CJAVA

Programación Orientada a Objeto

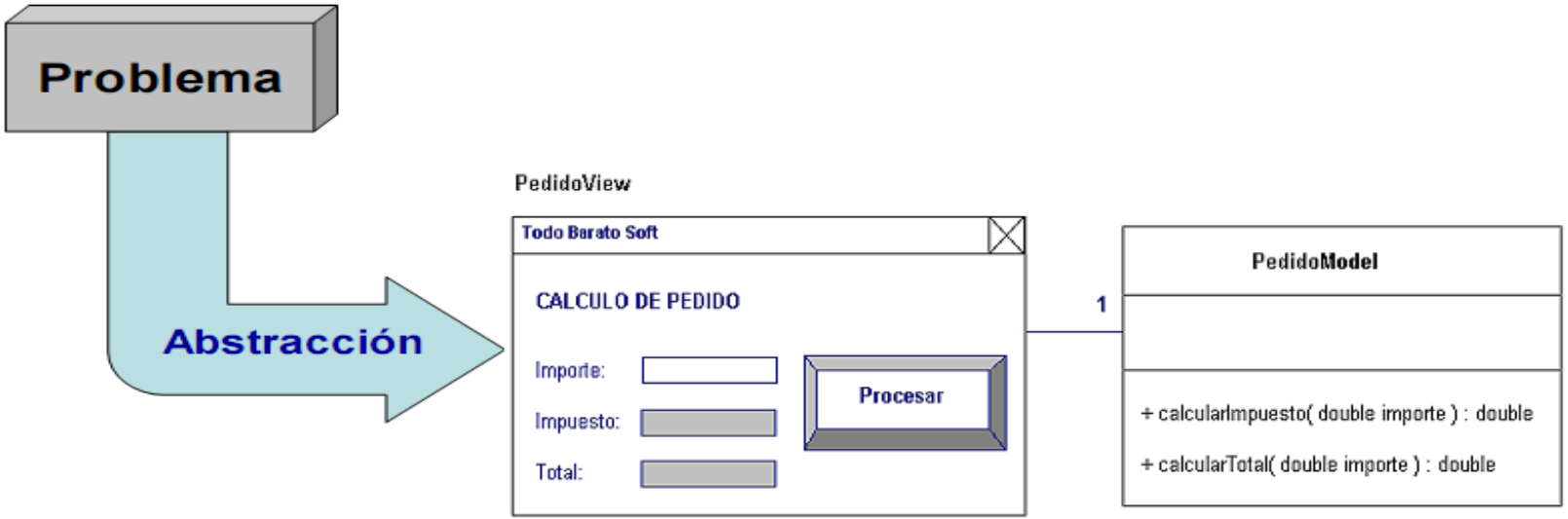
CJAVA
siempre para apoyarte

Teoría Orientada a Objeto

- Definición de clases y objeto
- Declaración de variables
- Vista de una clase en java
- Caso practico de declaración de objetos
- Definición de encapsulamiento
- Aplicaciones

Objetivo

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



Abstraccion

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

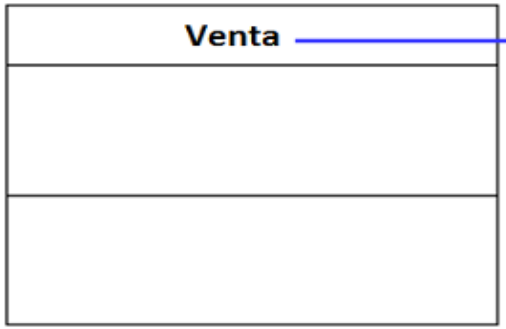
Animal
+ Raza : String + Genero : String
+ Comer()

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

Definicion de Clases y Objetos

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

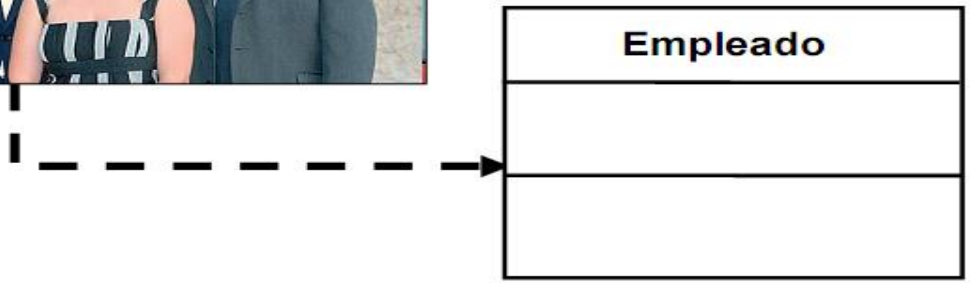
Ejemplos de Nombres de Clase

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

Definicion de Clases y Objetos

OBJETO

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



Definicion de Clases y Objetos

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

CJAVA
siempre para apoyarte

Implementacion de Clases

SINTAXIS

```
public class NombreClase {

    // Definición de variables

    // Definición de métodos

}
```

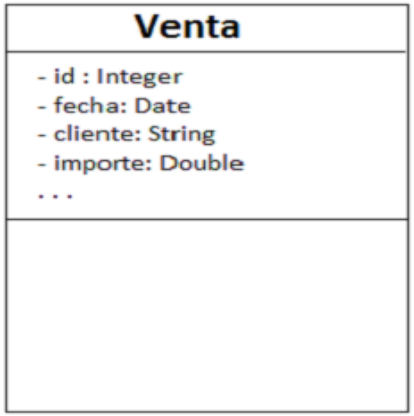
El nombre del archivo debe tener el mismo nombre de la clase.

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

Implementacion de Clases

ATRIBUTOS

- Representa una propiedad de una entidad.
- 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.

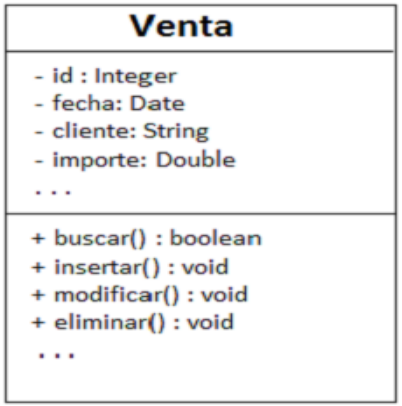



```
public class Venta {

    // 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 la clase que pueden ser solicitados por otras clases.
- 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( . . . ) {
        . . .
        . . .
    }

    . . .

}
```

CJAVA
siempre para apoyarte

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.

CJAVA
siempre para apoyarte

Creación y usos de Clases

OPERADOR NEW

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

ó

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

ACCESO A LOS MÉTODOS

```
variable.nombreMétodo ( … )
```

CJAVA
siempre para apoyarte

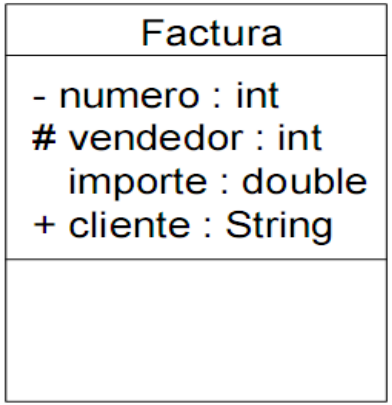
Declaracion de Variables

[modificadorAcceso] tipo nombreVariable [= valor] ;

El modificadorAcceso puede ser:

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

```
public class Factura{

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

}
```



CJAVA
siempre para apoyarte

Declaracion de metodos

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

        // Implementación

}
```

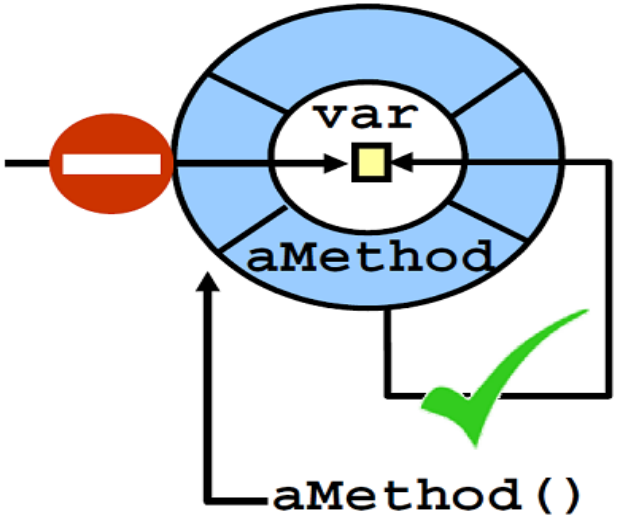
El **modificadorAcceso** puede ser:

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

Encapsulacion

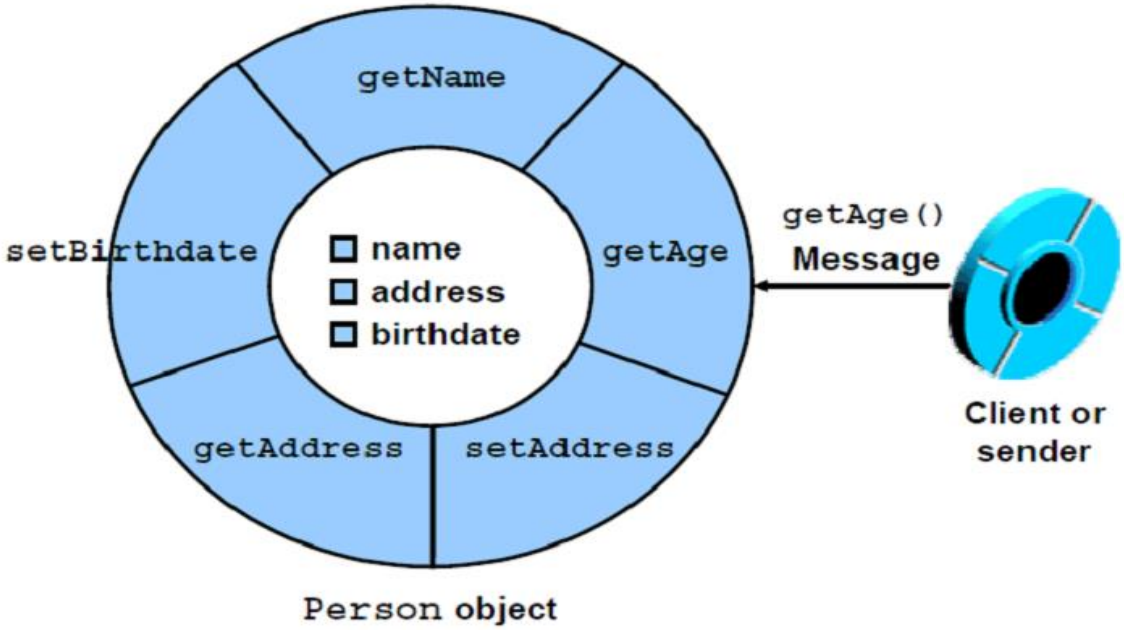
Características

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



Implementacion

Implementación



CJAVA
siempre para apoyarte

Encapsulacion

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

```
public class NombreClase {

    public NombreClase() {

        // Inicialización del objeto

    }

}
```

Cuando se aplica herencia y se quiere ejecutar el constructor del padre, la instrucción es la siguiente:

Destructor

```
class NombreClase {

    protected void finalize() throws Throwable {

        // Liberar recursos del objeto

    }

}
```

Cuando se aplica herencia y se quiere ejecutar el destructor del padre, la instrucción es la siguiente:

```
super.finalize();
```

Herencia de Clases

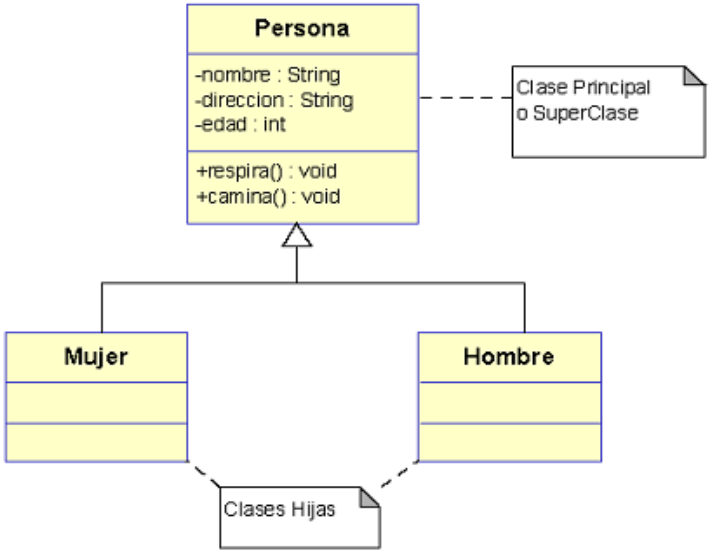
Objetivo

Aplicar la herencia para:

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

De esta manera:

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



CJAVA
siempre para apoyarte

Introducción

Las clases no son suficientes para conseguir los objetivos de:

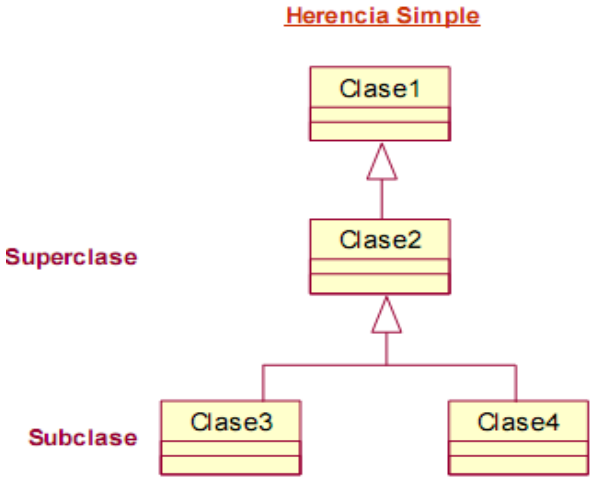
- REUTILIZACIÓN: Necesidad de mecanismos para generar código genérico:
  - ✓ Capturar aspectos comunes en grupos de estructuras similares
  - ✓ Independencia de la representación e implementación
  - ✓ Variación en estructuras de datos y algoritmos
- EXTENSIBILIDAD: Necesidad de mecanismos para favorecer:
  - ✓ "Principio abierto-cerrado" y "Principio Elección Única"
  - ✓ Estructuras polimórficas.

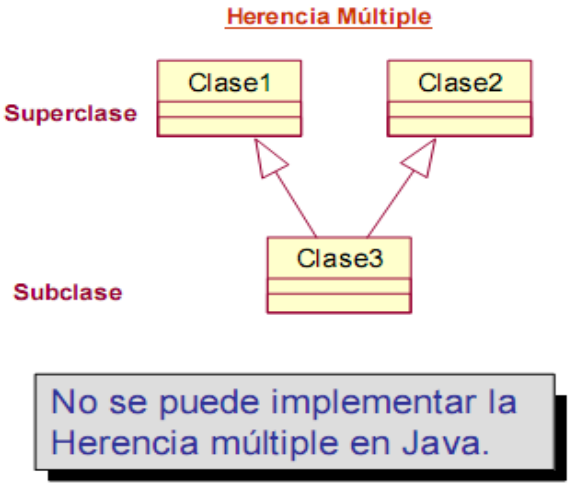
Definición

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)





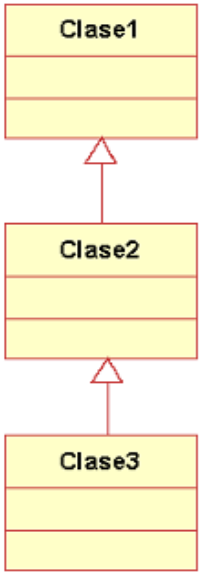
Caracteristicas

Si **Clase2** hereda de **Clase1**, entonces **Clase2** incorpora la estructura (atributos) y comportamiento (métodos) de **Clase1**, pero puede incluir adaptaciones:

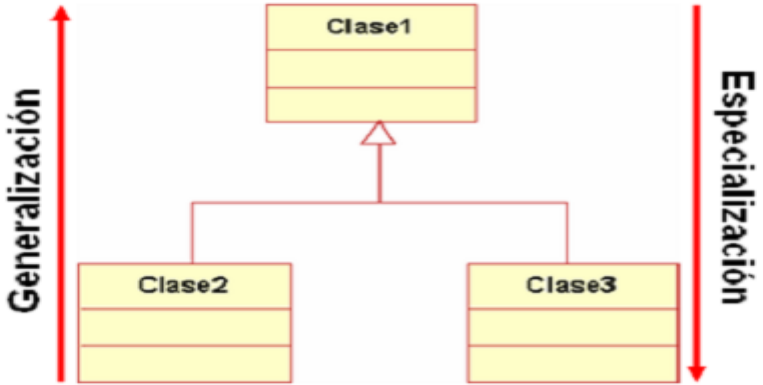
- Clase2 puede añadir nuevos atributos.
- Clase2 puede añadir nuevos métodos.
- Clase2 puede redefinir métodos heredados (refinar o reemplazar).

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



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.

No hay receta mágica para crear buenas jerarquías de herencia.

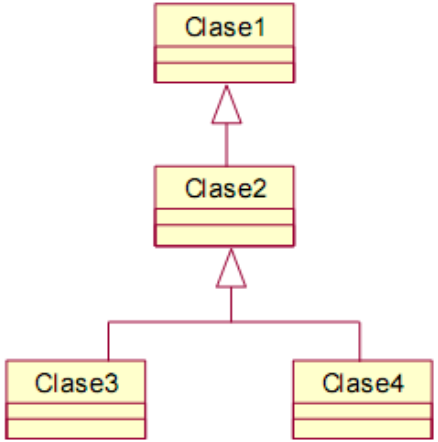
Implementación

```
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

CJAVA
siempre para apoyarte

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();
            . . .
      }
}
```

CJAVA
siempre para apoyarte

Acceso Protegido

Una subclase hereda todos los miembros definidos en la superclase, pero no puede acceder a los miembros privados.

Para permitir que un método de la subclase pueda acceder a un miembro (variable/método) de la superclase, éste tiene que declararse como **protected**.

- **private**: visible sólo en la clase donde se define.
- **Sin modificador (por defecto):** visible a las clases del paquete.
- **protected:** características visibles en las subclases y al resto de clases del paquete.
- **public:** visible a todas las clases.

CJAVA
siempre para apoyarte

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 (...) ;**

CJAVA
siempre para apoyarte

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.

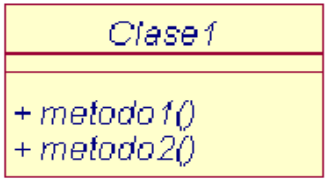
```
public final class Clase1 {
    ...
}
```

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();

}
```

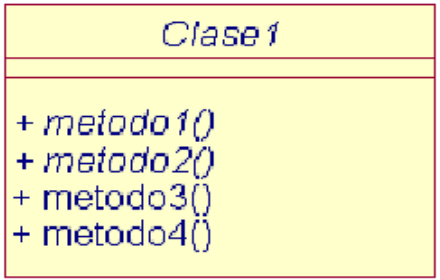
Clases Parcialmente Abstracta

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.

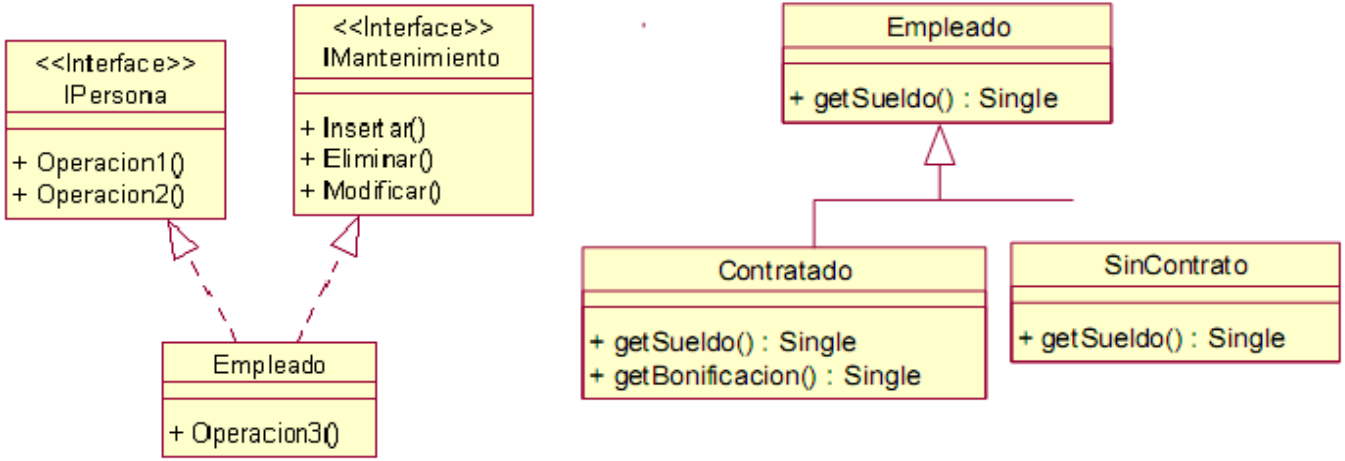



```
public abstract class Clase1 {

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

    public void metodo3() {
        . . .
    }
    public void metodo4() {
        . . .
    }

}
```

Interfaces

Objetivo

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

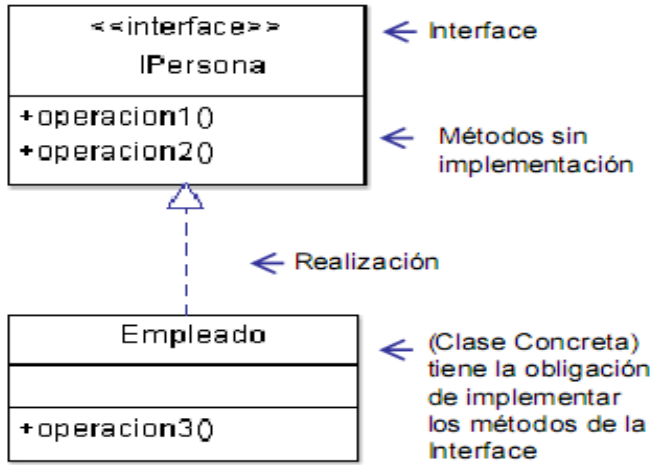


Interface

- Solo contienen operaciones (métodos) sin implementación, es decir solo la firma (signature).
- Las clases son las encargadas de implementar las operaciones (métodos) de una o varias Interfaces (*Herencia múltiple*).
- Se dice que se crean Interface cuando sabemos que queremos y no sabemos como hacerlo y lo hará otro o lo harán de varias formas (*polimorfismo*).

```
public interface IPersona {
    public void operacion1();
    public void operacion2();
}
```

```
public class Empleado implements IPersona {
    public void operacion1() {
        //implementar el método de la interface
    }
    public void operacion2() {
        //implementar el método de la interface
    }
    public void operacion3() {
        //implementación
    }
}
```



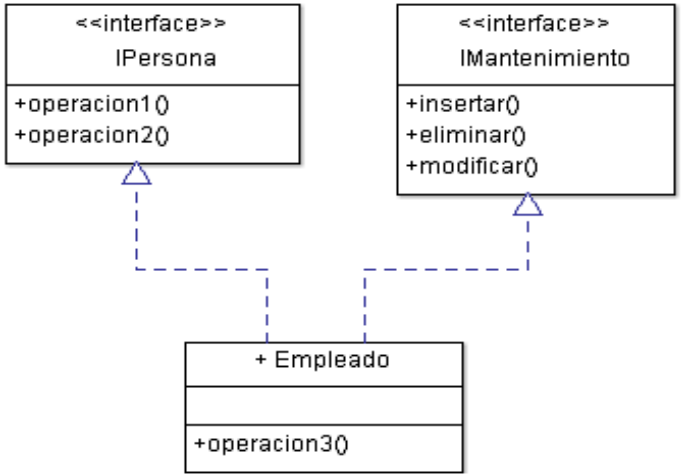
Interface

- Ejemplo de Herencia múltiple de Interface.

```
public interface IPersona {
   public void operacion1();
   public void operacion2();
}
```

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

```
public class Empleado
implements Ipersona, IMantenimiento {

     //implementar los métodos de la interface
     // . . .
     // . . .
     // . . .

}
```



CJAVA
siempre para apoyarte

Clase Concreta, Abstracta Interface

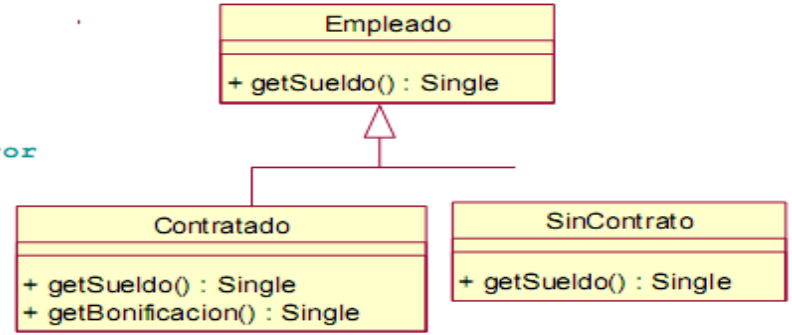
Tipo	Clase Concreta	Clase Abstracta	Interface
Herencia	extends (simple)	extends (simple)	implements (múltiple)
Instanciable	Si	No	No
Implementa	Métodos	Algunos métodos	Nada
Datos	Se permite	Se permite	No se permite

Polimorfismo

Polimorfismo

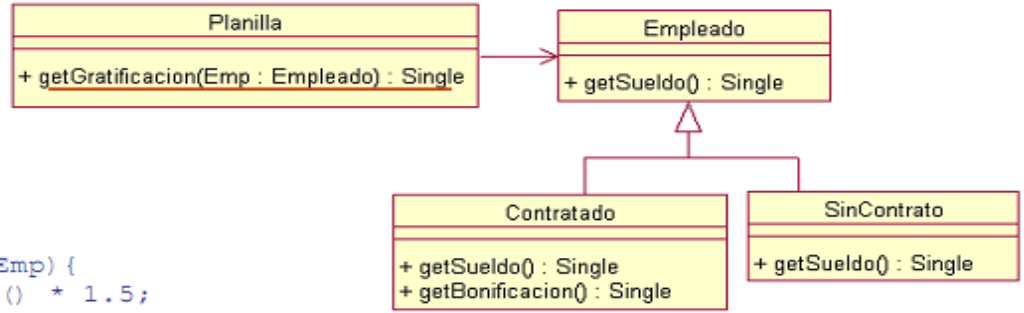
- Se dice que existe polimorfismo cuando un método de una clase es implementado de varias formas en otras clases.
- Algunos ejemplos de Polimorfismos de herencia son: *sobre-escritura*, *implementación* de métodos abstractos (clase abstracta e interface).
- Es posible apuntar a un objeto con una variable 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 Empleado o subtipos a este.
- Cuando invoque el método getSueldo se ejecutará la versión correspondiente al objeto referenciado.

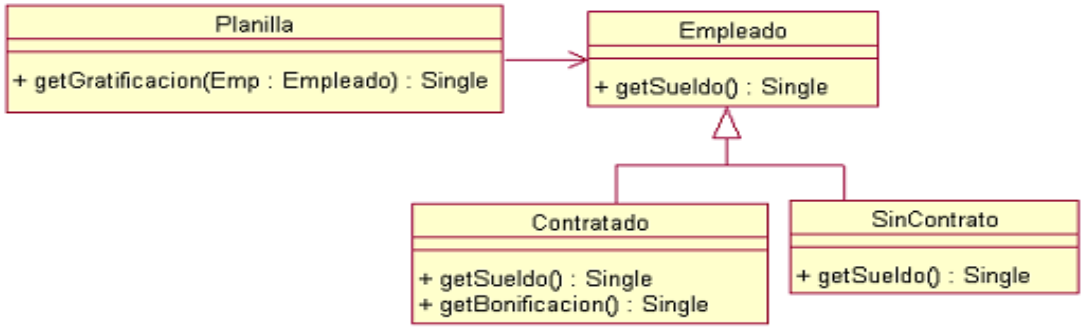



```
public class Planilla {
   public static double
   getGratificacion(Empleado Emp){
         return Emp.getSueldo()  * 1.5;
   }
}
```

```
//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 instancia de un tipo especifico.




```
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());
```

Proyecto propuesto 1

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.

Proyecto propuesto 2

La empresa **Vía Exitos** Necesita saber cuanto se le debe pagar a sus trabajadores y a cuanto asciende el importe de retención de 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 el importe supera los 1500 Nuevos Soles se debe retener el 10% del total.

CJAVA
siempre para apoyarte

Proyecto propuesto 3

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**.

CJAVA

siempre para apoyarte

Gracias