Metodología de la programación

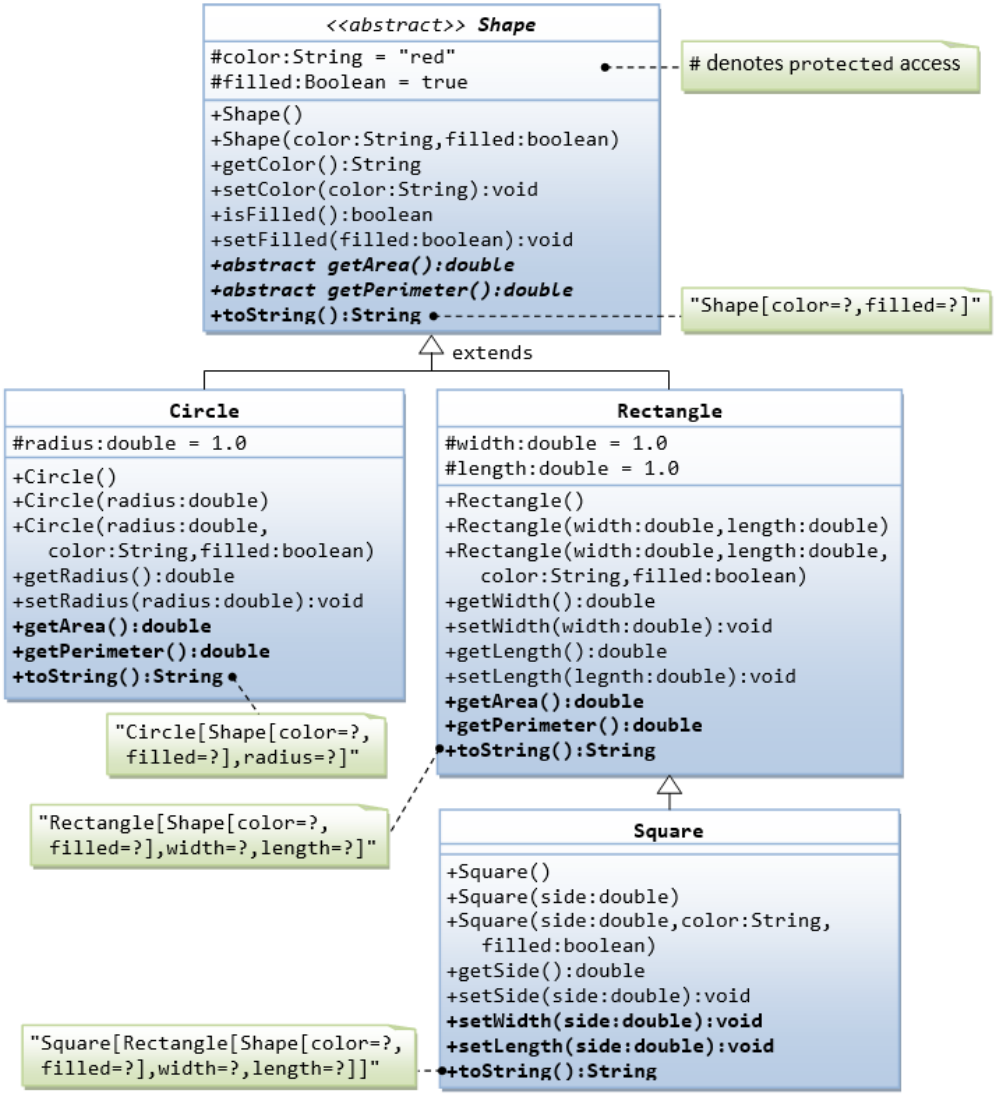
Ejercicios de Programación Orientada a Objetos

Estos ejercicios están basados en los ejercicios propuestos en la Universidad Tecnológica de Nanyang en Singapur.

6.Polimorfirsmo, Clases Abstractas e Interfaces

6.1 Superclase abstracta Shape y sus subclases concretas

Dado el siguiente diagrama, hay que rehacer las clases mostradas.
La clase Shape es una clase abstracta, que contiene dos métodos abstractos. A estos métodos abstractos, las subclases están obligadas a implementarlos.
Dado que estamos haciendo una jerarquía de clases, todos los atributos de las clases tendrán acceso `protected` *(#)*. Todos los métodos sobrecargados tendrán la anotación `@Override`.



Las subclases `Circle` y `Rectangle` deben sobrecargar los métodos abstractos de la superclase, proveyendo de la implementación correspondiente. También deben sobrecargar el método `toString()`.

Escriba un programa de prueba indicando cuáles son los casos de polimorfismo y explique por qué. Si se produjesen errores, indique por qué. Use como base el siguiente código:

```
Shape s1 = new Circle(5.5, "red", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
System.out.println(s1.getColor());

System.out.println(s1.isFilled());

System.out.println(s1.getRadius());


Circle c1 = (Circle)s1;                   // Downcast back to Circle
System.out.println(c1);

System.out.println(c1.getArea());

System.out.println(c1.getPerimeter());

System.out.println(c1.getColor());

System.out.println(c1.isFilled());

System.out.println(c1.getRadius());


Shape s2 = new Shape();


Shape s3 = new Rectangle(1.0, 2.0, "red", false);   // Upcast
System.out.println(s3);

System.out.println(s3.getArea());

System.out.println(s3.getPerimeter());

System.out.println(s3.getColor());

System.out.println(s3.getLength());


Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);

System.out.println(r1.getArea());

System.out.println(r1.getColor());

System.out.println(r1.getLength());


Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);

System.out.println(s4.getArea());

System.out.println(s4.getColor());

System.out.println(s4.getSide());


// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;

System.out.println(r2);
```

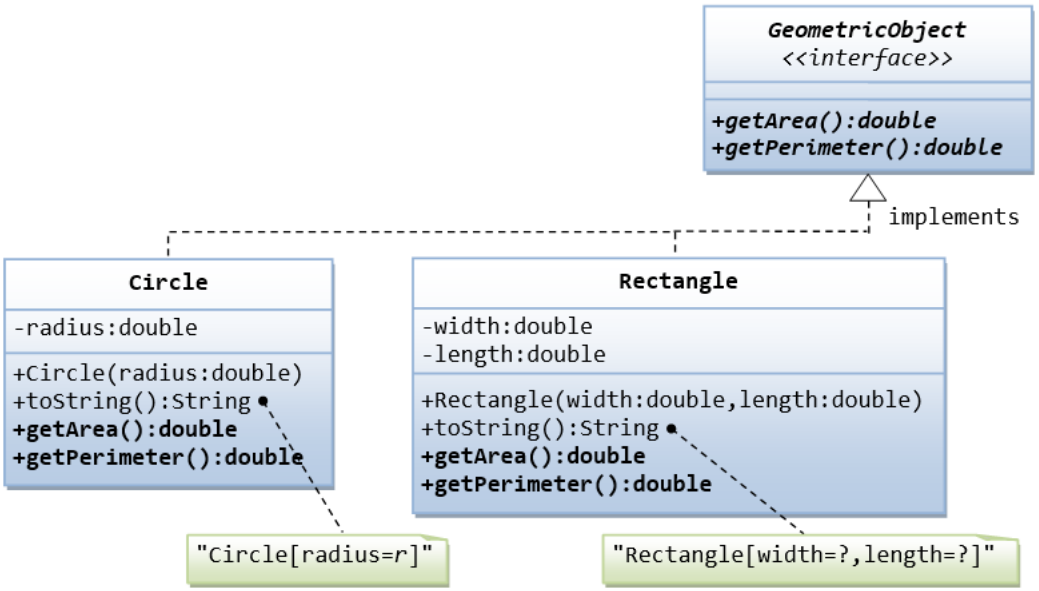
```
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

En tus palabras, ¿para qué sirven las clases abstractas y los métodos abstractos?

6.2 Interfaces: `GeometricObject` y su implementación en las clases `Circle` y `Rectangle`

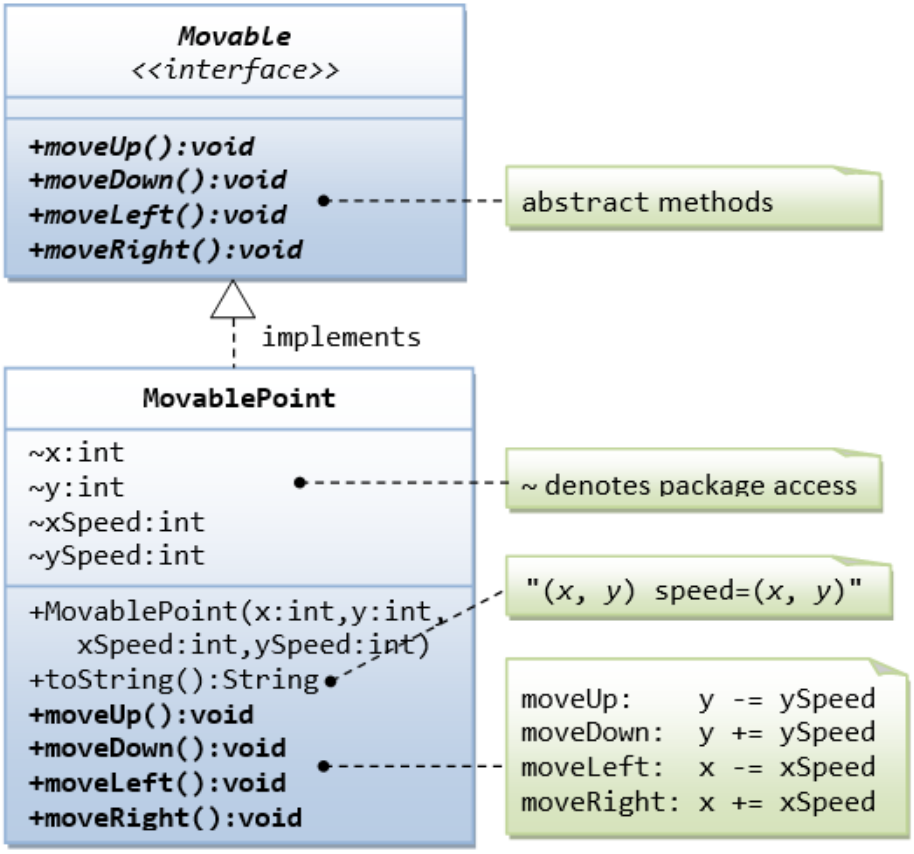
Un interfaz representa un contrato público. Aquellas clases que quieran disfrutar de un interfaz, deben implementar todos los métodos indicados. Los programas que usen un interfaz permitirán que cualquier clase de objetos que implemente el interfaz pueda ser manejado de manera transparente.

Escriba un interfaz llamado `GeometricObject`, que contiene dos métodos abstractos (OJO!!!, que sean abstractos aquí no quiere decir que tengamos que poner "abstract": en una interfaz todo es abstracto porque no hay implementación) `getArea()` y `getPerimeter()`. Escribe las clases que implementan la interfaz, y un programa de prueba similar al empleado en el ejercicio anterior, donde las variables sean de tipo GeometricObject, y los objetos creados mediante new Circle y new Rectangle.



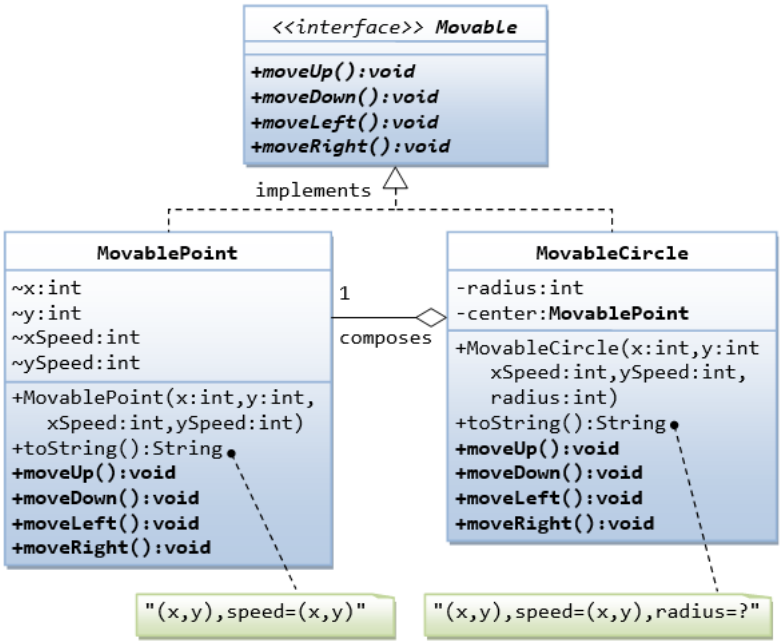
6.3 Interfaz `Movable` y su implementación en la clase `MovablePoint`

Construye la interfaz y la clase diseñada, generando también un programa de ejemplo.

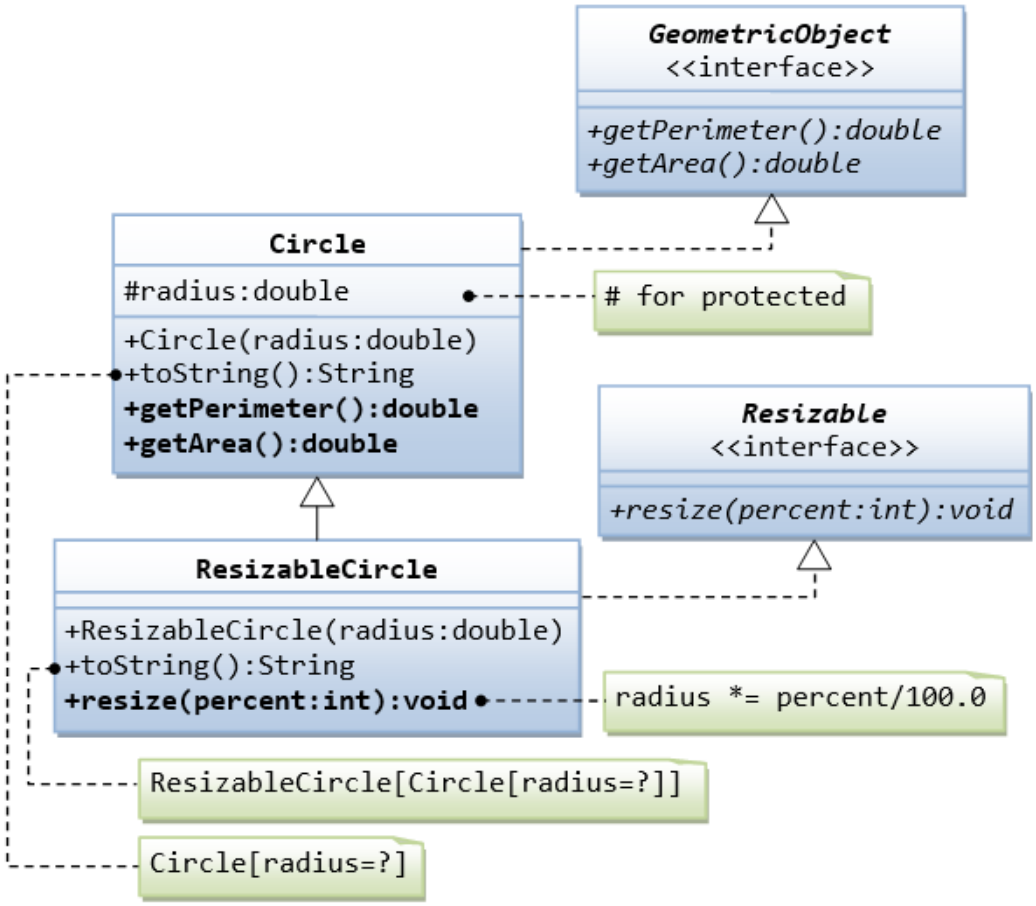


6.4 Interfaz `Movable` y su implementación en las clases `MovablePoint` y `MovableCircle`

Escriba el código de la interfaz y las clases asociadas, así como un programa de prueba que muestre el uso de dichas clases.



6.5 La interfaz `Resizable` y `GeometricObject`



Las interfaces se pueden combinar en la misma clase, para dotar a esta clase y sus subclases de capacidades múltiples.

1.Escribe la interfaz `GeometricObject`, ejemplo:

```
public interface GeometricObject {
    public double getPerimeter();

    ......

}
```

2.Escribe la implementación de la interfaz en la clase Circle:

```
public class Circle implements GeometricObject {
    // Private variable
    ......


    // Constructor
    ......


    // Implement methods defined in the interface GeometricObject
    @Override
    public double getPerimeter() { ...... }
```

```
    ......
}
```

3.Escribe el programa de prueba y muestra las capacidades de la clase `Circle`..

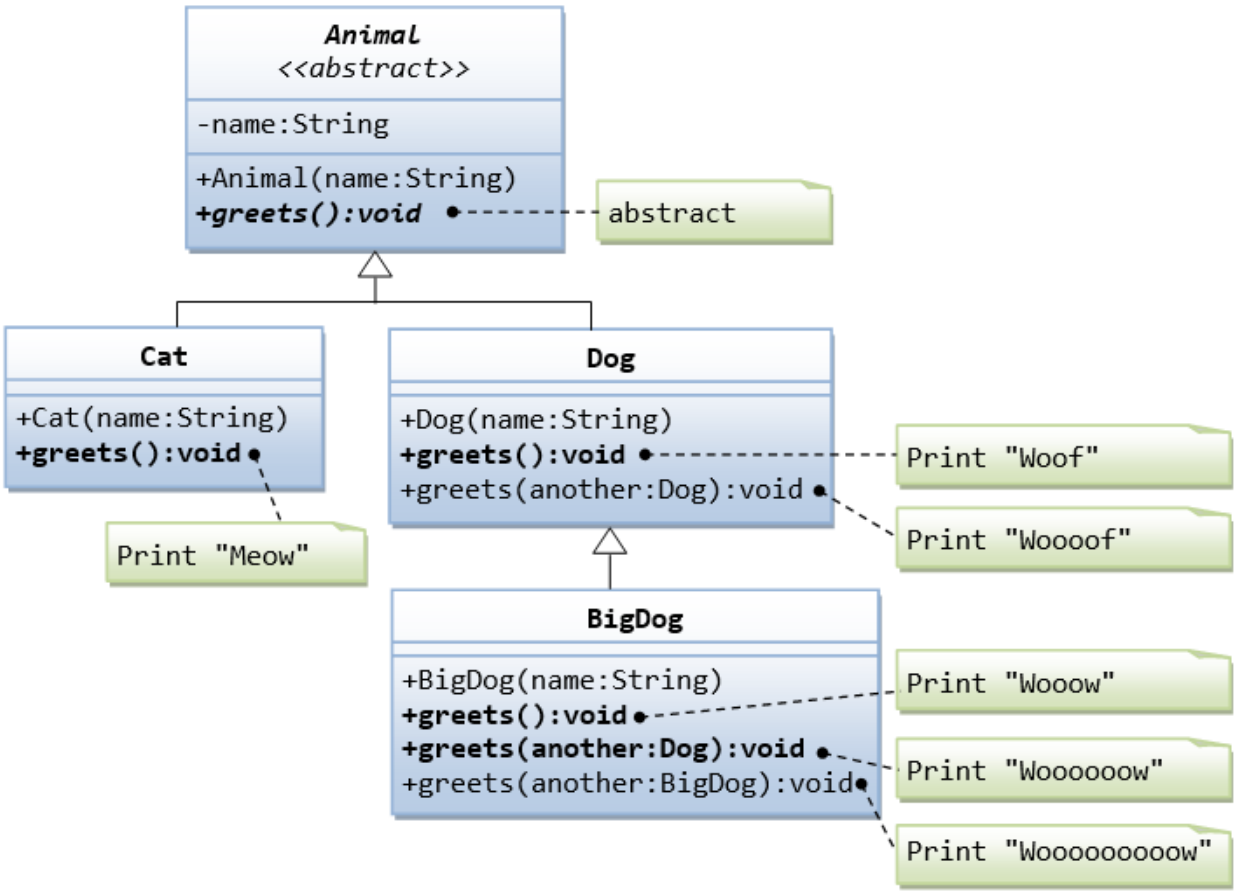
4.Escribe ahora la interfaz Resizable y la clase ResizableCircle:

```
public interface Resizable {
    public double resize(...);
}
public class ResizableCircle extends Circle implements Resizeable {

    // Constructor
    public ResizableCircle(double radius) {
        super(...);
    }

    // Implement methods defined in the interface Resizable
    @Override
    public double resize(int percent) { ...... }
}
```

5.Escribe ahora el programa de ejemplo para la nueva clase creada.

6.6 La superclase abstracta `Animal` y su implementación en subclases

Usando ahora clases abstractas, modifique las clases que se ha escrito anteriormente (ejercicio 3.6). Marque los métodos con @Override si es neceario. Escriba un programa de prueba para estas clases que demuestre sus capacidades.



6.7 Implementación alternativa

A partir del siguiente código, genere el diagrama de clases UML correspondiente.

```
abstract public class Animal {
    abstract public void greeting();
}
public class Cat extends Animal {
    @Override
    public void greeting() {
        System.out.println("Meow!");
    }
}
public class Dog extends Animal {
    @Override
    public void greeting() {
        System.out.println("Woof!");
    }

    public void greeting(Dog another) {
        System.out.println("Wooooooooof!");
    }
}
public class BigDog extends Dog {
    @Override
    public void greeting() {
        System.out.println("Woow!");
    }

    @Override
    public void greeting(Dog another) {
        System.out.println("Wooooowwwww!");
    }
}
```

Explique las salidas que se obtendrían con el siguiente código:

```
public class TestAnimal {
    public static void main(String[] args) {
        // Using the subclasses
        Cat cat1 = new Cat();
        cat1.greeting();
        Dog dog1 = new Dog();
```

```
        dog1.greeting();
        BigDog bigDog1 = new BigDog();
        bigDog1.greeting();

        // Using Polymorphism
        Animal animal1 = new Cat();
        animal1.greeting();
        Animal animal2 = new Dog();
        animal2.greeting();
        Animal animal3 = new BigDog();
        animal3.greeting();
        Animal animal4 = new Animal();

        // Downcast
        Dog dog2 = (Dog)animal2;
        BigDog bigDog2 = (BigDog)animal3;
        Dog dog3 = (Dog)animal3;
        Cat cat2 = (Cat)animal2;
        dog2.greeting(dog3);
        dog3.greeting(dog2);
        dog2.greeting(bigDog2);
        bigDog2.greeting(dog2);
        bigDog2.greeting(bigDog1);
    }
}
```

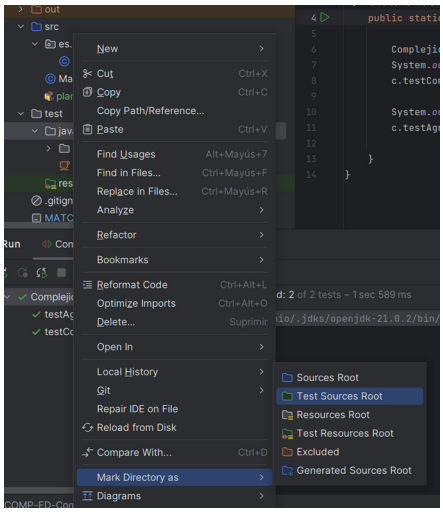
6.7 Mejorando la calidad de las clases

Dado que tenemos los programas de prueba de las clases anteriores, y en estos programas se muestra qué se espera que aparezca en pantalla según la ejecución de los ejercicios, añada para todas las clases de los ejercicios los tests necesarios para verificar la correcta ejecución de todos ellos.

Asegure que los tests tienen una cobertura del 100% del código desarrollado en las clases.

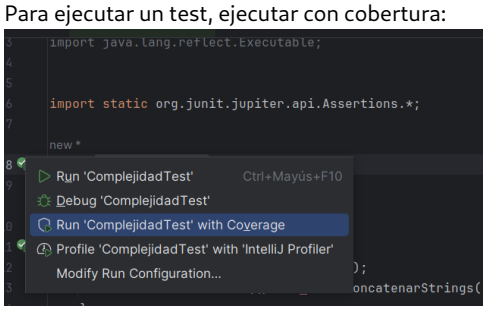
(Usamos JUnit 5)
Pasos:
Crear directorio test
Crear directorio test/java
Crear directorio test/resources

Marcar directorio test/java como “Test Source Root”
Marcar directorio test/resources como “Test Resources Root”



Sobre la clase a testear, botón derecho → “Generate” y en la ventana que aparece, elegir “Test”.

Para ejecutar un test, ejecutar con cobertura:



Referencia:
https://www.jetbrains.com/help/idea/tests-in-ide.html
https://www.jetbrains.com/help/idea/testing.html