# Ejercicio Teórico 2

INGENIERÍA DEL SOFTWARE II

MARCOS ISABEL LUMBRERAS
ALEJANDRO GUTIÉRREZ ROMERO-SALAZAR

# Introducción

Está wiki está destinada para el ejercicio número 2 del trabajo teórico. Este ejercicio será desarrollado por `Marcos Isabel Lumbreras` y `Alejandro Gutiérrez Romero-Salazar`

# Enunciado del Ejercicio

Escriba y pruebe los métodos necesarios para calcular las raíces de la ecuación de segundo grado. Además de las consiguientes consideraciones semánticas del problema, tenga en cuenta también las siguientes:

• La interfaz de usuario será en línea de comando. Se tienen que desarrollar clases con métodos específicos para leer de teclado y escribir en pantalla cadenas y números.

• Utilice excepciones en aquellos puntos del programa donde sea necesario controlar el flujo del programa.

• No acople las clases de dominio con interfaz de usuario (p.ej. no genere salidas por pantalla en las clases de la lógica de dominio)

# Problemas a tratar

1. Escribir el código que se identifique en base al anterior enunciado.
2. Identificar las variables a probar del código e identificar sus valores de prueba.
3. Calcular el número máximo de casos de prueba y hacer dos conjuntos de prueba (each use y pairwaise).
4. Realizar un conjunto de prueba para alcanzar la cobertura de decisiones.
5. Realizar un conjunto de prueba para alcanzar la cobertura MC/DC.
6. Conjunto de pruebas para alcanzar cobertura Pairwaise
7. Conjunto de casos de prueba para cobertura de decisiones
8. Conjunto de casos de prueba cobertura MC DC
9. Comentarios Apartados 4, 5 y 6

# 1. CÓDIGO

## ENUNCIADO

- Escribir el código que se identifique en base al anterior enunciado.

### Código

En este apartado, se podrá observar la implementación del problema. Este se ha dividido en 4 clases, la primera es el objeto Ecuación, donde se almacenará el método para el calculo de la ecuación. Después, tendremos la excepción InvalidInputException y la interfaz IU_Usuario para controlar los errores. Por último, se encuentra la clase Main donde se probarán todas las clases.

### Ecuación

Esta clase cuenta con 3 métodos. El primero, calcularD, sirve para comprobar si el resultado de dentro de la raíz es negativo. Si se diera el caso, el programa mostrará que no existe raíz para los datos dados. Los otros dos que hay, sirven para calcular los resultados de la ecuación.

```
public class Ecuacion {

private double a, b, c;

    public Ecuacion(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public double calcularD() {
        return b * b - 4 * a * c;
    }

    public String[] calcularRaices() {
        double d = calcularD();

        if (d >= 0) {
            // Caso de discriminante positivo o igual a cero
            double raiz1 = (-b + Math.sqrt(d)) / (2 * a);
            double raiz2 = (-b - Math.sqrt(d)) / (2 * a);

            return new String[]{String.valueOf(raiz1),
String.valueOf(raiz2)};
        } else {
            // Caso de discriminante negativo
            double parteReal = -b / (2 * a);
            double parteImaginaria = Math.sqrt(Math.abs(d)) / (2 * a);

            String raiz1 = parteReal + " + " + parteImaginaria + "i";
```

```
            String raiz2 = parteReal + " - " + parteImaginaria + "i";

            return new String[]{raiz1, raiz2};
        }
    }
}
```

## InvalidInputException

```
public class InvalidInputException extends Exception {
        public InvalidInputException(String message) {
                super(message);
            }
}
```

## IU_Usuario

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class IU_Usuario {
         private Scanner scanner;

     public IU_Usuario() {
         this.scanner = new Scanner(System.in);
     }

     public double obtenerNumero(String mensaje) throws InvalidInputException
{

         while (true) {
             try {
                 System.out.print(mensaje);
                 return scanner.nextDouble();
             } catch (InputMismatchException e) {
                 scanner.nextLine();
                 throw new InvalidInputException("Ingrese un número
válido.");
             }
         }
     }
}
```

## Main

```
import java.util.Arrays;
import java.util.InputMismatchException;
import java.util.Scanner;

public class main {

        private static IU_Usuario iu = new IU_Usuario();

        public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);

        try {
            double a = iu.obtenerNumero("Ingrese el coeficiente a: ");
            double b = iu.obtenerNumero("Ingrese el coeficiente b: ");
            double c = iu.obtenerNumero("Ingrese el coeficiente c: ");

            Ecuacion ecuacion = new Ecuacion(a, b, c);


            String[] raices = ecuacion.calcularRaices();

            System.out.println(Arrays.toString(raices));

        } catch (InvalidInputException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (ArithmeticException e) {
            System.out.println("Error: División por cero. Asegúrese de que
'a' no sea cero.");
        } finally {
            scanner.close();
        }
    }

}
```

# 2. Variables

## ENUNCIADO

- Identificar las variables a probar del código e identificar sus valores de prueba.

### Variables a tener en cuenta

Para la realización del problema, tendremos que tener en cuenta cada una de las variables que intervienen en los métodos de la clase Ecuación. Sabemos que una ecuación de segundo grado posee 3 números, en los que en el método hemos llamado 'a', 'b' y 'c'.

Observando las distintas variables, tendremos que tener en cuenta de que a nunca puede ser 0, ya que el denominador nunca puede ser igual a 0. Teniendo esto en cuenta, el resto de valores sean positivos o negativos podrán ser validos para los 3, incluido 0 para los valores de 'b' y 'c'.

Otro apartado que hay que tener en cuenta es el del método 'CalcularRaices'. En este método, hemos añadido dos partes. Una para cuando 'calcularD' es positivo o igual a cero en el que se mostrarán las raíces reales, y otro para cuando es menor que cero, en el que se mostrarán tanto la parte real como la imaginaria.

# 3. Valores de prueba

## ENUNCIADO

- Identificar los valores de pruebas para cada una de las variables anteriores usando las tres técnicas vistas en teoría, especificando para cada una cual es la que ha sido usada.

### Técnicas Utilizadas

### 1. Partición

Consistirá en dividir los conjuntos de números de entrada, en la que para cada conjunto haremos una selección de valores para comprobar el funcionamiento del proyecto. De esta manera podremos observar el resultado que nos proporciona. El conjunto de valores y lo números elegidos se podrá observar en las columnas 'Partición' y 'Valores'.

### 2. Valores límite

Consistirá en seleccionar dos valores en los límites de los conjuntos seleccionados en el apartado de Partición.

- Variante ligera: en variante tomaremos de prueba los valores que se encuentre en el límite y los valores adyacentes de las clases.
- Variante pesada: con esta variante, lo que haremos es tomar el valor inmediatamente inferior al límite.

Todos estos valores se podrán ver en la tabla de abajo, en las columnas llamadas: 'V. Ligera' y 'V. Pesada'.

### 3. Conjetura de errores

Tomaremos los distintos valores que creemos pueden dar error de alguna manera al comportamiento del sistema. Estos valores se podrán observar en la tabla de abajo, en la columna llamada 'Conj. Errores'.

TABLA DE VALORES

| Parametros | Particiones | V. Ligera | V. Pesada | Conj. Errores |
|---|---|---|---|---|
| a | (-∞,0) U (0,+∞) | -1, 1 | -0.00001, 0.00001 | -10000000, 10000000 |
| b | (-∞,+∞) | -1, 1, 0 | --0.00001, 0.00001 | -10000000, 10000000 |
| c | (-∞,+∞) | -1, 1, 0 | 0.00001, 0.00001 | -10000000, 10000000 |

# 4. Número máximo posible de casos de prueba

## Enunciado

- Calcular el número máximo posible de casos de pruebas que se podrían generar a partir de los valores de pruebas (combinatoria).

### Resolución

Como vimos en la tabla del apartado 2, definimos los distintos rango que podían adoptar las 3 variables del problema. Por lo tanto, para simplificar el número de casos de prueba, podremos ver la cantidad de valores que pueden coger las distintas variables:

| Variable | Nº Valores |
|---|---|
| a | 6 |

| Variable | Nº Valores |
|---|---|
| b | 7 |
| c | 7 |

### Cálculo Combinatorio

Para saber el número total de casos de prueba, lo que haremos es multiplicar la cantidad de valores distintos para cada variable:

TotalCombinaciones = a x b x c = 6 x 7 x 7

De esta manera, el número máximo de casos de prueba distintos que podriamos generar es de 294.

# 5. Conjunto de casos de pruebas para cumplir con Each Use

## ENUNCIADO

- Defina un conjunto de casos de pruebas para cumplir con each use (cada valor una vez)

### RESOLUCIÓN

Para poder cumplir el Each Use, definiremos un conjunto de casos de uso que necesitaremos que cumplan cada una de las variables. Como en el apartado 4 , en la tabla resultan, tenemos que para 'a' existen 6 valores de pruebas, y, para 'b' y 'c' existen 7 valores de prueba. Como resultando de todo esto, como el número mayor es 7, tendremos ese número de casos de prueba para nuestro proyecto.

| Caso de Prueba | a | b | c |
|---|---|---|---|
| 1 | -1 | 5 | 10 |
| 2 | 2 | 6 | -20 |
| 3 | 3 | 7 | 30 |
| 4 | -4 | 8 | 40 |
| 5 | 5 | 9 | 50 |
| 6 | 6 | -10 | 60 |
| 7 | -7 | -11 | -70 |

# 6. Conjunto de pruebas para alcanzar cobertura

## ENUNCIADO

- Defina conjuntos de pruebas para alcanzar cobertura pairwaise usando el algoritmo explicado en clase. Se pueden comprobar los resultados con el programa PICT2

### RESOLUCIÓN

Teniendo en cuenta la tabla realizada en el apartado 4:

| Variable | Nº Valores |
|---|---|
| a | 6 |
| b | 7 |
| c | 7 |

Teniendo en cuenta que los valores más altos son los dos 7, tendremos un total de 49 casos de pruebas a realizar. Como resultado del calculo anterior, haremos una donde resumiremos algunos casos de uso que consideramos importantes:

| Caso de Prueba | a | b | c |
|---|---|---|---|
| 1 | -1 | 5 | -10 |
| 2 | 2 | 6 | 20 |
| 3 | 3 | 7 | 30 |
| 4 | 4 | -1 | 40 |
| 5 | 5 | 2 | 50 |
| 6 | 6 | 3 | -60 |
| 7 | 7 | -4 | 70 |
| 8 | 8 | 10 | 80 |
| 9 | 9 | 9 | 90 |
| 10 | 10 | 8 | 100 |
| 11 | 1 | 6 | 70 |
| 12 | -2 | -7 | 60 |

| | | | |
|---|---|---|---|
| 13 | 3 | 8 | 50 |
| 14 | -4 | 9 | 40 |
| 15 | 5 | 10 | -30 |
| ... | ... | ... | ... |

# 7. Conjunto de casos de prueba para cobertura de decisiones

# ENUNCIADO

- Para los trozos de código que incluyan decisiones, proponga conjunto de casos de prueba para alcanzar cobertura de decisiones

## 1º Condicional

```
if (d >= 0) {
            // Caso de discriminante positivo o igual a cero
            double raiz1 = (-b + Math.sqrt(d)) / (2 * a);
            double raiz2 = (-b - Math.sqrt(d)) / (2 * a);

            return new String[]{String.valueOf(raiz1),
String.valueOf(raiz2)};
        }
```

En el primer condicional que nos encontramos, es cuando d es menor o igual que 0, en el que se mostrarán las distintas raíces. En caso de se 0, solo habrá una única raíz. Por lo tanto, la tabla de verdad resultante es la siguiente:

| Condiciones | a | b | c |
|---|---|---|---|
| T | 2 | -3 | -5 |
| F | 2 | 3 | 5 |

## 2º Condicional

```
else {
            // Caso de discriminante negativo
            double parteReal = -b / (2 * a);
            double parteImaginaria = Math.sqrt(Math.abs(d)) / (2 * a);

            String raiz1 = parteReal + " + " + parteImaginaria + "i";
            String raiz2 = parteReal + " - " + parteImaginaria + "i";

            return new String[]{raiz1, raiz2};
        }
```

En el segundo condicional, es cuando d es mayor que 0, en el que se mostrarán las distintas raíces imaginarias. Por lo tanto, la tabla de verdad resultante es la siguiente:

| Condiciones | a | b | c |
|---|---|---|---|
| F | 2 | -3 | -5 |
| T | 2 | 3 | 5 |

## 3º Condicional

```
 public double obtenerNumero(String mensaje) throws InvalidInputException {

         while (true) {
             try {
                 System.out.print(mensaje);
                 return scanner.nextDouble();
             } catch (InputMismatchException e) {
                 scanner.nextLine();
                 throw new InvalidInputException("Ingrese un número
válido.");
             }
         }
     }
```

En el tercer condicional, se encuentra cuando el usuario da un valor que no sea un número para alguna de las variables. La tabla de verdad sería la siguiente:

| Condiciones | a | b | c |
|---|---|---|---|
| F | m | | |
| T | 2 | | |

## 4º Condicional

```
catch (ArithmeticException e) {
            System.out.println("Error: División por cero. Asegúrese de que
'a' no sea cero.");
        }
```

En el cuarto condicional nos encontraremos una Excepción que salta cuando 'a' es igual a 0, esto ocurre porque el denominador no puede ser 0. La tabla de verdad resultante sería:

| Condiciones | a | b | c |
|---|---|---|---|
| F | 0 | | |
| T | 2 | | |

### Casos de prueba finales

Tras obtener cada uno de los condicionales, podremos sacar la tabla final sobre los casos de prueba generales que satisfaga el problema. El conjunto de casos de prueba es el siguiente:

| a | b | c |
|---|---|---|
| 2 | -3 | -5 |
| 2 | 3 | 5 |

Como podemos observa, no tendremos ningún problema con las excepciones, ya que todos son valores numéricos y 'a' nunca es 0. Por otro lados tenemos los dos ejemplos, uno con resultado de raíces normales, y otro de raíces imaginarias.

# 8. Conjunto de casos de prueba MC DC

## ENUNCIADO

- Para los trozos de código que incluyan decisiones, proponga conjunto de casos de prueba para alcanzar cobertura MC/DC.

1º Condicional

```
if (d >= 0) {
            // Caso de discriminante positivo o igual a cero
            double raiz1 = (-b + Math.sqrt(d)) / (2 * a);
            double raiz2 = (-b - Math.sqrt(d)) / (2 * a);
```

```
            return new String[]{String.valueOf(raiz1),
String.valueOf(raiz2)};
        }
```

En el primer condicional que nos encontramos, es cuando d es menor o igual que 0, en el que se mostrarán las distintas raíces. En caso de se 0, solo habrá una única raíz. Por lo tanto, para obtener la co

| Condiciones | DECISIÓN | a | b | c |
|---|---|---|---|---|
| T | T | 2 | -3 | -5 |
| F | F | 2 | 3 | 5 |

## 2º Condicional

```
else {
            // Caso de discriminante negativo
            double parteReal = -b / (2 * a);
            double parteImaginaria = Math.sqrt(Math.abs(d)) / (2 * a);

            String raiz1 = parteReal + " + " + parteImaginaria + "i";
            String raiz2 = parteReal + " - " + parteImaginaria + "i";

            return new String[]{raiz1, raiz2};
        }
```

En el segundo condicional, es cuando d es mayor que 0, en el que se mostrarán las distintas raíces imaginarias. Por lo tanto, la tabla de verdad resultante es la siguiente:

| Condiciones | Decisión | a | b | c |
|---|---|---|---|---|
| F | F | 2 | -3 | -5 |
| T | T | 2 | 3 | 5 |

## 3º Condicional

```
 public double obtenerNumero(String mensaje) throws InvalidInputException {

         while (true) {
             try {
                 System.out.print(mensaje);
                 return scanner.nextDouble();
```

```
            } catch (InputMismatchException e) {
                scanner.nextLine();
                throw new InvalidInputException("Ingrese un número
válido.");
            }
        }
    }
```

En el tercer condicional, se encuentra cuando el usuario da un valor que no sea un número para alguna de las variables. La tabla de verdad sería la siguiente:

| Condiciones | Decisión | a | b | c |
|---|---|---|---|---|
| F | F | m | | |
| T | T | 2 | | |

## 4º Condicional

```
catch (ArithmeticException e) {
            System.out.println("Error: División por cero. Asegúrese de que
'a' no sea cero.");
        }
```

En el cuarto condicional nos encontraremos una Excepción que salta cuando 'a' es igual a 0, esto ocurre porque el denominador no puede ser 0. La tabla de verdad resultante sería:

| Condiciones | Decisión | a | b | c |
|---|---|---|---|---|
| F | F | 0 | | |
| T | T | 2 | | |

## Casos de prueba finales

Tras obtener cada uno de los condicionales, podremos sacar la tabla final sobre los casos de prueba generales que satisfaga el problema. El conjunto de casos de prueba es el siguiente:

| a | b | c |
|---|---|---|
| 2 | -3 | -5 |
| 2 | 3 | 5 |

Como podemos observa, no tendremos ningún problema con las excepciones, ya que todos son valores numéricos y 'a' nunca es 0. Por otro lados tenemos los dos ejemplos, uno con resultado de raíces normales, y otro de raíces imaginarias.

# 9. Comentarios Apartados 4, 5 y 6

## ENUNCIADO

- Comente los resultados del número de los casos de pruebas conseguidos en los apartados 4, 5 y 6 ¿qué podría decirse de la cobertura alcanzada?

### Apartado 4

En el apartado 4, hemos calculado el número máximo de casos de prueba que podríamos obtener en nuestro problema. Como resultado de eso, hemos obtenido 294 casos de prueba, que aunque no parezcan demasiados casos, hemos considerado reducirlos para nuestro problema.

### Apartado 5

En el apartado 5, hemos implementado la cobertura Each Use. Siguiendo los pasos que hemos explicado en el apartado, hemos llegado a la conclusión de que eran necesarios 7 casos de pruebas, por ser el número mayor. Teniendo en cuenta el tipo de problema, hemos considera que era correcto para la realización de este.

## Apartado 6

En el apartado 6, hemos implementado la cobertura PairWaise. Teniendo este en cuenta, en este apartado hemos llegado a la conclusión de que eran necesarios 49 casos de prueba, pero al parecernos excesivos, solo hemos implementado 15.

## Conclusión

En conclusión, preferiríamos no elegir la opción de implementar todos los casos de pruebas posibles. Entre las otras dos, nos decantaríamos por la Each Use. Ya que para un ejercicio como el nuestro que es tan simple, no vemos necesario implementar tantos casos de pruebas para su realización. Sin embargo, consideramos que la PairWaise es mucho más completa y daría mejores resultados a la larga.