Universidad Continental

UNIDAD 01
Fundamentos

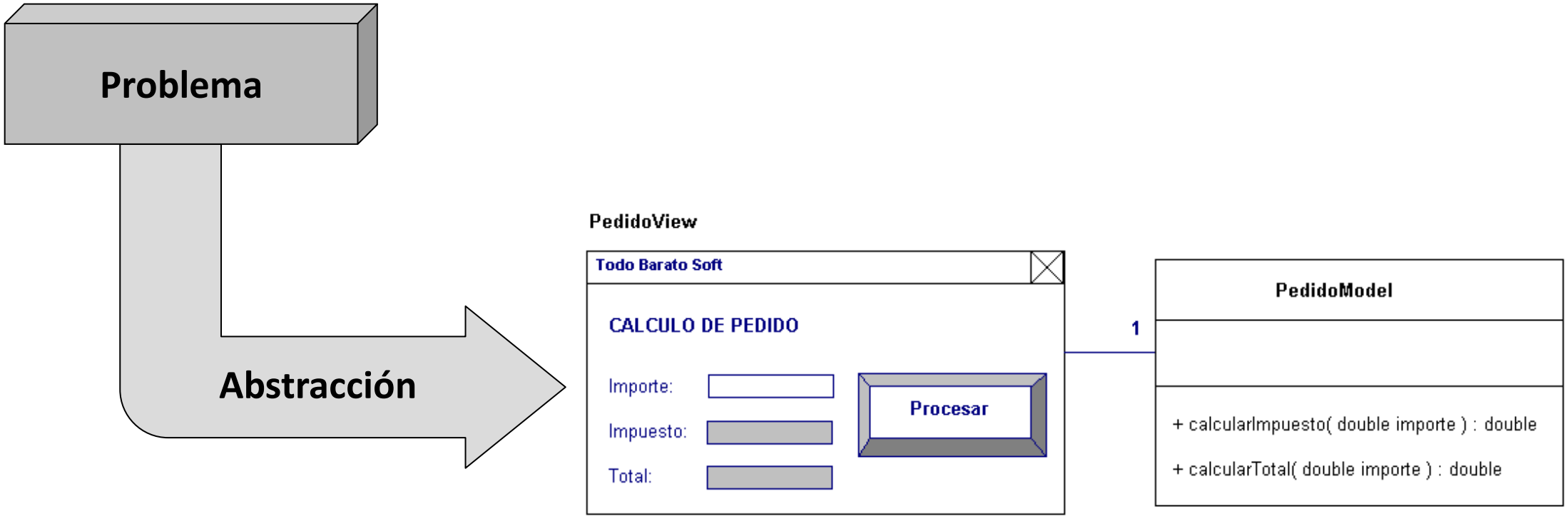
PROGRAMACIÓN ORIENTADA A OBJETOS

Eric Gustavo Coronel Castillo
ecoronel@continental.edu.pe

CLASES Y OBJETOS

OBJETIVO

Entender los conceptos de Clase y Objeto, y su aplicación en la solución de problemas sencillos.



ABSTRACCIÓN

Consiste en capturar, percibir y clasificar las características (datos-atributos) y comportamientos (operaciones) necesarias (relevantes) del mundo real (proceso a sistematizar) para dar solución al problema.

Notación UML

Abstracción

Persona
+ Nombre : String + Edad : Integer + Profesion : String
+ Caminar() + Correr() + Cantar() : String

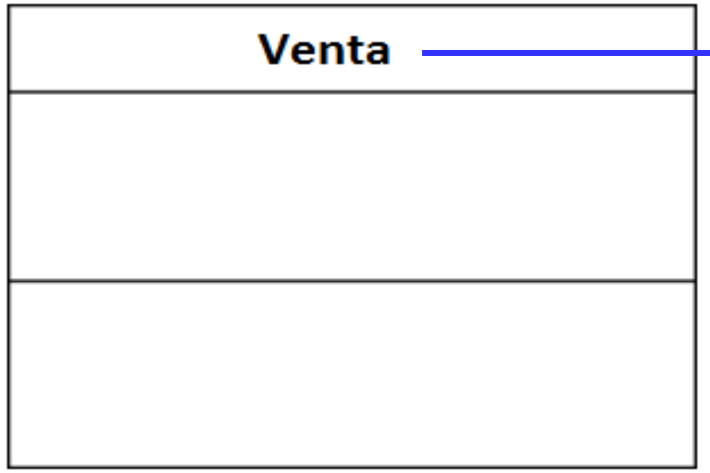
Animal
+ Raza : String + Genero : String
+ Comer()

Transporte
+ Tipo : String + Marca : String + Año : Integer
+ Encender() : Boolean + Acelerar(Velocidad : Integer)

DEFINICIÓN DE CLASE Y OBJETO

CLASE

- Una clase define un tipo de objeto en particular.
- Por ejemplo, la clase Empleado define a todos los trabajadores de una empresa.



Nombre de la Clase

Ejemplos de Nombres de Clase

- Cliente
- Factura
- NotaCredito
- Guia
- Pedido
- Matricula
- CuentaMaestra

DEFINICIÓN DE CLASE Y OBJETO

OBJETO

- Un objeto es una instancia de una clase.
- Por ejemplo, cada trabajador de una empresa es una instancia de la clase Empleado.

DEFINICIÓN DE CLASE Y OBJETO

Notación UML de OBJETO

objPersona1 : Persona
+ Nombre : Jennifer + Edad : 34 + Profesion : Cantante
+ Caminar() + Correr() + Cantar():String

objPersona2 : Persona
+ Nombre : Zidane + Edad : 40 + Profesion : Futbolista
+ Caminar() + Correr() + Cantar():String

Instancia

Instancia

Notación UML de Clase

Persona
+ Nombre : String + Edad : Integer + Profesion : String
+ Caminar() + Correr() + Cantar() : String

IMPLEMENTACIÓN DE CLASES

SINTAXIS

```
public class NombreClase {

   // Definición de variables

   // Definición de métodos

}
```

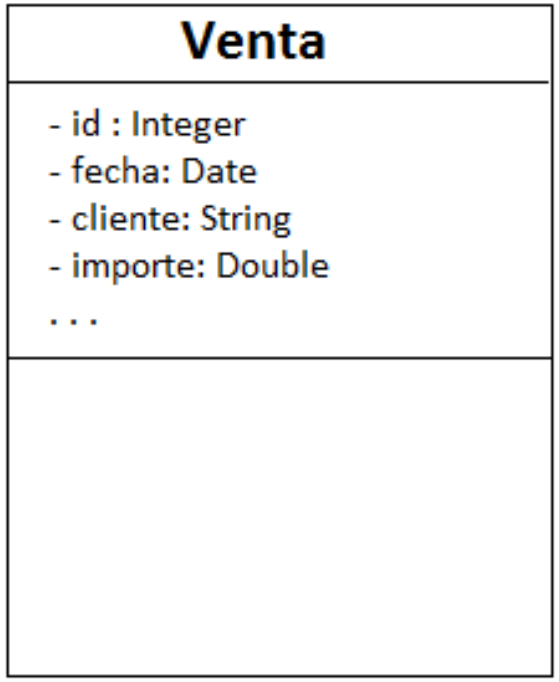
El nombre del archivo debe tener el mismo nombre de la clase.

Por ejemplo, si la clase se llama **Producto** el nombre del archivo que contiene a la clase se debe llamar **Producto.java**.

IMPLEMENTACIÓN DE CLASES

ATRIBUTOS

- Representa un dato del objeto.
- Cada atributo de un objeto tiene un valor que pertenece a un dominio de valores determinado.
- En Java se implementan creando variables a nivel de clase.

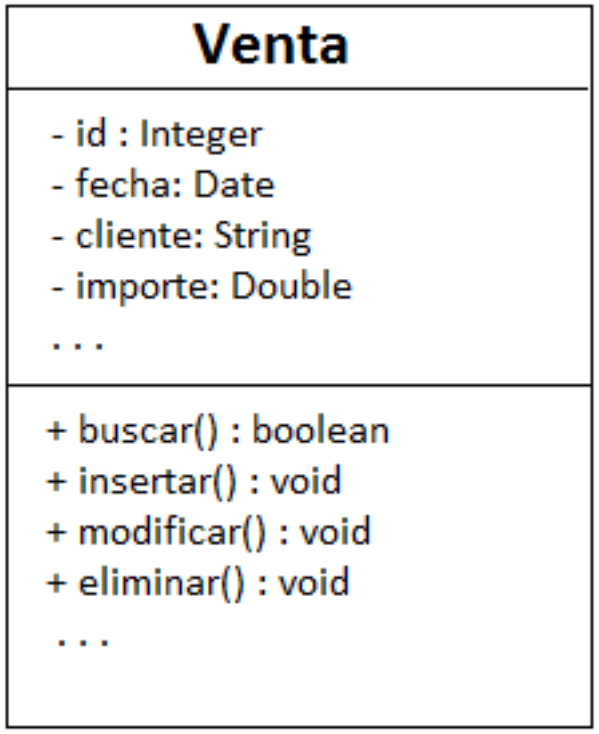



```
public class Venta {

    // Variables que implementación de atributos
    private Integer id;
    private Date fecha;
    private String cliente;
    private Double importe;



}
```

IMPLEMENTACIÓN DE CLASES

OPERACIONES

- Son servicios proporcionado por un objeto que pueden ser solicitados por otros objetos.
- Determinan el comportamiento del objeto.
- La implementación en Java se realiza mediante métodos.




```
public class Venta {

    // Implementación de atributos
    private Integer id;
    . . .

    // Implementación de operaciones
    public boolean buscar( . . . ) {
        . . .
        . . .
    }

    . . .

}
```

IMPLEMENTACIÓN DE CLASES

DEFINICIÓN DE MÉTODOS

```
public <tipo> nombreMétodo ( [ parámetros ] ) {

   // Implementación

   [ return valorRetorno; ]
}
```

<tipo> Determina el tipo de dato que retorna el método, si no retorna ningún valor se utiliza **void**.

return Esta sentencia finaliza la ejecución del método, se acompaña de un valor cuando el método debe retornar un resultado.

CREACIÓN Y USO DE OBJETOS

OPERADOR NEW

```
NombreClase variable = new NombreClase();

ó

NombreClase variable = null;
variable = new NombreClase();
```

ACCESO A LOS MÉTODOS

```
variable.nombreMétodo ( … )
```

PROYECTO EJEMPLO

ENUNCIADO

La empresa "Todo Barato" necesita facilitar la elaboración de los pedidos que realizan sus empleados a sus proveedores, el problema radica al momento de calcular el impuesto.

La empresa ha solicitado a su departamento de sistemas elaborar un programa en Java que permita ingresar el importe del pedido, y calcule el impuesto y el total que se debe pagar al proveedor.

MIEMBROS DE CLASE

DECLARACIÓN DE VARIABLES

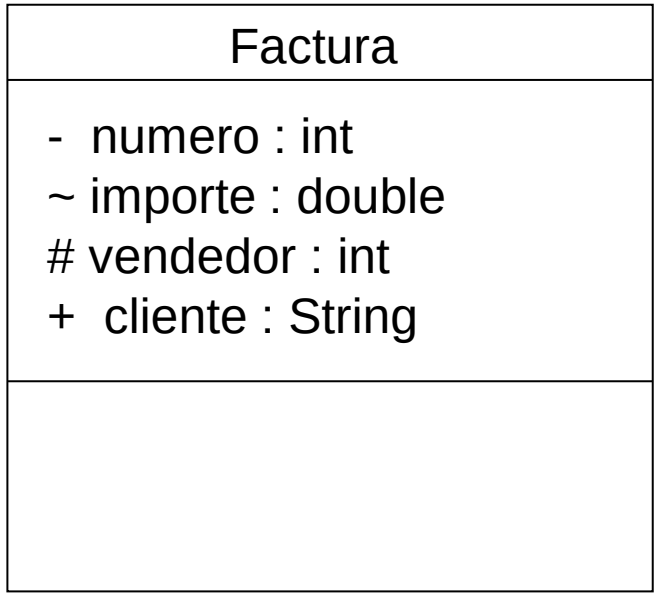
Sintaxis:

```
[modificadorAcceso] tipo nombreVariable [ = valor ] ;
```

El **modificadorAcceso** puede ser:

- privado (private)
- paquete
- protegido (protected)
- público (public)

```
public class Factura{

  private int numero = 54687;
  double importe = 5467.87;
  protected int vendedor = 528;
  public String cliente = "Banco de Crédito";

}
```



DECLARACIÓN DE VARIABLES

paquete: uno

```
public class ClaseA {
   private int n1;  // privada
   int n2;  // paquete
   protected int n3;  // protegida
   public int n4;  // publica

   public void metodoA ( ) {

   }

}

public class ClaseB {

   public void metodoB ( ) {

   }

}
```

paquete: dos

```
public class ClaseC extends ClaseA {

   public void metodoC ( ) {

   }

}

public class ClaseD {

   public void metodoD ( ) {

   }

}
```

DECLARACIÓN DE VARIABLES

Todas las variables se encuentran declaradas en **ClaseA**.

```
private int n1;  // privada
int n2;  // paquete
protected int n3;  // protegida
public int n4;  // publica
```

El siguiente cuadro explica el acceso a estas variables por métodos de diferentes clases.

Paquete	Clase	Métodos	Variables a las que tiene acceso			
			n1	n2	n3	n4
uno	ClaseA	metodoA	Si	Si	Si	Si
uno	ClaseB	metodoB	No	Si	Si	Si
dos	ClaseC	metodoC	No	No	Si	Si
dos	ClaseD	metodoD	No	No	No	Si

DECLARACIÓN DE MÉTODOS

Sintaxis:

```
[modificadorAcceso] tipo nombreMétodo ( [ parámetros ] ) {

   // Implementación

}
```

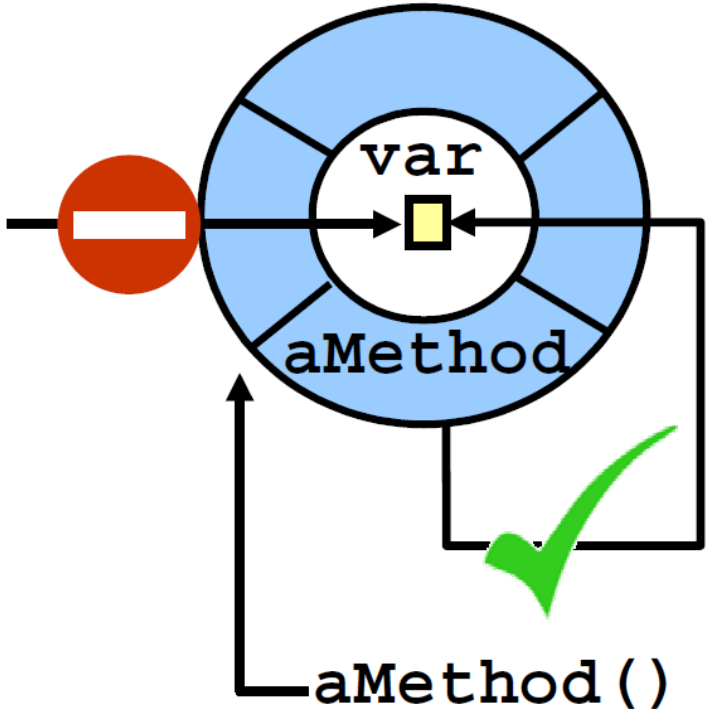
El modificadorAcceso puede ser:

- privado (private)
- paquete
- protegido (protected)
- público (public)

ENCAPSULACIÓN

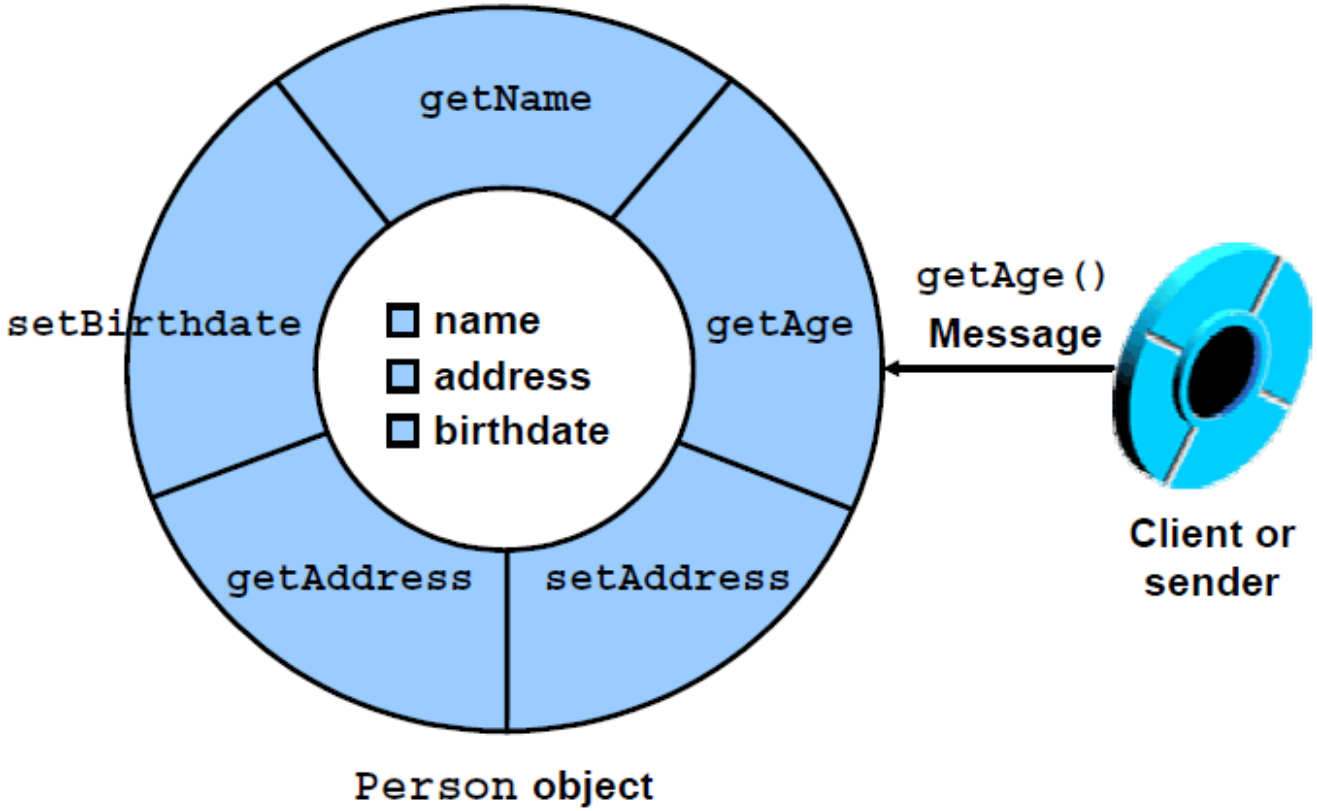
Características

- Las variables de instancia deben ser declaras como privadas.
- Los métodos de instancia sólo puede acceder a las variables de instancia privadas.



ENCAPSULACIÓN

Implementación



ENCAPSULACIÓN

Implementación

- Variable

```
private tipo variable[ = valor ] ;
```

- Método set

```
public void setVariable( tipo valor ) {
       this.variable = valor;
}
```

- Método get

```
public tipo getVariable() {
       return this.variable;
}
```

En caso que la propiedad sea de tipo **boolean** se utiliza **isPropiedad** en lugar de **getPropiedad**.

CONSTRUCTOR

Se utiliza para inicializar el objeto.

```
public class NombreClase {

    public NombreClase() {

      // Inicialización del objeto

    }

}
```

DESTRUCTOR

Se utiliza para liberar los recursos que el objeto está utilizando.

```
public class NombreClase {

  protected void finalize() throws Throwable {

        // Liberar recursos del objeto

  }

}
```

PROYECTOS EJEMPLO

ENUNCIADO

La empresa **Vía Éxitos** Necesita saber cuanto se le debe pagar a sus trabajadores y a cuanto asciende el importe del impuesto a la renta que debe retener.

Los datos son:

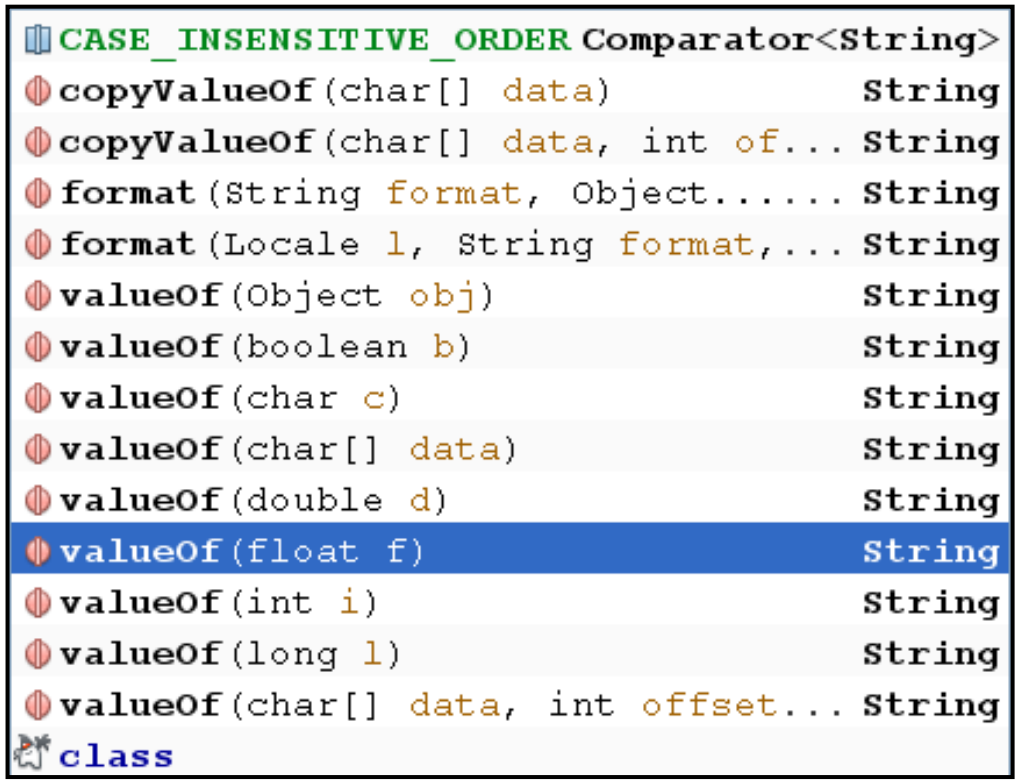
- Cantidad diaria de horas trabajadas.
- Cantidad de días trabajados.
- El pago por hora.

Se sabe que si los ingresos supera los 1500.00 Nuevos Soles, se debe retener el 8% de los ingresos correspondiente al impuesto a la renta.

SOBRECARGA

OBJETIVO

Aplicar la sobrecarga para disponer de diversas versiones de métodos y constructores que se puedan aplicar dependiendo de las necesidades que se tengan o se proyecten tener.



FIRMA DE UN METODO

- La firma de un método esta definida por:
  - Nombre del método
  - Parámetros del método
    - La cantidad de parámetros
    - El tipo de dato de sus parámetros
    - Orden de los parámetros

- En una clase no puede existir dos métodos con la misma firma.

FIRMA DE UN METODO

```
public class DemoService {

    public void procesar(double value1, int value2){

    }

    public void procesar(double value1, int value2){

    }

}
```

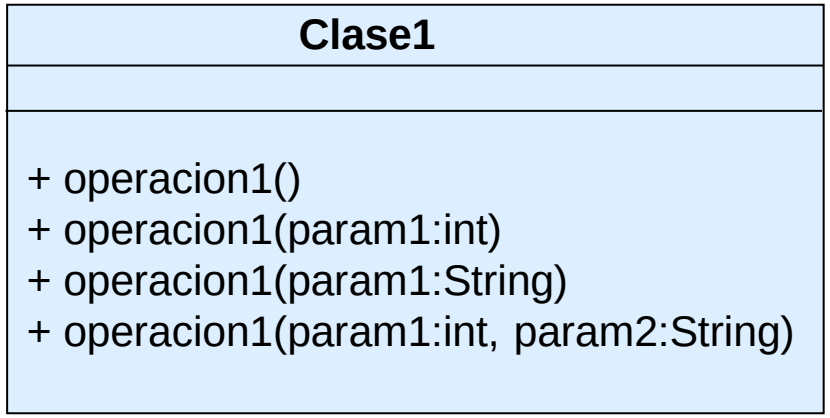
method procesar(double,int) is already defined in class DemoService
---
(Alt-Enter shows hints)

SOBRECARGA DE MÉTODOS

- Es la implementación de varios métodos con el mismo nombre, pero que se diferencian en:
  - La cantidad de parámetros
  - El tipo de dato de sus parámetros
  - Orden de los parámetros
- Por lo tanto podemos afirmar que los métodos tienen diferente firma.

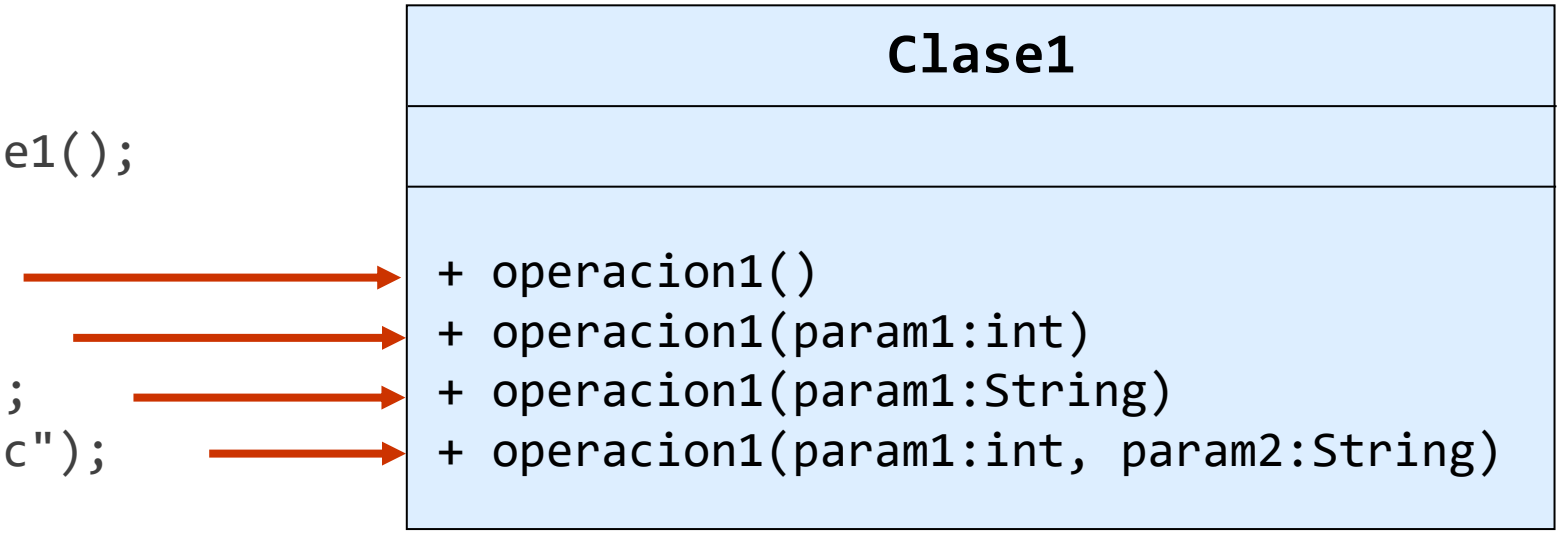
SOBRECARGA DE MÉTODOS

```
public class Clase1 {

	public void operacion1() {
		// Implementación
	}

	public void operacion1(int param1) {
		// Implementación
	}

	public void operacion1(String param1) {
		// Implementación
	}

	public void operacion1(int param1, String param2) {
		// Implementación
	}

}
```



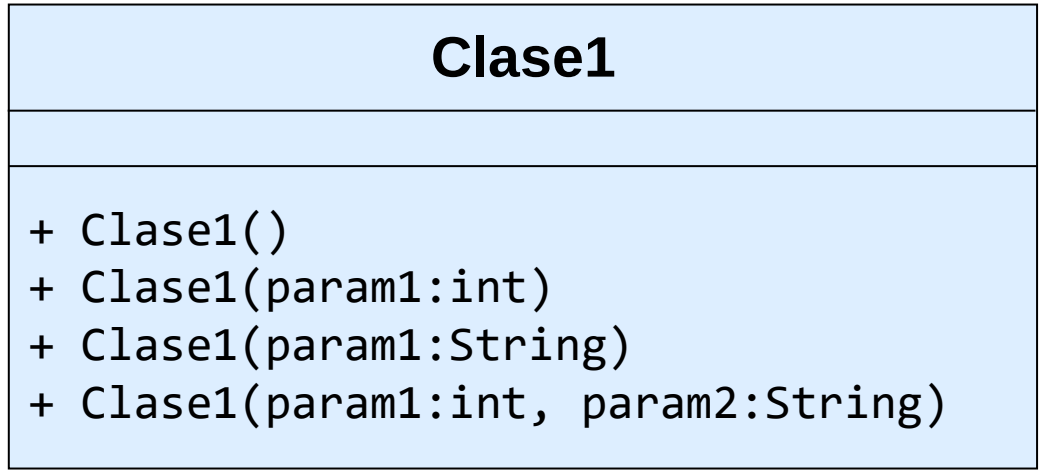
SOBRECARGA DE MÉTODOS

```
Clase1 obj = new Clase1();

obj.operacion1();
obj.operacion1(15);
obj.operacion1("abc");
obj.operacion1(15,"abc");
```



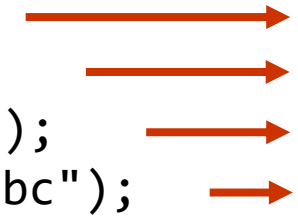
SOBRECARGA DE CONSTRUCTORES

```
public class Clase1 {

    public Clase1(){
        // implementación
    }

    public Clase1(int param1){
        // implementación
    }

    public Clase1(String param1){
        // implementación
    }

    public Clase1(int param1, String param2){
        // implementación
    }

}
```



SOBRECARGA DE CONSTRUCTORES

```
Clase1 obj = new Clase1();
Clase1 obj = new Clase1(20);
Clase1 obj = new Clase1("abc");
Clase1 obj = new Clase1(20,"abc");
```



Clase1
+ Clase1() + Clase1(param1:int) + Clase1(param1:String) + Clase1(param1:int, param2:String)

PROYECTO EJEMPLO

ENUNCIADO

La empresa **EduTec** necesita de una librería que permita calcular el promedio de un conjunto de números.

Se sabe que pueden ser 2, 3, 4 o 5 números.

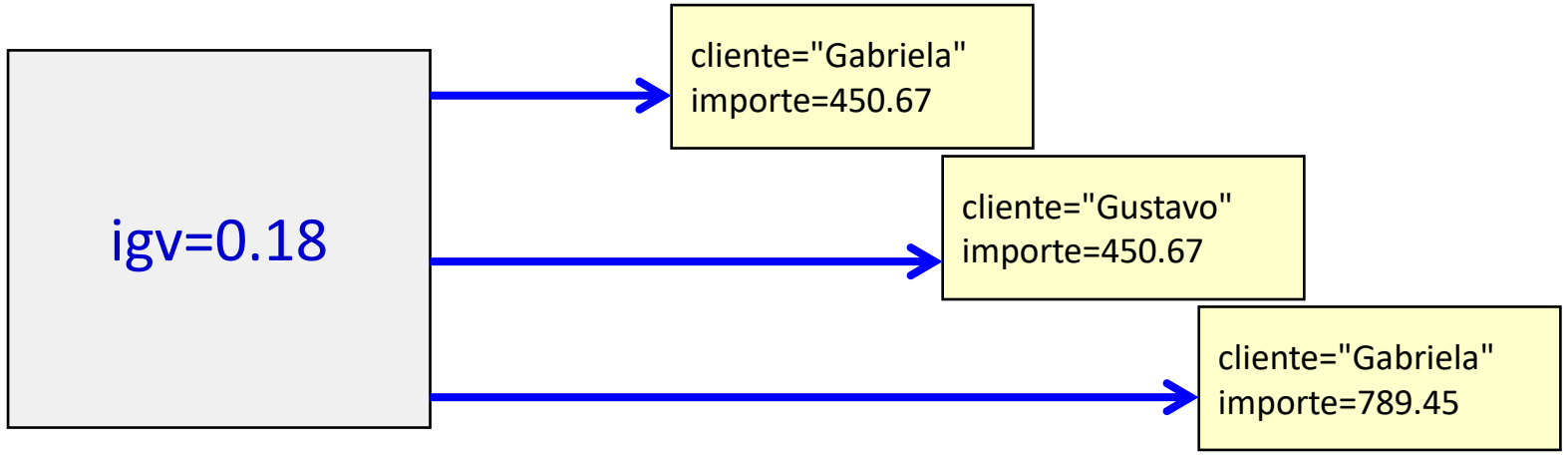
A usted se le ha encargado que desarrolle la librería que necesita Edutec y construya una aplicación de prueba.

VARIABLES Y MÉTODOS DE INSTANCIA Y DE CLASE

OBJETIVO

- Entender la diferencia entre variables y métodos de instancia y de clase.
- Aplicar correctamente variables y métodos de instancia y de clase.

```
public class Venta{
    private static double igv = 0.18;          // Variable de clase
    private String cliente;                    // Variable de instancia
    private double importe;                    // Variable de instancia
    . . .
}
```



Variable de la clase **Venta**

Objetos de tipo **Venta** tienen sus propias variables

DEFINICIONES

VARIABLES Y METODOS DE CLASE

- Se trata de variables y métodos que no requieren crear una instancia (objeto) para ser invocados, basta con anteponer el nombre de la clase para poder acceder a ellos.
  - NombreClase.variable
  - NombreClase.método(...)

VARIABLES Y METODOS DE INSTANCIA

- Se trata de variables y métodos que se crean en el objeto y por lo tanto para ser invocados se necesita una instancia de la clase, es decir, un objeto.
  - objeto.variable
  - objeto.método(...)

IMPLEMENTACIÓN

VARIABLES

[visibilidad] [**static**] tipo campo [=valor];

MÉTODOS

```
[visibilidad] [static] tipo nombreMétodo( … ) {

    // Implementación

    [return valor;]
}
```

La presencia de la palabra **static** determina que la declaración es de la clase y no de la instancia.

ACCESO A LAS VARIABLES Y MÉTODOS

DE INSTANCIA

- **Desde la misma clase**

  this.variable
  this.método(...)

- **Desde fuera de la clase**

  objeto.variable
  objeto.método(...)

DE CLASE

- **Desde la misma clase**

  variable
  método(...)

- **Desde fuera de la clase**

  NombreClase.variable
  NombreClase.método(...)

INICIALIZADOR ESTÁTICO

```
public class NombreClase {

    . . .
    . . .
    static {
      . . .
      . . .
    }
    . . .
    . . .

}
```

Se utiliza para inicializar variables de clase, similar a un constructor que se utiliza para inicializar las variables de instancia.

PROYECTO EJEMPLO

ENUNCIADO

El colegio "Ángeles del Cielo" esta solicitando un programa en Java para que los alumnos de primaria verifiquen sus ejercicios de matemáticas referidos a:

- Calculo de factorial
- Calculo del MCD y MCM de dos números
- La serie de Fibonacci
- Número primo

La programación de estos cálculos matemáticos deben estar implementados como métodos de clase en una clase de nombre **MyMath**.

HERENCIA

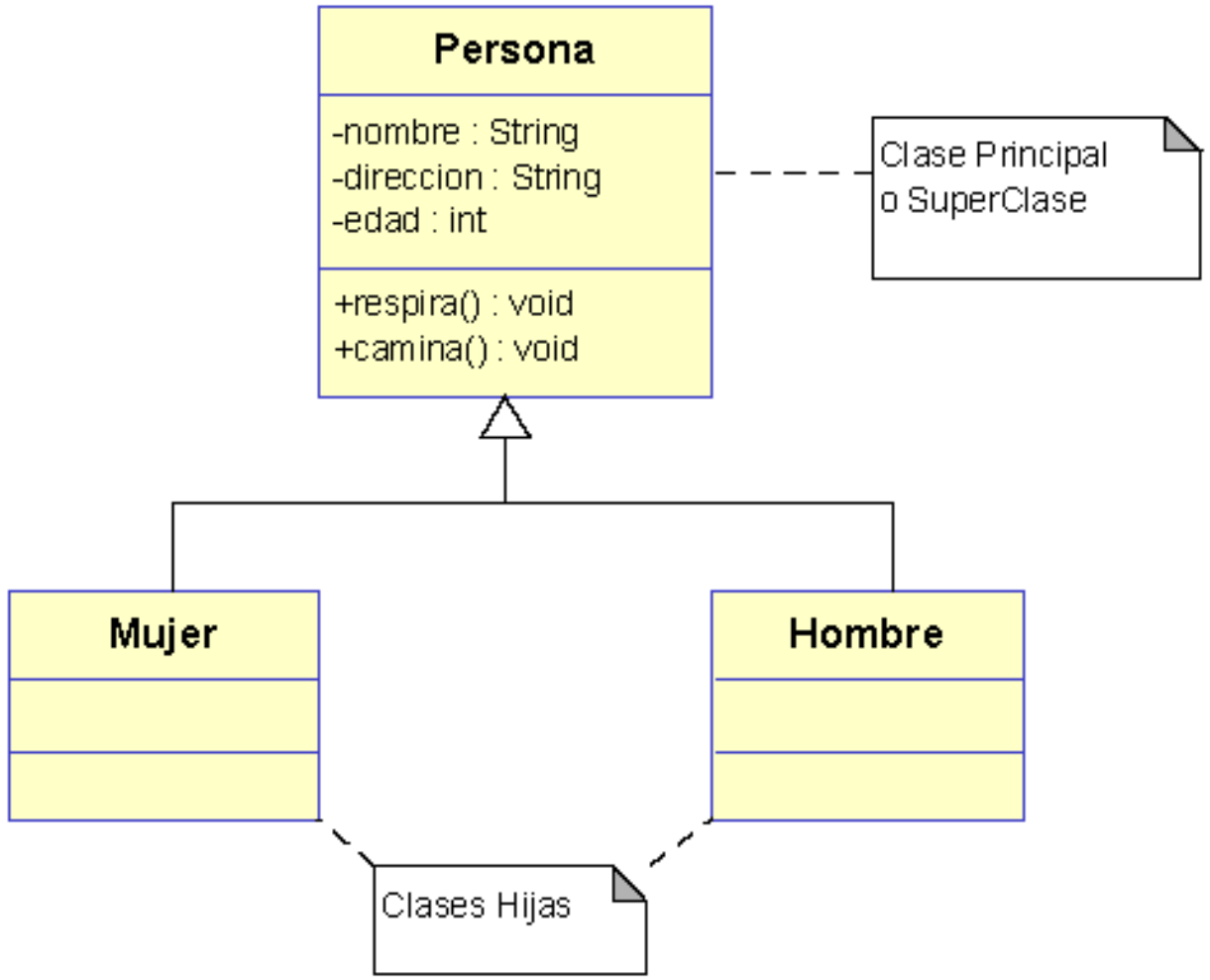
OBJETIVO

Aplicar la herencia para:

- Reutilizar código.
- Extender la funcionalidad de clases (Especialización).
- Aprovechar el polimorfismo.

De esta manera:

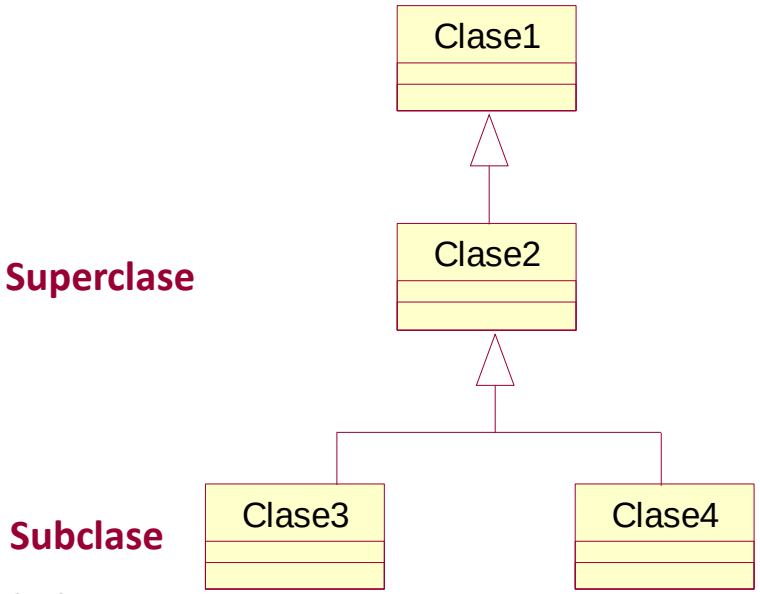
- Mejoramos la productividad.
- Disminuimos el esfuerzo de mantenimiento.
- Aumentamos la fiabilidad y eficiencia.



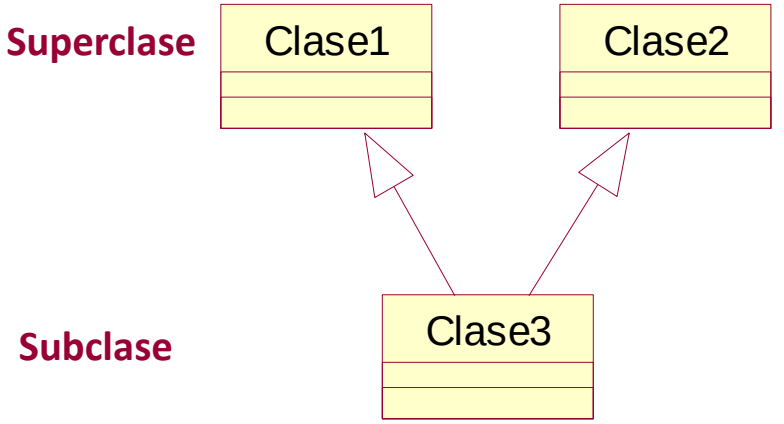
DEFINICIÓN

- La herencia es el mecanismo mediante el cual podemos definir una clase (**Subclase**) en función de otra ya existe (**Superclase**).
- Las subclases heredan los atributos y operaciones de sus superclases.
- Existen dos tipos de herencia (simple y múltiple)

Herencia Simple



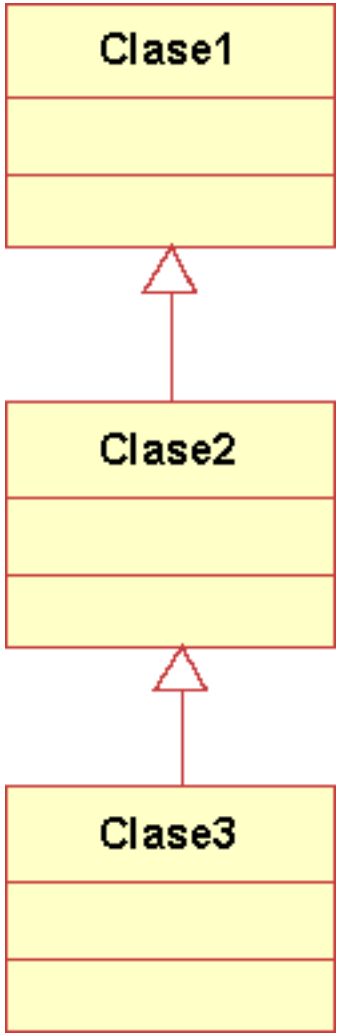
Herencia Múltiple



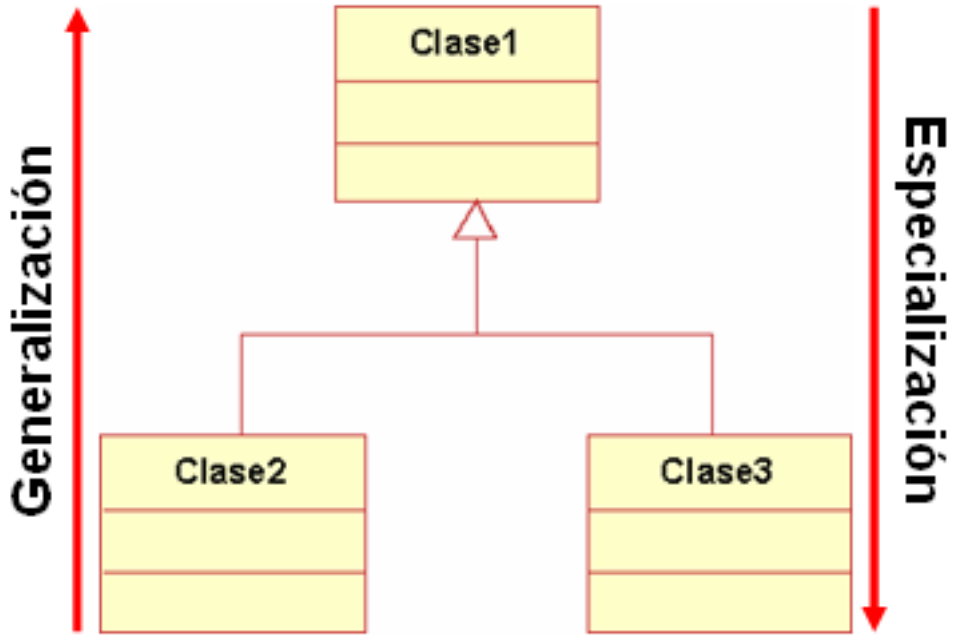
No se puede implementar la herencia múltiple en Java.

CARACTERÍSTICAS

- Si **Clase2** hereda de **Clase1**, entonces **Clase2** incorpora la estructura (atributos) y comportamiento (métodos) de **Clase1**, pero puede incluir adaptaciones:
  - Clase2 puede añadir nuevos atributos.
  - Clase2 puede añadir nuevos métodos.
  - Clase2 puede redefinir métodos heredados (refinar o reemplazar).
- La herencia es transitiva
  - Clase2 hereda de Clase1 (Clase1 es la superclase y Clase2 la subclase)
  - Clase3 hereda de Clase2 y Clase1
  - Clase2 y Clase3 son subclases de Clase1
  - Clase2 es un descendiente directo de Clase1
  - Clase3 es un descendiente indirecto de Clase1



DISEÑO



No hay receta mágica para crear buenas jerarquías de herencia.

- Generalización (Factorización): Se detectan dos clases con características comunes y se crea una clase padre con esas características.
  - Ejemplo: Libro, Revista → Publicación
- Especialización: Se detecta que una clase es un caso especial de otra.
  - Ejemplo: Rectángulo es un tipo de Polígono.

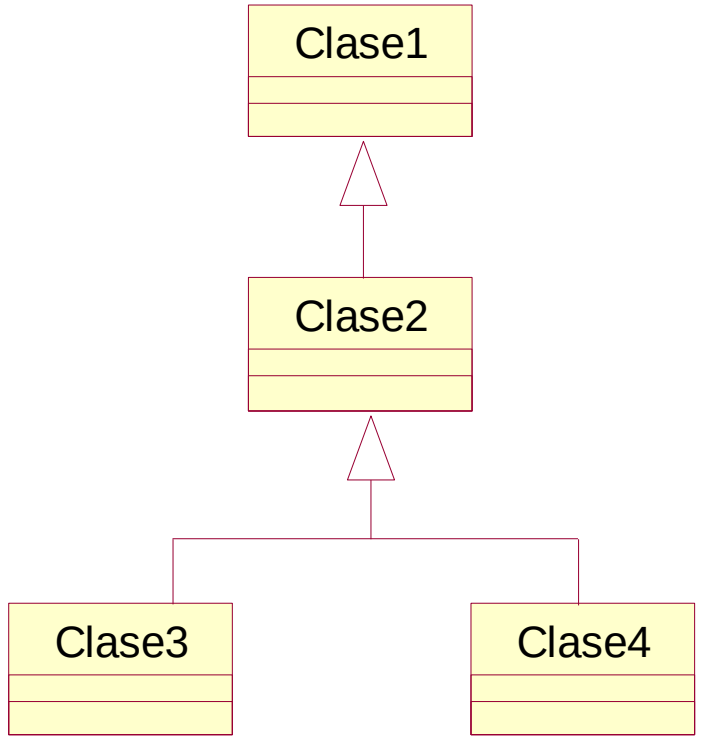
IMPLEMENTACIÓN

```
public class Clase1 {

}
```

```
public class Clase2 extends Clase1 {

}
```

```
public class Clase3 extends Clase2 {

}
```

```
public class Clase4 extends Clase2 {

}
```



Recuerde usar:
this: referencia a métodos del objeto actual.
super: referencia a métodos de la superclase.

HERENCIA Y CONSTRUCTORES

- En Java, los constructores no se heredan.
- Java permite invocar a los constructores de la clase padre dentro de un constructor utilizando la llamada **super(…)**.
- Cuando se aplica herencia, la llamada a un constructor de la clase padre es obligatoria.
- Debe ser la primera sentencia del código del constructor.
- Si se omite la llamada, el compilador asume que la primera llamada es **super()**.

```
public class Clase2 extend Clase1 {

    public Clase2() {
        super();
        . . .
    }

}
```

REDEFINICIÓN

- La redefinición reconcilia la reutilización con la extensibilidad.
- Las **variables** no se pueden redefinir, sólo se ocultan
  - Si la clase hija define una variable con el mismo nombre que un variable de la clase padre, éste no está accesible.
  - La variable de la superclase todavía existe pero no se puede acceder
- Un **método** de la subclase con la misma firma (nombre y parámetros) que un método de la superclase lo está redefiniendo.
  - Si se cambia el tipo de los parámetros se está sobrecargando el método original.
- Si un método redefinido refina el comportamiento del método original puede necesitar hacer referencia a este comportamiento.
  - **super:** se utiliza para invocar a un método de la clase padre:
    - **super.metodo (…) ;**

MODIFICADOR `final`

- Aplicado a una variable lo convierte en una constante.

```
protected final String NOMBRE= "Gustavo Coronel" ;
```

- Aplicado a un método impide su redefinición en una clase hija.

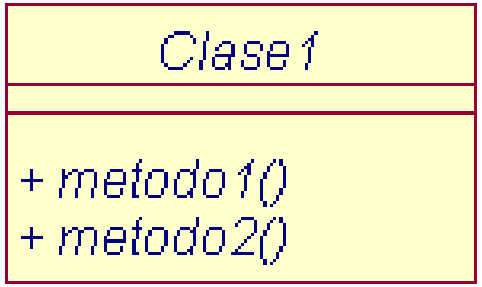
```
public final int suma( int a, int b ) { … }
```

- Aplicado a una clase indica que no se puede heredar.

```
public final class Clase1 {
  . . .
}
```

CLASES ABSTRACTAS

- Una clase abstracta define un tipo, como cualquier otra clase.
- Sin embargo, no se pueden construir objetos de una clase abstracta.
- Los constructores sólo tienen sentido para ser utilizados en las subclases.

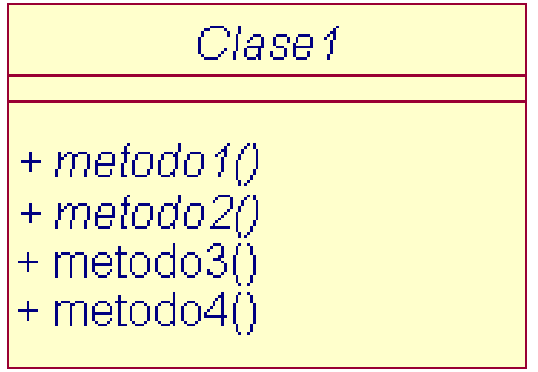


- ❖ Especifica una funcionalidad que es común a un conjunto de subclases aunque no es completa.
- ❖ Justificación de una clase abstracta:
  - Declara o hereda métodos abstractos.
  - Representa un concepto abstracto para el que no tiene sentido crear objetos.

```
public abstract class Clase1 {

    public abstract void metodo1();
    public abstract void metodo2();

}
```

CLASES PARCIALMENTE ABSTRACTAS

- Contienen métodos abstractos y concretos.
- Los métodos concretos pueden hacer uso de los métodos abstractos.
- Importante mecanismo para incluir código genérico.
- Incluyen comportamiento abstracto común a todos los descendientes.




```
public abstract class Clase1 {

    public abstract void metodo1();
    public abstract void metodo2();

    public void metodo3() {
        . . .
    }

    public void metodo4() {
        . . .
    }

}
```

OPERADOR `instanceof`

- Comprueba si el tipo de una variable es compatible con un tipo dado.
  - Es de ese tipo o alguna de sus subclases
- Si no se hace la comprobación, en el caso de que fallara el casting (en tiempo de ejecución) se abortaría el programa.
- No es lo mismo hacer la comprobación con instanceof que con el método getClass heredado de la clase.

```
if ( variable instanceof Clase ) {

    // Script

}
```

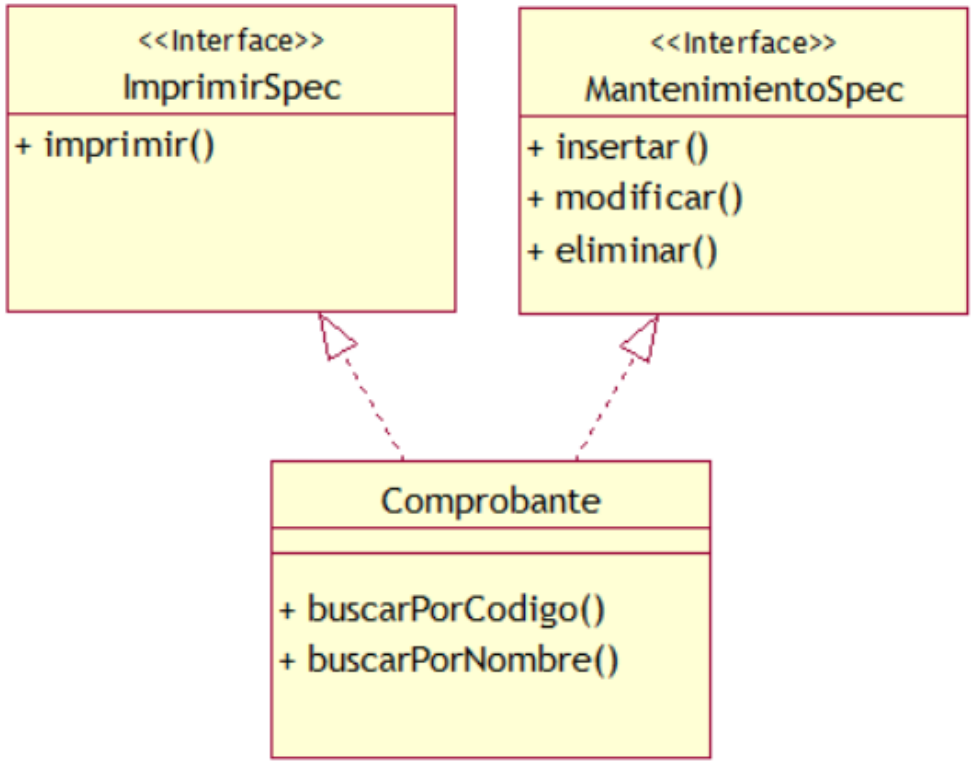
PROYECTO EJEMPLO

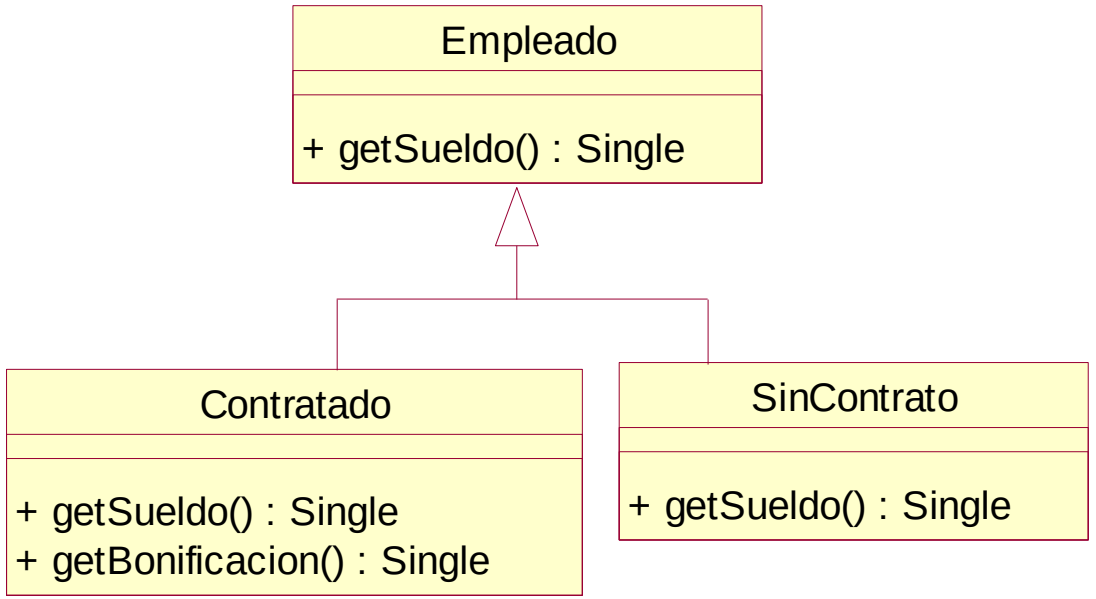
- El restaurante "El Buen Sabor" necesita implementar una aplicación que permita a sus empleados calcular los datos que se deben registrar en el comprobante de pago.
- Los conceptos que se manejan cuando se trata de una factura son los siguientes:
  - Consumo 100.00
  - Impuesto 19.00
  - Total 119.00
  - Servicio (10%) 11.90
  - Total General 130.90
- Cuando se trata de una boleta son los siguientes:
  - Total 119.00
  - Servicio (10%) 11.90
  - Total General 130.90
- Diseñe y desarrolle la aplicación que automatice el requerimiento solicitado por el restaurante.
- Se sabe que el dato que debe proporcionar el empleado es el **Total**.

INTERFACES

OBJETIVOS

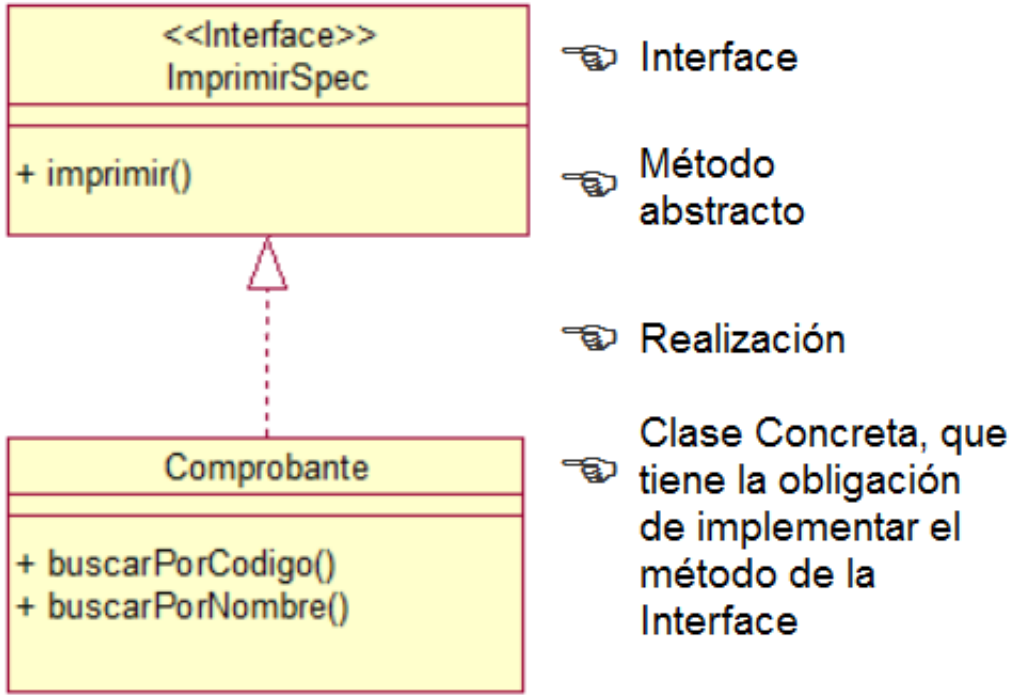
- Aplicar interfaces en el diseño de componentes software.
- Aplicar el polimorfismo en el diseño de componentes software





INTERFACE

- Solo contienen operaciones (métodos) sin implementación, es decir solo la firma de los métodos.
- Las clases son las encargadas de implementar las operaciones (métodos) de una o varias interfaces.
- A nivel de interfaces si existe la herencia múltiple, una interface puede heredar de varias interfaces.




```java
public interface ImprimirSpec{

   void imprimir();

}
```

```java
public class Comprobante implements ImprimirSpec{

   public void buscarPorCodigo() {
      // Implementa el método de la clase
   }

   public void buscarPorNombre() {
      // Implementa el método de la clase
   }

   @Override
   public void imprimir() {
     // Implementa el método de la interface
   }

}
```

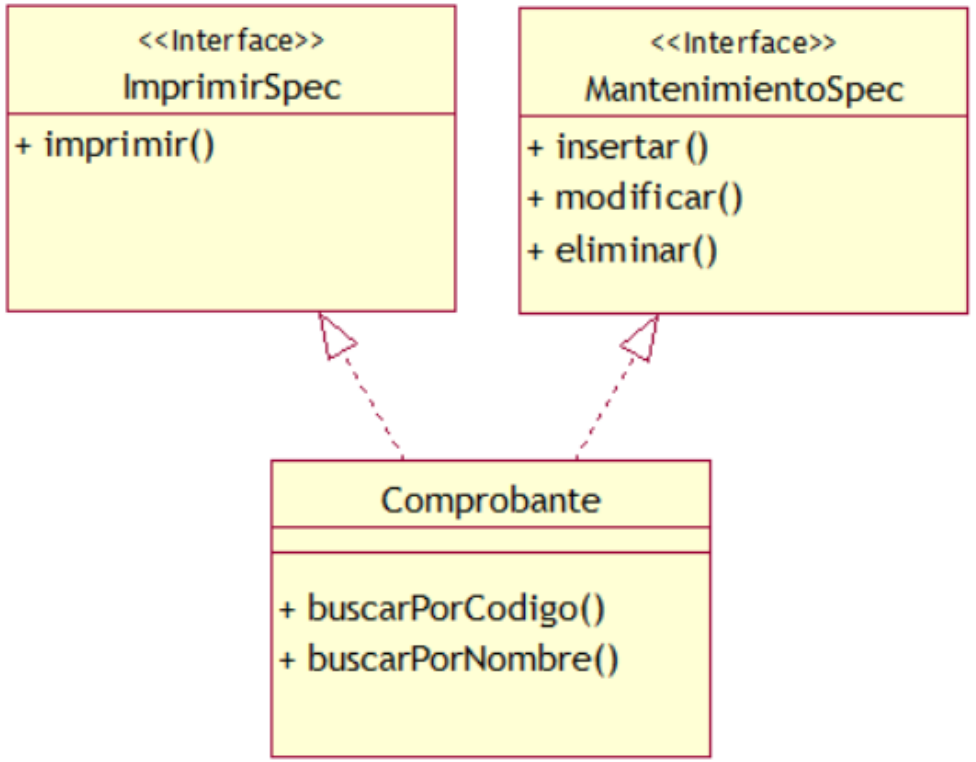
INTERFACE

Ejemplo de Herencia múltiple de Interface.

```
public interface ImprimirSpec {
   void imprimir();
}
```

```
public interface MantenimientoSpec {
   void insertar();
   void modificar();
   void eliminar();
}
```

```
public class Comprobante
implements ImprimirSpec, MantenimientoSpec {

   // Implementa sus propios métodos y
   // los métodos de las interfaces

}
```



CLASE CONCRETA, ABSTRACTA E INTERFACE

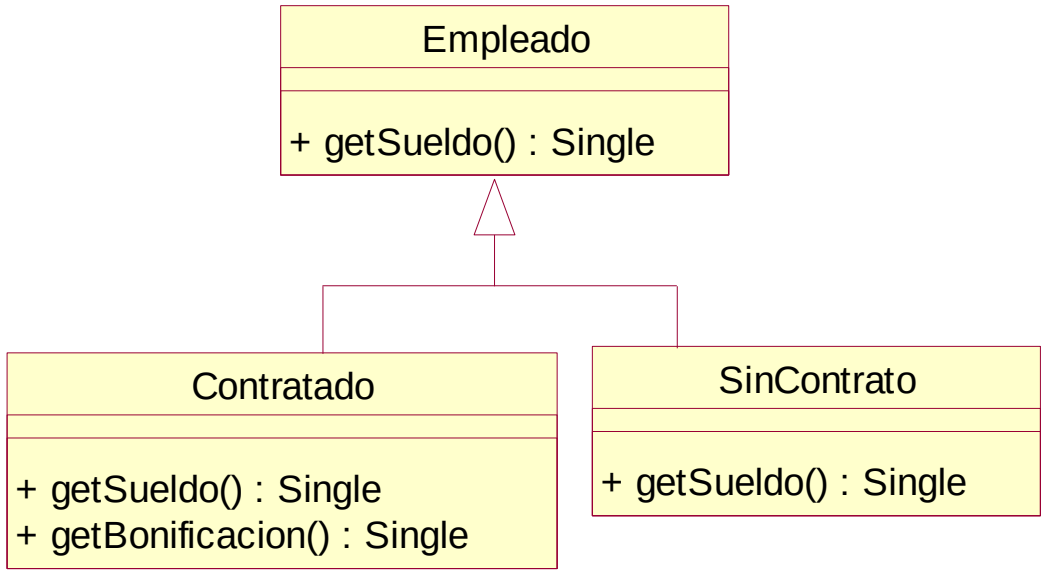
CARACTERISTICA	CLASE CONCRETA	CLASE ABSTRACTA	INTERFACE
HERENCIA	extends (simple)	extends (simple)	implements, * extends ** (múltiple)
INSTANCIABLE	Si	No	No
IMPLEMENTA	Métodos	Algunos métodos	Nada
DATOS	Se permite	Se permite	No se permite ***

* Una clase puede implementar varias interfaces
** Una interface puede heredar de varias interfaces
*** Las variables que se declaran en una interface son implícitamente estáticas, finales y públicas.

POLIMORFISMO

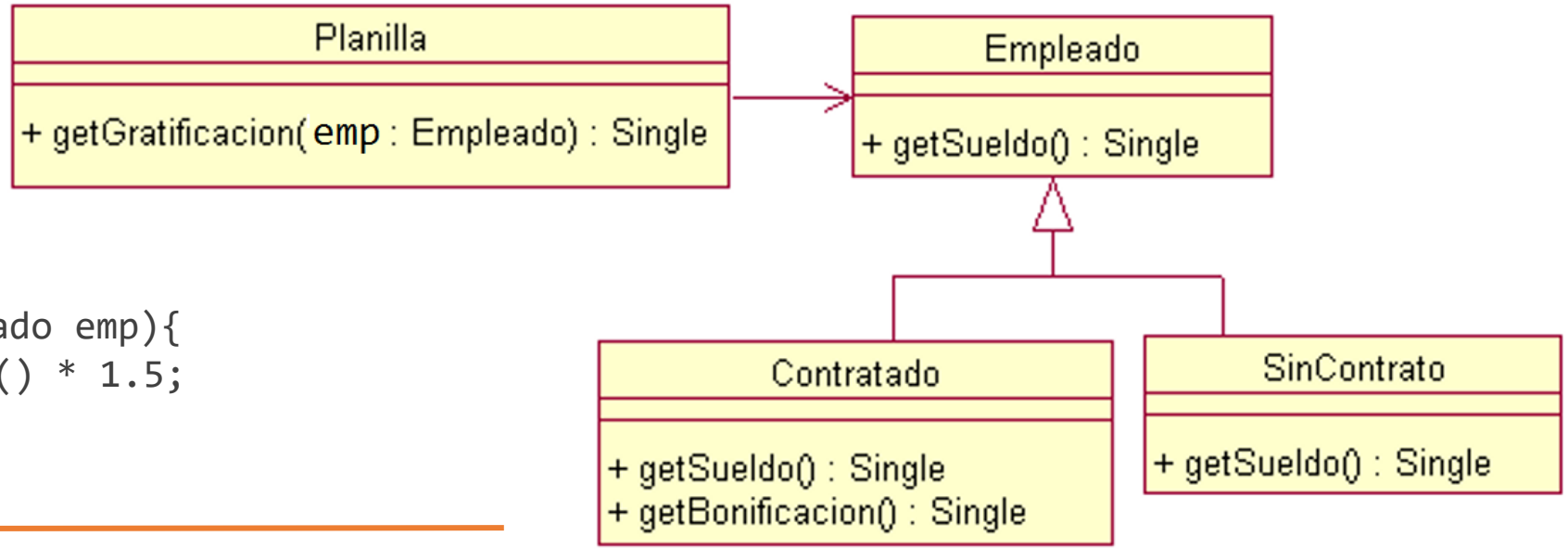
- Existe polimorfismo cuando un método definido en una clase o interface es implementado de varias formas en otras clases.
- Algunos ejemplos de polimorfismos de herencia son: *sobre-escritura*, *implementación* de métodos abstractos (clase abstracta e interface).
- Es posible apuntar a un objeto con una variable de tipo de *clase padre* (supercalse), esta sólo podrá acceder a los miembros (campos y métodos) que le pertenece.

```
// Variable de tipo Empleado y apunta a un
// objeto de tipo Contratado.
Empleado objEmp = new Contratado();

// Invocando sus métodos
double s = objEmp.getSueldo();        // OK
double b = objEmp.getBonificacion(); // Error
```



POLIMORFISMO

- El método **getGratificacion** puede recibir objetos de tipo **Empleado** o subtipos a este.
- Cuando invoque el método **getSueldo** se ejecutará la versión correspondiente al objeto referenciado.

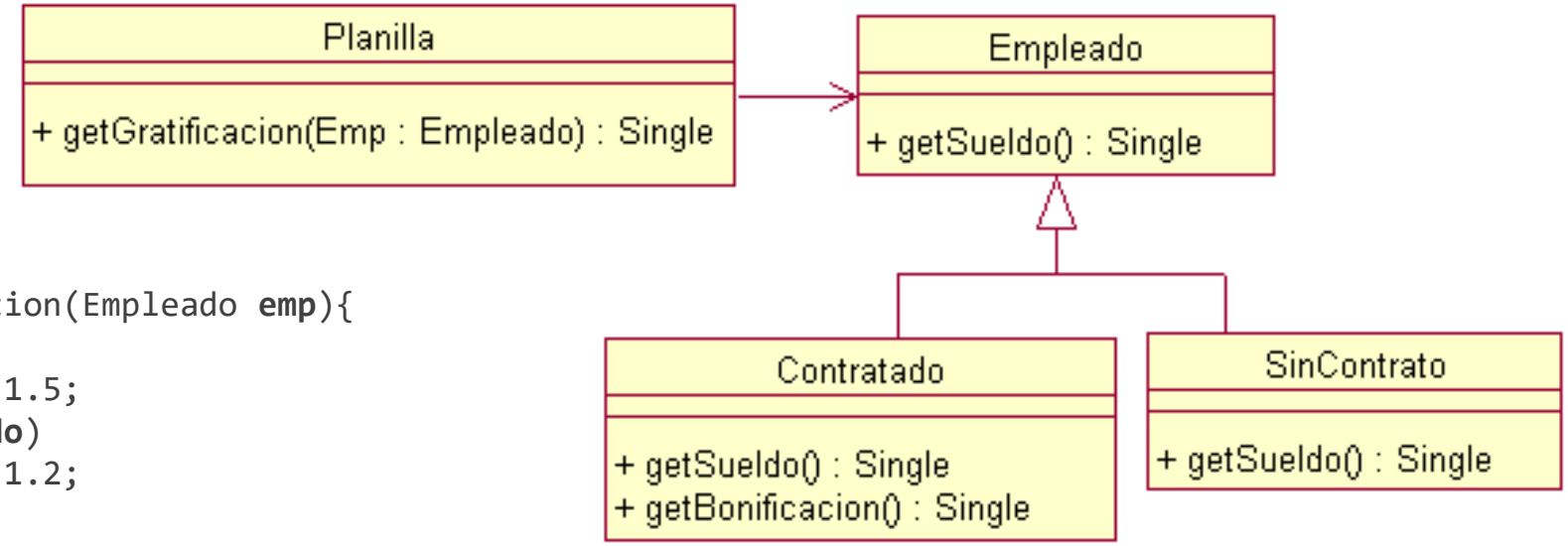



```
public class Planilla {
  public static double
  getGratificacion(Empleado emp){
    return emp.getSueldo() * 1.5;
  }
}
```

```
// Usando la clase Planilla
double g1 = Planilla.getGratificacion(new Contratado());
double g2 = Planilla.getGratificacion(new SinContratado());
```

OPERADOR instanceof

- Este operador permite verificar si el objeto es de un tipo determinado, es decir, el objeto debe pasar por la verificación ES-UN para una determinada clase o interface.




```
public class Planilla {
   public static double getGratificacion(Empleado emp){
      if (emp instanceof Contratado)
            return emp.getSueldo() * 1.5;
      if (emp instanceof SinContratado)
            return emp.getSueldo() * 1.2;
   }
}
```

```
// Usando la clase Planilla
double g1 = Planilla.getGratificacion(new Contratado());
double g2 = Planilla.getGratificacion(new SinContratado());
```

CASTING

- Para restablecer la funcionalidad completa de un objeto, que es de un tipo y hace referencia a otro tipo, debe realizar una conversión (Cast).
- **UpCasting:** Conversión a clases superiores de la jerarquía de clases (Herencia), es automático (conversión implícita), basta realizar la asignación.
- **DownCasting:** Conversión hacia abajo, es decir hacia las subclases de la jerarquía (Herencia), se debe realizar Cast (conversión explicita), si no es compatible genera un error (Excepción).

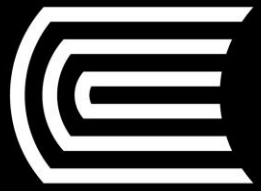
```
// UpCasting (Conversión implícita)
Contratado a = new Contratado();
Empleado b = a;

// DownCasting (Conversión explícita)
Empleado a = new Contrtado();
Contratado b = (Contratado) a;

// Error de compilación
SinContrato a = new SinContrato();
Contratado b = (Contratado) a;
```

PROYECTO EJEMPLO

- La empresa “Secure Money Exchange” es una casa de cambio, y necesita de un software que le permita a sus empleados realizar una atención ágil y segura.

- El software debe permitirles obtener el tipo de cambio según la moneda.

- Para la solución debe aplicar los conceptos desarrollados en este tema.



ucontinental.edu.pe