# Listado Excepciones.

1. Rellena los huecos:
    a. Bloque para detectar posibles errores: **try**
    b. Bloque para manejar/capturar los errores previamente detectados: **catch**
    c. Bloque relacionado con la detección/captura de excepciones que siempre se ejecuta a no ser que se halle con la sentencia System.exit(): **finally**
    d. Palabra reservada que indica en la declaración de un método que se puede lanzan una excepción: **throws**
    e. Palabra reservada que indica en el cuerpo de un método el lanzamiento de una excepción: **throw**
2. Todas las excepciones heredan de la clase Throwable. Utiliza la API de Java SE 6 y responde a las siguientes preguntas:
    a. La clase **Exception** es la superclase de todos los errores y excepciones en el lenguaje Java. Sólo los objetos que son instancias de esta clase (o una de sus subclases) son lanzados por la máquina virtual de Java o pueden ser lanzados por la sentencia throw de Java. Del mismo modo, sólo esta clase o una de sus subclases puede ser el tipo del argumento en una cláusula catch.
    b. La clase **Error** es una subclase de **Throwable** que indica graves problemas que una aplicación no debe tratar de capturar. La mayoría de tales errores son condiciones anormales.
    c. La clase **Exception** y sus subclases son subclases de **Throwable** que indican problemas que una aplicación sí debe tratar de capturar.
    d. **RuntimeException** es la superclase de las excepciones que pueden ser lanzadas durante la operación normal de la máquina virtual de Java. No es obligatorio capturarlas aunque es recomendable en algunas ocasiones. Subclases directas son AritmethicException, ClassCastException, IllegalArgumentException, NullPointerException y IndexOutOfBoundsException.
3. Indica verdadero o falso:
    a. El bloque try siempre necesita un bloque finally. **Falso**
    b. El bloque try no siempre necesita un bloque catch, porque la excepción puede propagarse al método que invocó al actual. **Falso**
    c. El bloque try siempre necesita un bloque catch o finally . Nunca puede ir solo. **Verdadero**
    d. El bloque try aislado provoca un error de compilación. **Verdadero**
    e. Todas las excepciones que heredan de Exception son verificadas. **Verdadero**
    f. Todas las excepciones que heredan de RuntimeException son no verificadas o unchecked exceptions. **Verdadero**

g. Todas las excepciones que heredan de Error son no verificadas o unchecked exceptions. **Verdadero**

4. A continuación se muestra un código de bloques try/catch/finally. Tanto ExceptionA como ExceptionB hereden de Exception. Indica el error de la siguiente implementación:

```
try {
        // instrucciones que pueden lanzar excepciones
} catch (Exception e) {
        // instrucciones para manejar la excepción del tipo Exception
} catch (ExceptionA e) {
        // instrucciones para manejar la excepción del tipo ExceptionA
} catch (ExceptionB e) {
        // instrucciones para manejar la excepción del tipo ExceptionB }
finally {
        // instrucciones para liberar recursos (cierre de archivos, cierre //
de conexiones de de bbdd, de redes...)
 }
```

**Al capturar la excepción de tipo Exception, nunca llegará a las demás capturas de excepciones, ya que todas las excepciones tomarán la primera captura porque todas heredan de Exception.**

5. El siguiente código lanza una excepción. Responde a las siguientes preguntas:
   a. ¿Es una excepción checked o unchecked? **Es una Checked Exception.**
   b. Clase a la que pertenece **ArrayIndexOutOfBoundsException**
   c. Modifica el código para capturarla (try/catch). Muestra el resultado de los siguientes métodos:
      i. getMessage(): **2**
      ii. getCause(): **null**
      iii. getLocalizedMessage(): **2**
      iv. toString() : **java.lang.ArrayIndexOutOfBoundsException: 2**
   d. ¿A qué clase pertenecen los métodos anteriores? : **A la clase Throwable**
   e. Asegúrate de que se muestre el mensaje "Después del lanzamiento de la excepción"

```
public class HolaMundoExcepcion {

    public static void main(String[] args) {

        String[] mensaje = new String[2]; mensaje[0] = "Hola ";
        mensaje[1] = "mundo!";

        // este bucle accederá a un índice fuera de rango

        // y lanzará una excepción

        for (int i = 0; i < 3; i++)

        System.out.println(mensaje[i]);

        System.out.println("Después del lanzamiento de la
        excepción");

        }

}
```

6. Explica los errores del siguiente código:

```
try {

    //Este código puede lanzar una excepción

     }System.out.print("Justo debajo del try");

catch(Exception e) {

    //Este código captura una excepción

}
```

**La cláusula catch debe ir junto a la cláusula try, da error de compilación.**

7. El siguiente código lanza una excepción. Captúrala (try/catch) y haz un System.out.println() de los siguientes métodos de la excepción capturada: getMessage(), getCause(), getLocalizedMessage(), toString().Realiza las siguientes actividades:
    a. ¿Cuál es la excepción lanzada y a qué paquete pertenece?: **ArithmeticException y pertenece al paquete lang.**
    b. Envía el código nuevo.

```
/**
 * 7. El siguiente código lanza una excepción. Captúrala (tr
 * System.out.println() de los siguientes métodos de la exce
 * getMessage(), getCause(), getLocalizedMessage(), toString
 * siguientes actividades:
 *
 * @author Antonio Luque Bravo
 * @version 1.0
 *
 */
public class TestCapturaExcepcion {
    public static void main(String[] args) {
        int dividendo = 7;
        int divisor = 0;
        try {
            int cociente = dividendo / divisor;
        } catch (ArithmeticException e) {
            System.err.println(e.getMessage());
            System.err.println(e.getCause());
            System.err.println(e.getLocalizedMessage());
            System.err.println(e.toString());
        }
        System.out.println("Aaaaaaaadios");
    }
}
```

8. Analiza la siguiente clase TestLanzaExcepcion. En el metodo2() se lanza una excepción NullPointerException en la línea nula.toString();
    a. Envía un pantallazo de la ejecución y explica quién maneja la excepción lanzada. Justifica por qué no se ejecuta ningún System.out.println("...");

```
Exception in thread "main" java.lang.NullPointerException
        at ejercicio8.TestLanzaExcepcion.metodo2(TestLanzaExcepcion.java:24)
        at ejercicio8.TestLanzaExcepcion.metodo1(TestLanzaExcepcion.java:18)
        at ejercicio8.TestLanzaExcepcion.main(TestLanzaExcepcion.java:13)
```

**No se ejecuta ningún System.out.println("..."); porque no llega a ejecutarlos ya que se sale antes por el NullPointerException.**

b. Hay que tratar la excepción. Dale distintas soluciones. Impleméntalas y analiza los distintos escenarios:
    i. TestCapturaExcepcion: La excepción se captura directamente en metodo2.

```
    private static void metodo2() {
        String nula = null;
        try {
            nula.toString();
        } catch (NullPointerException e) {
            System.err.println(e.toString());
        }
        System.out.println("Método2: Acabando...");
    }
```

    ii. TestLanzaMiExcepcion: Se crea una excepción MiExcepcion con el mensaje "Mi primera excepcion. " + e.getMessage() en metodo2() y se captura en metodo1()

```
private static void metodo1() {
    try {
        metodo2();
    } catch (MiExcepcion e) {
        System.err.println(e.getMessage());
    }
    System.out.println("Método1: Acabando...");
}

private static void metodo2() throws MiExcepcion {
    String nula = null;
    if (nula == null) {
        throw new MiExcepcion("Mi primera excepcion");
    }
    nula.toString();
    System.out.println("Método2: Acabando...");
}
```

iii. TestLanzaMiExcepcion2: Se crea una excepción MiExcepcion con el mensaje "Mi primera excepcion. " + e.getMessage() en metodo2() y se captura en el main().

```
public static void main(String[] args) {
    try {
        metodo1();
    } catch (MiExcepcion e) {
        System.err.println(e.getMessage());
    }
    System.out.println("main: Acabando...");
}

private static void metodo1() throws MiExcepcion {
    metodo2();
    System.out.println("Método1: Acabando...");
}

private static void metodo2() throws MiExcepcion {
    String nula = null;
    if (nula == null) {
        throw new MiExcepcion("Mi primera excepcion");
    }
    nula.toString();
    System.out.println("Método2: Acabando...");
}
```

iv. TestCapturaEnElMain: Se captura la excepción directamente en el main().

```
public static void main(String[] args) {
    try {
        metodo1();
    } catch (NullPointerException e) {
        System.err.println(e.toString());
    }
    System.out.println("main: Acabando...");
}
```

9. Ejecuta el siguiente código y responde:
    a. A continuación aparecen dos pilas de ejecución de métodos, es decir, dos instantáneas de la ejecución. Indica en qué instante se encuentra cada pila:
        i. El metodo3 se está ejecutando: **Primera pila.**
        ii. El método 3 ha finalizado y devuelve el control al metodo2: **Segunda pila.**

| |
|---|
| metodo3() |
| metodo2() |
| metodo1() |
| main() |

metodo2 invoca a metodo3
metodo1 invoca a metodo2
main invoca a metodo1
main comienza

| |
|---|
| metodo2() |
| metodo1() |
| main() |

metodo2 finalizará
metodo1 finalizará
main finalizará y la JVM acabará

b. Realiza una captura del error. Indica qué tipo de excepción es: verificada o no verificada: **Verificada.**

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ejercicio9.TestPropagaExcepcion.metodo3(TestPropagaExcepcion.java:17)
        at ejercicio9.TestPropagaExcepcion.metodo2(TestPropagaExcepcion.java:13)
        at ejercicio9.TestPropagaExcepcion.metodo1(TestPropagaExcepcion.java:9)
        at ejercicio9.TestPropagaExcepcion.main(TestPropagaExcepcion.java:5)
```

c. Indica dónde se lanza la excepción: **Se lanza cuando se intenta dividir un número entre cero, es decir en el método3.**
d. Indica dónde se captura la excepción: **La excepción no se captura.**
e. Modifica el código para que la excepción sea capturada en metodo3 mostrando el mensaje "División por cero". Utiliza el menú "Código fuente…" de Eclipse. Entrégalo en TestPropagaExcepcion2.

```
private static void metodo3() {
    try {
        int a = 7 / 0;
    } catch (Exception e) {
        System.err.println("División por cero.");
    }
}
```

f. Modifica el código para que la excepción sea capturada de la misma forma en metodo2. Entrégalo en TestPropagaExcepcion3.

```
private static void metodo2() {
    try {
        metodo3();
    } catch (Exception e) {
        System.err.println("División por cero.");
    }
}
```

g. Modifica el código para que la excepción sea capturada de la misma forma en el main. Entrégalo en TestPropagaExcepcion4.

```
public static void main(String[] args) {
    try {
        metodo1();
    } catch (Exception e) {
        System.err.println("División por cero.");
    }
}
```

10. El siguiente código utiliza la clase Scanner para la lectura desde el teclado:
    a. Indica la excepción que se lanza al introducir una letra en vez de un entero: **InputMismatchException**
    b. Captura la excepción para que el usuario sólo pueda introducir un valor válido. Muestra el mensaje de error mediante System.err.println(""); El programa no finalizará hasta que el usuario introduzca un valor válido. Quizás tengas que utilizar la sentencia scanner.nextLine();
11. Crea la clase TecladoScanner para la lectura de datos desde el teclado. Para ello, crea la clase TestScannerConMenu que compruebe todas las lecturas. La clase TecladoScanner dispondrá de:
    a. Una propiedad scanner para la lectura desde teclado (flujo System.in)
    b. Método leerEntero() que devuelva un entero válido introducido por el usuario.
    c. Método leerDecimal() que devuelva un decimal válido introducido por el usuario.
    d. Método leerCaracter() que devuelva un carácter válido introducido por el usuario.
    e. Método leerCadena() que devuelva una cadena introducida por el usuario.
    f. Todos los métodos estarán sobrecargados con un argumento de tipo String para mostrarlo como mensaje previo a la lectura.
    g. ¿Puede utilizarse el patrón de diseño Singleton? Si es así, úsalo.
12. Analiza la clase Teclado suministrada por la profesora para la entrada por teclado del usuario. a. Analiza la gestión de excepciones, indicando i. Nombre de la excepción ii. Cuándo se lanza iii. Si se captura o se lanza de nuevo iv. Solución al error en cuestión b. Modifica el código de la clase Teclado para obligar al usuario a que introduzca un valor válido. c. Pruébalo con la clase TestTeclado mediante un menú.