Exception-handling Problems

The purpose of all this is for you to get practice. As long as this is a good attempt, with most things right, you get 100%.

It may help to do this with another person, together. If you do that, put both names and netids at the top. If you have a question, ask the TA or the people around you. Go ahead, discuss things with those around you. Some of these questions are mechanical, asking for a definition or something like that. Some ask you to write code or execute a method call. You can look at notes or a book, get on the internet and read the exception-handling webpage, watch the videos, google something, whatever.

1. Answer the following short questions

   (a) Which Java class is the parent class of all Exceptions? Throwable

   (b) What is the difference between Exceptions and Errors?You can handle Exceptions, but don't handle Errors because they really mean something is messed up, like out of space.

   (c) What is the purpose of Exceptions? Give an example of what they're used for. Exceptions are thrown objects that you might want to catch and handle, if you know what to do with them. Examples are ArithmeticException, ArrayIndexOutOfBoundsException, and IllegalArgumentException.

2. Draw the basic exception-class hierarchy, with Throwable at the top. Include at least two subclasses of Error, Exception, and RuntimeException.

```
Throwable
    Error
        OutOfMemoryError
        InternalError
    Exception
        IOException
        RuntimeException
            ArithmeticException
            ArrayStoreException
            NullPointerException
```

3. In the following statements, suppose S1 throws an Exception, so S2 is executed. State what happens in two cases: (1) S2 throws an exception, (2) S2 does not throw an exception.

```
try { S1 }
catch (Exception e) { S2 }
S3
```

   If S2 throws an exception (and it is not within a second try-block), the exception is thrown out to the place of call. If S2 doesn't throw an exception, the try-statement execution ends —and S3 is executed.

4. Write a throw statement that throws an ArithmeticException with message "arg should not be negative". Remember: use a new-expression to create a throwable object.

   throw new ArithmeticException("arg should not be negative");

5. Consider the following class:

```
public class A {
    public static double m(int x) {
        int y= x;
        try {
            System.out.println("one");
            y= 5/x;
            System.out.println("two");
            return 5/(x + 2);
        } catch (RuntimeException e) {
            System.out.println("three");
            y= 5/(x+1);
            System.out.println("four");
        }
        System.out.println("five");
        y= 4/x;
        System.out.println("six");
        return 1/x;
    }
}
```

   (a) Below, write what is printed by execution of the call `m(0)` .
   one three four five (each on its own line)

   (b) Below, write what is printed by execution of the call `m(-2)` .
   one two three four five six (each on its own line)

   (c) Below, write what is printed by execution of the call `m(-1)` .
   one two (each on its own line)

6. Here is a method to return the minimum value of array segment b[m..n]. It is not correct because it does not throw the required exception if b[m..n] is empty. Place a suitable throw statement at the beginning of the method body.

```
/** Return the minimum value in b[m..n]. Throw a RuntimeException with
  * message "min of 0 values doesn't exist" if b[m..n] is empty. */
public int min(int[] b, int m, int n) {

    if (m > n) throw new RuntimeException("min of 0 values doesn't exist");

    int min= b[m];
    for (int k= m+1; k <= n; k= k+1) {
        if (b[k] > min) min= b[k];
    }
    return min;
}
```

7. Here is a method to return the minimum value of array b. It is not correct because it does not throw the required exception if b is empty. Yes, you can create an array with 0 values: int[] c= new c[0].

```
/** Return the minimum value in b. Throw a RuntimeException with
  * message "min of 0 values doesn't exist" if b is empty. */
public int min(int[] b) {
    int min= b[0];
    for (int k= 0; k < b.length; k= k+1) {
        if (b[k] > min) min= b[k];
    }
    return min;
}
```

Instead of inserting a throw statement in the method body, below, rewrite the method body to use a try statement, with the whole method body in the try block. Let the catch clause catch an ArrayIndexOutOfBoundsException. In the catch block, throw the required exception. You don't have to copy the method specification or header; just write the body.

```
/** Return the minimum value in b. Throw a RuntimeException with
  * message "min of 0 values doesn't exist" if b is empty. */
public int min(int[] b) {
    try {
        int min= b[0];
        for (int k= 0; k < b.length; k= k+1) {
            if (b[k] > min) min= b[k];
        }
        return min;
    } catch (ArrayIndexOutOfBoundsException e) {
          throw new RuntimeException("min of 0 values doesn't exist");

       }
    }
}
```

8. Consider the following class. Several possible sequences of numbers can be printed by a call first(i); depending on the value of i. List each sequences of numbers that can be printed by such a call, along with the value of i that causes that sequence to be printed.

```
public class B {
    public static void first(int i) {
        try {
            System.out.println("0");
            second(i);
            System.out.println("1");
        } catch (Exception e) {
            System.out.println("2");
        }
    }

    public static void second(int i) throws Exception {
        System.out.println("3");
        try {
            int b= 5/i;
            System.out.println("4");
            if (i == 6) throw new Exception();
            System.out.println("5");
        } catch (ArithmeticException e) {
            System.out.println("6");
        }
        System.out.println("7");
    }
}
```

Case i = 0: prints 0 3 6 7 1 (each on a separate line)

Case i = 6: prints 0 3 4 2 (each on a separate line)

Case i != 0 and i != 6: prints 0 3 4 5 7 1 (each on a separate line)

9. Consider class C given below. Function Integer.parseInt throws a `NumberFormatException` if its argument does not contain an integer. Below class C, rewrite the class so that if Integer.parseInt throws an exception, the number 1 is used. Note that Integer.parseInt is called in two places so you may need two try-statements.

   Don't be concerned with how one reads from the keyboard, pausing until something is typed.

```
public class C {
    /** Print the sum of two integers read from the keyboard */
    public static void main(String[] args) {
        System.out.println("Enter a number: ");
        String s;
        Read a line from the keyboard and store it in s;
        int a= Integer.parseInt(s);

        System.out.println("Enter another number: ");
        Read a line from the keyboard and store it in s;
        int b= Integer.parseInt(s);

        System.out.println("Product: " + a*b);
    }
}
```

```
 public class C {
    /** Print the sum of two integers read from the keyboard */
    public static void main(String[] args) {
        System.out.println("Enter a number: ");
        String s;
        Read a line from the keyboard and store it in s;
        int a= 1;
        try {a= Integer.parseInt(s);}
        catch (NumberFormatException nfe) {}

        System.out.println("Enter another number: ");
        Read a line from the keyboard and store it in s;
        int b= 1;
        try {b= Integer.parseInt(s);}
        catch (NumberFormatException nfe) {}

        System.out.println("Product: " + a*b);
    }
}
```

10. Below, write an unchecked Exception class `MyException`.

```
 public class MyException extends RuntimeException {
       public MyException() {}
       public MyException(String m) {super(m)}

 }
```