Содержание

2. Задание

3. Цель

4. Код программы

5. Вывод

2. Задание

Используя механизм инкапсуляции, описать объект реального мира. Добавить к имеющимся в задании атрибутам и методам еще по 3 произвольных пункта.

Вариант 1.

Объект: Шахматная доска.

Методы: 1) Поставить фигуру X на позицию Y.

2) Переместить фигуру с позиции X на позицию Y.

3) Удалить фигуру с позиции X.

4) Найти короля на доске.

5) Вывести доску.

3. Цель работы.

Научиться описывать объекты реального мира с использованием классов.

s

4. Код программы.

#include <iostream>

using namespace std;

class GamePiece

{

public:

GamePiece(char PieceColor) : mPieceColor(PieceColor) {}

virtual char GetPiece() = 0;

char GetColor() {

return mPieceColor;

}

private:

virtual bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, GamePiece\* GameBoard[8][8]) = 0;

const char mPieceColor;

};

class PawnPiece : public GamePiece

{

public:

PawnPiece(char PieceColor) : GamePiece(PieceColor) {}

private:

char GetPiece() {

return 'P';

}

bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, GamePiece\* GameBoard[8][8]) {

GamePiece\* Dest = GameBoard[iDestRow][iDestCol];

if (Dest == 0) {

// Destination square is unoccupied

if (iSrcCol == iDestCol) {

if (GetColor() == 'W') {

if (iDestRow == iSrcRow + 1) {

return true;

}

}

else {

if (iDestRow == iSrcRow - 1) {

return true;

}

}

}

}

else {

// Dest holds piece of opposite color

if ((iSrcCol == iDestCol + 1) || (iSrcCol == iDestCol - 1)) {

if (GetColor() == 'W') {

if (iDestRow == iSrcRow + 1) {

return true;

}

}

else {

if (iDestRow == iSrcRow - 1) {

return true;

}

}

}

}

return false;

}

};

class KnightPiece : public GamePiece

{

public:

KnightPiece(char PieceColor) : GamePiece(PieceColor) {}

private:

char GetPiece() {

return 'N';

}

bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, GamePiece\* GameBoard[8][8]) {

// Destination square is unoccupied or occupied by opposite color

if ((iSrcCol == iDestCol + 1) || (iSrcCol == iDestCol - 1)) {

if ((iSrcRow == iDestRow + 2) || (iSrcRow == iDestRow - 2)) {

return true;

}

}

if ((iSrcCol == iDestCol + 2) || (iSrcCol == iDestCol - 2)) {

if ((iSrcRow == iDestRow + 1) || (iSrcRow == iDestRow - 1)) {

return true;

}

}

return false;

}

};

class KingPiece : public GamePiece

{

public:

KingPiece(char PieceColor) : GamePiece(PieceColor) {}

private:

char GetPiece() {

return 'K';

}

bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, GamePiece\* GameBoard[8][8]) {

int iRowDelta = iDestRow - iSrcRow;

int iColDelta = iDestCol - iSrcCol;

if (((iRowDelta >= -1) && (iRowDelta <= 1)) &&

((iColDelta >= -1) && (iColDelta <= 1)))

{

return true;

}

return false;

}

};

class CBoard

{

public:

GamePiece\* GameBoard[8][8];

void SetPawnPiece(int iRow, int iCol, char PieceColor) {

GameBoard[iRow][iCol] = new PawnPiece(PieceColor);

}

void SetKnightPiece(int iRow, int iCol, char PieceColor) {

GameBoard[iRow][iCol] = new KnightPiece(PieceColor);

}

void SetKingPiece(int iRow, int iCol, char PieceColor) {

GameBoard[iRow][iCol] = new KingPiece(PieceColor);

}

void DeletePiece(int iRow, int iCol) {

delete GameBoard[iRow][iCol];

GameBoard[iRow][iCol] = 0;

}

void ReplacePiece(int iStartMove, int iEndMove) {

int iStartRow = (iStartMove / 10);

int iStartCol = (iStartMove % 10);

int iEndRow = (iEndMove / 10);

int iEndCol = (iEndMove % 10);

// Check that the indices are in range

// and that the source and destination are different

if ((iStartRow >= 0 && iStartRow <= 7) &&

(iStartCol >= 0 && iStartCol <= 7) &&

(iEndRow >= 0 && iEndRow <= 7) &&

(iEndCol >= 0 && iEndCol <= 7)) {

GamePiece\* CurrPiece = GameBoard[iStartRow][iStartCol];

// Additional checks in here

if (CurrPiece != 0) {

{

// Make the move if this. piece can do this

GamePiece\* Figure = GameBoard[iEndRow][iEndCol];

GameBoard[iEndRow][iEndCol] = GameBoard[iStartRow][iStartCol]; // New position set

GameBoard[iStartRow][iStartCol] = 0; // Deletes old position

}

}

}

}

CBoard() {

for (int iRow = 0; iRow < 8; ++iRow) {

for (int iCol = 0; iCol < 8; ++iCol) {

GameBoard[iRow][iCol] = 0;

}

}

// Allocate and place black pieces

for (int iCol = 0; iCol < 8; ++iCol) {

SetPawnPiece(6, iCol, 'B');

}

SetKnightPiece(7, 1, 'B'); SetKingPiece(7, 3, 'B'); SetKnightPiece(7, 6, 'B');

// Allocate and place white pieces

for (int iCol = 0; iCol < 8; ++iCol) {

SetPawnPiece(1, iCol, 'W');

}

SetKnightPiece(0, 1, 'W'); SetKingPiece(0, 3, 'W'); SetKnightPiece(0, 6, 'W');

}

void Print() { // Method of printing the chess desk

const int SquareWidth = 4;

const int SquareHeight = 3;

for (int iRow = 0; iRow < 8 \* SquareHeight; ++iRow) {

int iSquareRow = iRow / SquareHeight;

// Print the chess board

for (int iCol = 0; iCol < 8 \* SquareWidth; ++iCol) {

int iSquareCol = iCol / SquareWidth;

if (((iRow % 3) == 1) && ((iCol % 4) == 1 || (iCol % 4) == 2) && GameBoard[7 - iSquareRow][iSquareCol] != 0) {

//Positioning of the game pieces

if ((iCol % 4) == 1) {

cout << GameBoard[7 - iSquareRow][iSquareCol]->GetColor();

}

else {

cout << GameBoard[7 - iSquareRow][iSquareCol]->GetPiece();

}

}

else {

if ((iSquareRow + iSquareCol) % 2 == 0) { // Cell positioning

cout << '\*';

}

else {

cout << ' ';

}

}

}

cout << endl;

}

}

bool KingHere(char PieceColor) {

// Find the king

int iKingRow;

int iKingCol;

for (int iRow = 0; iRow < 8; ++iRow) {

for (int iCol = 0; iCol < 8; ++iCol) {

if (GameBoard[iRow][iCol] != 0) {

if (GameBoard[iRow][iCol]->GetColor() == PieceColor) {

if (GameBoard[iRow][iCol]->GetPiece() == 'K') {

iKingRow = iRow;

iKingCol = iCol;

cout << "King's place: " << iKingRow << " " << iKingCol;

return true;

}

}

}

}

}

return false;

}

};

class ChessBoard

{

public:

void Start() {

GameBoard.SetPawnPiece(0, 0, 'W'); // 1

GameBoard.ReplacePiece(10, 20); // 2

//GameBoard.DeletePiece(); // 3

//GameBoard.KingHere('B'); // 4

GameBoard.Print(); // 5

}

private:

CBoard GameBoard;

};

int main() {

ChessBoard Game;

Game.Start();

return 0;

}

Вывод:  
научился работать с классами, правильно использовать инкапсуляцию.