CS139 Algorithm Development
Activity 11A - Introduction to Arrays

Objectives
At the end of this exercise, students will:
- be able to determine the size and content of an array
- be able to allocate and initialize an array
- be able to write loops to perform simple array processes

Terminology
- Array: An ordered collection of data values that have a common name and data type
- Array Type: An array extension of a data type (native type or reference type), indicated by square brackets. For example, int[] is type *integer array*, String[] is type *String array*.
- Array Element: A single member of an array, designated by a subscript
- Subscript (index): The position (zero-based) of an individual array element; The first element of an array has subscript 0 (e.g., myArray[0]), and the last has subscript (length - 1)
(e.g., myArray[myArray.length - 1]). Note that *.length* is a built-in attribute of every array.
- ArrayIndexOutOfBoundsException: An exception (run-time error) caused by trying to access an array element that does not exist. For example trying to access myArray[6] when there are only 6 elements in the array (indexed 0-5), or trying to access myArray[-1].
- Initializer List: A list of values that is used to initialize the elements of an array.

Instructions
Answer each of the numbered questions below.

1. Hand in a record of your completed answers and Java solutions.

Part 1 - What is an array?

An array is an ordered collection of data, all with the same name and type. Array allocation is done with an array type, which is any data type followed by "[]". By comparison, a regular variable (called a "scalar" variable) contains only a single value.

```
int x;      // an integer (scalar)
int[] x;    // an integer array reference variable
```

Array variables are reference variables, meaning that they refer to an array object (just as String reference variables refer to a String object). This means that an extra step is required in order to actually allocate the array itself, in addition to the reference variable;

```
int[] x = new int[10];  // allocate an array of 10 integers
String[] s = new String[3];  // allocate an array of 3 Strings
```

1. Given that the number of bytes required for storing a scalar value of type char, int, and double is (2, 4, 8), respectively, determine how much memory each of these arrays will require.

```
a.  int[] x = new int[100];
b. char[] chs = new char[250];
c. final int SIZE = 125;
   double costs = new double[SIZE * 4];
```

Part 2 - Array initialization & subscripts

The elements of an array can be assigned an initial value individually or through enumeration. An individual assignment is done by specifying the (zero-based) position of the element in the array, called its "subscript". Subscripts are written in square brackets.

```
x[5] = 0;   // assign 0 to the 6th element of the array x
str[4] = "";   // assign "" to the 5th element of str
```

The elements of an array can also be specified by enumeration, which is an alternative way of allocating an array. Rather than specifying the size of the array, enumeration specifies the exact contents using an initializer list, which is contained in braces ({}).

```
int[] x = {3, 5, 7, 2, 1};  // allocate an array of 5 integers
// allocate an array of 3 Strings
String[] str = {"James", "Madison", "University");
```

1. Write the statements to allocate these arrays:

a.

3.23	1.52	4.23	32.5	2.45	5.23	3.33

b.

0	14	1024	127	5521	3

2. What is the value of each of the following expressions?

```
int[] a = {3, 6, 15, 22, 100, 0};
double[] b = {3.5, 4.5, 2.0, 2.0, 2.0};
String[] c ={"alpha", "beta", "gamma"};
```

a. a[3] + a[2]
b. b[2] - b[0] + a[4]
c. c[1].charAt(a[0])
d. a[4] * b[1] <= a[5] * a[0]

Part 3. Arrays and loops

The real power of arrays in programming languages comes from the ability to process arrays using iteration (loops). This allows a single process to be applied to multiple elements of an array.

The standard form for an array iteration is as follows (assuming "array" is the array variable):

```
for (int i = 0; i < array.length; i++) {
   ... process array[i] ...
}
```

For example:

```
// set all of the elements of x to -1
double[] x = new double[100];
for (int i=0; i<x.length; i++) {
   x[i] = -1;
}

// sum the elements of scores
int[] scores = {....};
int sum = 0;
for (int i=0; i<scores.length; i++) {
  sum = sum + scores[i];
}
```

1. Show the value of *array* and *accumulator* after the following iterations.

```
      int[] array = {5, 26, 13, 12, 37, 15, 16, 4, 1, 0};
      int accumulator;
      accumulator = 0;
      for (int i = 0; i < array.length; i++) {
        if (array[i]% 2 == 1 && i+1 < array.length) {
          accumulator = accumulator + array[i + 1];
        }
}
```

Part 4 - Coding

1. Complete the coding for this method.

```
/**
 * Return a new array containing the pairwise maximum value of
 * the arguments. For each subscript, i, the return value at [i]
 * will be the larger of x[i] and y[i]. If x and y do not have the
 * same length, return null.
 *
 * @param x an array of double values
 * @param y an array of double values
 * @return the pairwise max of x and y; null if x.length != y.length
 */
public static double[] pairwiseMax(double[] x, double[] y) {
```

Things to remember about arrays:

Feature	Syntax	Example
Declaration	*data-type* [] *variable-name*;	int [] gradeArray; Card [] deck;
Declaration (alternative)	*data-type variable-name*[];	int gradeArray []; Card deck [];
Initializer list	*data-type [] list* = { *value, value, ... value*}	String [] names = { "bob", "ann", "amy", "sue", "sam"};
Instantiation	*new data-type*[*int-value*]	tArray = new int [25]; eck = new Card [cardCount];
Instantiation and initialization	*new data-type*[] { *value, value, ... value* }	names = new String [] { "bob", "ann", "sue", "sam" }; return new int [] {0};

Referring to the array as a whole	*var-name*	return deck;
		intArray = grades;
		
		array2.equals(array1);
		(What do you think this does compares?)
		
		arrLength = names.length;

Specialized Arrays methods		java.util.Arrays.fill(grades, 100);
		java.util.Arrays.equals(section1, section2);
		java.util.Arrays.sort(names);
		java.util.Arrays.toString(names)

Referring to array elements	*var-name*[*int-value*]	return deck[0];
		intArray[ii] = grades[jj];
		rank = deck[ii].getRank();
		while (intArray[ii] == 5)