Пензенский государственный технологический университет

Факультет автоматизированных информационных технологий

Кафедра «Прикладная информатика»

ЕН.Ф.02

**КОНСТРУКТОРЫ, НАСЛЕДОВАНИЕ**

Отчет о лабораторной работе № 4

по дисциплине «Информатика и программирование»

Выполнил: ст-т гр.17СН1с

Тюняев А.В., Богданов М.О.

Проверил: ст. преп. каф. ПИ

Юранов В.С.

2018

**1 Формулировка задачи**

1. В приложение из 1 лабораторной работы необходимо добавить классы фигуры и соответствующие классы наследники – горизонтальная линия и вертикальная линия.
2. В основной программе создать объекты данных классов, которые формируют на экран прямоугольник.
3. Сделать коммит изменений. Отправить изменения в удаленный репозиторий

**2 Требования к программе**

Программа должна выводить символы на экран, которые будут образовывать прямоугольник.



**3 Описание программы**

**3.1 Общие сведения**

Программа разработана в среде Visual Studio 2017

**3.2 Функциональное назначение**

Программа предназначена для вывода символов на экран.

* 1. **Описание логической структуры**

*Класс Point:*

В классе объявляются три поля, значение оси x, значение оси y, символ, который будет выводится.

Конструктор Point.

Метод Draw служит для вывода символов на экран.

*Класс Figure:*

Создается список pList для записи координат точек

Метод Draw служит для вывода символов на экран используя список pList.

*Класс VerticalLine:*

Этот класс является наследником класса Figure.

Метод VerticalLine принимает значения смещения по оси y, координату по оси x и символ.

Создается экземпляр класса List<Point> (списка).

В цикл передаются значения начальной и конечной точки по оси y.

В цикле создается экземпляр класса Point и в список добавляются координаты точек по осям x и y, и необходимый символ.

*Класс HorizontalLine:*

Этот класс является наследником класса Figure.

Метод HorizontalLine принимает значения смещения по оси x, координату по оси y и символ.

Создается экземпляр класса List<Point> (списка).

В цикл передаются значения начальной и конечной точки по оси x.

В цикле создается экземпляр класса Point и в список добавляются координаты точек по осям x и y, и необходимый символ.

*Класс Program:*

Метод Main (основной метод) в нем создается экземпляры классов VerticalLine и HorizontalLine, передаются данные в эти классы, вызывается метод Draw, для вывода каждой прямой.

**3 Описание применения**

После запуска программы выводятся символы в заданной последовательности образуя прямоугольник.

**Выводы**

Познакомились с созданием конструкторовдля классов. Изучили механизмы наследования и полиморфизма. Познакомились с управляющими операторами языка C#.

**Контрольные вопросы**

1. **Что такое конструктор?**

Конструктор класса — специальный блок инструкций, вызываемый при создании объекта.

1. **Зачем нужен конструктор?**

Одна из ключевых особенностей ООП — инкапсуляция: внутренние поля объекта напрямую недоступны, и пользователь может работать с объектом только как с единым целым, через открытые (public) методы. Каждый метод, в идеале, должен быть устроен так, чтобы объект, находящийся в «допустимом» состоянии (то есть когда выполняется инвариант класса), после вызова метода также оказался в допустимом состоянии. И первая задача конструктора — перевести поля объекта в такое состояние.

Вторая задача — упростить пользование объектом. Объект — не «вещь в себе», ему часто приходится требовать какую-то информацию от других объектов: например, объект File, создаваясь, должен получить имя файла. Это можно сделать и через метод:

1. **Что такое наследование?**

Наследование — концепция объектно-ориентированного программирования, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения.

Наследование является механизмом повторного использования кода и способствует независимому расширению программного обеспечения через открытые классы и интерфейсы.

1. **Что такое методы и поля класса?**

По́ле кла́сса в объектно-ориентированном программировании — переменная, связанная с классом или объектом. Все данные объекта хранятся в его полях. Доступ к полям осуществляется по их имени. Обычно тип данных каждого поля задаётся в описании класса, членом которого является поле.

Класс— это сложный (структурированный, составной) тип данных, объединяющий переменные, которые называют полями класса, и функции для работы с этими полями, которые называют *методами* класса.

1. **Что такое модификаторы доступа и для чего они нужны?**

Все члены класса - поля, методы, свойства - все они имеют модификаторы доступа. Модификаторы доступа позволяют задать допустимую область видимости для членов класса. То есть контекст, в котором можно употреблять данную переменную или метод. В предыдущей теме мы уже с ними сталкивались, когда объявляли поля класса Book публичными (то есть с модификатором public).

В C# применяются следующие модификаторы доступа:

* public: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.
* private: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.
* protected: такой член класса доступен из любого места в текущем классе или в производных классах. При этом производные классы могут располагаться в других сборках.
* internal: класс и члены класса с подобным модификатором доступны из любого места кода в той же сборке, однако он недоступен для других программ и сборок (как в случае с модификатором public).
* protected internal: совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов.
* private protected: такой член класса доступен из любого места в текущем классе или в производных классах, которые определены в той же сборке.

Объявление полей класса без модификатора доступа равнозначно их объявлению с модификатором private. Классы, объявленные без модификатора, по умолчанию имеют доступ internal.

ТЕКСТ ПРОГРАММЫ

Приложение А

(обязательное)

namespace Lab4

{

class Point

{

int x;

int y;

char sym;

public Point(int x, int y, char sym)

{

this.x = x;

this.y = y;

this.sym = sym;

}

public void Draw()

{

Console.SetCursorPosition(x, y);

Console.Write(sym);

}

}

class Figure

{

protected List<Point> pList;

public void Draw()

{

foreach (Point p in pList)

{

p.Draw();

}

}

}

class VerticalLine : Figure

{

public VerticalLine(int yUp, int yDown, int x, char sym)

{

pList = new List<Point>();

for (int y = yUp; y <= yDown; y++)

{

Point p = new Point(x, y, sym);

pList.Add(p);

}

}

}

class HorizontalLine : Figure

{

public HorizontalLine(int xLeft, int xRight, int y, char sym)

{

pList = new List<Point>();

for (int x = xLeft; x <= xRight; x++)

{

Point p = new Point(x, y, sym);

pList.Add(p);

}

}

}

class Program

{

static void Main(string[] args)

{

VerticalLine v1 = new VerticalLine(3, 4, 1, '\*');

v1.Draw();

HorizontalLine h1 = new HorizontalLine(1, 10, 2, '\*');

h1.Draw();

//VerticalLine v1 = new VerticalLine(3, 4, 1, '\*');

//v1.Draw();

VerticalLine v2 = new VerticalLine(3, 4, 10, '\*');

v2.Draw();

HorizontalLine h2 = new HorizontalLine(1, 10, 5, '\*');

h2.Draw();

Console.ReadKey();

}

}

}

РЕЗУЛЬТАТЫ ИСПЫТАНИЙ

Приложение Б

(обязательное)

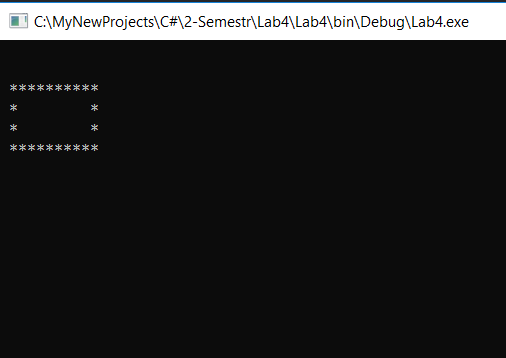


Рисунок 1 – Результат выполнения программы