

Herencia y Polimorfismo

Dr. José Lázaro Martínez Rodríguez

Paradigma Orientado a Objetos

- Ventajas:
 - Es más fácil de mantener
 - Es transportable
 - Es encapsulado y por lo tanto modular
 - Es mejor para proyectos grandes o en aquellos casos donde se reutilice código
- El atractivo intuitivo de la POO es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.
 - *el resultado es que la transformación radical normal de los requisitos del sistema (definidos en términos de usuario) a la especificación del sistema (definidos en términos de la computadora) se reduce considerablemente*

Paradigma Orientado a Objetos

- Las técnicas OO proporcionan mejoras y metodologías para construir sistemas de software complejos a partir de unidades de software modular y reutilizable.
- Los elementos más importantes del modelo Orientado a objetos son:
 - **Abstracción**
 - **Encapsulación**
 - **Jerarquía (Herencia)**
 - **Polimorfismo**

Paradigma Orientado a Objetos

Abstracción

- La **abstracción** es la propiedad que permite representar las características **esenciales** de un objeto, sin preocuparse de las características restantes (características no esenciales).
- Durante el proceso de abstracción se define qué características y qué comportamiento debe tener el modelo.

abstracción

- Definir una abstracción significa describir una entidad del mundo real, no importa lo compleja que pueda ser y a continuación utilizar esta descripción en un programa.
- Clase Persona
 - Date fechaNacimiento;
 - String nombre;
 - String comidaFavorita;

Encapsulamiento

- ▶ El encapsulamiento permite a los objetos ocultar su implementación de otros objetos; a este principio se le conoce como *ocultamiento de la información*.
- ▶ Aunque los objetos pueden comunicarse entre sí a través de *interfaces* bien definidas no están conscientes de cómo se implementan otros objetos.

- ▶ Cuando un programa crea (instancia) un objeto de algún tipo de clase, las variables de ese objeto se encapsulan en el objeto y sólo pueden utilizarse mediante los métodos de la clase de ese objeto.
- ▶ Por lo general los datos de una clase se declaran como `private` y los métodos como `public`.

Aspectos a considerar:

- ▶ Control del acceso a los miembros
 - ▶ Los modificadores de acceso `public` y `private` controlan el acceso a las variables y los métodos de una clase.
- ▶ Referencias a los miembros del objeto actual mediante `this`
- ▶ Inicialización de los objetos de una clase mediante: Constructores

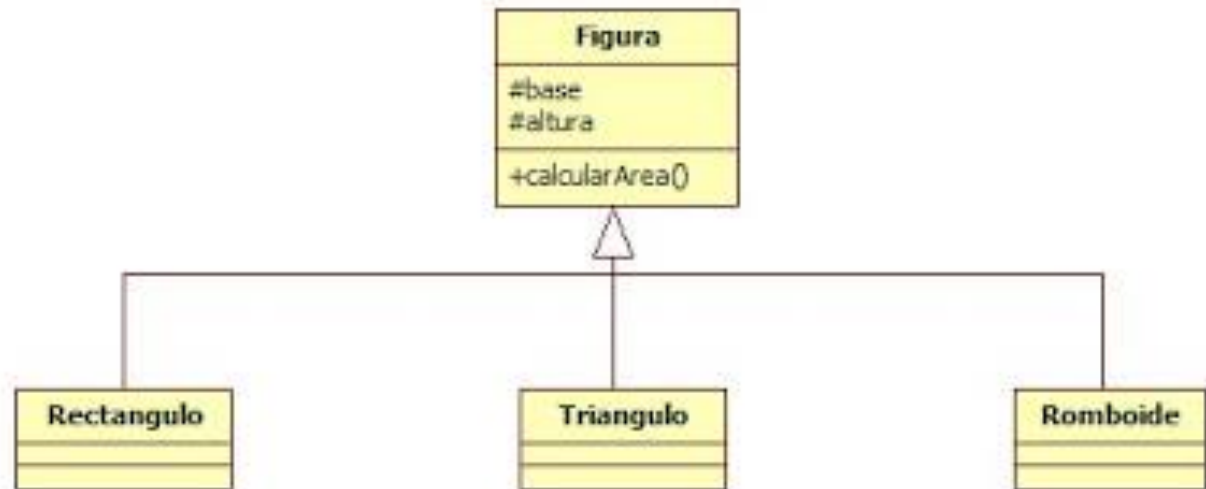

```
class Persona{  
    private String nombre;  
  
    public void setNombre (String nombre) {  
        this.nombre=nombre;  
    }  
  
    public String getNombre () {  
        return this.nombre;  
    }  
}
```

Herencia

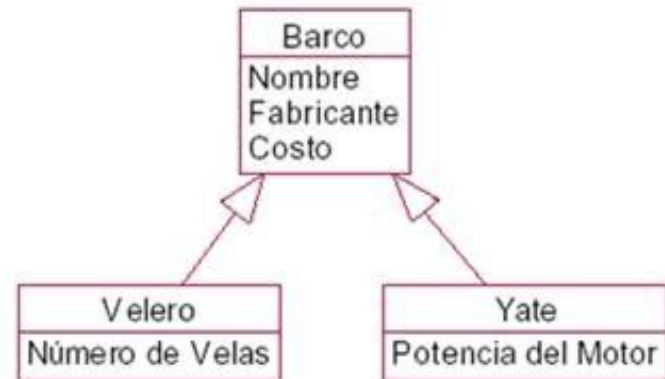
- ▶ ¿Qué es Herencia?
 - ▶ Es una propiedad esencial de la *programación orientada a objetos* que consiste en la creación de nuevas clases a partir de otras ya existentes.
 - ▶ Cuando una clase hereda de otra, contendrá los métodos y atributos de la clase padre.
 - ▶ La herencia permite:
 - ▶ La reutilización del código
 - ▶ Añadir nuevos comportamientos a las clases hijas
 - ▶ La redefinición de comportamientos
 - ▶ Creación de clasificaciones jerárquicas de clases

Ejemplos

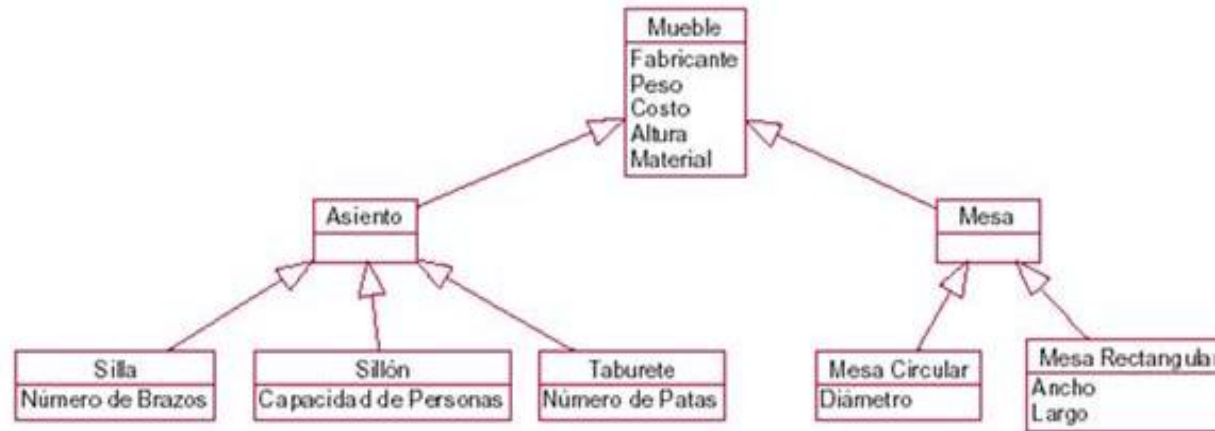
- ▶ Figura
 - ▶ Rectángulo
 - ▶ Triángulo
 - ▶ Romboide



- ▶ Barco
 - ▶ Velero
 - ▶ Yate

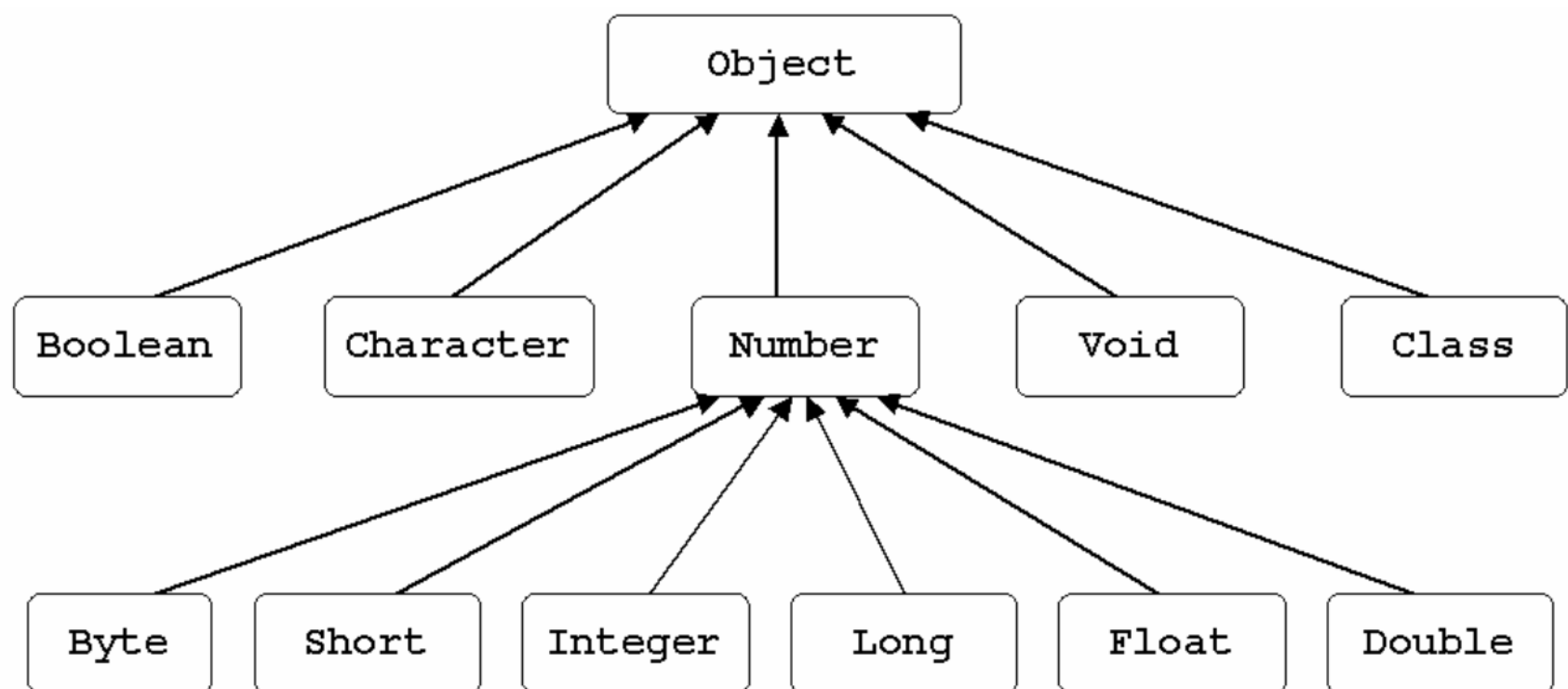


- ▶ Mueble
 - ▶ Asiento
 - ▶ Silla
 - ▶ Sillón
 - ▶ Taburete
 - ▶ Mesa
 - ▶ Mesa Circular
 - ▶ Mesa Rectangular



- ▶ Mesa Circular
- ▶ Mesa Rectangular

- ▶ En Java la clase `java.lang.Object` es la base de toda la jerarquía de clases de Java:
- ▶ Si al definir una clase no se especifica de qué clase deriva, por defecto la clase creada deriva de `Object`.
- ▶ La clase `Object` proporciona métodos heredados como `toString()`, `equals()`, `getClass()`,...



Declaración de herencia

- ▶ Para indicar que la clase **B** (clase descendiente, derivada, hija o subclase) hereda de la clase **A** (clase ascendiente, heredada, padre, base o superclase) se emplea la palabra reservada **extends** en la cabecera de la declaración de la clase descendiente

```
public class ClaseB extends ClaseA {  
    // Declaracion de atributos y metodos especificos de ClaseB  
    // y/o redeclaracion de componentes heredados  
}
```

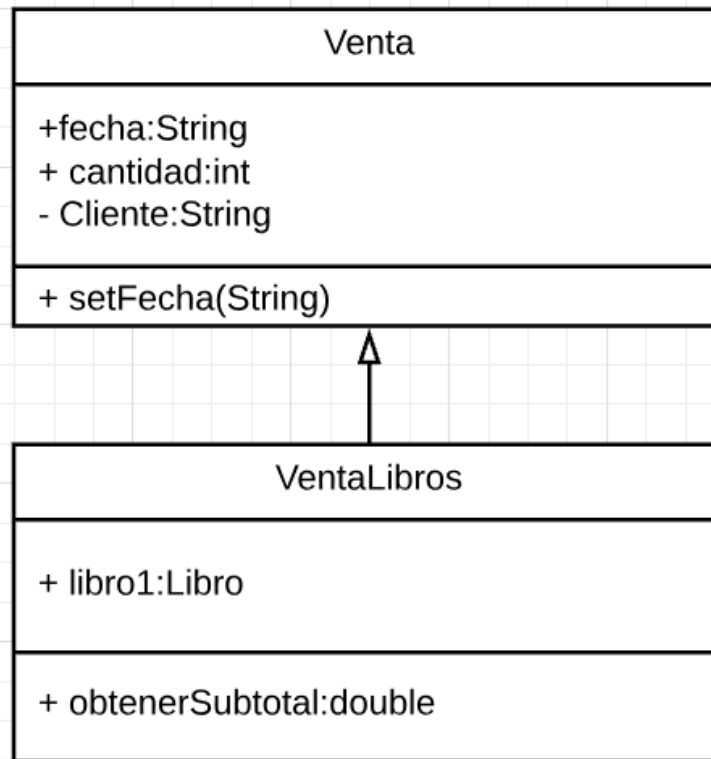
Herencia

- ▶ Representa una relación “es un” (isA)
- ▶ `extends` especifica que la *clase hija* hereda datos y métodos de la *clase padre*
- ▶ Aunque hereda todo, la clase hija solo tiene acceso a los miembros de la clase padre con modificadores de acceso `public` o `protected` o sin modificador

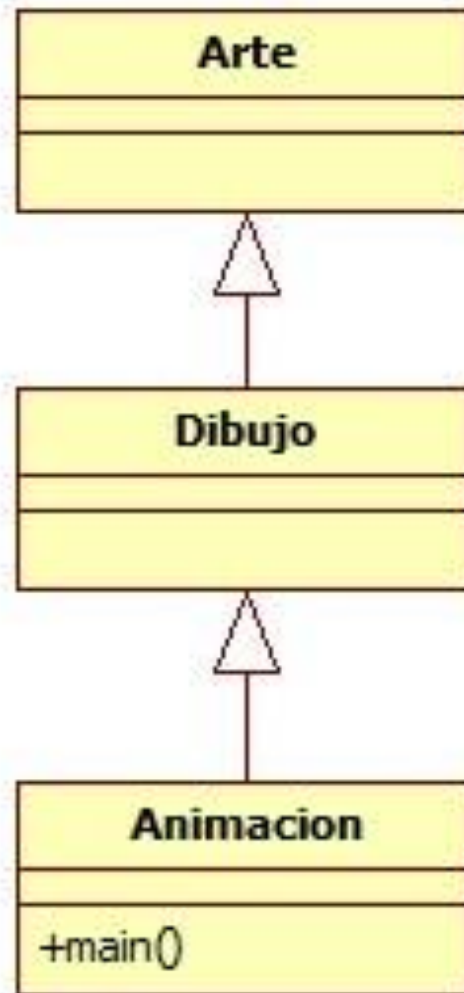
Ejemplo:



- ▶ Dado que ahora hay dos clases involucradas
 - ▶ La clase hija
 - ▶ La clase padre
- ▶ Es esencial que el objeto de la clase hija se inicialice correctamente y sólo hay una forma de garantizarlo, llevando a cabo la inicialización en el constructor.
- ▶ Java inserta automáticamente llamadas al constructor de la clase padre en el constructor de la clase hija.



Ejemplo:



► Arte.java

```
public class Arte {  
    public Arte() {  
        System.out.println("Constructor de Arte");  
    }  
}
```

► Dibujo.java

```
public class Dibujo extends Arte{  
    public Dibujo() {  
        System.out.println("Constructor de dibujo");  
    }  
}
```

► Animacion.java

```
public class Animacion extends Dibujo {  
    public Animacion() {  
        System.out.println("Constructor de animación");  
    }  
    public static void main(String[] args) {  
        Animacion x = new Animacion();  
    }  
}
```

- ▶ La salida de este programa muestra las llamadas automáticas a los constructores de las clases:
- ▶

```
Constructor de Arte
Constructor de Dibujo
Constructor de Animación
BUILD SUCCESSFUL (total time: 1 second)
|
```

Se inicializa la clase padre antes de que los constructores de la clase hija puedan acceder a ella.

Herencia - *super*

- ▶ **super** permite acceder tanto a los constructores como a los métodos de la superclase
- ▶ Se puede llamar al constructor de una superclase desde el constructor de la subclase utilizando la palabra **super**([Lista_de_parámetros]) en la primera sentencia, excepto si se llama a otro constructor de la misma clase con **this**
- ▶ Existe una llamada implícita al constructor por defecto de la superclase aunque no se especifique.

Ejemplo:



Ejemplo:

► Juego.java

```
public class Juego {  
    public Juego(int i) {  
        System.out.println("Constructor de Juego i = " + i);  
    }  
}
```

► JuegoMesa.java

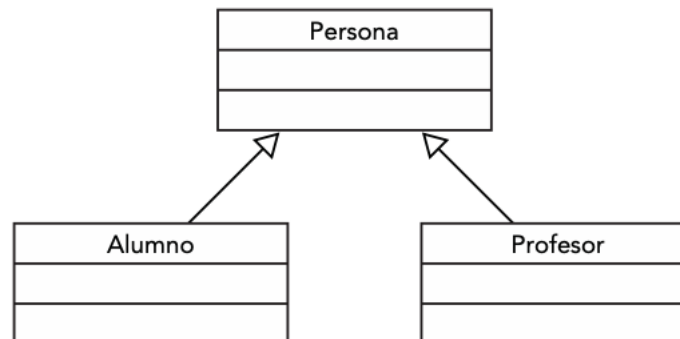
```
public class JuegoMesa extends Juego{  
    public JuegoMesa(int i){  
        super(i);  
        System.out.println("Constructor JuegoMesa i = " + i );  
    }  
}
```

► Ajedrez.java

```
public class Ajedrez extends JuegoMesa{  
    public Ajedrez() {  
        super(11);  
        System.out.println("Constructor Ajedrez");  
    }  
    public static void main(String[] args) {  
        Ajedrez a = new Ajedrez();  
    }  
}
```

Ejemplo

- Ejercicio 1. Supongamos que tenemos las clases `Alumno` y `Profesor`. Usando la generalización, podemos definir una clase padre que sea la clase `Persona`.
 - La ventaja de hacerlo así es que reutilizamos el código de la clase `Persona` en las clases hijas.
 - Además, el código relacionado con alumnos y con profesores queda por separado, lo que facilita el mantenimiento del programa. La figura ilustra esta relación de herencia.



Ejemplo

- La clase `Persona` define las propiedades comunes. Queremos hacer la clase `Persona` con los atributos privados `nombre` de clase `String` y `fechaNacimiento` de clase `Fecha`.
- El constructor de `Persona` inicializa los atributos con un `nombre` (`String`) y un objeto de clase `Fecha` con los atributos privados `anio`, `mes` y `dia` de clase `Integer`
- Los métodos de `Fecha` son: `toString()`, para regresar la cadena de caracteres con el día, el mes y el año, y `asignar()`, el cual es un método público que asigna un día, un mes y un año específicos al objeto. ¿Cómo codificarías estos métodos?

Ejemplo

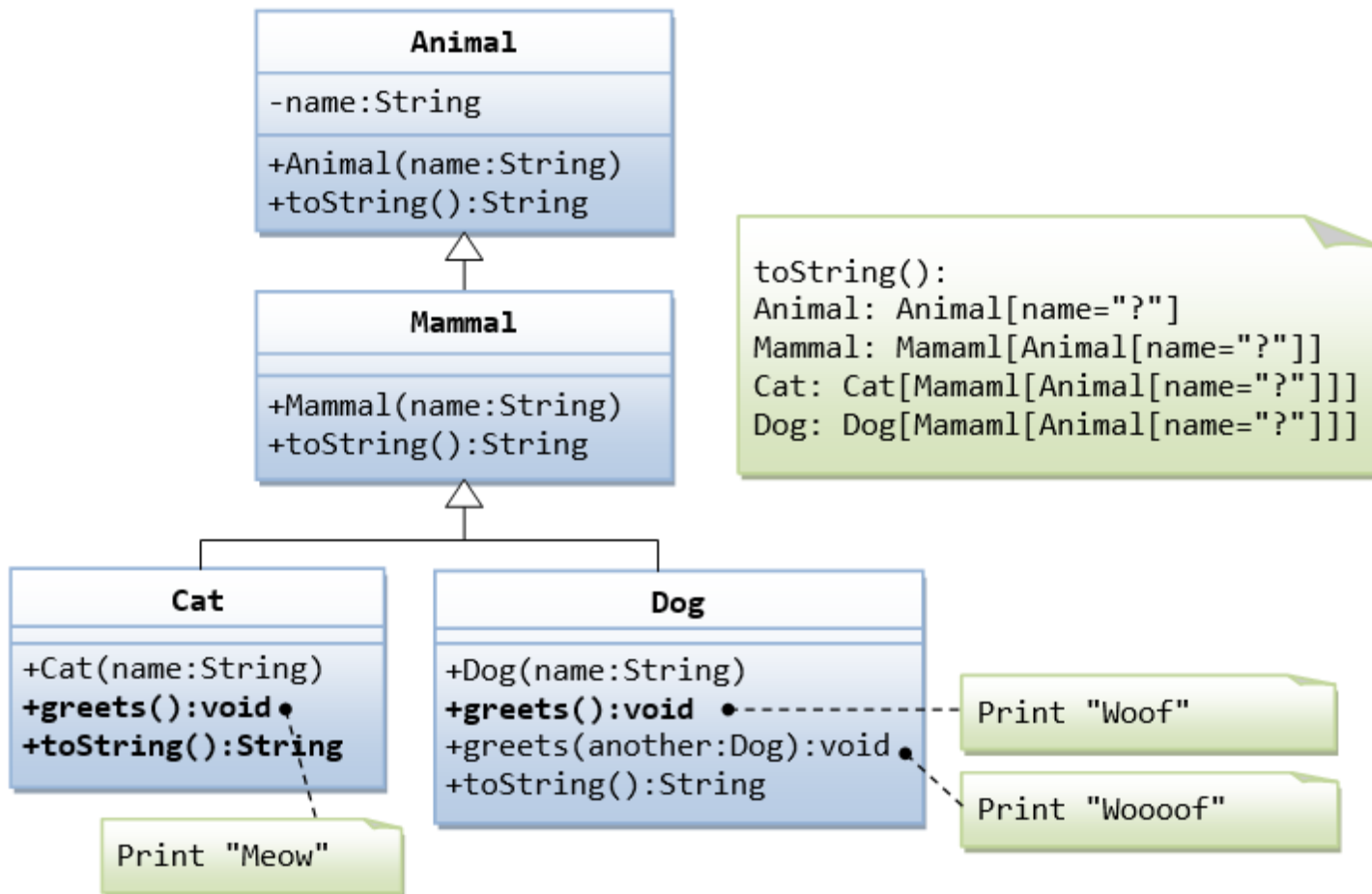
- Ejercicio 2. Una vez teniendo la clase `Fecha` que se utiliza en `Persona`, procedemos a definir la clase `Alumno`.
- La clase `Alumno` debe tener las mismas propiedades que `Persona`, más algunas otras que no toda persona tiene pero que todo alumno sí.
- Para no reescribir las propiedades de la persona cuando se define la clase `Alumno`, se indica que la clase `Alumno` hereda de `Persona`.
- `Alumno` tiene además el atributo privado `matricula` de tipo `String`.

Ejemplo

- Ejercicio 3. Ahora definamos la clase Profesor, la cual hereda también de Persona, pero añade propiedades que sólo tiene el profesor.
- Profesor tiene el atributo privado claveEmpleado de tipo Integer. En el constructor Profesor() se deben inicializar los atributos nombre, fechaNacimiento y claveEmpleado.

Ejercicio

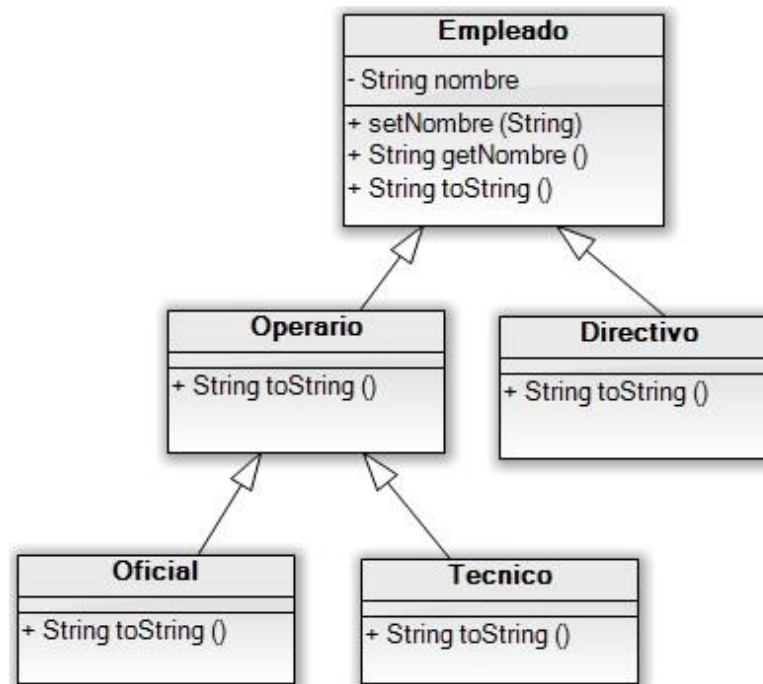
- Codifica el siguiente diagrama



Ejercicio

- Codifica la siguiente jerarquía de clases java representada por este diagrama UML:

-



Definición clase base y derivada

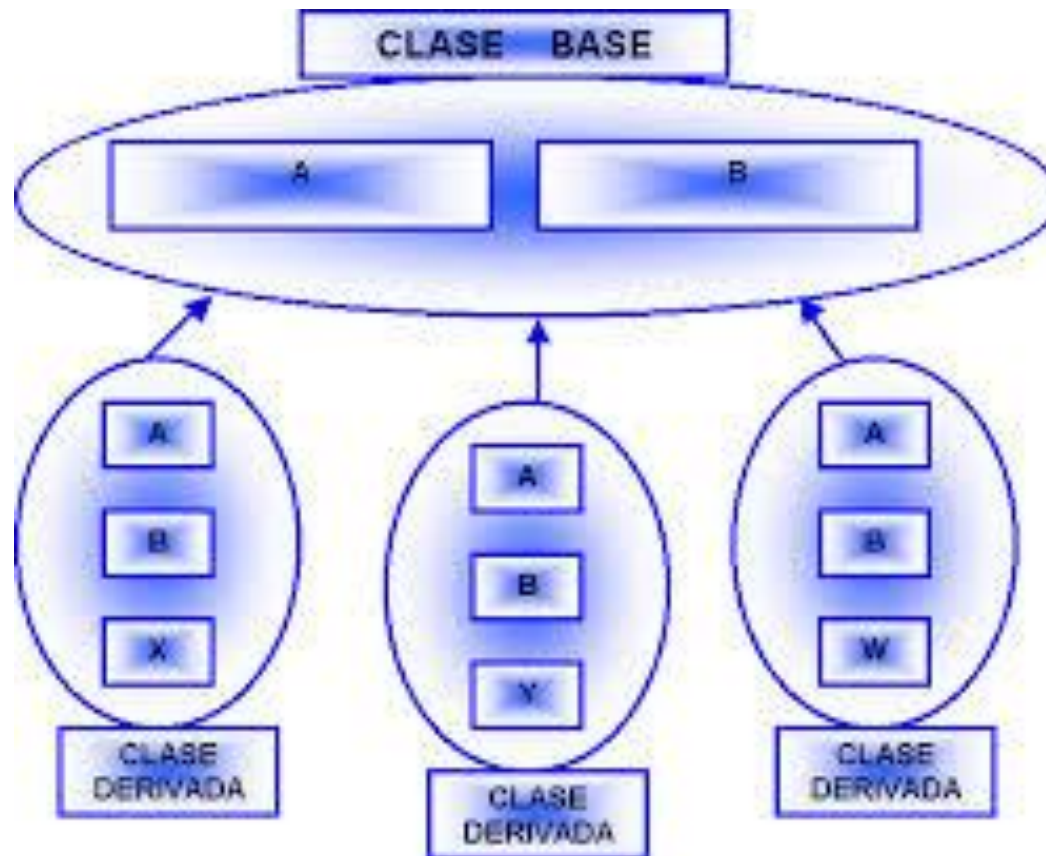
- Clase Base

- Una clase base es aquella que no dependen ninguno de sus atributos u objetos de la clase de alguna otra clase
- se podría decir que, en términos de herencia, sería la clase padre, la clase que se mantiene fija, en el aspecto de herencia.
- Es también por así llamarlo la clase principal de un programa, sería la clase primaria sin incluir la clase main en donde se corre todo el programa en si

Definición clase base y derivada

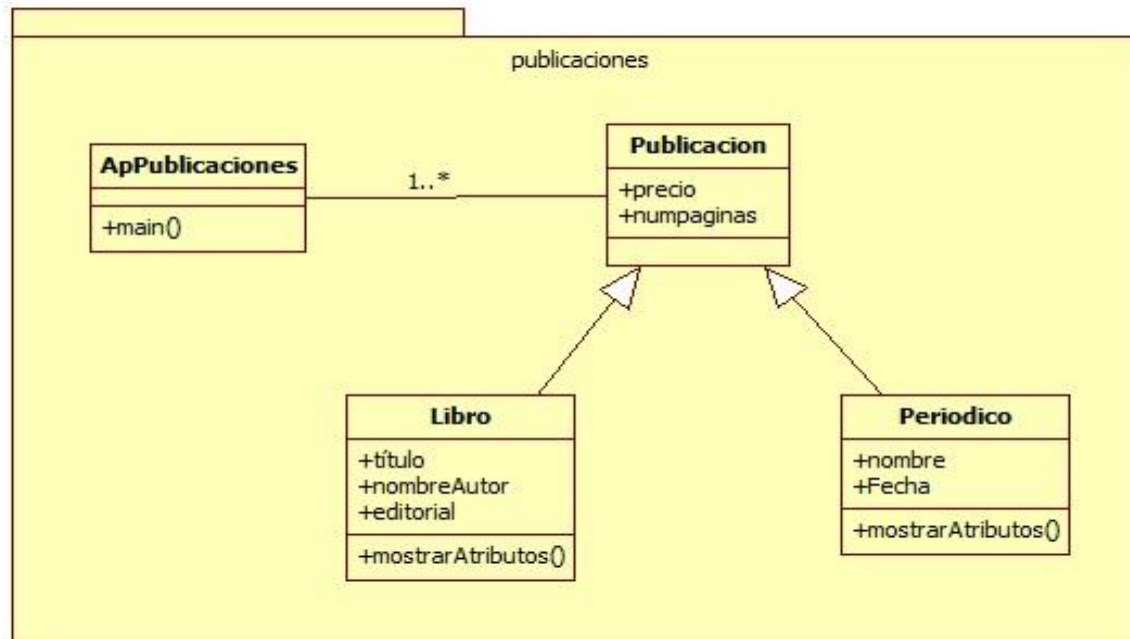
- Clase derivada
 - son clases que dependen de las clases bases, ya que algunos de sus métodos son también heredados, y muchas veces, el compilador arrojará malos resultados, ya que al ser dependientes estas clases, a veces podrán generar errores lógicos.

Clase base y derivada



Ejercicio:

- ▶ Realizar una aplicación en Java que cree la jerarquía siguiente y a partir de ella se creen varios objetos del Tipo Publicación
- ▶ Diagrama de clases de la aplicación:

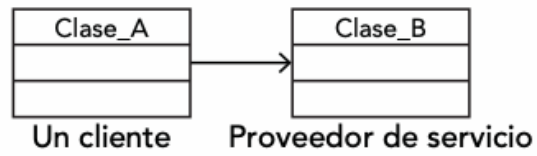


Niveles de asociación

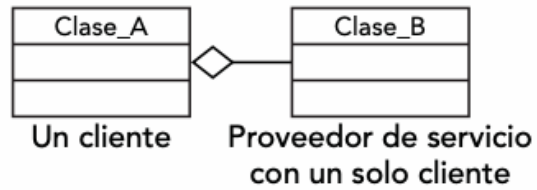
Asociación

- Se dice que un objeto de clase A está asociado con uno de clase B cuando el objeto de clase A hace uso de algún método del objeto de clase B.
- Existen tres niveles de asociación entre dos clases: la asociación simple, la agregación y la composición.

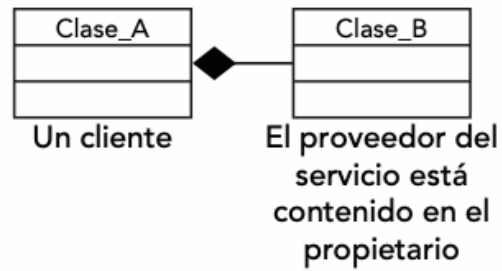
Asociación



Agregación



Composición



Asociación simple

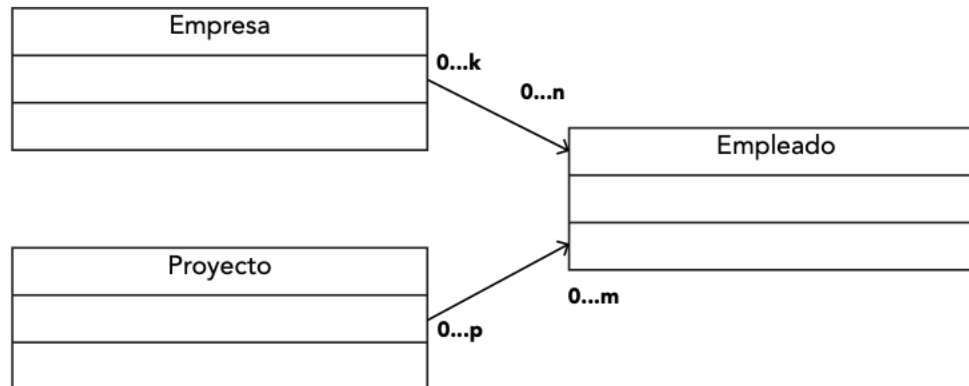
- Cuando una clase B está asociada a otra clase A se entiende que un objeto de clase B da servicio a otro de clase A.
- Este servicio puede exclusivo o compartido con otros objetos de la clase A o de alguna otra clase a la que se asocie la clase B
- Cada uno de los objetos es independiente del otro en cuanto a su creación o destrucción, es decir, si uno se crea o se destruye, el otro puede estar creado o no.
- La duración de la relación entre estos objetos es temporal, ya que la asociación puede crearse y destruirse en tiempo de ejecución sin que los objetos tengan que ser destruidos.

Asociación simple

- Por ejemplo, cuando se asigna un empleado para trabajar en dos proyectos.
- En este caso, el objeto empleado ya existe desde antes de que los dos objetos de clase Proyecto existan y seguirá existiendo aun cuando uno, o los dos proyectos terminen.
- Cada proyecto hace uso de las habilidades y conocimiento del empleado mientras el proyecto los necesite.

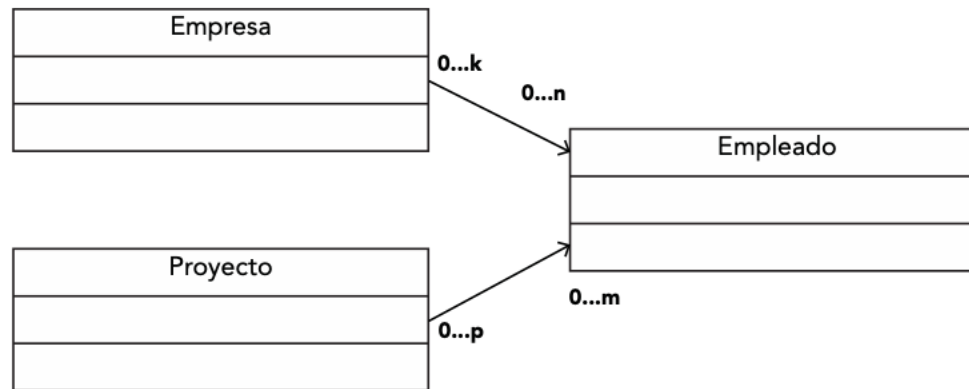
Ejemplo asociación

- Un objeto de clase Proyecto y un objeto de clase Empresa pueden necesitar cualquier cantidad de objetos de clase Empleado y un objeto de clase Empleado puede estar asignado a uno o a varios objetos de clase Proyecto y a uno o a varios objetos de clase Empresa.



Ejemplo de asociación

- Ejemplos: un empleado que trabaja en una empresa y también participa en proyectos fuera de su empresa, o un empleado que no tiene un trabajo fijo en una empresa pero que participa en varios proyectos, o un empleado que trabaja para dos empresas, etcétera.

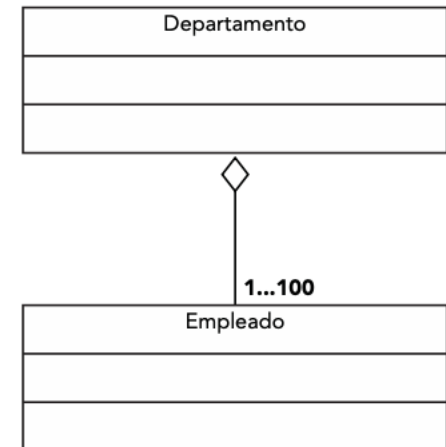


Agregación

- Cuando una clase B está agregada a una clase A se entiende que uno o varios objetos de la clase B le dan servicio exclusivo a un objeto de la clase A.
- En este caso, cada objeto de clase B agregado a uno de clase A puede reasignarse a otro objeto también de clase A.
 - Si el objeto de clase A desaparece, el objeto de clase B puede seguir existiendo, y este debe agregarse a otro objeto de clase A para que su existencia tenga sentido.

Agregación

- La agregación de un objeto de clase B a un objeto de clase A se hace mediante un método de la clase A, el cual recibe como parámetro la referencia al objeto que se le va a agregar, que ya fue **creado previamente**.
- Por ejemplo, a un objeto que se llama deptoVentas de la clase Departamento se le agregan varios objetos de la clase Empleado por medio del método agregarEmpleado que está en la clase Departamento:



Agregación

```
public class Departamento {  
    private static final int MAX_EMPLEADOS = 100;  
    private int clave;  
    private int numEmpleados;  
    private Empleado[] integrantes = new Empleado[MAX_EMPLEADOS];  
  
    public Departamento( int clave ) {  
        this.clave = clave;  
        numEmpleados = 0;  
    }  
  
    public Boolean agregarEmpleado( Empleado e ) {  
        if ( numEmpleados >= MAX_EMPLEADOS ) {  
            return false;  
        }  
        integrantes[numEmpleados] = e;  
        numEmpl++;  
        return true;  
    }  
  
    // los demas metodos de Departamento  
    ...  
}
```

Recibe una referencia al objeto e

El arreglo contiene ahora la referencia al objeto e

Paso por referencia

pass by reference



fillCup()

pass by value



fillCup()

Composición


- En una relación de composición entre A y B, en la que los objetos de la clase A tienen como componentes uno o más objetos de clase B.
- Los objetos de clase B son dependientes de la clase A ya que no pueden existir sin ser componentes de un objeto de clase A.
- Así, cuando desaparece el objeto de clase A, desaparecen también los objetos de clase B, porque no tiene sentido la existencia de B sin el objeto del que son componentes.

Composición

- En el siguiente ejemplo, un cliente puede tener hasta tres cuentas. Las cuentas no pueden existir si no existe ese cliente en particular(no pueden ser reasignadas a otro cliente).

```
Public class Cliente {  
    private int clave;  
    private String nombre ;  
    private int cont;  
    private Cuenta[] cuentas = new Cuenta[3];  
  
    public Cliente( int clave, String nombre ) {  
        this.clave = clave;  
        this.nombre = nombre;  
        cont = 0;  
    }  
  
    public Boolean agregarCuenta() {  
        if ( cont >= 3 ) {  
            return false;  
        }  
        cuentas[cont] = new Cuenta( this.clave );  
  
        cont++;  
        return true;  
    }  
  
    // los demás métodos de Cliente  
    ...  
}
```

La cuenta se instancia dentro de la clase Cliente



- ▶ Agregar a la aplicación una clase Contenedor que almacenara Libros / Periodicos y deberá permitir las operaciones de:
 - ▶ Insertar Publicación
 - ▶ Eliminar Publicación
 - ▶ Buscar Publicación
 - ▶ Desplegar Publicaciones