**Лабораторная работа №7.**

**Полиморфизм. Виртуальные методы**

**Задание 1.**

Составить программу с одним родительским классом и двумя потомками. Потомки должны содержать виртуальные функции. Создать виртуальную функцию выдачи результатов расчета методов на экран монитора с

указанием названий и полей и их значений соответствующего объекта. Со-

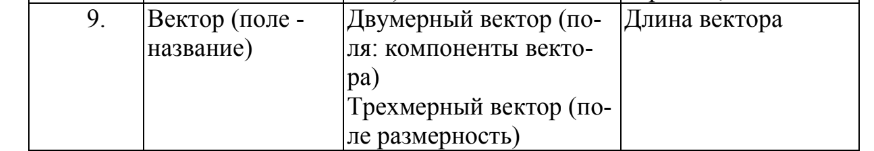
ставить тестирующую программу с выдачей протокола на экран монитора.

При этом создать объекты базового и производных типов, используя по-

лиморфный контейнер - массив ссылок базового класса на объекты базово-

го и производных классов (количество объектов >=5).





using System;

// Базовый класс (родительский класс)

class Vector

{

private string name; // Название вектора

// Конструктор с параметрами

public Vector(string name)

{

this.name = name;

}

// Виртуальный метод для расчета длины вектора

public virtual double CalculateLength()

{

return 0; // Базовый вектор имеет нулевую длину

}

// Метод для вывода информации о векторе

public void DisplayInfo()

{

Console.WriteLine($"Вектор \"{name}\":");

Console.WriteLine($"Длина вектора: {CalculateLength()}");

}

}

// Производный класс (потомок) для двумерных векторов

class TwoDimensionalVector : Vector

{

private double x; // Компонента x

private double y; // Компонента y

public TwoDimensionalVector(string name, double x, double y)

: base(name)

{

this.x = x;

this.y = y;

}

// Переопределение метода CalculateLength

public override double CalculateLength()

{

return Math.Sqrt(x \* x + y \* y);

}

}

// Производный класс (потомок) для трехмерных векторов

class ThreeDimensionalVector : Vector

{

private double x; // Компонента x

private double y; // Компонента y

private double z; // Компонента z

public ThreeDimensionalVector(string name, double x, double y, double z)

: base(name)

{

this.x = x;

this.y = y;

this.z = z;

}

// Переопределение метода CalculateLength

public override double CalculateLength()

{

return Math.Sqrt(x \* x + y \* y + z \* z);

}

}

class Program

{

static void Main()

{

// Создаем объекты базового и производных классов

Vector baseVector = new Vector("Базовый вектор");

TwoDimensionalVector twoDVector = new TwoDimensionalVector("2D вектор", 3, 4);

ThreeDimensionalVector threeDVector = new ThreeDimensionalVector("3D вектор", 1, 2, 3);

// Создаем массив ссылок на базовый класс

Vector[] vectors = { baseVector, twoDVector, threeDVector };

// Выводим информацию о векторах

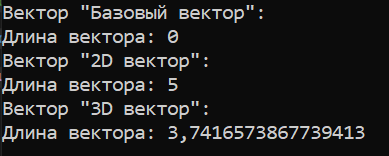
foreach (var vector in vectors)

{

vector.DisplayInfo();

}

}

}

Задание 2. Составить программу с абстрактным родительским классом и двумя объектами - потомками. Для этого модифицировать задание 1. Составить тестирующую программу с выдачей протокола на экран монитора. В ней нужно реализовать циклический вывод параметров объектов, используя полиморфный контейнер - массив объектов базового класса (количество объектов >=5).

using System;

// Базовый класс (родительский класс)

class Vector

{

protected string name; // Название вектора

// Конструктор с параметрами

public Vector(string name)

{

this.name = name;

}

// Виртуальный метод для расчета длины вектора

public virtual double CalculateLength()

{

return 0; // Базовый вектор имеет нулевую длину

}

// Метод для вывода информации о векторе

public void DisplayInfo()

{

Console.WriteLine($"Вектор \"{name}\":");

Console.WriteLine($"Длина вектора: {CalculateLength()}");

}

}

// Производный класс (потомок) для двумерных векторов

class TwoDimensionalVector : Vector

{

private double x; // Компонента x

private double y; // Компонента y

public TwoDimensionalVector(string name, double x, double y)

: base(name)

{

this.x = x;

this.y = y;

}

// Переопределение метода CalculateLength

public override double CalculateLength()

{

return Math.Sqrt(x \* x + y \* y);

}

}

// Производный класс (потомок) для трехмерных векторов

class ThreeDimensionalVector : Vector

{

private double x; // Компонента x

private double y; // Компонента y

private double z; // Компонента z

public ThreeDimensionalVector(string name, double x, double y, double z)

: base(name)

{

this.x = x;

this.y = y;

this.z = z;

}

// Переопределение метода CalculateLength

public override double CalculateLength()

{

return Math.Sqrt(x \* x + y \* y + z \* z);

}

}

class Program

{

static void Main()

{

Vector[] vectors = new Vector[5]; // Создаем массив объектов базового класса

vectors[0] = new TwoDimensionalVector("2D вектор 1", 3, 4);

vectors[1] = new TwoDimensionalVector("2D вектор 2", 1, 7);

vectors[2] = new ThreeDimensionalVector("3D вектор 1", 1, 2, 3);

vectors[3] = new ThreeDimensionalVector("3D вектор 2", 4, 5, 6);

vectors[4] = new TwoDimensionalVector("2D вектор 3", 8, 2);

// Выводим информацию о векторах

foreach (var vector in vectors)

{

vector.DisplayInfo();

}

// Вычисляем суммарную длину векторов

double totalLength = 0;

foreach (var vector in vectors)

{

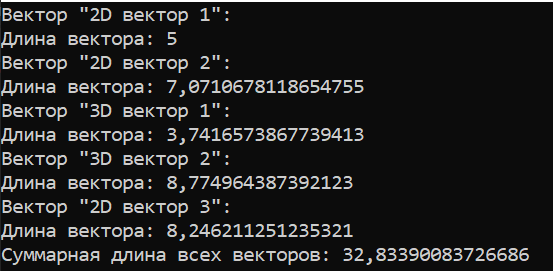
totalLength += vector.CalculateLength();

}

Console.WriteLine($"Суммарная длина всех векторов: {totalLength}");

}

}



Контрольные вопросы:

1. **Что означает принцип полиморфизма?**

Принцип полиморфизма в объектно-ориентированном программировании позволяет разным сущностям выполнять одни и те же действия. Независимо от того, как эти сущности устроены внутри и чем они различаются, программист может использовать общий код для работы с ними. Примером полиморфизма может быть метод “Отправить”, который может применяться как к картинкам, так и к видео1.

**2. Для чего используется позднее связывание?**

Позднее связывание — это механизм, при котором вызов метода происходит на основе типа объекта во время выполнения программы. Если в иерархии классов есть цепочка виртуальных методов (с помощью ключевых слов virtual и override), то компилятор использует позднее связывание. Это позволяет реализовать полиморфизм2.

1. **В каких случаях используются виртуальные методы?**

Виртуальные методы — это методы класса, которые могут быть переопределены в классах-наследниках. Они позволяют обеспечить разную реализацию метода для разных типов объектов, не зависимо от точного типа ссылки на объект3.

1. **Какие условия необходимо соблюдать при переопределении виртуального метода?**

При переопределении виртуального метода необходимо соблюдать следующие условия:

* Метод в базовом классе должен быть объявлен как виртуальный с помощью ключевого слова virtual.
* Метод в производном классе должен быть объявлен с помощью ключевого слова override.
* Сигнатура (параметры и возвращаемый тип) переопределенного метода должна совпадать с сигнатурой базового метода.

1. **Что представляют собой абстрактные классы? Для чего они предназначены?**

Абстрактные классы — это базовые классы, которые не предполагают создания экземпляров. Они могут содержать абстрактные методы, которые должны быть реализованы в классах-наследниках. Абстрактные классы предоставляют общий функционал для классов-потомков4.

1. **Могут ли в абстрактном классе быть неабстрактные методы?**

В абстрактном классе могут быть как абстрактные, так и неабстрактные методы. Неабстрактные методы могут содержать реализацию, а абстрактные — только декларацию без реализации. Абстрактные классы предоставляют базовый функционал, который может быть переопределен в классах-наследниках