```
#ifndef _PILE_H
#define _PILE_H

#include <string>
#include <vector>
using namespace std;

class Card
{
  public:

    enum suit{
        spades, hearts, diamonds,clubs
    };

    suit getSuit()    const;
    int getRank()     const;
    string toString() const;

    bool operator < (const Card& c) const;

    friend class Deck;

  private:
    Card(int n);

    int myNumber;
    static string ourRankStrings[];
    static string ourSuitStrings[];
};

inline ostream& operator << (ostream& out, const Card& c)
{
    out << c.toString();
    return out;
}


class Displayer;

class Pile
{
  public:
    virtual int size() const = 0;

    virtual bool addCard(const Card& c) = 0;
    virtual void display(Displayer* d) const = 0;
    virtual Card takeCard()             = 0;
};


class Deck : public Pile
{
  public:
    Deck();

    virtual int size() const;
    virtual void display(Displayer * d) const;

    virtual bool addCard(const Card& c);
    virtual Card takeCard();
    virtual void shuffle();

  private:

    vector<Card> myCards;
};
```



```
class BoardPile : public Pile
{
   public:
    BoardPile();
    BoardPile(const vector<Card>& cards);

    virtual int size() const;

    virtual bool addCard(const Card& c);

    virtual Card takeCard();

    virtual void display(Displayer * d) const;

  private:
    vector<Card> myCards;
};


#endif
```

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <stdexcept>
using namespace std;

#include "pile.h"
#include "displayer.h"

int main()
{
    vector<BoardPile *> boardPiles(8);
    Deck * d = new Deck();
    Displayer * textd = new TextDisplayer();

    if (d != 0){
        d->display(textd);
        d->shuffle();
    }

    vector<vector<Card> > forPiles(8);
    try{
        int count = 0;
        while (true){
            Card c = d->takeCard();
            forPiles[count % 8].push_back(c);
            count++;
        }

    }
    catch (runtime_error * re) {
        cerr << "oops " << re->what() << endl;
    }


    for(int k=0; k < 8; k++){
        boardPiles[k] = new BoardPile(forPiles[k]);
    }

    for(int k=0; k < 8; k++){
        cout << "board pile " << k << endl;
        boardPiles[k]->display(textd);
        cout << "---" << endl;
    }
}
```

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <stdexcept>
using namespace std;

#include <ctime>        // for time()
#include <cstdlib>      // for rand/srand

#include "pile.h"
#include "displayer.h"

class Randomizer
{
  public:
    Randomizer(){
        if (!ourInitialized){
            ourInitialized = true;
            srand(unsigned(time(0)));       // randomize
        }
    }

    int operator()(int n)
    {
        return int((rand() / (double(RAND_MAX) + 1))*n);
    }

  private:
    static bool ourInitialized;
};

bool Randomizer::ourInitialized = false;

string Card::ourRankStrings[] = {
    "ace","two", "three", "four", "five", "six",
        "seven","eight", "nine", "ten", "jack", "queen","king"
};

string Card::ourSuitStrings[] = {
        "spades","hearts","diamonds","clubs"
};


Card::Card(int n)
    : myNumber(n)
{

}

string Card::toString() const
{
    return ourRankStrings[getRank()] +
        " of " + ourSuitStrings[getSuit()];
}


bool Card::operator < (const Card& c) const
{
    int myRank = getRank();
    int cRank = c.getRank();

    if (myRank == cRank){
        return getSuit() < c.getSuit();
    }
    return myRank < cRank;
}

int Card::getRank() const
```



```
{
    return myNumber / 4;
}

Card::suit Card::getSuit() const
{
    return static_cast<suit> (myNumber % 4);
}

Deck::Deck()
{
    for(int k=0; k < 52; k++){
        myCards.push_back(Card(k));
    }
}

int Deck::size() const
{
    return myCards.size();
}

bool Deck::addCard(const Card& c)
{
    return false;
}

void Deck::display(Displayer * d) const
{
    for(int k=0; k < size(); k++){
        d->showCard(myCards[k]);
    }
}

Card Deck::takeCard()
{
    if (size() != 0) {
        Card c = myCards[size()-1];
        myCards.pop_back();
        return c;
    }
    throw new runtime_error("no cards left");
}

void Deck::shuffle()
{
    Randomizer rando;
    random_shuffle(myCards.begin(), myCards.end(), rando);
}


BoardPile::BoardPile()
{

}

BoardPile::BoardPile(const vector<Card>& cards)
    : myCards(cards)
{

}

int BoardPile::size() const
{
    return myCards.size();
}

bool BoardPile::addCard(const Card& c)
{
    return false;
```

```
}

Card BoardPile::takeCard()
{
    if (size() != 0) {
        return myCards[0];
    }
    throw new runtime_error("no cards left");
}

void BoardPile::display(Displayer* d) const
{
    for(int k=0; k < size(); k++){
        d->showCard(myCards[k]);
    }
}
```