QUESTION 1:

```
package os;

import java.util.Scanner;

class Process {
    int id;
    int burstTime;
    int remainingTime;

    public Process(int id, int burstTime) {
        this.id = id;
        this.burstTime = burstTime;
        this.remainingTime = burstTime;
    }
}

public class conSwitch {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Number of processes: ");
        int numProcesses = scanner.nextInt();

        Process[] processes = new Process[numProcesses];
        for (int i = 0; i < numProcesses; i++) {
            System.out.print("Enter Process ID for Process " + (i + 1) + ":
");
            int id = scanner.nextInt();
            System.out.print("Enter Burst Time for Process " + (i + 1) + ":
");
            int burstTime = scanner.nextInt();
            processes[i] = new Process(id, burstTime);
        }

        System.out.print("Time Quantum (TQ): ");
        int time = scanner.nextInt();

        simulate(processes, time);

        scanner.close();
    }

    public static void simulate(Process[] processes, int time) {
        int[] remainingTime = new int[processes.length];
        boolean[] completed = new boolean[processes.length];
        int currentTime = 0;
```

```
        while (true) {
            boolean allCompleted = true;
            for (int i = 0; i < processes.length; i++) {
                if (!completed[i]) {
                    allCompleted = false;
                    if (processes[i].remainingTime > 0) {
                        if (processes[i].remainingTime > time) {
                            System.out
                                    .println("Process " + processes[i].id + "
executing for " + time + " units");
                            processes[i].remainingTime -= time;
                            currentTime += time;
                        } else {
                            System.out.println("Process " + processes[i].id +
" executing for "
                                    + processes[i].remainingTime + " units");
                            currentTime += processes[i].remainingTime;
                            processes[i].remainingTime = 0;
                            completed[i] = true;
                        }
                    }
                }
            }
            if (allCompleted) {
                break;
            }
        }
    }
}
```

```
PS C:\Users\Ramesh Babu\.vscode\oopsjava>  c:; cd 'c:\Us
\Ramesh Babu\AppData\Roaming\Code\User\workspaceStorage\
Number of processes: 3
Enter Process ID for Process 1: 1
Enter Burst Time for Process 1: 4
Enter Process ID for Process 2: 2
Enter Burst Time for Process 2: 12
Enter Process ID for Process 3: 3
Enter Burst Time for Process 3: 9
Time Quantum (TQ): 3
Process 1 executing for 3 units
Process 2 executing for 3 units
Process 3 executing for 3 units
Process 1 executing for 1 units
Process 2 executing for 3 units
Process 3 executing for 3 units
Process 2 executing for 3 units
Process 3 executing for 3 units
Process 2 executing for 3 units
```

QUESTION 2:

```java
package os;

class Shared {
    private int value;

    public Shared() {
        this.value = 0;
    }

    public int getValue() {
        return value;
    }

    public void increment() {
        value++;
    }
}

class Worker extends Thread {
    private Shared shared;

    public Worker(Shared shared) {
```

```
        this.shared = shared;
    }

    @Override
    public void run() {
        for (int i = 0; i < 3; i++) {
            shared.increment();
            System.out.println(Thread.currentThread().getName() + ":
Incremented value to " + shared.getValue());
        }
    }
}

public class criticalSection {
    public static void main(String[] args) {
        Shared shared = new Shared();

        Thread thread1 = new Worker(shared);
        Thread thread2 = new Worker(shared);

        thread1.start();
        thread2.start();
    }
}
```

```
PS C:\Users\Ramesh Babu\.vscode\oopsjava>
\Ramesh Babu\AppData\Roaming\Code\User\wo

Thread-1: Incremented value to 2
Thread-0: Incremented value to 2
```

QUESTION 3

```
package os;

import java.util.concurrent.Semaphore;

class Q {
    int item;
    static Semaphore semCon = new Semaphore(0);

    static Semaphore semProd = new Semaphore(1);

    void get() {
        try {
```

```
            semCon.acquire();
        } catch (InterruptedException e) {
            System.out.println("InterruptedException caught");
        }

        System.out.println("Consumer consumed item : " + item);

        semProd.release();
    }

    void put(int item) {
        try {
            semProd.acquire();
        } catch (InterruptedException e) {
            System.out.println("InterruptedException caught");
        }

        this.item = item;

        System.out.println("Producer produced item : " + item);
        semCon.release();
    }
}

class Producer implements Runnable {
    Q q;

    Producer(Q q) {
        this.q = q;
        new Thread(this, "Producer").start();
    }

    public void run() {
        for (int i = 0; i < 5; i++)
            q.put(i);
    }
}

class Consumer implements Runnable {
    Q q;

    Consumer(Q q) {
        this.q = q;
        new Thread(this, "Consumer").start();
    }

    public void run() {
        for (int i = 0; i < 5; i++)
```

```
            q.get();
    }
}

class PC {
    public static void main(String args[]) {
        Q q = new Q();
        new Consumer(q);
        new Producer(q);
    }
}
```

```
\Ramesh Babu\AppData\Roaming\Code\User\workspaceStorage
Producer produced item : 0
Consumer consumed item : 0
Producer produced item : 1
Consumer consumed item : 1
Producer produced item : 2
Consumer consumed item : 2
Producer produced item : 3
Consumer consumed item : 3
Producer produced item : 4
Consumer consumed item : 4
```

QUESTION 4:

```
package os;

import java.util.concurrent.Semaphore;

class ReaderWriter {
    private Semaphore mutex;
    private Semaphore readerLock;
    private Semaphore writerLock;
    private int readers;

    public ReaderWriter() {
        mutex = new Semaphore(1);
        readerLock = new Semaphore(1);
        writerLock = new Semaphore(1);
        readers = 0;
    }
```

```
    public void startReading(int readerId) throws InterruptedException {
        readerLock.acquire();
        mutex.acquire();
        readers++;
        if (readers == 1) {
            writerLock.acquire();
        }
        mutex.release();
        readerLock.release();

        System.out.println("Reader " + readerId + ": Started reading");

        readerLock.acquire();
        mutex.acquire();
        readers--;
        if (readers == 0) {
            writerLock.release();
        }
        mutex.release();
        readerLock.release();
    }

    public void startWriting(int writerId) throws InterruptedException {
        writerLock.acquire();
        System.out.println("Writer " + writerId + ": Started writing");
        writerLock.release();
    }
}

class Reader extends Thread {
    private ReaderWriter rw;
    private int id;

    public Reader(ReaderWriter rw, int id) {
        this.rw = rw;
        this.id = id;
    }

    @Override
    public void run() {
        try {
            rw.startReading(id);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
class Writer extends Thread {
    private ReaderWriter rw;
    private int id;

    public Writer(ReaderWriter rw, int id) {
        this.rw = rw;
        this.id = id;
    }

    @Override
    public void run() {
        try {
            rw.startWriting(id);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class ReaderWriterProblem {
    public static void main(String[] args) {
        int numReaders = 3;
        int numWriters = 2;

        ReaderWriter rw = new ReaderWriter();

        for (int i = 1; i <= numReaders; i++) {
            new Reader(rw, i).start();
        }

        for (int i = 1; i <= numWriters; i++) {
            new Writer(rw, i).start();
        }
    }
}
```

```
PS C:\Users\Ramesh Babu\.vscode\oopsjava>  c:; cd 'c:\Users\
\Ramesh Babu\AppData\Roaming\Code\User\workspaceStorage\2ddb
em'
Reader 3: Started reading
Reader 2: Started reading
Reader 1: Started reading
Writer 1: Started writing
Writer 2: Started writing
```

QUESTION 5:

```java
package os;

import java.util.concurrent.Semaphore;
import java.util.Scanner;

public class ResourceAllocation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of processes: ");
        int numProcesses = scanner.nextInt();
        System.out.print("Enter the number of resources: ");
        int numResources = scanner.nextInt();
        System.out.print("Enter the maximum number of resources each process
can acquire: ");
        int maxResources = scanner.nextInt();
        Semaphore semaphore = new Semaphore(numResources-1, true);

        for (int i = 1; i <= numProcesses; i++) {
            Thread process = new Thread(new Process(i, semaphore,
maxResources));
            process.start();
        }
        scanner.close();
    }

    static class Process implements Runnable {
        private int processId;
        private Semaphore semaphore;
        private int maxResources;

        public Process(int processId, Semaphore semaphore, int maxResources) {
            this.processId = processId;
            this.semaphore = semaphore;
            this.maxResources = maxResources;
        }
```

```
        @Override
        public void run() {
            try {
                semaphore.acquire();
                System.out.println("Process " + processId + ": Acquired
resource");
                // Thread.sleep(1000); // Simulating some work
                semaphore.release();
                System.out.println("Process " + processId + ": Released
resource");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
PS C:\Users\Ramesh Babu\.vscode\oopsjava>  c:; cd 'c:\Users\Ramesh Bab
\Ramesh Babu\AppData\Roaming\Code\User\workspaceStorage\2ddb5e05245b95
n'
Enter the number of processes: 3
Enter the number of resources: 2
Enter the maximum number of resources each process can acquire: 1
Process 1: Acquired resource
Process 2: Acquired resource
Process 3: Acquired resource
Process 1: Released resource
Process 2: Released resource
Process 3: Released resource
```

QUESTION 6:

```
package os;

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class Deadlock {
    public static void main(String[] args) {
        Resource resourceA = new Resource();
        Resource resourceB = new Resource();
```

```
        Process process1 = new Process(1, resourceA, resourceB);
        Process process2 = new Process(2, resourceA, resourceB);

        process1.start();
        process2.start();
    }
}

class Resource {
    private Lock lock;

    public Resource() {
        lock = new ReentrantLock();
    }

    public void acquire() {
        lock.lock();
    }

    public void release() {
        lock.unlock();
    }
}

class Process extends Thread {
    private int id;
    private Resource resourceA;
    private Resource resourceB;

    public Process(int id, Resource resourceA, Resource resourceB) {
        this.id = id;
        this.resourceA = resourceA;
        this.resourceB = resourceB;
    }

    @Override
    public void run() {
        if (id == 1) {
            acquireResources(resourceA, resourceB);
        } else {
            acquireResources(resourceB, resourceA);
        }
    }

    private void acquireResources(Resource first, Resource second) {
        first.acquire();
        System.out.println("Process " + id + ": Acquired " + first);
        try {
```

```
            Thread.sleep(1000); // Simulate some work
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Process " + id + ": Waiting for " + second + "
(Deadlock occurs)");
        second.acquire();
        System.out.println("Process " + id + ": Acquired " + second);
        first.release();
        second.release();
    }
}
```

```
Process 2: Acquired Resource B
Process 1: Acquired Resource A
Process 1: Waiting for Resource B (Deadlock occurs)
Process 2: Waiting for Resource A
```

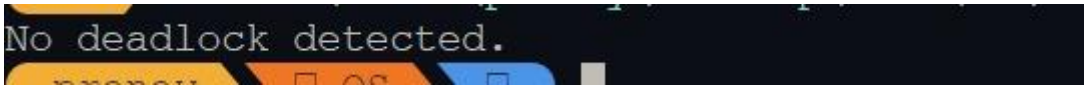
QUESTION 7:

```
package os;

import java.util.*;

class Resource {
    private int id;
    private boolean allocated;

    public Resource(int id) {
        this.id = id;
        this.allocated = false;
    }

    public int getId() {
        return id;
    }

    public boolean isAllocated() {
        return allocated;
    }

    public void allocate() {
        allocated = true;
    }

    public void deallocate() {
```

```
        allocated = false;
    }
}

class Process {
    private int id;
    private List<Resource> resourcesNeeded;

    public Process(int id) {
        this.id = id;
        this.resourcesNeeded = new ArrayList<>();
    }

    public int getId() {
        return id;
    }

    public List<Resource> getResourcesNeeded() {
        return resourcesNeeded;
    }

    public void addResource(Resource resource) {
        resourcesNeeded.add(resource);
    }

    public void removeResource(Resource resource) {
        resourcesNeeded.remove(resource);
    }
}

public class DeadlockDetection {
    public static void main(String[] args) {
        int numProcesses = 2;
        int numResources = 2;
        Resource resourceA = new Resource(1);
        Resource resourceB = new Resource(2);
        Process process1 = new Process(1);
        process1.addResource(resourceA);
        process1.addResource(resourceB);

        Process process2 = new Process(2);
        process2.addResource(resourceB);
        process2.addResource(resourceA);

        resourceA.allocate();
        resourceB.allocate();

        if (isDeadlock(numProcesses, numResources, process1, process2)) {
```

```
            System.out.println("Deadlock detected!");

            System.out.println("Recovery option: Terminate Process 1");
            terminateProcess(process1);
            System.out.println("Process 1 terminated");
            System.out.println("Process 2: Acquired Resource A (Deadlock
resolved)");
        } else {
            System.out.println("No deadlock detected.");
        }
    }

    private static boolean isDeadlock(int numProcesses, int numResources,
Process... processes) {
        boolean[] finished = new boolean[numProcesses];
        int[][] need = new int[numProcesses][numResources];
        int[][] allocation = new int[numProcesses][numResources];
        int[] available = new int[numResources];

        for (int i = 0; i < numProcesses; i++) {
            finished[i] = false;
            for (int j = 0; j < numResources; j++) {
                need[i][j] =
processes[i].getResourcesNeeded().get(j).isAllocated() ? 0 : 1;
                allocation[i][j] =
processes[i].getResourcesNeeded().get(j).isAllocated() ? 1 : 0;
            }
        }

        for (int i = 0; i < numResources; i++) {
            available[i] = 1;
        }

        boolean deadlock = true;
        int count = 0;
        while (deadlock && count < numProcesses) {
            deadlock = false;
            for (int i = 0; i < numProcesses; i++) {
                if (!finished[i]) {
                    boolean canExecute = true;
                    for (int j = 0; j < numResources; j++) {
                        if (need[i][j] > available[j]) {
                            canExecute = false;
                            break;
                        }
                    }
                    if (canExecute) {
                        deadlock = false;
```

```
                        finished[i] = true;
                        count++;
                        for (int j = 0; j < numResources; j++) {
                            available[j] += allocation[i][j];
                        }
                        break;
                    }
                }
            }
        }
        return deadlock;
    }

    private static void terminateProcess(Process process) {
        List<Resource> resourcesNeeded = process.getResourcesNeeded();
        for (Resource resource : resourcesNeeded) {
            resource.deallocate();
        }
    }
}
```



QUESTION 9:

```
package os;

import java.util.concurrent.Semaphore;

public class Priority {
    private static Semaphore semaphore = new Semaphore(1);

    public static void main(String[] args) {
        Thread thread1 = new Thread(new MyRunnable(semaphore), "Thread 1");
        Thread thread2 = new Thread(new MyRunnable(semaphore), "Thread 2");
        Thread thread3 = new Thread(new MyRunnable(semaphore), "Thread 3");

        thread1.setPriority(10);
        thread2.setPriority(5);
        thread3.setPriority(1);

        thread1.start();
        thread2.start();
        thread3.start();
    }
```

```
    static class MyRunnable implements Runnable {
        private Semaphore semaphore;

        public MyRunnable(Semaphore semaphore) {
            this.semaphore = semaphore;
        }

        @Override
        public void run() {
            try {
                semaphore.acquire();
                System.out.println(Thread.currentThread().getName() + "
running...");
                semaphore.release();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
PS C:\Users\Ramesh Babu\.vscode\oopsjava>  c
\Ramesh Babu\AppData\Roaming\Code\User\works
Thread 2 running...
Thread 1 running...
Thread 3 running...
```

QUESTION 10:

```
package os;

import java.io.IOException;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;

public class IPC {
    public static void main(String[] args) throws IOException {
        try {
            PipedInputStream inputPipe = new PipedInputStream();
            PipedOutputStream outputPipe = new PipedOutputStream(inputPipe);
            Thread writerThread = new Thread(() -> {
                try {
```

```
                    String inputMessage = "Hello from process 1!";
                    outputPipe.write(inputMessage.getBytes());
                    outputPipe.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
            Thread readerThread = new Thread(() -> {
                try {
                    byte[] buffer = new byte[1024];
                    int bytesRead = inputPipe.read(buffer);
                    String outputMessage = new String(buffer, 0, bytesRead);
                    System.out.println("Output (read by the other process):");
                    System.out.println(outputMessage);
                    inputPipe.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
            writerThread.start();
            readerThread.start();
            writerThread.join();
            readerThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
\Ramesh Babu\AppData\Roaming\Code\User\workspaceS
Output (read by the other process):
Hello from process 1!
```