Sesión 15: Polimorfismo

Programación 2

Ángel Herranz

Marzo 2019

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

En el capítulo de hoy

- Tema 5: Herencia y Polimorfismo
  - Herencia $\Rightarrow$ Subtipado

En el capítulo de hoy

- Tema 5: Herencia y Polimorfismo
  - Herencia $\Rightarrow$ Subtipado
  - Paramétrico

En el capítulo de hoy

- Tema 5: Herencia y Polimorfismo
  - Herencia $\Rightarrow$ Subtipado
  - Paramétrico
  - *Ad hoc*

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

A component must have a small well-defined set of responsabilities

Minimizar el acoplamiento

Ocultación de datos

+

Evitar variables globales

+

API

(tipos abstractos de datos)

Ejemplo tonto

- `Mamífero`: `hablar()`
- `Perro`: `hablar()` y `ladrar()`
- `Gato`: `hablar()` y `maullar()`
- Programa principal que *juegue* con objetos de dichas clases

¿Dudas?

`geometria` con herencia

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

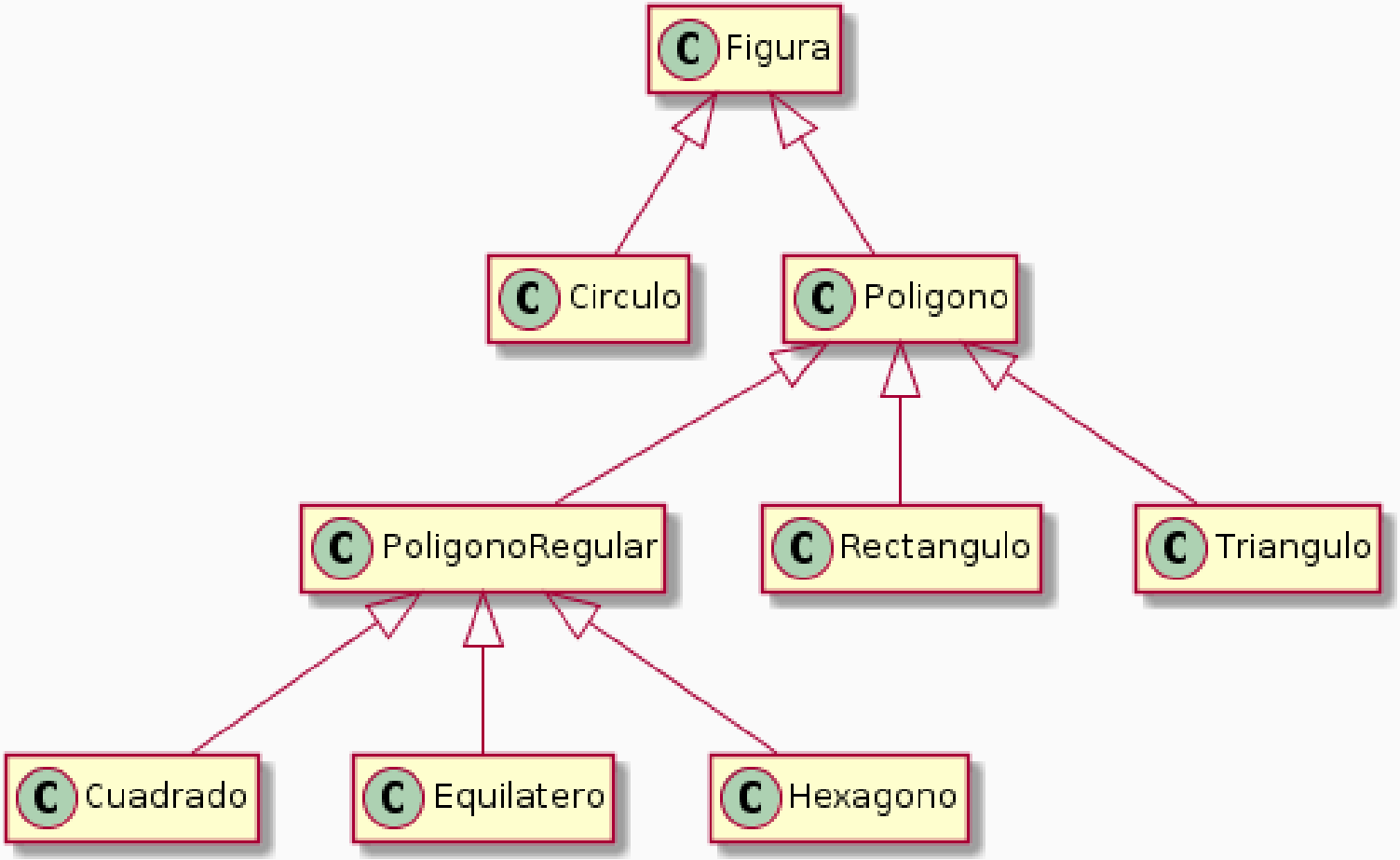
Elaborado en Clase[a]

[a] no mirar hasta haberlo intentado por ti mismo

¡Aprendiendo!

- Las siguientes transparencias contienen el resultado de un diseño colaborativo en clase
- 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 herencia

Jerarquía de herencia[1]



[1] Java no permite herencia múltiple: por ejemplo, no es posible hacer que `Equilatero` sea una subclase de `Triangulo` y de `PoligonoRegular`

Figura: clase abstracta[2] + protected[3]

```
public abstract class Figura {
  protected Punto2D centro;

  public abstract double area();

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

[2] Métodos sin implementar

[3] Visibilidad en subclases

Circulo: centro y centro() heredado

```
public class Circulo extends Figura
{
  private double radio;

  public Circulo(Punto2D centro,
                 double radio) {
    this.centro = centro;
    this.radio = radio;
  }

  public double area() {
    return Math.PI * radio * radio;
  }
}
```

`Poligono`: aún es demasiado abstracta[4]

```
public abstract class Poligono extends Figura
{
  protected int nLados;

  public abstract double perimetro();

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

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

`PoligonoRegular`: sólo para valientes i

```
public class PoligonoRegular extends Poligono {
  /* Suficiente con longLado y nLados pero...
     "sólo para valientes" ;) */
  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[5] + sobreescritura[6]

```
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;
  }
}
```

[5] Reusando el constructor del *padre*

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

Notas internas

Conceptos a contar

- UML
- Clases abstractas
- super
- Herencia múltiple
- Adding, replacement, refining
- Dynamic vs Static dispatching (binding)
- Replacement and substitutability
- American and Scandinavian semantics