**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: **Создание классов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2381 |  | Рыжиков И.С. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2023

## Цель работы

Изучить понятие класса, его методы, поля и модификаторы доступа. Научиться реализовывать простейшие классы и создавать межклассовые отношения.

## Задание

**Для данной лаб. работы UML диаграмма классов в отчете не обязательна.**

а) Создать класс игрока. У игрока должны быть поля, которые определяют его характеристики, например кол-во жизней, очков и.т.д. Также в классе игрока необходимо реализовать ряд методов для работы с его характеристиками. Данные методы должны контролировать значения характеристик (делать проверку на диапазон значений).

б) Создать класс, передвигающий игрока по полю и работу с характеристиками. Данный класс всегда должен знать об объекте игрока, которым управляет, но не создавать класс игрока. В следующих лаб. работах данный класс будет проводить проверку, может ли игрок совершить перемещение по карте.

Примечания:

* Не забывайте для полей и методов определять модификатор доступа
* Для указания направления движения можно использовать перечисление enum или дополнительную систему классов. Использования чисел или строк является для указания направления является плохой практикой
* Делать отдельный метод под каждое направление делает класс перегруженным, и в будущем ограничивает масштабирование класса

## Выполнение работы

Созданы два абстрактных класса: Observer и Observable для создание простой модели на основе шаблона *MVC (Model-View-Controller)*.

### Класс *Observer*

Класс *наблюдателя*. Используется как базовый класс, от которого наследуются другие классы.

*Подписан на сообщения* от экземпляров класса Observable. В случае, когда происходит уведомление об обновлении данных *вызывает метод* update, который производит необходимые действия, например, *рендер* интерфейса.

### Класс *Observable*

Класс, за которым наблюдают экземпляры класса Observer, другими словами *объекты наблюдения*. Используется как базовый класс, от которого наследуются другие классы.

Имеет два метода: addObserver и notifyUpdate.

С помощью метода addObserver происходит *подписание* одного из экземпляров Observer. При этом ссылка на экземпляр сохраняется в приватный массив, хранящий всех наблюдателей за данным объектом.

При вызове метода notifyUpdate происходит *обход всех наблюдателей, подписанных на обновление* этого объекта. При этом обходе у наблюдателей вызывается метод update.

### Класс Player

Класс содержащий *модель* игрока. Наследник Observable.

Обеспечивает *сокрытие данных игрока* от остального кода. А также предоставляет необходимые методы для работы с этими данными.

На данный момент создано два поля:

* name — для хранения *имени пользователя* (методы доступа setName для *установки* имени и getName для *получения* сохраненного имени);
* HP — для хранения *здоровья* игрока (методы доступа setHP и getHP)

### Класс PlayerView

Класс *представления* игрока. Наследник класса Observer. Тестовый класс, обеспечивает минимальную функциональность системы.

Конструктор принимает *ссылку на экземпляр класса игрока*. Сохраняет её в приватное поле для дальнейшего взаимодействия и *подписывается* на обновление игрока.

Имеет переопределенный метод update, который *отрисовывает* основные характеристики игрока.

### Класс PlayerController

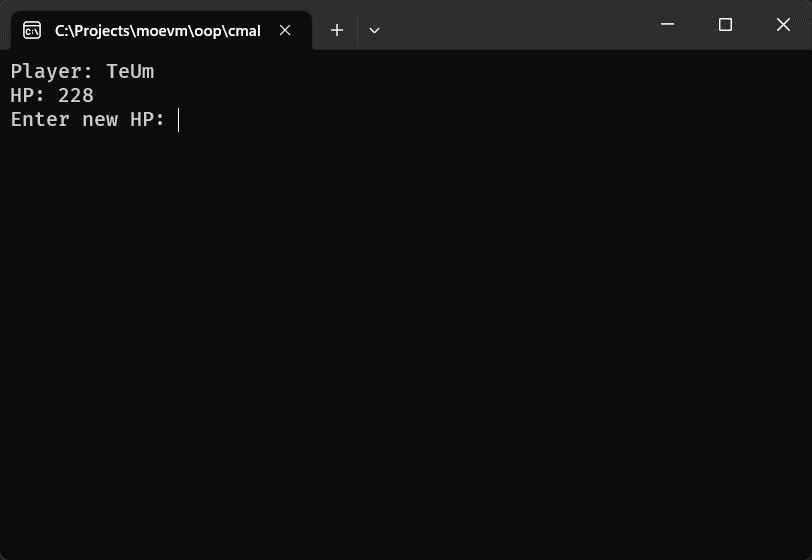
Класс *контроллер*. Используется для обновления данных пользователя. Связывает компоненты воедино и запускает приложение.

На текущем этапе развития приложения имеет единственный *метод* start. Который сначала *устанавливает имя пользователя*. А потом создает бесконечный *цикл* для работы приложения.

В этом цикле контроллер ожидает *ввода* новых данных от пользователя. Контроллер *обновляет* данные игрока. *Игрок уведомляет* модель представления PlayerView, которая в свою очередь *отрисовывает новые данные на экране*.

Так происходит, пока здоровье игрока не станет равно 0. После чего *программа завершится*.

## Работа программы



1. Пример работы программы в терминале

## Выводы

Были изученные основы объектно-ориентированного программирования на языке C++. В ходе работы создан класс игрока, классы отвечающие за вывод и взаимодействия пользователя с игрой, а также классы, определяющие дальнейшую структуру программы.

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

Код