Universidad César Vallejo

Pregrado

SESIÓN 04:
Relación de Herencia

Contexto

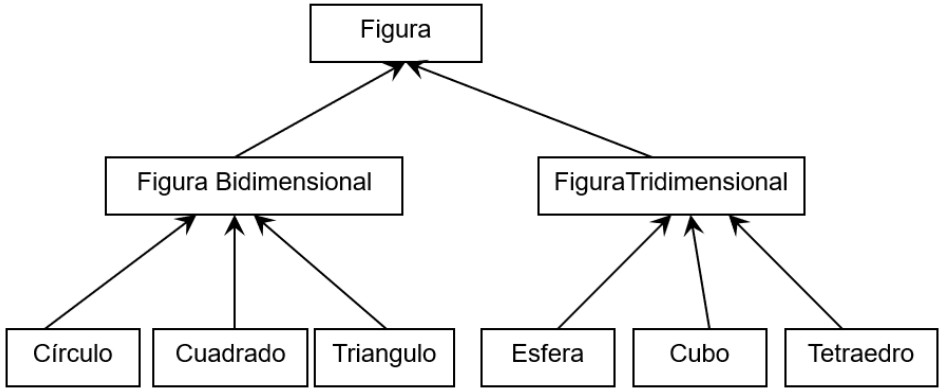
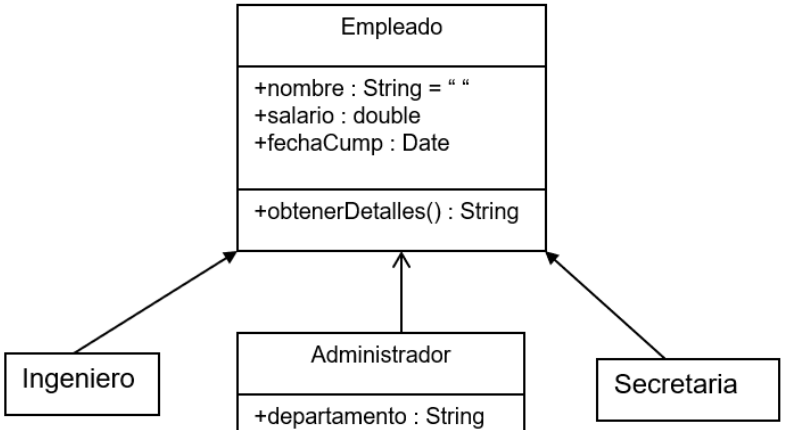
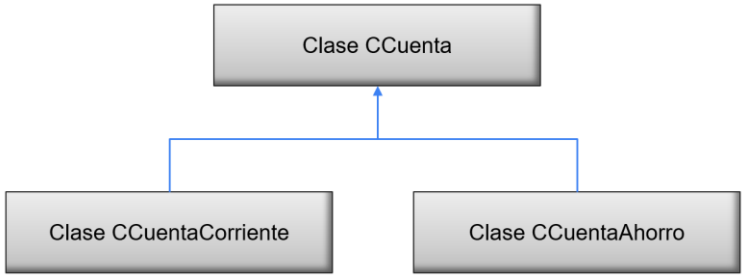


Figura
Figura Bidimensional
FiguraTridimensional
Círculo
Cuadrado
Triangulo
Esfera
Cubo
Tetraedro


Empleado
+nombre : String = " "
+salario : double
+fechaCump : Date
+obtenerDetalles() : String
Ingeniero
Administrador
+departamento : String
Secretaria
Director
+adjudicarVehiculo : double
+incrementarConcesion()


Clase CCuenta
Clase CCuentaCorriente
Clase CCuentaAhorro

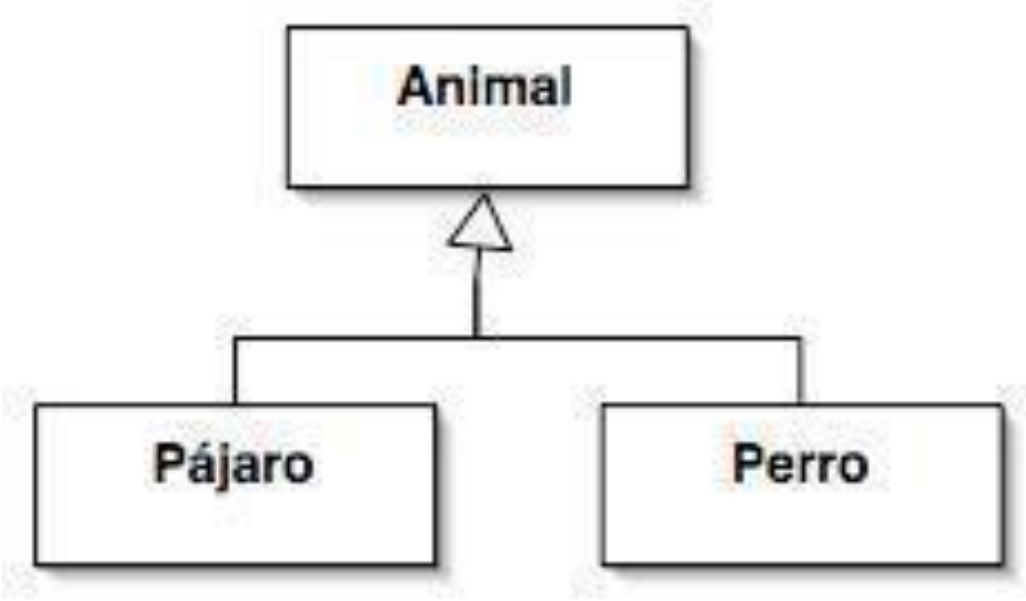
Herencia

- Es una propiedad que permite definir clases a partir de otras ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes.
- Es la relación entre una clase general y otra clase mas especifica.
- Es un mecanismo que nos permite crear clases derivadas a partir de una clase base.
- Permite compartir automáticamente métodos y datos entre clases, subclases y objetos.
- Ejemplo: Si declaramos una clase **Parrafo** derivada de una clase **Texto**, todos los métodos y variables asociadas con la clase **Texto**, son automáticamente heredados por la subclase **Parrafo**.

Herencia

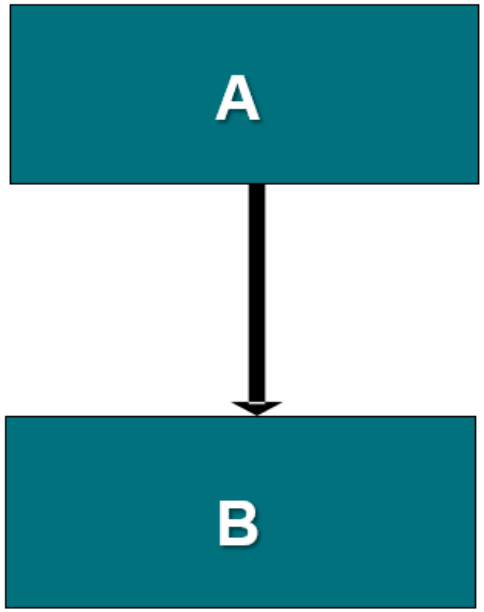
- Se trata de crear una clase hija (**subclase**) que hereda de la clase padre (**superclase**) sus atributos y métodos.
- La subclase puede tener sus propios atributos y métodos.
- Permite la reusabilidad del código.
- En Java se implementa mediante: **extends**

Herencia simple



Herencia múltiple

Herencia



A es un ascendiente o superclase de B. Si la herencia entre A y B es directa decimos que A es la clase padre de B.

B es un descendiente o subclase de A. Si la herencia entre A y B es directa decimos que B es una clase hija de A.

La clase derivada puede añadir nuevas variables y métodos y/o redefinir los métodos heredados.

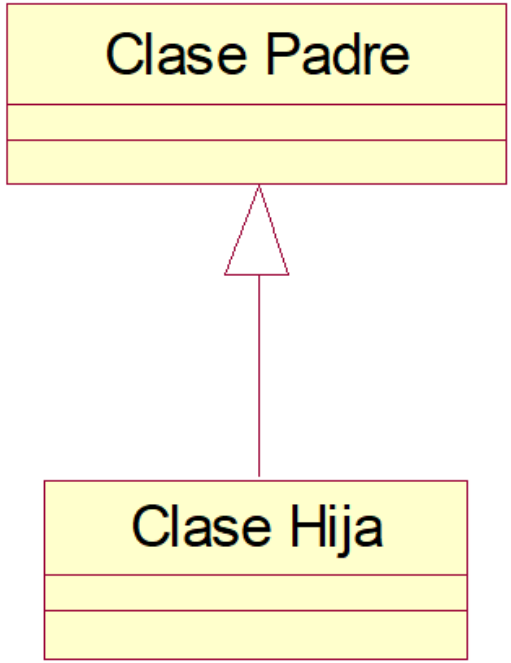
La herencia permite que se puedan definir nuevas clases basadas en clases existentes.

Herencia en Java

Java permite definir una clase como subclase de una clase padre.

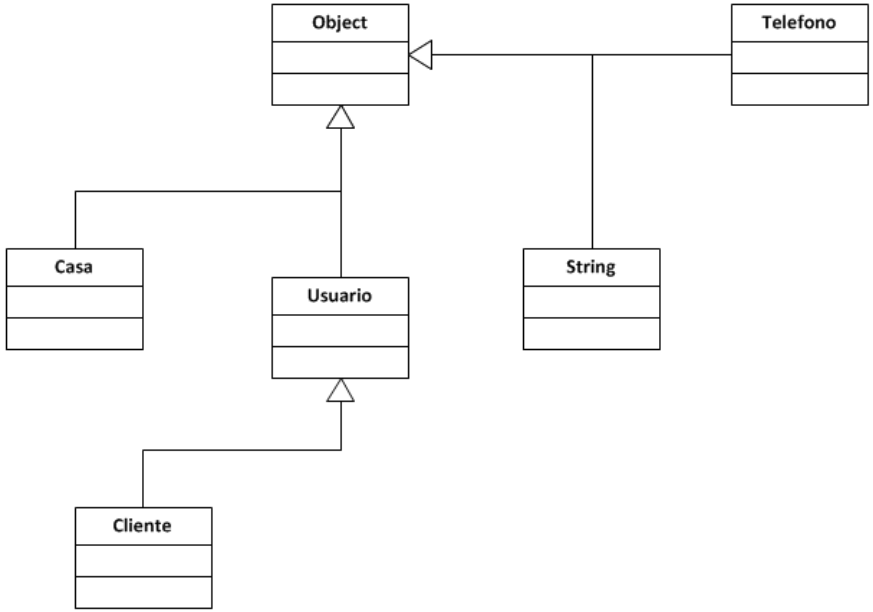
```
class clase_hija extends clase_padre

{

    ..........

}
```

Por ejemplo, podemos definir **CuentaAhorros** y **CuentaCorriente** como subclases que extienden de **CuentaBancaria**, ya que esta última agrupa las características comunes de ambos tipos de cuenta.



La clase Object

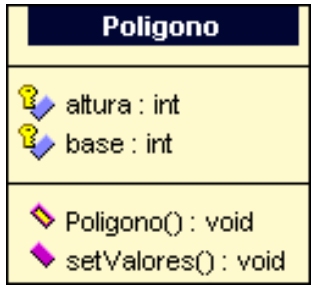
En Java existe una clase base que es la raíz de la jerarquía y de la cual heredan todas aunque no se diga explícitamente mediante la clausula **extends**.



Herencia en Java - Ejemplo


```java
public class Poligono {
  protected int base, altura;
  public Poligono() {
  }
  public void setValores(int a, int b) {
    base = a;
    altura = b;
  }
}
```

```java
public class Triangulo extends Poligono {
  public int area() {
    return (base * altura / 2);
  }
}
```




```java
public class Rectangulo extends Poligono {
  public int area() {
    return (base * altura);
  }
}
```

```java
public class AppPoligonos {
  public static void main(String[] args) {
    Rectangulo rect;
    Triangulo trgl;
    rect = new Rectangulo();
    trgl = new Triangulo();
    rect.setValores(4, 5);
    trgl.setValores(4, 5);
    System.out.print("Area Rectangulo : " +
                     rect.area() + '\n' +
                     "Area Triangulo : " +
                     trgl.area() + '\n');
  }
}
```

Constructores y Herencia

- Cuando se declara un objeto de una clase derivada, se ejecutan los constructores siguiendo el orden de derivación, es decir, primero el de la clase base, y después los constructores de las clases derivadas de arriba a abajo.

- Para pasar parámetros al constructor de la clase padre se utiliza la palabra reservada **super**.

```
super (para1,  para2 , ...,  paraN)
```

Ejemplo de super


```java
class Persona {
  protected String nombre;
  protected int edad;
  public Persona() {}

  public Persona(String n, int e) {
    nombre = n;
    edad = e;
  }
  public void mostarDatos(){
    System.out.println("Nombre: "+ nombre+
                       "Edad: "+edad);
  }
}
```

```java
class Alumno extends Persona {
  private String curso;
  private String nivelAcademico;
  public Alumno(String n, int e, String c, String nivel) {
    super(n, e);//constructor de la clase padre
    curso = c;
    nivelAcademico = nivel;
  }
  public void mostarDatos(){//sobrecarga del metodo de la clase padre
  System.out.println("Nombre: "+ nombre+"\n"+
                     "Edad: "+edad+"\n"+
                     "Curso: "+curso+"\n"+
                     "Nivel Academico: "+nivelAcademico);
}
  public static void main(String[] args) {
    Alumno a = new Alumno("Pepe", 19, "Calculo II", "bueno");
    a.mostarDatos();
  }
}
```

Redefinición de métodos


```
class Persona {
    private String nombre;
    private int edad;
    ..................
    public String Datos( ) {
        return nombre + edad;
    }
    public void setEdad(int e) {
        edad = e;
    }
}
```

```
class Alumno extends Persona {
    private int curso;
    private String nivelAcademico;
    ..................
    @override
    public String Datos( ) {
        return super.Datos() + curso + nivelAcademico;
    }
    public void setCurso(int c) {
        curso = c;
    }
}
```

Construya la subclase Profesor heredada de la clase Persona

Herencia - Ejemplo


```java
public class CuentaBancaria {
  protected double saldo;
  protected String nombreCuenta;
  //Constructor CuentaBancaria
  public CuentaBancaria() {
  }
  public CuentaBancaria(String nombreCuenta) {
    this.nombreCuenta = nombreCuenta;
  }
  public double getSaldo() {
    return saldo;
  }
  public String getNombreCuenta() {
    return nombreCuenta;
  }
  public void retirar(double valor) {
    saldo -= valor;
  }
  public void depositar(double valor) {
    saldo += valor;
  }
}
```

```java
public class CuentaCorriente extends CuentaBancaria {
  protected int numeroCheques;
  protected int numeroChequesUtilizados;

  public CuentaCorriente() {
  }
  public CuentaCorriente(String nombreCuenta) {
    //se llama al constructor de la clase padre
    super(nombreCuenta);
  }
  public void setCuentaCorriente(int nc, int nc_usados) {
    numeroCheques = nc;
    numeroChequesUtilizados = nc_usados;
  }
  public int getNumeroChequesRestantes() {
    return numeroCheques - numeroChequesUtilizados;
  }
}
```

Herencia - Ejemplo

Debido a que **CuentaCorriente** extiende de **CuentaBancaria**, todos los métodos de esta última son heredados por **CuentaCorriente**. Así que las siguientes líneas serían válidas:

```
CuentaCorriente ctaCte = new CuentaCorriente();
double saldo = ctaCte.getSaldo();
```

Herencia - Ejemplo


```
public class CuentaAhorros extends CuentaBancaria {
  public double totalInteresesMes;
  public CuentaAhorros(String nombreCuenta, double tim) {
  //se llama al constructor de la clase padre
    super(nombreCuenta);
    totalInteresesMes = tim;
  }
  public double getTotalInteresesMes() {
    return totalInteresesMes;
  }
  //metodo ya definido en CuentaBancaria
  public double getSaldo() {
    return saldo + totalInteresesMes;
  }
}
```

Como puede verse, la clase **CuentaAhorros** está redefiniendo el método **getSaldo**, que ya estaba definido en **CuentaBancaria**.

Ventajas

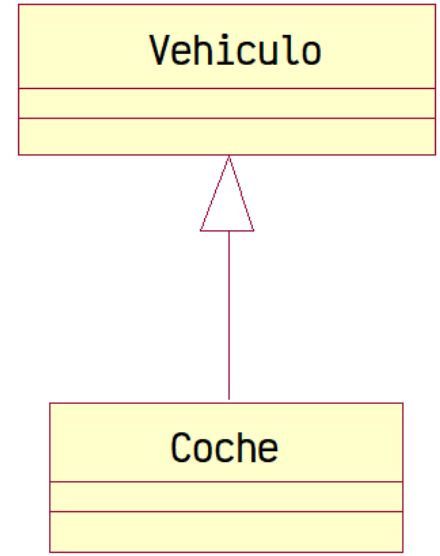
- Modelado de la realidad: las relaciones de especialización/generalización entre las entidades del mundo real.
- Evita redundancias
- Facilita reutilizar código previamente desarrollado.
- Si una clase deriva de otra (**extends**) hereda todas sus variables y métodos
- Sirve de soporte para el polimorfismo

Aplicación de la herencia

- Especialización
- Extensión
- Especificación
- Construcción

Especialización

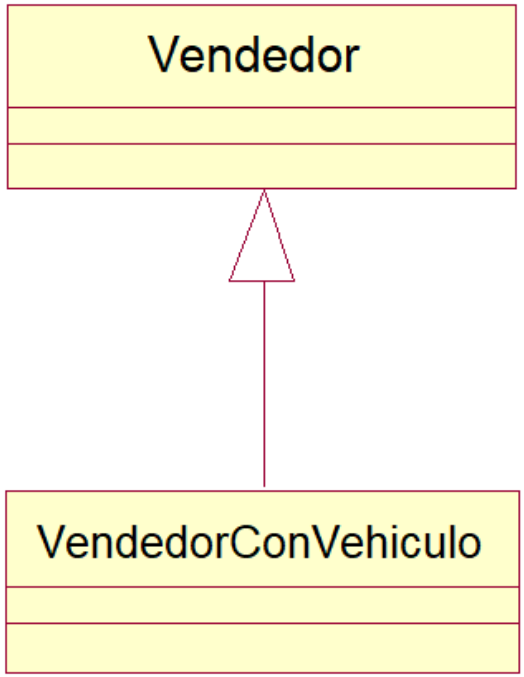
Dado un concepto B (vehiculo) y otro concepto A (coche) que representa una especialización de B, entonces puede establecerse una relación de herencia entre las clases de objetos que representan A y B.



A es B:
Un Ciche es un Vehiculo

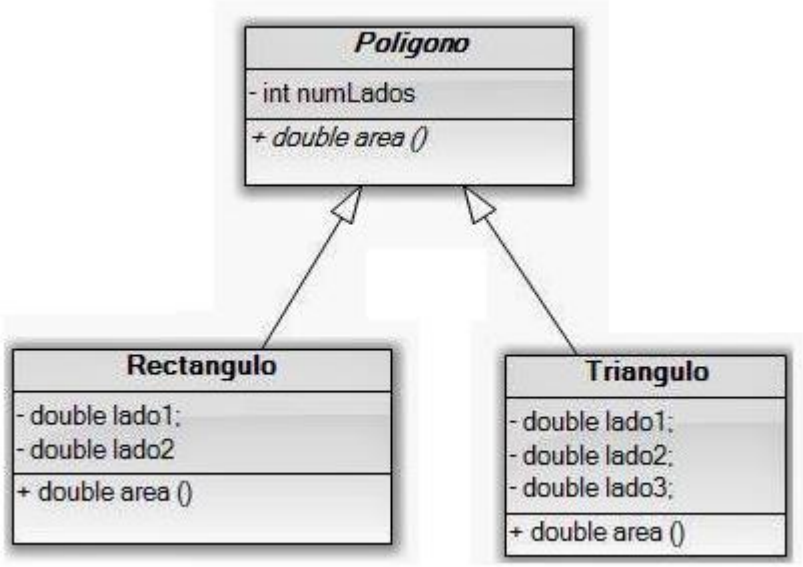
Extensión

Una clase puede servir para extender la funcionalidad de una superclase sin que represente necesariamente un concepto más específico.



Especificación

Una superclase puede servir para especificar la funcionalidad mínima común de un conjunto de descendientes.



Tipos de herencia

- Principalmente existen dos tipos de herencia.
  - **Herencia simple:** una clase solo puede tener un padre, por lo tanto la estructura de clases será en forma de arbol.
  - **Herencia múltiple:** Una clase puede tener uno o varios padres. La estructura de clases es un grafo

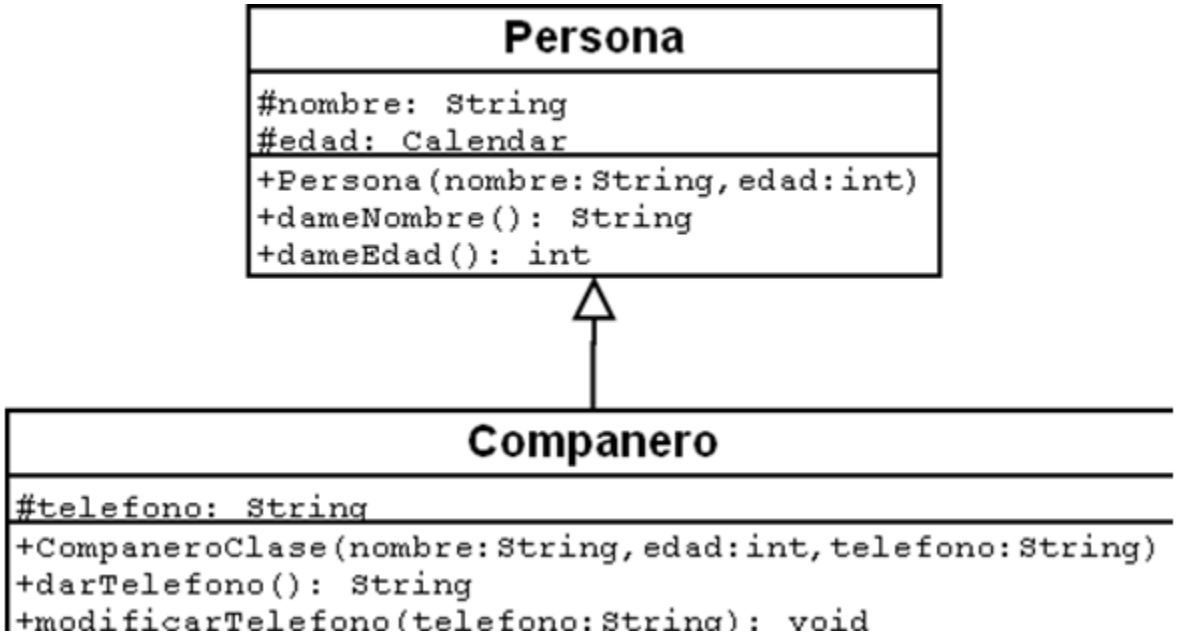
Tipos de herencia

Herencia simple

- Muy fácil de entender y manejar tanto por el lenguaje como por el programador.
- Limitada puesto que en el mundo real un objeto puede pertenecer a varias clases y sin embargo aquí esta situación no se puede modelar.
- Estructura jerárquica en árbol en donde en la raíz podemos encontrar la clase **Object**, de las que heredan todas las clases.
- Todas las clases tienen un padre
- Todos los objetos son "**Object**"

Tipos de herencia

Ejemplo de herencia simple



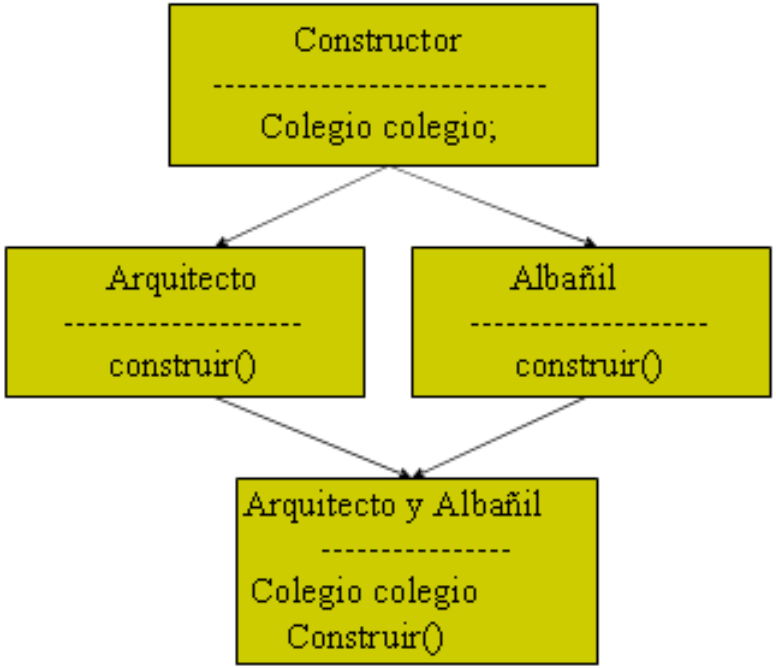
Tipos de herencia

Herencia múltiple

- Es más realista y da al programador más libertad y mas posibilidades de reutilización de código.
- Es mucho más difícil de utilizar por la posibilidad de ciclos y para los lenguajes de programación es muy costoso el permitirlo.

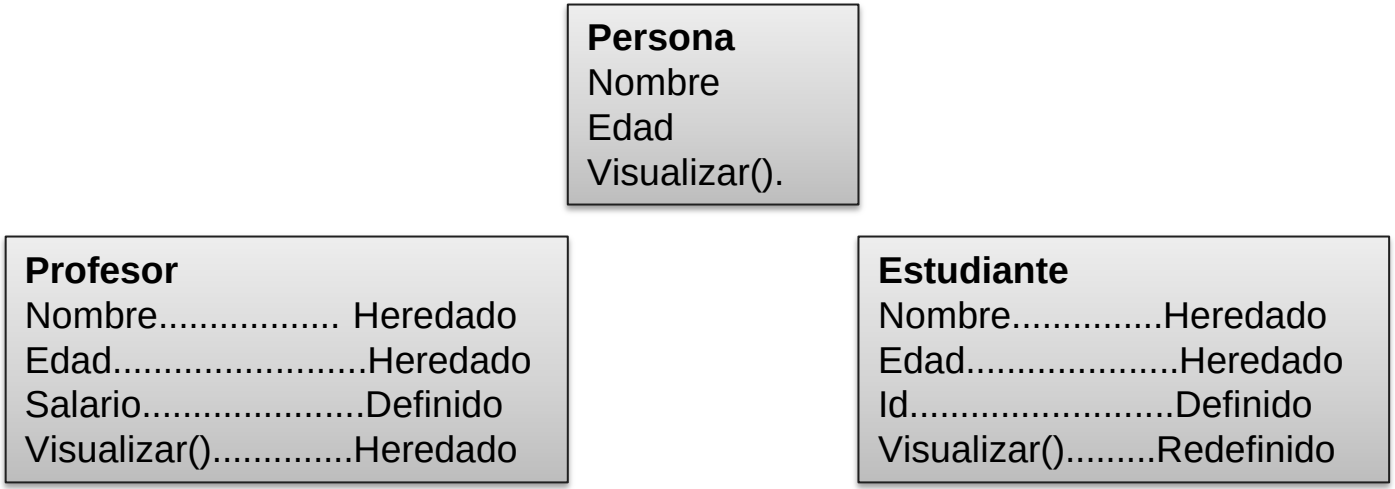
Tipos de herencia

Ejemplo de herencia múltiple



Ejercicio - Enunciado

Diseñar la siguiente jerarquía de clases



Escribir un programa que manipule la jerarquía de clases, lea un objeto de cada clase y lo visualice.

Ejercicio - Solución


```java
public class Persona {
  protected String Nombre;
  protected   int edad;
  public Persona(String Nombre, int edad) {
    this.Nombre = Nombre;
    this.edad = edad;
  }
  void Visualizar() {
    System.out.println("Hola " + Nombre +
                       " tu edad es " + edad + " anios");
  }
}
```

```java
public class Prueba {
  public static void main(String[] args) {
    Persona P = new Persona("persona", 27);
    Estudiante E = new Estudiante("estudiante", 23, 20);
    Profesor Pr = new Profesor("profesor", 30, 20);

    P.Visualizar();
    E.Visualizar();
    Pr.Visualizar();
  }
}
```

```java
class Estudiante
    extends Persona {
  int id;
  public Estudiante(String Nombre, int edad, int id) {
    super(Nombre, edad);
    this.id = id;
  }

  void Visualizar() {
    System.out.println("Hola " + Nombre +
                       " tu edad es " + edad + " anios "+
                       "Tu Id es " + id);
  }
}
```

```java
class Profesor
    extends Persona {
  int salario;

  public Profesor(String Nombre, int edad, int salario) {
    super(Nombre, edad);
    this.salario = salario;
  }
}
```

Herencia - Ejemplo

Desarrollar una clase llamada CuentaCorriente que:

- Tenga tres atributos private de tipo Titular (Nombre – String, Apellidos –String y Edad – int), de tipo String (el número de cuenta) y de tipo double(el saldo).
- Tenga un constructor con parámetros de tipo Titular, String y double.
- Tenga un constructor con parámetros de tipo Titular y String. El saldo se inicializará a 15,3.
- Tenga un getter para cada uno de los atributos.
- Tenga un setter solo para el saldo.
- Tenga un método ingresar que incremente el saldo en una cantidad.
- Tenga un método reintegro que decremente el saldo en una cantidad.
- Tenga un método para que al imprimir la cuenta salga por pantalla el número de cuenta y su saldo.
- Tenga un método para comparar cuentas, sabiendo que dos cuentas serán iguales si sus números de cuenta son iguales.

Nota 1: Al imprimir por pantalla un Titular saldrá su nombre, apellidos y edad.

Nota 2: Para comparar dos String utilizar su método compareTo(String) que devuelve 0 si son iguales.

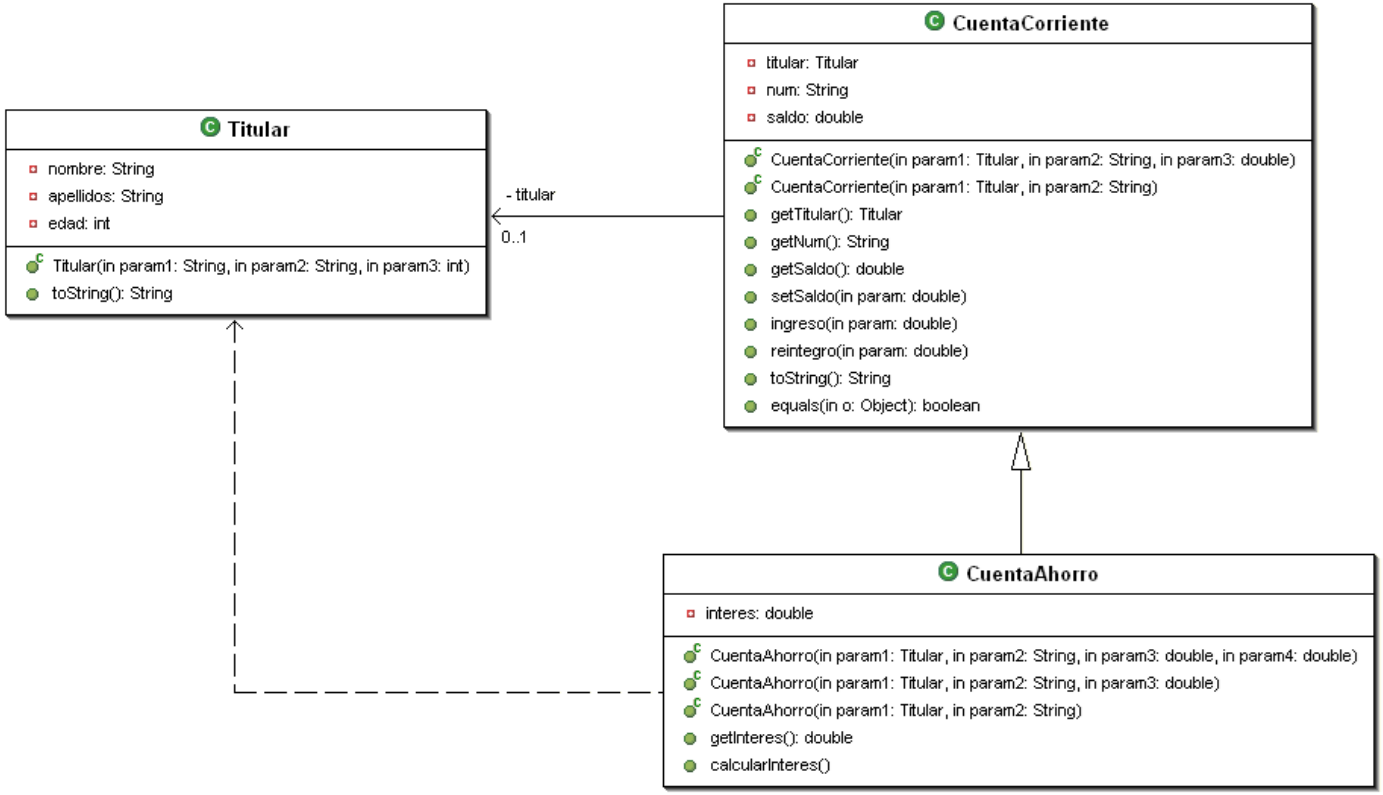
Herencia - Ejemplo

Desarrollar una clase llamada CuentaAhorro que:

- Es una especialización de CuentaCorriente.
- Tiene un atributo mas de tipo double (el interés).
- Tiene un constructor con parámetros de tipo Titular, String, double y double.
- Tiene un constructor con parámetros de tipo Titular, String y double. El saldo se inicializará a 15,3.
- Tiene un constructor con parámetros de tipo Titular y String. El saldo se inicializará a 15,3 y el interés a 2,5.
- Tiene un getter para cada uno de los atributos.
- Tiene un método calcularInteres que incremente el saldo según el interés.

Desarrollar la clase Practica6 que en su método main cree varias cuentas de distinto tipo y trabaje con ellas.

Herencia - Ejemplo



Ejercicio

- 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**.

Universidad **César Vallejo**

Licenciada por Sunedu
para que puedas salir adelante