1/18

JRE (Java Runtime Environment)

- Used to provide runtime environment
- physical implementation of JVM
- Contains set of libraries + other files that JVM uses at runtime

JVM

- abstract machine
- specification that provides runtime environment in which Java bytecode can be executed
- loads, verifies, and executes code

Arrays

Array Declaration

`<type>[] <name> = new <type>[<length>];` Ex: `int[] numbers = new int[10];`
Length can be any non-negative integer expression Ex: `int [] data = new int[x % 5 + 2];` Each element gets a "zero-equivalent" value

Accessing Elements

Access: `<name>[<index>]` Modify: `<name>[<index>] = <value>;`

Limitations of Arrays

- Cannot resize an existing array
- Cannot compare arrays with == or `equals`

```
int [] a1 = {42, -7, 1, 15};
int [] a2 = {42, -7, 1, 15};
if (a1 == a2) {}    // false!
if (a1.equals(a2))  // false!
```

- Does not know how to print itself

Arrays Class

- Class `arrays` in package `java.util` has useful static methods for manipulating arrays
- Syntax: `Arrays.<methodname>(<parameters)`

Arrays.toString

- Accepts an array as a parameter and returns a String representation of its elements

Reference Semantics

- **Value semantics**: Behavior where values are copied when assigned, passed as parameters, or returned
  - All primitive types in Java use value semantics
  - When one variable is assigned to another
- **Reference semantics**: Behavior where variables actualy store the address of an object in memory
  - When one variable is assigned to another, the object is not copied; both variables refer to the same object
  - Modifying the value of one variable will affect others

```
int[] a1 = {4, 15, 8};
int[] a2 = a1;
a2[0] = 7;
System.out.println(Arrays.toString(a1));    // [7, 15, 8]
```

- Arrays and objects use reference semantics. Why?
  - *Efficiency*. copying large objects slows down a program
  - *Sharing*. its useful to share an object's data among methods
- When an object is passed as a parameter, the object is not copied. The parameter refers to the same object
  - If the parameter is modified, it *will* affect the original object

Array Stuff

```
public class ArrayStuff {

public static void main(String[] args) {
  int[] a = {1, 2, 3, 4, 5};
  int i = 2;
  int j = 4;
  swap(a, i, j);
  System.out.println(Arrays.toString(a));     // {1, 2, 5, 4, 3}
  System.out.println(i);      // 2
  System.out.println(j);      // 4
}

// Swap the elements at index i and index j in the array a
// Assume i and j are within the bounds of array a
public static void swap(int[] a, int i, int j) {
  int tmp = a[i];
  int[] b = a;    // output remains the same due to reference semantics
  b[i] = a[j];
  b[j] = temp;
  i = 123432;     // wont be passed back to original i because of value
```

```
semantics
}
}
```

Software Testing

- Dynamic approach for checking software correctness
- Run code for some inputs, check outputs
- Checks correctness for some executions
- Main Questions
    - Test-input generation (generate inputs)
    - Test oracles (check outputs)
    - Test automation (run at scale)

    Testing Concepts

- Test case -- An execution of the software with a given input
- Test oracle

JUnit

```
import org.junit.Test;
import static org.junit.Assert.*;
```

```
public class myTest { @Test public void test1() { int x = 0; assertEquals(0, x); } }
```