CSCI 201L Final - Written
Summer 2014
12% of course grade

1. **Polymorphism –** Does the following code compile? If so, what is the output? If not, why not? Explain your answer. **(1.5%)**

```
class P1 {
	protected int num;
	P1(int num) {
		this.num = num;
	}
	public void print() {
		System.out.println("P1: " + num);
	}
}

public class Problem1 extends P1 {
	private int num;
	Problem1() {
		super(5);
		this.num = 10;
	}
	public void print() {
		System.out.println("Problem1: " + num);
	}

	public static void main(String [] args) {
		P1 p = new Problem1();
		p.print();
	}
}
```

2. **Subnetting –** Now that you have learned a little about networking, you have decided to set up a web server and mail server at your home. When you call your ISP, the technician gives you the following address: 215.139.55.120/29. What is the subnet mask? What IP addresses are available for you to assign to your web server and mail server? **(1.0% + 1.0%)**

```
215.139.55.120 = 1101 0111  1000 1011  0011 0111  0111 1000
```

3. **Networking –** Write two snippets of code. The first snippet will be server code for accepting an incoming connection and sending a string back to the client that says "Connected." The second snippet will be for connecting to the server and receiving a string. Assume the server is running on a machine named "csci201.usc.edu". Snippets do not need to be complete methods or have import statements. **(1.0% + 1.0%)**

4. **Concurrent vs Distributed vs Parallel –** Explain the differences among concurrent programming, distributed programming, and parallel programming. Provide one sample application that would use each paradigm. **(2.0%):**

5. **Monitors and Locks –** The use of the **synchronized** keyword can affect the execution of a program. Looking at the **changeNum** method below, explain how the output would be different if the keyword **synchronized** was removed. Provide two possible outputs of the program using the **synchronized** keyword and two possible outputs if **synchronized** keyword was removed. **(1.0% + 1.0% + 1.0%)**

```
public class Problem5 implements Runnable {

  private static int num = 5;
  private char c;
  Problem5(char c) {
    this.c = c;
  }
  public void run() {
    changeNum(c);
  }

  private static synchronized void changeNum(char ch) {
    System.out.println(ch + "1 = " + num);
    int num1 = num + 2;
    Thread.yield();
    num = num1;
    System.out.println(ch + "2 = " + num);
  }

  public static void main(String [] args) {
    Thread t1 = new Thread(new Problem5('a'));
    Thread t2 = new Thread(new Problem5('b'));
    Thread t3 = new Thread(new Problem5('c'));
    t1.start();
    t2.start();
    t3.start();
    try {
      t1.join();
      t2.join();
      t3.join();
    } catch (InterruptedException ie) {
      System.out.println("IE: " + ie.getMessage());
    }
    System.out.println("num = " + num);
  }
}
```

6. **Locks and Conditions –** A solution to the dining philosopher problem involved assigning unique values to the resources and obtaining locks on those resources in the same order regardless of the order of thread execution. In the Producer/Consumer example, we saw that deadlock still occurred even using this solution. Locks allowed us to circumvent the deadlock. Explain why locks were required instead of monitors. Could semaphores have been used instead in the Producer/Consumer problem and still avoided deadlock? (**1.0% + 0.5%**)