MINISTERUL EDUCAŢIEI AL REPUBLICII MOLDOVA

Centrul de Excelenţă în Informatică şi Tehnologii Informaţionale



**RAPORT**

**PENTRU PRACTICA TEHNOLOGICĂ**

**SPECIALITATEA** „Programarea și analiza produselor program”

A elaborat elevul: Darii Dumitru

Grupa: P-1932

Conducătorul practicii: Jumbei Olga

Chişinău

2022

# 2. Cuprins

[2. Cuprins 2](#_Toc103635703)

[Introducere 3](#_Toc103635704)

[Modul de elaborare a aplicației 4](#_Toc103635705)

[3. Introducere 5](#_Toc103635706)

[3.2 Competențele practicii 7](#_Toc103635707)

[3.3 Competențele profesionale 7](#_Toc103635708)

[3.4 Competențele specifice 8](#_Toc103635709)

[4. Enunțul sarcinii individuale 9](#_Toc103635710)

[5. Descrierea modului de elaborare a aplicației 11](#_Toc103635711)

[6. Listing-ul programului 12](#_Toc103635712)

[6. Rezultatele testării subprogramelor 28](#_Toc103635713)

[7. Concluzie 39](#_Toc103635714)

[8. Bibliografie 40](#_Toc103635715)

# Introducere

Sarcina stagiul dat de practică constă în realizarea unei aplicații vizuale în baza unei anumite teme.

Am ales să realizez o aplicație similară site-ului web Librarius.md, care reprezintă o librărie din Moldova. Aplicația dispune de două interfețe: una pentru clienți și alta pentru manager.

Abilitățile clienților:

* Dispun de înregistrare
* Dispun de autentificare
* Pot vizualiza catalogul de cărți
* Pot obține detalii pentru fiecare carte
* Pot căuta cărți în baza la mai multe filtre
* Pot adăuga cărți la lista de cărți favorite
* Pot exporta date din lista de cărți favorite în Excel (fișier .xlsx)

Abilitățile managerilor:

* Dispun de toate abilitățile clienților
* Au acces la modificarea bazei de date a librăriei
* Pot adăuga cărți noi în baza de date
* Pot șterge cărți din baza de date

# Modul de elaborare a aplicației

Pentru realizarea acestei sarcini au fost utilizate următoarele instrumente:

* IDE Microsoft Visual Studio
* Platforma .NET
* Limbajul de programare C#
* Framework-ul UI Windows Forms
* Microsoft SQLServer
* Microsoft SQL Server Management Studio

Instrumente opționale utilizate:

* Librăria .NET IronXL (predestinată pentru exportarea datelor în format .xlsx, adică Excel)
* Sistemul de control de versiuni GIT (predestinat facilitării lucrului asupra aplicației și administrarea versiunilor programului)
* GitHub.com - Hosting-ul de proiecte software, care utilizează GIT (predestinat pentru stocarea la distanță a fișierelor proiectului și pentru administrarea versiunilor în mod interactiv)

Părțile de bază ale interfeței grafice:

* Meniul este realizat printr-un sidebar, situat în stânga, care conține butoane cu linkuri către paginile aplicației.
* Zona de conținut este alăturată meniului, aceasta servind drept placeholder pentru paginile deschise în urma accesării link-urilor din meniu.
* Înregistrarea, autentificarea și detaliile despre cărțil se deschid la necesitate în ferestre separate.

# 3. Introducere

Fiecare din aceste practici reprezintă câte o etapă a procesului de pregătire continuă a tînărului specialist.

Studentul începător se familiarizează cu programarea, de regulă, în baza elaborării şi transcrierii într-un limbaj de programare a algoritmilor de rezolvare a unor probleme concrete

simple. Însă, rezolvarea problemelor reale presupune elaborarea unor complexe programate mari, care includ o gamă întreagă de probleme: proiectarea sistemului, elaborarea părţilor componente ale algoritmului, reunirea diverselor fragmente ale programului într-un produs final, documentarea etc. În cadrul lecţiilor este posibilă examinarea doar a principiilor generale ale metodicii de elaborare a programelor şi anumitor aspecte ale rezolvării problemelor. La rândul său, practica de instruire imită întregul proces de elaborare a unui complex programat mare, permite studentului să evolueze în rolul de elaborator şi organizator al proiectului.

E cunoscut faptul, că aplicarea tehnicilor de organizare a subprogramelor la elaborarea programelor mari, oferă anumite avantaje. De exemplu, utilizarea subprogramelor permite crearea şi depanarea componentele unui program, independent de celelalte componente ale acestuia.

Sarcina individuală pentru student se formulează în aşa mod, încât să cuprindă cât mai deplin toate compartimentele cursului studiat. Formularea problemei revine pe seama conducătorului practicii, întrucât aceasta trebuie să fie selectată astfel încât să fie suficient de complicată, prezentând, în acelaşi rând, interes pentru elaborator. Totodată, gradul complexităţii problemelor de rezolvat variază larg, ţinând cont de capacităţile şi pregătirea generală a studentului respectiv.

În procesul lucrului de sine stătător asupra sarcinii individuale, studenţii elaborează şi descriu algoritmi într-un limbaj de programare, testează şi depanează programele respective.

Practica de Specialitate I constituie o primă lucrare de programare vizuala a studenţilor şi va contribui la formarea şi dezvoltarea calităţilor strict necesare nu numai viitorilor programatori, dar şi fiecărui om cult care, la sigur, va trăi şi va activa într-un mediu bazat pe cele mai moderne tehnologii informaţionale. Are următoarele obiective generale:

* consolidarea cunoştinţelor teoretice, obţinute de studenţi pe parcursul studierii limbajului de programare Java şi mediului de programare IntelliJ Idea.
* însuşirea tehnologiei de elaborare a programelor mari;
* dezvoltarea abilităţilor muncii de sine stătătoare şi lucrul în echipă;
* formarea deprinderilor de cercetător.

Sarcina propusă mie spre realizare este cea cu numărul 13. Subiectul sarcinii este despre un teren minat împărțit în zone. Sarcina dată are 8 subpuncte cu propria descriere. Nu consider că voi întâlni dificultăți în procesul de realizare a sarcinii date.

# 3.2 Competențele practicii

1. Competenţe de învăţare de a învăţa să înveţi.
2. Competenţe de comunicare în limba maternă / limba de stat.
3. Competenţe de comunicare într-o limbă străină.
4. Competenţe acțional-strategice.
5. Competenţe de autocunoaştere şi autorealizare.
6. Competenţe interpersonale, civice, morale.
7. Competenţe de bază în matematică, ştiinţe şi tehnologie.
8. Competenţe digitale, în domeniul tehnologiei informaţiei şi comunicaţiilor (TIC).
9. Competenţe culturale, interculturale (de a recepta şi a crea valori).
10. Competenţe antreprenoriale, spirit de iniţiativă şi antreprenoriat.
11. Competenţe de comunicare în limba maternă / limba de stat.
12. Competenţe de bază în matematică, ştiinţe şi tehnologie.
13. Competenţe digitale, în domeniul tehnologiei informaţiei şi comunicaţiilor (TIC).

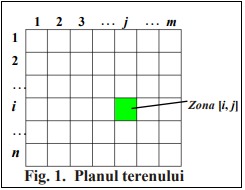
# 3.3 Competențele profesionale

1. Respectarea dreptului de autor asupra resurselor digitale, a normelor de etică şi securitate informaţională.
2. Identificarea principiilor de funcţionare, clasificare şi proiectare a conexiunii între echipamentele reţelelor de calculatoare. 3. Proiectarea algoritmilor şi codificarea lui într-un limbaj de programare.
3. Utilizarea metodelor şi tehnicilor eficiente de programare.
4. Elaborarea aplicaţiilor, utilizând elementele programării orientate pe obiecte.
5. Conceperea produselor multimedia.
6. Utilizarea tehnologiilor şi instrumentelor de proiectare, elaborare şi programare a aplicaţiilor WEB.
7. Modelarea, proiectarea şi implementarea sistemelor informatice.
8. Aplicarea fundamentelor teoretice ale ştiinţelor juridice, sociale şi economice în activitatea profesională.

# 3.4 Competențele specifice

1. Analiza structurală a problemei în studiu.
2. Proiectarea descendentă a algoritmilor.
3. Aplicarea principiilor programării structurate, programării procedurale şi programării orientate pe obiecte în scrierea algoritmilor şi elaborarea aplicaţiei.
4. Utilizarea tipurilor structurate de date (tablou, şir de caractere, mulţime, articol, fişier etc.) adecvate pentru organizarea şi prelucrarea datelor.
5. Utilizarea principiilor Programării Orientate spre Obiecte la elaborarea programelor.
6. Utilizarea tehnicilor eficiente la elaborarea algoritmilor şi programelor respective.
7. Aplicarea tehnicilor de testare şi depanare a programelor elaborate.
8. Elaborarea unei aplicaţii cu interfaţă grafică de interacţiune cu utilizatorul;
9. Gestionarea informaţiei utilizând resursele Internet.

# 4. Enunțul sarcinii individuale

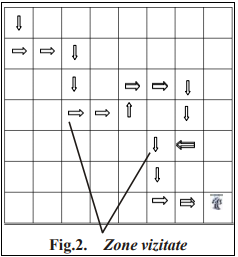
Planul unui teren minat, având forma unei table dreptunghiulare de dimensiunea **n×m (n, m ≤ 50)** este împărţit în zone pătrate de lungimea 1 (vezi figura 1). În fiecare zonă reală a terenului poate fi plasată o mină.

Informaţii mai concrete despre terenul în studiu sunt înregistrate în fişierul text **Teren.in**, care conţine pe prima linie numerele naturale **n** şi **m**, separate prin spaţiu. Fiecare din următoarele **n** linii ale acestui fişier conţine câte **m** cifre binare, separate prin câte un spaţiu – elementele unei matrice **T**, în care **T[i, j] =1**, dacă zona **[i, j]** conţine o mină, şi **T[i, j]=0** –dacă zona este liberă.

Să se elaboreze un program care, folosind meniuri şi subprograme, să realizeze, la solicitarea utilizatorului, următoarele prescripţii:

1. Înscrie în planul terenului un nou rând (marginal) / o nouă coloană (marginală); alternativa aleasă şi poziţia rândului (nord/sud) / coloanei (vest/est) de înscris se vor preciza de la tastatură;
2. „Demină” zonele unui rând / unei coloane; alternativa aleasă şi numărul de ordine al rândului / coloanei de „deminat” se va preciza de la tastatură;
3. Determină numărul liniei/coloanei cu un număr minimal de zone minate;
4. Determină media numerelor de zone minate de pe coloanele pare ale terenului în studiu;
5. Afişează pe ecran lista numerelor de ordine ale liniilor terenului în ordinea ascendentă a numărului total de mine plasate pe liniile respective; datele se vor sorta prin metoda selecției;
6. Creează fişierul text **Mine.txt**, în care se vor copia doar acele linii ale fişierului de intrare **Teren.in**, care nu conţin mine;
7. Determină numărul de obiecte din matricea binară **T**.

**Notă**. Un obiect este format din elemente cu valoarea 1 care se învecinează pe linii, pe coloane sau pe diagonale.

1. **Rezolvă problema.** Un soldat, având la dispoziţie un detector de mine, porneşte dintru –un colţ al terenului şi trebuie să ajungă în colţul opus. Pe terenul considerat, soldatul se poate deplasa doar ortogonal şi, evident, fără a nimeri în careva din zonele minate.

Să se afle traseul cel mai scurt, ce trebuie parcurs de soldat pentru a ajunge din zona [1, 1], considerată, evident, neminată, în zona [n, m].

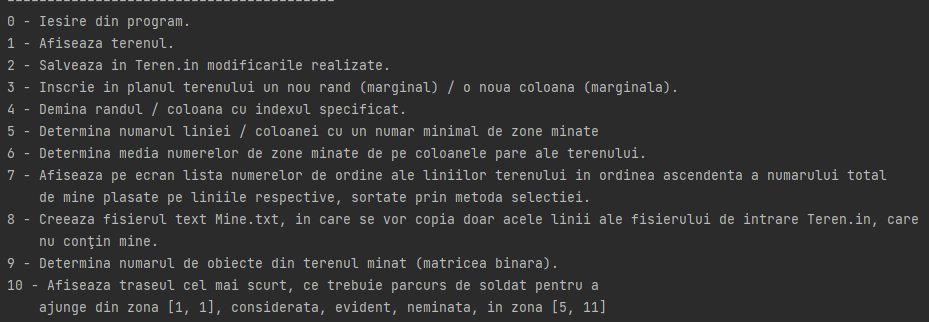
Date de intrare. Informaţiile despre dimensiunile terenului şi zonele minate din teritoriu se conţin în fişierul text Teren.in, descris anterior.

Date de ieşire. Se va afişa pe ecran drumul găsit, descris prin coordonatele zonelor respective.

De exemplu, pentru ilustrarea din figura 2, drumul parcurs se va afişa astfel: [1, 1]–[2, 1]–[2, 2]–[3, 2]–[3, 3]–[4, 3]–[4, 4]–[4, 5]–[3, 5]–[3, 6] –[3, 7]– [4, 7]– [5, 7] –[5, 6] – [6, 6] –[7, 6] –[7, 7] –[7, 8]

# 5. Descrierea modului de elaborare a aplicației

Am adăugat un meniu pentru a putea utiliza și executa logica implementată prin cod. În meniu sunt prezente opțiuni pentru fiecare dintre cele 8 subpuncte ale sarcinii, precum și câteva opțiuni bonus pentru afișare, salvare ș.a.

Meniul:

# 6. Listing-ul programului

**Main.java:**

import java.io.IOException;  
import java.util.Scanner;  
  
public class Main {  
  
 private static TerenMinat *teren*;  
  
 public static void main(String[] args) {  
 *teren* = new TerenMinat();  
 *menu*();  
 }  
  
 public static void printMenu() {  
 System.*out*.println("=========================================");  
 System.*out*.println("0 - Iesire din program.");  
 System.*out*.println("1 - Afiseaza terenul.");  
 System.*out*.println("2 - Salveaza in Teren.in modificarile realizate.");  
 System.*out*.println("3 - Inscrie in planul terenului un nou rand (marginal) / o noua coloana (marginala).");  
 System.*out*.println("4 - Demina randul / coloana cu indexul specificat.");  
 System.*out*.println("5 - Determina numarul liniei / coloanei cu un numar minimal de zone minate");  
 System.*out*.println("6 - Determina media numerelor de zone minate de pe coloanele pare ale terenului.");  
 System.*out*.println("7 - Afiseaza pe ecran lista numerelor de ordine ale liniilor terenului in ordinea ascendenta a numarului total\n" +  
 " de mine plasate pe liniile respective, sortate prin metoda selectiei.");  
 System.*out*.println("8 - Creeaza fisierul text Mine.txt, in care se vor copia doar acele linii ale fisierului de intrare Teren.in, care\n" +  
 " nu conţin mine.");  
 System.*out*.println("9 - Determina numarul de obiecte din terenul minat (matricea binara).");  
 System.*out*.println("10 - Afiseaza traseul cel mai scurt, ce trebuie parcurs de soldat pentru a\n" +  
 " ajunge din zona [1, 1], considerata, evident, neminata, in zona " +  
 String.*format*("[%d, %d]", *teren*.getN(), *teren*.getM()));  
 System.*out*.println("=========================================");  
 }  
  
  
  
 public static void menu() {  
 Scanner scanner = new Scanner(System.*in*);  
 boolean exit = false;  
  
 while (!exit) {  
 *printMenu*();  
 int option = scanner.nextInt();  
  
 switch (option) {  
 case 0:  
 exit = true;  
 break;  
 case 1:  
 System.*out*.println(*teren*);  
 break;  
 case 2:  
 try {  
 *teren*.writeData();  
 } catch (IOException e) {  
 System.*out*.println("Eroare la salvarea modificarilor in fisierul Teren.in.");  
 System.*out*.println(e.getMessage());  
 }  
 break;  
 case 3:  
 System.*out*.println("1 - Inscrie rand.");  
 System.*out*.println("2 - Inscrie coloana.");  
  
 int input = scanner.nextInt();  
  
 if (input == 1) { // inscrie rand  
 System.*out*.println("\t1 - Inscrie randul la NORD.");  
 System.*out*.println("\t2 - Inscrie randul la SUD.");  
  
 int rowDirection = scanner.nextInt();  
  
 if ((rowDirection != 1) && (rowDirection != 2)) {  
 System.*out*.println("Optiune invalida! Optiunile valide sunt: 1 / 2");  
 break;  
 }  
  
 int[] row = new int[*teren*.getM()];  
  
 System.*out*.println("Introduceti "+ *teren*.getM() + " elemente ale randului(0 / 1):");  
 for (int i = 0; i < *teren*.getM(); ++i) {  
 System.*out*.print((i + 1) + ": ");  
 int value = scanner.nextInt();  
  
 if ((value != 1) && (value != 0)) {  
 System.*out*.println("Ati introdus o valoare invalida. Valorile valide: 0 / 1");  
 row = null;  
 break;  
 } else {  
 row[i] = value;  
 }  
 }  
  
 if (row == null) {  
 break;  
 }  
  
 if (rowDirection == 1) {  
 *teren*.insertRow(TerenMinat.RowDirections.*NORTH*, row);  
 } else {  
 *teren*.insertRow(TerenMinat.RowDirections.*SOUTH*, row);  
 }  
 } else if (input == 2) { // inscrie coloana  
 System.*out*.println("\t1 - Inscrie coloana la EST.");  
 System.*out*.println("\t2 - Inscrie coloana la VEST.");  
  
 int columnDirection = scanner.nextInt();  
 scanner.nextLine();  
  
 if ((columnDirection != 1) && (columnDirection != 2)) {  
 System.*out*.println("Optiune invalida! Optiunile valide sunt: 1 / 2");  
 break;  
 }  
  
 int[] column = new int[*teren*.getN()];  
  
 System.*out*.println("Introduceti "+ *teren*.getN() + " elemente ale coloanei(0 / 1):");  
 for (int i = 0; i < *teren*.getN(); ++i) {  
 System.*out*.print((i + 1) + ": ");  
 int value = scanner.nextInt();  
  
 if ((value != 1) && (value != 0)) {  
 System.*out*.println("Ati introdus o valoare invalida. Valorile valide: 0 / 1");  
 column = null;  
 break;  
 } else {  
 column[i] = value;  
 }  
 }  
  
 if (column == null) {  
 break;  
 }  
  
 if (columnDirection == 1) {  
 *teren*.insertColumn(TerenMinat.ColumnDirections.*EAST*, column);  
 } else {  
 *teren*.insertColumn(TerenMinat.ColumnDirections.*WEST*, column);  
 }  
 } else {  
 System.*out*.println("Optiune invalida! Optiunile valide sunt: 1 / 2");  
 }  
 break;  
 case 4:  
 System.*out*.println("1 - Demina rand.");  
 System.*out*.println("2 - Demina coloana.");  
  
 input = scanner.nextInt();  
  
 if (input == 1) {  
 System.*out*.printf("Introduceti numarul randului (1-%d):\n", *teren*.getN());  
 int index = scanner.nextInt();  
 *teren*.demineRow(index - 1);  
 } else if (input == 2) {  
 System.*out*.printf("Introduceti numarul coloanei (1-%d):\n", *teren*.getM());  
 int index = scanner.nextInt();  
 *teren*.demineColumn(index - 1);  
 } else {  
 System.*out*.println("Optiune invalida! Optiunile valide sunt: 1 / 2");  
  
 }  
 break;  
 case 5:  
 System.*out*.println("1 - Randul.");  
 System.*out*.println("2 - Coloana.");  
  
 input = scanner.nextInt();  
  
 if (input == 1) {  
 int rowWithMinimalMines = *teren*.getRowWithMinimalMines() + 1;  
 System.*out*.printf("Randul #%d.\n", rowWithMinimalMines);  
 } else if (input == 2) {  
 int columnWithMinimalMines = *teren*.getColumnWithMinimalMines() + 1;  
 System.*out*.printf("Coloana #%d.\n", columnWithMinimalMines);  
 } else {  
 System.*out*.println("Optiune invalida! Optiunile valide sunt: 1 / 2");  
 }  
 break;  
 case 6:  
 double average = *teren*.getAverageOfEvenColumns();  
 System.*out*.println("\tMedia: " + average);  
 break;  
 case 7:  
 System.*out*.println("Randurile sortate ascendent:");  
 *teren*.printLinesSortedByMinesNumber();  
 break;  
 case 8:  
 System.*out*.println("Copierea liniilor fara mine...");  
 if (*teren*.copyDeminedLinesToFile()) {  
 System.*out*.println("Fisierul Mine.txt a fost creat cu succes.");  
 }  
 break;  
 case 9:  
 System.*out*.printf("In matricea binara sunt: %d obiecte.\n", *teren*.countObjects());  
 break;  
 case 10:  
 try {  
 *teren*.printShortestPath();  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 break;  
 default:  
 System.*out*.println("Comanda necunoscuta!");  
 }  
 }  
  
 }  
  
}

TerenMinat.java:

import java.io.\*;  
import java.util.\*;  
  
public class TerenMinat {  
  
 private static int *n*; // Randuri  
 private static int *m*; // Coloane  
 private static int[][] *teren*;  
  
 enum RowDirections {  
 *NORTH*,  
 *SOUTH*  
}  
  
 enum ColumnDirections {  
 *EAST*,  
 *WEST*  
}  
  
 static {  
 BufferedReader input = null;  
  
 try {  
 input = new BufferedReader(new FileReader("Teren.in"));  
  
 String line = input.readLine();  
 String[] lines = line.split(" ");  
 *n* = Integer.*parseInt*(lines[0]);  
 *m* = Integer.*parseInt*(lines[1]);  
 *teren* = new int[*n*][*m*];  
  
 for (int i = 0; i < *n*; ++i) {  
 line = input.readLine();  
 lines = line.split(" ");  
  
 for (int j = 0; j < *m*; ++j) {  
 *teren*[i][j] = Integer.*parseInt*(lines[j]);  
 }  
 }  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 } finally {  
 if (input != null) {  
 try {  
 input.close();  
 } catch (IOException e) {  
 System.*out*.println("Eroarea la citirea datelor din fisier: " + e.getMessage());  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 public int getM() {  
 return *m*;  
 }  
  
 public int getN() {  
 return *n*;  
 }  
  
 public void writeData() throws IOException {  
 try (BufferedWriter out =  
 new BufferedWriter(new FileWriter("Teren.in"))  
 ) {  
 StringBuilder builder = new StringBuilder();  
 builder.append(*n*).append(" ").append(*m*).append("\n");  
  
 for (int i = 0; i < *n*; ++i) {  
 for (int j = 0; j < *m*; ++j) {  
 builder.append(*teren*[i][j]).append(" ");  
 }  
 builder.append("\n");  
 }  
  
 out.write(builder.toString());  
 }  
 }  
  
 // Sarcina 1.  
 public void insertRow(RowDirections direction, int[] row) {  
 if (direction.equals(RowDirections.*NORTH*)) {  
 insertRowAtNorth(row);  
 } else if (direction.equals(RowDirections.*SOUTH*)) {  
 insertRowAtSouth(row);  
 }  
 }  
  
 public void insertColumn(ColumnDirections direction, int[] column) {  
 if (direction.equals(ColumnDirections.*EAST*)) {  
 insertColumnAtEast(column);  
 } else if (direction.equals(ColumnDirections.*WEST*)) {  
 insertColumnAtWest(column);  
 }  
 }  
  
 private boolean insertRowAtNorth(int[] row) {  
 if (row == null) {  
 return false;  
 }  
  
 int[][] arr = new int[*n* + 1][*m*];  
  
 arr[0] = row;  
  
 if (*n* >= 0) {  
 System.*arraycopy*(*teren*, 0, arr, 1, *n*);  
 }  
  
 *teren* = arr;  
 ++*n*;  
  
 return true;  
 }  
  
 private boolean insertRowAtSouth(int[] row) {  
 if (row == null) {  
 return false;  
 }  
  
 int[][] arr = new int[*n* + 1][*m*];  
  
 if (*n* >= 0) {  
 System.*arraycopy*(*teren*, 0, arr, 0, *n*);  
 }  
  
 arr[arr.length - 1] = row;  
  
 *teren* = arr;  
 ++*n*;  
  
 return true;  
 }  
  
 private boolean insertColumnAtEast(int[] column) {  
 if (column == null) {  
 return false;  
 }  
  
 int[][] arr = new int[*n*][*m* + 1];  
  
 for (int i = 0; i < *n*; ++i) {  
 System.*arraycopy*(*teren*[i], 0, arr[i], 0, *m*);  
 }  
  
 for (int i = 0; i < *n*; ++i) {  
 arr[i][arr[0].length - 1] = column[i];  
 }  
 *teren* = arr;  
 ++*m*;  
  
 return true;  
 }  
  
 private boolean insertColumnAtWest(int[] column) {  
 if (column == null) {  
 return false;  
 }  
  
 int[][] arr = new int[*n*][*m* + 1];  
  
 for (int i = 0; i < *n*; ++i) {  
 arr[i][0] = column[i];  
 }  
  
 for (int i = 0; i < *n*; ++i) {  
 if (*m* >= 0) {  
 System.*arraycopy*(*teren*[i], 0, arr[i], 1, *m*);  
 }  
 }  
  
 *teren* = arr;  
 ++*m*;  
  
 return true;  
 }  
  
 // sarcina 2  
 private boolean ensureRowIndex(int index) {  
 return ((index >= 0) && (index < *n*));  
 }  
  
 private boolean ensureColumnIndex(int index) {  
 return ((index >= 0) && (index < *m*));  
 }  
  
 public void demineRow(int index) {  
 if (!ensureRowIndex(index)) {  
 System.*out*.printf("Index invalid! Indecsii valizi: %d-%d\n", 0, (*n* - 1));  
 }  
  
 for (int i = 0; i < *m*; ++i) {  
 *teren*[index][i] = 0;  
 }  
 }  
  
 public void demineColumn(int index) {  
 if (!ensureColumnIndex(index)) {  
 System.*out*.printf("Index invalid! Indecsii valizi: %d-%d\n", 0, (*m* - 1));  
 }  
  
 for (int i = 0; i < *n*; ++i) {  
 *teren*[i][index] = 0;  
 }  
 }  
  
 // Sarcina 3  
 public int getRowWithMinimalMines() {  
 TreeMap<Integer, Integer> mines = new TreeMap<>((o1, o2) -> o2 - o1); // mines, index  
 for (int i = 0; i < *n*; ++i) {  
 int minesCounter = 0;  
  
 for (int j = 0; j < *m*; ++j) {  
 minesCounter += *teren*[i][j];  
 }  
 mines.putIfAbsent(minesCounter, i);  
 }  
  
 return mines.lastEntry().getValue();  
 }  
  
 public int getColumnWithMinimalMines() {  
 TreeMap<Integer, Integer> mines = new TreeMap<>((o1, o2) -> o2 - o1); // mines, index  
 for (int i = 0; i < *m*; ++i) {  
 int minesCounter = 0;  
  
 for (int j = 0; j < *n*; ++j) {  
 minesCounter += *teren*[j][i];  
 }  
 mines.putIfAbsent(minesCounter, i);  
 }  
  
 return mines.lastEntry().getValue();  
 }  
  
 //Sarcina 4  
 public double getAverageOfEvenColumns() {  
 int sum = 0;  
 int evenRows = *m* / 2;  
  
 for (int i = 1; i < *m*; i += 2) {  
 for (int j = 0; j < *n*; ++j) {  
 sum += *teren*[j][i];  
 }  
 }  
  
 return ((double) sum / evenRows);  
 }  
  
 // Sarcina 5  
 // Metoda selectiei  
 private void entrySelectionSort(List<Map.Entry<Integer, Integer>> entryList) {  
 if ((entryList == null) || (entryList.size() == 1)) {  
 return;  
 }  
  
 for (int i = 0; i < (entryList.size() - 1); ++i) {  
 int minIndex = i;  
  
 for (int j = i + 1; j < entryList.size(); ++j) {  
 int minValue = entryList.get(minIndex).getKey();  
 int actualValue = entryList.get(j).getKey();  
  
 if (actualValue < minValue) {  
 minIndex = j;  
 }  
 }  
 int value1 = entryList.get(i).getKey();  
 int value2 = entryList.get(minIndex).getKey();  
  
 if (value1 != value2) {  
 Collections.*swap*(entryList, i, minIndex);  
 }  
 }  
 }  
  
 public void printLinesSortedByMinesNumber() {  
 ArrayList<Map.Entry<Integer, Integer>> entries = new ArrayList<>(); // sum, row number  
  
 for (int i = 0; i < *n*; ++i) {  
 int sum = 0;  
 for (int j = 0; j < *m*; ++j) {  
 sum += *teren*[i][j];  
 }  
  
 entries.add(Map.*entry*(sum, (i + 1)));  
 }  
  
 entrySelectionSort(entries);  
  
 StringBuilder builder = new StringBuilder("{");  
 for (Map.Entry<Integer, Integer> entry : entries) {  
 builder.append(entry.getValue())  
 .append(", ");  
 }  
 builder.delete(builder.length() - 2, builder.length());  
 builder.append("}");  
  
 System.*out*.println(builder);  
 }  
  
 public boolean copyDeminedLinesToFile() {  
 ArrayList<Integer> emptyLines = new ArrayList<>();  
  
 for (int i = 0; i < *n*; ++i) {  
 boolean hasMines = false;  
  
 for (int j = 0; j < *m*; ++j) {  
 if (*teren*[i][j] == 1) {  
 hasMines = true;  
 break;  
 }  
 }  
  
 if (!hasMines) {  
 emptyLines.add((i + 1));  
 }  
 }  
  
 try (BufferedWriter out =  
 new BufferedWriter(new FileWriter("Mine.txt"))  
 ) {  
 for (Integer element : emptyLines) {  
 out.write(element + "\n");  
 }  
 } catch (IOException ex) {  
 System.*out*.println("Eroare la crearea/scrierea datelor in Mine.txt: " + ex.getMessage());  
 return false;  
 }  
  
 return true;  
 }  
  
 public int countObjects() {  
 int[][] matrixCopy = new int[*n*][*m*];  
 for (int i = 0; i < *n*; ++i) {  
 System.*arraycopy*(*teren*[i], 0, matrixCopy[i], 0, *m*);  
 }  
  
 int counter = 0;  
  
 for (int i = 0; i < *n*; ++i) {  
 for (int j = 0; j < *m*; ++j) {  
 if (matrixCopy[i][j] == 1) {  
 *cleanBlock*(matrixCopy, i, j);  
 ++counter;  
 }  
 }  
 }  
  
 return counter;  
 }  
  
 public static void cleanBlock(int[][] matrix, int i, int j) {  
 matrix[i][j] = 0;  
 Cell up = new Cell(i - 1, j);  
 Cell down = new Cell(i + 1, j);  
 Cell left = new Cell(i, j - 1);  
 Cell right = new Cell(i, j + 1);  
 Cell upLeft = new Cell(i - 1, j - 1);  
 Cell upRight = new Cell(i - 1, j + 1);  
 Cell downRight = new Cell(i + 1, j + 1);  
 Cell downLeft = new Cell(i + 1, j - 1);  
  
 if ((up.existsIn(matrix))  
 && (matrix[up.row][up.col] == 1)) {  
 *cleanBlock*(matrix, up.row, up.col);  
 }  
 if ((down.existsIn(matrix))  
 && (matrix[down.row][down.col] == 1)) {  
 *cleanBlock*(matrix, down.row, down.col);  
 }  
 if ((left.existsIn(matrix))  
 && (matrix[left.row][left.col] == 1)) {  
 *cleanBlock*(matrix, left.row, left.col);  
 }  
 if ((right.existsIn(matrix))  
 && (matrix[right.row][right.col] == 1)) {  
 *cleanBlock*(matrix, right.row, right.col);  
 }  
 if ((upLeft.existsIn(matrix))  
 && (matrix[upLeft.row][upLeft.col] == 1)) {  
 *cleanBlock*(matrix, upLeft.row, upLeft.col);  
 }  
 if ((upRight.existsIn(matrix))  
 && (matrix[upRight.row][upRight.col] == 1)) {  
 *cleanBlock*(matrix, upRight.row, upRight.col);  
 }  
 if ((downRight.existsIn(matrix))  
 && (matrix[downRight.row][downRight.col] == 1)) {  
 *cleanBlock*(matrix, downRight.row, downRight.col);  
 }  
 if ((downLeft.existsIn(matrix))  
 && (matrix[downLeft.row][downLeft.col] == 1)) {  
 *cleanBlock*(matrix, downLeft.row, downLeft.col);  
 }  
 }  
 // Sarcina 8  
 public void printShortestPath() throws Exception {  
 if ((*teren*[0][0] == 1) || *teren*[*n* - 1][*m* - 1] == 1) {  
 throw new Exception("Zona [1, 1] sau zona " +  
 String.*format*("[%d, %d] este minata, respectiv este imposibil de gasit traseul.", *n*, *m*));  
 }  
 Stack<Cell> path = new Stack<>();  
 shortestPath(*teren*, new Cell(0, 0), new Cell(*n* - 1, *m* - 1), path);  
  
 if (path.size() == 0) {  
 throw new Exception("Nu a fost gasit un traseu valid pentru a ajunge din zona [1, 1] in zona " +  
 String.*format*("[%d, %d].", *n*, *m*));  
 }  
 StringBuilder builder = new StringBuilder();  
 builder.append("Lungimea celui mai scurt traseu: ").append(path.size()).append("\n");  
  
 while (!path.empty()) {  
 Cell cell = path.pop();  
 builder.append("[")  
 .append(cell.row + 1)  
 .append(", ")  
 .append(cell.col + 1)  
 .append("]-");  
 }  
 // sterge ultimul '-'  
 builder.deleteCharAt(builder.length() - 1);  
 System.*out*.println(builder.toString());  
 }  
 private int shortestPath(int[][] map, Cell start, Cell end,  
 Stack<Cell> path) {  
 // initializeaza matricea cu distante  
 int[][] distances = new int[map.length][];  
 for (int i = 0; i < map.length; i++) {  
 distances[i] = new int[map[i].length];  
 Arrays.*fill*(distances[i], Integer.*MAX\_VALUE*); // initializeaza fiecare celula cu Integer.MAX\_VALUE  
 }  
 // celula de start are distanta 0  
 int distance = 0;  
 List<Cell> currentCells = Arrays.*asList*(start);  
  
 // ruleaza pana cand nu ajunge la sfarsit si celulele curente nu sunt goale  
 while (distances[end.row][end.col] == Integer.*MAX\_VALUE*  
&& !currentCells.isEmpty()) {  
 List<Cell> nextCells = new ArrayList<>(); // lista pentru urmatoarele celule  
 // itereaza asupra la toate celulele din iteratia precedenta  
 // seteaza distantele lor  
 // adauga vecinii fiecarei celule in lista, pentru a ii procesa la urmatoarea iteratie  
 for (Cell cell : currentCells) {  
 if ((distances[cell.row][cell.col] == Integer.*MAX\_VALUE*)  
 && !(map[cell.row][cell.col] == 1)) {  
 distances[cell.row][cell.col] = distance;  
 addNeighbors(cell, nextCells, map.length, map[0].length);  
 }  
 }  
 // pregateste pentru urmatoarea iteratie  
 currentCells = nextCells;  
 distance++;  
 }  
 // adauga coordonatele drumului gasit intr-un Stack  
 if (distances[end.row][end.col] < Integer.*MAX\_VALUE*) {  
 Cell cell = end;  
 path.push(end);  
 for (int d = distances[end.row][end.col] - 1; d >= 0; d--) {  
 cell = getNeighbor(cell, d, distances);  
 path.push(cell);  
 }  
 }  
 // returneaza cea mai scurta distanta  
 return distances[end.row][end.col];  
 }  
 // adauga toti vecinii valizi al unei celule in lista respectiva  
 private void addNeighbors(Cell cell, List<Cell> list,  
 int maxRow, int maxCol) {  
 // directiile  
 int[][] ds = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};  
  
 for (int[] d : ds) {  
 int row = cell.row + d[0];  
 int col = cell.col + d[1];  
 if (*isValid*(row, col, maxRow, maxCol)) // daca e valida, atunci adauga  
 list.add(new Cell(row, col));  
 }  
 }  
 // gaseste vecinul unei celule ce se afla la o anumita {distance} de la coordonatele de start  
 private Cell getNeighbor(Cell cell, int distance, int[][] distances) {  
 int[][] ds = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};  
  
 for (int[] d : ds) {  
 int row = cell.row + d[0];  
 int col = cell.col + d[1];  
 if (*isValid*(row, col, distances.length, distances[0].length)  
 && distances[row][col] == distance)  
 return new Cell(row, col);  
 }  
 return null;  
 }  
 // verifica daca coordonatele se afla in interiorul matricei  
 private static boolean isValid(int row, int col, int maxRow, int maxCol) {  
 return (row >= 0) && (row < maxRow) && (col >= 0) && (col < maxCol);  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder result = new StringBuilder(" ");  
 for (int i = 0; i < *m*; ++i) { // adaugarea numerelor de ordine ale coloanelor  
 result.append(i + 1)  
 .append((i < 9) ? " " : " ");  
 }  
 result.append("\n ");  
  
 // adaugarea dash-urilor  
 result.append("- ".repeat(Math.*max*(0, *m*)));  
 result.append("\n");  
  
 for (int i = 0; i < *n*; ++i) {  
 result.append((i + 1)) // adaugarea numerelor de ordine ale randurilor  
 .append((i < 9) ? " " : "")  
 .append(": ");  
  
 for (int j = 0; j < *m*; ++j) { // adaugarea elementelor randurilor  
 result.append(*teren*[i][j]);  
 result.append(" ");  
 }  
 result.append("\n");  
 }  
 return result.toString();  
 }  
}

**Cell.java:**

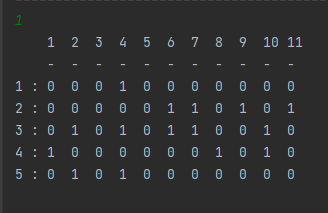
public class Cell {  
  
 public int row;  
 public int col;  
  
 public Cell(int row, int col) {  
 this.row = row;  
 this.col = col;  
 }  
  
 public boolean existsIn(int[][] matrix) {  
 return ((row >= 0) && (col >= 0) && (row < matrix.length) && (col < matrix[row].length));  
 }  
  
 @Override  
 public String toString() {  
 return "{" + row + ", " + col + "}";  
 }  
  
}

**Teren.in:**

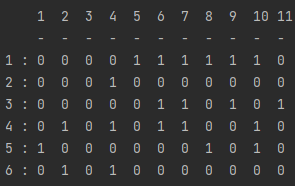
5 11  
0 0 0 1 0 0 0 0 0 0 0  
0 0 0 0 0 1 1 0 1 0 1  
0 1 0 1 0 1 1 0 0 1 0  
1 0 0 0 0 0 0 1 0 1 0  
0 1 0 1 0 0 0 0 0 0 0

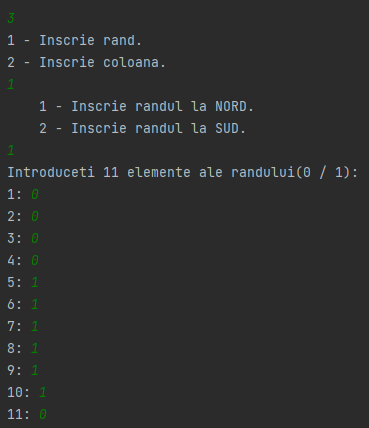
# 6. Rezultatele testării subprogramelor

Afișarea terenului:



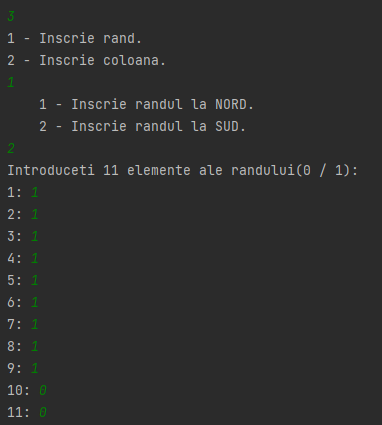
**Sarcina 1.** Înscrie în planul terenului un nou rând (marginal) / o nouă coloană (marginală); alternativa aleasă şi poziţia rândului (nord/sud) / coloanei (vest/est) de înscris se vor preciza de la tastatură.

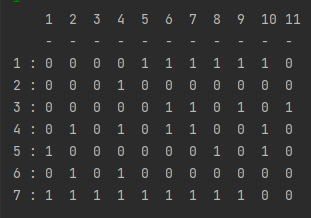
1. Înscrie rând la nord.



Rezultatul înscrierii:

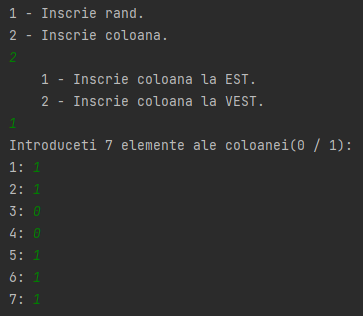
1. Înscrie rând la sud:



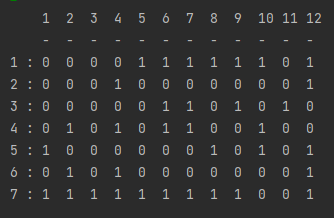


Rezultatul înscrierii:

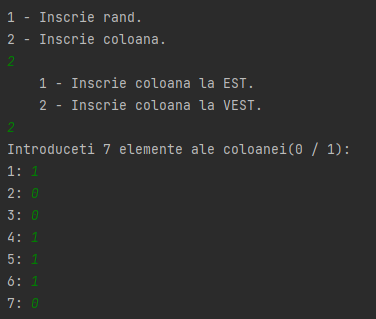
1. Înscrie coloană la est:



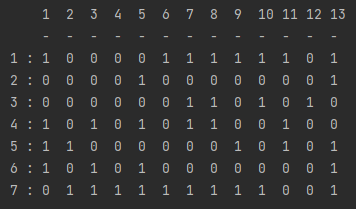
Rezultatul înscrierii:



1. Înscrie coloană la vest:



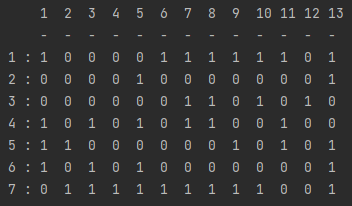
Rezultatul înscrierii:



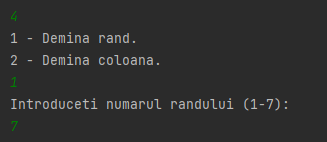
**Sarcina 2.** „Demină” zonele unui rând / unei coloane; alternativa aleasă şi numărul de ordine al rândului / coloanei de „deminat” se va preciza de la tastatură.

1. Demină rând.

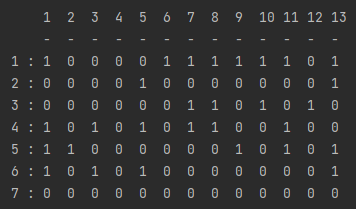
Terenul inițial:



Demină rândul 7:

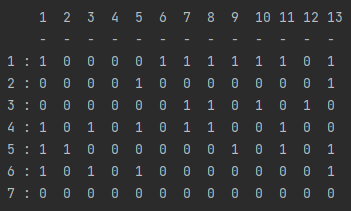


Terenul după deminare:

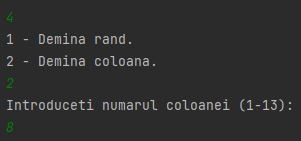


1. Demină coloană:

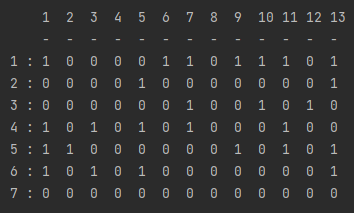
Terenul inițial:



Demină coloana 8:

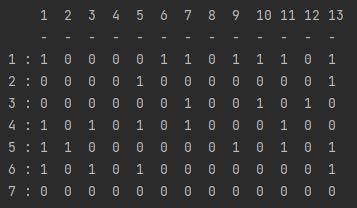


Terenul după deminare:

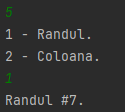


**Sarcina 3.** Determină numărul liniei/coloanei cu un număr minimal de zone minate.

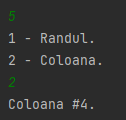
Terenul:



1. Pentru linie:

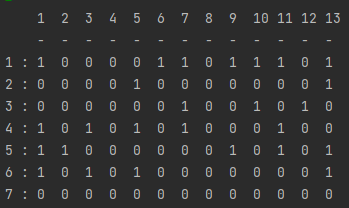


1. Pentru coloană:



**Sarcina 4.** Determină media numerelor de zone minate de pe coloanele pare ale terenului în studiu.

Terenul:

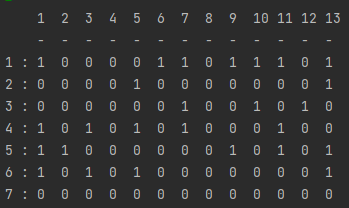


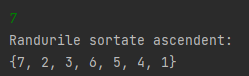
Rezultatul:



**Sarcina 5.** Afişează pe ecran lista numerelor de ordine ale liniilor terenului în ordinea ascendentă a numărului total de mine plasate pe liniile respective; datele se vor sorta prin metoda selecției.

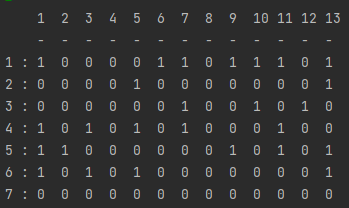
Terenul:

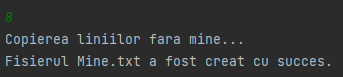




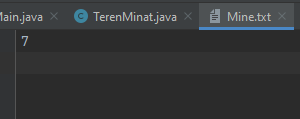
**Sarcina 6.** Creează fişierul text Mine.txt, în care se vor copia doar acele linii ale fişierului de intrare Teren.in, care nu conţin mine.

Terenul:



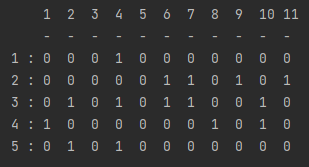


Mine.txt: (conține numărul de ordine ale liniilor)



**Sarcina 7.** Determină numărul de obiecte din matricea binară T. **Notă.** Un obiect este format din elemente cu valoarea 1 care se învecinează pe linii, pe coloane sau pe diagonale.

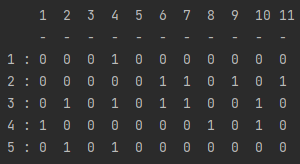
Terenul:



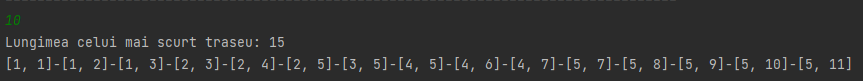


**Sarcina 8.** Rezolvă problema. Un soldat, având la dispoziţie un detector de mine, porneşte dintr-un colţ al terenului şi trebuie să ajungă în colţul opus. Pe terenul considerat, soldatul se poate deplasa doar ortogonal şi, evident, fără a nimeri în careva din zonele minate.

Terenul:



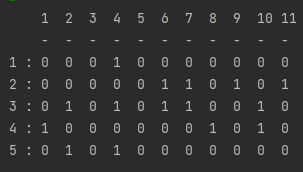
Rezultatul:



**Opțiuni bonus:**

0 – Ieșire din program.

1 – Afișează terenul.



2 – Salvează in Teren.in modificările realizate.

# 7. Concluzie

Datorită practicii date am obținut experiență în rezolvarea problemelor și folosirea algoritmilor. Am experimentat cu combinarea principiilor POO împreună cu diverși algoritmi, prin urmare am reușit să scriu mai puțin cod, care se descurcă de minune cu sarcinile propuse.

Practica dată m-a ajutat să înțeleg cum e să lucrezi într-un colectiv de oameni zilnic. Am înțeles că este nevoie de multă concentrație și răbdare pentru a face față sarcinilor propuse.

De asemenea practica te impune să gestionezi propriul timp într-un mod care ar permite realizarea tuturor sarcinilor și altor obligații în termenii prestabiliți.

Această practică m-a făcut să înțeleg importanța structurii programului, clean codului și a diferitor principii care îți permit dezvoltarea și întreținerea unui produs.

# 8. Bibliografie

1. A. Braicov. Turbo Pascal. Culegere de probleme, Chişinău, Editura "Prut Internaţional", 2007.

2. K. Sierra, Head First Java, 2nd Edition, Editura „O'Reilly Media” Martie 1, 2005.

3. J. Bloch, Effective Java 3rd Edition, Editura „‎ Addison-Wesley Professional” Decembrie 27, 2017.

4. H. Schildt, Java: The Complete Reference, 11th Edition, Editura „McGraw-Hill Education” December 12, 2018