T10 – Java Exceptions

T19 – SWING/AWT

1.

Escribe un programa, utilizando para ello el paradigma de POO, que juegue con el usuario a adivinar un número. Debe cumplir los siguientes requerimientos:

1. El ordenador debe generar un número entre 1 y 500, y el usuario tiene que intentar adivinarlo.
2. Cada vez que el usuario introduce un valor, el ordenador debe decirle al usuario si el número que tiene que adivinar es mayor o menor que el que ha introducido el usuario.
3. Cuando consiga adivinarlo, debe indicárselo e imprimir en pantalla el número de veces que el usuario ha intentado adivinar el número.
4. Si el usuario introduce algo que no es un número, debe indicarlo en pantalla, y contarlo como un intento indicando que no ha conseguido reconocer la entrada lanzando la excepción **InputMismatchException**.

```java
public TA10Ej01_AdivinaAdivinanza() {
    Random rand = new Random();
    this.numeroAdivinar = rand.nextInt(500) + 1;
    this.intentos = 0;
}
```

```
Bienvenido al juego de adivinar el número!
Intenta adivinar el número entre 1 y 500.
Introduce tu intento: 250
El número a adivinar es menor.
Introduce tu intento: 100
El número a adivinar es mayor.
```

```
Introduce tu intento: 154
¡Felicidades! ¡Has adivinado el número 154!
Número de intentos: 11
```

```
Bienvenido al juego de adivinar el número!
Intenta adivinar el número entre 1 y 500.
Introduce tu intento: hola
Entrada inválida. Debes ingresar un número.
Introduce tu intento:
```

Realmente no sé qué explicar sobre este código; es muy directo y fácil de entender.

El bucle try-catch nos asegura que el input de consola sea un entero (punto 4). Esto nos evita tener que volver a ejecutar el programa y poder seguir como si aún no hubiésemos hecho un input.

2.

Escribe un programa , utilizando para ello el paradigma de POO, que lance y capture una excepción customizada. Crea para ello una package diferente que puedas reutilizar para el resto de tus proyectos.

Recomendaciones:

El programa abre un bucle *try{}* en el que comienza mostrando un mensaje por pantalla. A continuación, crea un objeto de la clase *Exception*, indicando en su constructor un mensaje explicativo.

El resultado debe ser similar a esta captura de pantalla:

```
Mensaje mostrado por pantalla
Excepcion capturada con mensaje: Esto es un objeto Exception
Programa terminado
```

Me costó bastante entender este ejercicio, pero al final entendí la lógica de la clase Throwable de Java.

Dejo copiado aquí el resumen que puse en el código tras entenderlo.

```
/*
 * En Java, todas las excepciones deben ser subclases de la clase base Throwable, ya sea directa o indirectamente.
 * La clase Throwable tiene dos subclases principales: Exception y Error.
 * Como esperamos que sea una excepción que el programa pueda tratar, extendemos Exception.
 * Generamos el constructor con un string para poder introducir más tarde el mensaje que queremos.
 * Con throw new instaciamos la expeción de tpo CustomException y la capturamos y nombramos en catch.
 * En catch expresamos las instrucciones necesarias para que podamos tratar esa CustomException.
 * finally se ejecuta independientemente de que se haya generado y tratado una CustomException o no.
*/
```

```
<terminated> TA10Ej02_Exception [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (14 may 2024
Mensaje mostrado por pantalla
Excepción capturada con mensaje: Esto es un objeto Exception
Programa terminado
```

3.

Escribe un programa , utilizando para ello el paradigma de POO, que genere un número aleatorio e indique si el número generado es par o impar. El programa utilizará para ello el lanzamiento de una excepción.

Recomendaciones:

1. El programa utiliza la clase Random() para obtener un número aleatorio entre 0 y 999 (por poner un rango cualquiera).
2. Se determina si el número es par o impar y se lanza una excepción con el correspondiente mensaje para indicarlo (se limitará a mostrar el mensaje asociado a la excepción capturada).

El resultado debe ser similar a esta captura de pantalla:

```
Generando número aleatorio...
El numero aleatorio generado es: 203
Es impar
```

```
//      Determinamos si el número es par o impar
            if (esPar(numeroAleatorio)) {
                throw new ParException("Es PAR");
            } else {
                throw new ImparException("Es IMPAR");
            }
        } catch (ParException e) {
            System.out.println(e.getMessage());

        } catch (ImparException e) {
            System.out.println(e.getMessage());
        }
    }
```

Como hay dos escenarios posibles generamos una Exception para cada uno: ParException e ImparException.

Cada uno dirá si el número es PAR o IMPAR, respectivamente.

4.

Escribe un programa , utilizando para ello el paradigma de POO, que nos permita realizar cálculos simples (suma, resta, multiplicación, potencia, raíz cuadrada, raíz cubica y división). El programa ha des estar preparado para gestionar los posibles errores de calculo. Has de utilizar para ello el control de excepciones de JAVA.

Recomendaciones:

1. Utiliza siempre que sea posible las Excepciones definidas en la API de Java 8.
2. Puedes utilizar como interfaz visual Scanner o JOptionPane.
3. Estructura correctamente el código en diferentes packages.

He definido dos Exception:

- ArithmeticException (en División y Raíz Cuadrada): Captura errores de cálculo, como la división por cero y la raíz cuadrada de un número negativo.
- NumberFormatException (en JOptionPane): Captura errores de entrada, como cuando el usuario ingresa un valor no numérico.

```
} catch (ArithmeticException e) {
    JOptionPane.showMessageDialog(null, "Error: " + e.getMessage(),
            "Error de Cálculo", JOptionPane.ERROR_MESSAGE);
} catch (NumberFormatException e) {
    JOptionPane.showMessageDialog(null, "Error: Por favor ingrese un número válido",
            "Error de Entrada",JOptionPane.ERROR_MESSAGE);
```

Ejemplo:

```
case "Raíz Cuadrada":
    if (numero1 < 0) {
        throw new ArithmeticException(
                "No se puede calcular la raíz cuadrada de un número negativo");
```

5) Haz una clase llamada **Password** que siga las siguientes condiciones:

- Que tenga los atributos **longitud** y **contraseña** . Por defecto, la longitud sera de 8.

- Los constructores serán los siguiente:
  - ✓ Un constructor por defecto.
  - ✓ Un constructor con la longitud que nosotros le pasemos. Generara una contraseña aleatoria con esa longitud.

Los métodos que implementa serán:

- **esFuerte()**: devuelve un booleano si es fuerte o no, para que sea fuerte debe tener mas de 2 mayúsculas, mas de 1 minúscula y mas de 5 números.
- **generarPassword()**: genera la contraseña del objeto con la longitud que tenga.
- Método get para contraseña y longitud.
- Método set para longitud.

El ejercicio no especificaba que try-catch utilizar por lo que me tomé la libertad de realizar los que consideraba oportunos.

Seguí implementando el proceso de arrays de contraseña-boolean, pero añadí ciertos puntos extra al código, como son:

- Longitud mínima para crear la contraseña
- Opción de transformación de contraseñas generadas, de débil a fuerte (se mantienen las fuertes creadas anteriormente)
- Bucle para repetir la generación de otras contraseñas (reinicio del programa).
- Revisión de inputs del usuario

Podemos diferenciar entre tres bloques try-catch diferentes:

A. Leer longitud de la contraseña
B. Leer número de contraseñas a generar
C. Validar input del usuario

BLOQUES TRY-CATCH

A. Leer longitud de la contraseña

```java
// Método para obtener la longitud con manejo de excepciones
private static int obtenerLongitud(Scanner scanner) {
    while (true) {
        try {
            System.out.print("Ingrese la longitud de los passwords (mínimo "
                                        + Password.getLongitudMinima() + "): ");
            String input = scanner.next();
            if (!esNumeroEntero(input)) {
                throw new EntradaNoNumericaException("El valor ingresado no es un número entero.");
            }
            int longitud = Integer.parseInt(input);
            if (longitud < Password.getLongitudMinima()) {
                throw new LongitudInvalidaException("La longitud de la contraseña debe ser al menos "
                                        + Password.getLongitudMinima());
            }
            return longitud;
        } catch (LongitudInvalidaException | EntradaNoNumericaException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Error inesperado: " + e.getMessage());
            scanner.nextLine(); // Limpiar el buffer de entrada
        }
    }
```

BLOQUES TRY-CATCH

A. Leer longitud de la contraseña.

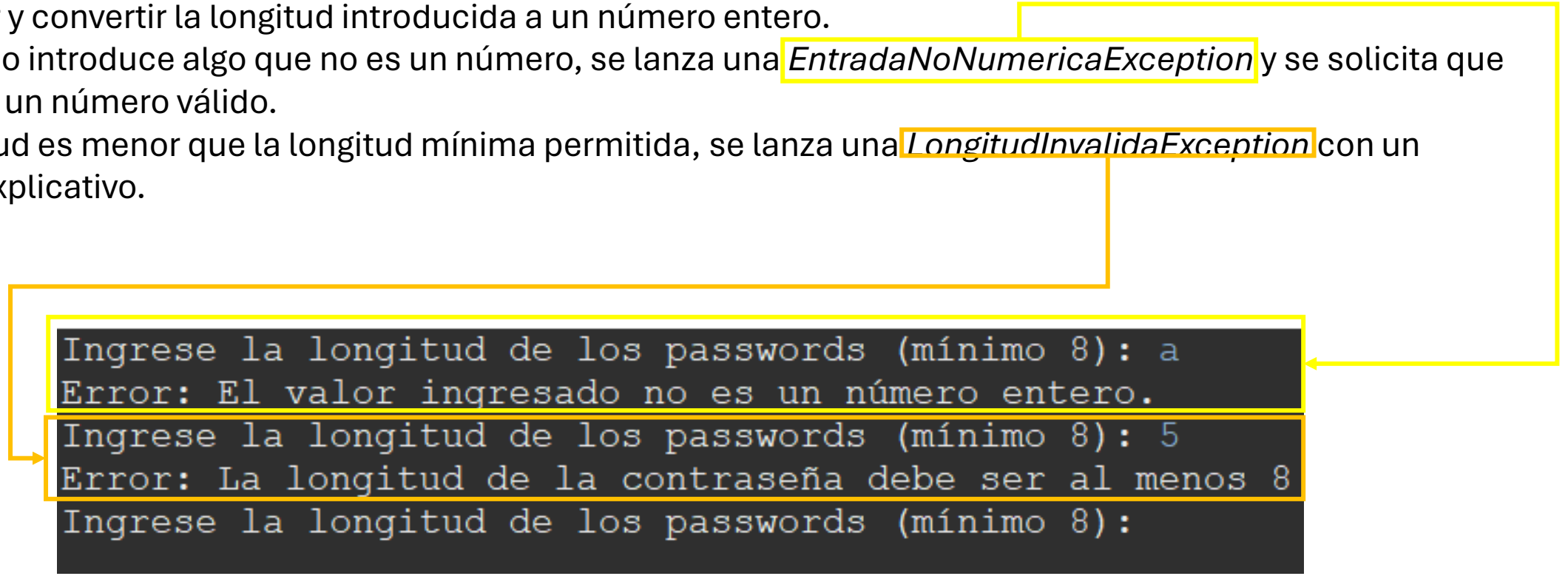
Propósito: Asegurarse de que el usuario introduce una longitud válida (un número entero positivo e igual o superior a la longitud mínima).

Funcionamiento:

try intenta leer y convertir la longitud introducida a un número entero.

a) Si el usuario introduce algo que no es un número, se lanza una *EntradaNoNumericaException* y se solicita que introduzca un número válido.

b) Si la longitud es menor que la longitud mínima permitida, se lanza una *LongitudInvalidaException* con un mensaje explicativo.



BLOQUES TRY-CATCH

B. Leer número de contraseñas a generar

```java
// Método para obtener la cantidad de passwords con manejo de excepciones
private static int obtenerCantidad(Scanner scanner) {
    while (true) {
        try {
            System.out.print("Ingrese la cantidad de passwords a generar: ");
            String input = scanner.next();
            if (!esNumeroEntero(input)) {
                throw new EntradaNoNumericaException("La cantidad ingresada no es un número entero.");
            }
            int cantidad = Integer.parseInt(input);
            if (cantidad <= 0) {
                throw new CantidadInvalidaException("La cantidad de passwords debe ser un número positivo.");
            }
            return cantidad;
        } catch (CantidadInvalidaException | EntradaNoNumericaException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("Error inesperado: " + e.getMessage());
            scanner.nextLine(); // Limpiar el buffer de entrada
        }
    }
}
```

BLOQUES TRY-CATCH

B. Leer número de contraseñas a generar.

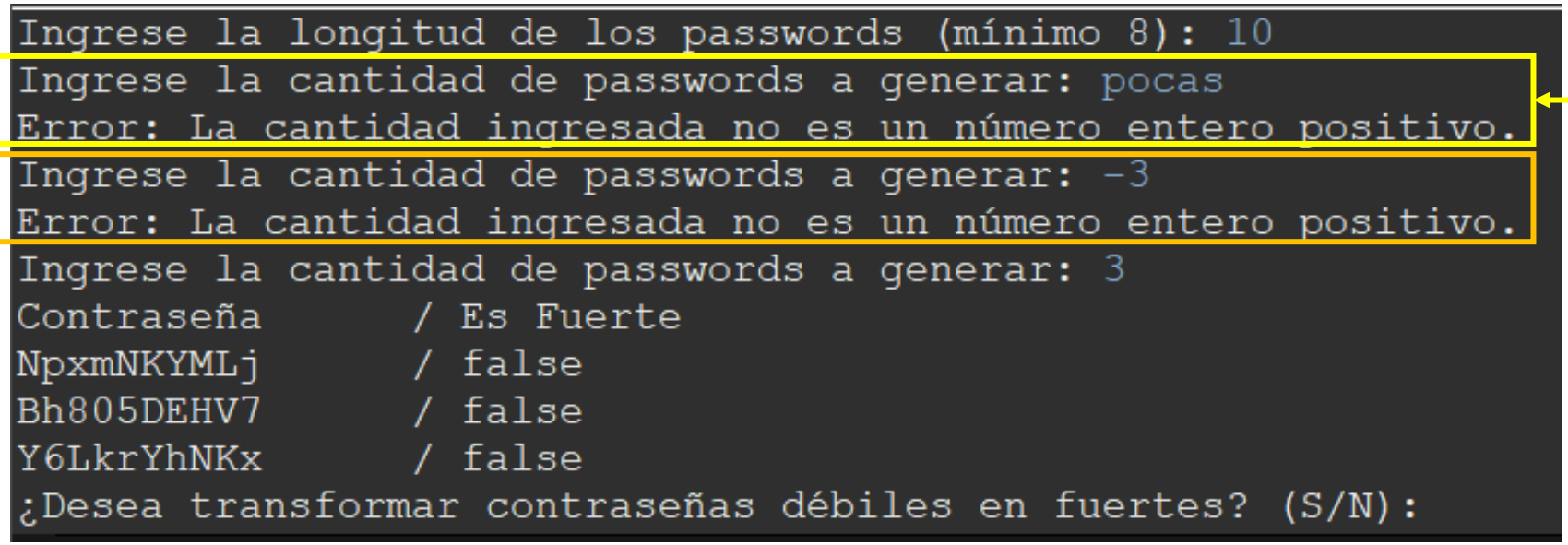
Propósito: Asegurarse de que el usuario introduce un número válido para la cantidad de contraseñas a generar.

Funcionamiento:

try intenta leer y convertir la cantidad introducida a un número entero.

a) Si el usuario introduce un número negativo, se lanza una *CantidadInvalidaException* y se solicita que introduzca un número entero positivo.

b) Si el usuario introduce caracteres que no sean numéricos, se lanza una *EntradaNoNumericaException* y se solicita que introduzca un numero entero positivo.



BLOQUES TRY-CATCH

C. Validar input del usuario (reinicio programa)

```java
// Método para preguntar al usuario si desea transformar contraseñas débiles en fuertes
private static boolean preguntarTransformarDebiles(Scanner scanner, Password[] passwords, boolean[] esFuerte) {
    while (true) {
        try {
            System.out.print("¿Desea transformar contraseñas débiles en fuertes? (S/N): ");
            String respuesta = scanner.next().toUpperCase();
            if (respuesta.equals("S")) {
                // Verificar si hay contraseñas débiles
                for (boolean fuerte : esFuerte) {
                    if (!fuerte) {
                        return true; // Si hay contraseñas débiles, retornar true
                    }
                }
                System.out.println("No hay contraseñas débiles para transformar.");
                return false; // Si no hay contraseñas débiles, retornar false
            } else if (respuesta.equals("N")) {
                return false;
            } else {
                throw new IllegalArgumentException("Ingrese S o N.");
            }
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

BLOQUES TRY-CATCH

C. Validar input del usuario: transformación de contraseñas y reinicio del programa.

Propósito: Asegurarse de que el usuario introduce una respuesta válida ('S' o 'N') para decidir si transformar las contraseñas débiles en fuertes.

Funcionamiento:

try lee la respuesta y la convierte a mayúsculas.

a) Si la respuesta no es 'S' o 'N', se lanza una *IllegalArgumentException* con un mensaje explicativo, solicitando al usuario que introduzca una respuesta válida.