Final Exam Review Questions
CSE110 - Arizona State University

1. What are the indexes for the first and last positions of array called x?

   a. `x[0]` and `x[x.length]`

   b. `x[0]` and `x[x.length - 1]`

   c. `x[1]` and `x[x.length]`

   d. `x[1]` and `x[x.length - 1]`

   The answer is choice b, the first index is `x[0]`, and the last is `x[x.length-1]`.

2. Immediately after instantiating a new array of primitives (`int`s, `double`s, etc.), what fills the array? What about an array of objects?

   If the array is of primitive types, then the default value of the primitive fills the array (such as 0 for `int`, 0.0 for `double`, `false` for `boolean`, etc.). If the array is of objects, then `null` fills the array.

3. What happens when you try to access an array element past the end of an array? An exception of type `ArrayIndexOutOfBoundsException` is thrown.

4. Instantiate three arrays called `x, y` and `z` of types `int, String`, and `BankAccount`, respectively, all of size 10.

```
int[] x = new int[10];
String[] y = new String[10];
BankAccount[] z = new BankAccount[10];
```

5. What is method overloading? Method overloading refers to when there are two methods in the same class such that they have the same name, but have different signatures. Remember, the signature of a method is the name of the method + the number of parameters in the method + the types of those parameters.

   For example, these two methods are overloaded (different signatures because they have different types for parameters):

```
int calc(double n1, int n2) { ... }
int calc(int n1, int n2) { ... }
```

6. Use the following full array, x: {4, 8, 5, 1, 6, 3, 2}

   a. What value is given by `x[1]`? 8

   b. What value is given by `x[6]`? 2

   c. What value is given by `x[7]`? `ArrayIndexOutOfBoundsException` is thrown.

   d. What value is given by `x.length`? 7

7. Write a `for`-loop to double each element in the array x given in question 6 above.

```
for (int i=0; i < x.length; i++) {
     x[i] *= 2;
}
```

8. What is a `static` variable? What is a `static` method? A `static` variable belongs to the class, as well as a `static` method - they are not bound to instances. For example, if I instantiate 100 instances of a class `A` that has a static variable `num`, then there is only one `num` in existence, since it is bound to the class `A`, and not each individual instance of `A`. Questions 9 and 10 address this.

9,10 Use the following code to answer the parts of question 9,10:

```
public class AmazingClass {
     private static int number;
     public AmazingClass(int a) {
          number = a;
     }
     public int twice() {
          number*=2;
          return number;
     }
}
```

9. What is the value of `number` after the following statements? (For each part, assume the preceding parts it have already been executed.)

   a. `AmazingClass ac1 = new AmazingClass(3);` 3

   b. `AmazingClass ac2 = new AmazingClass(7);` 7

   c. `ac1.twice();` 14

   d. `ac2.twice();` 28

10. Using the code from question 9, how many copies of the variable `number` exist after I instantiate 374 different `AmazingClass` objects? As hinted from Question 8, there is only one copy.

11. What is the meaning of each of `public, static, final void, main`, and `String[] args`?

    - `public` refers to a visibility modifier in which anyone can see this item.
    - `static` means that the item is bound to the class in which it is, and not to the instance (i.e. there is only one copy of this item in existence).
    - `final` Once a value has been assigned to a final variable, it always contains the same value and can not be changed.
    - `void` means literally "nothing" - for methods, in this case, this method does not return anything.
    - `main` means the main method (the entry point to the program).
    - `String[] args` is a `String` array that is all of the arguments that are passed to the `main` method from the command line. For example, if I compile `A.java` with `javac A.java`, when I run the program, I can pass arguments: `java A 100 200 300`, and `args` will then be the array filled with these inputs as `Strings`.

12. Given the array from question 6, write the code fragments requested. You don?t need to write them in methods, just assume you?re working within an established main method.

    a. Write code to store the largest number in the array into a variable called `max`.

```
int max = x[0]; // have to assume x is not empty
for (int i=1; i < x.length; i++) {
     if (max < x[i]) {
          max = x[i];
     }
     // Alternatively without if statement: max = Math.max(max, x[i]);
}
```

b. Write code to count how many numbers in the array are strictly larger than four, and store that total in a variable called `total`.

```
int total = 0;
for (int i=0; i < x.length; i++) {
     if (x[i] > 4) {
          total++;
     }
}
```

c. Write code to print out every other element in the array separated by tabs. I will assume that this means start at index 0, and then index 2, 4, etc. (all the even indices). If we mean odd indices, change `i=0` to `i=1`.

```
for (int i=0; i < x.length; i+=2) {
     System.out.print(x[i] + "\t");
}
```

d. Write code to shift each number one place to the right. There will be two copies of the first element when you're done with this.

```
for (int i=x.length-1; i > 0 ; i--) {
        x[i] = x[i-1];
}
```

e. Write code to print the contents of the array in reverse order, one element for each line.

```
for (int i=x.length-1; i >= 0; i--) {
     System.out.println(x[i]);
}
```

13. Circle the valid method headings assuming they are written inside a class named `SomeClass`.

- `public void Void( )` Valid (believe it or not)
- `public String string( int n )` Valid
- `public double void f2( )` Invalid, because there are two return types.
- `public BankAccount bankAccount( )` Valid
- `public double sum( int left, right )` Invalid because the 2nd parameter does not have a type.

14. Use the following array to answer parts of the question: `{Mike, Betsy, Aaron, Steven, Doug, Pat, Elise}`.

a. Write the contents after each step of selection sort (alphabetical).
The algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right at the front (left) of the list, and the sublist of items remaining to be sorted that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest element in the unsorted sublist, exchanging it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right. If the input has N elements, the algorithm will have N-1 passes.

Here are the results (first line is original array):
{Mike, Betsy, Aaron, Steven, Doug, Pat, Elise}
{Aaron, Betsy, Mike, Steven, Doug, Pat, Elise}
{Aaron, Betsy, Mike, Steven, Doug, Pat, Elise}
{Aaron, Betsy, Doug, Steven, Mike, Pat, Elise}
{Aaron, Betsy, Doug, Elise, Mike, Pat, Steven}
{Aaron, Betsy, Doug, Elise, Mike, Pat, Steven}
{Aaron, Betsy, Doug, Elise, Mike, Pat, Steven}

b. Write the contents after each step of insertion sort (alphabetical).
Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain. If the input has N elements, the algorithm will have N-1 passes.

Here are the results (first line is original array):
{Mike, Betsy, Aaron, Steven, Doug, Pat, Elise}
{Betsy, Mike, Aaron, Steven, Doug, Pat, Elise}
{Aaron, Betsy, Mike, Steven, Doug, Pat, Elise}
{Aaron, Betsy, Mike, Steven, Doug, Pat, Elise}
{Aaron, Betsy, Doug, Mike, Steven, Pat, Elise}
{Aaron, Betsy, Doug, Mike, Pat, Steven, Elise}
{Aaron, Betsy, Doug, Elise, Mike, Pat, Steven}

15. Use the sorted list and use a binary search to look for Mike in the list. Show all the names that are going to be compared with Mike before it finds it, and repeat the same process looking for Cathy, which is not in the list: `{Aaron, Betsy, Doug, Elise, Mike, Pat, Steven}`.
The order for Mike is:
Middle = (first + last) /2 = (0+6)/2 = 3 → Mike ⟩ Elise
Middle = (first + last) /2 = (4+6)/2 = 5 → Mike ⟨ Pat
Middle = (first + last) /2 = (4+4)/2 = 4 → Mike == Mike
reports that Mike is in the array

The order for Cathy is:
Middle = (first + last) /2 = (0+6)/2 = 3 → Cathy ⟨ Elise
Middle = (first + last) /2 = (0+2)/2 = 1 → Cathy ⟩ Betsy
Middle = (first + last) /2 = (2+2)/2 = 2 → Cathy ⟨ Doug
reports that Cathy is not in the array.

16. Write class `LittleStatistician` to maintain two descriptive statistics: `count` and `average`. Write method `add` to add numbers to the collection. Write methods `count` and `average` to return the correct values. Also, write method `toString` to return all elements in the collection as a `String` (see output below). You must use an array instance variable to store the elements. Let the capacity be 5 (kept small here for demonstration purposes). When an attempt is made to add to a `LittleStatistician` object when the array is full (trying to add a 6th or 7th number), print a message stating the number could not be added. The following code must generate the output shown below.

```
LittleStatistician tests = new LittleStatistician( );
tests.add( 80.0 );
tests.add( 70.5 );
tests.add( 75.0 );
tests.add( 82.5 );
tests.add( 99.5 );
tests.add( 65.0 );
tests.add( 52.0 );
System.out.println( "Average: " + tests.average( ) );
System.out.println( "Count: " + tests.count( ) );
System.out.println( tests.toString( ) );
```

The output is:

```
Could not add 65.0
Could not add 52.0
Average: 81.5
Count: 5
```

```
public class LittleStatistician2 {
     // instance variables
     private int my_count;
     private double[] my_data;

     // constructor
     public LittleStatistician2() {
          my_count = 0;
          my_data = new double[5];
     }

     public void add(double newNumber) {
          if( my_data.length > my_count) {
               my_data[my_count] = newNumber;
               my_count++;
          } else {
               System.out.println( "Could not add " + newNumber );
          }
     }

     public int count() {
          return my_count;
     }

     public double average() {
          double sum = 0.0;
          for(int j = 0; j < my_count; j++)
               sum += my_data[j];
          return sum / my_count;
     }

     public String toString() {
          String result = "[";
          for(int j = 0; j < my_count - 1; j++)
               result += my_data[j] + ", ";
          return result + my_data[my_count-1] + "]";
     }
}
```

17. In order to Swap the values of two integers (A and B), we do not need extra integer.
    A)true B)false
    The answer is false, you need a third integer to swap the values of A and B.

18. write a line of code to initialize an 2D array of integers called A with the following values. Afterward what is the value of A[1][2] and A[0][3]?

    1 2 3 4
    5 6 7 8

```
int[][] A = {{1, 2, 3, 4},{5, 6, 7, 8}};
A[1][2] = 7       A[0][3] = 4
```

19. If we have `<String S = "equanimity">`, what is the output of `<S.substring(1,5)>` and `<S.substring(0,3)>`?

    a. "quani" and "equa"

    b. "quani" and "equ"

    c. "quan" and "equa"

    d. "quan" and "equ"

    Answer is choice d. Recall that in `substring` method, `beginIndex` is inclusive and `endIndex` is exclusive.

20 what is the difference between primitive types and objects in Java? Give an example of each of them.
Primitive types are predefined by the language and is named by a reserved keyword. The eight primitive data types supported by the Java programming language are `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `char`.
Objects are entities that have state and behavior. Usually state is maintained with variables and behavior is maintained by methods inside the object. Example of objects are arrays and classes.