Exercise 10

Exceptions

By the end of this exercise you will be able to

- Understand the idea of *exceptions* as a way for a program to respond to "abnormal" events.
- Write a program that can accept input from the user, properly handling any exceptions that are thrown.

Introduction

In order to write a program that can receive input from the user you will need to understand how exceptions work, because the standard Java Input/Output libraries use exceptions for handling abnormal circumstances like reading past the end of a file or the user's keyboard blowing up!

The `readline` method of the `BufferedReader` class is a useful way to receive user input and question 1 shows you how to go about calling this method. Question 2 gives you practice in calling a method that can throw various kinds of exceptions, and how to use `try-catch` statements to keep track of how many of each kind of exception is thrown.

Questions

1. **Standard Input and Exceptions**

   (a) Fetch a copy of `Student.java`. Compile and run it, and check that you understand its code.

   (b) Add a method `readAge` which reads an `int` value from the standard input and assigns the value to the `age` property. Add a method `printAge` which prints the student's age on the standard output. Invoke both methods on the `me` object in the `main` method.

   Note that you will need to convert the `String` object returned by `stdIn.readLine()` to an `int` value. Look up the `Integer` class in `java.lang` for information on how to do this.

   (c) What happens when you type a series of alphabetical characters instead of a number for the age?

   (d) Add a `try-catch` statement to the method `readAge` so that the exceptions `NumberFormatException` and `IOException` cannot be thrown by the method. If an `IOException` exception occurs, the program should terminate with a suitable error message. If an `NumberFormatException` occurs, the age should be set to -1 and an error message should be printed out.

   HINT: The `readLine()` method call throws the I/O exceptions whenever the user's keyboard blows up or for some equally unlikely reason. The method from the `Integer` class throws Number Format exceptions. The `try` block should go around these two method calls.

(e) Add further statements to the `readAge` method so that it repeatedly loops until the user enters an age in the range 0 to 150. If `NumberFormatException` objects are thrown, then the user should be prompted to try again.

2. **Working with exceptions.** A very sloppy programmer has created a random number generator class called `BadRandom` that provides a method `randVal`. The method signature for `randVal` is as follows:

```
public static int randVal();
```

This method periodically throws the following exceptions:

- `ArithmeticException`
- `NullPointerException`
- `ArrayIndexOutOfBoundsException`
- `ClassCastException`
- `NegativeArraySizeException`.

(a) Fetch a copy of `BadRandom.class`

(b) You are to write a class called `BRTest` that collects data on calls to `BadRandom.randVal()` and the values it returns. Items that you should add to the `BRTest` class are detailed below.

(c) Add `private int` properties `calls`, `successfulCalls` and `totalReturned` which store the total number of times `BadRandom.randVal()` has been called, the number of times `BadRandom.randVal()` has successfully returned a random value, and the total of the successfully returned values.

(d) Add an array of `int` values called `excepCounts` (a `private` property). The `excepCounts` array will store the number of occurrences of each of the exceptions listed above.

(e) Add a method `public void callIt()` which is used to keep track of calls to the method `BadRandom.randVal()`, as follows.

- Each invocation of `callIt` increments the value in `calls`.
- `callIt` then calls `BadRandom.randVal()`.
- If the call successfully returns a random value, then `successfulCalls` is incremented and the value returned is added to `totalReturned`.
- If the call throws an exception then the exception is caught and the exception message is printed on the standard output. The array element associated with this type of exception in `excepCounts` is incremented.

(f) Add a method `public void resetCounts()` which resets all of the `BRTest` count values to zero: `calls`, `successfulCalls`, `totalReturned` and each of the exception counts in `excepCounts`.

(g) Add a method `public void nRandInts(int n)` which repeatedly calls `callIt` until `n` integer values have been successfully returned by `BadRandom.randVal()` (where `n` is an integer value greater than zero).

(h) Add a method `public void writeData()` which writes to the standard output the following data:

- The number of calls to `BadRandom.randVal()`.
- The number of successful calls to `BadRandom.randVal()`.
- The total of the values returned by the successful calls.

- The percentage of calls to `BadRandom.randVal()` which threw each type of exception identified above.
- The percentage of calls to `BadRandom.randVal()` which successfully returned an integer.

(i) Add statements to the `main` method of `BRTest` to test the methods you have written. Note that the following main program:

```
public static void main(String[] args) {
  BRTest me = new BRTest();
  me.resetCounts();
  me.nRandInts(30);
  me.writeData();
}
```

should produce output similar to this:

```
ArithmeticException: Division by zero!
NegativeArraySizeException: Squaaawk!  Screetch. Burp.
ArrayIndexOutOfBoundsException: Accessing 11th element of  10 element array!
NullPointerException: Uninitialised object reference.
ClassCastException: Casting apples into oranges.
ClassCastException: Casting apples into oranges.
ArithmeticException: Division by zero!
NegativeArraySizeException: Squaaawk!  Screetch. Burp.
NullPointerException: Uninitialised object reference.
ClassCastException: Casting apples into oranges.
ClassCastException: Casting apples into oranges.
ClassCastException: Casting apples into oranges.
ArithmeticException: Division by zero!


========================================
Number of calls:  43
Successful calls:  30
Total returned:  48
Percentage Arithmetic Exceptions:  6.9767447
Percentage Null Pointer Exceptions:  4.6511626
Percentage Array Index Exceptions:  2.3255813
Percentage Class Cast Exceptions:  11.627908
Percentage Negative Array Exceptions:  4.6511626
Percentage of successful calls:  69.76744
========================================
```