarnos en un programa Java extienden a **Object**.

La ilustración de la izquierda es una muestra de la clase **JFrame** de la API de Java. La API nos mostrará siempre en primer lugar la jerarquía de herencia de la clase consultada. Como se puede observar, la clase **Object** siempre estará en la cúspide de la jerarquía de herencia en cualquier clase que consultemos.

La clase **Object** se da por supuesta y jamás hace falta escribir **extends Object**. (extends es la palabra reservada utilizada en java para especificar que una clase hereda de otra)

Excepto los tipos primitivos, el resto de métodos campos etc de cualquier clase de Java, extienden o heredan de **Object**.

¿Cuál es la sintaxis entonces para crear una jerarquía de herencia entre dos clases? Lo puedes ver en la ilustración de la página siguiente.

```java
package es.pildorasinformaticas.pooAbstractas;

import java.util.Date;

abstract class Personas {

    public Personas(String nom){

        nombre=nom;
    }

    public String getNombre(){

        return nombre;
    }

    public abstract String getDescripcion();


    private String nombre;

}

class Empleados extends Personas {

    public Empleados(String nom, Date fechaAlta, double sueldo) {
        super(nom);
        // TODO Auto-generated constructor stub

        this.fechaAlta=fechaAlta;

        this.sueldo=sueldo;
    }
```

La palabra reservada "extends" indica que la clase "Empleados" hereda de la clase "Personas". Esto convierte a la clase "Personas" en clase padre o superclase y a la clase "Empleados" en clase hija o subclase