Турсунов Баходурхон

#14. .NET Object-Oriented Programming (OOP). Anonymous Types. Tuples.

1. **Объясните концепцию анонимных типов в C#. В чем основное преимущество их использования и каковы ограничения при создании анонимных типов?**

Анонимные типы позволяют создать объект с некоторым набором свойств без определения класса. Анонимный тип определяется с помощью ключевого слова **var** и инициализатора объектов:

var user = new { Name = "Tom", Age = 34 };  
 Console.WriteLine(user.Name);

В данном случае user – это объект анонимного типа, у которого определены два свойства Name и Age. И мы также можем использовать его свойства, как и у обычных объектов классов. Однако тут есть ограничение – свойства анонимных типов доступны только для чтения.

При создании анонимных типов мы не можем в дальнейшем переопределить наш объект, т. е. этот тип доступен только для чтения.

При использовании анонимных типов есть ряд ограничений:

* Анонимные типы не могут содержать методы или другие пользовательские члены, кроме автоматически созданных свойств.
* Анонимные типы не могут наследоваться от других типов.
* Поскольку анонимные типы не имеют явного имени типа, использование их в качестве параметров методов с ключевыми словами **out** или **ref** невозможно
* Анонимные типы ограничены областью видимости там, где они были созданы.

1. **Как C# обрабатывает информацию об именовании и типе для анонимных типов, и почему к имени типа нельзя получить доступ в коде приложения?**

Во время компиляции компилятор сам будет создавать для него имя типа и использовать это имя при обращении к объекту. Анонимные типы имеют имя наподобие “<>f\_AnonymousType0`2”. Для исполняющей среды CLR анонимные типы будут также, как и классы, представлять ссылочный тип. Если в программе используются несколько объектов анонимных типов с одинаковым набором свойств, то для них компилятор создаст одно определение анонимного типа:

var user = new { Name = "Tom", Age = 34 };  
 var student = new { Name = "Alice", Age = 21 };  
 var manager = new { Name = "Bob", Age = 26, Company = "Microsoft" };  
 Console.WriteLine(user.GetType().Name); // <>f\_\_AnonymousType0'2  
 Console.WriteLine(student.GetType().Name); // <>f\_\_AnonymousType0'2  
 Console.WriteLine(manager.GetType().Name); // <>f\_\_AnonymousType1'3

Почему нельзя получить доступ к имени типа в коде приложения? Это связано с тем, что анонимные типы предназначены для использования в локальных областях и имеют ограниченную область видимости. Их цель – обеспечить удобство в некоторых сценариях, но они не предназначены для длительного жизненного цикла или использования в широком контексте.

Имя анонимного типа является деталью реализации, и компилятор генерирует его автоматически. Если бы мы могли получить доступ к этому имени, это нарушило бы принцип анонимности и могло бы привести к несогласованности в случае изменения кода или добавления анонимных типов.

1. **Что такое кортежные типы в C# и чем они отличаются от обычных структур данных? Приведите базовый пример создания кортежа с именованными элементами.**

В C# кортежи (tuples) представляют собой структуры данных, позволяющие объединить несколько элементов различных типов в один составной объект.

Кортежи могут содержать элементы различных типов данных. Например, кортеж может включать в себя целое число, строку и др.

Элементы кортежа доступны только для чтения (т.е. они являются immutable). Если необходимо изменить значение, создается новый кортеж. Это способствует безопасности и избеганию неожиданных изменений данных.

Базовый пример создание кортежа с именованными элементами:

Tuple<int, double, float> tuple1 = new(1, 1.0d, 1.4f);

Practice:

1. Создайте программу на C#, моделирующую простую систему заказов электронной коммерции. Используйте анонимные типы для представления заказов клиентов. Каждый заказ должен иметь такие свойства, как OrderId, CustomerName, ProductName и Price. Инициализируйте коллекцию объектов анонимного типа и отобразите их.

2. namespace HW14.\_1

3. {

4. internal class Program

5. {

6. static void Main(string[] args)

7. {

8. var Order = new[]

9. {

10. new { OrderId = 1, CustomerName = "Saint John", ProductName = "Mac 14 pro", Price = 2999 },

11. new { OrderId = 2, CustomerName = "Sam Jack", ProductName = "Mac 13 pro", Price = 1999 },

12. new { OrderId = 3, CustomerName = "Antony Bonderos", ProductName = "Mac 12 pro", Price = 999 }

13. };

14. Console.WriteLine("Orders:");

15. foreach (var order in Order)

16. {

17. Console.WriteLine($"OrderId: {order.OrderId}, CustomerName: {order.CustomerName}, ProductName: {order.ProductName}, Price: {order.Price}");

18. }

19. }

20. }

21. }

22.

<https://metanit.com/sharp/tutorial/3.20.php>