Worksheet Five on Exception Handling

Principles of Programming II

Part-time cohort 2018

Learning goals

Before the next session, you should have achieved the following learning goals:

- Understand what exceptions are and how they are used.
- Create your own exceptions, both checked and runtime.
- Catch exceptions by means of try/catch.

1 Code flow

Look at the following code and write the code flow (use the line numbers to indicate which lines are executed in which order) in the following situations:

- `userInput` is 0.
- `userInput` is 2.
- `userInput` is 4.
- `userInput` is 6.

```
01   public void launch(int userInput) {
02       List<Integer> intList = new ArrayList<Integer>();
03       intList.add(1);
04       intList.add(2);
05       intList.add(3);
06       intList.add(4);
07       intList.add(5);
08       intList.add(6);
09       try {
10          intList.remove(0);
11          System.out.println(intList.get(userInput));
12          intList.remove(0);
13          System.out.println(intList.get(userInput));
14          intList.remove(0);
15          System.out.println(intList.get(userInput));
16          intList.remove(0);
17          System.out.println(intList.get(userInput));
18          intList.remove(0);
19          System.out.println(intList.get(userInput));
20          intList.remove(0);
21          System.out.println(intList.get(userInput));
22          intList.remove(0);
23          System.out.println(intList.get(userInput));
24       } catch (IndexOutOfBoundsException ex) {
```

```
25          ex.printStackTrace();
26       }
27    }
```

Incorporate this code into a simple class to verify your answers.

2 Exception

Read the following code and check whether there is anything wrong with it. Then write some similar code and check your answer.

```
// Some code here
try {
  // more code here
  list.add(newElement);
  // more code here
} catch (Exception ex) {
  ex.printStackTrace();
} catch (NullPointerException ex) {
  ex.printStackTrace();
}
```

3 Error handling on user input

a)

Write a program that reads 10 numbers from the user and then prints the mean average. If the user inputs something that is not a number, the program should complain and ask for a number again until 10 numbers have been introduced.

b)

Modify the program so that it first asks how many numbers the user wants to enter, and then asks for the numbers. The computer should complain if the user enters something that is not a number in both cases. Use methods to prevent code repetitions.

4 More patients

Write a class that asks for data (name and year of birth) about new patients in a hospital and keeps them in a list of `Patient`. The constructor of `Patient` must throw an `IllegalArgumentException` if the age of the patient is negative or greater than 130.

5 Prime divisors

Create a class `PrimeDivisorList`. Integers (as in class `Integer`) can be added to / removed from the list. If a `null` number is passed to the `add(Integer)` method, a `NullPointerException` must be thrown. If a non-prime number is added, an `IllegalArgumentException` must be thrown.

Override the method `toString()` so that it returns something like:

```
[ 2 * 3^2 * 7 = 126 ]
```

for a list containing one 2, two 3, and one 7.

Use the TDD methodology to create the class (interface, tests, implementation). You can base your class on classes and interfaces from the Java Collections Library.

6 Your first exceptions

Create two exceptions, one checked exception and one runtime exception. Then create a simple class that will throw each of them in two different situations, according to user input:

1. inside a `try` block.
2. outside a `try` block.

Note: Two exceptions times two situations means four different inputs from users. Create the new exceptions with four different messages (using the appropriate constructor), e.g. "I am a checked exception and I have been thrown out of a try block".

Assuming you do all of the above inside the `launch()` method of your class, did you have to make any changes to the method's declaration?

7 Hierarchies of classes, hierarchies of exceptions

(Borrowed from Bruce Eckel.) Create a three-level hierarchy of exceptions (i.e. an exception extends some exceptions that extends some exception). Now create a base-class A with a method that throws an exception at the base of your hierarchy. Inherit B from A and override the method so it throws an exception at level two of your hierarchy. Repeat by inheriting class C from B. In the `launch()` method of another class, create a C and upcast it to A, then call the method.

8 Exaggerating list (*)

Create a class `ExaggeratingList` that implements `java.util.List` for type integer. Implement all methods in the interface, with the following peculiarities:

- When a small element is added to the list, it is transformed into a bigger element. What "small" means is decided at construction time (i.e. as a parameter of the constructor of `ExaggeratingList`).
- The output of `size()` is actually twice the number of elements in the list.

Pay special attention to the exceptions that must be thrown at each method (according to the interface documentation), especially in the following two cases:

- No `null` elements can be added. Bear this in mind and implement the appropriate exceptions to be thrown (according to the interface documentation).
- The list does not support methods `removeAll()` and `retainAll()`.

9 Bonding (**)

As a curiosity, look at the following code. Can you understand how it works and predict its output? What would happen if method `gotBored()` returned always `false`? Test yours predictions against reality by compiling and executing the code.

Note: this code is **not** an example of well-written code and is provided only as a mental exercise.

```
public class Player {

   /*
    * This is the only method worth looking. The rest is boilerplate.
    */
   public void aim(Ball theBall) {
      try {
         throw theBall;
      } catch (Ball incomingBall) {
```

```
            if (gotBored()) {
               System.out.println(getName() + ": I got bored. Let's play something else. ");
            } else {
               System.out.println(getName() + ": good throw, " + target.getName());
               target.aim(incomingBall);
            }
         }
      }

      /*
       * All code below is basically boilerplate and it is
       * shown only for completeness so the code is compilable.
       */

      private Player target;
      private String name;

      public Player(String name, Player target) {
         this.name = name;
         this.setTarget(target);
      }

      public void setTarget(Player target) {
         this.target = target;
      }

      public String getName() {
         return name;
      }

      public boolean gotBored() {
         return (Math.random() < 0.001);
      }

      public static void main(String...args) {
         Player parent = new Player("Dad", null);
         Player child = new Player("Kid", parent);
         parent.setTarget(child);
         parent.aim(new Ball());
      }
   }

   class Ball extends Throwable {}
```

Acknowledgement: thanks to Randall Munroe for the idea.