LISTADO DE EJERCICIOS. EXCEPCIONES.

Con respecto a las excepciones:

a. *Una excepción es un evento que se produce durante la ejecución de un programa, que interrumpe el flujo normal de las instrucciones del programa.*
b. *Cuando se produce un error dentro de un método, el método crea un objeto y se lo pasa al sistema de ejecución. El objeto, denominado objeto excepción, contiene información sobre el error, incluido el tipo y el estado del programa cuando ocurrió el error. La creación de un objeto excepción y la entrega al sistema de ejecución es lo que se denomina **lanzar una excepción**.*
c. *Cuando un método lanza una excepción, el sistema de ejecución intenta encontrar algo para controlarla. La búsqueda se hace en orden inverso a las llamadas en la pila de ejecución. El control de la excepción se gestiona en un bloque denominado catch. Es a lo que se le denomina **capturar la excepción**. De todos los bloques catch, el sistema toma el apropiado, es decir, aquel cuyo tipo del objeto de excepción coincide con el tipo que puede ser manejado por el controlador.*
d. *Un programa puede capturar las excepciones utilizando una combinación de los bloques try, catch y finally.*
e. *El **bloque try** identifica un bloque de código en el que puede producirse una excepción.*
f. *El **bloque catch** identifica un bloque de código, conocido como gestor de excepciones, que puede manejar un tipo particular de excepción.*
g. *El **bloque finally** identifica un bloque de código que se ejecuta con seguridad, y es el sitio adecuado donde cerrar archivos y recuperar los recursos. En definitiva, sitio donde liberar los recursos del código incluido en el bloque try.*
h. *Existen 3 tipos de excepciones: checked exceptions (verificadas), errors y runtime exceptions*
i. *Todas las excepciones son **checked exceptions (verificadas)**, excepto las que heredan de Error y RuntimeException. Es obligatorio capturar (try/catch) o declarar (throws en la signatura) las checked exceptions.*
j. *El segundo tipo de excepciones es el **error**. Estas son las condiciones excepcionales que son externas a la aplicación, y que la aplicación por lo general no puede anticipar o recuperarse. Por ejemplo, supongamos que una aplicación abre con éxito un archivo para la entrada, pero no puede leer el archivo debido a un mal funcionamiento del hardware o del sistema. La lectura sin éxito lanzará java.io.IOError.*
k. *El tercer tipo de excepción es una **runtime exception**. Estas son las condiciones excepcionales que son internas a la aplicación, y que la aplicación por lo general no puede ni anticipar ni recuperar. Estos por lo general indican errores de programación, tales como errores lógicos o el uso indebido de una API. La aplicación puede capturar la excepción, pero probablemente tiene más sentido eliminar el error que causó la excepción*
l. *Errors y runtime exceptions se conocen colectivamente como **unchecked exceptions** o **excepciones no verificadas**.*

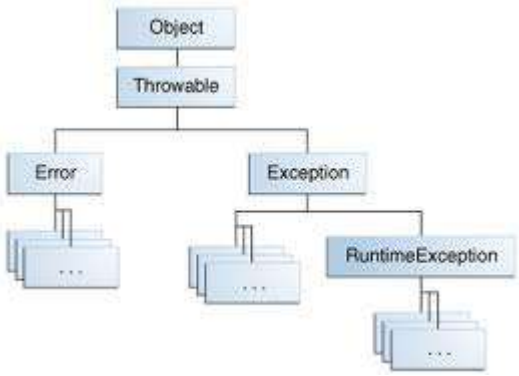
m. Checked exceptions incluyen a todos los subtipos de Exception, excluyendo las clases que heredan de RuntimeException.

n. Las subclases de Error o RuntimeException son unchecked exception, por lo que el compilador no obliga a la captura o declaración de la excepción. Pueden ser manejadas o declaradas, pero al compilador no le importa.

*o. Si se usa el bloque opcional **finally**, siempre será invocado, independientemente de si una excepción en el try correspondiente se lanza o no, e independientemente de si la excepción sea capturada o no.*

p. La única excepción a la regla finally-siempre-se-ejecuta es que finally no se invocará cuando la JVM se apague. Eso podría suceder si en los códigos del try o catch se llama a System.exit ()

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 Throwable 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. Falso
    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...)
}
```

5. El siguiente código lanza una excepción. Responde a las siguientes preguntas:
    a. ¿Es una excepción checked o unchecked?
    b. Clase a la que pertenece
    c. Modifica el código para capturarla (try/catch). Muestra el resultado de los siguientes métodos:
        i. getMessage(),
        ii. getCause(),
        iii. getLocalizedMessage(),
        iv. toString().
    d. ¿A qué clase pertenecen los métodos anteriores?
    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
}
```

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:

```
public class TestCapturaExcepcion {
    public static void main(String[] args) {

        int dividendo = 7;
        int divisor = 0;
        int cociente = dividendo / divisor;
        System.out.println("Aaaaaaaadios");
    }
}
```

   a. ¿Cuál es la excepción lanzada y a qué paquete pertenece? ArithmeticException
   b. Envía el código nuevo. java.lang

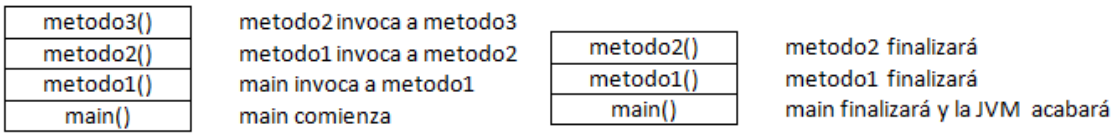
8. Analiza la siguiente clase TestLanzaExcepcion. En el metodo2() se lanza una excepción NullPointerException en la línea nula.toString();

```
public class TestLanzaExcepcion {
    public static void main(String[] args) {
        metodo1();
        System.out.println("main: Acabando...");
    }

    private static void metodo1() {
        metodo2();
        System.out.println("Metodo1: Acabando...");
    }

    private static void metodo2() {
        String nula = null;
        nula.toString();
        System.out.println("Metodo2: Acabando...");
    }

}
```

   a. Envía un pantallazo de la ejecución y explica quién maneja la excepción lanzada. Justifica por qué no se ejecuta ningún System.out.println("...");
   b. Hay que tratar la excepción. Dale distintas soluciones. Impleméntalas y analiza los distintos escenarios:
   c. TestCapturaExcepcion: La excepción se captura directamente en metodo2.

d. TestLanzaMiExcepcion: Se crea una excepción MiExcepcion con el mensaje "Mi primera excepcion. " + e.getMessage() en metodo2() y se captura en metodo1()
e. TestLanzaMiExcepcion2: Se crea una excepción MiExcepcion con el mensaje "Mi primera excepcion. " + e.getMessage() en metodo2() y se captura en el main().
f. TestCapturaEnElMain: Se captura la excepción directamente en el main().

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,
      ii. El método 3 ha finalizado y devuelve el control al metodo2



   b. Realiza una captura del error. Indica qué tipo de excepción es: verificada o no verificada.
   c. Indica dónde se lanza la excepción
   d. Indica dónde se captura la excepción
   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.
   f. Modifica el código para que la excepción sea capturada de la misma forma en metodo2. Entrégalo en TestPropagaExcepcion3.
   g. Modifica el código para que la excepción sea capturada de la misma forma en el main. Entrégalo en TestPropagaExcepcion4.

```
public class TestPropagaExcepcion {
    public static void main(String[] args) {
            metodo1();
    }

    private static void metodo1() {
            metodo2();
    }

    private static void metodo2() {
            metodo3();
    }

    private static void metodo3() {
            int a = 7 / 0;
    }
}
```

10. El siguiente código utiliza la clase Scanner para la lectura desde el teclado:

```
import java.util.Scanner;

public class TestScanner {

    /**
     * Probando la clase Scanner
```

```
     *
     * @param args
     */
    public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Introduce un entero: ");
            System.out.println(scanner.nextInt());
    }

}
```

   a. Indica la excepción que se lanza al introducir una letra en vez de un entero
   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.