Taller 05

1. Con ayuda de los IDEs[1] presentados en clase y teniendo en cuenta la definición de las clases `ComplexNumber` y `ComplexCalculator` (las cuales no tienen errores) **encuentre, mencione y explique** cuáles son los errores de compilación (aquellos que no permitan que el programa sea ejecutado correctamente) de los siguientes fragmentos de código Java. Algunos fragmentos de código pueden tener más de un error, o no tener ☺.

```java
package oop.assignments.a03.p1;

public class ComplexNumber {

    private double realPart;
    private double imaginaryPart;

    public ComplexNumber() {
        this(0.0, 0.0);
    }

    public ComplexNumber(double realPart, double imaginaryPart) {
        this.realPart = realPart;
        this.imaginaryPart = imaginaryPart;
    }

    public double getImaginaryPart() {
        return imaginaryPart;
    }

    public void setImaginaryPart(double imaginaryPart) {
        this.imaginaryPart = imaginaryPart;
    }

    public double getRealPart() {
        return realPart;
    }

    public void setRealPart(double realPart) {
        this.realPart = realPart;
    }
}

```

Figura 1 – Definición de la clase ComplexNumber

[1] IDE Integrated development environment (Entorno de desarrollo integrado)

```java
package oop.assignments.a02.p1;

public class ComplexCalculator {

    public static ComplexNumber sum(ComplexNumber numA, ComplexNumber numB) {

        double sumRealPart = numA.getRealPart() + numB.getRealPart();
        double sumImaginaryPart = numA.getImaginaryPart() + numB.getImaginaryPart();

        ComplexNumber result = new ComplexNumber();
        result.setRealPart(sumRealPart);
        result.setImaginaryPart(sumImaginaryPart);

        return result;
    }

    public static ComplexNumber substraction(ComplexNumber numA, ComplexNumber numB) {

        double subRealPart = numA.getRealPart() - numB.getRealPart();
        double subImaginaryPart = numA.getImaginaryPart() - numB.getImaginaryPart();

        ComplexNumber result = new ComplexNumber();
        result.setRealPart(subRealPart);
        result.setImaginaryPart(subImaginaryPart);

        return result;
    }

    public static ComplexNumber opposite(ComplexNumber number) {

        ComplexNumber opposite = new ComplexNumber();
        opposite.setRealPart(-number.getRealPart());
        opposite.setImaginaryPart(-number.getImaginaryPart());

        return opposite;
    }
}
```

Figura 2 - Definición de la clase ComplexCalculator

1.1

```java
package oop.assignments.a03.p1;

public class A03P11 {

    public static void main(String[] args) {

        ComplexNumber number = new ComplexNumber("2+5i");

        ComplexNumber result = ComplexCalculator.opposite(number);

        System.out.println("Opposite number is: " + result.getRealPart() + " + "
                + result.getImaginaryPart() + "i");

    }
}
```

1.2

```java
package oop.assignments.a03.p1;

public class A03P12 {

    public void main(String[] args) {

        ComplexCalculator.sum(new ComplexNumber(5f, 6f),
                new ComplexNumber('c', 'i'));

        System.out.println("Real Part is: " + ComplexNumber.getRealPart());

    }
}
```

1.3

```java
package oop.assignments.a03.p1;

public class A03P13 {

    private ComplexNumber numberA;
    public static void main(String[] args) {

        numberA = new ComplexNumber();
        ComplexNumber numberB = new ComplexNumber(14, 15, 56);

        ComplexNumber result = ComplexCalculator.substraction(numberA, numberB);

        System.out.println("Number is:" + result.getRealPart() + " + "
                + result.getImaginaryPart() + "i");

    }
}
```

1.4

```java
package oop.assignments.a03.p1;

public class A03P14 {

    private ComplexNumber numberA;

    public A03P14();

    public A03P14(ComplexNumber numberA, int sign) {
        if (sign > 0) {
            this.setNumberA(numberA);
        } else {
            this.setNumberA(numberA.opposite());
        }
    }

    public A03P14(ComplexNumber) {
        this(ComplexNumber, 1);
    }

    public ComplexNumber getNumberA() {
        return numberA;
    }

    public void setNumberA(ComplexNumber numberA) {
        this.numberA = numberA;
    }
}
```

1.5

```java
package oop.assignments.a03.p1;

public class A03P15 {

    public static void main(String[] args) {

        ComplexNumber[] array = new ComplexNumber[5];
        Arrays.fill(array, new ComplexNumber());

        array = Arrays.sort(array);

        System.out.println(Arrays.toString(array));
    }
}
```

2. De los siguientes enunciados identifique las clases involucradas y realice el respectivo diagrama de clases UML con atributos, métodos, nombres, relaciones con multiplicidad y rótulos.
   NOTA: se debe identificar al menos una clase de datos, una de lógica de negocio y una de interfaz de usuario.

2.1 Calculadora de áreas y perímetros

- La calculadora debe ofrecer al usuario la opción de calcular el área y perímetro para cuadrados, rectángulos, triángulos, círculos y trapecios.
- El sistema deberá solicitarle al usuario las diferentes medidas requeridas para realizar el cálculo.
- El programa no debe terminar hasta que el usuario lo indique en el menú principal del programa.
- El sistema debe ser amigable, es decir, debe guiar al usuario indicándole en que forma debe ingresar los valores así como también debe validar que el usuario ingrese los valores correctamente.

2.2 Skytale

En criptografia el skytale es una herramienta que antiguamente fue usada para cifrar información, en este caso usted debe implementar la versión en Java, el funcionamiento puede ser consultado en esta página o en esta otra. Adicionalmente:

- El programa no debe terminar hasta que el usuario lo indique en el menú principal del programa.
- El sistema debe ser amigable, es decir, debe guiar al usuario indicándole en que forma debe ingresar los valores así como también debe validar

que el usuario ingrese los valores correctamente.

2.3 Analizador de frecuencias

- El analizador de frecuencias permite que para un texto ingresado por el usuario el programa calcule las frecuencias para los caracteres individuales, bigramas y trigramas encontrados dentro del texto, estas frecuencias pueden ser mostradas numéricamente o mediante un diagrama de frecuencias (de manera similar a la tarea 00), en esta página se puede ver un ejemplo de un contador de bigramas, tigramas y caracteres individuales.
- La salida del programa independiente del formato (numérico o diagrama) debe mostrar la lista de caracteres, bigramas o trigramas ordenados de forma descendente.
- El programa no debe terminar hasta que el usuario lo indique en el menú principal del programa.
- El sistema debe ser amigable, es decir, debe guiar al usuario indicándole en que forma debe ingresar los valores así como también debe validar que el usuario ingrese los valores correctamente.

2.4 Picas y fijas

El juego de picas y fijas es un juego de adivinación y destreza matemática (detalles en esta página).

- El programa debe simular al jugador que genera el número y evalúa las suposiciones del jugador humano indicándole el número de picas y fijas logradas por cada turno
- El máximo número de intentos será de 9
- Después de cada turno el programa debe mostrar el historial de las jugadas realizadas, indicando que número se usó como intento y el número de picas y fijas logradas en dicho turno.
- El programa no debe terminar hasta que el usuario lo indique en el menú principal del programa.
- El sistema debe ser amigable, es decir, debe guiar al usuario indicándole en que forma debe ingresar los valores así como también debe validar que el usuario ingrese los valores correctamente.

3. Realice las respectivas implementaciones funcionales (en Java) de cada uno de los 3 enunciado escogidos del punto 2.