Использование модификаторов доступа в C#

# Цель: научиться применять различные модификаторы доступа для управления видимостью классов и их членов, а также понять, как это влияет на инкапсуляцию и безопасность кода.

**Вопрос 1**. Какие есть модификаторы доступа в C#?

**public**: код в любой сборке может получить доступ к этому типу или члену. Уровень специальных возможностей содержащего типа определяет уровень доступности общедоступных членов типа.

**private**: только код, объявленный в том же или class struct может получить доступ к этому члену.

**protected**: только код в том же class или в производном class может получить доступ к этому типу или члену.

**internal**: доступ к этому типу или члену может получить только код в одной сборке.

**protected internal**: доступ к этому типу или члену может получить только код в той же сборке или в производном классе в другой сборке.

**private protected**: только код в одной сборке и в том же классе или производном классе может получить доступ к типу или члену.

file: доступ к типу или члену может получить только код в одном файле.

**Вопрос 2**. Вам надо определить в классе переменную, которая должна быть доступна из любого места в текущем проекте. Какой модификатор (или модификаторы, если их несколько) вы будете использовать?

Модификатор public.

**Вопрос 3**. В чем различие между модификаторами protected и private protected?

Модификатор protected позволяет получить доступ к вызывающему объекту в файле, классе, производном классе той же сборки, производном классе другой сборки.

**Вопрос 4**. Если классы и члены класса не имеют никаких модификаторов, какие модификаторы доступа к ним применяются по умолчанию?

Если классы и члены класса не имеют установленный модификатор доступа, то по дефолту им задается модификаторы доступа internal и private (для классов – internal, для членов классов – private).

**Вопрос 5**. Что выведет на консоль следующая программа и почему?

class Person

{

int age = 26;

string name = "Tom";

public Person(int age, string name)

{

this.age = age;

this.name = name;

}

}

class Program

{

static void Main(string[] args)

{

Person person = new Person(19, "Bob");

Console.WriteLine(person.name);

Console.ReadKey();

}

В коде будет ошибка компиляции, потому что поле name класса Person имеет модификатор доступа по умолчанию — private, что означает, что это поле не доступно извне класса. Поскольку в методе Main мы пытаемся обратиться к полю name объекта person, мы получим ошибку, которая укажет на то, что name недоступно.

**Свойства**

**Вопрос 1**. Что будет выведено на консоль в результате выполнения следующей программы?

class Person

{

private int age = 15;

public int Age

{

get { return age; }

set { }

}

}

class Program

{

static void Main(string[] args)

{

Person tom = new Person();

tom.Age = 25;

Console.WriteLine(tom.Age);

Console.ReadKey();

}

}

**Варианты ответов**:

1. 15
2. 25
3. 0
4. Программа не скомпилируется

**Ответ**: 15 (при выводе tom.Age, сработает только get, который возвращает текущее значение age, то есть 15)

**Вопрос 2**. Что будет выведено на консоль в результате выполнения следующей программы и почему?

class Person

{

internal string Name { get; set; } = "Bob";

}

class Program

{

static void Main(string[] args)

{

Person tom = new Person { Name = "Tom" };

Console.WriteLine(tom.Name);

Console.ReadKey();

}

}

На консоль выведется значение "Tom", потому что в конструкторе класса Person создан экземпляр объекта tom, и свойству Name было дано значение "Tom". Хотя у класса Person есть значение по умолчанию для свойства Name ("Bob"), при создании объекта tom это значение было переопределено. Поэтому при вызове Console.WriteLine(tom.Name) на экран выводится именно "Tom".

**Вопрос 3**. Что будет выведено на консоль в результате выполнения следующей программы и почему?

class Person

{

internal string Name { internal get; set; } = "Bob";

}

class Program

{

static void Main(string[] args)

{

Person tom = new Person { Name = "Tom" };

Console.WriteLine(tom.Name);

Console.ReadKey();

}

}

На консоль выведется значение "Tom", потому что когда создается объект tom типа Person, значение свойства Name устанавливается в "Tom". Затем после выполнения строки Console.WriteLine(tom.Name); программа выводит текущее значение свойства Name, которое равно "Tom".