**Наследование**

Наследование в объектно-ориентированном программировании (ООП) — это механизм, который позволяет создавать новые классы на основе уже существующих, наследуя их свойства и методы. Это позволяет повторно использовать код, расширять функциональность и обеспечивать логическую структуру кода.

**Основные концепции наследования:**

Базовый класс (Superclass, Parent class):

Это класс, от которого наследуют другие классы.

Содержит общие свойства и методы, которые будут унаследованы производными классами.

**Производный класс (Subclass, Child class):**

Это класс, который наследует свойства и методы базового класса.

Может добавлять свои свойства и методы или переопределять унаследованные.

**Переопределение методов (Method Overriding):**

Производный класс может переопределить методы базового класса, чтобы изменить их поведение.

Для этого используется ключевое слово override.

**Модификаторы доступа:**

Модификаторы, такие как public, protected, и private, контролируют доступ к членам класса.

protected делает члены доступными в производных классах, но не вне их.

**(примеры 1 - 6);**

**Описание примера 4:**

Модификатор private ограничивает доступ к членам класса, делая их доступными только внутри самого класса. При наследовании такие члены не видны в производных классах.

Объяснение:

В классе Device поле serialNumber и метод ShowSerialNumber объявлены как private, что делает их недоступными в классе Phone.

Попытка вызвать или получить доступ к serialNumber и ShowSerialNumber из класса Phone приведет к ошибке компиляции.

Однако, метод DisplayInfo, который публичный, может вызывать ShowSerialNumber внутри класса Device.

**Описание примера 5**

Модификатор protected позволяет доступ к членам класса внутри самого класса и всех его производных классов.

Объяснение:

Поля Model и Year, а также метод DisplayCarInfo объявлены как protected в классе Car.

Они доступны в производном классе ElectricCar, что позволяет производному классу вызывать DisplayCarInfo и использовать защищенные поля.

**Описание примера 6:**

В этом примере оба модификатора используются для иллюстрации их взаимодействия.

Объяснение:

В классе Staff поле staffID объявлено как private, и оно недоступно в производном классе Teacher.

Поле Position, объявленное как protected, доступно в производном классе Teacher.

Поле subject, объявленное как private в классе Teacher, недоступно за пределами этого класса.

**Преобразование типов**

**Восходящее преобразование (Upcasting)**

Upcasting — это процесс преобразования объекта производного класса в объект базового класса. Это всегда безопасное преобразование, и оно может быть выполнено неявно, поскольку производный класс содержит все члены базового класса. **(пример 7).**

**Описание примера 7:**

Переменная myAnimal типа Animal указывает на объект Dog.

После восходящего преобразования (upcasting) переменная может использовать только те члены, которые определены в базовом классе Animal.

**Нисходящее преобразование (Downcasting)**

Downcasting — это процесс преобразования объекта базового класса в объект производного класса. Такое преобразование требует явного указания, поскольку может быть небезопасным, если объект не является экземпляром производного класса. Если преобразование невозможно, оно вызовет исключение InvalidCastException. **(пример 8).**

**Описание примера 8:**

Переменная myAnimal типа Animal фактически указывает на объект Dog.

Перед выполнением нисходящего преобразования (downcasting) с использованием оператора is проверяется, действительно ли объект принадлежит типу Dog.

Если преобразование успешно, объект типа Dog может использовать все члены класса Dog, включая метод Bark.

**Виртуальные методы и свойства**

Виртуальные методы и свойства в C# позволяют базовым классам определять функциональность, которая может быть изменена или расширена в производных классах. Это одна из ключевых концепций объектно-ориентированного программирования (ООП), обеспечивающая гибкость и полиморфизм.

**Виртуальные методы**

Виртуальный метод — это метод, который объявлен в базовом классе с использованием ключевого слова virtual и может быть переопределён (override) в производном классе. Это позволяет производному классу изменить поведение метода, унаследованного от базового класса.

(пример 9).

Обьяснение примера 9:

В базовом классе Shape метод CalculateArea объявлен как виртуальный (virtual), что позволяет производным классам его переопределить.

В классе Square и Circle этот метод переопределён с использованием ключевого слова override, чтобы выполнять соответствующие вычисления для каждой фигуры.

**Виртуальные свойства**

Виртуальные свойства аналогичны виртуальным методам, но используются для определения характеристик, которые могут быть изменены в производных классах. Они позволяют переопределять геттеры и сеттеры для получения и установки значений свойств. (пример 10)

Обьяснение примера 10:

В базовом классе Person свойства Name и Age объявлены как виртуальные (virtual).

В производном классе Employee эти свойства переопределены (override), чтобы добавить дополнительную логику при их получении и установке.

Свойство Age в классе Employee изменено так, что возраст не может быть меньше 18 лет, а свойство Name возвращает имя с префиксом "Сотрудник".

**Скрытие методов и свойств**

Скрытие методов и свойств в C# — это процесс, при котором производный класс может предоставить собственную реализацию метода или свойства, которая скрывает (заменяет) член с таким же именем, унаследованный от базового класса. В отличие от переопределения (override), скрытие не предоставляет полиморфного поведения и не связано с виртуальными методами или свойствами. Для скрытия используется ключевое слово new.

**Скрытие методов**

Когда метод скрыт в производном классе, метод базового класса с тем же именем и сигнатурой становится недоступным через ссылку на производный класс, если не использовать явное приведение типа. (пример 11)

Описание примера 11:

В базовом классе Animal метод MakeSound определен без использования ключевого слова virtual.

В классе Dog метод MakeSound скрывает метод базового класса с использованием ключевого слова new.

Метод MakeSound из класса Dog заменяет (скрывает) метод MakeSound из класса Animal, когда он вызывается через ссылку на объект Dog.

Однако, если объект Dog приводится к типу Animal, вызывается метод MakeSound из базового класса.

**Скрытие свойств**

Скрытие свойств работает аналогично скрытию методов. Производный класс может скрыть свойство базового класса, предоставив свою собственную реализацию с тем же именем. (пример 12).

Описание примера 12:

В базовом классе Vehicle определено свойство Speed.

В производном классе Car это свойство скрывается с использованием ключевого слова new.

При доступе к объекту Car через ссылку на производный класс используется скрытое свойство Speed из класса Car.

Однако, если объект Car приводится к типу Vehicle, используется оригинальное свойство Speed из базового класса.

**Различия между скрытием и переопределением**

Переопределение (override):

Применяется к виртуальным методам или свойствам.

Поддерживает полиморфизм, что означает, что вызов метода или свойства определяется на основе фактического типа объекта, даже если он используется через ссылку на базовый тип.

Скрытие (new):

Применяется к любым членам класса, включая методы и свойства, которые не являются виртуальными.

Не поддерживает полиморфизм. Вызов скрытого члена зависит от типа ссылки, а не фактического типа объекта.

**Абстрактные классы**

Абстрактные классы в C# — это классы, которые не могут быть созданы в виде объектов и предназначены для предоставления общей функциональности другим классам через наследование. Абстрактные классы часто используются, когда необходимо определить некоторый общий функционал, но конкретная реализация этого функционала должна быть предоставлена производными классами.

Основные особенности абстрактных классов:

Абстрактные методы: Абстрактный класс может содержать абстрактные методы — методы без тела, которые должны быть реализованы в производных классах. Эти методы объявляются с ключевым словом abstract и не могут содержать реализацию.

Частично реализованный функционал: Абстрактный класс может содержать как абстрактные, так и полностью реализованные методы и свойства. Это позволяет комбинировать общую реализацию с требованиями к конкретной реализации в производных классах.

Наследование: Абстрактные классы предназначены для наследования. Производный класс, который наследует абстрактный класс, должен реализовать все его абстрактные методы и свойства.

Запрещение создания объектов: Нельзя создать экземпляр абстрактного класса напрямую, то есть нельзя использовать оператор new для создания объекта абстрактного класса.

Рассмотрим пример, в котором абстрактный класс Shape предоставляет общую структуру для различных типов геометрических фигур, таких как Circle и Rectangle. (пример 13).

Описание примера 13:

Абстрактный класс Shape определяет общую структуру для всех фигур. Он содержит абстрактное свойство Area и абстрактный метод CalculatePerimeter, которые должны быть реализованы в производных классах.

Класс Circle и класс Rectangle наследуют от Shape и предоставляют свои реализации методов и свойств, которые были объявлены в Shape как абстрактные.

Метод DisplayInfo в абстрактном классе Shape предоставляет общую реализацию для отображения информации о фигуре. Этот метод может быть использован в любом производном классе без необходимости его переопределения.

**Преимущества использования абстрактных классов:**

Упрощение кода: Абстрактные классы позволяют сократить дублирование кода за счет реализации общей функциональности в одном месте, которую затем могут использовать все производные классы.

Полиморфизм: Абстрактные классы позволяют использовать полиморфизм. Вы можете работать с объектами различных производных классов через ссылки на абстрактный класс, что упрощает разработку и поддержку кода.

Ясная архитектура: Абстрактные классы позволяют разработчикам задавать общие шаблоны для семейств классов, предоставляя основу для создания сложных иерархий.