Due: Activity (in-lab) Monday, October 8, 2012 by the end of lab

Goals:

By the end of this activity you should be able to do the following:

➢ Understand the basics of instantiating arrays and assigning / accessing array elements.
➢ How to iterate through arrays using loops.

Description:

In this activity, you will create a class called Scores that will hold an array of numerical values and provide methods that allow users to interact with the Scores class.

Directions:

Part 1: Scores - method stubs

- Create a class called Scores.
    - o Add a constructor that has a parameter declared as an array of int values.

```
public Scores(int[] numbersIn) {

}
```

- Add method stubs for the following methods. **The first one is given; do the rest on your own**.

    - o findEvens: no parameter, returns an array of ints (all of the even-valued scores)

```
public int[] findEvens() {
    return null;
}
```
> An array is an object, so null is a placeholder return.

    - o findOdds: no parameter, returns an array of ints (all of the odd-valued scores)

    - o calculateAverage: no parameters; returns a double (the average of all scores)

    - o toReverseString: no parameters; returns a String (all scores in reverse order)

    - o toString: no parameters; returns a String (all scores)

Compile Scores and run the following in interactions. **Do not continue until your program compiles and the following code runs without error in interactions.**

```
Scores s = new Scores(null);
int[] e = s.findEvens();
int[] o = s.findOdds();
double avg = s.calculateAverage();
```

Part 2: Scores - instance variable and constructor

- Add an instance variable with the name *numbers* to your class that is an array of int values:

```
int[] numbers;
```

- In your constructor, add code that will set the value of *numbers* to *numbersIn*. **You access the entire array object using its variable name with no brackets.**

```
numbers = _____;
```

- Compile Scores. In the interactions pane set up an array of int values using an initializer list and send it to the constructor of scores:

```
    int
[] nums = {2, 5, 8};
    Scores s = new Scores(nums);
```

Open the Scores object *s* on the Workbench in the upper left corner and open the instance variable numbers. You should be able to see your values.
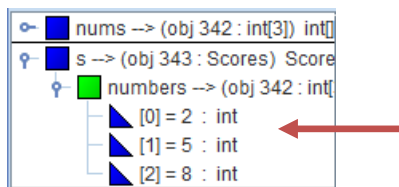
Open a viewer on *s* by dragging the blue square from the debug tab.

Part 3: Scores – toString and toReverseString methods

- The toString method will create a local String and then concatenate all of the values of numbers to the String.

```
public String toString() {
    String output = "";

    for (int i = 0; i < numbers.length; i++) {

    }

    return output;
}
```

- The variable i iterates from 0 the length of numbers - 1. Within the for loop, add the number at each index to the output:

```
output += numbers[i] + "\t";
```

- Check the toString return in interactions:

```
    int[] nums = {2, 5, 8};
    Scores s = new Scores(nums);
    s
    2    5    8
```

- The toReverseString method will be exactly the same as toString, but will iterate from the length of numbers - 1 to 0.

```
String output = "";

for (int i = numbers.length - 1; _____; _____) {
    output += numbers[i] + " ";
}

return output;
```

Compile Scores and run the following code in the interactions pane. **Do not continue until the following code runs without error in interactions.**

```
int[] nums = {2, 5, 8};
Scores s = new Scores(nums);
s.toReverseString()
8  5  2
```

Part 4: Scores – findEvens and findOdds methods

- There are two parts to the findEvens method. First, count the number of evens in the array:

```
int numberEvens = 0;

for (int i = 0; i < numbers.length; i++) {
    if (_____ % 2 == 0) {
        numberEvens++;
    }
}
```

- You will then need to create an array with the appropriate length to store the number of even numbers.

```
int[] evens = new int[numberEvens];
```

- Add the even numbers to the evens array. In the following loop, *i* represents the current index of numbers and *count* is the current index of evens.

```
int count = 0;
for (int i = 0; i < numbers.length; i++) {
    if (numbers[_____] % 2 == 0) {
        evens[_____] = numbers[_____];
        count++;
    }
}
return evens;
```

- Compile Scores and test the return of findEvens. The array return does not have a toString representation including the value at each index, so you will use a method from the Arrays class.

```
import java.util.Arrays;
int[] nums = {2, 5, 8, 1, 10};
Scores s = new Scores(nums);
int[] evens = s.findEvens();
evens // toString output of an array object (will vary)
[D@5abb7465
Arrays.toString(evens)
[2, 8, 10]
```

- **Create the findOdds method. It will perform the exact same function as findEvens, but it will find all odd numbers in the array (numbers that are not divisible by 2).**

- Test findOdds in the interactions pane. **Do not continue until your output is correct.**

```
import java.util.Arrays;
int[] nums = {1, 5, 8, 3, 10};
Scores s = new Scores(nums);
Arrays.toString(s.findOdds())
[1, 5, 3]
```

Part 5: Scores - calculateAverage method
- First, find the sum of all values in the numbers array.

```
int sum = 0;

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

- Return the sum divided by the number of elements in the array. Remember that sum and arrays.length are both integers.

```
return _____ / _____;
```

Compile NumberOperations and run the following code in the interactions pane.

```
int[] nums = {2, 5, 8, 7, 19};
Scores s = new Scores(nums);
s.calculateAverage()
8.2
```

Your GTA will ask you to demonstrate all methods in the interactions pane with a different set of values than shown above.