

UT07. CLASES ABSTRACTAS

Programación – 1º DAW

Clases abstractas

- En ocasiones definimos clases de las que no pretendemos crear objetos → su único objetivo es servir de superclases para las clases “reales”, es decir, servir de modelo para las clases derivadas.
- **Ejemplos:** No tiene sentido crear objetos **Figura**, ya que es algo demasiado abstracto.
- A este tipo de clases las denominaremos **clases abstractas**.

Clases abstractas

- Las clases abstractas en Java se identifican mediante la palabra reservada “**abstract**”:

```
public abstract class Figura {  
    ...  
}
```

- Es un **error** tratar de **crear un objeto** de una clase abstracta

```
Figura f = new Figura(...);
```

← ERROR detectado por el compilador

- Pero **no es un error** utilizar **referencias** a clases abstractas, ya que pueden apuntar a objetos de cualquiera de sus subclases (lo hemos visto en el **polimorfismo**)

```
Figura f1 = new Círculo(...); // correcto  
Figura f2 = new Cuadrado(...); // correcto
```

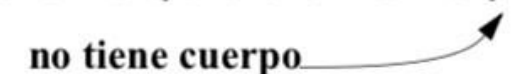
Clases abstractas – Métodos abstractos

- Definir una **clase abstracta** permite manipular un conjunto de clases con una **interfaz común** (conjunto de métodos públicos).
- Las clases abstractas permiten definir **cómo es una clase sin tener que implementar** todos sus métodos.
- Esto es especialmente útil cuando **las distintas subclases deben proporcionar los mismos métodos** definidos en la clase base pero con una implementación específica.
- Una **clase abstracta** puede tener **métodos abstractos**:
 - Se trata de **métodos sin cuerpo**, sin implementación.
 - Es **obligatorio redefinirlos** en las subclases no abstractas.
 - Permiten declarar en la superclase un **comportamiento común** que deberán verificar todas sus subclases.
 - Pero sin decir nada sobre su implementación.
- **Toda clase no abstracta que herede** debe escribir el código para los métodos abstractos.

Clases abstractas – Métodos abstractos

```
public abstract int métodoAbstracto(double d);
```

no tiene cuerpo



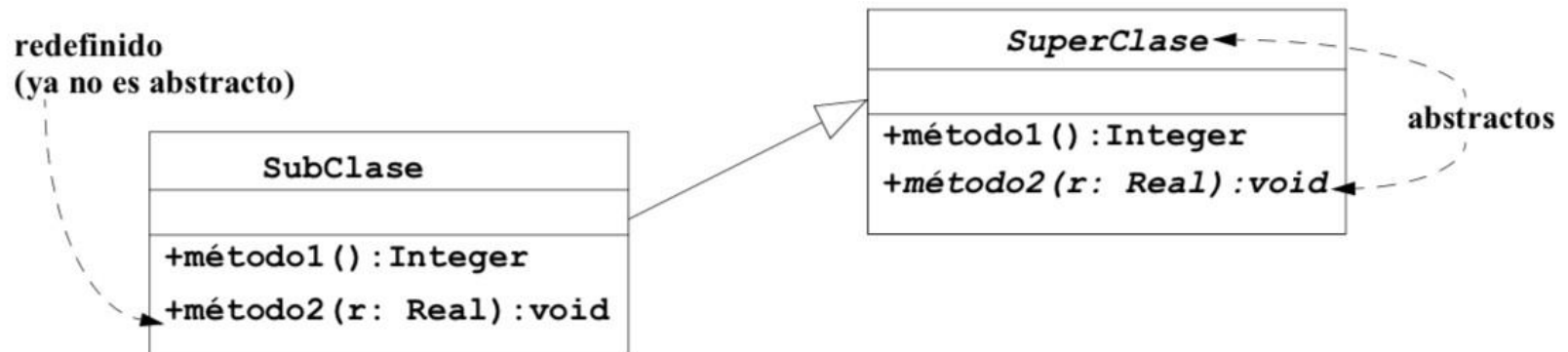
- Debes **tener en cuenta cuando trabajes con métodos abstractos**:
 - Un **método abstracto** implica que la clase a la que pertenece tiene que ser **abstracta**, pero eso no significa que todos los métodos de esa clase tengan que ser abstractos.
 - Un **método abstracto no puede ser privado** (no se podría implementar en las clases derivadas ya que no tendrían acceso a él).
 - Los **métodos abstractos no pueden ser estáticos**, pues los métodos estáticos **no pueden ser redefinidos** (y los métodos abstractos necesitan ser redefinidos).

Clases abstractas – Métodos abstractos

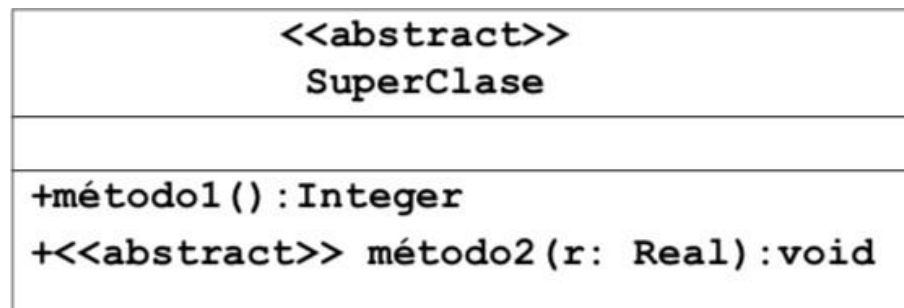
- Los **métodos abstractos** permiten establecer una **interfaz** para definir el **comportamiento común** de superclase que deberían compartir las subclases
- **Debes tener en cuenta cuando trabajes con clases abstractas:**
 - La finalidad de una **clase abstracta** es servir de base para construir la jerarquía de herencia entre clases y aplicar el polimorfismo.
 - **No se puede instanciar una clase abstracta. No** se puede hacer un **new** de una clase abstracta. Se producirá un **error de compilación**.
 - Una **clase abstracta** puede contener métodos totalmente definidos (**no abstractos**) y métodos sin definir (**métodos abstractos**).

Clases abstractas – Métodos abstractos

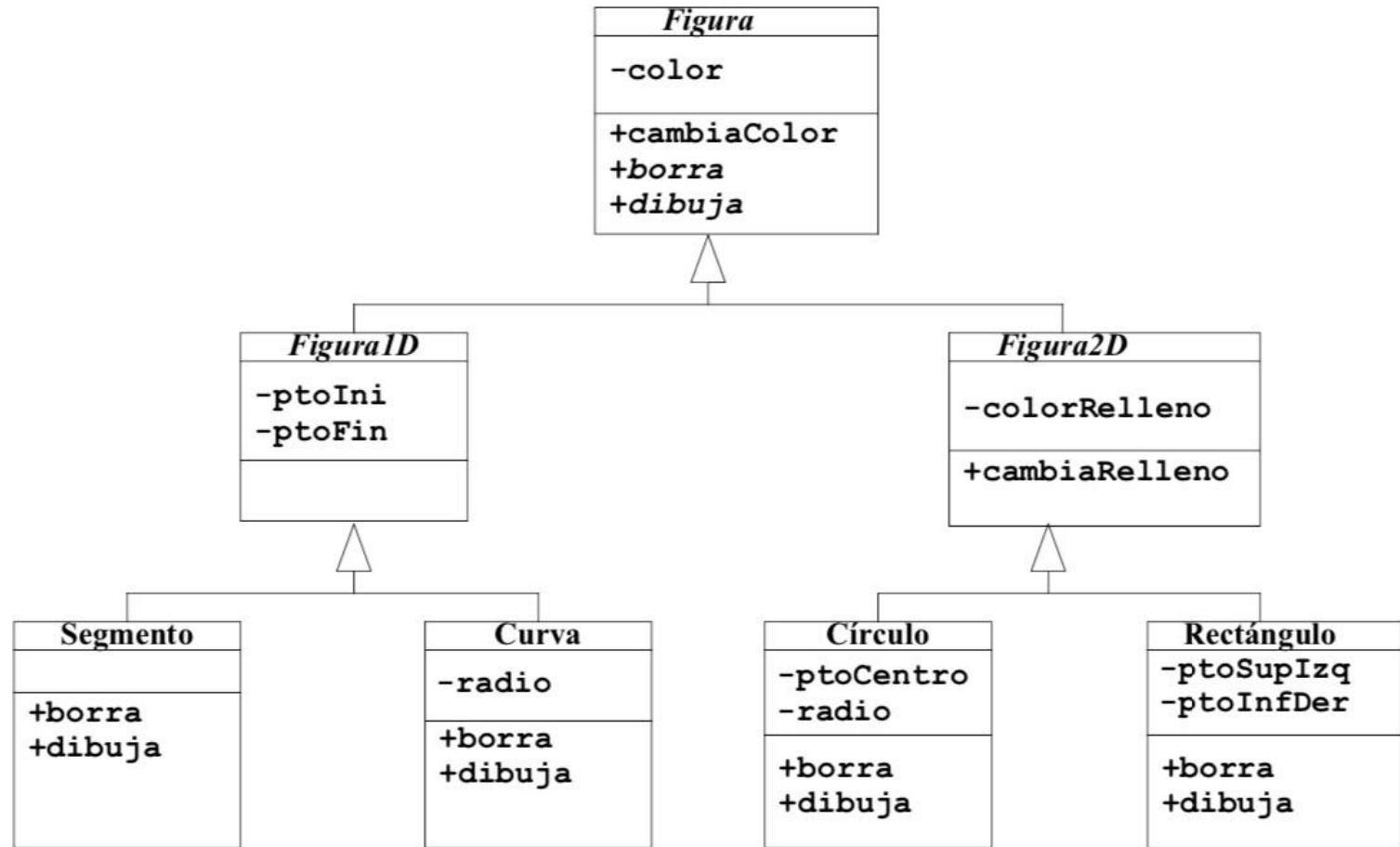
- Las clases y los métodos abstractos se escriben **en cursiva** en los **diagramas de clases**:



- También puede utilizarse el estereotipo <<abstract>>



Jerarquía de clases (clases abstractas)



Implementación de la jerarquía anterior

```
public abstract class Figura {  
    // color del borde de la figura  
    private int color;  
    /** Constructor ... */  
    public Figura(int color) {  
        this.color=color;  
    }  
    /** cambia el color del borde de la figura ... */  
    public void cambiaColor(int color) {  
        this.color=color;  
    }  
    /** borra la figura (abstracto) ... */  
    public abstract void borra();  
    /** dibuja la figura (abstracto) ... */  
    public abstract void dibuja();  
}
```

Implementación de la jerarquía anterior

54

```
public abstract class Figura1D extends Figura {  
  
    // puntos de comienzo y final de la figura  
    private Punto ptoIni, ptoFin;  
  
    /** Constructor ... */  
    public Figura1D(int color, Punto ptoIni,  
                    Punto ptoFin) {  
        super(color);  
        this.ptoIni = ptoIni;  
        this.ptoFin = ptoFin;)  
    }  
  
    // NO redefine ningún método abstracto  
}
```

Implementación de la jerarquía anterior

55

```
public abstract class Figura2D extends Figura {  
    // color de relleno de la figura  
    private int colorRelleno;  
  
    /** Constructor ... */  
    public Figura2D(int color, int colorRelleno) {  
        super(color);  
        this.colorRelleno=colorRelleno;  
    }  
  
    /** cambia el color de relleno ... */  
    public void cambiaRelleno(int color) {  
        colorRelleno=color;  
    }  
  
    // NO redefine ningún método abstracto  
}
```

Implementación de la jerarquía anterior

56

```
public class Recta extends Figura1D {  
  
    /** Constructor ... */  
    public Recta(int color,  
                 Punto ptoIni, Punto ptoFin) {  
        super(color, ptoIni, ptoFin);  
    }  
  
    /** implementa el método abstracto borra ... */  
    @Override  
    public void borra() { implementación...; }  
  
    /** implementa el método abstracto dibuja ... */  
    @Override  
    public void dibuja() { implementación...; }  
    ...;  
}
```

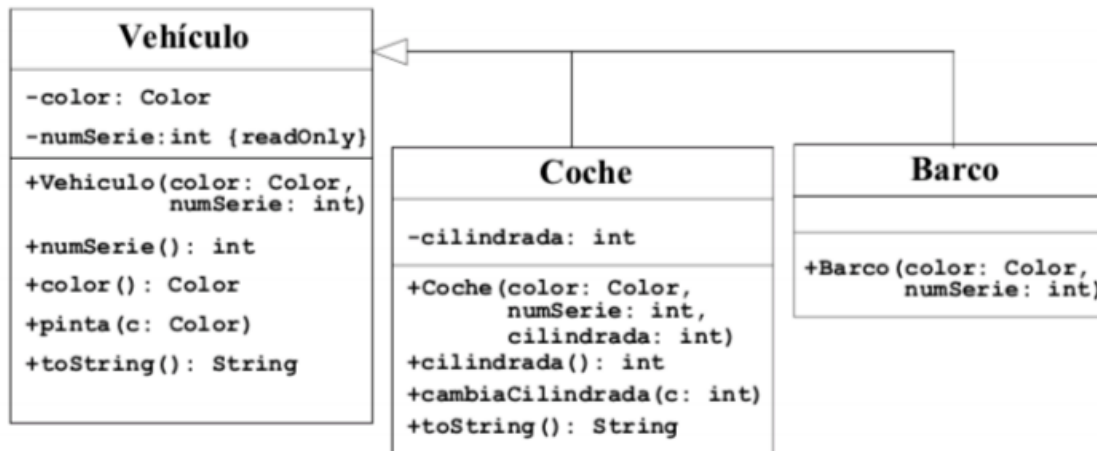
Implementación de la jerarquía anterior

57

```
public class Círculo extends Figura2D {  
    private Punto ptoCentro;  
    private double radio;  
    /** Constructor ... */  
    public Círculo(int color, int colorRelleno,  
                  Punto ptoCentro, double radio){  
        super(color, colorRelleno);  
        this.ptoCentro = ptoCentro;  
        this.radio = radio;  
    }  
    /** implementa el método abstracto borra ... */  
    @Override  
    public void borra() { implementación...; }  
    /** implementa el método abstracto dibuja ... */  
    @Override  
    public void dibuja() { implementación...; }  
}
```

Herencia en Java - Clases abstractas

Vehículo puede ser una clase abstracta → nunca crearemos un vehículo, crearemos un coche, un barco , un avión, etc



Una aplicación para un centro educativo puede necesitar trabajar con las clases **Alumno** y **Profesor**, ambas subclases de **Persona**. La clase **Persona** reúne las generalidades de **Alumno** y **Profesor**, por lo que es posible que nunca necesitemos instanciarla, sino que únicamente nos sirva como base para construir las clases **Alumno** y **Profesor**. **Persona puede ser una clase abstracta.**

Implementacion de la Jerarquía Figuras

57

```
public class EditorGraficoMain {  
    public static void main(String[] args) {  
        Figura f1;  
        f1 = new Rectangulo(new Punto(0,3),new Punto(5,0),Color.BLUE,  
Color.GREEN);  
        Figura f2 = new Circulo(new Punto(0,0), 5, Color.magenta,Color.WHITE);  
        //No podemos instanciar una clase abstracta  
        //Figura f3 = new Figura(Color.BLACK); //No permitido.  
        //f1 es una referencia de tipo Figura por tanto solo podemos invocar sobre  
        //f1 métodos definidos en Figura  
        //f1.base(); //No permitido, ya que figura() es un método de Rectangulo  
        //Casting  
        Rectangulo r = (Rectangulo) f1;  
        r.base(); //Ahora si podemos invocar métodos de Rectangulo  
    }  
}
```


Más información

Vídeos de apoyo

- Herencia
 - <https://youtu.be/wqoyQ3BxK4A>
 - <https://youtu.be/rEOFpdl3HY0>
 - https://youtu.be/3g_3cbH97cs
- Polimorfismo y clases abstractas
 - <https://youtu.be/sdJgcMaazml>
 - <https://youtu.be/ztpYmmecfQs>
 - <https://youtu.be/LDZUBY0mxv8>