Page No.
Date: / /

SPOSL C2

Title: Implementation of Banker's Algorithm.

Problem Statement: Write a JAVA program to implement Banker's Algorithm.

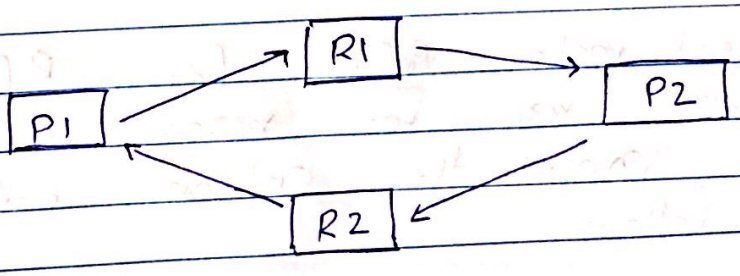
Objective: To study the algorithm for finding and whether a system is safe & study resource request for deadlock

Learning Outcomes: The student will be able to implement deadlock avoidance algorithm resource allocation sequences & demonstrate limitations of deadlock avoidance alogorithm.

Theory:

Deadlock:

A set of procedures are in deadlock when every procers is set in waiting for an event that can only cause by another procers in the set is waiting for an event. As shown in the diagram P1 is holding R2 & requesting R1. P2 is holding R1 & requesing R2



Page No.
Date: / /

Four basic conditions for deadlock to happen:
1. Mutual exclusion - at least one resource must be used in non-share
2. Hold & wait : There must be a process holding one resource & waiting for another.
3. No premption : Resource cant be preempted.

Approaches used to avoid deadlock are:
1) Deadlock avoidance
2) Deadlock prevention
3) Deadlock Detection & Recovering

Bankers algorithm is a deadlock avoidance algorithm. When a new process enters the system it must declare the max number of instanas of each type of resource that it may need.

Steps:
- Input matrix (C) & allocation mat. (A)
- Calculate (C - A) & avaliable vector (V)
- Test for safety condition
- Decide if to allocate resources.

Request vector : Req[i] for R[i]
When req. for resource is made by process the actions will be as follows

If req[i] ≤ Need [i] goto step 2 otherwise an excess condition since process has exceeded it's max claim
If req[i] ≤ available go ahead or else wait
How system pretend to have allocated the req resources
Available = Available − Req [i]
Allocation :- Allocation [i] - Req [i]

If safe then transaction is completed
However if unsafe the P[i] must wait for req [i] & old resource allocation state is restored

* Test Cases:
No. of Processes = 5
No. of resources = 4

Allocation Mat

	A	B	C	D
P_0	0	0	1	2
P_1	1	0	0	0
P_2	1	3	5	4
P_3	0	6	3	2
P_4	0	e	1	4

Total resources: [3 14 12 12]

Page No.
Date: / /

Max matrix

$$\begin{bmatrix} 0 & 0 & 1 & 2 \\ 1 & 7 & 5 & 0 \\ 2 & 3 & 5 & 6 \\ 0 & 6 & 5 & 2 \\ 0 & 0 & 5 & 6 \end{bmatrix}$$

Output Sequence → P_0 P_2 P_1 P_3 P_4
Safe process

max matrix

$$\begin{bmatrix} 6 & 0 & 1 & 2 \\ 1 & 7 & 5 & 0 \\ 2 & 3 & 5 & 6 \\ 1 & 6 & 5 & 2 \\ 0 & 6 & 5 & 50 \end{bmatrix}$$

Output Sequence → P_0 P_2 P_1 P_3
Not Safe Process

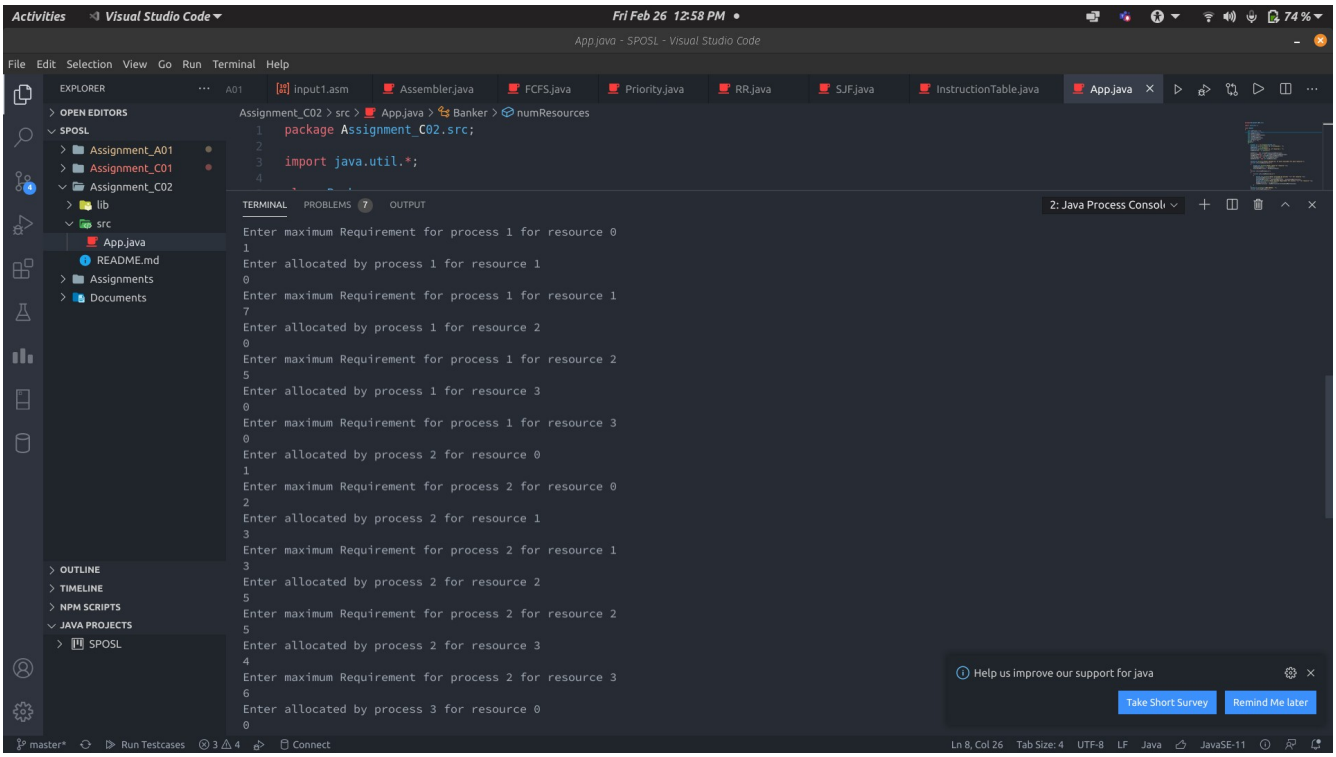
* Conclusion:
Thus we successfully implemented bankers algorithm to avoid deadlock.

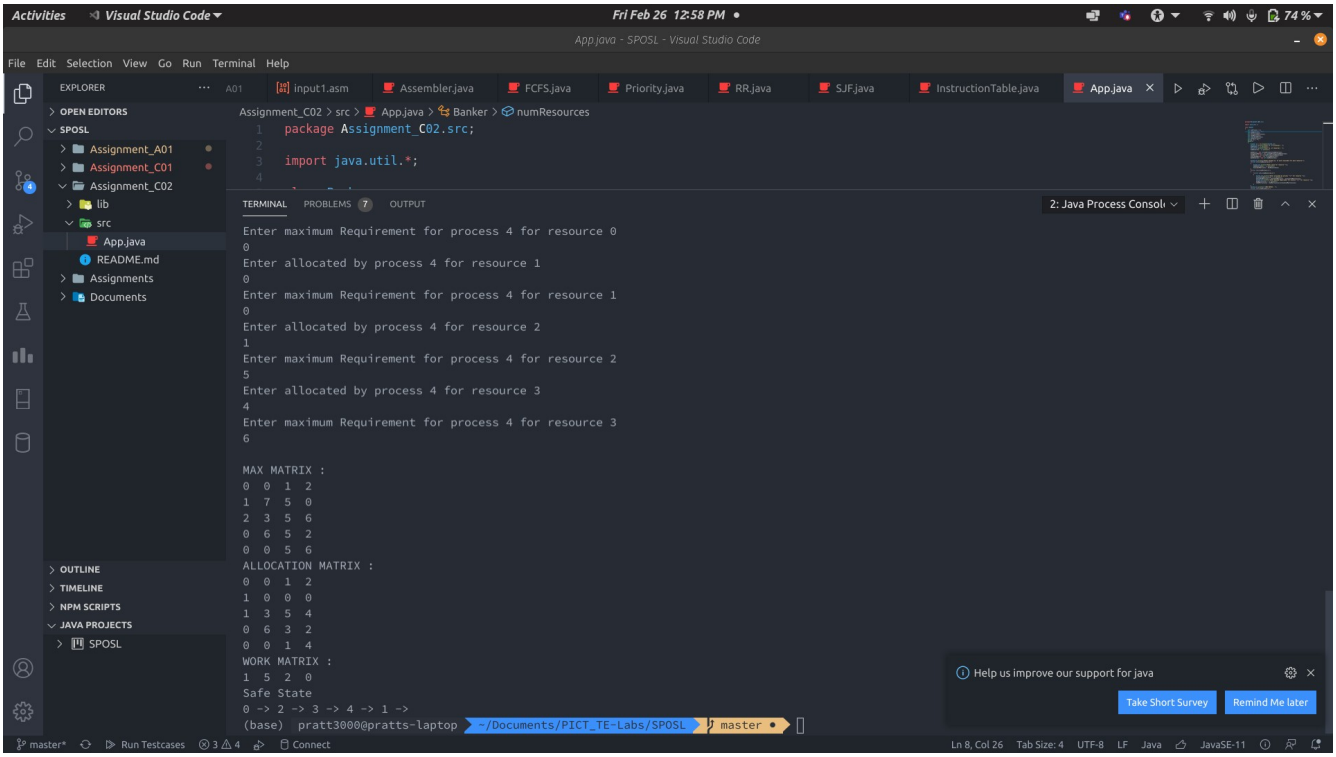
```
package Assignment_C02.src;

import java.util.*;

class Banker
{
int numProcess = 0;
int numResources = 0;
int maxMatrix[][];
int allocationMatrix[][];
int needMatrix[][];
int availableMatrix[];
int maxResources[];
String str = "";
Banker()
{
Scanner sc = new Scanner(System.in);
System.out.println("Enter no. of processes : ");
numProcess = sc.nextInt();
System.out.println("Enter no. of resources : ");
numResources = sc.nextInt();
maxMatrix = new int[numProcess][numResources];
allocationMatrix = new int[numProcess][numResources];
needMatrix = new int[numProcess][numResources];
availableMatrix = new int[numResources];
maxResources = new int [numResources];
System.out.println("Enter maximum no. of units available for each resource:");
for(int i=0;i<numResources;i++)
{
System.out.println("Enter value for resource "+i);
maxResources[i] = sc.nextInt();
availableMatrix[i] = maxResources[i];
}
for(int i=0;i<numProcess;i++)
{
for(int j=0;j<numResources;j++)
{
System.out.println("Enter allocated by process "+i+" for resource "+j);
allocationMatrix[i][j] = sc.nextInt();
availableMatrix[j] = availableMatrix[j] - allocationMatrix[i][j];
System.out.println("Enter maximum Requirement for process "+i+" for resource "+j);
maxMatrix[i][j] = sc.nextInt();
needMatrix[i][j] = maxMatrix[i][j]-allocationMatrix[i][j];
}
}
System.out.println("\nMAX MATRIX : ");
for(int i=0;i<numProcess;i++)
{
for(int j=0;j<numResources;j++)
{
System.out.print(maxMatrix[i][j]+" ");
}
System.out.print("\n");
}

System.out.println("ALLOCATION MATRIX : ");
for(int i=0;i<numProcess;i++)
```

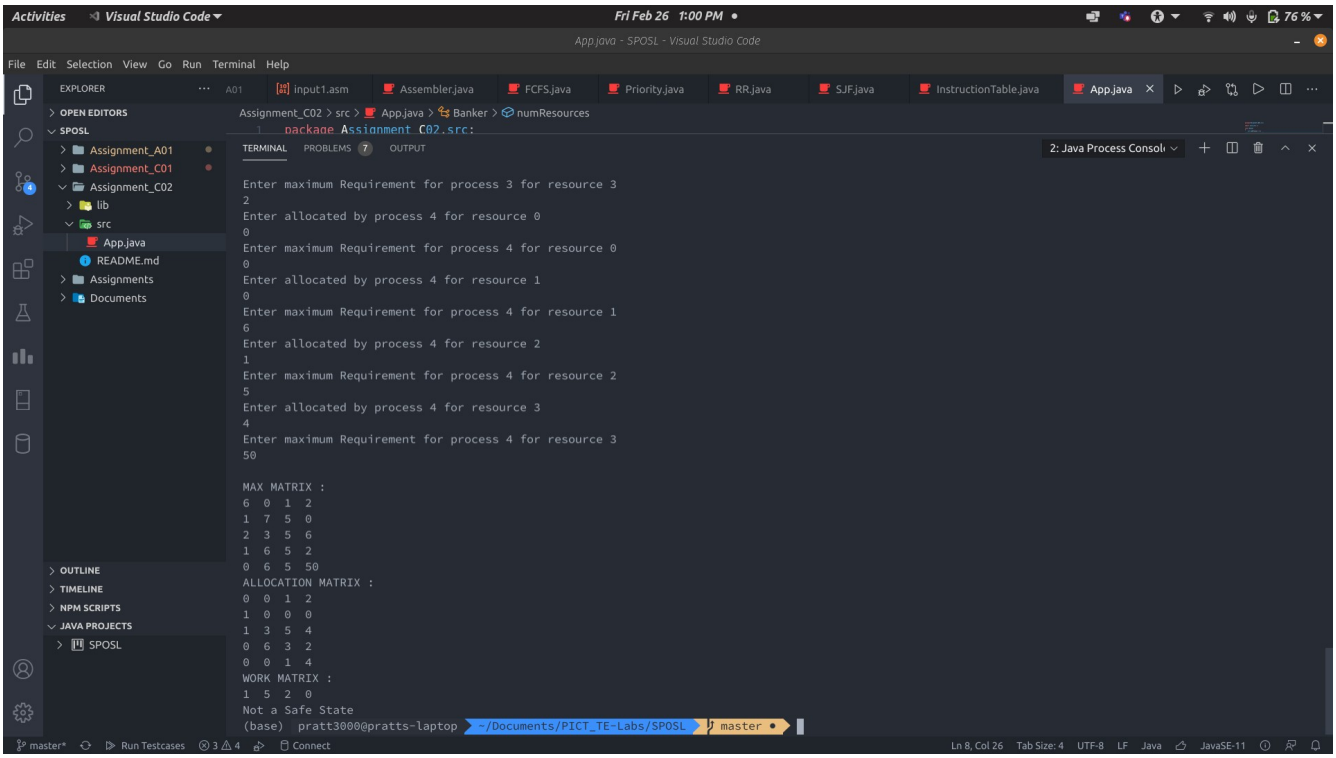
```
{
for(int j=0;j<numResources;j++)
{
System.out.print(allocationMatrix[i][j]+" ");
}
System.out.print("\n");
}
System.out.println("WORK MATRIX : ");
for(int j=0;j<numResources;j++)
{
int temp=0;
for(int i=0;i<numProcess;i++)
{
temp += allocationMatrix[i][j];
}
System.out.print((maxResources[j]-temp)+" ");
}
System.out.print("\n");
}
// check if process doesnt req. more than max avaliable resources
boolean checkmaxMatrix()
{
for(int i=0;i<numProcess;i++)
{
for(int j=0;j<numResources;j++)
{
if(maxMatrix[i][j]>maxResources[j])
{
return true;
}
}
}
return false;
}
// check same as checkmaxmatrix()
boolean checkNeed()
{
for(int i=0;i<numProcess;i++)
{
for(int j=0;j<numResources;j++)
{
if(needMatrix[i][j]<0)
{
return true;
}
}
}
return false;
}

boolean checkSafe()
{
if(checkmaxMatrix() || checkNeed())
{
return false;
}
int work[] = new int[numResources];
```

```
int need1[][] = new int [numProcess][numResources];
for(int i=0;i<numProcess;i++)
{
for(int j=0;j<numResources;j++)
{
need1[i][j] = needMatrix[i][j];
}
}
for(int i=0;i<numResources;i++)
{
work[i] = availableMatrix[i];
}
int flag = 0;
int flag1 = 0;
boolean exe[] = new boolean[numProcess];
for(int i=0;i<numProcess;i++)
{
exe[i] = false;
}
while(flag == 0)
{
flag = 1;
for(int i=0; i<numProcess;i++)
{
for(int j=0;j<numResources;j++)
{
if(need1[i][j]>work[j])
{
flag1 = 1;
break;
}
}
if(flag1 == 0)
{
for(int j=0;j<numResources;j++)
{
work[j] = work[j] + allocationMatrix[i][j];
need1[i][j] = maxResources[j] + 10000;
}
exe[i] = true;
str = str + i +" -> ";
flag = 0;
}
else
{
flag1 = 0;
}
}
}
for(int i=0;i<numProcess;i++)
{
if(exe[i]==false)
return false;
}
return true;
}
void menu()
```

```
{
Scanner sc = new Scanner(System.in);
if(checkSafe())
{
System.out.println("Safe State \n"+str);
}
else
{
System.out.println("Not a Safe State");
}
}
public static void main(String args[])
{
Banker b = new Banker();
b.menu();
}
}
```




Activities
Visual Studio Code
Fri Feb 26 12:58 PM
App.java - SPOSL - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
SPOSL
Assignment_A01
Assignment_C01
Assignment_C02
lib
src
App.java
README.md
Assignments
Documents
OUTLINE
TIMELINE
NPM SCRIPTS
JAVA PROJECTS
SPOSL
input1.asm
Assembler.java
FCFS.java
Priority.java
RR.java
SJF.java
InstructionTable.java
App.java
Assignment_C02 > src > App.java > Banker > numResources
package Assignment_C02.src;
import java.util.*;
TERMINAL
PROBLEMS 7
OUTPUT
2: Java Process Consol
Enter maximum Requirement for process 4 for resource 0
0
Enter allocated by process 4 for resource 1
0
Enter maximum Requirement for process 4 for resource 1
0
Enter allocated by process 4 for resource 2
1
Enter maximum Requirement for process 4 for resource 2
5
Enter allocated by process 4 for resource 3
4
Enter maximum Requirement for process 4 for resource 3
6
MAX MATRIX :
0 0 1 2
1 7 5 0
2 3 5 6
0 6 5 2
0 0 5 6
ALLOCATION MATRIX :
0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
0 0 1 4
WORK MATRIX :
1 5 2 0
Safe State
0 -> 2 -> 3 -> 4 -> 1 ->
(base) pratt3000@pratts-laptop ~/Documents/PICT_TE-Labs/SPOSL master
Help us improve our support for java
Take Short Survey
Remind Me later
master*
Run Testcases
Connect
Ln 8, Col 26
Tab Size: 4
UTF-8
LF
Java
JavaSE-11


Activities
Visual Studio Code
Fri Feb 26 1:00 PM
App.java - SPOSL - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
SPOSL
Assignment_A01
Assignment_C01
Assignment_C02
lib
src
App.java
README.md
Assignments
Documents
OUTLINE
TIMELINE
NPM SCRIPTS
JAVA PROJECTS
SPOSL
input1.asm
Assembler.java
FCFS.java
Priority.java
RR.java
SJF.java
InstructionTable.java
App.java
Assignment_C02 > src > App.java > Banker > numResources
TERMINAL
PROBLEMS 7
OUTPUT
2: Java Process Consol
Enter maximum Requirement for process 3 for resource 3
2
Enter allocated by process 4 for resource 0
0
Enter maximum Requirement for process 4 for resource 0
0
Enter allocated by process 4 for resource 1
0
Enter maximum Requirement for process 4 for resource 1
6
Enter allocated by process 4 for resource 2
1
Enter maximum Requirement for process 4 for resource 2
5
Enter allocated by process 4 for resource 3
4
Enter maximum Requirement for process 4 for resource 3
50
MAX MATRIX :
6 0 1 2
1 7 5 0
2 3 5 6
1 6 5 2
0 6 5 50
ALLOCATION MATRIX :
0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
0 0 1 4
WORK MATRIX :
1 5 2 0
Not a Safe State
(base) pratt3000@pratts-laptop ~/Documents/PICT_TE-Labs/SPOSL master
master*
Run Testcases
Connect
Ln 8, Col 26
Tab Size: 4
UTF-8
LF
Java
JavaSE-11