

MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII

AL REPUBLICII MOLDOVA

Centrul de Excelență în Informatică și Tehnologii Informaționale

RAPORT

LA PRACTICA DE ÎNȚIERE ÎN SPECIALITATE

SPECIALITATEA "Programare și analiza produselor program"

ТЕМА : "Банк и Минное поле(13)"

A elaborat elevul

Ганзюк Александр, Р-2124Р

Conducătorul practicii

Cerbu Olga

Chișinău 2023

Содержание

Введение	3
Задание 1 (C++)	4
Условие(вариант 13)	4
Входные данные	5
Описание подпрограммы:	6
<i>Меню</i>	6
<i>Вывод матрицы в главном меню</i>	10
<i>Замена первого и последнего столбца</i>	11
<i>Функция для очистки строки и/или столбца</i>	13
<i>Вывод строки и/или столбца с максимальным количеством мин</i>	15
<i>Вывод среднего количества мин в нечётных строках</i>	17
<i>Вывод колонок с минами в убывающем порядке</i>	19
<i>Копирование строк с минами в отдельный файл</i>	21
<i>Функция для определения количества объектов в бинарной матрице T</i>	23
<i>Функция для поиска кратчайшего пути с левого верхнего до правого нижнего угла</i>	25
<i>Функция для повторного считывания матрицы из файла</i>	29
Вспомогательная функция	30
<i>Смена цвета</i>	30
Задание 2 (Java)	31
Условие(вариант 13)	31
.....	65
Вывод	66
Библиография	66
C++	66
Java	66
Приложение	67
C++	67
Java	77
BankSystem.java	77
BankBranch.java.....	78
Client.java.....	79
IndividualClient.java	81
CorporateClient.java	82
Credit.java	84
Contract.java	86
BankManagementSystem.java	88

Введение.

Период практики: 3 недели.

С 26 мая 2023 по 19 июня 2023.

Время практики с 14:00 по 17:00

Цели практики введения в специальность:

- ✓ укрепление теоретических знаний и углубление практических навыков, полученных студентами в период изучения модулей;
- ✓ применение технологий разработки программных продуктов, электронных таблиц, текстовых документов и электронных презентаций;
- ✓ развитие навыков самостоятельной и командной работы;

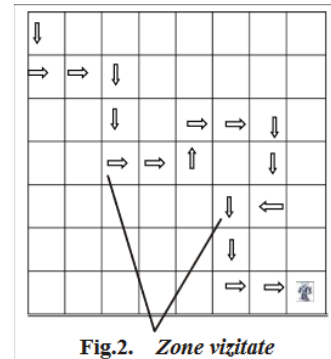
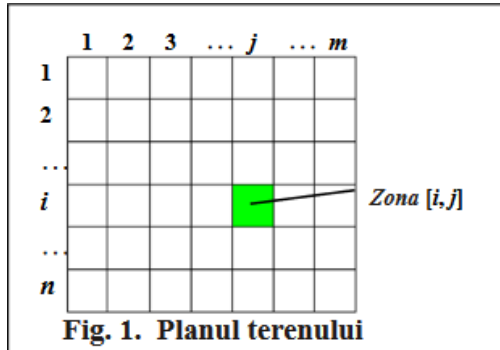
Практика проходила в нашем колледже. Начиная с 26 мая до 19 июня мы приходили в колледж в 14:00. Мне очень понравился период практики и представленные задания, было интересно их выполнять.

Во время практики я использовал такие инструменты как: Code Blocks, IntelliJ IDEA и Microsoft Word.

Задание 1 (C++)

Условие(вариант 13)

Напишите программу на языке C++, решающую следующую задачу:

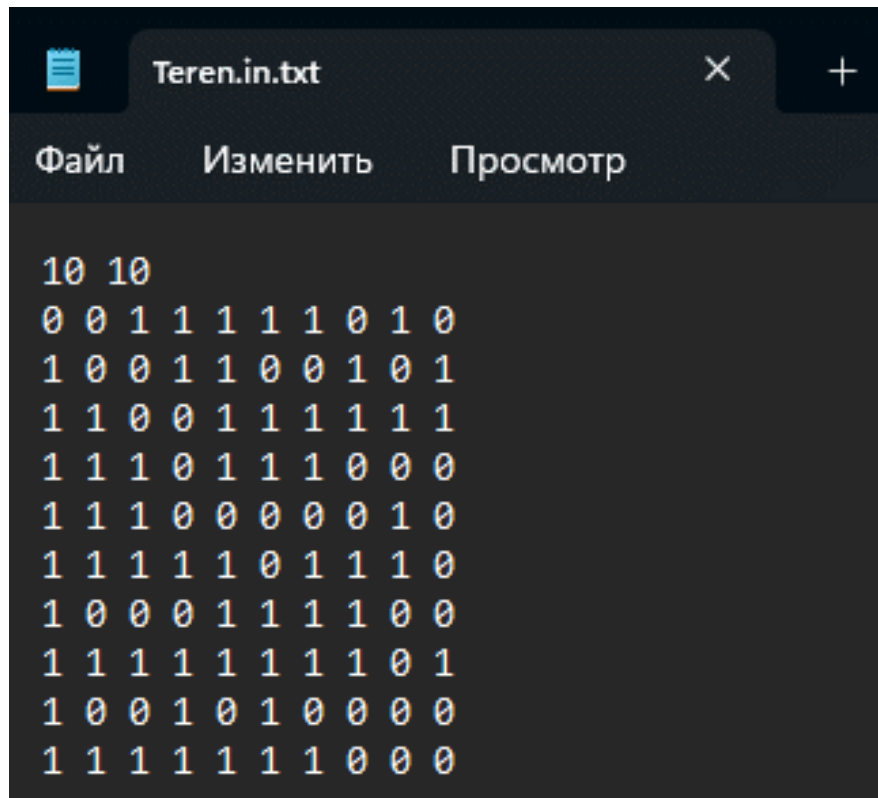


Минное поле. План минного поля имеет форму прямоугольной таблицы размером $n \times m$ ($n, m \leq 50$) и разделен на квадратные зоны длиной 1 (см. рисунок 1). В каждой реальной зоне поля может быть размещена мина. Более конкретная информация о исследуемом поле содержится в текстовом файле *Teren.in*, в котором на первой строке записаны натуральные числа n и m , разделенные пробелом. Каждая из следующих n строк этого файла содержит по m бинарных цифр, разделенных пробелами - элементы матрицы T , в которой $T[i, j] = 1$, если зона $[i, j]$ содержит мину, и $T[i, j] = 0$, если зона свободна. Создайте приложение, которое по запросу пользователя будет выполнять следующие инструкции: (Дизайн приложения на ваше усмотрение).

1. Поменять местами первый и последний столбец поля;
2. "Разминировать" зоны строки/столбца; выбор альтернативы и номер строки/ столбца для "разминирования" будут указаны с клавиатуры;
3. Определить номер строки/столбца с максимальным числом заминированных зон;
4. Определить среднее количество заминированных зон на нечетных строках исследуемого поля;
5. Вывести на экран список порядков столбцов поля в порядке убывания общего количества размещенных в них мин; данные должны быть отсортированы методом подсчета;
6. Создать текстовый файл *Mine.txt*, в котором будут скопированы только те строки исходного файла *Teren.in*, которые содержат мины;
7. Определить количество объектов в бинарной матрице T . Примечание. Объект состоит из элементов со значением 1, соседствующих по строкам, столбцам или диагоналям.
8. Решить проблему. Солдат, имеющий металлоискатель, начинает движение из одного угла поля и должен достичь противоположного угла. На рассматриваемом поле солдат может двигаться только ортогонально и, конечно, не попадать в минные зоны. Определить кратчайший путь, который должен пройти солдат, чтобы достичь зоны $[1, 1]$, считая ее свободной, в зону $[n, m]$. Входные данные. Информация о размерах поля и заминированных зонах территории содержится в текстовом файле *Teren.in*, описанном ранее. Выходные данные. На экран выводится найденный путь, описанный координатами соответствующих зон. Например, для иллюстрации на рисунке 2 путь будет выведен следующим образом: $[1, 1] - [2, 1] - [2, 2] - [3, 2] - [3, 3] - [4, 3] - [4, 4] - [4, 5] - [3, 5] - [3, 6] - [3, 7] - [4, 7] - [5, 7] - [5, 6] - [6, 6] - [7, 6] - [7, 7] - [7, 8]$

Входные данные

Teren.in.txt



```
10 10
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0
```

Описание подпрограммы:

Меню

Описание: в моем проекте меню используется для того, чтобы взаимодействовать с функциями. Моё меню предоставляет пользователю много возможностей, такие как: замена первой и последней колонки, очистка строки и/или столбца, получение строки и/или столбца с наибольшим количеством мин, вычисление среднего значения мин в нечётных строках, вывод колонок с минами в убывающем порядке, копирование строк с минами в отдельный файл, подсчёт объектов в матрице, поиск кратчайшего пути с левого верхнего до правого нижнего угла, повторное считывание данных из файла и выход. Для управления в меню используется клавиатура

Меню

```
Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice:
```

Фрагмент кода:

```
int main() {
    setcolor(3,0);
    int field[MAX_SIZE][MAX_SIZE];
    int n, m;
    if(readField(field, n, m) == 0) return 10;

    int choice;
    do{
        cout << "Matrix\n";
        writeField(field,n,m);
        cout << "\nMenu:\n";
        cout << "1. Swap first and last columns\n";
        cout << "2. Clear zones in a row/column\n";
        cout << "3. Get row/column with max mines\n";
        cout << "4. Calculate average mined zones in odd rows\n";
        cout << "5. Sort columns by number of mines\n";
        cout << "6. Copy rows with mines to a file\n";
        cout << "7. Count objects\n";
        cout << "8. Find shortest path from top-left to bottom-right\n";
        cout << "9. Read data from a file\n";
        setcolor(4,0);
        cout << "10. Exit\n";
        setcolor(3,0);
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:{
                swapColumns(field, n, m);
                cout << "\nField after swapping first and last columns:" << endl;
                writeField(field, n, m);
                system("pause");
                system("CLS");
                break;
            }case 2:{
                int rowToClear, columnToClear,ch=0;
                cout << "\nWant to clear everything(1), row(2),column(3) or
nothing(0)?\n";
                do{
                    if(ch<0 or ch>3){
                        setcolor(4,0);
                        cout << "\nError! Invalid choice! Try again! \n";
                        setcolor(3,0);
                    }

                    cout << "Enter your choice: ";
                    cin >> ch;
                }while(ch<0 or ch>3);

                switch(ch){
                    case 1 :
                    case 2 :
                        cout << "\nEnter the row to clear: ";
                        cin >> rowToClear;
                        clearRow(field, rowToClear - 1, m);
                        cout << "\nField after clearing row " << rowToClear << ":" << endl;
                        writeField(field, n, m);
                        if(ch!=1)break;
                    case 3:

```

```

        cout << "\nEnter the column to clear: ";
        cin >> columnToClear;
        clearColumn(field, columnToClear - 1, n);
        cout << "\nField after clearing column " << columnToClear << ":" <<
endl;

        writeField(field, n, m);
        if(ch!=1)break;

        case 0 :
            setcolor(4,0);
            cout << "\nExit...\n"<<endl;
            setcolor(3,0);
            break;

    }

    system("pause");
    system("CLS");
    break;

}case 3:{
    int ch;
    cout << "\nWant to see all(1), row(2),column(3) or nothing(0) with the maximum number
of mined zones?\n";

        do{
            if(ch<0 or ch>3){
                setcolor(4,0);
                cout << "\nError! Invalid choice! Try again! \n";
                setcolor(3,0);
            }

            cout << "Enter your choice: ";
            cin >> ch;
        }while(ch<0 or ch>3);

        switch(ch){
        case 1 :
        case 2 :{
            cout << "\nRow with the maximum number of mined zones: ";
            getMaxMinedRow(field, n, m);
            if(ch!=1)break;
        }case 3:{
            cout << "\nColumn with the maximum number of mined zones: ";
            getMaxMinedColumn(field, n, m);
            cout << endl;
            if(ch!=1)break;

        }case 0 :
            setcolor(4,0);
            cout << "\nExit...\n\n";
            setcolor(3,0);
            break;
        }
        system("pause");
        system("CLS");
        break;
    }case 4:{
        float averageMinedOddRows = getAverageMinedOddRows(field, n, m);
        cout << "\nAverage mined zones in odd rows: " << averageMinedOddRows <<
endl;

        system("pause");
        system("CLS");
        break;
    }
}

```



```

}case 5:{
    int columns[MAX_SIZE];
    int mineCounts[MAX_SIZE];
    for (int j = 0; j < m; j++) {
        columns[j] = j + 1;
    }
    countMinesInColumns(field, n, m, mineCounts);
    sortColumnsByMines(columns, mineCounts, m);
    cout << "\nColumns in descending order of mine count:" << endl;
    for (int j = 0; j < m; j++) {
        cout << columns[j] << " ";
    }
    cout << endl;
    system("pause");
    system("CLS");
    break;
}case 6:{
    copyRowsWithMines(field, n, m);
    cout << "\nRows with mines copied to Mine.txt" << endl;
    system("pause");
    system("CLS");
    break;
}case 7:{
    int objects = countObjects(field, n, m);
    cout << "\nNumber of objects: " << objects << endl;
    system("pause");
    system("CLS");
    break;
}case 8:{
    cout << "\nShortest path from top-left to bottom-right:" << endl;
    findShortestPath(field, n, m);
cout << "\nPath: \n";
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++)
                if(field[i][j] == 4){
                    field[i][j] = 0;
                    setcolor(10,0);
                    cout << field[i][j]<< " ";
                    setcolor(3,0);
                }else cout << field[i][j] << " ";

        cout << endl;
    }

        system("pause");
        system("CLS");
        break;
}case 9:{
    readField(field, n, m);
    setcolor(10,0);
    cout << "Data from the file has been read"<<endl;
    setcolor(3,0);
    system("pause");
    system("CLS");
    break;
}case 10:{
    setcolor(4,0);
    cout << "Exit..."<<endl;
    setcolor(3,0);
    return 10;
    system("pause");
    system("CLS");
}

```

```

    }
    default:
        setcolor(4,0);
        cout << "\nInvalid choice. Please enter a valid choice.\n";
        setcolor(3,0);
        system("pause");
        system("CLS");
        break;
    }
}while(choice!=10);

return 0;
}

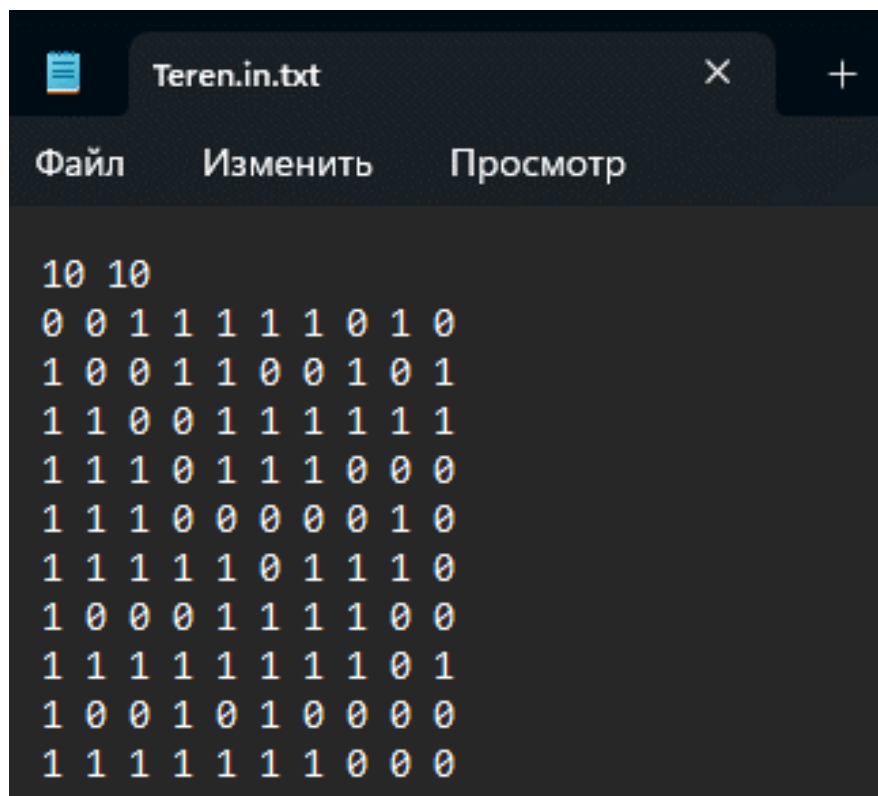
```

Вывод матрицы в главном меню

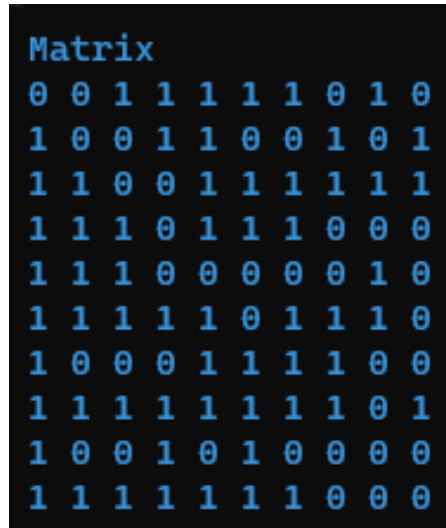
Текст задания:

Вывод матрицы в главном меню

Скриншот вводимых данных:



Скриншот результата на экране:



```
Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0
```

Описание: Функция writeField выводит содержимое двумерного массива "field" размером n x m на экран.

Фрагмент кода:

```
//функция для вывода информации
void writeField(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) cout << field[i][j] << " ";

        cout << endl;
    }
}
```

Замена первого и последнего столбца

Текст задания:

Замена первого и последнего столбца матрицы

Скриншот результата на экране:

До

После

```
Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0
```

```
Field after swapping first and last columns:
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
0 1 1 0 1 1 1 0 0 1
0 1 1 0 0 0 0 0 1 1
0 1 1 1 1 0 1 1 1 1
0 0 0 0 1 1 1 1 0 1
1 1 1 1 1 1 1 1 0 1
0 0 0 1 0 1 0 0 0 1
0 1 1 1 1 1 1 0 0 1
```

Описание: Данная функция, с именем используется для замены значений в первом и последнем столбце матрицы. Она принимает три параметра: "field" - двумерный массив, "n" - количество строк в массиве, и "m" - количество столбцов в массиве.

Функция использует цикл "for" для перебора строк матрицы. В каждой итерации цикла, она вызывает функцию "swap" для замены значений первого и последнего элемента в текущей строке матрицы. Функция "swap" меняет значения двух переменных местами.

Таким образом, данная функция заменяет значения в первом и последнем столбце матрицы.

Фрагмент кода:

```
//функция для замены 1 и последней колонки матрицы
void swapColumns(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    for (int i = 0; i < n; i++) swap(field[i][0],field[i][m-1]);
}
```

Функция для очистки строки и/или столбца

Текст задания:

Очистить строку и/или столбец.

Скриншот вводимых данных и результата на экране:

```
Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice: 2

Want to clear everything(1), row(2),column(3) or nothing(0)?
Enter your choice: 1

Enter the row to clear: 1

Field after clearing row 1:
0 0 0 0 0 0 0 0 0 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Enter the column to clear: 1

Field after clearing column 1:
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 1 0 1
0 1 0 0 1 1 1 1 1 1
0 1 1 0 1 1 1 0 0 0
0 1 1 0 0 0 0 0 1 0
0 1 1 1 1 0 1 1 1 0
0 0 0 0 1 1 1 1 0 0
0 1 1 1 1 1 1 1 0 1
0 0 0 1 0 1 0 0 0 0
0 1 1 1 1 1 1 0 0 0
```

Описание: Функция *"clearRow"* используется для очистки строки матрицы. Она принимает двумерный массив *"field"*, номер строки *"row"* и количество столбцов *"m"*. В цикле она устанавливает значение 0 для каждого элемента в указанной строке, эффективно очищая ее.

Функция *"clearColumn"* используется для очистки столбца матрицы. Она принимает двумерный массив *"field"*, номер столбца *"column"* и количество строк *"n"*. В цикле она устанавливает значение 0 для каждого элемента в указанном столбце, очищая его.

Обе функции изменяют значения внутри массива *"field"*, чтобы очистить соответствующие строки или столбцы.

Фрагмент кода:

```
//функция для очистки строки матрицы
void clearRow(int field[MAX_SIZE][MAX_SIZE], int row, int m) {
    for (int j = 0; j < m; j++)
        field[row][j] = 0;
}

//функция для очистки столбца матрицы
void clearColumn(int field[MAX_SIZE][MAX_SIZE], int column, int n) {
    for (int i = 0; i < n; i++)
        field[i][column] = 0;
}
```

Вывод строки и/или столбца с максимальным количеством мин

Текст задания:

Вывод строки и/или столбца с максимальным количеством мин

Скриншот вводимых данных:

```
Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice: 3

Want to see all(1), row(2),column(3) or nothing(0) with the maximum number of mined zones?
Enter your choice: 1
```

Скриншот результата на экране:

```
Row with the maximum number of mined zones: 8

Column with the maximum number of mined zones: 1

Exit...

Для продолжения нажмите любую клавишу . . .
```

Описание: Первая функция, "getMaxMinedRow", используется для поиска строки с максимальным количеством мин в двумерном массиве "field" размером MAX_SIZE x MAX_SIZE. Она принимает три параметра: "field" - двумерный массив, "n" - количество строк в массиве, и "m" - количество столбцов в массиве.

Функция начинает с инициализации переменных "maxMinedRow" и "maxMines" со значениями 0. Затем она использует два вложенных цикла "for" для перебора элементов массива.

Внешний цикл итерируется по строкам, а внутренний цикл - по столбцам. В каждой итерации внутреннего цикла, она проверяет, равен ли текущий элемент массива единице. Если это так, то увеличивает счетчик "mines" на единицу.

После завершения внутреннего цикла для каждой строки, она сравнивает значение "mines" с текущим максимальным количеством мин ("maxMines"). Если "mines" больше "maxMines", то обновляет значения "maxMines" и "maxMinedRow" соответствующими значениями.

Затем функция выводит номер строки с максимальным количеством мин ("maxMinedRow+1") на экран с помощью оператора "cout". После этого, она снова использует циклы "for", чтобы найти другие строки с таким же количеством мин и выводит их номера на экран.

Вторая функция, "getMaxMinedColumn" делает то же самое, но с столбцами матрицы.

Фрагмент кода:

```
//функция для поиска строки с максимальным количеством мин
void getMaxMinedRow(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    int maxMinedRow = 0;
    int maxMines = 0;
    for (int i = 0; i < n; i++) {
        int mines = 0;
        for (int j = 0; j < m; j++)
            if (field[i][j] == 1) mines++;

        if (mines > maxMines) {
            maxMines = mines;
            maxMinedRow = i;
        }
    }
    cout << maxMinedRow+1 << " ";
    for (int i = 0; i < n; i++) {
        int mines = 0;
        for (int j = 0; j < m; j++)
            if (field[i][j] == 1) mines++;

        if (mines == maxMines and i!=maxMinedRow) cout << i+1 << " ";
    }
    cout << endl;
}

//функция для поиска колонки с максимальным количеством мин
void getMaxMinedColumn(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    int maxMines = 0;
    int maxMinedColumn = 0; // Установите начальное значение для maxMinedColumn
```



```

for (int j = 0; j < m; j++) {
    int mines = 0;
    for (int i = 0; i < n; i++)
        if (field[i][j] == 1) mines++;

    if (mines > maxMines) {
        maxMines = mines;
        maxMinedColumn = j;
    }
}

cout << maxMinedColumn+1 << " ";

for (int j = 0; j < m; j++) {
    int mines = 0;
    for (int i = 0; i < n; i++)
        if (field[i][j] == 1) mines++;

    if (mines == maxMines and j!=maxMinedColumn) {
        cout << j+1 << " ";
    }
}
}

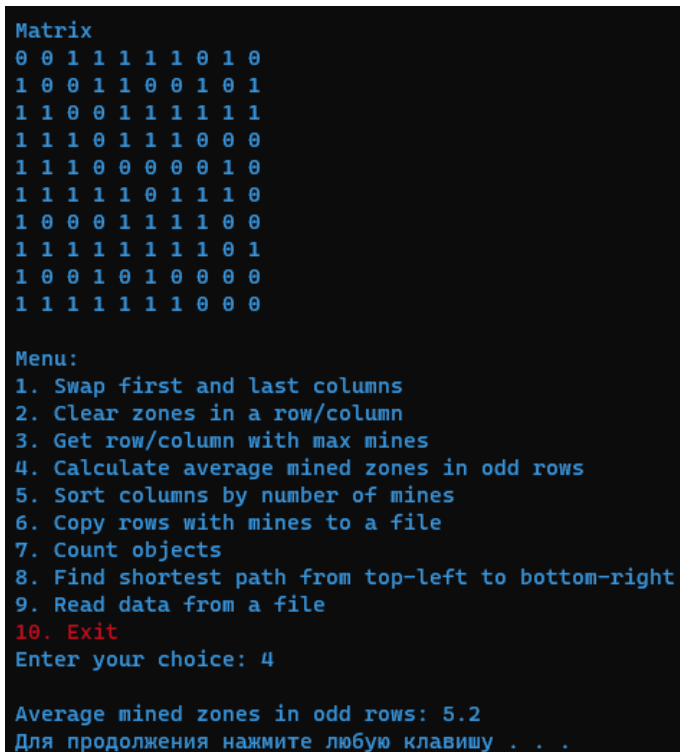
```

Вывод среднего количества мин в нечётных строках.

Текст задания:

Вывод среднего количества мин в нечётных строках.

Скриншот результата на экране:



```

Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice: 4

Average mined zones in odd rows: 5.2
Для продолжения нажмите любую клавишу . . .

```

Описание: Эта функция "используется для вычисления среднего количества мин в нечетных строках двумерного массива "field" размером MAX_SIZE x MAX_SIZE. Она принимает три параметра: "field" - двумерный массив, "n" - количество строк в массиве, и "m" - количество столбцов в массиве.

Функция начинает с инициализации переменных "minedRows" и "minedZones" со значениями 0. Затем она использует два вложенных цикла "for" для перебора элементов массива. Внешний цикл итерируется по строкам, начиная с первой нечетной строки (индекс 0), и увеличивая индекс на 2 в каждой итерации. Внутренний цикл итерируется по столбцам. В каждой итерации внутреннего цикла, она проверяет, равен ли текущий элемент массива единице. Если это так, то увеличивает счетчик "minedZones" на единицу.

После завершения внутреннего цикла для каждой нечетной строки, она увеличивает счетчик "minedRows" на единицу.

Затем функция возвращает вычисленное значение среднего количества мин в нечетных строках, которое представлено выражением: "static_cast<float>(minedZones) / minedRows". Если переменная "minedRows" больше 0, то происходит деление числа мин в нечетных строках на количество нечетных строк и возвращается полученный результат в виде значения типа float. Если "minedRows" равно 0 (отсутствуют нечетные строки), то функция возвращает 0.0.

Таким образом, эта функция вычисляет среднее количество мин в нечетных строках массива и возвращает результат в виде значения типа float.

Фрагмент кода:

```
//функция для поиска среднего количества мин в нечётных строках
float getAverageMinedOddRows(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    int minedRows = 0;
    int minedZones = 0;
    for (int i = 0; i < n; i += 2) {
        for (int j = 0; j < m; j++)
            if (field[i][j] == 1) minedZones++;

        minedRows++;
    }
    return (minedRows > 0) ? static_cast<float>(minedZones) / minedRows : 0.0;
}
```

Вывод колонок с минами в убывающем порядке.

Текст задания:

Вывод колонок с минами в убывающем порядке

Скриншот вводимых данных:

```
Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice: 5
```

Скриншот результата на экране:

```
Columns in descending order of mine count:
1 5 6 7 2 3 4 8 9 10
Для продолжения нажмите любую клавишу . . .
```

Описание: Первая функция, "countMinesInColumns", используется для подсчета количества мин в каждом столбце двумерного массива "field" размером MAX_SIZE x MAX_SIZE. Она принимает четыре параметра: "field" - двумерный массив, "n" - количество строк в массиве, "m" - количество столбцов в массиве и "mineCounts" - массив, в котором будут сохранены результаты подсчета мин.

Функция использует вложенные циклы "for" для перебора элементов массива. Внешний цикл итерируется по столбцам, а внутренний цикл - по строкам. В каждой итерации внутреннего

цикла, она проверяет, равен ли текущий элемент массива единице. Если это так, то увеличивает счетчик "mines" на единицу.

После завершения внутреннего цикла для каждого столбца, она сохраняет значение счетчика "mines" в соответствующий элемент массива "mineCounts".

Вторая функция, "sortColumnsByMines", используется для сортировки столбцов в убывающем порядке на основе количества мин, сохраненного в массиве "mineCounts". Она принимает три параметра: "columns" - массив, содержащий номера столбцов, "mineCounts" - массив с количеством мин в каждом столбце и "m" - количество столбцов.

Функция использует два вложенных цикла "for" для сравнения количества мин в каждом столбце. Внешний цикл итерируется по индексам столбцов, а внутренний цикл - по следующим столбцам. В каждой итерации внутреннего цикла, она сравнивает количество мин в текущем столбце с количеством мин в следующем столбце. Если количество мин в следующем столбце больше, чем в текущем, то она меняет местами значения количества мин и номера столбцов, используя функцию "swap".

Таким образом, эти две функции работают вместе для подсчета количества мин в каждом столбце и сортировки столбцов в убывающем порядке на основе этого подсчета.

Фрагмент кода:

```
//функция для поиска количества мин в колонках
void countMinesInColumns(int field[MAX_SIZE][MAX_SIZE], int n, int m, int
mineCounts[MAX_SIZE]) {
    for (int j = 0; j < m; j++) {
        int mines = 0;
        for (int i = 0; i < n; i++)
            if (field[i][j] == 1) mines++;

        mineCounts[j] = mines;
    }
}

//функция для сортировки колонок в убывающем порядке
void sortColumnsByMines(int columns[MAX_SIZE], int mineCounts[MAX_SIZE], int m)
{
    for (int i = 0; i < m; i++)
        for (int j = i + 1; j < m; j++)
            if (mineCounts[j] > mineCounts[i]) {
                swap(mineCounts[i], mineCounts[j]);
                swap(columns[i], columns[j]);
            }
}
```

Копирование строк с минами в отдельный файл

Текст задания:

Копирование строк с минами в отдельный файл

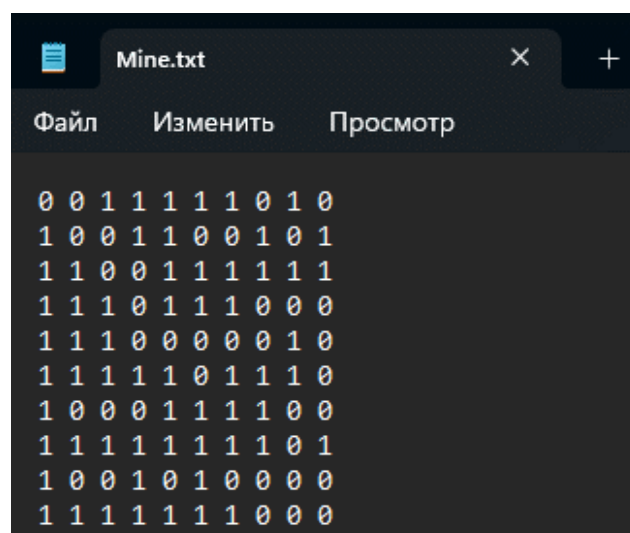
Скриншот вводимых данных:

```
Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice: 6

Rows with mines copied to Mine.txt
Для продолжения нажмите любую клавишу . . .
```

Скриншот результата в файле:



```
Mine.txt
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0
```

Описание: Эта функция используется для копирования строк, содержащих мины, из двумерного массива "field" размером MAX_SIZE x MAX_SIZE, в отдельный файл с именем "Mine.txt". Она принимает три параметра: "field" - двумерный массив, "n" - количество строк в массиве, и "m" - количество столбцов в массиве.

Функция начинает с открытия выходного файла "Mine.txt" с помощью объекта ofstream. Если файл успешно открыт, то выполняется следующий код.

В цикле "for" она перебирает каждую строку массива "field". Для каждой строки, она устанавливает флаг "hasMines" в значение false. Затем, с помощью вложенного цикла "for", она проверяет каждый элемент в строке и если находит мину (значение 1), устанавливает флаг "hasMines" в значение true и выходит из цикла.

Если флаг "hasMines" имеет значение true, значит текущая строка содержит мину, и она записывает содержимое строки в файл "Mine.txt". Для этого она использует вложенный цикл "for", чтобы перебрать каждый элемент в строке и записать его значение в файл с помощью оператора "outputFile << field[i][j] << " ". После записи всех элементов строки, она добавляет символ новой строки с помощью оператора "outputFile << endl;".

По завершении цикла "for" для всех строк массива, она закрывает файл с помощью оператора "outputFile.close()".

Если файл не удалось открыть, она выводит сообщение "Unable to open file Mine.txt" с помощью оператора "cout".

Таким образом, эта функция копирует строки, содержащие мины, из двумерного массива в отдельный файл "Mine.txt".

Фрагмент кода:

```
//функция для копирования строк с минами в отдельный файл
void copyRowsWithMines(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    ofstream outputFile("Mine.txt");
    if (outputFile.is_open()) {
        for (int i = 0; i < n; i++) {
            bool hasMines = false;
            for (int j = 0; j < m; j++)
                if (field[i][j] == 1) {
                    hasMines = true;
                }
        }
    }
}
```

```

        break;
    }

    if (hasMines) {
        for (int j = 0; j < m; j++)
            outputFile << field[i][j] << " ";

        outputFile << endl;
    }
}
outputFile.close();
} else {
    cout << "Unable to open file Mine.txt";
}
}

```

Функция для определения количества объектов в бинарной матрице T

Текст задания:

Определить количество объектов в бинарной матрице T

Скриншот вводимых данных:

```

Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice: 7

```

Скриншот результата на экране:

```

Number of objects: 24
Для продолжения нажмите любую клавишу . . .

```

Описание: Эта функция используется для подсчета объектов в двумерном массиве "field" размером MAX_SIZE x MAX_SIZE. Она принимает три параметра: "field" - двумерный массив, "n" - количество строк в массиве, и "m" - количество столбцов в массиве.

Функция начинает с инициализации переменных. Она устанавливает значение "objects" (количество объектов) в 0 и создает массив "visited" размером MAX_SIZE x MAX_SIZE, заполняя его значениями false. Массив "visited" используется для отслеживания посещенных элементов массива "field".

Затем, с помощью вложенных циклов "for", она перебирает каждый элемент в массиве "field". Если текущий элемент равен 1 и не был посещен ранее (соответствующий элемент в массиве "visited" равен false), то это означает наличие нового объекта.

Она увеличивает счетчик "objects" на 1 и устанавливает значение "visited[i][j]" в true, чтобы отметить текущий элемент как посещенный.

Затем, с помощью массивов "dx" и "dy", определены смещения для проверки соседних элементов. Внутренний цикл "for" с индексом "k" перебирает все возможные смещения. Для каждого смещения, она вычисляет новые индексы "ni" и "nj" для соседнего элемента. Затем, она проверяет, что новые индексы находятся в пределах массива и соответствующий элемент равен 1, и что элемент еще не был посещен. Если эти условия выполняются, она устанавливает "visited[ni][nj]" в true, чтобы отметить соседний элемент как посещенный.

По завершении перебора всех элементов в массиве "field", она возвращает значение переменной "objects" - количество найденных объектов.

Таким образом, эта функция выполняет подсчет объектов в двумерном массиве на основе связности элементов, используя алгоритм обхода в ширину.

Фрагмент кода:

```
//функция для подсчёта объектов в матрицы
int countObjects(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    int objects = 0;
    bool visited[MAX_SIZE][MAX_SIZE] = { false };

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (field[i][j] == 1 && !visited[i][j]) {
```



```

        objects++;
        visited[i][j] = true;

        int dx[] = { -1, 1, 0, 0, -1, -1, 1, 1 };
        int dy[] = { 0, 0, -1, 1, -1, 1, -1, 1 };
        for (int k = 0; k < 8; k++) {
            int ni = i + dx[k];
            int nj = j + dy[k];
            if (ni >= 0 && ni < n && nj >= 0 && nj < m && field[ni][nj]
== 1 && !visited[ni][nj])
                visited[ni][nj] = true;
        }
    }
    return objects;
}

```

Функция для поиска кратчайшего пути с левого верхнего до правого нижнего угла.

Текст задания:

Найти кратчайший путь с левого верхнего до правого нижнего угла.

Скриншот вводимых данных:

```

Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice: 8

```

Скриншот результата на экране:

```
Shortest path from top-left to bottom-right:
[1, 1] - [1, 2] - [2, 2] - [2, 3] - [3, 3] - [3, 4] - [4, 4] - [5, 4] - [5, 5] - [5, 6] - [5, 7] - [5, 8] -
[4, 8] - [4, 9] - [4, 10] - [5, 10] - [6, 10] - [7, 10] - [7, 9] - [8, 9] - [9, 9] - [9, 10] - [10, 10]

Path:
0 0 1 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Для продолжения нажмите любую клавишу . . .
```

Описание: Эта функция используется для поиска кратчайшего пути в матрице от начальной точки (верхний левый угол) до конечной точки (нижний правый угол).

Сначала она создает структуру "Cell" с полями "row" (строка) и "col" (столбец), которая представляет ячейку в матрице.

Затем, она создает двумерный массив "visited" размером MAX_SIZE x MAX_SIZE для отслеживания посещенных ячеек. И массив "parent" того же размера для сохранения родительских ячеек, чтобы восстановить кратчайший путь.

Далее, она создает очередь "q" для обхода ячеек в ширину. Задает начальную точку "start" (верхний левый угол) и конечную точку "end" (нижний правый угол).

Затем, она отмечает начальную точку как посещенную, помещает ее в очередь "q" и устанавливает флаг "pathFound" в false.

В цикле while она извлекает ячейку из очереди "q" и проверяет, является ли она конечной точкой. Если текущая ячейка равна конечной точке, то устанавливает флаг "pathFound" в true и выходит из цикла.

Затем, с помощью массивов "dx" и "dy", определены смещения для проверки соседних ячеек вверх, вниз, влево и вправо. Внутренний цикл "for" с индексом "k" перебирает все возможные смещения. Для каждого смещения, она вычисляет новые индексы "nx" и "ny" для соседней

ячейки. Затем, она проверяет, что новые индексы находятся в пределах матрицы, соответствующая ячейка равна 0 и ячейка еще не была посещена. Если эти условия выполняются, она отмечает соседнюю ячейку как посещенную, сохраняет текущую ячейку как родительскую для соседней ячейки, и помещает соседнюю ячейку в очередь "q" для дальнейшего обхода.

По завершении цикла *while*, она проверяет значение флага "pathFound". Если путь найден, то она восстанавливает кратчайший путь, начиная от конечной точки до начальной точки, используя массив "parent". Она также помечает посещенные ячейки пути значением 4 в массиве "field". Если путь не найден, то выводит сообщение "No path found".

Таким образом, эта функция выполняет поиск кратчайшего пути в матрице с помощью алгоритма обхода в ширину (BFS) и сохраняет путь с помощью массива "parent".

Фрагмент кода:

```
// создание структуры для поиска кратчайшего пути в матрице
struct Cell {
    int row;
    int col;
};

// функция для поиска кратчайшего пути от начальной точки к конечной
void findShortestPath(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    bool visited[MAX_SIZE][MAX_SIZE] = { false };
    Cell parent[MAX_SIZE][MAX_SIZE];

    queue<Cell> q;
    Cell start = { n - 1, m - 1 };
    Cell end = { 0, 0 };

    visited[start.row][start.col] = true;
    q.push(start);

    bool pathFound = false;

    while (!q.empty()) {
        Cell current = q.front();
        q.pop();

        int cx = current.row;
        int cy = current.col;

        if (cx == end.row && cy == end.col) {
            pathFound = true;
            break;
        }
    }
```

```

int dx[] = { -1, 1, 0, 0 };
int dy[] = { 0, 0, -1, 1 };

for (int k = 0; k < 4; k++) {
    int nx = cx + dx[k];
    int ny = cy + dy[k];

    if (nx >= 0 && nx < n && ny >= 0 && ny < m && field[nx][ny] == 0 &&
!visited[nx][ny]) {
        visited[nx][ny] = true;
        parent[nx][ny] = { cx, cy };
        q.push({ nx, ny });
    }
}

if (pathFound) {
    Cell current = end;

    Cell showPath[MAX_SIZE];
    int count = 0;
    while (current.row != start.row || current.col != start.col) {
        cout << "[" << current.row + 1 << ", " << current.col + 1 << "]" -
";

        field[current.row][current.col] = 4;
        current = parent[current.row][current.col];
    }
    cout << "[" << start.row + 1 << ", " << start.col + 1 << "]" << endl;
    field[start.row][start.col] = 4;
} else {
    cout << "No path found." << endl;
}
}

```

Функция для повторного считывания матрицы из файла.

Текст задания:

Считать данные из файла в программу.

Скриншот вводимых данных:

```
Matrix
0 0 1 1 1 1 1 0 1 0
1 0 0 1 1 0 0 1 0 1
1 1 0 0 1 1 1 1 1 1
1 1 1 0 1 1 1 0 0 0
1 1 1 0 0 0 0 0 1 0
1 1 1 1 1 0 1 1 1 0
1 0 0 0 1 1 1 1 0 0
1 1 1 1 1 1 1 1 0 1
1 0 0 1 0 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0

Menu:
1. Swap first and last columns
2. Clear zones in a row/column
3. Get row/column with max mines
4. Calculate average mined zones in odd rows
5. Sort columns by number of mines
6. Copy rows with mines to a file
7. Count objects
8. Find shortest path from top-left to bottom-right
9. Read data from a file
10. Exit
Enter your choice: 9
```

Скриншот результата на экране:

```
Data from the file has been read
Для продолжения нажмите любую клавишу . . .
```

Описание: Эта функция используется для чтения информации из файла "Teren.in.txt" и сохранения данных в двумерный массив "field". Она также принимает ссылки на переменные "n" и "m", которые будут использоваться для хранения размеров матрицы.

Сначала она пытается открыть файл "Teren.in.txt" для чтения с помощью объекта "inputFile". Если файл успешно открыт и не достигнут конец файла, она считывает значения "n" и "m" из файла, указывающие размеры матрицы.

Затем, с помощью двух вложенных циклов "for", она считывает элементы матрицы из файла и сохраняет их в соответствующие ячейки массива "field".

По завершении чтения из файла, она закрывает файл "inputFile" и возвращает значение true, указывая успешное выполнение операции чтения.

Если файл не может быть открыт или достигнут конец файла, она выводит сообщение об ошибке и возвращает значение false, указывая на неудачное выполнение операции чтения.

Таким образом, эта функция обеспечивает чтение данных матрицы из файла и их сохранение в массиве для дальнейшей обработки.

Фрагмент кода:

```
//функция для чтения информации из файла
bool readField(int field[MAX_SIZE][MAX_SIZE], int& n, int& m) {
    ifstream inputFile("Teren.in.txt");
    if (inputFile.is_open() && !inputFile.eof()) {
        inputFile >> n >> m;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                inputFile >> field[i][j];

        inputFile.close();
    } else {
        cout << "Unable to open file Teren.in.txt";
        return false;
    }
    return true;
}
```

Вспомогательная функция

Смена цвета

Описание: данная функция нужна для того, чтобы менять цвет текста и заднего фона. Благодаря этой функции, не нужно писать длинную строку для смены цвета.

Фрагмент кода

```
void setcolor(int text, int backG=0){ // функция для изменения цвета
HANDLE color = GetStdHandle(STD_OUTPUT_HANDLE); // дескриптор устройства
стандартного вывода.
SetConsoleTextAttribute(color,(WORD)((backG << 4)| text)); // задаем цвет
фона и цвет текста
}
```

Задание 2 (Java)

Условие(вариант 13)

Напишите программу на языке Java, которая моделирует следующую деятельность:

Создайте программу для учета деятельности коммерческого банка по предоставлению кредитов.

- 1) Коммерческий банк имеет несколько филиалов в стране. Каждый филиал характеризуется именем, адресом, телефоном и т. д.
- 2) Одной из деятельности коммерческого банка является предоставление кредитов клиентам. Клиентами могут быть как физические лица, так и юридические лица. Физические лица характеризуются именем, личным кодом, адресом, телефоном. Юридические лица характеризуются налоговым идентификационным номером, наименованием, типом собственности, адресом, телефоном, именем администратора, контактным лицом и т. д.
- 3) Каждый кредит определяется типом кредита, наименованием, валютой и годовой процентной ставкой. Для каждого предоставленного кредита заключается договор, который содержит его номер, дату, клиента, тип кредита, общую сумму кредита и срок полного погашения суммы.
- 4) Программа должна обеспечивать учет этих кредитов, расчет дохода банка, который должен быть конвертирован и представлен в местной валюте и т. д. За каждый заключенный договор сотрудник банка получает определенный процент от общей суммы предоставленного кредита.

Требования к заданию: Реализуйте концепции ООП

- 1) Создайте классы с методами (функциями) для чтения и вывода каждого поля (сеттеры и геттеры), а также метод, который вычисляет новое значение одного или нескольких полей (например, конвертирует цену из лей в евро или рассчитывает продолжительность поездки). Добавьте как минимум три вида конструкторов (один без параметров, второй со всеми указанными параметрами, третий с несколькими указанными параметрами, в зависимости от темы, в последнем случае некоторые поля могут иметь одинаковые значения или некоторые стандартные значения).

- 2) Классы, наследующие поля и методы родительского класса, будут иметь свои собственные свойства, создавая соответствующие конструкторы (соответствующие конструкторы базового класса). Используйте отношения ассоциации и агрегации между объектами (где это необходимо).
- 3) Создайте абстрактный класс и интерфейс, содержащий хотя бы один абстрактный метод, а затем унаследуйте их в процессе наследования и полностью реализуйте. Каждый класс должен содержать перегруженные конструкторы и переопределенный метод `toString()`.
- 4) Обработайте различные типы исключений (например, при делении на ноль, извлечении корня из отрицательного числа, попытке доступа к несуществующему элементу массива, вводе пользователем букв вместо чисел, вызове метода объекта с нулевой ссылкой и т. д.). Создайте собственное исключение.
- 5) Запишите объекты в список, не менее 20 записей. Используйте список для вывода на экран.
- 6) Программа должна содержать минимальное меню.
- 7) Если необходимо, добавьте классы, члены или методы к существующим классам.

Интерфейс BankSystem

```
/**
 * Интерфейс BankSystem
 * Данный интерфейс определяет набор методов, которые должны быть
 реализованы в классе, который использует этот интерфейс.
 * Он служит для описания функциональности банковской системы, а
 именно для управления филиалами банка, клиентами, кредитами,
 контрактами и другими операциями.
 * Вот краткое описание каждого метода в интерфейсе:
 * 1.    displayBankBranches(): Этот метод используется для
 отображения информации о всех филиалах банка.
 * 2.    displayClients(List<IndividualClient> individualClients,
 List<CorporateClient> corporateClients): Этот метод используется
 для отображения информации о клиентах банка. Он принимает два
 списка клиентов: individualClients (список физических лиц) и
 corporateClients (список юридических лиц).
 * 3.    displayCredits(): Этот метод используется для отображения
 информации о кредитах, доступных в банковской системе.
 * 4.    addBankBranch(): Этот метод используется для добавления
 нового филиала банка в систему.
 * 5.    addClient(): Этот метод используется для добавления нового
 клиента в систему. Новый клиент может быть как физическим лицом
 (IndividualClient), так и юридическим лицом (CorporateClient).
 * 6.    addCredit(): Этот метод используется для добавления нового
 кредита в систему.
 * 7.    createContract(): Этот метод используется для создания
 нового контракта в системе.
 * 8.    calculateBankIncome(): Этот метод используется для расчета
 дохода банка.
 * 9.    displayContracts(): Этот метод используется для
 отображения информации о контрактах, заключенных в банковской
 системе.
 * 10.   convertPriceToEuro(): Этот метод используется для
 конвертации цены в евро.
 * Интерфейс предоставляет абстракцию для операций, которые могут
 быть выполнены в банковской системе.
 * Реализация этих методов будет зависеть от конкретного класса,
 который реализует данный интерфейс и предоставляет конкретную
 функциональность для работы с банковской системой.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
import java.util.List;

public interface BankSystem {
    void displayBankBranches();
    void displayClients(List<IndividualClient> individualClients,
 List<CorporateClient> corporateClients);
    void displayCredits();
    void addBankBranch();
    void addClient();
}
```

```
void addCredit();  
void createContract();  
void calculateBankIncome();  
void displayContracts();  
void convertPriceToEuro();  
}
```

Описание: Данный интерфейс, названный "BankSystem" (Банковская система), определяет набор методов, которые должны быть реализованы в классе, который использует этот интерфейс. Он служит для описания функциональности банковской системы, а именно для управления филиалами банка, клиентами, кредитами, контрактами и другими операциями.

Вот краткое описание каждого метода в интерфейсе:

1. `displayBankBranches()`: Этот метод используется для отображения информации о всех филиалах банка.
2. `displayClients(List<IndividualClient> individualClients, List<CorporateClient> corporateClients)`: Этот метод используется для отображения информации о клиентах банка. Он принимает два списка клиентов: `individualClients` (список физических лиц) и `corporateClients` (список юридических лиц).
3. `displayCredits()`: Этот метод используется для отображения информации о кредитах, доступных в банковской системе.
4. `addBankBranch()`: Этот метод используется для добавления нового филиала банка в систему.
5. `addClient()`: Этот метод используется для добавления нового клиента в систему. Новый клиент может быть как физическим лицом (`IndividualClient`), так и юридическим лицом (`CorporateClient`).
6. `addCredit()`: Этот метод используется для добавления нового кредита в систему.
7. `createContract()`: Этот метод используется для создания нового контракта в системе.
8. `calculateBankIncome()`: Этот метод используется для расчета дохода банка.
9. `displayContracts()`: Этот метод используется для отображения информации о контрактах, заключенных в банковской системе.
10. `convertPriceToEuro()`: Этот метод используется для конвертации цены в евро.

Интерфейс предоставляет абстракцию для операций, которые могут быть выполнены в банковской системе. Реализация этих методов будет зависеть от конкретного класса, который реализует данный интерфейс и предоставляет конкретную функциональность для работы с банковской системой.

Класс BankBranch

```
/**
 * Базовый класс "Филиал банка"
 * Этот класс служит в качестве базового класса для представления
 * филиалов банка и содержит некоторые свойства и методы для работы с
 * ними.
 * 1.    Приватные переменные:
 *    name: хранит имя филиала банка.
 *    address: хранит адрес филиала банка.
 *    phoneNumber: хранит номер телефона филиала банка.
 * 2.    Конструктор BankBranch инициализирует объект класса
 * BankBranch с заданными именем, адресом и номером телефона филиала.
 * 3.    Геттеры и сеттеры для свойств name, address и phoneNumber:
 * позволяют получать значения и устанавливать новые значения для
 * соответствующих свойств филиала банка.
 * 4.    Переопределенный метод toString(): возвращает строковое
 * представление филиала банка, содержащее его имя, адрес и номер
 * телефона.
 * Этот класс предоставляет базовый функционал для представления
 * филиала банка.
 * Можно использовать его как основу для создания объектов
 * филиалов и управления их свойствами.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public class BankBranch {
    private String name;
    private String address;
    private String phoneNumber;

    public BankBranch(String name, String address, String
phoneNumber) {
        this.name = name;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }

    public String getName() {return name;}

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}
```

```

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber)
    {this.phoneNumber = phoneNumber;}
    @Override
    public String toString() {
        return "Филиал банка: " +
            "Имя = " + name +
            ", Адрес = " + address +
            ", Телефон = " + phoneNumber;
    }
}

```

Описание: Данный код представляет класс `BankBranch` (Филиал банка). Этот класс служит в качестве базового класса для представления филиалов банка и содержит некоторые свойства и методы для работы с ними. Вот краткое описание каждого элемента класса:

1. Приватные переменные:
 - o `name`: хранит имя филиала банка.
 - o `address`: хранит адрес филиала банка.
 - o `phoneNumber`: хранит номер телефона филиала банка.
2. Конструктор `BankBranch(String name, String address, String phoneNumber)`: инициализирует объект класса `BankBranch` с заданными именем, адресом и номером телефона филиала.
3. Геттеры и сеттеры для свойств `name`, `address` и `phoneNumber`: позволяют получать значения и устанавливать новые значения для соответствующих свойств филиала банка.
4. Переопределенный метод `toString()`: возвращает строковое представление филиала банка, содержащее его имя, адрес и номер телефона.

Этот класс предоставляет базовый функционал для представления филиала банка. Можно использовать его как основу для создания объектов филиалов и управления их свойствами.

Абстрактный класс Client

```
/**
 * Данный код представляет абстрактный класс Client (Клиент).
 * Этот класс служит базовым классом для представления клиентов
 банка и содержит общие свойства и методы для работы с клиентами.
 * Краткое описание каждого элемента класса:
 * 1. Приватные переменные:
 *   name: хранит имя клиента.
 *   address: хранит адрес клиента.
 *   phoneNumber: хранит номер телефона клиента.
 * 2. Конструктор Client инициализирует объект класса Client с
 заданными именем, адресом и номером телефона клиента.
 * 3. Геттеры и сеттеры для свойств name, address и phoneNumber:
 позволяют получать значения и устанавливать новые значения для
 соответствующих свойств клиента.
 * 4. Абстрактный метод calculateCreditLimit(): предоставляет
 абстрактную реализацию для расчета кредитного лимита клиента. Этот
 метод должен быть реализован в подклассах, наследующих абстрактный
 класс Client.
 * 5. Переопределенный метод toString(): возвращает строковое
 представление клиента, содержащее его имя, адрес и номер телефона.
 * Этот абстрактный класс предоставляет базовый функционал для
 представления клиентов банка.
 * Он может быть использован в качестве основы для создания
 подклассов, представляющих конкретные типы клиентов
 * (например, физические лица и юридические лица), и расширения
 функциональности клиентов в этих подклассах.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public abstract class Client {
    private String name;
    private String address;
    private String phoneNumber;

    public Client(String name, String address, String phoneNumber)
    {
        this.name = name;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }
}
```

```

    public void setAddress(String address) {
        this.address = address;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    @Override
    public String toString() {
        return "Клиент: " +
            "Имя = " + name +
            ", Адрес = " + address +
            ", Телефон = " + phoneNumber;
    }
}

```

Описание: Данный код представляет абстрактный класс `Client` (Клиент). Этот класс служит базовым классом для представления клиентов банка и содержит общие свойства и методы для работы с клиентами. Вот краткое описание каждого элемента класса:

1. Приватные переменные:
 - o `name`: хранит имя клиента.
 - o `address`: хранит адрес клиента.
 - o `phoneNumber`: хранит номер телефона клиента.
2. Конструктор `Client(String name, String address, String phoneNumber)`: инициализирует объект класса `Client` с заданными именем, адресом и номером телефона клиента.
3. Геттеры и сеттеры для свойств `name`, `address` и `phoneNumber`: позволяют получать значения и устанавливать новые значения для соответствующих свойств клиента.
4. Абстрактный метод `calculateCreditLimit()`: предоставляет абстрактную реализацию для расчета кредитного лимита клиента. Этот метод должен быть реализован в подклассах, наследующих абстрактный класс `Client`.
5. Переопределенный метод `toString()`: возвращает строковое представление клиента, содержащее его имя, адрес и номер телефона.

Этот абстрактный класс предоставляет базовый функционал для представления клиентов банка. Он может быть использован в качестве основы для создания подклассов, представляющих конкретные типы клиентов (например, физические лица и юридические лица), и расширения функциональности клиентов в этих подклассах.

Класс IndividualClient

```
/**
 * Данный код представляет класс IndividualClient (физическое
 * лицо), который является подклассом класса Client.
 * Этот класс расширяет функциональность базового класса Client,
 * добавляя специфичные свойства и методы для представления
 * физических лиц в банковской системе. Вот краткое описание каждого
 * элемента класса:
 * 1. Приватная переменная:
 *   personalCode: хранит личный код физического лица.
 * 2. Конструктор IndividualClient инициализирует объект класса
 * IndividualClient с заданными именем, адресом, номером телефона и
 * личным кодом физического лица. Конструктор также вызывает
 * конструктор базового класса Client с помощью ключевого слова
 * super, чтобы инициализировать общие свойства клиента.
 * 3. Геттер и сеттер для свойства personalCode: позволяют
 * получать значение и устанавливать новое значение личного кода
 * физического лица.
 * 4. Переопределенный метод toString(): возвращает строковое
 * представление физического лица, содержащее его имя, адрес, номер
 * телефона и личный код.
 * Этот класс предоставляет конкретную реализацию для
 * представления физического лица в банковской системе.
 * Он наследует общий функционал от базового класса Client и
 * добавляет специфичные свойства и методы для физических лиц.
 * Это позволяет использовать объекты класса IndividualClient для
 * представления и управления информацией о физических лицах в
 * банковской системе.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public class IndividualClient extends Client {
    private String personalCode;

    public IndividualClient(String name, String address, String
phoneNumber, String personalCode) {
        super(name, address, phoneNumber);
        this.personalCode = personalCode;
    }

    public String getPersonalCode() {
        return personalCode;
    }

    public void setPersonalCode(String personalCode) {
        this.personalCode = personalCode;
    }

    @Override
    public String toString() {
```

```
        return "Имя: " + getName() + "\n" +  
               "Адрес: " + getAddress() + "\n" +  
               "Телефон: " + getPhoneNumber() + "\n" +  
               "Личный код: " + personalCode;  
    }  
}
```

Описание: Данный код представляет класс `IndividualClient` (Физическое лицо), который является подклассом класса `Client`. Этот класс расширяет функциональность базового класса `Client`, добавляя специфичные свойства и методы для представления физических лиц в банковской системе. Вот краткое описание каждого элемента класса:

1. Приватная переменная:
 - o `personalCode`: хранит личный код физического лица.
2. Конструктор `IndividualClient(String name, String address, String phoneNumber, String personalCode)`: инициализирует объект класса `IndividualClient` с заданными именем, адресом, номером телефона и личным кодом физического лица. Конструктор также вызывает конструктор базового класса `Client` с помощью ключевого слова `super`, чтобы инициализировать общие свойства клиента.
3. Геттер и сеттер для свойства `personalCode`: позволяют получать значение и устанавливать новое значение личного кода физического лица.
4. Переопределенный метод `toString()`: возвращает строковое представление физического лица, содержащее его имя, адрес, номер телефона и личный код.

Этот класс предоставляет конкретную реализацию для представления физического лица в банковской системе. Он наследует общий функционал от базового класса `Client` и добавляет специфичные свойства и методы для физических лиц. Это позволяет использовать объекты класса `IndividualClient` для представления и управления информацией о физических лицах в банковской системе.

Класс CorporateClient

```
/**
 * Данный код представляет класс CorporateClient (Юридическое
 * лицо), который также является подклассом класса Client.
 * Этот класс расширяет функциональность базового класса Client,
 * добавляя специфичные свойства и методы для представления
 * юридических лиц в банковской системе. Вот краткое описание каждого
 * элемента класса:
 * 1. Приватные переменные:
 *   taxIdentificationNumber: хранит налоговый идентификационный
 *   номер юридического лица.
 *   administratorName: хранит имя администратора юридического
 *   лица.
 *   contactPerson: хранит имя контактного лица юридического лица.
 * 2. Конструктор CorporateClient инициализирует объект класса
 * CorporateClient с заданными именем, адресом, номером телефона,
 * налоговым идентификационным номером, именем администратора и
 * контактным лицом юридического лица.
 * Конструктор также вызывает конструктор базового класса Client с
 * помощью ключевого слова super, чтобы инициализировать общие
 * свойства клиента.
 * 3. Геттеры и сеттеры позволяют получать значения и
 * устанавливать новые значения для соответствующих свойств
 * юридического лица.
 * 4. Переопределенный метод toString(): возвращает строковое
 * представление юридического лица.
 * Этот класс предоставляет конкретную реализацию для
 * представления юридического лица в банковской системе.
 * Он наследует общий функционал от базового класса Client и
 * добавляет специфичные свойства и методы для юридических лиц.
 * Это позволяет использовать объекты класса CorporateClient для
 * представления и управления информацией о юридических лицах в
 * банковской системе.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public class CorporateClient extends Client {
    private String taxIdentificationNumber;
    private String administratorName;
    private String contactPerson;

    public CorporateClient(String name, String address, String
    phoneNumber, String taxIdentificationNumber, String
    administratorName, String contactPerson) {
        super(name, address, phoneNumber);
        this.taxIdentificationNumber = taxIdentificationNumber;
        this.administratorName = administratorName;
        this.contactPerson = contactPerson;
    }
}
```

```

    public String getTaxIdentificationNumber() {
        return taxIdentificationNumber;
    }

    public void setTaxIdentificationNumber(String
taxIdentificationNumber) {this.taxIdentificationNumber =
taxIdentificationNumber;}

    public String getAdministratorName() {
        return administratorName;
    }

    public void setAdministratorName(String administratorName) {
        this.administratorName = administratorName;
    }

    public String getContactPerson() {return contactPerson;}

    public void setContactPerson(String contactPerson) {
        this.contactPerson = contactPerson;
    }

    @Override
    public String toString() {
        return "Наименование: " + getName() + "\n" +
            "Адрес: " + getAddress() + "\n" +
            "Телефон: " + getPhoneNumber() + "\n" +
            "Налоговый идентификационный номер: " +
taxIdentificationNumber + "\n" +
            "Имя администратора: " + administratorName + "\n"
+
            "Контактное лицо: " + contactPerson;
    }
}

```

Описание: Данный код представляет класс `CorporateClient` (Юридическое лицо), который также является подклассом класса `Client`. Этот класс расширяет функциональность базового класса `Client`, добавляя специфичные свойства и методы для представления юридических лиц в банковской системе. Вот краткое описание каждого элемента класса:

1. Приватные переменные:

- o `taxIdentificationNumber`: хранит налоговый идентификационный номер юридического лица.
- o `administratorName`: хранит имя администратора юридического лица.
- o `contactPerson`: хранит имя контактного лица юридического лица.

2. Конструктор `CorporateClient(String name, String address, String phoneNumber, String taxIdentificationNumber, String administratorName, String contactPerson)`: инициализирует объект

класса `CorporateClient` с заданными именем, адресом, номером телефона, налоговым идентификационным номером, именем администратора и контактным лицом юридического лица. Конструктор также вызывает конструктор базового класса `Client` с помощью ключевого слова `super`, чтобы инициализировать общие свойства клиента.

3. Геттеры и сеттеры для свойств `taxIdentificationNumber`, `administratorName` и `contactPerson`: позволяют получать значения и устанавливать новые значения для соответствующих свойств юридического лица.
4. Переопределенный метод `toString()`: возвращает строковое представление юридического лица, содержащее его наименование, адрес, номер телефона, налоговый идентификационный номер, имя администратора и контактное лицо.

Этот класс предоставляет конкретную реализацию для представления юридического лица в банковской системе. Он наследует общий функционал от базового класса `Client` и добавляет специфичные свойства и методы для юридических лиц. Это позволяет использовать объекты класса `CorporateClient` для представления и управления информацией о юридических лицах в банковской системе.

Класс Credit

```
/**
 * Данный код представляет класс Credit (Кредит), который
 * предназначен для представления информации о кредите в банковской
 * системе.
 * Краткое описание каждого элемента класса:
 * 1. Приватные переменные:
 *   creditType: хранит тип кредита.
 *   name: хранит наименование кредита.
 *   currency: хранит валюту кредита.
 *   annualInterestRate: хранит годовую процентную ставку по
 *   кредиту.
 * 2. Конструктор Credit инициализирует объект класса Credit с
 *   заданными типом кредита, наименованием, валютой и годовой
 *   процентной ставкой.
 *   Конструктор принимает эти значения и устанавливает их в
 *   соответствующие приватные переменные класса.
 * 3. Геттеры и сеттеры позволяют получать значения и
 *   устанавливать новые значения для соответствующих свойств кредита.
 * 4. Переопределенный метод toString(): возвращает строковое
 *   представление кредита, содержащее его тип, наименование, валюту и
 *   годовую процентную ставку.
 * Этот класс позволяет представлять информацию о кредите в
 * банковской системе.
 * Объекты класса Credit могут использоваться для хранения и
 * управления данными о кредитах, таких как тип, наименование, валюта
 * и процентная ставка.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public class Credit {
    private String creditType;
    private String name;
    private String currency;
    private double annualInterestRate;

    public Credit(String creditType, String name, String currency,
double annualInterestRate) {
        this.creditType = creditType;
        this.name = name;
        this.currency = currency;
        this.annualInterestRate = annualInterestRate;
    }

    public String getCreditType() {
        return creditType;
    }

    public void setCreditType(String creditType) {
        this.creditType = creditType;
    }
}
```

```

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getCurrency() {
    return currency;
}

public void setCurrency(String currency) {this.currency =
currency;}

public double getAnnualInterestRate() {
    return annualInterestRate;
}

public void setAnnualInterestRate(double annualInterestRate) {
    this.annualInterestRate = annualInterestRate;
}
@Override
public String toString() {
    return "Кредит: " +
        "Тип = " + creditType +
        ", Наименование = " + name +
        ", Валюта = " + currency +
        ", Годовая процентная ставка = " +
annualInterestRate;
}
}

```

Описание: Данный код представляет класс Credit (Кредит), который предназначен для представления информации о кредите в банковской системе. Вот краткое описание каждого элемента класса:

1. Приватные переменные:
 - o creditType: хранит тип кредита.
 - o name: хранит наименование кредита.
 - o currency: хранит валюту кредита.
 - o annualInterestRate: хранит годовую процентную ставку по кредиту.
2. Конструктор Credit(String creditType, String name, String currency, double annualInterestRate): инициализирует объект класса Credit с заданными типом кредита, наименованием, валютой и годовой процентной ставкой. Конструктор принимает эти значения и устанавливает их в соответствующие приватные переменные класса.

3. Геттеры и сеттеры для свойств `creditType`, `name`, `currency` и `annualInterestRate`: позволяют получать значения и устанавливать новые значения для соответствующих свойств кредита.
4. Переопределенный метод `toString()`: возвращает строковое представление кредита, содержащее его тип, наименование, валюту и годовую процентную ставку.

Этот класс позволяет представлять информацию о кредите в банковской системе. Объекты класса `Credit` могут использоваться для хранения и управления данными о кредитах, таких как тип, наименование, валюта и процентная ставка.

Класс Contract

```
/**
    Данный код представляет класс Contract (Договор), который
    используется для представления информации о договоре кредита в
    банковской системе.
    Краткое описание каждого элемента класса:
    1. Приватные переменные:
        contractNumber: хранит номер договора.
        date: хранит дату заключения договора.
        client: объект класса Client, представляющий клиента,
        связанного с договором.
        credit: объект класса Credit, представляющий кредит, связанный
        с договором.
        totalLoanAmount: хранит общую сумму займа по договору.
        loanRepaymentTerm: хранит срок погашения займа по договору.
    2. Конструктор Contract инициализирует объект класса Contract с
    заданными параметрами.
        Конструктор принимает данные и устанавливает их в
    соответствующие приватные переменные класса.
    3. Геттеры и сеттеры позволяют получать значения и устанавливать
    новые значения для соответствующих свойств договора.
    4. Метод calculateIncome(): вычисляет доход банка по договору на
    основе общей суммы займа и годовой процентной ставки кредита.
    Результат выводится на экран.
    5. Переопределенный метод toString(): возвращает строковое
    представление договора.
    Этот класс позволяет представлять информацию о договоре кредита в
    банковской системе.
    Объекты класса Contract используются для хранения и управления
    данными о договорах. Класс также предоставляет метод для
    вычисления дохода банка по договору.
    * @author Ganziuc Alexandr
    * @version 17.0.3.1
    */
public class Contract {
    private String contractNumber;
    private String date;
    private Client client;
    private Credit credit;
    private double totalLoanAmount;
    private int loanRepaymentTerm;

    public Contract(String contractNumber, String date, Client
    client, Credit credit, double totalLoanAmount, int
    loanRepaymentTerm) {
        this.contractNumber = contractNumber;
        this.date = date;
        this.client = client;
        this.credit = credit;
        this.totalLoanAmount = totalLoanAmount;
    }
}
```

```

        this.loanRepaymentTerm = loanRepaymentTerm;
    }

    public String getContractNumber() {
        return contractNumber;
    }

    public void setContractNumber(String contractNumber) {
        this.contractNumber = contractNumber;
    }

    public String getDate() {
        return date;
    }

    public void setDate(String date) {
        this.date = date;
    }

    public Client getClient() {
        return client;
    }

    public void setClient(Client client) {
        this.client = client;
    }

    public Credit getCredit() {
        return credit;
    }

    public void setCredit(Credit credit) {
        this.credit = credit;
    }

    public double getTotalLoanAmount() {
        return totalLoanAmount;
    }

    public void setTotalLoanAmount(double totalLoanAmount) {
        this.totalLoanAmount = totalLoanAmount;
    }

    public int getLoanRepaymentTerm() {
        return loanRepaymentTerm;
    }

    public void setLoanRepaymentTerm(int loanRepaymentTerm) {
        this.loanRepaymentTerm = loanRepaymentTerm;
    }

    public void calculateIncome() {
        double income = totalLoanAmount *
(credit.getAnnualInterestRate() / 100);
        System.out.println("Доход банка: " + income + " "
+credit.getCurrency());
    }

```



```

    }

    @Override
    public String toString() {
        return "Номер контракта: " + contractNumber + "\n" +
            "Дата: " + date + "\n" +
            "Клиент: " + client.toString() + "\n" +
            "Кредит: " + credit.toString() + "\n" +
            "Общая сумма займа: " + totalLoanAmount + "\n" +
            "Срок погашения займа: " + loanRepaymentTerm;
    }
}

```

Описание: Данный код представляет класс `Contract` (Договор), который используется для представления информации о договоре кредита в банковской системе. Вот краткое описание каждого элемента класса:

1. Приватные переменные:
 - o `contractNumber`: хранит номер договора.
 - o `date`: хранит дату заключения договора.
 - o `client`: объект класса `Client`, представляющий клиента, связанного с договором.
 - o `credit`: объект класса `Credit`, представляющий кредит, связанный с договором.
 - o `totalLoanAmount`: хранит общую сумму займа по договору.
 - o `loanRepaymentTerm`: хранит срок погашения займа по договору.
2. Конструктор `Contract(String contractNumber, String date, Client client, Credit credit, double totalLoanAmount, int loanRepaymentTerm)`: инициализирует объект класса `Contract` с заданными параметрами. Конструктор принимает номер договора, дату, объекты клиента и кредита, общую сумму займа и срок погашения займа, и устанавливает их в соответствующие приватные переменные класса.
3. Геттеры и сеттеры для свойств `contractNumber`, `date`, `client`, `credit`, `totalLoanAmount` и `loanRepaymentTerm`: позволяют получать значения и устанавливать новые значения для соответствующих свойств договора.
4. Метод `calculateIncome()`: вычисляет доход банка по договору на основе общей суммы займа и годовой процентной ставки кредита. Результат выводится на экран.
5. Переопределенный метод `toString()`: возвращает строковое представление договора, содержащее его номер, дату, информацию о клиенте и кредите, общую сумму займа и срок погашения.

Этот класс позволяет представлять информацию о договоре кредита в банковской системе. Объекты класса `Contract` используются для хранения и управления данными о договорах, включая информацию о клиенте, кредите, сумме займа и сроке погашения. Класс также предоставляет метод для вычисления дохода банка по договору.

Управляющий класс BankManagementSystem

```
/**
 * Класс BankManagementSystem управляет всей деятельностью банка
 * Этот класс содержит методы:
 * Метод addIndividualClient добавляет нового физического клиента
 в список individualClients.
 * Пользователь вводит личный код клиента, а затем создается
 экземпляр класса IndividualClient с указанными данными и
 добавляется в список.
 * Метод addClient() предлагает пользователю выбрать тип клиента
 (физическое или юридическое лицо) и вводит соответствующие данные.
 * Затем вызывается соответствующий метод (addIndividualClient или
 addLegalClient), который добавляет клиента в соответствующий
 список.
 * Метод addCredit() позволяет пользователю добавить новый кредит.
 * Пользователь вводит информацию о кредите. Создается экземпляр
 класса Credit с указанными данными и добавляется в список credits.
 * Метод createContract() позволяет пользователю создать новый
 контракт. Созданный контракт добавляется в список contracts.
 * Метод calculateBankIncome() предназначен для расчета дохода
 банка.
 * Метод ищет введенный номер контракта в списке contracts по
 указанному номеру и вызывает метод calculateIncome() для расчета
 дохода контракта.
 * Метод convertPriceToEuro() позволяет пользователю
 конвертировать цену из леев в евро.
 * Пользователь вводит текущую цену в лях и курс обмена (1 евро в
 лях), а затем производится конвертация цены в евро и результат
 выводится на экран.
 * Метод findContractByNumber выполняет поиск контракта по
 указанному номеру в списке contracts и возвращает соответствующий
 контракт.
 * В коде также присутствуют блоки try-catch для обработки
 исключений, таких как некорректный ввод данных и многие другие.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
import java.text.DecimalFormat;
import java.util.*;

public class BankManagementSystem implements BankSystem{
    List<BankBranch> bankBranches;
    List<IndividualClient> individualClients;
    List<CorporateClient> corporateClients;
    private static List<Credit> credits;
    private static List<Contract> contracts;
    private static final Scanner scanner = new Scanner(System.in);

    public BankManagementSystem() {
        bankBranches = new ArrayList<>();
        individualClients = new ArrayList<>();
    }
}
```

```

        corporateClients = new ArrayList<>();
        contracts = new ArrayList<>();
        credits = new ArrayList<>();
    }
    //Добавление информации для тестов
    private void addForTest() {
        //<editor-fold desc="Добавление филиалов банка">
        bankBranches.add(new BankBranch("MAIB", "Stefan Cel Mare
16", "+37368773211"));
        bankBranches.add(new BankBranch("MAIB", "Stefan Cel Mare
36", "+3736812321"));
        bankBranches.add(new BankBranch("MAIB", "Ismail
26", "+37368563213"));
        bankBranches.add(new BankBranch("MAIB", "Dacia
47", "+373687950214"));
        bankBranches.add(new BankBranch("MAIB", "Milestiu
22", "+373687123215"));
        bankBranches.add(new BankBranch("MAIB", "Ciuflea
48", "+37368655223"));
        bankBranches.add(new BankBranch("MAIB", "Dacia
116", "+37368224423"));
        bankBranches.add(new BankBranch("MAIB", "Cuza-Voda
116", "+37368223344"));
        bankBranches.add(new BankBranch("MAIB", "Cuza-Voda
11", "+373682233445"));
        bankBranches.add(new BankBranch("MAIB", "Lev Tolstoi
111", "+37368214906"));
        //</editor-fold>

        //<editor-fold desc="Добавление физических клиентов">
        individualClients.add(new IndividualClient("Ulises
Patterson", "Bucuresti 5", "+37368234567", "2004023456789"));
        individualClients.add(new IndividualClient("Thatcher
Jackson", "Stefan cel Mare 12", "+37369345678", "2004067890123"));
        individualClients.add(new IndividualClient("Sterling
Sanchez", "Ion Creanga 27", "+37366456789", "2004012345678"));
        individualClients.add(new IndividualClient("Shepherd Green",
"Mitropolit Grigore Banulescu-Bodoni 14", "+37367987654",
"2004098765432"));
        individualClients.add(new IndividualClient("Rhys Young",
"Mircea cel Batran 33", "+37368876543", "2004054321098"));
        individualClients.add(new IndividualClient("Quinnton Moore",
"Vasile Alecsandri 19", "+37369543210", "2004076543210"));
        individualClients.add(new IndividualClient("Otis Butler",
"Decebal 22", "+37366765432", "2004087654321"));
        individualClients.add(new IndividualClient("Israel Rogers",
"Alecu Russo 9", "+37367432109", "2004032109876"));
        individualClients.add(new IndividualClient("Griffin Watson",
"Cetatea Alba 6", "+37368109876", "2004043210987"));
        individualClients.add(new IndividualClient("Desmond Torres",
"Dacia 8", "+37369098765", "2004009876543"));
        //</editor-fold>

```

```

//<editor-fold desc="Добавление юридических клиентов">
    corporateClients.add(new CorporateClient("GlobalTech
Solutions", "Victoriei 10", "+37368234567", "1021234567", "John
Smith", "Alex Turner"));
    corporateClients.add(new CorporateClient("PrimeFinancial
Group", "Stefan cel Mare 25", "+37369345678", "1029876543", "John
Smith", "Emma Davis"));
    corporateClients.add(new CorporateClient("TechNova
Corporatio", "Mihai Eminescu 7", "+37366456789", "1025432167",
"Michael Brown", "Benjamin Scott"));
    corporateClients.add(new CorporateClient("EliteMarketing
Strategies", "Dacia 14", "+37367987654", "1028765432", "Olivia
Davis", "Olivia Hughes"));
    corporateClients.add(new CorporateClient("PowerTech
Industries", "Primaverii 3", "+37368876543", "1023145678", "Ethan
Wilson", "Ethan Carter"));
    corporateClients.add(new CorporateClient("OptiPro Logistics",
"Decebal 21", "+37369543210", "1027654321", "Sophia Thompson",
"Sophia Brooks"));
    corporateClients.add(new CorporateClient("NovaTech
Solutions", "Ion Creanga 12", "+37366765432", "1022345678",
"William Clark", "William Mitchell"));
    corporateClients.add(new CorporateClient("ApexCorp Holdings",
"Renasterii 9", "+37367432109", "1028765432", "Ava Anderson",
"Ava Reed"));
    corporateClients.add(new CorporateClient("InnovateTech
Solutions", "Unirii 17", "+37368109876", "1027654321", "James
Green", "James Foster"));
    corporateClients.add(new CorporateClient("SecureNet Systems",
"Alexandru cel Bun 6", "+37369098765", "1021234567", "Mia Taylor",
"Mia Collins"));
//</editor-fold>

//<editor-fold desc="Добавление кредитов">
    credits.add(new Credit("Mortgage", "Home Loan", "MDL", 3.5));
    credits.add(new Credit("Personal", "Personal Loan", "EUR",
8.2));
    credits.add(new Credit("Auto", "Car Loan", "USD", 5.9));
    credits.add(new Credit("Business", "Small Business Loan",
"EUR", 12.5));
    credits.add(new Credit("Student", "Education Loan", "MDL",
4.8));
    credits.add(new Credit("Mortgage", "Property Loan", "EUR",
4.1));
    credits.add(new Credit("Personal", "Wedding Loan", "MDL",
9.5));
    credits.add(new Credit("Auto", "Lease Financing", "EUR",
6.4));
    credits.add(new Credit("Business", "Equipment Loan", "MDL",
10.8));
    credits.add(new Credit("Student", "Student Line of Credit",
"EUR", 3.9));
//</editor-fold>

```

```

        //<editor-fold desc="Добавление контрактов">
        contracts.add(new Contract("1", "2023-06-01",
individualClients.get(0), credits.get(0), 250000, 20));
        contracts.add(new Contract("2", "2023-06-02",
individualClients.get(1), credits.get(1), 5000, 3));
        contracts.add(new Contract("3", "2023-06-03",
individualClients.get(2), credits.get(2), 30000, 5));
        contracts.add(new Contract("4", "2023-06-04",
individualClients.get(3), credits.get(3), 1000000, 10));
        contracts.add(new Contract("5", "2023-06-05",
individualClients.get(4), credits.get(4), 15000, 2));
        contracts.add(new Contract("6", "2023-06-06",
corporateClients.get(5), credits.get(5), 200000, 15));
        contracts.add(new Contract("7", "2023-06-07",
corporateClients.get(6), credits.get(6), 10000, 2));
        contracts.add(new Contract("8", "2023-06-08",
corporateClients.get(7), credits.get(7), 40000, 4));
        contracts.add(new Contract("9", "2023-06-09",
corporateClients.get(8), credits.get(8), 500000, 7));
        contracts.add( new Contract("10", "2023-06-10",
corporateClients.get(9), credits.get(9), 20000, 3));

        //</editor-fold>
    }

    public static void main(String[] args) {
        BankManagementSystem bankManagementSystem = new
BankManagementSystem(); // Создаем экземпляр класса
        bankManagementSystem.addForTest();
        bankManagementSystem.displayMenu(); // Используем
экземпляр для вызова метода displayMenu()
    }

    // ВЫВОД МЕНЮ
    private void displayMenu() {
        int choice = 0;
        do {
            System.out.println("\n\nМеню:");
            System.out.println("1. Добавить/создать");
            System.out.println("2. Вывести");
            System.out.println("3. Рассчитать доход банка");
            System.out.println("4. Конвертировать цену в евро");
            System.out.println("5. Выход");
            System.out.print("Выберите пункт меню: ");

            try {
                choice = Integer.parseInt(scanner.nextLine());

                switch (choice) {
                    case 1:
                        System.out.println("1. Добавить филиал
банка");

```

```

        System.out.println("2. Добавить клиента");
        System.out.println("3. Добавить кредит");
        System.out.println("4. Заключить
договор");

        System.out.println("5. Обратно");
        System.out.print("Выберите пункт меню: ");
        int subChoice =
Integer.parseInt(scanner.nextLine());
        System.out.println(subChoice);
        switch (subChoice) {
            case 1:
                addBankBranch();
                break;

            case 2:
                addClient();
                break;

            case 3:
                addCredit();
                break;

            case 4:
                createContract();
                break;
            case 5:
                System.out.println("Возвращаемся
обратно...");
                break;
            default:
                System.out.println("Неверный пункт
меню. Повторите выбор.");
                break;
        }
        break;

    case 2:
        System.out.println("1. Вывести список
договоров");
        System.out.println("2. Вывести список
филиалов");
        System.out.println("3. Вывести список
клиентов");
        System.out.println("4. Вывести список
кредитов");
        System.out.println("5. Обратно");
        System.out.print("Выберите пункт меню: ");
        subChoice =
Integer.parseInt(scanner.nextLine());

        switch (subChoice) {
            case 1:
                displayContracts();

```

```

        break;

        case 2:
            displayBankBranches();
            break;

        case 3:
            displayClients(individualClients,
corporateClients);
            break;

        case 4:
            displayCredits();
            break;
        case 5:
            System.out.println("Возвращаемся
обратно...");
            break;
        default:
            System.out.println("Неверный пункт
меню. Повторите выбор.");
            break;
    }
    break;

    case 3:
        calculateBankIncome();
        break;

    case 4:
        convertPriceToEuro();
        break;

    case 5:
        System.out.println("Программа
завершена.");
        break;

    default:
        System.out.println("Неверный пункт меню.
Повторите выбор.");
        break;
    }
} catch (NumberFormatException e) {
    System.out.println("Неверный пункт меню. Повторите
выбор.");
}
} while (choice != 5);
}

// вывод филиалов банка
@Override
public void displayBankBranches() {

```

```

        if(bankBranches.size()>0)
            for (BankBranch branch : bankBranches)
                System.out.println(branch.toString());
            else System.out.println("Филиалов нет!");
        }

        // вывод клиентов банка
        @Override
        public void displayClients(List<IndividualClient>
individualClients, List<CorporateClient> corporateClients) {
            if(individualClients.size()>0) {
                System.out.println("Список клиентов:");
                System.out.println("\n\nФизические лица:");
                for (IndividualClient client : individualClients)
                    System.out.println(individualClients.indexOf(client)+1+" ". +
client.toString()+"\n");
            }else System.out.println("\n\nФизических лиц нет!");
            if(corporateClients.size()>0) {
                System.out.println("\n\nЮридические лица:");
                for (CorporateClient client : corporateClients)
                    System.out.println(corporateClients.indexOf(client)+1+individualCl
ients.size() + ". "+client.toString()+"\n");
            }else System.out.println("\n\nЮридических лиц нет!");
        }

        // вывод кредитов
        @Override
        public void displayCredits() {
            if (credits.size()>0) {
                System.out.println("Список кредитов:");
                for (Credit credit : credits) {
                    System.out.println(credits.indexOf(credit) + ". "
+ credit);
                }
            }else System.out.println("\nКредитов нет!\n");
        }

        // вывод контрактов
        @Override
        public void displayContracts() {
            if(contracts.size()>0) {
                System.out.println("Список договоров:");
                for (Contract contract : contracts) {
                    System.out.println("Номер договора: " +
contract.getContractNumber());
                    System.out.println("Дата договора: " +
contract.getDate());
                    System.out.println("Имя клиента: " +
contract.getClient().getName());
                    System.out.println("Тип кредита: " +
contract.getCredit().getCreditType());
                    System.out.println("Сумма кредита: " +
contract.getTotalLoanAmount());
                    System.out.println("Срок погашения кредита: " +

```



```

contract.getLoanRepaymentTerm() + " месяцев");
        System.out.println();
    }
    }else System.out.println("\nДоговоров нет!\n");
}
// добавление нового филиала банка
@Override
public void addBankBranch() {
    System.out.println("Добавление филиала банка:");
    System.out.print("Введите имя филиала: ");
    String name = scanner.nextLine();
    System.out.print("Введите адрес филиала: ");
    String address = scanner.nextLine();
    System.out.print("Введите телефон филиала: ");
    String phoneNumber = scanner.nextLine();

    BankBranch bankBranch = new BankBranch(name, address,
phoneNumber);
    bankBranches.add(bankBranch);
    // Добавление филиала в список или другую структуру данных
    System.out.println("Филиал банка добавлен.");
}
// добавление нового юридического лица
public void addLegalClient(String name, String address, String
phoneNumber) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Введите идентификационный номер
налогоплательщика: ");
    String taxIdentificationNumber = scanner.nextLine();

    System.out.print("Введите имя администратора: ");
    String administratorName = scanner.nextLine();

    System.out.print("Введите имя контактного лица: ");
    String contactPerson = scanner.nextLine();

    CorporateClient client = new CorporateClient(name,
address, phoneNumber, taxIdentificationNumber, administratorName,
contactPerson);
    corporateClients.add(client);
    System.out.println("Юридический клиент добавлен.");
}
// добавление нового физического лица
public void addIndividualClient(String name, String address,
String phoneNumber) {
    System.out.print("Введите личный код: ");
    String personalCode = scanner.nextLine();

    IndividualClient client = new IndividualClient(name,
address, phoneNumber, personalCode);
    individualClients.add(client);
    System.out.println("Физический клиент добавлен.");
}

```

```

    }

    // общая функция добавления клиента
    @Override
    public void addClient() {

        System.out.println("Выберите тип клиента:");
        System.out.println("1. Физическое лицо");
        System.out.println("2. Юридическое лицо");
        int choice = 0;
        try {
            choice = scanner.nextInt();
            scanner.nextLine();
        } catch (InputMismatchException e) {
            System.out.println("Введены некорректные данные. Пожалуйста, введите данные в правильном формате.");
            scanner.nextLine(); // очистка буфера
            return;
        }

        System.out.println("Введите имя:");
        String name = scanner.nextLine();

        System.out.println("Введите адрес:");
        String address = scanner.nextLine();

        System.out.println("Введите номер телефона:");
        String phoneNumber = scanner.nextLine();

        if (choice == 1) addIndividualClient(name, address,
phoneNumber);
        else if (choice == 2) addLegalClient(name, address,
phoneNumber);
        else System.out.println("Неверный выбор типа клиента.");
    }

    // добавление нового кредита
    @Override
    public void addCredit() {
        System.out.println("Добавление кредита:");
        System.out.print("Введите тип кредита: ");
        String creditType = scanner.nextLine();
        System.out.print("Введите наименование кредита: ");
        String name = scanner.nextLine();
        System.out.print("Введите валюту кредита: ");
        String currency = scanner.nextLine();
        System.out.print("Введите годовую процентную ставку
кредита: ");
        double annualInterestRate;
        try {
            annualInterestRate = scanner.nextDouble();
            scanner.nextLine(); // Очистка буфера
        } catch (InputMismatchException e) {
            System.out.println("Введены некорректные данные.

```

```

        Пожалуйста, введите данные в правильном формате.");
        return;
    }
    Credit credit = new Credit(creditType, name, currency,
annualInterestRate);
    credits.add(credit);
}
// создание нового контракта
@Override
public void createContract() {
    try {
        System.out.println("Заключение договора:");

        System.out.print("Введите номер контракта: ");
        String contractNumber = scanner.nextLine();
        System.out.print("Введите дату контракта dd.mm.yyyy:
");

        String date = scanner.nextLine();

        System.out.println("Выберите клиента:");
        displayClients(individualClients, corporateClients);
        int clientChoice = scanner.nextInt();
        scanner.nextLine(); // Очистка буфера

        Client client;
        if (clientChoice > 0 && clientChoice <=
individualClients.size()) {
            client = individualClients.get(clientChoice - 1);
        } else if (clientChoice > individualClients.size() &&
clientChoice <= individualClients.size() +
corporateClients.size()) {
            client = corporateClients.get(clientChoice -
individualClients.size() - 1);
        } else {
            System.out.println("Неверный выбор клиента.");
            return;
        }

        System.out.println("Выберите кредит:");
        displayCredits();
        int creditChoice = scanner.nextInt();
        scanner.nextLine(); // Очистка буфера

        Credit credit;
        if (creditChoice > 0 && creditChoice <=
credits.size()) {
            credit = credits.get(creditChoice - 1);
        } else {
            System.out.println("Неверный выбор кредита.");
            return;
        }

        System.out.print("Введите общую сумму займа: ");

```

```

        double totalLoanAmount = scanner.nextDouble();
        scanner.nextLine(); // Очистка буфера
        System.out.print("Введите срок погашения займа: ");
        int loanRepaymentTerm = scanner.nextInt();
        scanner.nextLine(); // Очистка буфера

        Contract contract = new Contract(contractNumber, date,
client, credit, totalLoanAmount, loanRepaymentTerm);
        contracts.add(contract);
        System.out.println("Договор заключен.");
    } catch (InputMismatchException e) {
        System.out.println("Введены некорректные данные.
Пожалуйста, введите данные в правильном формате.");
    } catch (NoSuchElementException e) {
        System.out.println("Не удалось считать ввод.
Пожалуйста, введите данные корректно.");
    }
}

// расчет дохода банка
@Override
public void calculateBankIncome() {
    System.out.println("Расчет дохода банка:");
    if (contracts == null) {throw new
IllegalArgumentException("Контрактов нет!");}

    System.out.print("Введите номер договора: ");
    String contractNumber = scanner.nextLine();
    // Найти договор по номеру
    Contract contract = findContractByNumber(contractNumber);

    if (contract != null) {
        contract.calculateIncome();
    } else {
        System.out.println("Договор не найден.");
    }
}

// поиск контракта по номеру
private static Contract findContractByNumber(String
contractNumber) {
    if (contractNumber == null) {throw new
IllegalArgumentException("Номер контракта не может быть [" +
contractNumber+ "]");}

    for (Contract contract : contracts) {
        if
(contract.getContractNumber().equals(contractNumber)) {
            return contract;
        }
    }
    return null;
}

// перевод цены из леев в евро

```

```

@Override
public void convertPriceToEuro() {
    System.out.println("Конвертация цены в евро:");
    try {
        System.out.print("Введите текущую цену в лей: ");
        double price = scanner.nextDouble();
        System.out.print("Введите курс (1 евро в леях): ");
        double rate = scanner.nextDouble();
        if(rate == 0) throw new ArithmeticException();
        double euroPrice = price / rate;
        DecimalFormat df = new DecimalFormat("#.00");
        System.out.println("Цена в евро: " +
df.format(euroPrice));
    } catch (InputMismatchException e) {
        System.out.println("\nОшибка ввода. Пожалуйста,
введите числовое значение.\n");
    } catch (ArithmeticException e) {
        System.out.println("\nОшибка (деление на ноль):" +
e.getMessage() + "\n");
    } finally {
        scanner.nextLine(); // Очистка буфера
    }
}
}

```

Описание: Класс BankManagementSystem управляет всей деятельностью банка. Этот класс содержит методы:

- 1) Метод addIndividualClient добавляет нового физического клиента в список individualClients.

Пользователь вводит личный код клиента, а затем создается экземпляр класса IndividualClient с указанными данными и добавляется в список.

- 2) Метод addClient() предлагает пользователю выбрать тип клиента (физическое или юридическое лицо) и вводит соответствующие данные.

Затем вызывается соответствующий метод (addIndividualClient или addLegalClient), который добавляет клиента в соответствующий список.

- 3) Метод addCredit() позволяет пользователю добавить новый кредит.

Пользователь вводит информацию о кредите. Создается экземпляр класса Credit с указанными данными и добавляется в список credits.

- 4) Метод createContract() позволяет пользователю создать новый контракт. Созданный контракт добавляется в список contracts.

- 5) Метод `calculateBankIncome()` предназначен для расчета дохода банка.
- 6) Метод ищет введенный номер контракта в списке `contracts` по указанному номеру и вызывает метод `calculateIncome()` для расчета дохода контракта.
- 7) Метод `convertPriceToEuro()` позволяет пользователю конвертировать цену из леев в евро.

Пользователь вводит текущую цену в леях и курс обмена (1 евро в леях), а затем производится конвертация цены в евро и результат выводится на экран.

- 8) Метод `findContractByNumber` выполняет поиск контракта по указанному номеру в списке `contracts` и возвращает соответствующий контракт.
- 9) В коде также присутствуют блоки `try-catch` для обработки исключений, таких как некорректный ввод данных и многие другие.

Работа программы:

```

Меню:
1. Добавить/создать
2. Вывести
3. Рассчитать доход банка
4. Конвертировать цену в евро
5. Выход
Выберите пункт меню: 2
1. Вывести список договоров
2. Вывести список филиалов
3. Вывести список клиентов
4. Вывести список кредитов
5. Обратно
Выберите пункт меню: 1
Список договоров:
Номер договора: 1
Дата договора: 2023-06-01
Имя клиента: Ulises Patterson
Тип кредита: Mortgage
Сумма кредита: 250000.0
Срок погашения кредита: 20 месяцев

Номер договора: 2
Дата договора: 2023-06-02
Имя клиента: Thatcher Jackson
Тип кредита: Personal
Сумма кредита: 5000.0
Срок погашения кредита: 3 месяцев

Номер договора: 3
Дата договора: 2023-06-03
Имя клиента: Sterling Sanchez
Тип кредита: Auto
Сумма кредита: 30000.0
Срок погашения кредита: 5 месяцев

```

```

Номер договора: 4
Дата договора: 2023-06-04
Имя клиента: Shepherd Green
Тип кредита: Business
Сумма кредита: 1000000.0
Срок погашения кредита: 10 месяцев

Номер договора: 5
Дата договора: 2023-06-05
Имя клиента: Rhys Young
Тип кредита: Student
Сумма кредита: 15000.0
Срок погашения кредита: 2 месяцев

Номер договора: 6
Дата договора: 2023-06-06
Имя клиента: OptiPro Logistics
Тип кредита: Mortgage
Сумма кредита: 200000.0
Срок погашения кредита: 15 месяцев

Номер договора: 7
Дата договора: 2023-06-07
Имя клиента: NovaTech Solutions
Тип кредита: Personal
Сумма кредита: 10000.0
Срок погашения кредита: 2 месяцев

Номер договора: 8
Дата договора: 2023-06-08
Имя клиента: ApexCorp Holdings
Тип кредита: Auto
Сумма кредита: 40000.0
Срок погашения кредита: 4 месяцев

```

```

Номер договора: 9
Дата договора: 2023-06-09
Имя клиента: InnovateTech Solutions
Тип кредита: Business
Сумма кредита: 500000.0
Срок погашения кредита: 7 месяцев

Номер договора: 10
Дата договора: 2023-06-10
Имя клиента: SecureNet Systems
Тип кредита: Student
Сумма кредита: 20000.0
Срок погашения кредита: 3 месяцев

Меню:
1. Добавить/создать
2. Вывести
3. Рассчитать доход банка
4. Конвертировать цену в евро
5. Выход
Выберите пункт меню: 2
1. Вывести список договоров
2. Вывести список филиалов
3. Вывести список клиентов
4. Вывести список кредитов
5. Обратно
Выберите пункт меню: 2

```

Филиал банка: Имя = MAIB, Адрес = Stefan Cel Mare 16, Телефон = +37368773211
Филиал банка: Имя = MAIB, Адрес = Stefan Cel Mare 36, Телефон = +3736812321
Филиал банка: Имя = MAIB, Адрес = Ismail 26, Телефон = +37368563213
Филиал банка: Имя = MAIB, Адрес = Dacia 47, Телефон = +373687950214
Филиал банка: Имя = MAIB, Адрес = Milestiu 22, Телефон = +373687123215
Филиал банка: Имя = MAIB, Адрес = Ciuflea 48, Телефон = +37368655223
Филиал банка: Имя = MAIB, Адрес = Dacia 116, Телефон = +37368224423
Филиал банка: Имя = MAIB, Адрес = Cuza-Voda 116, Телефон = +37368223344
Филиал банка: Имя = MAIB, Адрес = Cuza-Voda 11, Телефон = +373682233445
Филиал банка: Имя = MAIB, Адрес = Lev Tolstoi 111, Телефон = +37368214906

Меню:

1. Добавить/создать
2. Вывести
3. Рассчитать доход банка
4. Конвертировать цену в евро
5. Выход

Выберите пункт меню: 2

1. Вывести список договоров
2. Вывести список филиалов
3. Вывести список клиентов
4. Вывести список кредитов
5. Обратно

Выберите пункт меню: 3

Список клиентов:

Физические лица:

1. Имя: Ulises Patterson

Адрес: Bucuresti 5

Телефон: +37368234567

Личный код: 2004023456789

2. Имя: Thatcher Jackson

Адрес: Stefan cel Mare 12

Телефон: +37369345678

Личный код: 2004067890123

3. Имя: Sterling Sanchez

Адрес: Ion Creanga 27

Телефон: +37366456789

Личный код: 2004012345678

4. Имя: Shepherd Green

Адрес: Mitropolit Grigore Banulescu-Bodoni 14

Телефон: +37367987654

Личный код: 2004098765432

5. Имя: Rhys Young

Адрес: Mircea cel Batran 33

Телефон: +37368876543

Личный код: 2004054321098

6. Имя: Quinnton Moore

Адрес: Vasile Alecsandri 19

Телефон: +37369543210

Личный код: 2004076543210

7. Имя: Otis Butler

Адрес: Decebal 22

Телефон: +37366765432

Личный код: 2004087654321

8. Имя: Israel Rogers

Адрес: Alecu Russo 9

Телефон: +37367432109

Личный код: 2004032109876

9. Имя: Griffin Watson

Адрес: Cetatea Alba 6

Телефон: +37368109876

Личный код: 2004043210987

10. Имя: Desmond Torres

Адрес: Dacia 8

Телефон: +37369098765

Личный код: 2004009876543

Юридические лица:

11. Наименование: GlobalTech Solutions
Адрес: Victoriei 10
Телефон: +37368234567
Налоговый идентификационный номер: 1021234567
Имя администратора: John Smith
Контактное лицо: Alex Turner

12. Наименование: PrimeFinancial Group
Адрес: Stefan cel Mare 25
Телефон: +37369345678
Налоговый идентификационный номер: 1029876543
Имя администратора: John Smith
Контактное лицо: Emma Davis

13. Наименование: TechNova Corporatio
Адрес: Mihai Eminescu 7
Телефон: +37366456789
Налоговый идентификационный номер: 1025432167
Имя администратора: Michael Brown
Контактное лицо: Benjamin Scott

14. Наименование: EliteMarketing Strategies
Адрес: Dacia 14
Телефон: +37367987654
Налоговый идентификационный номер: 1028765432
Имя администратора: Olivia Davis
Контактное лицо: Olivia Hughes

15. Наименование: PowerTech Industries
Адрес: Primaverii 3
Телефон: +37368876543
Налоговый идентификационный номер: 1023145678
Имя администратора: Ethan Wilson
Контактное лицо: Ethan Carter

16. Наименование: OptiPro Logistics
Адрес: Decebal 21
Телефон: +37369543210
Налоговый идентификационный номер: 1027654321
Имя администратора: Sophia Thompson
Контактное лицо: Sophia Brooks

17. Наименование: NovaTech Solutions
Адрес: Ion Creanga 12
Телефон: +37366765432
Налоговый идентификационный номер: 1022345678
Имя администратора: William Clark
Контактное лицо: William Mitchell

18. Наименование: ApexCorp Holdings
Адрес: Renasterii 9
Телефон: +37367432109
Налоговый идентификационный номер: 1028765432
Имя администратора: Ava Anderson
Контактное лицо: Ava Reed

19. Наименование: InnovateTech Solutions
Адрес: Unirii 17
Телефон: +37368109876
Налоговый идентификационный номер: 1027654321
Имя администратора: James Green
Контактное лицо: James Foster

20. Наименование: SecureNet Systems
Адрес: Alexandru cel Bun 6
Телефон: +37369098765
Налоговый идентификационный номер: 1021234567
Имя администратора: Mia Taylor
Контактное лицо: Mia Collins

Меню:

1. Добавить/создать
2. Вывести
3. Рассчитать доход банка
4. Конвертировать цену в евро
5. Выход

Выберите пункт меню: 2

1. Вывести список договоров
2. Вывести список филиалов
3. Вывести список клиентов
4. Вывести список кредитов
5. Обратно

Выберите пункт меню: 4

Список кредитов:

0. Кредит: Тип = Mortgage, Наименование = Home Loan, Валюта = MDL, Годовая процентная ставка = 3.5
1. Кредит: Тип = Personal, Наименование = Personal Loan, Валюта = EUR, Годовая процентная ставка = 8.2
2. Кредит: Тип = Auto, Наименование = Car Loan, Валюта = USD, Годовая процентная ставка = 5.9
3. Кредит: Тип = Business, Наименование = Small Business Loan, Валюта = EUR, Годовая процентная ставка = 12.5
4. Кредит: Тип = Student, Наименование = Education Loan, Валюта = MDL, Годовая процентная ставка = 4.8
5. Кредит: Тип = Mortgage, Наименование = Property Loan, Валюта = EUR, Годовая процентная ставка = 4.1
6. Кредит: Тип = Personal, Наименование = Wedding Loan, Валюта = MDL, Годовая процентная ставка = 9.5
7. Кредит: Тип = Auto, Наименование = Lease Financing, Валюта = EUR, Годовая процентная ставка = 6.4
8. Кредит: Тип = Business, Наименование = Equipment Loan, Валюта = MDL, Годовая процентная ставка = 10.8
9. Кредит: Тип = Student, Наименование = Student Line of Credit, Валюта = EUR, Годовая процентная ставка = 3.9

Меню:

1. Добавить/создать
2. Вывести
3. Рассчитать доход банка
4. Конвертировать цену в евро
5. Выход

Выберите пункт меню: 3

Расчет дохода банка:

Введите номер договора: 1

Доход банка: 8750.0 MDL

Меню:

1. Добавить/создать
2. Вывести
3. Рассчитать доход банка
4. Конвертировать цену в евро
5. Выход

Выберите пункт меню: 4

Конвертация цены в евро:

Введите текущую цену в лей: 25000

Введите курс (1 евро в леях): 19,58

Цена в евро: 1276,81

Меню:

1. Добавить/создать
2. Вывести
3. Рассчитать доход банка
4. Конвертировать цену в евро
5. Выход

Выберите пункт меню: 5

Программа завершена.

Process finished with exit code 0

Вывод

Мне очень понравилась практика в колледже, мне было интересно справляться с задачами, узнавать что-то новое. Во время практики мы получаем новые знания, можем дорабатывать программы на своё усмотрение.

Мне понравились предложенные задания, возможность использовать всё, что позволяет фантазия и нет никаких ограничений. Также мне понравилось работать со всеми вместе в одном компьютерном классе и узнавать что-то новое друг от друга.

Во время практики отрицательных моментов не возникало.

Было бы интересно выделить 1 день практики для того, чтобы посмотреть проекты других групп, посмотреть какие-то идеи и может взять что-то для себя.

Библиография

C++

- 1) Брайан У. Керниган, Д. М. (2017). *Язык программирования C. Вильямс.*
- 2) Дэвис, С. Р. (2015). *C++ для чайников.* Москва - Санкт-Петербург- Киев : Диалектика.
- 3) Лафоре, Р. (2013). *Объектно-ориентированное программирование в C++.* ПИТЕР.
- 4) Страуструп, Б. (2015). *Язык программирования C++.* БИНОМ.

Java

- 1) Sierra, K., Bates, B. (2018). *Изучаем Java.* 9-е издание. ДМК Пресс.
- 2) Horstmann, C.S., Cornell, G. (2018). *Core Java, Volume I - Fundamentals.* 11th Edition. Prentice Hall.
- 3) Bloch, J. (2018). *Effective Java.* 3rd Edition. Addison-Wesley Professional.
- 4) Eckel, B. (2017). *Thinking in Java.* 4th Edition. Prentice Hall.
- 5) Naftalin, M., Wadler, P. (2006). *Java Generics and Collections.* O'Reilly Media.

Приложение

C++

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <queue>
#include <windows.h>
using namespace std;
const int MAX_SIZE = 50;

//функция для изменения цвета консоли
void setcolor(int text, int backG=0){
HANDLE color = GetStdHandle(STD_OUTPUT_HANDLE);
SetConsoleTextAttribute(color,(WORD)((backG << 4)| text));
}

//функция для чтения информации из файла
bool readField(int field[MAX_SIZE][MAX_SIZE], int& n, int& m) {
    ifstream inputFile("Teren.in.txt");
    if (inputFile.is_open() && !inputFile.eof()) {
        inputFile >> n >> m;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                inputFile >> field[i][j];

        inputFile.close();
    } else {
        cout << "Unable to open file Teren.in.txt";
        return false;
    }
    return true;
}

//функция для вывода информации
void writeField(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) cout << field[i][j] << " ";

        cout << endl;
    }
}

//функция для замены 1 и последней колонки матрицы
void swapColumns(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    for (int i = 0; i < n; i++) swap(field[i][0],field[i][m-1]);
}
```

```

//функция для очистки строки матрицы
void clearRow(int field[MAX_SIZE][MAX_SIZE], int row, int m) {
    for (int j = 0; j < m; j++)
        field[row][j] = 0;
}

//функция для очистки столбца матрицы
void clearColumn(int field[MAX_SIZE][MAX_SIZE], int column, int n) {
    for (int i = 0; i < n; i++)
        field[i][column] = 0;
}

//функция для поиска строки с максимальным количеством мин
void getMaxMinedRow(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    int maxMinedRow = 0;
    int maxMines = 0;
    for (int i = 0; i < n; i++) {
        int mines = 0;
        for (int j = 0; j < m; j++)
            if (field[i][j] == 1) mines++;

        if (mines > maxMines) {
            maxMines = mines;
            maxMinedRow = i;
        }
    }
    cout << maxMinedRow+1 << " ";
    for (int i = 0; i < n; i++) {
        int mines = 0;
        for (int j = 0; j < m; j++)
            if (field[i][j] == 1) mines++;

        if (mines == maxMines and i!=maxMinedRow) cout << i+1 << " ";
    }
    cout << endl;
}

//функция для поиска колонки с максимальным количеством мин
void getMaxMinedColumn(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    int maxMines = 0;
    int maxMinedColumn = 0; // Установите начальное значение для
maxMinedColumn

    for (int j = 0; j < m; j++) {
        int mines = 0;
        for (int i = 0; i < n; i++)
            if (field[i][j] == 1) mines++;

        if (mines > maxMines) {

```

```

        maxMines = mines;
        maxMinedColumn = j;
    }
}

cout << maxMinedColumn+1 << " ";

    for (int j = 0; j < m; j++) {
        int mines = 0;
        for (int i = 0; i < n; i++)
            if (field[i][j] == 1) mines++;

        if (mines == maxMines and j!=maxMinedColumn) {
            cout << j+1 << " ";
        }
    }
}

//функция для поиска среднего количества мин в нечётных строках
float getAverageMinedOddRows(int field[MAX_SIZE][MAX_SIZE], int n, int m)
{
    int minedRows = 0;
    int minedZones = 0;
    for (int i = 0; i < n; i += 2) {
        for (int j = 0; j < m; j++)
            if (field[i][j] == 1) minedZones++;

        minedRows++;
    }
    return (minedRows > 0) ? static_cast<float>(minedZones) / minedRows :
0.0;
}

//функция для поиска количества мин в колонках
void countMinesInColumns(int field[MAX_SIZE][MAX_SIZE], int n, int m, int
mineCounts[MAX_SIZE]) {
    for (int j = 0; j < m; j++) {
        int mines = 0;
        for (int i = 0; i < n; i++)
            if (field[i][j] == 1) mines++;

        mineCounts[j] = mines;
    }
}

//функция для сортировки колонок в убывающем порядке
void sortColumnsByMines(int columns[MAX_SIZE], int mineCounts[MAX_SIZE],
int m) {
    for (int i = 0; i < m; i++)
        for (int j = i + 1; j < m; j++)
            if (mineCounts[j] > mineCounts[i]) {

```

```

        swap(mineCounts[i], mineCounts[j]);
        swap(columns[i], columns[j]);
    }
}

//функция для копирования строк с минами в отдельный файл
void copyRowsWithMines(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    ofstream outputFile("Mine.txt");
    if (outputFile.is_open()) {
        for (int i = 0; i < n; i++) {
            bool hasMines = false;
            for (int j = 0; j < m; j++)
                if (field[i][j] == 1) {
                    hasMines = true;
                    break;
                }

            if (hasMines) {
                for (int j = 0; j < m; j++)
                    outputFile << field[i][j] << " ";

                outputFile << endl;
            }
        }
        outputFile.close();
    } else {
        cout << "Unable to open file Mine.txt";
    }
}

//функция для подсчёта объектов в матрицы
int countObjects(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
    int objects = 0;
    bool visited[MAX_SIZE][MAX_SIZE] = { false };

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (field[i][j] == 1 && !visited[i][j]) {
                objects++;
                visited[i][j] = true;

                int dx[] = { -1, 1, 0, 0, -1, -1, 1, 1 };
                int dy[] = { 0, 0, -1, 1, -1, 1, -1, 1 };
                for (int k = 0; k < 8; k++) {
                    int ni = i + dx[k];
                    int nj = j + dy[k];
                    if (ni >= 0 && ni < n && nj >= 0 && nj < m &&
field[ni][nj] == 1 && !visited[ni][nj])
                        visited[ni][nj] = true;
                }
            }
}

```

```

        return objects;
    }

    // создание структуры для поиска кратчайшего пути в матрице
    struct Cell {
        int row;
        int col;
    };

    // функция для поиска кратчайшего пути от начальной точки к конечной
    void findShortestPath(int field[MAX_SIZE][MAX_SIZE], int n, int m) {
        bool visited[MAX_SIZE][MAX_SIZE] = { false };
        Cell parent[MAX_SIZE][MAX_SIZE];

        queue<Cell> q;
        Cell start = { n - 1, m - 1 };
        Cell end = { 0, 0 };

        visited[start.row][start.col] = true;
        q.push(start);

        bool pathFound = false;

        while (!q.empty()) {
            Cell current = q.front();
            q.pop();

            int cx = current.row;
            int cy = current.col;

            if (cx == end.row && cy == end.col) {
                pathFound = true;
                break;
            }

            int dx[] = { -1, 1, 0, 0 };
            int dy[] = { 0, 0, -1, 1 };

            for (int k = 0; k < 4; k++) {
                int nx = cx + dx[k];
                int ny = cy + dy[k];

                if (nx >= 0 && nx < n && ny >= 0 && ny < m && field[nx][ny]
== 0 && !visited[nx][ny]) {
                    visited[nx][ny] = true;
                    parent[nx][ny] = { cx, cy };
                    q.push({ nx, ny });
                }
            }
        }
    }

```

```

    if (pathFound) {
        Cell current = end;

        Cell showPath[MAX_SIZE];
        int count = 0;
        while (current.row != start.row || current.col != start.col) {
            cout << "[" << current.row + 1 << ", " << current.col + 1 <<
"] - ";
            field[current.row][current.col] = 4;
            current = parent[current.row][current.col];

        }
        cout << "[" << start.row + 1 << ", " << start.col + 1 << "]" <<
endl;
        field[start.row][start.col] = 4;
    } else {
        cout << "No path found." << endl;
    }
}

int main() {
    setcolor(3,0);
    int field[MAX_SIZE][MAX_SIZE];
    int n, m;
    if(readField(field, n, m) == 0) return 10;

    int choice;
    do{
        cout << "Matrix\n";
        writeField(field,n,m);
        cout << "\nMenu:\n";
        cout << "1. Swap first and last columns\n";
        cout << "2. Clear zones in a row/column\n";
        cout << "3. Get row/column with max mines\n";
        cout << "4. Calculate average mined zones in odd rows\n";
        cout << "5. Sort columns by number of mines\n";
        cout << "6. Copy rows with mines to a file\n";
        cout << "7. Count objects\n";
        cout << "8. Find shortest path from top-left to bottom-right\n";
        cout << "9. Read data from a file\n";
        setcolor(4,0);
        cout << "10. Exit\n";
        setcolor(3,0);
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:{
                swapColumns(field, n, m);

```



```

        cout << "\nField after swapping first and last columns:"
<< endl;
        writeField(field, n, m);
        system("pause");
        system("CLS");
        break;
    }case 2:{
        int rowToClear, columnToClear, ch=0;
        cout << "\nWant to clear everything(1), row(2),column(3)
or nothing(0)?\n";
        do{
            if(ch<0 or ch>3){
                setcolor(4,0);
                cout << "\nError! Invalid choice! Try again!
\n";
                setcolor(3,0);
            }

            cout << "Enter your choice: ";
            cin >> ch;
        }while(ch<0 or ch>3);

        switch(ch){
        case 1 :
        case 2 :
            cout << "\nEnter the row to clear: ";
            cin >> rowToClear;
            clearRow(field, rowToClear - 1, m);
            cout << "\nField after clearing row " << rowToClear <<
":" << endl;
            writeField(field, n, m);
            if(ch!=1)break;
        case 3:
            cout << "\nEnter the column to clear: ";
            cin >> columnToClear;
            clearColumn(field, columnToClear - 1, n);
            cout << "\nField after clearing column " << columnToClear
<< ":" << endl;
            writeField(field, n, m);
            if(ch!=1)break;

        case 0 :
            setcolor(4,0);
            cout << "\nExit...\n"<<endl;
            setcolor(3,0);
            break;

        }

        system("pause");
        system("CLS");
        break;
    }
}

```

```

        }case 3:{
            int ch;
cout << "\nWant to see all(1), row(2),column(3) or nothing(0) with the
maximum number of mined zones?\n";
            do{
                if(ch<0 or ch>3){
                    setcolor(4,0);
                    cout << "\nError! Invalid choice! Try again!
\n";
                    setcolor(3,0);
                }

                cout << "Enter your choice: ";
                cin >> ch;
            }while(ch<0 or ch>3);

            switch(ch){
case 1 :
case 2 :{
cout << "\nRow with the maximum number of mined zones: ";
    getMaxMinedRow(field, n, m);
    if(ch!=1)break;
}case 3:{
    cout << "\nColumn with the maximum number of mined zones:
";

    getMaxMinedColumn(field, n, m);
    cout << endl;
    if(ch!=1)break;

    }case 0 :
        setcolor(4,0);
        cout << "\nExit...\n\n";
        setcolor(3,0);
        break;
    }
    system("pause");
    system("CLS");
    break;
}case 4:{
    float averageMinedOddRows = getAverageMinedOddRows(field,
n, m);

    cout << "\nAverage mined zones in odd rows: " <<
averageMinedOddRows << endl;
    system("pause");
    system("CLS");
    break;
}case 5:{
    int columns[MAX_SIZE];
    int mineCounts[MAX_SIZE];
    for (int j = 0; j < m; j++) {

```

```

        columns[j] = j + 1;
    }
    countMinesInColumns(field, n, m, mineCounts);
    sortColumnsByMines(columns, mineCounts, m);
    cout << "\nColumns in descending order of mine count:" <<
endl;

    for (int j = 0; j < m; j++) {
        cout << columns[j] << " ";
    }
    cout << endl;
    system("pause");
    system("CLS");
    break;
}case 6:{
    copyRowsWithMines(field, n, m);
    cout << "\nRows with mines copied to Mine.txt" << endl;
    system("pause");
    system("CLS");
    break;
}case 7:{
    int objects = countObjects(field, n, m);
    cout << "\nNumber of objects: " << objects << endl;
    system("pause");
    system("CLS");
    break;
}case 8:{
    cout << "\nShortest path from top-left to bottom-right:"
<< endl;
    findShortestPath(field, n, m);
    cout << "\nPath: \n";
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++)
                if(field[i][j] == 4){
                    field[i][j] = 0;
                    setcolor(10,0);
                    cout << field[i][j]<< " ";
                    setcolor(3,0);
                }else cout << field[i][j] << " ";

        cout << endl;
    }

    system("pause");
    system("CLS");
    break;
}case 9:{
    readField(field, n, m);
    setcolor(10,0);
    cout << "Data from the file has been read"<<endl;
    setcolor(3,0);
    system("pause");

```

```

        system("CLS");
        break;
    }case 10:{
        setcolor(4,0);
        cout << "Exit..."<<endl;
        setcolor(3,0);
        return 10;
        system("pause");
        system("CLS");
    }
    default:
        setcolor(4,0);
        cout << "\nInvalid choice. Please enter a valid
choice.\n";
        setcolor(3,0);
        system("pause");
        system("CLS");
        break;
    }
}while(choice!=10);

return 0;
}

```

```
/**
 * Интерфейс BankSystem
 * Данный интерфейс определяет набор методов, которые должны быть
 реализованы в классе, который использует этот интерфейс.
 * Он служит для описания функциональности банковской системы, а
 именно для управления филиалами банка, клиентами, кредитами,
 контрактами и другими операциями.
 * Вот краткое описание каждого метода в интерфейсе:
 * 1.    displayBankBranches(): Этот метод используется для
 отображения информации о всех филиалах банка.
 * 2.    displayClients(List<IndividualClient> individualClients,
 List<CorporateClient> corporateClients): Этот метод используется
 для отображения информации о клиентах банка. Он принимает два
 списка клиентов: individualClients (список физических лиц) и
 corporateClients (список юридических лиц).
 * 3.    displayCredits(): Этот метод используется для отображения
 информации о кредитах, доступных в банковской системе.
 * 4.    addBankBranch(): Этот метод используется для добавления
 нового филиала банка в систему.
 * 5.    addClient(): Этот метод используется для добавления нового
 клиента в систему. Новый клиент может быть как физическим лицом
 (IndividualClient), так и юридическим лицом (CorporateClient).
 * 6.    addCredit(): Этот метод используется для добавления нового
 кредита в систему.
 * 7.    createContract(): Этот метод используется для создания
 нового контракта в системе.
 * 8.    calculateBankIncome(): Этот метод используется для расчета
 дохода банка.
 * 9.    displayContracts(): Этот метод используется для
 отображения информации о контрактах, заключенных в банковской
 системе.
 * 10.   convertPriceToEuro(): Этот метод используется для
 конвертации цены в евро.
 * Интерфейс предоставляет абстракцию для операций, которые могут
 быть выполнены в банковской системе.
 * Реализация этих методов будет зависеть от конкретного класса,
 который реализует данный интерфейс и предоставляет конкретную
 функциональность для работы с банковской системой.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
import java.util.List;

public interface BankSystem {
    void displayBankBranches();
    void displayClients(List<IndividualClient> individualClients,
 List<CorporateClient> corporateClients);
    void displayCredits();
    void addBankBranch();
}
```

```

    void addClient();
    void addCredit();
    void createContract();
    void calculateBankIncome();
    void displayContracts();
    void convertPriceToEuro();
}

```

BankBranch.java

```

/**
 * Базовый класс "Филиал банка"
 * Этот класс служит в качестве базового класса для представления
 * филиалов банка и содержит некоторые свойства и методы для работы с
 * ними.
 * 1.    Приватные переменные:
 *    name: хранит имя филиала банка.
 *    address: хранит адрес филиала банка.
 *    phoneNumber: хранит номер телефона филиала банка.
 * 2.    Конструктор BankBranch инициализирует объект класса
 * BankBranch с заданными именем, адресом и номером телефона филиала.
 * 3.    Геттеры и сеттеры для свойств name, address и phoneNumber:
 * позволяют получать значения и устанавливать новые значения для
 * соответствующих свойств филиала банка.
 * 4.    Переопределенный метод toString(): возвращает строковое
 * представление филиала банка, содержащее его имя, адрес и номер
 * телефона.
 * Этот класс предоставляет базовый функционал для представления
 * филиала банка.
 * Можно использовать его как основу для создания объектов
 * филиалов и управления их свойствами.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public class BankBranch {
    private String name;
    private String address;
    private String phoneNumber;

    public BankBranch(String name, String address, String
phoneNumber) {
        this.name = name;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }

    public String getName() {return name;}

    public void setName(String name) {
        this.name = name;
    }

```

```

    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber)
    {this.phoneNumber = phoneNumber;}
    @Override
    public String toString() {
        return "Филиал банка: " +
            "Имя = " + name +
            ", Адрес = " + address +
            ", Телефон = " + phoneNumber;
    }
}

```

Client.java

```

/**
 * Данный код представляет абстрактный класс Client (Клиент).
 * Этот класс служит базовым классом для представления клиентов
 * банка и содержит общие свойства и методы для работы с клиентами.
 * Краткое описание каждого элемента класса:
 * 1. Приватные переменные:
 *   name: хранит имя клиента.
 *   address: хранит адрес клиента.
 *   phoneNumber: хранит номер телефона клиента.
 * 2. Конструктор Client инициализирует объект класса Client с
 * заданными именем, адресом и номером телефона клиента.
 * 3. Геттеры и сеттеры для свойств name, address и phoneNumber:
 * позволяют получать значения и устанавливать новые значения для
 * соответствующих свойств клиента.
 * 4. Абстрактный метод calculateCreditLimit(): предоставляет
 * абстрактную реализацию для расчета кредитного лимита клиента. Этот
 * метод должен быть реализован в подклассах, наследующих абстрактный
 * класс Client.
 * 5. Переопределенный метод toString(): возвращает строковое
 * представление клиента, содержащее его имя, адрес и номер телефона.
 * Этот абстрактный класс предоставляет базовый функционал для
 * представления клиентов банка.
 * Он может быть использован в качестве основы для создания
 * подклассов, представляющих конкретные типы клиентов

```

```

* (например, физические лица и юридические лица), и расширения
функциональности клиентов в этих подклассах.

* @author Ganziuc Alexandr
* @version 17.0.3.1
*/
public abstract class Client {
    private String name;
    private String address;
    private String phoneNumber;

    public Client(String name, String address, String phoneNumber)
    {
        this.name = name;
        this.address = address;
        this.phoneNumber = phoneNumber;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    @Override
    public String toString() {
        return "Клиент: " +
            "Имя = " + name +
            ", Адрес = " + address +
            ", Телефон = " + phoneNumber;
    }
}

```



```

/**
 * Данный код представляет класс IndividualClient (Физическое
 * лицо), который является подклассом класса Client.
 * Этот класс расширяет функциональность базового класса Client,
 * добавляя специфичные свойства и методы для представления
 * физических лиц в банковской системе. Вот краткое описание каждого
 * элемента класса:
 * 1. Приватная переменная:
 *   * personalCode: хранит личный код физического лица.
 * 2. Конструктор IndividualClient инициализирует объект класса
 *   IndividualClient с заданными именем, адресом, номером телефона и
 *   личным кодом физического лица. Конструктор также вызывает
 *   конструктор базового класса Client с помощью ключевого слова
 *   super, чтобы инициализировать общие свойства клиента.
 * 3. Геттер и сеттер для свойства personalCode: позволяют
 *   получать значение и устанавливать новое значение личного кода
 *   физического лица.
 * 4. Переопределенный метод toString(): возвращает строковое
 *   представление физического лица, содержащее его имя, адрес, номер
 *   телефона и личный код.
 * Этот класс предоставляет конкретную реализацию для
 * представления физического лица в банковской системе.
 * Он наследует общий функционал от базового класса Client и
 * добавляет специфичные свойства и методы для физических лиц.
 * Это позволяет использовать объекты класса IndividualClient для
 * представления и управления информацией о физических лицах в
 * банковской системе.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public class IndividualClient extends Client {
    private String personalCode;

    public IndividualClient(String name, String address, String
phoneNumber, String personalCode) {
        super(name, address, phoneNumber);
        this.personalCode = personalCode;
    }

    public String getPersonalCode() {
        return personalCode;
    }

    public void setPersonalCode(String personalCode) {
        this.personalCode = personalCode;
    }

    @Override
    public String toString() {

```

```

        return "Имя: " + getName() + "\n" +
               "Адрес: " + getAddress() + "\n" +
               "Телефон: " + getPhoneNumber() + "\n" +
               "Личный код: " + personalCode;
    }
}

```

CorporateClient.java

```

/**
 * Данный код представляет класс CorporateClient (Юридическое
 * лицо), который также является подклассом класса Client.
 * Этот класс расширяет функциональность базового класса Client,
 * добавляя специфичные свойства и методы для представления
 * юридических лиц в банковской системе. Вот краткое описание каждого
 * элемента класса:
 * 1. Приватные переменные:
 *   taxIdentificationNumber: хранит налоговый идентификационный
 *   номер юридического лица.
 *   administratorName: хранит имя администратора юридического
 *   лица.
 *   contactPerson: хранит имя контактного лица юридического лица.
 * 2. Конструктор CorporateClient инициализирует объект класса
 * CorporateClient с заданными именем, адресом, номером телефона,
 * налоговым идентификационным номером, именем администратора и
 * контактным лицом юридического лица.
 * Конструктор также вызывает конструктор базового класса Client с
 * помощью ключевого слова super, чтобы инициализировать общие
 * свойства клиента.
 * 3. Геттеры и сеттеры позволяют получать значения и
 * устанавливать новые значения для соответствующих свойств
 * юридического лица.
 * 4. Переопределенный метод toString(): возвращает строковое
 * представление юридического лица.
 * Этот класс предоставляет конкретную реализацию для
 * представления юридического лица в банковской системе.
 * Он наследует общий функционал от базового класса Client и
 * добавляет специфичные свойства и методы для юридических лиц.
 * Это позволяет использовать объекты класса CorporateClient для
 * представления и управления информацией о юридических лицах в
 * банковской системе.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public class CorporateClient extends Client {
    private String taxIdentificationNumber;
    private String administratorName;
    private String contactPerson;

    public CorporateClient(String name, String address, String
phoneNumber, String taxIdentificationNumber, String
administratorName, String contactPerson) {
        super(name, address, phoneNumber);
    }
}

```

```

        this.taxIdentificationNumber = taxIdentificationNumber;
        this.administratorName = administratorName;
        this.contactPerson = contactPerson;
    }

    public String getTaxIdentificationNumber() {
        return taxIdentificationNumber;
    }

    public void setTaxIdentificationNumber(String
taxIdentificationNumber) {this.taxIdentificationNumber =
taxIdentificationNumber;}

    public String getAdministratorName() {
        return administratorName;
    }

    public void setAdministratorName(String administratorName) {
        this.administratorName = administratorName;
    }

    public String getContactPerson() {return contactPerson;}

    public void setContactPerson(String contactPerson) {
        this.contactPerson = contactPerson;
    }

    @Override
    public String toString() {
        return "Наименование: " + getName() + "\n" +
            "Адрес: " + getAddress() + "\n" +
            "Телефон: " + getPhoneNumber() + "\n" +
            "Налоговый идентификационный номер: " +
taxIdentificationNumber + "\n" +
            "Имя администратора: " + administratorName + "\n"
+
            "Контактное лицо: " + contactPerson;
    }
}

```

```

/**
 * Данный код представляет класс Credit (Кредит), который
 * предназначен для представления информации о кредите в банковской
 * системе.
 * Краткое описание каждого элемента класса:
 * 1.    Приватные переменные:
 *    creditType: хранит тип кредита.
 *    name: хранит наименование кредита.
 *    currency: хранит валюту кредита.
 *    annualInterestRate: хранит годовую процентную ставку по
 *    кредиту.
 * 2.    Конструктор Credit инициализирует объект класса Credit с
 *    заданными типом кредита, наименованием, валютой и годовой
 *    процентной ставкой.
 *    Конструктор принимает эти значения и устанавливает их в
 *    соответствующие приватные переменные класса.
 * 3.    Геттеры и сеттеры позволяют получать значения и
 *    устанавливать новые значения для соответствующих свойств кредита.
 * 4.    Переопределенный метод toString(): возвращает строковое
 *    представление кредита, содержащее его тип, наименование, валюту и
 *    годовую процентную ставку.
 * Этот класс позволяет представлять информацию о кредите в
 * банковской системе.
 * Объекты класса Credit могут использоваться для хранения и
 * управления данными о кредитах, таких как тип, наименование, валюта
 * и процентная ставка.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
public class Credit {
    private String creditType;
    private String name;
    private String currency;
    private double annualInterestRate;

    public Credit(String creditType, String name, String currency,
double annualInterestRate) {
        this.creditType = creditType;
        this.name = name;
        this.currency = currency;
        this.annualInterestRate = annualInterestRate;
    }

    public String getCreditType() {
        return creditType;
    }

    public void setCreditType(String creditType) {
        this.creditType = creditType;
    }
}

```

```

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCurrency() {
        return currency;
    }

    public void setCurrency(String currency) {this.currency =
currency;}

    public double getAnnualInterestRate() {
        return annualInterestRate;
    }

    public void setAnnualInterestRate(double annualInterestRate) {
        this.annualInterestRate = annualInterestRate;
    }
    @Override
    public String toString() {
        return "Кредит: " +
            "Тип = " + creditType +
            ", Наименование = " + name +
            ", Валюта = " + currency +
            ", Годовая процентная ставка = " +
annualInterestRate;
    }
}

```

```

/**
    Данный код представляет класс Contract (Договор), который
    используется для представления информации о договоре кредита в
    банковской системе.
    Краткое описание каждого элемента класса:
    1. Приватные переменные:
        contractNumber: хранит номер договора.
        date: хранит дату заключения договора.
        client: объект класса Client, представляющий клиента,
        связанного с договором.
        credit: объект класса Credit, представляющий кредит, связанный
        с договором.
        totalLoanAmount: хранит общую сумму займа по договору.
        loanRepaymentTerm: хранит срок погашения займа по договору.
    2. Конструктор Contract инициализирует объект класса Contract с
    заданными параметрами.
        Конструктор принимает данные и устанавливает их в
        соответствующие приватные переменные класса.
    3. Геттеры и сеттеры позволяют получать значения и устанавливать
    новые значения для соответствующих свойств договора.
    4. Метод calculateIncome(): вычисляет доход банка по договору на
    основе общей суммы займа и годовой процентной ставки кредита.
    Результат выводится на экран.
    5. Переопределенный метод toString(): возвращает строковое
    представление договора.
    Этот класс позволяет представлять информацию о договоре кредита в
    банковской системе.
    Объекты класса Contract используются для хранения и управления
    данными о договорах. Класс также предоставляет метод для
    вычисления дохода банка по договору.
    * @author Ganziuc Alexandr
    * @version 17.0.3.1
    */
public class Contract {
    private String contractNumber;
    private String date;
    private Client client;
    private Credit credit;
    private double totalLoanAmount;
    private int loanRepaymentTerm;

    public Contract(String contractNumber, String date, Client
client, Credit credit, double totalLoanAmount, int
loanRepaymentTerm) {
        this.contractNumber = contractNumber;
        this.date = date;
        this.client = client;
        this.credit = credit;
        this.totalLoanAmount = totalLoanAmount;
        this.loanRepaymentTerm = loanRepaymentTerm;
    }
}

```

```

public String getContractNumber() {
    return contractNumber;
}

public void setContractNumber(String contractNumber) {
    this.contractNumber = contractNumber;
}

public String getDate() {
    return date;
}

public void setDate(String date) {
    this.date = date;
}

public Client getClient() {
    return client;
}

public void setClient(Client client) {
    this.client = client;
}

public Credit getCredit() {
    return credit;
}

public void setCredit(Credit credit) {
    this.credit = credit;
}

public double getTotalLoanAmount() {
    return totalLoanAmount;
}

public void setTotalLoanAmount(double totalLoanAmount)
{this.totalLoanAmount = totalLoanAmount;}

public int getLoanRepaymentTerm() {
    return loanRepaymentTerm;
}

public void setLoanRepaymentTerm(int loanRepaymentTerm) {
    this.loanRepaymentTerm = loanRepaymentTerm;
}

public void calculateIncome() {
    double income = totalLoanAmount *
(credit.getAnnualInterestRate() / 100);
    System.out.println("Доход банка: " + income + " "
+credit.getCurrency());
}

```

```

    }

    @Override
    public String toString() {
        return "Номер контракта: " + contractNumber + "\n" +
            "Дата: " + date + "\n" +
            "Клиент: " + client.toString() + "\n" +
            "Кредит: " + credit.toString() + "\n" +
            "Общая сумма займа: " + totalLoanAmount + "\n" +
            "Срок погашения займа: " + loanRepaymentTerm;
    }
}

```

BankManagementSystem.java

```

/**
 * Класс BankManagementSystem управляет всей деятельностью банка
 * Этот класс содержит методы:
 * Метод addIndividualClient добавляет нового физического клиента
 в список individualClients.
 * Пользователь вводит личный код клиента, а затем создается
 экземпляр класса IndividualClient с указанными данными и
 добавляется в список.
 * Метод addClient() предлагает пользователю выбрать тип клиента
 (физическое или юридическое лицо) и вводит соответствующие данные.
 * Затем вызывается соответствующий метод (addIndividualClient или
 addLegalClient), который добавляет клиента в соответствующий
 список.
 * Метод addCredit() позволяет пользователю добавить новый кредит.
 * Пользователь вводит информацию о кредите. Создается экземпляр
 класса Credit с указанными данными и добавляется в список credits.
 * Метод createContract() позволяет пользователю создать новый
 контракт. Созданный контракт добавляется в список contracts.
 * Метод calculateBankIncome() предназначен для расчета дохода
 банка.
 * Метод ищет введенный номер контракта в списке contracts по
 указанному номеру и вызывает метод calculateIncome() для расчета
 дохода контракта.
 * Метод convertPriceToEuro() позволяет пользователю
 конвертировать цену из леев в евро.
 * Пользователь вводит текущую цену в леех и курс обмена (1 евро в
 леех), а затем производится конвертация цены в евро и результат
 выводится на экран.
 * Метод findContractByNumber выполняет поиск контракта по
 указанному номеру в списке contracts и возвращает соответствующий
 контракт.
 * В коде также присутствуют блоки try-catch для обработки
 исключений, таких как некорректный ввод данных и многие другие.
 * @author Ganziuc Alexandr
 * @version 17.0.3.1
 */
import java.text.DecimalFormat;
import java.util.*;

```



```

public class BankManagementSystem implements BankSystem{
    List<BankBranch> bankBranches;
    List<IndividualClient> individualClients;
    List<CorporateClient> corporateClients;
    private static List<Credit> credits;
    private static List<Contract> contracts;
    private static final Scanner scanner = new Scanner(System.in);

    public BankManagementSystem() {
        bankBranches = new ArrayList<>();
        individualClients = new ArrayList<>();
        corporateClients = new ArrayList<>();
        credits = new ArrayList<>();
        contracts = new ArrayList<>();
    }
    //Добавление информации для тестов
    private void addForTest() {
        //<editor-fold desc="Добавление филиалов банка">
        bankBranches.add(new BankBranch("MAIB", "Stefan Cel Mare
16", "+37368773211"));
        bankBranches.add(new BankBranch("MAIB", "Stefan Cel Mare
36", "+3736812321"));
        bankBranches.add(new BankBranch("MAIB", "Ismail
26", "+37368563213"));
        bankBranches.add(new BankBranch("MAIB", "Dacia
47", "+373687950214"));
        bankBranches.add(new BankBranch("MAIB", "Milestiu
22", "+373687123215"));
        bankBranches.add(new BankBranch("MAIB", "Ciuflea
48", "+37368655223"));
        bankBranches.add(new BankBranch("MAIB", "Dacia
116", "+37368224423"));
        bankBranches.add(new BankBranch("MAIB", "Cuza-Voda
116", "+37368223344"));
        bankBranches.add(new BankBranch("MAIB", "Cuza-Voda
11", "+373682233445"));
        bankBranches.add(new BankBranch("MAIB", "Lev Tolstoi
111", "+37368214906"));
        //</editor-fold>

        //<editor-fold desc="Добавление физических клиентов">
        individualClients.add(new IndividualClient("Ulises
Patterson", "Bucuresti 5", "+37368234567", "2004023456789"));
        individualClients.add(new IndividualClient("Thatcher
Jackson", "Stefan cel Mare 12", "+37369345678", "2004067890123"));
        individualClients.add(new IndividualClient("Sterling
Sanchez", "Ion Creanga 27", "+37366456789", "2004012345678"));
        individualClients.add(new IndividualClient("Shepherd Green",
"Mitropolit Grigore Banulescu-Bodoni 14", "+37367987654",
"2004098765432"));
        individualClients.add(new IndividualClient("Rhys Young",
"Mircea cel Batran 33", "+37368876543", "2004054321098"));
    }
}

```

```

        individualClients.add(new IndividualClient("Quinnton Moore",
"Vasile Alecsandri 19", "+37369543210", "2004076543210"));
        individualClients.add(new IndividualClient("Otis Butler",
"Decebal 22", "+37366765432", "2004087654321"));
        individualClients.add(new IndividualClient("Israel Rogers",
"Alecu Russo 9", "+37367432109", "2004032109876"));
        individualClients.add(new IndividualClient("Griffin Watson",
"Cetatea Alba 6", "+37368109876", "2004043210987"));
        individualClients.add(new IndividualClient("Desmond Torres",
"Dacia 8", "+37369098765", "2004009876543"));
    //</editor-fold>

    //<editor-fold desc="Добавление юридических клиентов">
        corporateClients.add(new CorporateClient("GlobalTech
Solutions", "Victoriei 10", "+37368234567", "1021234567", "John
Smith", "Alex Turner"));
        corporateClients.add(new CorporateClient("PrimeFinancial
Group", "Stefan cel Mare 25", "+37369345678", "1029876543", "John
Smith", "Emma Davis"));
        corporateClients.add(new CorporateClient("TechNova
Corporatio", "Mihai Eminescu 7", "+37366456789", "1025432167",
"Michael Brown", "Benjamin Scott"));
        corporateClients.add(new CorporateClient("EliteMarketing
Strategies", "Dacia 14", "+37367987654", "1028765432", "Olivia
Davis", "Olivia Hughes"));
        corporateClients.add(new CorporateClient("PowerTech
Industries", "Primaverii 3", "+37368876543", "1023145678", "Ethan
Wilson", "Ethan Carter"));
        corporateClients.add(new CorporateClient("OptiPro Logistics",
"Decebal 21", "+37369543210", "1027654321", "Sophia Thompson",
"Sophia Brooks"));
        corporateClients.add(new CorporateClient("NovaTech
Solutions", "Ion Creanga 12", "+37366765432", "1022345678",
"William Clark", "William Mitchell"));
        corporateClients.add(new CorporateClient("ApexCorp Holdings",
" Renasterii 9", "+37367432109", "1028765432", "Ava Anderson",
"Ava Reed"));
        corporateClients.add(new CorporateClient("InnovateTech
Solutions", "Unirii 17", "+37368109876", "1027654321", "James
Green", "James Foster"));
        corporateClients.add(new CorporateClient("SecureNet Systems",
"Alexandru cel Bun 6", "+37369098765", "1021234567", "Mia Taylor",
"Mia Collins"));
    //</editor-fold>

    //<editor-fold desc="Добавление кредитов">
        credits.add(new Credit("Mortgage", "Home Loan", "MDL", 3.5));
        credits.add(new Credit("Personal", "Personal Loan", "EUR",
8.2));
        credits.add(new Credit("Auto", "Car Loan", "USD", 5.9));
        credits.add(new Credit("Business", "Small Business Loan",
"EUR", 12.5));
        credits.add(new Credit("Student", "Education Loan", "MDL",

```

```

4.8));
    credits.add(new Credit("Mortgage", "Property Loan", "EUR",
4.1));
    credits.add(new Credit("Personal", "Wedding Loan", "MDL",
9.5));
    credits.add(new Credit("Auto", "Lease Financing", "EUR",
6.4));
    credits.add(new Credit("Business", "Equipment Loan", "MDL",
10.8));
    credits.add( new Credit("Student", "Student Line of Credit",
"EUR", 3.9));
    //</editor-fold>

    //<editor-fold desc="Добавление контрактов">
    contracts.add(new Contract("1", "2023-06-01",
individualClients.get(0), credits.get(0), 250000, 20));
    contracts.add(new Contract("2", "2023-06-02",
individualClients.get(1), credits.get(1), 5000, 3));
    contracts.add(new Contract("3", "2023-06-03",
individualClients.get(2), credits.get(2), 30000, 5));
    contracts.add(new Contract("4", "2023-06-04",
individualClients.get(3), credits.get(3), 1000000, 10));
    contracts.add(new Contract("5", "2023-06-05",
individualClients.get(4), credits.get(4), 15000, 2));
    contracts.add(new Contract("6", "2023-06-06",
corporateClients.get(5), credits.get(5), 200000, 15));
    contracts.add(new Contract("7", "2023-06-07",
corporateClients.get(6), credits.get(6), 10000, 2));
    contracts.add(new Contract("8", "2023-06-08",
corporateClients.get(7), credits.get(7), 40000, 4));
    contracts.add(new Contract("9", "2023-06-09",
corporateClients.get(8), credits.get(8), 500000, 7));
    contracts.add( new Contract("10", "2023-06-10",
corporateClients.get(9), credits.get(9), 20000, 3));

    //</editor-fold>
}

public static void main(String[] args) {
    BankManagementSystem bankManagementSystem = new
BankManagementSystem(); // Создаем экземпляр класса
    bankManagementSystem.addForTest();
    bankManagementSystem.displayMenu(); // Используем
экземпляр для вызова метода displayMenu()
}

// ВЫВОД МЕНЮ
private void displayMenu() {
    int choice = 0;
    do {
        System.out.println("\n\nМеню:");
        System.out.println("1. Добавить/создать");
        System.out.println("2. Вывести");
    }
}

```

```

        System.out.println("3. Рассчитать доход банка");
        System.out.println("4. Конвертировать цену в евро");
        System.out.println("5. Выход");
        System.out.print("Выберите пункт меню: ");

        try {
            choice = Integer.parseInt(scanner.nextLine());

            switch (choice) {
                case 1:
                    System.out.println("1. Добавить филиал
банка");

                    System.out.println("2. Добавить клиента");
                    System.out.println("3. Добавить кредит");
                    System.out.println("4. Заключить
договор");

                    System.out.println("5. Обратно");
                    System.out.print("Выберите пункт меню: ");
                    int subChoice =
Integer.parseInt(scanner.nextLine());
                    System.out.println(subChoice);
                    switch (subChoice) {
                        case 1:
                            addBankBranch();
                            break;

                        case 2:
                            addClient();
                            break;

                        case 3:
                            addCredit();
                            break;

                        case 4:
                            createContract();
                            break;
                        case 5:
                            System.out.println("Возвращаемся
обратно...");

                            break;
                        default:
                            System.out.println("Неверный пункт
меню. Повторите выбор.");

                            break;
                    }
                    break;

                case 2:
                    System.out.println("1. Вывести список
договоров");

                    System.out.println("2. Вывести список
филиалов");

```

```

        System.out.println("3. Вывести список
клиентов");
        System.out.println("4. Вывести список
кредитов");
        System.out.println("5. Обратно");
        System.out.print("Выберите пункт меню: ");
        subChoice =
Integer.parseInt(scanner.nextLine());

        switch (subChoice) {
            case 1:
                displayContracts();
                break;

            case 2:
                displayBankBranches();
                break;

            case 3:
                displayClients(individualClients,
corporateClients);
                break;

            case 4:
                displayCredits();
                break;
            case 5:
                System.out.println("Возвращаемся
обратно...");
                break;
            default:
                System.out.println("Неверный пункт
меню. Повторите выбор.");
                break;
        }
        break;

    case 3:
        calculateBankIncome();
        break;

    case 4:
        convertPriceToEuro();
        break;

    case 5:
        System.out.println("Программа
завершена.");
        break;

    default:
        System.out.println("Неверный пункт меню.
Повторите выбор.");

```

```

        break;
    }
    } catch (NumberFormatException e) {
        System.out.println("Неверный пункт меню. Повторите
выбор.");
    }
    } while (choice != 5);
}

// вывод филиалов банка
@Override
public void displayBankBranches() {
    if(bankBranches.size()>0)
        for (BankBranch branch : bankBranches)
System.out.println(branch.toString());
        else System.out.println("Филиалов нет!");
    }

// вывод клиентов банка
@Override
public void displayClients(List<IndividualClient>
individualClients, List<CorporateClient> corporateClients) {
    if(individualClients.size()>0) {
        System.out.println("Список клиентов:");
        System.out.println("\n\nФизические лица:");
        for (IndividualClient client : individualClients)
System.out.println(individualClients.indexOf(client)+1+" "+
client.toString()+"\n");
    }else System.out.println("\n\nФизических лиц нет!");
    if(corporateClients.size()>0) {
        System.out.println("\n\nЮридические лица:");
        for (CorporateClient client : corporateClients)
System.out.println(corporateClients.indexOf(client)+1+individualCl
ients.size() + ". "+client.toString()+"\n");
    }else System.out.println("\n\nЮридических лиц нет!");
    }

// вывод кредитов
@Override
public void displayCredits() {
    if (credits.size()>0) {
        System.out.println("Список кредитов:");
        for (Credit credit : credits) {
            System.out.println(credits.indexOf(credit) + ". "
+ credit);
        }
    }else System.out.println("\nКредитов нет!\n");
    }

// вывод контрактов
@Override
public void displayContracts() {
    if(contracts.size()>0) {
        System.out.println("Список договоров:");
    }
}

```

```

        for (Contract contract : contracts) {
            System.out.println("Номер договора: " +
contract.getContractNumber());
            System.out.println("Дата договора: " +
contract.getDate());
            System.out.println("Имя клиента: " +
contract.getClient().getName());
            System.out.println("Тип кредита: " +
contract.getCredit().getCreditType());
            System.out.println("Сумма кредита: " +
contract.getTotalLoanAmount());
            System.out.println("Срок погашения кредита: " +
contract.getLoanRepaymentTerm() + " месяцев");
            System.out.println();
        }
    }else System.out.println("\nДоговоров нет!\n");
}
// добавление нового филиала банка
@Override
public void addBankBranch() {
    System.out.println("Добавление филиала банка:");
    System.out.print("Введите имя филиала: ");
    String name = scanner.nextLine();
    System.out.print("Введите адрес филиала: ");
    String address = scanner.nextLine();
    System.out.print("Введите телефон филиала: ");
    String phoneNumber = scanner.nextLine();

    BankBranch bankBranch = new BankBranch(name, address,
phoneNumber);
    bankBranches.add(bankBranch);
    // Добавление филиала в список или другую структуру данных
    System.out.println("Филиал банка добавлен.");
}
// добавление нового юридического лица
public void addLegalClient(String name, String address, String
phoneNumber) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Введите идентификационный номер
налогоплательщика: ");
    String taxIdentificationNumber = scanner.nextLine();

    System.out.print("Введите имя администратора: ");
    String administratorName = scanner.nextLine();

    System.out.print("Введите имя контактного лица: ");
    String contactPerson = scanner.nextLine();

    CorporateClient client = new CorporateClient(name,
address, phoneNumber, taxIdentificationNumber, administratorName,
contactPerson);
    corporateClients.add(client);
}

```

```

        System.out.println("Юридический клиент добавлен.");
    }
    // добавление нового физического лица
    public void addIndividualClient(String name, String address,
String phoneNumber) {
        System.out.print("Введите личный код: ");
        String personalCode = scanner.nextLine();

        IndividualClient client = new IndividualClient(name,
address, phoneNumber, personalCode);
        individualClients.add(client);
        System.out.println("Физический клиент добавлен.");
    }

    // общая функция добавления клиента
    @Override
    public void addClient() {

        System.out.println("Выберите тип клиента:");
        System.out.println("1. Физическое лицо");
        System.out.println("2. Юридическое лицо");
        int choice = 0;
        try {
            choice = scanner.nextInt();
            scanner.nextLine();
        } catch (InputMismatchException e) {
            System.out.println("Введены некорректные данные.
Пожалуйста, введите данные в правильном формате.");
            scanner.nextLine(); // очистка буфера
            return;
        }

        System.out.println("Введите имя:");
        String name = scanner.nextLine();

        System.out.println("Введите адрес:");
        String address = scanner.nextLine();

        System.out.println("Введите номер телефона:");
        String phoneNumber = scanner.nextLine();

        if (choice == 1) addIndividualClient(name, address,
phoneNumber);
        else if (choice == 2) addLegalClient(name, address,
phoneNumber);
        else System.out.println("Неверный выбор типа клиента.");
    }

    // добавление нового кредита
    @Override
    public void addCredit() {
        System.out.println("Добавление кредита:");
        System.out.print("Введите тип кредита: ");
        String creditType = scanner.nextLine();
    }

```



```

        System.out.print("Введите наименование кредита: ");
        String name = scanner.nextLine();
        System.out.print("Введите валюту кредита: ");
        String currency = scanner.nextLine();
        System.out.print("Введите годовую процентную ставку
кредита: ");
        double annualInterestRate;
        try {
            annualInterestRate = scanner.nextDouble();
            scanner.nextLine(); // Очистка буфера
        } catch (InputMismatchException e) {
            System.out.println("Введены некорректные данные.
Пожалуйста, введите данные в правильном формате.");
            return;
        }
        Credit credit = new Credit(creditType, name, currency,
annualInterestRate);
        credits.add(credit);
    }
    // создание нового контракта
    @Override
    public void createContract() {
        try {
            System.out.println("Заключение договора:");

            System.out.print("Введите номер контракта: ");
            String contractNumber = scanner.nextLine();
            System.out.print("Введите дату контракта dd.mm.yyyy:
");

            String date = scanner.nextLine();

            System.out.println("Выберите клиента:");
            displayClients(individualClients, corporateClients);
            int clientChoice = scanner.nextInt();
            scanner.nextLine(); // Очистка буфера

            Client client;
            if (clientChoice > 0 && clientChoice <=
individualClients.size()) {
                client = individualClients.get(clientChoice - 1);
            } else if (clientChoice > individualClients.size() &&
clientChoice <= individualClients.size() +
corporateClients.size()) {
                client = corporateClients.get(clientChoice -
individualClients.size() - 1);
            } else {
                System.out.println("Неверный выбор клиента.");
                return;
            }

            System.out.println("Выберите кредит:");
            displayCredits();
            int creditChoice = scanner.nextInt();

```

```

        scanner.nextLine(); // Очистка буфера

        Credit credit;
        if (creditChoice > 0 && creditChoice <=
credits.size()) {
            credit = credits.get(creditChoice - 1);
        } else {
            System.out.println("Неверный выбор кредита.");
            return;
        }

        System.out.print("Введите общую сумму займа: ");
        double totalLoanAmount = scanner.nextDouble();
        scanner.nextLine(); // Очистка буфера
        System.out.print("Введите срок погашения займа: ");
        int loanRepaymentTerm = scanner.nextInt();
        scanner.nextLine(); // Очистка буфера

        Contract contract = new Contract(contractNumber, date,
client, credit, totalLoanAmount, loanRepaymentTerm);
        contracts.add(contract);
        System.out.println("Договор заключен.");
    } catch (InputMismatchException e) {
        System.out.println("Введены некорректные данные.
Пожалуйста, введите данные в правильном формате.");
    } catch (NoSuchElementException e) {
        System.out.println("Не удалось считать ввод.
Пожалуйста, введите данные корректно.");
    }
}

// расчет дохода банка
@Override
public void calculateBankIncome() {
    System.out.println("Расчет дохода банка:");
    if (contracts == null) {throw new
IllegalArgumentException("Контрактов нет!");}

    System.out.print("Введите номер договора: ");
    String contractNumber = scanner.nextLine();
    // Найти договор по номеру
    Contract contract = findContractByNumber(contractNumber);

    if (contract != null) {
        contract.calculateIncome();
    } else {
        System.out.println("Договор не найден.");
    }
}

// поиск контракта по номеру
private static Contract findContractByNumber(String
contractNumber) {
    if (contractNumber == null) {throw new

```

```

IllegalArgumentException("Номер контракта не может быть [" +
contractNumber+ "]);}

        for (Contract contract : contracts) {
            if
(contract.getContractNumber().equals(contractNumber)) {
                return contract;
            }
        }
        return null;
    }
    // перевод цены из леев в евро
    @Override
    public void convertPriceToEuro() {
        System.out.println("Конвертация цены в евро:");
        try {
            System.out.print("Введите текущую цену в лей: ");
            double price = scanner.nextDouble();
            System.out.print("Введите курс (1 евро в лях): ");
            double rate = scanner.nextDouble();
            if(rate == 0) throw new ArithmeticException();
            double euroPrice = price / rate;
            DecimalFormat df = new DecimalFormat("#.00");
            System.out.println("Цена в евро: " +
df.format(euroPrice));
        }catch (InputMismatchException e){
            System.out.println("\nОшибка ввода. Пожалуйста,
введите числовое значение.\n");
        }catch (ArithmeticException e){
            System.out.println("\nОшибка(деление на ноль):" +
e.getMessage() + "\n");
        }finally {
            scanner.nextLine(); // Очистка буфера
        }
    }
}

```