МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп'ютерні інформаційні технології»

**Лабораторна робота №4**

**з дисципліни «Основи програмної інженерії»**

**на тему: «**Класи та об’єкти.**»**

Виконав:

студент гр.ПЗ1911

Сафонов Д.Є.

Прийняла:

Куроп'ятник О. С.

Дніпро, 2020

**Тема.** Класи та об’єкти

**Мета.** Ознайомитися з основними принципами та поняттями об’єктноорієнтованого програмування та отримати практичні навички розробки об’єктно-орієнтованих програм з використанням принципу інкапсуляції.

# **Постановка задачі згідно загального та індивідуального завдання.**

Виконати розробку програми згідно індивідуального завдання лабораторної роботи № 3.

Вимоги до програми:

− наявність щонайменше трьох класів: матриці, роботи з файлом, інтерфейсу користувача;

− передбачити заповнення матриці з клавіатури, генератором випадкових чисел у заданому користувачем діапазоні, з файлу;

− глобальні змінні не допускаються;

− контроль значень вхідних даних (перевірка діапазону значень для чисел, коректна реакція на введення різних символів тощо).

Відповідальність класу матриці: зберігання матриці у ОП та її обробка (обробка може бути винесена у окремий клас).

Відповідальність класу роботи з файлом: зчитування даних для ініціалізації матриці, збереження результатів роботи програми.

Відповідальність класу інтерфейсу користувача: введення та виведення інформації.

Вимоги до тексту програми:

− коментарі щодо призначення кожного методу, його входу та виходу;

− самодокументованість коду: всі ідентифікатори повинні мати назви, що відповідають суті змінної або дії, яку вони позначають. Перевага надається англомовним назвам;

− єдині стилі іменування змінних та функцій.

Дано символьну матрицю 10\*10, яка є полем гри «Морський бій». Згенерувати випадкове розташування кораблів, яке відповідає таким правилам:

− кораблі: 4 палуби – 1 шт., 3 палуби – 2 шт., 2 – 3 шт., 1 – 4 шт.;

− кораблі можуть розташовуватися лише по прямій;

− між кораблями має бути хоча б одна вільна клітинка;

− кораблі не можуть доторкатися один одного по діагоналі.

# **Текст програми.**

“main.cpp”

#include "battleshipsBoard.h"

#include "UI.h"

int main() {

battleshipsBoard bboard;

UI::menu(bboard);

return 0;

}

“battleshipsBoard.h”

#ifndef \_\_BATTLESHIPSBOARD\_H\_\_

#define \_\_BATTLESHIPSBOARD\_H\_\_

#include <random>

#include <chrono>

#include "UI.h"

#include "fileHandle.h"

enum seaField { empty, shipBody, bodyHit, miss, shipNeighbour };

enum direction { horizontal, vertical };

struct point {

int x;

int y;

};

class battleshipsBoard{

private:

seaField\*\* board;

//generates, checks and places x-body ship in this object

void genXbody(int Xbodies);

public:

battleshipsBoard();

~battleshipsBoard();

//checks if given ship can be placed in this object

bool validateShip(point start, direction dir, int length);

//places ship(use validation before!)

void placeShip(point start, direction dir, int length);

//generates board

void genBoard();

friend class UI;

friend class fileHandle;

};

#endif

“battleshipsBoard.cpp”

#include "battleshipsBoard.h"

bool battleshipsBoard::validateShip(point start, direction dir, int length) {

bool usable = true;

if (dir == horizontal)

for (int i = 0; i < length && usable; i++)

if (start.y + i < 10)

usable = (this->board[start.x][start.y + i] == empty);

else

usable = false;

else//vertical

for (int i = 0; i < length && usable; i++)

if (start.x + i < 10)

usable = (this->board[start.x + i][start.y] == empty);

else

usable = false;

return usable;

}

void battleshipsBoard::placeShip(point start, direction dir, int length) {

if (dir == horizontal) {

for (int i = 0; i < length; i++)

this->board[start.x][start.y + i] = shipBody;

if (start.y - 1 >= 0)

this->board[start.x][start.y - 1] = shipNeighbour;

if (start.y + length < 10)

this->board[start.x][start.y + length] = shipNeighbour;

if (start.x - 1 >= 0) {

for (int i = 0; i < length; i++)

this->board[start.x - 1][start.y + i] = shipNeighbour;

if (start.y - 1 >= 0)

this->board[start.x - 1][start.y - 1] = shipNeighbour;

if (start.y + length < 10)

this->board[start.x - 1][start.y + length] = shipNeighbour;

}

if (start.x + 1 < 10) {

for (int i = 0; i < length; i++)

this->board[start.x + 1][start.y + i] = shipNeighbour;

if (start.y - 1 >= 0)

this->board[start.x + 1][start.y - 1] = shipNeighbour;

if (start.y + length < 10)

this->board[start.x + 1][start.y + length] = shipNeighbour;

}

}//

else {//vertical

for (int i = 0; i < length; i++)

this->board[start.x + i][start.y] = shipBody;

if (start.x - 1 >= 0)

this->board[start.x - 1][start.y] = shipNeighbour;

if (start.x + length < 10)

this->board[start.x + length][start.y] = shipNeighbour;

if (start.y - 1 >= 0) {

for (int i = 0; i < length; i++)

this->board[start.x + i][start.y - 1] = shipNeighbour;

if (start.x - 1 >= 0)

this->board[start.x - 1][start.y - 1] = shipNeighbour;

if (start.x + length < 10)

this->board[start.x + length][start.y - 1] = shipNeighbour;

}

if (start.y + 1 < 10) {

for (int i = 0; i < length; i++)

this->board[start.x + i][start.y + 1] = shipNeighbour;

if (start.x - 1 >= 0)

this->board[start.x - 1][start.y + 1] = shipNeighbour;

if (start.x + length < 10)

this->board[start.x + length][start.y + 1] = shipNeighbour;

}

}

}

void battleshipsBoard::genBoard() {

for (int i = 0; i < 10; i++)

for (int j = 0; j < 10; j++)

this->board[i][j] = empty;

genXbody(4);

for (int i = 0; i < 2; i++)

genXbody(3);

for (int i = 0; i < 3; i++)

genXbody(2);

for (int i = 0; i < 4; i++)

genXbody(1);

}

void battleshipsBoard::genXbody(int Xbodies) {//Xbodies supposed to be 1-4

std::uniform\_int\_distribution<int> XYdistribution(0, 9);

std::uniform\_int\_distribution<int> directioDistribution(0, 1);

std::default\_random\_engine generator(time(NULL));

point start;

direction dir;

bool gen = true;

do {

start.x = XYdistribution(generator);

start.y = XYdistribution(generator);

dir = static\_cast<direction>(directioDistribution(generator));

gen = validateShip(start, dir, Xbodies);

} while (!gen);

placeShip(start, dir, Xbodies);

}

battleshipsBoard::battleshipsBoard() {

this->board = new seaField \* [10];

for (int i = 0; i < 10; i++)

board[i] = new seaField[10];

}

battleshipsBoard::~battleshipsBoard() {

for (int i = 0; i < 10; i++)

delete[]board[i];

delete[]board;

}

“UI.h”

#ifndef \_\_UI\_H\_\_

#define \_\_UI\_H\_\_

#include <iostream>

#include <Windows.h>

#include <fstream>

#include "inp\_val7.h"

class battleshipsBoard;//in main or etc include UI then battleshipsBoard

class UI{

static void setShip(battleshipsBoard& bboard, int Xbodies);//from user input

public:

static void printBoard(battleshipsBoard& bboard);

static void setBoard(battleshipsBoard& bboard);//from user inputs

static void menu(battleshipsBoard& bboard);

};

#endif

“UI.cpp”

#include "UI.h"

#include "battleshipsBoard.h"

#include "inp\_val7.h"

void UI::printBoard(battleshipsBoard& bboard) {

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

std::cout << " A B C D E F G H I J" << std::endl;

for (int i = 0; i < 10; i++) {

std::cout << i;

for (int j = 0; j < 10; j++)

if (bboard.board[i][j] == shipBody) {

SetConsoleTextAttribute(hConsole, 127);

std::cout << "[]";

SetConsoleTextAttribute(hConsole, 15);

}

else

std::cout << "[]";

std::cout << std::endl;

}

}

void UI::setBoard(battleshipsBoard& bboard) {

for (int i = 0; i < 10; i++)

for (int j = 0; j < 10; j++)

bboard.board[i][j] = empty;//set all to empty

for (int Xbodies = 4; Xbodies > 0; Xbodies--)

for (int Xships = 5 - Xbodies; Xships > 0; Xships--) {

system("cls");

UI::printBoard(bboard);

UI::setShip(bboard, Xbodies);

}

}

void UI::setShip(battleshipsBoard& bboard, int Xbodies) {

point start;

direction dir = horizontal;//for 1body

bool gen = true;

do {

std::cout << "placing " << Xbodies << "-body ship" << std::endl;

std::cout << "input coordinates of start of the ship(use correct start because direction can be only right/down)" << std::endl;

start.x = cinum("input row:", "wrong input, try again", 1, 1, 0, 9);

char y;

std::cout << "input column: ";

std::cin >> y;

while (!((y >= 'A' && y <= 'J') || (y >= 'a' && y <= 'j')) || (std::cin.peek() != '\n')) {//until correct input

while (std::cin.get() != '\n');//clear istream

std::cout << "wrong input, try again." << std::endl;

std::cout << "input column: ";

std::cin >> y;

}

//std::cin.get();//enter

start.y = toupper(y) - 65;//map(map(y, A\_J, a\_j), A\_J, 0\_9)

if (Xbodies != 1) {//dont ask if 1body

bool dirInt;

dirInt = cinum("input direction(0 - right, 1 - down):", "wrong input, try again.", 1, 1, 0, 1);

dir = static\_cast<direction>(dirInt);

}

gen = bboard.validateShip(start, dir, Xbodies);

if (!gen)

std::cout << "wrong input, try again" << std::endl;

} while (!gen);

bboard.placeShip(start, dir, Xbodies);

}

void UI::menu(battleshipsBoard& bboard) {

bool boardExists = false;//let !exists on first turn

int menuChoice;

do {

system("cls");

if (boardExists) {

UI::printBoard(bboard);

}

std::cout << "1. Generate board." << std::endl;

if(boardExists)

std::cout << "2. Save this board to file." << std::endl;

std::cout << "3. Load board from file." << std::endl;

std::cout << "4. Set the board." << std::endl;

std::cout << "5. Exit." << std::endl;

menuChoice = cinum("Choose what to do:", "Wrong input, try again", 1, 1, 1, 5);

switch (menuChoice) {

case 1:

bboard.genBoard();

boardExists = true;

break;

case 2:

if (boardExists) {

std::ofstream binarySave;

char fileOutName[256];

errno\_t err = tmpnam\_s(fileOutName, 255);

binarySave.open(fileOutName, std::ofstream::binary);

fileHandle::writeBoard(bboard, binarySave);

std::cout << "board is saved in " << fileOutName << " file" << std::endl;

system("pause");

}

break;

case 3: {

std::ifstream binarySave;

char fileInName[256];

do {

std::fill\_n(fileInName, 256, 0);//0 = '\0'

std::cout << "input address of file(or only name if it is in the same folder as programm) starting with disk:\\ and ending with \*.txt: " << std::endl;

std::cin.get();//enter

std::cin.getline(fileInName, 255);

binarySave.open(fileInName, std::ifstream::binary);

if (!binarySave.is\_open())

std::cout << "Can't open that file!" << std::endl;

} while (!binarySave.is\_open());

fileHandle::readBoard(bboard, binarySave);

boardExists = true;

break;

}

case 4:

UI::setBoard(bboard);

boardExists = true;

break;

}

} while (menuChoice != 5);

}

“fileHandle.h”

#ifndef \_\_FILEHANDLE\_H\_\_

#define \_\_FILEHANDLE\_H\_\_

#include <fstream>

class battleshipsBoard;

class fileHandle{

public:

static void readBoard(battleshipsBoard& bboard, std::ifstream& file);//binary

static void writeBoard(battleshipsBoard& bboard, std::ofstream& file);//binary

};

#endif

“fileHandle.cpp”

#include "fileHandle.h"

#include "battleshipsBoard.h"

void fileHandle::readBoard(battleshipsBoard& bboard, std::ifstream& file) {

seaField temp[100];

file.read(reinterpret\_cast<char\*>(temp), sizeof(temp));

for (int i = 0; i < 10; i++)

for (int j = 0; j < 10; j++)

bboard.board[i][j] = static\_cast<seaField>(temp[i \* 10 + j]);

}

void fileHandle::writeBoard(battleshipsBoard& bboard, std::ofstream& file) {

seaField temp[100];

for (int i = 0; i < 10; i++)

for (int j = 0; j < 10; j++)

temp[i \* 10 + j] = bboard.board[i][j];

file.write(reinterpret\_cast<char\*>(temp), sizeof(temp));

}

“inp\_val7.h”

//cinum - numerical only (R)

//char msg\_invite - your message for inviting user to input their number

//char msg\_error - if error input occures

//int m\_ - check if input is multiple of this number

//bool intFlag - =1(only integer input); =0(any input)

//double left - leftmost point in diapasone for input

//double right - rightmost point in diapasone for input

#include <iostream>

#include <windows.h>

#undef max

#include <cmath>

#include <climits>

int cinum(const char\* msg\_invite, const char\* msg\_error, int m\_, bool intFlag, float left, float right);

//input to char array with lenght validation

void val\_char\_lenght(char array[], int ary\_size);

“inp\_val7.cpp”

#include "inp\_val7.h"

using namespace std;

int cinum(const char\* msg\_invite, const char\* msg\_error, int m\_ = 1, bool intFlag = 0, float left = -2147483648.0, float right = 2147483647.0) {

bool i = 0;

float aaa;

cout << msg\_invite << endl;

do {

cin >> aaa;

int iaaa;

iaaa = (int)aaa;

if (cin.fail() || (aaa < left) || (aaa > right) || ((intFlag == 1) && (ceil(aaa) != floor(aaa))) || ((m\_ != 1) && ((iaaa % m\_) != 0))) {

cout << msg\_error << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

else

i = 1;

} while (!i);

return (int)aaa;

}

void val\_char\_lenght(char array[], int ary\_size) {//input to char array with lenght validation

bool correct = false;

do {

std::cin.getline(array, 24);

if ((strlen(array) > 23) || cin.fail()) {

std::cout << "wrong input, try again" << std::endl;

cin.clear();

std::cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

else

correct = true;

} while (!correct);

}

# **Аналіз результатів та висновки щодо переваг та недоліків об’єктноорієнтованого програмування. Визначення у чому полягає суть абстракції та інкапсуляції у виконаній роботі. Пропозиції щодо покращення організації коду (інші доречні варіанти розбиття на класи, зменшення зв’язності класів, розподіл задач у методах тощо).**



Рисунок 1(Меню одразу після запуску програми)



Рисунок 2(Сгенерована матриця)



Рисунок 3(Сгенерована інша матриця) 

Рисунок 4(Збереження матриці у файлі)

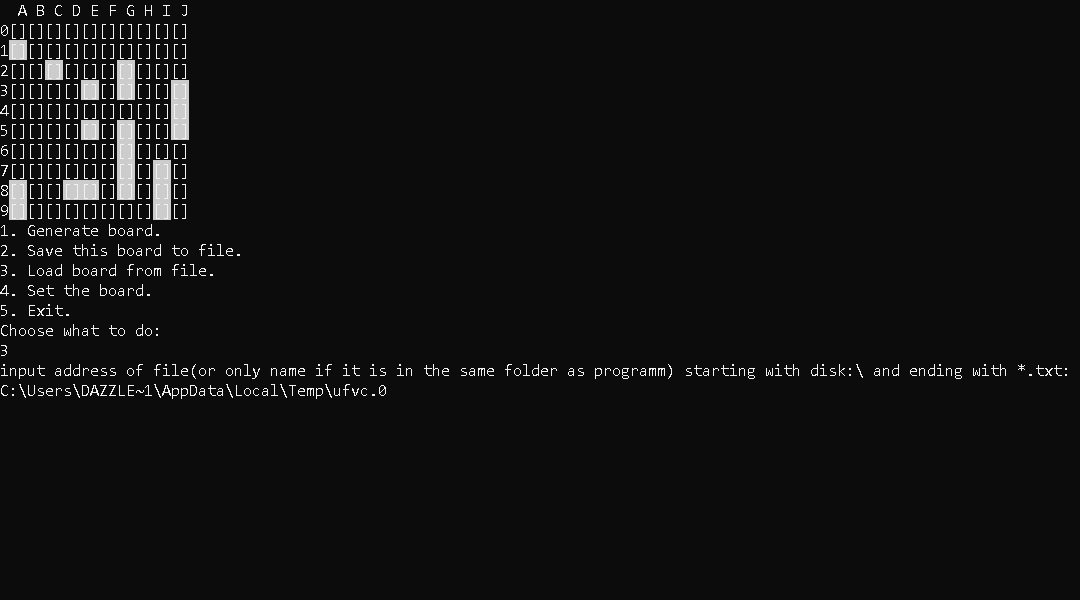


Рисунок 5(Зчитування матриці з файлу)

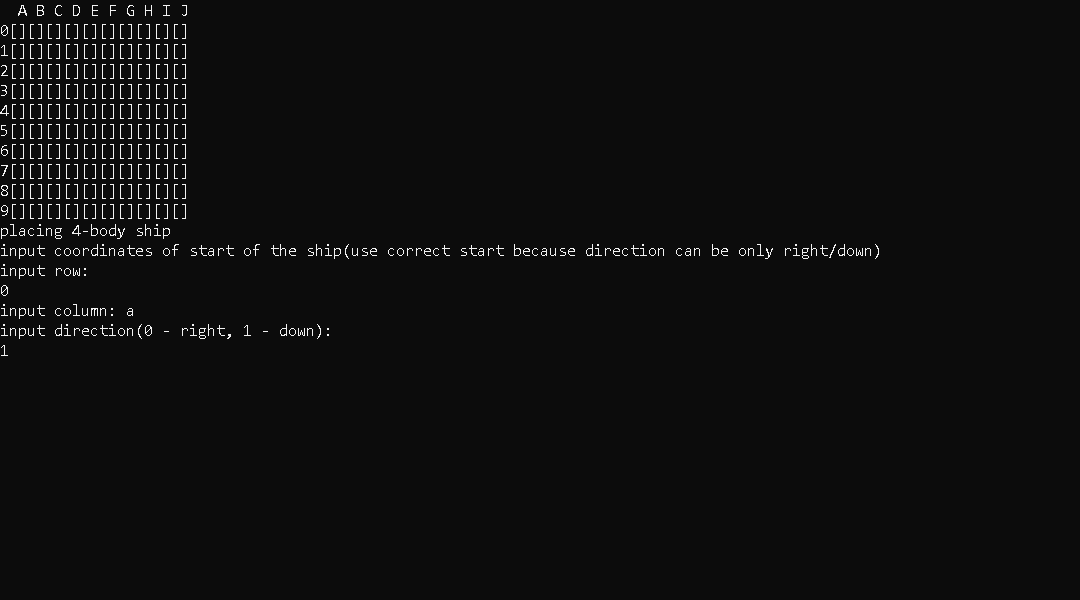


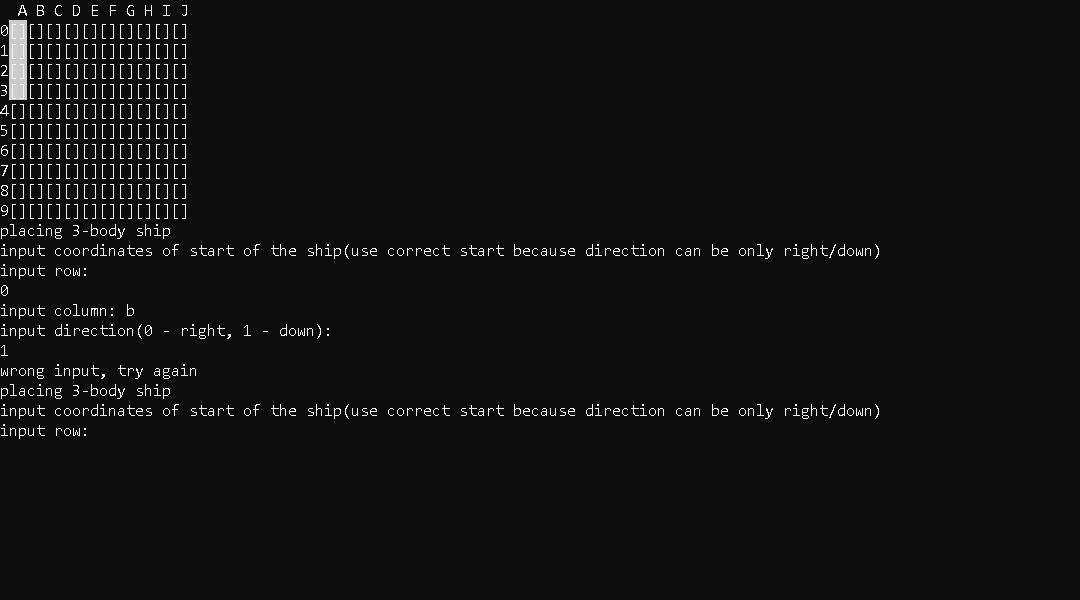
Рисунок 6(Ручне заповнення матриці) 

Рисунок 7(Некоректні дані при ручному заповнені матриці)

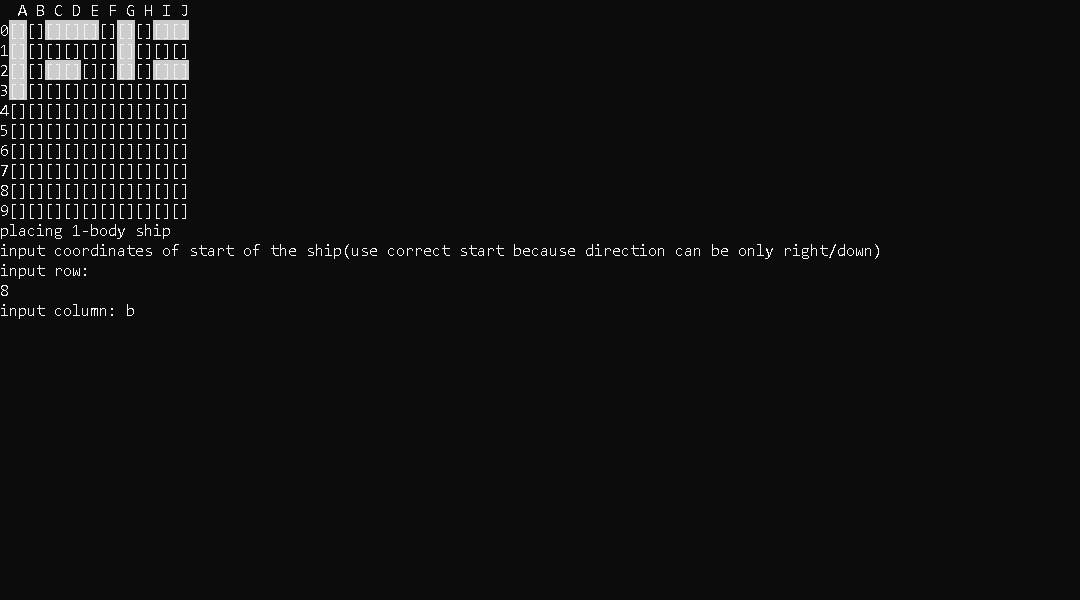


Рисунок 8(При додавані однопалубних не потрібно вказувати напрямок)



Рисунок 9(Результат ручного заповненя)

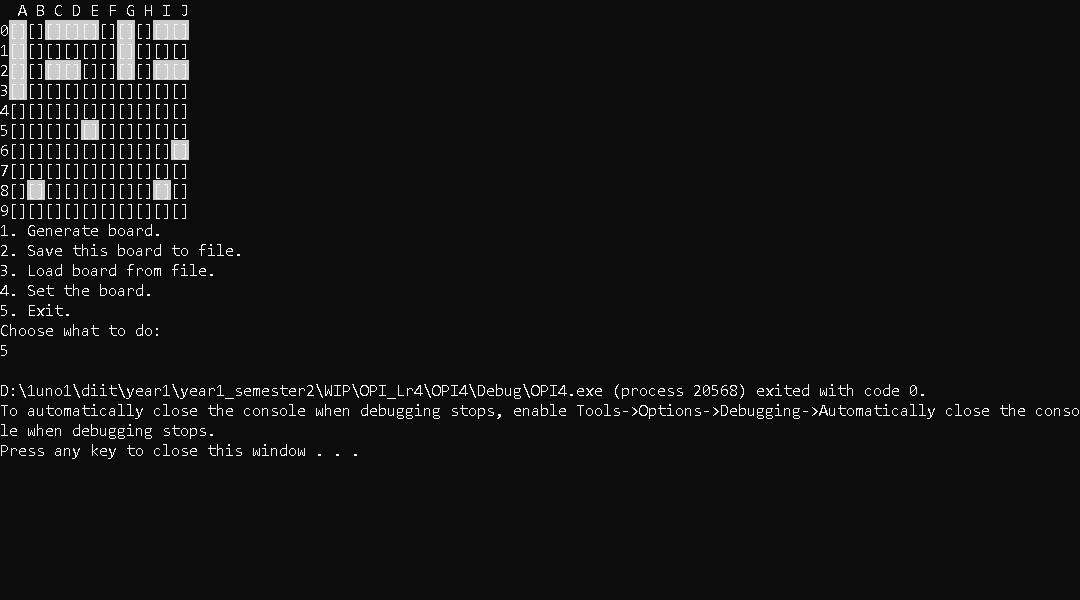


Рисунок 10(Завершення програми)

Абстракція – представлення класу найменшою кількістю полей, достатньою для вирішення конкретної задачі.

Інкапсуляція – розміщення в класі методів які з ним працюють, приховування реалізації цих методів, приховування деяких змінних та методів.

Найбільша перевага ООП – розбиття даних та функцій на класи з якими вони працюють, за рахунок цього набагато простіше розбиратися із кодом. Але через це ж додавати новий функціонал дуже складно, бо зазвичай розбиття робиться під деякий фіксований функціонал, бо передбачити об’єми розширення майже неможливо. З першої переваги витікає захищеність даних. А з цього витікає новий недолік – погіршення швидкості. У розробленій програмі можна було розбити функцію меню на меню та сесію, щоб сесію визивала меню в циклі.