Lab 6

Exceptions and Exception Handling

Due: during your lab session on or before April 12nd

Description:

Worth **4%** **of your total mark**

In this lab you will:

1. Explore exceptions and exception handling
2. Write code to throw and catch exceptions
3. Understand the role polymorphism plays in determining the order in which exceptions are caught

You will have completed this lab when you:

- ☐ Can successfully create and throw the exceptions indicated.
- ☐ Can answer questions such as:
  - What is the advantage of using exceptions over an if…else… structure to capture errors?
  - Where is the getMessage() method actually declared, and how do you set the string to be output by this message in your code for the BadAccountInputException constructor?
  - If you inserted 'throw new Exception("General exception triggered; source unknown")' in try block of the OurDate() class's setOurDate(String) method, where would this exception be caught?
  - Why must superclass exceptions be caught *after* subclass exceptions?
  - What is the purpose of a check digit, and how would you use it to catch an incorrect value entered into code?

Lab 6

Exceptions and Exception Handling

I. Load the code for this lab, available on Brightspace, into Eclipse.

a. Download the CST8284_Lab06.zip file from Brightspace and import the file into Eclipse package explorer, using the same instructions as you did for Lab4.

b. Inspect the contents of the files just imported. Note that:

i. `AccountLauncher` contains the `public static void main()` method that launches your program. This class is complete and should not need modification.

ii. The `Account` class contains 3 TODOs for you to complete. The description of your task, and the output to be printed, will be generated by the Lab 6 TODO Generator.jar file provided. Double click on the .jar to launch it (the JRE must be correctly installed), enter your student number in the box at the top, click the button, and follow the instructions for the three TODOs indicated.

iii. The `OurDate` class has been modified to accept a string in an OurDate constructor. The code is provided for you, but you must supply the catch statements indicated in TODO #4.

iv. In the same package as the above classes, load a new class, `BadAccountInputException`, which must extend from `java.lang.RuntimeException`. Your class must contain two constructors, a no-arg constructor that chains to a one-arg constructor that loads a single string. This is the string that is output using ex.getMessage() in AccountLauncher. This constructor uses super() to load its String message. But you'll need to be able to explain how the BadAccountInputException subclass does this. And for this, expect to do some research in the Oracle documentation. In particular, you should be able to answer two questions: (1) where is the getMessage() method located? And (2) how does your error message String get loaded into it?

Note: You'll need to add the line

```
private static final long
        serialVersionUID = 1L;
```

to this class, to prevent serialization warnings and errors

II. Implement the code indicated in the TODOs

a. The first two TODOs, along with TODO #4 are relatively self-explanatory. In each case, throw a new BadAccountInputException() and pass the String message indicated so that ex.getMessage() outputs the error message passed to the new Exception. Be sure you understand how exception handling works, and be prepared to insert breakpoints in your code if unsure of the program's execution.

Note that each account number consists of a three digit branch code, followed by a hyphen ("-"), followed by a six-digit customer account number.

The following String methods may be of some use to you in testing the first name, last name, and account number for correctness. However, you'll need to research these at

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html to find out how they work:

trim()	substring()
matches()	contains()
split()	valueOf()

Additionally, you may find Integer.parseInt() useful to convert String values to their equivalent int values.

Be sure to test your code with values designed to throw the particular exception your code is designed to detect.

b. TODO #3 requires you to write code that calculates a checksum value, which should equal the last digit in the account number string. If the checksum of the account number is correct, isCheckDigitCorrect() returns true, otherwise false. Your error message should be: "Checksum failed"

For example, the checksum algorithm for the 10-digit ISBN number on the back of your text book is 0-13-474335-0. The algorithm for this says: ignoring hyphens, multiply the first digit by 1, add it to the second digit multiplied by 2, etc. Take the sum modulus 11. The last value, '0' in this case, should equal the result. Mathematically, for **013474335** the calculation is

```
(1)0 + (2)1 + (3)3 + (4)4 + (5)7
+ (6)4 + (7)3 + (8)3 + (9)5 = 176
```

`(176 % 11) == 0`, i.e. equal to the last digit, hence the checksum digit is correct. Recall that % is the modulus operator; it returns the remainder of a calculation.

Note that your code cannot perform the calculation in the fashion show above, taking each digit in the sum and multiplying it by 1, 2, 3, etc. *So you wouldn't' get any marks for code like this:*

```
sum =
   1*Integer.ParseInt(accountNum.charAt(0)) +
   2*Integer.ParseInt(accountNum.charAt(1)) +
   3*Integer.ParseInt(accountNum.charAt(2)) +
   4*Integer.ParseInt(accountNum.charAt(3)) +
      etc.
```

At this point, you should be well beyond this kind of code. You must be able to write your code in a for() loop, in under a half-dozen statements inside the loop (and probably considerably less). Your code should be scalable for any length of account number.

III. Demonstrate your program, and your understanding of the code, to the lab professor

Run the .jar file before demoing your code, and then clearly indicate that your program catches the errors requested

Show your code corresponding to the four TODOs to the lab professor, along with the BadAccountInputException constructors, and be prepared to answer questions.

Marking

Requirement	*Mark*
BadAccountInputException correct	**2**
TODO #1 Account number error caught correctly	**3**
TODO #2 Name error caught correctly	**3**
TODO #3 Checksum implemented correctly	**5**
TODO #4 implemented correctly and in correct order	**3**
Can answer questions about your code indicating you understand its operation	**4**
Total:	**20**