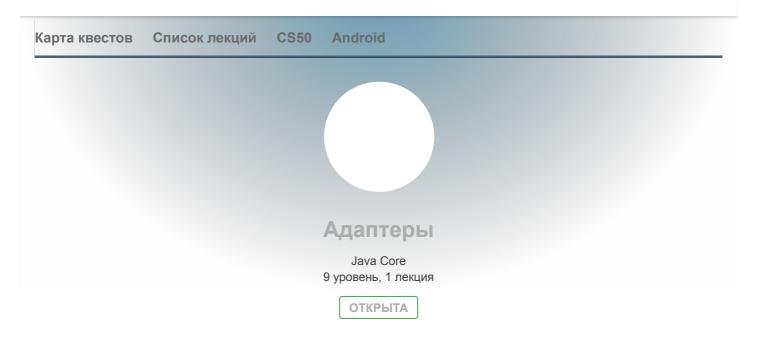
Лекции





— Привет, Амиго! Сегодня я расскажу тебе, что же такое «**адаптер**». Надеюсь, что после его изучения ты начнешь понимать потоки ввода-вывода гораздо лучше.



Представь, что в твоей программе ты используешь два фреймворка, написанные другими программистами/компаниями. Оба фреймворка очень хорошие и используют принципы ООП: абстракцию, полиморфизм, инкапсуляцию. Они вместе практически полностью покрывают задачи твоей программы. За тобой осталось простая задача — объекты, которые создает один фреймворк нужно передать во второй. Но оба фреймворка совершенно разные и «не знают друг о друге» — т.е. не имеют общих классов. Тебе нужно как-то преобразовывать объекты одного фреймворка в объекты другого.

Эту задачу можно красиво решить, применив подход (паттерн проектирования) «адаптер»:

Код на Java Описание

```
class MyClass implements Interface2
2
    {
3
     private Interface1 object;
    MyClass(Interface1 object)
4
5
6
     this.object = object;
7
8
     // тут располагаются методы Interface2,
9
     // которые вызывают методы Interface1
10
```

Это схематическое описание «паттерна проектирования адаптер».

Суть его в том, что класс MyClass является преобразователем (адаптером) одного интерфейса к другому.

— А можно более конкретный пример?

— Ок. Допустим, что у каждого фреймворка есть свой уникальный интерфейс «список», вот как это может выглядеть:

Код на Java

```
1 interface AlphaList
2 {
3  void add(int value);
4  void insert(int index, int value);
5  int get(int index);
6  void set(int index, int value);
7  int count();
8  void remove(int index);
9 }
```

Описание

Код из первого(**Alpha**) фреймворка.

AlphaList – это один из интерфейсов, для взаимодействия кода фреймворка и кода, который будет использовать этот фреймворк.

AlphaListManager –

класс фреймворка,

метод которого

объект типа

AlphaList

createList создает

```
1 class AlphaListManager
2 {
3 public static AlphaList createList()
4 {
5 //какой-то код по созданию объекта
6 }
7 }
```

```
Код из второго(Beta) фреймворка.
```

```
1 interface BetaList
2 {
3  int getValue(int index);
4  void setValue(int index, int value);
5  int getSize();
```

```
6
     void setSize(int newSize);
7
    }
8
    class BetaSaveManager
9
10
     public static void saveList(BetaList list)
11
12
      //какой-то код по сохранению объекта
13
      //типа BetaList в файл на диске
14
15
    }
```

BetaList — это один из интерфейсов, для взаимодействия кода фреймворка и кода, который будет использовать этот фреймворк.

BetaSaveManager -

класс фреймворка, метод которого saveList сохраняет на диск объект типа

```
BetaList
```

```
1
    class ListAdapter implements BetaList
2
3
     private AlphaList list;
     ListAdapter(AlphaList list)
4
5
6
     this.list = list;
7
8
9
     int getValue(int index)
10
      return this.list.get(index);
11
12
13
14
     void setValue(int index, int value)
15
      this.list.set(index, value);
16
17
18
19
     int getSize()
20
21
     return this.list.count();
22
23
24
     void setSize(int newSize)
25
26
      if (newSize > this.list.count()
27
       while (this.list.count() < newSize)</pre>
28
29
30
       this.list.add(null);
31
      }
32
     }
33
     else if (newSize < this.list.count() {</pre>
```

Класс «адаптер» (т.е. переходник) от интерфейса

AlphaList к интерфейсу **BetaList**

Класс ListAdapter реализует интерфейс BetaList из второго фреймворка.

Когда кто-то
вызывает эти
методы, код класса
перевызывает
методы переменной
list, которая имеет
тип AlphaList из
первого фреймворка.

Объект типа
AlphaList передается
в конструктор
ListAdapter в момент
создания

```
34  while (this.list.count() > newSize)
35      {
36          list.remove(list.count() - 1);
37      }
38      }
39      }
40  }
```

Метод setSize
работает по
принципу: если
нужно увеличить
размер списка —
добавим туда пустых
(null) элементов.
Если нужно
уменьшить — удалим
несколько
последних.

— Больше всего понравился пример использования. Очень компактно и понятно.

< Предыдущая

Следующая >

Программистами не рождаются © 2018