Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г. И. Носова»
(ФГБОУ ВО «МГТУ им. Г.И. Носова»)

Кафедра вычислительной техники и программирования

Лабораторная работа №5

по дисциплине «Метрология и стандартизация программного обеспечения»

название лабораторной работы: «Оценка качества программы с помощью объектно-ориентированных метрик. Часть 2»

Исполнитель: Варламов М.Н., студент 3 курса, группа АВб-19-1

Руководитель: Сибилева Н.С., ст. преподаватель каф. ВТиП

Магнитогорск, 2022

Оглавление

Задание

Разработать структуру классов. Базовый класс – помещения. Производные – квартира и офис. Создать класс Дом, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно квартир и офисов (использовать оператор instanceof). Также, необходимо дополнить/переработать функционал программы таким образом, чтобы организовать расширенную информацию об исследуемой задаче. Добавить минимум 2 новых класса-родителя или наследника с соответствующими членами классов.

Структурный анализ программного кода

На рисунке 1 представлены блок-схемы алгоритмов каждого из методов класса.

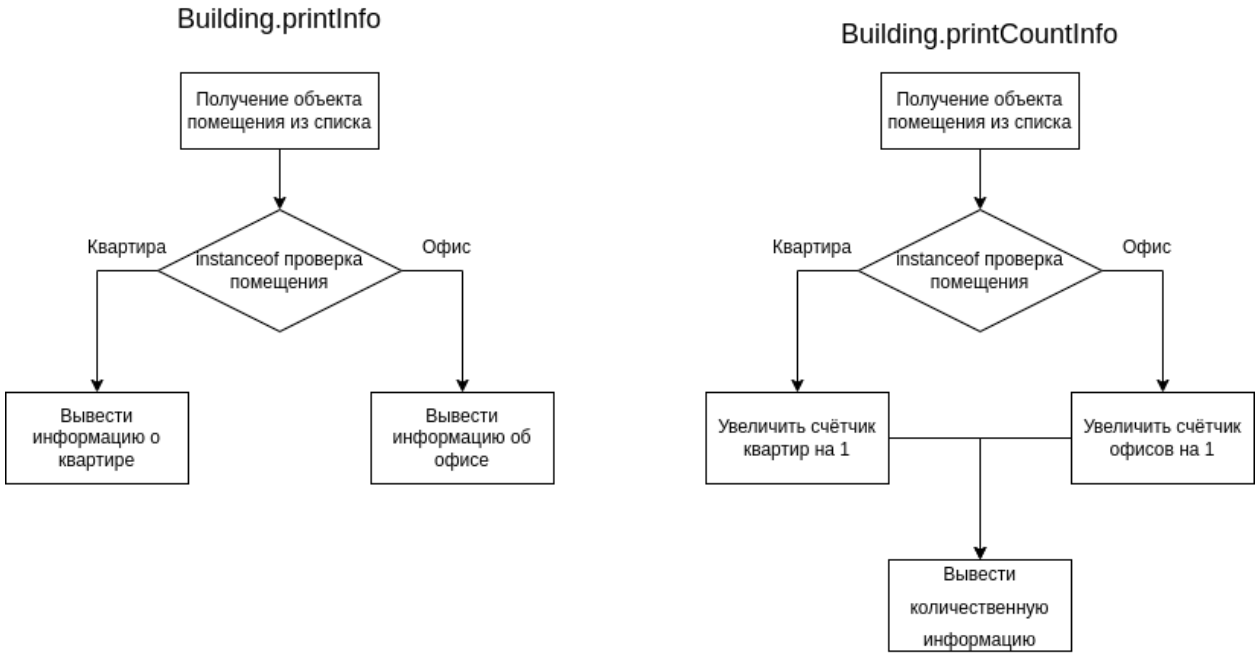

Рисунок 1 - блок-схемы алгоритмов методов класса.

Листинг

```
1.  public class Lab4 {
2.          public static void main(String[] args) {
3.          Building building = new Building();
4.          Scanner scanner = new Scanner(System.in);
5.          while (true) {
6.          System.err.print(
7.                  "1 - Add office \n" +
8.                  "2 - Add flat \n" +
9.                  "3 - Add shop \n" +
10.                 "4 - Add technical room \n" +
11.                 "5 - Print building info\n" +
```

```
                "6 - Print building count info\n" +
                "7 - Exit \n" +
                "Choose action:"
        );
        int action = scanner.nextInt();
        switch (action) {
                case 1: {
                System.err.print("Enter work places count: ");
                int workPlacesCount = scanner.nextInt();
                building.addRoom(new Office(workPlacesCount));
                System.err.println("\n Added!");
                break;
                }
                case 2: {
                System.err.print("Enter rooms count: ");
                int roomsCount = scanner.nextInt();
                building.addRoom(new Flat(roomsCount));
                System.err.println("\n Added!");
                break;
                }
                case 3: {
                System.err.print("Enter shop name: ");
                String name = scanner.nextLine();
                building.addRoom(new Shop(name));
                System.err.println("\n Added!");
                break;
                }
                case 4: {
                System.err.print("Enter technical room function: ");
                String function = scanner.nextLine();
                building.addRoom(new Technical(function));
                System.err.println("\n Added!");
                break;
                }
                case 5: {
                building.printInfo();
                break;
                }
                case 6: {
                building.printCountInfo();
                break;
                }
                default:
                return;
        }
        }
    }
}
```

```
1.  public class Building {
2.  private final List<Room> rooms;
3.          public Building() {
4.          rooms = new ArrayList<>()
5.           }
6.          public void addRoom(Room room) {
7.          rooms.add(room);
8.          }
9.           public void printInfo() {
10.         for (int i = 0; i < rooms.size(); i++) {
11.         Room room = rooms.get(i);
12.         String type = "";
13.         if (room instanceof Office)
14.                 type = "office";
15.
16.         if (room instanceof Flat)
17.                 type = "flat";
18.
19.         if (room instanceof Shop)
20.                 type="shop";
21.
22.         if (room instanceof Technical)
23.                 type="technical room";
24.
25.         System.err.println("On " + (i + 1) + " floor placed " + type + ". Info " +
    room.getInfo());
26.         }}
27.         public void printCountInfo() {
28.         int officeCount = 0;
29.         int flatCount = 0;
30.         int shopCount = 0;
31.         int technical = 0;
32.
33.         for (Room room : rooms) {
34.         if (room instanceof Office)
35.                 officeCount++;
36.         if (room instanceof Flat)
37.                 flatCount++;
38.         if (room instanceof Shop)
39.                 shopCount++;
40.         if (room instanceof Technical)
41.                 technical++;
42.         }
43.
44.         System.err.println(
45.                 "Flats count: " + flatCount + "\n" +
46.                 "Offices count: " + officeCount + "\n" +
47.                 "Shops count: " + shopCount + "\n" +
48.                 "Technical rooms count: " + technical + "\n" +
49.                 "Technical rooms count: " + officeCount + "\n" +
50.                 "Common count " + rooms.size());}}
```

На рисунке 2 представлена блок схема общей работы программного продукта.

Рисунок 2 - блок схема общей работы программного продукта.

- **Анализ качества программного кода с помощью метрик Лоренца и Кидда**

Метрика 1: Размер класса CS (Class Size)

Большие значения CS указывают, что класс имеет слишком много обязанностей. Они уменьшают возможность повторного использования класса, усложняют его реализацию и тестирование. Чем меньше среднее значение размера, тем больше вероятность повторного использования класса.

- Building: 5
- Flat: 3
- Office: 3
- Room: 1
- Shop: 3
- Technical: 3

Метрика 2: Количество операций, переопределяемых подклассом, NOO

Переопределением называют случай, когда подкласс замещает операцию, унаследованную от суперкласса, своей собственной версией. Большие значения NOO обычно указывают на проблемы проектирования.

- NOO: 1

Метрика 3: Количество операций, добавленных подклассом, NOA

Подклассы специализируются добавлением приватных операций и свойств. С ростом NOA подкласс удаляется от абстракции суперкласса. Обычно при увеличении высоты иерархии классов (увеличении DIT) должно уменьшаться значение NOA на нижних уровнях иерархии.

- NOA общ. 0

Метрика 4: Индекс специализации SI (Specialization Index)

Обеспечивает грубую оценку степени специализации каждого подкласса. Специализация достигается добавлением, удалением или переопределением операций:

- Office: (1 * 1) / 1 = 1
- Room: (1 * 1) / 1 = 1
- Shop: (1 * 1) / 1 = 1
- Technical: (1 * 1) / 1 = 1

Метрика 5: Средний размер операции OSAVG (Average Operation Size)

В качестве индикатора размера может использоваться количество строк программы, однако LOC-оценки приводят к известным проблемам.

- Building: 63
- Flat: 15
- Office: 15
- Room: 5
- Shop: 15
- Technical: 13

Метрика 6: Сложность операции OC (Operation Complexity)

Параметр	Вес
Вызовы функций API	5
Присваивания	0.5
Арифметические операции	2
Сообщения с параметрами	3

Параметры	0.3
Временные переменные	0.5

Значение метрики ОС вычисляется суммированием оценок с весовыми коэффициентами, приведенными в таблице.

- Building : ОС = 15*0,5 + 2*2 + 3*3 + 4 * 0.3 + 0.5 = 22.2
- Technical: ОС = 5*0,5 + 5*2 + 7*3 + 8*0.3 + 0.5 = 36.4
- Flat: ОС = 5*0,5 + 4*2 + 6*3 + 7*0.3 + 0.5 = 31.1
- Office: ОС = 4*0,5 + 3*2 + 5*3 + 6*0.3 + 0.5 = 25.8
- Room: ОС = = 5*0,5 + 3*2 + 5*3 + 6*0.3 + 0.5 = 26.3

Метрика 7: Среднее количество параметров на операцию NP_{AVG}

Чем больше параметров у операции, тем сложнее сотрудничество между объектами. Поэтому значение NP_{AVG}должно быть как можно меньшим.

Рекомендуемое значение NP_{AVG} = 0,7.

NP_{AVG} = 1.

Метрика 8: Количество описаний сценариев NSS (Number of Scenario Scripts)

Это количество прямо пропорционально количеству классов, требуемых для реализации требований, количеству состояний для каждого класса, а также количеству методов, свойств и сотрудничеств. Метрика NSS — эффективный индикатор размера программы.

Рекомендуемое значение NSS — не менее одного сценария на публичный протокол подсистемы, отражающий основные функциональные требования к подсистеме.

NSS = 8

Метрика 9: Количество ключевых классов NKC (Number of Key Classes)

Ключевой класс прямо связан с коммерческой проблемной областью, для которой предназначена система. Маловероятно, что ключевой класс может появиться в результате повторного использования существующего класса. Поэтому значение NKC достоверно отражает предстоящий объем разработки. М. Лоренц и Д. Кидд предполагают, что в типовой ОО-системе на долю ключевых классов приходится 20-40% от общего количества

классов. Как правило, оставшиеся классы реализуют общую инфраструктуру (GUI, коммуникации, базы данных).

NKC = 8

Метрика 10: Количество подсистем NSUB (NumberofSUBsystem)

Количество подсистем обеспечивает понимание следующих вопросов: размещение ресурсов, планирование (с акцентом на параллельную разработку), общие затраты на интеграцию.

Рекомендуемое значение: NSUB > 3.
NSUB = 0

Выводы

Была разработана структура классов. Базовый класс – помещения. Производные – квартира и офис. Создан класс Дом, который может содержать оба вида объектов. Предусмотреть метод подсчета отдельно квартир и офисов (использовать оператор instanceof).

С помощью метрик Лоренца и Кидда было выяснено, что написанная структура является полностью стабильна и мало абстрактна.