La Sénia
Institut d'Educació Secundària
Paiporta

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

POO. Conceptos avanzados

Curso 2023 - 2024

1. POO

Conceptos:

- Relación entre clases
- Agregación/Composición
- Herencia. Superclases y subclases
- Clases Abstractas y finales
- Herencia de Constructores. Sobreescritura
- Interfaces
- Polimorfismo

2. Relaciones entre las clases

Las clases y los objetos no existen de modo aislado.

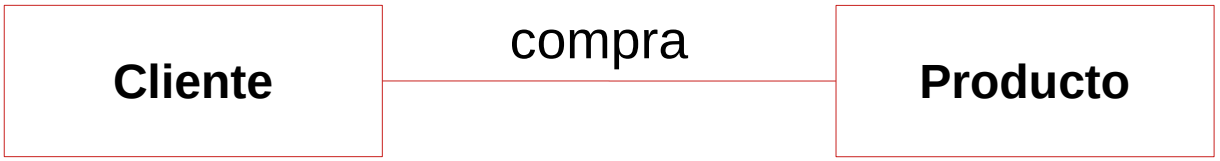
Existen relaciones entre ellas:

- **Asociación**: son relaciones entre objetos independientes
- **Agregación/Composición**: relaciones de todo-parte, tiene-un, es-parte-de
- **Dependencia**: Una clase usa los atributos o métodos de otra clase para realizar algo
- **Herencia**: Una clase se **especializa** en otras clases y unas clases se **generalizan** en una clase superior

3. Agregación y composición

La **Asociación** permite relacionar objetos sin ninguna relación entre sus clases.

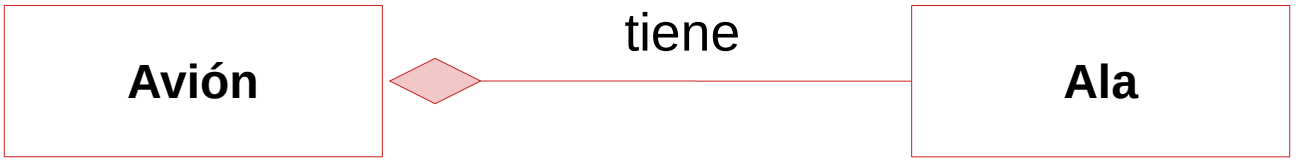
Pueden participar distintas instancias en lo que se define como multiplicidad o cardinalidad: 1 a 1, 1 a N, N a N



3. Agregación y composición

En la **Composición** una clase es **parte** de otra clase **todo** (objeto contenedor que tiene otros objetos). La clase parte es atributo de la clase todo.

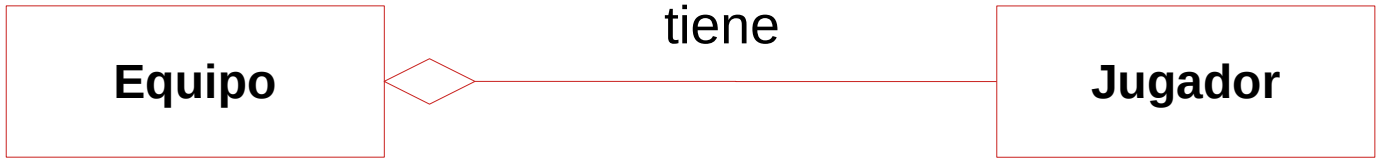
Las partes sólo se pueden acceder a través del compuesto, no de forma independiente. No puede existir una sin la otra



3. Agregación y composición

En la **Agregación** la clase contiene objetos y no otras clases.

Las partes se pueden acceder de forma independiente al compuesto.



3. Agregación y composición

En la relación de **Dependencia** la clase usa o necesita otra clase, es una relación de dependencia o instanciación. La relación no es persistente, al finalizar la misma termina el contacto entre los objetos.

En el ejemplo, la clase Ecuación necesita recurrir a la función sqrt de la clase Math para calcular la raíz cuadrada.

4. Herencia

La **Herencia** es una relación entre clases que comparten estructura y comportamiento.

Herencia simple: Una clase comparte la estructura y el comportamiento de una sola clase superior.

Herencia múltiple: Una clase comparte la estructura y comportamiento de varias clases superiores.

Java sólo permite implementar **herencia simple**

- Reutilización de código.
- Organización de clases en estructuras jerárquicas.

4. Herencia

Jerarquía entre dos clases:

La clase superior se denomina **superclase** o clase padre. La clase que hereda se llama **subclase**, clase derivada o clase hija.

La herencia transmite tanto atributos como métodos.

La relación de herencia es **transitiva**: A hereda de B y B hereda de C, por lo que A hereda de C

4. Herencia

Especialización

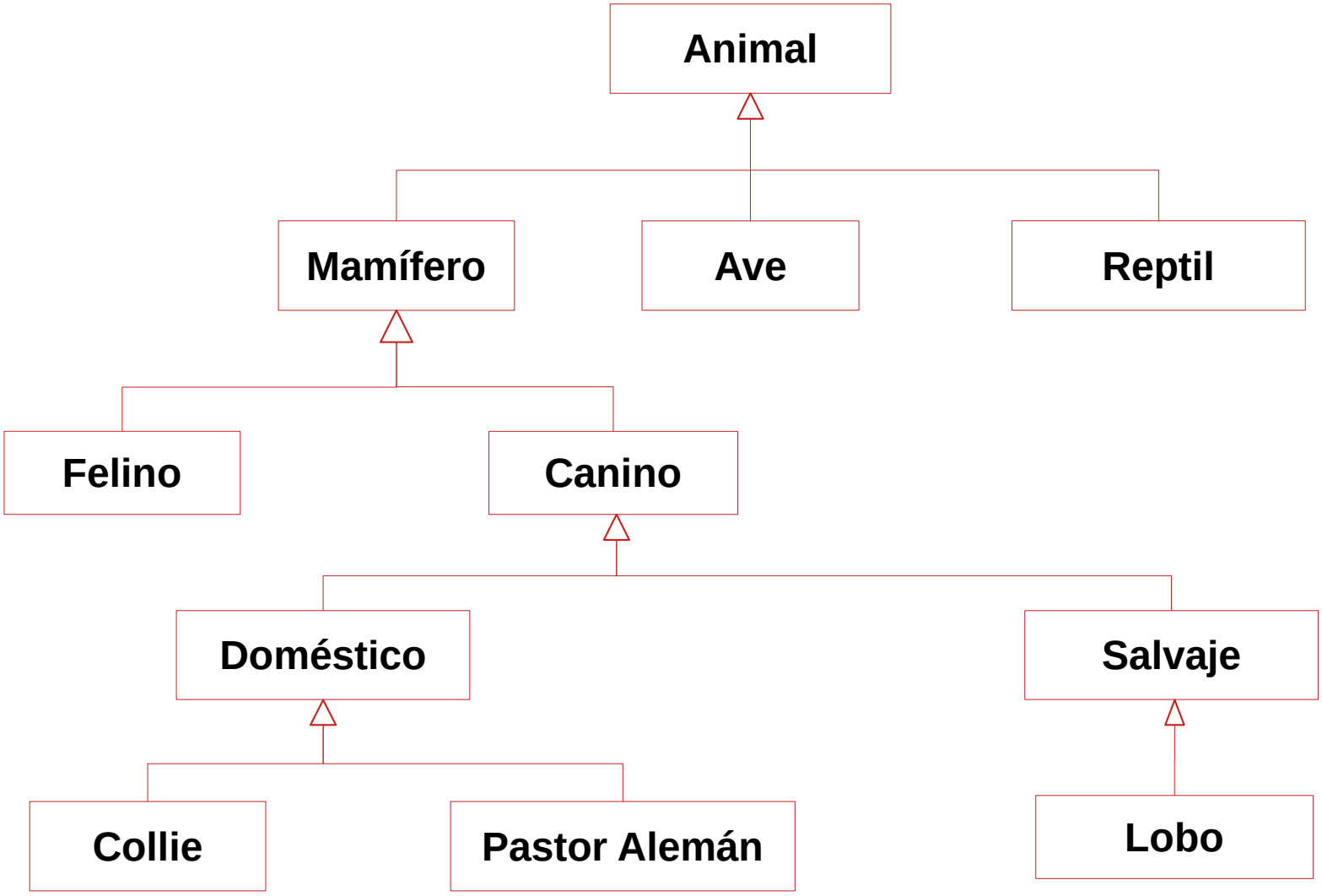
La superclase se **especializa** en subclases.

Generalización:

La **abstracción** de las subclases que reune los métodos y atributos comunes que contenga una superclase

- Una **superclase** se detalla en las subclases.
- Una **subclase** se generaliza o abstrae en la superclase.

4. Herencia


Animal
Mamífero
Ave
Reptil
Felino
Canino
Doméstico
Salvaje
Collie
Pastor Alemán
Lobo

4. Herencia

```
[modificador] class ClasePadre {
    //Atributos
    [public | private | protected ] tipoDato nombreVariable;
    //Constructores
    [public | private | protected ] constructor(tipoDato Valor) {
        this.nombreVariable=Valor;
    }
    //Métodos
}
```

```
[modificador] class ClaseHija extends ClasePadre {
    //Atributos
    [public | private | protected ] tipoDato nombreVariable;
    //Constructores
    //Métodos
}
```

5. Clases Abstractas y finales

Clase Abstracta

- No va a tener instancias (objetos) de forma directa.
- Se crean instancias en las subclases
- Se utiliza para herencia. Todas las subclases la heredan
- Una clase abstracta pueden tener métodos totalmente definidos (**no abstractos**) y métodos sin definir (**abstractos**)

```
[modificador] abstract class ClaseAbstracta {
    // Cuerpo de la clase
    ...
}
```

5. Clases Abstractas y finales

Método Abstracto

- Método sin implementar declarado en una clase.
- Si una clase tiene al menos un método abstracto es una clase abstracta.
- Toda subclase (obligatoriamente) debe implementar el **método abstracto** o declararlo como abstracto.

```
[modificador] abstract <tipo> metodoAbstracto ([parametros]){
    // Cuerpo del método
    ...
}
```

5. Clases Abstractas y finales

Clase Final

- Es una clase que no puede ser heredada por subclases.
- La jerarquía de clases termina en ella.

```
[modificador] final class ClaseFinal {
    // Cuerpo de la clase
    ...
}
```

Método Final

- No podrá ser redefinido en una clase derivada (error de compilación)

```
[modificador] final <tipo> metodoAbstracto ([parametros]){
// Cuerpo de la clase
...
}
```

6. Herencia de Constructores

Un **constructor** de una clase puede llamar a otro constructor de la misma clase con **this()** en la primera línea.

```
public class Persona{

    private String nombre;

    public Persona () { }

    public Persona (String nombre, String apellidos) {
        this.nombre= nombre;
        this.apellidos= apellidos;
    }

    public Persona (String nombre, String apellidos, GregorianCalendar fechaNacim) {
        this(nombre,apellidos);
        this.fechaNacim= new GregorianCalendar (fechaNacim);
    }

}
```

6. Herencia de Constructores

Un **constructor** de una clase derivada puede llamar al constructor de su clase padre usando la palabra **super**.

La llamada a **super()** debe ser la primera sentencia del constructor (con la única excepción de que exista una llamada a otro constructor de la clase mediante this).

```
public class Deportista extends Persona{

    private String deporte;

    public Deportista (String nombre, String apellidos) {
        super(nombre,apellidos);
    }

    public Deportista (String nombre, String apellidos, String deporte) {
        this(nombre,apellidos);
        this.deporte = deporte;
    }
}
```

6. Herencia de Constructores

Sobreescritura

Permite modificar la implementación de un método heredado de la clase padre.

Para indicar al compilador que se va a sobreescribir se usa la notación **@Override**

```
public class MiClase {

    @Override
    public String ToString() {
        return "Hola, esta es MiClase";
    }
}
```

7. Interfaces

Interfaz

- Una clase padre que contiene una serie de métodos sin implementar.
- Métodos con cabeceras con las entradas y salidas esperadas.
- Si una clase implementa una interfaz, se implementan los métodos de la interfaz para definir cómo se va a comportar
- Se usan sufijos como: -able, -or, -ente (serializable, comparable, iterable, etc.)

Sus atributos son finales y sus métodos abstractos

7. Interfaces

```
[public] interface nombreInterfaz {
    //Atributos
    [public] [final] tipoDato nombreVariable = Valor;
    //Métodos
    [public] abstract tipoDato metodo1(tipoDato Valor);
    [public] abstract void metodo2();
}

[modificador] class ClaseEjemplo implements nombreInterfaz {
    //Atributos
    //Constructores
    //Métodos
    [public] abstract tipoDato metodo1(tipoDato Valor){
        return Valor
    }
    [public] abstract void metodo2(){
        //código
    }
}
```

7. Interfaces

Interfaces o Clases Abstractas

- No pueden ser instanciadas (new).
- No pueden ser selladas (final). Deben poder ser heredadas/redefinidas .
- Las **Interfaces**:
  - No pueden contener ninguna implementación, sólo la cabecera de los métodos.
  - Todos los atributos y métodos son públicos.
  - No pueden extender clases (aunque pueden extender otras interfaces).

8. Polimorfismo

Polimorfismo

- Un comportamiento pueda realizarse de múltiples formas distintas.

  Ejemplo, **emitir un sonido** puede ser hablar una persona, el pitido de un tren, el ladrido de un perro.

- **Sobrecarga**: Se desarrollan distintos métodos con el mismo nombre pero distinta implementación.

  Comparten nombre pero distintos parámetros en número, en tipo y/o en orden. El tipo devuelto no es válido como distinción

- También es conocido como **Sobreescritura** o **Redefinición** de métodos (**Override**)

Ejercicio

Deberás crear una clase silla que no se pueda instanciar y que lleve la cuenta de las sillas creadas y en el constructor por defecto se asignarán valores por defecto a los atributos.

A continuación crea las clases SillaDirector, SillaPupitre, SillaMadera, SillaCocina, Butaca, etc. cada una con sus características propias.

Hay comportamientos que sólo pueden hacer ciertas sillas, identifícalos y crea la interfaz correspondiente con al menos dos métodos a implementar por las clases que la utilicen.

Finalmente haz el menú de prueba.