PROGRAMACIÓN

Unidad 5. Programación Avanzada Orientada a Objetos

1. HERENCIA

Característica principal, capacidad que tienen los objetos de heredar estados y comportamientos de otros objetos. Con el objetivo de conseguir especialización.

En el diagrama de clases se representa con un triángulo.

Automáticamente los estados y comportamientos se heredan de una clase a otra clase. Ejemplo de persona a profesor y alumno.

En la herencia se hereda de arriba a abajo, cuando se busca de abajo a arriba

1.1. TIPOS DE HERENCIA

Se usará dependiendo del lenguaje que usemos.

- Herencia simple: una clase solo puede tener un padre y solo uno.

  JAVA, KOTLIN, JAVA SCRIPT

- Herencia múltiple: una clase puede heredar de dos o más padres.

  PHYTON, C++

2. POLIMORFISMO

Un objeto se puede comportar de una forma u otra o ser un objeto dependiendo de la situación. El profesor se puede comportar como profesor, se puede dar el caso de que se comporte como persona.

3. LENGUAJES DE PROGRAMACIÓN

Para que profesor herede de persona:

```
public class Profesor extends Persona {
        private String modulo;
public Profesor (String nombre, String modulo){
        super(nombre);
        this.modulo = modulo;
        }
}
```

Sí no quiero que haya herencia del comportamiento implementamos @Override para sobrescribir.

Modificadores:

protected --> está protegido para todos, pero va a ser accesible para los descendientes.

Instanceof(¿Qué eres?) hace referencia sí (p2 instanceof Padre) será verdadero ya que hace instancia a esa clase. Se suele utilizar para hacer un código genérico. Para hacer referencia será el mínimo común.

Entre padre, hijo y nieto el mínimo común será padre. Sí quiero acceder a Nieto hacemos lo siguiente:

```
For (int i = 0; i <padres.length; i++){
        If (padres[i] instanceof Nieto){
                ((Nieto)(padres[i])).jugar();
        }
}
```

En el ejemplo del cine tendremos diferentes tipos de butacas, cada butaca tendrá un precio hacemos una matriz de diferentes tipos de butacas.

Ejercicio:
Tenemos una banda de música. Tenemos un músico que toca(), afinar() y tenemos diferentes tipos de musico: tenemos percusionista, viento, cuerda. En viento tenemos dos: vientoMadrea, vientoMetal. En cuerda tenemos otros dos: violín y violista. Percusionista aporrear(). Viento soplar() : vientoMadera cambiarBoquilla(), vientoMetal ponerseRojo(). Cuerda afinar() violín taponar() violista encerar().

Banda de 25 de percusionista
5 percusionista
10 viento --- 6 vientoMadera
------ 4 vientoMetal
10 cuerda --- 5 violines
------- 5 violas
Crear la banda completa, sacar solo percusionista, sacar viento, sacar vientoMadera, sacar vientoMetal, sacar cuerda, sacar violines, sacar violas, preguntar quien toca, preguntar quien afina, preguntar quien encera, preguntar quien aporrea, preguntar quien sopla, quien puede cambiarBoquilla…

4. INTERFAZ FLUIDA

Constructor vacio, getter y setter quitamos void, setter → return mismo objeto es decir this
this.ram;
return this;
En IntelliJ con generate fluent para ello instalando el plugin fluent code, sino se hace de forma manual.
OrdernadoPro p2 = new OrdernadorPro().setProcesador("i5").setModelo("elmodel")……

5. INTERFACES

Una interfaz es un contrato que toda clase que implemente dicha interfaz tiene que cumplir. Tengo que dar respuesta a todos los métodos del contrato de esa interfaz.

```
Interface Mamifero{
        Public void amamantat();
}

Interface Oviparo{
        Public void empollar();
}
```

```
Class Ornitorrinco extends Animal implements Mamifero, Oviparo{
        @Override
        Public void amamantar(){
                --------
        }
        @Override
        Public void empollar(){
                -------
        }
}
```

Siempre o casi siempre que queramos extender el estado se utiliza la herencia, cuando estemos trabajando con el objetivo de mejorar, ampliar, extender a nivel de estado se utiliza la herencia. Cuando queramos definir comportamientos y que luego esos comportamientos sean las clases encarguen de implementarlos.

Las interfaces también pueden implementar constantes, sería public static final …
Podemos crear un vector de interfaces que contiene todos aquellos que heredan esa interfaz.