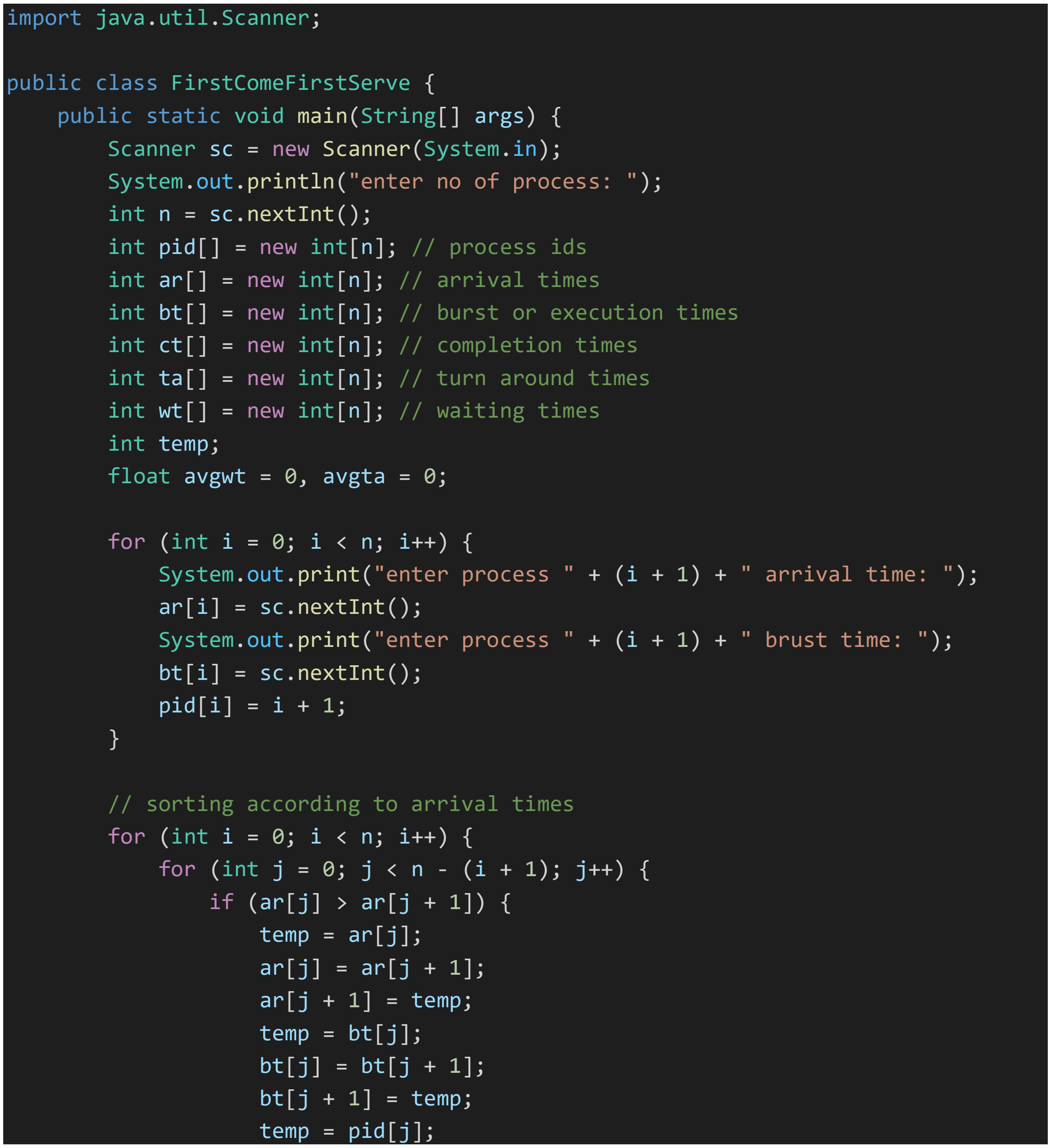
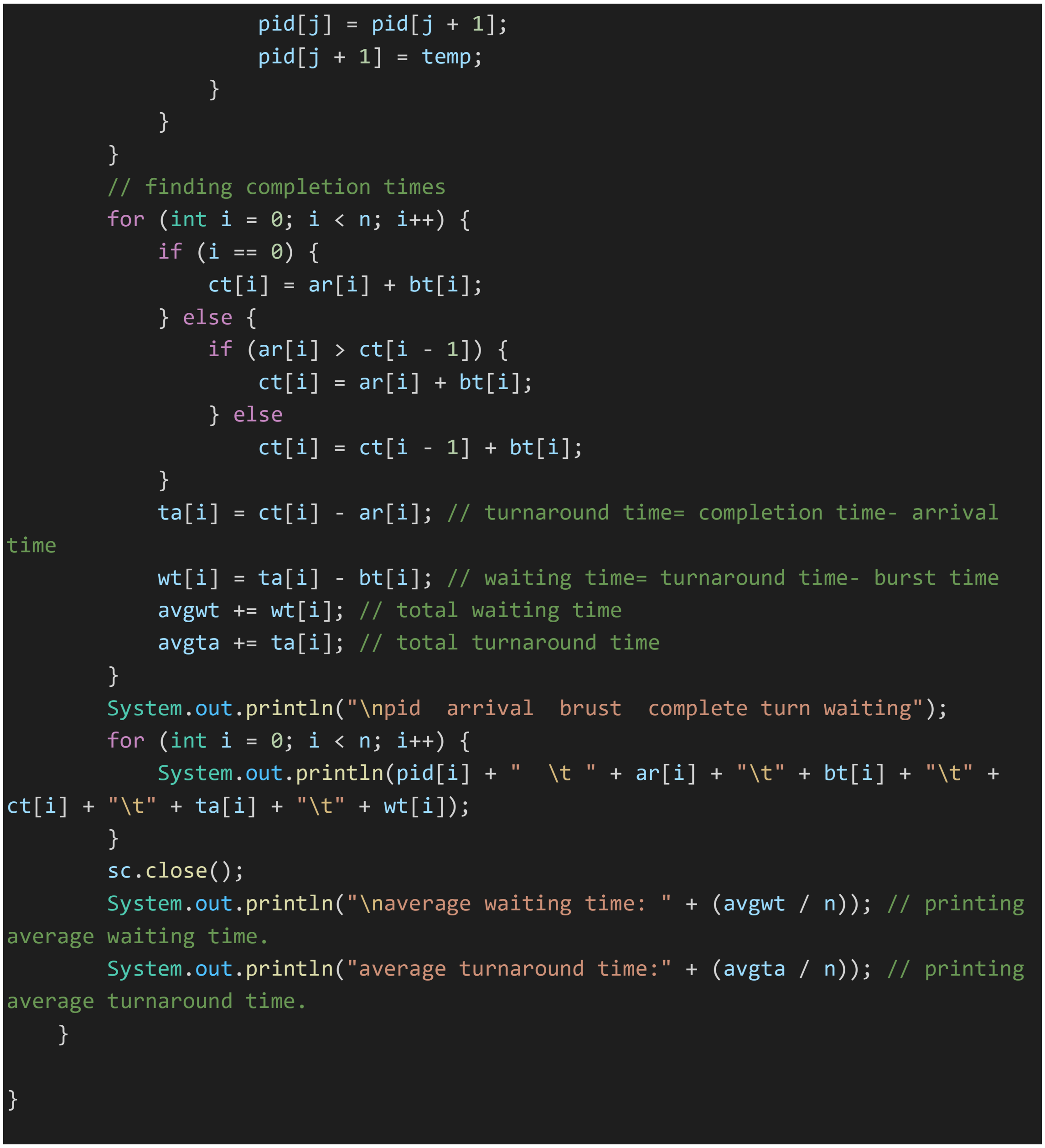
CPU SCHEDULING PROGRAMS

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time First (SRTF)
4. Priority Scheduling (Preemptive)
5. Priority Scheduling (Non-pre-emptive)
6. Round Robin

PROGRAMS 1: FIRST COME FIRST SERVE (FCFS)

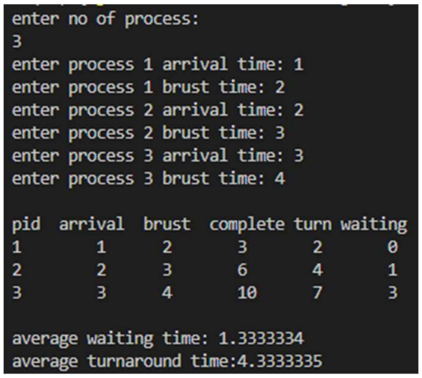


```java
import java.util.Scanner;

public class FirstComeFirstServe {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter no of process: ");
        int n = sc.nextInt();
        int pid[] = new int[n]; // process ids
        int ar[] = new int[n]; // arrival times
        int bt[] = new int[n]; // burst or execution times
        int ct[] = new int[n]; // completion times
        int ta[] = new int[n]; // turn around times
        int wt[] = new int[n]; // waiting times
        int temp;
        float avgwt = 0, avgta = 0;

        for (int i = 0; i < n; i++) {
            System.out.print("enter process " + (i + 1) + " arrival time: ");
            ar[i] = sc.nextInt();
            System.out.print("enter process " + (i + 1) + " brust time: ");
            bt[i] = sc.nextInt();
            pid[i] = i + 1;
        }

        // sorting according to arrival times
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n - (i + 1); j++) {
                if (ar[j] > ar[j + 1]) {
                    temp = ar[j];
                    ar[j] = ar[j + 1];
                    ar[j + 1] = temp;
                    temp = bt[j];
                    bt[j] = bt[j + 1];
                    bt[j + 1] = temp;
                    temp = pid[j];
```



```java
                    pid[j] = pid[j + 1];
                    pid[j + 1] = temp;
                }
            }
        }
        // finding completion times
        for (int i = 0; i < n; i++) {
            if (i == 0) {
                ct[i] = ar[i] + bt[i];
            } else {
                if (ar[i] > ct[i - 1]) {
                    ct[i] = ar[i] + bt[i];
                } else
                    ct[i] = ct[i - 1] + bt[i];
            }
            ta[i] = ct[i] - ar[i]; // turnaround time= completion time- arrival
time
            wt[i] = ta[i] - bt[i]; // waiting time= turnaround time- burst time
            avgwt += wt[i]; // total waiting time
            avgta += ta[i]; // total turnaround time
        }
        System.out.println("\npid  arrival  brust  complete turn waiting");
        for (int i = 0; i < n; i++) {
            System.out.println(pid[i] + "  \t " + ar[i] + "\t" + bt[i] + "\t" +
ct[i] + "\t" + ta[i] + "\t" + wt[i]);
        }
        sc.close();
        System.out.println("\naverage waiting time: " + (avgwt / n)); // printing
average waiting time.
        System.out.println("average turnaround time:" + (avgta / n)); // printing
average turnaround time.
    }

}
```

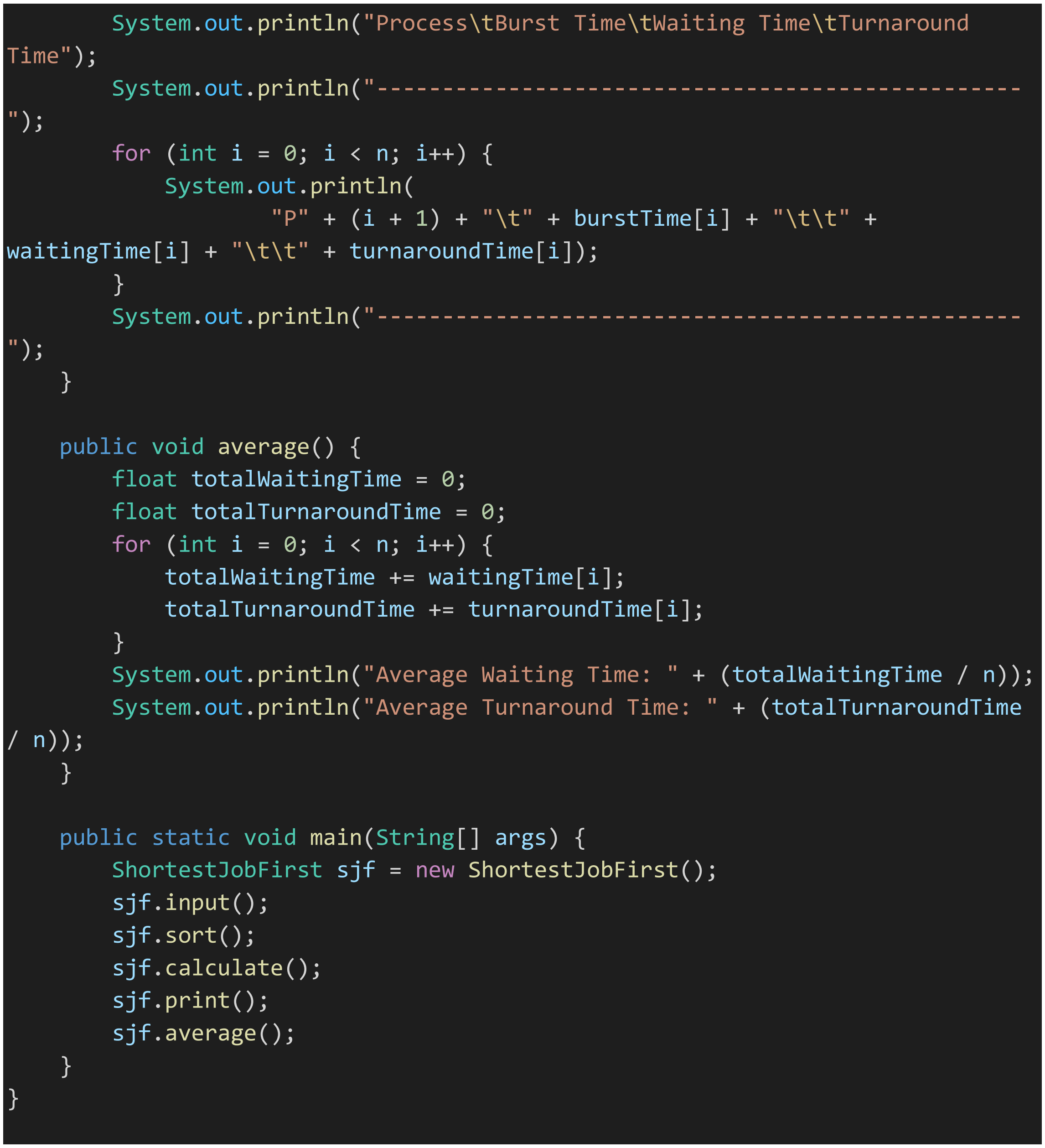
OUTPUT:



```
enter no of process:
3
enter process 1 arrival time: 1
enter process 1 brust time: 2
enter process 2 arrival time: 2
enter process 2 brust time: 3
enter process 3 arrival time: 3
enter process 3 brust time: 4

pid  arrival  brust  complete turn waiting
1         1      2       3       2       0
2         2      3       6       4       1
3         3      4       10      7       3

average waiting time: 1.3333334
average turnaround time:4.3333335
```

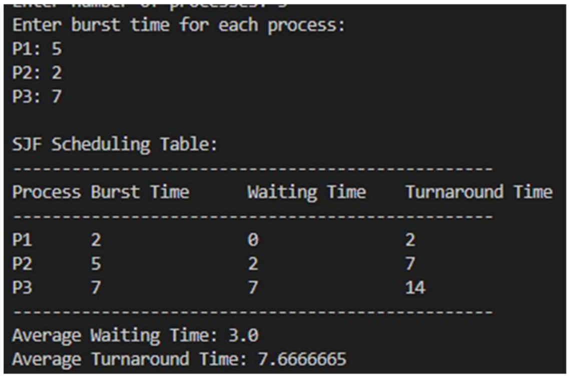
PROGRAM 2: SHORTEST JOB FIRST (SJF)

```java
import java.util.Scanner;

public class ShortestJobFirst {
    Scanner sc = new Scanner(System.in);
    int n;
    int[] burstTime;
    int[] waitingTime;
    int[] turnaroundTime;

    public void input() {
        System.out.print("Enter number of processes: ");
        n = sc.nextInt();
        burstTime = new int[n];
        waitingTime = new int[n];
        turnaroundTime = new int[n];
        System.out.println("Enter burst time for each process:");
        for (int i = 0; i < n; i++) {
            System.out.print("P" + (i + 1) + ": ");
            burstTime[i] = sc.nextInt();
        }
    }

    public void sort() {
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (burstTime[i] > burstTime[j]) {
                    int temp = burstTime[i];
                    burstTime[i] = burstTime[j];
                    burstTime[j] = temp;
                }
            }
        }
    }

    public void calculate() {
        waitingTime[0] = 0;
        turnaroundTime[0] = burstTime[0];
        for (int i = 1; i < n; i++) {
            waitingTime[i] = waitingTime[i - 1] + burstTime[i - 1];
            turnaroundTime[i] = waitingTime[i] + burstTime[i];
        }
    }

    public void print() {
        System.out.println("\nSJF Scheduling Table:");
        System.out.println("-------------------------------------------------
");
```



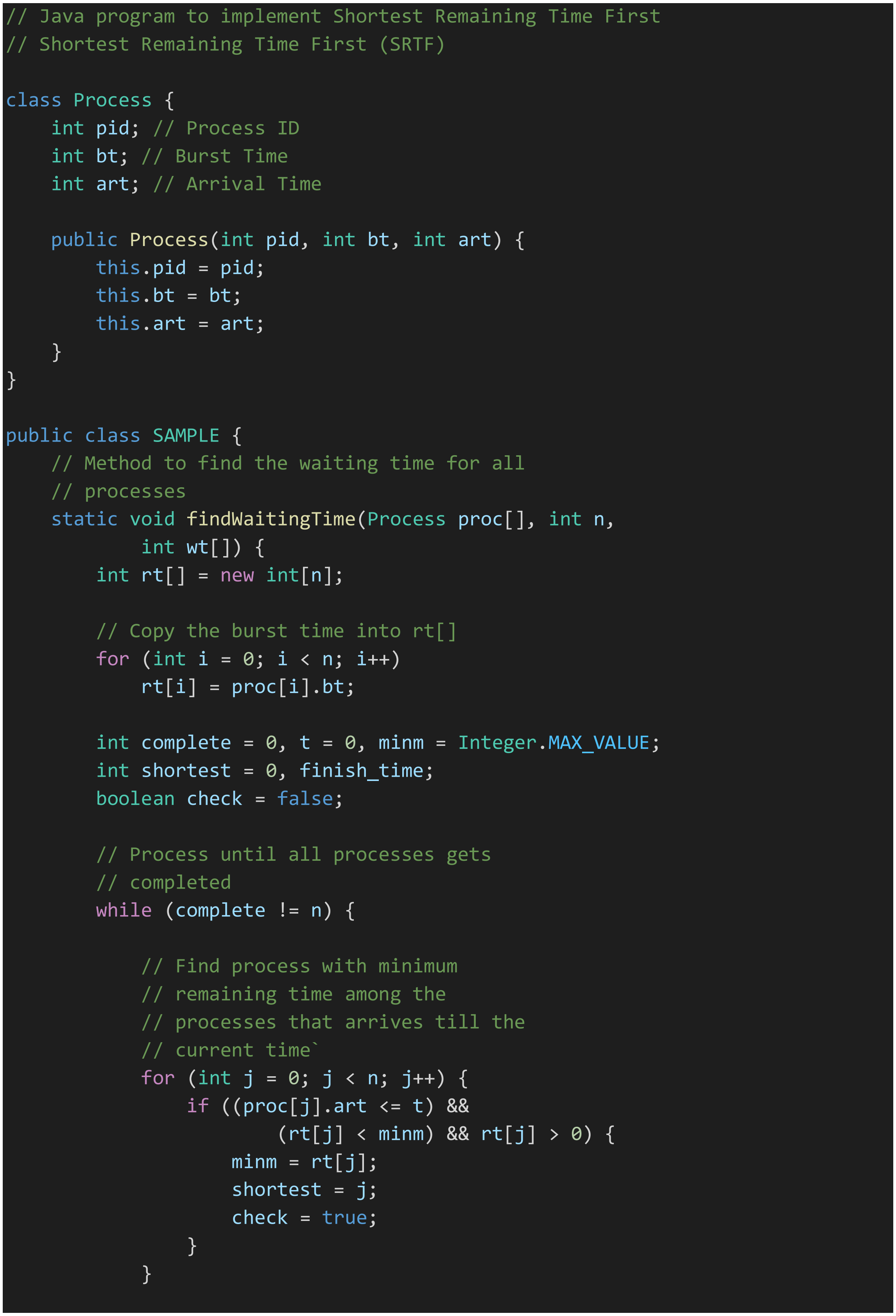
```
        System.out.println("Process\tBurst Time\tWaiting Time\tTurnaround
Time");
        System.out.println("-------------------------------------------------
");
        for (int i = 0; i < n; i++) {
            System.out.println(
                    "P" + (i + 1) + "\t" + burstTime[i] + "\t\t" +
waitingTime[i] + "\t\t" + turnaroundTime[i]);
        }
        System.out.println("-------------------------------------------------
");
    }

    public void average() {
        float totalWaitingTime = 0;
        float totalTurnaroundTime = 0;
        for (int i = 0; i < n; i++) {
            totalWaitingTime += waitingTime[i];
            totalTurnaroundTime += turnaroundTime[i];
        }
        System.out.println("Average Waiting Time: " + (totalWaitingTime / n));
        System.out.println("Average Turnaround Time: " + (totalTurnaroundTime
/ n));
    }

    public static void main(String[] args) {
        ShortestJobFirst sjf = new ShortestJobFirst();
        sjf.input();
        sjf.sort();
        sjf.calculate();
        sjf.print();
        sjf.average();
    }
}
```

OUTPUT:

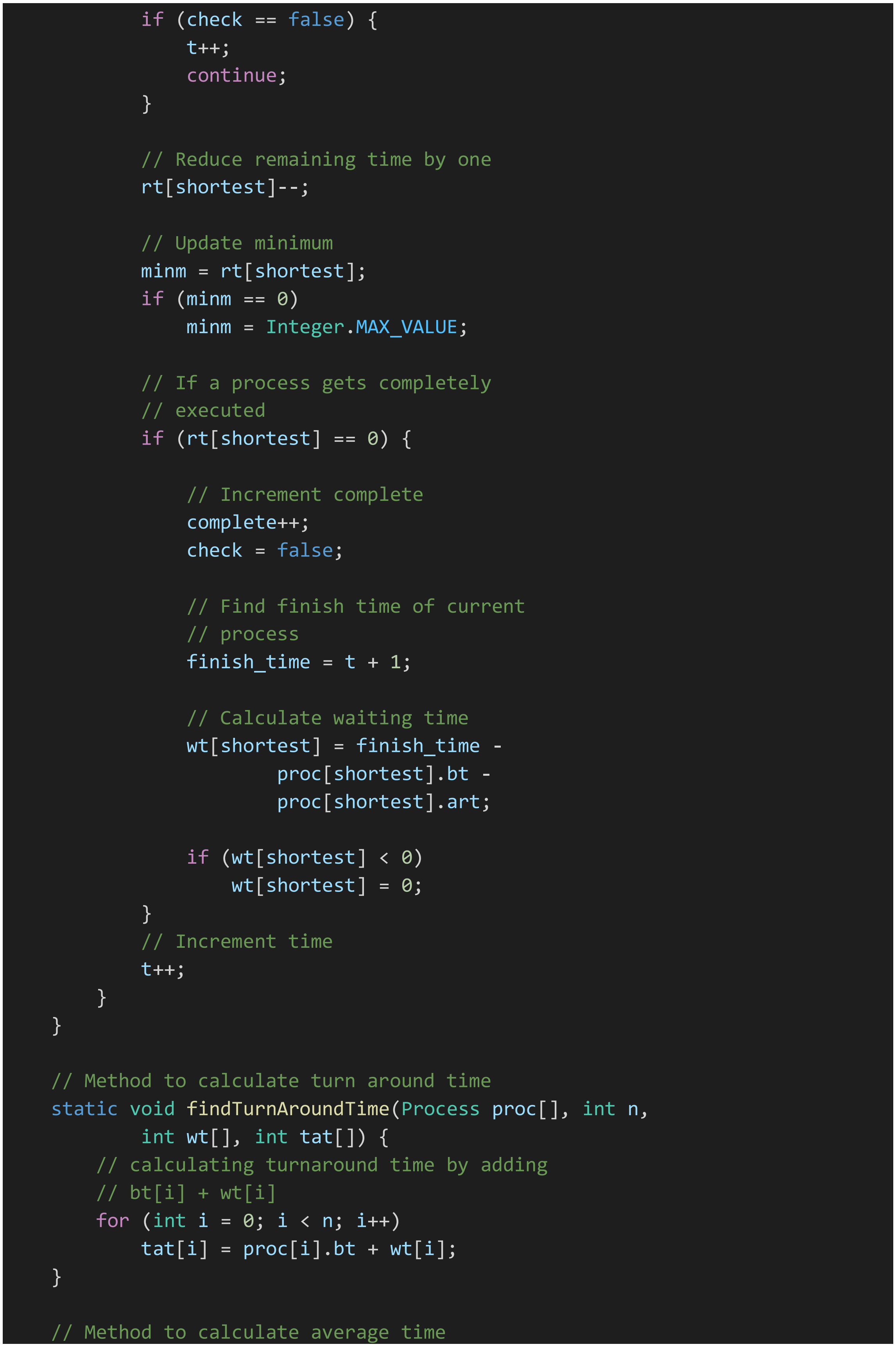


```
Enter number of processes: 3
Enter burst time for each process:
P1: 5
P2: 2
P3: 7

SJF Scheduling Table:
-------------------------------------------------
Process Burst Time      Waiting Time    Turnaround Time
-------------------------------------------------
P1      2               0               2
P2      5               2               7
P3      7               7               14
-------------------------------------------------
Average Waiting Time: 3.0
Average Turnaround Time: 7.6666665
```

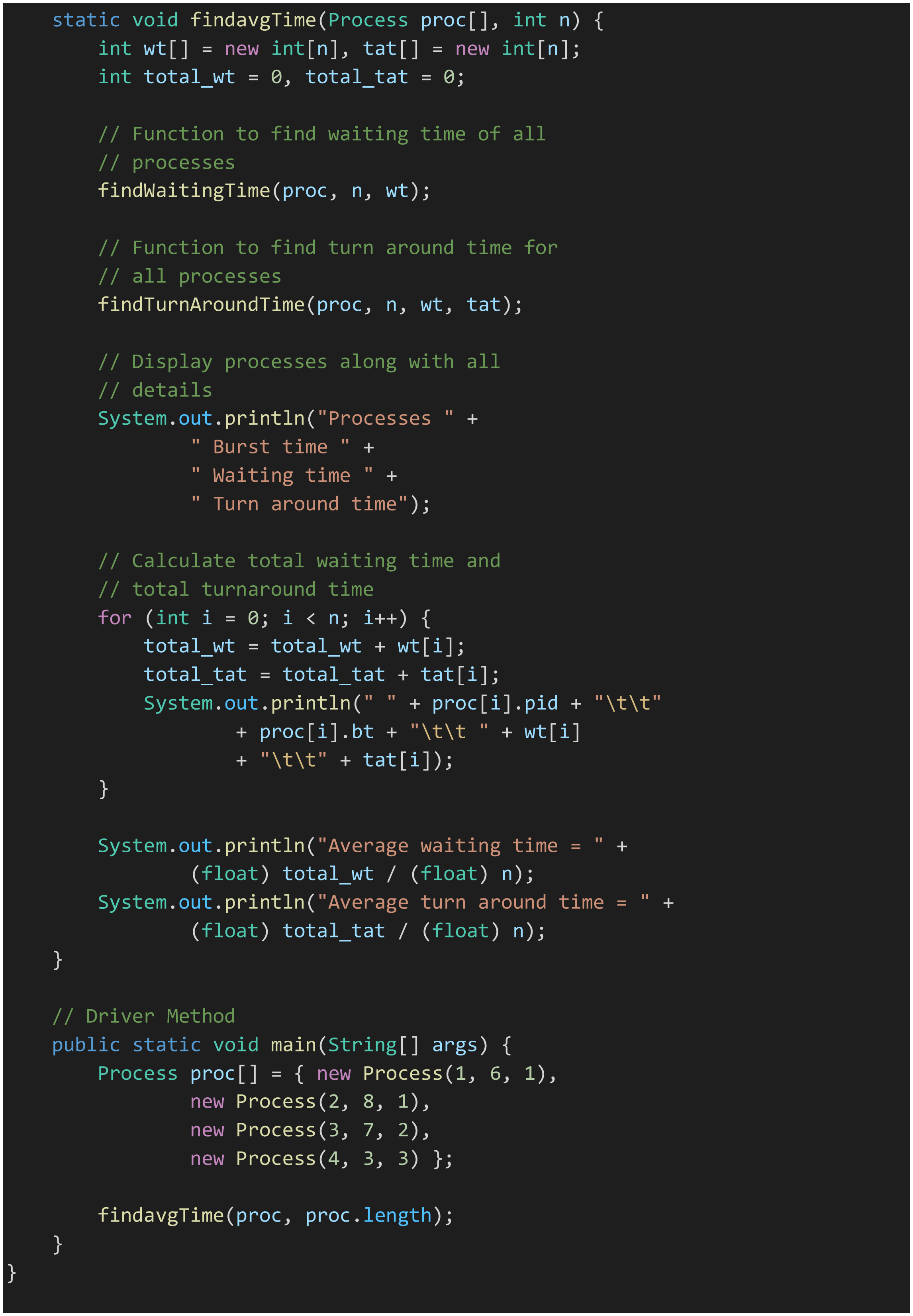
PROGRAM 3: SHORTEST REMAINING TIME FIRST (SRTF)



```
// Java program to implement Shortest Remaining Time First
// Shortest Remaining Time First (SRTF)

class Process {
    int pid; // Process ID
    int bt; // Burst Time
    int art; // Arrival Time

    public Process(int pid, int bt, int art) {
        this.pid = pid;
        this.bt = bt;
        this.art = art;
    }
}

public class SAMPLE {
    // Method to find the waiting time for all
    // processes
    static void findWaitingTime(Process proc[], int n,
            int wt[]) {
        int rt[] = new int[n];

        // Copy the burst time into rt[]
        for (int i = 0; i < n; i++)
            rt[i] = proc[i].bt;

        int complete = 0, t = 0, minm = Integer.MAX_VALUE;
        int shortest = 0, finish_time;
        boolean check = false;

        // Process until all processes gets
        // completed
        while (complete != n) {

            // Find process with minimum
            // remaining time among the
            // processes that arrives till the
            // current time`
            for (int j = 0; j < n; j++) {
                if ((proc[j].art <= t) &&
                        (rt[j] < minm) && rt[j] > 0) {
                    minm = rt[j];
                    shortest = j;
                    check = true;
                }
            }
```

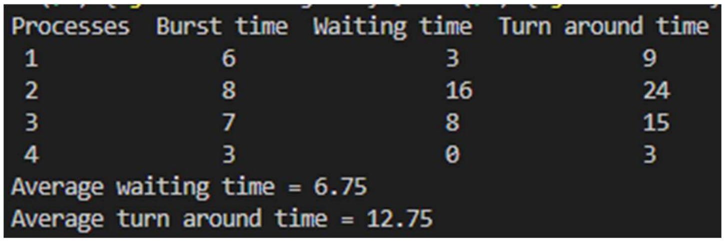


```
            if (check == false) {
                t++;
                continue;
            }

            // Reduce remaining time by one
            rt[shortest]--;

            // Update minimum
            minm = rt[shortest];
            if (minm == 0)
                minm = Integer.MAX_VALUE;

            // If a process gets completely
            // executed
            if (rt[shortest] == 0) {

                // Increment complete
                complete++;
                check = false;

                // Find finish time of current
                // process
                finish_time = t + 1;

                // Calculate waiting time
                wt[shortest] = finish_time -
                        proc[shortest].bt -
                        proc[shortest].art;

                if (wt[shortest] < 0)
                    wt[shortest] = 0;
            }
            // Increment time
            t++;
        }
    }

    // Method to calculate turn around time
    static void findTurnAroundTime(Process proc[], int n,
            int wt[], int tat[]) {
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n; i++)
            tat[i] = proc[i].bt + wt[i];
    }

    // Method to calculate average time
```



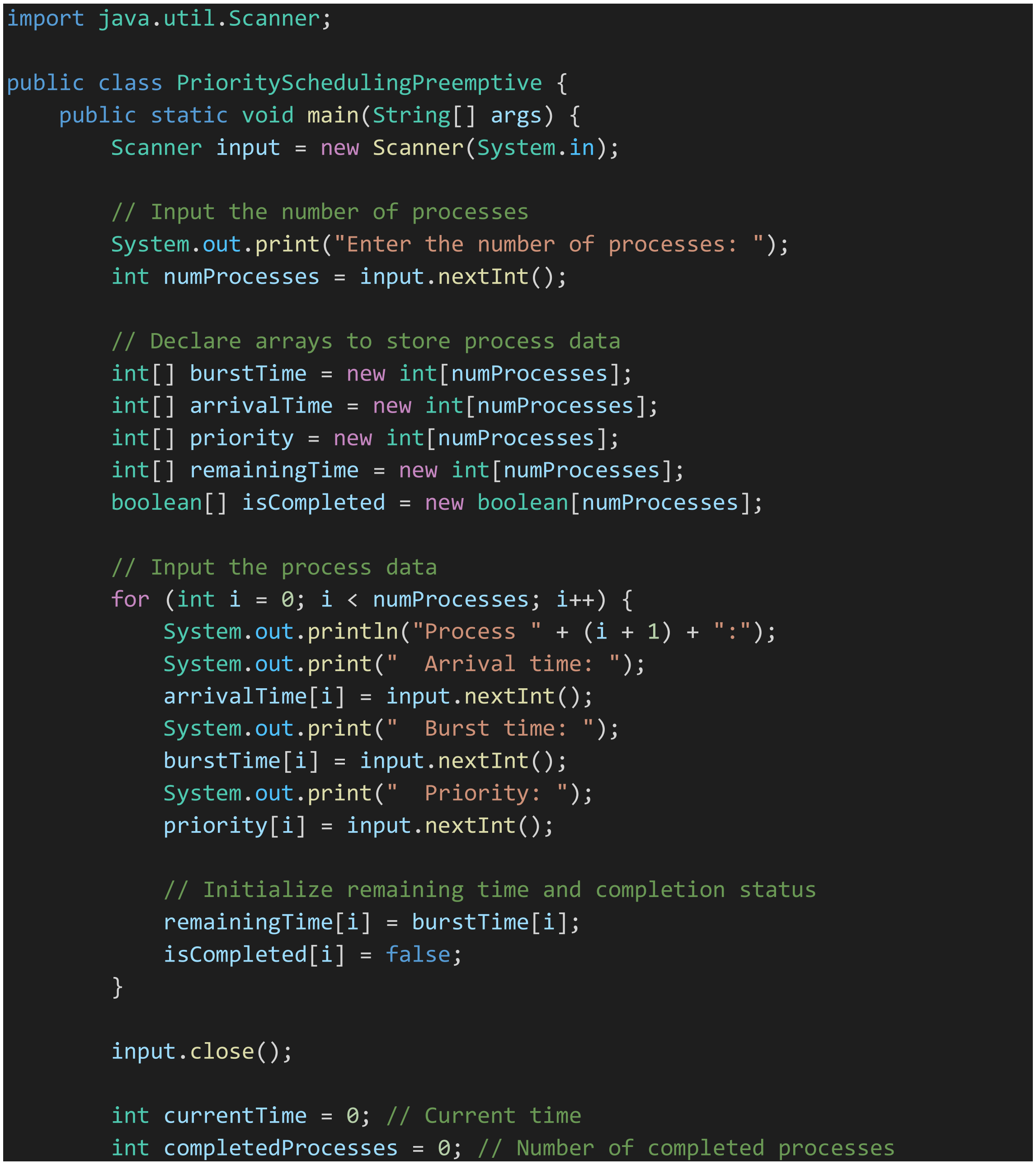
```
    static void findavgTime(Process proc[], int n) {
        int wt[] = new int[n], tat[] = new int[n];
        int total_wt = 0, total_tat = 0;

        // Function to find waiting time of all
        // processes
        findWaitingTime(proc, n, wt);

        // Function to find turn around time for
        // all processes
        findTurnAroundTime(proc, n, wt, tat);

        // Display processes along with all
        // details
        System.out.println("Processes " +
                " Burst time " +
                " Waiting time " +
                " Turn around time");

        // Calculate total waiting time and
        // total turnaround time
        for (int i = 0; i < n; i++) {
            total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i];
            System.out.println(" " + proc[i].pid + "\t\t"
                    + proc[i].bt + "\t\t " + wt[i]
                    + "\t\t" + tat[i]);
        }

        System.out.println("Average waiting time = " +
                (float) total_wt / (float) n);
        System.out.println("Average turn around time = " +
                (float) total_tat / (float) n);
    }

    // Driver Method
    public static void main(String[] args) {
        Process proc[] = { new Process(1, 6, 1),
                new Process(2, 8, 1),
                new Process(3, 7, 2),
                new Process(4, 3, 3) };

        findavgTime(proc, proc.length);
    }
}
```

OUTPUT:

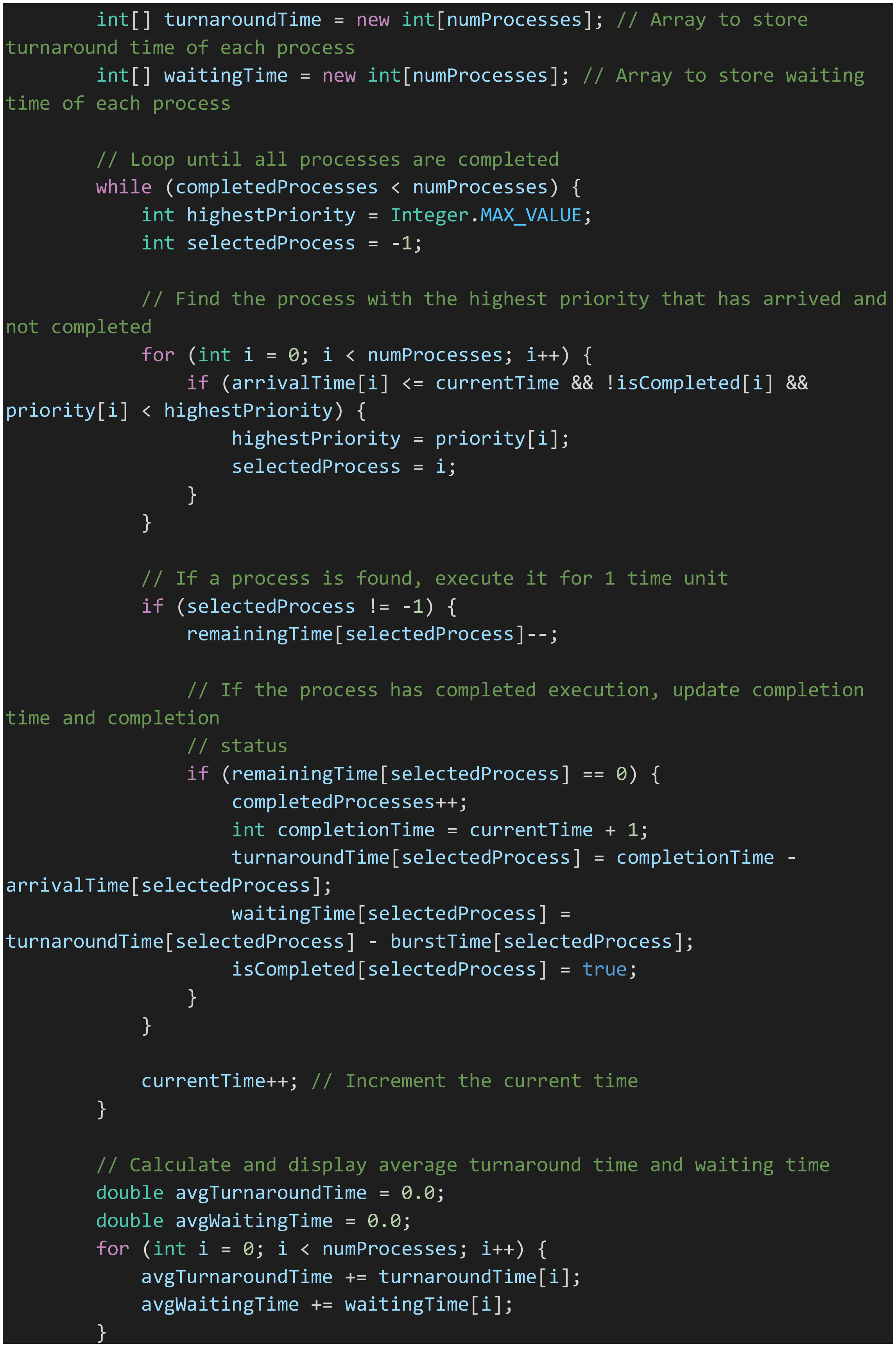


```
Processes  Burst time  Waiting time  Turn around time
 1           6            3             9
 2           8            16            24
 3           7            8             15
 4           3            0             3
Average waiting time = 6.75
Average turn around time = 12.75
```

PROGRAM 4: PRIORITY (PRE-EMPTIVE)

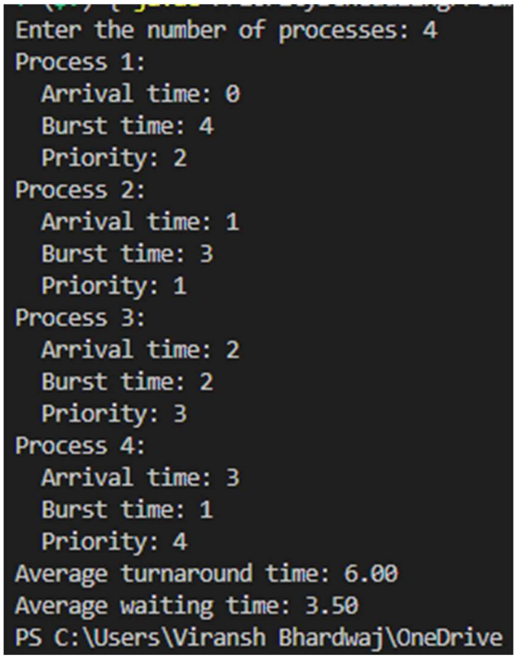


```java
import java.util.Scanner;

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

        // Input the number of processes
        System.out.print("Enter the number of processes: ");
        int numProcesses = input.nextInt();

        // Declare arrays to store process data
        int[] burstTime = new int[numProcesses];
        int[] arrivalTime = new int[numProcesses];
        int[] priority = new int[numProcesses];
        int[] remainingTime = new int[numProcesses];
        boolean[] isCompleted = new boolean[numProcesses];

        // Input the process data
        for (int i = 0; i < numProcesses; i++) {
            System.out.println("Process " + (i + 1) + ":");
            System.out.print("  Arrival time: ");
            arrivalTime[i] = input.nextInt();
            System.out.print("  Burst time: ");
            burstTime[i] = input.nextInt();
            System.out.print("  Priority: ");
            priority[i] = input.nextInt();

            // Initialize remaining time and completion status
            remainingTime[i] = burstTime[i];
            isCompleted[i] = false;
        }

        input.close();

        int currentTime = 0; // Current time
        int completedProcesses = 0; // Number of completed processes
```



```
        int[] turnaroundTime = new int[numProcesses]; // Array to store 
turnaround time of each process
        int[] waitingTime = new int[numProcesses]; // Array to store waiting 
time of each process

        // Loop until all processes are completed
        while (completedProcesses < numProcesses) {
            int highestPriority = Integer.MAX_VALUE;
            int selectedProcess = -1;

            // Find the process with the highest priority that has arrived and 
not completed
            for (int i = 0; i < numProcesses; i++) {
                if (arrivalTime[i] <= currentTime && !isCompleted[i] && 
priority[i] < highestPriority) {
                    highestPriority = priority[i];
                    selectedProcess = i;
                }
            }

            // If a process is found, execute it for 1 time unit
            if (selectedProcess != -1) {
                remainingTime[selectedProcess]--;

                // If the process has completed execution, update completion 
time and completion
                // status
                if (remainingTime[selectedProcess] == 0) {
                    completedProcesses++;
                    int completionTime = currentTime + 1;
                    turnaroundTime[selectedProcess] = completionTime - 
arrivalTime[selectedProcess];
                    waitingTime[selectedProcess] = 
turnaroundTime[selectedProcess] - burstTime[selectedProcess];
                    isCompleted[selectedProcess] = true;
                }
            }

            currentTime++; // Increment the current time
        }

        // Calculate and display average turnaround time and waiting time
        double avgTurnaroundTime = 0.0;
        double avgWaitingTime = 0.0;
        for (int i = 0; i < numProcesses; i++) {
            avgTurnaroundTime += turnaroundTime[i];
            avgWaitingTime += waitingTime[i];
        }
        avgTurnaroundTime /= numProcesses;
        avgWaitingTime /= numProcesses;
        System.out.printf("Average turnaround time: %.2f\n",
avgTurnaroundTime);
        System.out.printf("Average waiting time: %.2f\n", avgWaitingTime);
    }
}
```

OUTPUT:

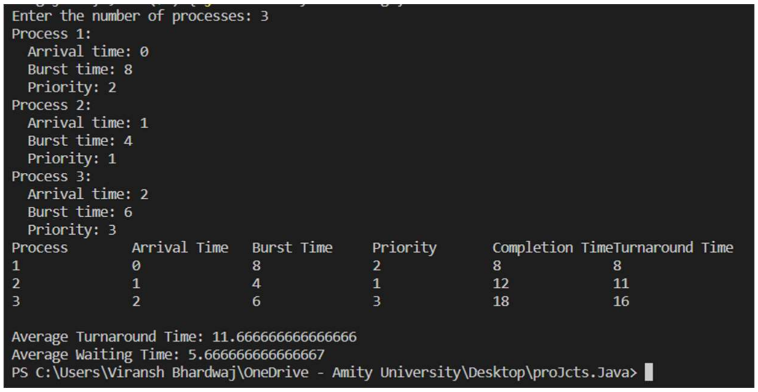


```
Enter the number of processes: 4
Process 1:
  Arrival time: 0
  Burst time: 4
  Priority: 2
Process 2:
  Arrival time: 1
  Burst time: 3
  Priority: 1
Process 3:
  Arrival time: 2
  Burst time: 2
  Priority: 3
Process 4:
  Arrival time: 3
  Burst time: 1
  Priority: 4
Average turnaround time: 6.00
Average waiting time: 3.50
PS C:\Users\Viransh Bhardwaj\OneDrive
```

PROGRAM 5: PRIORITY (NON- PRE-EMPTIVE)

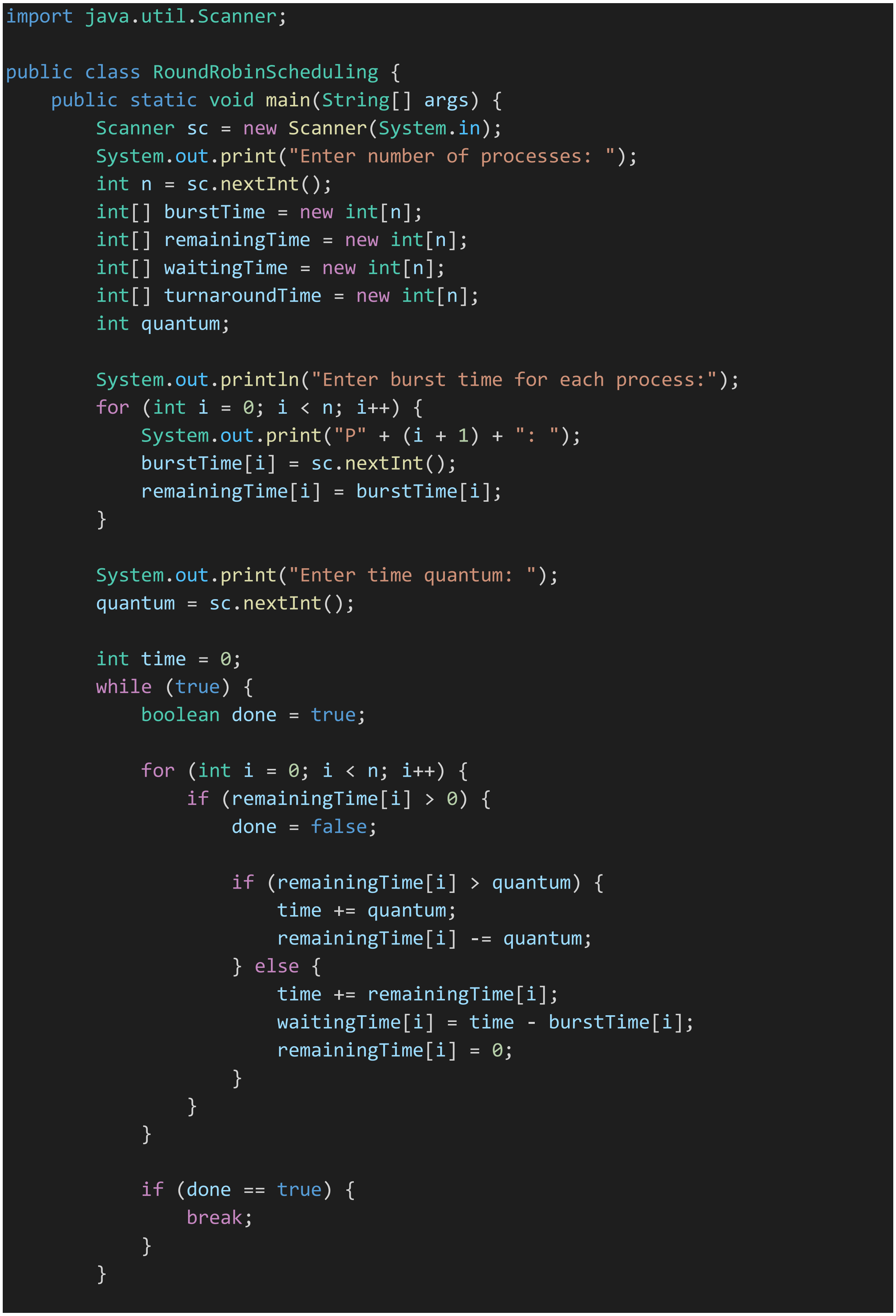
```java
import java.util.*;

public class PriorityScheduling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of processes: ");
        int n = sc.nextInt();

        int[] arrivalTime = new int[n];
        int[] burstTime = new int[n];
        int[] priority = new int[n];
        boolean[] visited = new boolean[n];

        for (int i = 0; i < n; i++) {
            System.out.println("Process " + (i + 1) + ":");
            System.out.print("  Arrival time: ");
            arrivalTime[i] = sc.nextInt();
            System.out.print("  Burst time: ");
            burstTime[i] = sc.nextInt();
            System.out.print("  Priority: ");
            priority[i] = sc.nextInt();
        }

        int totalTime = 0;
        int currentProcess = -1;
        int[] completionTime = new int[n];
        int[] turnaroundTime = new int[n];
        int[] waitingTime = new int[n];

        while (true) {
            int highestPriority = Integer.MAX_VALUE;
            int highestPriorityProcess = -1;

            for (int i = 0; i < n; i++) {
                if (!visited[i] && arrivalTime[i] <= totalTime && priority[i]
< highestPriority) {
                    highestPriority = priority[i];
                    highestPriorityProcess = i;
                }
            }

            if (highestPriorityProcess == -1) {
                break;
            }

            visited[highestPriorityProcess] = true;
            currentProcess = highestPriorityProcess;
            completionTime[currentProcess] = totalTime +
burstTime[currentProcess];
            turnaroundTime[currentProcess] = completionTime[currentProcess] -
arrivalTime[currentProcess];
            waitingTime[currentProcess] = turnaroundTime[currentProcess] -
burstTime[currentProcess];

            totalTime += burstTime[currentProcess];
        }

        double totalTurnaroundTime = 0;
        double totalWaitingTime = 0;

        System.out.printf("%-15s%-15s%-15s%-15s%-15s%-15s\n", "Process",
"Arrival Time", "Burst Time", "Priority",
                "Completion Time", "Turnaround Time");
        for (int i = 0; i < n; i++) {
            System.out.printf("%-15d%-15d%-15d%-15d%-15d%-15d\n", (i + 1),
arrivalTime[i], burstTime[i], priority[i],
                    completionTime[i], turnaroundTime[i]);
            totalTurnaroundTime += turnaroundTime[i];
            totalWaitingTime += waitingTime[i];
        }

        System.out.println("\nAverage Turnaround Time: " +
(totalTurnaroundTime / n));
        System.out.println("Average Waiting Time: " + (totalWaitingTime / n));
    }
}
```

OUTPUT:

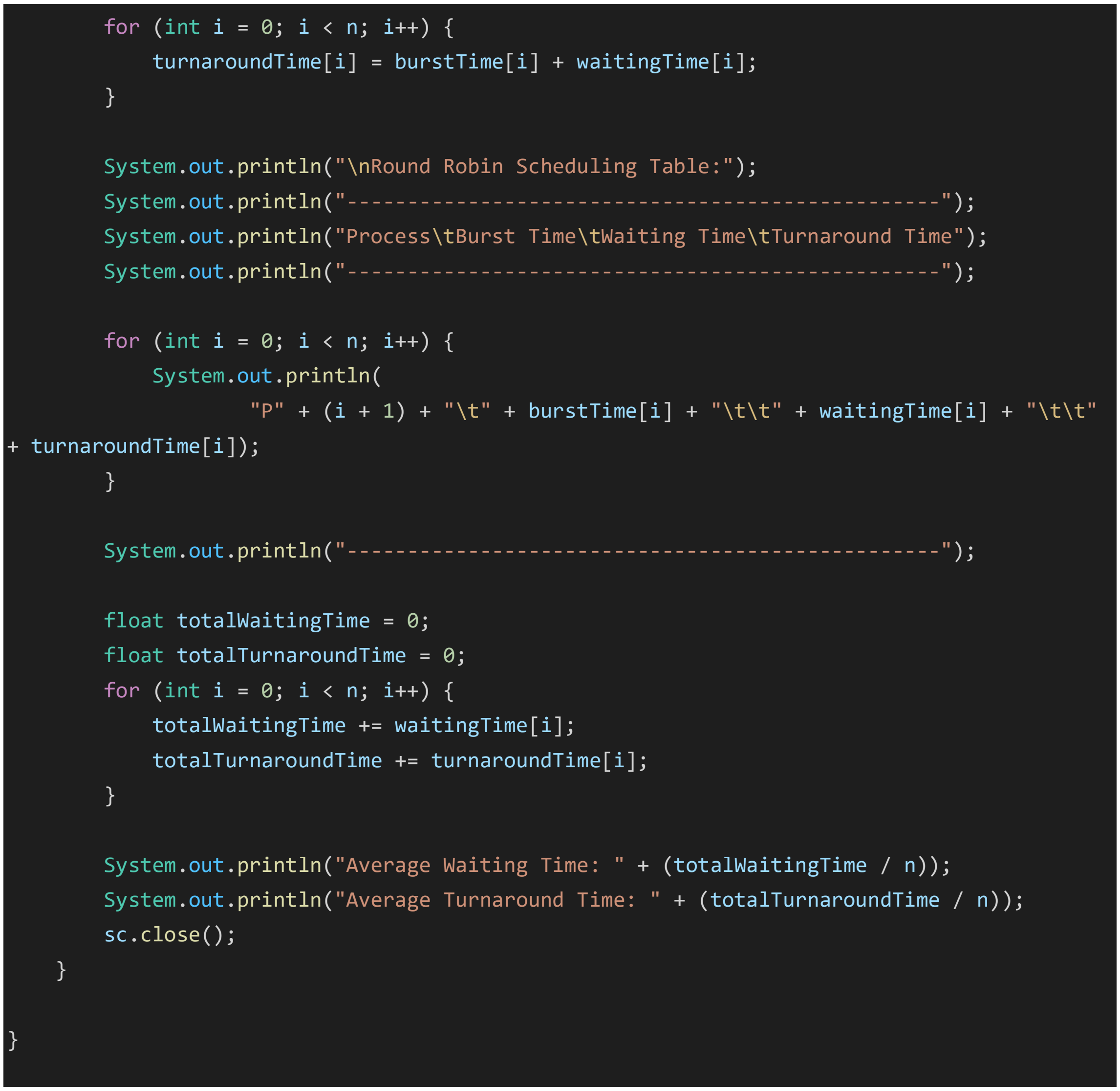


```
Enter the number of processes: 3
Process 1:
  Arrival time: 0
  Burst time: 8
  Priority: 2
Process 2:
  Arrival time: 1
  Burst time: 4
  Priority: 1
Process 3:
  Arrival time: 2
  Burst time: 6
  Priority: 3
Process        Arrival Time   Burst Time     Priority       Completion TimeTurnaround Time
1              0              8              2              8              8
2              1              4              1              12             11
3              2              6              3              18             16

Average Turnaround Time: 11.666666666666666
Average Waiting Time: 5.666666666666667
PS C:\Users\Viransh Bhardwaj\OneDrive - Amity University\Desktop\proJcts.Java> 
```

PROGRAM 6: ROUND ROBIN

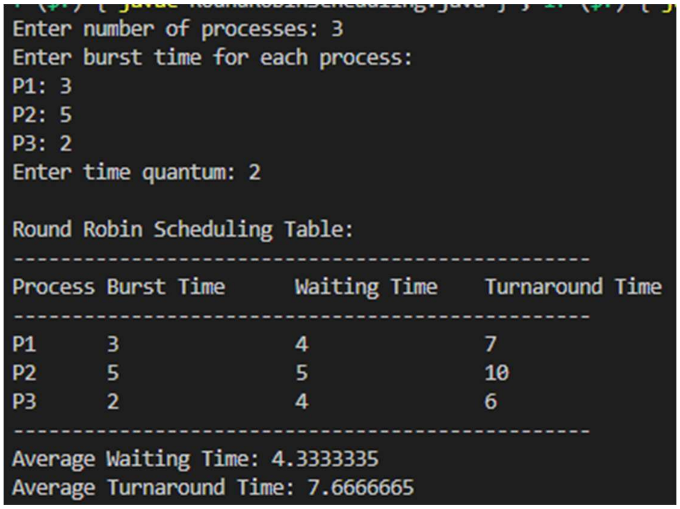


```java
import java.util.Scanner;

public class RoundRobinScheduling {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of processes: ");
        int n = sc.nextInt();
        int[] burstTime = new int[n];
        int[] remainingTime = new int[n];
        int[] waitingTime = new int[n];
        int[] turnaroundTime = new int[n];
        int quantum;

        System.out.println("Enter burst time for each process:");
        for (int i = 0; i < n; i++) {
            System.out.print("P" + (i + 1) + ": ");
            burstTime[i] = sc.nextInt();
            remainingTime[i] = burstTime[i];
        }

        System.out.print("Enter time quantum: ");
        quantum = sc.nextInt();

        int time = 0;
        while (true) {
            boolean done = true;

            for (int i = 0; i < n; i++) {
                if (remainingTime[i] > 0) {
                    done = false;

                    if (remainingTime[i] > quantum) {
                        time += quantum;
                        remainingTime[i] -= quantum;
                    } else {
                        time += remainingTime[i];
                        waitingTime[i] = time - burstTime[i];
                        remainingTime[i] = 0;
                    }
                }
            }

            if (done == true) {
                break;
            }
        }
```



```
        for (int i = 0; i < n; i++) {
            turnaroundTime[i] = burstTime[i] + waitingTime[i];
        }

        System.out.println("\nRound Robin Scheduling Table:");
        System.out.println("-------------------------------------------------");
        System.out.println("Process\tBurst Time\tWaiting Time\tTurnaround Time");
        System.out.println("-------------------------------------------------");

        for (int i = 0; i < n; i++) {
            System.out.println(
                    "P" + (i + 1) + "\t" + burstTime[i] + "\t\t" + waitingTime[i] + "\t\t"
+ turnaroundTime[i]);
        }

        System.out.println("-------------------------------------------------");

        float totalWaitingTime = 0;
        float totalTurnaroundTime = 0;
        for (int i = 0; i < n; i++) {
            totalWaitingTime += waitingTime[i];
            totalTurnaroundTime += turnaroundTime[i];
        }

        System.out.println("Average Waiting Time: " + (totalWaitingTime / n));
        System.out.println("Average Turnaround Time: " + (totalTurnaroundTime / n));
        sc.close();
    }

}
```

OUTPUT:

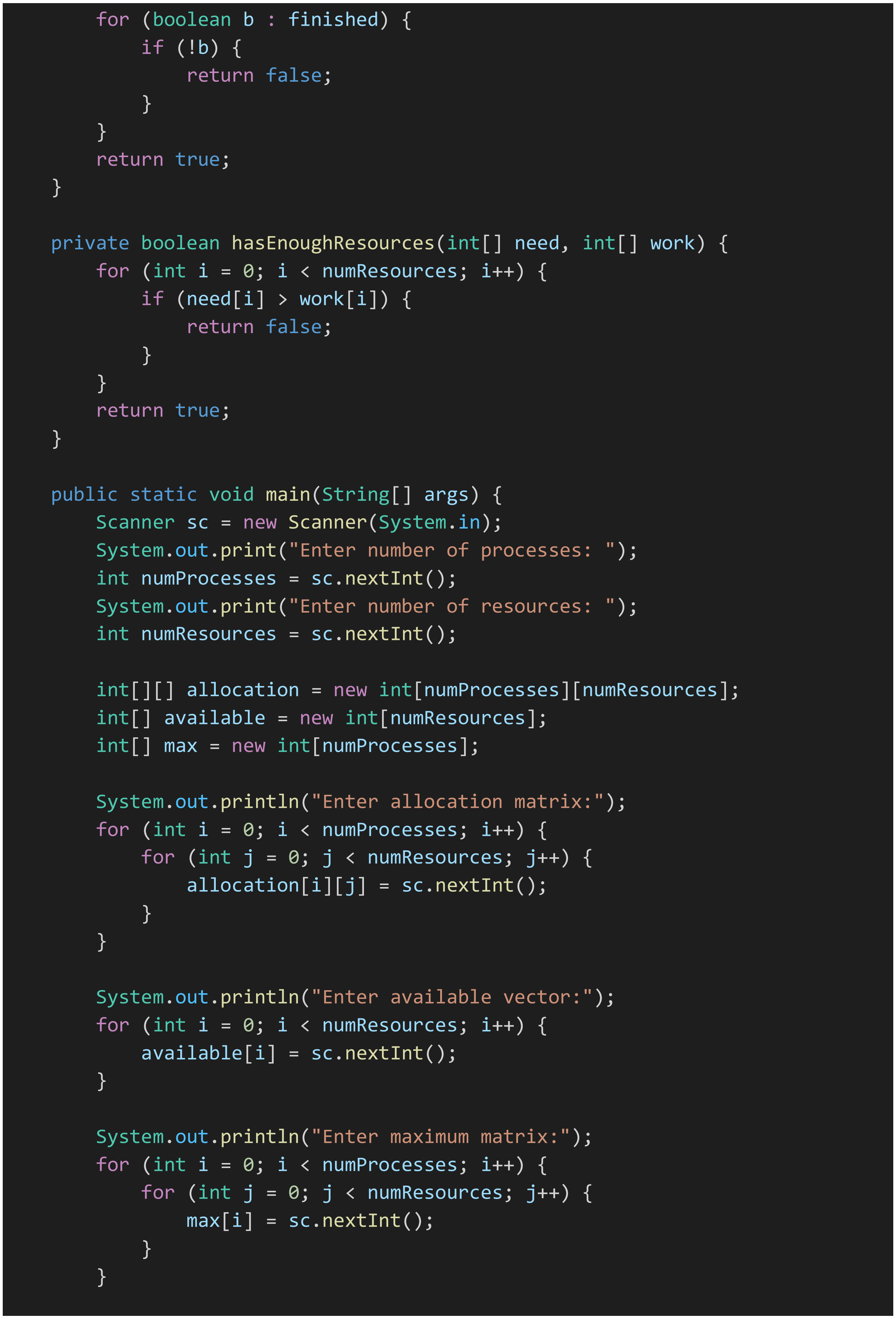


```
Enter number of processes: 3
Enter burst time for each process:
P1: 3
P2: 5
P3: 2
Enter time quantum: 2

Round Robin Scheduling Table:
-------------------------------------------------
Process Burst Time      Waiting Time    Turnaround Time
-------------------------------------------------
P1      3               4               7
P2      5               5               10
P3      2               4               6
-------------------------------------------------
Average Waiting Time: 4.3333335
Average Turnaround Time: 7.6666665
```

BANKER'S ALGORITHM

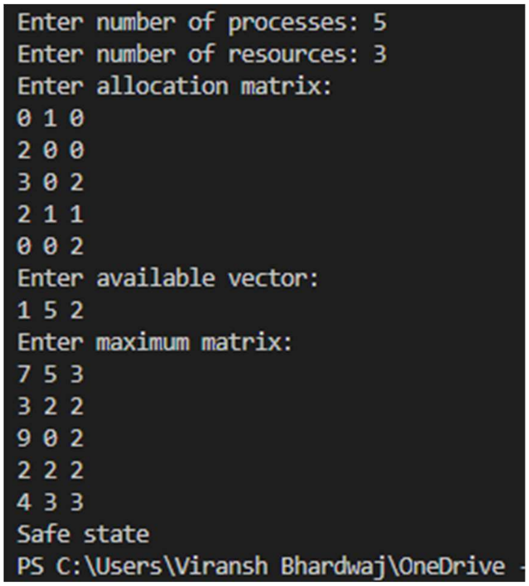
PROGRAM:

```java
import java.util.*;

public class BankersAlgorithm {
    private int[][] need; // Need matrix
    private int[][] allocation; // Allocation matrix
    private int[] available; // Available resource vector
    private int[] max; // Maximum resource vector
    private int numProcesses; // Number of processes
    private int numResources; // Number of resource types

    public BankersAlgorithm(int[][] allocation, int[] available, int[] max) {
        this.allocation = allocation;
        this.available = available;
        this.max = max;
        this.numProcesses = allocation.length;
        this.numResources = available.length;
        this.need = new int[numProcesses][numResources];
        for (int i = 0; i < numProcesses; i++) {
            for (int j = 0; j < numResources; j++) {
                need[i][j] = max[i] - allocation[i][j];
            }
        }
    }

    public boolean isSafeState() {
        boolean[] finished = new boolean[numProcesses];
        int[] work = Arrays.copyOf(available, numResources);

        while (true) {
            boolean foundProcess = false;
            for (int i = 0; i < numProcesses; i++) {
                if (!finished[i] && hasEnoughResources(need[i], work)) {
                    foundProcess = true;
                    finished[i] = true;
                    for (int j = 0; j < numResources; j++) {
                        work[j] += allocation[i][j];
                    }
                }
            }
            if (!foundProcess) {
                break;
            }
        }
```



```
        for (boolean b : finished) {
            if (!b) {
                return false;
            }
        }
        return true;
    }

    private boolean hasEnoughResources(int[] need, int[] work) {
        for (int i = 0; i < numResources; i++) {
            if (need[i] > work[i]) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of processes: ");
        int numProcesses = sc.nextInt();
        System.out.print("Enter number of resources: ");
        int numResources = sc.nextInt();

        int[][] allocation = new int[numProcesses][numResources];
        int[] available = new int[numResources];
        int[] max = new int[numProcesses];

        System.out.println("Enter allocation matrix:");
        for (int i = 0; i < numProcesses; i++) {
            for (int j = 0; j < numResources; j++) {
                allocation[i][j] = sc.nextInt();
            }
        }

        System.out.println("Enter available vector:");
        for (int i = 0; i < numResources; i++) {
            available[i] = sc.nextInt();
        }

        System.out.println("Enter maximum matrix:");
        for (int i = 0; i < numProcesses; i++) {
            for (int j = 0; j < numResources; j++) {
                max[i] = sc.nextInt();
            }
        }
        BankersAlgorithm ba = new BankersAlgorithm(allocation, available, 
max);

        if (ba.isSafeState()) {
            System.out.println("Safe state");
        } else {
            System.out.println("Unsafe state");
        }
        sc.close();
    }
}
```

OUTPUT:



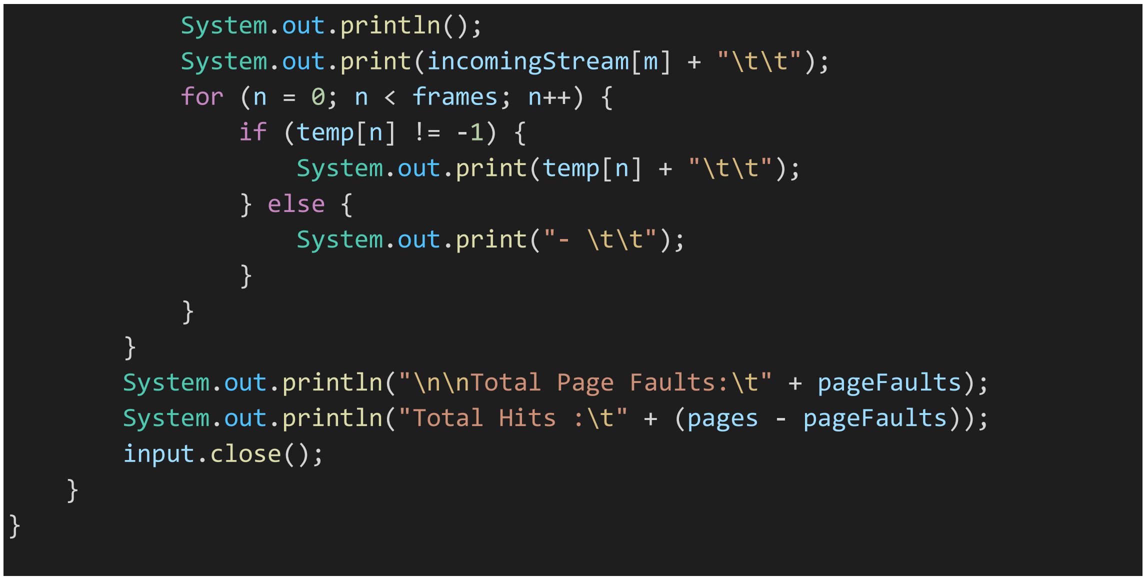
```
Enter number of processes: 5
Enter number of resources: 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter available vector:
1 5 2
Enter maximum matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Safe state
PS C:\Users\Viransh Bhardwaj\OneDrive -
```

PAGE REPLACEMENT ALGORITHMS

1. First In First Out (FIFO)
2. Optimal Page Replacement (ORU)
3. Least Recently Used (LRU)

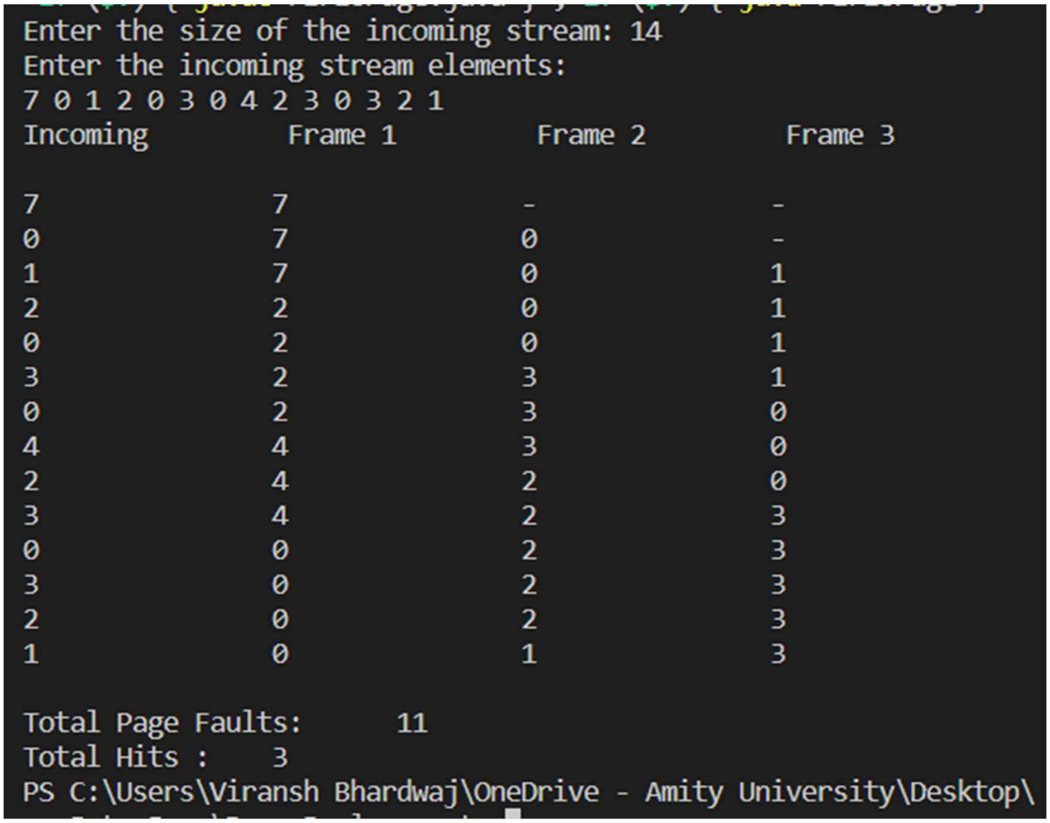
PROGRAMS 1: FIRST COME FIRST SERVE (FCFS)

```java
import java.util.Scanner;

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

        System.out.print("Enter the size of the incoming stream: ");
        int size = input.nextInt();

        int[] incomingStream = new int[size];

        System.out.println("Enter the incoming stream elements:");
        for (int i = 0; i < size; i++) {
            incomingStream[i] = input.nextInt();
        }

        int pageFaults = 0;
        int frames = 3;
        int m, n, s, pages;
        pages = incomingStream.length;
        System.out.println("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
        int[] temp = new int[frames];
        for (m = 0; m < frames; m++) {
            temp[m] = -1;
        }
        for (m = 0; m < pages; m++) {
            s = 0;
            for (n = 0; n < frames; n++) {
                if (incomingStream[m] == temp[n]) {
                    s++;
                    pageFaults--;
                }
            }
            pageFaults++;
            if ((pageFaults <= frames) && (s == 0)) {
                temp[m] = incomingStream[m];
            } else if (s == 0) {
                temp[(pageFaults - 1) % frames] = incomingStream[m];
            }
```



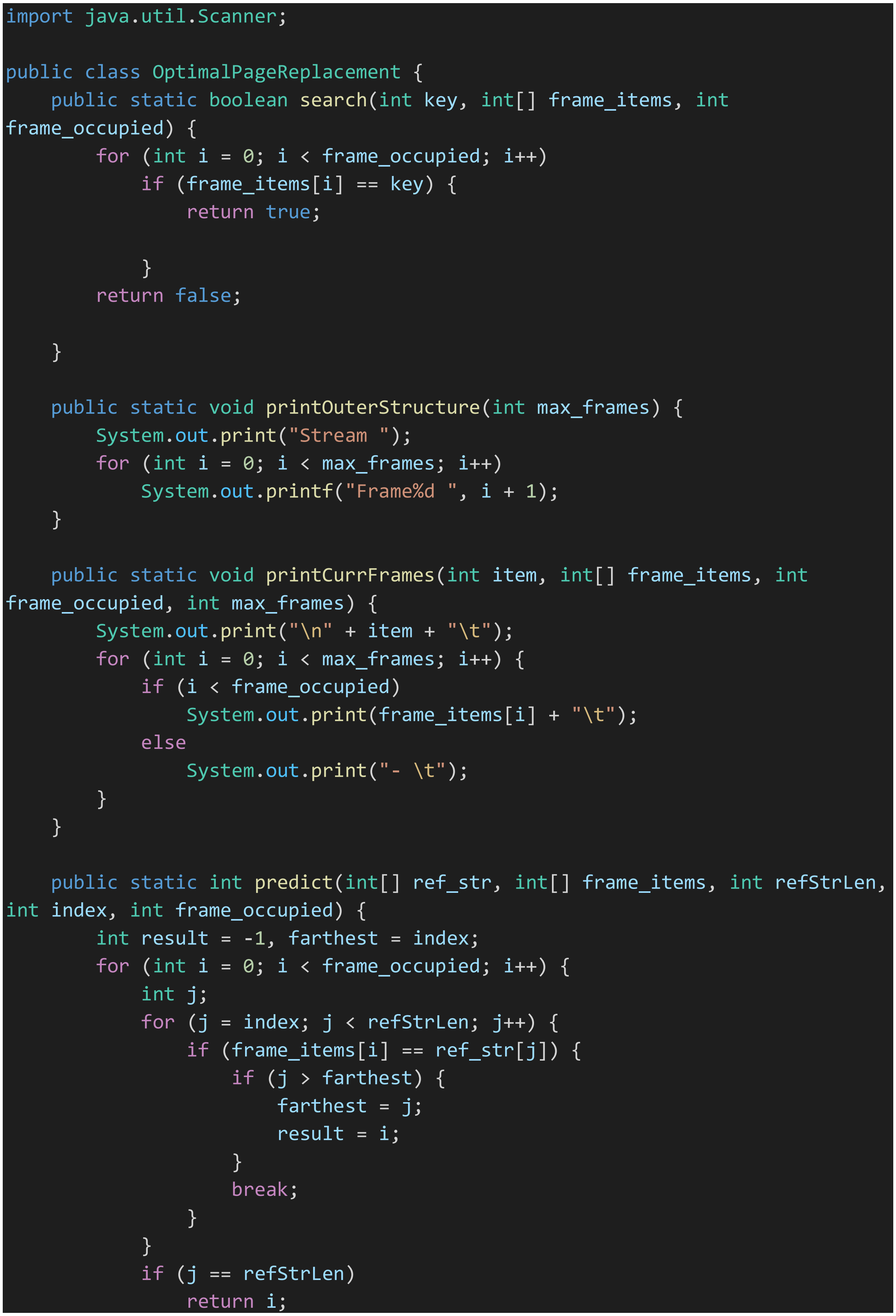
```
            System.out.println();
            System.out.print(incomingStream[m] + "\t\t");
            for (n = 0; n < frames; n++) {
                if (temp[n] != -1) {
                    System.out.print(temp[n] + "\t\t");
                } else {
                    System.out.print("- \t\t");
                }
            }
        }
        System.out.println("\n\nTotal Page Faults:\t" + pageFaults);
        System.out.println("Total Hits :\t" + (pages - pageFaults));
        input.close();
    }
}
```

OUTPUT:



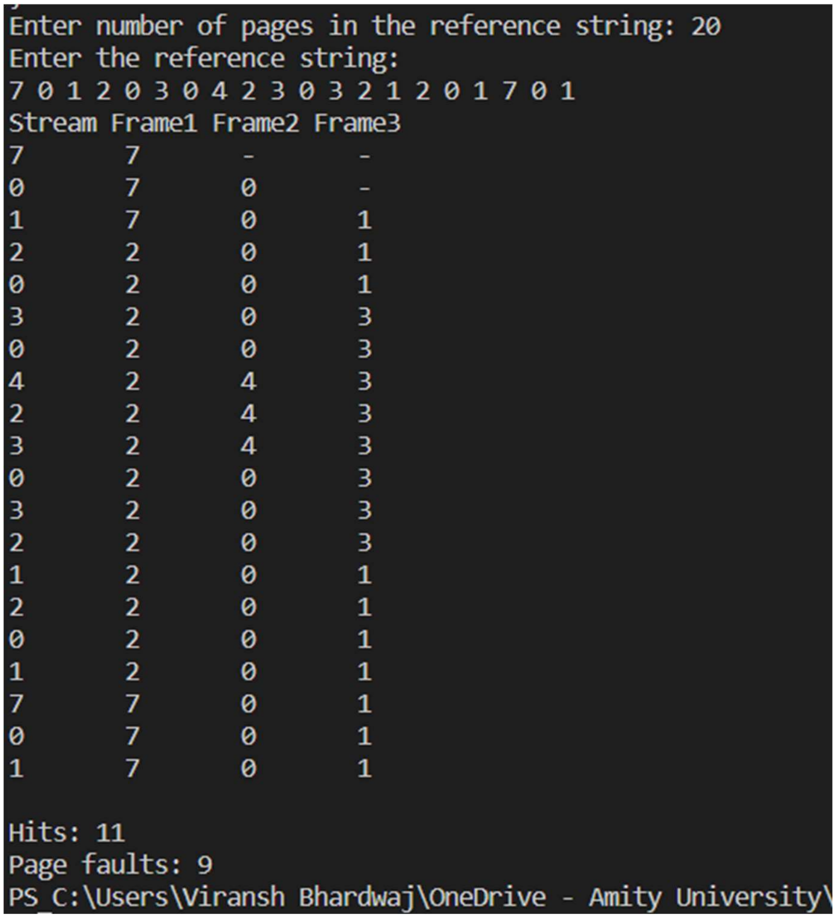
```
Enter the size of the incoming stream: 14
Enter the incoming stream elements:
7 0 1 2 0 3 0 4 2 3 0 3 2 1
Incoming         Frame 1         Frame 2         Frame 3

7               7               -               -
0               7               0               -
1               7               0               1
2               2               0               1
0               2               0               1
3               2               3               1
0               2               3               0
4               4               3               0
2               4               2               0
3               4               2               3
0               0               2               3
3               0               2               3
2               0               2               3
1               0               1               3

Total Page Faults:      11
Total Hits :    3
PS C:\Users\Viransh Bhardwaj\OneDrive - Amity University\Desktop\
```

PROGRAMS 2: OPTIMAL PAGE REPLACEMENT(OPR)



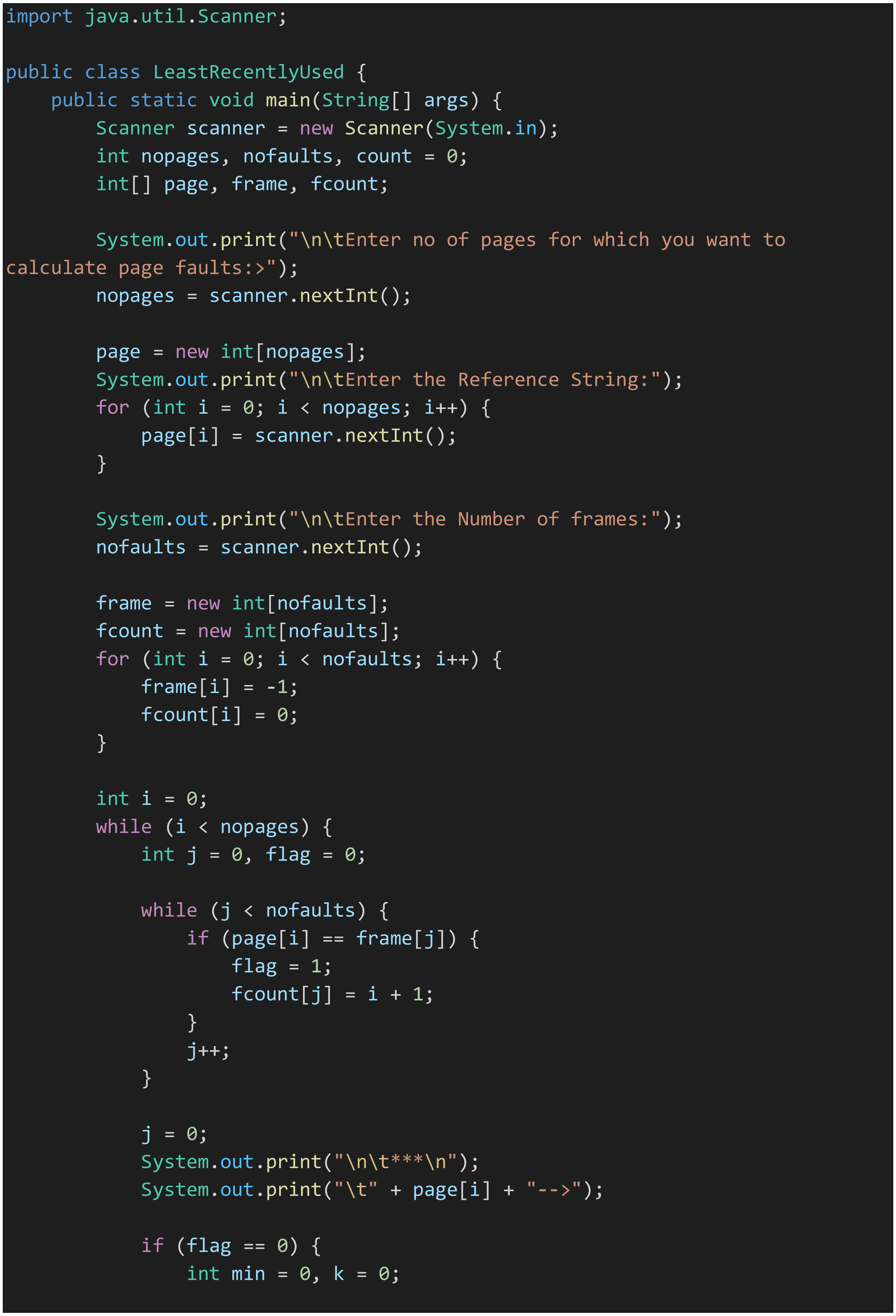
```java
import java.util.Scanner;

public class OptimalPageReplacement {
    public static boolean search(int key, int[] frame_items, int
frame_occupied) {
        for (int i = 0; i < frame_occupied; i++)
            if (frame_items[i] == key) {
                return true;

            }
        return false;

    }

    public static void printOuterStructure(int max_frames) {
        System.out.print("Stream ");
        for (int i = 0; i < max_frames; i++)
            System.out.printf("Frame%d ", i + 1);
    }

    public static void printCurrFrames(int item, int[] frame_items, int
frame_occupied, int max_frames) {
        System.out.print("\n" + item + "\t");
        for (int i = 0; i < max_frames; i++) {
            if (i < frame_occupied)
                System.out.print(frame_items[i] + "\t");
            else
                System.out.print("- \t");
        }
    }

    public static int predict(int[] ref_str, int[] frame_items, int refStrLen,
int index, int frame_occupied) {
        int result = -1, farthest = index;
        for (int i = 0; i < frame_occupied; i++) {
            int j;
            for (j = index; j < refStrLen; j++) {
                if (frame_items[i] == ref_str[j]) {
                    if (j > farthest) {
                        farthest = j;
                        result = i;
                    }
                    break;
                }
            }
            if (j == refStrLen)
                return i;
        }
        return (result == -1) ? 0 : result;
    }

    public static void optimalPage(int[] ref_str, int refStrLen, int[]
frame_items, int max_frames) {
        int frame_occupied = 0;
        printOuterStructure(max_frames);
        int hits = 0;
        for (int i = 0; i < refStrLen; i++) {
            if (search(ref_str[i], frame_items, frame_occupied)) {
                hits++;
                printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);
                continue;
            }
            if (frame_occupied < max_frames) {
                frame_items[frame_occupied] = ref_str[i];
                frame_occupied++;
                printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);
            } else {
                int pos = predict(ref_str, frame_items, refStrLen, i + 1,
frame_occupied);
                frame_items[pos] = ref_str[i];
                printCurrFrames(ref_str[i], frame_items, frame_occupied,
max_frames);
            }

        }
        System.out.println("\n\nHits: " + hits);
        System.out.println("Page faults: " + (refStrLen - hits));
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of pages in the reference string: ");
        int refStrLen = sc.nextInt();
        int[] ref_str = new int[refStrLen];
        System.out.println("Enter the reference string:");
        for (int i = 0; i < refStrLen; i++) {
            ref_str[i] = sc.nextInt();
        }
        int max_frames = 3;
        int[] frame_items = new int[max_frames];

        optimalPage(ref_str, refStrLen, frame_items, max_frames);
        sc.close();
    }
}
```

OUTPUT:

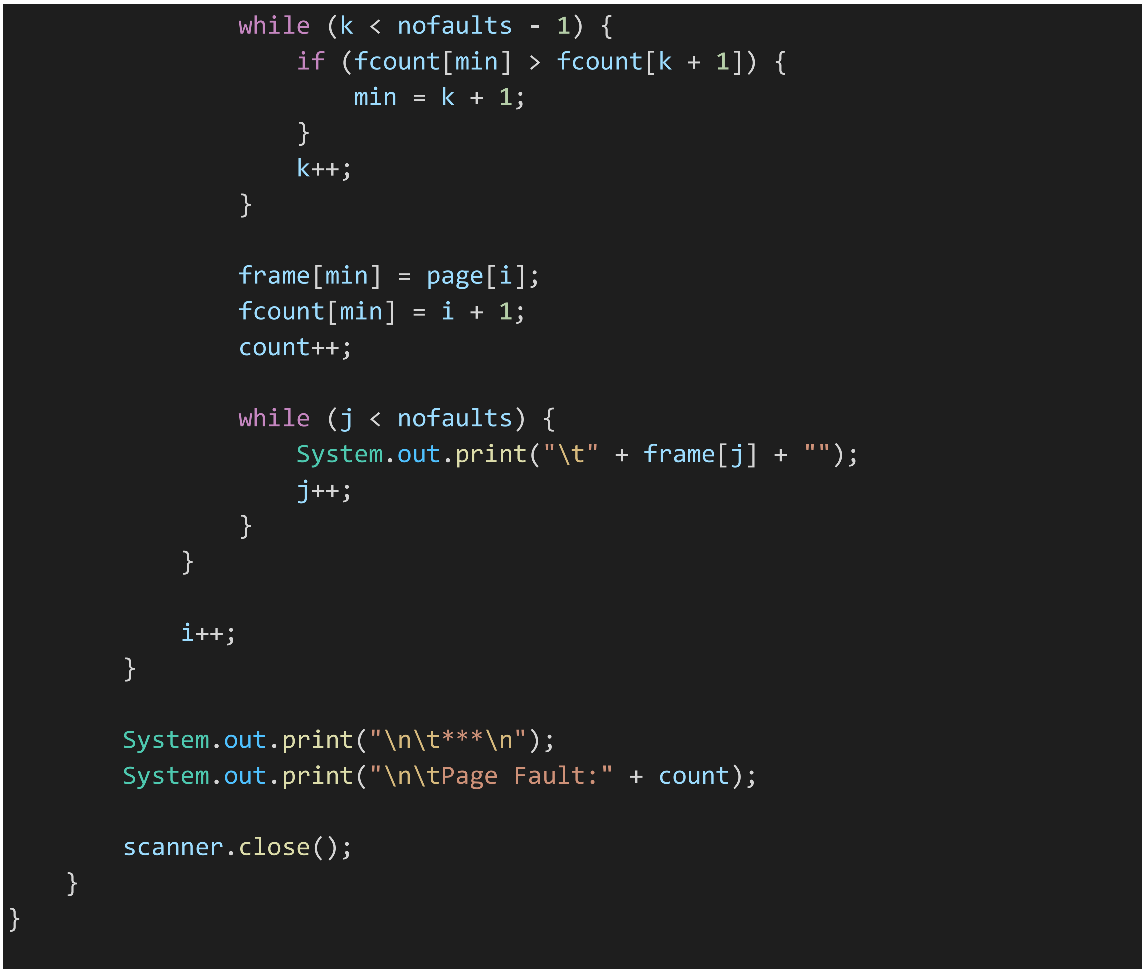


```
Enter number of pages in the reference string: 20
Enter the reference string:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Stream Frame1 Frame2 Frame3
7       7       -       -
0       7       0       -
1       7       0       1
2       2       0       1
0       2       0       1
3       2       0       3
0       2       0       3
4       2       4       3
2       2       4       3
3       2       4       3
0       2       0       3
3       2       0       3
2       2       0       3
1       2       0       1
2       2       0       1
0       2       0       1
1       2       0       1
7       7       0       1
0       7       0       1
1       7       0       1

Hits: 11
Page faults: 9
PS C:\Users\Viransh Bhardwaj\OneDrive - Amity University\
```

PROGRAMS 3: LEAST RECENTLY USED (LRU)

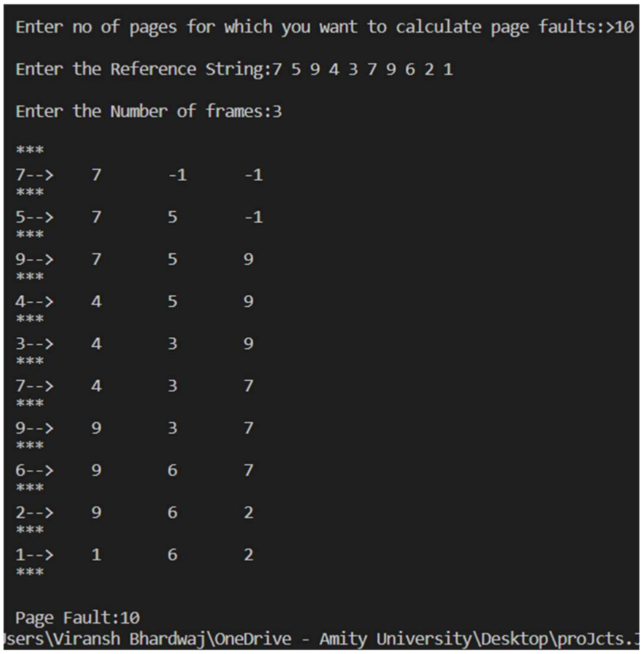


```java
import java.util.Scanner;

public class LeastRecentlyUsed {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int nopages, nofaults, count = 0;
        int[] page, frame, fcount;

        System.out.print("\n\tEnter no of pages for which you want to
calculate page faults:>");
        nopages = scanner.nextInt();

        page = new int[nopages];
        System.out.print("\n\tEnter the Reference String:");
        for (int i = 0; i < nopages; i++) {
            page[i] = scanner.nextInt();
        }

        System.out.print("\n\tEnter the Number of frames:");
        nofaults = scanner.nextInt();

        frame = new int[nofaults];
        fcount = new int[nofaults];
        for (int i = 0; i < nofaults; i++) {
            frame[i] = -1;
            fcount[i] = 0;
        }

        int i = 0;
        while (i < nopages) {
            int j = 0, flag = 0;

            while (j < nofaults) {
                if (page[i] == frame[j]) {
                    flag = 1;
                    fcount[j] = i + 1;
                }
                j++;
            }

            j = 0;
            System.out.print("\n\t***\n");
            System.out.print("\t" + page[i] + "-->");

            if (flag == 0) {
                int min = 0, k = 0;
```



```
                while (k < nofaults - 1) {
                    if (fcount[min] > fcount[k + 1]) {
                        min = k + 1;
                    }
                    k++;
                }

                frame[min] = page[i];
                fcount[min] = i + 1;
                count++;

                while (j < nofaults) {
                    System.out.print("\t" + frame[j] + "");
                    j++;
                }
            }

            i++;
        }

        System.out.print("\n\t***\n");
        System.out.print("\n\tPage Fault:" + count);

        scanner.close();
    }
}
```

OUTPUT:



```
Enter no of pages for which you want to calculate page faults:>10

Enter the Reference String:7 5 9 4 3 7 9 6 2 1

Enter the Number of frames:3

***
7-->    7       -1      -1
***
5-->    7       5       -1
***
9-->    7       5       9
***
4-->    4       5       9
***
3-->    4       3       9
***
7-->    4       3       7
***
9-->    9       3       7
***
6-->    9       6       7
***
2-->    9       6       2
***
1-->    1       6       2
***


Page Fault:10
sers\Viransh Bhardwaj\OneDrive - Amity University\Desktop\proJcts.J
```