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

using System;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

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

Employee empl = tom as Employee;

tom.Name = "Bob";

Console.WriteLine(empl.Name);

Console.ReadKey();

}

}

class Person

{

public string Name { get; set; }

}

class Employee : Person { }

}

В результате выведется исключение NullReferenceException, потому что объект tom (типа Person) не может быть приведен к типу Employee, и переменная empl станет null. При попытке доступа к empl.Name программа попытается обратиться к свойству Name у null, что приведет к исключению.

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

using System;

namespace HelloApp

{

class Program

{

static void Main(string[] args)

{

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

Person bob = tom;

Employee empl = bob as Employee;

tom.Name = "Bob";

Console.WriteLine(empl.Name);

Console.ReadKey();

}

}

class Person

{

public string Name { get; set; }

}

class Employee : Person { }

}

В результате выведется "Bob", потому что переменная tom ссылается на объект Employee, у которого свойство Name сначала устанавливается в "Tom". Затем в строке tom.Name = "Bob"; свойство Name обновляется на "Bob". Поскольку empl является ссылкой на тот же объект Employee, что и tom, вызов Console.WriteLine(empl.Name); выводит "Bob".

**Вопрос 3.** Есть два класса:

class Person

{

public string Name { get; set; }

}

class Employee : Person { }

Для преобразования объекта Person к типу Employee мы можем использовать следующие два способа:

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

Employee empl1 = tom as Employee;

Employee empl2 = (Employee)tom;

Какой из способов является более предпочтительным и почему?

Лучше будет использовать оператор as из-за его безопасности и возможности избежать неожиданного выхода программы при неудачном приведении типов.

**Виртуальные методы и свойства**

**Вопрос 1**. Какое ключевое слово применяется для переопределения виртуальных методов и свойств:

• overridable

• override

• overriding

• overriden

**Ответ**: override

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

class Person

{

public string Name { get; set; }

public virtual void Display()

{

Console.WriteLine($"Person {Name}");

}

}

class Employee : Person

{

public string Company { get; set; }

public override void Display()

{

Console.WriteLine($"Employee {Name}");

}

}

class Program

{

static void Main(string[] args)

{

Person person = new Employee { Name = "Sam", Company = "Microsoft" };

person.Display();

Console.ReadKey();

}

}

В результате выведется Employee Sam, потому что в классе Employee метод Display переопределён, и при вызове Display на объекте person, который на самом деле является экземпляром Employee, будет выполнен переопределённый метод из класса Employee. В итоге отображается имя "Sam" с префиксом "Employee".

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

class Person

{

public string Name { get; set; }

public virtual void Display()

{

Console.WriteLine($"Person {Name}");

}

}

class Employee : Person

{

public string Company { get; set; }

public override void Display()

{

Console.WriteLine($"Employee {Name}");

}

}

class Manager : Employee

{

public override void Display()

{

Console.WriteLine($"Manager {Name}");

}

}

class Program

{

static void Main(string[] args)

{

Person person = new Manager { Name = "Bob", Company = "Microsoft" };

Employee employee = (Employee)person;

employee.Display();

Console.ReadKey();

}

}

В результате выведется Manager Bob, потому что переменная person ссылается на объект типа Manager, который переопределяет метод Display. Когда мы вызываем employee.Display(), реальный тип объекта — Manager, поэтому вызывается именно этот переопределённый метод.

**Вопрос 4**. Что будет выведено на консоль в результате выполнения следующей программы и почему:

class Auto

{

protected internal virtual void Move()

{

Console.WriteLine("Auto is moving");

}

}

class Track : Auto

{

public override void Move()

{

Console.WriteLine("Track is moving");

}

}

class Program

{

static void Main(string[] args)

{

Track track = new Track();

Auto auto = track;

auto.Move();

Console.ReadKey();

}

}

В результате выведется Track is moving, потому что метод Move в классе Track переопределяет виртуальный метод Move из базового класса Auto. Когда вызывается auto.Move(), объект track (типа Track) вызывает свою собственную реализацию метода Move, что и приводит к выводу "Track is moving".

**Вопрос 5**. Для чего нужен модификатор sealed?

Модификатор sealed нужен для предотвращения дальнейшего наследования от класса. Если класс объявлен как sealed, то его нельзя использовать в качестве базового класса для других классов.

**Вопрос 6**. Что будет выведено на консоль в результате выполнения следующей программы и почему?

class Auto

{

public sealed void Display()

{

Console.WriteLine("Auto");

}

}

class Truck : Auto

{

public void DisplayInfo()

{

base.Display();

}

}

class Program

{

static void Main(string[] args)

{

Truck truck = new Truck();

truck.DisplayInfo();

Console.ReadKey();

}

}

В результате выведется Auto, потому что у класса Display используется модификатор sealed, что не позволяет переопределять его в производных классах.