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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Программирование»
Тема: Введение в ООП: создание классов, конструкторов классов, методов классов; наследование

Студентка гр. 0324 _____ Жигалова Д. А.

Преподаватель _____ Глущенко А. Г.

Санкт-Петербург
2021

Цель работы.

Знакомство с ООП: овладение навыками создания классов, их конструкторов и методов, а также изучение концепции наследования.

Основные теоретические положения.

Класс является абстрактным типом данных, определяемым пользователем, и представляет собой модель реального объекта в виде данных и функций для работы с ними.

Данные класса называются полями (по аналогии с полями структуры), а функции класса – методами. Поля и методы называются элементами класса.

Класс может определять переменные и константы для хранения состояния объекта и функции для определения поведения объекта.

Спецификаторы доступа private и public управляют видимостью элементов класса. Элементы, описанные после служебного слова private, видимы только внутри класса. Этот вид доступа принят в классе по умолчанию. Интерфейс класса описывается после спецификатора public. Действие любого спецификатора распространяется до следующего спецификатора или до конца класса. Можно задавать несколько секций private и public, порядок их следования значения не имеет.

В каждом классе есть хотя бы один метод, имя которого совпадает с именем класса. Он называется конструктором и вызывается автоматически при создании объекта класса. Конструктор предназначен для инициализации объекта. Автоматический вызов конструктора позволяет избежать ошибок, связанных с использованием неинициализированных переменных.

Наследование

Механизм наследования классов позволяет строить иерархии, в которых производные классы получают элементы родительских, или базовых, классов и могут дополнять их или изменять их свойства. При большом количестве никак не связанных классов управлять ими становится невозможным. Наследование позволяет справиться с этой проблемой путем упорядочивания и ранжирования

классов, то есть объединения общих для нескольких классов свойств в одном классе и использования его в качестве базового.

Классы, находящиеся ближе к началу иерархии, объединяют в себе наиболее общие черты для всех нижележащих классов. По мере продвижения вниз по иерархии классы приобретают все больше конкретных черт. Множественное наследование позволяет одному классу обладать свойствами двух и более родительских классов.

Фабричный метод

Фабричный метод – это порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Для того, чтобы система оставалась независимой от различных типов объектов, паттерн Factory Method использует механизм полиморфизма - классы всех конечных типов наследуют от одного абстрактного базового класса, предназначенного для полиморфного использования. В этом базовом классе определяется единый интерфейс, через который пользователь будет оперировать объектами конечных типов.

Постановка задачи.

Разработать и реализовать набор классов:

- Класс игрового поля
- Набор классов юнитов

Игровое поле является контейнером для объектов, представляющим прямоугольную сетку. Основные требования к классу игрового поля:

- Создание поля произвольного размера
- Контроль максимального количества объектов на поле
- Возможность добавления и удаления объектов на поле
- Возможность копирования поля (включая объекты на нем)

Юнит является объектов, размещаемым на поля боя. Один юнит представляет собой отряд. Основные требования к классам юнитов:

- Все юниты должны иметь как минимум один общий интерфейс
- Реализованы 3 типа юнитов (например, пехота, лучники, конница)
- Реализованы 2 вида юнитов для каждого типа (например, для пехоты могут быть созданы мечники и копейщики)
- Юниты имеют характеристики, отражающие их основные атрибуты, такие как здоровье, броня, атака.
- Юнит имеет возможность перемещаться по карте

Выполнение работы.

Для решения поставленных задач была создана программа на языке программирования C#.

Для создания поля произвольного размера Grid был создан массив Map.

Класс Grid имеет поля map, height и width со значениями которых происходит отрисовка Поля.

Проходя по массиву Map происходит отрисовка. Класс одиночка gameManager позволяет хранить данные поля и фабрики юнитов на нем в единственном экземпляре. GameManager содержит свойства: grid, player, BaseUnitFactory и MortalUnitFactory.

Для создания юнитов был использован Фабричный метод. Был создан абстрактный класс фабрики UnitFactory содержащий метод получения и создания Юнита. Классы наследники BaseUnitFactory и MortalUnitFactory переписывают метод, создающий IUnit. Интерфейс IUnit – интерфейс всех юнитов, объявляющий функцию getID.

Созданы 2 типа Юнитов: Base и Mortal. Base наследуют интерфейс IUnit, Mortal наследуют интерфейс IUnit и абстрактный класс Mortal со свойством здоровья. Base юниты это стены и пустой пол. Mortal юниты это игрок и враги.

Юнит Игрок имеет возможность передвижения по полю методом Move, учитывающий столкновение со стенами и врагами.

Выводы.

При разработке программы были изучены основные порождающие паттерны проектирования и основы ООП.

ИТОГОВЫЙ КОД ПРОГРАММЫ

Исходный код файла Program.cs

```
using System;

namespace course_work
{
    class Program
    {
        static void Main(string[] args)
        {
            gameManager.Instance.BaseUnitFactory = new BaseFactory();
            gameManager.Instance.MortalUnitFactory = new MortalFactory();
            // 5 5
            byte[] map1 = { 1, 1, 1, 1, 1,
                            1, 0, 0, 0, 1,
                            1, 0, 2, 0, 1,
                            1, 0, 0, 4, 1,
                            1, 1, 1, 1, 1
            };// 8 5
            byte[] map2 = { 1, 1, 1, 1, 1, 1, 1, 1,
                            1, 0, 1, 0, 0, 0, 0, 1,
                            1, 0, 1, 0, 1, 0, 0, 1,
                            1, 0, 0, 4, 1, 0, 2, 1,
                            1, 1, 1, 1, 1, 1, 1, 1
            };// 6 8
            byte[] map3 = { 1, 1, 1, 1, 1, 1,
                            1, 0, 0, 0, 0, 1,
                            1, 0, 1, 1, 0, 1,
                            1, 0, 1, 4, 0, 1,
                            1, 0, 1, 0, 0, 1,
                            1, 0, 1, 1, 1, 1,
                            1, 0, 0, 0, 0, 1,
                            1, 1, 1, 1, 1, 1
            };
            gameManager.Instance.grid = new Grid(map1, 5, 5);

            string message = "";
            while (true)
            {
                Console.Clear();
                // foreach (var item in gameManager.Instance.grid.Units)
                // {
                //     Console.WriteLine(item);
                // }
                if (message != "") { Console.WriteLine(message); message = ""; }
                gameManager.Instance.grid.draw();
                Console.WriteLine($"Здоровье:
{gameManager.Instance.player.health}\nПеремещение:\n1 - Шаг вверх\n2 - Шаг
вправо\n3 - Шаг вниз\n4 - Шаг влево\n0 - Выйти");
```

```
                string input = Console.ReadLine();
                bool success = Int32.TryParse(input, out int parsedInput);
                if (!success) { message = "Вводить можно только цифры"; continue; }
                if (parsedInput == 0) break;
                switch (parsedInput)
                {
                    case >=1 and <=4:
                        message = gameManager.Instance.player.move(parsedInput);
                        break;
                    default:
                        message = "Моя твоя не понимать, еретик!";
                        break;
                }
            }
        }
    }
}
```

Исходный код файла grid.cs

```
using System.Collections.Generic;

namespace course_work {
    public enum cellType : byte {
        Floor = 0,
        Wall = 1,
        Player = 4,
        Warrior = 2
    }

    public class Grid {
        private List<IUnit> board {get; set;}
        public int width {get; set;}
        public int height {get; set;}
        public Grid(Grid copy) {
            this.width = copy.width;
            this.height = copy.height;
            this.board = new List<IUnit>(copy.board);
        }

        public Grid(byte[] map, int width, int height) {
            this.board = new List<IUnit>();
            this.width = width;
            this.height = height;
            foreach (byte item in map)
            {
                IUnit newUnit;
                switch (item)
                {
                    case 0 or 1:
```

```
                        newUnit =
gameManager.Instance.BaseUnitFactory.createUnit((cellType)item);
                        break;
                    case 2 or 4:
                        newUnit =
gameManager.Instance.MortalUnitFactory.createUnit((cellType)item);
                        if (newUnit as Player != null) {
                            gameManager.Instance.player = (Player)newUnit;
                        }
                        break;
                    default:
                        newUnit = null;
                        break;
                }
                this.board.Add(newUnit);
            }
        }
        public void draw() {
            for (int i = 0; i < this.height; i++) {
                for (int j = 0; j < this.width; j++){
                    System.Console.Write($"{this.board[Grid.oneDIndex(i, j,
this.width)]} ");
                }
                System.Console.WriteLine();
            }
        }
        public IUnit getCellDir(IUnit unit, int dir) {
            int index = this.getUnitIndex(unit);
            IUnit r;
            switch (dir)
            {
                case 1:
                    r = this.board[index-this.width];
                    break;
                case 2:
                    r = this.board[index+1];
                    break;
                case 3:
                    r = this.board[index+this.width];
                    break;
                case 4:
                    r = this.board[index-1];
                    break;
                default:
                    r = new Wall();
                    break;
            };
            return r;
        }
        public void moveUnit(IUnit unit, int i) {
```

```
            int index = this.getUnitIndex(unit);
            this.board[index] = new Floor();
            this.board[i] = unit;
        }
        public void addUnit(IUnit unit, int i) => this.board[i] = unit;
        public void removeUnit(IUnit unit) {
            int i = this.board.IndexOf(unit);
            if (i != -1) {
                this.board[i] = new Floor();
            }
        }
        public int dir2offset(int dir) {
            int offset = 0;
            switch (dir)
            {
                case 1:
                    offset -= this.width;
                    break;
                case 2:
                    offset += 1;
                    break;
                case 3:
                    offset += this.width;
                    break;
                case 4:
                    offset -= 1;
                    break;
                default:
                    break;
            };
            return offset;
        }
        public static int oneDIndex(int x, int y, int w) => (x * w) + y;
        public static cellType IUnit2cellType(IUnit unit) =>
(cellType)unit.getID();
        public int getUnitIndex(IUnit unit) => this.board.IndexOf(unit);
        public IEnumerable<IUnit> Units
        {
            get {
                for (int i = 0; i < this.board.Count; i++) yield return
this.board[i];
            }
        }
        private IEnumerator<IUnit> GetEnumerator()
        {
            for (int i = 0; i < this.board.Count; i++) yield return this.board[i];
        }
    }

}
```

Исходный код файла factory.cs

```
namespace course_work
{
    public abstract class UnitFactory {
        public IUnit GetUnit(cellType UnitT) => createUnit(UnitT);
        public abstract IUnit createUnit(cellType UnitT);
    }

    public class BaseFactory : UnitFactory {
        public override IUnit createUnit(cellType UnitT) {
            switch (UnitT)
                {
                    case cellType.Floor:
                        return new Floor();
                    case cellType.Wall:
                        return new Wall();
                    default:
                        System.Console.WriteLine("Can't create type " + UnitT);
                        return null;
                }
        }
    }
    public class MortalFactory : UnitFactory {
        public override IUnit createUnit(cellType UnitT) {
            switch (UnitT)
                {
                    case cellType.Warrior:
                        return new Warrior();
                    case cellType.Player:
                        return new Player();
                    default:
                        System.Console.WriteLine("Can't create type " + UnitT);
                        return null;
                }
        }
    }
}
```

Исходный код файла units.cs

```
namespace course_work {
    public interface IUnit
    {
        public byte getID();
    }

    public abstract class Mortal {
        public byte health { get; set; }
    }
```

```
public class Player : Mortal, IUnit {
    private static byte ID = 4;
    private bool canWalk {get; set;}

    public Player () {this.health = 10; this.canWalk = true;}
    public byte getID() => ID;
    public string move(int dir) {
        if (!this.canWalk) {
            return "Game Over";
        }
        Grid g = gameManager.Instance.grid;
        IUnit cell = g.getCellDir(this, dir);
        cellType cellT = Grid.IUnit2cellType(cell);
        switch (cellT)
        {
            case cellType.Floor:
                g.moveUnit(this, g.getUnitIndex(this) + g.dir2offset(dir));
                return "";
            case cellType.Wall:
                return "Не могу туда пойти!";
            case cellType.Warrior:
                Mortal enemy = (Mortal)cell;
                this.health -= enemy.health;
                this.health = (this.health >= 200) ? (byte)0 : this.health;

                if (this.health == 0) {
                    this.canWalk = false;
                    gameManager.Instance.grid.removeUnit(this);
                    return "Game Over";
                }
                g.moveUnit(this, g.getUnitIndex(this) + g.dir2offset(dir));
                return $"Игрок потерял 2 здоровья, но победил врага";
            default:
                return "ОшибОчка";
        }
    }
    public override string ToString() => "@";
}

public class Warrior : Mortal, IUnit {
    private static byte ID = 2;
    public Warrior () {this.health = 2;}

    public byte getID() => ID;

    public override string ToString() => "!";
}
public class Floor : IUnit {
    private static byte ID = 0;
    public Floor () {}
```

```
        public byte getID() => ID;

        public override string ToString() => ".";
    }

    public class Wall : IUnit {
        private static byte ID = 1;

        public Wall () {}
        public byte getID() => ID;

        public override string ToString() => "#";
    }

}
```

Исходный код файла gameManager.cs

```
namespace course_work{
    public sealed class gameManager {
        private static gameManager _instance;
        private gameManager() {}
        public static gameManager Instance {
            get {
                if (_instance == null) {
                    _instance = new gameManager();
                }
                return _instance;
            }
        }
        public Grid grid {get; set;}
        public Player player {get; set;}
        public BaseFactory BaseUnitFactory {get; set;}
        public MortalFactory MortalUnitFactory {get; set;}
    }
}
```

РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

Обозначения:

- Стена – « # »
- Пустые клетки – « . »
- Игрок – « @ »
- Враг – « ! »

```
# # # # #
# . . . #
# . ! . #
# . . @ #
# # # # #
Здоровье: 10
Перемещение:
1 - Шаг вверх
2 - Шаг вправо
3 - Шаг вниз
4 - Шаг влево
0 - Выйти
```

Рисунок 1 - Демонстрационный пример поля 1

```
# # # # # # # #
# . # . . . . #
# . # . # . . #
# . . @ # . ! #
# # # # # # # #
Здоровье: 10
Перемещение:
1 - Шаг вверх
2 - Шаг вправо
3 - Шаг вниз
4 - Шаг влево
0 - Выйти
```

Рисунок 2 - Демонстрационный пример поля 2

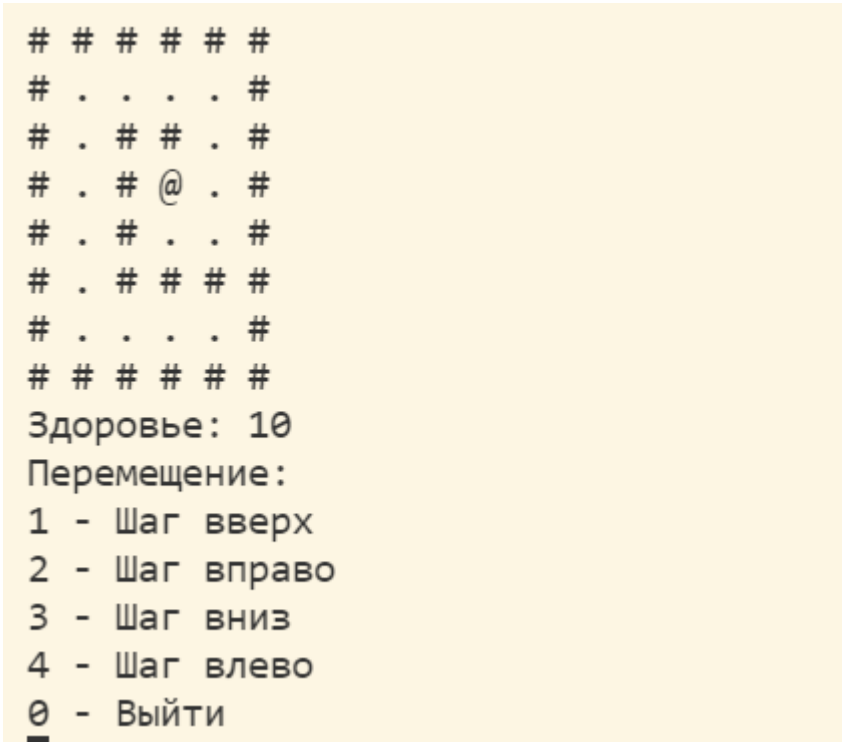

Рисунок 3 - Демонстрационный пример поля 3