TÉCNICAS DE PROGRAMACIÓN ORIENTADA A OBJETOS

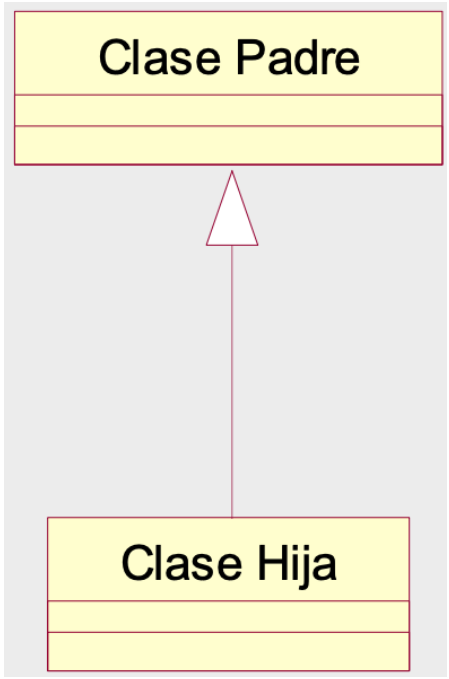
23/04/2024 y 25/04/2024

UPN.EDU.PE

UNIDAD 2: RELACIONES DE CLASES DE HERENCIA SIMPLE Y MÚLTIPLE, COMPONENTES SWING Y ACCESO A DATOS

Sesión 9

- DIAGRAMAS DE CLASES Y SUS RELACIONES.
- RELACIÓN DE HERENCIA SIMPLE



UNIDAD 2: RELACIONES DE CLASES DE HERENCIA SIMPLE Y MÚLTIPLE, COMPONENTES SWING Y ACCESO A DATOS

Sesión 10

- Diagramas de clases y sus relaciones.
- Relación de herencia multiple

LOGRO DE LA SESIÓN:

Al término de la sesión, el estudiante implementa ejercicios prácticos de herencia multiple en modelamiento UML y codificación en Java utilizando lógica y buenas prácticas.

Temario:

- Representación UML en la relación de herencia
- Tipos de herencia
- Herencia Multiple

HERENCIA

- Es una propiedad que permite que los objetos sean creados a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes.
- Es la relación entre una clase general y otra clase mas especifica.
- Es un mecanismo que nos permite crear clases derivadas a partir de clase base, Nos permite compartir automáticamente métodos y datos entre clases subclases y objetos.

HERENCIA – EJEMPLO:

- Si declaramos una clase párrafo derivada de un clase texto todos los métodos y variables asociadas con la clase texto son automáticamente heredados por la subclase párrafo.

HERENCIA – DECLARACIÓN:

- Para declarar la herencia en Java usamos la palabra clave extends.
- Ejemplo: public class MiClase2 extends Miclase1.

HERENCIA – EJEMPLO / DECLARACIÓN:

```
public class Persona {
        private String nombre;
        private String apellidos;
        private int edad;
        //Constructor
        public Persona (String nombre, String apellidos, int edad) {
                this.nombre = nombre;
                this.apellidos = apellidos;
                this.edad = edad;                                   }
        //Métodos
        public String getNombre () { return nombre;   }
        public String getApellidos () { return apellidos;   }
        public int getEdad () { return edad;     }
} //Cierre de la clase
```

HERENCIA – EJEMPLO / DECLARACIÓN:

```
public class Profesor extends Persona {
      //Campos específicos de la subclase.
      private String IdProfesor;
      //Constructor de la subclase: incluimos como parámetros al menos los del constructor de la superclase
      public Profesor (String nombre, String apellidos, int edad) {
            super(nombre, apellidos, edad);
            IdProfesor = "Unknown";     } //Cierre del constructor
      //Métodos específicos de la subclase
      public void setIdProfesor (String IdProfesor) { this.IdProfesor = IdProfesor;     }
      public String getIdProfesor () { return IdProfesor;     }
      public void mostrarNombreApellidosYCarnet() {
            // nombre = "Paco"; Si tratáramos de acceder directamente a un campo privado de la superclase,
salta un error
             // Si podemos acceder a variables de instancia a través de los métodos de acceso públicos de la
superclase
            System.out.println ("Profesor de nombre: " + getNombre() + " " +   getApellidos() +
             " con Id de profesor: " + getIdProfesor() ); }
} //Cierre de la clase
```

HERENCIA – EJEMPLO / DECLARACIÓN:

```
public class TestHerencia1 {
        public static void main (String [ ] Args) {
                Profesor profesor1 = new Profesor ("Juan", "Hernández García", 33);
                profesor1.setIdProfesor("Prof 22-387-11");
                profesor1.mostrarNombreApellidosYCarnet();}
} //Cierre de la clase
```

ANÁLISIS:

- La clase persona es una clase “normal” definida tal y como lo venimos haciendo habitualmente mientras que la clase Profesor es una subclase de Persona con ciertas peculiaridades.

- Los objetos de la subclase van a tener campos nombre, apellidos y edad (heredados de Persona) y un campo específico IdProfesor. El constructor de una subclase ha de llevar obligatoriamente como parámetros al menos los mismos parámetros que el constructor de la superclase.

ANÁLISIS:

- El constructor de la subclase invoca al constructor de la superclase.

- Para ello se incluye, obligatoriamente, la palabra clave super como primera línea del constructor de la subclase. La palabra super irá seguida de paréntesis dentro de los cuales pondremos los parámetros que requiera el constructor de la superclase al que queramos invocar.

ANÁLISIS:

- En este caso solo teníamos un constructor de superclase que requería tres parámetros.

- Si p.ej. hubiéramos tenido otro constructor que no requiriera ningún parámetropodríamoshaberusadounou otro,esdecir,super (nombre, apellidos, edad) ó super(), o bien ambos teniendo dos constructores para la superclase y dos constructores para la subclase.

ANÁLISIS:

En la superclase:

```
public Persona() {
nombre = "";
apellidos = "";
edad = 0;
}

Public Persona (String nombre, String apellidos, int edad) {
this.nombre = nombre;
this.apellidos = apellidos;
this.edad = edad;
}
```

ANÁLISIS:

En la subclase:

```
public Profesor () {
super();
IdProfesor = "Unknown";
}

public Profesor (String nombre, String apellidos, int edad) {
super(nombre, apellidos, edad);
IdProfesor = "Unknown";
 }
```

LOGRO DE LA SESIÓN:

Al término de la sesión, el estudiante implementa ejercicios prácticos de herencia multiple en modelamiento UML y codificación en Java utilizando lógica y buenas prácticas.

Temario:

- Representación UML en la relación de herencia
- Tipos de herencia
- Herencia Multiple

REFLEXIONA

- ¿Qué entiende por herencia multiple?
- ¿Cómo se representa la herencia enprogramación?

HERENCIA

- Es una propiedad que permite que los objetos sean creados a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes. Es la relación entre una clase general y otra clase más específica.
- Es un mecanismo que nos permite crear clases derivadas a partir deuna clase base.
- Permite compartir automáticamente métodos y datos entre clases subclases y objetos.
- Ejemplo: Si declaramos una clase párrafo derivada de una clase texto, todos los métodos y variables asociadas con la clase texto, son automáticamente heredados por la subclase párrafo.

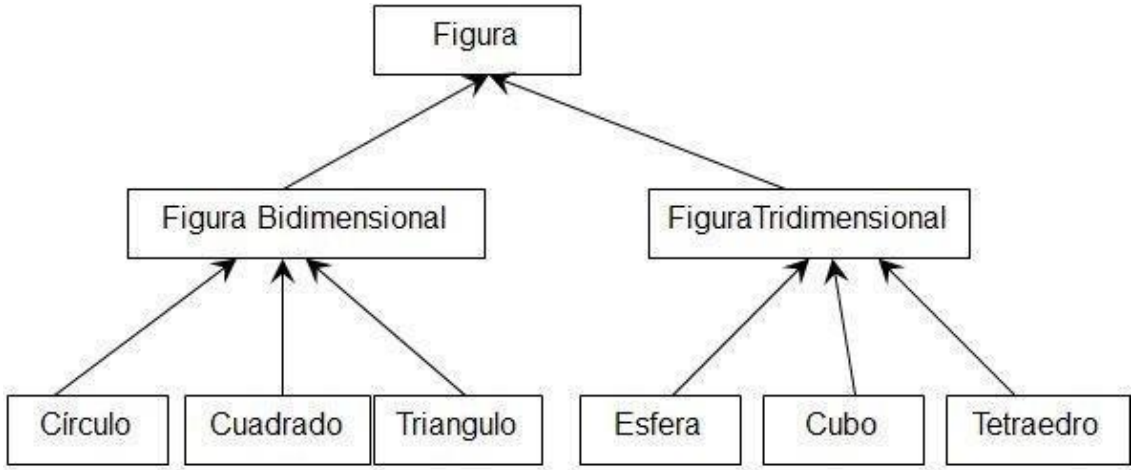
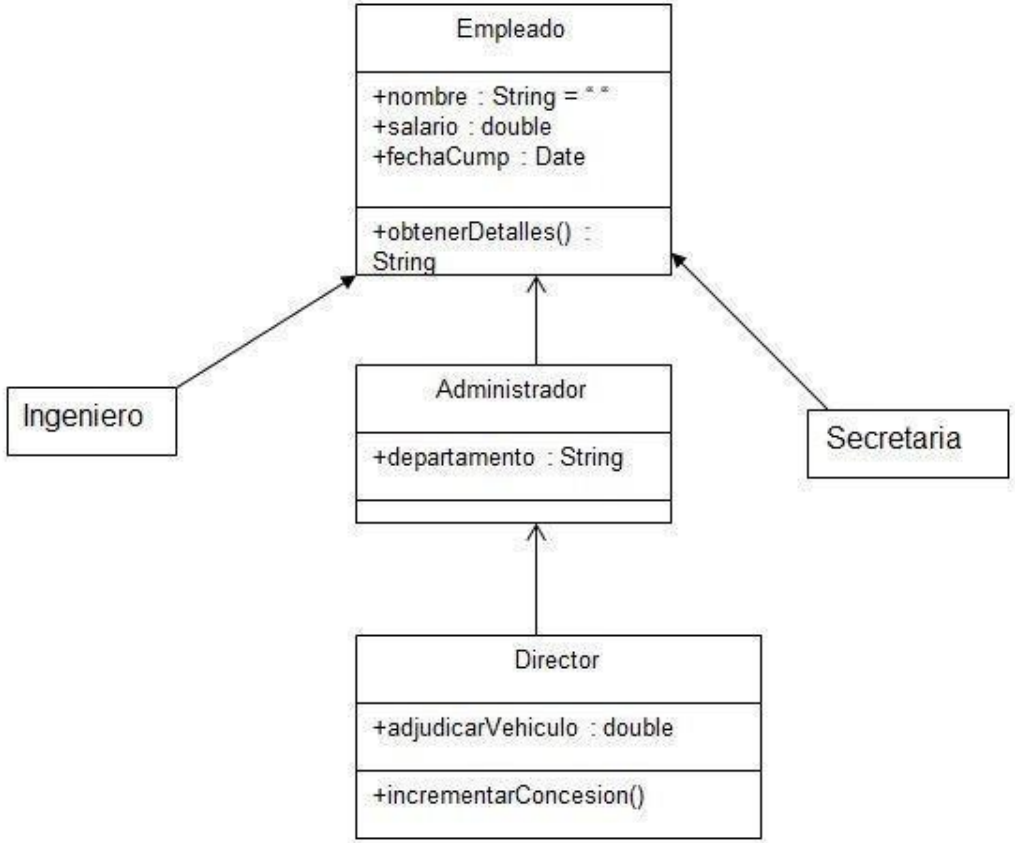
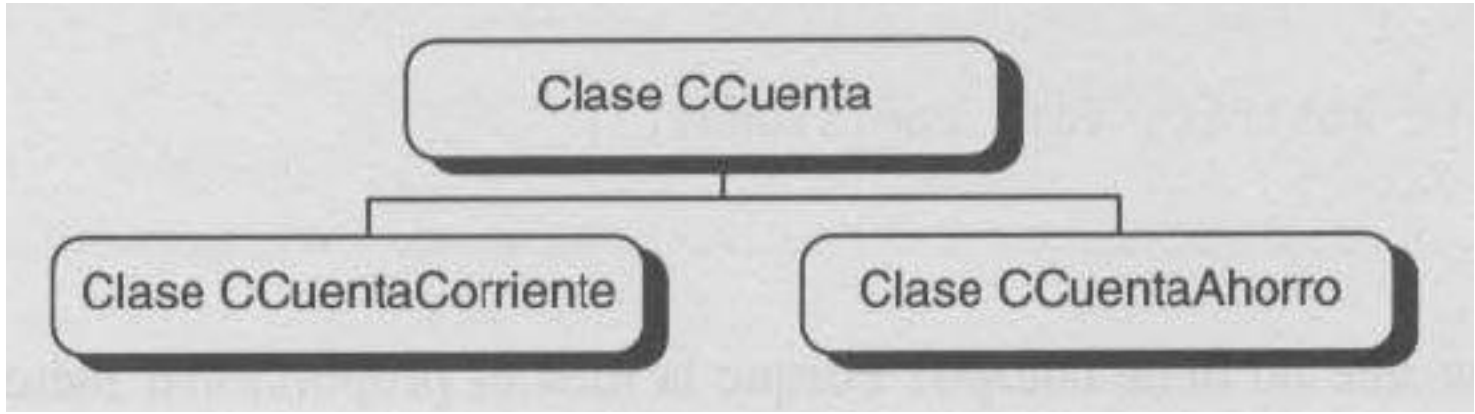


Figura
Figura Bidimensional
FiguraTridimensional
Círculo
Cuadrado
Triangulo
Esfera
Cubo
Tetraedro


Empleado
+nombre : String = " "
+salario : double
+fechaCump : Date
+obtenerDetalles() : String
Ingeniero
Administrador
+departamento : String
Secretaria
Director
+adjudicarVehiculo : double
+incrementarConcesion()


Clase CCuenta
Clase CCuentaCorriente
Clase CCuentaAhorro

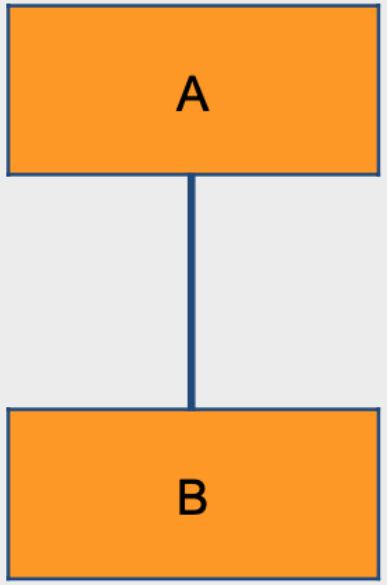
HERENCIA

- Se trata de crear una clase hija – subclase – que hereda de la clase padre – superclase – todos sus atributos, constructores, getters, setters y métodos que pueden ser modificados.
- La subclase puede tener sus propios atributos y métodos. •Permite la reusabilidad delcódigo.
- En Java se implementa mediante: extends

TIPOS DE HERENCIA

- **Simple**
- **Múltiple**, java no las soporta.

HERENCIA



A es un ascendiente o superclase de B. Si la herencia entre A y B es directa decimos que A es la clase padre de B.

B es un descendiente o subclase de A. Si la herencia entre A y B es directa decimos que B es una clase hija de A

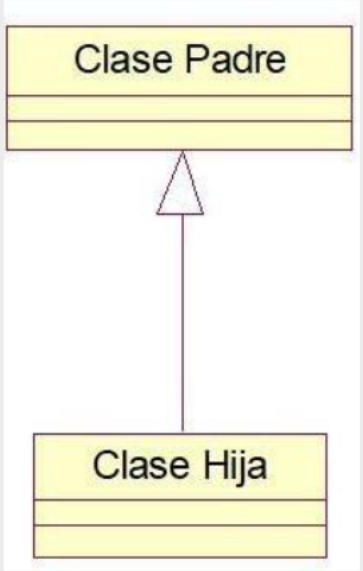
La clase derivada puede *añadir* nuevas variables y métodos y/o *redefinir* las variables y métodos heredados.
La herencia permite que se puedan definir nuevas clases basadas en clases existentes.

HERENCIA

- Java permite definir una clase como subclase de una clase padre.

```
class Clase Hija extends Clase Padre
{


}
```



La clase Object:

En Java existe una clase base que es la raíz de la jerarquía y de la cual heredan todas aunque no se diga explícitamente mediante la cláusula extends.

Por ejemplo, podemos definir **CuentaAhorros** y **CuentaCorriente** como subclases que extienden de **CuentaBancaria**, ya que esta última agrupa las características comunes de ambos tipos de cuenta

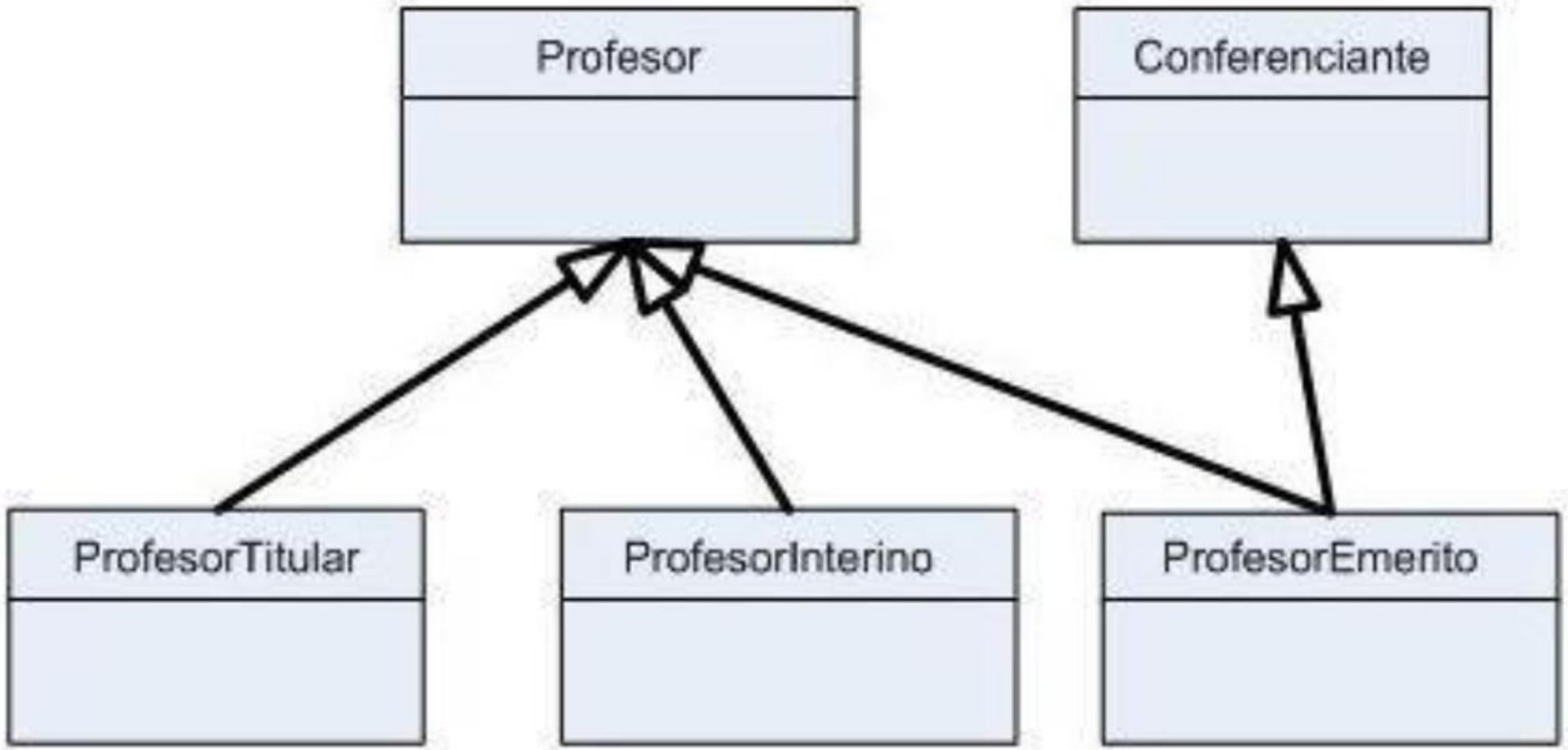
TIPOS DE HERENCIA

Principalmente existen dos tipos de herencia.

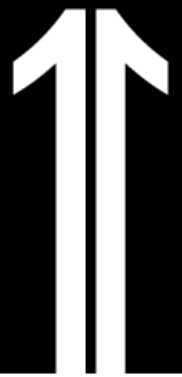
- Herencia simple: una clase solo puede tener un padre, por lo tanto la estructura de clases será en forma de árbol.
- Herencia múltiple: Una clase puede tener uno o varios padres. La estructura de clases es un grafo.

HERENCIA MÚLTIPLE (NO ES SOPORTADA POR JAVA):

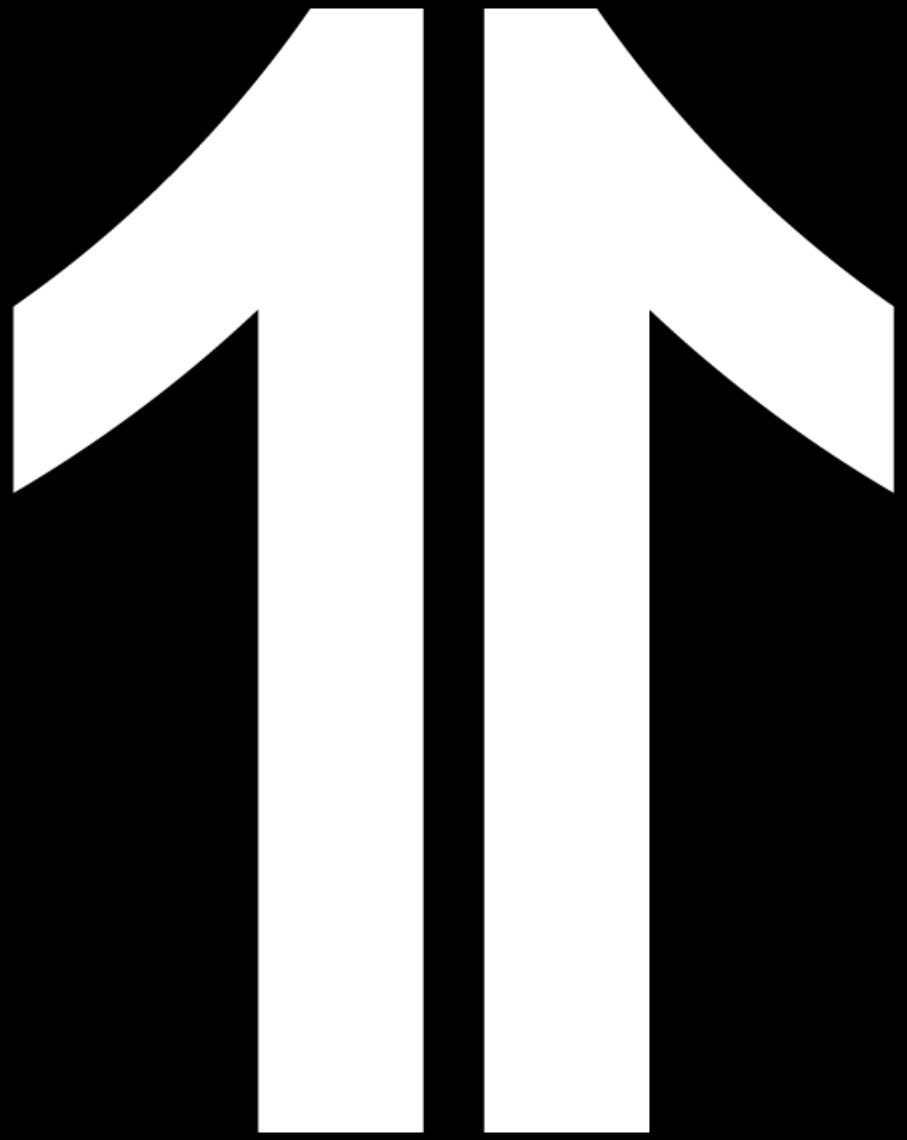
- Es más realista y da al programador más libertad y mas posibilidades de reutilización de código.
- Es mucho más difícil de utilizar por la posibilidad de ciclos y para los lenguajes de programación es muy costoso el permitirlo.


Profesor
Conferenciante
ProfesorTitular
ProfesorInterino
ProfesorEmerito

REFERENCIAS DIGITALES



- https://javadesdecero.es/poo/herencia-java-tipos-ejemplos/
- https://www.mundojava.net/la-herencia-en-java.html?Pg=java_inicial_4_4_6.html
- https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=653: ejemplo-de-herencia-en-java-uso-de-palabras-clave-extends-y-super-constructores-con- herencia-cu00686b&catid=68&Itemid=188
- http://profesores.fi-b.unam.mx/carlos/java/java_basico3_4.html



GRACIAS


UPN
UNIVERSIDAD
PRIVADA
DEL NORTE