ТЕОРИЯ

Наследование − это механизм получения нового класса на основе уже существующего. Формирование иерархии и повторное использование кода.

Полиморфизм - заключается в подмене реализация одного и того же метода. Типичной фразой полиморфизм можно объяснить фразой "Один интерфейс, множество реализаций"

Статический - перегрузка методов и операторов

Динамический - наследование и переопределение методов. Использование интерфейсов

Инкапсуляция – скрытие реализации объекта и доступ к нему только через интерфейсы или методы.

Абстракция – представление объекта только с помощью основных характеристик.

Интерфейс – правила, которые определяют набор методов, свойств, событий, которые должны быть реализованы классами.

"virtual" используется для объявления методов, свойств и индексаторов в базовом классе, которые могут быть переопределены в производных классах. Это позволяет реализовать полиморфизм и изменить поведение метода, свойства или индексатора в производных классах. Правила: параметры совпадают, не статик и не абстракт,

В реальности не существует геометрической фигуры как таковой. И для описания подобных сущностей, которые не имеют конкретного воплощения, предназначены абстрактные классы. Абстрактный класс похож на обычный класс. Он также может иметь переменные, методы, конструкторы, свойства. Единственное, что при определении абстрактных классов используется ключевое слово abstract. Но главное отличие абстрактных классов от обычных состоит в том, что мы НЕ можем использовать конструктор абстрактного класса для создания экземпляра класса.

Абстрактные члены классов не должны иметь модификатор private. Производный класс обязан переопределить и реализовать все абстрактные методы и свойства, которые имеются в базовом абстрактном классе(override). Если класс имеет хотя бы один абстрактный, то он должен быть определен как абстрактный.

1. Определение иерархии и композиции классов:
   * Иерархия классов представляет собой организацию классов в виде древовидной структуры, где классы на более высоком уровне общности наследуются от классов на более низком уровне. Например, можно создать иерархию классов "Фигура" -> "Прямоугольник" -> "Квадрат", где каждый класс наследует свойства и методы от предыдущего класса в иерархии.
   * Композиция классов представляет собой отношение, когда один класс содержит экземпляры других классов в качестве своих членов. Например, класс "Автомобиль" может содержать объекты классов "Двигатель", "Колесо" и "Салон" в качестве своих членов.
2. Интерфейсы и абстрактные классы:
   * Интерфейс представляет собой контракт, который определяет набор методов и свойств, которые должны быть реализованы классами, подписывающимися на этот интерфейс.
   * Абстрактный класс является базовым классом, который не может быть инстанциирован, но может содержать абстрактные и виртуальные члены. Абстрактные члены не имеют реализации в базовом классе и должны быть реализованы в производных классах.
3. Ключевое слово sealed:
   * Ключевое слово sealed используется для запрета наследования от класса или переопределения методов в производных классах. Класс, отмеченный как sealed, является конечным в иерархии наследования.
4. Одноименные методы в интерфейсах и абстрактных классах:
   * Если в интерфейсе и абстрактном классе определены методы с одинаковыми именами, то класс, реализующий интерфейс, должен предоставить реализацию для обоих методов.
5. Работа с объектами через ссылки на абстрактные классы и интерфейсы:
   * Создание объектов различных классов и использование ссылок на абстрактные классы и интерфейсы позволяет обращаться к объектам общими методами и свойствами, определенными в абстрактных классах и интерфейсах. Для идентификации типов объектов можно использовать операторы is или as.

Вопросы

1. Статические классы используются для группировки методов и свойств, которые не требуют создания экземпляра класса для их вызова. Они могут быть использованы для организации утилитарных функций, математических операций, расширений и других подобных задач.

2. Статический класс может содержать только статические методы, статические свойства, статические поля и статические события. Он не может иметь экземпляров и не может быть унаследован.

3. Базовый класс (родительский класс) - это класс, от которого производный класс наследует свойства и методы. Производный класс (дочерний класс) - это класс, который наследует свойства и методы от базового класса и может добавлять свои собственные свойства и методы.

4. Ключевое слово "base" используется в производном классе для обращения к членам базового класса. Оно может использоваться для вызова конструктора базового класса или для доступа к методам и свойствам базового класса, скрытым или переопределенным в производном классе.

5. Основная задача наследования - обеспечить повторное использование кода и создать иерархию классов, где производные классы наследуют свойства и методы от базовых классов. Наследование позволяет создавать специализированные классы на основе уже существующих классов и вносить изменения или добавлять новую функциональность без необходимости повторного написания кода.

6. Если объект производного класса создан, то он не может напрямую вызывать методы базового класса, если такого метода нет в производном классе. Однако, если метод в базовом классе объявлен с модификатором доступа `protected`, то объект производного класса может получить доступ к этому методу через наследование.

7. Объявление конструктора без аргументов для производного класса B, который будет вызывать конструктор без аргументов базового класса A, выглядит следующим образом:

```csharp

class B : A

{

public B() : base()

{

// Код конструктора производного класса B

}

}

```

8. Полиморфизм - это возможность объектов разных классов обладать одним и тем же интерфейсом и вести себя одинаково, при этом каждый класс может иметь свою собственную реализацию методов этого интерфейса. Примером полиморфизма может быть использование базового класса для работы с объектами разных производных классов, вызывая одинаковые методы у этих объектов через общий интерфейс.

```csharp

class Shape

{

public virtual void Draw()

{

Console.WriteLine("Drawing a shape");

}

}

class Circle : Shape

{

public override void Draw()

{

Console.WriteLine("Drawing a circle");

}

}

class Square : Shape

{

public override void Draw()

{

Console.WriteLine("Drawing a square");

}

}

Shape shape1 = new Circle();

Shape shape2 = new Square();

shape1.Draw(); // Вывод: "Drawing a circle"

shape2.Draw(); // Вывод: "Drawing a square"

```

9. Виртуальные функции в языке C# предназначены для реализации полиморфизма. Они позволяют производным классам переопределить методы базового класса и предоставляют механизм динамического связывания вызовов методов на этапе выполнения программы. Виртуальные функции объявляются с использованием ключевого слова `virtual` в базовом классе и могут быть переопределены в производных классах с помощью ключевого слова `override`.

10. Переменные с модификатором `protected` доступны только внутри класса, в производных классах и в том же самом пакете (assembly). Они не доступны извне класса или пакета.

11. Переменные с модификатором `private` не наследуются производными классами и не доступны извне класса, в котором они объявлены. Они являются закрытыми и доступны только в пределах того же класса.

12. В C# ключевое слово `as` используется для явного приведения типов объектов. Если приведение типов невозможно, то возвращается значение `null`. Ключевое слово `is` используется для проверки, является ли объект экземпляром определенного типа. Оно возвращает значение `true`, если объект является экземпляром указанного типа, и `false` в противном случае.

13. Нет, C# не поддерживает множественное наследование классов, т.е. класс не может наследовать от нескольких классов. Однако, множественное наследование интерфейсов поддерживается.

14. В C# нельзя явно запретить наследование от класса. Возможно, есть некоторые косвенные способы ограничить наследование, например, объявить класс с `private` конструктором, чтобы предотвратить создание экземпляров производного класса.

15. В C# нельзя разрешить наследование класса, но запретить перекрытие метода. Если метод в базовом классе не объявлен как `virtual` или `abstract`, то его нельзя переопределить в производном классе.

16. Абстрактный класс - это класс, который содержит один или несколько абстрактных методов. Абстрактные методы не имеют реализации в абстрактном классе и должны быть переопределены в производных классах. Абстрактные классы не могут быть инстанциированы, они служат только в качестве базовых классов для других классов.

17. Класс должен быть объявлен абстрактным, если в нем содержится хотя бы один абстрактный метод. Абстрактные классы предназначены для предоставления общего интерфейса и определения базовой функциональности, оставляя детали реализации для производных классов.

18. Разница между абстрактными и виртуальными классами заключается в том, что абстрактные классы могут содержать абстрактные методы, которые должны быть переопределены в производных классах, а виртуальные классы не содержат абстрактных методов и их методы могут быть переопределены, но не обязательно.

Разница между виртуальными и абстрактными методами заключается в реализации. Виртуальные методы имеют реализацию в базовом классе и могут быть переопределены в производных классах, а абстрактные методы не имеют реализации в абстрактном классе и должны быть

19. Виртуальными могут быть методы и свойства класса.

20. Интерфейс - это контракт, определяющий набор методов и свойств, которые должны быть реализованы классами. Он определяет только сигнатуры методов и свойств, но не их реализацию.

21. Интерфейс может содержать методы, свойства, индексаторы и события. Он также может наследовать другие интерфейсы.

22. Для работы с объектом через унаследованный интерфейс, нужно создать экземпляр класса, реализующего этот интерфейс, и использовать его для вызова методов и доступа к свойствам, определенным в интерфейсе.

23. Пример явной реализации интерфейса:

```csharp

interface IExample

{

void Method();

}

class MyClass : IExample

{

void IExample.Method()

{

Console.WriteLine("Explicit implementation of IExample.Method");

}

}

// Использование явной реализации

MyClass obj = new MyClass();

((IExample)obj).Method(); // Вывод: Explicit implementation of IExample.Method

```

24. Модификатор видимости не указывается для методов интерфейса, потому что все методы интерфейса являются public и доступны для всех классов, реализующих интерфейс.

25. Да, можно наследовать от нескольких интерфейсов. Класс может реализовывать любое количество интерфейсов.

26. Основные отличия между интерфейсом и абстрактным классом:

- Интерфейс определяет только сигнатуры методов и свойств, а абстрактный класс может содержать и реализацию методов.

- Класс может реализовывать несколько интерфейсов, но может наследоваться только от одного абстрактного класса.

- Интерфейс может быть реализован как классами, так и структурами, а абстрактный класс может быть унаследован только другими классами.

27. Стандартные интерфейсы используются для предоставления общего функционала:

- ICloneable позволяет создавать копию объекта.

- IComparable позволяет сравнивать объекты для определения их относительного порядка.

- IComparer позволяет выполнять сравнение объектов на основе пользовательского критерия.

- IEnumerable предоставляет возможность перечисления элементов коллекции.

28. В приведенном фрагменте листинга не содержится ошибка.

29. Результат выполнения фрагмента будет:

3 4

30. Результат выполнения фрагмента будет:

A B

31. Ответ: M4 – интерфейс или класс.

32. Верное присваивание для объектов, определенных в листинге, это:

1) b = a;

33. Если раскомментировать строчку 1, то будет выведено:

1) A B

34. Ошибка содержится в строке:

3) public string m() { return "C"; }//3

35. Приведенный фрагмент содержит ошибку, потому что абстрактный класс не может быть создан экземпляром. То есть, строка 4 содержит ошибку.

36. Ошибка компиляции может возникнуть в строке:

4) one = two;

37. В результате выполнения фрагмента листинга будет выведено:

3) new G

Лаб5

Теория

1) Перечисление (enum) - это тип данных в языках программирования, который позволяет определить набор именованных значений. Перечисления часто используются для создания наборов констант, которые могут быть использованы в программе. Каждому значению в перечислении автоматически назначается целочисленное значение, начиная с 0, и увеличивается на 1 для каждого последующего элемента.

Пример использования перечисления:

`````csharp

enum DaysOfWeek

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

DaysOfWeek today = DaysOfWeek.Monday;

Console.WriteLine(today); // Выводит: "Monday"

```

Структура (struct) - это тип данных, который позволяет объединить различные типы данных в одно логическое целое. В отличие от класса, структура является значимым типом и передается по значению, а не по ссылке.

Пример использования структуры:

````csharp

struct Point

{

public int X;

public int Y;

}

Point p;

p.X = 10;

p.Y = 20;

Console.WriteLine($"X: {p.X}, Y: {p.Y}"); // Выводит: "X: 10, Y: 20"

```

2) Ключевое слово `partial` позволяет разделить определение класса на несколько частей, которые могут быть определены в разных файлах. Компилятор автоматически объединяет эти части в один класс при компиляции. Это полезно, когда несколько разработчиков работают над одним классом или когда класс очень большой и сложный.

Пример использования partial-класса:

\*\*File1.cs\*\*

````csharp

partial class MyClass

{

public void Method1()

{

// Реализация метода 1

}

}

```

\*\*File2.cs\*\*

````csharp

partial class MyClass

{

public void Method2()

{

// Реализация метода 2

}

}

```

\*\*Program.cs\*\*

````csharp

MyClass obj = new MyClass();

obj.Method1();

obj.Method2();

```

Компилятор объединит определения из `File1.cs` и `File2.cs`, и код в `Program.cs` будет работать с полным определением класса `MyClass`.

3) Класс-контейнер - это класс, который служит для хранения и управления другими объектами. Класс-контейнер может использовать список или массив для хранения объектов, а также предоставлять методы для добавления, удаления и получения объектов из контейнера.

Пример класса-контейнера:

````csharp

class Container<T>

{

private List<T> items = new List<T>();

public void Add(T item)

{

items.Add(item);

}

public void Remove(T item)

{

items.Remove(item);

}

public T Get(int index)

{

return items[index];

}

public void Print()

{

foreach (T item in items)

{

Console.WriteLine(item);

}

}

}

// Пример использования класса-контейнера

Container<string> container = new Container<string>();

container.Add("Item 1");

container.Add("Item 2");

container.Print(); // Выводит: "Item 1", "Item 2"

```

4) Управляющий класс-контроллер (Controller) - это класс, который управляет объектом-контейнером (Container) и реализует определенные запросы и операции с этим контейнером. В зависимости от варианта, контроллер может использовать стандартные интерффейсы (например, IComparable, ICloneable) для дополнительной функциональности.

Пример управляющего класса-контроллера:

````csharp

class Controller

{

private Container<object> container = new Container<object>();

public void AddObject(object obj)

{

container.Add(obj);

}

public void RemoveObject(object obj)

{

container.Remove(obj);

}

public void PrintObjects()

{

container.Print();

}

// Реализация запросов по варианту

public object FindObject(Predicate<object> predicate)

{

return container.Find(predicate);

}

public object CloneObject(object obj)

{

if (obj is ICloneable)

{

ICloneable cloneable = (ICloneable)obj;

return cloneable.Clone();

}

return null;

}

public int CompareObjects(object obj1, object obj2)

{

if (obj1 is IComparable)

{

IComparable comparable = (IComparable)obj1;

return comparable.CompareTo(obj2);

}

return 0;

}

}

// Пример использования управляющего класса-контроллера

Controller controller = new Controller();

controller.AddObject("Item 1");

controller.AddObject("Item 2");

controller.PrintObjects(); // Выводит: "Item 1", "Item 2"

object foundObject = controller.FindObject(item => item.ToString() == "Item 1");

object clonedObject = controller.CloneObject(foundObject);

int comparisonResult = controller.CompareObjects(clonedObject, "Item 2");

```

В этом примере управляющий класс-контроллер `Controller` управляет объектом-контейнером `Container` и реализует запросы по варианту, такие как поиск объекта, клонирование объекта и сравнение объектов. В случае, если объект поддерживает интерфейсы `ICloneable` или `IComparable`, контроллер использует их для выполнения соответствующих операций.

Вопросы

1. Основное отличие между классом и структурой заключается в том, как они хранятся и передаются. Классы являются ссылочными типами и передаются по ссылке, тогда как структуры являются значимыми типами и передаются по значению. Классы могут быть унаследованы другими классами, поддерживают наследование и полиморфизм, а структуры не могут быть унаследованы и не поддерживают наследование или полиморфизм.

2. В структуре могут быть поля, свойства, методы, события и конструкторы. Они также могут реализовывать интерфейсы. Однако структуры не могут иметь явно объявленного безпараметрического конструктора (default constructor) и не могут унаследовать другую структуру или класс.

3. Перечисление (enum) - это тип данных, который определяет набор именованных констант. Перечисления используются для определения набора допустимых значений. Например, можно определить перечисление `DaysOfWeek`, чтобы представить дни недели:

```csharp

enum DaysOfWeek

{

Monday,

Tuesday,

Wednesday,

Thursday,

Friday,

Saturday,

Sunday

}

```

Перечисления предоставляют удобный способ работы с ограниченным набором значений и могут использоваться вместо числовых констант.

4. Некоторые стандартные интерфейсы .NET:

- IComparable: Позволяет объектам сравниваться с другими объектами по определенному критерию сравнения.

- ICloneable: Позволяет объекту создать глубокую копию самого себя.

- IDisposable: Предоставляет механизм для освобождения ресурсов, используемых объектом.

- IEnumerable: Предоставляет возможность перечисления элементов коллекции.

- INotifyPropertyChanged: Позволяет объекту уведомлять другие объекты о изменении своих свойств.

5. Интерфейс IComparable используется для определения порядка сортировки объектов. Класс, реализующий интерфейс IComparable, должен предоставить реализацию метода CompareTo, который сравнивает текущий объект с другим объектом и возвращает значение, указывающее их относительный порядок.

6. Интерфейс ICloneable используется для создания глубокой копии объекта. Класс, реализующий интерфейс ICloneable, должен предоставить реализацию метода Clone, который создает и возвращает копию текущего объекта.

7. Полиморфизм - это возможность объектов разных типов обрабатываться единообразно. Формы полиморфизма включают:

- Параметрический полиморфизм (обобщения): позволяет создавать обобщенные типы и методы, которые могут работать с различными типами данных.

- Наследование: позволяет объектам дочерних классов использовать методы и свойства родительских классов, при этом их можно переопределить или расширить.

- Полиморфизм через интерфейсы: позволяет объектам разных классов использовать одинаковые интерфейсы для взаимодействия с ними.

8. Виртуальные методы в классе позволяют их переопределять в производных классах. Они предоставяют базовую реализацию метода в базовом классе, которая может быть изменена или расширена в производных классах. Зачастую виртуальные методы используются для создания полиморфизма и обеспечения гибкости в проектировании классов.

9. Чтобы запретить переопределение методов в классе, можно использовать ключевое слово `sealed` в объявлении метода. Если метод объявлен с модификатором `sealed`, то он не может быть переопределен в производных классах. Пример:

```csharp

public class MyBaseClass

{

public sealed virtual void MyMethod()

{

// Реализация метода

}

}

public class MyDerivedClass : MyBaseClass

{

// Попытка переопределения метода MyMethod вызовет ошибку компиляции

// так как метод в базовом классе объявлен как sealed

}

```