ArgentinaPrograma
YoPrograma

# Herencia

Leonardo Blautzik, Federico Gasior, Lucas Videla

Julio / Diciembre 2021

Ministerio de
Desarrollo Productivo
Argentina

# Las Subclases

Supongamos que estamos modelando un Empleado, y que luego se necesita una versión mas **especializada** del mismo, un Gerente. Sabemos que un Gerente **es un** Empleado pero con características adicionales.

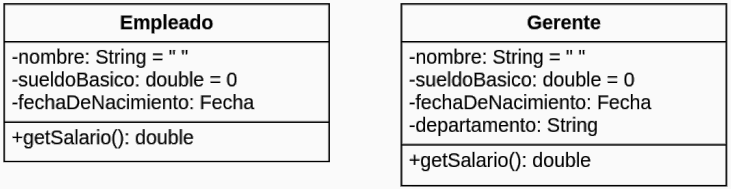



**Figure 1:** Diagrama de clases UML para Empleado y Gerente

# Las Subclases

**Una posible implementación de la class Empleado:**

```
public class Empleado {

    private String nombre = " ";
    private double sueldoBasico = 0;
    private Fecha fechaDeNacimiento;

    public double getSalario() { ... }
}
```

# Las Subclases

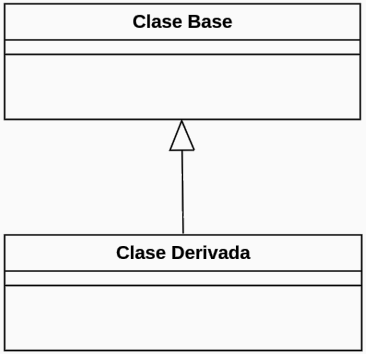
**Una posible implementación de la class Gerente:**

```
public class Gerente {

    private String nombre = " ";
    private double sueldoBasico = 0;
    private Fecha fechaDeNacimiento;
    private String departamento;

    public double getSalario() { ... }
}
```

# Veamos ambas implementaciones en eclipse


Vamos a eclipse, preparamos los tests e implementamos ambas clases...

# Volvimos y vamos a mejorar lo que hicimos...




**usando H E R E N C I A**

# HERENCIA

- La herencia es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente.
- Podemos definir la herencia como la capacidad de crear clases que **heredan** de manera automática los miembros (atributos y métodos) de otras clases que ya existen, pudiendo al mismo tiempo, añadir atributos y métodos propios.
- Una clase que se hereda se denomina **superclase** o **clase base**.
- La clase que hereda de otra, se denomina **subclase** o **clase derivada**.

# Las Subclases

**Modelamos la misma situación Empleado Gerente usando Herencia**

Para evitar la duplicación de datos, vamos a crear a la class Gerente como una **clase derivada** o **subclase** de Empleado.

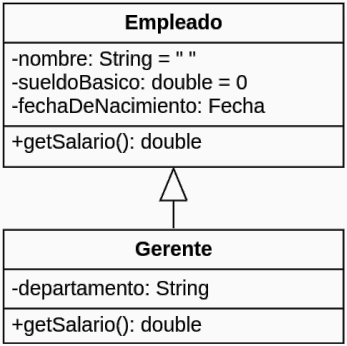



**Figure 2:** Diagrama de clases UML para Empleado y Gerente usando Herencia

# Las Subclases

**La class Gerente usando Herencia:**

En los lenguajes Orientados a Objetos, se proveen mecanismos especiales para que se pueda definir una clase a partir de otra definida previamente.

```
public class Gerente  extends Empleado {
    private String departamento;

    public double getSalario(){...}
}
```

# Las Subclases

La figura muestra una jerarquía de clases donde Empleado tiene tres subclases: Secretaria, Gerente e Ingeniero. La clase Director es subclase de Gerente.

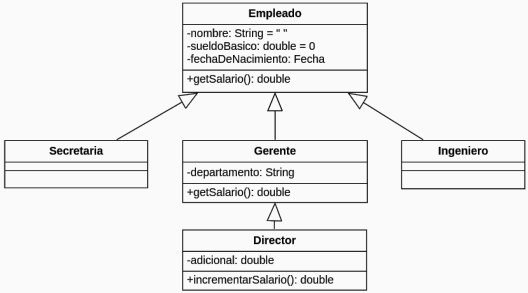



**Figure 3:** Diagrama de clases UML para la jerarquía Empleado

La clase Director hereda todos los miembros de Empleado, los de Gerente y especifica los que le son propios.

# Los controles de Acceso

Las clases solo pueden estar a nivel public o default. Una variable, un método o una clase tienen accesibilidad por defecto si no poseen un modificador explícito.

| Modificador | Misma Clase | Mismo Paquete | Subclase | Universo |
|---|---|---|---|---|
| private | Si | No | No | No |
| default | Si | Si | No | No |
| protected | Si | Si | Si | No |
| public | Si | Si | Si | Si |

# La sobreescritura de métodos

- Con la Herencia, además de crear una clase derivada de una anterior agregando características **(Herencia de Tipos)**, se puede crear una nueva clase basada en una anterior, cambiando el comportamiento de la clase base **(Herencia de Comportamiento)**.
- Si un método se define en una subclase con la misma firma que en la clase base, se dice entonces que el nuevo método **sobreescribe** al anterior.

# La sobreescritura de métodos

Considere estos métodos ejemplo en las clases Empleado y Gerente:

```
public class Empleado {
    protected String nombre = " ";
    protected double sueldoBasico = 0;
    protected Fecha fechaDeNacimiento;

    public double getSalario() {
        return this.sueldoBasico;
    }
}
```

# La sobreescritura de métodos

```
public class Gerente  extends Empleado {
    private String departamento;

    public double getSalario() {
        return super.getSalario() * 1.20;
    }
}
```

- El método getSalario() de Gerente sobreescribe al de Empleado pagando un 20% mas del salario de un Empleado.
- Invoca al méto getSalario() de la clase base a usando de la palabra reservada **super**.
- Un método que sobreescribe no pueden ser menos accesibles que el método sobreescrito.

# Veamos como quedan ambas clase y si pasan los Tests


Vamos a eclipse y veamos si la nueva implementación sigue pasando nuestros tests...

# La sobrecarga de métodos

**Definición:**

Dos o más métodos están sobrecargados, cuando tienen el mismo nombre, pero difieren en sus argumentos(tipo o cantidad) o en su tipo de retorno.

Ejemplo: sobrecarga del método imprimir dónde el tipo del argumento es diferente en cada caso:

```
public void imprimir(int x) { ... }
public void imprimir(float y) { ... }
public void imprimir(String s) { ... }
```

Ejemplo: sobrecarga del método promedio dónde la cantidad de los argumetos difiere en cada caso:

```
public double promedio(int x1, int x2) { ... }
public double promedio(int x1, int x2, int x3) { ... }
public double promedio(int x1, int x2, int x3, int x4 ) { ... }
```

# La Sobrecarga de Constructores

Cuando un objeto es instanciado, el programa debería ser capaz de proveer múltiples constructores basados en los datos que pueden ser necesarios para la constucción del objeto que está siendo creado.

Veamos un ejemlo de ésto con la class Empleado:

```
public class Empleado {
    public static final double SALARIO_BASICO = 150000;
    public String nombre = " ";
    public double sueldoBasico = 0;
    public Fecha fechaDeNacimiento;

    public Empleado(String nombre, double sueldoBasico, Fecha feNac){
        this.nombre = nombre;
        this.sueldoBasico = sueldoBásico;
        this.fechaDeNacimiento = feNac;
    }
```

# La Sobrecarga de Constructores

```
    public Empleado(String nombre, double sueldoBasico){
        this(nombre, sueldoBasico, null);
    }

    public Empleado(String nombre){
        this(nombre, SALARIO_BASICO);
    }

    public double getSalario() { ... }
}
```

# Los Constructores no se heredan

Aunque una subclase hereda todos los métodos y variables desde una clase base, **no hereda sus constructores**.

Hay solo dos maneras de que una clase tenga un constructor:

- Que el programador escriba un constructor.
- Que el programador no escriba ningún constructor y se asigne un **constructor por defecto**.

# Invocación a los Constructores de la clase Padre

Los constructores pueden llamar a los constructores no privados de la superclase inmediata.

Se debe colocar **super(...)** o **this(...)** en la primera línea del constructor. Si no se hace explícitamente, el compilador inserta automáticamente una llamada al constructor de la clase base sin argumentos super() aunque ésto no se vea reflejado en el código.

```
public class Gerente extends Empleado {
    private String departamento;
```

```
    public Gerente(String nombre, double sueldoBasico, String departamento){
        super(nombre, sueldoBasico);
        this.departamento = departamento;
    }

    public Gerente(String nombre, String departamento){
        super(nombre);
        this.departamento = departamento;
    }

    public Gerente(String departamento){ //Fallará porque no disponemos
                                         //de super() en Empleado
        this.departamento = departamento;
    }
}
```

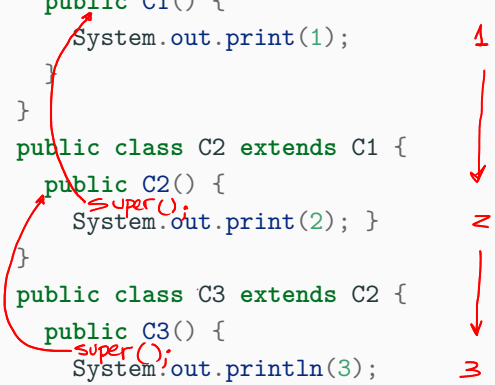
# Desafío:

Nos tomamos un café, meditamos el proceso de inicialización de objetos y analizamos dos fragmentos de código...

# El proceso de inicialización de objetos, un ejemplo práctico:

¿Cual será la salida por consola cuando el siguiente código sea compilado y ejecutado?

```
public class C1 {
  public C1() {
    System.out.print(1);
  }
}
public class C2 extends C1 {
  public C2() {
    System.out.print(2); }
}
public class C3 extends C2 {
  public C3() {
    System.out.println(3);
  }
}
```



```
public class Test {
  public static void main(String[] a) {
    C3 c= new C3();
  }
}
```

**Veamos las ociones posibles:**

Elija, elija!!!

a. 3
b. 23
c. 32
d. 123
e. 321
f. Compilation fails.
g. An exception is thrown at runtime.

# Otro ejemplo:

```
public class Parent {
    public Parent(int x) {
      System.out.print("A");
    }
     // public Parent(){}
}
class Child extends Parent {
    public Child(int x) {// Falla porque Parent no implementa el constructor por defecto
        super(); -> Aunque no lo veamos, siempre está.
        System.out.print("B");
    }
    public Child() {
        super(123);
        System.out.print("C");
    }
```

```
    public static void main(String[] args) {
        new Child();
      }
    }
```

**Opciones posibles:**

Elija, elija!!

¿Cuál es el resultado de la sentencia new Child();

a. ABC
b. ACB
c. AB
d. AC
e. Error de compilación

# FIN

Continuará...

# ¡Muchas Gracias!

continuará...

Ministerio de
Desarrollo Productivo
Argentina