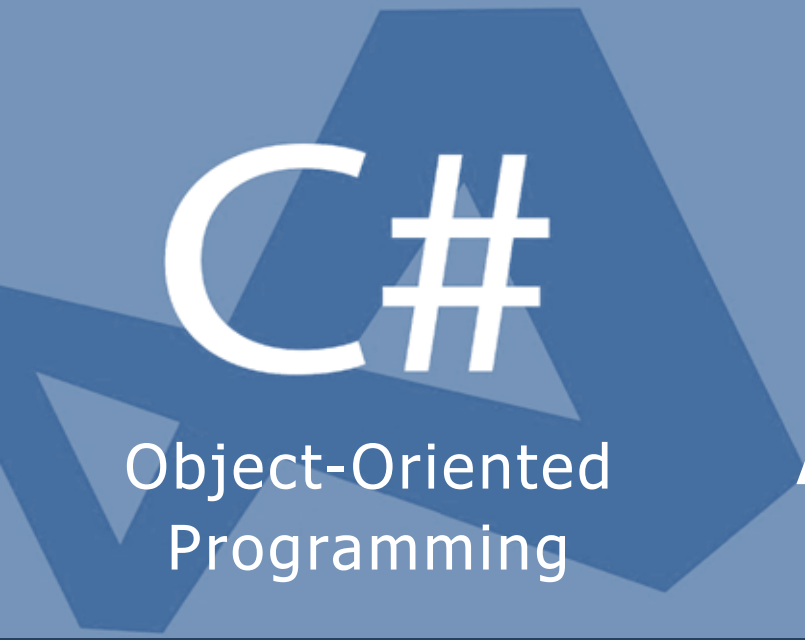


Object-Oriented Programming

3

Abstract class vs Interface

д.т.н. Емельянов Виталий Александрович

: v.yemelyanov@gmail.com

Виды наследования

От простого класса

От абстрактного класса

Реализация интерфейса

Абстрактные классы

Абстрактный класс:

- используется для создания производных классов;
- базовый класс, который не предполагает создания экземпляров.

Назначение:

Абстрактные классы призваны предоставлять базовый функционал (общие поля и методы) для классов-наследников. А производные классы уже реализуют этот функционал.

При определении абстрактных классов используется ключевое слово `abstract`:

C#

```
1  abstract class Employee
2  {
3  }
```

Абстрактные классы

Кроме обычных методов абстрактный класс может содержать **абстрактные методы**. Такие методы определяются с помощью ключевого слова `abstract` и не имеют никакого функционала:

C#

```
1
2   abstract class Employee
3   {
4     private String name;
5
6     public String getName()    //обычный метод
7     {
8         return name;           //реализация функционала обычного метода
9     }
10
11    public abstract void GiveBonus(float amount);  //абстрактный метод
12  }
13
```

Абстрактные классы

Особенности:

- производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе;
- если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный.
- производный класс в котором не будет замещен абстрактный метод сам считается абстрактным, и мы не сможем создавать объекты этого класса.

Ограничения наследования от классов

В современных языках программирования (Java, C#, PHP и др.) **наследовать можно только от одного класса**, в отличие, например, от языка C++, где имеется множественное наследование.

Интерфейсы

Интерфейс (interface) – именованный набор абстрактных членов

Назначение:

- **Интерфейс описывает поведение**, которое конкретный класс может выбрать для реализации.
- Класс может при необходимости поддерживать много интерфейсов, тем самым поддерживая множество стилей поведения.

Чтобы определить интерфейс, используется ключевое слово `interface`:

C#

```
1  interface IMovable
2  {
3  }
```

Интерфейсы

Особенности интерфейса:

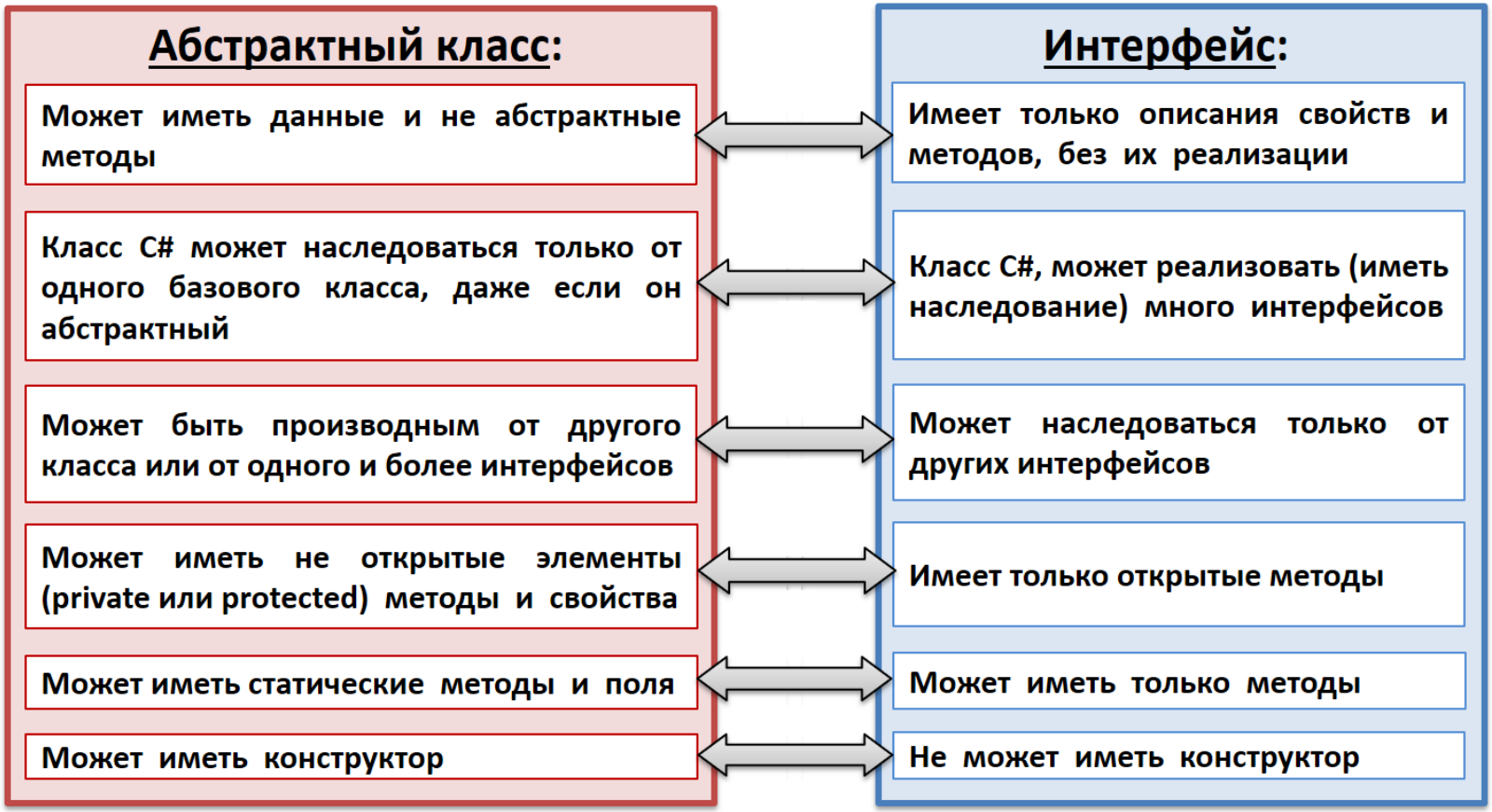
- Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ `public`, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

Интерфейсы

C#

```
1  interface IMovable
2  {
3      void Move();
4  }
5
6  class Animal : IMovable
7  {
8      public void Move()
9      {
10         //код для реализации передвижения животного
11     }
12 }
13
14 class Human : IMovable
15 {
16     public void Move()
17     {
18         //код для реализации передвижения человека
19     }
20 }
```

Различия абстрактного класса и интерфейса



Когда следует использовать абстрактные классы?

- **Если необходимо определить общий функционал (поведение) для родственных объектов**
- **Если** проектируем довольно большую функциональную единицу, которая содержит много базового функционала
- **Если** необходимо, чтобы все производные классы на всех уровнях наследования имели некоторую общую реализацию

С абстрактными классами, если надо изменить базовый функционал во всех наследниках, то достаточно поменять его в абстрактном базовом классе.

Если надо будет поменять название или параметры метода интерфейса, то придется вносить изменения и во все классы, которые данный интерфейс реализуют.

Когда следует использовать интерфейсы?

- **Если необходимо определить функционал для группы разрозненных объектов, которые могут быть никак не связаны между собой**

- **Если** проектируем небольшой функциональный тип

Пример использования абстрактных классов

Задача: **Пусть есть система транспортных средств: легковой автомобиль, автобус, трамвай, поезд и т.д. Поскольку данные объекты являются родственными, мы можем выделить у них общие признаки, то в данном случае можно использовать абстрактные классы:**

C#

```
1  public abstract class Vehicle
2  {
3    public abstract void Move();
4  }
5
6
7  public class Car : Vehicle
8  {
9    public override void Move()
10   {
11      Console.WriteLine("Машина едет");
12   }
13 }
14
15
```

```
16
17 public class Bus : Vehicle
18 {
19    public override void Move()
20    {
21       Console.WriteLine("Автобус едет");
22    }
23 }
24
25 public class Tram : Vehicle
26 {
27    public override void Move()
28    {
29       Console.WriteLine("Трамвай едет");
30    }
31 }
```

Пример использования интерфейса

Задача: Предположим, что система транспорта не ограничивается вышеперечисленными транспортными средствами. Например, можно добавить самолеты, лодки. Можно добавить лошадь - животное, которое может также выполнять роль транспортного средства. Т.е. получается широкий круг объектов, которые связаны только тем, что являются транспортным средством и должны реализовать метод `Move()`, выполняющий перемещение.

Решение: **Так как объекты малосвязанные между собой, то для определения общего для всех них функционала лучше определить интерфейс**. Тем более некоторые из этих объектов могут существовать в рамках параллельных систем классификаций. Например, лошадь может быть классом в структуре системы классов животного мира.

Пример использования интерфейса

C#

```
1  public interface IMovable
2  {
3      void Move();
4  }
5
6
7  public abstract class Vehicle
8  { }
9
10
11 public class Car : Vehicle, IMovable
12 {
13   public void Move()
14   {
15     Console.WriteLine("Машина едет");
16   }
17 }
18
19
20
21
22
```

```
23 public class Bus : Vehicle, IMovable
24 {
25    public void Move()
26    {
27       Console.WriteLine("Автобус едет");
28    }
29 }
30
31 public class Hourse : IMovable
32 {
33    public void Move()
34    {
35       Console.WriteLine("Лошадь скачет");
36    }
37 }
38
39 public class Aircraft : IMovable
40 {
41    public void Move()
42    {
43       Console.WriteLine("Самолет летит");
44    }
45 }
```

Заключение

- **абстрактные классы фокусируются на общем состоянии классов-наследников**, т.е. **для одноплановых классов**, которые имеют общее состояние, лучше определять **абстрактный класс**.

- **интерфейсы строятся вокруг какого-либо общего действия**. Если **разноплановые классы** обладают каким-то общим действием, то это действие лучше выносить в **интерфейс.**