1. Define the following terms: “instance variable”, “static variable”, “instance method”, “static method”, ”constructor”, “primitive type” (What are the eight primitive types?), “subclasses of the Object class” (Are there any other classes?)

2. In the code at the end of this document, identify:
   (i) any instance variables
   (ii) any static variables
   (iii) any instance methods
   (iv) any static methods
   (v) any constructors

3. On PCRS, you learned many properties of the String. Which of those properties are properties of all subclasses of Object? Are any of those properties specific to instances of String and not other classes? If so, which ones?

4. Write a main method that creates an instance of each class from the Supplementary Code, and also an instance of DirectedGraphNode that is of type GraphNode. Try calling each of the methods through each instance. When do the lookup rules apply? Does DirectedGraphNode inherit any variables or methods from GraphNode? When can you use casting to access the methods from the subclass class? super class? When you call getter methods, do they return values of variables in the same class or a different class? Write equals methods for each of these classes.

5. In general, what is the toString() method supposed to do? What is the equals() method supposed to do? Why do we usually want to override them in a subclass?

6. What is type-casting?
   If ClassA is a class and ClassB extends ClassA, consider the following code:

   ```java
   ClassA ob1 = new ClassA();
   ClassB ob2 = new ClassB();
   ClassA ob3 = new ClassB();
   ```

   Can you think of a reason to type-cast ob2 or ob3 as an instance of ClassA? If so, what is the syntax for this? Is it possible to type-cast any of ob1, ob2, or ob3 as an instance of ClassB? Why or why not?
7. **Integer** is the class associated with the primitive type **int**. When we use the **==** operator on primitive types, Java compares the values of the primitives. But for subclasses of **Object**, **==** compares the memory addresses that each instance points to. What will happen if we try to compile the following code? After fixing it so that it compiles, what will the output be?

```java
int w = 5;
int x = 5
Integer y = new Integer(5);
Integer z = new Integer(5);
System.out.println(w == x);   // What is wrong with this line?
System.out.println(w == y);
System.out.println(y == z);
```

8. Consider the constructors for **EventSpace**. If we change the number of arguments in the constructor, will the code compile without also changing the constructor in **LectureHall**? If not, why not?

9. If we include a second (empty) constructor in **DirectedGraphNode**, will the code compile without also including an empty constructor in **GraphNode**? If not, why not?

10. Write a class called **TopTwoSwitch** that extends **Stack**, but keeps track of the top two items on the stack separately. It should have a method that can switch the order of the top two objects. It should have a different method that takes in two new objects as arguments and stores the previous top two objects with all of the other objects.

11. What is the difference between an abstract class and an interface? How are they similar?

12. Look up **ArrayList** on the Oracle website. Which interfaces does it implement? What is its parent class? Give three examples of useful methods contained in the **ArrayList** class. What are their arguments? What are their return types?
package test;

public class GraphNode {
    public String address;
    public static int numNodes;

    public GraphNode(String address){
        this.address = address;
        numNodes++;
    }

    public String getAddress(){
        return address;
    }

    public static int getNumNodes(){
        return numNodes;
    }

    public String toString(){
        return "This graph node is at: " + address;
    }

    public void printGraphNode(){
        System.out.println("This is a node.");
    }
}

package test;

public class DirectedGraphNode extends GraphNode {
    public String address;
    private String previous;
    private String next;
    public static int numNodes;

    public DirectedGraphNode(String address, String p, String n){
        super(address);
        previous = p;
        next = n;
    }
}
public String getAddress() {
    return address;
}

public static int getNumNodes() {
    return numNodes;
}

public String toString() {
    return "This directed graph node is at: " + address;
}

public void printDirectedGN() {
    System.out.println("This is a directed graph node.");
}