PRG (E.T.S. d'Enginyeria Informàtica) - Curs 2019-2020

Pràctica 4. Tractament d'excepcions i fitxers

Primera part: Una llibreria d'utilitats de lectura des de l'entrada estàndard

(Una sessió)

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Índex

1 Context i objectius

En el marc acadèmic, aquesta pràctica correspon al "*Tema 3. Elements de la POO: herència i tractament d'excepcions*" i al "*Tema 4. E/S: fitxers i fluxos*". L'objectiu principal que es pretén aconseguir amb ella és reforçar i posar en pràctica els conceptes introduïts en les classes de teoria sobre el tractament d'excepcions i la gestió de la E/S mitjançant fitxers i fluxes. En concret:

- Llançar, propagar i capturar excepcions local i remotament.
- Llegir/escriure des de/en un fitxer de text.
- Tractar les excepcions relacionades amb l'E/S.

Per a això, en aquesta pràctica, es desenvoluparà una xicoteta aplicació en la qual es processaran les dades que es lligen de fitxers de text, guardant el resultat en un altre fitxer.

El treball s'organitza en dues parts. En aquesta primera part de la pràctica es desenvoluparà una xicoteta llibreria d'utilitats per a facilitar la lectura de dades numèriques des de l'entrada estàndard, i en la segona part es completaran les classes de l'aplicació.

2 La llibreria `utilPRG`

La llibreria d'utilitats `utilPRG` consistirà en un paquet que contindrà la classe `CorrectReading`, alguns dels mètodes de la qual es completaran en aquesta part de la pràctica.

2.1 Activitat 1: preparar el paquet `pract4`

- Si la pràctica es realitza en un equip propi, es crearà un projecte de nom `prg`, en la ubicació que es considere oportuna. Si es treballa en el laboratori mitjançant connexió remota, s'entén que el projecte `prg` serà el situat en `DiscoW`.
- Obrir amb *BlueJ* el projecte `prg` i crear un paquet `pract4` específic per a aquesta pràctica. Situats en la finestra del paquet `pract4`, crear dins d'aquest un subpaquet anomenat `utilPRG`.
- Descarregar des de `Recursos/Laboratorio/Práctica 4` de *PoliformaT* de `PRG`, els fitxers `CorrectReading.java` i `TestCorrectReading.class` i situar-los en el subpaquet `utilPRG` (recordar que el fitxer `.class` no es pot agregar des de BlueJ, sinó que cal copiar-ho en la carpeta del sistema corresponent al subpaquet).

En la figura 1 es mostren obertes les tres finestres corresponents al projecte `prg` i als paquets que s'han d'haver creat.

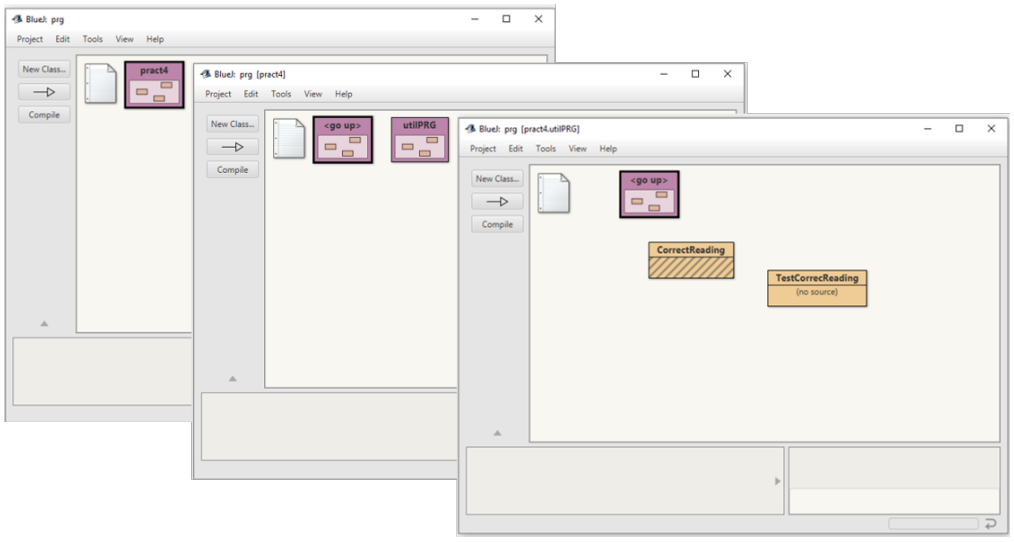

Figura 1: Projecte `prg`, paquet `prg[pract4]` i subpaquet `prg[pract4.utilPRG]`.

3 Detecció d'errors en la lectura des de teclat

El mètode `nextInt()` de `Scanner` s'ha utilitzat sovint. En la documentació (`http://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html`)

es pot observar que si el valor introduït per l'usuari no és un enter, pot llançar l'excepció `InputMismatchException` la documentació del qual també es pot consultar.

Aquesta és una excepció *unchecked* o *no comprovada* (derivada de `RuntimeException`), de manera que la seua situació en la jerarquia de classes coincideix amb la que es mostra en la figura 2. Java no obliga a preveure el tractament les excepcions d'aquesta classe, encara que pot ser útil capturar-les i tractar-les quan es produeixen, com en els mètodes que es tracten en les següents activitats.

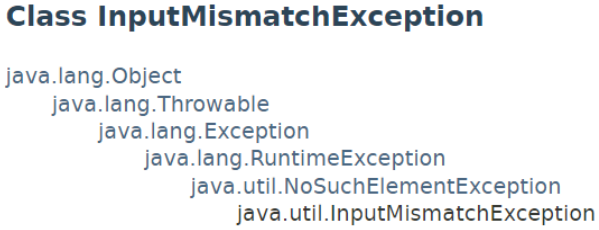

Figura 2: Jerarquia de classes de l'excepció `InputMismatchException`.

3.1 Activitat 2: provar i examinar `nextInt(Scanner, String)`

El mètode `nextInt(Scanner, String)` de la classe `CorrectReading` permet realitzar la lectura d'un valor de tipus `int` des de teclat usant el mètode `nextInt()` de la classe `Scanner`, capturant l'excepció en cas que es produïsca i mostrant un missatge d'error per a indicar a l'usuari quina acció correctiva és necessària.

Es pot provar aquest mètode en *la zona de codi* (*Code Pad*) de *BlueJ*. Per a això, executar les instruccions següents:

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

Des de la *finestra del terminal* de *BlueJ*, introduir un valor no enter (per exemple, teclejar els caràcters de la paraula *hola*) i acabar prement *Enter* per a enviar els caràcters teclejats al programa, incloent aquest salt de línia. Es pot observar el missatge mostrat indicant que el valor no és vàlid i que l'execució del mètode no acaba fins que s'introdueix un valor enter.

Convé examinar el codi del mètode, i veure que utilitza un bloc `try-catch-finally` per a tractar l'excepció: el mètode repeteix tots els intents de lectura que faça falta fins que s'aconseguisca llegir un enter.

Cal observar amb especial detall la clàusula `finally` del mètode `nextInt(Scanner, String)`, usada per a "netejar el buffer"de l'entrada. Si no es netejara el buffer, el mètode funcionaria incorrectament.

Per exemple, suposem en primer lloc que s'introdueix des de teclat la seqüència de caràcters `23`, i que es prem la tecla *Enter*. Llavors `sc.nextInt()` retorna el `int` 23 i el nostre mètode acaba retornant aquest valor, però en el buffer d'entrada ha quedat el caràcter `'\n'` del salt de línia; si a continuació, en un altre mètode se li demanara a l'usuari que introduira una línia per a llegir-la amb el mètode `nextLine()` usant el mateix objecte `Scanner`, s'obtindria el `String` buit `""`, sense esperar que es teclejara res nou.

Suposem en segon lloc que es tecleja la seqüència de caràcters `hola` i es prem la tecla *Enter*. En aqueix cas, `nextInt()` produeix l'excepció `InputMismatchException` sense extraure cap caràcter del buffer, amb el que el token disponible per a la següent iteració del bucle continua sent el mateix.

Com sabem, una clàusula `finally` s'executa tant si totes les instruccions del bloc `try` s'executen (i cap bloc `catch`) o si es produeix una excepció i un dels blocs `catch` s'executa. Així doncs, ací s'ha usat el `finally` per a executar la instrucció `sc.nextLine()` en qualsevol possible cas de lectura:

- el token llegit amb `sc.nextInt()` és correcte, i s'acaba l'execució del bloc `try`. Llavors, `sc.nextLine()` serveix per a extraure el salt de línia corresponent a la tecla *Enter* premuda per l'usuari.

- el token llegit és incorrecte, `sc.nextInt()` fracassa sense extraure el token del buffer, i es llança l'excepció (el bloc `catch` s'encarrega d'escriure l'avís corresponent). Llavors, `sc.nextLine()` serveix per a extraure aquest token del buffer, salt de línia inclòs; si no fóra així, els següents intents es trobarien repetidament amb el mateix token incorrecte: bucle infinit.

La classe conté el mètode anàleg `nextDouble(Scanner, String)`, per a tractar les excepcions `InputMismatchException` que puga produir `sc.nextDouble()`.

3.2 Activitat 3: completar `nextDoublePositive(Scanner, String)`

En la classe `CorrectReading`, el mètode `nextDoublePositive(Scanner, String)` ha de capturar l'excepció `InputMismatchException` si el valor introduït per l'usuari no és un `double`, de manera similar al mètode `nextInt(Scanner, String)`, mostrant un missatge d'error apropiat en lloc d'avortar l'execució. En la documentació (comentaris) del mètode es troben exemples dels missatges que han de mostrar-se. Completar el mètode afegint el bloc `try-catch-finally` necessari. Amb açò, s'acaba d'afegir un controlador d'excepcions per a detectar una excepció a nivell local, és a dir, en el mateix mètode on es produeix la fallada.

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

- En la classe `CorrectReading`, el mètode `nextInt(Scanner, String, int, int)` ha de capturar l'excepció `InputMismatchException` si el valor introduït per l'usuari no és un `int`, de manera similar al mètode `nextInt(Scanner, String)`, mostrant un missatge d'error apropiat en lloc d'avortar l'execució.
- Aquest mètode, a més, ha de controlar que el valor introduït està en el rang `[lowerBound,upperBound]`. Hi ha dues formes de realitzar aquest control:

  – la primera consisteix en afegir la condició apropiada en la guarda del bucle,
  – la segona en llançar una excepció.

  En aquest mètode s'ha d'optar per aquesta última; per a això, aprofitant que es disposa de la instrucció `throw`, s'ha d'afegir una instrucció condicional tal que, si el valor introduït no està en el rang anterior, llance l'excepció `IllegalArgumentException` amb un missatge que indique que el valor llegit no està en aquest rang. A continuació, afegir una clàusula `catch` per a capturar localment aquesta excepció, de forma similar a la captura de l'excepció `InputMismatchException`, mostrant el missatge de l'excepció mitjançant el mètode `getMessage()` (heretat de la classe `Throwable`).

3.4 Activitat 5: test dels mètodes. Classe `TestCorrectReading`

Es pot comprovar que els mètodes de la llibreria `CorrectReading` funcionen correctament amb la classe `TestCorrectReading` que es proporciona; no obstant això, aquest test NO donarà informació d'on està la fallada, sinó només si tot és correcte o si no ho és. Per a trobar l'error, s'han de realitzar les proves pertinents, bé escrivint i executant un programa de prova, bé amb el *CodePad*.