**Test your knowledge**

1. What are the six combinations of access modifier keywords and what do they do?

* Public:can be accessed anywhere
* Private.: can be accessed same class
* Protected.: can be accessed current and child class
* Internal.: can be accessed current assembly
* Protected Internal.: can be accessed current class
* Private Protected.: can be accessed by types derived from the containing class, but only within its containing assembly

2. What is the difference between the static, const, and readonly keywords when applied

to a type member?

static:The static keyword is used to specify a static member, which means static members are common to all the objects and they do not tie to a specific object. This keyword can be used with classes, fields, methods, properties, operators, events, and constructors, but it cannot be used with indexers, destructors, or types other than classes.

const:Constant fields or local variables must be assigned a value at the time of declaration and after that, they cannot be modified.

readonly:A readonly field can be initialized either at the time of declaration or within the constructor of the same class. Therefore, readonly fields can be used for run-time constants.

3. What does a constructor do?

construct is used to create objects of class and initialize class members.

4. Why is the partial keyword useful?

The partial keyword indicates that other parts of the class, struct, or interface can be defined in the namespace

5. What is a tuple?

A tuple is a data structure that contains a sequence of elements of different data types

6. What does the C# record keyword do?

A record is a reference type and follows value-based equality semantics.

7. What does overloading and overriding mean?

Overloading: refers to in same class, same method, same access modifier but different input parameter

Overriding: refers to between base class, derived class, same method, same access modifier, same input parameter. Derived class can have different implementation for same method

8. What is the difference between a field and a property?

A field is a variable of any type that is declared directly in a class. A property is a member that provides a flexible mechanism to read, write or compute the value of a private field. A field can be used to explain the characteristics of an object or a class

9. How do you make a method parameter optional?

You can use optional parameters in [Methods](https://www.geeksforgeeks.org/c-sharp-methods/), [Constructors](https://www.geeksforgeeks.org/c-sharp-constructors/), [Indexers](https://www.geeksforgeeks.org/c-sharp-indexers/), and [Delegates](https://www.geeksforgeeks.org/c-sharp-delegates/).Each and every optional parameter contains a default value which is the part of its definition.If we do not pass any parameter to the optional arguments, then it takes its default value.The default value of an optional parameter is a constant expression. The optional parameters are always defined at the end of the parameter list. Or in other words, the last parameter of the method, constructor, etc. is the optional parameter.

10. What is an interface and how is it different from abstract class?

Interface cannot be implemented but abstract class can be implemented

11. What accessibility level are members of an interface?

12. True Polymorphism allows derived classes to provide different implementations

of the same method.

13. True The override keyword is used to indicate that a method in a derived class is

providing its own implementation of a method.

14. False. The new keyword is used to indicate that a method in a derived class is

providing its own implementation of a method.

15. False. Abstract methods can be used in a normal (non-abstract) class.

16. True Normal (non-abstract) methods can be used in an abstract class.

17. True Derived classes can override methods that were virtual in the base class.

18. True Derived classes can override methods that were abstract in the base class.

19. False. In a derived class, you can override a method that was neither virtual non

abstract in the base class.

20. False. A class that implements an interface does not have to provide an

implementation for all of the members of the interface.

21. True. A class that implements an interface is allowed to have other members that

aren’t defined in the interface.

22. False. A class can have more than one base class.

23. True A class can implement more than one interface.

**Designing and Building Classes using object-oriented principles**

1. Abstraction/, /Encapsulation/, /Inheritance/ and /Polymorphism/.

Abstraction: Hide details of implementation and display only details

Encapsulation: group of properties and methods are considered to single unit or object

Inheritance: derived class can use the value (properties and methods) of base class.

Polymorphism: many forms. different forms and similar methods

namespace ConsoleApp2

{

public abstract class Person

{

public Person()

{

}

public int Id { get; set; }

public string Name { get; set; }

public string Email { get; set; }

public string Phone { get; set; }

public string Address { get; set; }

public abstract void PerformWork();

}

public class Student : Person

{

public Student()

{

}

public string Course { get; set; }

public override void PerformWork()

{

//base.PerformWork();

Console.WriteLine("Students enrolled class");

}

}

public class Instructor : Person

{

public decimal TeachClass { get; set; }

public void AttendMeeting()

{

Console.WriteLine("Instructors have to attend meetings");

}

}

3. public abstract class Employee

{

public Employee()

{

}

public int Id { get; set; }

public string Name { get; set; }

public string Email { get; set; }

public string Phone { get; set; }

public string Address { get; set; }

public abstract void PerformWork();

public virtual void VitualMethodDemo()

{

Console.WriteLine("This is a virtual method");

}

}

4. namespace ConsoleApp2

{

public abstract class Person

{

public Person()

{

}

public int Id { get; set; }

public string Name { get; set; }

public string Email { get; set; }

public string Phone { get; set; }

public string Address { get; set; }

public abstract void PerformWork();

}

public class Student : Person

{

public Student()

{

}

public string Course { get; set; }

public override void PerformWork()

{

//base.PerformWork();

Console.WriteLine("Students enrolled class");

}

}

public class Instructor : Person

{

public decimal TeachClass { get; set; }

public void AttendMeeting()

{

Console.WriteLine("Instructors have to attend meetings");

}

}

5. public abstract class Employee

{

public Employee()

{

}

public int Id { get; set; }

public string Name { get; set; }

public string Email { get; set; }

public string Phone { get; set; }

public string Address { get; set; }

public abstract void PerformWork();

public virtual void VitualMethodDemo()

{

Console.WriteLine("This is a virtual method");

}

}

public class FullTimeEmployee : Employee

{

public FullTimeEmployee()

{

}

public decimal BiweeklyPay { get; set; }

public string Benefits { get; set; }

public override void PerformWork()

{

//base.PerformWork();

Console.WriteLine("Full-time employee works 40 hours a week!");

}

public override void VitualMethodDemo()

{

//base.VitualMethodDemo();

Console.WriteLine("I want to override the virtual method in full time employee class");

}

}

public sealed class PartTimeEmployee : Employee

{

public decimal HourlyPay { get; set; }

public override void PerformWork()

{

//base.PerformWork();

Console.WriteLine("Part-time employee works 20 hours a week!");

}

}

6.