Polimorfismo

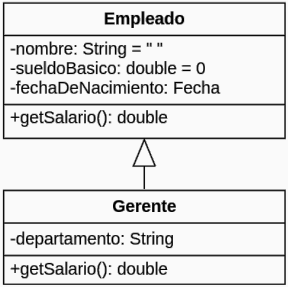
CESSI #ArgentinaPrograma #YoProgramo

Leonardo Blautzik - Federico Gasior - Lucas Videla

Agosto -Diciembre de 2021

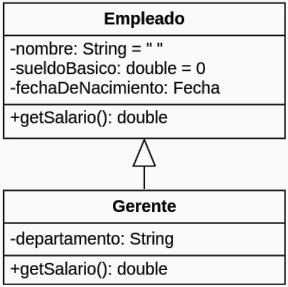
POLIMORFISMO

Polimorfismo



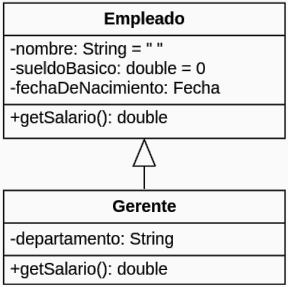
- Decir que un Gerente **es un** Empleado **no** es solo una manera conveniente de describir la relación entre las dos clases.

Polimorfismo



- Decir que un Gerente **es un** Empleado **no** es solo una manera conveniente de describir la relación entre las dos clases.
- Gerente tiene todos los miembros de Empleado.

Polimorfismo



- Decir que un Gerente **es un** Empleado **no** es solo una manera conveniente de describir la relación entre las dos clases.
- Gerente tiene todos los miembros de Empleado.
- Toda operación que puede ser aplicada a un Empleado, puede también ser aplicada a un Gerente.

Polimorfismo

Un Objeto tiene solamente **una forma**, la que se le da cuando es construido.

```
Empleado e = new Empleado();
Gerente g = new Gerente();
```

Polimorfismo

Una variable (referencia), sin embargo, es **polimórfica** porque puede referenciar a Objetos con **formas** diferentes.

```
Empleado e = new Empleado();
Gerente g = new Gerente();
e = g  // ¡Legal! un Gerente ES UN Empleado. Sólo serán accesibles
       // las partes de e que son propias de Empleado.
e.getDepartamento(); // Ilegal
g = e  // ¡Ilegal¡ un Gerente NO ES UN Empleado.
```

Polimorfismo

```
Empleado e = new Empleado();
Gerente g = new Gerente();
e.getSalario(); // return this.sueldoBasico
g.getSalario(); // return super.getSalario() * 1.2;
```

Es claro por lo visto en Herencia que las solicitudes a `getSalario()` invocan a **comportamientos** diferentes

Polimorfismo

En Java, así como en la mayoría de los lenguajes orientados a objetos, se permite referenciar a un objeto con una variable que tiene uno de los tipos de de las superclases del objeto, por lo tanto es posible hacer:

```
Empleado e = new Empleado();
Empleado g = new Gerente();
e.getSalario(); // return this.sueldoBasico
g.getSalario(); // ??
```

Es menos obvio ahora el comportamiento asociado a `g.getSalario()`

Polimorfismo

El **comportamiento** está asociado con el objeto al cual la variable hace **referencia** en tiempo de ejecución.
El **comportamiento** no está determinado por el **tipo** de la variable al momento de la compilación.
Este es un aspecto del **polimorfismo** y una característicaimportante de los lenguajes orientados a objetos.

Arreglos Heterogéneos

```
Empleado [] staff = new Empleado[124];
staff[0] = new Gerente();
staff[1] = new Ingeniero():
staff[2] = new Empleado();
```

Se podría por lo tanto, escribir un método que ordene al staff por edad o por salario, sin importar de que tipo es cada uno, **todos** son Empleado.
¿Magia? No, Polimorfismo.

Arreglos de `Object`

Nota: Cada clase es una subclase de `Object`. Por consiguiente, es posible usar un arreglo de `Object` como un contenedor para cualquier combinación de objetos.
Para agregar variables de tipos primitivos a un arreglo de `Object` se deben usar las clases de envoltura.

Los Argumentos Polimorficos

Se podría crear un método en una clase que reciba un Empleado como parámetro y calcule, por ejemplo, los impuestos correspondientes a ese Empleado. Usando las características polimórficas ésto se puede resolver de la siguiente manera:

```
public class CalculadoraDeImpuestos {
    public TasaDeImpuesto calcularTasa(Empleado e) {
        //realiza los calculos y devuelve la tasa de impuestos del Empleado e
    }
}
```

Los Argumentos Polimorficos

Será válido entonces, en algún punto de la aplicación:

```
CalculadoraDeImpuestos calcuDeImp = new CalculadoraDeImpuestos();
Gerente g = new Gerente();
TasaDeImpuesto tasa = calcuDeImp.calcularTasa(g);
```

Esto es legal, puesto que g **es un** Empleado. Sin embargo el método `calcularTasa` solo tiene accesos a los miembros de g que están definidos en Empleado.

La conversión de objetos (casting)

En circunstancias en que se ha recibido como parámetro, una referencia a un objeto de una super clase (Empleado), y se ha determinado efectivamente que la referencia es del tipo de alguna de sus subclases, por ejemplo un Gerente, se puede acceder a la funcionalidad completa de ese objeto convirtiendo (casting) la referencia.

```
public void hacerAlgo(Empleado e) {
  if( e instanceof Gerente ){
  //el operador instanceof devolverá true si e fue construido como un Gerente
    Gerente g = (Gerente) e;
    System.out.println(g.getDepartamento());
  }
}
```

La conversión de objetos (casting)

- La conversión hacia arriba siempre está permitida y basta con una asignación.
- Si el compilador permite la conversión hacia abajo, el tipo del objeto es verificado en tiempo de ejecución.
- Si se omite la comprobación por medio de `instanceof` y se recibe por parámetro un objeto que no puede ser convertido, se lanzará una exception en tiempo de ejecución.

CLASES ABSTRACTAS

Las Clases Abstractas

- El lenguaje de programación Java permite al diseñador de clases especificar que una superclase declare un método pero que no provea ninguna implementación para él.
- A estos métodos se los denomina **métodos abstractos**.
- La implementación de éstos métodos queda a cargo de las subclases (están obligadas a implementarlos).
- Cualquier clase con uno o más métodos abstractos es una **clase abstracta**.

Las Clases Abstractas

- Las clases abstractas pueden tener además atributos, métodos concretos y constructores.
- Los constructores de una clase abstracta solo pueden ser usados para inicializar sus atributos desde las subclases, ya que **una clase abstracta no puede ser instanciada**
- Una clase puede ser daclarada como abstracta aunque no tenga métodos abstractos, tal vez con el fin de que no sea instanciada.
- Una clase que tiene métodos abstractos no puede no ser abstracta.

Las Clases Abstractas

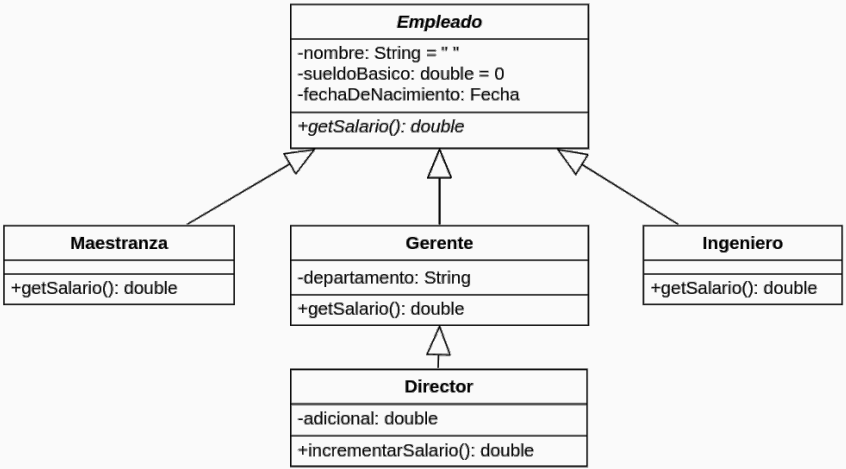


Figure 1: Modelo UML de una jerarquía donde Empleado es una clase abstracta

Desafío:

Nos tomamos un café, meditamos todo esto y volvemos renovados para aplicarlo a un problema lleno de empleados y gerentes ;-)

Empresas ACME: Modele una empresa con empleados.

Una empresa conoce a todos sus empleados. Los empleados pueden ser de planta permanente o temporaria, además hay gerentes, que también son empleados de planta permanente, pero siguen un régimen salarial particular.
Cuando un empleado es de planta permanente cobra la cantidad de horas trabajadas por $3000, más antigüedad ($1000 por año de antigüedad), más salario familiar.
Cuando es de planta temporaria, no cobra antigüedad y cobra la cantidad de horas trabajadas por $2000, más salario familiar.
El salario familiar es $2000 por cada hijo, los empleados casados además cobran $1000 por su esposa/o.
Un gerente cobra de manera similar a un empleado de planta permanente pero su hora trabajada vale $4000, por antigüedad se le pagan $1500 por año, mientras que el salario familiar es el mismo que el de los empleados de planta permanente y temporal.

Empresas ACME: Continuación

1. Defina e implemente el mensaje montoTotal() en la clase Empresa, que retorna el monto total que la empresa debe pagar en concepto de sueldos a sus empleados.
2. Provea un TestEmpresa para instanciar y testear su sistema creando el siguiente escenario y envíe a la empresa el mensaje montoTotal() para obtener la liquidación total:
    - Una empresa, con el CUIT y Razón Social que desee (ACME), y con los cuatro empleados que se describen a continuación:
    - Un empleado de Planta Temporaria con 80 horas trabajadas, con esposa y sin hijos.
    - Un empleado de Planta Permanente (que no sea gerente) con 80 horas trabajadas, con esposa, 2 hijos y 6 años de antigüedad.
    - Un empleado de Planta Permanente (que no sea gerente) con 160 horas trabajadas, sin esposa, sin hijos y con 4 años de antigüedad.
    - Un Gerente con 160 horas trabajadas, con esposa, un hijo y 10 años de antigüedad.

Veamos como resolverlo...

Vamos a eclipse y veamos si podemos modelarlo aplicando todo lo visto hasta ahora.