Lab 7: Exceptions

1. Catching `NumberFormatException`

Consider the following code which computes the sum of integers input by the user.

```
// ***********************************************************
// ParseInts.java
// Reads a line of text and prints the integers in the line.
//***********************************************************
import java.util.Scanner;
public class ParseInts
{  public static void main(String[] args)
   {     int val, sum=0;
         Scanner scan = new Scanner(System.in);
         String line;
         System.out.println("Enter a line of text");
         Scanner scanLine = new Scanner(scan.nextLine());
         while (scanLine.hasNext()){
               val = Integer.parseInt(scanLine.next());
               sum += val;}
         System.out.println("The sum of the integers on this line is " +
         sum);
   }
}
```

- Run the code inputting integers only : 10 20 30 40
- Run the code inputting integers and strings : 1 “one” 2 “two” 3
  - What kind of exception you get?
  - Which line of code cause the exception?
- Add a try-catch block to handle the exception
  - Wrap the loop with the catch bloc
  - Make the catch bloc inside the loop
  - Test both the above scenarios with different outputs and tell what is the difference

2. Throwing `IllegalArgumentException`

Reconsider your code of computing the factorial of a number from the first lab. Change the code so that if the `fact` function throws `IllegalArgumentException` if the `fact` parameter is negative. Make the message of the exception as informative as possible. The main method, when calling `fact,` should catch the exception and handles by asking for a valid number.

3. Call built-in methods of Exception class

The Exception class provides a set of methods which manipulate the exception object. Complete the following code by invoking all the methods and understand what each method does.

```
public class Main {
    public static void main(String[] args) {
        try {
            throw new Exception("My Exception");
        } catch (Exception e) {
            System.err.println("Caught Exception");
            System.err.println("getMessage():" + e.getMessage());
            System.err.println("toString():" + e);
           …
        }
    }
}
```

4. Custom and multiple exceptions

ˆ Create a checked exception called AnimalException

ˆ Create a checked exception called MammalException which extends the class AnimalException ˆ
Create a checked exception called TigerException which extends the class MammalException
Each exception should have two constructors:

- o One without parameters
- o and one with a string parameters that holds the error message.
  - – For AnimalException print: An animal exception occurred instead of Animal
  - – For MammalException print: A mammal exception occurred instead of Mammal
  - – For TigerException print: A tiger exception occurred instead of Tiger

Given the following code:

```
public class Calculations {
  public static void animal() throws AnimalException {
       Random random = new Random();
       int val = random.nextInt(4); // it can: not throw an exception
       switch(val) {
             case 0: System.out.println("Animal");
                     throw new AnimalException("Animal");
             case 1: System.out.println("Mammal");
                     throw new MammalException("Mammal");
             case 2: System.out.println("Tiger");
                     throw new TigerException("Tiger");
       }
  }
  public static void main(String[] args) {
       animal();
  }
}
```

Surround the call to the method animal with a try and catch block. For each exception a different message is printed according to the exception raised. Use getMessage() to get the message of each exception. ˆ
At the end, ensure that whether any exception occurred, or not, you should print the message: Finished!

5. **Inherited exceptions**

- Create a checked exception ParentException
- Create a second exception ChildException which extends the ParentException
- Create two methods for throwing each exception
- In the main method, experiment with the following serrations

Call the first method:

  - Catch the child Exception
  - Catch the parent Exception
  - Catch the two exceptions

Call the second method

  - Catch the child Exception
  - Catch the parent Exception
  - Catch the two exceptions

In which case you get a compilation error? Explain.