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[1] 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*

[1] Atributos y métodos.

En el capítulo de hoy

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

`geometria` con herencia

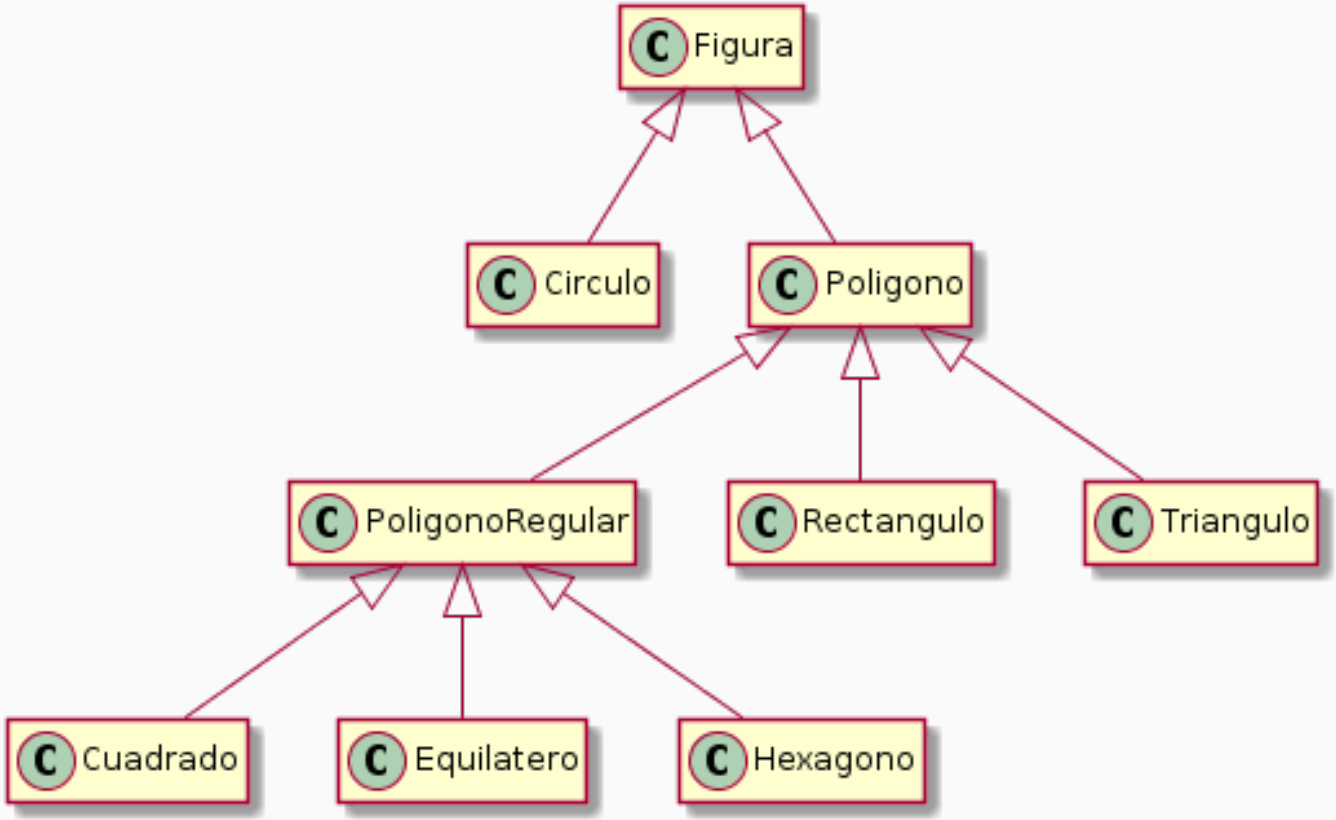
- `Figura`
- `Circulo`
- `Poligono`
- `PoligonoRegular`
- `Rectangulo`
- `Cuadrado`
- `Triangulo`
- `Equilatero`
- `Hexagono`
- Y un programa principal para probar

¡Aprendiendo!

- Las siguientes transparencias contienen el resultado de un diseño colaborativo en clase[2]
- 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

[2] Curso 2018-2019.

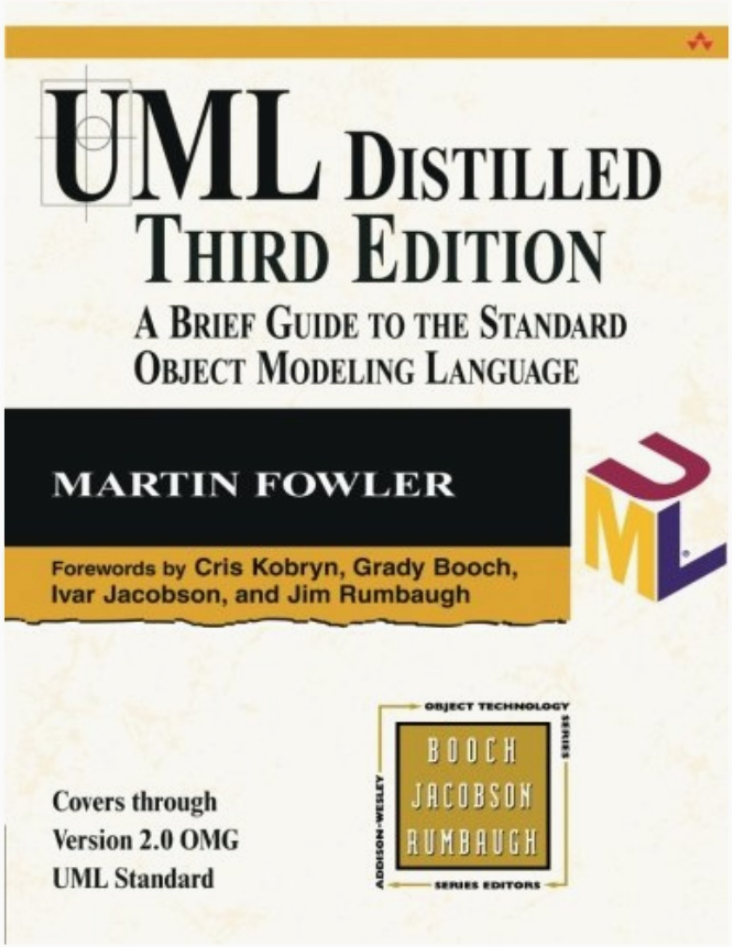
Jerarquía de herencia[3]



[3] 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

Deberías estar programando. . . [a]

[a] no mirar hasta haberlo intentado por ti mismo

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

Empezamos por *el top*: `Figura`

Deberías estár programando...

Empezamos por *el top*: `Figura`

Deberías estár 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: `Circulo`

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

Primera subclase: `Circulo`

Deberías estár programando...

Primera subclase: `Circulo`

Deberías estár programando...

```
package geometria;
public class Circulo extends Figura {
  private double radio;
  public Circulo(Punto2D c, double r) {
    centro = c;
    radio = r;
  }
  public float area() {
    return Math.PI * radio * radio;
  }
}
```

Probemos ya: `GeoTest`

Deberías estár programando...

Probemos ya: GeoTest

Deberías estár 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
```

Intersección de círculos

- Método en `Circulo` que diga si interseccione con otro círculo

Intersección de círculos

Deberías estár programando...

Intersección de círculos

Deberías estár programando...

```
public boolean intersectan(Circulo c) {
  double d = centro().distancia(c.centro());
  return d < radio + c.radio;
}
```

¿Algún problema?

Pruebas sobre `Figura`

```
// Más tests
Figura f = new Figura();
assert f.centro() != null;
assert f.area() < 0.001;
```

¿Algún problema?

Figura: clase abstracta[4] + protected[5]

```
package geometria;

public abstract class Figura {
  protected Punto2D centro;

  public abstract double area();

  public Punto2D centro() {
    return centro;
  }
}
```

[4] Métodos sin implementar

[5] Visibilidad en subclases

`Poligono`: aún es demasiado abstracta[6]

```
package geometria;

public abstract class Poligono extends Figura
{
  protected int nLados;

  public abstract double perimetro();

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

[6] Aún no se puede programar `area()` y se añade `perimero()` que tampoco se sabe cómo implementar

`PoligonoRegular`: sólo para valientes 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`: sólo para valientes 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;
  }
}
```

Hexagono: super[7] + sobreescritura[8]

```
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;
  }
}
```

[7] Reusando el constructor del *padre*

[8] *Overriding*: sobreescribimos el método `area()` con mayor "eficiencia"