CLEAN CODE

Carolina Dranicer Dranicer

Partimos con el siguiente código que consiste en calcular el área y perímetro de un círculo.

```java
package es.ed.carolinadranicerdranicer_cleancode;

import java.util.Scanner;

public class CarolinaDranicerDranicer_CleanCode {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(source:System.in);

        System.out.println(x: "Ingrese el radio de la circunferencia:");
        double radio = scanner.nextDouble();

        double area = Math.PI * Math.pow(a: radio, b: 2);
        double perimetro = 2 * Math.PI * radio;

        System.out.println("El área de la circunferencia es: " + area);
        System.out.println("El perímetro de la circunferencia es: " + perimetro);
    }
}
```

OBJETOS Y ESTRUCTURAS DE DATOS

Vamos a crear una clase Circunferencia para encapsular la lógica relacionada con el cálculo del área y el perímetro de la circunferencia. Así ocultamos los datos detrás de la clase y solo haremos uso de las funciones que calculan el área y perímetro llamandolas en el main.

```java
package es.ed.carolinadranicerdranicer_cleancode;

import java.util.Scanner;

public class CarolinaDranicerDranicer_CleanCode {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(source:System.in);
        System.out.print(s: "Ingrese el radio de la circunferencia: ");
        double radio = scanner.nextDouble();
        Circunferencia circunferencia = new Circunferencia(radio);

        double area = circunferencia.calcularArea();
        double perimetro = circunferencia.calcularPerimetro();

        System.out.println("El área de la circunferencia es: " + area);
        System.out.println("El perímetro de la circunferencia es: " + perimetro);

    }
}
```

```java
package es.ed.carolinadranicerdranicer_cleancode;

public class Circunferencia {
    private double radio;

    public Circunferencia(double radio) {
        this.radio = radio;
    }

    public double calcularArea() {
        return Math.PI * Math.pow(a: radio, b: 2);
    }

    public double calcularPerimetro() {
        return 2 * Math.PI * radio;
    }
}
```

MANEJO DE ERRORES

Vamos a agregar un validador para el radio de la circunferencia por si se introduce un radio cero o negativo ,así se lanzará una excepción. Buscamos que el programa devuelva la excepción en lugar del retorno. También se ha implementado un try-catch para que en el caso de que introduzcamos un valor no numérico, vuelva a pedir el valor.

```java
package es.ed.carolinadranicerdranicer_cleancode;

import java.util.Scanner;

public class CarolinaDranicerDranicer_CleanCode {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(source:System.in);
        double radio = 0;
        boolean validInput = false;

        while (!validInput) {
            try {
                System.out.print(s: "Introduzca el radio de la circunferencia: ");
                radio = scanner.nextDouble();
                validInput = true;
            } catch (Exception e) {
                System.out.println(x: "Error, introduzca un número válido.");
                scanner.nextLine();
            }
        }

        Circunferencia circunferencia = new Circunferencia(radio);

        double area = circunferencia.calcularArea();
        double perimetro = circunferencia.calcularPerimetro();

        System.out.println("El área de la circunferencia es: " + area);
        System.out.println("El perímetro de la circunferencia es: " + perimetro);

    }
```

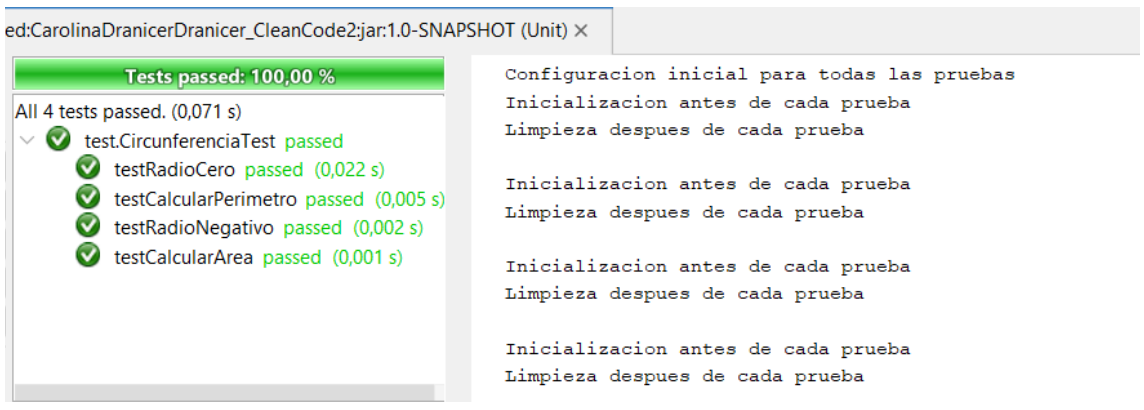
```java
package es.ed.carolinadranicerdranicer_cleancode;

public class Circunferencia {
    private double radio;

    public Circunferencia(double radio) {
        if (radio <= 0) {
            throw new IllegalArgumentException(s: "El radio debe ser mayor que cero");
        }
        this.radio = radio;
    }

    public double calcularArea() {
        return Math.PI * Math.pow(a: radio, b: 2);
    }

    public double calcularPerimetro() {
        return 2 * Math.PI * radio;
    }
}
```

TESTS DE ERRORES

Vamos a crear pruebas unitarias utilizando JUnit para cubrir todos los posibles casos. Primero comprobamos que los métodos calcularArea() y calcularPerimetro() realizan los cálculos correctamente.

También implementamos pruebas unitarias para comprobar que la excepción IllegalArgumentException se lanza siempre que el radio es negativo o cero. Así como las pruebas Before y After.

Nos aseguramos de que cada test realiza solo una cosa, manteniendo los tests limpios, cumpliendo la ley de TDD y la regla FIRST, además se ha implementado al menos dos asserts.



```java
public class CircunferenciaTest {

    @Test
    @DisplayName("Area")
    public void testCalcularArea() {
        Circunferencia circunferencia = new Circunferencia(radio: 5);
        assertEquals(expected: 78.53981633974483, actual: circunferencia.calcularArea(), delta: 0.0000001);
    }

    @Test
    @DisplayName("Perimetro")
    public void testCalcularPerimetro() {
        Circunferencia circunferencia = new Circunferencia(radio: 5);
        assertEquals(expected: 31.41592653589793, actual: circunferencia.calcularPerimetro(), delta: 0.0000001);
    }

    @Test
    @DisplayName("RadioNegativo")
    public void testRadioNegativo() {
        try {
            new Circunferencia(radio: -5);
        } catch (IllegalArgumentException e) {
            // Se espera que se lance la excepción.
        }
    }

    @Test
    @DisplayName("RadioCero")
    public void testRadioCero() {
        try {
            new Circunferencia(radio: 0);
        } catch (IllegalArgumentException e) {
            // Se espera que se lance la excepción.
        }
    }
        @BeforeAll
    public static void setUpAll() {
        System.out.println(x: "Configuracion inicial para todas las pruebas");
    }

    @BeforeEach
    public void setUp() {
        System.out.println(x: "Inicializacion antes de cada prueba");
    }

    @AfterEach
    public void tearDown() {
        System.out.println(x: "Limpieza despues de cada prueba");
    }

    @AfterAll
    public static void tearDownAll() {
        System.out.println(x: "Limpieza despues de todas las pruebas");
    }
}
```

CLASES

Para cumplir con este bloque de clean code los atributos y métodos están organizados siguiendo la estructura recomendada. He mantenido la clase Circunferencia pequeña y enfocada en su objetivo que es calcular el área y el perímetro de una circunferencia.
La clase sigue el principio de responsabilidad única y la organización del código está preparada para el cambio, cuya única razón para cambiar es si las fórmulas de cálculo cambian. Esto se ha logrado siguiendo el principio de diseño Open-Closed y aplicando el principio de inversión de dependencias, no mezclado la construcción de la clase con su uso.
La clase está cohesionada ya que los métodos manipulan solo la variable de instancia radio.

REPOSITORIO GITHUB : https://github.com/CarolinaDranicer