COMP132 – Computer Science II
Fall 2015

Homework #6
Exceptions
Solutions

1. Consider the following class definitions:

```
public class WhatsTheOutput {

  public static void main(String[] args) {
    System.out.println("Main Starting");
    try {
      methodOne();
    }
    catch (ExceptionTypeE e) {
      System.out.println("Caught type E");
    }
    System.out.println("Main Ending");
  }

  public static void methodOne() {
    System.out.println("MethodOne Starting");
    try {
      methodTwo();
    }
    catch (ExceptionTypeC e) {
      System.out.println("Caught type C");
    }
    System.out.println("MethodOne Ending");
  }

  public static void methodTwo() {
    System.out.println("MethodTwo Starting");
    try {
      // THROW EXCEPION HERE!
    }
    catch (ExceptionTypeA e) {
      System.out.println("Caught type A");
    }
    System.out.println("MethodTwo Ending");
  }
}

public class ExceptionTypeA extends RuntimeException {…}
public class ExceptionTypeB extends RuntimeException {…}
public class ExceptionTypeC extends ExceptionTypeB {…}
public class ExceptionTypeD extends ExceptionTypeC {…}
public class ExceptionTypeE extends RuntimeException {…}
```

Indicate the output that the above program would produce if the following exception types were thrown on the line labeled "THROW EXCEPTION HERE". If the program would crash also indicate that by writing "CRASH!" at the end of the output.

a. ExceptionTypeA

```
Main Starting
MethodOne Starting
MethodTwo Starting
Caught type A
MethodTwo Ending
MethodOne Ending
Main Ending
```

The main method invokes methodOne, which invokes methodTwo, which then throws an exception of type ExceptionTypeA. The point where this exception is thrown is within a try/catch that catches ExceptionTypeA. So at the point where the exception is thrown the program jumps to the catch. In this case, the catch statement prints "Caught type A". Then, because the exception was caught, the execution of the program then continues as normal: methodTwo completes and control returns to methodOne, which completes normally, and control returns to main, which also completes normally.

b. ExceptionTypeB

```
Main Starting
MethodOne Starting
MethodTwo Starting
Crash!!
```

The main method invokes methodOne, which invokes methodTwo, which then throws an exception of type ExceptionTypeB. The point where this exception is thrown is within a try/catch, but it does not catch ExceptionTypeB (or any of its super classes). Thus, methodTwo is terminated immediately and control returns to the point where methodTwo was called from methodOne (i.e. the exception is propagated). This point is also in a try/catch, but it does not catch ExceptionTypeB (or any of its super classes). Thus, methodOne is terminated immediately and control returns to the point where methodOne was called from main. The situation in main is similar and thus the exception is also propagated from main, which causes the program to crash.

c. `ExceptionTypeC`

```
Main Starting
MethodOne Starting
MethodTwo Starting
Caught type C
MethodOne Ending
Main Ending
```

The `main` method invokes `methodOne`, which invokes `methodTwo`, which then throws an exception of type `ExceptionTypeC`. The point where this exception is thrown is within a `try/catch`, but it does not catch `ExceptionTypeC` (or any of its super classes). Thus, `methodTwo` is terminated immediately and control returns to the point where `methodTwo` was called from `methodOne` (i.e. the exception is propagated). This point is also in a `try/catch`, however this `try/catch` does catch `ExceptionTypeC`. Thus, control is transferred to the `catch` clause. The code in the `catch` executes, printing "`Caught type C`". Then, because the exception was caught, the execution of the program then continues as normal: `methodOne` completes and control returns `main`, which also completes normally.

d. `ExceptionTypeD`

```
Main Starting
MethodOne Starting
MethodTwo Starting
Caught type C
MethodOne Ending
Main Ending
```

The `main` method invokes `methodOne`, which invokes `methodTwo`, which then throws an exception of type `ExceptionTypeD`. The point where this exception is thrown is within a `try/catch`, but it does not catch `ExceptionTypeD` (or any of its super classes). Thus, `methodTwo` is terminated immediately and control returns to the point where `methodTwo` was called from `methodOne` (i.e. the exception is propagated). This point is in a `try/catch` that catches `ExceptionTypeC`. Because `ExceptionTypeD` "is a" `ExceptionTypeC` (i.e. `ExceptionTypeD` is a sub-class of `ExceptionTypeC`) the `ExceptionTypeD` is caught. Thus, control is transferred to the `catch` clause. The code in the `catch` executes, printing "`Caught type C`". Then, because the exception was caught, the execution of the program then continues as normal: `methodOne` completes and control returns `main`, which also completes normally.

e. ExceptionTypeE

```
Main Starting
MethodOne Starting
MethodTwo Starting
Caught type E
Main Ending
```

The `main` method invokes `methodOne`, which invokes `methodTwo`, which then throws an exception of type `ExceptionTypeE`. The point where this exception is thrown is within a `try/catch`, but it does not catch `ExceptionTypeE` (or any of its super classes). Thus, `methodTwo` is terminated immediately and control returns to the point where `methodTwo` was called from `methodOne` (i.e. the exception is propagated). This point is also in a `try/catch`, but it does not catch `ExceptionTypeE` (or any of its super classes). Thus, `methodOne` is terminated immediately and control returns to the point where `methodOne` was called from `main`. This point is also in a `try/catch`, but this try/catch does catch `ExceptionTypeE`. Thus, control is transferred to the `catch` clause. The code in the `catch` executes, printing "`Caught type E`". Then, because the exception was caught, the execution of the program then continues and `main` completes normally.

2. Rewrite `methodOne` from question #1 such that it catches exceptions for type `ExceptionTypeD` and `ExceptionTypeE` and prints distinct messages in each case. Give just your modified code for `methodOne`.

```java
public static void methodOne() {
    System.out.println("MethodOne Starting");
    try {
        methodTwo();
    }
    catch (ExceptionTypeD e) {
        System.out.println("Caught type D");
    }
    catch (ExceptionTypeE e) {
        System.out.println("Caught type E");
    }
    System.out.println("MethodOne Ending");
 }
```

3. Many methods in the Java Development Kit (JDK) throw exceptions when they are unable to perform the requested action. The JavaDoc for each method describes the type of exceptions that the method might throw and the circumstances under which they are thrown. Use the JDK documentation (linked to from the course home page) to identify the types of exceptions that might be thrown by the methods below. Also indicate if each exception type is **Checked** or **Unchecked**.

a. `Double.parseDouble(String val)`

NumberFormatException	**Unchecked**
NullPointerException	**Unchecked**

The JavaDoc for the `Double` class can be found in the `java.lang` package. Looking at the documentation for the `parseDouble` method shows that it may throw these exceptions. Looking at the JavaDoc for each of these exceptions shows that they are sub-classes of `RuntimeException`, and thus are unchecked exceptions.

b. `ArrayList.get(int index)`

IndexOutOfBoundsException	**Unchecked**

The JavaDoc for the `ArrayList` class can be found in the `java.util` package. Looking at the documentation for the `get` method shows that it may throw this type of exception. Looking at the JavaDoc for the `IndexOutOfBoundsException` shows that it is a sub-class of `RuntimeException`, and thus is an unchecked exception.

c. `Thread.sleep(long millis, int nanos)`

InterruptedException	**Checked**
IllegalArgumentException	**Unchecked**

The JavaDoc for the `Thread` class can be found in the `java.lang` package. Looking at the documentation for the `sleep` method shows that it may throw these exceptions. Looking at the JavaDoc for the `InterruptedException` shows that it is a sub-class of `Exception`, and thus is a checked exception. Looking at the JavaDoc for the `IllegalArgumentException` shows that it is a sub-class of `RuntimeException`, and thus is an unchecked exception.

d. `File.getCanonicalPath()`

`IOException` **`Checked`**
`SecurityException` **`Unchecked`**

The JavaDoc for the `File` class can be found in the `java.io` package. Looking at the documentation for the `getCanonicalPath` method shows that it may throw these exceptions. Looking at the JavaDoc for the `IOException` shows that it is a sub-class of `Exception`, and thus is a checked exception. Looking at the JavaDoc for the `SecurityException` shows that it is a sub-class of `RuntimeException`, and thus is an unchecked exception.

e. `JTextField.setColumns(int columns)`

`IllegalArgumentException` **`Unchecked`**

The JavaDoc for the `JTextField` class can be found in the `javax.swing` package. Looking at the documentation for the `setColumns` method shows that it may throw this type of exception. Looking at the JavaDoc for the `IllegalArgumentException` shows that it is a sub-class of `RuntimeException`, and thus is an unchecked exception.

4. The following questions use the `PhoneNumber` example from the `comp132.examples.exceptions.phone` package in the `132SampleCode` project.

a. Try to create a new `PhoneNumber` with an invalid number (e.g. "`(abc) 245-1401`"). What type of exception do you receive? In which method in the PhoneNumber class does the exception occur? On which line of the `PhoneNumber` class is the exception generated?

The statement:

```
PhoneNumber p = new PhoneNumber("(abc) 245-1401");
```

generates a `NumberFormatException` on line 26 of the `parseAreaCode` method where it invokes the `parseInt` method in the `Integer` class.

b. Try to create a new `PhoneNumber` that does not contain enough digits (e.g. "`(abc) 245-140`"). What type of exception do you receive? In which method in the PhoneNumber class does the exception occur? On which line of the `PhoneNumber` class is the exception generated?

The statement:

```
PhoneNumber p = new PhoneNumber("(717) 245-140");
```

generates a `StringIndexOutOfBoundsException` on line 37 of the `parseNumber` method where it invokes the `substring` method in the `String` class.

c. Modify the `PhoneNumber` constructor so that it throws an `IllegalArgumentException`, with a descriptive message, instead of the different exception types identified in parts a and b. Paste your constructor code as your solution to this question.

```
public PhoneNumber(String phNumber) {
   try {
      areaCode = parseAreaCode(phNumber);
      exchange = parseExchange(phNumber);
      number = parseNumber(phNumber);
   }
   catch(NumberFormatException e) {
      throw new IllegalArgumentException(
         "Bad phone number: " + phNumber);
   }
   catch(StringIndexOutOfBoundsException e) {
      throw new IllegalArgumentException(
         "Bad phone number: " + phNumber);
   }
}
```

d. Write a snippet of code that reads a phone number from the user. If an invalid phone number is entered your code should print an error message and the user should be prompted again. Paste your code as your solution to this exercise.

```java
public static void main(String[] args) {

   Scanner scr = new Scanner(System.in);
   PhoneNumber pn = null;
   while (pn == null) {
      System.out.print("Enter a phone number: ");
      String numStr = scr.nextLine();
      try {
         pn = new PhoneNumber(numStr);
      }
      catch(IllegalArgumentException e) {
         System.out.println(e.getMessage());
         System.out.println("Please try again.");
      }
   }
}
```

e. Write a JUnit test that checks that the `PhoneNumber` constructor now throws the correct exception when an invalid phone number is provided. Give the code for your JUnit test as the answer to this question.

```java
@Test
public void testBadNumberFormat() {
   try {
      new CopyOfPhoneNumber("(abc) 245-1401");
      fail("Should throw exception.");
   }
   catch(IllegalArgumentException e) {
      // pass
   }
   catch(Exception e) {
      fail("Threw incorrect exception type.");
   }
}
```

5. The following questions also use the PhoneNumber example from the comp132.examples.exceptions.phone package in the 132SampleCode project. It turns out that not all 3 digit numbers are valid area codes. The first digit of an area code must be [2…9], the second digit must be [0…8], but the third digit may be [0…9].

a. Create a new type of Checked Exception called InvalidAreaCodeException. Give the code for your InvalidAreaCodeException class as the answer for this question.

```
public class InvalidAreaCodeException extends Exception {
   public InvalidAreaCodeException(String msg) {
      super(msg);
   }
}
```

A Checked Exception is a sub-class of Exception. So InvalidAreaCodeException extends Exception. The constructor for InvalidAreaCodeException simply accepts a String as a descriptive message of the exception and passes that to the constructor of the Exception class. Nothing else is required as InvalidAreaCodeException inherits everything else from Exception.

b. Modify the parseAreaCode method so that it throws an InvalidAreaCodeException if the number can be parsed but is not valid according to the rules outlined above. Paste your modified parseAreaCode method as your solution to this question.

```
private int parseAreaCode(String phNumber) throws
   InvalidAreaCodeException {

   String acStr = phNumber.substring(1, 4);

   if (acStr.charAt(0) == '0' || acStr.charAt(0) == '1'
        || acStr.charAt(1) == '9') {
      throw new InvalidAreaCodeException("Bad area code: " + acStr);
   }

   int acInt = Integer.parseInt(acStr);
   return acInt;
}
```

c. Modify your `PhoneNumber` Constructor from problem #4c so that it will propagate an `InvalidAreaCodeException` exception if it occurs, but will still throw `IllegalArgumentExceptions` as it did before. Paste your modified constructor as your solution to this question.

```
public CopyOfPhoneNumber(String phNumber) throws
  InvalidAreaCodeException {
     …
}
```

The only change necessary is to add the throws clause to the end of the method declaration. This indicates that code in this method may throw a Checked Exception and that this method is going to allow that exception to propagate.