CISC 275: Introduction to Software Engineering

Lab 8:
Handling Exceptions in Java

Charlie Greenbacker
University of Delaware
Fall 2010

Overview

- Walkthrough of basic example
- Defining & throwing custom Exceptions
- Lab exercise

Walkthrough of basic

- Let's say we want to read from a file...

```
FileInputStream input = new
      FileInputStream("/path/to/file.txt");
```

Walkthrough of basic

- Let's say we want to read from a file...

```
FileInputStream input = new
      FileInputStream("/path/to/file.txt");
```

- But FileInputStream constructor throws FileNotFoundException, which we must handle!

Walkthrough of basic

- Surround it with a try/catch block...

```
try {
     FileInputStream input = new
          FileInputStream("/path/to/file.txt");
} catch (FileNotFoundException e) { ...
```

Walkthrough of basic

- Surround it with a try/catch block...

```
try {
     FileInputStream input = new
          FileInputStream("/path/to/file.txt");
} catch (FileNotFoundException e) { ...
```

- Inside catch statement, we need to do something as a result of the Exception
  - We can print an error, or return some default value, but for now we'll just print out the Exception we caught...

```
} catch (FileNotFoundException e) {
     System.out.println("Could not open file, "
          + "cause: " + e);
}
```

Walkthrough of basic

- Since we've opened the file, let's read from it...

```
FileInputStream input = new
  FileInputStream("/path/to/file.txt");
BufferedReader reader = new
  BufferedReader(new InputStreamReader(input));
System.out.println(reader.readLine());
```

Walkthrough of basic

- Since we've opened the file, let's read from it...

```
FileInputStream input = new
  FileInputStream("/path/to/file.txt");
BufferedReader reader = new
  BufferedReader(new InputStreamReader(input));
System.out.println(reader.readLine());
```

- But BufferedReader.readLine() throws IOException, which we must handle!
  - We can either add a new catch block to our existing try/catch statement, but since FileNotFoundException extends IOException, and we want to handle them the same, we can just modify catch

Walkthrough of basic

- So here's what will work...

```
try {
  FileInputStream input = new
       FileInputStream("/path/to/file.txt");
  BufferedReader reader = new
       BufferedReader(new InputStreamReader(input));
  System.out.println(reader.readLine());
} catch (IOException e) {
  System.out.println("Could not open file, cause: "
       + e);
}
```

- Use bad path to cause FileNotFoundException, or call input.close() before readLine() to get IOException: Bad file descriptor
- Review complete code: BasicException.java

Define & throw custom

- Defining your own custom Exception is as easy as this:

```
public class NegativeException extends Exception {
     public NegativeException(String s) {
          super(s);
     }
}
```

- Review class file: NegativeException.java

Define & throw custom

- We can now throw our new Exception by calling the constructor & passing a String message

```
new NegativeException("number is negative!");
```

Define & throw custom

- In a new class called MilliDate we'll write a method that throws NegativeException...

```
public static Date millisecondDate(int num)
  throws NegativeException {
      if (num < 0) {
          throw new NegativeException("number is "
              + "negative!");
      } else {
          return new Date(num);
      }
  }
```

Define & throw custom

- If we want to call millisecondDate(), we'll have to catch the Exception it throws...

```
public static void main(String[] args) {
      try {
            Date date = millisecondDate(987983607);
            System.out.println(date);
      } catch (NegativeException e) {
            System.out.println("Couldn't create Date, "
                  + e);
      }
  }
```

- Review complete code: MilliDate.java

Lab Exercise

- Define a new Exception named OddException, that will be just like NegativeException
- Create a new class named EvenOdd
  - Write a method called halfOf(), which takes an int, throws an OddException if the int is odd or zero, otherwise returns (int / 2)
  - Write a main method that calls halfOf() three times (once each with an even int, an odd int, and zero), with three try/catch blocks, and prints either the output of halfOf() or the caught OddException
- *Work alone, show me before leaving or email*

Lab Exercise

- Hints:
  - Use a different Exception message string depending on whether the int is odd or zero
  - Use the modulus operator to test if the int is odd: `if (num % 2 != 0)`...