**КОД**

import java.util.Random;

public class Lab2 {

public static void main(String[] args) {

// -------------------------------

// 1. Вихідні дані та обчислення

// -------------------------------

int recordBookNumber = 123456; // Номер залікової книжки (приклад)

int C5 = recordBookNumber % 5;

int C7 = recordBookNumber % 7;

int C11 = recordBookNumber % 11;

// Для прикладу оберемо розмірність матриць

int rows = 4;

int cols = 5;

// Додаткова константа a для випадку C5 = 0 (C = a × B)

double a = 2.5;

// -------------------------------

// 2. Створення матриць A і B

// -------------------------------

// За умовою, тип елементів залежить від C7

// C7: 0->double, 1->byte, 2->short, 3->int, 4->long, 5->char, 6->float

// Для демонстрації нижче - реалізація для декількох типів.

// За потреби можна розширити.

Object A = null; // "Обгортка" під будь-який тип

Object B = null;

try {

switch (C7) {

case 0: // double

{

double[][] aMatrix = generateDoubleMatrix(rows, cols);

double[][] bMatrix = generateDoubleMatrix(rows, cols);

A = aMatrix;

B = bMatrix;

break;

}

case 3: // int

{

int[][] aMatrix = generateIntMatrix(rows, cols);

int[][] bMatrix = generateIntMatrix(rows, cols);

A = aMatrix;

B = bMatrix;

break;

}

// Можна додати інші випадки: byte, short, long, char, float тощо

default:

// Для спрощення - якщо тип нам не прописаний у switch,

// все одно згенеруємо double, щоб код продовжився

System.out.println("Увага! Тип не реалізований. Використовую double за замовчуванням.");

double[][] defaultMatrixA = generateDoubleMatrix(rows, cols);

double[][] defaultMatrixB = generateDoubleMatrix(rows, cols);

A = defaultMatrixA;

B = defaultMatrixB;

break;

}

} catch (Exception e) {

System.out.println("Помилка під час генерування матриць: " + e.getMessage());

return; // Завершимо роботу програми, якщо не можемо згенерувати матриці

}

// -------------------------------

// 3. Виконання дії з матрицями A і/або B (залежить від C5)

// -------------------------------

// C5: 0->C=a×B; 1->C=B^T; 2->C=A+B; 3->C=A⊕B; 4->C=A×B (матричне множення)

// Результат дії покладемо в "Object C"

Object C = null;

try {

switch (C5) {

case 0:

// C = a × B

C = multiplyMatrixByConstant(B, a);

System.out.println("Дія (C5=0): C = a × B (a=" + a + ")");

break;

case 1:

// C = B^T

C = transposeMatrix(B);

System.out.println("Дія (C5=1): C = B^T (транспонування B)");

break;

case 2:

// C = A + B

C = addMatrices(A, B);

System.out.println("Дія (C5=2): C = A + B (покомпонентне додавання)");

break;

case 3:

// C = A XOR B (логічна операція XOR, актуально здебільшого для цілочисельних типів)

C = xorMatrices(A, B);

System.out.println("Дія (C5=3): C = A ⊕ B (XOR)");

break;

case 4:

// C = A × B (матричне множення)

C = multiplyMatrices(A, B);

System.out.println("Дія (C5=4): C = A × B (матричне множення)");

break;

default:

System.out.println("Невідома операція з матрицями для C5=" + C5);

return;

}

} catch (Exception e) {

System.out.println("Помилка при виконанні операції з матрицями: " + e.getMessage());

return;

}

// Виведемо (для наочності) початкові матриці і матрицю C

System.out.println("\nМатриця A:");

printMatrix(A);

System.out.println("\nМатриця B:");

printMatrix(B);

System.out.println("\nРезультуюча матриця C:");

printMatrix(C);

// -------------------------------------------

// 4. Виконати дію з матрицею C (залежить від C11)

// -------------------------------------------

// За таблицею для C11 можуть бути різні підсумкові дії:

// 0->сума мінімальних ел. кожного стовпця, 1->сума мінімальних ел. кожного рядка, 2->сума максимальних ел. стовпця, 3->сума макс. ел. рядка, ...

// 8->середнє значення елементів у кожному рядку, 9->середнє значення в кожному стовпці, 10->середнє значення всіх ел. в матриці

// і т.д.

try {

double resultC11 = processMatrixC(C, C11);

System.out.println("\nРезультат дії над матрицею C (C11=" + C11 + "): " + resultC11);

} catch (Exception e) {

System.out.println("Помилка при опрацюванні матриці C (C11=" + C11 + "): " + e.getMessage());

}

System.out.println("\nПрограма завершена успішно.");

}

// ------------------------------------------------------

// Методи для генерації матриць заданого типу та розміру

// ------------------------------------------------------

private static double[][] generateDoubleMatrix(int rows, int cols) {

double[][] matrix = new double[rows][cols];

Random rand = new Random();

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

// Генеруємо випадкові числа у діапазоні [-10; 10]

matrix[i][j] = -10 + rand.nextDouble() \* 20;

}

}

return matrix;

}

private static int[][] generateIntMatrix(int rows, int cols) {

int[][] matrix = new int[rows][cols];

Random rand = new Random();

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

// Генеруємо випадкові цілі числа у діапазоні [-10; 10]

matrix[i][j] = rand.nextInt(21) - 10;

}

}

return matrix;

}

// -------------------------------------

// Методи для друку матриці у консоль

// -------------------------------------

private static void printMatrix(Object mat) {

if (mat instanceof double[][]) {

double[][] m = (double[][]) mat;

for (double[] row : m) {

for (double val : row) {

System.out.printf("%8.2f ", val);

}

System.out.println();

}

} else if (mat instanceof int[][]) {

int[][] m = (int[][]) mat;

for (int[] row : m) {

for (int val : row) {

System.out.printf("%5d ", val);

}

System.out.println();

}

} else if (mat == null) {

System.out.println("Матриця відсутня (null).");

} else {

// Для інших типів у разі розширення

System.out.println("Формат виведення для даного типу не реалізований.");

}

}

// -------------------------------------

// Методи виконання різних операцій

// -------------------------------------

// C = a × B (помножити матрицю на число)

private static Object multiplyMatrixByConstant(Object B, double a) throws Exception {

if (B instanceof double[][]) {

double[][] bMatrix = (double[][]) B;

double[][] result = new double[bMatrix.length][bMatrix[0].length];

for (int i = 0; i < bMatrix.length; i++) {

for (int j = 0; j < bMatrix[0].length; j++) {

result[i][j] = a \* bMatrix[i][j];

}

}

return result;

} else if (B instanceof int[][]) {

int[][] bMatrix = (int[][]) B;

double[][] result = new double[bMatrix.length][bMatrix[0].length];

for (int i = 0; i < bMatrix.length; i++) {

for (int j = 0; j < bMatrix[0].length; j++) {

result[i][j] = a \* bMatrix[i][j];

}

}

return result;

}

// За потреби реалізувати інші типи

throw new Exception("multiplyMatrixByConstant: Непідтримуваний тип матриці");

}

// C = B^T (транспонування)

private static Object transposeMatrix(Object B) throws Exception {

if (B instanceof double[][]) {

double[][] bMatrix = (double[][]) B;

double[][] result = new double[bMatrix[0].length][bMatrix.length];

for (int i = 0; i < bMatrix.length; i++) {

for (int j = 0; j < bMatrix[0].length; j++) {

result[j][i] = bMatrix[i][j];

}

}

return result;

} else if (B instanceof int[][]) {

int[][] bMatrix = (int[][]) B;

int[][] result = new int[bMatrix[0].length][bMatrix.length];

for (int i = 0; i < bMatrix.length; i++) {

for (int j = 0; j < bMatrix[0].length; j++) {

result[j][i] = bMatrix[i][j];

}

}

return result;

}

// За потреби реалізувати інші типи

throw new Exception("transposeMatrix: Непідтримуваний тип матриці");

}

// C = A + B (покомпонентне додавання)

private static Object addMatrices(Object A, Object B) throws Exception {

if (A instanceof double[][] && B instanceof double[][]) {

double[][] aMatrix = (double[][]) A;

double[][] bMatrix = (double[][]) B;

if (aMatrix.length != bMatrix.length || aMatrix[0].length != bMatrix[0].length) {

throw new Exception("addMatrices: Розміри матриць не збігаються!");

}

double[][] result = new double[aMatrix.length][aMatrix[0].length];

for (int i = 0; i < aMatrix.length; i++) {

for (int j = 0; j < aMatrix[0].length; j++) {

result[i][j] = aMatrix[i][j] + bMatrix[i][j];

}

}

return result;

} else if (A instanceof int[][] && B instanceof int[][]) {

int[][] aMatrix = (int[][]) A;

int[][] bMatrix = (int[][]) B;

if (aMatrix.length != bMatrix.length || aMatrix[0].length != bMatrix[0].length) {

throw new Exception("addMatrices: Розміри матриць не збігаються!");

}

int[][] result = new int[aMatrix.length][aMatrix[0].length];

for (int i = 0; i < aMatrix.length; i++) {

for (int j = 0; j < aMatrix[0].length; j++) {

result[i][j] = aMatrix[i][j] + bMatrix[i][j];

}

}

return result;

}

// За потреби реалізувати інші типи

throw new Exception("addMatrices: Непідтримуваний тип матриці");

}

// C = A XOR B (логічна операція XOR для відповідних типів)

private static Object xorMatrices(Object A, Object B) throws Exception {

// Має сенс лише для цілочисельних типів: byte, short, int, long, char

if (A instanceof int[][] && B instanceof int[][]) {

int[][] aMatrix = (int[][]) A;

int[][] bMatrix = (int[][]) B;

if (aMatrix.length != bMatrix.length || aMatrix[0].length != bMatrix[0].length) {

throw new Exception("xorMatrices: Розміри матриць не збігаються!");

}

int[][] result = new int[aMatrix.length][aMatrix[0].length];

for (int i = 0; i < aMatrix.length; i++) {

for (int j = 0; j < aMatrix[0].length; j++) {

result[i][j] = aMatrix[i][j] ^ bMatrix[i][j];

}

}

return result;

}

throw new Exception("xorMatrices: Непідтримуваний тип матриці або непогоджені розміри");

}

// C = A × B (матричне множення)

private static Object multiplyMatrices(Object A, Object B) throws Exception {

if (A instanceof double[][] && B instanceof double[][]) {

double[][] aMatrix = (double[][]) A;

double[][] bMatrix = (double[][]) B;

if (aMatrix[0].length != bMatrix.length) {

throw new Exception("multiplyMatrices: Неможливо перемножити (число стовпців A != число рядків B)");

}

double[][] result = new double[aMatrix.length][bMatrix[0].length];

for (int i = 0; i < aMatrix.length; i++) {

for (int j = 0; j < bMatrix[0].length; j++) {

double sum = 0;

for (int k = 0; k < bMatrix.length; k++) {

sum += aMatrix[i][k] \* bMatrix[k][j];

}

result[i][j] = sum;

}

}

return result;

} else if (A instanceof int[][] && B instanceof int[][]) {

int[][] aMatrix = (int[][]) A;

int[][] bMatrix = (int[][]) B;

if (aMatrix[0].length != bMatrix.length) {

throw new Exception("multiplyMatrices: Неможливо перемножити (число стовпців A != число рядків B)");

}

int[][] result = new int[aMatrix.length][bMatrix[0].length];

for (int i = 0; i < aMatrix.length; i++) {

for (int j = 0; j < bMatrix[0].length; j++) {

int sum = 0;

for (int k = 0; k < bMatrix.length; k++) {

sum += aMatrix[i][k] \* bMatrix[k][j];

}

result[i][j] = sum;

}

}

return result;

}

// За потреби - інші типи

throw new Exception("multiplyMatrices: Непідтримуваний тип матриці або непогоджені розміри");

}

// -------------------------------------

// 5. Дії з матрицею C, залежно від C11

// -------------------------------------

private static double processMatrixC(Object C, int c11) throws Exception {

// Для спрощення повертаємо одне число (як суму, середнє тощо)

// Якщо потрібно, можна повертати іншу структуру даних

if (C instanceof double[][]) {

double[][] cMatrix = (double[][]) C;

return processDoubleMatrix(cMatrix, c11);

} else if (C instanceof int[][]) {

int[][] cMatrix = (int[][]) C;

// Перетворимо в double[][] для уніфікації обчислень:

double[][] tmp = new double[cMatrix.length][cMatrix[0].length];

for (int i = 0; i < cMatrix.length; i++) {

for (int j = 0; j < cMatrix[0].length; j++) {

tmp[i][j] = cMatrix[i][j];

}

}

return processDoubleMatrix(tmp, c11);

}

throw new Exception("processMatrixC: Непідтримуваний тип матриці");

}

// Метод для обчислення з double-матрицею (усі операції переведені до double)

private static double processDoubleMatrix(double[][] mat, int c11) throws Exception {

int rows = mat.length;

int cols = mat[0].length;

switch (c11) {

case 0:

// Сума найменших елементів кожного стовпця

return sumOfColumnMinElements(mat);

case 1:

// Сума найменших елементів кожного рядка

return sumOfRowMinElements(mat);

case 2:

// Сума найбільших елементів кожного стовпця

return sumOfColumnMaxElements(mat);

case 3:

// Сума найбільших елементів кожного рядка

return sumOfRowMaxElements(mat);

case 8:

// Знайти середнє значення елементів кожного рядка (сума середніх)

// Якщо за умовою потрібно саме “кожного рядка окремо”, можна виводити масив.

// Тут для прикладу повернемо суму середніх кожного рядка

double sumAvgRow = 0;

for (int i = 0; i < rows; i++) {

double rowSum = 0;

for (int j = 0; j < cols; j++) {

rowSum += mat[i][j];

}

double rowAvg = rowSum / cols;

sumAvgRow += rowAvg;

}

return sumAvgRow;

case 9:

// Знайти середнє значення елементів кожного стовпця (сума середніх)

double sumAvgCol = 0;

for (int j = 0; j < cols; j++) {

double colSum = 0;

for (int i = 0; i < rows; i++) {

colSum += mat[i][j];

}

double colAvg = colSum / rows;

sumAvgCol += colAvg;

}

return sumAvgCol;

case 10:

// Середнє значення всіх елементів матриці

double totalSum = 0;

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

totalSum += mat[i][j];

}

}

return totalSum / (rows \* cols);

// За потреби реалізувати й інші варіанти (4,5,6,7 тощо) –

// сума найбільших/найменших у парних/непарних індексах і т.д.

default:

throw new Exception("processDoubleMatrix: Операція для C11=" + c11 + " не реалізована у прикладі");

}

}

// --------------------------------

// Допоміжні методи для пункту C11

// --------------------------------

private static double sumOfColumnMinElements(double[][] mat) {

int rows = mat.length;

int cols = mat[0].length;

double sum = 0;

for (int j = 0; j < cols; j++) {

double minVal = mat[0][j];

for (int i = 1; i < rows; i++) {

if (mat[i][j] < minVal) {

minVal = mat[i][j];

}

}

sum += minVal;

}

return sum;

}

private static double sumOfRowMinElements(double[][] mat) {

int rows = mat.length;

int cols = mat[0].length;

double sum = 0;

for (int i = 0; i < rows; i++) {

double minVal = mat[i][0];

for (int j = 1; j < cols; j++) {

if (mat[i][j] < minVal) {

minVal = mat[i][j];

}

}

sum += minVal;

}

return sum;

}

private static double sumOfColumnMaxElements(double[][] mat) {

int rows = mat.length;

int cols = mat[0].length;

double sum = 0;

for (int j = 0; j < cols; j++) {

double maxVal = mat[0][j];

for (int i = 1; i < rows; i++) {

if (mat[i][j] > maxVal) {

maxVal = mat[i][j];

}

}

sum += maxVal;

}

return sum;

}

private static double sumOfRowMaxElements(double[][] mat) {

int rows = mat.length;

int cols = mat[0].length;

double sum = 0;

for (int i = 0; i < rows; i++) {

double maxVal = mat[i][0];

for (int j = 1; j < cols; j++) {

if (mat[i][j] > maxVal) {

maxVal = mat[i][j];

}

}

sum += maxVal;

}

return sum;

}

}

**Пояснення**

1. **Зчитування та обчислення C5, C7, C11**У коді визначається номер залікової книжки (наприклад, 123456). Далі обчислюються залишки від ділення цього номера на 5, 7 та 11, що відповідає значенням змінних C5, C7 та C11.
2. **Створення матриць A та B** Типи елементів матриць визначаються за значенням C7 (наприклад, якщо C7 = 0 – тоді це double[][]). У коді для прикладу реалізовані два варіанти: double та int.  
   * Якщо C7 = 0 – програма викликає методи generateDoubleMatrix, щоб згенерувати дві матриці (A і B) типу double.
   * Якщо C7 = 3 – створюються дві матриці типу int.
   * Якщо C7 має інше значення (наприклад 1, 2 тощо), для спрощення (у прикладі) все одно створюється double-матриця і виводиться попередження, що інший тип не реалізований.
3. **Виконання дії з матрицями (залежить від C5)** Значення C5 визначає операцію, яку слід виконати з матрицями A, B (або лише B):  
   * C5 = 0: C = a × B (помножити кожен елемент B на певну константу a)
   * C5 = 1: C = B^T (транспонування B)
   * C5 = 2: C = A + B (покомпонентне додавання)
   * C5 = 3: C = A XOR B (логічна операція XOR, зазвичай застосовується для цілих типів)
   * C5 = 4: C = A × B (матричне множення)
4. У програмі кожен метод (multiplyMatrixByConstant, transposeMatrix, addMatrices, xorMatrices, multiplyMatrices) виконує потрібну операцію та повертає результуючу матрицю C.
5. **Виведення результатів** Після отримання матриці C програма виводить на екран (у методі printMatrix):  
   * Спочатку матрицю A
   * Потім матрицю B
   * І наостанок матрицю C
6. У методі printMatrix тип матриці перевіряється (якщо це double[][], то виводимо з плаваючою точкою, якщо int[][], то виводимо цілі числа). Якщо тип не обробляється – виводиться відповідне повідомлення.
7. **Дія з матрицею C (залежить від C11)** Друга частина завдання – виконати обчислення, визначене C11. У прикладі реалізовані такі варіанти:  
   * C11 = 0: сума найменших елементів кожного стовпця
   * C11 = 1: сума найменших елементів кожного рядка
   * C11 = 2: сума найбільших елементів кожного стовпця
   * C11 = 3: сума найбільших елементів кожного рядка
   * C11 = 8: сума середніх значень для кожного рядка (для прикладу)
   * C11 = 9: сума середніх значень для кожного стовпця
   * C11 = 10: середнє значення всіх елементів матриці
8. Якщо в коді попадається інше значення (наприклад, 4, 5, 6, 7), програма поки що виведе повідомлення, що операція не реалізована.  
     
    Результат цього другого обчислення (наприклад, сума чи середнє) повертається у змінну resultC11 і виводиться на екран.
9. **Обробка винятків** У програмі передбачені блоки try-catch, які перехоплюють ситуації, коли:  
   * Розміри матриць не збігаються (наприклад, при додаванні або множенні).
   * Використовується тип, що не обробляється (наприклад, XOR для double).
   * Інші виняткові ситуації (наприклад, некоректно ініціалізована матриця).
10. Якщо виникає критична помилка, програма виводить повідомлення і припиняє подальше виконання.
11. **Завершення роботи** Після успішного виконання всіх операцій програма виводить повідомлення про успішне завершення.

Код демонструє, як залежно від C5 і C7 виконується створення й обробка матриць (A, B, C), а залежно від C11 – виконується додаткова підсумкова операція над матрицею C. При потребі можна додавати обробку усіх інших типів (byte, short, long, float, char) або впроваджувати додаткові операції для C5 та C11, дотримуючись такої ж структури і методології.