# Лабораторная работа 4. Наследование.

## Задание.

Выделить в предметной области 2-3 варианта сущности (например, "Геометрическая фигура", "Точка", "Треугольник"), отличающиеся несколькими полями и методами. Каждый класс имеет поля, свойства и методы.

Спроектировать UML-диаграммы классов.

**Базовый** класс для вашей иерархии объявите **абстрактным**. Он должен содержать **абстрактные методы** и **методы с реализацией**.

Один из наследников должен **перегружать (скрывать)** метод родителя.

Один из классов должен содержать **виртуальный метод**, который **переопределяется** в одном наследнике и **не переопределяется** в другом.

Продемонстрировать работу всех объявленных методов.

Продемонстрировать **вызов конструктора родительского класса** при наследовании.

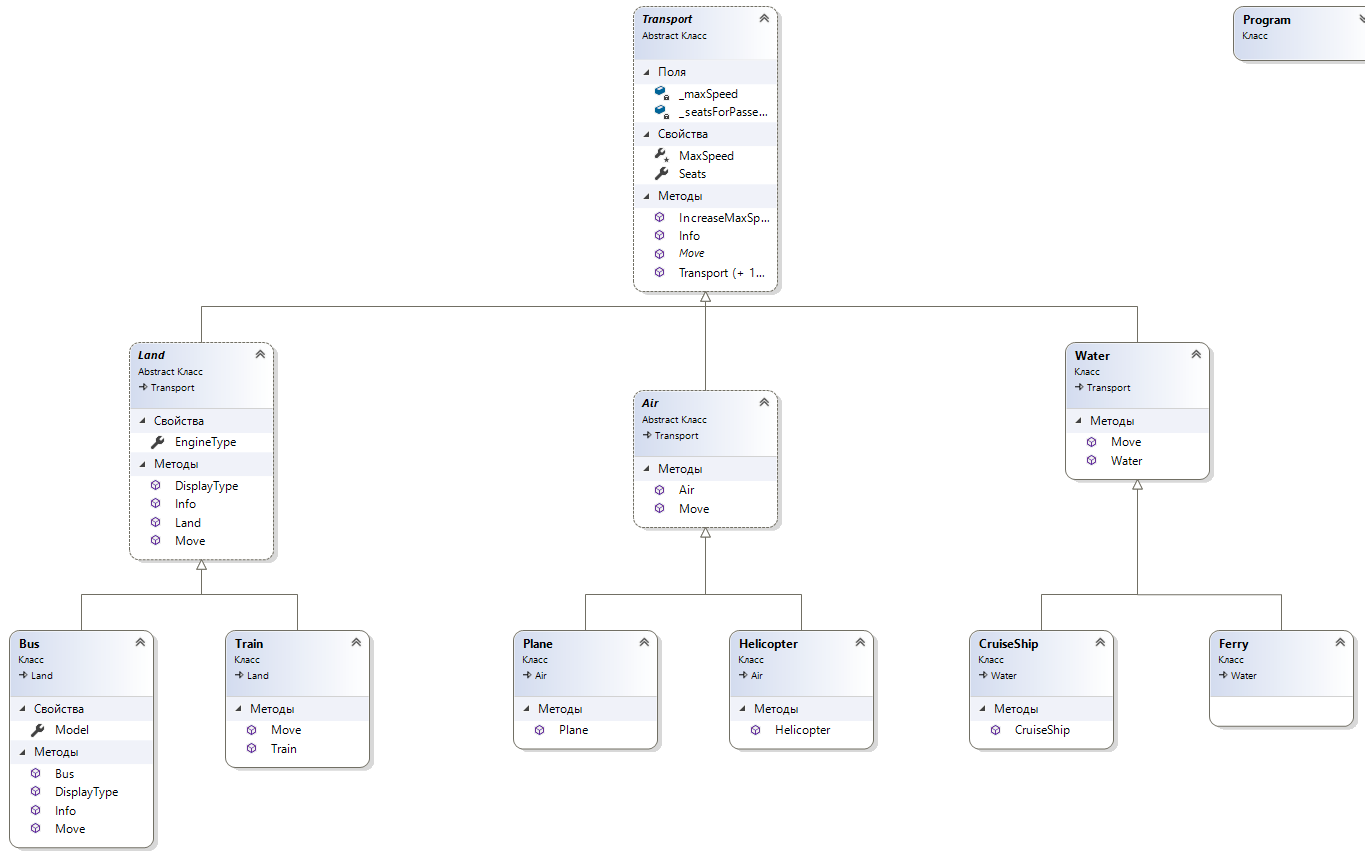
Продемонстрировать вызов метода родительского класса при его **скрытии**.

Создать класс, **закрытый для наследования**.

Примеры методов: вывод полей класса, изменение числовых полей класса (скорости, дохода, численности штата и т.д.), изменение текстовых полей класса (переименование), какая-либо динамика (полет, ремонт, переезд, открытие/закрытие и.д.)

**Номер варианта выбирается по желанию!**

1. Геометрические фигуры
2. Здания
3. Транспорт
4. Автомобили
5. Товары
6. Игровые юниты
7. Компьютерная техника
8. Программное обеспечение
9. Игрушки
10. Налоги
11. Банковские счета
12. Врачи
13. Лекарства
14. Мебель
15. Фильмы
16. Одежда
17. Техника
18. Еда
19. Алкогольные напитки
20. Коктейли
21. Фрукты или овощи
22. Десерты
23. Космические объекты
24. Спорт
25. Животные
26. Растения
27. Научные дисциплины



public abstract class Transport

{

private int \_maxSpeed; // скорость

private int \_seatsForPassenger; // количество пассажирских мест

protected int MaxSpeed { get => \_maxSpeed; set => \_maxSpeed = value; }

public int Seats { get => \_seatsForPassenger; set => \_seatsForPassenger = value; }

public Transport() {

MaxSpeed = 0;

Seats = 0;

}

public Transport(int maxspeed, int seats)

{

MaxSpeed = maxspeed;

Seats = seats;

}

abstract public void Move(); // абстрактный метод

public virtual string Info() // метод с реализацией который может быть перегружен в дочерних классах

{

return ($"максимальная скорость:{MaxSpeed}; кол-во сидений для пассажиров:{Seats}");

}

public void IncreaseMaxSpeed(int sp)

{

MaxSpeed += sp;

}

}

abstract public class Land : Transport

{

public string EngineType { get; set; }

public Land(int maxspeed, int seats, string engineType) : base(maxspeed, seats)

{

EngineType = engineType;

}

public override void Move()

{

Console.WriteLine("Я еду по земле.");

}

public void DisplayType()

{

Console.WriteLine("Я являюсь наземным транспортом");

}

public override string Info() // переопределение метода абстрактного родительского класса Transport

{

return $"тип двигателя:{EngineType};\n" + base.Info();

}

}

public class Bus : Land

{

public string Model { get; set; }

public Bus(string model, int maxspeed, int passenger, string typeEngine) : base(maxspeed, passenger, typeEngine)

{

Model = model;

}

public override void Move() // реализация абстрактного метода базового класса Transport

{

Console.WriteLine("Я еду по автодороге");

}

public new void DisplayType() // скрываем метод родительского класса Land

{

Console.WriteLine($"Я являюсь автобусом {Model}");

}

public override string Info() // переопределение метода абстрактного родительского класса Bus

{

return $"Название модели:{Model};" + base.Info();

}

}

public class Train : Land

{

public Train(int maxspeed, int passengers, string typeEngine) : base(maxspeed, passengers, typeEngine) { }

public override void Move()

{

Console.WriteLine("Я еду по железной дороге");

}

}

abstract public class Air : Transport

{

public Air( int maxspeed, int passengers): base(maxspeed, passengers) { }

public override sealed void Move() // перегружаем метод и закрываем его для дальнейшего наследования

{

Console.WriteLine("Я лечу над землей");

}

}

public class Helicopter : Air

{

public Helicopter(int maxspeed, int passengers) : base(maxspeed, passengers) { }

// public override void Move() { }; Нельзя перегрузить метод Move т.к. он "запечатан"

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Лайнер:");

Transport titanik = new CruiseShip(); // создаем новый объект транспорта, который использует конструктор базового класса без параметров

titanik.Move();

Console.WriteLine("Информация о titanik - " + titanik.Info());

Console.WriteLine("\nВертолет:");

Helicopter mi5 = new Helicopter(300, 15);

mi5.Move();

Console.WriteLine("Информация о mi5 - " + mi5.Info());

Console.WriteLine("\nПоезд:");

Land shtadler = new Train(170, 500, "электрический");

shtadler.Move();

Console.WriteLine("Информация о shtadler - " + shtadler.Info());

shtadler.DisplayType();

Console.WriteLine("\nАвтобус:");

Transport maz\_base = new Bus("МАЗ 2257", 90, 48, "дизельный");

maz\_base.Move();

Console.WriteLine("Информация о maz\_base - " + maz\_base.Info() + "\n"); //не смотря на то, что maz\_base хранится в переменной типа Transport, для него будет вызываться перегруженый метод класса Bus!

Land maz = (Land)maz\_base;

maz.DisplayType(); // => наземный транспорт; вызывается DisplayType() для класса Land

maz.Move();

Console.WriteLine("Информация о maz - " + maz.Info() + "\n");

Bus maz\_new = (Bus)maz;

maz\_new.DisplayType(); // => Model:МАЗ 2257; вызывается DisplayType() для класса Bus, т.к. родительский метод скрыт

maz\_new.IncreaseMaxSpeed(55);

maz\_new.Move();

Console.WriteLine("Информация о maz\_new - " + maz\_new.Info() + "\n");

Console.ReadKey();

}

}

