Clases

- Herencia
- Generalización y Especialización
- Clases abstractas
- Reglas de visibilidad
- Herencia simple vs. herencia múltiple

Panorama General de las Clases

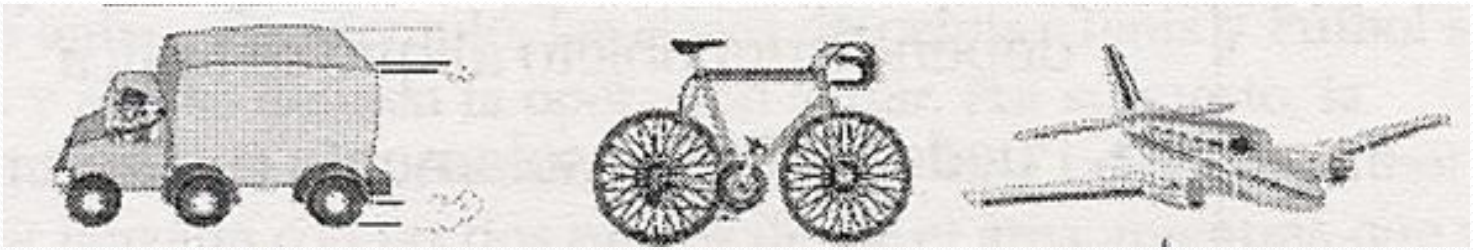
Objetos de una Clase Hoja

Objetos de una Clase Pelota

- Un objeto es una instancia de: una hoja, pelota, coche o moneda específico.
- “Un coche” es una clase: “mi coche” es un objeto

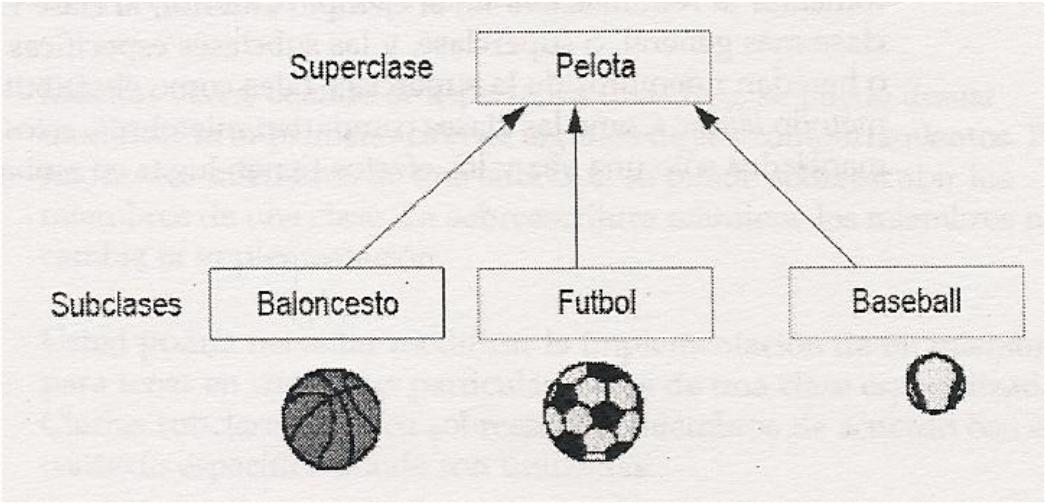
Generalización

- Ejemplo: Transporte es la generalización de varias clases que ofrecen servicios de traslado
- La Generalización identifica y define los atributos y operaciones comunes en una colección de objetos
- Transporte -> moverse de un lado a otro



Herencia

- Ejemplo: La Clase PelotaFutbol puede heredar los miembros de la Pelota
  - Método: lanzar
  - Atributos: forma y tamaño

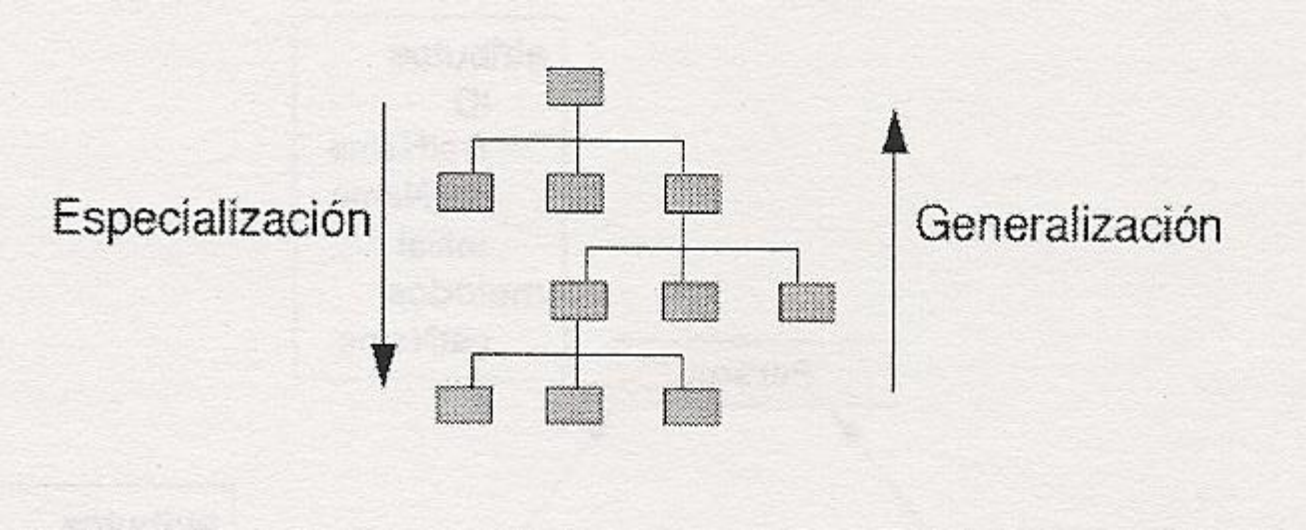


Herencia

- Es un mecanismo para definir una nueva clase a partir de una clase existente
- Permite agrupar clases relacionadas, de forma que puedan ser manejadas colectivamente
- Promueve el reúso
- Permite ocultar y sobreescribir métodos heredados
- Generalización, Especialización y Sobreescritura

Herencia

- Suele representarse como un árbol
  - Hacia las hojas: refinamiento
  - Hacia la raíz: más general



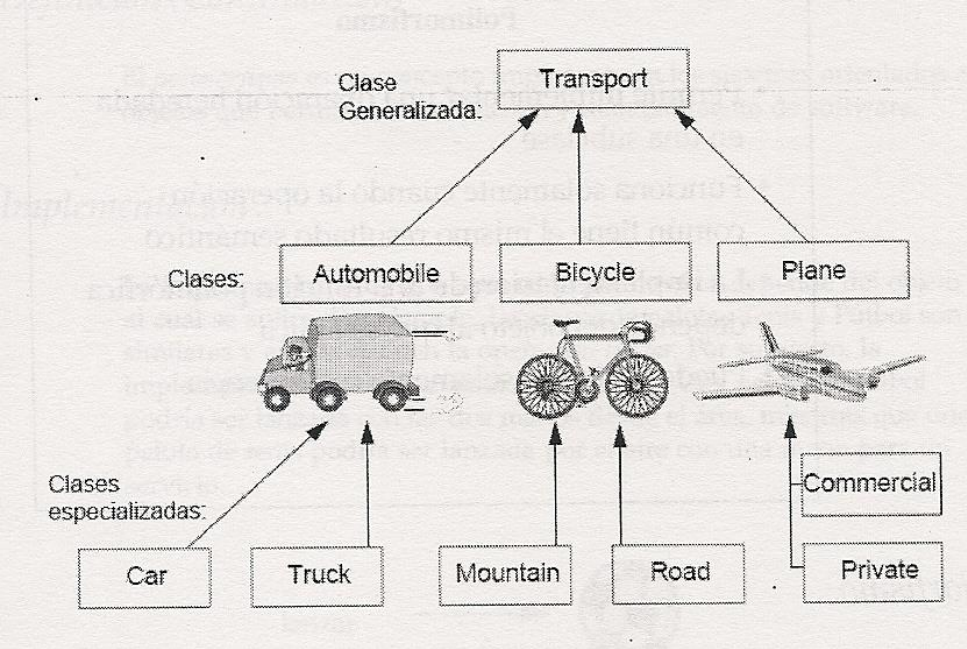
Herencia

Ejercicio: Encontrar la jerarquía de clases adecuada para un Sistema de empleados con las siguientes características:

- Se solicita realizar un Sistema de empleados que almacene el nombre, el salario y fecha de cumpleaños.
- El organigrama de la empresa determina que está organizada en diferentes departamentos, siendo que cada departamento está administrado por un gerente, quienes a su vez tienen una cochera asignada.
- El director de la empresa, además de tener su cochera, tiene recibe un extra a su salario para los gastos del vehículo.

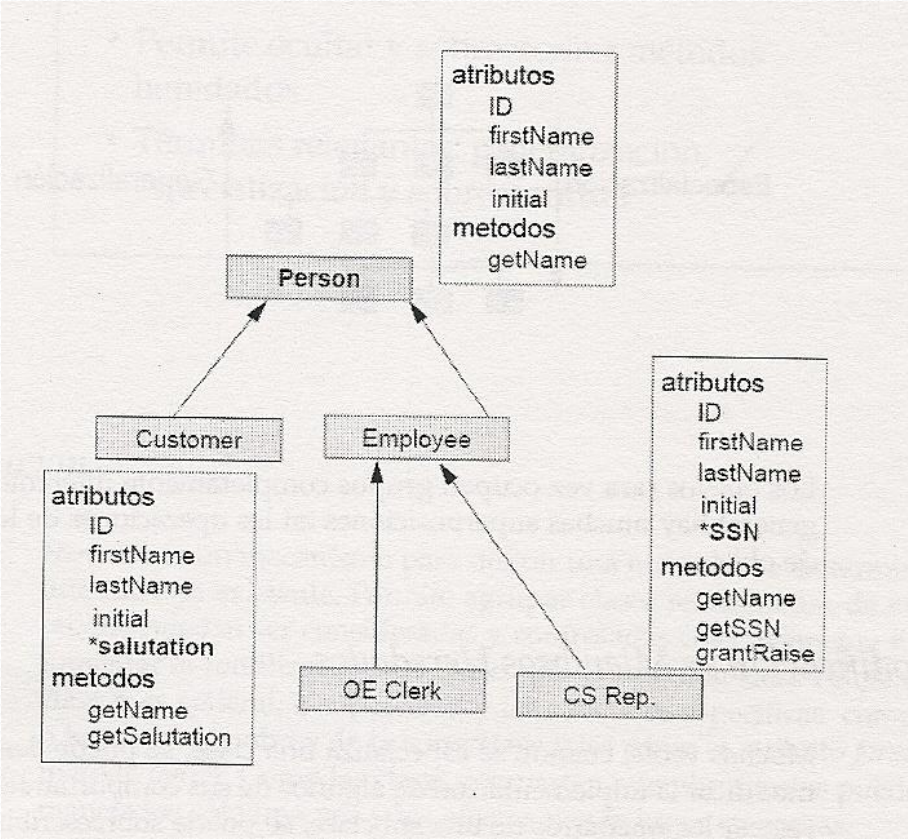
Especialización

- La especialización es una herencia con métodos agregados y modificados para resolver un problema específico



Especialización

- Ejemplo:



Herencia

- Ejemplo: Una compañía necesita u sistema de personal que almacene objetos Gerente, Ingeniero, Vendedor, etc.
- Las Clases Manager(Gerente) y Clerk (Vendedor) tienen características de un Employee (Empleado)
- Los Ítems comunes se definen en una clase y las siguientes clases se basan en ella

Verificación de herencia

- Una clase puede heredar únicamente de una superclase
- Es importante utilizarla solamente si es válido e inevitable
- Frase “es un/a”
  - Un Gerente “es un” Empleado

```
class Manager extends Employee
{
   int numberOfWorkers;
   //etcetera.
}
```

UML – Diagrama de Clase

- Representa la estructura estática de un sistema. Estos diagramas están compuestos por clases y sus relaciones

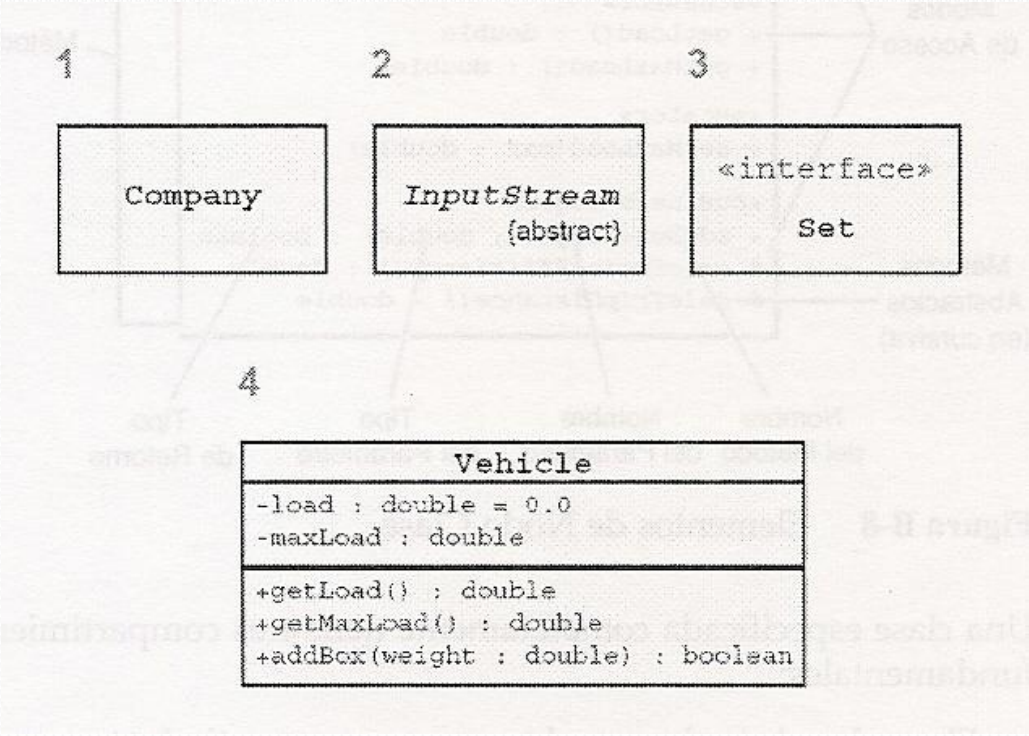


Diagrama de Clase – Elementos de un Nodo

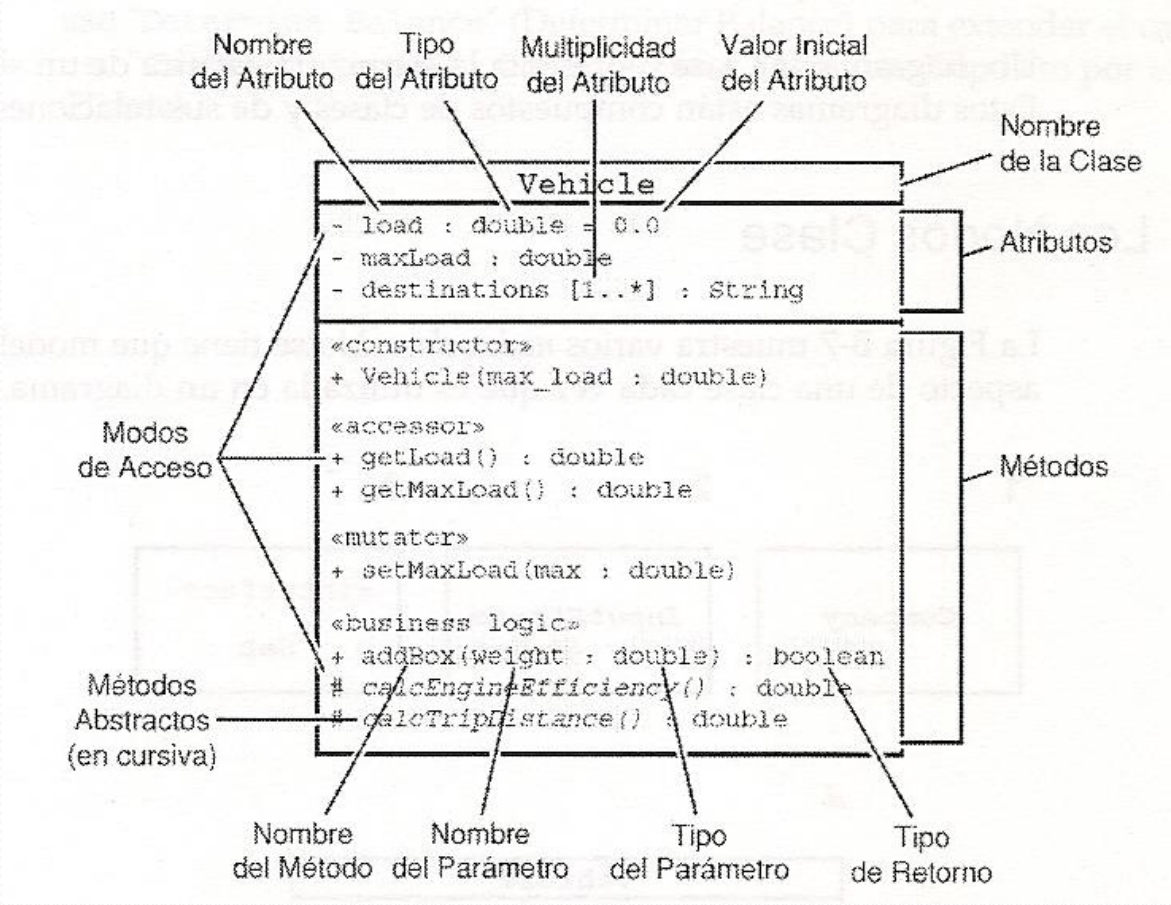
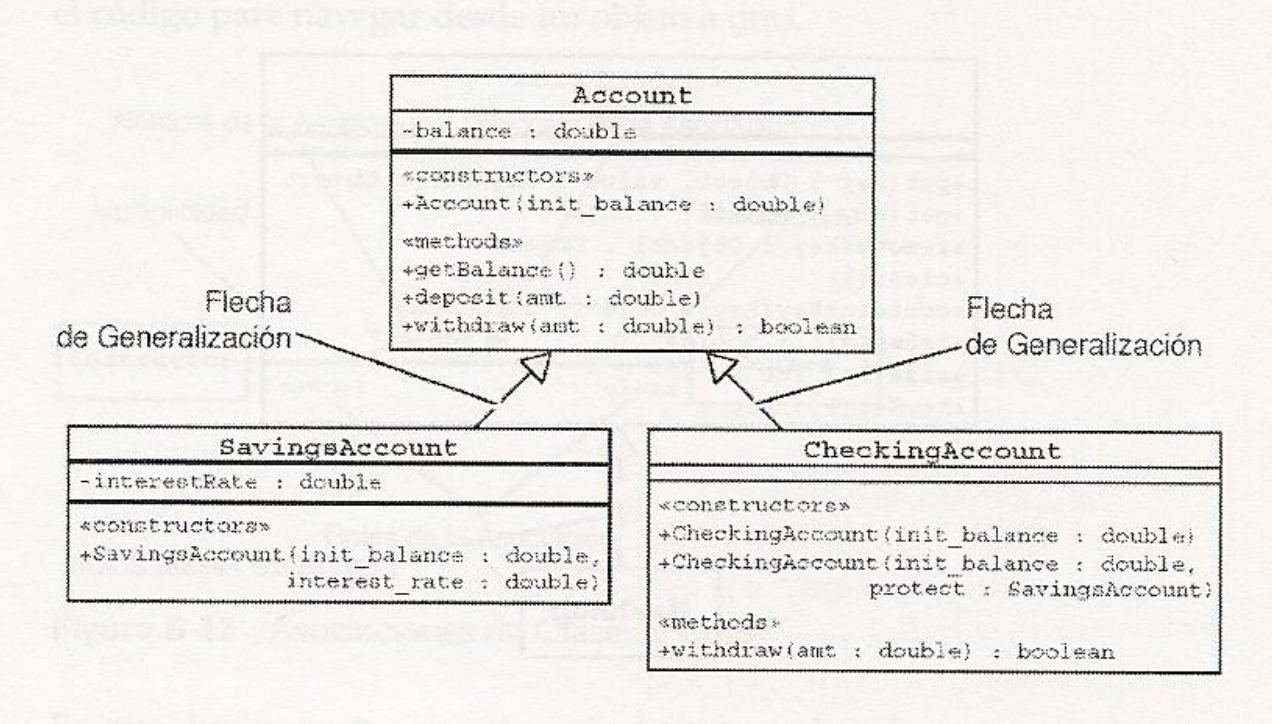


Diagrama de Clase – La Herencia



Ejemplo: descubriendo herencias

```
class Transporte {
  // omitiremos los atributos
  Double calcularIndiceDesgasteTotal() { ... }
  Boolean debeArreglarse() { ... }
  void agregarPaquete(Paquete paquete) { ... }
  Double capacidadRestante() { ... }
}
```

Un pequeño reconto de los métodos nos permite saber que:

- `calcularIndiceDesgasteTotal():Double` devolverá un número entre 0 y 1 que nos indicará el desgaste del vehículo.
- `debeArreglarse():Boolean` se basará en cuán desgastado está para indicar si debe o no pasar por el taller.
- `agregarPaquete(Paquete):void` permitirá agregar carga.
- `capacidadRestante():Double` nos informará cuántos kilogramos pueden cargarse aún.

Nuestro “mini mundo” cambia

```
class Transporte {
  String tipo = "bicicleta";
  Transporte () { }
  Transporte (String tipo) { this.tipo = tipo; }
  Boolean debeArreglarse() {
    Double valorLimiteDesgaste
    if ("bicicleta".equals(this.tipo)) {
      valorLimiteDesgaste = 0.3;
    } else {
      valorLimiteDesgaste = 0.5;
    }
    return this.desgasteActual < valorLimiteDesgaste;
  }
  // se omiten los otros métodos para simplificar el ejemplo
}
```

```
class Transporte {
  Double calcularIndiceDesgasteTotal() { ... }
  Boolean debeArreglarse() { ... }
  void agregarPaquete(Paquete paquete) { ... }
  Double capacidadRestante() { ... }
}

class Bicicleta extends Transporte {
  Double calcularIndiceDesgasteTotal() {
    // implementación específica de la bicicleta
  }
  // ...
}

class Automovil extends Transporte {
  Double calcularIndiceDesgasteTotal() {
    // implementación específica del automóvil
  }
  // ...
}

class Camion extends Transporte {
  Double calcularIndiceDesgasteTotal() {
    // implementación específica del camión
  }
  // ...
}
```

- Una `Bicicleta` es un `Transporte`
- Un `Automovil` es un `Transporte`
- Un `Camion` es un `Transporte`

Herencia y composición

```
class Boca {
  void comer(Alimento alimento) { ... }
}
```

```
class Persona extends Boca {
}
```

Cómo debimos haberlo hecho

- Simplemente necesitábamos reutilizar el comportamiento, pero no generar esa relación

```
class Persona {
  private Boca boca;
  Persona() { ... }
  void comer(Alimento alimento) {
    this.boca.comer(alimento);
  }
}
```

Especialización/generalización

- **Sobreescritura de métodos**

```
class Transporte {
  Double calcularDesgaste() {
    return kilometrosRecorridos / kilometrosTotalesPosibles;
  }
}

class Camion extends Transporte {
  @Override Double calcularDesgaste() {
    return kilometrosRecorridos / kilometrosTotalesPosibles + cargaTransportada / cargaTotalPosible;
  }
}
```

Clases abstractas

- Ciertos conceptos de nuestro software no tienen sentido más que taxonómico: sólo nos sirven para formar jerarquías

```
abstract class Transporte {
  abstract void cargarPaquete(Paquete paquete);
}
```

Reglas de visibilidad

- Si la superclase define un miembro como **privado**, la subclase no podrá accederlo bajo ningún concepto. Las clases ajenas a la jerarquía, como es de esperar, tampoco.
- Si la superclase define un miembro como **protegido**, la subclase podrá accederlo como si fuera propio pero privado. Las clases ajenas a la jerarquía no podrán accederlo.
- Si la superclase define un miembro como **público**, la subclase lo poseerá con la misma visibilidad. Las clases ajenas a la jerarquía podrán acceder al mismo.