Herencia

Programació Orientada a Objectes

Herencia

- Entre una clase A y B hay una relación de herencia cuando
  - A es una generalización de B, y B es más específica.
  - B hereda las propiedades de A, y añade otras nuevas
- Es un mecanismo fundamental para aprovechar la reutilización y extensibilidad del software.
  - es uno de los tres pilares básicos de la Programación Orientada a Objetos
- La herencia facilita la creación de nuevas clases a partir de otras ya existentes.

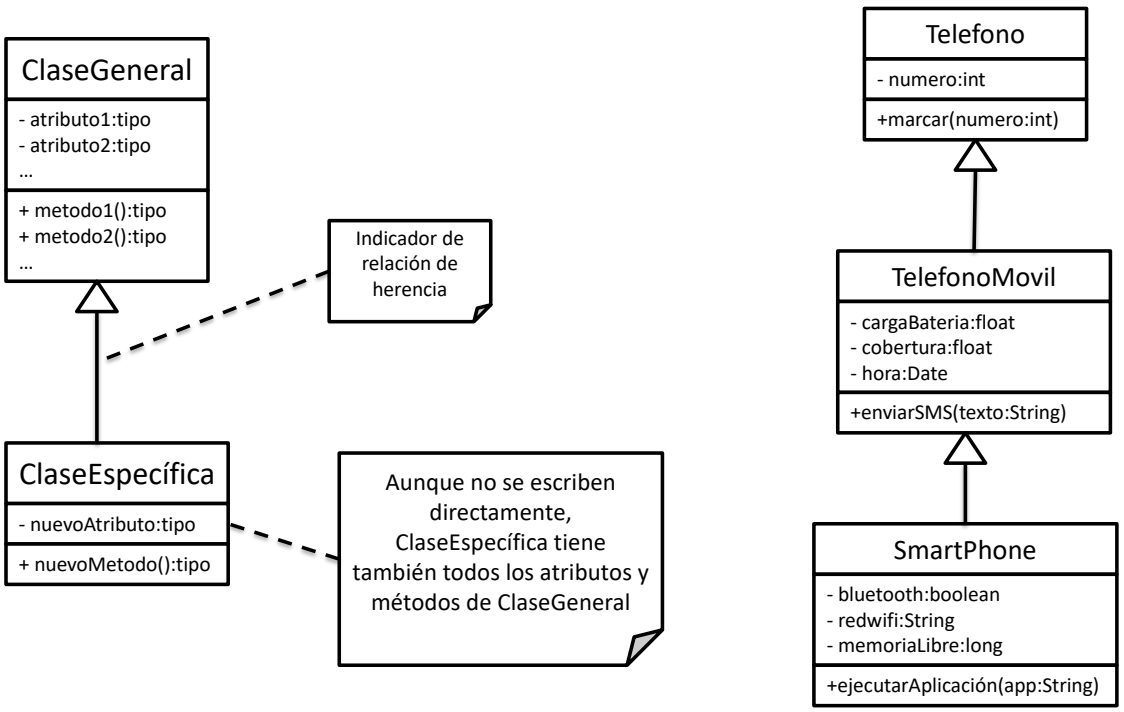
Ejemplo de Herencia

Teléfono
Funcionalidades: marcar un número, hablar

Teléfono móvil
Funcionalidades: todas las de un teléfono fijo
+ conectarse a redes GSM, enviar SMS, reloj y alarma

Smartphone
Funcionalidades: todas las de un teléfono móvil
+ conectarse a redes Wifi, Bluetooth, Aplicaciones, reproducir música, navegador web…

La herencia en UML



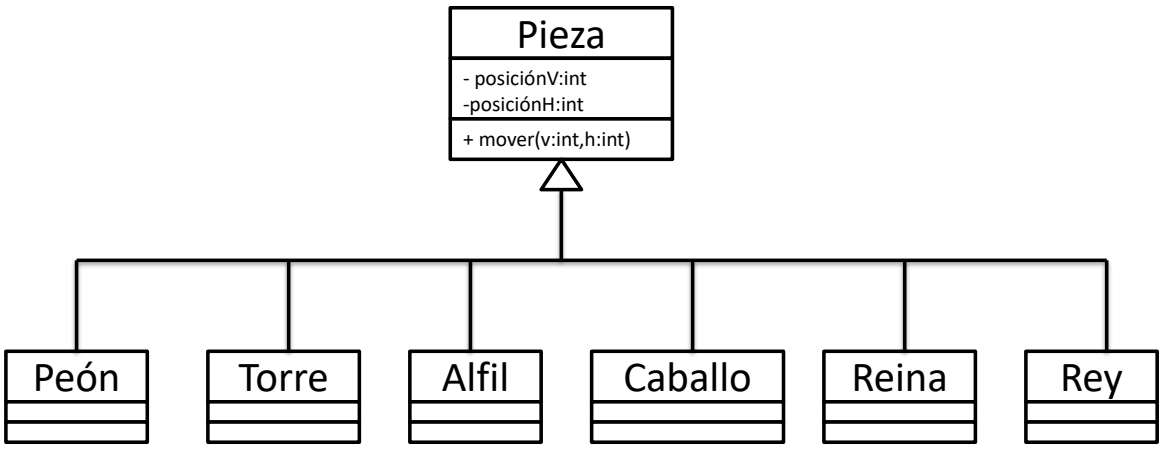
Léxico

- A es **superclase** de B.
- B es **subclase** de A.

- Sinónimos
  - Superclase = clase base, clase padre
  - Subclase = clase derivada, clase hija

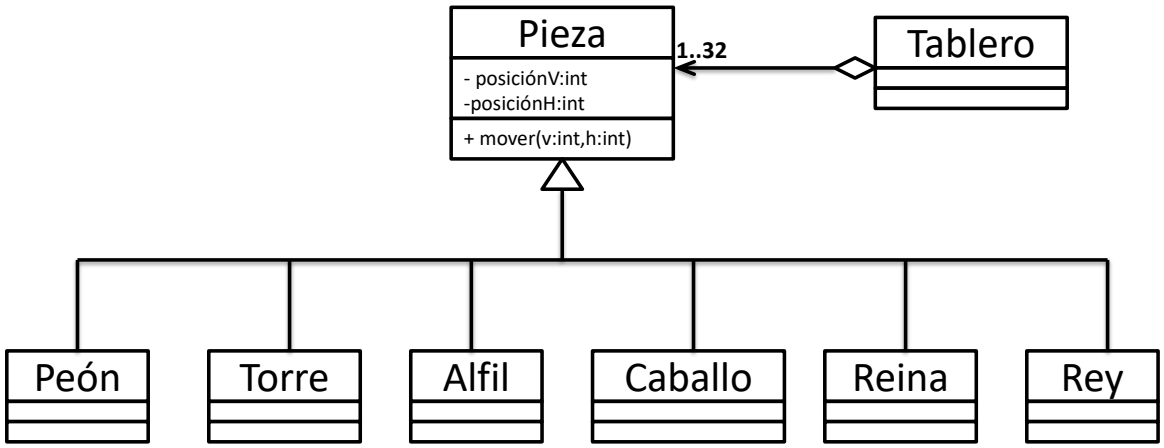
Jerarquía de clases

- La herencia nos ayuda a definir una Jerarquía de clases



Jerarquía de clases

- Las subclases también heredan las asociaciones de la superclase con otras clases.
  - … pero no a la inversa.

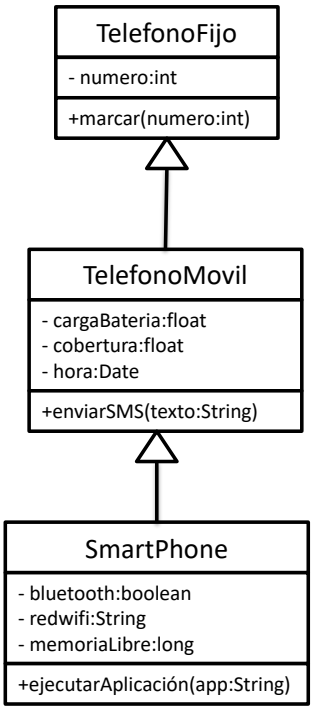


Herencia en Java

- Una clase puede tener muchas subclases.
- Sin embargo, una clase solo puede tener una superclase
  - En otros lenguajes no tiene por qué ser así
- En Java, una subclase hereda de su superclase:
  - Atributos
  - Métodos
- Una subclase **NO** hereda de su superclase:
  - Constructor

Herencia en java

```
public class TelefonoFijo {
    private int numero;
    public void marcar(int numero) {…}
}
public class TelefonoMovil extends TelefonoFijo {
    private float cargaBateria;
    private float cobertura;
    private Date hora;
    public void enviarSMS(String texto) {…}
}
public class SmartPhone extends TelefonoMovil {
    private boolean bluetooth;
    private String redWifi;
    private long memoriaLibre;
    publc void ejecutarAplicación(String app) {…}
}
```



Redefinición de métodos

- Los métodos de una subclase se pueden redefinir para ampliar/adaptar el comportamiento de la superclase.

```
public class Ecuación {
    // p1 x + p0 = 0
    protected float p0, p1;
    public float resolver() {
        return -p1/p0;
    }
}
public class Ecuación2oGrado extends Ecuación{
  // p2 x^2 + p1 x + p0 = 0
  protected float p2;
  public float resolver() { //redefinición
    return (-p1+Math.sqrt(p1*p1-4*p2*p0))/(2*p2);
  }
}
```

Esta operación también se conoce como sobreescritura (*Overriding*)

Referencias “this” y “super”

- Toda clase tiene una referencia “this”, que apunta a sí misma.
- Mediante “this” podemos acceder tanto a los atributos y métodos de la propia clase como a los de la superclase.
- Cuando la subclase redefine algún método de la superclase, aún podemos llamar al método de la superclase mediante la referencia “super”

Referencias *this* y *super*

```
public class Alumno {
  protected HashSet<Asignatura> matriculas;
  public float getPrecioMatricula() {
    float precio = 0;
    for(Asignatura a : matriculas) {
       precio += a.getPrecio();
    }
    return precio;
}}
public class AlumnoConDescuento extends Alumno {
 protected float descuento;
 public float getPrecioMatricula() {
   return super.getPrecioMatricula()*this.descuento;
  }}
```

Constructores

- Una subclase hereda todos los métodos y atributos de la superclase, excepto los constructores.

Eso significa que si tenemos:

```
public class SuperClase {
  public SuperClase(int x) { … }
}
public class SubClase extends SuperClase {…}
```

NO podemos hacer:

```
SubClase obj = new SubClase(3);
```

Constructores

- Si la superclase no tiene constructor por defecto, tenemos que llamar explícitamente al constructor de la superclase.

```
public class Ecuacion {
  protected float a,b;
  public Ecuacion(float a, float b) {
    this.a = a; this.b = b;
  }
}
public class Ecuacion2oGrado extends Ecuacion{
  protected float c;
  public Ecuacion2oGrado(float a, float b, float c){
    this.c = c;
  }
}
```

No sabe cómo construir la parte de la superclase: **ERROR!**

Constructores

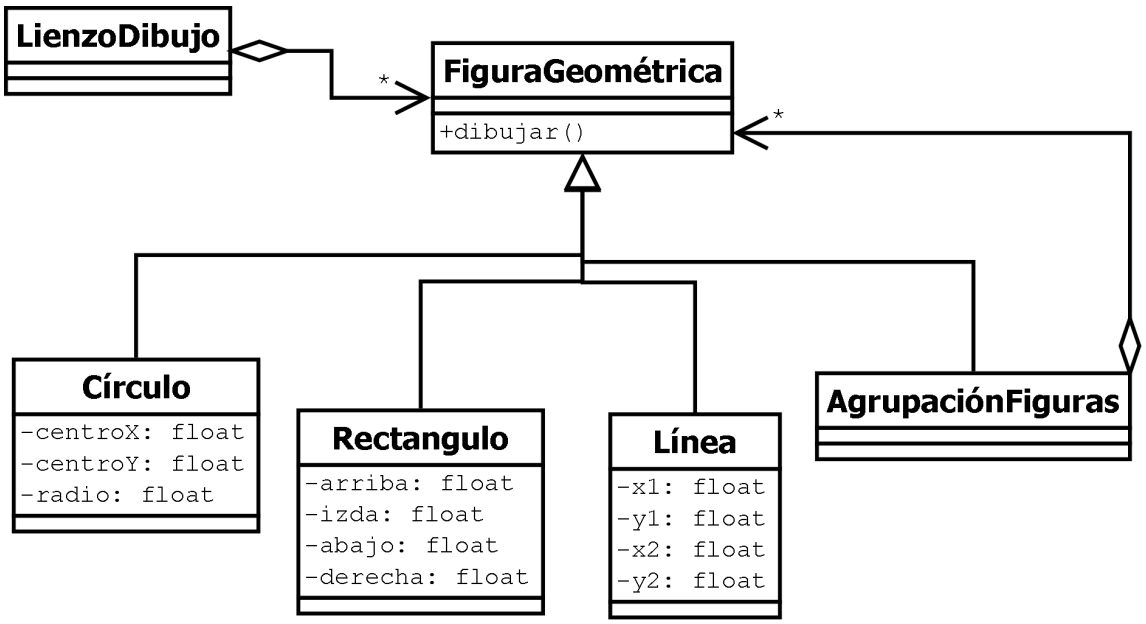
- Si la superclase no define constructor por defecto, es obligatorio llamar al constructor con parámetros desde el constructor de la subclase.

```
public class Ecuacion {
  protected float a,b;
  public Ecuacion(float a, float b) {
    this.a = a; this.b = b;
  }
}
public class Ecuacion2oGrado extends Ecuacion{
  protected float c;
  public Ecuacion2oGrado(float a, float b, float c){
    super(a,b);
    this.c = c;
  }
}
```

Ejemplo

- Queremos modelar un editor de dibujos sencillo.
  - Un lienzo contiene figuras que son dibujadas en él.
  - Hay varios tipos de figuras:
    - Líneas rectas, que se caracterizan por las coordenadas (x,y) de sus dos extremos.
    - Rectángulos, que se caracterizan por las coordenadas de cada uno de sus lados.
    - Círculos, que se caracterizan por las coordenadas de su centro y su radio.
    - Agrupaciones de figuras: permiten agrupar muchas figuras como las anteriores, y tratarlas como una sola.

Solución: UML



Solución: "Esqueleto" en Java

```
public class Figura {
  public void dibujar() {…}
}
public class Circulo extends Figura {
  private float centroX, centroY, radio;
  public void dibujar() { … }
}
public class Rectangulo extends Figura {
  private float arriba, abajo, izda, derecha;
  public void dibujar() { … }
}
```

Solución: “Esqueleto” en Java

```
public class Linea extends Figura {
   private float x1, y1, x2, y2;
   public void dibujar() { … }
}

public class AgrupacionFiguras extends Figura {
   private HashSet<Figura> agrupacion;
   public void dibujar() { … }
}

public class Lienzo {
   private HashSet<Figura> figuras;
}
```

Ejercicio

- Hacer el diagrama UML de la siguiente situación
  - Por una carretera circulan vehículos de diferentes tipos: coches, motos, camiones…
  - Cada carretera puede estar conectada con otras carreteras de diversos tipos: nacionales, autovías, urbanas…
  - El conductor de cada vehículo debe vigilar a los demás vehículos que circulan a su alrededor para no colisionar con ellos.
- **Considerar relaciones de herencia!**