UDGVIRTUAL

POO

HERENCIA

GENERALIZACIÓN - ESPECIALIZACIÓN

FECHA	AGOSTO, 2018 GUADALAJARA, JALISCO. MÉXICO	REALIZADO POR	PROF. SABRNA LIZBETH VEGA MALDONADO

Definición

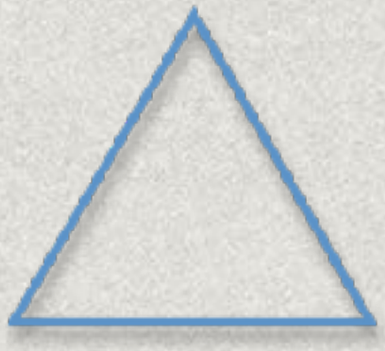
- La definición de herencia para la programación orientada a objetos se basa en la definición que se usa en la vida cotidiana que es trasmitir valores, bienes materiales o características físicas de una persona a otra persona

- En la programación en lugar de ser valores, bienes materiales o características físicas lo que se trasmite son atributos y objetos y en lugar de ser de una persona a otra persona es de una clase a otra clase

- Por lo tanto podemos definir formalmente a la herencia como un mecanismo que permite compartir atributos y métodos entre clases tomando como base una relación jerárquica

- Una clase denominada “clase padre” o “superclase” trasmite o hereda sus características a otra clase denominada “clase hija” o “subclase”

- Su simbología en el diseño es un triángulo vacío que apunta a la clase padre o superclase y de su base sale una línea a las clases hijos o subclases



- La palabra reservada en el lenguaje de java para la herencia es “extends”

- y su sintaxis en el lenguaje de java es:

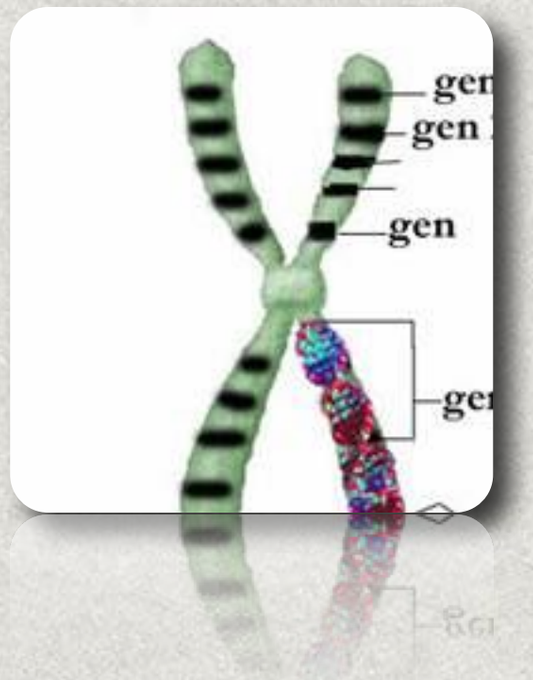
```
[<tipoAcceso>] class <SuperClase> {

}

[<tipoAcceso>] class <SubClase> extends <SuperClase> {

}
```

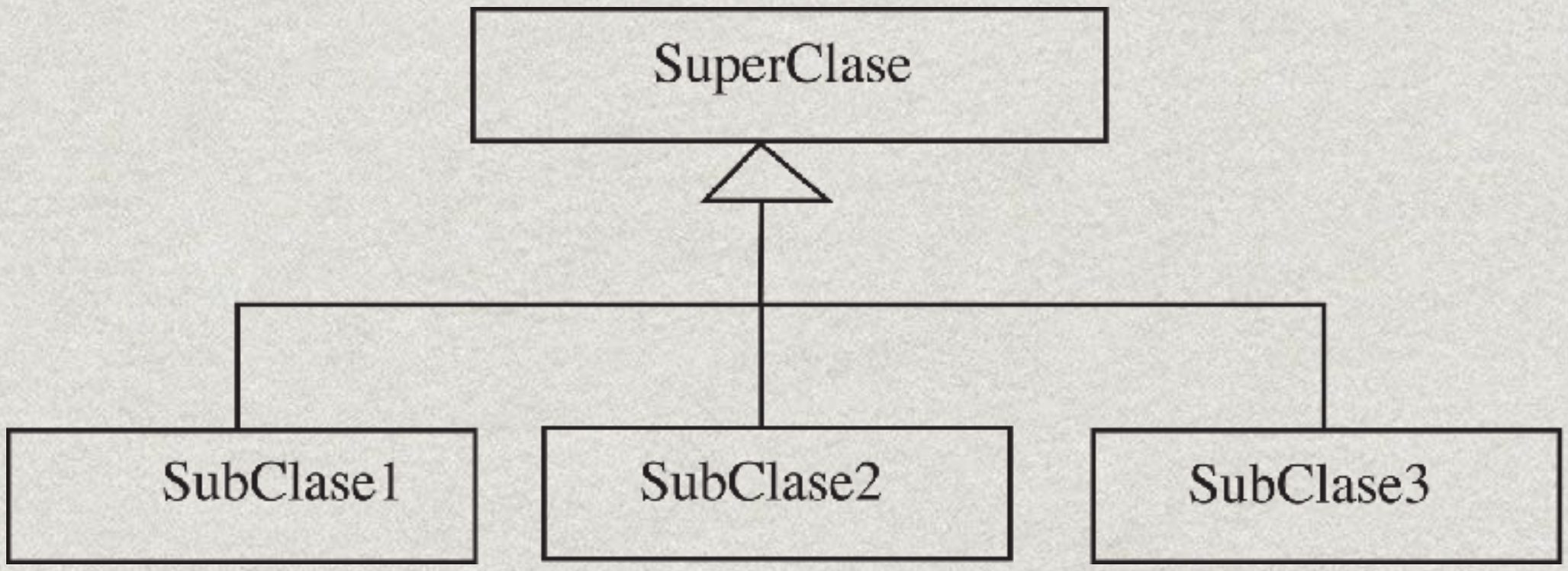
Tipos de Herencia

- Herencia Simple
- Herencia Multiple

Herencia Simple

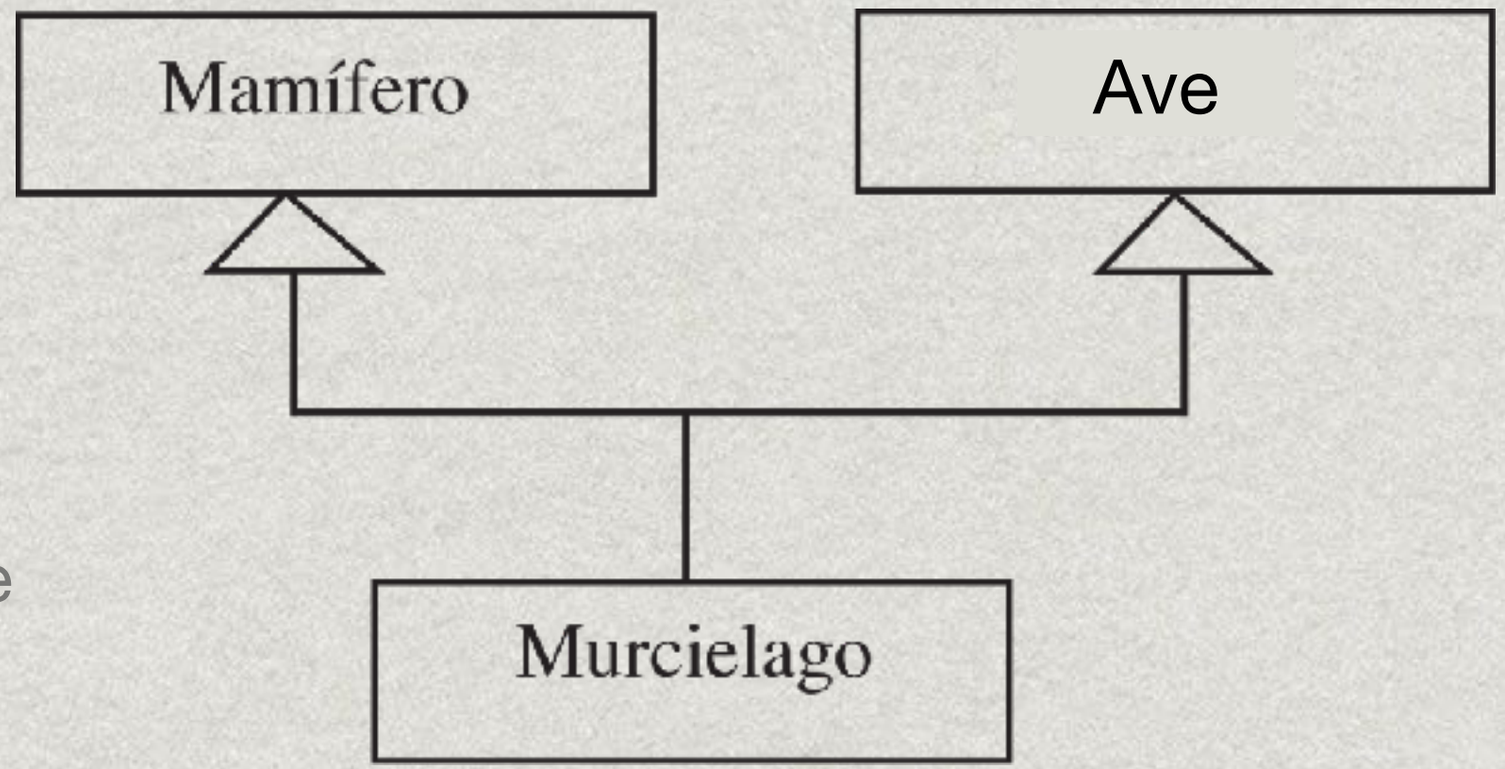
- Es cuando una o varias clases sólo pueden tener una clase padre





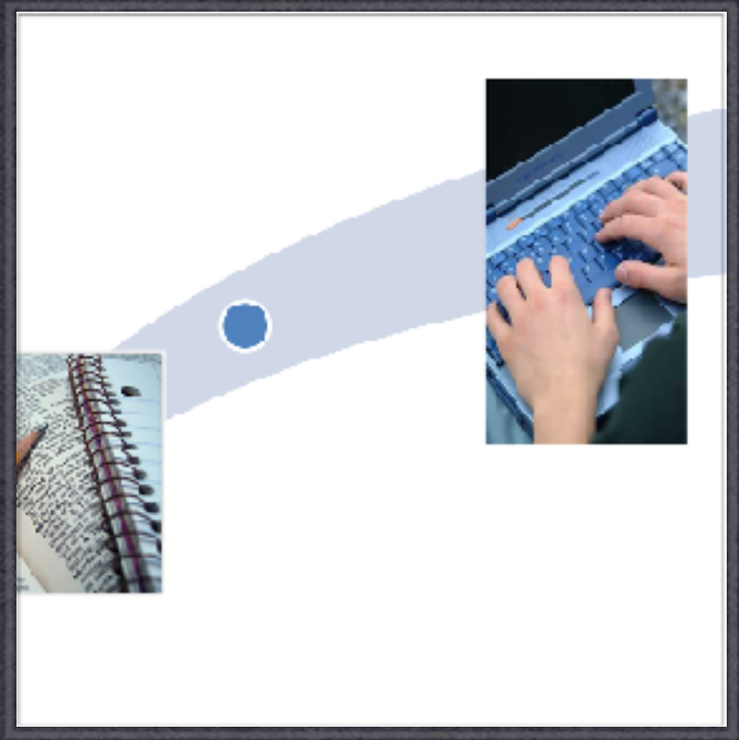
Herencia Multiple

- Es cuando una o varias clases pueden tener más de dos clase padre



- En el lenguaje de java no existe la herencia múltiple

Generalización - Especialización



- La generalización-especialización es la forma en cómo se va a analizar y diseñar la herencia. Es importante remarcar que NO es un tipo de herencia.

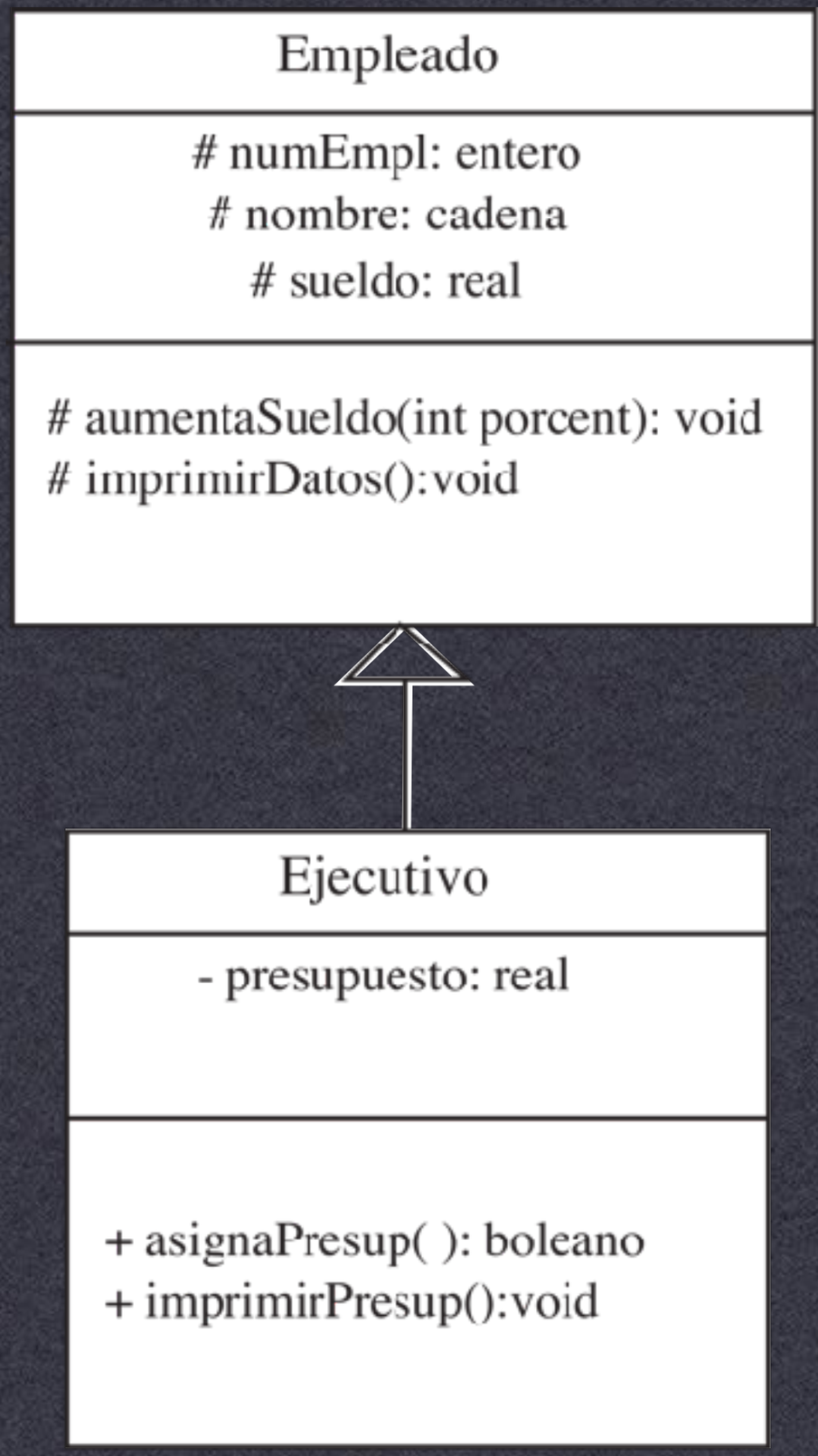
Generalización

- La generalización consiste en identificar las características iguales de los distintos objetos que quedaron en las subclases para formar y así generalizar a la superclase.

Especialización

- La especialización hace alusión al hecho consistente en que las subclases especializan a la superclase, se parte de tener las superclases para ir formando las subclases y que éstas especialicen a la superclase.

Ejemplo de Herencia



```
//definición clase padre o superclase
class Empleado{

    //declaración atributos
    protected String nombre;
    protected int numEmpl;
    protected double sueldo;
    private static int cont=0; //variable global estática
    //definición constructor
    protected Empleado(string n, double sueldo){
        nombre=n;
        this.sueldo=sueldo;
        numEmpl=++cont;
    }
    //definición métodos
    protected void aumentaSueldo(int porcent){
        sueldo+=(int)(sueldo*porcent/100);
    }
    protected void imprimirDatos( ){
        System.out.println("Num Empleado: "+numEmpl);
        System.out.println("Nombre: "+nombre);
        System.out.println("Sueldo: "+sueldo);
    }

}
```

```
//definición clase hijo o sublcase
class Ejecutivo extends Empleado{
    private double presupuesto; //declaración atributo
    //definición constructor
    public Ejecutivo (String n, double s, double presupuesto)
        super(n,s); //llamada del constructor de la superclase
        this.presupuesto=presupuesto;
    }
    //definición métodos
    private boolean asignaPresup( ){
        if (presupuesto>=50000)
            return false;
        else
            return true;
    }
    public void imprimirPresup(  ){
        super.imprimirDatos(  ); //llamada del método de la superclase
        System.out.println("Presupuesto:"+presupuesto);
        if  (asignaPresup( ))
            System.out.println("Presupuesto aprobado");
        else
            System.out.println("Presupuesto No aprobado");
    }
}
```

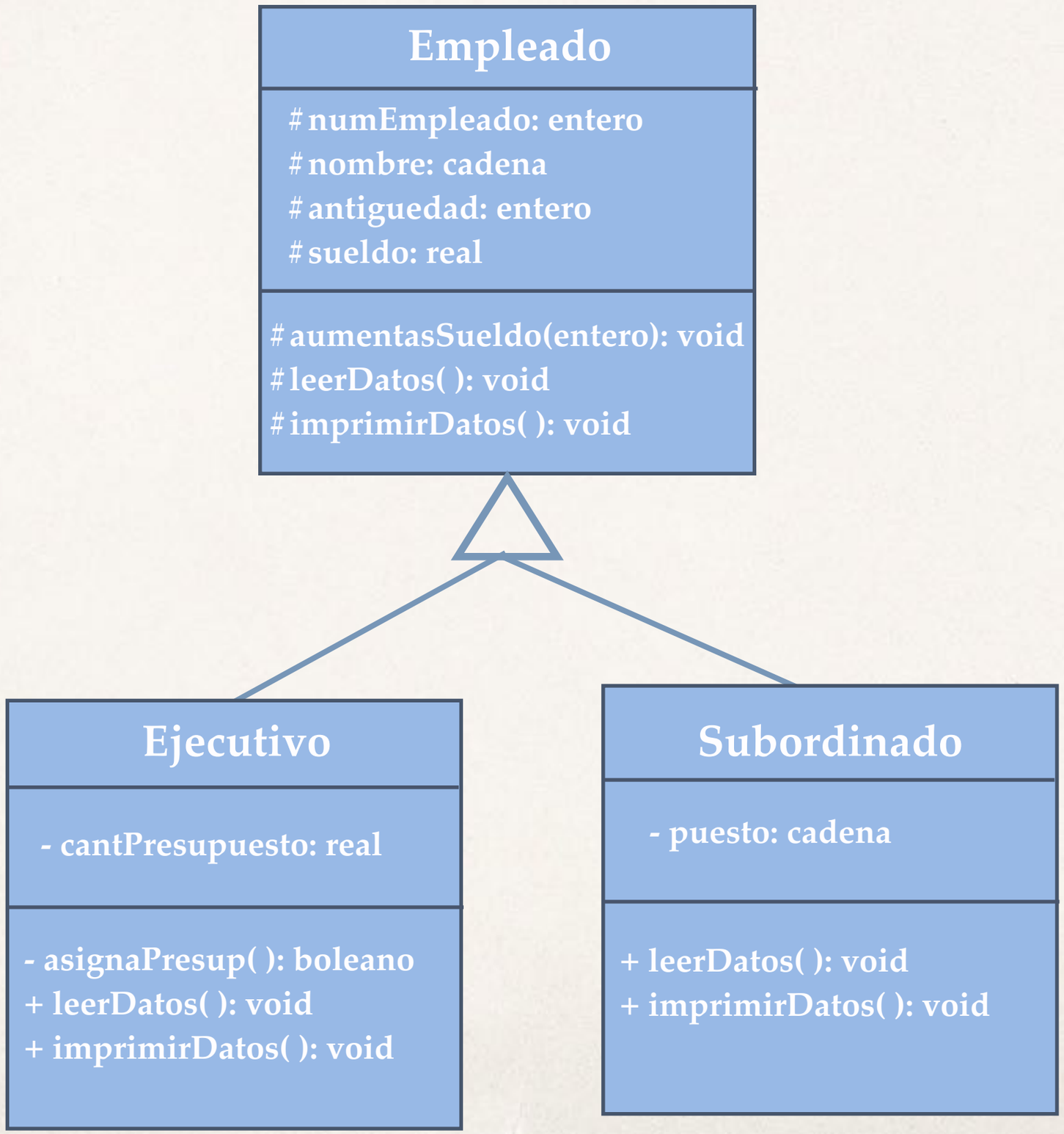
```
//implementación de la herencia

//definición de la clase que contiene a la función principal
class HerenciaApp{

    //definición función principal
    public static void main(String arg[]){
        //declaración variable referencia y creación del objeto
        Ejecutivo jefe=new Ejecutivo("Oscar Lasca", 10000, 25000); //instancia
        jefe.aumentaSueldo(5); //llamada del método (ejecución comportamiento obj)
        jefe.imprimirPresup( );
    }
}
```

Sobre-escritura de métodos

- Se declaran en **diferente clase**
- Se declaran con los **mismos parámetros**
- Tienen el **mismo identificador**
- Tienen **diferente código**

La sobre-escritura se utiliza en la herencia

Ejemplo de herencia con sobre-escritura de métodos



```java
//definición SuperClase en el archivo Empleado.java
class Empleado {
    //declaración atributos
    protected int numEmpl;
    protected String nombre;
    protected int antiguedad;
    protected double sueldo;
    protected static int cont=0;
    //definición métodos
    private void aumentaSueldo(int porcent) {
        sueldo+=(int)(sueldo*porcent/100);
    }
    protected void leerDatos(){
       //asignación de un valor al atributo desde teclado
        System.out.print("Dame tu numero de empleado: “);
        numEmpl=Leer.ent(); //método estático de clase Leer
        System.out.print("Dame tu nombre: ");
        nombre=Leer.cad(); //método estático de clase Leer
        System.out.print("Dime que antiguedad tienes: ");
        antiguedad=Leer.ent(); //método estático clase Leer
        System.out.print("Dime cual es tu sueldo: ");
        sueldo=Leer.real(); //método estático de clase Leer
    }
    protected void imprimirDatos(){
        System.out.println("Numero de Empleado: "+numEmpl);
        System.out.println("Nombre: "+nombre);
        if(antiguedad>10)
            aumentaSueldo(5);
        System.out.println("Sueldo Total: $"+sueldo);
    }
}
```

```java
//definición de subclase en el archivo Ejecutivo.java
class Ejecutivo extends Empleado {
    //declaración atributos
    private double cantPresupuesto;
    //definición métodos
    private boolean asignaPresup(){
        if(cantPresupuesto>=50000)
            return false;
        else
            return true;
    }
    public void leerDatos(){
        super.leeDatos();
        System.out.print("Presupuesto requerido: ");
        presupuesto=Leer.real();
    }
    public void imprimirDatos(){
        super.imprimeDatos();
        System.out.println("Presupuesto pedido: $"+cantPresupuesto);
        if(asignaPresup())
            System.out.println("Presupuesto Aprobado\n");
        else
            System.out.println("Presupuesto No Aprobado\n");
    }
}
```

```java
//definición de subclase en el archivo Subordinado.java
class Subordinado extends Empleado{
    //declaración atributos
    private String puesto;
    public void leerDatos(){
        super.leeDatos();
        System.out.print("Dime tu puesto: ");
        puesto=Leer.cad();
    }
    public void imprimirDatos(){
        super.imprimeDatos();
        System.out.println("Puesto del Empleado: "+puesto+"\n");
    }
}
```

```java
//definición de la clase que contiene al main en el archivo
SobreEscritura.java
public class SobreEscritura {
    //definición de la función principal
    public static void main(String[] args) {
        // declaracion variable referencia y creación de objetos
        Ejecutivo ej=new Ejecutivo();
        Subordinado s=new Subordinado();
        //llamada de los metodos sobreescritos
        ej.leerDatos();
        ej.imprimirDatos();
        System.out.println();
        s.leerDatos();
        s.imprimirDatos();
    }

}
```

Analicemos la clase Leer

Es la misma clase de DatosEntrada pero esta clase se llama Leer; recuerda que es una clase que no se encuentra en el lenguaje de Java por lo que nosotros mismos la creamos y le podemos poner el nombre que queramos a la clase y a sus métodos estáticos

```
protected void leerDatos(){
      //asignación de un valor al atributo desde teclado
       System.out.print("Dame tu numero de empleado: “);
       numEmpl=Leer.ent(); //método estático de clase Leer
       System.out.print("Dame tu nombre: ");
       nombre=Leer.cad(); //método estático de clase Leer
       System.out.print("Dime que antiguedad tienes: ");
       antiguedad=Leer.ent(); //método estático clase Leer
       System.out.print("Dime cual es tu sueldo: ");
       sueldo=Leer.real(); //método estático de clase Leer
}
```

En este ejemplo el método leerDatos( ) manda llamar al método estático ent( ) de la clase Leer que debe estar guardada en la misma carpeta de nuestros archivos Empleado.java, Ejecutivo.java, Subordinado.java y SobreEscritura.java

A continuación se escribe el código del archivo **Leer.java** que si lo comparas con el archivo **DatosEntrada.java** es el mismo

Archivo: Leer.java

```
import java.io.*;

class Leer {

  public static String cad(){
     try{
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        return br.readLine();
     }catch(Exception e){ return " "; }
  }
  public static int ent(){
     try{
        return Integer.parseInt(cad());
     }catch(Exception e){ return 0; }
  }
  public static double real(){
     try{
        return Double.parseDouble(cad());
     }catch(Exception e){ return 0.0; }
  }
  public static char caracter(){
     try{
        return cad().charAt(0);
     }catch(Exception e){ return ' '; }
  }

}
```

Ejercicio para la actividad 2

Problemática

- Se quiere imprimir en pantalla el desglose de la venta de un producto. Se requiere que en pantalla también se impriman los siguientes datos: código del producto, descripción del producto, cantidad a comprar del producto, precio unitario (tomando en cuenta que el precio unitario ya trae el iva incluido) así como los siguientes procesos: el subtotal, el iva y total.

- Se debe considerar que si el producto vendido es perecedero, en la pantalla se imprimirá (a parte de lo ya mencionado) la fecha de caducidad como dato y como proceso se tendrá que determinar si ya está caduco o no a través de un mensaje en la pantalla.

- Por otro lado, si el producto es no perecedero, en la pantalla se imprimirá (a parte de lo ya mencionado) el proceso de determinar un descuento del 7% en caso de que se hayan vendido más de 5 del mismo producto.

- El programa debe preguntar a través de un menú si se va hacer la venta de un producto perecedero o de un producto no perecedero. También se debe considerar que al final del programa se pregunte si quiere salir del programa en caso de que la respuesta sea no el programa debe regresar al menú.