1. **Inheritance** - Explain the difference between an abstract class that only contains abstract methods and an interface. Why would a programmer choose to use an abstract class that only contains abstract methods instead of an interface? (1.0%)
3. **Multi-threaded Programming** – In the following diagram, explain when a thread moves from one state to another. There are eight transitions to explain. (1.5%)
4. More Multi-Threadiing – What are three possible outputs of the following program? What will always be true in the output? (2.0%):

```java
public class Problem4 {
    public static void main(String[] args) {
        System.out.println("First line");
        try {
            for (int i=0; i < 10; i++) {
                MyThread t = new MyThread(i);
                t.start();
                t.join();
            }
        } catch (InterruptedException ie) {
            System.out.println("IE: "+ ie.getMessage());
        }
        System.out.println("Last line");
    }
}

class MyThread extends Thread {
    private int num;
    public MyThread(int num) {
        this.num = num;
    }
    public void run() {
        for (char c='A'; c <= 'Z'; c++) {
            System.out.print(num + "" + c + "");
        }
        System.out.println("");
    }
}
```
5. **Monitors and Locks** – Explain the difference between the following two sections of code. Why would a programmer choose to use each? (2.0%) 

```java
// code block #1
class PiggyBank {
    private int balance = 0;
    public int getBalance() {
        return balance;
    }
    public synchronized void deposit(int amount) {
        int newBalance = balance + amount;
        try {
            Thread.sleep(1);
        } catch (InterruptedException ie) {
            System.out.println("IE: " + ie.getMessage());
        }
        balance = newBalance;
    }
}

// code block #2
class PiggyBank {
    private int balance = 0;
    private Lock lock = new ReentrantLock();
    public int getBalance() {
        return balance;
    }
    public void deposit(int amount) {
        lock.lock();
        try {
            int newBalance = balance + amount;
            Thread.sleep(1);
            balance = newBalance;
        } catch (InterruptedException ie) {
            System.out.println("IE: " + ie.getMessage());
        } finally {
            lock.unlock();
        }
    }
}
```
6. **Semaphores** – Given the following code, explain how the following output is generated. What properties will always hold true about the output? (2.0%)

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

public class Problem6 {
    public static void main(String [] args) {
        for (int i=0; i < 10; i++) {
            SemThread st = new SemThread(i);
            st.start();
        }
    }
}

class SemThread extends Thread {
    private static Semaphore sem = new Semaphore(3);
    private int num;
    public SemThread(int num) {
        this.num = num;
    }
    public void run() {
        try {
            System.out.println(num + " done: " + sem.availablePermits());
            sem.acquire();
            Thread.sleep(1000);
        } catch (InterruptedException ie) {
            System.out.println("IE: " + ie.getMessage());
        } finally {
            sem.release();
            System.out.println(num + " done: " + sem.availablePermits());
        }
    }
}
```

0: 3
2: 2
5: 1
6: 1
8: 0
9: 0
4: 0
3: 0
7: 0
1: 0
5 done: 3
2 done: 1
0 done: 3
9 done: 2
8 done: 0
6 done: 2
3 done: 1
4 done: 2
1 done: 2
7 done: 3
7. **Distributed Computing** – Explain the difference between the functionality of the ExecutorService using a thread pool and the ForkJoinPool. *(1.0%)*

8. **RMI, CORBA, Web Services** – Distributed computing technologies allow a program to execute on another computer though the call occurs locally. We learned that networked code could provide this same functionality. Explain the difference between writing a client-server program to execute some task on the server compared to utilizing one of the distributed computing technologies such as RMI, CORBA, or Web Services. *(1.5%)*