Due:
Activity (in-lab): Monday, September 26, 2011 by the end of lab
Homework: Tuesday, September 27, 2011 by 11:59 PM on Web-CAT

Goals:
By the end of this activity you should be able to do the following:
- Understand the basics of switch statements and the ternery operator (homework)
- Understand the basics of do-while and for loops

Description:
In this activity you will create two classes. Temperatures will hold a set of integer values representing daily temperatures. TemperatureInfo will allow users to interact with the Temperatures class.

Directions:
Part 1: Temperatures: Method Stubs
- Create a class called Temperatures, which will hold a set of integer values representing daily temperatures.
- Add method stubs for the following methods**.**
    - The constructor takes an ArrayList of integer values
```
public Temperatures(ArrayList temperaturesIn) {

}
```
    - getLowTemp: takes no parameters; returns an integer value
    - getHighTemp: takes no parameters; returns an integer value
    - lowerMinimum: takes an int parameter; returns an integer value
    - higherMaximum: takes an int parameter; returns an integer value
    - toString: no parameters; returns a String

Part 2: Temperatures: instance variable, Constructor, getLowValue
- Add an instance variable with the name *temperatures* to your class that is of type ArrayList with generic type integer.
- In your constructor, set *temperatures* equal to *temperaturesIn*.
- In getLowValue, first return 0 if the ArrayList is empty:
```
if(temperatures.isEmpty()) {
   return 0;
}
```
- Now iterate through the entire list and find the lowest temperature:
```
for (int i = _____; i < __________; i++) {
   if (temperatures.get(i) < low) {
      low = temperatures.get(i);
   }
}
```

- Finally, return the lowest temperature:

```
return low;
```

Part 3: getHighValue

- In getHighValue, again return 0 if there are no temperatures in the ArrayList:

```
if(temperatures.isEmpty()) {
   return 0;
}
```

- This time, use a for-each loop to iterate through the list of temperatures to find the highest value.

```
int high = temperatures.get(0);
for (Integer currentTemp : temperatures) {
   if (______________ > high) {
      high = _______________;
   }
}

return high;
```

- Add code to the toString method to return a string containing the low and high temperatures (hint: make a method call to getLowValue and getHighValue):

```
public String toString() {
   return "Low: " + ____________()
      + "\r\nHigh: " + ____________();
}
```

Part 4: lowerMinimum & higherMaximum

- The lowerMinimum method takes an int value and returns the parameter if it is lower than the value returned by getLowerValue. Otherwise, it returns the return of getLowerValue.

```
public int lowerMinimum(int lowIn) {
   return lowIn < getLowValue() ? lowIn : getLowValue();
}
```

- The higherMaximum method takes an int value and returns the parameter if it is greater than than the value returned by getHigherValue. Otherwise, it returns the return of getHigherValue.

```
public int higherMaximum(int highIn) {
   return _______ ? ______________;
}
```

- Test your methods in the interactions pane:

```
import java.util.ArrayList;
ArrayList tempList = new ArrayList();
tempList.add(34);
tempList.add(52);
tempList.add(36);
```

```
    tempList.add(65);
    Temperatures temps = new Temperatures(tempList\);
    temps.getLowValue()
    34
    temps.getHighValue()
    65
    temps.lowerMinimum(33)
    33
    temps.lowerMinimum(35)
    34
    temps.higherMaximum(64)
    65
    temps.higherMaximum(67)
    67
```

Part 5: TemperatureInfo

- Create a class called TemperatureInfo with a main method. Declare and instantiate an ArrayList with generic type Integer called tempList. Declare and instantiate a Scanner object called userInput that reads from System.in.
- Create the following do-while loop that will add numbers to the list until the user enters a -1 input. Also create a Temperatures object with the ArrayList as input and print the temperatures object:

```
int tempInput = -1;
do {
   System.out.print("Enter a postive temperature (-1 to stop): ");
   tempInput = userInput.nextInt();
   if (tempInput > -1) {
      ________.add(tempInput);
   }
} while (tempInput > -1);

Temperatures temps = new Temperatures(___________);
System.out.println(_____________);
```

Run your program as below to test its output:

```
     ----jGRASP exec: java Temperatures

    Enter a postive temperature (-1 to stop): 45
    Enter a postive temperature (-1 to stop): 44
    Enter a postive temperature (-1 to stop): 12
    Enter a postive temperature (-1 to stop): 33
    Enter a postive temperature (-1 to stop): 24
    Enter a postive temperature (-1 to stop): -1
    Low: 12
    High: 45

     ----jGRASP: operation complete.
```

Homework

- Download EmployeeReviewer, Review, and employees.txt from the lecture slides on Chapter 6 (you'll have to upzip the lecture notes zip file). Run the program and ensure that it has the following output:

```
 ----jGRASP exec: java EmployeeReviewer

Enter the file name to be processed: employees.txt

Lane, Jane: Great
Smith, Bob: Average
Michaels, Gary: Superb

 ----jGRASP: operation complete.
```

- Modify the program so that it prints out the average of all employee's reviews (the average of all review objects:

```
 ----jGRASP exec: java EmployeeReviewer

Enter the file name to be processed: employees.txt

Lane, Jane: Great
Smith, Bob: Average
Michaels, Gary: Superb

Average of all scores: 2.9

 ----jGRASP: operation complete.
```

- Modify your overallPerformance method to return the String as it does now, but to also add "(has superb)" to the end if any of the scores are superb (do not include the quotes). You must keep the switch statement for full credit. HINT: you can add a local String variable before the switch statement and then modify the switch to set the string variable rather than return a value. You can then iterate through the scores and add (has superb) if any score is of value SUPERB.

```
 ----jGRASP exec: java EmployeeReviewer

Enter the file name to be processed: employees.txt

Lane, Jane: Great (has superb)
Smith, Bob: Average (has superb)
Michaels, Gary: Superb (has superb)

Average of all scores: 2.9

 ----jGRASP: operation complete.
```

* You may want to modify your file to test for conditions where (has superb) would not be present.