Теория

Объект - автономный модуль со своим состоянием и поведением

Класс – это некоторое абстрактное понятие - шаблон, по которому определяется форма объекта

Объект – это физическая реализация класса(шаблона).

Описанием объекта является **класс**, а объект представляет экземпляр этого класса. Для хранения данных в классе применяются **поля**. По сути **поля класса** - это переменные, определенные на уровне класса. Кроме того, класс может определять некоторое поведение или выполняемые действия. Для определения поведения в классе применяются методы. Методы класса имеют доступ к его поля. После определения класса мы можем создавать его объекты. Для создания объекта применяются **конструкторы**.

Конструкторы - это специальные методы класса, которые используются для инициализации объектов. Они выполняются при создании нового экземпляра класса и позволяют задать начальные значения полей объекта. Сначала идет оператор **new**, который выделяет память для объекта, а после него идет вызов **конструктора**.

Конструктор по умолчанию - Если в классе не определено ни одного конструктора то для этого класса автоматически создается пустой конструктор по умолчанию, который не принимает никаких параметров.

Статический конструктор (конструктор типа) выполняется один раз при первом обращении к классу и используется для инициализации статических полей класса или выполнения других инициализирующих операций. У статического конструктора нет параметров и модификатора доступа public.

Закрытый конструктор - это конструктор класса, объявленный с модификатором доступа private, что означает, что он не может быть вызван за пределами самого класса. То есть он недоступен для внешнего кода и может быть вызван только изнутри самого класса.

Поле только для чтения - это поле, которое можно установить только один раз, обычно в конструкторе, и затем оно становится доступным только для чтения.

Поле-константа - это поле, значение которого не может быть изменено после его инициализации. Оно объявляется с ключевым словом const.

Свойства (get, set) используются для получения и установки значений полей класса. Они обеспечивают контроль над доступом к полям и позволяют выполнить дополнительную логику при чтении или записи значения.

ref и out - это два модификатора параметров метода в языках программирования, таких как C# или C++. Они позволяют передавать аргументы методу по ссылке, а не по значению, что может быть полезно в некоторых ситуациях.

Статическое поле класса хранит информацию, которая является общей для всех экземпляров класса. Например, можно создать статическое поле для подсчета количества созданных объектов класса как в лабе.

this - это ключевое слово, которое используется в языках программирования, таких как C# или Java, для ссылки на текущий экземпляр класса. Оно представляет собой ссылку на сам объект, в контексте которого выполняется код.

Класс можно разделить на несколько частей с помощью ключевого слова partial. Каждая часть содержит часть определения класса и все части должны находиться в одном и том же пространстве имен.

Методы класса Equals, GetHashCode и ToString являются методами, унаследованными от базового класса Object и могут быть переопределены в пользовательском классе.

Анонимный тип (anonymous type) - это тип данных, который создается автоматически компилятором на основе его структуры и инициализации в момент компиляции. Анонимные типы предоставляют удобный способ создания объектов без явного определения класса.

В C# анонимные типы объявляются с использованием ключевого слова var и инициализации объекта с набором свойств. Каждому свойству присваивается имя и значение, и компилятор автоматически определяет типы свойств на основе значений.

Производные классы - это классы, которые наследуются от других классов, называемых базовыми классами или родительскими классами.

Доступ к членам типов

► public - доступ не ограничен – все члены во всех сборках

► private - по умолчанию для членов класса (используется для вложенных классов). Доступен только методам в определяющем типе и вложенных в него типах

► protected - (используется для вложенных классов) Доступен только методам в определяющем типе (и вложенных в него типах) или в одном из его производных типов независимо от сборки

► internal - доступ только из данной сборки

1. public (общедоступный) - это модификатор доступа, который позволяет членам класса быть доступными из любого места кода в любых сборках. Это означает, что любой другой класс или код может использовать и обращаться к общедоступным членам.
2. private (частный) - это модификатор доступа по умолчанию для членов класса. Он ограничивает доступ к этим членам только в пределах самого класса. Это значит, что только методы внутри самого класса могут использовать или обращаться к частным членам. Вложенные классы также имеют доступ к частным членам внешнего класса.
3. protected (защищенный) - этот модификатор доступа используется в основном для вложенных классов. Он позволяет методам внутри определяющего типа и его вложенных типов использовать или обращаться к защищенным членам. члены класса были доступны только внутри этого класса и его производных классов.
4. internal (внутренний) - этот модификатор доступа ограничивает доступ к членам только внутри текущей сборки. Это означает, что только другие классы и код, находящиеся в той же сборке, могут использовать или обращаться к внутренним членам.

Вопросы

1. Принципы ООП (Объектно-Ориентированного Программирования) включают в себя следующие основные концепции:

a. Инкапсуляция (пакетирование) - механизм, связывающий вместе данные и код, обрабатывающий эти данные, и сохраняющий их от внешнего воздействия и ошибочного использования. Инкапсуляция - скрытие некоторых моментов реализации класса от других частей программы.

b. Наследование: процесс, благодаря которому один объект может наследовать (приобретать) свойства от другого объекта.

c. Полиморфизм: способность вызывать метод потомка через экземпляр предка

d. Абстракция подразумевает разделение и независимое рассмотрение интерфейса и реализации

2. В .NET Framework базовым классом, от которого наследуются все классы, является класс `System.Object`.

3. Открытые методы `System.Object` включают:

- `ToString()`: Возвращает строковое представление объекта.

- `Equals(Object obj)`: Проверяет, равен ли текущий объект указанному объекту.

- `GetHashCode()`: Возвращает хеш-код объекта.

- `GetType()`: Возвращает тип объекта.

4. Методы с закрытым доступом (protected или private):

* + MemberwiseClone(): Создает неполную копию текущего объекта.
  + Finalize(): Выполняет очистку или другие завершающие действия перед уничтожением объекта.
  + ReferenceEquals(object objA, object objB): Определяет, являются ли два указанных объекта одним и тем же объектом.
  + ToString(): Может быть переопределен в производных классах для возврата более конкретного представления объекта.

5. Пример определения класса на языке C#:

public class Person

{

// Поля (переменные)

public string Name;

public int Age;

// Метод для вывода информации о персоне

public void DisplayInfo()

{

Console.WriteLine($"Name: {Name}, Age: {Age}");

}

}

6. Ключевые слова, которые можно использовать при определении класса в C#, включают:

- `class`: Определяет класс.

- `public`, `private`, `protected`, `internal`: Модификаторы доступа, которые определяют видимость класса и его членов.

- `static`: Определяет статический класс или член класса.

- `abstract`: Определяет абстрактный класс или метод.

- `sealed`: Запрещает наследование от класса или переопределение метода.

7. Основное отличие между объектом и классом заключается в следующем:

- Класс - это шаблон или описание, по которому создаются объекты. Класс определяет состояние (поля) и поведение (методы) объектов.

- Объект - это конкретный экземпляр класса, созданный на основе этого класса. Объект имеет конкретные значения полей, определенные в классе, и может вызывать методы, определенные в классе.

8. Конструктор - это специальный метод, который вызывается при создании объекта класса. Конструкторы используются для инициализации полей объекта и выполнения других начальных действий. Они вызываются с помощью оператора `new`. Конструкторы могут быть перегружены, что позволяет создавать объекты с разными наборами параметров.

9. Свойства конструктора включают:

- Имя: Имя конструктора совпадает с именем класса.

- Параметры: Конструктор может принимать параметры, которые передаются при создании объекта.

- Тело: Тело конструктора содержит код, который выполняется при создании объекта.

10. В C# деструктор (destructor) отсутствует. Вместо деструкторов в C# используется механизм сборки мусора, который автоматически освобождает память, занимаемую объектами, когда они больше не используются.

11. Ключевое слово `this` используется внутри методов класса для обращения к текущему экземпляру объекта. Оно позволяет различать локальные переменные и поля объекта, когда их имена совпадают.

13. В C# существуют следующие спецификаторы доступа для класса и методов класса:

- `public`: Класс или метод доступен из любого места.

- `private`: Класс или метод доступен только внутри своего объявления.

- `protected`: Класс или метод доступен внутри своего объявления и в производных классах.

- `internal`: Класс или метод доступен внутри текущей сборки.

- `protected internal`: Класс или метод доступен внутри текущей сборки и в производных классах (комбинация `protected` и `internal`).

14. Модификатор `protected internal` означает, что член класса (например, метод или поле) доступен внутри текущей сборки и в производных классах (по отношению к наследованию), но не доступен извне текущей сборки.

15. `ref` и `out` параметры функции используются для передачи аргументов по ссылке, что позволяет функции изменять значения переданных аргументов.

- `ref` используется, когда аргумент должен быть проинициализирован до передачи в функцию и изменения внутри функции должны отразиться на вызывающей стороне.

- `out` используется, когда аргумент может быть не проинициализирован до передачи в функцию, и функция должна обязательно его проинициализировать внутри себя.

16. Пример необязательных параметров:

```csharp

public void ExampleMethod(int x, int y = 10)

{

// Ваш код

}

```

Пример именованных параметров:

```csharp

ExampleMethod(y: 20, x: 5);

```

17. Примеры полей класса:

- Статические поля: `public static int Count;`

- Константные поля: `public const double Pi = 3.14159265359;`

- Поля только для чтения: `public readonly string Name;`

18. Пример определения свойства класса:

```csharp

private int \_age;

public int Age

{

get { return \_age; }

set { \_age = value; }

}

```

Свойства связаны с инкапсуляцией, так как они обеспечивают контролируемый доступ к данным поля `\_age`, скрывая его от прямого доступа извне класса.

19. Явное имя параметра, передаваемого в метод `set` свойства класса, можно получить с помощью ключевого слова `value`.

20. Автоматические свойства позволяют определить свойство без явного задания бэкфилда. Пример:

```csharp

public int Age { get; set; }

```

21. Индексаторы класса позволяют обращаться к объекту как к массиву или коллекции. Ограничения на индексатор включают в себя тип индекса и обязательное наличие `get` и/или `set` методов.

22. Перегруженный метод - это метод с тем же именем, что и другой метод в классе, но с разными параметрами (разными типами или количеством параметров).

23. Парциальные классы (partial classes) позволяют разделять определение класса на несколько файлов. Это удобно для генерации кода и совместной разработки. Преимущества включают поддержку разделения логики и генерируемого кода.

24. Анонимный тип в C# - это безымянный тип, который позволяет создавать объекты с набором свойств на лету. Пример:

```csharp

var person = new { Name = "Alice", Age = 30 };

```

25. Статические классы используются для группировки связанных методов и данных. Они не могут быть инстанциированы, и их члены доступны без создания объекта класса.

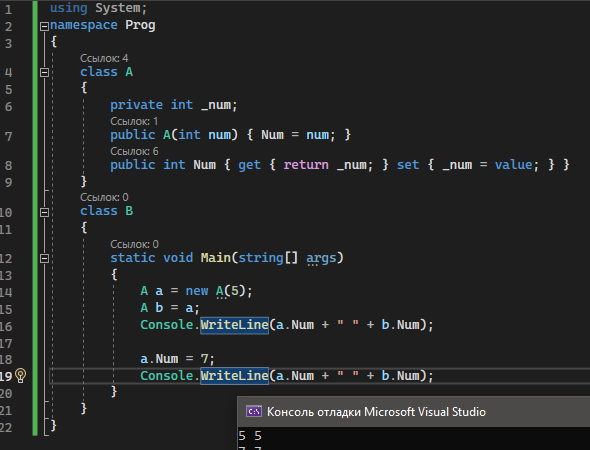
26. Статическое поле принадлежит классу, а экземплярное поле принадлежит объекту класса. Статическое поле существует в единственном экземпляре для всего класса, в то время как экземплярное поле имеется отдельно для каждого объекта.

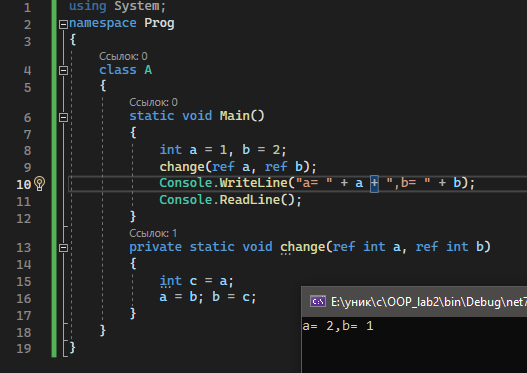
27. Статические конструкторы используются для инициализации статических членов класса. Они вызываются один раз, когда класс впервые используется, и выполняют инициализацию.

28. Поверхностное (shallow) копирование копирует объект, но не его вложенные объекты, в то время как глубокое (deep) копирование копирует объект и все его вложенные объекты рекурсивно.

29. Равенство (equality) проверяет, имеют ли два объекта одинаковые значения, в то время как тождество (identity) проверяет, являются ли два объекта одним и тем же экземпляром.

30. Частичные классы (partial classes) и частичные методы (partial methods) позволяют разделять определение класса или метода на несколько файлов. Они используются, например, для разделения кода, который генерируется автоматически, и пользовательского кода. При этом можно иметь часть класса или метода, которая может быть определена в одном месте и другую часть, которая может быть определена в другом месте.

31. 

32. 

33. Конструкторы в C# не имеют возвращаемого типа, и их имя должно совпадать с именем класса. Правильные конструкторы должны иметь имя класса и не иметь возвращаемого типа. В вашем коде:

1. Правильный конструктор: `public A() { }`

2. Этот конструктор имеет возвращаемый тип `int`, что является неверным для конструктора. Конструкторы не должны иметь возвращаемого типа. Правильный синтаксис должен быть таким: `public A() { }`

3. Правильный конструктор: `public A(int someI) { }`

4. Правильный конструктор: `public A(A somA) { }`

Таким образом, конструктор №2 задан неверно из-за наличия возвращаемого типа `int`.

34. При вызове конструктора класса `Motorcycle`, можно задать два аргумента:

1. `int intensity` - опциональный аргумент с значением по умолчанию равным 0.

2. `string name` - опциональный аргумент с значением по умолчанию равным пустой строке "".

При вызове конструктора вы можете указать ноль, один или оба из этих аргументов, в зависимости от вашей потребности. В случае, если аргументы не указаны при вызове конструктора, будут использованы их значения по умолчанию.

34. Проблема с созданием объекта класса `A` в вашем коде заключается в том, что конструкторы класса `A` имеют модификаторы доступа по умолчанию, что означает, что они доступны только внутри текущей сборки (internal).

В методе `Main` вы пытаетесь создать объект класса `A` извне сборки (в другом проекте или вне текущей сборки), что вызывает ошибку компиляции, так как конструкторы не доступны извне.

Чтобы решить эту проблему, вы можете сделать одно из следующего:

1. Изменить модификатор доступа конструкторов на `public`, чтобы они стали доступными извне текущей сборки:

```csharp

internal class A

{

public A() {} //1

public A(string st) { } //2

public A(int a) { } //3

public A(int a, int b) { } //4

}

```

2. Если вы хотите оставить конструкторы с модификатором `internal`, то убедитесь, что код, который пытается создать объект класса `A`, находится в той же сборке или проекте, что и класс `A`.

36. При выполнении данной программы будет выведено следующее:

```

A static

A

```

Объявление `static A()` представляет собой статический конструктор класса `A`. Статический конструктор вызывается один раз при первом использовании класса или при создании первого объекта класса, и он выполняется до вызова конструктора экземпляра.

Затем вызывается конструктор `public A()`, который выводит "A".В приведенном классе `Points` следующая строка вызовет ошибку компиляции:

```csharp

b = 30; //3

```

Ошибка компиляции произойдет, потому что вы пытаетесь присвоить новое значение статическому `readonly` полю `b`, которое уже было проинициализировано при его объявлении. Поля, объявленные как `readonly`, могут быть проинициализированы только один раз, их значение нельзя изменить после этой инициализации.37.