Lab Questions
Big Java, Late Objects / Java for Everyone, 2e
Chapter Number: 7 Input/Output and Exception Handling

1.1) Using a text editor, copy the data below into a text file called `ziptable.txt`. Each line of the table contains a state abbreviation, the state name, and one or more ZIP code specifiers. The ZIP code specifiers consist of the first three digits of a ZIP code. For example 350-369 denotes ZIP codes in the range 35000 to 36999.

Tables like this one are used in mailing applications to verify that an address has a reasonable ZIP code. Each ZIP code specifier is either a pair of three-digit integers separated with a hyphen, or a single three-digit integer. A ZIP code for a state is "valid" if the first three digits falls in the range of a ZIP code specifier pair or matches a single three-digit ZIP code specifier.

In order to standardize the data in the table, we will create a new file by modifying the old table data so that every ZIP code specifier is a pair or integers separated with a hyphen. For example, in the case of New York, we will change the single ZIP code 005 into 005-005. We will divide the task into three steps.

As the first step, write a program that reads and prints each line in the table exactly as it appears in the file.

Here is the table data:

AL Alabama 350-369
AK Alaska 995-999
AS American Samoa 967-967
AZ Arizona 850-865
AR Arkansas 716-729 755
CA California 900-966
CO Colorado 800-816
CT Connecticut 060-069
DE Delaware 197-199
DC District of Columbia 200-205
FM FS of Micronesia 969-969
FL Florida 320-349
GA Georgia 300-319 398-399
GU Guam 969-969
HI Hawaii 967-968
ID Idaho 832-838
IL Illinois 600-629
IN Indiana 460-479
IA Iowa 500-528
KS Kansas 660-679
KY Kentucky 400-427
LA Louisiana 700-714
ME Maine 039-049

MH Marshall Islands 969-969
MD Maryland 206-219
MA Massachusetts 010-027 055
MI Michigan 480-499
MN Minnesota 550-567
MS Mississippi 386-397
MO Missouri 630-658
MT Montana 590-599
NE Nebraska 680-693
NV Nevada 889-898
NH New Hampshire 030-039
NJ New Jersey 070-089
NM New Mexico 870-884
NY New York 005 063 090-149
NC North Carolina 269-289
ND North Dakota 580-588
MP N. Mariana Islands 969-969
OH Ohio 430-459
OK Oklahoma 730-749
OR Oregon 970-979
PW Palau Island 969-969
PA Pennsylvania 150-196
PR Puerto Rico 006-009
RI Rhode Island 028-029
SC South Carolina 290-299
SD South Dakota 570-577
TN Tennessee 370-385
TX Texas 750-799 885
UT Utah 840-847
VT Vermont 050-059
VA Virginia 201 220-246
VI Virgin Islands 008-008
WA Washington 980-994
WI Wisconsin 530-549
WV West Virginia 247-268
WY Wyoming 820-831

1.2) Modify the `ZipsReader` program so that it passes each line of the table file as a `String` to another `Scanner` constructor. Use the new `Scanner` to read the single line of the table as a sequence of strings by invoking the `next` method. Print out the state name, state abbreviation, and each ZIP code specifier.

In order to make the output consistent, single integer ZIP code specifiers should be printed as a two-integer range. For example, New York's ZIP code of 005 should be printed as 005-005. There are a couple of hurdles to completing this:

1) Some states like New Jersey have multi-part names, and
2) We need to determine which strings are ZIP code specifiers.

Fortunately, the `String` class supports the `matches` method and we can use that method to determine if a string matches a specific pattern. We will denote the pattern as a regular expression. For example, if we have a `String` identifier called `token`, we can invoke `token.matches("\\d{3}-\\d{3}")` to make sure the string referenced by token consists of three digits followed by a "-" followed by three more digits. We can use `token.matches("\\d{3}")` to make sure the string referenced by `token` consists of exactly three digits. In each case, `matches` returns `true` or `false`, so we can use this expression in an `if` statement to detect when we have scanned a ZIP code specifier.

1.3) Modify the `ZipsReader` program by directing the output to a file rather than `System.out`. Create a `PrintWriter` object that is associated with a file for output. Use `print` and `println` to write your output to the file. Be sure to invoke `close` on the stream when you are finished. Use a text editor to examine the output file and verify that it is correct.

2.1) Write a program to store multiple memos in a file. Allow a user to enter a topic and the text of a memo (the text of the memo is stored as a single line and thus cannot contain a return character). Store the topic, the date stamp of the memo, and the memo message.

Creating a `java.util.Date` object with no arguments will initialize the `Date` object to the current time and date. A date stamp is obtained by calling the `Date.toString()` method.

Use a text editor to view the contents of the output and to check that the information is stored correctly.

Part of the program code has been provided for you:

```
import . . .

public class MemoPadCreator
{
   public static void main(String[] args) throws FileNotFoundException
   {
      Date now;
      Scanner console = new Scanner(System.in);
      System.out.print("Output file: ");
      String filename = console.nextLine();

      PrintWriter out = . . .;

      boolean done = false;
      while (!done)
      {
         System.out.println("Memo topic (enter -1 to end):");
         String topic = console.nextLine();
         if (topic.equals("-1"))
         {
            done = true;
```



```
         }
         else
         {
            System.out.println("Memo text:");
            String message = console.nextLine();
            // Create the new date object and obtain a dateStamp
            out.println(topic + "\n" + dateStamp + "\n" + message);
         }
      }
      // Close the output file
   }
}
```

2.2) Modify your memo writing program to read multiple memos stored in a text file. Memos are stored in three lines. The first line is the memo topic, the second line is the date stamp, and the third line is the memo message. Display the memos one at a time, and allow the user to choose to view the next memo (if there is one).

Part of the program code has been provided for you:

```
. . .

public class MemoPadReader
{
   public static void main(String[] args) throws IOException
   {
      Scanner console = new Scanner(System.in);
      System.out.print("Input file: ");
      String inputFileName = console.nextLine();

      File inFile = . . .;
      Scanner in = new Scanner(inFile);

      boolean done = false;
      while (in.hasNextLine() && !done)
      {
         String topic = . . .;
         String dateStamp = . . .;
         String message = . . .;
         System.out.println(topic + "\n" + dateStamp + "\n" + message);

         if (. . .) // You should only ask to display the next memo if
                    // there are more memos in the file
         {
            System.out.println("Do you want to read the next memo (y/n)?");
            String ans = console.nextLine();
            if (ans.equalsIgnoreCase("n"))
            {
               done = true;
            }
         }
      }
   }
}
```


2.3) Modify your simple memo reader program. Use a `JFileChooser` dialog box to allow the user to choose the file from which the memos will be read.

You can use the `showOpenDialog` method to enable the user to select a file to open. This method returns either `JFileChooser.APPROVE_OPTION`, if the user has chosen a file, or `JFileChooser.CANCEL_OPTION`, if the user canceled the selection.

If a file was chosen, then you call the `getSelectedFile` method to obtain a `File` object that describes the file. Here is a complete example:

```
JFileChooser chooser = new JFileChooser();
if (chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
{
   File selectedFile = chooser.getSelectedFile();
   Scanner in = new Scanner(selectedFile);
   . . .
}
```

Provide the complete code of your modified `MemoPadReader` program below.

3) In this exercise you will develop skills with input and output of text strings. Start with the code below and complete the following tasks:

a) Read `String input1` with a `Scanner` and print each word.
b) Read `String input2` with a `Scanner` and print each character.
c) Read `String input3` with a `Scanner` and print each character and whether it is a letter, a digit, or whitespace.
d) Read `String input4` with a `Scanner` and print each line on a different line.

```
public class TextIO
{
   public static void main(String[] args)
   {
      String input1 = "Now is the time for all good men to come to the aid of their
country.";
      String input2 = "abcdefghijklmnopqrstuvwxyz0123456789";
      String input3 = "a1b2c3 d4";
      String input4 = "Line 1\nLine2\nLine3\nLine4";
   }
}
```

4) Scanners are handy for reading input that is stored in a file. For this exercise, create a file called `"c:\\aaa\\numbers.txt"` that contains the following data:

```
      1.2   2.3  3.4  4.5
      2.0   3.0  4.0  5.0
      6.0   7.0  8.0  9.0
```


Read each line, convert the numbers in each line to doubles, and print each row of numbers and their total.

5) Write a program that processes command line arguments. The arguments are a mixture of numbers (`int`s and `double`s). Concatenate all the arguments into a single string. Scan the string, then print each number on a separate line and whether it is an `int` or a `double`.

Test your program with each of the following lists of arguments:

a) 1 2 3 4 5
b) 1.1 2.2 3.3 4.4
c) 1 2.9 3 4.9 5 6.9

6.1) Start with the code below and complete the `getInt` method. The method should prompt the user to enter an integer. Scan the input the user types. If the input is not an `int`, throw an `IllegalArgumentException`; otherwise, return the `int`.

```
import java.util.Scanner;

public class Throwing
{
   public static void main(String[] args)
   {
      int x = getInt();
      System.out.println(x);
   }

   public static int getInt()
   {
      // your code goes here
   }
}
```

6.2) Modify the program from Lab 7.6.1 so that `getInt` throws an `IOException` instead of an `IllegalArgumentException`. Modify the main program so that it catches and prints the `IOException`.

7.7) Write a program that prompts the user to enter an `int`. Print the integer the user enters. If the user doesn't enter an `int`, keep looping, prompting the user to enter the required `int`.

