

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ МОЛДОВА

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ МОЛДОВЫ

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ, ИНФОРМАТИКИ И
МИКРОЭЛЕКТРОНИКИ

ДЕПАРТАМЕНТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И АВТОМАТИКИ

ОТЧЕТ

По лабораторной работе №1

по дисциплине: АРРОО

Тема: Принципы ООП

Выполнил: студент группы ТІ-154

Сокол М.

Проверил:

Ресари М.

Кишинев 2018

Цель: целью лабораторной является изучение и понимание принципов ООП. И выполнить реализацию всех 4 принципов: абстракция, полиморфизм, инкапсуляция, наследование.

Ход работы: выполнить реализацию всех 4 принципов:

1. абстракция,
2. полиморфизм,
3. инкапсуляция,
4. наследование.

Ссылка на репозиторий: <https://github.com/ghipermax/APPOO.git>

Теория:

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Абстракция — в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.

Инкапсуляция — свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента (что у него внутри?), а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: `public`.

Пользователь не может использовать закрытые данные и методы. Реализуется с помощью ключевых слов: `private`, `protected`, `internal`.

Наследование — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Полиморфизм — возможность объектов с одинаковой спецификацией иметь различную реализацию.

Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования.

Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Плюсы ООП:

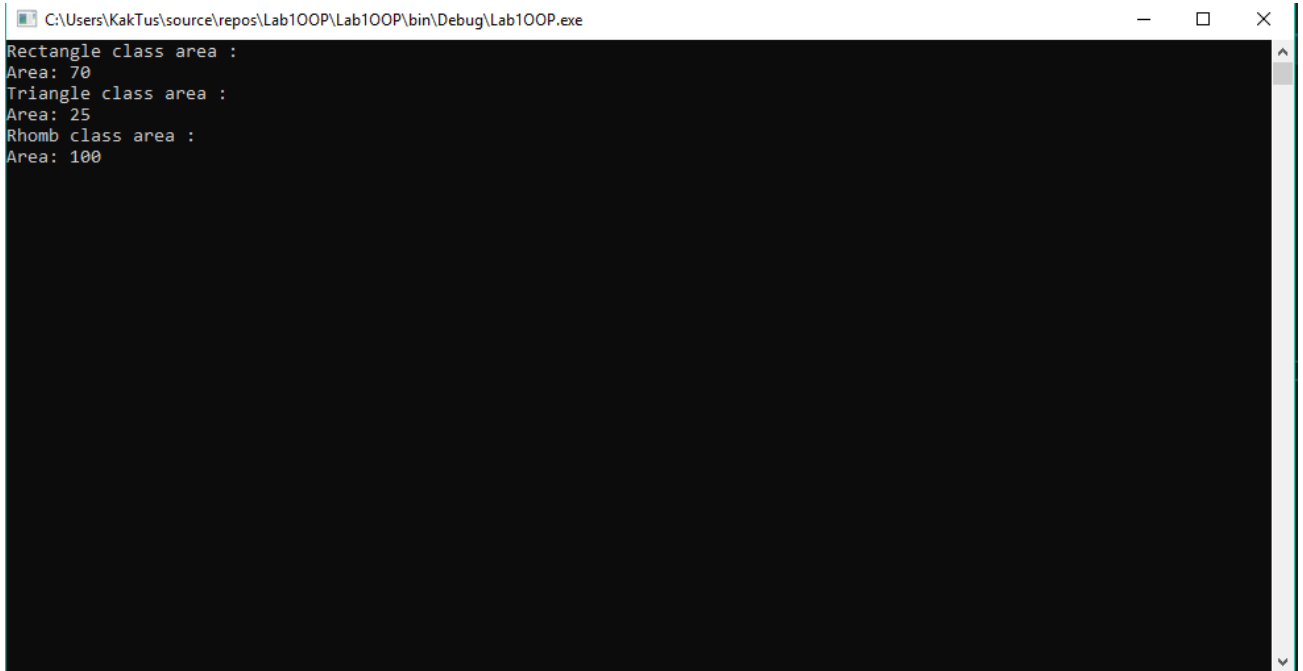
1. основным достоинством объектно-ориентированного программирования по сравнению с модульным программированием является «более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку.
2. объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения.
3. эти механизмы позволяют конструировать сложные объекты из сравнительно простых. В результате существенно увеличивается показатель повторного использования кодов и появляется возможность создания библиотек классов для различных применений.

Минусы ООП:

1. освоение базовых концепций ООП не требует значительных усилий. Однако разработка библиотек классов и их использование требуют существенных трудозатрат.
2. документирование классов — задача более трудная, чем это было в случае процедур и модулей.
3. в сложных иерархиях классов поля и методы обычно наследуются с разных уровней. И не всегда легко определить, какие поля и методы фактически относятся к данному классу.

4. основной недостаток ООП - некоторое снижение быстродействия за счет более сложной организации программной системы.

Результаты:



```
C:\Users\KakTus\source\repos\Lab1OOP\Lab1OOP\bin\Debug\Lab1OOP.exe
Rectangle class area :
Area: 70
Triangle class area :
Area: 25
Rhomb class area :
Area: 100
```

Рисунок 1: Результат выполнения.

Выводы: в этой лабораторной работе реализованы четыре основных принципа объектно-ориентированного программирования: инкапсуляция, абстракция, наследование и полиморфизм. В ходе выполнения лабораторной работы я выполнил 4 принципа в одном исходном коде. Тема, выбранная мною где я использовал все 4 принципа: “Вычисления площади фигуры”. ООП дает возможность создавать расширяемые системы (extensible systems). Это одно из самых значительных достоинств ООП и именно оно отличает данный подход от традиционных методов программирования. Расширяемость (extensibility) означает, что существующую систему можно заставить работать с новыми компонентами, причем без внесения в нее каких-либо изменений. Компоненты могут быть добавлены на этапе выполнения

Листинг программы:

Program.cs

```
class Program
{
    static void Main(string[] args)
    {
        Caller c = new Caller(); //создание объектов
        Rectangle r = new Rectangle(10, 7);
        Triangle t = new Triangle(10, 5);
        Rhomb rh = new Rhomb(20, 10);

        c.CallArea(r); //явное проявление полиморфизма
        c.CallArea(t);
        c.CallArea(rh);
        Console.ReadKey();
    }
}
```

Shape.cs

```
abstract class Shape { //абстрактный класс

    protected int width, height; //инкапсуляция - можем изменять данные в дочерних
    классах

    public Shape(int a, int b)
    {
        width = a;
        height = b;
    }
    public abstract int area(); //не нужно реализовать этот метод в базовом абстрактном
    классе
}
```

Rectangle.cs

```
class Rectangle : Shape { //наследование
    public Rectangle(int a, int b) : base(a, b) //Конструктор производного класса
    {

    }
    public override int area() //полиморфизм
    {
        Console.WriteLine("Rectangle class area :");
        return (width * height);
    }
}
```

Triangle.cs

```
class Triangle : Shape
{
    public Triangle(int a, int b) : base(a, b)
    {
    }
    public override int area()
    {
    }
}
```

```

        Console.WriteLine("Triangle class area :");
        return (width * height / 2);
    }
}

```

Rhomb.cs

```

class Rhomb : Shape
{
    public Rhomb(int a, int b) : base(a, b)
    {
    }
    public override int area()
    {
        Console.WriteLine("Rhomb class area :");
        return (width * height / 2);
    }
}

```

Caller.cs

```

class Caller
{
    public void CallArea(Shape sh) //полиморфизм
    {
        Console.WriteLine("Area: {0}", sh.area());
    }
}

```