Programación con Java

TAREA 10

Alumno: Alejandro Rodríguez Jiménez
Profesor: Marc Esteve García
Curso 2024

Índice

Introducción

Durante el procedimiento de esta práctica aprenderemos sobre los bloques *try_catch* sus funcionalidades, como utilizarlo y cómo crear excepciones personalizadas. Este bloque que estaremos utilizando se conforma de dos partes obligatorias pero contiene un bloque "*finally*" el cual nos permitirá ejecutar una fracción de código independientemente del bloque "*try_catch", haya errores en el try y pasemos al catch o no.*

Ejercicio 01:

Para el primer ejercicio, donde tenemos que hacer un pequeño juego donde generamos uno aleatorio y tendremos que adivinarlos, solo he utilizado un *try_catch* donde en él intentamos pasar la entrada del usuario a un entero para poder compararlo con el número a adivinar y responder.

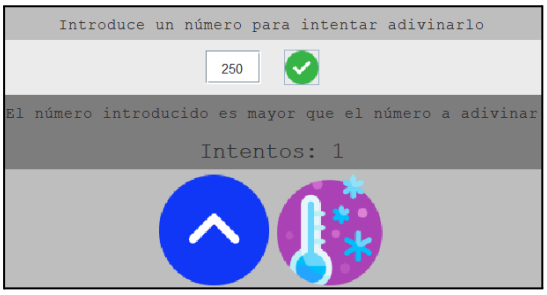
NumberFormatException

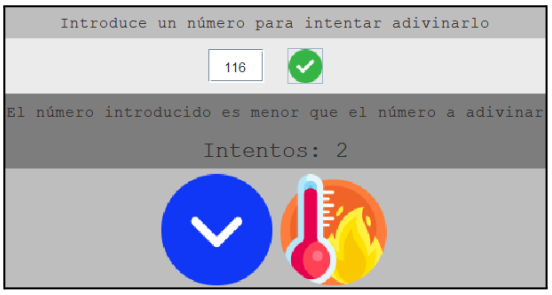
Se lanza para indicar que la aplicación ha intentado convertir una cadena a uno de los tipos numéricos, pero que la cadena no tiene el formato apropiado.

Try_catch

```
if (!numeroEntrada.getText().equals(anObject:"")) {
    int numeroIntroducido = 0;
    try {
        numeroIntroducido = Integer.parseInt(numeroEntrada.getText());
        mensajeMostrar.setLength(newLength:0);
        intentosMostrar.setLength(newLength:0);
    } catch (NumberFormatException exception) {
        mensajeMostrar.setLength(newLength:0);
        mensajeMostrar.append(str:"Entrada inválida. Debes ingresar un número.");
        vibrar(frame);
        error.set(newValue:true);
    }
```

Juego





Ejercicio 02:

En el segundo programa lo que haremos principalmente será crear un archivo aparte al programa del ejercicio que el ejercicio utilizará, y próximos proyectos también. En este archivo a parte, crearemos una clase para crear excepciones personalizadas dando el mensaje que queremos que muestre.

Programa

```
public class ejercicio02 {
    Run | Debug
    public static void main(String[] args) {
        try {
            System.out.println(x:"\033[32mIniciación del programa.\u001b[0m");
            throw new CustomEx(mensaje:"\033[35mExcepción personalizada\u001b[0m");
        } catch (CustomEx e) {
            System.out.println("\033[33mTipo de excepción capturada:\u001b[0m [ " + e.getMessage() + " ]");
        } finally {
            System.out.println(x:"\033[31mFinal programa.\u001b[0m");
        }
    }
}
```

Clase

```
public class CustomEx extends Exception {
    public CustomEx(String mensaje) {
        super(mensaje);
    }
}
```

Resultado

```
Iniciación del programa.
Tipo de excepción capturada: [ Excepción personalizada ]
Final programa.
```

Ejercicio 03:

El tercero será un programa que generará un número aleatorio entre 0 y 999, y dirá si el número generado es par o impar. Para ello, generamos el número con Random() y para determinar si es par o impar he pensado en añadirlo a la clase "*CustomEx*" para posibles necesidades.

Programa

```
public class ejercicio03 {
    Run | Debug
    public static void main(String[] args) {
        Random aleatorio = new Random();
        int numero = aleatorio.nextInt(origin:0, bound:1000);
        try {
            System.out.println(x:"\033[32mNúmero generado.\u001b[0m");
            CustomEx.esPar(numero);
        } catch (CustomEx e) {
            System.out.println("\033[33mTipo de excepción capturada:"+
            "\u001b[0m [ " + e.getMessage() + " ]");
        } finally {
            System.out.println(x:"\033[31mFinal programa.\u001b[0m");
        }
    }
}
```

Clase

```
public static void esPar(int numero) throws CustomEx {
    if (numero % 2 == 0) {
        throw new CustomEx(mensaje:"\033[35mNúmero par\u001b[0m");
    } else {
        throw new CustomEx(mensaje:"\033[35mNúmero impar\u001b[0m");
    }
}
```

Resultado

```
Número generado.
Tipo de excepción capturada: [ Número impar ]
Final programa.
```

```
Número generado.
Tipo de excepción capturada: [ Número par ]
Final programa.
```

Ejercicio 04:

Crearemos una pequeña calculadora muy simple para el ejercicio 4, en la que tendremos que pasarle los dos operadores y después decirle la operación que hará. Las disponibles son: sumar, restar, multiplicar, dividir, potencias, raíz cuadrada y raíz cúbica. Para el funcionamiento he creado una función para verificar que el contenido del "*JTextField*" sea un número, en caso contrario lanzará una excepción con un mensaje advirtiendo el error.

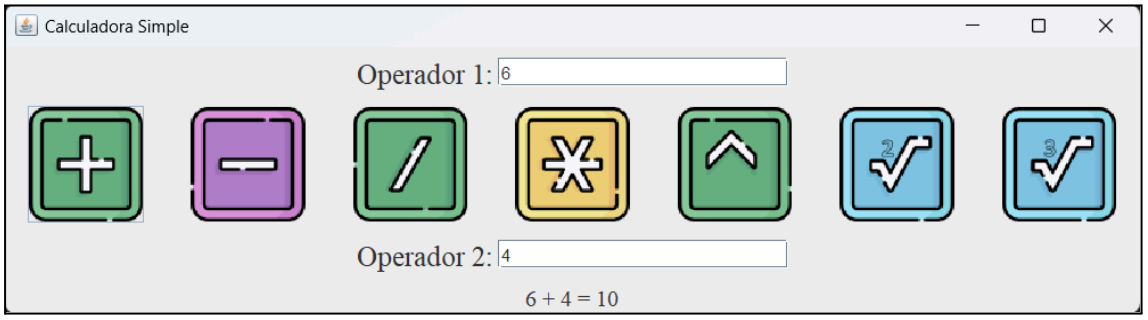
Try_catch

```
JButton mas = crearYEscalarBotonImagen(ruta:"imagenes/mas.png");
mas.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            verificacion(operador1, numOp:"1");
            verificacion(operador2, numOp:"2");
            double resultado = Double.parseDouble(operador1.getText()) + Double.parseDouble(operador2.getText());
            // double resultado = Double.parseDouble(operador1.getText()) - Double.parseDouble(operador2.getText()); // BOTÓN RESTAR
            // double resultado = Double.parseDouble(operador1.getText()) / Double.parseDouble(operador2.getText()); // BOTÓN DIVIDIR
            // double resultado = Double.parseDouble(operador1.getText()) * Double.parseDouble(operador2.getText()); // BOTÓN MULTIPLICAR
            // double resultado = Math.pow(Double.parseDouble(operador1.getText()), Double.parseDouble(operador2.getText())); // BOTÓN POTENCIA
            // double resultado1 = Math.sqrt(Double.parseDouble(operador1.getText())); // BOTÓN RAÍZ CUADRADA
            // double resultado1 = Math.cbrt(Double.parseDouble(operador1.getText())); // BOTÓN CÚBICA
            mostrarMensaje(frame, panelResultado, resultadoStringFormat(resultado));
        } catch (CustomEx custom) {
            System.out.println(custom.getMessage());
        }
    }
});
```

Función

```
public static void verificacion(JTextField operador, String numOp) throws CustomEx {
    if (operador.getText().equals(anObject:"")) {
        throw new CustomEx("Operador "+ numOp +": Está vacío");
    } else if (!operador.getText().matches(regex:"\\d+(\\.\\d+)?")) {
        throw new CustomEx("Operador "+ numOp +": No és un número");
    }
}
```

Resultado



6 - 4 = 2 | 6 / 4 = 1,5 | 6 * 4 = 24 | 6 ^ 4 = 1296 | ²√ de 6 = 2,449
²√ de 4 = 2 | ³√ de 6 = 1,817
³√ de 4 = 1,587

Ejercicio 05:

El quinto ejercicio es una ampliación de un ejercicio anterior que hicimos para generar contraseñas. Estas ampliaciones son: un método booleano para ver si es una contraseña fuerte, un método para generar una contraseña, métodos getters para contraseña y para longitud y un método set para longitud. Después crearemos un main donde ingresamos la cantidad de contraseñas que generará y la longitud de estas, para mostrarse he hecho un estilo tabla antes de los valores generados y con colores que los relacionan.

Ampliación clase “Password”

```java
public void setLongitud(int longitud) {
    this.longitud = longitud;
}

public int getLongitud() {
    return longitud;
}

public String getPass() {
    return password;
}

public Password generarPassword() {
    Password password = new Password(longitud);
    return password;
}
```

```java
public boolean verFuerza (String password) {
    int numeros = 0;
    int mayus = 0;
    int minus = 0;
    for (char caracter : password.toCharArray()) {
        if (Character.isDigit(caracter)) {
            numeros++;
        } else if (Character.isUpperCase(caracter)) {
            mayus++;
        } else if (Character.isLowerCase(caracter)) {
            minus++;
        }
    }
    if (mayus > 2 && minus > 1 && numeros > 5) {
        return true;
    } else {
        return false;
    }
}
```

Generación de contraseñas

```java
public class ejercicio05 {
    Run | Debug
    public static void main(String[] args) {
        Password contra = new Password();
        Scanner sc = new Scanner(System.in);

        System.out.println(x:"\033[33mIndique la cantidad de contraseñas que se generarán:\u001b[0m");
        int cantidad = sc.nextInt();

        System.out.println(x:"\033[33mIndique la longitud de las contraseñas:\u001b[0m");
        int longitud = sc.nextInt();
        sc.close();
        Password contrasenas[] = new Password[cantidad];
        boolean fuerzasContras[] = new boolean[cantidad];
        System.out.println("\033[35m  Contraseña \u001b[0m |  \u001B[34mEs fuerte?\u001b[0m  "+
        "|  \033[32mLongitud\u001b[0m  |  \033[31mNº Contraseña\u001b[0m  ");
        contra.setLongitud(longitud);
        for (int i = 0; i < contrasenas.length; i++) {
            contrasenas[i] = contra.generarPassword();
            fuerzasContras[i] = contrasenas[i].getFuerza();
            System.out.println("\033[35m"+contrasenas[i].getPass()+"\u001b[0m | \u001B[34m"+"\u001b[0m | \033[31m["+i+"]\u001b[0m");
        }
    }
}
```

```
Indique la cantidad de contraseñas que se generarán:
1
Indique la longitud de las contraseñas:
10
  Contraseña  |  Es fuerte?  |  Longitud  |  Nº Contraseña
rChQ4CtX=^ | false | 10 | [0]
```

Webgrafía

- Para ver API's de java:
  Java API

- Raíces con Math:
  Math.cbrt y Math.sqrt