1. Главное отличие структур и классов: **структуры передаются по значению (то есть копируются), объекты классов — по ссылке**. Именно это является существенным различием в их поведении, а не то, где они хранятся. 4. Структуру тоже можно передать по ссылке, используя модификаторы out и ref.

Иногда прямой доступ к объектам как к значениям простых типов оказывается полезно иметь, например, ради повышения эффективности программы. Ведь каждый доступ к объектам (даже самым мелким) по ссылке связан с дополнительными издержками на расход вычислительных ресурсов и оперативной памяти.

Для разрешения подобных затруднений в C# предусмотрена **структура**, которая подобна классу, но относится к типу значения, а не к ссылочному типу данных. Т.е. структуры отличаются от классов тем, как они сохраняются в памяти и как к ним осуществляется доступ (классы — это ссылочные типы, размещаемые в куче, структуры — типы значений, размещаемые в [стеке](https://professorweb.ru/my/csharp/charp_theory/level12/12_6.php)), а также некоторыми свойствами (например, структуры не поддерживают наследование). Из соображений производительности вы будете использовать структуры для небольших типов данных. Однако в отношении синтаксиса структуры очень похожи на классы.

Как и у классов, у каждой структуры имеются свои члены: методы, поля, индексаторы, свойства, операторные методы и события. В структурах допускается также определять конструкторы, но не деструкторы. В то же время для структуры нельзя определить конструктор, используемый по умолчанию (т.е. конструктор без параметров). Дело в том, что конструктор, вызываемый по умолчанию, определяется для всех структур автоматически и не подлежит изменению. Такой конструктор инициализирует поля структуры значениями, задаваемыми по умолчанию. А поскольку структуры не поддерживают наследование, то их члены нельзя указывать как abstract, virtual или protected.

Обратите внимание, когда одна структура присваивается другой, создается копия ее объекта. В этом заключается одно из главных отличий структуры от класса. Когда ссылка на один класс присваивается ссылке на другой класс, в итоге ссылка в левой части оператора присваивания указывает на тот же самый объект, что и ссылка в правой его части. А когда переменная одной структуры присваивается переменной другой структуры, создается копия объекта структуры из правой части оператора присваивании.



Перечисления представляют набор логически связанных констант.

Объявление перечисления происходит :

enum название\_перечисления

{

    // значения перечисления

    значение1,

    значение2,

    .......

    значениеN

}

Каждое перечисление фактически определяет новый тип данных, с помощью которых мы также, как и с помощью любого другого типа, можем определять переменные, константы, параметры методов и т.д. В качестве значения переменной, константы и параметра метода, которые представляют перечисление, должна выступать одна из констант этого перечисления, например:

const DayTime dayTime = DayTime.Morning;

1. Интерфейс IComparable определен в пространстве имен System. Он содержит всего один метод CompareTo, возвращающий результат сравнения двух объектов — текущего и переданного ему в качестве параметра:

Интерфейс IComparer определен в пространстве имен System.Collections. Он содержит один метод CompareTo, возвращающий результат сравнения двух объектов, переданных ему в качестве параметров:

Интерфейс IEnumerable (*перечислимый*) определяет всего один метод —GetEnumerator, возвращающий объект типа IEnumerator (*перечислитель*), который можно использовать для просмотра элементов объекта.

IDisposable - интерфейс, который позволяет освободить ресурсы, занятые объектом.

Интерфейс IEnumerable (*перечислимый*) определяет всего один метод —GetEnumerator, возвращающий объект типа IEnumerator (*перечислитель*), который можно использовать для просмотра элементов объекта.

Стандартные интерфейсы являются также важной частью классов коллекций, предоставляющих различные средства, в том числе стеки и очереди, для хранения целых групп объектов. Так, в интерфейсеSystem.Collections.ICollectionопределяются функции для всей коллекции, а в интерфейсеSystem.Collections.IEnumerator— способ последовательного обращения к элементам коллекции. Эти и многие другие интерфейсы подробнее рассматриваются в части II данной книги.

7. Полиморфизм в объектно-ориентированном программировании – это возможность обработки разных типов данных, т. е. принадлежащих к разным классам, с помощью "одной и той же" функции, или метода. На самом деле одинаковым является только имя метода, его исходный код зависит от класса.

Перегрузка методов обеспечивает статический полиморфизм, а виртуальный метод – динамический**. раннее связывание** – адрес функции назначается во время компиляции, и именно этот адрес используется при вызове функции

**позднее связывание** (только для методов классов) – во время выполнения приложения определяется действительный класс объекта, адрес которого находится в указателе, и вызывается метод нужного класса. Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания.

8. Они **позволяют создавать общий код, который может работать как с объектами базового класса, так и с объектами любого его класса-наследника**.

9. override seald метод