```java
/**
 * Implementing the concept of parallel processing on matrix multiplication
 *
 * @author ashra.
 *         Created Sep 22, 2019.
 */

import java.util.*;
import java.io.*;
public class MatrixMultiplication extends Thread {
    /*** declaration ***/
    private int i,j,k;
    static int m1[][];
    static int m2[][];
    static int result[][];
    /*** multi-threading ***/
    public MatrixMultiplication ( int i, int j, int k) {
        this.i = i;
        this.j = j;
        this.k = k;
    }
    public void run() {
        result[i][j] += m1[i][k]*m2[k][j];
    }
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the dimension of the matrix:");
        int dm = reader.nextInt(); //getting dimension of the matrix
        /*** initialization ***/
        m1 = new int[dm][dm];
        m2 = new int[dm][dm];
        result = new int [dm][dm];
        /*** input ***/
        System.out.println("Enter the first matrix:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                try {
                    m1[i][j]=reader.nextInt();
                }catch(Exception e) {}
            }
        }
        System.out.println("Enter the second matrix:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                try {
                    m2[i][j]=reader.nextInt();
                }catch(Exception e) {}
            }
        }
        /*** matrix multiplication ***/
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                for (int k=0; k<dm; k++) {
                    try {
                    new MatrixMultiplication(i, j, k).start(); //formula for multiplying two
                    matrices
                    }catch(Exception e) {}
                }
            }
        }
        /*** Output ***/
        System.out.println("The matrix after parallel processing multiplication:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++)
                System.out.print(result[i][j]+" ");
            System.out.println();
```

```
                }
        }
}
```

```java
/**
 * Implementing the concept of parallel processing on matrix multiplication
 *
 * @author ashra.
 *         Created Sep 22, 2019.
 */

import java.util.*;
import java.io.*;
public class MatrixMultiplication extends Thread {
    /*** declaration ***/
    private int i,j,k;
    static int m1[][];
    static int m2[][];
    static int result[][];
    /*** multi-threading ***/
    public MatrixMultiplication ( int i, int j, int k) {
        this.i = i;
        this.j = j;
        this.k = k;
    }
    public void run() {
        result[i][j] += m1[i][k]*m2[k][j];
    }
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the dimension of the matrix:");
        int dm = reader.nextInt(); //getting dimension of the matrix
        /*** initialization ***/
        m1 = new int[dm][dm];
        m2 = new int[dm][dm];
        result = new int [dm][dm];
        /*** input ***/
        System.out.println("Enter the first matrix:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                try {
                    m1[i][j]=reader.nextInt();
                }catch(Exception e) {}
            }
        }
        System.out.println("Enter the second matrix:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                try {
                    m2[i][j]=reader.nextInt();
                }catch(Exception e) {}
            }
        }
        /*** matrix multiplication ***/
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                for (int k=0; k<dm; k++) {
                    try {
                    new MatrixMultiplication(i, j, k).start(); //formula for multiplying two
                    matrices
                    }catch(Exception e) {}
                }
            }
        }
        /*** Output ***/
        System.out.println("The matrix after parallel processing multiplication:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++)
                System.out.print(result[i][j]+" ");
            System.out.println();
        }
    }
}
```

```
/**
 * Implementing the concept of parallel processing on matrix multiplication
 *
 * @author ashra.
 *         Created Sep 22, 2019.
 */

import java.util.*;
import java.io.*;
public class MatrixMultiplication extends Thread {
    /*** declaration ***/
    private int i,j,k;
    static int m1[][];
    static int m2[][];
    static int result[][];
    /*** multi-threading ***/
    public MatrixMultiplication ( int i, int j, int k) {
        this.i = i;
        this.j = j;
        this.k = k;
    }
    public void run() {
        result[i][j] += m1[i][k]*m2[k][j];
    }
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the dimension of the matrix:");
        int dm = reader.nextInt(); //getting dimension of the matrix
        /*** initialization ***/
        m1 = new int[dm][dm];
        m2 = new int[dm][dm];
        result = new int [dm][dm];
        /*** input ***/
        System.out.println("Enter the first matrix:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                try {
                    m1[i][j]=reader.nextInt();
                }catch(Exception e) {}
            }
        }
        System.out.println("Enter the second matrix:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                try {
                    m2[i][j]=reader.nextInt();
                }catch(Exception e) {}
            }
        }
        /*** matrix multiplication ***/
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                for (int k=0; k<dm; k++) {
                    try {
                    new MatrixMultiplication(i, j, k).start(); //formula for multiplying two
                    matrices
                    }catch(Exception e) {}
                }
            }
        }
        /*** Output ***/
        System.out.println("The matrix after parallel processing multiplication:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++)
                System.out.print(result[i][j]+" ");
            System.out.println();
        }
    }
}
```

```java
/**
 * Implementing the concept of parallel processing on matrix multiplication on 1000x1000 matrix
 *
 * @author ashra.
 *         Created Sep 22, 2019.
 */
import java.util.*;
import java.io.*;

public class MatrixMultiplication extends Thread {
    /*** declaration ***/
    private int i,j,k;
    static int m1[][];
    static int m2[][];
    static int result[][];
    /*** multi-threading ***/
    public MatrixMultiplication ( int i, int j, int k) {
        this.i = i;
        this.j = j;
        this.k = k;
    }
    public void run() {
        result[i][j] += m1[i][k]*m2[k][j];
    }
    public static void main(String args[]) {
        long start = System.currentTimeMillis(); //timestamp before execution
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the dimension of the matrix:");
        int dm = reader.nextInt(); //getting dimension of the matrix

        /*** initialization ***/
        m1 = new int[dm][dm];
        m2 = new int[dm][dm];
        result = new int [dm][dm];
        int max = 10;
        int min = 1;
        int range = max - min + 1;

        /*** input ***/
        //System.out.println("First matrix:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                try {
                    m1[i][j]=(int)(Math.random()*range)+min;
                }catch(Exception e) {}
            }
        }
        //System.out.println("Second matrix:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                try {
                    m2[i][j]=(int)(Math.random()*range)+min;
                }catch(Exception e) {}
            }
        }
        //the output matrix 1
        System.out.println("Matrix 1:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++)
                System.out.print(m1[i][j]+" ");
            System.out.println();
        }
        //the output matrix 2
        System.out.println("Matrix 2:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++)
```

```
                System.out.print(m2[i][j]+" ");
            System.out.println();
        }


        /*** matrix multiplication ***/
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++) {
                for (int k=0; k<dm; k++) {
                    try {
                    new MatrixMultiplication(i, j, k).start(); //formula for multiplying two
                    matrices
                    Thread.sleep(120);
                    }catch(Exception e) {}
                }
            }
        }
        /*** Output ***/
        System.out.println("The matrix after parallel processing multiplication:");
        for(int i=0; i<dm; i++) {
            for(int j=0; j<dm; j++)
                System.out.print(result[i][j]+" ");
            System.out.println();
        }
        long end = System.currentTimeMillis(); //timestamp after execution
        float sec = (end - start) / 1000F;
        System.out.println(sec + " seconds");
    }
}
```

```java
/**
 * Implementing the concept of booth multiplication
 *
 * @author ashra.
 *         Created Sep 22, 2019.
 */

import java.util.*;

public class BoothMultiplication {

    /*** multiplication function ***/
    public int multiply(int n1, int n2) {
        int m[] = binary(n1);
        int m1[] = binary(-n1);
        int r[] = binary(n2);
        int A[] = new int[9];
        int S[] = new int[9];
        int P[] = new int[9];
        for (int i=0; i<4; i++) {
            A[i] = m[i];
            S[i] = m1[i];
            P[i + 4] = r[i];
        }
        display(A, 'A');
        display(S, 'S');
        display(P, 'P');
        System.out.println();

        for (int i=0; i<4; i++) {
            if(P[7]==0 && P[8]==0);
                // do nothing
            else if(P[7]==1 && P[8]==0)
                add(P, S);
            else if(P[7]==0 && P[8]==1)
                add(P, A);
            else if(P[7]==1 && P[8]==1);
                // do nothing

            rightShift(P);
            display(P, 'P');
        }
        return getDecimal(P);
    }
    /*** decimal equivalent of P ***/
    public int getDecimal(int[] B) {
        int p = 0;
        int t = 1;
        for(int i=7; i>=0; i--, t*=2)
            p += (B[i] * t);
        if(p>64)
            p = -(256-p);
        return p;
    }
    /*** right shift array function ***/
    public void rightShift(int[] A) {
        for(int i=8; i>=1; i--)
            A[i] = A[i - 1];
    }
    /*** adding two binary arrays ***/
    public void add(int[] A, int[] B) {
        int carry = 0;
        for(int i=8; i>=0; i--) {
            int temp = A[i]+B[i]+carry;
            A[i] = temp%2;
            carry = temp/2;
        }
    }
    /*** binary of a number ***/
    public int[] binary(int n) {
        int bin[] = new int[4];
        int ctr = 3;
        int num = n;
        /*** for negative numbers 2's complement ***/
        if(n<0)
            num = 16+n;
        while(num!=0) {
            bin[ctr--] = num%2;
            num/=2;
        }
        return bin;
    }
    /*** Function to print array ***/
    public void display(int[] P, char ch) {
        System.out.print("\n"+ ch +" : ");
        for(int i=0; i<P.length; i++) {
            if(i==4)
                System.out.print(" ");
            if(i==8)
                System.out.print(" ");
            System.out.print(P[i]);
        }
    }
    public static void main (String[] args) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Booth Algorithm Test\n");
        /** Make an object of Booth class **/
        BoothMultiplication b = new BoothMultiplication();

        /** Accept two integers **/
        System.out.println("Enter two integer numbers\n");
        int n1 = reader.nextInt();
        int n2 = reader.nextInt();
        int result = b.multiply(n1, n2);
        System.out.println("\n\nResult : "+ n1 +" * "+ n2 +" = "+ result);
    }
}
```

```java
/**
 * Implementing the concept of collision vector
 *
 * @author ashra.
 *         Created Sep 26, 2019.
 */
import java.util.*;

public class CollisionVector {
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the number of columns in the reservation table:");
        int n = reader.nextInt(); // getting number of columns in the reservation table
        System.out.println("Enter the forbidden latencies:");
        long fl = reader.nextLong(); // getting the forbidden latencies
        getCollisionVector(n, fl);
    }
    public static void getCollisionVector(int n, Long fl) {
        int C[] = new int [n-1]; // defining the length of the collision vector
        int rC[] = new int [n-1]; //reverse array for calculation
        int flLength = String.valueOf(fl).length();
        /*** Checking for the final collision vector ***/
        if(flLength <= C.length) {
            while(fl != 0) {
                if((fl%10) != 0)
                    C[((int)(fl%10))-1] = 1;
                fl/=10;
            }
        }
        else
            System.out.println("Error! The number of forbidden latencies are more than the
            column length!");

        System.out.print("\nCollision Vector: ");
        for(int i=(C.length-1); i>=0; i--)
            rC[(C.length-1)-i] = C[i];

        System.out.println(Arrays.toString(rC));
    }
}
```

```java
/**
 * Implementing the concept of state diagram
 *
 * @author ashra.
 *          Created Sep 26, 2019.
 */
import java.util.*;

public class StateDiagram {
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the number of columns in the reservation table:");
        int n = reader.nextInt(); // getting number of columns in the reservation table
        System.out.println("Enter the forbidden latencies:");
        long fl = reader.nextLong(); // getting the forbidden latencies
        getCollisionVector(n, fl);
    }
    public static void getCollisionVector(int n, Long fl) {
        int C[] = new int [n-1]; // defining the length of the collision vector
        int rC[] = new int [n-1]; //reverse array for calculation
        int flLength = String.valueOf(fl).length();
        /*** Checking for the final collision vector ***/
        if(flLength <= C.length) {
            while(fl != 0) {
                if((fl%10) != 0)
                    C[((int)(fl%10))-1] = 1;
                fl/=10;
            }
        }
        else
            System.out.println("Error! The number of forbidden latencies are more than the
            column length!");

        System.out.print("\nCollision Vector: ");
        for(int i=(C.length-1); i>=0; i--)
            rC[(C.length-1)-i] = C[i];

        System.out.println(Arrays.toString(rC));
        /*** Checking for the state diagram ***/
        Arrays.setAll(C, i -> 0);
        /*** Performing the right shift on CV ***/
        System.out.print("\nCV 1-bit right shifted: ");
        for(int i=0; i<C.length; i++) {
            if(i==(C.length-1))
                C[i] = rC[0];
            else
                C[i] = rC[i+1];
        }
        System.out.println(Arrays.toString(C));
        int sd[] = new int [n-1];
        System.out.print("\nState Diagram: ");
        for(int i=0; i<sd.length; i++) {
            if(C[i]==1 || rC[i]==1)     //condition for getting the state diagram
                sd[i] = 1;
        }
        System.out.println(Arrays.toString(sd));
    }
}
```

```java
/**
 * Implementing the concept of speed-up factor
 *
 * @author ashra.
 *          Created Oct 2, 2019.
 */
import java.util.*;

public class SpeedUpFactor {
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("1: SpeedUp by Performance \n2: SpeedUp by Execution time \nChoose
        one of the options to proceed further!");
        int choice = reader.nextInt();
        double speedup = 0.0;
        switch(choice) {
            case 1:
                System.out.println("Enter the value of performance for entire task using
                enhancement:");
                int pe = reader.nextInt();
                System.out.println("Enter the value of performance for entire task without
                enhancement:");
                int pw = reader.nextInt();
                speedup = (double)pe/pw; //formula for speedup factor
                break;
            case 2:
                System.out.println("Enter the value of execution time in seconds without using
                enhancement:");
                int ew = reader.nextInt();
                System.out.println("Enter the value of execution time in seconds using
                enhancement:");
                int ee = reader.nextInt();
                speedup = (double)ew/ee;  //formula for speedup factor
                break;
            default:
                System.out.println("Invalid choice!");
        }
        System.out.println("Speedup Factor: "+speedup); //speedup enhanced

        /*** Overall Speedup ***/
        System.out.println("Enter the value for fraction enhanced:");
        double fraction = reader.nextDouble();
        double overallspeedup = 1/((1-fraction)+(fraction/speedup));//formula for overall speedup
        System.out.println("Overall Speedup: "+overallspeedup);
    }
}
```

```java
/**
 * Implementing the concept of direct mapping
 *
 * @author ashra.
 *         Created Oct 10, 2019.
 */
import java.util.*;
import java.math.*;

public class DirectMapping {
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the Total Memory Size in bits:");
        int tM = reader.nextInt(); // getting size of Main memory
        System.out.println("Enter the number of blocks:");
        int nB = reader.nextInt(); // getting the number of blocks in the cache memory
        getAssociativeMappedValue(tM, nB);
    }
    public static void getAssociativeMappedValue(int tM, int nB) {
        int block = (int)(Math.log(nB)/Math.log(2));
        int index = (int)(Math.log(block)/Math.log(2));
        int tag = tM - (block+index);
        System.out.println("Word: "+block+"\nIndex: "+index +"\nTag: "+tag);
    }
}
```

```java
/**
 * Implementing the concept of associative mapping
 *
 * @author ashra.
 *         Created Oct 10, 2019.
 */
import java.util.*;
import java.math.*;

public class AssociativeMapping {
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the Total Memory Size in bits");
        int tM = reader.nextInt(); // getting size of Main memory
        System.out.println("Enter the number of blocks:");
        int nB = reader.nextInt(); // getting the number of blocks in the cache memory
        getAssociativeMappedValue(tM, nB);
    }
    public static void getAssociativeMappedValue(int tM, int nB) {
        int block = (int)(Math.log(nB)/Math.log(2));
        int tag = tM - block;
        System.out.println("Word: "+block+"\nTag: "+tag);
    }
}
```

```
/**
 * Implementing the concept of set associative mapping
 *
 * @author ashra.
 *         Created Oct 10, 2019.
 */
import java.util.*;
import java.math.*;

public class SetAssociativeMapping {
    public static void main(String args[]) {
        Scanner reader = new Scanner(System.in);
        System.out.println("Enter the Total Memory Size in bits:");
        int tM = reader.nextInt(); // getting size of Main memory
        System.out.println("Enter the number of blocks:");
        int nB = reader.nextInt(); // getting the number of blocks in the cache memory
        System.out.println("Enter the number of sets in cache memory:");
        int nSC = reader.nextInt(); // getting the number of sets in cache
        getAssociativeMappedValue(tM, nB, nSC);
    }
    public static void getAssociativeMappedValue(int tM, int nB, int nSC) {
        int block = (int)(Math.log(nB)/Math.log(2));
        int set = (int)(Math.log(nSC)/Math.log(2));
        int tag = tM - (block+set);
        System.out.println("Word: "+block+"\nSet: "+set+"\nTag: "+tag);
    }
}
```