Турсунов Баходурхон

#11. .NET Object-Oriented Programming (OOP). Class. Properties (get, set, init). Read-only properties. Required properties.

**1. Объясните принципиальное различие между классом и объектом в объектно-ориентированном программировании. Приведите примеры, иллюстрирующие это различие.**

Класс – это абстрактное понятие, описывающее некую сущность, которая объединяет свойства (атрибуты) и действия (методы). Он представляет собой шаблон, по которому создаются объекты. Класс является неким чертежом, по которому нужно построить конкретный объект.

Объект – это конкретный экземпляр класса. Он имеет доступ ко всем атрибутам и методам, определенным в классе. Объект является реализацией класса. Он может быть создан, модифицирован и удален в течение выполнения программы.

Таким образом, основная разница между классом и объектом заключается в том, что класс представляет абстракцию, описывающую все возможные свойства и действия, которые могут иметь объекты данного класса, а объект – это конкретная реализация этого класса, с определенными значениями свойств и возможностью вызывать его методы

Перейдем к иллюстрационным примерам:

Посути класс представляет новый тип, который определяется пользователем. Класс определяется с помощью ключевого слова **class:**

static void Main(string[] args)

{

Person person = new Person(); // в этой части мы создали экземпляр класса **Person** ***person***

person.Print(); // а тут мы вызвали у этого экземпляра метод Print

}

class Person // мы создали класс Person и добавили поля и методы описывающие данный класс.

{

public string Name = "Tom";

public int Age = 22;

public void Print()

{

Console.WriteLine($"Name: {Name} Age: {Age}");

}

}

**2. Когда создается экземпляр класса, что хранится в переменной? Опишите процесс выделения памяти и присвоения ссылок.**

При создании экземпляра класса в переменной строкового типа хранится **null**  это говорит нам о том, что наша переменная **string Name** не ссылается ни на что, а переменная **int Age** равна 0.

После запуска данного кода, в куче выделилась память в размере 394КБ но он еще не применим. И после того как создается экземпляр класса и вызывается метод Print(), выделенная память чуть увеличилась в размере. И теперь переменная Name ссылается на адрес, для которого создалась память в управляемой куче (heap).

**3. Опишите концепцию свойств в C# и то, как они обеспечивают гибкий механизм для работы с данными. Приведите примеры сценариев, в которых свойства могут быть полезны.**

В C#, свойства представляют с собой механизм для доступа к данным класса. Они объединяют в себе преимущества полей класса и методов доступа, обеспечивая гибкий и контролируемый доступ к данным объекта. Свойства предоставляют интерфейс для чтения и записи значений, а их синтаксис похож на синтаксис полей, но позволяет выполнять дополнительные операции при чтении или записи значений

public class Person

{

private string name;

// Свойство Name с автоматическими геттером и сеттером

public string Name

{

get { return name; }

set { name = value; }

}

}

Этот код создает класс Person с приватным полем name и публичным свойством Name. Преимущество использования свойства вместо прямого доступа к полю заключается в том, что вы можете добавить логику или проверки при чтении или записи значенияю

**4. Различайте аксессоры свойств "get", "set" и "init". Как они влияют на доступность и модифицируемость данных?**

Аксессоры свойств get set и init управляют доступностью и модифицируемостью данных в свойствах классов.

Get (получение) – отвечает за чтение значения свойства. Если свойство имеет только аксессор get, то оно называется readonly (только для чтения). Это означает, что значение свойства можно только читать, и нельзя изменять.

Set (установка) – отвечает за установку значения свойства. Если свойство имеет только аксессор set, то оно называется writeonly (только для записи). Это означает, что значение свойства можно только устанавливать, но нельзя читать.

Init (инициализация) – введен в C# 9.0 и представляет возможность установки значения только во время инициализации объекта и не позволяет его изменять после этого. Часто используется в объектах, где требуется установить значение только один раз при создании объекта и предотвратить изменение этого значения в дальнейшем.

**5. Каково назначение модификатора "required modifier" в C# 11 и где он может быть применен? Объясните правила и ограничения, связанные с использованием модификатора required.**

Модификатор required указывает, что **поле** или **свойства** с этим модификатором обязательно должны быть инициализированы. Если не инициализировать какое-нибудь свойство, то будет ошибка.

Этот модификатор может быть применен, к примеру, при регистрации в приложение пользователь обязательно должен ввести свои имя и фамилию, а также год рождения и тп.

6. Обсудите роль атрибута "SetsRequiredMembers" и его последствия для типов с обязательными членами. Как он влияет на поведение конструктора?

SetsRequiredMembers указывает, что этот конструктор устанавливает все необходимые члены для текущего типа, и вызывающим его пользователям не нужно устанавливать какие-либо необходимые члены самостоятельно