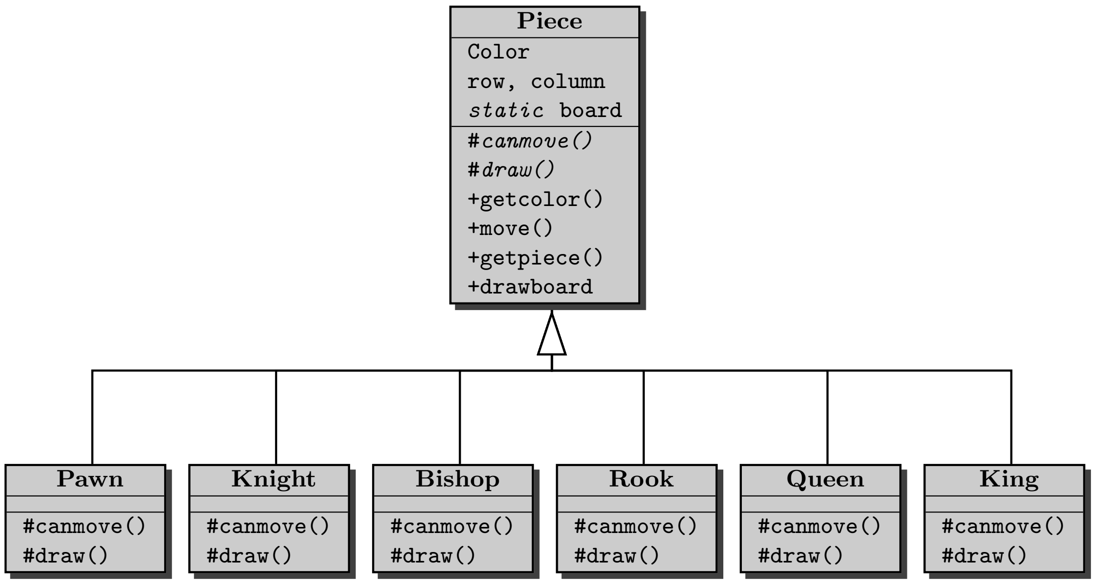
# METU NCC, CNG 242, Spring 2010, Homework 5

Due 25th May 2010

In this homework you will implement some utility classes for the Chess game. The game engine interacts with an abstract class to make players move. Each kind of chess piece inherits this abstract class and implements virtual member functions and specify if the given move is valid or not.

## 1 Class Diagram



Board state is defined as a *static member variable* to the Piece class and each constructed piece has to be implemented on this board. When a piece captures another, the capture piece is deleted from the board. All moves are done through Piece class member function move(). Those moves are checked against the board and updated on the board if valid. All pieces are heap variables created with new.

## 2 Header File

You are given the following header file hw5.h:

```
#include<iostream>
using namespace std;
class Exception {
        const char *mess;
public:
        Exception(const char *m) { mess = m ; }
        void output() { cerr << mess << '\n';}
};

// A piece with invalid color value is tried to be constructed
class InvalidColor: public Exception {
public:
        InvalidColor():Exception("Invalid piece color") {}
};

// A move outside of the board is attempted
class OutofBoard: public Exception {
public:
        OutofBoard():Exception("Out of the Board") {}
};
```

```
// A piece is tried to be placed on an already occupied cell
class Occupied: public Exception {
public:
        Occupied():Exception("Location is occupied") {}
};

// An invalid move is targetted for the piece kind
class CannotMove: public Exception {
public:
        CannotMove():Exception("Cannot move there") {}
};

// A piece is expected on the cell but, the cell is empty
class EmptyCell: public Exception {
public:
        EmptyCell():Exception("The cell is empty, no piece there") {}
};

enum Color { BLACK, WHITE};
enum Column {A, B, C, D, E, F, G, H, OUT};
const char ctoname[]="ABCDEFGH    ";  // convert Column type to character
                                      // for printing. ctoname[A] is 'A'
const char coltoname[][10]={"BLACK","WHITE"};  // convert Color type to color
                                           // name. coltoname[BLACK] is "BLACK"
// macro to convert character input into Column name nametoc('a') is A ,
//  nametoc('H') is H.
#define nametoc(a)  (('a'<= a && a <= 'h')?(Column) (a-'a'):\
                     ('A' <= a && a <= 'H')?(Column) (a - 'A'):OUT)

// Base abstract class.
class Piece {
protected:
        static Piece * board[9][OUT];     // board is a class member variable
        Color color;                       // BLACK or WHITE
        Column x;                          // A to H
        int y;                             // 1 to 8. 8 is top, 1 is bottom
        // each piece kind (Pawn, Knight, Bishop, Rook, Queen, King) decides on its own
        virtual int canmove(Column, int) const = 0;
        // each pice returns a string denoting itself (P, N, B, R, Q, K)
        virtual const char * draw() const  = 0;
public:
        // throws OutofBoard, Occupied
        Piece(Color clr, Column clmn, int row);
        Color getcolor() const { return color; }
        // throws OutofBoard, CannotMove
        void move(Column clmn, int row);
        //throws EmptyCell
        static Piece * getpiece(Column clmn, int row);
        // draw the curent state of the board, uses draw functions from pieces
        static void drawboard() {
                const char empty[]=
"|       |       |       |       |       |       |       |       |\n";
                const char line[]=
"+-------+-------+-------+-------+-------+-------+-------+-------+\n";

                cout << line << empty ;
                for (int r = 8; r>0 ; r--) {
                        for (int c = A ; c <= H ; ++c)
                                if (board[r][c] == NULL)
                                        cout << "|       ";
                                else cout << "| " << board[r][c]->draw()
                                          << " ";
                        cout << "|\n" << empty << line;
                        if ( r > 1)  cout << empty;
                }
        }
};

// initialize cells to be empty
Piece * Piece::board[9][OUT]={
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL},
        {NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL}};



// create a piece depending on the name given in first argument
// pawn, knight, bishop, rook, queen and king
// returns NULL if no such piece is defined
extern Piece *create(const char *piecename, Color clr, Column clmn, int row);
```

## 3 Implementation

You are asked:

1. to implement unimplemented member functions of Piece class in file `hw5.cpp`
2. to define all classes corressponding to the actual chess pieces derived from abstract class Piece and create() function in file `chess.cpp`

The functions to implement in `hw5.cpp` and their meanings are given as:

`Piece::Piece(Color, Column, int)`
: Constructs a new piece on the board. Make all necessary controls before placing the piece on the board, makes sure that the requested location is on the board and it is not occupied. Then updates the board member variable to contain pointer to the constructed object. All derived classes have constructors calling this constructor properly.

`void Piece::move(Column, int)`
: For any piece, it tries to make the move. It make couple of checks, make sure that the position requested is on the board and not occupied by a piece with same color. It also asks the derived class if it can move this location by calling canmove() function. When move is valid, it makes the move, if there is an opponnent piece, it captures the piece (it is deleted).

  In case a piece is captured this function outputs a message like:
  `captured:  * P * at B7`

`static Piece * Piece::getpiece(Column, int)`
: Returns the pointer of the piece in the given position. If position is empty, it raises the exception EmptyCell

The functions to implement in `chess.cpp` and their meanings are given as:

`int canmove(Column, int)`
: It is implemented for all derived classes specifically to test if the move from pieces current location to given position is possible for the piece.

`const char * draw() const`
: It is implemented for all derived classes specifically to return the corressponding string for the piece. The string from the following table is returned:

| Piece \ Color | BLACK | WHITE |
|---|---|---|
| Pawn | `* P *` | `. P .` |
| Knight | `* N *` | `. N .` |
| Bishop | `* B *` | `. B .` |
| Rook | `* R *` | `. R .` |
| Queen | `* Q *` | `. Q .` |
| King | `* K *` | `. K .` |

Note that `drawboard()` function uses this function.

Constructors

For each of the derived classes, the corressponding class constructor should be implemented. The parameters are passed to `Piece` constructor.

`Player *create(const char *, Column, int)`

It is a not object oriented function. Implemented in global scope. It is in the `chess.cpp` and has access to all derived piece classes. So it can call `new` for them and return their pointers. The first parameter is the name of the class as a string, one of:
`"pawn"`, `"knight"`, `"bishop"`, `"rook"`, `"queen"`, `"king"`.
If any other string is specified, `NULL` is returned.

## 4 Chess Rules

In order to implement `canmove()` you need to know the rules of the game. You can find it on web page:
`http://en.wikipedia.org/wiki/Chess_rules#Movement`
Here are couple of reminders and simplifications:

- Do not implement *Castling* (Rok in Turkish) move switching King and the Rook.
- Do not implement *Pawn promotion*
- Do not implement *Mate* situation, allow moves of the King to a threatened location.
- Do not implement *En passant* capture of a Pawn by another Pawn.
- Implement all other moves of the Pawn, diagonal move if an opponent is to be captured, initial two step move.
- Note that you should check intermediate cells for being empty when pieces other than Knight is moving.
- `move()` should already check if move is outside of the board and if the move is already occupied by a same color piece. So you don't have to test them again inside of `canmove()`.

## 5 Main File

You are given the following file `hw5main.cpp`:

```
#include<iostream>
using namespace std;
#include "hw5.h"

int main() {
        // place 16 pawns
        for (int c = A ; c <= H ; ++c) {
                create("pawn", BLACK, (Column) c, 7);
                create("pawn", WHITE, (Column) c, 2);
        }
        // white pieces
        create("rook", WHITE, A, 1);
        create("knight", WHITE, B, 1);
        create("bishop", WHITE, C, 1);
        create("queen", WHITE, D, 1);
        create("king", WHITE, E, 1);
        create("bishop", WHITE, F, 1);
        create("knight", WHITE, G, 1);
        create("rook", WHITE, H, 1);

        // black pieces
        create("rook", BLACK, A, 8);
        create("knight", BLACK, B, 8);
        create("bishop", BLACK, C, 8);
        create("king", BLACK, D, 8);
        create("queen", BLACK, E, 8);
        create("bishop", BLACK, F, 8);
        create("knight", BLACK, G, 8);
        create("rook", BLACK, H, 8);

        // white starts
        Color turn = WHITE ;

        Piece::drawboard();

        while (1) {
                char sc,ec;
                int sr, er;
                cout << "Turn: " << coltoname[turn] << '\n';
                // input start pos and end pos as "D7 E5"
                cin >> ws >> sc >> sr;
                cin >> ws >> ec >> er;

                try {
                        Piece *p;
                        // ask the piece  on start position
                        // if no piece exist on pos EmptyCell
                        p = Piece::getpiece(nametoc(sc),sr);

                        // if piece with the wrong color leave iteration
                        if (p->getcolor() != turn) {
                                cout << "not turn of that player\n";
                                continue;
                        }

                        // move the piece on start location
                        p->move(nametoc(ec),er);

                        Piece::drawboard();

                        // switch turns
                        turn = (Color) (! (int) turn);
                } catch (Exception &e) {
                        e.output();   // output exception message
                }
        }
}
```

This code interacts with the user and makes all moves given on input and executes them through `Piece` member functions and `create()`

## 6 Submission

Put two files `hw5.cpp` and `chess.cpp` in a `.zip` file and submit. Note that your code should be compiled separately with the `hw5main.cpp` specified above. `hw5main.cpp` should not include any source from `chess.cpp`. You can download given sources from the web page.