Study questions set #6

1. In your own words explain the difference between “call by reference” and “call by value.”
2. In your own words explain the difference between static binding and dynamic binding (covered on July 10)
3. Implement the swap method below, without using the **new** keyword. The input parameter is the memory address of an array with two elements. Once the method returns, the positions of those two values must be interchanged in the original array. For example, the following snippet should produce the output indicated in the comments:

```
int[] test = {1, 2};
System.out.println(Arrays.toString(test)); //Output: [1, 2]
swap(test);
System.out.println(Arrays.toString(test)); //Output: [2, 1]
…
public static void swap(int[] twoVals) {




}
```

4. Implement the rotateLeft method below, which is a generalization of swap to arbitrary length. It will rotate every element one position to the left (the element at the front will move to the back). For example, the following snippet should produce the output indicated in the comments:

```
int[] test = {1, 2, 3};
System.out.println(Arrays.toString(test)); //Output: [1, 2, 3]
rotateLeft(test);
System.out.println(Arrays.toString(test)); //Output: [2, 3, 1]
```

Again, do not use the **new** keyword.

```
public static void rotateLeft(int[] manyVals) {









}
```

5. Write the standard equals method, which accepts a parameter of type Object, for the abridged MyVector class below (covered on July 10).

```
public class MyVector {

	double[] components;

	public MyVector(double[] components) {
		this.components = components;
	}

	public boolean equals(Object other) {









	}

}
```

6. Consider the Building interface below:

```
import java.awt.Color;

public interface Building {

	/**
	 * Changes the color of the building to the given parameter
	 * @param color The new color of the building.
	 */
	public void paint(Color color);

	/**
	 * Returns a new object identical to this one.
	 * @return The copy
	 */
	public Building copy();

}
```

Buildings might ***not*** be immutable. Implement the Street class below, which contains an array of Buildings. This class must be immutable.

```java
import java.awt.Color;

public final class Street {

	private final Building[] blds;

	/**
	 * Constructs a new Street with buildings identical
	 * to the input parameter.
	 * @param blds
	 */
	public Street(Building[] blds) {









	}

	/**
	 * Returns a copy of this Street's buildings.
	 * @param blds
	 */
	public Building[] getBlds() {









	}
	/**
	 * Returns a new Street identical to this one,
	 * except with all buildings painted a new color.
	 * @param color
	 * @return
	 */
	public Street paintAll(Color color) {









	}

}
```