Grado en Ingeniería Informática

Metodología de la Programación

Sesión 10

Tratamiento de Excepciones

Docentes:

Raúl Marticorena

Félix Nogal

Índice de contenidos

1. Introducción

En este tema se introduce el concepto de **excepciones y su tratamiento** para la construcción de software **más robusto**. Se plantean dos ejercicios, **independientes entres sí,** revisando en primer lugar el **concepto de programación defensiva** y en segundo lugar el **tratamiento de excepciones** concreto que tenemos en Java.

2. Objetivos

- Familiarizarse con el uso y sintaxis de las excepciones en Java.
- Introducir algunos pequeños cambios en el lenguaje, desde la versión Java 7.

3. Evolución del tratamiento de excepciones en Java

A lo largo de la evolución del lenguaje, solo ha habido un **par de modificaciones en Java, relativas a las excepciones**. En concreto, desde su versión 7 se permite **adicionalmente:**

- **Atrapar dos o más tipos de excepciones en un solo `catch`** con el operador tubería: de esta forma se pueden capturar varios tipos, no relacionados entre sí por herencia:
  - Ej: `catch(XException | YException ex)` captura excepciones de subtipo `XException` o `Yexception`, indistintamente, manejando el objeto excepción con la variable `ex`.
- Abrir uno o más recursos a utilizar en la declaración **`try`** de tal forma que **el cierre se hace automáticamente** sin necesidad de hacerlo en el bloque `finally`. La única restricción es que el objeto instanciado debe implementar la interfaz `java.lang.AutoCloseable`. Esto se denomina en Java, un **"try-with-resource**".
  - `Ej: try(Scanner scanner = new Scanner(System.in)) { … } // sin cerrar en finally, se puede utilizar la variable scanner en el cuerpo del try. El cierre se hace automático al finalizar el bloque de instrucciones del catch`

Estos añadidos se consideran **"azúcar sintáctico"** (coloquialmente, una forma más "elegante" o simple de escribir el código), y no es obligatorio su uso, aunque en algún caso pueden simplificar y mejorar la legibilidad del código, por lo que su uso es discrecional.

4. Ejercicios

4.1 Programación defensiva con lanzamiento de excepción

Suponiendo que se programa una clase `tiempo.Alarma` que permite establecer una alarma a una hora y minuto determinado (solo se permite una alarma, en formato de 24 horas), se debe completar el siguiente código como base (se deja en UBUVirtual el correspondiente fichero `Alarma.java` para completar):

```java
package tiempo;

public class Alarma {

	/** Hora. */
	private int hora;

	/** Minuto. */
	private int minuto;

	/**
	 * Inicializa la hora y minuto de una alarma.
	 *
	 * @param hora hora
	 * @param minuto minuto
	 * @throws Exception si se intenta establecer una alarma con valores incorrectos
	 */
	public Alarma(int hora, int minuto) throws Exception {
		// COMPLETAR ENFOQUE DE PROGRAMACIÓN DEFENSIVA POR LOS ALUMNOS
		this.hora = lhora;
		this.minuto = lminuto;
	}

	/**
	 * Consulta la hora.
	 *
	 * @return hora
	 */
	public int consultarHora() {
		return hora;
	}

	/**
	 * Consulta el minuto.
	 *
	 * @return alarma
	 */
	public int consultarMinuto()  {
		return minuto;
	}
}
```

Código 1: Código a completar de la clase `tiempo.Alarma`

Se considera que una hora es correcta si está en el intervalo [0, 23] y que un minuto es correcto si está en el intervalo [0, 59].

En concreto, hay que completar el constructor `Alarma` de tal forma que:

a) Si se introduce una hora incorrecta lanza una excepción `java.lang.Exception` con el texto `"Hora incorrecta: X"` donde en `X` debe aparecer el valor incorrecto que genera la excepción. Ej: si se pasa un valor de hora 24, el texto será `"Hora incorrecta: 24"`.

b) Si se introduce un minuto incorrecto lanza una excepción `java.lang.Exception` con el texto `"Minuto incorrecto: X”` donde en X debe aparecer el valor incorrecto que genera la excepción. Ej: si se pasa un valor de minuto 60, el texto será `"Minuto incorrecto: 60"`.

c) Si se introduce una hora y minuto incorrectos lanza una excepción `java.lang.Exception` con el texto `"Hora y minutos incorrectos: X:Y"` donde en X e Y deben aparecer los valores incorrectos de hora y minuto respectivamente que genera la excepción. Ej: si se pasan valores de hora 24 y minuto 60, el texto será `"Hora y minuto incorrectos: 24:60"`.

- Cuestión: ¿es necesario que el constructor `Alarma` incluya una cláusula **`throws`** `Exception` en su declaración? Eliminar y comprobar si se produce algún error de compilación.

Se recuerda que se proporciona el test automático correspondiente en UBUVirtual parar probar la implementación correcta de la clase.

4.2 Captura y lanzamiento de excepciones.

Se plantea un ejercicio a resolver, utilizando una jerarquía predefinida de excepciones (`AException`, `BException` y `CException`) y una clase auxiliar `Conversor` que utiliza (lanza) dichas excepciones en sus dos métodos (`aMinusculas` y `aMayusculas`). Recordemos que:

- `CException` **`extends`** `BException`
- `BException` **`extends`** `AException`
- `AException` **`extends`** `java.lang.Exception`

Los códigos completos de las excepciones y la clase `Conversor`, están disponibles en la plataforma UBUVirtual.

A partir de dichas clases y jerarquía de herencia, se solicita construir una nueva clase `Principal` con un método `main`, cuyo código parcial se corresponde al mostrado en el Código 2. A partir de dicha clase `Principal`, se plantean una serie de modificaciones sobre el código del `main`. Se proporciona el código de partida en el fichero `Principal.java` (para completar).

```
Scanner scanner = new Scanner(System.in);
String s = scanner.next();
Conversor conversor = new Conversor();
System.out.println(conversor.aMinúsculas(s));
System.out.println(conversor.aMayúsculas(s));
scanner.close();
```

Código 2: Código en el método main de `excepcion.Principal`

1. Utilizar la sintaxis **`try-catch-finally`** para atrapar **cada una de las posibles excepciones**, dando una **salida por pantalla diferente** en cada caso (bien ha saltado una excepción `BException` o `CException`). Comprobar el efecto de cambiar el orden de las cláusulas **`catch`**.
2. Modificar el código anterior para utilizar **solo un bloque `catch`**, dando un mensaje general sea cual sea el tipo de excepción inicial.
3. Modificar la captura anterior, lanzando ahora en el bloque `catch` una nueva excepción de tipo `java.lang.RuntimeException` utilizando además el **mecanismo de encadenamiento de excepciones** guardando la información de la excepción inicialmente atrapada.
4. ¿Por qué no es necesario añadir una cláusula **`throws`** al método `main`? Si en su lugar se lanza una excepción `java.lang.Exception` ¿qué cambios hay que hacer al método `main`?
5. Partiendo de la solución dada al ejercicio 3) modificar el código para realizar un máximo de 3 reintentos (con error) en la lectura de la cadena por teclado. Agotado el máximo número de posibles errores se informa al usuario. Para simplificar, el programa nunca termina, salvo que se agote el número máximo de reintentos.

5. Resumen

En esta sesión de prácticas se utiliza una jerarquía de excepciones, y una clase con métodos que pueden lanzar dichas excepciones, estudiando su uso y posibles variantes en un código cliente.

6. Bibliografía

[Eckel 2007]. Eckel, B. (2007). Piensa en Java. (4ª Edición). Prentice Hall.

7. Recursos

Bibliografía complementaria:

[Oracle, 2020] The Java Tutorials. Lesson: Exceptions (2020). Disponible en http://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

Licencia

Autor: Raúl Marticorena

Área de Lenguajes y Sistemas Informáticos

Departamento de Ingeniería Informática

Escuela Politécnica Superior

UNIVERSIDAD DE BURGOS

2020