Лабораторна робота № 8

Робота з двовимірними масивами. Читання даних з текстового файлу та запис в текстовий файл.

Мета роботи: Оволодіння практичними навичками роботи з двовимірними масивами, читання-запису текстових файлів та опрацювання виняткових ситуацій.

Короткі теоретичні відомості.

Двовимірний масив в Java є не що інше як масив одновимірних масивів. Елементи двовимірних масивів, відповідно, індексуються парою індексів – кожен записують в окремих квадратних дужках. Загальна форма оголошення двовимірного масиву:

```
<тип елементів> <назва масиву>[<розмірність1>][<розмірність2>];
```

Як і у випадку одновимірного масиву дозволяється записувати розмірність масиву відразу і після типу елементів. Приклади оголошення двовимірних масивів:

```
int[][] A = new int[5][5];
float B[][] = new float[3][2];
```

Для опрацювання двовимірних масивів використовують вкладені цикли for, наприклад:

```
int max=A[1][1];
for(int i=0;i<5;i++)
      for(int j=0;j<5;j++)
            if (max < A[i][j]) max = A[i][j];
```

Дозволяється також ініціалізація двовимірного масиву елементами під час опису, наприклад:

```
int[][]A ={
      {5, 6, 1, 3},
      {3, 4, 2, 1},
      {1, 2, 2, 2}
};
```

Для **читання даних** з текстового файлу можна використовувати клас BufferedReader, як і для зчитування даних з консолі. Для цього його слід ініціалізувати представником класу FileReader, який, в свою чергу, конструюється на основі об’єкта класу File:

```
BufferedReader input = new BufferedReader(new FileReader(new File("data.txt")))
```

Але такий спосіб передбачає використання методів read() чи readLine(), якими складно опрацьовувати стрічки тексту, що містять по декілька числових даних. Набагато зручнішим в таких випадках є використання класу java.util.Scanner, що має метод hasNext() для перевірки того чи є у вхідному потоці чергова порція даних. На випадок вводу цілих чисел використовують метод hasNextInt() (для дійсних hasNextFloat(), hasNextDouble()). Якщо наступна порція даних є, то її можна прочитати методами next(), nextInt() (nextFloat, nextDouble), відповідно. Організувати такий ввід даних можна аналогічно до попереднього:

```
Scanner input = new Scanner(new FileReader(new File("data.txt")))
```

Для **запису даних** використовують потік виводу що є екземпляром классу PrintWriter. Цей об’єкт взаємодіє з потоком виводу в файл типу FileWriter, через відповідний буфер BufferedWriter. Наприклад

```
File fB = new File("D:\\B.txt");
FileWriter fileWrite = new FileWriter(fB);
BufferedWriter writeBufer = new BufferedWriter(fileWrite);
PrintWriter printB = new PrintWriter(writeBufer);
```

Або коротше з використанням анонімних об’єктів:

```
FileWriter fileWrite = new FileWriter(new File("D:\\B.txt"));
PrintWriter printB = new PrintWriter(new BufferedWriter(fileWrite));
```

При цьому можна використовувати ті ж методи print(), println(), printf(), format(), що і при виводі даних на консоль.

Зчитування даних з файлів – процес при якому часто можуть виникати помилки, наприклад, відсутність файла з даними, чи неправильний запис даних у файлі. У класичних мовах програмування, наприклад, в C, доводилося перевіряти якусь умову, що вказувала на наявність помилки, і в залежності від цього робити ті чи інші дії. В Java з’явилося більш просте рішення - обробка виняткових ситуацій.

```
try{
   someAction();
   anotherAction();
} catch(Exception e) {
   // обробка винятку
}
```

При виникненні виняткової ситуації управління передається від коду, що викликав виняткову ситуацію, на найближчий блок catch (або вгору по стеку) і створюється об'єкт, успадкований від класу Throwable, або його нащадків який містить інформацію про виняткову ситуацію і використовується при її обробці. Власне, в блоці catch вказується саме клас оброблюваної ситуації.

Допускається створення власних класів виняткових ситуацій. Здійснюється це за допомогою механізму успадкування, тобто клас виняткової ситуації, створений користувачем повинен бути успадкований від класу Throwable, або його нащадків.

Конструкція try-catch в загальному випадку виглядає так:

```
try {
   ...
} catch(SomeExceptionClass e) {
   ...
} catch(AnotherExceptionClass e) {
   ...
}
```

Спочатку виконується код в фігурних дужки після оператора try. Якщо під час його виконання не відбувається ніяких позаштатних ситуацій, то далі управління передається

за закриваючу фігурну дужку останнього оператора catch, асоційованого з даним оператором try. Якщо в межах try виникає виняткова ситуація, то далі виконання коду проводиться по одному з перерахованих нижче сценаріїв:

- Якщо виникла виняткова ситуація, клас якої вказаний як параметр одного з блоків catch, то виконується блок коду, асоційований з даним catch. Далі, якщо код в цьому блоці завершується нормально, то і весь оператор try завершується нормально і управління передається на оператор після закриваючої фігурної дужки останнього catch.
- Якщо код в catch завершується некоректно, то і весь try завершується некоректно з тієї ж причини.
- Якщо виникла виняткова ситуація, клас якої не зазначений в якості аргументу ні в одному catch, то виконання всього try завершується некоректно.

Конструкція **try-catch-finally** дозволяє за допомогою оператора finally гарантувати виконання будь-якого фрагмента коду, незалежно від того виникла виняткова ситуація чи ні. Послідовність виконання такої конструкції наступна:

- якщо оператор try виконаний нормально, то буде виконаний блок finally. У свою чергу, якщо оператор finally виконується нормально, то і весь оператор try виконується нормально.
- Якщо під час виконання блоку try виникає виняток і існує оператор catch, який перехоплює даний тип винятку, відбувається виконання пов'язаного з catch блоку. Якщо блок catch виконується нормально, або ненормально, все одно потім виконується блок finally. Якщо блок finally завершується нормально, то оператор try завершується так само, як завершився блок catch.
- Якщо в списку операторів catch чи не знаходиться такого, який обробив б виникло виняток, то все одно виконується блок finally. У цьому випадку, якщо finally завершиться нормально, весь try завершиться ненормально з тієї ж причини, по якій було порушено виконання try.

У всіх випадках, якщо блок finally завершується ненормально, то весь try завершиться ненормально з тієї ж причини.

```
try {
   byte [] buffer = new byte[128];
   FileInputStream fis = new FileInputStream("file.txt");
   while(fis.read(buffer) > 0) {
      ... опрацювання даних ...
   }
} catch(IOException es) {
      ... опрацювання винятку ...
} finally {
   fis.flush();
   fis.close();
}
```

Якщо в даному прикладі помістити оператори очищення буфера і закриття файлу відразу після закінчення обробки даних, то при виникненні помилки вводу / виводу коректного закриття файлу не відбудеться

Завдання. Скласти Java-програму розв'язання наступної задачі: Задано цілочисельний масив (матриця) А розмірності 5х5. Елементи вхідного масиву А записані в текстовому файлі на диску (файл створити самостійно за допомогою текстового редактора, заповнити довільно – 5 рядків по 5 чисел). Потрібно:

прочитати елементи масиву А із відповідного текстового файлу;

в масиві А знайти вказані у варіанті завдання величини та вивести результат у вигляді повідомлення на консоль;

утворити новий масив В, згідно з вказівками відповідного варіанту, утворений масив вивести у новий текстовий файл на диску;

В програмі передбачити опрацювання виняткових ситуацій: відсутність файлу з масивом А, неправильний запис елементів масиву в файлі, недостатня кількість елементів у файлі та помилки при записі у файл масиву В. В якості обробки виняткових ситуацій реалізувати вивід повідомлення із текстом по відповідну помилку на консоль.

Варіанти завдань

1. В масиві А знайти кількість, суму та середнє значення додатних елементів. Масив В утворити з масиву А, замінивши в ньому всі додатні елементи числом 5, а елементи, менші за -5 – середнім значенням додатних елементів.
2. В масиві А знайти різницю середніх значень окремо взятих додатних та від'ємних елементів. Масив В утворити з масиву А, замінивши в ньому всі елементи менші за -5 на протилежні.
3. В масиві А знайти кількості елементів, значення яких лежать в діапазонах від 1 до 10, від 11 до 25 та від 1 до 20. Масив В утворити з масиву А, замінивши в ньому всі елементи на протилежні.
4. В матриці А знайти добуток мінімальних елементів кожного рядка. Матрицю В утворити транспонуванням матриці А.
5. В матриці А знайти суму середніх значень елементів рядків. Матрицю В утворити з матриці А, замінивши всі елементи під головною діагоналлю знайденою сумою.
6. В матриці А знайти відношення кількості додатних елементів до кількості від'ємних. Матрицю В утворити з матриці А, відобразивши її симетрично відносно середнього рядка.
7. В матриці А знайти кількості додатних елементів у кожному рядку. Матрицю В утворити з матриці А, видаливши з неї перший стовпчик, змістивши решту її стовпців на один вліво та додавши останнім стовпцем знайдені кількості додатних елементів рядка.
8. В матриці А знайти суму елементів, розташованих над головною діагоналлю та суму елементів, розташованих під нею. Матрицю В утворити з матриці А, замінивши елементи головної діагоналі різницею знайдених сум.
9. В матриці А знайти відношення мінімального з додатних елементів до максимального з від'ємних. Матрицю В утворити з матриці А перестановкою її стовпців в зворотному порядку.
10. В масиві А знайти відношення значень максимального та мінімального елементів. Масив В утворити з масиву А, замінивши в ньому всі елементи більші за 10 на максимальний елемент, а від'ємні елементи на 0.
11. В матриці А знайти відношення середнього значення елементів, розташованих над головною діагоналлю до середнього значення елементів цієї діагоналі. Матрицю В утворити з матриці А перестановкою її рядків в зворотному порядку.
12. В матриці А знайти середнє арифметичне елементів, розташованих над головною діагоналлю та середнє елементів, розташованих під нею. Матрицю В утворити з матриці А, відобразивши її симетрично відносно другорядної діагоналі.

13. В матриці А знайти кількості додатних елементів у кожному стовпці. Матрицю В утворити з матриці А, видаливши з неї перший рядок, змістивши решту її рядків на один вгору та додавши останнім рядком знайдені кількості додатних елементів стовпця.
14. В матриці А знайти відношення кількості нульових елементів до кількості ненульових. Матрицю В утворити з матриці А, відобразивши її симетрично відносно середнього стовпця.
15. В матриці А знайти суму середніх значень елементів стовпців. Матрицю В утворити з матриці А, замінивши всі елементи над головною діагоналлю знайденою сумою.
16. В масиві А знайти розмах значень його елементів (різницю значень максимального та мінімального елементів). Масив В утворити з масиву А, замінивши в ньому всі елементи більші за 5 на мінімальний елемент, а елементи, менші за -5 – на максимальний.

Приклад виконання лабораторної роботи.
Варіант 16. В масиві А знайти розмах значень його елементів (різницю значень максимального та мінімального елементів). Масив В утворити з масиву А, замінивши в ньому всі елементи більші за 5 на мінімальний елемент, а елементи, менші за -5 – на максимальний.

Код програми програми для розв'язання задачі може бути таким:

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.InputMismatchException;
import java.util.NoSuchElementException;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Lab8 {

public static void main(String[] args) {
	FileReader fA;
	File fB = new File("D:\\B.txt");
	int[][] A = new int[5][5];
	int[][] B = new int[5][5];
	try {
		fA = new FileReader("D:\\A.txt");
		Scanner input = new Scanner(fA);
		for(int i=0;i<5;i++)
			for(int j=0;j<5;j++)
				A[i][j] = input.nextInt();
		input.close();
	} catch (FileNotFoundException e) {
		System.out.println("Файл з елементами масиву A не створено");
		return;
	} catch (InputMismatchException e) {
		System.out.println("Перевірте запис елементів масиву A у файлі");
		return;
	} catch (NoSuchElementException e) {
		System.out.println("У файлі недостатньо елементів для масиву 5x5");
		return;
```

```
}
	int max=A[1][1];
	int min=A[1][1];
	for(int i=0;i<5;i++)
		for(int j=0;j<5;j++){
			if (max < A[i][j]) max = A[i][j];
			if (min > A[i][j]) min = A[i][j];
		}
	System.out.println("Розмах значень елементів масиву A: " + (max- min));

	for(int i=0;i<5;i++)
		for(int j=0;j<5;j++)
			if (A[i][j]<-5) B[i][j] = max;
				else {
					if(A[i][j]>5) B[i][j] = min;
					else B[i][j]= A[i][j];
				}
	try {
		FileWriter fileWrite = new FileWriter(fB);
		BufferedWriter writeBufer = new BufferedWriter(fileWrite);
		PrintWriter printB = new PrintWriter(writeBufer);
		for(int i=0;i<5;i++){
			for(int j=0;j<5;j++) printB.print(B[i][j] + " ");
			printB.println();
		}
		printB.close();
	} catch (IOException e) {
		System.out.println("Помилка при створенні файла з масивом B");
	}
	}
}
```

Основні елементи запропонованого коду:
На початку тіла методу main оголошено масиви, та файли для читання / запису даних. Файл для виводу масиву B можна оголошувати відразу, при фізичній відсутності цього файла на диску його буде створено, оскільки в цей файл буде виконуватися запис. Відсутність файла для зчитування масиву A призведе до помилки, тому створення FileReader-a fA на основі дискового файла D:\\A.txt виконується всередині блоку try разом з читанням елементів масиву A.
Для читання даних з файла використано сканер input та метод nextInt() класу Scanner, про який йшлося в теоретичних відомостях.
Три блоки catch охоплюють найбільш поширені при читанні з файлів види винятків. Послідовність в якій вони записані, очевидно, є суттєвою. Кожен catch завершується оператором return, що завершує виконання методу main – при виникненні хоча б однієї з описаних помилок подальше виконання методу є безглуздим.
У випадку успішного зчитування масиву A метод main продовжить виконання з коду, записаного після закриваючої фігурної дужки останнього catch. Там за допомогою вкладених циклів for реалізовано опрацювання масиву A та утворення масиву B, задекларованe в умові задачі.
Завершується код методом створенням потрібних для запису буферів, та власне записом елементів масиву B у файл. Перехоплення виняткової ситуації в цій частині коду не обов'язкове – помилки при записі даних у файл виникають значно рідше. Хоча і цілком можливі, наприклад запис файла на диск до якого користувач не має доступу і ін.