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: Ежов Г.А., студент 3 курса, группа АВб-21-12

Руководитель: Козлова А.Н., ст. преподаватель каф. ВТиП

Магнитогорск, 2024

**Оглавление**

|  |  |  |
| --- | --- | --- |
| **Задание** |  | **3** |
| **Структурный анализ программного кода** |  | **3** |
| **Анализ качества программного кода с Кидда** | **помощью** | **метрик Лоренца и**  **7** |

**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

class Room:

def get\_info(self):

raise NotImplementedError("Subclasses should implement this method.")

class Office(Room):

def \_\_init\_\_(self, work\_places\_count):

self.work\_places\_count = work\_places\_count

def get\_info(self):

return f"Work places count: {self.work\_places\_count}"

class Flat(Room):

def \_\_init\_\_(self, rooms\_count):

self.rooms\_count = rooms\_count

def get\_info(self):

return f"Rooms count: {self.rooms\_count}"

class Shop(Room):

def \_\_init\_\_(self, name):

self.name = name

def get\_info(self):

return f"Shop name: {self.name}"

class Technical(Room):

def \_\_init\_\_(self, function):

self.function = function

def get\_info(self):

return f"Function: {self.function}"

class Building:

def \_\_init\_\_(self):

self.rooms = []

def add\_room(self, room):

self.rooms.append(room)

def print\_info(self):

for i, room in enumerate(self.rooms):

type\_ = ""

if isinstance(room, Office):

type\_ = "office"

elif isinstance(room, Flat):

type\_ = "flat"

elif isinstance(room, Shop):

type\_ = "shop"

elif isinstance(room, Technical):

type\_ = "technical room"

print(f"On {i + 1} floor placed {type\_}. Info: {room.get\_info()}")

def print\_count\_info(self):

office\_count = 0

flat\_count = 0

shop\_count = 0

technical\_count = 0

for room in self.rooms:

if isinstance(room, Office):

office\_count += 1

elif isinstance(room, Flat):

flat\_count += 1

elif isinstance(room, Shop):

shop\_count += 1

elif isinstance(room, Technical):

technical\_count += 1

print(

f"Flats count: {flat\_count}\n"

f"Offices count: {office\_count}\n"

f"Shops count: {shop\_count}\n"

f"Technical rooms count: {technical\_count}\n"

f"Common count: {len(self.rooms)}"

)

def main():

building = Building()

while True:

print(

"1 - Add office \n"

"2 - Add flat \n"

"3 - Add shop \n"

"4 - Add technical room \n"

"5 - Print building info\n"

"6 - Print building count info\n"

"7 - Exit \n"

"Choose action:"

)

action = int(input().strip())

if action == 1:

work\_places\_count = int(input("Enter work places count: ").strip())

building.add\_room(Office(work\_places\_count))

print("\n Added!")

elif action == 2:

rooms\_count = int(input("Enter rooms count: ").strip())

building.add\_room(Flat(rooms\_count))

print("\n Added!")

elif action == 3:

name = input("Enter shop name: ").strip()

building.add\_room(Shop(name))

print("\n Added!")

elif action == 4:

function = input("Enter technical room function: ").strip()

building.add\_room(Technical(function))

print("\n Added!")

elif action == 5:

building.print\_info()

elif action == 6:

building.print\_count\_info()

elif action == 7:

break

if \_\_name\_\_ == "\_\_main\_\_":

main()

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 ОС (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 OC вычисляется суммированием оценок с весовыми коэффициентами, приведенными в таблице.

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

**Метрика 7: Среднее количество параметров на операцию NPAVG**

Чем больше параметров у операции, тем сложнее сотрудничество между объектами. Поэтому значение NPAVGдолжно быть как можно меньшим.

Рекомендуемое значение NPAVG = 0,7. NPAVG = 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.