Ejercicios de Excepciones

Relación de ejercicios

Ejercicio 1. Implementa un programa que pida al usuario dos números enteros y devuelva las operaciones matemáticas entre ellos (suma, resta, multiplicación y división).

```
Introduzca un número entero entre 0 y 100: 10
Introduzca un número entero entre 1 y 100: 3
OPERACIONES
        -> Suma: 10 + 3 = 13
        -> Resta: 10 - 3 = 7
        -> Multiplicación: 10 * 3 = 30
        -> División: 10 / 3 = 3
```

El programa debe ejecutarse mientras que ambos números (*num1* y *num2*) sean distintos. La última ejecución debe mostrar por pantalla las operaciones entre ambos números (que van a ser el mismo número). Se indica un ejemplo de salida.

Una vez implementado el programa, introduce los siguientes valores para *num1* y *num2* analizando y razonando lo que ocurre en cada iteración.

Iteración	num1	num2	Iteración	num1	num2
1	10	3	5	0	4
2	101		6	prog	
3	0	0	7	-1	
4	0	17,5	8	4	4

Ejercicio 2. Modifica el programa anterior para que capture la excepción que se produce en la iteración 4 y vuelva a pedirle el dato al usuario. Repite las iteraciones de la tabla anterior.

Ejercicio 3. Modifica el programa anterior para que si se produce cualquier otra excepción no contemplada, no se acabe la ejecución del programa, se muestre el mensaje de la excepción y se le sigan pidiendo números al usuario.

Ejercicio 4. Implementa una clase llamada `ValoracionException` que permita implementar una clase de tipo excepción y que sea lanzada cuando la valoración de un `Hardware` no sea correcta (no esté dentro de los valores posibles para una valoración).

Modifica la clase `Hardware`, que implementamos en su momento, para que lance esta nueva excepción. La nueva clase tendrá el constructor sin parámetros y un constructor que reciba y guarde una cadena de texto. El contenido de dicha cadena de texto podrá obtenerse llamando al método *getMessage*. Implementa un programa que permita validar el correcto funcionamiento de esta excepción (se debe imprimir en pantalla un aviso cuando se produzca la excepción). Para ello crea un nuevo `Monitor` y, posteriormente, accede a la media de valoraciones del `Monitor` y agrega una nueva valoración con valor 8 (en ese orden). Se produzca o no se produzca excepción se debe mostrar la información del monitor.

Ejercicio 5. Implementa las siguientes excepciones en un paquete llamado *excepciones*. Cada excepción deberá tener su mensaje correspondiente.

- `SaldoNegativoException`: cuyo mensaje será "*El saldo no puede ser negativo*".
- `CantidadInvalidaException`: cuyo mensaje será "*La cantidad debe ser un valor positivo*".

Ejercicio 6. Refactoriza la clase `Cuenta` que nos permitía modelizar una cuenta bancaria. Posteriormente, modifícala para conseguir lo siguiente:

- Cuando intenten setear un saldo negativo, se deberá lanzar la excepción `SaldoNegativoException`.
- Al retirar una cantidad de dinero, se debe capturar la excepción provocada por `SaldoNegativoException` y devolver que no se puede retirar dinero.
- Al intentar ingresar una cantidad de dinero negativa o de cero, se deberá lanzar la excepción `CantidadInvalidaException`.

Ejercicio 7. Mejora el siguiente código para que no se produzca ninguna salida abrupta del programa, es decir, debemos capturar cualquier excepción que se pueda producir, informar al usuario sobre dicha circunstancia y finalizar el programa de forma controlada.

```
public static void main(String[] args) {
	System.out.println("Introduzca una frase y pulse intro para finalizar:");
	String frase = Entrada.leerTexto();

	System.out.print("Introduzca un número de letra de la anterior frase: ");
	int posicion = Entrada.leerEntero();

	System.out.println(String.format("La letra en la posición %d es la '%c'", posicion,
frase.charAt(posicion-1)));

	System.out.println("[OK] Finalización controlada del programa");
}
```

Ejercicio 8. Dado el siguiente código fuente, trata de anticipar lo que ocurrirá si lo ejecutamos. Posteriormente, ejecuta el código y comprueba si estabas en lo cierto.

```
public class Ejercicio8 {

	public static String devuelveMensaje(int numero) {
		String mensaje = "Inicio del método";

		try {
			if (numero % 2 == 0) throw new Exception("El número es par");
			else mensaje = "El número es impar";
		}
		catch (Exception ex) {
			mensaje = "El número es lo contrario de impar";
		}
		finally {
			mensaje = "No sabemos si el número es par o impar";
		}

		return mensaje;
	}

	public static void main(String[] args) {
		System.out.println(devuelveMensaje(1));
		System.out.println(devuelveMensaje(2));
	}
}
```

Ejercicio 9. Analiza el siguiente programa e indica cuál sería la salida por pantalla al ejecutarlo. Una vez hechas tus predicciones, ejecútalo y valida si estabas en lo correcto.

```
private static int ejecuta() {
	int valor = 0;
	try {
		valor += 1;
		valor += Integer.parseInt("13");
		valor += 1;

		System.out.println("Valor al final del try: " + valor);
	}
	catch (NumberFormatException e) {
		valor += Integer.parseInt("13,0");
		System.out.println ("Valor al final del catch: " + valor);
	}
	finally {
		valor += 1;
		System.out.println ("Valor al final de finally: " + valor);
	}

	valor += 1;
	System.out.println ("Valor antes del return: " + valor);

	return valor;
}

public static void main(String[] args) {
	try {
		System.out.println(ejecuta());
	}
	catch (Exception e) {
		System.out.println("Se ha producido una excepción en la ejecución del código");
		e.printStackTrace();
	}
}
```

Ejercicio 10. Dada las siguientes clases, las cuales nos permiten modificar los valores de un array, trata las posibles excepciones que se puedan producir, es decir, todas aquellas excepciones que consideres relevante manejar. El programa no debe finalizar nunca su ejecución (salvo que el usuario cierre la ventana en la que se está ejecutando).

```
public class Datos {
	private double[] valores;

	public Datos() {
		valores = new double[] {9.17, 4.05, -3.00, 0.15, 2.18, -11.25, 7.35, 3.5, -30, 105.85};
	}

	public String toString() {
		return Arrays.toString(valores);
	}

	public void modificarValor(int indice, double nuevoValor) {
		this.valores[indice] = nuevoValor;
	}

	public void modificarValor(int indice, double nuevoValor) {
		this.valores[indice] = nuevoValor;
	}
}
```

```
public class EntradaSalidaView {
	private static final int OPCION_MENU_MINIMA = 1;
	private static final int OPCION_MENU_MAXIMA = 2;

	public void mostrarMenu() {
		System.out.println("\nMENÚ");
		System.out.println("********");
		System.out.println("1. Mostrar datos");
		System.out.println("2. Modificar datos");
	}
```

```
    public int pedirOpcion() {
        int opcion;
        boolean correcto;

        do {
            System.out.print("Introduzca una opción válida del menú: ");
            opcion = Entrada.leerEntero();
            correcto = Validador.estaEntre(opcion, OPCION_MENU_MINIMA, OPCION_MENU_MAXIMA);

            if(!correcto) {
                System.out.println("=> ¡Aviso! La opción de menú debe estar entre " +
OPCION_MENU_MINIMA + " y " + OPCION_MENU_MAXIMA);
            }
        }
        while(!correcto);

        return opcion;
    }

    public int solicitarIndiceArray(int longitudArray) {
        int posicion;
        boolean valido;

        do {
            System.out.print("Introduzca la posición del array que desea modificar: ");
            posicion = Entrada.leerEntero();
            valido = Validador.esPositivo(posicion) && posicion <= longitudArray;

            if(!valido) {
                System.out.println("=> El valor introducido debe ser un número entero
positivo menor o igual a " + longitudArray);
                }
        }
        while(!valido);
        return posicion - 1;
    }
```

```
    public double solicitarValor() {
        double valor;
        boolean valido = true;

        do {
            System.out.print("Introduzca un valor para almacenar en el array: ");
            valor = Entrada.leerDoble();
        }
        while(!valido);
        return valor;
    }

    public void mostrarMensaje(String mensaje)
    {
        System.out.println(mensaje);
    }
}
```

```
public class App {
    private static Datos datos;
    private static EntradaSalidaView vista;

    private static void inicializar() {
        datos = new Datos();
        vista = new EntradaSalidaView();
    }

    private static void gestionarOpcion(int opcion) {
        switch(opcion) {
            case 1:
                vista.mostrarMensaje("DATOS -> " + datos.toString());
                break;

            case 2:
                int indice = vista.solicitarIndiceArray(datos.longitud());
                double valor = vista.solicitarValor();
                datos.modificarValor(indice, valor);
                break;

            default: vista.mostrarMensaje("La opción de menú introducida no está contemplada");
        }
    }
```

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

        int opcion;
        do {
            vista.mostrarMenu();
            opcion = vista.pedirOpcion();
            gestionarOpcion(opcion);
        }
        while(true);
    }
}
```

El programa debe estar preparado para, como mínimo, no finalizar su ejecución si se introduce la siguiente información:

- Introducir una letra como opción de menú.
- Introducir un número decimal como opción del menú.
- Introducir una letra como posición del array.
- Introducir un número decimal como posición del array.
- Introducir una letra como valor del array.
- Introducir un valor numérico en una posición del array inexistente.