

```

1 #include "Card.h"
2 #include "Pile.h"
3
4 #define _CRTDBG_MAP_ALLOC
5 #include <crtdbg.h>
6 #ifdef _DEBUG
7 #ifndef DBG_NEW
8 #define DBG_NEW new ( _NORMAL_BLOCK , __FILE__ , __LINE__ )
9 #define new DBG_NEW
10 #endif
11 #endif // _DEBUG
12
13 //there are no static methodes to initialize!
14
15 Pile::Pile()
16     :m_size(0), m_left(0), m_right(m_size)
17 {
18     m_queue = new Card * [m_capacity];
19     memset(m_queue, NULL, m_capacity * sizeof(Card*));
20 }
21
22 Pile::~~Pile()
23 {
24     //Delete all created dynamic allocations:
25     for (int i = 0; i < m_size; i++)
26     {
27         delete m_queue[i];
28     }
29     delete[] m_queue;
30 }
31
32 //////////////////////////////////////// ↗
33 void Pile::push_back(Card* card)
34 {
35     if (is_empty())    //empty Pile
36         m_queue[m_right] = card;
37     else if (is_full()) //full Pile
38     {
39         cout << "Pile is full ! - cant push a card !" << endl;
40         return;
41     }
42     else
43     {
44         for_push_next_index("push_back");
45         m_queue[m_right] = card;
46     }
47     m_size++;
48 }
49
50 Card* Pile::pop_back()
51 {
52     if (is_empty())

```

```

53     return NULL;
54     Card* card = m_queue[m_right];
55     m_queue[m_right] = NULL;           //card pointer is not in ↗
        m_queue after this line
56     for_pop_prev_index("pop_back");
57     m_size--;
58     return card;
59 }
60
61 void Pile::push_front(Card* card)
62 {
63     if (is_empty())
64         m_queue[m_left] = card;
65     else if (is_full())
66     {
67         cout << "Pile is full ! - cant push a card !" << endl;
68         return;
69     }
70     else {
71         for_push_next_index("push_front");
72         m_queue[m_left] = card;
73     }
74     m_size++;
75 }
76
77
78 Card* Pile::pop_front()
79 {
80     if (is_empty())
81         return NULL;
82     Card* card = m_queue[m_left];
83     m_queue[m_left] = NULL;           //card pointer is not in ↗
        m_queue after this line
84     for_pop_prev_index("pop_front");
85     m_size--;
86     return card;
87 }
88 //////////////////////////////////////// ↗
89     ////////////////////////////////////////
90
91
92 const Card& Pile::front() const
93 {
94     return *m_queue[m_left];
95 }
96
97 const Card& Pile::back() const
98 {
99     return *m_queue[m_right];
100 }
101
102 unsigned int Pile::size() const

```

```
103 {
104     return m_size;
105 }
106
107 bool Pile::is_empty() const
108 {
109     if (m_size == 0)
110         return true;
111     else
112         return false;
113 }
114
115 ///
116
117 //Extras:
118
119 bool Pile::is_full() const
120 {
121     if (m_size == m_capacity)
122         return true;
123     else
124         return false;
125 }
126
127 //////////////////////////////////////// ↗
128 void Pile::for_push_next_index(const string& push_back_or_push_front)
129 {
130     if (m_size == m_capacity)
131         return;
132     if (push_back_or_push_front == "push_back") {
133         if (m_right >= m_capacity - 1)
134             m_right = 0;
135         else
136             m_right++;
137     }
138     else if (push_back_or_push_front == "push_front") {
139         if (m_left == 0)
140             m_left = m_capacity - 1;
141         else
142             m_left--;
143     }
144     else
145         return;
146 }
147
148 void Pile::for_pop_prev_index(const string& pop_back_or_pop_front)
149 {
150     if (is_empty())
151         return;
152     if (pop_back_or_pop_front == "pop_back") {
153         if (m_right == 0) //m_capacity - 1
154             m_right = m_capacity - 1;
```

```

155         else
156             m_right--;
157     }
158     else if (pop_back_or_pop_front == "pop_front") {
159         if (m_left == m_capacity - 1)
160             m_left = 0;
161         else
162             m_left++;
163     }
164     else
165         return;
166 }
167
168 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
169
170 //void Pile::new_pile()
171 //{
172 //    //int i = 0;
173 //    int rand_creating_card;
174 //    while ( Card::get_number_of_cards_already_made() <
175 //           Card::get_total_number_of_cards_in_1_ratatat_pile())
176 //    {
177 //        rand_creating_card = rand() % 5;
178 //        switch (rand_creating_card)
179 //        {
180 //            //Cat_cards
181 //            case 0:
182 //                while (Cat_card::get_total_cards() <
183 //                       Cat_card::get_total_Cat_cards_in_1_pile())
184 //                {
185 //                    //m_queue[i] = new Cat_card(Cat_card::toss_val());
186 //                    push_front(new Cat_card(Cat_card::toss_val()));
187 //                    //i++;
188 //                }
189 //                break;
190 //            //Rat_cards
191 //            case 1:
192 //                while (Rat_card::get_total_cards() <
193 //                       Rat_card::get_total_Rat_cards_in_1_pile())
194 //                {
195 //                    //m_queue[i] = new Rat_card(Rat_card::toss_val());
196 //                    push_front(new Rat_card(Rat_card::toss_val()));
197 //                    //i++;
198 //                }
199 //                break;
200 //            //Peek_cards
201 //            case 2:
202 //                while (Peek_card::get_total_cards() <
203 //                       Peek_card::get_total_Peek_cards_in_1_pile())
204 //                {
205 //                    //m_queue[i] = new Peek_card();
206 //                    push_front(new Peek_card);

```

```

203 //
204 //          //i++;
205 //      }
206 //      break;
207 //      //Draw 2_cards
208 //      case 3:
209 //          while (Draw2_card::get_total_cards() <
210 //              Draw2_card::get_total_Draw2_cards_in_1_pile())
211 //          {
212 //              //m_queue[i] = new Draw2_card();
213 //              push_front(new Draw2_card);
214 //              //i++;
215 //          }
216 //          break;
217 //          //Swap_cards
218 //          case 4:
219 //              while (Swap_card::get_total_cards() <
220 //                  Swap_card::get_total_Swap_cards_in_1_pile())
221 //              {
222 //                  //m_queue[i] = new Swap_card();
223 //                  push_front(new Swap_card);
224 //                  //i++;
225 //              }
226 //              break;
227 //          }
228 //      }
229 //  }
230 //}
231
232 void Pile::new_pile()
233 {
234     //int i = 0;
235     int rand_creating_card;
236     while (Card::get_number_of_cards_already_made() <
237         Card::get_total_number_of_cards_in_1_ratatat_pile())
238     {
239         rand_creating_card = rand() % 5;
240         switch (rand_creating_card)
241         {
242             //Cat_cards
243             case 0:
244                 if (Cat_card::get_total_cards() <
245                     Cat_card::get_total_Cat_cards_in_1_pile())
246                 {
247                     //m_queue[i] = new Cat_card(Cat_card::toss_val());
248                     push_front(new Cat_card(Cat_card::toss_val()));
249                     //i++;
250                 }
251                 break;
252             //Rat_cards
253             case 1:
254                 if (Rat_card::get_total_cards() <
255                     Rat_card::get_total_Rat_cards_in_1_pile())
256                 {

```

```
251         //m_queue[i] = new Rat_card(Rat_card::toss_val());
252         push_front(new Rat_card(Rat_card::toss_val()));
253         //i++;
254     }
255     break;
256     //Peek_cards
257     case 2:
258         if (Peek_card::get_total_cards() <
259             Peek_card::get_total_Peek_cards_in_1_pile())
260         {
261             //m_queue[i] = new Peek_card();
262             push_front(new Peek_card);
263             //i++;
264         }
265         break;
266         //Draw 2_cards
267     case 3:
268         if (Draw2_card::get_total_cards() <
269             Draw2_card::get_total_Draw2_cards_in_1_pile())
270         {
271             //m_queue[i] = new Draw2_card();
272             push_front(new Draw2_card);
273             //i++;
274         }
275         break;
276         //Swap_cards
277     case 4:
278         if (Swap_card::get_total_cards() <
279             Swap_card::get_total_Swap_cards_in_1_pile())
280         {
281             //m_queue[i] = new Swap_card();
282             push_front(new Swap_card);
283             //i++;
284         }
285         break;
286     }
287 }
288
289 void Pile::shuffle()
290 {
291     unsigned int number_of_swaps = 101;
292     unsigned int index_a = 0;
293     unsigned int index_b = 0;
294     for (unsigned i = 0; i < number_of_swaps; i++)
295     {
296         random_2_indexes_0_to_m_capacity_minus1(index_a, index_b);
297         swap_cards(m_queue[index_a], m_queue[index_b]);
298     }
299 }
300
```

```
301 void Pile::random_2_indexes_0_to_m_capacity_minus1(unsigned int& index_a, ↗
    unsigned int& index_b)
302 {
303     index_a = rand() % m_capacity;
304     index_b = rand() % m_capacity;
305 }
306
307 void Pile::swap_cards(Card*& card_a_pointer, Card*& card_b_pointer)
308 {
309     Card* tmp_card_pointer = card_a_pointer;
310     card_a_pointer = card_b_pointer;
311     card_b_pointer = tmp_card_pointer;
312     return;
313 }
314
```