Ejercicios de tratamiento de errores

Índice

1. Captura de excepciones (0.5 puntos)

En el proyecto `lja-excepciones` de las plantillas de la sesión tenemos una aplicación `Ej1.java` que toma un número como parámetro, y como salida muestra el logaritmo de dicho número. Sin embargo, en ningún momento comprueba si se ha proporcionado algún parámetro, ni si ese parámetro es un número. Se pide:

a) Compilar el programa y ejecutadlo de tres formas distintas:

- Sin parámetros

```
java Ej1
```

- Poniendo un parámetro no numérico

```
java Ej1 pepe
```

- Poniendo un parámetro numérico

```
java Ej1 30
```

Anotad las excepciones que se lanzan en cada caso (si se lanzan)

b) Modificar el código de `main` para que capture las excepciones producidas y muestre los errores correspondientes en cada caso:

- Para comprobar si no hay parámetros se capturará una excepción de tipo `ArrayIndexOutOfBoundsException` (para ver si el *array* de `String` que se pasa en el `main` tiene algún elemento).
- Para comprobar si el parámetro es numérico, se capturará una excepción de tipo `NumberFormatException`.

Así, tendremos en el `main` algo como:

```
try
{
        // Tomar parámetro y asignarlo a un double
} catch (ArrayIndexOutOfBoundsException e1) {
        // Codigo a realizar si no hay parametros
} catch (NumberFormatException e2) {
        // Codigo a realizar con parametro no numerico
}
```

Probad de nuevo el programa igual que en el caso anterior comprobando que las excepciones son capturadas y tratadas.

2. Lanzamiento de excepciones (0.5 puntos)

El fichero `Ej2.java` es similar al anterior, aunque ahora no vamos a tratar las excepciones del `main`, sino las del método `logaritmo`: en la función que calcula el logaritmo se comprueba si el valor introducido es menor o igual que 0, ya que para estos

valores la función logaritmo no está definida. Se pide:

a) Buscar entre las excepciones de Java la más adecuada para lanzar en este caso, que indique que a un método se le ha pasado un argumento ilegal. (Pista: Buscar entre las clases derivadas de `Exception`. En este caso la más adecuada se encuentra entre las derivadas de `RuntimeException`).

b) Una vez elegida la excepción adecuada, añadir código (en el método `logaritmo`) para que en el caso de haber introducido un parámetro incorrecto se lance dicha excepción.

```
throw new ... // excepcion elegida
```

Probar el programa para comprobar el efecto que tiene el lanzamiento de la excepción.

c) Al no ser una excepción del tipo *checked* no hará falta que la capturemos ni que declaremos que puede ser lanzada. Vamos a crear nuestro propio tipo de excepción derivada de `Exception` (de tipo *checked*) para ser lanzada en caso de introducir un valor no válido como parámetro. La excepción se llamará `WrongParameterException` y tendrá la siguiente forma:

```
public class WrongParameterException extends Exception
{
        public WrongParameterException(String msg) {
                super(msg);
        }
}
```

Deberemos lanzarla en lugar de la escogida en el punto anterior.

```
throw new WrongParameterException(...);
```

Intentar compilar el programa y observar los errores que aparecen. ¿Por qué ocurre esto? Añadir los elementos necesarios al código para que compile y probarlo.

d) Por el momento controlamos que no se pase un número negativo como entrada. ¿Pero qué ocurre si la entrada no es un número válido? En ese caso se producirá una excepción al convertir el valor de entrada y esa excepción se propagará automáticamente al nivel superior. Ya que tenemos una excepción que indica cuando el parámetro de entrada de nuestra función es incorrecto, sería conveniente que siempre que esto ocurra se lance dicha excepción, independientemente de si ha sido causada por un número negativo o por algo que no es un número, pero siempre conservando la información sobre la causa que produjo el error. Utilizar *nested exceptions* para realizar esto.

Ayuda

Deberemos añadir un nuevo constructor a `WrongParameterException` en el que se proporcione la excepción que causó el error. En la función `logaritmo` capturaremos cualquier excepción que se produzca al convertir la cadena a número, y lanzaremos una excepción `WrongParameterException` que incluya la excepción causante.

3. Excepciones como tipos genéricos en la aplicación filmotecas(0.5 puntos)

Realiza una copia del proyecto `lja-filmoteca` con otro nombre: `lja-filmoteca-exc`. En este nuevo branch realizaremos cambios que **no** mantendremos para el siguiente ejercicio.

Los métodos de nuestros DAO pueden devolver excepciones. Según si el DAO se dedica a escribir en disco, en memoria o en una base de datos, podrá devolver unas excepciones u otras. Busca la clase de excepción que te parezca adecuada para cada uno de los DAO que tiene la aplicación. Por ejemplo, la `IOException` para `FilePeliculaDAO`.

Añade un tipo genérico `E` a `IPeliculaDAO`, que sólo pueda ser una `Exception` o clases derivadas. De esta manera si tenemos que crear un nuevo `FilePeliculaDAO` y asignarlo a una variable que cumple con la interfaz, tendremos que introducir el tipo de excepción:

```
IPeliculaDAO<IOException> dao = new FilePeliculaDAO();
```

Reliza los cambios necesarios a `FilePeliculaDAO` para que, efectivamente, sus métodos lancen esa excepción (sólo con `throws`, sin implementar código dentro de los métodos).

Supongamos que `MemoryPeliculaDAO` no tuviera que devolver ninguna excepción. Declara adecuadamente la clase

```
public class MemoryPeliculaDAO implements IPeliculaDAO<...> {
```

Comprueba que todos los DAO se pueden devolver correctamente desde GestorDAO, tanto los que devuelven excepciones como los que no.

¿Qué limitación tiene esta aproximación? En el siguiente ejercicio añadiremos tratamiento de excepciones siguiendo otra aproximación diferente, sin usar tipos genéricos.

4. Excepciones anidadas en la aplicación filmotecas (1.5 puntos)

En este ejercicio plantearemos el tratamiento de errores de los distintos DAO anidando excepciones en una excepción general para todos los DAO. Así las partes de código que utilicen un DAO de cualquier tipo, sabrán qué excepciones se deben al DAO y además se incluirá excepción concreta que causó el error, por si es necesario saberlo.

Para empezar, en el proyecto `lja-filmoteca`, añadiremos excepciones para tratar los errores en la clase `MemoryPeliculaDAO`. Cuando se produzca un error en esta clase lanzaremos una excepción `DAOException`, de tipo *checked*, que deberemos implementar.

Se pide:

a) La excepción `DAOExcepcion` se creará en el mismo paquete que el resto de las clases del DAO. Utilizaremos las facilidades que nos ofrece Eclipse para generar automáticamente los constructores de dicha clase (tendremos suficiente con tener acceso a los constructores de la super-clase).

b) ¿Qué ocurre si declaramos que los métodos de `MemoryPeliculaDAO` pueden lanzar la excepción creada, pero no lo hacemos en la interfaz `IPeliculaDAO`? ¿Y si en `IPeliculaDAO` si que se declara, pero en alguna de las clases que implementan esta interfaz no se hace (`FilePeliculaDAO`, `JDBCPeliculaDAO`)? ¿Por qué crees que esto es así? Todos los métodos de `IPeliculaDAO` podrán lanzar este tipo de excepción.

c) El método `addPelicula` lanzará la excepción si le pasamos como parámetro `null`, si intentamos añadir una película cuyo título sea `null` o cadena vacía, o si ya existe una película en la lista con el mismo título.

d) El método `delPelicula` lanzará la excepción cuando no exista ninguna película con el identificador proporcionado.

e) En algunos casos es posible que fallen las propias operaciones de añadir, eliminar o listar los elementos de las colecciones. Vamos a utilizar *nested exceptions* para tratar estos posibles errores. Para hacer esto añadiremos bloques `try-catch` alrededor de las operaciones con colecciones, y en caso de que se produjese un error lanzaremos una excepción de tipo `DAOException`. Podemos conseguir que se produzca una excepción de este tipo si utilizamos como colección un tipo `TreeSet`. ¿Por qué se producen errores al utilizar este tipo? ¿Qué habría que hacer para solucionarlo?

f) Comprobar que los errores se tratan correctamente utilizando el programa de pruebas que tenemos (clase `Main`). Habrá que hacer una serie de modificaciones en dicho programa principal para tratar las excepciones. Debemos destacar que el haber tratado las posibles excepciones internas de las colecciones mediante *nested exceptions* ahora nos facilitará el trabajo, ya que sólo deberemos preocuparnos de capturar la excepción de tipo `DAOException`, sin importarnos cómo se haya implementado internamente el DAO.