Технологии функционального программирования

Студент группы
ИВТ-13МО
Швецов Илья

О задаче

В качестве задачи была выбрана задача для разработки ПО №8 из курса по изучению языка Ruby на сайте “lms.crafted.su”

Структура проекта

Написать программу, моделирующую процесс купли-продажи квартир.

Сведения о каждой доступной квартире: метраж, количество комнат, адрес (район, улица, дом), этаж, вид дома (панельный, кирпичный), количество этажей, стоимость.

Сведения о каждой заявке на покупку квартиры: количество комнат, район, вид дома.

Начальное формирование данных осуществляется из файла (или файлов).

Функционал проекта:

1. Добавить/удалить квартиру/заявку.
2. Вывести на экран все заявки заданного района, отсортированные по количеству комнат.
3. Вывести на экран все квартиры, сгруппированные по району и отсортированные по метражу.
4. Вывести на экран все квартиры по заданному диапазону стоимости.
5. Для заданной заявки вывести на экран список подходящих и «почти подходящих» квартир: с количеством комнат, отличающимся не более, чем на 1.
6. Выполнить заявку: для заявки из списка «почти подходящих» квартир выбрать одну. В результате и заявка, и выбранная квартира удаляются.
7. Вывести статистику по каждому району: количество предложений продажи, средний метраж, средняя стоимость, количество заявок на покупку, потенциальный процент покрытия заявок (сколько заявок на покупку относительно общего числа заявок в этом районе имеют хотя бы одну полностью подходящую квартиру).

Хранение данных

```
21 usages
public class RequestInfo {
    6 usages
    public int roomCount; // количество комнат
    8 usages
    public String district; // район
    8 usages
    public BuildingTypeEnum.BuildingType buildingType; // вид дома
```

Здесь приведён пример хранения данных о заказах.

Данные о квартирах хранятся в аналогичном формате

```
6 usages
public class Requests {

    6 usages
    List<RequestInfo> requests = new ArrayList<>();
```

Так же изначальный вариант приложения содержал в классе Main глобальные переменных

```
1 usage
public class Main {

    9 usages
    Apartments apartments = new Apartments();
    6 usages
    Requests requests = new Requests();
    31 usages
    int stage = 0;
    28 usages
    ApartmentInfo newOrSelectedApartment = null;
    18 usages
    RequestInfo newOrSelectedRequest = null;
    6 usages
    String districtToSearch = "";
    9 usages
    double startRange = -1, endRange = -1;
```

Вычисление/изменение данных происходит в публичных функциях классов: Apartments и Request, вызываемых из класса Main

```
2 usages
public boolean add(RequestInfo a) { return requests.add(a); }

2 usages
public boolean remove(RequestInfo a) { return requests.remove(a); }

no usages
public RequestInfo get(int index) { return requests.get(index); }

1 usage
public List<RequestInfo> search(String districtToSearch) {
    List<RequestInfo> list = new ArrayList<>();
    for (RequestInfo info : requests) {
        if (info.district.equals(districtToSearch)) {
            list.add(info);
        }
    }
    return list;
}
```

Переходим к применению техник функционального программирования

В каждом классе было уменьшено количество обращений к глобальным переменным. За счёт этого удалось увеличить количество так называемых “чистых функций”

```
1 usage
public List<RequestInfo> search(String districtToSearch, Requests req) {
    List<RequestInfo> list = new ArrayList<>();
    for (RequestInfo info : req.requests) {
        if (info.district.equals(districtToSearch)) {
            list.add(info);
        }
    }
    return list;
}
```

Изменение данных

Для достижения неизменяемости данных было применено копирование при записи, при котором создаётся копия данных перед её модификацией

```
1 usage
public RequestInfo setDistrict(RequestInfo info, String district) {
    RequestInfo copy = info.copy(info);
    copy.district = district;
    return copy;
}

1 usage
public RequestInfo setBuildingType(RequestInfo info, BuildingTypeEnum.BuildingType type) {
    RequestInfo copy = info.copy(info);
    copy.buildingType = type;
    return copy;
}

1 usage
public RequestInfo setRoomCount(RequestInfo info, int count) {
    RequestInfo copy = info.copy(info);
    copy.roomCount = count;
    return copy;
}
```

```
2 usages
public Requests add(RequestInfo a, Requests req) {
    Requests reqCopy = new Requests();
    reqCopy.requests = req.requests;
    reqCopy.requests.add(a);
    return reqCopy;
}
```

Для добавления новых данных так же применяется копировани.
И все изменения происходят с копией экземпляра. После чего функция возвращает изменённую копию.

Подведение итогов

В результате работы был переработан подход к использованию данных, в результате чего было увеличено количество “чистых функций”. Что в свою очередь благоприятно сказалось на потенциальном переиспользовании кода.

Однако в связи со спецификой рассматриваемой задачи, сократить количество кода не получилось.