Государственный университет – Высшая школа экономики

Факультет Бизнес-Информатики

Кафедра Основ информатики
и прикладного программного обеспечения

C#

Объектно-ориентированный язык программирования

Дополнение 1

к проекту

Компьютерная игра Блэк джек (Blackjack)

Проф. Забудский Е.И.

Москва 2008

Тема 9. Разработка компьютерных моделей
на основе объектно-ориентированной методологии:
практический пример

Дополнение 1
к проекту
Компьютерная игра Блэк джек (Blackjack)

Изучаются две ключевые концепции ООП: *абстракция* и *инкапсуляция*. Рассмотрение этих концепций, в данном случае, осуществляется на примере описания **колоды карт**. Иллюстрируются шаблоны проектирования Итератор (Iterator) и Перечисление с сохранением типа (Typesafe Enum), используемые для улучшения кода [8].

Unified Modeling Language (UML) – популярный язык графического моделирования, используемый для представления объектно-ориентированных программ (**www.omg.org/uml**).

// **КОММЕНТАРИЙ**. На сайте http://www.firststeps.ru/ “Первые шаги” представлено много интересных обучающих материалов по различным интегрированным средам и языкам программирования, в том числе представлены C# и платформа .NET (step by step).

Содержание

1. Вокруг инкапсуляции

Эффективная инкапсуляция характеризуется тремя свойствами:
a) **абстракция;**
b) **сокрытие реализации;**
c) **распределение ответственности.**

Чтобы правильно инкапсулировать объекты в класс, необходимо наделить их всеми этими свойствами.

1.1. Применение трех свойств инкапсуляции на примере игры в карты

a) Сначала **рассмотрим** **абстракцию**.Применяя ее, не нужно заходить слишком далеко, так как поставлена конкретная задача, а попытка разрешить все задачи сразу явно обречена на провал.

Что можно сказать *о какой-то* карточной игре?

Очевидно, что обсуждение начнем с такого объекта как колода карт. Что можно сказать о ней?

1. Стандартная колода карт состоит из 52 карт.
2. Можно тасовать колоду,
3. Можно выбирать с какой-то позиции карту из колоды.
4. Можно возвратить выбранную ранее карту в другую позицию.

Что еще известно о картах?

1. Все карты имеют общую структуру.
2. Каждая карта имеет **масть;** мастей четыре: бубны ♦, черви ♥, трефы ♣ и пики ♠.
3. Карта имеет и собственное **достоинство**, выражаемое числами от **2** до **10** и "**титулами**" валет, дама, король, туз.

Карты отличаются только значениями этих двух атрибутов — масти и достоинства.

b) **Рассмотрим** **сокрытие реализации**

1. Узнать, какая карта лежит в колоде, можно только вытащив карту из колоды.
2. В колоду нельзя вставить карту, которая не является частью колоды.

c) **Рассмотрим** **распределение ответственности**

Как отмечено выше карты обозначаются своей мастью и достоинством и имеют два состояния (положения): лицом вверх и лицом вниз.

В компьютерной модели карта будет помнить: 1) свою масть, 2) достоинство и 3) состояние.

Тот, кто раздает карты **(dealer),** сначала тасует колоду, а затем раздает их.

1. В простых программах даже карты умеют показать себя. **2.** Колода будет создаваться для того, чтобы хранить карты. **3.** А раздающий карты будет знать, как тасовать их и осуществлять раздачу.

1.1.1. Постановка задачи

Приведенное выше описание колоды карт достаточно, чтобы спроектировать следующие классы:

1) класс **Card (карта)**,
2) класс **Deck of cards (колода карт),**
3) класс **Dealer (раздающий карты)**.

Затем необходимо написать небольшой метод **Main()**, который выполняет следующее:

1) создает экземпляр класса **dealer** (раздающий карты), **с. 10, л-г 4а, стр. 6**
2) создает экземпляр класса **deck** of cards (колода карт), **с. 10, л-г 4а, стр. 5**

3) **перетасовывает** карты, **с. 11, л-г 4а, стр. 10** и **с. 9, л-г 3а, стр. 11…22**
4) **распечатывает** полученную колоду карт. **с. 10, л-г 4а, стр. 18…29** и **с. 6, л-г 2а, стр. 70…94**

Перед проектированием класса полезно обдумать сокрытие реализации и разделение ответственности.

Примечание: возлагать ответственность действительно на ответственных "лиц".

1.1.2. Решения и их обсуждение

В **листинге 1а** представлена одна из возможных реализаций класса **Card (карта)**.

Листинг 1а. файл Card.CS **см. папку CAp_C#_gl3_PZ3_c76.NET**

```
Это техническое представление карты в виде класса Card.
Карты отличаются только значением масти и достоинством, а также положением
1:  public class Card
2:  {
3:  private int suit;          // 1) масть карты (suit)
4:  private int rank;          // 2) достоинство карты (values, rank)
5:  private bool face_up; // признак положения карты: лицом вниз (false), вверх (true)
6:
7:          // константы, используемые при присвоении:
8:                                          // 1) масти (suits) – 10-тичный ASCII-код:
9:      public const int DIAMONDS = 4;   //бубны   ♦
10:     public const int HEARTS    = 3;   //черви   ♥
11:     public const int SPADES    = 6;   //пики    ♠
12:     public const int CLUBS     = 5;   //трефы   ♣
13:                                         // 2) достоинства (rank):
14:         public const int TWO    = 2;
15:         public const int THREE = 3;
16:         public const int FOUR   = 4;
17:         public const int FIVE    = 5;
18:         public const int SIX     = 6;
19:         public const int SEVEN = 7;
20:         public const int EIGHT  = 8;
21:         public const int NINE    = 9;
22:         public const int TEN    = 10;
23:         public const int JACK    = 74;      // валет – 10-тичный ASCII-код: J
24:         public const int QUEEN = 81;      // дама                          : Q
25:         public const int KING    = 75;      // король                        : K
26:         public const int ACE     = 65;      // туз                           : A
27:
28:          /* создание новой карты - при этом используются только те константы,
                                        которые были инициализированы выше */
29:         public Card(int suit, int rank)                  // создание новой карты
30:         {
31:      // В реальной программе, нужно проверять правильность аргументов
32:                  this.suit = suit;
33:                  this.rank = rank;
34:         }
35:
36:          public int Suit  // свойство – получить значение масти карты
37:         {
38:                  get
39:                  {
40:                  return suit;
```

```
41:                 }
42:         }
43:
44:           public int Rank  // свойство – получить значение достоинства карты
45:         {
46:                 get
47:                 {
48:                         return rank;
49:                 }
50:         }
51:
52:   public bool FaceUp  // свойство – получить значение признака положения карты
53:         {
54:         get
55:                 {
56:                         return face_up;
57:                 }
58:         }
59:
60:         public   void  faceUp()  // метод - Положение карты  лицом вверх
61:         {
62:         face_up = true;
63:         }
64:
65:         public   void  faceDown()  // метод - Положение карты  лицом вниз
66:         {
67:                 face_up = false;
68:         }
69:
70:         public    String display()     // формирование строк вывода на печать
71:         {
72:                  String display;
73:
74:                 if (rank > 10)
75:                 {
76:                         display =  Convert.ToString((char) rank);  //  = J, Q, K или A
77:                 }
78:                 else
79:                 {
80:                         display =  Convert.ToString(rank);   // = 2, 3, … или 10
81:                 }
82:
83:         switch (suit)                                          // suit = 3, 4 , 5  или 6
84:         {
85:         case DIAMONDS:                                                  // бубны (4)  ♦
86:         return display +  Convert.ToString((char) DIAMONDS);
87:         case HEARTS:                                                    // черви (3)  ♥
88:         return display +  Convert.ToString((char) HEARTS);
89:         case SPADES:                                                    // пики   (6)  ♠
90:         return display +  Convert.ToString((char) SPADES);
91:         default:                                                        // трефы (5)  ♣
92:         return display +  Convert.ToString((char) CLUBS);
93:         }
94:         }
95: }
```

Класс **Card** отвечает за сохранение **масти (1)** (**строки 3 и 36…42**) и **достоинства (2)** карт (**строки 4 и 44…49**). Кроме того, карта также знает, как вернуть свое строковое (типа **string**) **представление (3)** (**см. выше строки 70…95**).

Сначала в классе **Card** определяется множество констант. Эти константы перенумеровывают существующие достоинства и масти карт (**см. выше строки 9…26**).

Заметьте, что если карте приписано достоинство, то изменить его невозможно (**объявлены const**). Поэтому экземпляры класса **Card** не изменяются. И поскольку карту изменить нельзя, то никто не сможет изменить ее достоинство **(важно)**.

В **листинге 2а** представлена одна из возможных реализаций класса **Deck (колода карт)**.

Листинг 2а. файл Deck.CS

```
Класс Deck создает колоду карт (1), а также (2) содержит методы:
2.1. для перетасовки колоды, 2.2. для извлечения и 2.3. для возврата карт
1:  public class Deck
2:      {
3:
4:      private System.Collections.ArrayList deck; // динамический массив
5:
6:      public Deck()
7:      {
8:          buildCards(); // обращение к методу - создание колоды карт
9:      }
10:
11:     public  Card get_Renamed(int index)
12:     {
13:         if (index < deck.Count)     // Count - свойство класса ArrayList
14:         {
15:             return (Card) deck[index];
16:         }
17:             return null;
18:     }
19:
20:     public   void  replace(int index, Card card)        // перестановка карт
21:     {
22:         deck[index] = card;
23:     }
24: // Count возвращает кол-во эл-тов в массиве desk, т.е. к-во карт в колоде
25:     public   int size()
26:     {
27:         return deck.Count;          // Count - свойство класса ArrayList
28:     }
29:
30:     public   Card  removeFromFront()      // метод – извлечение карты
31:     {
32:         if (deck.Count > 0)                 // свойство класса ArrayList
33:         {
34:             Object tempObject;
35:             tempObject = deck[0];
36:             deck.RemoveAt(0);           // метод класса ArrayList
37:             Card card = (Card) tempObject;
38:             return card;
39:         }           // м-д RemoveAt(0) – извлечение карты из колоды
40:         return null;
```

```
41:         }
42:
43:   public   void  returnToBack(Card card) // метод – возврат карты в колоду
44:         {
45:                 deck.Add(card);
46:         }
47:
48:         private void  buildCards()       // метод - создание колоды карт
49:         {
50:            // объект deck класса ArrayList()- динамический массив
51:                         deck = new System.Collections.ArrayList();
52: //трефы (5) ♣           Add - метод класса ArrayList
53:                 deck.Add(new Card(Card.CLUBS, Card.TWO));
54:                 deck.Add(new Card(Card.CLUBS, Card.THREE));
55:                 deck.Add(new Card(Card.CLUBS, Card.FOUR));
56:                 deck.Add(new Card(Card.CLUBS, Card.FIVE));
57:                 deck.Add(new Card(Card.CLUBS, Card.SIX));
58:                 deck.Add(new Card(Card.CLUBS, Card.SEVEN));
59:                 deck.Add(new Card(Card.CLUBS, Card.EIGHT));
60:                 deck.Add(new Card(Card.CLUBS, Card.NINE));
61:                 deck.Add(new Card(Card.CLUBS, Card.TEN));
62:                 deck.Add(new Card(Card.CLUBS, Card.JACK));
63:                 deck.Add(new Card(Card.CLUBS, Card.QUEEN));
64:                 deck.Add(new Card(Card.CLUBS, Card.KING));
65:                 deck.Add(new Card(Card.CLUBS, Card.ACE));
66: //пики   (6) ♠
67:                         deck.Add(new Card(Card.SPADES, Card.TWO));
68:                         deck.Add(new Card(Card.SPADES, Card.THREE));
69:                         deck.Add(new Card(Card.SPADES, Card.FOUR));
70:                         deck.Add(new Card(Card.SPADES, Card.FIVE));
71:                         deck.Add(new Card(Card.SPADES, Card.SIX));
72:                         deck.Add(new Card(Card.SPADES, Card.SEVEN));
73:                         deck.Add(new Card(Card.SPADES, Card.EIGHT));
74:                         deck.Add(new Card(Card.SPADES, Card.NINE));
75:                         deck.Add(new Card(Card.SPADES, Card.TEN));
76:                         deck.Add(new Card(Card.SPADES, Card.JACK));
77:                         deck.Add(new Card(Card.SPADES, Card.QUEEN));
78:                         deck.Add(new Card(Card.SPADES, Card.KING));
79:                         deck.Add(new Card(Card.SPADES, Card.ACE));
80: //черви (3) ♥
81:                 deck.Add(new Card(Card.HEARTS, Card.TWO));
82:                 deck.Add(new Card(Card.HEARTS, Card.THREE));
83:                 deck.Add(new Card(Card.HEARTS, Card.FOUR));
84:                 deck.Add(new Card(Card.HEARTS, Card.FIVE));
85:                 deck.Add(new Card(Card.HEARTS, Card.SIX));
86:                 deck.Add(new Card(Card.HEARTS, Card.SEVEN));
87:                 deck.Add(new Card(Card.HEARTS, Card.EIGHT));
88:                 deck.Add(new Card(Card.HEARTS, Card.NINE));
89:                 deck.Add(new Card(Card.HEARTS, Card.TEN));
90:                 deck.Add(new Card(Card.HEARTS, Card.JACK));
91:                 deck.Add(new Card(Card.HEARTS, Card.QUEEN));
92:                 deck.Add(new Card(Card.HEARTS, Card.KING));
93:                 deck.Add(new Card(Card.HEARTS, Card.ACE));
94: // бубны (4) ♦
95:                         deck.Add(new Card(Card.DIAMONDS, Card.TWO));
```

```
96:                         deck.Add(new Card(Card.DIAMONDS, Card.THREE));
97:                         deck.Add(new Card(Card.DIAMONDS, Card.FOUR));
98:                         deck.Add(new Card(Card.DIAMONDS, Card.FIVE));
99:                         deck.Add(new Card(Card.DIAMONDS, Card.SIX));
100:                        deck.Add(new Card(Card.DIAMONDS, Card.SEVEN));
101:                        deck.Add(new Card(Card.DIAMONDS, Card.EIGHT));
102:                        deck.Add(new Card(Card.DIAMONDS, Card.NINE));
103:                        deck.Add(new Card(Card.DIAMONDS, Card.TEN));
104:                        deck.Add(new Card(Card.DIAMONDS, Card.JACK));
105:                        deck.Add(new Card(Card.DIAMONDS, Card.QUEEN));
106:                        deck.Add(new Card(Card.DIAMONDS, Card.KING));
107:                        deck.Add(new Card(Card.DIAMONDS, Card.ACE));
108:            }
109:    }
```

Класс **Deck** отвечает за **создание экземпляров карт (1)**, а также **обеспечивает доступ к ним (2)**. **Deck** снабжен методами для **извлечения (3)** и **возврата (4)** карт.

В **листинге 3а** представлена реализация раздающего карты (класс **Dealer**).

Листинг 3а. файл Dealer.CS

```
                Это класс раздающего карты.
        Dealer умеет тасовать (shuffle) (1) и раздавать (deal) (2) карты
1:  public class Dealer
2:          {
3:        // колода карт deck в прямую недоступна
4:          private Deck deck;
5:
6:          public Dealer(Deck d)                   // конструктор
7:          {
8:                  deck = d;
9:          }
10:
11:         public   void  shuffle()          // карты тасуются
12:         {
13:                 // расположить карты в массиве карт в случайном порядке
14:                 int num_cards = deck.size();              // с. 7, л-г 2а, стр. 25…
15:                 for (int i = 0; i < num_cards; i++)
16:                 {                                  // класс SupportClass – см. с.10, стр.34..
17:         int index = (int) (SupportClass.Random.NextDouble() * num_cards);
18:                 Card card_i =       (Card) deck.get_Renamed(i); // с. 7, л-г 2а, стр. 11..
19:                 Card card_index = (Card) deck.get_Renamed(index);
20:                         deck.replace(i, card_index);              // с. 7, л-г 2а, стр. 20..
21:                         deck.replace(index, card_i);
22:                 }
23:         }                                  // index – индекс массива карт
24:
25:         public   Card dealCard()                    // этот метод не используется
26:         {
27:                 if (deck.size() > 0)
28:                 {
29:                         return deck.removeFromFront();
30:                 }
31:                 return null;
32:         }
```

```
33:         // внутренний класс SupportClass
34:         public class SupportClass
35:         {       // обеспечивается доступ к статической переменной Random
36:             static public System.Random Random = new System.Random();
37:         }
38: }
```

Раздающий карты (**Dealer**) отвечает: **1) за тасование** колоды, **2) за раздачу** карт.

В данной реализации класса **Dealer** раздающий карты начинает раздачу **с начала колоды**!

Ответственность распределена (3) между всеми представленными здесь тремя классами:

- класс **Card** представляет собой карты для игры.
- класс **Deck** хранит карты,
- класс **Dealer** раздает карты игрокам.

Все три класса скрывают свою реализацию. Ничто даже не наводит на мысль, что в классе **Deck** колода фактически хранится в виде динамического массива (класс **ArrayList**) карт **(строка 4:)**.

Хотя в классе **Card** было определено множество констант **(строки 9:…26:)**, это не вредит целостности его реализации, поскольку класс **Card** может использовать константы по своему усмотрению. Он также может в любое время изменить значения констант.

С помощью метода **buildCards()** класса Deck (колода) **(строки 48: … 108:)** можно увидеть недостаток сокрытия реализации. В цикле **for** можно создать карты с достоинством от двойки до десятки. Если вы посмотрите на константы, то увидите, что последовательность констант от **TWO** до **TEN** соответствует последовательности чисел от **2** до **10**. Такой цикл значительно проще, чем отдельное создание экземпляра каждой карты.

Тем не менее, такое допущение привязывает вас к текущим значениям констант. Но вообще-то программа не должна зависеть от конкретных значений, закрепленных за константами. Вместо этого следовало бы использовать константы **Card.TWO**, **Card.THREE** и т.д. Не следует делать какого-либо предположения о достоинстве карты. **Класс Card** мог бы переопределить постоянные величины в любое время. В случае метода **buildCards()** легко прельститься и непосредственно использовать постоянные величины.

В данной программе сотрудничество между **классом Card** и его пользователем основано на том, что постоянными остаются имена констант, а не их значения **(строки 9:…26:)**. В дальнейшем будет представлено решение, чуть более элегантное, чем использование констант.

Листинг 4а. Класс CardDriver

```
Это начальный класс CardDriver распечатывает колоду,
тасует ее, а затем распечатывает ее снова
1: public class CardDriver
2: {
3:         public static void  Main( String[ ] args)
4:         {
5: Deck   deck   =  new Deck();     // объект  deck - колода карт.     с. 7, л-г 2а, стр. 6…
6: Dealer dealer = new Dealer(deck);  // объект  deck - передается дилеру.     с. 9, л-г 3а, стр. 6…
7:    Console.WriteLine("\nРаспечатка упорядоченной колоды карт");
8:                 printDeck(deck); // распечатать упорядоченную колоду карт, с.11, стр. 18…29
9:
```

```
10:                     dealer.shuffle(); // перетасовать колоду карт.  с. 9, л-г 3а, стр. 11…
11:
12:     Console.WriteLine("\nРаспечатка перетасованной колоды карт");
13:                 printDeck(deck); // распечатать перетасованную  колоды карт, с.11, стр. 18…29
14:
15:                     Console.ReadLine();
16:             }
17:
18:             public static void  printDeck(Deck deck) // метод – печать массива карт
19:             {
20:                     for (int i = 0; i < 4; i++)                 // четыре масти
21:                     {
22:                             for (int j = 0; j < 13; j++)    // карты 13-ти достоинств
23:                             {
24:                                     Card card = deck.removeFromFront();         // с. 7, л-г 2а, стр. 30…
25:                                     deck.returnToBack(card);                    // с. 8, л-г 2а, стр. 43…
26:                     Console.Write(card.display() + " ");    // с. 6,  л-г 1а, строки 70 … 95
27:                             }
28:                              Console.WriteLine(" ");
29:                     }
30:             }
31: }
```

Метод **Main()** выполняет следующее: **1)** создает **экземпляр** класса **dealer** (**раздающий карты**), **2)** создает **экземпляр** класса **deck** of cards (**колода карт**), **3) перетасовывает** карты, **4) распечатывает** полученную колоду карт.

1.1.3. Результаты работа программы

Распечатка упорядоченной колоды карт

2♣ 3♣ 4♣ 5♣ 6♣ 7♣ 8♣ 9♣ 10♣ J♣ Q♣ K♣ A♣

2♠ 3♠ 4♠ 5♠ 6♠ 7♠ 8♠ 9♠ 10♠ J♠ Q♠ K♠ A♠

2♥ 3♥ 4♥ 5♥ 6♥ 7♥ 8♥ 9♥ 10♥ J♥ Q♥ K♥ A♥

2♦ 3♦ 4♦ 5♦ 6♦ 7♦ 8♦ 9♦ 10♦ J♦ Q♦ K♦ A♦

Распечатка перетасованной колоды карт

4♥ 10♦ 5♠ 8♣ 5♦ 9♣ A♣ 5♣ 7♦ 4♣ 3♣ 2♦ A♠

7♥ J♠ J♣ 10♠ 9♥ 9♦ 2♠ Q♥ 10♣ Q♠ 4♦ 5♥ K♥

J♦ 6♠ 3♥ K♣ 7♠ 9♠ 2♣ 8♥ 3♦ A♦ J♥ 7♣ 2♥

Q♣ 10♥ 3♠ 8♠ 4♠ 8♦ Q♦ 6♥ 6♦ A♥ 6♣ K♦ K♠

1.1.4. Диаграмма классов 1

2. Многократное использование проектов с помощью шаблонов проектирования

Применяем шаблоны проектирования с целью повышения качества проекта.

A. Многократное использование проекта

Одной из основных задач объектно-ориентированного программирования (ООП) является многократное использование кода. При многократном использовании кода можно чувствовать себя спокойнее, зная, что программа использует надежный, проверенный временем код.

В течение долгих лет программирования многие проектировщики и программисты замечали, что повсюду в их проектах много раз появляются те же самые модули (элементы проекта). Сообщество сторонников объектно-ориентированного подхода решило идентифицировать, назвать и описать эти повторяющиеся концепции проектирования. Результат — постоянно растущий список шаблонов проектирования.

Шаблон проектирования — это концепция проектирования, которую можно применять многократно.

Оказывается, мы можем многократно применять шаблон проектирования точно так же, как мы многократно используем классы при программировании. Это позволяет перенести все преимущества многократного использования из объектно-ориентированного программирования (ООП) в объектно-ориентированное проектирование (ООПр). При использовании шаблонов проектирования мы можем быть уверены, что наш проект основан на надежных, проверенных временем проектах. При этом мы получаем дополнительное подтверждение того, что мы на верном пути к надежному решению. При очередном применении уже существующего шаблона проектирования мы применяем тот проект, который неоднократно успешно применялся ранее.

B. Шаблоны проектирования

Шаблоны проектирования — многократно используемые элементы объектно-ориентированного программного обеспечения.

Шаблон проектирования состоит из четырех элементов:

a) Название шаблона
b) Задача
c) Решение
d) Последствия

***a)* Название шаблона.** Название позволяет однозначно идентифицировать каждый шаблон проектирования. Точно так же, как UML определяет универсальный язык проектирования, названия шаблонов составляют общий словарь, при помощи которого описываются элементы проекта. Это позволит другим разработчикам легко и быстро разобраться в вашем проекте.
Название позволяет свести задачу, ее решение и последствия до уровня одного термина. Так же, как объекты переводят программирование на более высокий уровень абстракции, применение таких терминов позволит заниматься проектированием на более высоком, более абстрактном уровне, а не увязать в мелочах, которые повторяются из проекта в проект.

***b)* Задача.** Каждый шаблон проектирования создается для решения задач из некоего дискретного набора, причем в каждом шаблоне должно быть описано множество решаемых им задач. Значит, описание задач помогает определить, применим ли шаблон проектирования к конкретной задаче.

***c)* Решение.** В решении описывается, каким образом задача решается при помощи шаблона проектирования, и определяются все архитектурно важные объекты в решении, их назначения и взаимосвязи.

Примечание Следует отметить, что шаблоны проектирования применимы для целых классов задач. Решение является общим решением, а не ответом на конкретные вопросы.

Допустим, что мы хотели бы узнать, как наилучшим образом просмотреть список товаров в тележке для магазинов самообслуживания из лекции 14 "Введение в объектно-ориентированное проектирование". Для решения выбирается шаблон проектирования Iterator (Итератор). Однако решение, предлагаемое этим шаблоном, сформулировано не в терминах товаров и тележки для магазинов самообслуживания. Оно описывает процесс просмотра элементов любого списка.

При использовании шаблонов проектирования следует интерпретировать общее решение в терминах конкретной задачи. Иногда такую интерпретацию дать нелегко. Тем не менее, шаблоны проектирования должны быть достаточно общими, чтобы их можно было применять ко многим конкретным задачам.

***d)* Последствия.** Идеальных проектов не существует. В каждом хорошем проекте придется искать оптимальные компромиссы, и у каждого компромисса будут свои последствия. В шаблоне проектирования должны быть перечислены все те последствия, которые приобретет проект.

Последствия применения шаблона проектирования не являются чем-то новым. Каждый раз при выборе между альтернативными путями построения программы приходится идти на уступки. Возьмем, к примеру, выбор между представлением *последовательности:* **1)** в виде массива или **2)** в виде связного списка. Массив **(1)** позволяет **быстро** получать доступ к данным по индексу, но работает гораздо медленнее, когда нужно вставить или удалить элементы из **середины** последовательности. С другой стороны, связный список **(2)** позволяет легко добавлять или удалять элементы, но менее рационально использует память, да и просмотр списка выполняется медленнее. В данном случае последствиями использования связного списка будут менее рациональное использование памяти и более медленный просмотр, последствиями использования массива будут, соответственно, сложности с изменением размера массива и быстрый доступ по индексу элемента [6, с. 450] . Выбор представления следует делать в зависимости от требований к объему памяти и быстродействию программы, частоты добавления и удаления элементов, а также частоты просмотра.

C. Практика применения шаблонов проектирования

При первом знакомстве с шаблонами проектирования следует сразу понять, что могут и чего не могут делать шаблоны проектирования. Следующие списки помогут четко определить назначение шаблонов:

Шаблоны — это:

- Проекты многократного использования, которые доказали свое право на жизнь
- Абстрактные решения общих проблем проектирования
- Решения часто возникающих задач
- Способ построения словаря проекта
- Общедоступная запись опыта проектирования
- Решение задач одного типа

Шаблоны не являются:

- Решением конкретной задачи
- Волшебным средством для решения всех задач
- Опорой для решения поставленной перед вами задачи; проектировать решение поставленной перед вами задачи все равно придется самостоятельно
- Конкретными классами, библиотеками, готовыми решениями

2.1. Использование шаблона Итератор (Iterator)

В табл. 1 очерчена область применения шаблона Итератор (**Iterator**).

Таблица 1. Шаблон Итератор (**Iterator**)

Название шаблона	Итератор (**Iterator**)
Задача	Перебор элементов коллекции вне зависимости от его реализации
Решение	Создается объект, который занимается деталями перебора, – последние прячутся от пользователя
Последствия	Развязка обхода, упрощение интерфейса коллекции, инкапсуляция логики перебора

С помощью шаблона итератор Iterator реализуем механизм перебора элементов в следующей коллекции – колода карт (см. листинги в разд. 2.1.1): осуществим перебор колоды в прямом и обратном порядке, а также перебор содержимого массива (см. результаты работы программы в разд. 2.1.2).

2.1.1. Код программы (см. разд. 1.1.2)

В листинге 5 приведен интерфейс объекта **Iterator** (Итератор).

Интерфейс объекта **Iterator** (Итератор) является универсальным для перебора элементов коллекции. Вместо того, чтобы писать методы перебора для каждой отдельной коллекции, можно использовать общий интерфейс объекта **Iterator**, который скрывает реализацию коллекции.

см. папку CAp_J_g11_Iterator_c264.NET

Листинг 5. *Интерфейс объекта* **Iterator**

```
1:  public interface Iterator
2:  {
3:      bool Done  // свойство Done - признак конца коллекции
4:      {
5:      get;
6:      }
7:                  void  first();  // первый элемент коллекции (0-й)
8:                        void  next(); // след-й элемент коллекции
9:    System.Object currentItem();// текущ-й элемент коллекции
10: }
```

Перебор колоды карт в прямом порядке (упорядоченно)

Листинг 6. *Реализация (***Forward***) интерфейса* **Iterator**

```
1:  public class ForwardIterator : Iterator
2:  {
3:          private System.Object[ ] items; // предметы (карты)
4:          private int index;
5:
6:  public ForwardIterator(System.Collections.ArrayList items)
7:          {                    // items – динамический массив
8:          this.items = items.ToArray(); // м-д класса ArayList
9:          }
10:
11:         virtual public bool Done      // с. 15, л-г 5, стр. 3…
```

```
12:         {
13:                 get
14:                 {
15:                         if (index >= items.Length)
16:                         {
17:                                 return true;
18:                         }
19:                         return false;
20:                 }
21:         }
22:
23:         public virtual void  first()       // с. 15, л-г 5, стр. 7
24:         {
25:                 index = 0;
26:         }
27:
28:         public virtual void  next()        // с. 15, л-г 5, стр. 8
29:         {
30:                 index++;
31:         }
32:
33:         public virtual System.Object currentItem()
34:         {                                  // с. 15, л-г 5, стр. 9
35:                 if (!Done)       // если еще не все сделано
36:                 {
37:                         return items[index];
38:                 }
39:                 return null;
40:         }
41:
42: }
```

Перебор колоды карт в обратном порядке (упорядоченно)

Листинг 7. *Реализация (**Reverse**) интерфейса* **Iterator**

```
1:          public class ReverseIterator : Iterator
2:          {
3:                  private System.Object[ ] items; // предметы (карты)
4:                  private int index;
5:
6:  public ReverseIterator(System.Collections.ArrayList items)
7:                  {                   // items – динамический массив
8:                          this.items = items.ToArray();// м-д класса ArayList
8a:                         first();                        // с. 15, л-г 5, стр. 7
9:                  }
10:
11:                 virtual public bool Done      // с. 15, л-г 5, стр. 3…
12:                 {
```

```
13:                         get
14:                         {
15:                             if (items.Length == 0 || index <= - 1)
16:                             {
17:                                 return true;
18:                             }
19:                             return false;
20:                         }
21:                 }
22:
23:                 public virtual void  first()      // с. 15, л-г 5, стр. 7
24:                 {
25:                         index = items.Length - 1;
26:                 }
27:
28:                 public virtual void  next()       // с. 15, л-г 5, стр. 8
29:                 {
30:                         index--;
31:                 }
32:
33:                 public virtual System.Object currentItem()
34:                 {                                          // с. 15, л-г 5, стр. 9
35:                         if (!Done)  // если еще не все сделано
36:                         {
37:                             return items[index];
38:                         }
39:                         return null;
40:                 }
41:
42:     }
```

Листинг 2**b**. Класс Deck создает колоду карт (1), а также (2) содержит методы: 2.1. для перетасовки колоды, 2.2. для извлечения и 2.3. для возврата карт

```
1:  public class Deck
2:  {
3:          private System.Collections.ArrayList deck; // deck – динамический массив
4:
5:          public Deck()
6:          {
7:                  buildCards();// обращение к методу - создание колоды карт
8:          }
9:
10:                 public virtual Iterator iterator()                  // добавление
11:                 {               // стр. 10…13 - это единственное отличие от л-га 2a, с. 7…9
12:                     return new ForwardIterator(deck);               // с. 15, л-г 6, стр. 6…9
13:                 }
14:
15:                 public virtual Card get_Renamed(int index)
```

```
16:         {
17:             if (index < deck.Count)
18:             {                       // deck[index] – это элемент массива
19:                 return (Card) deck[index];
20:             }
21:             return null;
22:         }
23:
24:     public virtual void  replace(int index, Card card)              // перестановка карт
25:     {
26:         deck[index] = card;
27:     }
28:     // Count возвращает кол-во эл-тов в массиве desk, т.е. к-во карт в колоде
29:         public virtual int size()
30:         {
31:             return deck.Count;              // Count - свойство класса ArrayList
32:         }
33:
34:     public virtual Card removeFromFront()               // метод – извлечение карты
35:     {
36:         if (deck.Count > 0)                             // свойство класса ArrayList
37:         {
38:             System.Object tempObject;
39:             tempObject = deck[0];
40:             deck.RemoveAt(0);                           // метод класса ArrayList
41:             Card card = (Card) tempObject;
42:             return card;
43:         }
44:         return null;
45:     }
46:
47:     public virtual void  returnToBack(Card card)    // метод – возврат карты
48:     {
49:         deck.Add(card);                 // Add – м-д класса ArrayList
50:     }
51:
52:         private void  buildCards()// метод - создание колоды карт
53:         {                       // объект deck класса ArrayList()- динамический массив
54:         deck = new System.Collections.ArrayList();
55:                                                                     // трефы (5) ♣
56:         deck.Add(new Card(Card.CLUBS, Card.TWO));
57:         deck.Add(new Card(Card.CLUBS, Card.THREE));
58:         deck.Add(new Card(Card.CLUBS, Card.FOUR));
59:         deck.Add(new Card(Card.CLUBS, Card.FIVE));
60:         deck.Add(new Card(Card.CLUBS, Card.SIX));
61:         deck.Add(new Card(Card.CLUBS, Card.SEVEN));
62:         deck.Add(new Card(Card.CLUBS, Card.EIGHT));
```

```
63:                 deck.Add(new Card(Card.CLUBS, Card.NINE));
64:                 deck.Add(new Card(Card.CLUBS, Card.TEN));
65:                 deck.Add(new Card(Card.CLUBS, Card.JACK));
66:                 deck.Add(new Card(Card.CLUBS, Card.QUEEN));
67:                 deck.Add(new Card(Card.CLUBS, Card.KING));
68:                 deck.Add(new Card(Card.CLUBS, Card.ACE));
69:                                                 // пики  (6) ♠
70:             deck.Add(new Card(Card.SPADES, Card.TWO));
71:             deck.Add(new Card(Card.SPADES, Card.THREE));
72:             deck.Add(new Card(Card.SPADES, Card.FOUR));
73:             deck.Add(new Card(Card.SPADES, Card.FIVE));
74:             deck.Add(new Card(Card.SPADES, Card.SIX));
75:             deck.Add(new Card(Card.SPADES, Card.SEVEN));
76:             deck.Add(new Card(Card.SPADES, Card.EIGHT));
77:             deck.Add(new Card(Card.SPADES, Card.NINE));
78:             deck.Add(new Card(Card.SPADES, Card.TEN));
79:             deck.Add(new Card(Card.SPADES, Card.JACK));
80:             deck.Add(new Card(Card.SPADES, Card.QUEEN));
81:             deck.Add(new Card(Card.SPADES, Card.KING));
82:             deck.Add(new Card(Card.SPADES, Card.ACE));
83:                                                 // черви (3) ♥
84:                 deck.Add(new Card(Card.HEARTS, Card.TWO));
85:                 deck.Add(new Card(Card.HEARTS, Card.THREE));
86:                 deck.Add(new Card(Card.HEARTS, Card.FOUR));
87:                 deck.Add(new Card(Card.HEARTS, Card.FIVE));
88:                 deck.Add(new Card(Card.HEARTS, Card.SIX));
89:                 deck.Add(new Card(Card.HEARTS, Card.SEVEN));
90:                 deck.Add(new Card(Card.HEARTS, Card.EIGHT));
91:                 deck.Add(new Card(Card.HEARTS, Card.NINE));
92:                 deck.Add(new Card(Card.HEARTS, Card.TEN));
93:                 deck.Add(new Card(Card.HEARTS, Card.JACK));
94:                 deck.Add(new Card(Card.HEARTS, Card.QUEEN));
95:                 deck.Add(new Card(Card.HEARTS, Card.KING));
96:                 deck.Add(new Card(Card.HEARTS, Card.ACE));
97:                                                 // бубны (4) ♦
98:             deck.Add(new Card(Card.DIAMONDS, Card.TWO));
99:             deck.Add(new Card(Card.DIAMONDS, Card.THREE));
100:            deck.Add(new Card(Card.DIAMONDS, Card.FOUR));
101:            deck.Add(new Card(Card.DIAMONDS, Card.FIVE));
102:            deck.Add(new Card(Card.DIAMONDS, Card.SIX));
103:            deck.Add(new Card(Card.DIAMONDS, Card.SEVEN));
104:            deck.Add(new Card(Card.DIAMONDS, Card.EIGHT));
105:            deck.Add(new Card(Card.DIAMONDS, Card.NINE));
106:            deck.Add(new Card(Card.DIAMONDS, Card.TEN));
107:            deck.Add(new Card(Card.DIAMONDS, Card.JACK));
108:            deck.Add(new Card(Card.DIAMONDS, Card.QUEEN));
109:            deck.Add(new Card(Card.DIAMONDS, Card.KING));
```

```
110:            deck.Add(new Card(Card.DIAMONDS, Card.ACE));
111:                    }
112:
113:    }
```

Листинг 1а. Это техническое представление карты в виде класса Card. Карты отличаются только значением масти и достоинством, а также положением

```
1:  public class Card
2:  {
3:          private int rank;   // достоинство карты
4:          private int suit;  // масть карт
5:          private bool face_up; // признак положения карты: вверх – true, вниз – false
6:
7:      // использованы константы для обозначения:
8:                                              // 1) масти карт (suits) – 10-тичный ASCII-код:
9:          public const int DIAMONDS = 4; // бубны♦
10:         public const int HEARTS      = 3;       // червы♥
11:         public const int SPADES      = 6;             // пики♠
12:         public const int CLUBS       = 5;              // трефы♣
13:                                             // 2) достоинства карт (rank):
14:         public const int TWO      =  2;
15:         public const int THREE    =  3;
16:         public const int FOUR     =  4;
17:         public const int FIVE     =  5;
18:         public const int SIX      =  6;
19:         public const int SEVEN    =  7;
20:         public const int EIGHT    =  8;
21:         public const int NINE     =  9;
22:         public const int TEN      = 10;
23:         public const int JACK     = 74;      // валет – 10-тичный ASCII-код: J
24:         public const int QUEEN = 81;      // дама                          : Q
25:         public const int KING     = 75;      // король                        : K
26:         public const int ACE      = 65;      // туз                           : A
27:
28:                                     // для создания новой карты –
                                    используются только инициализированные константы
29:         public Card(int suit, int rank)
30:         {
31:                 this.suit = suit;
32:                 this.rank = rank;
33:         }
34:
35:         virtual public int Suit  // свойство – получить значение масти карты
36:         {
37:                 get
38:                 {
39:                         return suit;
40:                 }
41:         }
42:
43:         virtual public int Rank  // свойство – получить значение достоинства карты
```

```
44:         {
45:                 get
46:                 {
47:                         return rank;
48:                 }
49:         }
50:
51: virtual public bool FaceUp  // свойство – получить значение признака положения карты
52:         {
53:                 get
54:                 {
55:                         return face_up;
56:                 }
57:         }
58:
59:         public virtual void  faceUp()  // метод - Положение карты  лицом вверх
60:         {
61:                 face_up = true;
62:         }
63:
64:         public virtual void  faceDown()  // метод - Положение карты  лицом вниз
65:         {
66:                 face_up = false;
67:         }
68:
69:         public virtual System.String display()    // формирование строк вывода на печать
70:         {
71:                 System.String display;
72:
73:                 if (rank > 10)
74:                 {
75:                         display = System.Convert.ToString((char) rank);  // = J, Q, K или A
76:                 }
77:                 else
78:                 {
79:                         display = System.Convert.ToString(rank);   // = 2, 3, … или 10
80:                 }
81:
82:                 switch (suit)                                   // suit = 3, 4 , 5  или 6
83:                 {
84:                 case DIAMONDS:                                          // бубны (4) ♦
85:         return display + System.Convert.ToString((char) DIAMONDS);
86:                 case HEARTS:                                            //черви (3) ♥
87:         return display + System.Convert.ToString((char) HEARTS);
88:                 case SPADES:                                            //пики   (6) ♠
89:         return display + System.Convert.ToString((char) SPADES);
90:                 default:                                                //трефы (5) ♣
91:          return display + System.Convert.ToString((char) CLUBS);
92:                 }
93:         }
94: }
```

Листинг 4b.Это начальный класс CardDisplayDriver распечатывает колоду

```
1:  public class CardDisplayDriver
2:  {
3:
4:          public virtual System.String deckToString(Deck deck)
5:          {     // перебор содержимого колоды карт (прямой)
6:                  System.String cards = "";
7:                  for (int i = 0; i < deck.size(); i++) // с. 18, л-г 2b, стр. 29…
8:                  {
9:                          Card card = deck.get_Renamed(i); // с. 17, л-г 2b, стр. 15…
10:                         cards = cards + card.display();  // с. 21, л-г 1a, стр. 69…93
11:                 }
12:                 return cards;
13:         }
14:
15:         public virtual System.String deckToString(Iterator i)
16:         { // перебор содержимого экз-ра i интерфейса Iterator (прямой)
17:                 System.String cards = "";
18:                 for (i.first(); !i.Done; i.next()) // с. 15 и 16, л-г 6, стр. 23…, 11…, 28…
19:                 {
20:                         Card card = (Card) i.currentItem(); // с. 16, л-г 6, стр. 23…
21:                         cards = cards + card.display();  // с. 21, л-г 1a, стр. 69…93
22:                 }
23:                         return cards;
24:         }
25:
26:         public virtual System.String deckToString(Card[ ] deck)
27:         {        // перебор содержимого массива карт
28:                 System.String cards = "";
29:                 for (int i = 0; i < deck.Length; i++)
30:                 {
31:                         cards = cards + deck[i].display();  // с. 21, л-г 1a, стр. 69…93
32:                 }
33:                 return cards;
34:         }
35:
36:         public virtual System.String reverseDeckToString(Deck deck)
37:         { // перебор содержимого колоды карт (реверс)
38:                 System.String cards = "";
39:                 for (int i = deck.size() - 1; i > - 1; i--)  // с. 18, л-г 2b, стр. 29…
40:                 {
41:                         Card card = deck.get_Renamed(i);  // с. 17, л-г 2b, стр. 15…22
42:                         cards = cards + card.display();      // с. 21, л-г 1a, стр. 69…93
43:                 }
44:                 return cards;
45:         }
46:
47:         public static void  Main(System.String[ ] args)
48:         {
49:         Deck deck = new Deck();    // с. 17, л-г 2b, стр. 5; создание колоды карт
```

```
50:
51:                 Card[ ] cards = new Card[3];   // с. 20, л-г 1a
52:                 cards[0] = deck.get_Renamed(0);    // с. 17, л-г 2b, стр. 15…22
53:                 cards[1] = deck.get_Renamed(1);    // - « -
54:                 cards[2] = deck.get_Renamed(2);     // - « -
55:
56:           CardDisplayDriver display = new CardDisplayDriver();
57:
58:      System.Console.Out.WriteLine("\nПрямой ход");
59:      System.Console.Out.WriteLine(display.deckToString(deck));     // стр. 4…13
60:
61:      System.Console.Out.WriteLine("\nРеверс");
62:      System.Console.Out.WriteLine(display.reverseDeckToString(deck));    // стр. 36…45
63:
64:      System.Console.Out.WriteLine("\nМассив из 3-х карт");          // стр. 51…53
65:      System.Console.Out.WriteLine(display.deckToString(cards));    // стр. 26…34
66:
67:              System.Console.Out.WriteLine("\nПрямой ход");
68:          Iterator i = deck.iterator(); // i – экз-р интерфейса Iterator; с. 17, л-г 2b, стр. 10…13
69:                 System.Console.Out.WriteLine(display.deckToString(i));    // стр. 15…24
70:
71:       Console.ReadLine();
72:           }
73:  }
```

2.1.2. Результаты работы программы: иллюстрация использования шаблона Итератор

Прямой ход (см. строку 59:); перебор содержимого колоды карт (прямой) (см. строки 4: … 14:)

2♣3♣4♣5♣6♣7♣8♣9♣10♣J♣Q♣K♣A♣2♠3♠4♠5♠6♠7♠8♠9♠10♠J♠Q♠K♠A♠2♥3♥4♥5♥6♥7♥8♥9♥10♥J♥Q♥K♥A♥2♦3♦4♦5♦6♦7♦8♦9♦10♦J♦Q♦K♦A♦

Реверс (см. строку 62:); перебор содержимого колоды карт (реверс) (см. строки 36: … 45:)

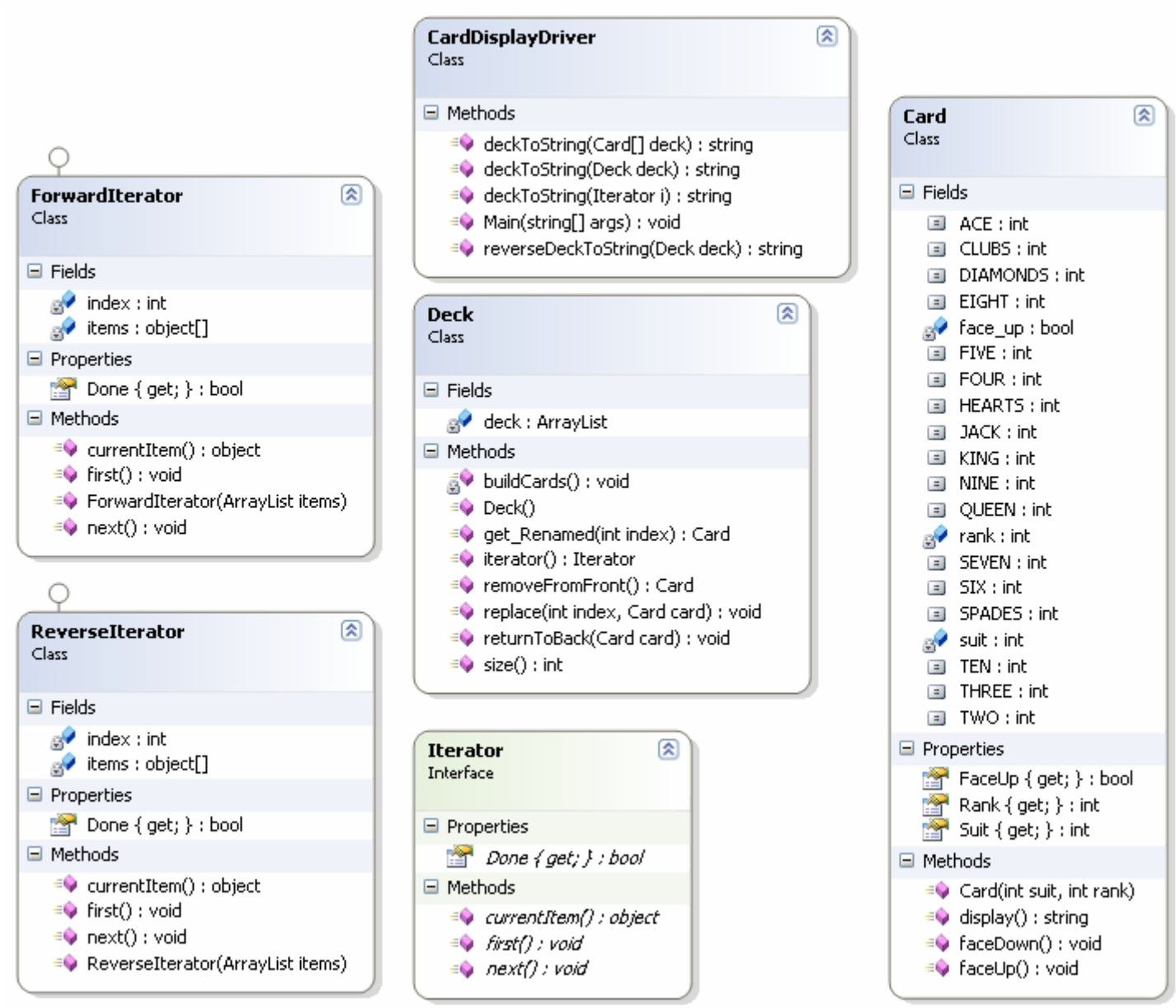
A♦K♦Q♦J♦10♦9♦8♦7♦6♦5♦4♦3♦2♦A♥K♥Q♥J♥10♥9♥8♥7♥6♥5♥4♥3♥2♥A♠K♠Q♠J♠10♠9♠8♠7♠6♠5♠4♠3♠2♠A♣K♣Q♣J♣10♣9♣8♣7♣6♣5♣4♣3♣2♣

Массив из 3 карт (см. строку 65:); перебор содержимого массива карт (прямой) (см. строки 26: … 34:)

2♣3♣4♣

Прямой ход(см. строки 68: и 69:); перебор содержимого экз-ра i интерфейса Iterator (прямой) (см. строки 15: … 24:)

2♣3♣4♣5♣6♣7♣8♣9♣10♣J♣Q♣K♣A♣2♠3♠4♠5♠6♠7♠8♠9♠10♠J♠Q♠K♠A♠2♥3♥4♥5♥6♥7♥8♥9♥10♥J♥Q♥K♥A♥2♦3♦4♦5♦6♦7♦8♦9♦10♦J♦Q♦K♦A♦



2.1.3. Диаграмма классов 2: иллюстрация использования шаблона Итератор

3. Шаблон продвинутого проектирования

Шаблон **Typesafe Enum** (перечисление с сохранением типа)

3.1. Особенности класса Card [см. листинг 1а. файл Card.CS, (с. 5,сл.)]

В листинге 1а. файл Card.cs, (с. 5сл.) представлен класс **Card**.

При создании объектов **Card** необходимо передать правильные константы ранга и масти **(строки 9:…26:)**. Подобное использование констант может привести к неисчислимым проблемам. Ничто не мешает вам передавать любое целое (типа int), какое вы хотите. И хотя следует передавать лишь имя константы, возможность передавать любое целое позволяет вскрыть внутреннее представление **Card**. Например, для того, чтобы узнать масть в **Card**, необходимо извлечь целое значение и сравнить его с константами. Это решение нельзя назвать идеальным, хотя оно и работает.

Проблема заключается в том, что и **suit (строки 9:…12:)**, и **rank (строки 14:…26:)** сами по себе являются объектами. Использование целого типа **int** не может решить эту проблему, поскольку ему необходимо присвоить значение. Опять же, это является смешением обязанностей, поскольку каждый раз, когда вы сталкиваетесь с целым типа **int**, которое представляет **rank** или **suit**, необходимо заново определять смысл целого типа **int**.

В языке C#, есть конструкция известная как *перечисление (enumeration)* [см. Ключевые слова языка CSharp, слово #20 - **enum**]. Перечисление, по сути, — это всего лишь быстрый способ объявления списка целых констант. Но эти константы ограничены. К примеру, они не могут представлять поведение. Добавление дополнительных констант также затруднено.

Ключевое слово **enum** используется для объявления перечисления. Перечисление представляет собой список числовых констант

В отличие от перечисления, шаблон перечисления с сохранением типа (Typesafe **Enum**) предлагает объектно-ориентированный подход к объявлению констант. Вместо простого объявления целочисленных констант, для каждого типа констант создается свой собственный класс. Например, для **Card** создаются классы **Rank** и **Suit**. Для каждого константного значения создается экземпляр и делается общедоступным из класса (через объявление **public sealed** подобно другим константам).

От класса, объявленного **sealed,** нельзя создать производных классов.

3.2. Реализация шаблона перечисления с сохранением типа (Typesafe Enum)

3.2.1. Код программы

Рассмотрим реализацию классов **Rank** (Ранг) и **Suit** (Масть) в листингах 8 и 9.

Листинг **8.** Suit.cs **см. папку CAp_C#_g12_c289.NET**

```
1:  // Общий закрытый класс Suit (масть карты)
2:  public sealed class Suit
3:  {
4:      // Статически определяет все допустимые значения масти
5:      public static readonly Suit DIAMONDS = new Suit((char) 4);  // ♦
6:      public static readonly Suit HEARTS = new Suit((char) 3);    //♥
7:      public static readonly Suit SPADES = new Suit((char) 6);    //♠
8:      public static readonly Suit CLUBS = new Suit((char) 5);     //♣
9:
10:     // Помогает выполнить итерации по перечислимым значениям
11: public static readonly Suit[ ] SUIT = new Suit[ ]{DIAMONDS, HEARTS, SPADES, CLUBS};
```

```
12:
13:         // Переменная экземпляра display для хранения отображаемого значения
14:             private readonly char display;
15:
16:                 // Не позволить создавать экземпляры вне объектов
17:             private Suit(char display)              // конструктор
18:             {
19:                 this.display = display;
20:             }
21:
22:  // ToString() – переопределяющий м-д, л-г 1с, с. 28, стр. 40: возвратить значение масти
23:             public override System.String ToString() // ToString() – м-д класса String
24:             {
25:                 return System.Convert.ToString(display);
26:             }
27:         }
```

Класс **Suit** реализован предельно просто. Конструктор принимает переменную символьного типа (**char**), которая представляет **Suit** (**строки 17…20**). Поскольку **Suit** — это полноценный объект, он может иметь собственные методы. В примере **Suit** обладает методом **toString()** (**строки 23…26**). Перечисление с сохранением типа позволяет добавлять любые полезные методы.

Обратите внимание на то, что константа **display** объявлена как **private** (**строка 14**). Это не позволяет объектам создавать экземпляры класса **Suit** напрямую. Разрешается только лишь использовать экземпляры констант, объявленных в классе (**строки 5…8, строка 11**). Кроме того, класс объявляется как **закрытый** (**sealed**), чтобы от него нельзя было создать производные классы. Бывает, что наследование все-таки необходимо. В этом случае конструктор делается защищенным (объявляется как **protected**), а объявление **sealed** (**закрытый**) удаляется.

Обратите внимание на то, что в классе **Suit** определено множество экземпляров констант (**строка 11**), по одной для каждого допустимого значения масти. Когда потребуется значение статической константы класса **Suit**, можно написать выражение **Suit.DIAMONDS**.

Класс **Rank**, приведенный в листинге 9, работает аналогично классу **Suit**, за исключением того, что он добавляет несколько новых методов. Метод **getRank()** возвращает значение **Rank** (числовое значение достоинства карты). Оно может потребоваться для вычисления кона. В отличие от исходных констант (**листинг 1а, с. 5,сл.**), выяснять смысл констант **Rank** и **Suit** уже не нужно. Ведь они имеют смысл сами по себе, поскольку являются объектами. Например, уже не нужно придавать смысл целочисленному (типа **int**) значению 4 и помнить, что 4 обозначает бубны (DIAMONDS) ♦.

Листинг 9. Rank.cs

```
1:          // Общий закрытый класс Rank (достоинство карты)
2:      public sealed class Rank
3:      {
4:          public static readonly Rank TWO = new Rank(2, "2");
5:          public static readonly Rank THREE = new Rank(3, "3");
6:          public static readonly Rank FOUR = new Rank(4, "4");
7:          public static readonly Rank FIVE = new Rank(5, "5");
8:          public static readonly Rank SIX = new Rank(6, "6");
9:          public static readonly Rank SEVEN = new Rank(7, "7");
10:         public static readonly Rank EIGHT = new Rank(8, "8");
11:         public static readonly Rank NINE = new Rank(9, "9");
12:         public static readonly Rank TEN = new Rank(10, "10");
```

```
13:                 public static readonly Rank JACK = new Rank(11, "J");          // валет
14:                 public static readonly Rank QUEEN = new Rank(12, "Q");         // дама
15:                 public static readonly Rank KING = new Rank(13, "K");          // король
16:                 public static readonly Rank ACE = new Rank(14, "A");           // туз
17:                         // Общий статический Rank (достоинство карты)
18: public static readonly Rank[ ] RANK = new Rank[ ]{TWO, THREE, FOUR, FIVE, SIX, SEVEN,
    EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE};
19:
20:                 private readonly int rank;
21:             // Переменная экземпляра rank для хранения отображаемого значения
22:                 private System.String display;
23:
24:                 private Rank(int rank, System.String display)          // конструктор
25:                 {
26:                         this.rank = rank;
27:                         this.display = display;
28:                 }
29:
30:                 public int getRank()                                   // возврат достоинства карты
31:                 {
32:                         return rank;
33:                 }
34:                                         // ToString() – переопределяющий м-д, л-г 1с, с. 28, стр. 40
35:                 public override System.String ToString()
36:                 {
37:                         return display;
38:                 }
39:         }
```

В листинге 1с показано, как нужно изменить класс **Card**, чтобы использовать в нем новые константы (**сравнить с листингом 1а, с.5,сл.**).

Листинг 1с. Обновленный класс Card

```
                /*Это техническое представление карты в виде класса Card.
            Карты отличаются только значением масти и достоинством, а также положением
                Это техническое представление карты в виде класса Card*/
1:  public class Card                                                          // Листинг 1с
2:  {       private Rank rank;
3:          private Suit suit;
4:          private bool face_up;
5:
6:              /* создание новой карты - при этом используются только те константы,
                                        которые были инициализированы */
7:          public Card(Suit suit, Rank rank)                  // конструктор – создается карта
8:          {// В реальной программе следовало бы сделать проверку правильности аргументов
9:                  this.suit = suit;
10:                 this.rank = rank;
11:         }
12:
13:         virtual public Suit Suit                    // свойство – получить значение масти карты
14:         {
15:                 get     {       return suit;    }
16:         }
17:
```

```
18:         virtual public Rank Rank            // свойство – получить значение достоинства карты
19:         {
20:                 get     {       return rank;   }
21:         }
22:
23:         virtual public bool FaceUp// свойство – получить значение признака положения карты
24:         {
25:                 get     {       return face_up;
26:         }
27:
28:         public virtual void  faceUp()                   // метод - Положение карты  лицом вверх
29:         {
30:                 face_up = true;
31:         }
32:
33:         public virtual void  faceDown()                 // метод - Положение карты  лицом вниз
34:         {
35:                 face_up = false;
36:         }
37:
38:         public virtual System.String display()          // строковое представление карты
39:         {         // л-г 9, с. 27, стр. 35 +  л-г 8, с. 26, стр. 23
40:                 return rank.ToString() + suit.ToString();
41:         }
42: }
```

Возможно, вы уже заметили, что и в **Rank**, и в **Suit** объявлены массивы констант. Это облегчает доступ к значениям констант внутри цикла. Объявление массивов констант помогает значительно упростить метод **buildCards**() класса **Deck** (см. в листингом 2а, с.7, строки 48…108 и в листинге 2с, с.29, строки 52…64).

В **листинге 2с** приведен обновленный классы **Deck (сравнить с листингом 2а, с. 7,сл.)**.

Листинг 2с. Обновленный класс **Deck**

```
        Класс Deck создает колоду карт из 52-х карт (1), а также (2) содержит методы:
          2.1. для перетасовки колоды, 2.2. для извлечения и 2.3. для возврата карт
1:  public class Deck                                                       // Листинг 2с
2:  {
3:          private System.Collections.ArrayList deck;          // динамический массив
4:
5:          public Deck()
6:          {       // обращение к методу - создание колоды карт – л-г 2с, с. 29, стр. 52…64
7:                  buildCards();
8:          }
9:
10: /*      public virtual Iterator iterator()                  // этот метод не используется
11:         {
12:                 return new ForwardIterator(this);
13:         } */
14:
15:         public virtual Card get_Renamed(int index)
16:         {
17:                 if (index < deck.Count)          // Count – это свойство класса ArrayList
18:                 {                       // deck[index] – это элемент массива
19:                         return (Card) deck[index];
```

```
20:                 }
21:                 return null;
22:         }
23:
24:         public virtual void  replace(int index, Card card)              // перестановка карт
25:         {
26:                 deck[index] = card;
27:         }
28:
29:         public virtual int size()
30:         {
31:                 return deck.Count;          // Count – это свойство класса ArrayList
32:         }
33:
34:         public virtual Card removeFromFront()                  // метод – извлечение карты
35:         {
36:                 if (deck.Count > 0)         // Count – это свойство класса ArrayList
37:                 {
38:                         System.Object tempObject;
39:                         tempObject = deck[0];
40:                         deck.RemoveAt(0);           // RemoveAt – это м-д класса ArrayList
41:                         Card card = (Card) tempObject;
42:                         return card;
43:                 }
44:                 return null;
45:         }
46:
47:         public virtual void  returnToBack(Card card)   // метод – возврат карты
48:         {
49:                 deck.Add(card);                    // Add – это м-д класса ArrayList
50:         }
51:
52:         private void  buildCards()          // Сравнить с листингом 2а, с.8 ,сл. (строки 48…108)
53:         {                       // создание колоды карт
54:                                 // объект deck класса ArrayList()- динамический массив
55:                 deck = new System.Collections.ArrayList();
56:
57:                 for (int i = 0; i < Suit.SUIT.Length; i++)          // Масть: л-г 8, с. 25, стр. 11
58:                 {
59:                         for (int j = 0; j < Rank.RANK.Length; j++)  // Ранг: л-г 9, с. 27, стр. 18
60:                         {     // Add – это м-д класса ArrayList
61:                 deck.Add(new Card(Suit.SUIT[i], Rank.RANK[j])); // л-г 1с, с. 27, стр. 7…11
62:                         }
63:                 }
64:         }
65: }
```

В листинге 3а приведен класс Dealer (с. 9,сл.).

Листинг 3а. Класс Dealer

```
                Это класс раздающего карты.
        Dealer умеет тасовать (shuffle) (1) и раздавать (deal) (2) карты
1:  public class Dealer                                                    //Листинг 3а
2:          {
3:          private Deck deck;                      // прямой доступ к колоде закрыт
```

```
4:
5:          public Dealer(Deck d)
6:          {
7:                  deck = d;
8:          }
9:
10:         public virtual void  shuffle()          // этот метод перетасовывает карты
11:         {
12:                 // расположить карты случайным образом
13:                 int num_cards = deck.size();                    // л-г 2c, с. 29, стр. 29…32
14:                 for (int i = 0; i < num_cards; i++)
15:                 {                                       // NextDouble() – м-д класса Random
16:                         int index = (int) (SupportClass.Random.NextDouble() * num_cards);
17:                         Card card_i = (Card) deck.get_Renamed(i); // л-г 2c, с. 28, стр. 15…22
18:                         Card card_index = (Card) deck.get_Renamed(index);
19:                         deck.replace(i, card_index);                    // л-г 2c, с. 29, стр. 24…27
20:                         deck.replace(index, card_i);
21:                 }
22:         }
23:
24:         public virtual Card dealCard()              // метод раздачи карт не используется
25:         {
26:                 if (deck.size() > 0)
27:                 {
28:                         return deck.removeFromFront();
29:                 }
30:                 return null;
31:         }
32:
33: public class SupportClass
34: {
35:                 // обеспечивается доступ к статической переменной Random
36:                 static public System.Random Random = new System.Random();
37: }
38: }
```

В Листинге 4а (с. 10) **приведен начальный класс CardDriver, который распечатывает колоду, тасует ее, а затем распечатывает ее снова** (с. 10).

Листинг 4а. Класс CardDriver

```
Это начальный класс CardDriver распечатывает колоду,
тасует ее, а затем распечатывает ее снова
1:  public class CardDriver
2:  {
3:          public static void  Main( String[ ] args)
4:          {
5:          Deck   deck   =  new Deck(); // объект  deck - колода карт – л-г 2c, с. 28, стр. 5…8
6:  Dealer dealer = new Dealer(deck);  // колода карт передается дилеру – л-г 3а, с. 30, стр. 5…8
7:     Console.WriteLine("\nРаспечатка упорядоченной колоды карт");
8:                  printDeck(deck); // распечатать упорядоченную колоду карт – стр. 18…30
9:
10:                 dealer.shuffle();          // перетасовать колоду карт – л-г 3а, с. 30, стр. 10…22
11:
12:    Console.WriteLine("\nРаспечатка перетасованной колоды карт");
13:                 printDeck(deck); // распечатать перетасованную  колоды карт – стр. 18…30
```

```
14:
15:                 Console.ReadLine();
16:         }
17:
18:         public static void  printDeck(Deck deck) // метод – печать массива карт
19:         {
20:                 for (int i = 0; i < 4; i++) // четыре масти
21:                 {
22:                         for (int j = 0; j < 13; j++) // карты 13-ти достоинств
23:                         {
24:                                 Card card = deck.removeFromFront();  //л-г 2c, с. 29, стр. 34…45
25:                                 deck.returnToBack(card);                  //л-г 2c, с. 29, стр. 47…50
26:                                  Console.Write(card.display() + " ");      // л-г 1c, с. 28, стр. 38…41
27:                         }
28:                          Console.WriteLine(" ");
29:                 }
30:         }
31: }
```

Метод **Main()** выполняет следующее: **1)** создает **экземпляр** класса **dealer** (**раздающий карты**), **2)** создает **экземпляр** класса **deck** of cards (**колода карт**), **3) перетасовывает** карты, **4) распечатывает** полученную колоду карт.

В коде, рассмотренном в **Разд. 3**, интерфейс **Iterface (л-г 5, с. 15)** не используется.

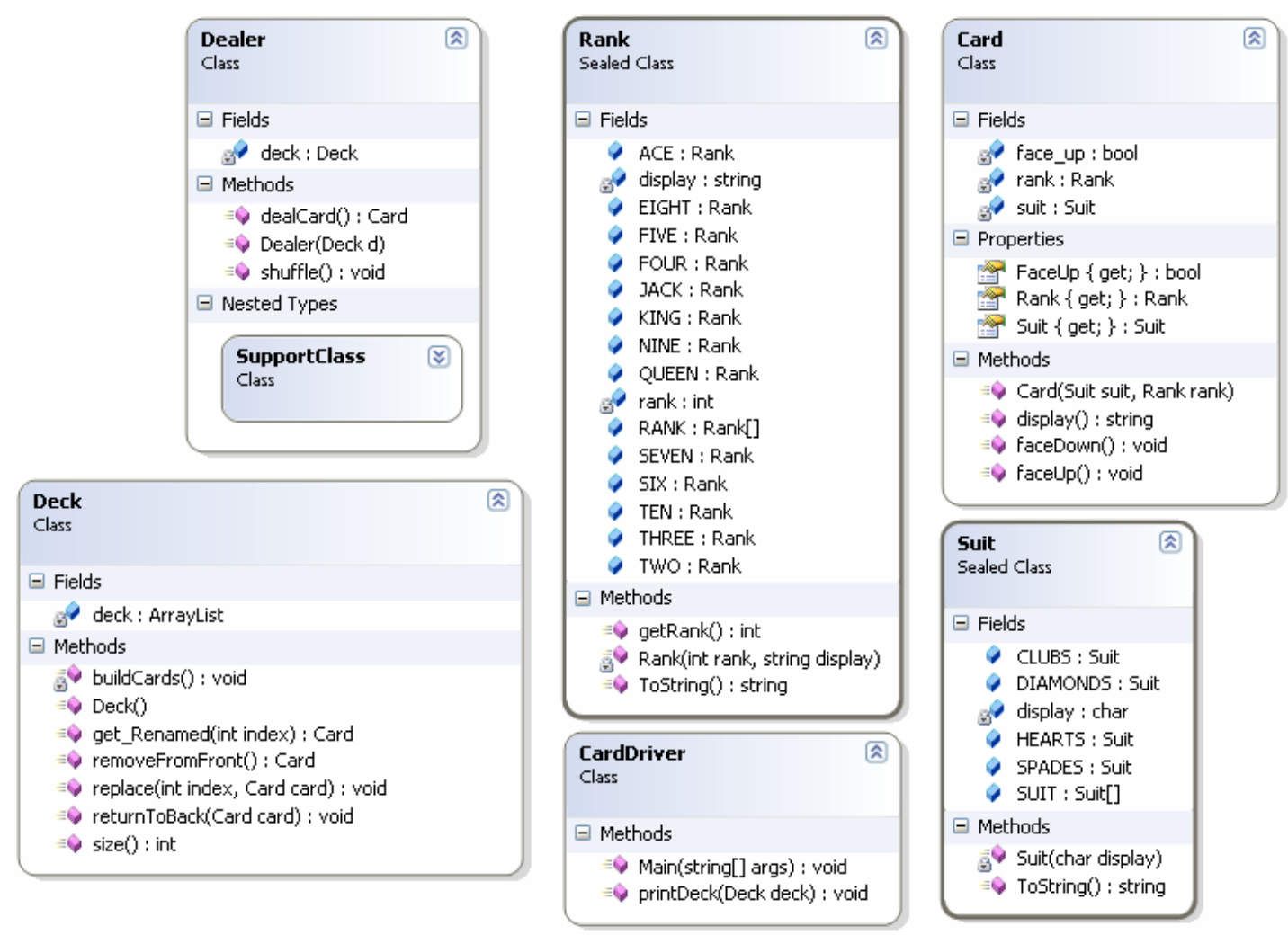
3.2.2. Результаты работа программы

Распечатка упорядоченной колоды карт

2♣ 3♣ 4♣ 5♣ 6♣ 7♣ 8♣ 9♣ 10♣ J♣ Q♣ K♣ A♣
2♠ 3♠ 4♠ 5♠ 6♠ 7♠ 8♠ 9♠ 10♠ J♠ Q♠ K♠ A♠
2♥ 3♥ 4♥ 5♥ 6♥ 7♥ 8♥ 9♥ 10♥ J♥ Q♥ K♥ A♥
2♦ 3♦ 4♦ 5♦ 6♦ 7♦ 8♦ 9♦ 10♦ J♦ Q♦ K♦ A♦

Распечатка перетасованной колоды карт

4♥ 2♦ 4♠ 7♥ 5♠ 3♣ 4♦ K♣ 8♦ J♦ 10♠ 7♠ 2♥
2♠ 9♦ 7♦ K♥ A♠ K♠ Q♥ 10♦ J♥ J♠ 8♣ 2♣ 3♥
6♣ 9♠ A♥ 6♠ 9♥ Q♦ A♦ 4♣ 6♥ 3♠ J♣ 6♦ 5♥
Q♣ 10♣ Q♠ 10♥ 8♠ 9♣ 5♣ 8♥ A♣ 7♣ 5♦ K♦ 3♦



3.2.3. Диаграмма классов 3

3.2.4. Когда используется шаблон перечисления Typesafe Enum

Используйте шаблон перечисления **Typesafe Enum** при:

- написании многочисленных общедоступных исходных постоянных или строковых констант (констант типа string);
- попытке идентифицировать значение (т.е. придать значению смысл) вместо того, чтобы значение идентифицировало само себя.

В табл. 2 содержится описание шаблона перечисления Typesafe Enum.

Таблица 2. Шаблон перечисления Typesafe Enum

Название шаблона	Typesafe Enum (перечисление)
Задача	Использование целых констант нежелательно
Решение	Создает класс для каждого типа констант и предоставляет экземпляры для каждого значения константы
Результаты	Расширяемый набор констант, удовлетворяющий объектно-ориентированному подходу. Полезные константы, имеющие поведение. По мере добавления новых констант для их использования код все равно придется обновить. Перечисления требуют больше памяти, чем простые константы

Контрольные вопросы

1. Для решения какой задачи предназначен шаблон перечисления Typesafe Enum?
2. Почему следует применять шаблон перечисления Typesafe Enum?

Ответы на вопросы

1. Использование исходных постоянных идет вразрез с принципами объектно-ориентированного подхода, поскольку смысл постоянной не определяется самой постоянной, чтобы определить его, приходится прибегать к внешним средствам. А вы уже убедились, сколько проблем может создать нарушение распределения обязанностей!

   Шаблон перечисления Typesafe Enum решает эту проблему путем превращения постоянной в объект более высокого уровня. Использование объекта более высокого уровня позволяет инкапсулировать обязанности константы внутри постоянного объекта.

2. Всякий раз, при объявлении общедоступных констант, которые сами по себе должны быть объектами, следует использовать шаблон перечисления Typesafe Enum.

Литература

Базовый учебник

1. Мейер Б. Объектно-ориентированное конструирование программных систем. М.: Русская Редакция, 2005.

Основная

2. Буч Г., Якобсон А., Рамбо Дж. **UML**. С.-Петербург: Питер, 2006.
3. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования. С.-Петербург: Питер, 2006.
4. Забудский Е.И. Учебно-методический комплекс дисциплины «Объектно-ориентированный анализ и программирование». М.: Кафедра ОИиППО ГУ-ВШЭ, 2007,
   Internet-ресурс – http://new.hse.ru/C7/C17/zabudskiy-e-i/default.aspx .
5. Кватрани Т. Визуальное моделирование с помощью Rational Rose 2002 и UML. М.: Вильямс, 2003.
6. Лафоре Р. Объектно-ориентированное программирование в C++. С.-Петербург: Питер, 2005.
7. Троелсен Э. C# и платформа .NET. С.-Петербург: Питер, 2006.
8. Синтес А. Освой самостоятельно объектно-ориентированное программирование за 21 день. Москва; С.-Петербург; Киев: Вильямс, 2002.

Дополнительная – Internet-ресурсы

9. Новые книги раздела **C#** – http://books.dore.ru/bs/f6sid16.html .
10. **C#** и ***.NET*** по шагам – http://www.firststeps.ru .
11. ***UML*** – язык графического моделирования – http://www.uml.org/ .
12. ***NUnit, JUnit*** – каркасы тестирования для испытания классов – http://www.junit.org , http://www.nunit.org .
13. Пакет объектного моделирования ***Rational Rose*** – http://www-306.ibm.com/software/rational/ .

13*a*. Steve Burbeck "Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)" – http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html .

13*b*. Информация о языке C# и платформе .NET – http://msdn2.microsoft.com/ru-ru/default.aspx .

13*c*. Информация о языке C# и платформе .NET – http://www.gotdotnet.com http://www.gotdotnet.ru

Дополнительная – книги

14. Мэтт Вайсфельд. Объектно-ориентированный подход: Java, .NET, C++. М.: КУДИЦ-ОБРАЗ, 2005.
15. Дж. Кьоу, М. Джеанини. Объектно-ориентированное программирование. С.-Петербург: Питер, 2005.
16. Уоткинз Д., Хаммонд М, Эйбрамз Б. Программирование на платформе .NET.: М.: Вильямс, 2003.