

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, конструкторов и методов классов»

Студент гр. 1303

Коренев Д.А.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2022

Цель работы.

Изучить основы объектно-ориентированного программирования, научиться реализовывать простые классы и связывать их между собой.

Задание.

Реализовать прямоугольное игровое поле, состоящее из клеток. Клетка - элемент поля, которая может быть проходима или нет (определяет, куда может стать игрок), а также содержит какое-либо событие, которое срабатывает, когда игрок становится на клетку. Для игрового поля при создании должна быть возможность установить размер (количество клеток по вертикали и горизонтали). Игровое поле должно быть зациклено по вертикали и горизонтали, то есть если игрок находится на правой границе и идет вправо, то он оказывается на левой границе (аналогично для всех краев поля).

Реализовать класс игрока. Игрок - сущность контролируемая пользователем. Игрок должен иметь свой набор характеристик и различный набор действий (например, разные способы перемещения, попытка избежать событие, и так далее).

Требования:

- Реализован класс игрового поля
- Для игрового поля реализован конструктор с возможностью задать размер и конструктор по умолчанию (то есть конструктор, который можно вызвать без аргументов)
- Реализован класс интерфейс события (в данной лабораторной это может быть пустой абстрактный класс)
- Реализован класс клетки с конструктором, позволяющим задать ей начальные параметры.

- Для клетки реализованы методы реагирования на то, что игрок перешел на клетку.
- Для клетки реализованы методы, позволяющие заменять событие. (То есть клетка в ходе игры может динамически меняться)
- Реализованы конструкторы копирования и перемещения, и соответствующие им операторы присваивания для игрового поля и при необходимости клетки
- Реализован класс игрока минимум с 3 характеристиками. И соответствующие ему конструкторы.
- Реализовано перемещение игрока по полю с проверкой допустимости на переход по клеткам.

Примечание:

- При написании конструкторов учитывайте, что события должны храниться по указателю для соблюдения полиморфизма
- Для управления игроком можно использовать медиатор, команду, цепочку обязанностей

Выполнение работы.

Для выполнения лабораторной работы были созданы классы, отвечающие за игрока, поле, клетки поля, их отображение, взаимодействие пользователя с программой.

- 1) Интерфейс MediatorObject имеет один virtual метод — notify. Далее применяется для создания паттерна — Mediator. Причина его создания — достижение слабой связи между объектами CommandReader и Controller (описаны далее): не ссылаются друг на друга.
- 2) Класс CommandReader реализует интерфейс MediatorObject. Его смысл — считывание данных от пользователя, однако он не «знает» для чего

эти данные нужны. Методы `getFieldWidth` и `getFiledHeight` считывают беззнаковое число, а метод `checkUIData` проверяет корректность ввода. Метод `getPlayerMove` считывает символ. Метод `notifu` из родительского интерфейса переопределен — он вызывает `getPlayerMove`.

- 3) Класс `Controller` тоже реализует интерфейс `MediatorObject`. Его смысл — преобразование команд, полученных от пользователя, во внутренние команды программы. Имеет поля `Field`, `FieldView`, `Player` чтобы управлять игровым полем (оно создается с помощью метода `createField`), выводит его внешний вид в консоль (для этого используется метод `printFieldView`), управлять игроком (метод `movePlayerPosition`). Метод `createField` принимает два значения: ширина и высота игрового поля, и присваивает в поле `field` объект `Field`. Метод `printFieldView` делегирует метод `printFieldView` у `fieldView` с передачей в качестве аргумента — ссылку на поле `field`. Метод `movePlayerPosition` принимает символ, преобразует его в команду и вызывает метод `movePlayer` у поля `field` с передаче в качестве аргумента команду. Переопределенный метод родительского класса `notifu` вызывает методы `movePlayerPosition` и `printFieldView`.
- 4) Класс `Mediator` взаимодействует с объектами `Controller` и `CommandReader`, ссылки на которые он принимает в своем конструкторе. Метод `start` вызывает у `CommandReader` метод считывания от пользователя параметров поля и передает их в качестве аргументов методу `Controller`, отвечающий за создания поля. Далее вызывает метод `update`, пока тот не прекратит свою работу. Она заключается в вызове метода `notifu` с аргументами `CommandReader` и `Controller`, а так же проверяет условие остановки. Метод `notifu`

принимает ссылку на объект `MediatorObject` и вызывает у этого объекта метод `notify`.

- 5) Класс `Field` представляет из себя игровое поле, имеет двумерный динамический массив клеток (дочерние экземпляры абстрактного класса `Cell`), созданный с помощью `vector`'а — поле `field`, так же имеет поля `fieldSize` и `playerPosition` в которых хранятся размеры поля и координата игрока, соответственно. Реализованы конструкторы копирования и перемещения, оператор присваивания. Метод `setField` позволяет создать игровое поле: зная требуемые размеры игрового поля, выделяется память и создает для каждой координаты клетку, для некоторых из которых присваивает событие, наконец, создает игрока и «ставит» его на поле. Метод `movePlayer` меняет положения игрока на поле, приняв в качестве аргумента команду, преобразует ее в действие, проверяет корректность данных или корректирует их, перемещает игрока на другую клетку и вызывая ее событие.
- 6) Класс `Cell` представляет из себя общую сущность клетки, единицу игрового поля. Имеет поле `id` и указатель на событие, хранящееся в клетке (`nullptr` по умолчанию). Имеет 4 метода, один из которых `virtual`: `getId` возвращает `id` клетки, `setEvent` устанавливает другое событие в клетку, `callEvent` вызывает событие клетки, `isPassable` возвращает булево значение — может ли игрок находится на этой клетке. Его наследуют следующие классы:
 - 1) `CellGrass`, имеет единственный метод — конструктор, который присваивает значение 0 в поле `id` и `true` в родительское поле `passable`
 - 2) `CellPlayer`, имеет единственный метод — конструктор, который присваивает значение 1 в поле `id` и `false` в родительское поле `passable`

- 3) CellWall, имеет единственный метод — конструктор, который присваивает значение 2 в поле id и false в родительское поле passable
- 7) Класс FieldView создан для отображения игрового поля в консоли. Имеет поле cellView - объект CellView. И методы: printFiledView, конструктор копирования и оператор присваивания. Первый из них принимает указатель на объект класса Field и итерационно вызывает метод getView у объекта в поле cellView для каждого элемента поля filed объекта Field.
- 8) Класс CellView умеет преобразовывать клетку в ее внешний вид. Для этого создано поле view типа std::map<int, char> заполненное парами ключ-значение. Метод getView принимает ссылку на объект Cell, возвращает значение из view, по ключу равному id клетки.
- 9) Интерфейс Event имеет два виртуальных метода: деструктор и callReaction. Его реализуют классы
- 1) TrapEventBanana
 - 2) TrapEventJoker

В каждом переопределен деструктор и метод callReactio

- 10) Класс Player описывает сущность игрока, он не знает где он, например, находится, но знает свои характеристика. Имеет поля health, shield, xp и методы способные изменять значения этих полей. Имеет перечисление STEP характеризующее направление игрока.

Строение системы классов:

- 5) Между FiledView и CellView композиция т.к. у конкретного представления поля должно быть конкретное представление клеток, FieldView управляет временем жизни CellView.
- 6) Между Filed и Cell композиция т.к. поле состоит из клеток, Field управляет временем жизни каждого объекта Cell.
- 7) Между FieldView и Field зависимость, т.к. FieldView всего лишь принимает ссылку на объект Field в одном из методов.
- 8) Controller композитует FieldView, Field, Player и управляет временем жизни каждого из них.
- 9) Mediator агрегирует Controller и CommandReader, не управляет временем их жизни, но взаимодействует с ними, имеет поля-ссылки на их экземпляры.
- 10) Controller и CommanReader реализуют интерфейс MedoatorObject, это сделано для создания паттерна «Mediator».

Тестирование

Тестирование программы


```
Enter filed Height (minimum 10)
Height: 5
Enter filed width (minimum 10)
Width: 15

+ - - - - - - - - - - - - - - +
|      #  #      |
|      #  #      |
|       #  #      |
|        #  #      |
|         #  #      |
|          #  #      |
|           #  #      |
|            #  #      |
|             #  #      |
|              #  #      |
| *                #  #      |
+ - - - - - - - - - - - - - - +
a
+ - - - - - - - - - - - - - - +
|      #  #      |
|      #  #      |
|       #  #      |
|        #  #      |
|         #  #      |
|          #  #      |
|           #  #      |
|            #  #      |
|             #  #      |
|              #  #      |
|                #  #      *
+ - - - - - - - - - - - - - - +
b
Process finished with exit code 0
```

Рис. 2 - Тестирование

Выводы.

В ходе выполнения лабораторной работы были приобретены знания об объектно-ориентированном программировании и опыт работы с ним на примере практической задачи на C++. Были приобретены навыки написания классов, конструкторов и методов