**Вопросы**

1. Для чего используют статические классы?

Статические классы предназначены для хранения статических членов и не могут быть инстанцированы. Их используют для группировки методов и свойств, которые не требуют создания экземпляра класса.

2. Что может содержать статический класс?

Статический класс может содержать статические поля, статические методы, статические свойства и статические события. Он не может содержать экземплярных полей или методов, так как не допускает создание экземпляров.

3.Что такое производный и базовый классы?

 **Базовый класс** (родительский класс) — это класс, от которого происходит наследование других классов. Он содержит общие для всех своих производных классов поля, методы, свойства и другие члены.

 **Производный класс** (дочерний класс) — это класс, который наследует функциональность от базового класса. Производный класс может добавлять новые члены или переопределять унаследованные члены.

1. Как используют ключевое слово base?

Ключевое слово base используется в C# для доступа к членам базового класса из производного класса. Оно позволяет вызывать конструктор базового класса из конструктора производного класса и обращаться к унаследованным полям, методам и свойствам.

1. В чем заключена основная задача наследования?

Оно позволяет определить дочерний класс, который использует (наследует), расширяет или изменяет возможности родительского класса.

1. Пусть базовый класс содержит метод basefunc(), а производный класс не имеет метода с таким именем. Может ли объект производного класса иметь доступ к методу basefunc()? Если да, то при каких условиях?

Да, объект производного класса может получить доступ к методу basefunc() из базового класса, даже если в производном классе нет метода с таким именем. Для этого метод basefunc() должен быть объявлен в базовом классе как public, protected или internal (при условии, что производный класс находится в том же сборке). В таком случае экземпляр производного класса сможет вызывать метод basefunc() напрямую.

1. Напишите объявление конструктора без аргументов для производного класса B, который будет вызывать конструктор без аргументов базового класса A.

public class A

{

public A() { /\* Конструктор базового класса \*/ }

}

public class B : A

{

public B() : base() { /\* Конструктор производного класса \*/ }

}

1. Что такое полиморфизм? Приведите пример.

ключевой аспект объектно-ориентированного программирования

способность к изменению функций, унаследованных от базового класса

public abstract class Animal // Базовый класс "Животное"

{

public abstract void Speak(); // Метод "говорить"

}

public class Dog : Animal // Производный класс "Собака"

{

public override void Speak() // Переопределение метода

{

Console.WriteLine("Гав!");

}

}

public class Cat : Animal // Производный класс "Кошка"

{

public override void Speak() // Переопределение метода

{

Console.WriteLine("Мяу!");

}

}

Полиморфизм позволяет нам использовать общий код для работы с разными объектами, но при этом каждый объект может выполнять действия по-своему.

1. Определите назначение виртуальных функций.

**Виртуальные функции** (или виртуальные методы) — это методы в базовом классе, которые можно переопределить в производных классах. Они позволяют задать общее поведение для всех классов, которые будут унаследованы от базового, но при этом дают возможность каждому производному классу изменить это поведение по-своему.

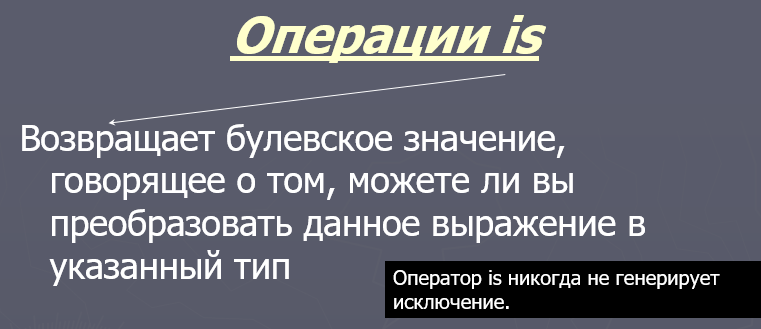
1. Кому доступны переменные с модификатором protected?

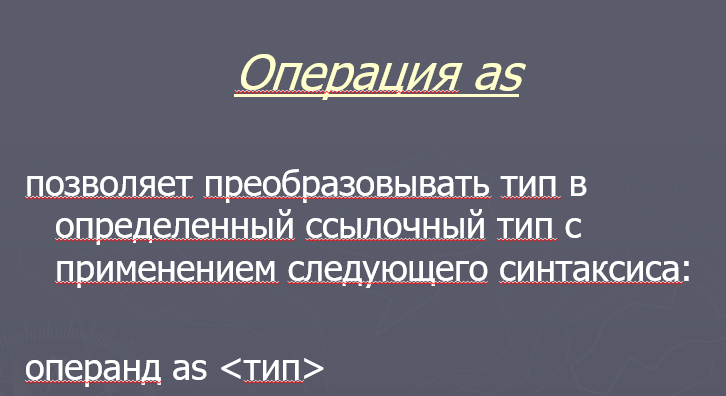
Переменные с модификатором protected доступны внутри самого класса и во всех производных классах, даже если они находятся в других сборках.

1. Наследуются ли переменные с модификатором private?

Да, переменные с модификатором private наследуются, но они недоступны в производных классах напрямую. Они остаются приватными для класса, в котором были объявлены, и доступны только через методы или свойства, если они предоставлены в базовом классе.

1. As, is – что это, как применяется? В чем между ними отличие ?





1. Поддерживает ли C# множественное наследование?

Нет, C# не поддерживает множественное наследование классов (класс может иметь только одного родителя). Однако C# поддерживает реализацию нескольких интерфейсов, что частично компенсирует отсутствие множественного наследования.

1. Можно ли запретить наследование от класса?

Да, чтобы запретить наследование от класса, его можно объявить с модификатором sealed. Класс sealed не может быть унаследован.

1. Можно ли разрешить наследование класса, но запретить перекрытие метода?

Да, в C# можно разрешить наследование класса, но запретить перекрытие метода. Для этого используется модификатор доступа sealed в сочетании с модификатором virtual или abstract.

1. Что такое абстрактный класс?

Абстрактный класс — это класс, который не может быть создан как объект напрямую и предназначен для использования в качестве базового класса. Абстрактный класс может содержать как обычные, так и абстрактные методы (методы без реализации). Абстрактные классы позволяют создать общий интерфейс и частичную реализацию для группы связанных классов.

1. В каком случае вы обязаны объявить класс абстрактным?

Вы обязаны объявить класс **абстрактным**, если он содержит хотя бы один **абстрактный метод**. Абстрактный класс — это класс, от которого нельзя создать объект напрямую; он служит основой для производных классов, которым необходимо реализовать все абстрактные методы базового класса.

18. В чем разница между абстрактными и виртуальными классами? Между виртуальными и абстрактными методами?

 **Абстрактный класс** — это класс, который не может быть инстанцирован напрямую и может содержать как обычные, так и абстрактные методы. Абстрактный класс является основой для других классов, задавая общую структуру.

 **Виртуальный класс** — это термин, который не используется в C# для классов. Виртуальными могут быть методы, свойства и события.

 **Абстрактный метод**: объявляется без реализации (тела), и каждый класс-наследник обязан его реализовать.

 **Виртуальный метод**: содержит реализацию, но производные классы могут её изменить. При этом реализация базового класса может быть использована без изменений.

1. Какие компоненты класса могут быть виртуальными?

В C# виртуальными могут быть следующие компоненты класса:

* Методы
* Свойства (get и set)
* Индексаторы
* События

Эти компоненты можно объявить как virtual в базовом классе, чтобы производные классы могли их переопределить.

1. Что такое интерфейс?

**Интерфейс** — это тип, который определяет контракт, набор методов, свойств, событий или индексаторов, которые класс или структура обязаны реализовать. Интерфейс не содержит реализации, он лишь описывает, **что** должно быть реализовано, оставляя производным классам выбор, **как** это сделать.

1. Что может содержать интерфейс?

Интерфейс может содержать:

* **Методы** (без реализации)
* **Свойства** (без реализации)
* **События** (без реализации)
* **Индексаторы** (без реализации)

В новых версиях C# также допускается добавлять **методы с реализацией** по умолчанию в интерфейсы, что позволяет создавать более гибкие контракты.

1. Как работать с объектом через унаследованный интерфейс?

Создайте интерфейс, описывающий требуемые методы и свойства

Создайте классы, реализующие интерфейс.

1. Приведите пример явной реализации интерфейса.

public interface IExample

{

void Method();

}

public class ExampleClass : IExample

{

void IExample.Method() // Явная реализация интерфейса

{

Console.WriteLine("Метод интерфейса IExample реализован явно.");

}

public void CallMethod()

{

// Вызов метода через интерфейс

IExample example = this;

example.Method();

}

}

1. Почему нельзя указать модификатор видимости для методов интерфейса?

Методы интерфейса по умолчанию являются **public**, и все реализации должны быть доступными. Поскольку интерфейс описывает контракт, по которому классы должны следовать, возможность указывать модификаторы доступа создала бы путаницу о том, какие методы доступны. Это противоречило бы принципам интерфейса как обобщенного контракта.

25. Можно ли наследовать от нескольких интерфейсов?

26. Назовите отличия между интерфейсом и абстрактным классом.

**Интерфейс**:

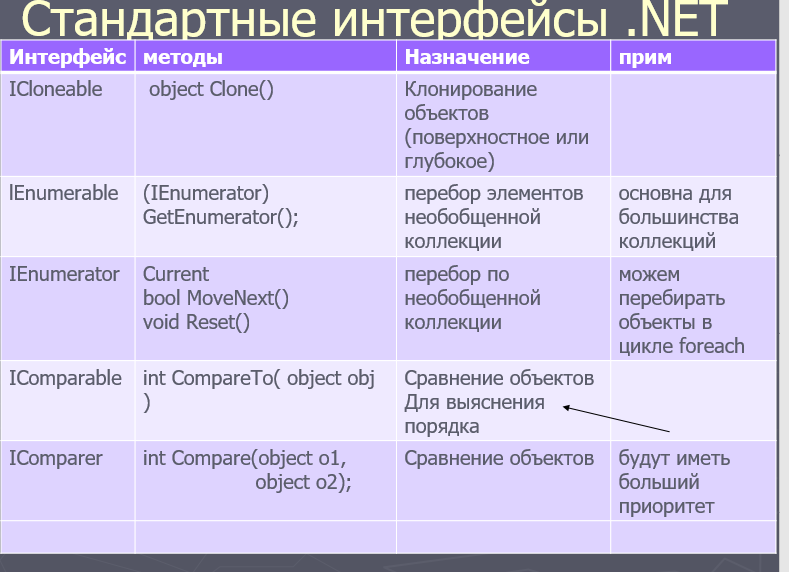
* Представляет собой **контракт**. Он определяет набор методов и свойств, которые класс должен реализовать, но не содержит их реализации. Интерфейсы используются, чтобы гарантировать, что класс будет иметь определённый набор методов, но не определяют, как именно эти методы должны работать.

**Абстрактный класс**:

* Это класс, который может содержать как абстрактные методы (без реализации), так и методы с реализацией. Абстрактные классы служат для создания базовых классов, от которых можно наследоваться.

Также может иметь поля (состояние) и различные модификаторы доступа (public, protected, private)

27. Для чего используются стандартные интерфейсы ICloneable, IComparable, IComparer, lEnumerable?



28. В какой строке приведенного ниже фрагмента листинга не содержится ошибки?

class A

{

public virtual abstract void m() { } //1 В C# **абстрактный метод** уже предполагает, что он может быть переопределён, и не может иметь реализации. Ключевое слово virtual здесь избыточно и не допускается для абстрактных методов.

public virtual void g() { }//2 правильно

public virtual new new void f() { }//3

public static virtual void h() { }//4 В C# **статические методы** не могут быть виртуальными, так как их нельзя переопределять.

}

29. Что будет выведено на консоль в результате выполнения следующего фрагмента?

|  |
| --- |
| class A  {  public int x = 1;  }  class B : A  {  public new int x = 2;  public void m(int a, int b)  {  x = a;  base.x = b;  Console.Write(x + " " + base.x); |
| }  }  class Test  {  static void Main(string[] args)  {  A a = new A();  B b = new B();  b.m(3, 4);  }  } |

3 4.

30. Что будет выведено на консоль в результате выполнения следующего фрагмента кода?

class A

{

public class B : A

{

public override void mA()

{

Console.WriteLine("B ");

}

}

public virtual void mA()

{

Console.WriteLine("A ");

}

}

class Prоgrаm

{

static void Main(string[] args)

{

A a = new A();

A.B b = new A.B();

a.mA();

b.mA();

}

}

A B.

31. Чем может быть M4 если дано следующее определение:

|  |  |
| --- | --- |
| **public class** C1 : M1, M2 { } **public struct** S1 : M3, M4 { }; | Варианты ответа:  1) M4 - только интерфейс  2) M4 - интерфейс или класс  3) M4 - только класс  4) M4 - только структура  5) M4 - делегат |

Ответ: **1) M4 - только интерфейс**. В C# структура может реализовывать **только интерфейсы**, но не может наследовать другие структуры, классы или делегаты. В приведённом примере S1 : M3, M4 говорит о том, что S1 реализует интерфейсы M3 и M4, поэтому M4 может быть только интерфейсом.

32. Выберите верное присваивание для объектов, определенных в листинге.

|  |  |
| --- | --- |
| class A { }  class B : A { }  class C : B { }  class D { }  class Test  {  static void Main(string[] args)  {  A a = new A();  B b = new B();  C c = new C();  D d = new D();  }  } | Варианты ответа:  1) b = a;  2) a = b;  3) c = a;  4) d = a;  5) с = b; |

33. Что будет выведено на консоль в результате выполнения следующего фрагмента, если раскомментировать строчку 1?

|  |  |
| --- | --- |
| public abstract class A  {  public virtual void method()  { Console.Write("A "); }  }  public class B : A  {  public override void method()  { // base.method(); // 1  // this.method(); // 2  Console.Write("B ");  }  }  class Program2  {  static void Main(string[] args)  {  A my = new B();  my.method();  }  } | Варианты ответа:  1) B  2) A  3) A B  4) B A  5) 0 |

A B.

34. В какой строке приведенного ниже фрагмента листинга содержится ошибка?

|  |  |  |  |
| --- | --- | --- | --- |
| public abstract class A  {  public virtual string m() { return "A"; }//1  }  public class B : A  {  public override new string m() { return "B"; }//2 ошибка В C# нельзя использовать одновременно override и new для одного метода. Ключевое слово override указывает, что метод должен переопределить виртуальный или абстрактный метод базового класса, тогда как new используется для скрытия метода из базового класса без его переопределения.  }  public class C : B  {  public string m() { return "C"; }//3  }  class Prоgrаm  {  static void Main(string[] args)  {  A ac = new C(); //4  Console.WriteLine(ac.m());  }  }  35. Почему приведенный ниже фрагмент листинга содержит ошибку?   |  | | --- | | abstract class Student //1  {  public int Age { get; set; } //2  public string Name { get; set; } //3  }  static void Main(string[] args)  {  Student Olga = new Student();//4 строке 1 класс Student объявлен как abstract. Абстрактные классы предназначены для того, чтобы служить базой для других классов, и **их нельзя создавать напрямую**.  } |   Можно создать класс который наследует класс student  36. В какой строке может быть ошибка компиляции?   |  | | --- | | class A{}  class B : A { }  class C : A { } //1  class Program4  {  static void Main()  {  A one = new B(); //2  A two = new C(); //3  one = two; //4  }  } |   37. Что будет выведено на консоль в результате выполнения следующего фрагмента листинга:   |  | | --- | | interface Interface1  {  void f();  void g();  }  class A  {  public void f() { System.Console.WriteLine("F"); }  public void g() { System.Console.WriteLine("G"); }  }  class B : A, Interface1  {  new public void g() { System.Console.WriteLine("new G"); }  }  class Program5  {  static void Main(string[] args)  {  //Interface1 obj = new B();  //obj.g();  B obj = new B();  obj.g();  }  } | |

New G.