PRG (E.T.S. de Ingeniería Informática) - Curso 2019-2020

Práctica 4. Tratamiento de excepciones y ficheros
Primera parte: Una librería de utilidades de lectura desde la entrada estándar

(Una sesión)

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Índice

1. Contexto y objetivos

En el marco académico, esta práctica corresponde al "*Tema 3. Elementos de la POO: herencia y tratamiento de excepciones*" y al "*Tema 4. E/S: ficheros y flujos*". El objetivo principal que se pretende alcanzar con ella es reforzar y poner en práctica los conceptos introducidos en las clases de teoría sobre el tratamiento de excepciones y la gestión de la E/S mediante ficheros y flujos. En concreto:

- Lanzar, propagar y capturar excepciones local y remotamente.
- Leer/escribir desde/en un fichero de texto.
- Tratar las excepciones relacionadas con la E/S.

Para ello, en esta práctica, se va a desarrollar una pequeña aplicación en la que se procesarán los datos que se lean de ficheros de texto, guardando el resultado en otro fichero.

El trabajo se organiza en dos partes. En esta primera parte de la práctica se va a desarrollar una pequeña librería de utilidades para facilitar la lectura de datos numéricos desde la entrada estándar, y en la segunda parte se completarán las clases de la aplicación.

2. La librería `utilPRG`

La librería de utilidades `utilPRG` va a consistir en un paquete que contendrá la clase `CorrectReading`, algunos de cuyos métodos se completarán en esta parte de la práctica.

2.1. Actividad 1: preparar el paquete `pract4`

- Si la práctica se realiza en un equipo propio, se creará un proyecto de nombre `prg`, en la ubicación que se considere oportuna. Si se va a trabajar en el laboratorio mediante conexión remota, se entiende que el proyecto `prg` será el ubicado en `DiscoW`.
- Abrir con *BlueJ* el proyecto `prg` y crear un paquete `pract4` específico para esta práctica. Situados en la ventana del paquete `pract4`, crear dentro de este un subpaquete llamado `utilPRG`.
- Descargar desde `Recursos/Laboratorio/Práctica 4` de *PoliformaT* de `PRG`, los ficheros `CorrectReading.java` y `TestCorrectReading.class` y situarlos en el subpaquete `utilPRG` (recordar que el fichero `.class` no se puede agregar desde BlueJ, sino que hay que copiarlo en la carpeta del sistema correspondiente al subpaquete).

En la figura 1 se muestran abiertas las tres ventanas correspondientes al proyecto `prg` y a los paquetes que se deben haber creado.

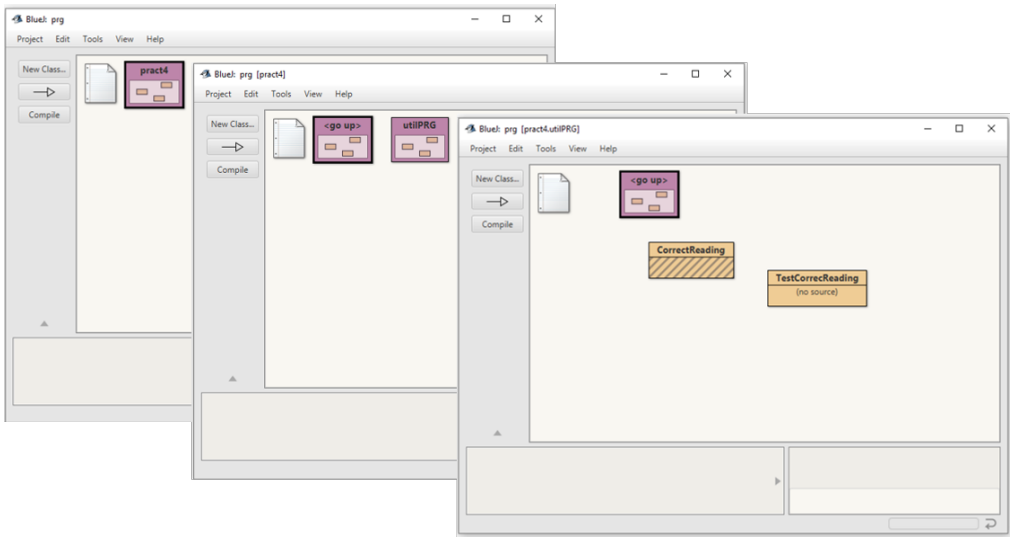

Figura 1: Proyecto `prg`, paquete `prg[pract4]` y subpaquete `prg[pract4.utilPRG]`.

3. Detección de errores en la lectura desde teclado

El método `nextInt()` de `Scanner` se ha utilizado frecuentemente. En la documentación (`http://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html`)

se puede observar que si el valor introducido por el usuario no es un entero, puede lanzar la excepción `InputMismatchException` cuya documentación también se puede consultar.

Esta es una excepción *unchecked* o *no comprobada* (derivada de `RuntimeException`), de manera que su situación en la jerarquía de clases coincide con la que se muestra en la figura 2. Java no obliga a prever el tratamiento las excepciones de esta clase, aunque puede ser útil capturarlas y tratarlas cuando se producen, como en los métodos que se tratan en las siguientes actividades.

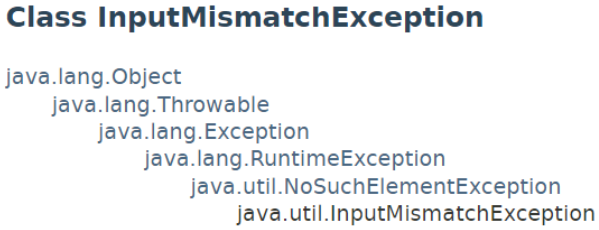

Figura 2: Jerarquía de clases de la excepción `InputMismatchException`.

3.1. Actividad 2: probar y examinar `nextInt(Scanner, String)`

El método `nextInt(Scanner, String)` de la clase `CorrectReading` permite realizar la lectura de un valor de tipo `int` desde teclado usando el método `nextInt()` de la clase `Scanner`, capturando la excepción en caso de que se produzca y mostrando un mensaje de error para indicar al usuario qué acción correctiva es necesaria.

Se puede probar este método en la *zona de código* (*Code Pad*) de *BlueJ*. Para ello, ejecutar las instrucciones siguientes:

```
import java.util.Scanner;
Scanner t = new Scanner(System.in);
int valor = CorrectReading.nextInt(t, "Valor: ");
```

Desde la *ventana del terminal* de *BlueJ*, introducir un valor no entero (por ejemplo, teclear los caracteres de la palabra *hola*) y terminar pulsando *Enter* para enviar los caracteres tecleados al programa, incluyendo dicho salto de línea. Se puede observar el mensaje mostrado indicando que el valor no es válido y que la ejecución del método no acaba hasta que se introduce un valor entero.

Conviene examinar el código del método, y ver que utiliza un bloque `try-catch-finally` para tratar la excepción: el método repite todos los intentos de lectura que haga falta hasta que se consiga leer un entero.

Hay que observar con especial detalle la cláusula `finally` del método `nextInt(Scanner, String)`, usada para "limpiar el buffer"de la entrada. Si no se limpiase el buffer, el método funcionaría mal.

Por ejemplo, supongamos en primer lugar que se introduce desde teclado la secuencia de caracteres 23, y que se pulsa la tecla *Enter*. Entonces `sc.nextInt()` devuelve el `int` 23 y nuestro método termina devolviendo dicho valor, pero en el buffer de entrada ha quedado el carácter `'\n'` del salto de línea; si a continuación, en otro método se le pidiese al usuario introducir una línea para leerla con el método `nextLine()` usando el mismo objeto `Scanner`, se obtendría el `String` vacío `""`, sin esperar a que se teclease nada nuevo.

Supongamos en segundo lugar que se teclea la secuencia de caracteres `hola` y se pulsa la tecla *Enter*. En ese caso, `nextInt()` produce la excepción `InputMismatchException` sin extraer

ningún carácter del buffer, con lo que el token disponible para la siguiente iteración del bucle sigue siendo el mismo.

Como sabemos, una cláusula `finally` se ejecuta tanto si todas las instrucciones del bloque `try` se ejecutan (y ningún bloque `catch`) o si se produce una excepción y uno de los bloques `catch` se ejecuta. Así pues, aquí se ha usado el `finally` para ejecutar la instrucción `sc.nextLine()` en cualquier posible caso de lectura:

- el token leído con `sc.nextInt()` es correcto, y se termina la ejecución del bloque `try`. Entonces, `sc.nextLine()` sirve para extraer el salto de línea correspondiente a la tecla *Enter* pulsada por el usuario.
- el token leído es incorrecto, `sc.nextInt()` fracasa sin extraer el token del buffer, y se lanza la excepción (el bloque `catch` se encarga de escribir el aviso correspondiente). Entonces, `sc.nextLine()` sirve para extraer dicho token del buffer, salto de línea incluído; si no fuese así, los siguientes intentos se encontrarían repetidamente con el mismo token incorrecto: bucle infinito.

La clase contiene el método análogo `nextDouble(Scanner, String)`, para tratar las excepciones `InputMismatchException` que pueda producir `sc.nextDouble()`.

3.2. Actividad 3: completar `nextDoublePositive(Scanner, String)`

En la clase `CorrectReading`, el método `nextDoublePositive(Scanner, String)` debe capturar la excepción `InputMismatchException` si el valor introducido por el usuario no es un `double`, de manera similar al método `nextInt(Scanner, String)`, mostrando un mensaje de error apropiado en lugar de abortar la ejecución. En la documentación (comentarios) del método se encuentran ejemplos de los mensajes que deben mostrarse.

Completar el método añadiendo el bloque `try-catch-finally` necesario. De este modo, se acaba de añadir un controlador de excepciones para detectar una excepción a nivel local, es decir, en el mismo método en donde se produce el fallo.

Para probar este método, ejecutar en la zona de código las instrucciones siguientes, en las que, para poder leer números reales con la notación de `double`, el `Scanner t` creado para las pruebas se configura con `Locale.US`:

```
import java.util.Scanner;
import java.util.Locale;
Scanner t = new Scanner(System.in).useLocale(Locale.US);
double valor = CorrectReading.nextDoublePositive(t, "Valor: ");
```

3.3. Actividad 4: completar `nextInt(Scanner, String, int, int)`

- En la clase `CorrectReading`, el método `nextInt(Scanner, String, int, int)` debe completarse para que capture la excepción `InputMismatchException` si el valor introducido por el usuario no es un `int`, de manera similar al método `nextInt(Scanner, String)`, mostrando un mensaje de error apropiado en lugar de abortar la ejecución.
- El mismo método, además, ha de controlar que el valor introducido está en el rango `[lowerBound,upperBound]`. Hay dos formas de realizar este control:
  - la primera consiste en añadir una condición apropiada en la guarda del bucle,
  - la segunda en lanzar una excepción.

En este método se debe optar por esta última; para ello, aprovechando que se dispone de la instrucción `throw`, se debe añadir una instrucción condicional tal que, si el valor introducido no está en el rango anterior, se lance la excepción `IllegalArgumentException` con un mensaje que indique que el valor leído no está en dicho rango.

A continuación, añadir una cláusula `catch` para capturar localmente dicha excepción, de forma similar a la captura de la excepción `InputMismatchException`, mostrando el mensaje de la excepción mediante el método `getMessage()` (heredado de la clase `Throwable`).

3.4. Actividad 5: test de los métodos. Clase `TestCorrectReading`

Se puede comprobar que los métodos de la librería `CorrectReading` funcionan correctamente con la clase `TestCorrectReading` que se proporciona; sin embargo, este test NO dará información de dónde está el fallo, sino sólo si todo es correcto o si no lo es. Para encontrar el error, se deben realizar las pruebas pertinentes, bien escribiendo y ejecutando un programa de prueba, bien con el *Code Pad* o zona de código.