

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

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Объектно-ориентированное программирование»
Тема: Уровни абстракции, управление игроком

Студент гр. 1303

Коренев Д.А.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Реализовать набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы.

Задание.

Реализовать набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы (начать новую игру, завершить игру, сохраниться, управление игроком, и т.д.). Команды/клавиши, определяющие управление должны считываться из файла.

Требования:

- Реализован класс/набор классов обрабатывающие команды.
- Управление задается из файла (определяет какая команда/нажатие клавиши отвечает за управление, например, w - вверх, s - вниз, и т.д.).
- Реализованные классы позволяют добавить новый способ ввода команд без изменения существующего кода (например, получать команды из файла или по сети). По умолчанию, управление из терминала или через GUI, другие способы реализовывать не надо, но должна быть такая возможность.
- Из метода, считывающего команду, не должно быть “прямого” управления игроком.

Примечания:

- Для реализации управления можно использовать цепочку обязанностей, команду, посредника, декоратор, мост, фасад.

Выполнение работы.

Для выполнения лабораторной работы созданы и дополнены классы, отвечающие за считывание, преобразование и обработку данных пользователя.

Интерфейс `Configuration` содержит в себе словарь где ключ — название команды, значение — ее команда в программе — элемент `enum` класса `Control`. Также имеет виртуальный деструктор.

Класс `FileConfig` реализует вышеописанный интерфейс. Имеет метод `setConfig(string)`, принимающий строку (название файла), он открывает файл конфигурации (бросается ошибка, если его нельзя открыть). Если файл получилось открыть из него считываются данные из которых формируется конфигурация — словарь `settings` (ключ — `char`, значение — элемент класса `Control`). Процесс формирования конфигурации гарантируется корректные настройки: отсутствуют повторы действий (нельзя вызвать одно действие разными кнопками), отсутствуют повторы кнопок (нельзя вызвать несколько действий нажатием на одну кнопку). Метод `getSettings()` возвращает словарь `setting`. Конструктор и деструктор стандартные.

Интерфейс `InteractionUser` служит описанием всех объектов считывающих данные от пользователя и вывод ему необходимой информации. Имеет несколько чистых виртуальных методов: `void getCommand(Control&)` отвечает за считывание у пользователя команды во время игрового процесса, `bool getAnswerLevel(string)` отвечает за запрос от пользователя на считывание уровней логирования, `bool getAnswerConfig()` отвечает за требования использования конфига, `string readConfigName()` отвечает за считывание названия файла конфига, `bool getAnswerLogger(string)` отвечает за запрос от пользователя на типы логирования, `void getValue(string&)` отвечает за считывание какого-либо значения (например, размеров поля).

Класс `InteractionConsole` реализует интерфейс `InteractionUser`. Все виртуальные методы интерфейса определены на считывание(вывод) данных в(из) консоль(консоли): выводит данные в консоль, считывает данные от

пользователя тоже. Дополнительно создан метод `convertInput(char)` принимающий символ и возвращает элемент класса `Control` в соответствии со словарем `settings`. Данный словарь, на момент создания экземпляра, содержит стандартные значения конфигурации (ключ — `char`, значение — элемент `Control`), но может измениться если вызвать метод `setConfig()`. Данный метод запрашивает у пользователя необходимо ли изменять конфигурацию управления и при положительном ответе присваивает в поле `config` указатель на новый экземпляр `FileConfig()`. Далее вызывает у `config` метод `setConfig` с аргументом считанным названием файла и метод `getSettings`, возвращаемое значение присваивается в переменную `newSettings`. Проверяется полнота конфигурации (у всех команд присутствует ключ), присваивается в поле `settings`.

Класс `CommandReader` отвечает за запрос у пользователя данных, вызывая соответствующие методы у имеющегося объекта, который реализует интерфейс `InteractionUser`. Имеет указатель на `InteractionUser` т.е. может запрашивать у пользователя данные независимо от того какой объект хранится по данной ссылке, т.к. он реализует `InputReader`.

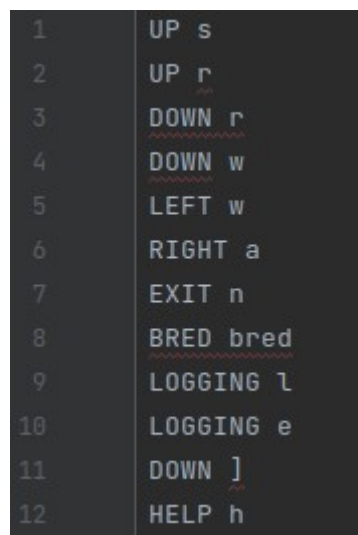
В конструкторе `Mediator` создают экземпляры `CommandReader` и `Controller`. Запрашивает у `CommandReader` данные для инициализации уровней и объектов вывода логов, использование конфига. Далее медиатор вызывает метод `notify` у `CommandReader` и `Controller`, пока игра не завершена. Вызывая этот метод у первого из них, он получает считанные данные, а у второму передает эти данные.

`Controller` имеет метод `notify(Control&)` и делегирует обработку методу `movePlayerPosition` полю `model`. Данный метод умеет обрабатывать передвижения игрока, другие игнорируются. Так как данные команд формируются в `InteractionConsole` (или другом классе, реализующий класс

InteractionUser), то для Controller не имеет значения способ ввода данных от пользователя.

Тестирование.

Тестирование программы: создание и обработка файла конфигурации. Пример файла конфигурации, который будет корректно преобразован (все ошибки обработаются) и установлен в качестве конфигурации, представлен на рисунке 1.



```
1 UP s
2 UP r
3 DOWN r
4 DOWN w
5 LEFT w
6 RIGHT a
7 EXIT n
8 BRED bred
9 LOGGING l
10 LOGGING e
11 DOWN ]
12 HELP h
```

Рисунок 1 - Пример
файла конфигурации

UML-диаграмма межклассовых отношений.

UML-диаграмма межклассовых отношений, созданных и измененных во время выполнения лабораторной работы, представлена на рисунке 2.

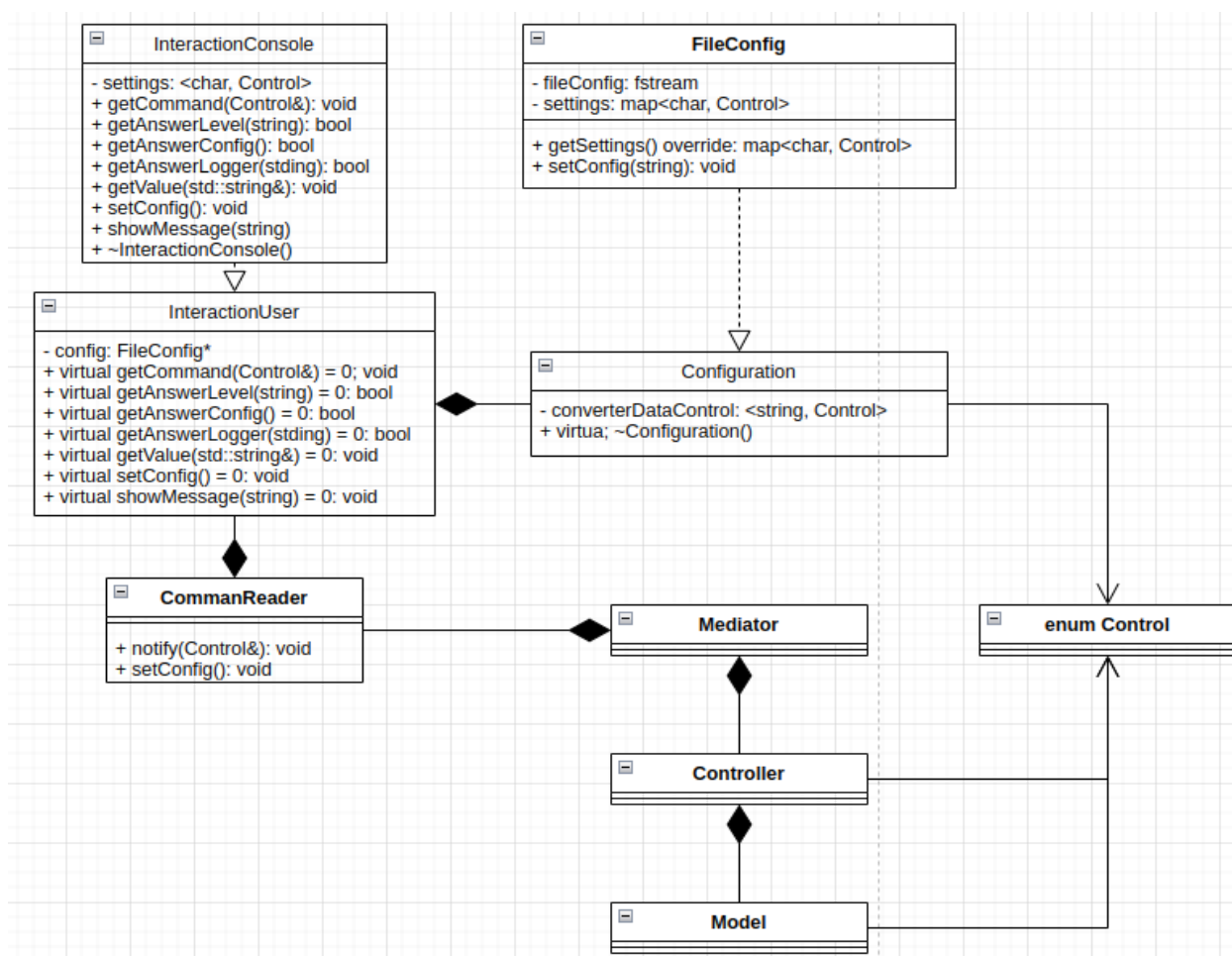


Рисунок 2. UML диаграмма межклассовых отношений

Выводы.

Реализован набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы.