**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Лабораторная работа № 1

по ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

«Объекты и классы. Инкапсуляция»

Студент \_\_\_\_\_\_\_\_\_\_\_ Целищев А.Е.

Группа ПМ-21-2

Руководитель

Доцент \_\_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Липецк 2022г.

**Цель работы:**

Научиться описывать объекты реального мира с использованием классов.

**Задание кафедры:**

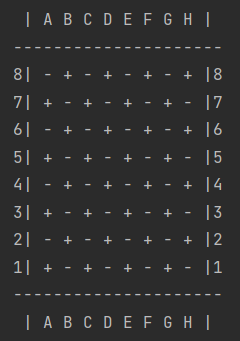
Используя механизм инкапсуляции, описать объект реального мира. Добавить к имеющимся в задании атрибутам и методам ещё по три произвольных пункта.

Методы и свойства объектов реального мира

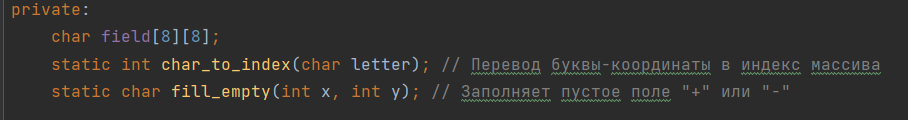
|  |  |  |
| --- | --- | --- |
| Вариант | Объект | Методы |
| 1 | Шахматная доска | Поставить фигуру Х на позицию Y  Переместить фигуру с позиции Х на позицию Y |

**Ход работы:**

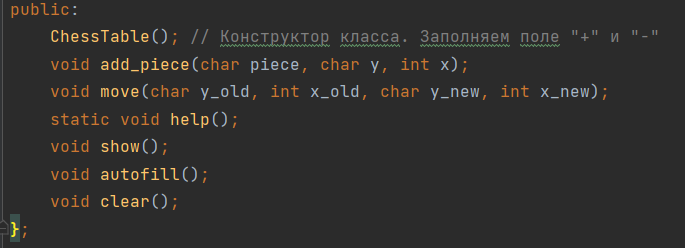
Для определения игрового поля используется двумерный массив 8х8 типа *char*. Пустые ячейки поля заполняются символами «+» и «-», соответствующими чёрным и белым клеткам:



Определение игрового поля, а также два метода - для перевода символьной координаты в индекс массива и для заполнения пустых ячеек «+» или «-» - были помещены в блок закрытым доступом, так как эти методы нужны только для промежуточных вычислений в публичных методах, а массив поля скрыт во избежание ошибок вследствие обращений к нему вне класса и объекта.

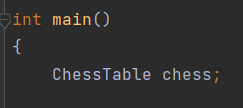


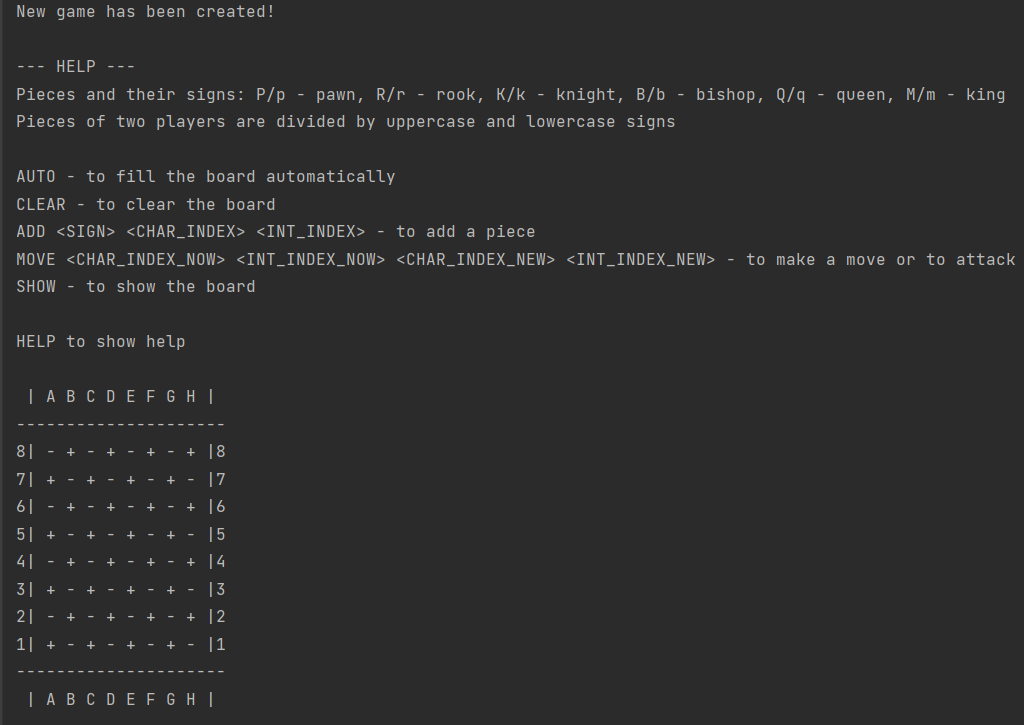
Общий доступимеют следующие функции: поставить фигуру Х на ячейку Y, сделать ход с ячейки X на ячейку Y, автоматически расставить фигуры, показать игровое поле в данный момент, очистить поле от всех фигур. Этих методов достаточно, чтобы пользователь мог полноценно взаимодействовать с классом. Здесь же определён конструктор, инициализирующий новое поле при создании нового объекта класса.



***Пример работы программы***

1. Создаём объект. При создании срабатывает конструктор.





1. Используем автоматическое заполнение поля и выведем результат.

|  |  |
| --- | --- |
|  |  |

1. Очистим поле и выведем результат.

|  |  |
| --- | --- |
|  |  |

1. Расставим несколько фигур и обыграем некоторую ситуацию.

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

***Код программы***

**<ChessTable.h>**

#ifndef DECISIONS\_CHESSTABLE\_H#define DECISIONS\_CHESSTABLE\_H

#endif //DECISIONS\_CHESSTABLE\_H

#include <iostream>

**using** **namespace** std;

// ChessTable - задает шахматную доску с возможностью// добавлять фигуры вручную либо заполнять поле автоматически,// делать ходы и атаковать фигуры противника.

**class** ChessTable

{

**private**:

**char** field[8][8];

**static** **int** char\_to\_index(**char** letter); // Перевод буквы-координаты в индекс массива

**static** **char** fill\_empty(**int** x, **int** y); // Заполняет пустое поле "+" или "-"

**public**:

ChessTable(); // Конструктор класса. Заполняем поле "+" и "-"

**void** add\_piece(**char** piece, **char** y, **int** x);

**void** move(**char** y\_old, **int** x\_old, **char** y\_new, **int** x\_new);

**static** **void** help();

**void** show();

**void** autofill();

**void** clear();

};

**<ChessTable.cpp>**

#include "ChessTable.h"

**int** ChessTable::char\_to\_index(**char** letter)

{

**return** (**int**)letter - 65; // Используем код буквы в ASCII и вычитаем код буквы "А". Тогда А -> 0, B -> 1...

}

**char** ChessTable::fill\_empty(**int** x, **int** y)

{

**if** ((x + y) % 2 == 0) **return** '-'; // Сумма координат чётная => клетка "белая", иначе - "чёрная"

**else** **return** '+';

}

ChessTable::ChessTable() // Заполняем поле "+" и "-" по указанному в fill\_empty правилу

{

**for** (**int** i = 0; i < 8; i++)

{

**for** (**int** j = 0; j < 8; j++)

{

**if** ((i + j) % 2 == 0) field[i][j] = '-';

**else** field[i][j] = '+';

}

}

// Для удобства пользователя сразу выводим помощь и показываем поле

cout << "New game has been created!\n";

help();

show();

}

// Добавляет фигуру. Параметры: буква-код фигуры, буква-координата, цифра-координата

**void** ChessTable::add\_piece(**char** piece, **char** y, **int** x)

{

field[8-x][char\_to\_index(y)] = piece;

cout << "The piece " << piece << " is added to the square " << y << x << endl;

}

// Ход фигурой со "старых" координат на "новые". Если на "новой" клетке противник - бьёт его.

**void** ChessTable::move(**char** y\_old, **int** x\_old, **char** y\_new, **int** x\_new)

{

**if**(field[8-x\_old][char\_to\_index(y\_old)] != '-' && field[8-x\_old][char\_to\_index(y\_old)] != '+') // Если "старая" клетка занята...

{

**if**(field[8-x\_new][char\_to\_index(y\_new)] != '-' && field[8-x\_new][char\_to\_index(y\_new)] != '+') // Если "новая" клетка занята...

{

**char** killed = field[8-x\_new][char\_to\_index(y\_new)];

field[8-x\_new][char\_to\_index(y\_new)] = field[8-x\_old][char\_to\_index(y\_old)];

field[8-x\_old][char\_to\_index(y\_old)] = fill\_empty(8-x\_old, char\_to\_index(y\_old));

cout << "The piece " << field[8-x\_new][char\_to\_index(y\_new)] << " beats a piece " << killed << endl;

}

**else**

{

field[8-x\_new][char\_to\_index(y\_new)] = field[8-x\_old][char\_to\_index(y\_old)];

field[8-x\_old][char\_to\_index(y\_old)] = fill\_empty(8-x\_old, char\_to\_index(y\_old));

cout << "The piece " << field[8-x\_new][char\_to\_index(y\_new)] << " makes a move: " << x\_old << y\_old << " -> " << x\_new << y\_new << endl;

}

}

**else** cout << "Selected square is empty!" << endl;

}

**void** ChessTable::help()

{

cout << "\n--- HELP ---\n";

cout << "Pieces and their signs: P/p - pawn, R/r - rook, K/k - knight, B/b - bishop, Q/q - queen, M/m - king\n";

cout << "Pieces of two players are divided by uppercase and lowercase signs\n\n";

cout << ".autofill() - to fill the board automatically\n";

cout << ".clear() - to clear the board\n";

cout << ".add\_piece(SIGN, CHAR\_INDEX, INT\_INDEX) - to add a piece\n";

cout << ".move(CHAR\_INDEX\_NOW,INT\_INDEX\_NOW,CHAR\_INDEX\_NEW,INT\_INDEX\_NEW) - to make a move or to attack\n";

cout << ".show() - to show the board\n\n";

cout << ".help() to show help\n";

}

**void** ChessTable::show() // Выводит игровое поле на данный момент

{

cout << "\n | A B C D E F G H |\n---------------------\n";

**for** (**int** i = 0; i < 8; i++)

{

cout << 8-i << "| ";

**for** (**int** j = 0; j < 8; j++)

{

cout << field[i][j] << ' ';

}

cout << '|' << 8-i << '\n';

}

cout << "---------------------\n | A B C D E F G H |\n\n";

}

**void** ChessTable::autofill() // Автозаполнение поля для игры в шахматы

{

**for** (**int** i = 0; i < 8; i++)

{

field[1][i] = 'p';

field[6][i] = 'P';

}

field[0][0] = 'r';

field[0][7] = 'r';

field[7][0] = 'R';

field[7][7] = 'R';

field[0][1] = 'k';

field[0][6] = 'k';

field[7][1] = 'K';

field[7][6] = 'K';

field[0][2] = 'b';

field[0][5] = 'b';

field[7][2] = 'B';

field[7][5] = 'B';

field[0][3] = 'q';

field[7][3] = 'Q';

field[0][4] = 'm';

field[7][4] = 'M';

cout << "Autofill completed!\n";

}

**void** ChessTable::clear() // Очистить игровое поле от всех фигур

{

**for** (**int** i = 0; i < 8; i++)

{

**for** (**int** j = 0; j < 8; j++)

{

field[i][j] = fill\_empty(i, j);

}

}

cout << "\nThe field has been cleared!\n";

}

***Вывод:***

При помощи механизма инкапсуляции был описан объект реального мира (шахматная доска) и реализованы методы работы с ним.

***Контрольные вопросы***

1. **Что такое класс?**

*Класс* - это абстрактная модель, описывающая структуру, характеристики, поведение экземпляров (объектов) класса и определяющая методы для работы с ними.

1. **Чем отличается экземпляр класса от класса?**

*Класс* - это общая абстрактная модель, задающая структуру и поведение для всех *экземпляров* данного класса, которые работают по заданным их классом правилам.

1. **Что такое свойство и метод класса?**

*Свойство* - это данные, которые как-либо характеризуют класс.

*Метод* - это набор команд, выполняющих действия над объектами класса. Равносильно функции.

1. **Какие существуют уровни доступа к свойствам и методам класса?**

*Public* - к свойствам/методам можно обратиться вне определения класса. Например, к public-методам и свойствам относятся те, которыми программист может свободно оперировать в своей программе.

*Protected* - к свойствам/методам может обратиться определяемый класс и его наследники в своих методах.

*Private* - к свойствам/методам может обратиться только сам класс в своём определении. Например, здесь могут храниться методы, которые нужны для промежуточных вычислений в public-методах, которые имеет скрыть от пользователя, т.е. не позволить ему воспользоваться ими отдельно.

1. **Что означает понятие инкапсуляция?**

*Инкапсуляция -* объединение в объекте его свойств (характеристик) и методов (способов изменения значения характеристик), сопровождающееся сокрытием от пользователя деталей реализации.