CSCI 201L Final – Written SOLUTION
Fall 2015
13% of course grade

1. **Generics** – C++ has had templates long before Java even existed as a language. When Java was created, there were no templates or generics. It wasn't until many years and many versions later before Java created generics in the language.
    a. Name two ways Java could deal with objects needing to take variables of varying types before generics existed. **(0.5% + 0.5%)**

1. Method overloading.
2. All objects inherit from Object, so methods could take an Object as a parameter.

b. Generics have more functionality than templates. Name one piece of functionality related to generics that is not included with templates. **(0.5%)**

1. Requiring a generic to be a derived class of another class.
2. Requiring a generic to be a super class of another class.

2. **Serialization** – Explain why Serializable objects must have a `serialVersionUID` variable. What is the purpose of this variable? **(1.0%)**

The serialVersionUID variable helps maintain consistency when multiple versions of the same object can be serialized. The serialVersionUID of the object that was serialized must be the same as the serialVersionUID of the object that is deserialized. This ensures that all of the variables are the same in the object into which the deserialized data is going.

3. **Networking Theory** – Given the following IP address and subnet mask, answer the following questions.

IP Address – 222.93.209.96
Subnet Mask – 255.255.255.128

IP – 1101 1110 0101 1101 1101 0001 0110 0000
Mask – 1111 1111 1111 1111 1111 1111 1000 0000

a. What is the network address? Provide this in the dotted IP notation, not in binary. **(0.5%)**

This is a Class C IP address, so the network address is 222.93.209.0.

b. What is the subnet address? Provide this in the dotted IP notation, not in binary. **(0.5%)**

222.93.209.0

c. How many hosts can be on the subnetwork? **(0.5%)**

$2^7 - 2 = 126$. 126 since the hosts with all 0s (network address) and all 1s (broadcast address) are reserved.

d. What is the range of IP addresses for this subnet? In other words, what is the starting IP address and ending IP address of this subnet? Provide this in the dotted IP notation, not in binary. **(1.0%)**

Starting address is 222.93.209.1
Ending address is 222.93.209.126

4. **Locks and Conditions** – Answer the following questions about locks and conditions.
   a. Monitors and locks (excluding conditions) can be used interchangeably. Give two reasons why a programmer would choose to use a monitor instead of a lock. **(0.5% + 0.5%)**

1. Simpler code
2. No need to have a try-finally
3. Guarantee that deadlock won't occur by the lock not being released

   b. Conditions add functionality to locks, but before any methods can be called on a condition, the lock associated with that condition must be obtained. Explain why this makes sense? **(1.0%)**

Conditions allow a thread to be moved into the waiting state until that condition is signaled by another thread. If the thread doing the signaling or waiting doesn't have the lock associated with that condition, there is no requirement that the thread be in a critical section of code. That means that the condition functionality does not need to be synchronized, and any thread could signal another thread waiting on a condition while another thread holds the lock. This is not how synchronization works.

5. **Concurrent Programming –** Modify the following code using concurrency theory to ensure all 100 values for each thread are printed before any other values are printed. **(2.0%)**

```
import java.util.ArrayList;
public class Problem5 extends Thread {
    public static ArrayList<Integer> al = new ArrayList<Integer>();
    public void run() {
           for (int i=0; i < al.size(); i++) {
                  System.out.println("A" + al.get(i));
           }
    }
    public static void main(String[] args) {
           for (int i=0; i < 100; i++) {
                  al.add(i);
           }
           Problem5 p5 = new Problem5();
           Problem5_1 p5_1 = new Problem5_1();
           p5.start();
           p5_1.start();
    }
}
class Problem5_1 extends Thread {
    public void run() {
           for (int i=0; i < Problem5.al.size(); i++) {
                  System.out.println("B" + Problem5.al.get(i));
           }
    }
}
```

Put a lock around the content of both run methods, making sure they are using the same lock, such as by making the lock static in the Problem5 class.

This also could be synchronized on the same object, such as al, around the content of each run method.

6. **Multithreading –** Give three rules that will always be true about the output of the following program. **(0.5% + 0.5% + 0.5%)**

```
import java.util.ArrayList;
import java.util.concurrent.Semaphore;

public class Problem6 extends Thread {
   public static ArrayList<Integer> al = new ArrayList<Integer>();
   public static Semaphore sem = new Semaphore(2);
   private int num;
   public Problem6(int num) {
         this.num = num;
   }
   public void run() {
         try {
               sem.acquire();
               System.out.println(num + " starting ");
               for (int i=0; i < al.size(); i++) {
                     System.out.println(al.get(i));
               }
         } catch(InterruptedException ie) {

         } finally {
               System.out.println(num + " ending ");
               sem.release();
         }
   }
   public static void main(String[] args) {
         for (int i=0; i < 5; i++) {
               al.add(i);
         }
         for (int i=0; i < 100; i++) {
               Problem6 p6 = new Problem6(i);
               P6.start();
         }
   }
}
```

1. **“1 starting” will always get printed before “1 ending”.**
2. **The values of al will always be printed in order, though there may be other numbers in the middle.**
3. **Only two threads will have printed “starting” before there must be an “ending”.**

7. **Distributed Programming –** Answer the following questions about distributed programming.
    a. Considering RMI, CORBA, and Web Services, what is the order from oldest to newest that they were created? **(0.5%)**

    CORBA, RMI, Web Services

    b. CORBA and Web Services both have to use an IDL. Why is an IDL needed? **(0.5%)**

    An IDL is needed because CORBA and Web Services can have the clients and servers written in different languages. An IDL provides a mapping to a common variable type so multiple languages can communicate.

    c. Why does RMI not have an IDL? **(0.5%)**

    RMI does not have an IDL because both the client and server are written in Java. RMI uses serialization to send variables back and forth.

8. **Multi-Threading and Parallel Programming** – Answer the following questions about concurrent programming.
    a. With multi-threaded programming, what is our ultimate goal? In other words, what are we trying to do the program? **(0.5%)**

    Add functionality.

    b. With parallel programming, what is our ultimate goal? In other words, what are we trying to do to the program? **(0.5%)**

    Improve the actual running time.

9. **Bonus Question –** Of all the topics we covered in the class, which do you think was the most useful? Which do you think was the least useful? Please explain your reason for both. **(0.25% + 0.25%)**
   NOTE: This question will only improve your exam grade up to the full 13%.

Week	Lecture	Lecture Topic	Most Useful	Least Useful
1	1	Introduction, Environment, Methods		
	2	Classes, Polymorphism, Interfaces		
2	3	Garbage Collection, Exception Handling		
	4	File I/O, Serialization, Generics		
	5	User Interface Design, JPanels		
4	6	Layout Managers		
	7	Event-Driven Programming, Inner Classes		
5	8	Menus, Toolbars, Option Panes		
	9	GUI Components, Graphics		
6	10	Tables, Trees		
	11	Software Engineering, Project Discussion		
7	12	Concurrent Computing		
	13	Multi-threaded Programming		
8	14	Networking Theory		
	15	Networking Programming		
9	16	More Networking, Serialization Revisited		
	17	Databases, JDBC		
10	18	Concurrency, Critical Sections, Monitors		
	19	Locks, Conditions		
11	20	Monitors and Locks Programming		
	21	Semaphores		
12	22	Semaphore Programming		
	23	Parallel Computing		
13	24	Distributed Computing, RPC		
	25	RMI		