Sample Quiz #3 Questions

Note: These might be longer than actual quiz questions but they illustrate the type of logic that will be required to answer the quiz questions.

1) Complete the Sudoku class below which has an instance variable of a 9 x 9 integer array and another 9 x 9 boolean array. Create a constructor that takes in a 9 x 9 integer array which is guaranteed to store values 0 – 9 only, where 0s represent unfilled squares. In the boolean array, store true in the locations that initially store 0 (meaning that those are editable) and false in all other locations (indicating that they are not editable.)

In addition to the constructor, add the following methods:

```
// Returns true if there is no value on row r equal to v and
// the square on row r, column c is currently unfilled.
// 0 <= r,c <= 8
private boolean canPlaceOnRow(int r, int c, int v)

// Returns true if there is no value on column c equal to v and
// the square on row r, column c is currently unfilled.
// 0 <= r,c <= 8
private boolean canPlaceOnCol(int r, int c, int v)

// Returns true if there is no value in the 3 x 3 box of
// row r, column c that contains v and that square is currently
// unfilled, 0 <= r <= c
private boolean canPlaceInBox(int r, int c, int v)

// Returns true if it's possible to place v in location row r,
// column c, false otherwise.
public boolean canPlace(int r, int c, int v)

// Erases the temporary value placed in location row r, col c,
// returns false if this is an unerasable value and also returns
// false if there was no value filled in the square to begin
// with. Returns true otherwise and resets value in row r col c
// to 0.
public boolean undo(int r, int c)

// Attempts to place value v in row r, column c. Returns true
// if successful and false otherwise.
public boolean place(int r, int c, int v)

// Returns true iff all squares
public boolean win()

// Returns a string representation of the this object.
public String toString()
```

2) What is the output of the program below? (Hint: there will be exactly 16 lines of output. Also, note that the constructors produce a total of 10 lines of output.)

```
class A {
	protected int x;

	public A() {
		System.out.println("In Default A.");
		x = 0;
	}

	public A(int val) {
		System.out.println("In A(int).");
		x = val;
	}

	public void function() {
		System.out.println("In A's function.");
	}

	public void functionA() {
		System.out.println("Only A has this function.");
	}
}

class B extends A {
	protected int y;

	public B() {
		System.out.println("In Default B.");
		y = 0;
	}

	public B(int val) {
		super(val);
		System.out.println("In B(int).");
		y = val;
	}

	public void function() {
		System.out.println("In B's function.");
	}
}
```

```
public class C extends B {
      protected int z;

      public C() {
            super(5);
            System.out.println("In Default C.");
            z = 2;
      }

      public C(int val) {
            super();
            System.out.println("In C(int).");
      }

      public C(int one, int two, int three) {
            System.out.println("In C(int,int,int).");
            x = one;
            y = two;
            z = three;
      }

      public void functionA() {
            System.out.println("A lied.");
      }

      public static void main(String[] args) {
            A test1 = new A();
            C test2 = new C(7);
            C test3 = new C(4,6,8);
            test3.function();
            test2.functionA();
            A temp = test2;
            temp.function();
            temp.functionA();
            temp = new A();
            temp.functionA();
            B test4 = new B(5);
            test4.functionA();
      }
}
```

3) Consider the following interface for modes of transportation:

```
public interface ModesOfTransportation {
    public double getAvgSpeed();
    public double getMaxSpeed();
}
```

Write a complete class called Walking that implements this interface. This class should have two instance variables:

```
private double minWalkSpeed;
private double maxWalkSpeed;
```

It should have two constructors: a default constructor that sets these two instance variables to 1.5 and 3.5, respectively, as well as a second constructor that takes in two doubles and sets both instance variables accordingly. For this class, define the average walking speed to simply be the average of the minimum and maximum walking speeds.

4) Consider creating a class Sprinter, which implements Comparable<Sprinter>. The Sprinter object has the following instance variables:

```
private String name;
private double hundredMeterTime;
private double twoHundredMeterTime;
```

Show the implementation of the compareTo method of this class so that if Sprinters were to be sorted, they would be sorted in order of their hundredMeterTimes. If those times are within .000000001 of each other, then break the tie by twoHundredMeterTime. Finally if these are within .000000001 of each other, break the tie by lexicographical ordering of the names.

```
public int compareTo(Sprinter other) {
    // Fill in code
}
```