Bucknell UNIVERSITY

CSCI 204 – Introduction to Computer Science II

Lab 6 – Recursion and Exceptions

1 Objectives

In this lab, you will learn the following:

- Recursion
- Implementing a directory listing using recursion
- Implementing your own exception and throwing it

2 Getting Started

As usual, you should set up your Eclipse workspace for today's lab:

1. Start Eclipse and create a new project called `lab06`. (Select **File → New ▪ Java Project**)

2. Next, import `PrintDirectory.java`, `TestFactorial.java`, and `Factorial.java` from the lab folder for today's lab into your workspace. (Start by highlighting the `src` folder and select **File ▪ Import...**)

   `~csci204/2011-spring/student/labs/lab06`

   `You should see the following files:`

- `Factorial.java`
- `TestFactorial.java`
- `PrintDirectory.java`

3. Finally, add the files into your Subversion repository. In Eclipse, click on `lab06` so that it is highlighted, and then right click. A popup menu will appear. Select **Team ▪ Share Project...** Then, select **SVN** as the repository type. and select your repository for your labs.

For future labs, you will simply be told to make a new project, import files for the lab (if necessary), and then add the new lab project into subversion. This will also be true of assignments and team projects as well, though you will need to set up a new repository for assignments and for projects. (The assignment and project guidelines will remind you of this.)

3 Practice with Recursion

Run `TestFactorial`. Observe the behavior of the program. Read the `Factorial` class and understand the logic of the program. Create a `readme.txt` file that answers the following questions.

4. If we rewrite the first branch of the if-else in the `factorial()` method as:

   ```
   if (n == 1)
       return 1;
   ```

   will the final result change? Why?

5. if we rewrite the branch as:

   ```
   if (n == 0)
       return 0;
   ```

   will the final result change? Why? Restore the program to its original state before proceeding.

6. If we revise the second branch of the if-else in the method as:

   ```
   else
       return 2 * n * factorial(n – 1);
   ```

   What will happen to the final result? Can you come up with a closed form mathematical equivalent of it?

Based on the `factorial()` method given above, write a similar recursive method that adds up all of the odd numbers between 1 and a given limit. Do not assume that this limit is an odd number. Add a method to the Factorial class to do the work, using the method signature:

```
public static long addOddInts(int n)
```

Uncomment the relevant lines in `main()` to test `addOddInts()`.

4 Review Fundamentals of Exceptions

Let's review some basic fundamental information about the exception framework in Java. In general, apart from for those exceptions that represent serious errors that are out of your control, exceptions will be classified as either *checked* exceptions or *unchecked* exceptions.

- *Checked exceptions* are those that must be handled by your program, and usually represent something that your program can recover from. These exceptions are all derived from the general exception class called `Exception`. One example that we've had to deal with already is the `FileNotFoundException`. You need to use a try-catch block to catch and recover from these, or declare that your method can throw the exceptions (using the throws keyword in the method signature) to allow a method higher up the call stack to handle it. It is generally unacceptable to simply let these types of exceptions terminate your program.

- *Unchecked exceptions* are those that indicate that you may likely have a bug in your program, such as an `ArithmeticError` or `NullPointerException`. You do not necessarily need to catch these in your final code (though you might if it makes sense for your design). You should not allow your program to be delivered with these being thrown. These exceptions are all derived from the general exception class `RuntimeException`.

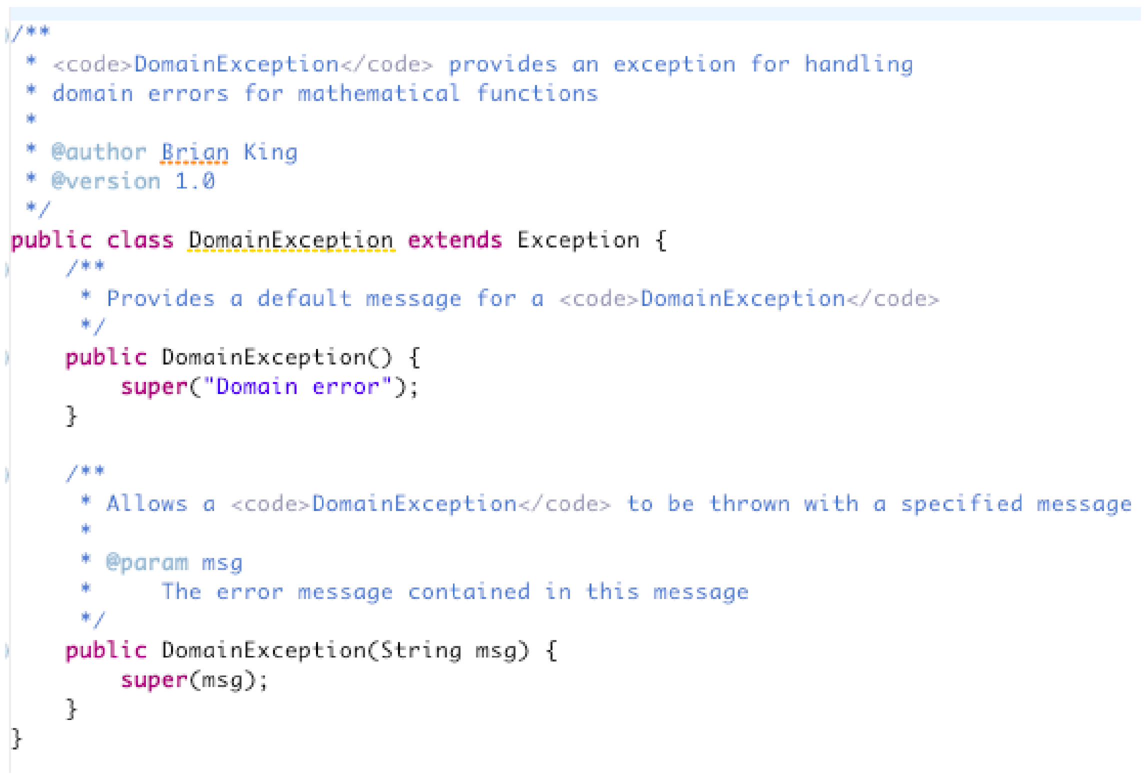
If you examine the Factorial program as it stands right now, we have no provision for doing any type of domain checking. Mathematical functions have a domain of values over which the function is defined. Factorial is only defined when the input is >= 0. What if the user tried to calculate the factorial of -5? Right now, we get some sort of result that is meaningless. We need to perform some sort of domain checking

5 Implement Your Own Exception

For this part of the lab, you are going to create your very own exception, and then use the same framework we've already learned about for handling it.

Create a new Java class called `DomainException`. It should be extended from the general `Exception` class (i.e. be sure to list this as the **Superclass**). The class as created will be empty at first. You will make two simple modifications.

First, examine the base class `Exception` in the Java API. (http://download.oracle.com/javase/6/docs/api/java/lang/Exception.html) You will notice that it is quite a simple class. We will overload the default constructor and the explicit constructor that takes a string message, and that is all. Your final class should look similar to the following (with your own Javadoc, of course):



```
/**
 * <code>DomainException</code> provides an exception for handling
 * domain errors for mathematical functions
 *
 * @author Brian King
 * @version 1.0
 */
public class DomainException extends Exception {
    /**
     * Provides a default message for a <code>DomainException</code>
     */
    public DomainException() {
        super("Domain error");
    }

    /**
     * Allows a <code>DomainException</code> to be thrown with a specified message
     *
     * @param msg
     *      The error message contained in this message
     */
    public DomainException(String msg) {
        super(msg);
    }
}
```

5.1 A note about `Serializable`

See that yellow line under the class name above? This warning is occurring because the base class `Exception` implements the `Serializable` interface. It requires a private static final long variable called `serialVersionUID`. Don't worry -- this is beyond the scope of the course. There is substantial information available online if you are interested. I encourage you to delve further on your own time. For now, you can simply let Eclipse generate a `serialVersionUID` for you, and the warning will disappear. (Hover the mouse cursor over the warning, and you'll see an obvious choice to fix the warning.)

5.2 Throwing `DomainException`

ThIs new exception of yours will be thrown by our `factorial` and `addOddInts` methods. Mathematically speaking, factorial is defined only for values that are >= 0. We will assume that the `addOddInts` is defined only for values >= 1.

Make the required modifications to *throw* the exception in these methods. Pass an appropriate error message to the constructor of the exception indicating the bad value that was passed to the function.

5.3 Catching `DomainException`

This is a checked exception. You'll notice that once you modify your two recursive methods in the `Factorial` class, now you'll have errors in the `main` method in the `TestFactorial` program. This is because you are not handling the new exception you just created. Since it is a checked exception, you need to handle it.

Write the code to *catch* the occurrence of `DomainException` that is thrown by both methods. Output the error message associated with the exception object caught using the `getMessage()` method of the exception object. Be sure to enclose the try-catch blocks within a do-while loop to allow the program to repeat getting user supplied input until no `DomainException` is thrown.

6 Recursively Listing Files

Recursive algorithms are natural for solutions to problems with hierarchical structure. An example problem is listing all the files in a directory and all of its subdirectories. Since the UNIX file system is hierarchical, we should immediately think of using a recursive approach.

For this part of the lab, you are to write a Java program to list all of the files in a directory and, recursively, in all of its subdirectories.

Read, and run the program contained within `PrintDirectory.java`. Observe the behavior of the program as it stands before you make any modifications. `PrintDirectory` is a Java program that lists the names of the files and directories in a directory, but does *not* recursively list the subdirectories. You'll notice that it checks to see if there are any command-line arguments passed to it. (Optional command line arguments to your program can be easily configured in Eclipse, or you can run the program from the command line yourself using the `java` command.) Assuming that no command line arguments are passed, it'll list the contents of the `labs` directory for the class.

You are to modify the `PrintDirectory` class to *recursively* print the names of all the files in all subdirectories. Here are some details.

7. You must use a recursive s   olution.
8. File names should be printed one per line.
9. Just *before* your recursive call, print out "Entering *name*" where you fill in the directory name.
10. Just *after* your recursive call, print out "Leaving *name*" where you fill in the directory name.
11. If you print a `File` object, Java will print the *full* pathname. You can print the file/directory name only by using the `getName()` method.
12. You may check if a `File` object is a file with the method `isFile()` which returns true if it is a file and false if it is a directory. If you find it more convenient, there is a similar method `isDirectory()` that tests whether a `File` object is a directory.

To check your program's output, use the following UNIX command:

```
ls –R ~csci204/2011-spring/student/labs
```

where it –R option is used to recursively list subdirectories encountered. Your listing should contain the same folders and files listed (albeit in a different format, in order to satisfy the requirements of the lab.)

Sample output:

```
Entering lab04
HTMLTagCounter.java
TestStubbornInput.java
```

```
WordCount.java
Leaving lab04


Entering lab02
Date.java
TestDates.java
cat-lion.jpg
garfield.gif
ShowImage.java
Leaving lab02


Entering lab05
BankAccount.class.violet
BankAccount.java
CashRegister.java
Money.java
Person.java
Leaving lab05


Entering lab06
Factorial.java
PrintDirectory.java
TestFactorial.java
Leaving lab06


Entering lab03
Point.java
TestPoint.java
TestWord.java
Word.java
Leaving lab03
```

(Note that if your directories come out in a different order they are still correct, this is an example of format, not an exact one correct answer).

7 Finishing up

Commit all source code files and your `readme.txt` file to your subversion repository.

NOTE: Be sure to have good Javadoc comments for any methods that you have written. You do not need to generate the Javadoc output.

At the end of this lab you will have created or edited the following:

- readme.txt for the recursion questions
- Factorial.java to throw exceptions
- TestFactorial.java to catch exceptions
- DomainException.java
- PrintDirectory.java