Система управления списком контактов

Чтобы мы могли дальше двигаться не просто рассматривая новое, а со смыслом и практикой, предлагаю начать разработку очень простого приложения, которое позволит нам задействовать описываемые технологии. Оно будет даже проще варианта «Студенческий отдел кадров», который я планирую переделать на новый лад.
Чтобы не придумывать что-то очень сложное, я буду делать простой список контактов, который будет иметь возможность делать основные операции (назовем их для дальнейших рассуждений «бизнес-действие»):

1. Просмотр списка контактов
2. Добавление контакта
3. Редактирование контакта
4. Удаление контакта

Также предлагаю делать свой вариант — например список машин в автосалоне, список заказов пиццы, список книг. Пусть он будет выглядеть практически так же, как мой, но это будет ваш собственный проект.

В этой статье мы создадим каркас нашего приложения — основные классы и методы. При изучении нового материала будем расширять эту функциональность.

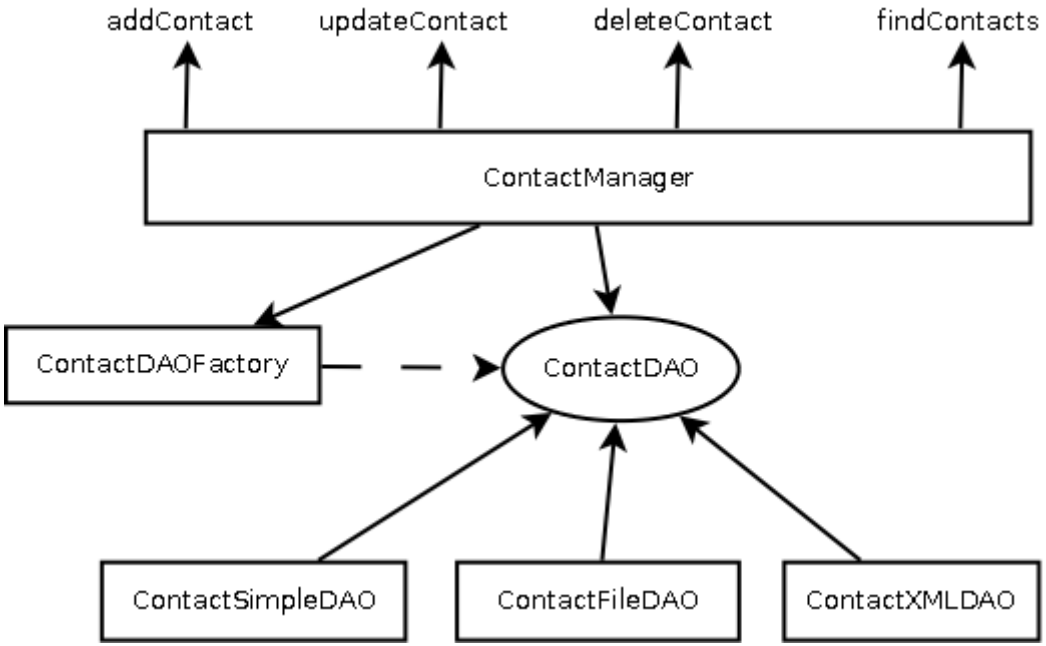
Для начала проектирования воспользуемся нашим набором функций и существительным «контакт». Эти слова на самом деле очень помогают при начальной разработке — вы по сути определяете предметную область и можете выделить какие-то базовые классы и их методы.
Естественно, что при наработке дальнейшего опыта и усложнении систем, которые вы будете разрабатывать, набор классов и их функциональность будет увеличиваться, но основные принципы останутся такие же, как и в нашем случае — смотрим описание и выделяем существительные и глаголы. В них основная идея и будет скрываться :).

Достаточно легко можно увидеть класс **Contact**, который будет использоваться как хранилище информации об одном контакте. Также нам потребуется класс, который предоставить нам нужные фукнции для работы со списокм контактов — удаление, добавление, редактирование, получение списка. Назовем его **ContactManager**. Также нам потребуется класс (классы) для хранения контактов в каком-то постоянном хранилище. Т.к. систем хранения у нас будет несколько — для фалов, для базы данных.
В нашем случае это не очень выразительно видно, но я очень хочу, чтобы вы уловили разницу между хранением данных в хранилище и действиями по редактированию. Этот момент важен по следующим соображениям:

1. Функции редактирования могут быть гораздо сложнее обычного сохранения данных в хранилище. Например, добавление контакта может потребовать проверку данных на правильность полей, проверку на наличие такого контакта (причем такая проверка может быть весьма нетривиальной задачей — по имени или по e-mail или еще как). Также надо принимать во внимание возможность дальнейшего расширения системы. Добавление контакта может потребовать выполнить какие-то дополнительные действия — поиск сведений о контакте в социальных сетях, поиск через Гугл или синхронизацию данных с удаленным хранилищем. В итоге мы однозначно можем сказать, что бизнес-действие нередко включает несколько элементарных действия, среди которых будет и действия с хранилищем.
2. Хранилища могут быть совершенно разными. Это может быть файл — причем в разных форматах: CSV, XML, какой-то текст, бинарное представлние (MS Word, MS Excel). Это может быть база данных — причем совершенно разных типов — например SQL и NoSQL. Это может быть та же 1С-Бухгалтерия или сервер MS Exchange. Или IBM Lotus Notes. В общем систем хранения может быть много и надо быть готовым к тому, чтобы перевести ваше приложение на новый вариант с текущего

Суммируя все вышеизложенное мы будем строить взаимодействие класса **ContactManager** с хранилищами через интерфейс , реализовать который должны все хранилища, которые мы будем создавать.

Давайте нарисуем упрощенную диаграмму наших классов:



Как видите, наш класс **ContactManager** взаимодействует с хранилищем не напрямую, а через интерфейс **ContactDAO**. Т.к. все хранилища его реализуют, то нам будет легко менять их фактически «на лету».

Коллекции

Мы еще не рассматривали классы-колекции — в нашем случае это **List** и **ArrayList**. Мы займемся этим сразу после этой главы, а пока примите это как данность. Обещаю, что мы очень скоро расставим все по своим местам.

Класс Contact

Этот класс не должен вызывать каких-то больших вопросов. Там перечислены нужные поля и для них сделаны сетеры и гетеры. Может вызывать сомнение набор конструкторов — я сделал для всех полей (в случае редактирования) и все поля кроме ИД — для случая добавления. Мне показалось, что так будет удобно. Переопределение метода **toString** сделано для удобства вывода информации о контакте.

```
1 package edu.javacourse.contact.entity;
2
3 /**
4  *   Класс для хранения данных контакта
5  */
6 public class Contact
```

```
{
    // Идентификатор контакта
    private Long contactId;
    // Имя
    private String firstName;
    // Фамилия
    private String lastName;
    // Телефон
    private String phone;
    // email
    private String email;

    public Contact() {
    }

    public Contact(String firstName, String lastName, String phone, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.phone = phone;
        this.email = email;
    }

    public Contact(Long contactId, String firstName, String lastName, String phone, String email) {
        this.contactId = contactId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.phone = phone;
        this.email = email;
    }

    public Long getContactId() {
        return contactId;
    }

    public void setContactId(Long contactId) {
        this.contactId = contactId;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
```

```
54          return lastName;
55      }
56
57      public void setLastName(String lastName) {
58          this.lastName = lastName;
59      }
60
61      public String getPhone() {
62          return phone;
63      }
64
65      public void setPhone(String phone) {
66          this.phone = phone;
67      }
68
69      public String getEmail() {
70          return email;
71      }
72
73      public void setEmail(String email) {
74          this.email = email;
75      }
76
77      @Override
78      public String toString() {
79          return "Contact{" + "contactId=" + contactId + ", firstName=" + firstName + ", lastName=" + lastName + ", phone
80      }
81  }
```

Класс ContactManager

Здесь тоже особых сложностей нет — в данном случае я просто перечислил те фукнции, которые мы хотели реализовать. Выглядит это конечно несколько натянуто, т.к. кроме вызовов методов для работы с хранилищем в них ничего нет, но мы уже осбуждали возможность расширения. Единственное, на что надо обратить внимание — мы используем интерфейс **ContactDAO**, а не реальный класс. Почему мы так сделали — я тоже объяснял выше.

```
1  package edu.javacourse.contact.business;
2
3  import edu.javacourse.contact.dao.ContactDAO;
4  import edu.javacourse.contact.dao.ContactDAOFactory;
5  import edu.javacourse.contact.entity.Contact;
6  import java.util.List;
```

```java

/**
 * Класс для реализации функций над списком контактов
 */
public class ContactManager
{
    private ContactDAO dao;

    public ContactManager() {
        dao = ContactDAOFactory.getContactDAO();
    }

    // Добавление контакта - возвращает ID добавленного контакта
    public Long addContact(Contact contact) {
        return dao.addContact(contact);
    }

    // Редактирование контакта
    public void updateContact(Contact contact) {
        dao.updateContact(contact);
    }

    // Удаление контакта по его ID
    public void deleteContact(Long contactId) {
        dao.deleteContact(contactId);
    }

    // Получение одного контакта
    public Contact getContact(Long contactId) {
        return dao.getContact(contactId);
    }

    // Получение списка контактов
    public List<Contact> findContacts() {
        return dao.findContacts();
    }
}
```

Почему мы создаем экземпляр **ContactDAO** с использованием класса **ContactDAOFActory** я объясню чуть позже.

Интерфейс ContactDAO

Прибавочка DAO — это Data Access Object — объект доступа к данным. Для работы с хранилищем любого типа мы создали интерфейс, который определяет контракт, по которому любой хранилище для работы с контактами должно реализовать определенный набор функций. Это позволит нам в дальнейшем заменять реализации хранилища без особых хлопот.

```java
package edu.javacourse.contact.dao;

import edu.javacourse.contact.entity.Contact;
import java.util.List;

/**
 * Интерфейс для определения функций хранлиза данных о контактах
 */
public interface ContactDAO
{
    // Добавление контакта - возвращает ID добавленного контакта
    public Long addContact(Contact contact);
    // Редактирование контакта
    public void updateContact(Contact contact);
    // Удаление контакта по его ID
    public void deleteContact(Long contactId);
    // Получение контакта
    public Contact getContact(Long contactId);
    // Получение списка контактов
    public List<Contact> findContacts();

}
```

Класс ContactDAOFactory

Идея появления этого класса лежит в плоскости шаблонов проектирования. В данном случае это шаблон **Abstract Factory**. Поробую ответить на вопрос — а зачем он нужен ?
Если простыми словами — классы для хранилища могут использоваться во многих местах. Это у нас все достаточо просто — мы к нему по сути обращаемся только водном месте — в классе **ContactManager**. Но представим ситуацию, когда хранилище контактов потребуется не в одном классе, а например в 25-ти. Тогда если мы захотим поменять наше хранилище на другое, то создание надо будет переопределять в 25-ти местах. Можно запутаться. Класс **ContactDAOFactory** легко решает эту задачу. Создание хранилища происходит исключительно в нем и больше нигде. Это удобно.

```java
package edu.javacourse.contact.dao;

/**
 * Фабрика для создания экземпляра ContactDAO
 */
public class ContactDAOFactory
{
    public static ContactDAO getContactDAO() {
        return new ContactSimpleDAO();
```

```
    }
}
```

Классы ContactSimpleDAO и ContactTest

Классы имеют исключительно утилитарное назначние — мы можем их использовать для проверки системы. **ContactSimpleDAO** реализует интерфейс **ContactDAO** и это дает нам возможность «смоделировать» хранилище. Что же касается класса **ContactTest** — он используется для вызова методов класса **ContactManager** и вывода результатов. Мы можем убедиться, что наша система (пусть пока и очень сырая) работает.

```
package edu.javacourse.contact.dao;

import edu.javacourse.contact.entity.Contact;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ContactSimpleDAO implements ContactDAO
{
    private final List<Contact> contacts = new ArrayList<Contact>();

    @Override
    public Long addContact(Contact contact) {
        Long id = generateContactId();
        contact.setContactId(id);
        contacts.add(contact);
        return id;
    }

    @Override
    public void updateContact(Contact contact) {
        Contact oldContact = getContact(contact.getContactId());
        if(oldContact != null) {
            oldContact.setFirstName(contact.getFirstName());
            oldContact.setLastName(contact.getLastName());
            oldContact.setPhone(contact.getPhone());
            oldContact.setEmail(contact.getEmail());
        }
    }

    @Override
    public void deleteContact(Long contactId) {
        for(Iterator<Contact> it = contacts.iterator(); it.hasNext();) {
            Contact cnt = it.next();
```

```
            if(cnt.getContactId().equals(contactId)) {
                it.remove();
            }
        }
    }

    @Override
    public Contact getContact(Long contactId) {
        for(Contact contact : contacts) {
            if(contact.getContactId().equals(contactId)) {
                return contact;
            }
        }
        return null;
    }

    @Override
    public List<Contact> findContacts() {
        return contacts;
    }

    private Long generateContactId() {
        Long contactId = Math.round(Math.random() * 1000 + System.currentTimeMillis());
        while(getContact(contactId) != null) {
            contactId = Math.round(Math.random() * 1000 + System.currentTimeMillis());
        }
        return contactId;
    }
}
```

```
package edu.javacourse.contact.test;

import edu.javacourse.contact.business.ContactManager;
import edu.javacourse.contact.entity.Contact;
import java.util.List;

/**
 * Класс для запуска тестовых вызовов
 */
public class ContactTest
{
    public static void main(String[] args) {
        ContactManager cm = new ContactManager();

```

```
15          Contact c1 = new Contact("Андрей", "Соколов", "+7-911-890-7766", "sokolov@yandex.ru");
16          Contact c2 = new Contact("Сергей", "Иванов", "+7-911-890-7755", "ivanov@google.com");
17          Contact c3 = new Contact("Татьяна", "Семенова", "+7-911-890-1164", "semenova@mail.ru");
18
19          System.out.println("ADD CONTACT ==============");
20          Long cId1 = cm.addContact(c1);
21          Long cId2 = cm.addContact(c2);
22          Long cId3 = cm.addContact(c3);
23          List<Contact> result1 = cm.findContacts();
24          for(Contact c : result1) {
25              System.out.println(c);
26          }
27
28          System.out.println("UPDATE CONTACT ==============");
29          Contact change = new Contact(cId1, "Алексей", "Соколов", "+7-911-890-7766", "sokolov@yandex.ru");
30          cm.updateContact(change);
31          List<Contact> result2 = cm.findContacts();
32          for(Contact c : result2) {
33              System.out.println(c);
34          }
35
36          System.out.println("DELETE CONTACT ==============");
37          cm.deleteContact(cId1);
38          List<Contact> result3 = cm.findContacts();
39          for(Contact c : result3) {
40              System.out.println(c);
41          }
42
43          System.out.println("GET CONTACT ==============");
44          Contact contact = cm.getContact(cId2);
45          System.out.println(contact);
46      }
47  }
```

Проект на NetBeans вы можете скачать по этой ссылке: ContactProject_01.zip

И теперь нас ждет следующая статья: Коллекции — базовые принципы.

21 comments to *Список контактов — начало*

- 

Декабрь 28, 2015 at 16:59
Евгений says:

Класс ContactManager тоже реализует паттерн Проектирования, если добавить проверку на доступность совершения операций вставки/удаления/изменения, то получится Заместитель, а так получается Фасад.

Reply

- 
Май 17, 2016 at 10:26
zdeniz says:

в методе generateContactId() не может так получиться, что сгенерится ID, который уже был?

Reply

  - 
Май 17, 2016 at 12:48
admin says:

Если используется однопотоковое приложение, то не может — см. код. В данном случае это (на мой взгляд) вполне допустимое предположение, т.к. это ДАО мы используем исключительно для демонстрационных целей.

Reply

- 
Октябрь 18, 2016 at 13:52
Антон says:

Тип данных char в методе toString всё равно же будет переведён в String? И написанное Вами ‘}’ само преобразует в «}»?

Reply

Октябрь 18, 2016 at 14:16
admin says:

Этот код генерируется автоматически IDE. Что получилось, то и получилось.

Reply

Октябрь 26, 2016 at 06:24
Alexandr says:

Небольшие опечатки:
В данном случае это шаблон **Anstract** Factory
Интерфейс для определения функций **хранлиза** данных о контактах

Reply

Ноябрь 18, 2016 at 23:12
Кирилл says:

В классе ContactTest при создании ContactManager нужно передать параметр ContactDAO
ContactDAO dao = ContactDAOFactory.getContactDAO();
ContactManager manager = new ContactManager(dao);

Reply

Ноябрь 21, 2016 at 11:26
admin says:

На данный момент инициализация DAo сделана в конструкторе. Так что не требуется. Хотя в принципе сеттер для DAO надо сделать — но я хочу это сделать позже.

Reply

- 

Декабрь 2, 2016 at 00:35
Василий Фрайман says:

Привет! Во-первых, хотел поблагодарить за отличный, пожалуй, лучший самоучитель по Java)

Я так и не понял разницу между хранением данных в хранилище и действиями по редактированию — пожалуйста, коллеги, кто разобрался — поясните=)

И, вторая просьба: так и не понял, зачем нам сразу 6(!) классов. Чисто умозрительно кажется, что хватит всего 3 классов:
1 — класса для работы с контактами (из создание, обновление данных) — Contact;
2 — класса для управления списком (внутри класса создаем массив-хранилище и обслуживающие его методы) — типа ContactSimpleDAO;
3 — управляющего класса чтобы это просто запустить — типа ContactTest.

Поясните, пожалуйста, зачем к этому списку добавлять еще и интерфейс, странную конструкцию ContactDAOFactory, которая только что и вызывает ContactSimpleDAO, и посредника ContactManager, который только и умеет, что через ContactManager оперировать методами ContactSimpleDAO.

Вот честно — не понял(((

Reply

  - 

Декабрь 2, 2016 at 17:19
admin says:

ContactManager — есть ли разница между операцией «добавить пользователя» в систему и «добавить запись в базу данных» ? В первом случае это моежт быть НЕ ТОЛЬКО добавление в базу — это может породить и другие действия.
Регистрация почтового адреса, выделение квоты на файл-сервере, регистрация пользователя в домене, отправка почты секретарю, в отдел кадров и в бухгалтерию.
Второй случай — это просто добавление в базу данных. Если мы уберем ContactManager, то потом, когда нам захочется что-то добавить, то куда мы это сможем добавить ? В какой класс ?

Точно так же и интерфейс ContactDAO — там же не только SQL база данных может использоваться для хранения данных. А вдруг появится что-то другое — логику опять менять ?

Reply

- 

  Декабрь 24, 2016 at 16:51
  Станислав says:

  Ошибка в тексте. Исправьте Abstract Factory.

  Reply

  - 

    Декабрь 26, 2016 at 10:08
    admin says:

    Я не понял замечание — где именно Abstract Factory находится ?

    Reply

    - 

      Декабрь 26, 2016 at 13:40
      Yu says:

      В тексте статьи: «В данном случае это шаблон A**n**stract Factory». Надо поменять слово на «A**b**stract».

      Reply

      - 

        Декабрь 26, 2016 at 16:27
        admin says:

Спасибо — теперь нашел и исправил.

Reply

- 
  Июнь 30, 2017 at 11:42
  Yury says:

  В тему данной статьи пытаюсь выяснить вопрос:
  Предположим имеется список строк с именами text1, text2, text3 Строки с такими именами имеют различное содержимое (содержимое неизвестно). Сам список выглядит примерно так.
  private List lstKey = new ArrayList();
  String t1 = «text1»;
  String t2 = «text2»;
  String t3 = «text3»;
  lstKey.add(t1);
  lstKey.add(t2);
  lstKey.add(t3);
  Можно-ли как-то получить содержимое строк имея только список их имен?

  Reply

  - 
    Июль 5, 2017 at 10:50
    admin says:

    Вопрос не понял — если мы говорим о ключе и значении, то это не список, это Map (ассоциативный массив). А у вас — List. Не укладывается в голове 🙂

    Reply

- 
  Январь 3, 2018 at 00:46
  javazitz says:

Если я хочу добавить еще одно хранилище, допустим ContactXMLDAO, мне нужно будет в класс фабрики добавить строчку return new ContactXMLDAO(); ?

Reply

- 

  Январь 3, 2018 at 13:08
  admin says:

  Да, именно так. Но дальше мы это решение сделаем более гибким — не бросайте чтение 🙂 Удачи.

  Reply

  - 

    Январь 4, 2018 at 00:07
    javazitz says:

    Спасибо за быстрый ответ 🙂

    Reply

- 

  Октябрь 2, 2018 at 12:15
  Дмитрий says:

  В целом все понятно кроме одного, откуда взялся объект contact класса Contact, его нигде не объявляли. Изучаю java второй месяц, так что всех тонкостей еще не знаю, проясните пожалуйста.

  Reply

  - 

    Октябрь 4, 2018 at 04:57
    admin says:

Добрый день. К какому конкретно месту в статье относится вопрос? Там много где встречается слово «contact».

Reply

Leave a reply

Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <em> <i> <q cite=""> <s> <strike> <strong> <pre class="" title="" data-url=""> <span class="" title="" data-url="">`

Имя *

E-mail *

Сайт

8 × 7 =

Add comment