**Адаптеры**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 9](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=9), Лекция 1

— 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](https://javarush.ru/groups/posts/principy-oop): 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  {  private Interface1 object;  MyClass(Interface1 object)  {  this.object = object;  }  // тут располагаются методы Interface2,  // которые вызывают методы Interface1  } | 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** | **Описание** |
| interface **AlphaList**  {  void add(int value);  void insert(int index, int value);  int get(int index);  void set(int index, int value);  int count();  void remove(int index);  } | Код из первого(**Alpha**) фреймворка.  **AlphaList** – это один из интерфейсов, для взаимодействия кода фреймворка и кода, который будет использовать этот фреймворк. |
| class **AlphaListManager**  {  public static **AlphaList createList**()  {  //какой-то код по созданию объекта  }  } | **AlphaListManager** – класс фреймворка, метод которого createList создает объект типа **AlphaList** |
| interface BetaList  {  int getValue(int index);  void setValue(int index, int value);  int getSize();  void setSize(int newSize);  }  class BetaSaveManager  {  public static void saveList(BetaList list)  {  //какой-то код по сохранению объекта  //типа BetaList в файл на диске  }  } | Код из второго(**Beta**) фреймворка.  **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** |
| class ListAdapter implements BetaList  {  private **AlphaList** list;  ListAdapter(AlphaList list)  {  this.list = list;  }  int getValue(int index)  {  return this.list.get(index);  }  void setValue(int index, int value)  {  this.list.set(index, value);  }  int getSize()  {  return this.list.count();  }  void setSize(int newSize)  {  if (newSize > this.list.count()  {  while (this.list.count() < newSize)  {  this.list.add(null);  }  }  else if (newSize < this.list.count() {  while (this.list.count() > newSize)  {  list.remove(list.count() - 1);  }  }  }  } | 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  Метод setSize работает по принципу: если нужно увеличить размер списка – добавим туда пустых (null) элементов. Если нужно уменьшить – удалим несколько последних. |
| public static void main(String[] args)  {  AlphaList listAlpha = AlphaListManager.createList();  BetaList listBeta = new ListAdapter(listAlpha);  BetaSaveManager.saveList(listBeta);  } | Пример использования |

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