

# Sesión 15: Herencia

## Programación 2

### 4. Herencia y polimorfismo

---

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

# En capítulos anteriores

- 👍 Tema 1: Clases y Objetos
- 👍 Tema 2: Colecciones acotadas de Objetos
- 👍 Tema 4: Tipos Abstractos de Datos
- 👍 Tema 3: Programación Modular
- 🕒 Tema 5: Herencia y Polimorfismo
  - Primera **toma de contacto** con la **herencia**



# Cohesión y Acoplamiento

## Acoplamiento

relaciones entre componentes

## Cohesión

relaciones dentro de un componente



# Cohesión y Acoplamiento

Acoplamiento

relaciones entre componentes

Cohesión ↑

relaciones dentro de un componente



# Cohesión y Acoplamiento

Acoplamiento ↓

relaciones entre componentes

Cohesión ↑

relaciones dentro de un componente

# Maximizar la cohesión

*Un componente debe tener un pequeño conjunto de responsabilidades bien definidas*

- Y deberíamos añadir:

altamente cohesionadas

- Máxima cohesión = 1! responsabilidad
- Aunque no siempre es posible ni bueno

# Minimizar el acoplamiento

Ocultación de datos

+

Evitar variables globales

+

API

(tipos abstractos de datos)



# Herencia

- Concepto **fundamental** en OO
- En Java, para empezar:

**class** *A* **extends** *B* {...}

Las instancias de la clase *A* **tienen todas las propiedades**<sup>1</sup> declaradas en *B*

- Decimos que una clase *A* **hereda de** otra clase *B*
- También decimos que *A* **es subclase** de *B*
- También decimos que *B* **es superclase** de *A*

---

<sup>1</sup>Atributos y métodos.



# En el capítulo de hoy

- 🕒 Tema 5: Herencia y Polimorfismo
  - **Profundizamos** en la herencia

# geometria con herencia

- Figura
- Circulo
- Poligono
- PoligonoRegular
- Rectangulo
- Cuadrado
- Triangulo
- Equilatero
- Hexagono
- Y un programa principal para probar

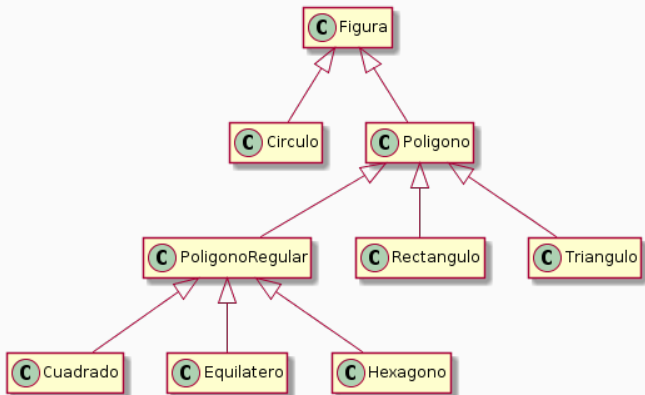
# ¡Aprendiendo!

- Las siguientes transparencias contienen el resultado de un diseño colaborativo en clase<sup>2</sup>
- La idea es que fueran surgiendo necesidades y soluciones a medida que se avanzaba
- La clase comenzó con una “clasificación” de las clases por herencia
- Las clases se fueron implementando de arriba a abajo siguiendo el árbol de herencia

---

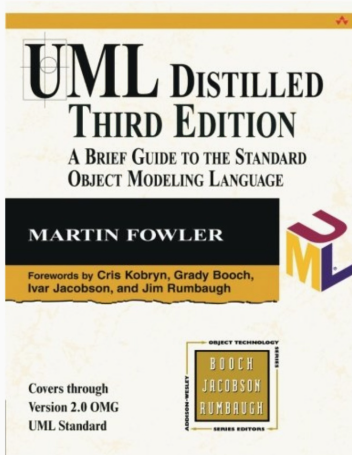
<sup>2</sup>Curso 2018-2019.

# Jerarquía de herencia<sup>3</sup>



<sup>3</sup>Java **no permite herencia múltiple**: por ejemplo, no es posible hacer `public class Equilatero extends Triangulo, PoligonoRegular...`

# Unified Modelling Language (UML)



- Lenguaje gráfico de diagramas (muy orientado a objetos)
- Podemos representar modelos del sistema a construir
- Uno de los diagramas nos permite representar clases



## Empezamos por *el top*: Figura

- Damos por supuesta la existencia de la clase Punto2D (distancia, equals, etc.)
- Todas las figuras tienen un centro
- Todas las figuras tienen un área

# Deberías estar programando...<sup>a</sup>

---

<sup>a</sup>no mirar hasta haberlo intentado por ti mismo



## Empezamos por *el top*: Figura

Deberías estar programando...





# Empezamos por *el top*: Figura

Deberías estar programando...

```
package geometria;
public class Figura {
    private Punto2D centro;
    public Punto2D centro() {
        return this.centro;
    }
    public double area() {
        return 0; // ¿Qué otra cosa podemos hacer?
    }
}
```

## Primera subclase: **Círculo**

- Todos los círculos tienen centro (**¡son figuras!**)
- Todos los círculos tienen área (**¡son figuras!**)
- Todos los círculos, **además**, tienen **un radio**



# Primera subclase: **Círculo**

Deberías estar programando...

## Primera subclase: **Círculo**

Deberías estar programando...

```
package geometria;  
  
public class Circulo extends Figura {  
    private double radio;  
    public Circulo(Punto2D c, double r) {  
        centro = c;  
        radio = r;  
    }  
}
```

# Probemos ya: GeoTest

Deberías estar programando. . .



# Problemos ya: GeoTest

Deberías estar programando...

```
import geometria.*;
public class GeoTest {
    public static void main(String[] argv) {
        Circulo c = new Circulo(new Punto2D(0,0), 2.0);
        assert c.centro().equals(new Punto2D(0,0));
        assert Math.abs(c.area() - 4 * Math.PI) < 0.001;
    }
}
```

```
javac -d lib -cp .:lib -sourcepath .:src src/GeoTest.java
java -ea -cp lib GeoTest
```

# ¿Problemillas?

- ¿Algún problema con el constructor de `Circulo`?
- ¿Quizás con `centro`?



¡Intenta resolverlo *como sea*!

# ¿Problemillas?

- ¿No pasa los tests?
- ¿Quizás el área?



Deberías de ser capaz de resolverlo

Sobreescribe el método área



# Modificador de visibilidad: **protected**

*Para que una **subclase** pueda **usar una propiedad** de alguna de su **superclases**, la propiedad debe etiquetarse al menos con el **modificador **protected*****

# Clases *abstractas*

La clase *Figura* *no sabe calcular* el área de cualquier figura, es *demasiado abstracta*, cuando esto sucede podemos marcar el método con el *modificador **abstract***

# Figura: clase abstracta<sup>4</sup> + **protected**<sup>5</sup>

```
package geometria;  
  
public abstract class Figura {  
    protected Punto2D centro;  
  
    public abstract double area();  
  
    public Punto2D centro() {  
        return centro;  
    }  
}
```

---

<sup>4</sup>Métodos sin implementar: serán implementados en las subclases

<sup>5</sup>Visibilidad en la clases y en sus subclases

# Pruebas con **Figura**

Deberías estar programando. . .

## Pruebas con Figura

Deberías estar programando...

```
Figura f = new Figura();  
assert f.centro() == null;
```

## Pruebas con Figura

Deberías estar programando...

```
Figura f = new Figura();  
assert f.centro() == null;
```

No es posible crear, **directamente**,  
instancias de clases abstractas

# Polígono

- Todos los polígonos son figuras
- Los polígonos tienen, además, un número de lados
- Los polígonos tienen, además, un perímetro

Deberías estar programando...



# Poligono: aún es demasiado abstracta<sup>6</sup>

```
package geometria;

public abstract class Poligono extends Figura
{
    protected int nLados;

    public abstract double perimetro();

    public int nLados() {
        return nLados;
    }
}
```

---

<sup>6</sup>Aún no se puede programar `area()` y se añade `perimetro()` que tampoco se sabe cómo implementar

- Todos los polígonos regulares son polígonos
- Los los lados de un polígono regular son iguales (longitud del lado)
- Calcular el área y el perímetro de un polígono regular es “fácil”: ¡ya no es abstracta!

Deberías estar programando...

# PoligonoRegular i

```
package geometria;

public class PoligonoRegular extends Poligono {
    protected double longLado;

    public PoligonoRegular(Punto2D centro,
                           int nLados,
                           double longLado) {
        this.centro = centro;
        this.nLados = nLados;
        this.longLado = longLado;
    }
}
```

# PoligonoRegular ii

```
public double perimetro() {  
    return longLado * nLados;  
}  
  
public double lado() {  
    return longLado;  
}  
  
public double area() {  
    double apotema = longLado / (2 * Math.tan(Math.PI/nLados));  
    return nLados * apotema * longLado / 2;  
}  
}
```

Deberías estar programando...

# Hexagono: **super**<sup>7</sup> + sobreescritura<sup>8</sup>

```
package geometria;

public class Hexagono extends PoligonoRegular
{
    public Hexagono(Punto2D centro,
                    double longLado) {
        super(centro, 6, longLado);
    }

    public double area() {
        return 3 * Math.sqrt(3) * longLado * longLado / 2;
    }
}
```

---

<sup>7</sup>Reusando el constructor del *padre*

<sup>8</sup>*Overriding*: sobreescribimos el método `area()` con mayor “eficiencia”