Турсунов Баходурхон

#18. .NET Object-Oriented Programming (OOP). Access Modifier. Interceptors.

**1. Каковы основные атрибуты доступа в C# и для чего он и используются?**

Все поля, методы, структуры, интерфейсы и остальные компоненты класса имеют модификаторы доступа. Модификаторы доступа позволяют задать допустимую область видимости для компонентов класса. То есть модификаторы доступа определяют контекст, в котором можно употреблять данную переменную или метод.

**2. Какой вид доступа предоставляет атрибут "protected" и когда он оправдан?**

Атрибут доступа **Protected** предоставляет доступ к членам класса внутри самого класса, а также в производных (наследуемых) классах. Это означает, что член с атрибутом protected виден внутри класса, где он объявлен и в любом другом классе, который наследуется от этого класса.

Когда атрибут оправдан?

* Наследование: используется, когда мы хотим, чтобы член класса был доступен в производных классах для повторного использования кода. Например, базовый класс может содержать защищенный метод, который предоставляет общую функциональность, которую наследники могут использовать.
* Расширение функциональности. Позволяет наследникам изменять или расширять поведение родительского класса, предоставляя доступ к его защищенным членам.
* Сокрытие деталей реализации. Используется для сокрытия деталей реализации от внешнего мира, и предоставляет доступ для наследников, что обеспечивает более гибкую и управляемую архитектуру.

**3. В чем смысл атрибута "private protected" и при каких условиях его следует использовать?**

Атрибут доступа private protected предоставляет интересную комбинацию доступа, которая позволяет члену класса быть видимыми только внутри самого класса и в его производных классах в переделах той же сборки.

**public** **class** BaseClass  
{  
 **private** **protected** int PrivateProtectedField;  
}  
  
**public** **class** DerivedClass1 : BaseClass  
{  
 **public** void AccessPrivateProtectedField()  
 {  
 // Доступ к private protected полю из производного класса внутри той же сборки  
 PrivateProtectedField = 42;  
 }  
}  
  
**public** **class** DerivedClass2  
{  
 **public** void AccessPrivateProtectedField(BaseClass instance)  
 {  
 // Недопустимо: PrivateProtectedField не видно из этого класса  
 // instance.PrivateProtectedField = 42;  
 }

}

4. Чем "частный" доступ отличается от других атрибутов доступа и в каких случаях его следует использовать?

Атрибут доступа private обозначает, что член(поле, метод, свойство и т.д.) доступен только внутри того же класса, в котором он объявлен. Он является самым строгим уровнем доступа и не предоставляет доступ к члену из других классов, даже если они являются производным от того же базового класса.

Особенности **private:**

* Ограниченный доступ обеспечивает максимально возможную защиту для членов класса, предоставляя доступ только внутри того же класса. Это предотвращает прямой доступ из других классов.
* Private играет важную роль в инкапсуляции, скрывая детали реализации от внешнего мира. Это позволяет изменять внутреннюю реализацию класса, не затрагивая внешний код.
* Использование private предотвращает прямой доступ к внутренним данным класса из других частей программы. Это способствует соблюдению принципов инкапсуляции и сокрытия деталей реализации.

**5. Опишите "внутренний" доступ и укажите, когда его следует использовать.**

Атрибут доступа internal обозначает, что член(класс, метод, свойство и т.д.) доступен только в пределах текущей сборки. Это означает, что любой класс внутри той же сборки может получить доступ к члену с атрибутом internal, но член остается недоступным для кода в других сборках.

Его следует использовать, когда мы хотим обеспечить контроль доступа в пределах одной сборки, что делает его полезным для создания модульного и согласованного кода в рамках больших проектов. Классы и члены, которые должны взаимодействовать между собой, но не предназначены для использования в других сборках, могут быть помечены как Internal.

Internal полезен при создании библиотек и компонентов, где некоторые классы или функциональность предназначены только для внутреннего использования.

Practice:

1. Создайте собственные классы и структуры, используя различные атрибуты доступа. Создайте методы и поля с различными атрибутами доступа, чтобы продемонстрировать различия в их доступности в разных частях вашего кода. Попробуйте вызвать или получить доступ к членам класса из разных областей вашего проекта, чтобы увидеть, как атрибуты доступа влияют на доступ к членам класса.