Что такое интерфейсы? Чем интерфейсы отличаются от классов?

Интерфейс - ссылочный тип данных, который содержит в себе методы и свойства, но не реализует их. По умолчанию имеет доступ public, может быть internal.

Интерфейс не может иметь полей, только методы.  
Интерфейс может наследоваться только от интерфейса.  
У Интерфейса члены не могут иметь модификаторов доступа, они по умолчанию открытые.  
  
Абстрактный Класс может наследоваться от множества интерфейсов, но только от одного  
класса. Интерфейс от класса наследоваться не может.  
  
Самое главное:   
Абстрактные классы применяются когда необходимо установить какое-то поведение для всей иерархии по умолчанию.  
А интерфейс применяется когда нужно расширить поведение класса.

Как осуществляется описание интерфейса?

interface IMovable

{

    void Move();

}

Как осуществляется описание членов интерфейса?

interface имя{

возвращаемый\_тип имя\_метода\_1 (список\_параметров);

возвращаемый\_тип имя\_метода\_2 (список\_параметров);

// ...

возвращаемый\_тип имя\_метода\_N (список\_параметров);

}

где имя — это конкретное имя интерфейса. В объявлении методов интерфейса используются только их возвращаемый\_тип и сигнатура. Они, по существу, являются абстрактными методами. Как пояснялось выше, в интерфейсе не может быть никакой реализации. Поэтому все методы интерфейса должны быть реализованы в каждом классе, включающем в себя этот интерфейс. В самом же интерфейсе методы неявно считаются открытыми, поэтому доступ к ним не нужно указывать явно.

Как сделать метод интерфейса методом по умолчанию?

Начиная с версии C# 8.0 интерфейсы поддерживают реализацию методов и свойств по умолчанию. Это значит, что мы можем определить в интерфейсах полноценные методы и свойства, которые имеют реализацию как в обычных классах и структурах. Например, определим реализацию метода Move по умолчанию:

interface IMovable

{

    // реализация метода по умолчанию

    void Move()

    {

        Console.WriteLine("Walking");

    }

}

С реализацией свойств по умолчанию в интерфейсах дело обстоит несколько сложнее, поскольку мы не можем определять в интерфейсах нестатические переменные, соответственно в свойствах интерфейса мы не можем манипулировать состоянием объекта. Тем не менее реализацию по умолчанию для свойств мы тоже можем определять:

|  |  |
| --- | --- |
|  | interface IMovable  {      void Move()      {          Console.WriteLine("Walking");      }      // реализация свойства по умолчанию      // свойство только для чтения      int MaxSpeed { get { return 0; } }  } |

Стоит отметить, что если интерфейс имеет приватные методы и свойства (то есть с модификатором private), то они должны иметь реализацию по умолчанию. То же самое относится к любым статическим методам и свойствам (не обязательно приватным):

Ранее до JDK 8 при реализации интерфейса мы должны были обязательно реализовать все его методы в классе. А сам интерфейс мог содерать только определения методов без конкретной реализации. В JDK 8 была добавлена такая функциональность как **методы по умолчанию**. И теперь интерфейсы кроме определения методов могут иметь их реализацию по умолчанию, которая используется, если класс, реализующий данный интерфейс, не реализует метод

Что нужно сделать для того чтобы в классе можно было бы реализовать больше одного интерфейса?

Класс может реализовать несколько интерфейсов. В этом случае все интерфейсы определяются списком через запятую.

Общая форма класса реализующего несколько интерфейсов:

class имя\_класса : имя\_интерфейса1, имя\_интерфейса2, ..., имя\_интерфейсаN

{

// тело класса

...

}

Как определить метод интерфейса, который может быть использован только внутри него самого

По умолчанию все методы в интерфейсе фактически имеют модификатор public. Однако начиная с Java 9 мы также можем определять в интерфейсе методы с модификатором **private**. Они могут быть статическими и нестатическими, но они не могут иметь реализации по умолчанию.

Подобные методы могут использоваться только внутри самого интерфейса, в котором они определены. То есть к примеру нам надо выполнять в интерфейсе некоторые повторяющиеся действия, и в этом случае такие действия можно выделить в приватные методы:

Как определить метод интерфейса, который может быть использован только внутри него самого?

Однако в отличие от классов мы не можем применять к интерфейсам модификатор **sealed**, чтобы запретить наследование интерфейсов.

Также мы не можем применять к интерфейсам модификатор **abstract**, поскольку интерфейс фактически итак, как правило, предоставляет абстрактный функционал, который должен быть реализован в классе или структуре (за исключением методов и свойств с реализацией по умолчанию).

interface IAction

{

    void Move();

}

interface IRunAction : IAction

{

    void Run();

}

class BaseAction : IRunAction

{

    public void Move()

    {

        Console.WriteLine("Move");

    }

    public void Run()

    {

        Console.WriteLine("Run");

    }

}

Каким образом интерфейсы используются в механизме обратного вызова?

Одним из распространенных способов использования интерфейсов в Java является создание обратного вызова. Суть обратного вызова состоит в том, что мы создаем действия, которые вызываются при других действиях. То есть одни действия вызываются другими действиями. Стандартный пример - нажатие на кнопку. Когда мы нажимаем на кнопку, мы производим действие, но в ответ на это нажатие запускаются другие действия. Например, нажатие на значок принтера запускает печать документа на принтере и т.д.

**1. Назначение интерфейсов. Особенности применения интерфейсов в C#**

Интерфейс определяет ряд методов (свойств, индексаторов, событий), которые должны быть реализованы в классе, который наследует (реализует) данный интерфейс. Интерфейсы используются для того, чтобы указать классам, что именно нужно реализовать в этих классах. Реализовывать нужно методы (свойства, индексаторы, события). Таким образом, интерфейс описывает функциональные возможности без конкретной реализации. Иными словами интерфейс определяет спецификацию но не реализацию.

Использование интерфейсов есть эффективным в случаях, когда нужно создать альтернативу множественного наследования. Любой класс может унаследовать несколько интерфейсов. При этом все методы унаследованных интерфейсов должны быть реализованы в классе.

Структура также как и класс может реализовывать любое количество интерфейсов.

**Особенности интерфейсов**

* в интерфейсе нельзя вписывать реализацию его элементов;
* невозможно создать экземпляр интерфейса;
* можно создать ссылку на интерфейс;
* в интерфейсе не может быть конструкторов;
* интерфейс не может содержать поля;
* в интерфейсе не может быть осуществлена перегрузка операторов;
* все методы интерфейса по умолчанию объявлены как public.

При использовании интерфейсов**в классах-наследниках**:

* запрещено изменять модификатор доступа для метода при его реализации;
* невозможно объявить методы интерфейса как virtual;
* запрещено объявлять методы интерфейса с ключевым словом static (как статические).

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**2. Какое отличие между интерфейсами и абстрактными классами?**

В языке программирования C# существуют следующие отличия между интерфейсами и абстрактными классами:

1. В интерфейсе запрещено прописывать реализацию его членов. В абстрактном классе часть членов может иметь реализацию. Иными словами, интерфейс это тот же абстрактный класс, у которого все методы абстрактные.
2. В интерфейсе запрещено описывать поля (переменные, объекты), в абстрактном классе можно.
3. Интерфейс не может содержать конструктор. В абстрактном классе может быть объявлен конструктор.
4. Любой класс может быть унаследован от нескольких интерфейсов. При этом любой класс может быть унаследован только от одного абстрактного класса (и не более).

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**3. Сколько классов могут иметь реализацию методов интерфейса?**

Если интерфейс определен, то он может быть реализован в любом количестве классов.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**4. Сколько интерфейсов может быть реализовано в одном классе?**

В одном классе может быть реализовано любое количество интерфейсов.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**5. Какой общий вид описания интерфейса?**

Интерфейсы объявляются с помощью ключевого слова **interface**. Общая форма описания интерфейса, в котором определяются методы, следующая:

**interface *имя***

**{**

***возвращаемый\_тип1* *имя\_метода1*(*параметры1*);**

***возвращаемый\_тип2* *имя\_метода2*(*параметры2*);**

**// ...**

***возвращаемый\_типN* *имя\_методаN*(*параметрыN*);**

**}**

где

* *имя* – конкретное имя интерфейса;
* *имя\_метода1*, *имя\_метода2*, …, *имя\_методаN* – имена методов интерфейсов;
* *возвращаемый\_тип1*, *возвращаемый\_тип2*, …, *возвращаемый\_типN* – типы, которые возвращаются методами интерфейса;
* *параметры1*, *параметры2*, …, *параметрыN* – списки параметров методов интерфейса.

Кроме методов, в интерфейсах можно указывать свойства, события и индексаторы.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**6. Какие элементы языка программирования можно указывать в интерфейсах?**

В интерфейсах можно указывать:

* методы;
* свойства;
* индексаторы;
* события.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**7. Как выглядит общая форма реализации интерфейса в классе?**

Общая форма реализации интерфейса в классе имеет следующий вид:

**class *имя\_класса* : *имя\_интерфейса***

**{**

**// тело класса**

**...**

**}**

где *имя\_интерфейса* – имя интерфейса, методы (свойства, индексаторы, события) которого реализуются в классе. Класс обязательно должен реализовать все методы интерфейса.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**8. Какая общая форма класса реализующего несколько интерфейсов?**

Класс может реализовать несколько интерфейсов. В этом случае все интерфейсы определяются списком через запятую.

Общая форма класса реализующего несколько интерфейсов:

**class *имя\_класса* : *имя\_интерфейса1*, *имя\_интерфейса2*, ..., *имя\_интерфейсаN***

**{**

**// тело класса**

**...**

**}**

где *имя\_интерфейса1*, *имя\_интерфейса2*, …, *имя\_интерфейсаN* – имена интерфейсов, которые должен реализовать класс. Класс должен реализовать все методы всех интерфейсов.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**9. Пример объявления интерфейса и класса наследующего этот интерфейс**

В данном примере интерфейсу присваивается имя IMyInterface. Рекомендовано к имени интерфейса добавить префикс ‘**I**’ в соответствии с общераспространенной практикой.

Интерфейс объявлен как public.

**public interface IMyInterface**

**{**

**int MyGetInt(); // метод, возвращающий число типа int**

**double MyGetPi(); // метод, возвращающий число Pi**

**int MySquare(int x); // метод, возвращающий x в квадрате**

**double MySqrt(double x); // метод возвращающий корень квадратный из x**

**}**

В данном примере, в интерфейсе объявлено описание четырех методов, которые должны быть реализованы во всех классах, определяющих эти интерфейсы. Это методы: MyGetInt(), MyGetPi(), MySquare(), MySqrt().

Пример описания класса использующего этот интерфейс.

**public class MyClass : IMyInterface**

**{**

**// модификатор доступа public**

**public int MyGetInt()**

**{**

**return 25;**

**}**

**public double MyGetPi()**

**{**

**return Math.PI;**

**}**

**public int MySquare(int x)**

**{**

**return (int)(x \* x);**

**}**

**public double MySqrt(double x)**

**{**

**return (double)Math.Sqrt(x);**

**}**

**}**

Все методы, которые определяются в классе, должны иметь тип доступа public. Если установить другой тип доступа (private или protected), то Visual Studio выдаст следующее сообщение:

**"MyClass does not implement interface member MyFun() because it is not public."**

где MyFun() – название функции, которая реализована в классе с модификатором доступа private или protected.

Это связано с тем, что в самом интерфейсе эти методы неявно считаются открытыми (public). Поэтому их реализация должна быть открытой.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**10. Пример объявления двух интерфейсов и класса, который реализует методы этих интерфейсов**

В нижеследующем примере объявлено два интерфейса с именами MyInterface и MyInterface2. Первый интерфейс содержит 4 методы. Второй интерфейс содержит 1 метод.

Также объявлен класс MyClass, использующий эти два интерфейса. Класс обязательно должен реализовать все методы обоих интерфейсов, то есть в сумме 5 методов.

**public interface IMyInterface**

**{**

**int MyGetInt(); // метод, возвращающий число типа int**

**double MyGetPi(); // метод, возвращающий число Pi**

**int MySquare(int x); // метод, возвращающий x в квадрате**

**double MySqrt(double x); // метод, возвращающий корень квадратный из x**

**}**

**public interface IMyInterface2**

**{**

**double MySqrt2(double x); // корень квадратный из x**

**}**

**public class MyClass : IMyInterface, IMyInterface2**

**{**

**// методы из интерфейса MyInterface**

**public int MyGetInt()**

**{**

**return 25;**

**}**

**public double MyGetPi()**

**{**

**return Math.PI;**

**}**

**public int MySquare(int x)**

**{**

**return (int)(x \* x);**

**}**

**public double MySqrt(double x)**

**{**

**return (double)Math.Sqrt(x);**

**}**

**// метод из интерфейса MyInterface2**

**public double MySqrt2(double x)**

**{**

**return (double)Math.Sqrt(x);**

**}**

**}**

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**11. Пример использования ссылки на интерфейс для доступа к методам класса**

В C# допускается описывать ссылки на интерфейс. Если описать переменную-ссылку на интерфейс, то с ее помощью можно вызвать методы класса, который использует этот интерфейс.

**Пример.**

**public interface IMyInterface**

**{**

**double MyGetPi(); // метод, возвращающий число Pi**

**}**

**class MyClass : IMyInterface**

**{**

**// методы из интерфейса MyInterface**

**public double MyGetPi()**

**{**

**return Math.PI;**

**}**

**}**

**// вызов из программного кода**

**private void button1\_Click(object sender, EventArgs e)**

**{**

**MyClass mc = new MyClass(); // создание объекта класса mc**

**IMyInterface mi; // ссылка на интерфейс**

**double d;**

**mi = mc; // mi ссылается на объект класса mc**

**d = mi.MyGetPi(); // d = 3.14159265358979**

**label1.Text = d.ToString();**

**}**

В данном примере создается объект (экземпляр) класса MyClass с именем mc. Затем описывается ссылка на интерфейс IMyInterface с именем mi.

Строка

**mi=mc;**

приводит к тому, что ссылка mi указывает на объект класса mc. Таким образом, через ссылку mi можно иметь доступ к методам класса MyClass, так как класс MyClass реализует методы интерфейса IMyInterface.

С помощью ссылки на интерфейс можно иметь доступ к методам классов, которые реализуют описанные в этом интерфейсе методы.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**12. Каким образом в интерфейсе описывается свойство?**

Свойство описывается в интерфейсе без тела. Общая форма объявления интерфейсного свойства следующая:

***тип* *имя\_свойства***

**{**

**get;**

**set;**

**}**

Если свойство предназначено только для чтения, то используется один только аксессор get.

Если свойство предназначено для записи, то используется только один аксессор set.

**Пример.** Описывается интерфейс и класс. Класс возвращает свойство MyPi.

**public interface IMyInterface**

**{**

**double MyGetPi(); // метод, возвращающий число Pi**

**// свойство, возвращающее число Pi**

**double MyPi**

**{**

**get;**

**}**

**}**

**class MyClass : IMyInterface**

**{**

**// метод**

**public double MyGetPi()**

**{**

**return Math.PI;**

**}**

**// реализация свойства в классе**

**public double MyPi**

**{**

**get**

**{**

**return Math.PI;**

**}**

**}**

**}**

**// использование интерфейсного свойства в обработчике события клика на кнопке**

**private void button1\_Click(object sender, EventArgs e)**

**{**

**MyClass mc = new MyClass(); // создание объекта класса mc**

**label1.Text = mc.MyPi.ToString(); // чтение свойства**

**}**

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**13. Пример интерфейса, в котором описывается индексатор.**

Общая форма объявления интерфейсного индексатора имеет вид:

***тип* this[int *индекс*]**

**{**

**get;**

**set;**

**}**

Пример описания и использования интерфейсного индексатора, который считывает элемент из массива, состоящего из 5 элементов типа double.

**public interface IMyInterface**

**{**

**// интерфейсный индексатор**

**double this[int index]**

**{**

**get;**

**}**

**}**

**class MyClass : IMyInterface**

**{**

**double[] mas = { 3, 2.9, 0.5, 7, 8.3 };**

**public double this[int index]**

**{**

**get**

**{**

**return mas[index];**

**}**

**}**

**}**

**private void button1\_Click(object sender, EventArgs e)**

**{**

**MyClass mc = new MyClass(); // создание объекта класса mc**

**double d;**

**d = mc[2]; // d = 0.5**

**label1.Text = d.ToString();**

**}**

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**14. Какие элементы программирования языка C# нельзя описывать в интерфейсах?**

Интерфейсы не могут содержать:

* члены данных;
* конструкторы;
* деструкторы;
* операторные методы.

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**15. Как работает механизм наследования интерфейсов?**

Интерфейс может наследовать другой интерфейс. Синтаксис наследования интерфейсов такой же, как и у классов.

Общая форма наследования интерфейса следующая:

**interface *имя\_интерфейса* : *имя\_интерфейса1*, *имя\_интерфейса2*, ..., *имя\_интерфейсаN***

**{**

**// методы, свойства, индексаторы и события интерфейса**

**...**

**}**

где *имя\_интерфейса* – имя интерфейса, который наследует другие интерфейсы;

*имя\_интерфейса1*, *имя\_интерфейса2*, …, *имя\_интерфейсаN* – имена интерфейсов-предков.

**Пример.** В данном примере класс MyClass использует интерфейс, который наследует другой интерфейс. В классе нужно реализовать все методы (свойства, индексаторы, события) интерфейса MyInterface1 и интерфейса MyInterface2.

**// базовый интерфейс**

**interface MyInterface1**

**{**

**void Int1\_Meth();**

**}**

**// интерфейс, который наследует другой интерфейс**

**interface MyInterface2 : MyInterface1**

**{**

**void Int2\_Meth();**

**}**

**// класс, который использует интерфейс MyInterface2**

**class MyClass : MyInterface2**

**{**

**// реализация метода интерфейса MyInterface1**

**public void Int1\_Meth()**

**{**

**// тело метода**

**// ...**

**return;**

**}**

**// реализация метода интерфейса MyInterface2**

**public void Int2\_Meth()**

**{**

**// тело метода**

**// ...**

**return;**

**}**

**}**

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**16. Что такое явная реализация члена интерфейса?**

Если перед именем метода (свойства, индексатора, события) стоит имя интерфейса через разделитель ‘ **.** ‘ (точка), то это называется явной реализацией члена интерфейса.

**Пример** явной реализации.

**// базовый интерфейс**

**interface MyInterface1**

**{**

**void Method();**

**}**

**// класс, который реализует интерфейс MyInterface1**

**class MyClass : MyInterface1**

**{**

**// явная реализация метода интерфейса MyInterface1**

**void MyInterface1.Method() // указывается имя интерфейса**

**{**

**// тело метода**

**// ...**

**return;**

**}**

**}**

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**17. Когда целесообразно применять явную реализацию члена интерфейса? Примеры.**

Явная реализация члена интерфейса применяется в следующих случаях:

* когда нужно, чтобы интерфейсный метод был доступен по интерфейсной ссылке, а не по объекту класса, реализующего данный интерфейс. В этом случае интерфейсный метод не является открытым (public) членом класса (см. пример 1);
* когда в одном классе реализованы два интерфейса, в которых методы имеют одинаковые имена и сигнатуру (см. пример 2).

**Пример 1**. Явная реализация интерфейсного метода. По интерфейсной ссылке метод есть доступен, а по объекту класса недоступен.

**// Интерфейс**

**interface MyInterface1**

**{**

**void Method();**

**}**

**// класс, вызывающий интерфейс**

**class MyClass : MyInterface1**

**{**

**// явная реализация метода интерфейса MyInterface1**

**// модификатор доступа, должен отсутствовать**

**void MyInterface1.Method() // указывается имя интерфейса**

**{**

**// тело метода**

**// ...**

**return;**

**}**

**void InternalMethod()**

**{**

**MyInterface1 mi = this; // mi - интерфейсная ссылка**

**mi.Method(); // работает!**

**MyClass mc = this; // mc - объект класса MyClass**

**// mc.Method() - невозможно вызвать - метод не открыт для объекта**

**}**

**}**

**Пример 2.**Есть два интерфейса MyInterface1 и MyInterface2. Каждый из них имеет методы с одинаковыми именами и сигнатурами. В данном случае это метод Method(), не возвращающий параметров (void). С помощью явной реализации класс распознает эти методы.

**// интерфейс 1**

**interface MyInterface1**

**{**

**void Method();**

**}**

**// интерфейс 2**

**interface MyInterface2**

**{**

**void Method();**

**}**

**// класс, который использует два интерфейса**

**class MyClass : MyInterface1, MyInterface2**

**{**

**// явная реализация - модификатор доступа (public) должен отсутствовать**

**// метод из интерфейса MyInterface1**

**void MyInterface1.Method()**

**{**

**// тело метода**

**// ...**

**return;**

**}**

**// метод из интерфейса MyInterface2**

**void MyInterface2.Method()**

**{**

**// тело метода**

**// ...**

**return;**

**}**

**}**

[**⇑**](https://www.bestprog.net/ru/2016/12/29/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d1%8b/#contents)

**18. В каких случаях лучше использовать интерфейс, а в каких абстрактный класс?**

Интерфейс целесообразно использовать в случаях, если некоторые понятия должны быть описаны с точки зрения функционального назначения, без уточнения деталей реализации.

Абстрактный класс целесообразно использовать тогда, когда все же нужно уточнять некоторые детали реализации.