Ejercicios Java

Excepciones

Ejercicio 1

Escribe en un programa Java en el que se provoque el lanzamiento de la excepción ArithmeticException. No se permite lanzarla directamente usando la sentencia throw.

Ejercicio 2

Crea una clase que defina los siguientes métodos estáticos:

• `isInt(String n)`, que recibe una cadena en el parámetro `n` y retorna el valor booleano `true` si la cadena representa un número de tipo `int`, o el valor booleano `false` en caso contrario.

• `isDouble(String n)`, que recibe una cadena en el parámetro `n` y retorna el valor booleano `true` si la cadena representa un número de tipo `double`, o el valor booleano `false` en caso contrario.

Da una solución basada en captura de excepciones y pon a prueba ambos métodos en el método `main`.

Ejercicio 3

Crea una clase se defina un método estático para resolver una ecuación de segundo grado $ax^2 + bx + c = 0$ según las especificaciones siguientes:

- Declara tres parámetros formales de tipo `double` para recibir el valor de los coeficientes.
- Retorna las soluciónes $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- Lanza la excepción `IllegalArgumentExeption` si el coeficiente de mayor grado es igual a 0;
- Lanza la excepción `ArithmeticException` si no tiene solución (el discriminante es negativo).

Añade un método `main` para poner a prueba el método anterior pidiendo por teclado los coeficientes `a`, `b` y `c` y mostrando en la salida estándar uno de los resultados siguientes:

- Las soluciones de la ecuación.
- El mensaje "*no es una ecuación de segundo grado*" si el coeficiente `a` tiene el valor cero.
- El mensaje "*la ecuación no tiene una solución real*" si la ecuación no tiene solución.

Ejercicio 4

Crea un programa que calcule la hipotenusa de un triángulo rectángulo a partir de la longitud de los catetos. El programa funcionará leyendo los comandos siguientes de la entrada estándar:

COMANDO	ACCIÓN
A *número*	Establece la longitud de uno de los catetos.
B *número*	Establece la longitud del otro cateto.
C	Calcula el valor de la hipotenusa y muestra el resultado.
F	Finaliza el programa.

No se mostrará en la salida estándar ningún mensaje para indicar la entrada de datos salvo un indicador de línea de comando, por ejemplo, el carácter '>' seguido de un espacio en blanco.

Después de cada ejecución sin error del comando `C`, el programa reiniciará la longitud de los catetos con el valor 0.

El programa se ha de desarrollar usando el manejo de excepciones para comprobar la validez de la entrada de datos y evitar que el programa finalice, además de informar al usuario del error cometido. Se entienden como entradas no válidas las siguientes:

- Valores no numéricos donde se esperan valores numéricos.
- Valores numéricos incorrectos.
- Comandos distintos de `A`, `B`, `C` o `F`.

Prueba el funcionamiento del programa con la siguiente secuencia de comandos:

```
A 3↵
B 4↵
C↵
A 3↵
B 0↵
C↵
A 0↵
B 3↵
C↵
A X↵
B X↵
A3↵
C↵
B3↵
C↵
X↵
Q↵
```

Ejercicio 5

Crea una clase copiando el código siguiente y trata de averiguar cuál será la salida antes de ejecutarlo:

```
public class Ejercicio4 {

	public static void main(String[] args) {
		try {
			System.out.println("Antes de metodoA");
			metodoA();
			System.out.println("Despues de metodoA");
		} catch (Exception e) {
			System.out.println("main: " + e);
		} finally {
			System.out.println("main: finally");
		}
	}

	public static void metodoA() {
		try {
			System.out.println("Antes de metodoB");
			metodoB();
			System.out.println("Despues de metodoB");
		} catch (Exception e) {
			System.out.println("metodoA: " + e);
		} finally {
			System.out.println("metodoA: finally");
		}
	}

	public static void metodoB() {
		try {
			System.out.println("Antes de metodoC");
```

```
                        metodoC();
                        System.out.println("Despues de metodoC");
                } catch (ArrayIndexOutOfBoundsException e) {
                        System.out.println("metodoB: " + e);
                } finally {
                        System.out.println("metodoB: finally");
                }
        }

        public static void metodoC() {
                try {
                        System.out.println("Antes de metodoD");
                        metodoD();
                        System.out.println("Despues de metodoD");
                } catch (NumberFormatException e) {
                        System.out.println("metodoC: " + e);
                } finally {
                        System.out.println("metodoC: finally");
                }
        }

        public static void metodoD() {
                try {
                        int a[] = new int[4];
                        a[a.length] = a.length;
                } catch (ClassCastException e) {
                        System.out.println("metodoD: " + e);
                } finally {
                        System.out.println("metodoD: finally");
                }
        }

}
```

Ejecuta el programa anterior y comprueba la predicción que has hecho.

Ejercicio 6

Crea una clase copiando el código siguiente y trata de averiguar antes de ejecutarlo cuál será la cadena que se muestra en la salida:

```
public class Ejercicio5 {
        StringBuilder cadena = new StringBuilder("a");

        void metodoA() {
                try {
                        cadena.append("b");
                        metodoB();
                } catch (Exception e) {
                        cadena.append("c");
                }
        }

        void metodoB() throws Exception {
                try {
                        cadena.append("d");
                        metodoC();
                } catch (Exception e) {
                        throw new Exception();
                } finally {
                        cadena.append("e");
                }
                cadena.append("f");

        }

        void metodoC() throws Exception {
                throw new Exception();
        }
```

```
        String getCadena() {
                return cadena.toString();
        }

        public static void main(String[] args) {
                Ejercicio5 e = new Ejercicio5();
                e.metodoA();
                System.out.println(e.getCadena());
        }

}
```

Ejecuta el programa anterior y comprueba la predicción que has hecho.

Ejercicio 7

Completa el método `main` en la clase siguiente para manejar las excepciones que lanza el método `lanzarExcepcion` en un `try/catch` con la cantidad mínima de bloques *catch*, pero de forma que por cada excepción capturada se muestre por pantalla un mensaje que indique de qué excepción se trata y si es *checked* o *unchecked*:

```
import java.io.FileInputStream;
import java.util.Scanner;

public class Ejercicio2 {

        Ejercicio2 e = new Ejercicio2();

        public static void main(String[] args) {
                Scanner in = new Scanner(System.in);
                String opcion;
                while (!(opcion = in.next()).equalsIgnoreCase("fin"))
                        lanzarExcepcion(Integer.parseInt(opcion));
                in.close();
        }

        static void lanzarExcepcion(int opcion) throws Exception {
                switch (opcion) {
                case 1:
                        System.out.println(1 / 0);
                        break;
                case 2:
                        int [] a = new int[10];
                        a[a.length] = a.length;
                        break;
                case 3:
                        FileInputStream in = new FileInputStream("este fichero no existe");
                        break;
                case 4:
                        Object o = null;
                        System.out.println(o.toString());
                        break;
                case 5:
                        Ejercicio2 e = new Ejercicio2();
                        break;
                default:
                        Class.forName("UnaClase");
                }
        }

}
```