Sentinel Controlled Loops (Outsource: 5-3 – 5-7)

A sentinel controlled loop is a loop that continues to repeat until a specific value (the sentinel value) is entered by the user.

For example, the following code reads numbers from the keyboard and stores them in the array `list` until the user enters the *sentinel value* -999:

```
int[] list = new int[100];
int count = 0;

Scanner reader = new Scanner(System.in);
System.out.print("Enter a number: "); // Prompt the user for input
int n = reader.nextInt();             // Get a number from the keyboard
while ( n != -999)                    // The sentinel value is 999
{
   list[count++] = n;                 // Action to be performed

   System.out.print("Enter a number: "); // Prompt the user for input AGAIN
   n = reader.nextInt();              // Get another number from the keyboard
}
```

Comparing Objects

Primitive data types can be compared by using the standard comparison operators such as the = symbol or the < symbol. Using these symbols to compare objects, such as two Strings does not yield the results you are probably looking for. An object does not contain a value. Instead, it contains a reference to a memory location. Therefore, if you use the comparison operators to compare to objects, you are comparing their memory locations, not the data contained in them.

Comparing Strings (Outsource: 7-15)

The String class provides two different methods to compare Strings for equality.

Method Summary – String class	
`boolean`	**`equals`**`(Object anotherObject)` Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.
`boolean`	**`equalsIgnoreCase`**`(String anotherString)` Compares this `String` to another `String`, ignoring case considerations.

```
String str = "JVHS";
If ( str.equals("JVHS") ) /* if str contains "JVHS" */
    /* body of the if statement */
```

Comparing Strings For Inequality

To test a string for **inequality** (NOT EQUAL TO) place a *not* operator before the call to the string's equal method.

```
String str = reader.next();
while ( !str.equals("stop") )  /* loop while str DOES NOT contain "stop" */
{
    /*  process the variable str  */
    str = reader.next();
}
```

Lab 15 - ASSIGNMENT

Lab 15A - 70 points

OBJECTIVE

WAP that reads numbers from the keyboard until the user enters **0**. Store the numbers in an array of type int. Sort the array and output the smallest value, the largest value, and the median value.

NOTE: The Median is the "middle number" if there is an odd number of values (in a sorted list of numbers). If there is an even number of values then the Median is the average of the middle two numbers.

FIELD SUMMARY

- **int[ ] list** – an array of eight integers
- **int count** – the number of elements actually stored in the array.
- **int median** – the sum of all the numbers.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to `input`, and `output`.
- **input** – declare a `Scanner` object and read a series of integers from the keyboard storing them in an array. Read and store numbers until the user enters **0**.
- **median** – Arguments: an array of integers and the number of elements in the array. **Return:** an int. Sort the array and return the Median value.
- **output** – output the smallest, largest, and median values in the array.

SAMPLE KEYBOARD INPUT

Enter first integer (0 to quit): **12**
Enter an integer (0 to quit): **-3**
Enter an integer (0 to quit): **4**
Enter an integer (0 to quit): **8**
Enter an integer (0 to quit): **0**

SAMPLE OUTPUT

The smallest value is -3.
The largest value is 12.
The median value is 6.

Lab 15B - 80 points

OBJECTIVE

WAP that reads student's first names from the keyboard until the user enters **"done"**. Store the names in an array of type String. Sort the array and output the names and the number of names that were entered.

FIELD SUMMARY

- **String[ ] list** – an array of names
- **int count** – the number of elements actually stored in the array.

METHOD SUMMARY

- **main** – instantiate an object of your class. Make method calls to `input`, and `output`.
- **input** – declare a `Scanner` object and read student's first names from the keyboard until the user enters **"done"**.
- **process** – sort the array of names.
- **output** – output all the student's names and the number of names that were processed.

SAMPLE KEYBOARD INPUT

```
Enter a name: Sue
Enter a name: John
Enter a name: David
Enter a name: Mary
Enter a name: Henry
Enter a name: Phillip
Enter a name: Margaret
Enter a name: done
```

SAMPLE OUTPUT

```
David
Henry
John
Margaret
Mary
Phillip
Sue
7 names were entered.
```

Lab 15C - 90 points

OBJECTIVE

WAP to read a series of integers from the keyboard. Continue to read numbers until the user enters *-99*. Sum up all the numbers and calculate the average. Output the number of vales entered, the sum and the average. Do not process the control value (-99).

FIELD SUMMARY

- **int[ ] list** – an array of integer values of unknown size.
- **int count** – the number of elements stored in the array *list*.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to `input` and `output`.
- **input** – declare a **`Scanner`** object and read a series of integers from the keyboard. Continue reading numbers until the value -99 is entered. Prompt the user before every read statement.
- **sum** – Arguments: an array of integers. **Return:** an int. Calculate and return the sum of all the values in the array received as an argument.
- **average** – Arguments: two integers – sum and count. **Return:** a double. Calculate and return the average of all the numbers processed.
- **output** – Output the sum and average of all the numbers. Make method calls to `sum` and `average` from the println statements. Formate all doubles to show one decimal place with no trailing zeros.

SAMPLE KEYBOARD INPUT

Enter a number: **12**
Enter a number: **7**
Enter a number: **11**
Enter a number: **8**
Enter a number: **21**
Enter a number: **33**
Enter a number: **-99**

SAMPLE OUTPUT

The sum of the 5 numbers is 92.
The average of the 5 numbers is 15.3.

Lab 15D - 100 points

OBJECTIVE

WAP that reads the names of grocery store items and their prices and stores them in a set of parallel arrays. Read items and prices until the user enters *quit* for an item name. Output all the items purchased and their cost along with the total cost for all items purchased.

FIELD SUMMARY

- **String[ ] items** – a list of items purchased from the grocery store.
- **String[ ] prices** – the price of each item purchased from the grocery store.
- **int count** – the number of items and their prices read from the keyboard.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to input and output.
- **input** – declare a Scanner object and read items and prices from the keyboard until the user enters ***quit*** for an item name.
- **totalCost** – Arguments: an array of doubles representing the cost of each item purchased and an int representing the number of items purchased. **Return:** a double. Calculate and return the total cost for all the items purchased.
- **output** – A list of all the items purchased and their prices followed by the total cost. Format all doubles to look like money with no leading zeros.

SAMPLE KEYBOARD INPUT

```
Enter the name of an item: onion
Enter the price for onion: .29
Enter the name of the next item: milk
Enter the price for milk: 3.79
Enter the name of the next item: bread
Enter the price for bread: 1.29
Enter the name of the next item: potato
Enter the price for potato: .49
Enter the name of the next item: quit
```

SAMPLE OUTPUT

```
At the market you bought:
   onion for .29
   milk for 3.79
   bread for 1.29
   potato for .49
The total cost was 5.86
```