### 1. Чем класс отличается от структуры?

- \*\*Ссылочный тип vs. Значимый тип\*\*: Классы являются ссылочными типами, а структуры — значимыми. Это означает, что при передаче структуры создается ее копия, а при передаче класса передается ссылка на объект.

- \*\*Наследование\*\*: Классы поддерживают наследование, структуры — нет.

- \*\*Инициализация\*\*: Классы могут иметь явно заданный конструктор без параметров, структуры всегда имеют неявный конструктор, который инициализирует все поля значениями по умолчанию.

- \*\*Размер\*\*: Структуры обычно используются для хранения небольших данных, так как они более эффективны при передаче по значению.

### 2. Что может и чего не может быть в структуре?

\*\*Что может быть в структуре:\*\*

- Поля

- Методы

- Свойства

- Конструкторы

- Индексаторы

- События

\*\*Что не может быть в структуре:\*\*

- Наследование от других структур или классов (можно только реализовывать интерфейсы).

- Структуры не могут быть абстрактными или содержать абстрактные методы.

- Структуры не могут содержать неявные конструкторы.

### 3. Что такое перечисление? Приведите пример определения и использования перечисления.

\*\*Перечисление\*\* — это специальный тип данных, который позволяет создавать набор именованных целых значений.

\*\*Пример:\*\*

```csharp

enum Days

{

Sunday,

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday

}

// Использование

Days today = Days.Monday;

Console.WriteLine(today); // Вывод: Monday

```

### 4. Перечислите и поясните стандартные интерфейсы .NET.

- \*\*IEnumerable\*\*: Позволяет итерировать по коллекции. Реализуется в коллекциях, чтобы использовать `foreach`.

- \*\*ICollection\*\*: Расширяет IEnumerable, добавляет методы для работы с коллекцией (например, `Add`, `Remove`, `Count`).

- \*\*IList\*\*: Расширяет ICollection, предоставляет доступ по индексу, позволяет изменять элементы коллекции.

- \*\*IComparable\*\*: Определяет метод для сравнения объектов, что позволяет сортировать и сравнивать.

- \*\*ICloneable\*\*: Позволяет создать копию объекта.

- \*\*IDisposable\*\*: Позволяет освобождать неуправляемые ресурсы.

### 5. Как используется интерфейс IComparable?

Интерфейс `IComparable` используется для определения порядка сортировки объектов. Классы, реализующие этот интерфейс, должны переопределить метод `CompareTo`.

\*\*Пример:\*\*

```csharp

public class Person : IComparable<Person>

{

public string Name { get; set; }

public int CompareTo(Person other)

{

return Name.CompareTo(other.Name);

}

}

// Использование

List<Person> people = new List<Person>

{

new Person { Name = "Alice" },

new Person { Name = "Bob" }

};

people.Sort(); // Сортировка по имени

```

### 6. Как используется интерфейс ICloneable?

Интерфейс `ICloneable` используется для создания копий объектов. Классы, реализующие этот интерфейс, должны переопределить метод `Clone`.

\*\*Пример:\*\*

```csharp

public class Person : ICloneable

{

public string Name { get; set; }

public object Clone()

{

return new Person { Name = this.Name };

}

}

// Использование

Person original = new Person { Name = "Alice" };

Person clone = (Person)original.Clone();

```

### 7. Что такое полиморфизм? Перечислите его формы. Приведите примеры.

\*\*Полиморфизм\*\* — это способность объектов разных классов обрабатываться одинаково через общий интерфейс.

\*\*Формы полиморфизма:\*\*

- \*\*Полиморфизм времени компиляции (перегрузка)\*\*: Один и тот же метод может иметь разные сигнатуры.

```csharp

public void Display(int number) { /\*...\*/ }

public void Display(string text) { /\*...\*/ }

```

- \*\*Полиморфизм времени выполнения (переопределение)\*\*: Методы базового класса могут быть переопределены в производных классах.

```csharp

class Animal

{

public virtual void Speak() { Console.WriteLine("Animal speaks"); }

}

class Dog : Animal

{

public override void Speak() { Console.WriteLine("Bark"); }

}

Animal myDog = new Dog();

myDog.Speak(); // Вывод: Bark

```

### 8. Зачем в классе определяют виртуальные методы?

Виртуальные методы позволяют производным классам переопределять их реализацию, что обеспечивает гибкость и возможность полиморфизма. Это полезно для создания обобщенного кода, который может работать с различными объектами.

### 9. Как сделать запрет переопределения методов?

Для запрета переопределения методов в производных классах используется модификатор `sealed`. Можно его применять к виртуальным методам.

\*\*Пример:\*\*

```csharp

public class BaseClass

{

public virtual void Display() { Console.WriteLine("BaseClass Display"); }

}

public class DerivedClass : BaseClass

{

public sealed override void Display() { Console.WriteLine("DerivedClass Display"); }

}

```

Таким образом, производные классы от `DerivedClass` не смогут переопределить метод `Display`.

Если у вас есть дополнительные вопросы или нужна более подробная информация, дайте знать!

public IEnumerable<TransportVehicle> GetEnumerator()

{

return vechiles.GetEnumerator();

}

public void CopyTo(TransportVehicle[] array, int arrayIndex) => vechiles.CopyTo(array, arrayIndex);

public TransportVechile this[int index]

{

get => vechiles[index];

set => vechiles[imdex] = value

}

public int IndexOf(TransportVehicle item) => vechiles.IndexOf(item);

public int CompareTo(PortController other) => other == null ? 1 : Count.CompareTo(other.Count);

class PortController : ICollection<TransportVehicle>, IList<TransportVehicle>, IComparable<PortController>, ICloneable, IDisposable