Programación Orientada a Objetos

Tema 2:
Fundamentos de la programación orientada a objetos

Tema 2-4: Conceptos básicos de POO 2

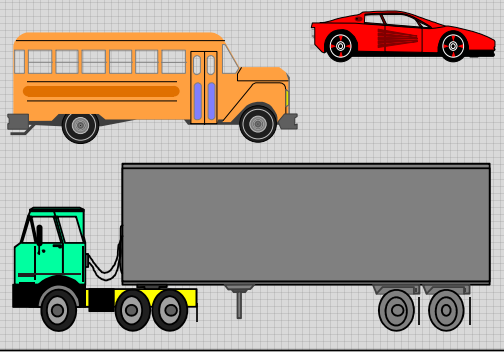
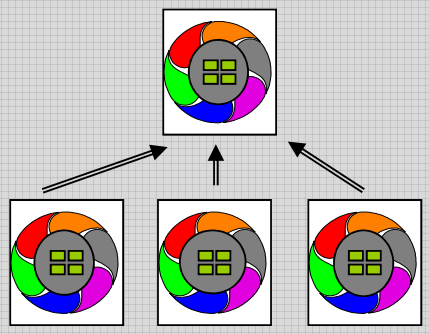
Universidad de Alcalá
Departamento Ciencias de la Computación

Contenidos

- Tema 2-4: Conceptos básicos de POO 2

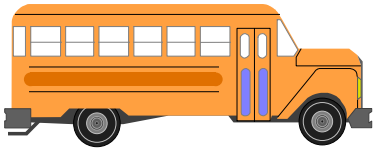
1. Super-clase/Herencia
2. Herencia en Java
3. Relaciones entre objetos
4. Asociación
5. Agregación/Composición
6. Cardinalidad
7. Ejemplo Java

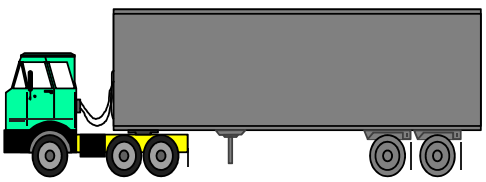
Super-clase

MUNDO REAL	SIMULACIÓN INFORMÁTICA
Las categorías de elementos pueden reagruparse en SUPER-CATEGORIAS : 	Las clases pueden reagruparse en SUPER-CLASES :  HERENCIA

Universidad de Alcalá
Departamento Ciencias de la Computación

Super-clase





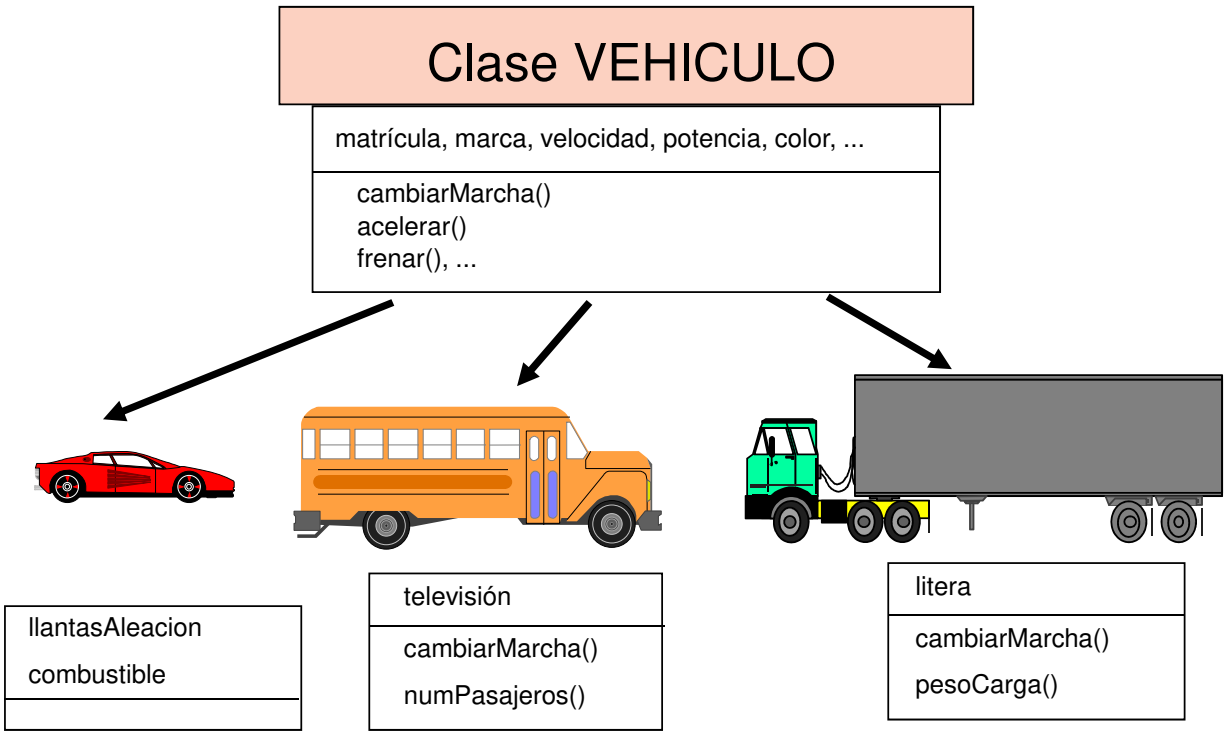
Automóviles, Autocares y Camiones se pueden agrupar dentro de la Super-Clase **VEHICULO**

Herencia

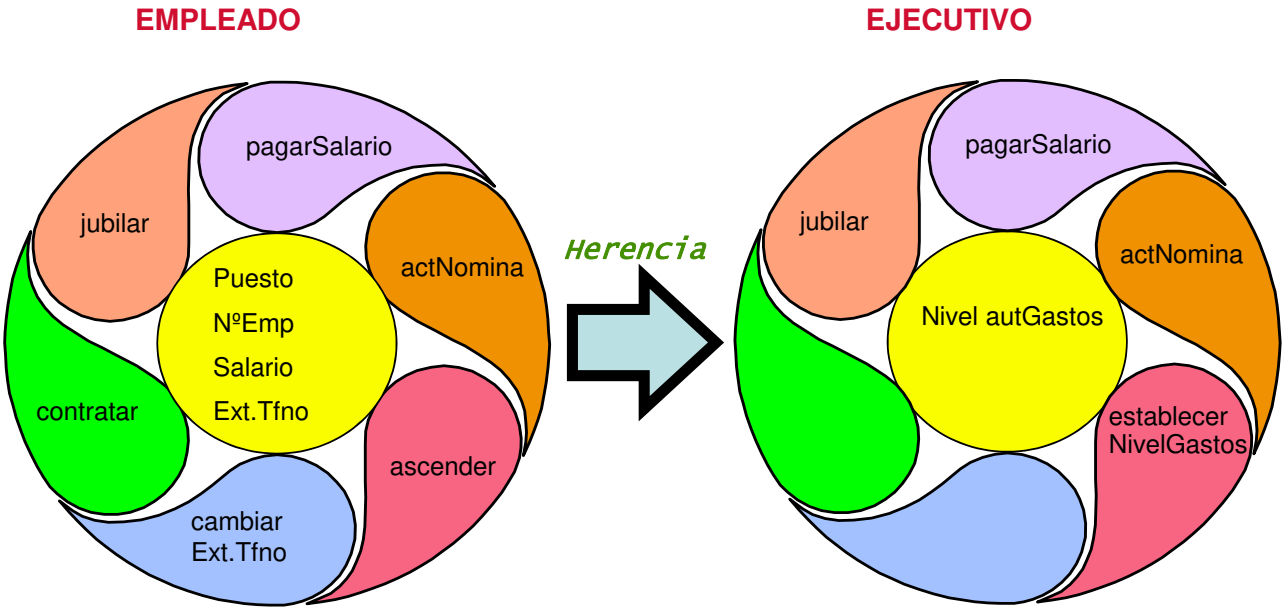
- Se establece una estructura jerárquica en la que cada clase hereda atributos y métodos de las clases que están por encima de ella. La clase derivada (Subclase) puede usar los procedimientos y los atributos de su Super-Clase.
- Cada subclase puede tener nuevos atributos y métodos y/o redefinir los heredados.
- Objetivo: permitir el análisis por clasificación.
- Ventajas: granularidad, reutilización.

Universidad de Alcalá
Departamento Ciencias de la Computación

Herencia

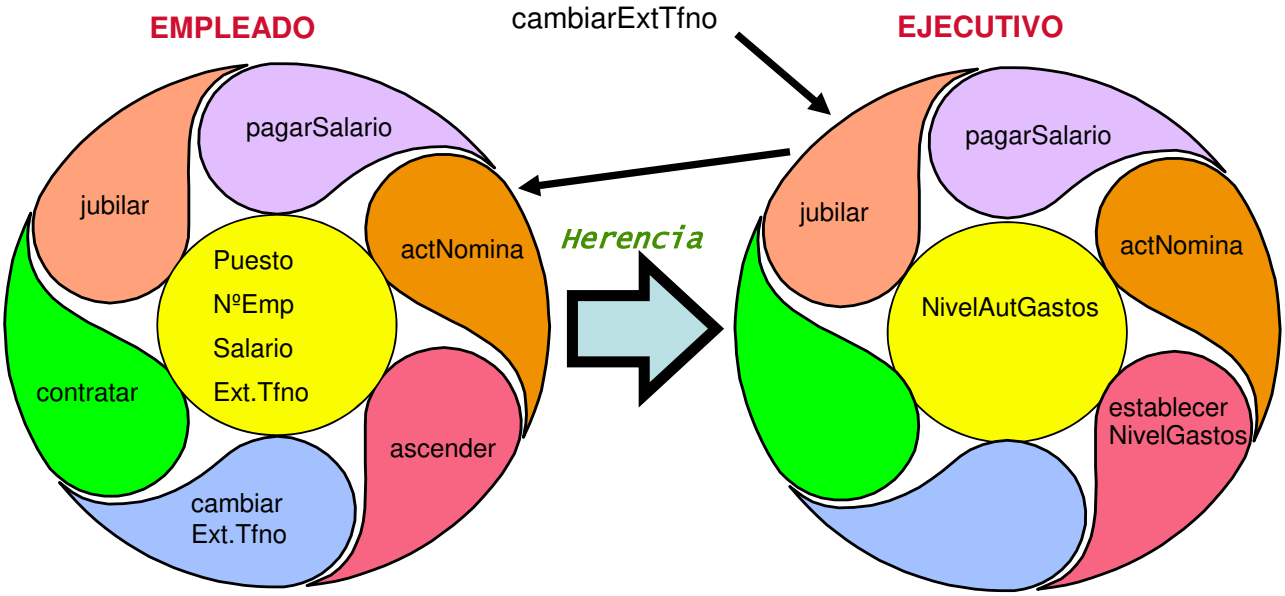


Herencia

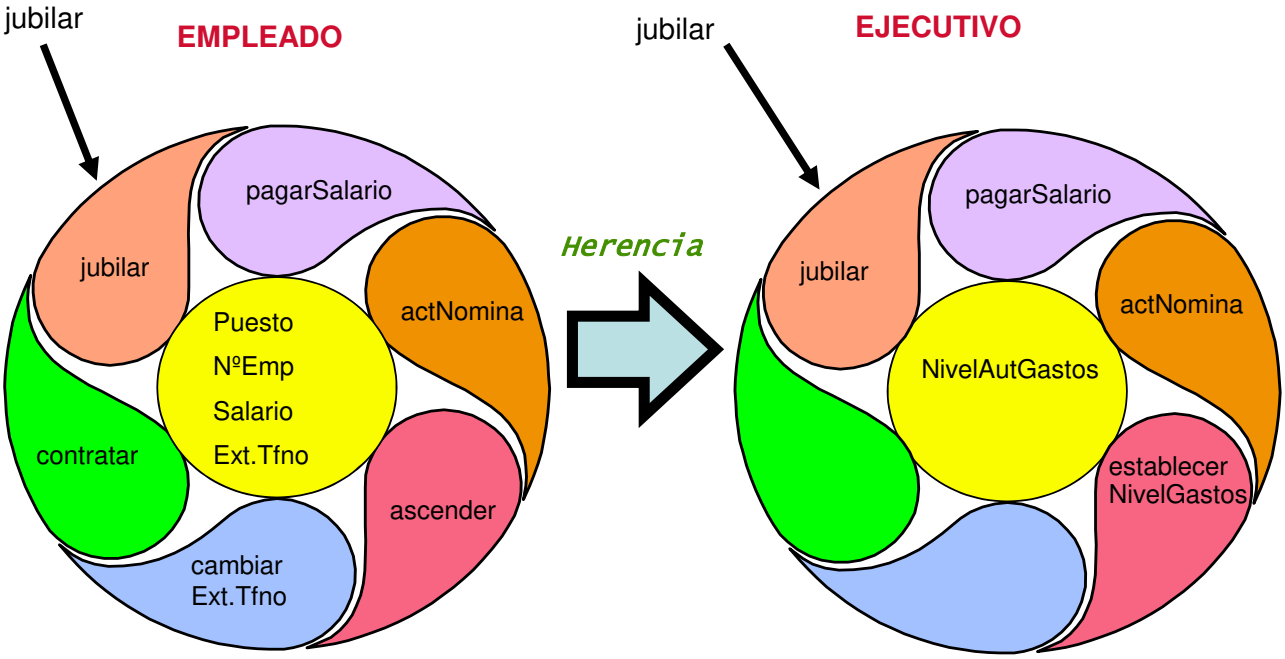


Herencia

- La búsqueda de los métodos y de los atributos sigue la jerarquía de las clases.
- Un objeto busca tanto los métodos y los atributos en su clase, después en sus super-clases.



Herencia



HERENCIA JAVA

• La **Herencia** es el mecanismo por el que se crean nuevos objetos definidos en término de objetos ya existentes.

• Para heredar de una clase utilizamos la palabra clave **extends**:

```
class Empleado extends Persona {
        . . .
}
```

La clase Empleado hereda todos los atributos y métodos (la interfaz y la implementación) de Persona. A partir de aquí, podemos:

- **Extender** la clase Persona añadiendo nuevos métodos a la clase Empleado. Aumentamos la interfaz.
- **Anular** métodos de Persona con la propia implementación de Empleado.

• En Java sólo es posible extender de una clase (***herencia simple***).

```
public class MiClase {
        int i;
        public MiClase() {
            i = 10;
        }
        public void suma_a_i( int j ) {
            i = i + j;
        }
}
```

```
public class MiNuevaClase extends MiClase {
        public void suma_a_i( int j ) {
            i = i + ( j/2 );
            }
}
```

```
MiNuevaClase mnc;
mnc = new MiNuevaClase();
mnc.suma_a_i( 10 );
```

- Ahora cuando se crea una instancia de MiNuevaClase, el valor de i también se inicializa a 10 porque los constructores también se heredan.
- Pero la llamada al método suma_a_i() produce un resultado diferente.

- **This:**
- La referencia **this** se usa para referirse al propio objeto y se usa explícitamente para referirse tanto a las variables de instancia como a los métodos de un objeto.
- Dentro de los métodos no estáticos, podemos utilizar el identificador especial **this** para referirnos a la instancia que está recibiendo el mensaje.
- Algunos usos habituales son:
  - Evitar conflictos de nombres con los argumentos de métodos.

```
class Persona {
        String nombre;
        public void setNombre(String nombre) {
                this.nombre = nombre;
        ...
```

  - Pasar una referencia de ese objeto a otro método.

```
Button incrementar = new Button("Incrementar");
incrementar.addActionListener(this);
```

- **Super:**

- Super se usa para referirse a métodos de la clase padre.

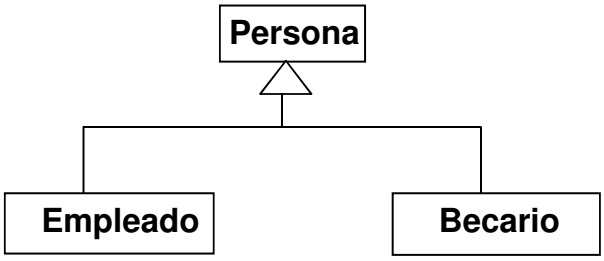
```
import MiClase;
  public class MiNuevaClase extends MiClase {
        public void suma_a_i( int j ) {
         i = i + ( j/2 );
        }
        public void suma_a_i_padre( int j ) {
         super.suma_a_i( j )
        }
  }
```

- Cuando extendemos una clase, los constructores de la subclase deben invocar algún constructor de la superclase como primera sentencia.

```
public Circulo(double radio) {
        super();
        this.radio = radio;
        }
```

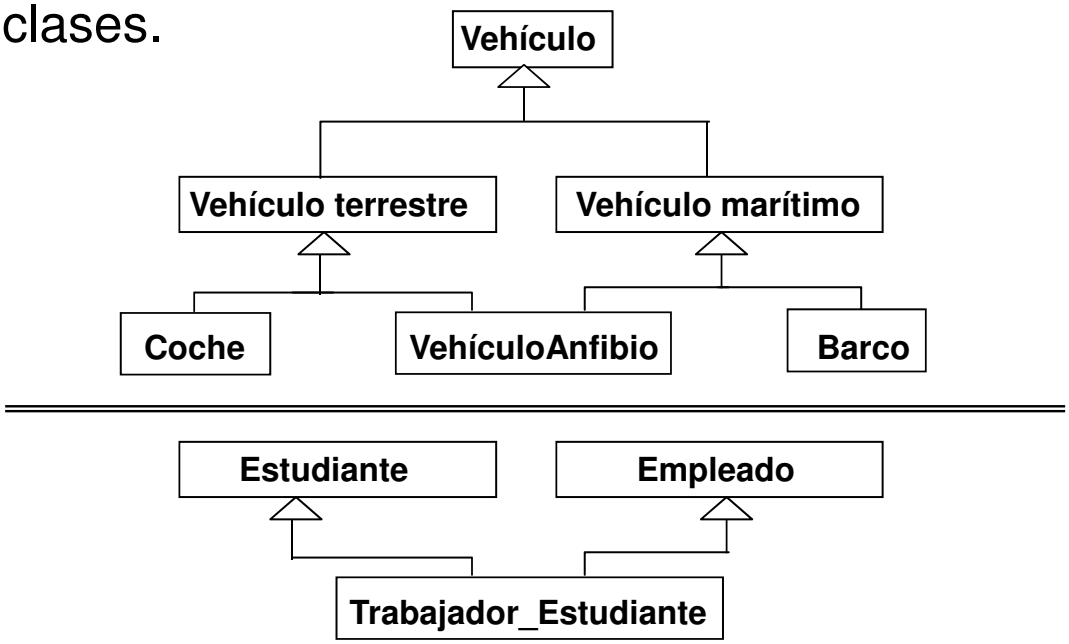
- Si no se invoca explícitamente un constructor de la superclase con super, debe existir un constructor sin parámetros en la superclase o no debe haber ningún constructor (el compilador genera uno por defecto sin parámetros).

- Simple: una clase hereda de una única super-clase.



Tipos de herencia

- Múltiple: una clase hereda de varias super-clases.



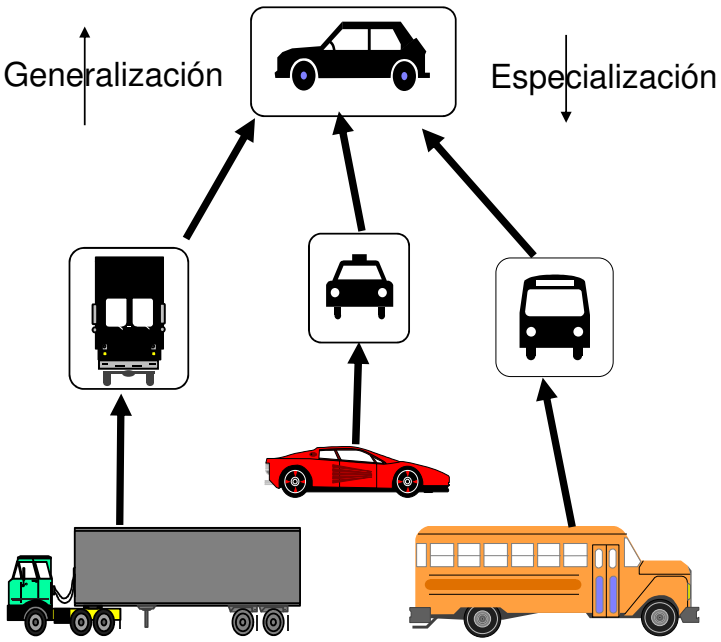
Tipos de herencia

H. Múltiple - Problemas:

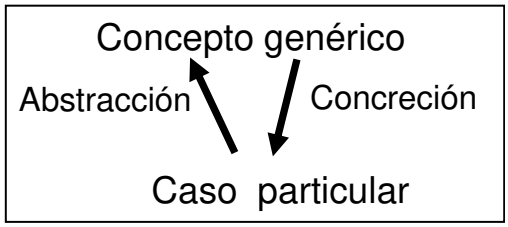
- Conflictos de nombres
- Conflictos de valores
- Conflictos de valores por defecto
- Conflictos de dominio
- Conflictos de restricciones

- Solución:
  - Lista de precedencia de clases.
  - Herencia selectiva: Algunas de las propiedades de las superclases se heredan selectivamente.

Jerarquías de clases

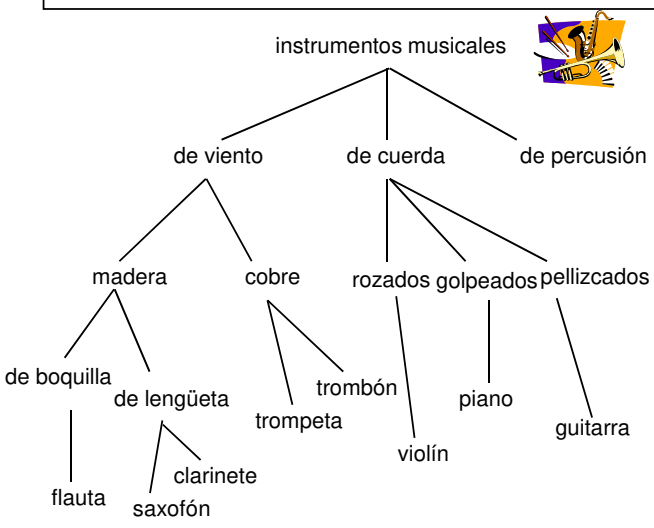


- Cada sub-clase puede establecer sus propias características, bien añadiéndolas a las definidas por la clase padre, o suprimiendo algunas de estas características.
- Para construir sistemas de software complejos y además comprensibles, la tecnología orientada a objetos utiliza nuestros mecanismos conceptuales innatos.

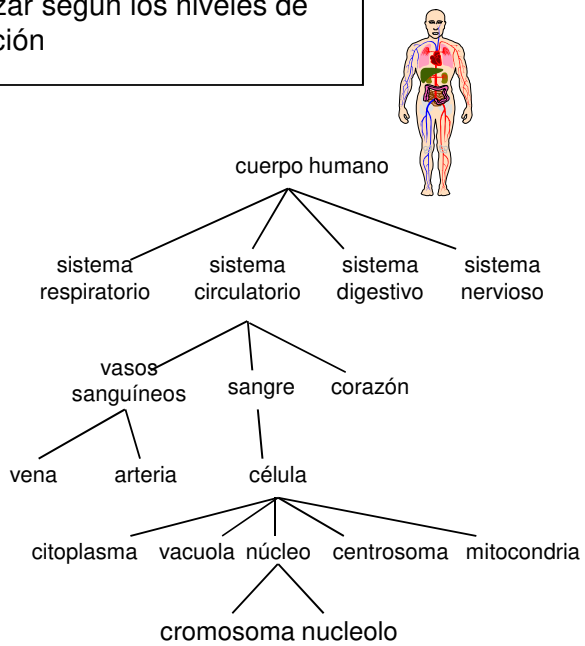


Clasificación vs. Composición

Clasificar y ordenar las abstracciones. Jerarquizar según los niveles de abstracción o según las relaciones de composición



Jerarquía "es-un" (herencia)



Jerarquía "parte-de" (compuesto-componente)

Herencia

- Ventajas:
  - Reutilización de software.
    - Mayor seguridad, menor coste.
  - Reutilización de componentes.
    - Rápido prototipado.
  - Consistencia de la interface.
    - Comportamiento similar de todas las clases que heredan de una superclase.

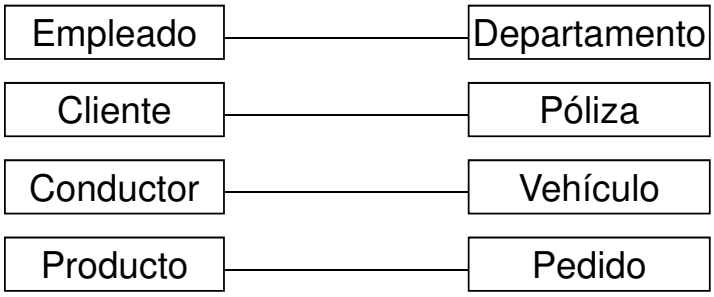
Relaciones entre Objetos

Los objetos tienen, aparte de atributos y métodos y relaciones de herencia, relaciones con otros objetos. Las relaciones más importantes son:

- **Asociación**
- **Agregación (composición)**

Asociación

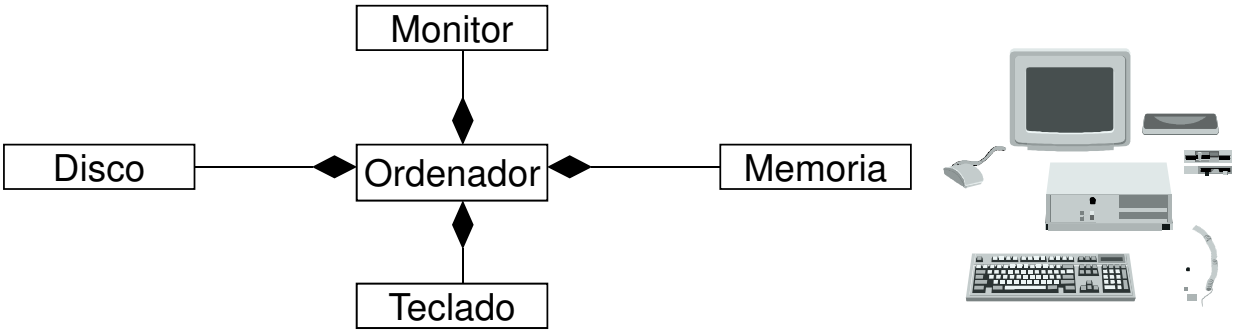
- Una asociación es una relación estructural que describe una conexión entre objetos. La idea es que los objetos se unen para trabajar juntos.
- Es una conexión entre clases, una conexión (enlace) semántica entre objetos de las clases implicadas en la relación. Esto se consigue usando algún objeto de una de las clases como atributo de la clase asociada.



Agregación/Composición

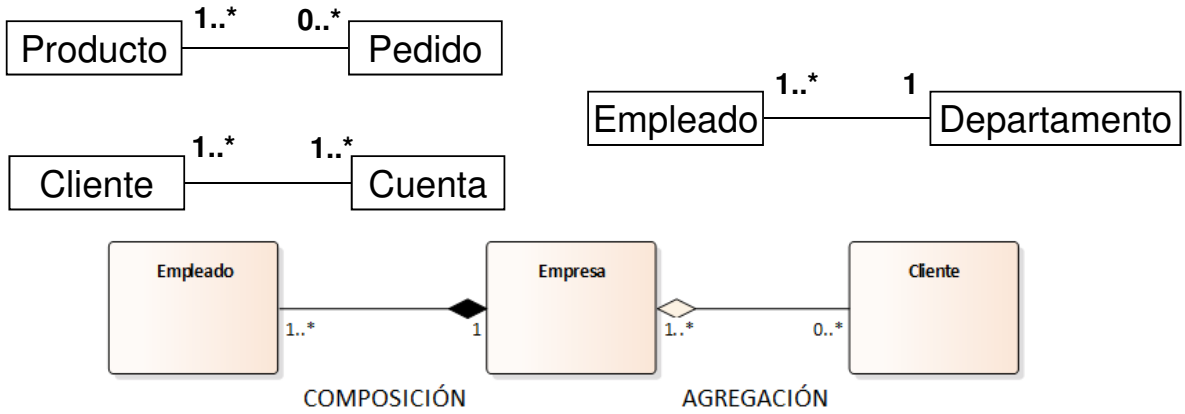
Se da cuando una clase está estructuralmente compuesta de otras clases.
Son relaciones de tipo parte-todo.
Una clase está formada por objetos de objetos (objeto compuesto) de otra u otras clases.

- Composición: para la existencia del objeto compuesto deben existir los demás objetos.
- Agregación: el objeto agregado puede existir sin la instancias de los demás objetos.



Cardinalidad

Las relaciones de Asociación y Agregación/Composición tienen cardinalidad:
Número de instancias de una clase que están relacionadas con una instancia de la otra clase.



EJEMPLO HERENCIA Y ASOCIACIÓN

- En Java se implementan las asociaciones mediante la inserción en una clase de un atributo que representa a otra clase (la clase asociada).

- A continuación veremos un ejemplo práctico donde usaremos herencia y asociación. Tenemos una clase Empleado que hereda de la clase Persona definida anteriormente y se asocia con la clase Departamento donde trabaja.

EJEMPLO HERENCIA Y ASOCIACIÓN

```
public class Departamento {
    // Atributos
    // Identificador del departamento
    private String id;
    // Nombre del departamento
    private String nombre;
    // Localización del departamento
    private String localizacion;

    //Constructor
    public Departamento(String id,
          String nombre,
          String localizacion) {
        this.id = id;
        this.nombre = nombre;
        this.localizacion = localizacion;
    }

    //Métodos
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getLocalizacion() {
        return localizacion;
    }
    public void setLocalizacion(String localizacion) {
        this.localizacion = localizacion;
    }
    @Override
    public String toString() {
        return "Departamento{" + "id=" + id
          + ", nombre=" + nombre
          + ", localizacion=" + localizacion + '}';
    }

}
```


EJEMPLO HERENCIA Y ASOCIACIÓN

```
public class Empleado extends Persona {

    //Atributos
    private String cargo;
    private double sueldo;
    //asociación
    private Departamento departamento;

    //Constructor
    public Empleado(String dni,
          String nombre,
          int edad,
          String estado,
          String cargo,
          double sueldo,
          Departamento departamento) {
        super(dni, nombre, edad, estado);
        this.cargo = cargo;
        this.sueldo = sueldo;
        this. departamento = departamento;
    }

    //Métodos
    public String getCargo() {
        return cargo;
    }
    public void setCargo(String cargo) {
        this.cargo = cargo;
    }
    public double getSueldo() {
        return sueldo;
    }
    public void setSueldo(double sueldo) {
        this.sueldo = sueldo;
    }
    public void subeSueldo(int cantidad) {
        sueldo += cantidad;
    }
    public Departamento getDepartamento() {
        return departamento;
    }
    public void setDepartamento(Departamento departamento) {
        this.departamento = departamento;
    }
    @Override
    public String toString() {
        super.toString() + "\nEmpleado{"
    + "cargo=" + cargo + ", sueldo=" + sueldo
    + ", departamento=" + departamento + '}';
    }
}
```

EJEMPLO HERENCIA Y ASOCIACIÓN

```
public class PruebaEmpleados {
    public static void main(String args[]) {
        //Creamos un departamento
        Departamento informatica = new Departamento("1Inf", "Informática", "Madrid");
        //Creamos empleados
        Empleado emp1 = new Empleado("153647458S", "Pepe", 35, "casado", "Analista", 1500, informatica);
        Empleado emp2 = new Empleado("452697419Z", "Maria", 25, "soltera", "Programadora", 1000, informatica);

        //Aplicamos métodos
        System.out.println(emp2.toString());
        emp2.subeSueldo(100);
        System.out.println("Sueldo: " + emp2.getSueldo());
    }
}
```

Ejecución:

Persona{dni=452697419Z, nombre=Maria, edad=25, estado=soltera}
Empleado{cargo=Programadora, sueldo=1000.0, departamento=Departamento{id=1Inf, nombre=Informática, localizacion=Madrid}}
Sueldo: 1100.0