

Trabajo Práctico N° 2: Excepciones

1) Dado el método metodoA de la clase A, indique:

- ¿qué sentencias y en qué orden se ejecutan si se produce la excepción MioException?
- ¿qué sentencias y en qué orden se ejecutan si no se produce la excepción MioException?
- ¿qué sentencias y en qué orden se ejecutan si se produce la excepción TuException?

```
class A {  
    void metodoA() {  
        sentencia_a1  
        sentencia_a2  
        try {  
            sentencia_a3  
            sentencia_a4  
        } catch (MioException e)  
        { sentencia_a6 }  
        sentencia_a5  
    }  
}
```

2) Dado el método metodoB de la clase B, indique:

- ¿qué sentencias y en qué orden se ejecutan si se produce la excepción MioException?
- ¿qué sentencias y en qué orden se ejecutan si no se produce la excepción MioException?
- ¿qué sentencias y en qué orden se ejecutan si se produce la excepción TuException?

```
class B {  
    void metodoB() {  
        sentencia_b1  
        try {  
            sentencia_b2  
        } catch (MioException e)  
        { sentencia_b3 }  
        finally  
            sentencia_b4  
    }  
}
```

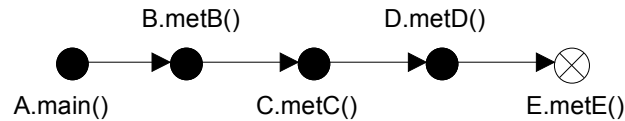
3) Dado el método metodoC de la clase C, indique:

- ¿qué sentencias y en qué orden se ejecutan si se produce la excepción MioException?
- ¿qué sentencias y en qué orden se ejecutan si no se produce la excepción MioException?
- ¿qué sentencias y en qué orden se ejecutan si se produce la excepción TuException?

```
class C {  
    void metodoC() throws TuException{  
        sentencia_c1  
        try {  
            sentencia_c2  
            sentencia_c3  
        } catch (MioException e)  
        { sentencia_c4 }  
        } catch (TuException e)  
        { sentencia_c5  
            throw (e) }  
        finally  
            sentencia_c6  
    }  
}
```

4. En el método metE de la clase E se produce una excepción, al atrapar la excepción se ejecuta el método printStackTrace(), qué salida tiene el método si:

- la excepción es atrapada en D.metD()
- la excepción es atrapada en C.metC()
- la excepción es atrapada en B.metB()
- la excepción es atrapada en A.main()



5. Sistema de Ficheros

Se debe desarrollar una aplicación que va a gestionar los ficheros de una computadora. Como ya se sabe en un sistema de ficheros hay diferentes clases de ficheros. Todos los ficheros se caracterizan por su nombre, extensión, tamaño (en bytes), fecha de la última modificación, permiso de escritura y permiso de lectura. El dato correspondiente a los permisos sólo admite dos valores, es decir si se puede o no leer/escribir. Hay ficheros binarios y ficheros de texto.

El sistema de ficheros sobre el que va a trabajar la aplicación, se puede describir con las siguientes características: identificador del disco duro, capacidad del disco duro y lista de ficheros. Usar la clase Vector de java.util para implementar la lista de ficheros.

Se pide:

- Diseñar un diagrama de clases que represente los objetos que se han descrito en el enunciado.
- Implemente las clases del diagrama
- Suponemos que *InterfazGrafica* es una clase que tiene los siguientes métodos estáticos:

```
public class InterfazGrafica {
...
    public static void mostrar (Fichero f) throws Error1, Error2{
        // Muestra un fichero por la pantalla
        // Lanza la excepción Error1 cuando el fichero que se pretende mostrar es binario
        // Lanza la excepción Error2 cuando no puede seguir mostrando ficheros
        ...
    }
    public static void mensajeError (String s) {
        ...
    }
}
```

Modificar el siguiente código de la clase *SistemaDeFicheros* que muestra por la pantalla todos sus ficheros tratando adecuadamente las excepciones. En caso de que ocurra el *Error1* se indicará con un mensaje y se continuará; cuando ocurra el *Error2* se indicará con otro mensaje y se parará la ejecución.

```
class SistemaDeFicheros {
    Vector Ficheros;

    SistemaDeFicheros() {} // inicializa el vector
    void agregarFichero(Fichero f) {} // agrega un objeto Fichero en el vector
    void eliminarFichero(String id) {} // elimina del vector un objeto Fichero
    void listarFicheros()// lista los ficheros del vector por pantalla
    {
        Por cada fichero f de Ficheros
            InterfazGrafica.mostrar(f)
    }
}
```

Completar la implementación de la aplicación con la codificación de las clases *Fichero*, *Error1* y *Error2* y un programa que pruebe todas las funcionalidades.

6. La impresora es un periférico que puede fallar (cuando se ordena la impresión de un documento) ante las siguientes condiciones: a) fuera de línea b) no conectada c) falta papel d) falta tinta.

La clase *Printer* tiene las siguientes operaciones básicas:

```
cargarTinta (double);
cargarPapel(int);
conectar()
desconectar()
on_line ()
of_line ()
print(Document) // esta operación puede fallar si falta papel o tinta o no esta conectada o esta fuera de línea.
```

La clase Document (cantidad de hojas) representa el documento a imprimir.
La clase PrinterException representa todas las posibles excepciones de la clase Printer.

Tener en cuenta:

- Al enviar a imprimir un documento se decrementa la cantidad de hojas y tinta correspondiente de la impresora.
- Si la cantidad de tinta o papel son mayores que 0 pero son insuficientes para imprimir completamente un documento, se imprimirán las hojas hasta que la cantidad de papel y/o tinta alcance y luego se produce la excepción.
- Si la cantidad de tinta y/o papel es/son igual a 0 al enviar a imprimir un documento, no se imprime nada y solo se produce la excepción.
- Para poder imprimir la impresora debe estar conectada y en línea.
- Una carga de tinta imprime 125 hojas. Es decir una hoja insume 1/125 carga de tinta.
- El usuario puede cargar más de una carga de tinta. (hay cartuchos de distintos tamaños).

Codificar un programa que utilice la impresora (use un menú).

7. Un gestor de base de datos (DBMS) es una aplicación que administra bases de datos. Una base de datos es un conjunto de tablas. Cada base de datos tiene su nombre, como así cada una de las tablas. Las operaciones del DBMS son: crear, eliminar y consultar, abrir, cerrar las bases de datos disponibles. A cada base de datos se le pueden agregar, remover y listar sus tablas (estas operaciones solo se pueden realizar si la base de datos existe y esta abierta).

Crear la clase DataBaseManagementSystem, DataBase y Table teniendo en cuenta que:

Operación	Condición de falla
CreateDB (db)	Falla si db ya existe
OpenDB(db)	Falla si db no existe
CloseDB(db)	Falla si db no existe
CreateTable (db, t)	Falla si db no esta abierta o t ya existe en db.
RemoveDB(db)	Falla si db no esta abierta.
QueryTable(db)	Falla si db no esta abierta
RemoveTable(db, t)	Falla si db no esta abierta o t no existe en db.

Usar como contenedor para las bases de datos que gestiona DataBaseManagementSystem la clase Hashtable (del API de Java). Idem para las tablas que gestiona DataBase.

Crear las clases DBException y TableException para manejar las excepciones.

Pruebe las clases con un menú con las siguientes opciones.

- Crear Base de Datos
- Eliminar Base de Datos
- Abrir Base de Datos
- Cerrar Base de Datos
- Agregar una Tabla a una Base de Datos
- Eliminar una Tabla de una Base de Datos
- Consultar las Tablas de una Base de Datos.