```cpp
#include "Card.h"
#include "Pile.h"

#define _CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#ifdef _DEBUG
#ifndef DBG_NEW
#define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
#define new DBG_NEW
#endif
#endif // _DEBUG

//there are no static methodes to initialize!

Pile::Pile()
    :m_size(0), m_left(0), m_right(m_size)
{
    m_queue = new Card * [m_capacity];
    memset(m_queue, NULL, m_capacity * sizeof(Card*));
}

Pile::~Pile()
{
    //Delete all created dynamic allocations:
    for (int i = 0; i < m_size; i++)
    {
        delete m_queue[i];
    }
    delete[] m_queue;
}

////////////////////////////////////////////////////////////////////////////////
void Pile::push_back(Card* card)
{
    if (is_empty())    //empty Pile
        m_queue[m_right] = card;
    else if (is_full()) //full Pile
    {
        cout << "Pile is full ! - cant push a card !" << endl;
        return;
    }
    else
    {
        for_push_next_index("push_back");
        m_queue[m_right] = card;
    }
    m_size++;
}

Card* Pile::pop_back()
{
    if (is_empty())
```

```
        return NULL;
    Card* card = m_queue[m_right];
    m_queue[m_right] = NULL;                    //card pointer is not in m_queue after this line
    for_pop_prev_index("pop_back");
    m_size--;
    return card;
}

void Pile::push_front(Card* card)
{
    if (is_empty())
        m_queue[m_left] = card;
    else if (is_full())
    {
        cout << "Pile is full ! - cant push a card !" << endl;
        return;
    }
    else {
        for_push_next_index("push_front");
        m_queue[m_left] = card;
    }
    m_size++;
}


Card* Pile::pop_front()
{
    if (is_empty())
        return NULL;
    Card* card = m_queue[m_left];
    m_queue[m_left] = NULL;                    //card pointer is not in m_queue after this line
    for_pop_prev_index("pop_front");
    m_size--;
    return card;
}
////////////////////////////////////////////////////////////////////////////////////////////////////

////

const Card& Pile::front() const
{
    return *m_queue[m_left];
}

const Card& Pile::back() const
{
    return *m_queue[m_right];
}

unsigned int Pile::size() const
```

```
{
    return m_size;
}

bool Pile::is_empty() const
{
    if (m_size == 0)
        return true;
    else
        return false;
}

///

//Extras:

bool Pile::is_full() const
{
    if (m_size == m_capacity)
        return true;
    else
        return false;
}

//////////////////////////////////////////////////////////////////////////////////////////////////
void Pile::for_push_next_index(const string& push_back_or_push_front)
{
    if (m_size == m_capacity)
        return;
    if (push_back_or_push_front == "push_back") {
        if (m_right >= m_capacity - 1)
            m_right = 0;
        else
            m_right++;
    }
    else if (push_back_or_push_front == "push_front") {
        if (m_left == 0)
            m_left = m_capacity - 1;
        else
            m_left--;
    }
    else
        return;
}

void Pile::for_pop_prev_index(const string& pop_back_or_pop_front)
{
    if (is_empty())
        return;
    if (pop_back_or_pop_front == "pop_back") {
        if (m_right == 0) //m_capacity - 1)
            m_right = m_capacity - 1;
```

```
        else
            m_right--;
    }
    else if (pop_back_or_pop_front == "pop_front") {
        if (m_left == m_capacity - 1)
            m_left = 0;
        else
            m_left++;
    }
    else
        return;
}

////////////////////////////////////////////////////////////////////////////////////////////////////

//void Pile::new_pile()
//{
//  //int i = 0;
//  int rand_creating_card;
//  while ( Card::get_number_of_cards_already_made() < Card::get_total_number_of_cards_in_1_ratatat_pile())
//  {
//      rand_creating_card = rand() % 5;
//      switch (rand_creating_card)
//      {
//          //Cat_cards
//      case 0:
//          while (Cat_card::get_total_cards() < Cat_card::get_total_Cat_cards_in_1_pile())
//          {
//              //m_queue[i] = new Cat_card(Cat_card::toss_val());
//              push_front(new Cat_card(Cat_card::toss_val()));
//              //i++;
//          }
//          break;
//          //Rat_cards
//      case 1:
//          while (Rat_card::get_total_cards() < Rat_card::get_total_Rat_cards_in_1_pile())
//          {
//              //m_queue[i] = new Rat_card(Rat_card::toss_val());
//              push_front(new Rat_card(Rat_card::toss_val()));
//              //i++;
//          }
//          break;
//          //Peek_cards
//      case 2:
//          while (Peek_card::get_total_cards() < Peek_card::get_total_Peek_cards_in_1_pile())
//          {
//              //m_queue[i] = new Peek_card();
//              push_front(new Peek_card);
```

```
//
//              //i++;
//          }
//          break;
//          //Draw 2_cards
//      case 3:
//          while (Draw2_card::get_total_cards() < Draw2_card::get_total_Draw2_cards_in_1_pile())
//          {
//              //m_queue[i] = new Draw2_card();
//              push_front(new Draw2_card);
//              //i++;
//          }
//          break;
//          //Swap_cards
//      case 4:
//          while (Swap_card::get_total_cards() < Swap_card::get_total_Swap_cards_in_1_pile())
//          {
//              //m_queue[i] = new Swap_card();
//              push_front(new Swap_card);
//              //i++;
//          }
//          break;
//      }
//  }
//}

void Pile::new_pile()
{
    //int i = 0;
    int rand_creating_card;
    while (Card::get_number_of_cards_already_made() < Card::get_total_number_of_cards_in_1_ratatat_pile())
    {
        rand_creating_card = rand() % 5;
        switch (rand_creating_card)
        {
            //Cat_cards
        case 0:
            if (Cat_card::get_total_cards() < Cat_card::get_total_Cat_cards_in_1_pile())
            {
                //m_queue[i] = new Cat_card(Cat_card::toss_val());
                push_front(new Cat_card(Cat_card::toss_val()));
                //i++;
            }
            break;
            //Rat_cards
        case 1:
            if (Rat_card::get_total_cards() < Rat_card::get_total_Rat_cards_in_1_pile())
            {
```

```
                //m_queue[i] = new Rat_card(Rat_card::toss_val());
                push_front(new Rat_card(Rat_card::toss_val()));
                //i++;
            }
            break;
            //Peek_cards
        case 2:
            if (Peek_card::get_total_cards() < Peek_card::get_total_Peek_cards_in_1_pile())
            {
                //m_queue[i] = new Peek_card();
                push_front(new Peek_card);

                //i++;
            }
            break;
            //Draw 2_cards
        case 3:
            if (Draw2_card::get_total_cards() < Draw2_card::get_total_Draw2_cards_in_1_pile())
            {
                //m_queue[i] = new Draw2_card();
                push_front(new Draw2_card);
                //i++;
            }
            break;
            //Swap_cards
        case 4:
            if (Swap_card::get_total_cards() < Swap_card::get_total_Swap_cards_in_1_pile())
            {
                //m_queue[i] = new Swap_card();
                push_front(new Swap_card);
                //i++;
            }
            break;
        }
    }
}


void Pile::shuffle()
{
    unsigned int number_of_swaps = 101;
    unsigned int index_a = 0;
    unsigned int index_b = 0;
    for (unsigned i = 0; i < number_of_swaps; i++)
    {
        random_2_indexes_0_to_m_capacity_minus1(index_a, index_b);
        swap_cards(m_queue[index_a], m_queue[index_b]);
    }
}

```

```
void Pile::random_2_indexes_0_to_m_capacity_minus1(unsigned int& index_a,
  unsigned int& index_b)
{
    index_a = rand() % m_capacity;
    index_b = rand() % m_capacity;
}

void Pile::swap_cards(Card*& card_a_pointer, Card*& card_b_pointer)
{
    Card* tmp_card_pointer = card_a_pointer;
    card_a_pointer = card_b_pointer;
    card_b_pointer = tmp_card_pointer;
    return;
}

```