Chapter 11

Question R11.1:

R11.1 What happens if you try to open a file for reading that doesn't exist? What happens if you try to open a file for writing that doesn't exist?

The system throws an IOException in each situation.

Question R11.2.

R11.2 What happens if you try to open a file for writing, but the file or device is write-protected (sometimes called read-only)? Try it out with a short test program.

IOException might occur if you are trying to create a file on a disk that is "write-protected," meaning that it cannot be modified. But you can open a file in "read-only" mode.

Question R11.3.

R11.3 How do you open a file whose name contains a backslash, like c:\temp\output.dat?

A backslash — like is used in a file path — can't be used directly in a Java String. Java uses the backslash to mean "the next character means something special". Doubling the backslash in Java indicates that "the next character (the second backslash) should be inserted into the String".

Question R11.4.

R11.4 What is a command line? How can a program read its command line arguments?

A Java application can accept any number of arguments from the command line. This allows the user to specify configuration information when the application is launched.

The user enters command-line arguments when invoking the application and specifies them after the name of the class to be run. For example, suppose a Java application called Sort sorts lines in a file. To sort the data in a file named friends.txt, a user would enter:

```
C:\>java Sort friends.txt
```

When an application is launched, the runtime system passes the command-line arguments to the application's main method via an array of Strings. In the previous example, the command-line arguments passed to the Sort application in an array that contains a single String: "friends.txt".

Question R11.7.

R11.7 What is the difference between throwing an exception and catching an exception?

Before catching an exception it is must to be thrown first. This means that there should be a code somewhere in the program that could catch the exception. We use throw statement to throw an exception or simply use the throw keyword with an object reference to throw an exception. A single argument is required by the throw statement i.e. a throwable object. Throwable objects are instances of any subclass of the Throwable class.

Question R11.8.

R11.8 What is a checked exception? What is an unchecked exception? Is a NullPointer-Exception checked or unchecked? Which exceptions do you need to declare with the throws reserved word?

Unchecked exceptions:

- Represent defects in the program (bugs) - often invalid arguments passed to a non-private method. To quote from The Java Programming Language, by Gosling, Arnold, and Holmes : "Unchecked runtime exceptions represent conditions that, generally speaking, reflect errors in your program's logic and cannot be reasonably recovered from at run time."
- Are subclasses of RuntimeException, and are usually implemented using IllegalArgumentException, NullPointerException, or IllegalStateException.
- A method is not obliged to establish a policy for the unchecked exceptions thrown by its implementation (and they almost always do not do so)

Checked exceptions:

- Represent invalid conditions in areas outside the immediate control of the program (invalid user input, database problems, network outages, absent files)
- Are subclasses of Exception
- A method is obliged to establish a policy for all checked exceptions thrown by its implementation (either pass the checked exception further up the stack, or handle it somehow)

It is somewhat confusing, but note as well that RuntimeException (unchecked) is itself a subclass of Exception (checked).

Question R11.10.

R11.10 When your program executes a throw statement, which statement is executed next?
After running the exception handler, the interpreter continues execution at the statement immediately following the handler code.

Question R11.12.

R11.12 What can your program do with the exception object that a catch clause receives?

The variable exception contains a reference to the exception object that was thrown. The catch clause can analyze that object to find out more details about the failure. For example, you can get a printout of the chain of method calls that lead to the exception, by calling

```
exception.printStackTrace()
```

In these sample catch clauses, we merely inform the user of the source of the problem.
A better way of dealing with the exception would be to give the user another chance to provide a correct input.

Question R11.14.
R11.14 What kind of values can you throw? Can you throw a string? An integer?

Only instances of subclasses of throwable can be used in conjunction with the throw keyword. In java all exceptions and errors are subclasses of throwable.

Question R.11.16.
R11.16 What happens when an exception is thrown, the code of a finally clause executes, and that code throws an exception of a different kind than the original one? Which one is caught by a surrounding catch clause?

Usually, when you work with streams, sockets, database connections and that sort of things, you should 'think safe'. The usual approach is indeed to put a try/catch clause inside the finally clause, so that it won't crash your app if something goes wrong while closing the stream. Don't worry about your code being a bit cluttered: safety has a price.
It is sometime necessary to put a try catch block within a catch or finally block.
A Finally block will be executed after a try block if no exception has been thrown or after a catch if an exception was thrown. This means that a finally block can be used as 'clean up' - a place to close files or connections etc., whether an exception is thrown or not.

Question R11.17.
R11.17 Which exceptions can the next and nextInt methods of the Scanner class throw? Are they checked exceptions or unchecked exceptions?
int nextInt(): Returns the next token as an int. If the next token is not an integer, InputMismatchException is thrown.
String next(): Finds and returns the next complete token from this scanner and returns it as a string; a token is usually ended by whitespace such as a blank or line break. If not token exists, NoSuchElementException is thrown.

Both are checked exceptions.

Question R11.18.
R11.18 Suppose the code in Quality Tip 11.3 on page 489 had been condensed to a single try/catch/finally statement:

```
PrintWriter out = new PrintWriter(filename);
Try {
  Write output
}
catch (IOException exception) {
   Handle exception
}
Finally {
   out.close();
}
```

What is the disadvantage of this version? (Hint: What happens when the PrintWriter constructor throws an exception?) Why can't you solve the problem by moving the declaration of the out variable inside the try block?

The instruction `out.close()` can throws an exception that not be catch.
Why can't you solve the problem by moving the declaration of the out variable inside the try block?
Because the problem can appear again.

Question P11.1

P11.1 Write a program that asks a user for a file name and prints the number of characters, words, and lines in that file.

```
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;


public class FileAnalyzer
{
   public static void main(String[] args) throws IOException
   {
      System.out.println("Filename: ");
      Scanner in = new Scanner(System.in);
      String name = in.nextLine();
      FileCounter counter = new FileCounter();
      FileReader reader = new FileReader(name);
      Scanner fileIn = new Scanner(reader);
      counter.read(fileIn);
      fileIn.close();
      System.out.println("Characters: " + counter.getCharacterCount());
      System.out.println("Words: " + counter.getWordCount());
      System.out.println("Lines : " + counter.getLineCount());
   }
}


import java.util.Scanner;
/**
   A class to count the number of characters, words, and lines in files.
*/
public class FileCounter
{
   private int numWords;
   private int numLines;
   private int numCharacters;
   public FileCounter()
   {
     this.numWords=0;
     this.numLines=0;
     this.numCharacters=0;
   }

   /**
      Processes an input source and adds its character, word, and line
      counts to this counter.
      @param in the scanner to process
   */
   public void read(Scanner in)
   {
         String s;
      while(in.hasNextLine()){
          this.numLines++;
          this.numWords++;
          s=in.nextLine();
          for(int i=0;i<s.length();i++){
                  if(s.charAt(i)==' '){this.numWords++;} else {this.numCharacters++;};
          }

      }

   }

   /**
      Gets the number of words in this counter.
      @return the number of words
   */
   public int getWordCount()
   {
      return numWords;
   }

   /**
      Gets the number of lines in this counter.
      @return the number of lines
   */
   public int getLineCount()
   {
     return numLines;
   }
   /**
      Gets the number of characters in this counter.
      @return the number of characters
   */
   public int getCharacterCount()
   {
     return numCharacters;
   }

}
```

Question P11.2.

P11.2 Write a program that asks the user for a file name and counts the number of characters, words, and lines in that file. Then the program asks for the name of the next file. When the user enters a file that doesn't exist, the program prints the total counts of characters, words, and lines in all processed files and exits.

```
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;

public class FileAnalyzer
{
   public static void main(String[] args) throws IOException
   {

      Scanner in = new Scanner(System.in);
      FileCounter counter = new FileCounter();
      int nFiles=0;
      boolean Fail=false;
      while(!Fail)
      try{
          System.out.println("Filename: ");
           String name = in.nextLine();
           FileReader reader = new FileReader(name);
           Scanner fileIn = new Scanner(reader);
           counter.read(fileIn);
           fileIn.close();
           nFiles++;
           System.out.println("Characters: " + counter.getCharacterCount());
           System.out.println("Words: " + counter.getWordCount());
           System.out.println("Lines : " + counter.getLineCount());
      }catch(Exception e){
          System.out.println("" + nFiles + " Files read...");
          System.out.println("Total Characters: " + counter.getTotalCharacterCount());
           System.out.println("Total Words: " + counter.getTotalWordCount());
           System.out.println("Total Lines : " + counter.getTotalLineCount());
           Fail=true;
      }

   }
}


import java.util.Scanner;
/**
   A class to count the number of characters, words, and lines in files.
*/
public class FileCounter
{
   private int numWords;
   private int numLines;
   private int numCharacters;
   private int numTotalWords;
   private int numTotalLines;
   private int numTotalCharacters;
   public FileCounter()
   {
     this.numWords=0;
     this.numLines=0;
     this.numCharacters=0;
     this.numTotalWords=0;
     this.numTotalLines=0;
     this.numTotalCharacters=0;
   }

   /**
      Processes an input source and adds its character, word, and line
      counts to this counter.
      @param in the scanner to process
   */
   public void read(Scanner in)
   {
          String s;
          this.numWords=0;
          this.numLines=0;
          this.numCharacters=0;
      while(in.hasNextLine()){
          this.numLines++;
          this.numWords++;
          s=in.nextLine();
          for(int i=0;i<s.length();i++){
                  if(s.charAt(i)==' '){this.numWords++;} else {this.numCharacters++;};
          }
      }
         this.numTotalLines+=this.numLines;
         this.numTotalCharacters+=this.numCharacters;
         this.numTotalWords+=this.numWords;

   }

   /**
      Gets the number of words in this counter.
      @return the number of words
   */
   public int getWordCount()
   {
      return numWords;
   }
   public int getTotalWordCount()
   {
      return numTotalWords;
   }
   /**
      Gets the number of lines in this counter.
      @return the number of lines
   */
   public int getLineCount()
   {
     return numLines;
   }
   public int getTotalLineCount()
   {
     return numTotalLines;
   }
   /**
      Gets the number of characters in this counter.
      @return the number of characters
   */
   public int getCharacterCount()
   {
     return numCharacters;
   }
   public int getTotalCharacterCount()
   {
     return numTotalCharacters;
   }
}
```

Question P11.4.

P11.4 Write a program that concatenates the contents of several files into one file. For example,

```
java CatFiles chapter1.txt chapter2.txt chapter3.txt book.txt
```

makes a long file, book.txt, that contains the contents of the files chapter1.txt, chapter2.txt, and chapter3.txt. The output file is always the last file specified on the command line.

```
import java.io.*;

public class ConcatenatedFiles {

        static public void main(String arg[]) throws java.io.IOException {
                PrintWriter pw = new PrintWriter(new FileOutputStream(arg[arg.length-1]));
                for (int nFile = 0; nFile < arg.length-1; nFile++) {
                        System.out.println("Processing " + arg[nFile] + "... ");
                        BufferedReader br = new BufferedReader(new FileReader(arg[nFile]));
                        String line = br.readLine();
                        while (line != null) {
                                pw.println(line);
                                line = br.readLine();
                        }
                        br.close();
                }
                pw.close();
                System.out.println("All files have been concatenated into " + arg[arg.length-1]);
        }
}
```

Question P11.5.

P11.5 Write a program Find that searches all files specified on the command line and prints out all lines containing a reserved word. For example, if you call

```
java Find ring report.txt address.txt Homework.java
```

then the program might print

```
report.txt: has broken up an international ring of DVD bootleggers that
address.txt: Kris Kringle, North Pole
address.txt: Homer Simpson, Springfield
Homework.java: String filename; The reserved word is always the first command line argument
```

```
import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.PrintWriter;

public class P11_5 {
        // this method return true if Pattern is substring of Line
        public static boolean hasString(String Line,String Pattern){
                boolean has=false;
                String Temp;
                for(int i=0;i<(Line.length()-Pattern.length()+1);i++){
                        Temp=Line.substring(i, i+Pattern.length());
                        if(Temp.equals(Pattern)) {has=true;i=Line.length()-Pattern.length()+1;}
                }
                return has;
        }
        static public void main(String arg[]) throws java.io.IOException {
                    String Pattern = arg[0];

                 for (int nFile = 1; nFile < arg.length; nFile++) {

                        System.out.println("Processing " + arg[nFile] + "... ");
                        BufferedReader br = new BufferedReader(new FileReader(arg[nFile]));
                        String line = br.readLine();
                        if(hasString(line,Pattern)) System.out.println(arg[nFile]+ ": " + line);
                        while (line != null) {
                                line = br.readLine();
                                if(hasString(line,Pattern)) System.out.println(arg[nFile]+ ": " +
line);
                        }
                        br.close();
                 }

        }
}
```

Question 11.11.

P11.6 Write a program that checks the spelling of all words in a file. It should read each word of a file and check whether it is contained in a word list. A word list is available on most UNIX systems in the file /usr/dict/words. (If you don't have access to a UNIX system, your instructor should be able to get you a copy.) The program should print out all words that it cannot find in the word list.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.Scanner;

public class P11_6 {
        // This method return true if Word is in Dictionary File
        public static boolean isCorrect(String Dictionary,String Word)throws java.io.IOException{
               boolean is=false;
               BufferedReader br = new BufferedReader(new FileReader(Dictionary));
            String line = br.readLine();
            if(line.equals(Word)) is=true;
            while (line != null) {
                line = br.readLine();
                if(line.equals(Word)) is=true;
            }
            br.close();
               return is;
        }
        static public void main(String arg[]) throws java.io.IOException {
                   String Dictionary = arg[0];
                String File = arg[1];

                System.out.println("Processing " + File + "... ");
                Scanner sc = new Scanner(new File(File));
                String Word;
                while(sc.hasNext()){
                   Word = sc.next();
                   if(!isCorrect(Dictionary,Word)) System.out.println(Word);
                }
                sc.close();


        }
}
```

Question P11.14.

P11.14 Write a program that asks the user to input a set of floating-point values. When the user enters a value that is not a number, give the user a second chance to enter the value. After two chances, quit reading input. Add all correctly specified values and print the sum when the user is done entering data. Use exception handling to detect improper inputs.

```
import java.util.InputMismatchException;
import java.util.Scanner;


public class P11_14 {

       public static void main(String Args[]){
               boolean flagA=true;
               boolean flagB=true;
               float num;
               float sum=0;
               Scanner input = new Scanner(System.in);

               while (flagA)
```

```
                {
                        try {
                                while (flagB)
                                {
                                        try {
                                                System.out.print("Value: ");
                                                num = input.nextFloat();
                                                sum += num;
                                        }
                                        catch (InputMismatchException e) {
                                                System.out.println("Input error. Try again.");
                                                flagB = false;
                                        }
                                }
                                System.out.print("Value: ");
                                num = input.nextFloat();
                                sum += num;
                                flagB = true;
                        }
                        catch (InputMismatchException f) {
                                System.out.println("Input error. Try again");
                                flagA = false;
                        }
                }
                System.out.println(sum);
        }

}
```