МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Объектно-ориентированное программирование»
Тема: Логическое разделение классов

Студентка гр. 8381 _____ Лисок М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Разработать и реализовать наборы классов для взаимодействия пользователя с юнитами и базой.

Задание.

Основные требования:

- Должен быть реализован функционал управления юнитами
- Должен быть реализован функционал управления базой

Дополнительные требования:

- Реализован паттерн “Фасад”, через который пользователь управляет программой
- Объекты между собой взаимодействуют через паттерн “Посредника”
- Для передачи команд используется паттерн “Команда”
- Для приема команд от пользователя используется паттерн “Цепочка обязанностей”

Основные классы, фасад, команды

Основные классы, добавленные в программу, и их назначение представлены в табл. 1.

Таблица 1 – Основные добавленные классы

Название	**Назначение**
Game (класс игры) (не относится к требованиям)	В классе хранится поле, массив баз, а также информация о юнитах. Класс обладает функционалом: создание баз, нейтральных объектов и помещение их на поле. С помощью посредника GameMediator между игрой и базами ведется учет юнитов: информация о них своевременно добавляется или удаляется.

GameCommand, (команда игры), FieldCommand (команда поля), BaseCommand (команда базы)	Реализованы по принципу паттерна «команда». Данные классы наследуется от абстрактного класса Command, описывающего все основные поля и методы необходимые для совершения определенного действия. Основные поля: • ассоциативный массив параметров map<string, Data> params, где Data структура, описывающая координаты x, y и string, описывающий что за координаты лежат в Data • тип запроса Actions action. Action представляет собой набор перечисление со всеми используемыми запросами(описан в файле enum.h) Основные методы: • map<string, int> mainInfoAboutObj() - основной метод, который выполняет команду. Функция в зависимости от action вызывает метод своего класса либо создает объект другой команды и передает все данные объекту, который в свою очередь выполняет указанное действие. Возвращают ассоциативный массив map<string, int>. В string записывается лог, характеризующий объект, с которым ведется действие. В int основы данные о предмете. • map<string, int> noSuchAct() - записывает лог, о том что не существует указанного действия в классе Команда поля дополнительно принимает на вход указатель на объект, если нужно определить его координаты. Команда поля отдельным методом может возвращать указатель на объект, если его нужно найти по координатам.
Facade (фасад)	Реализован по принципу паттерна «фасад». Фасад имеет множество методов, которые умеют работать с командами: передают параметры и обрабатывают выходные значения. Фасад не пользуется методами классов логики игры (напр. класса Game, Field, Base, Unit) – он создает и выполняет команду GameCommand, тем самым запуская цепочку обязанностей, в которой уже реализуется требуемый функционал.

Команды

Указанные в табл. 1 классы, реализованные по принципу паттерна «команда», имеют следующие свойства

- Каждая команда может пользоваться методами определенных классов, а именно
    - Команда GameCommand пользуется методами класса Game
    - Команда FieldCommand пользуется методами класса Field
    - Команда BaseCommand пользуется методами класса Base
- Входные параметры, указатель на используемый класс, указатель на посредника для команд, тип команды принимаются конструктором команды

Цепочки обязанностей, функционал

В программе реализован функционал управления базой, юнитами, а также вывод различной информации. Цепочки обязанностей формируются из классов команд и запускаются из фасада:

- Facade
    - GameCommand
        - FieldCommand
        - BaseCommand

Функциональные возможности программы, их описание и цепочка обязанностей представлены в табл. 2.

Таблица 2 – Функционал программы

Вывод игрового поля	Выводится сеткой	Facade > GameCommand > FieldCommand

Вывод информации об i-й базе	Вывод характеристик, местоположения на поле, состава базы (юнитов) Примечание к цепочке: GameCommand сначала получает от FieldCommand координаты базы, а затем уже вызывается BaseCommand	Facade > GameCommand > FieldCommand > GameCommand > BaseCommand
Вывод информации об объекте на поле	Информация берется по координатам (для юнитов - все характеристики, для базы - аналогично пункту выше, для нейтрального объекта - только название) Примечание к цепочке: GameCommand получает от FieldCommand указатель на объект, а далее в зависимости от его типа формирует ответ	Facade > GameCommand > FieldCommand
Добавление базы	Есть возможность добавления нескольких баз, и весь функционал учитывает эту возможность	Facade > GameCommand
Добавление юнита	Добавление юнита любого типа по координатам, для юнита выбирается база, которая его создает	Facade > GameCommand > BaseCommand
Добавление нейтрального объекта	Добавление нейтрального объекта любого типа по координатам	Facade > GameCommand
Перемещение юнита	Юнит выбирается по координатам, его перемещение определяется смещением по координатам Примечание к цепочке: GameCommand получает от FieldCommand указатель на объект, а затем ищет его в учете юнитов	Facade > GameCommand > FieldCommand
Атака юнита	Юнит выбирается по координатам, его атака определяется смещением по координатам Примечание к цепочке: аналогично перемещению юнита	Facade > GameCommand > FieldCommand

Посредник для взаимодействия юнитов

Атака юнитов реализована посредником AttackMediator, который хранит всех юнитов, а также указатель на поле. Доступ к посреднику тоже есть у всех юнитов. Посредник хранится в классе Game. Этапы взаимодействия юнитов представлены в табл. 3.

Таблица 3 – Основные этапы процесса атаки

1. Юнит методом attack() передает посреднику координаты смещения для атаки и указатель на себя
2. Посредник проверяет, что объект принадлежит полю и находит его координаты
3. Идет проверка, что атакуемый - тоже размещен на поле, и что он принадлежит другой базе
4. Сама атака осуществляется методом юнита receiveAttack(), который возвращает false, если юнит умер. В этом случае посредник обращается к полю для удаления юнита.

Выводы.

В ходе выполнения лабораторной работы была написана программа, в которой реализованы классы для функционала программы и взаимодействия пользователя с программой. Был использован объектно-ориентированный стиль программирования, были изучены и применены его основные положения, а также реализованы некоторые паттерны проектирования.