Announcements

- No School on Wednesday
  - Rescheduling normal Wednesday Office Hours to Tuesday (2p – 3p)

- Lab 1 is graded
  - Check Sakai for results and comments
Questions?

- Last Friday:
  - Even More Loops
    - Common Loop Bugs
    - Designing Loops
  - Debugging Techniques
Today in COMP 110

- Lecture: Arrays
- Lab 4
Familiar Example

- Sum a series of positive values, terminated by a negative value
  - Scanner keyboard = new Scanner(System.in);
  - double input;
  - double sum = 0;
  - do
  - {
    - input = keyboard.nextDouble();
    - if (input >= 0)
    - {
      - sum += input;
    - }
  - } while (input >= 0);
  - System.out.println(sum);
Basic Looping Algorithm

1. Read in value from user
2. While value is valid
   1. Process value into a composite value
   2. Read in value from user
3. Output composite value
Could we use that basic structure to output?

- Sum
- Mean
- Min/Max
- The values that are more than the mean
- Median
Alterations

- Could we use that basic structure to output?
  - Sum
  - Mean
  - Min/Max
  - The values that are more than the mean
  - Median

Yes

No
One possibility for mean

Suppose that we knew that we had 5 input values

Scanner keyboard = new Scanner(System.in);
double value1 = keyboard.nextDouble();
double value2 = keyboard.nextDouble();
double value3 = keyboard.nextDouble();
double value4 = keyboard.nextDouble();
double value5 = keyboard.nextDouble();

double mean = (value1 + value2 + value3 + value4 + value5) / 5;

if (value1 < mean)
    System.out.println(value1 + " is less than the mean");
else if(value1 > mean)
    System.out.println(value1 + " is more than the mean");
else
    System.out.println(value1 + " is equal to the mean");

if (value2 < mean)
    System.out.println(value2 + " is less than the mean");
// ...
What if we had 100 input values?

Scanner keyboard = new Scanner(System.in);
double value1 = keyboard.nextDouble();
double value2 = keyboard.nextDouble();
// ...
double value99 = keyboard.nextDouble();
double value100 = keyboard.nextDouble();

double mean = (value1 + value2 + /* ... */ value99 + value100) / 100;

if (value1 < mean)
    System.out.println(value1 + " is less than the mean");
else if (value1 > mean)
    System.out.println(value1 + " is more than the mean");
else
    System.out.println(value1 + " is equal to the mean");

if (value2 < mean)
    System.out.println(value2 + " is less than the mean");
// ...
Arrays to the Rescue

- An array is a collection of items of the same type
  - Can store any type (primitives, classes, arrays)

- They are like a list of variables, with a compact naming scheme
  - Each item in the array is an element

- Arrays are special kinds of objects in Java
  - Like a class type, not a primitive type
Declaring an Array

- Syntax
  - `Type[] arrayVar = new Type[Integer_Expression];`

- `Type` indicates the kinds of values the array can hold
  - Can be primitive, class, or other array types

- The value of `Integer_Expression` determines the number of elements the array can hold
Declaring an Array

- Declaration & Creation Example:
  - `int[] values = new int[5];`

- This is like creating 5 strangely named values and initializing them to zero
  - `values[0] = 0;`
  - `values[1] = 0;`
  - `values[2] = 0;`
  - `values[3] = 0;`
  - `values[4] = 0;`
Declaring an Array

- When the number of elements is constant, use a named constant
  - public static final int NUMBER_OF_READINGS = 42;
    
    double[] readings = new double[NUMBER_OF_READINGS];

- Can also read in the length at runtime
  - System.out.println("How many scores?");
    int numScores = keyboard.nextInt();
    int[] scores = new int[numScores];
Declaring an Array

- Can also assign a list of values directly to an array
  - `int[] scores = { 47, 52, 94 };`

- This is equivalent to
  - `int[] scores = new int[3];`
    `scores[0] = 47;`
    `scores[1] = 52;`
    `scores[2] = 94;`
Indexing

- Syntax
  - \texttt{arrayVar[Integer\_Expression]}

- These variables are known as indexed variables, subscript variables, array elements, or elements

- The value of \texttt{Integer\_Expression} determines which element is being accessed
Indexing

- Integer_EXPRESSION can be any integer expression
  - An integer: \texttt{values[3]}
  - An integer variable: \texttt{values[index]}
  - An expression that evaluates to int: \texttt{values[index * 3]}

- Can use array elements like any other variable
  - \texttt{values[3] = 68;}
  - \texttt{values[4] += 3;}
  - \texttt{System.out.println(values[16]);}
Indexing

- Where have we seen indexing before?
  - Strings, e.g. `myStr.charAt(5)`

- Like Strings, array indices always start at zero

- Strings are read-only, while arrays are read/write
Indexing

- Like Strings, arrays have a size (number of elements)
  - String: `myStr.length()` // method call, uses ()
  - Array: `myArray.length` // instance variable, no ()

- You cannot change the length of an array
  - `myArray.length` is read-only

- The value of `length` is equal to the number of elements in the array
  - `int[] values = new int[20]`
    // `values.length` is now 20
int[] scores = new int[5];
scores[0] = 68;
scores[1] = 73;
scores[2] = 57;
scores[3] = 102;
scores[4] = 94;

The array scores

scores.length is 5

Array Indices

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>68</td>
<td>73</td>
<td>57</td>
<td>102</td>
<td>94</td>
</tr>
</tbody>
</table>
Array Syntax Summary

- Declaration
  - $Type[] \ arrayVar1, \ arrayVar2, \ ...;$

- Array Creation
  - $arrayVar = new \ Type[\ Integer\_Expression];$

- Element Assignment
  - $arrayVar[\ Integer\_Expression] = \ Expression;$
Returning to the Example

Scanner keyboard = new Scanner(System.in);
double[] values = new double[100];
double sum = 0;
for (int i = 0; i < values.length; i++)
{
    values[i] = keyboard.nextDouble();
    sum += values[i];
}

double mean = sum / values.length;

for (int i = 0; i < values.length; i++)
{
    if (values[i] < mean)
        System.out.println(values[i] + " is less than the mean");
    else if (values[i] > mean)
        System.out.println(values[i] + " is more than the mean");
    else
        System.out.println(values[i] + " is equal to the mean");
}
For Each Loop

- Syntax
  - for (Type variable : Set)
    - Body_Statement

- Arrays can be used as Set
  - `double[] readings = new double[50];`
  - `// ...`
  - `// Fill in readings`
  - `// ...`
  - for (double reading : readings)
    - `{`
      - `System.out.println(reading);`
    - `}`

- Note that only the value of each element is accessible in this loop
  - Indices are not available
Returning to the Example Again

Scanner keyboard = new Scanner(System.in);
double[] values = new double[100];
double sum = 0;
for (int i = 0; i < values.length; i++)
{
    values[i] = keyboard.nextDouble();
    sum += values[i];
}

double mean = sum / values.length;

for (double value : values)
{
    if (value < mean)
        System.out.println(value + " is less than the mean");
    else if (value > mean)
        System.out.println(value + " is more than the mean");
    else
        System.out.println(value + " is equal to the mean");
}
Array Gotchas: Assignment

- Need to assign an array to a variable before giving its elements values
  
  ```
  int[] scores;
  scores[2] = 5; // Compiler Error
  ```

- The code will not compile
Array Gotchas: Assignment

- Assigning existing arrays to other variables does not create new copies
  - The second variable will refer to the same instance as the original variable

- Consider this example:
  ```java
  int[] myArr1 = new int[5];
  myArr1[2] = 5;
  int[] myArr2 = myArr1;
  myArr2[2] = 3;
  System.out.println(myArr1[2]); // Prints 3
  ```

- Why?
  - Arrays are like class types
  - We’ll cover more on this in lectures about classes
Array Gotchas: Bounds

- Need to stay within the bounds of the array
  - `int[] scores = new int[7];
  scores[7] = 5; // Runtime Error`

- The code will compile, but will produce an Array Index Out of Bounds error at runtime
Array Gotchas: Equality

- Like Strings (and any other class typed object), don’t use == to test equality
  - It will not behave like == does for primitives

- Need to write your own code if array equality testing is required
Sorting

- Given an array of numbers, sort the numbers into ascending order.

- For example

  4  7  3  9  6  2  8

becomes

  2  3  4  6  7  8  9
Selection Sort

- **Pseudocode**
  - ``` for (index = 0; index < length; index++)
      {
        // 1. Find index of smallest value of array
        //    between index and end of array
        // 2. Swap values of current index and the
        //    index with the smallest value
      }```

- **Java code is available on the class website (SelectionSort.java)**
Selection Sort Exercise

- Fun Exercise Time!
for (index = 0; index < length; index++)
{
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value
}

Indices

0  1  2  3  4  5  6
Selection Sort Exercise

```c
for (index = 0; index < length; index++)
{
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value
}
```

Indices

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort Exercise

```c
for (index = 0; index < length; index++)
{
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value
}
```

Indices

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Selection Sort Exercise

```java
for (index = 0; index < length; index++)
{
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value
}
```

Indices

<table>
<thead>
<tr>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
Selection Sort Exercise

```c
for (index = 0; index < length; index++)
{
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value
}
```

Indices

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>index</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7/2/2012
Selection Sort Exercise

for (index = 0; index < length; index++)
{
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value

    Indices

    0 1 2 3 4 5 6

    index
Selection Sort Exercise

```java
for (index = 0; index < length; index++)
{
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value
}
```

Indices

0  1  2  3  4  5  6

index
Selection Sort Exercise

```java
for (index = 0; index < length; index++) {
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value
}
```

Indices

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>

index
Selection Sort Exercise

for (index = 0; index < length; index++)
{
    // 1. Find index of smallest value of array
    //    between index and end of array
    // 2. Swap values of current index and the
    //    index with the smallest value

Indices

0 1 2 3 4 5 6
Sorting

- Other sorting algorithms exist
  - Bubble Sort
  - Insertion Sort
  - Quick Sort
  - Merge Sort

- Each uses different algorithms and has different performance characteristics
Multidimensional Arrays

- Arrays having more than one index tend to be useful
  - Tables
  - Grids
  - Matrices
  - Images
Multidimensional Arrays

- Recall 1-D Array Declaration Syntax
  
  ```c
  Type[] arrayVar = new Type[Integer_Expression];
  ```

- Arrays are types themselves
2-D Arrays

- Declaration
  - Type[][] arrayVar1, arrayVar2, ...;

- Array Creation
  - arrayVar = new Type[Int_Expr1][Int_Expr2];

- Element Assignment
  - arrayVar[Int_Expr1][Int_Expr2] = Expression;
2-D Arrays & Loops

- Use a single loop on 1-D arrays
  ```java
  int[] scores = { 4, 6, 12, 2, 6 };
  for(int i = 0; i < scores.length; i++)
  {
      System.out.println(scores[i]);
  }
  ```

- Use nested loops on 2-D arrays
  ```java
  int[][] table = new int[4][5];
  for (int row = 0; row < table.length; row++)
  {
      for (int column = 0; column < table[row].length; column++)
      {
          System.out.println(table[row][column]);
      }
  }
  ```
2-D Array Lengths

- int[][][] table = new int[4][5];
  - table.length is the number of rows: 4
  - table[row].length is the number of columns: 5
n-D Arrays

- Declaration
  - Type[ ]...[ ] arrayVar1, arrayVar2, ...;
  
- Array Creation
  - arrayVar = new Type[Int_Expr1]...[Int_ExprN];

- Element Assignment
  - arrayVar[Int_Expr1]...[Int_ExprN] = Expression;
Ragged Arrays

- Equivalent Statements (normal, not ragged arrays):
  - `double[][][] table = new double[2][3];`
  - `double[][][] table = new double[2][];
    table[0] = new double[3];
    table[1] = new double[3];`

- The sub-arrays need not all be the same size (ragged array), for example:
  - `double[][][] table = new double[2][];
    table[0] = new double[3];
    table[1] = new double[5];`
Arrays Summary

- Arrays are class-like variables
  - Contain a list of values of all the same type
  - These values are indexed using integer expressions within [ ]’s: myArr[index]
    - Indices Range: 0 <= index < myArr.length

- Can iterate through the values of an array using a loop
  - Use nested loops for multidimensional arrays

- Sorting is the process of organizing the values of an array into ascending order
Questions?
Logistics

- Next:
  - Lab 4

- Tomorrow:
  - Introduction to Classes
  - Lab 5