**1.По текстовому описанию задачи построить диаграмму прецедентов.**

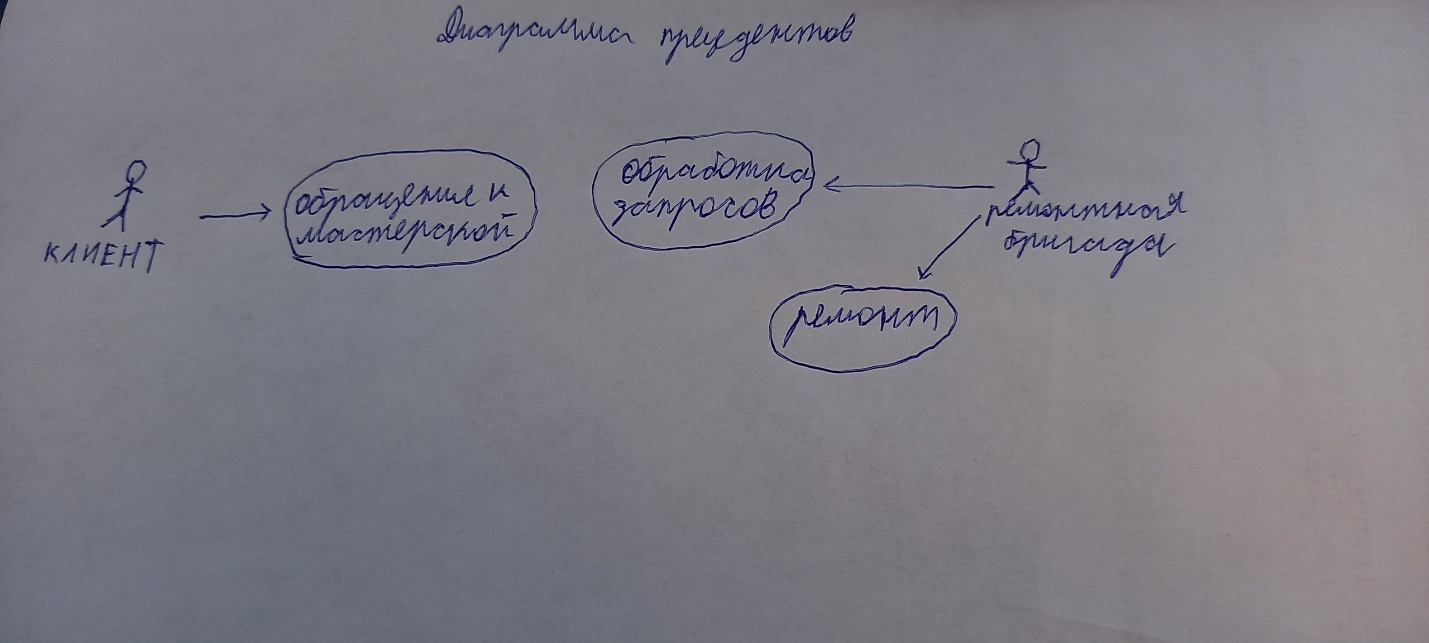


Рисунок 1.Диаграмма прецедентов

**2. По описанию и диаграмме прецедентов построить диаграмму классов.**

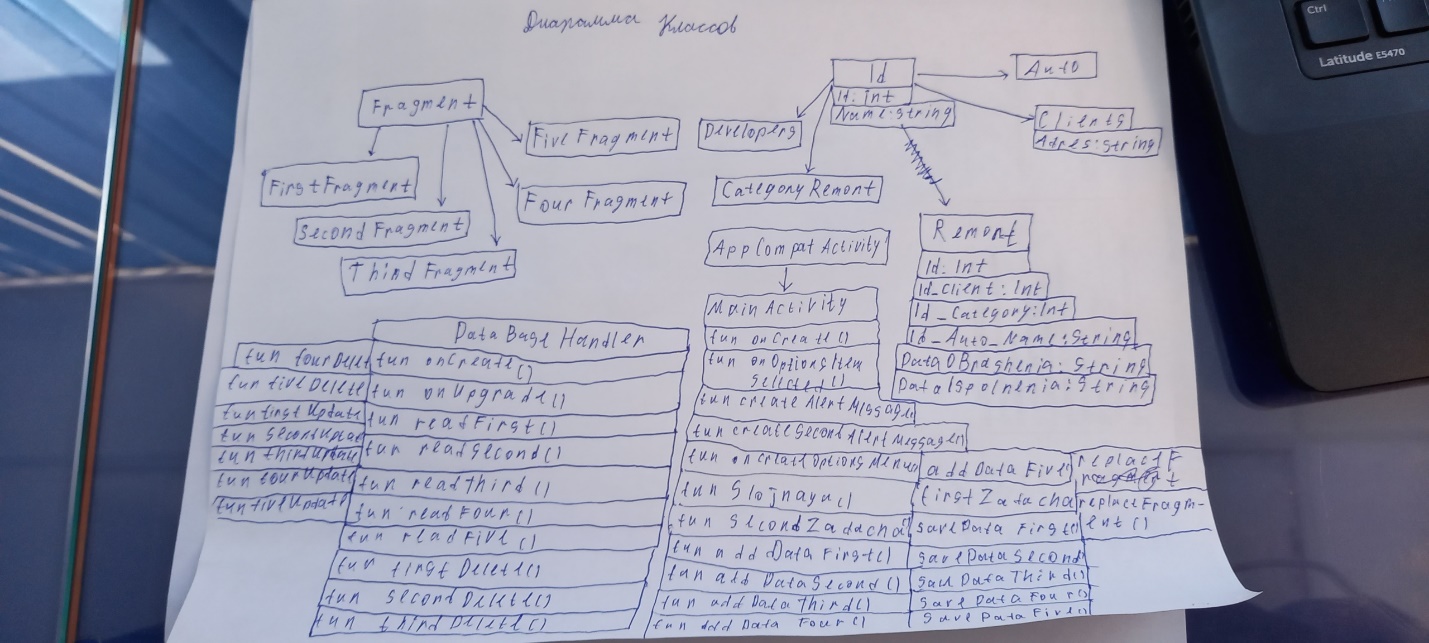


Рисунок 2.Диаграмма классов

**3.Разработать определения классов по диаграмме.**

Класс используемый для дальнейшего наследования, содержащий Id класса и название объекта класса, чтобы не пересоздавать его в каждом классе

public open class Id {

var Id:Int = 0

var Name:String = ""

}

Класс производителей, который наследует все свойства класса Id. Cодержит поля Кода производителя и название.

class Developers: Id() {}

Класс видов техники, который наследует все свойства класса Id. Cодержит поля Кода вида и название.

class Auto: Id() {}

Класс клиентов, который наследует все свойства класса Id. Cодержит поля Кода клиентов, ФИО и адреса.

class Clients: Id() {

var Adres:String = ""

}

Класс категорий ремонтов, который наследует все свойства класса Id. Cодержит поля Кода категории и название.

class CategoryRemont: Id() {}

Класс Ремонта не наследует какие либо классы. Он содержит поля: Код ремонта, Код клиента, Код вида техники, Код категории ремонта, Название техники, дата обращения, дата исполнения.

class Remont{

var Id:Int = 0

var Id\_Client:Int = 0

var Id\_Auto:Int = 0

var Id\_Category:Int = 0

var Id\_Auto\_Name:String = ""

var DataObrashenie:String = ""

var DataIspolnenia:String = ""

**4. Разработать проект с классами для выполнения задания.**

Проект с классами был разработан, все перечисленные классы из третьего пункта содержатся в файле Classes.kt. Помимо этого, был создан файл DataBase.kt, который содержит класс DataBaseHandler, используемый для взаимодействия с базой данных. Кроме того, были созданы классы FirstFragment, SecondFragment, ThirdFragment, FourFragment и FiveFragment, которые используются для графического отображения таблиц с классами.

**5. Показать откладку одного из модулей при разработке.**

В визуальной среде разработки Android Studio есть встроенные методы откладки. К примеру, на Рисунок 1 показана точка остановки, которая стоит на 113 строке. При выполнении программы, она остановится, когда достигнет этой строки.

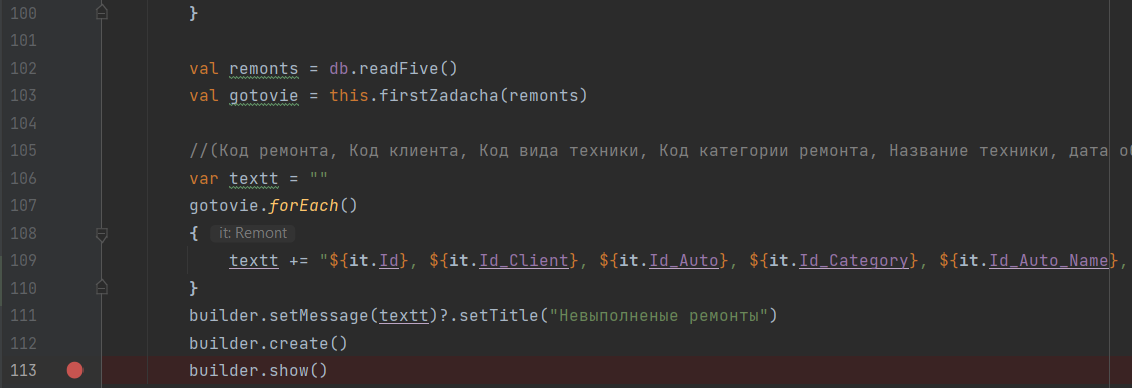


Рисунок 3. Точка остановки

На Рисунок 2 отображено окно дебаггера, который показывает все переменные используемые в блоке, в котором находится строка.

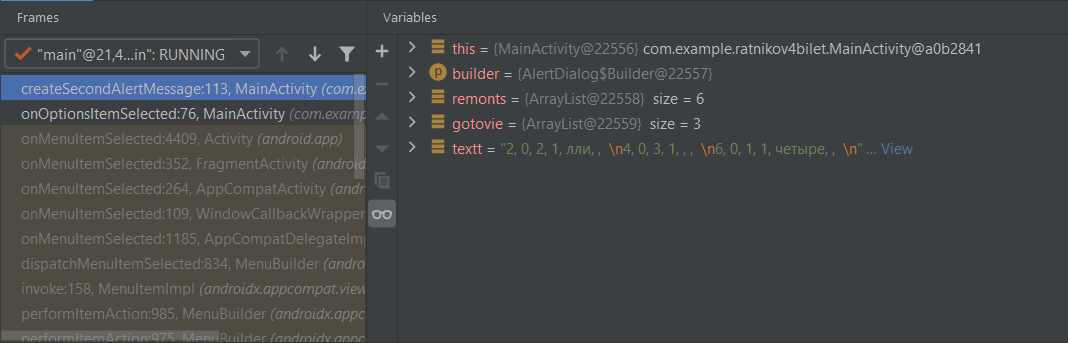


Рисунок 4. Дебаггер

**6. Подготовить тестовые наборы и провести тестирование одного из модулей.**

Для проверки и тестирования модулей программы используется библиотека junit.framework.TestCase. Для проведения тестирования достаточно выделить необходимый блок кода и нажать клавиши Ctrl + Shift + T. Создаём новый файл MainActivityTest и в него переносим функцию Slojnaya, которая в списке MutableList<CategoryRemont> находит объект с названием “Сложная”, и возвращает его Id.



Рисунок 5.Тестирование

Метод assertEquals сравнивает две переменные и выводит результат. Если они равны, значит тест пройден. Создадим пустой список и попробуем найти в нём объект, содержащий в названии “Сложная”. Запускаем тест

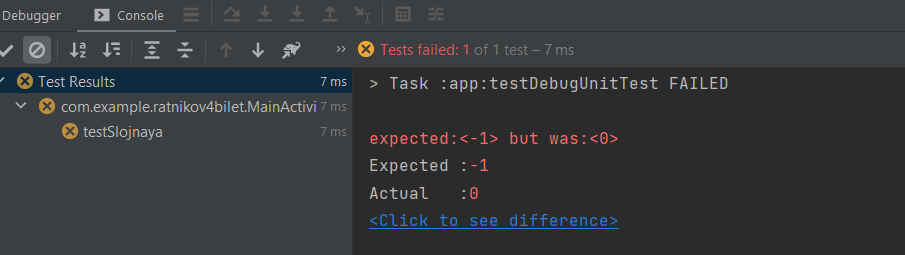


Рисунок 6. Окно результата

На Рисунок 6 показано Окно результата. Тест выдал ошибку, так как полученный результат из реализации функции Slojnaya не соответствует нулю.

**7. Определить значение временной сложности О для одного из методов. Указать значение О для оптимального кода.**

**Листинг 1: функция Slojnaya**

public fun Slojnaya(categoryRemont: MutableList<CategoryRemont>): Int {

categoryRemont.forEach(){

if (it.Name == "Сложная"){

return it.Id

}

}

return -1

}

Значение временной сложности данной функции равна O(n) – линейная сложность. Этот код оптимален. Проходим по списку и сравниваем каждую переменную объекта списка со строкой “Сложная”. Если они равны, возвращаем Id объекта. Если итератор доходит до конца, функция возвращает -1.