Тема 4 «Объектно-ориентированное программирование. Продолжение»

АБСТРАКТНЫЕ КЛАССЫ И ЧЛЕНЫ КЛАССОВ
АБСТРАКТНЫЕ КЛАССЫ

Кроме обычных классов в C# есть абстрактные классы.

При определении абстрактных классов используется ключевое слово **abstract**

АБСТРАКТНЫЕ КЛАССЫ И ЧЛЕНЫ КЛАССОВ
АБСТРАКТНЫЕ КЛАССЫ

Кроме обычных классов в C# есть абстрактные классы.

При определении абстрактных классов используется ключевое слово **abstract**

```
abstract class Transport
{
    public void Move()
    {
        Console.WriteLine("Транспортно средство движется");
    }
}
```

АБСТРАКТНЫЕ КЛАССЫ И ЧЛЕНЫ КЛАССОВ
АБСТРАКТНЫЕ КЛАССЫ

Главное отличие абстрактных классов от обычных состоит в том, что мы **НЕ можем** использовать **конструктор** абстрактного класса для создания экземпляра класса. Например, следующим образом:

```
Transport tesla = new Transport();
```

АБСТРАКТНЫЕ КЛАССЫ И ЧЛЕНЫ КЛАССОВ
АБСТРАКТНЫЕ КЛАССЫ

```
car.Move();
ship.Move();
aircraft.Move();
abstract class Transport
{
    public void Move()
    {
        Console.WriteLine("Транспортное средство движется");
    }
}
// класс корабля
class Ship : Transport { }
// класс самолета
class Aircraft : Transport { }
// класс машины
class Car : Transport { }
```

АБСТРАКТНЫЕ ЧЛЕНЫ КЛАССОВ

В частности, абстрактными могут быть:

- Методы
- Свойства
- Индексаторы
- События

АБСТРАКТНЫЕ ЧЛЕНЫ КЛАССОВ

Абстрактные члены классов не должны иметь модификатор **private**

ОБОБЩЕНИЯ

Кроме обычных типов фреймворк .NET также поддерживает обобщенные типы (**generics**), а также создание обобщенных методов.

```
class Person
{
    public int Id { get;}
    public string Name { get;}
    public Person(int id, string name)
    {
        Id = id;
        Name = name;
    }
}
```

ОБОБЩЕНИЯ

Для решения этих проблем в язык C# была добавлена **поддержка обобщенных типов** (универсальными типами). Обобщенные типы позволяют указать конкретный тип, который будет использоваться. Поэтому определим класс Person как обощенный:

```
class Person<T>
{
    public T Id { get; set; }
    public string Name { get; set; }
    public Person(T id, string name)
    {
        Id = id;
        Name = name;
    }
}
```

ОБОБЩЕНИЯ

```
Person<int> tom = new Person<int>(546, "Tom");  // упаковка не нужна
Person<string> bob = new Person<string>("abc123", "Bob");

int tomId = tom.Id;       // распаковка не нужна
string bobId = bob.Id;  // преобразование типов не нужно

Console.WriteLine(tomId);   // 546
Console.WriteLine(bobId);   // abc123
```

ОБОБЩЕНИЯ

Использование нескольких универсальных параметров

```
class Person<T, K>
{
    public T Id { get;}
    public K Password { get; set; }
    public string Name { get;}
    public Person(T id, K password, string name)
    {
        Id = id;
        Name = name;
        Password = password;
    }
}
```

ОБОБЩЕНИЯ

Обобщенные методы

```
int x = 7;
int y = 25;
Swap<int>(ref x, ref y); // или так Swap(ref x, ref y);
Console.WriteLine($"x={x}    y={y}");   // x=25   y=7

string s1 = "hello";
string s2 = "bye";
Swap<string>(ref s1, ref s2); // или так Swap(ref s1, ref s2);
Console.WriteLine($"s1={s1}    s2={s2}"); // s1=bye   s2=hello

void Swap<T>(ref T x, ref T y)
{
    T temp = x;
    x = y;
    y = temp;
}
```