

Assignment 04

Part 01

Question 1:

What is the primary purpose of an interface in C#?

- a. To provide a way to implement multiple inheritance
 - b. To define a blueprint for a class
 - c. To declare abstract methods and properties
 - d. To create instances of objects
-

Question 2:

Which of the following is NOT a valid access modifier for interface members in C#?

- a. private
 - b. protected
 - c. internal
 - d. public
-

Question 3:

Can an interface contain fields in C#?

- a. Yes
 - b. No
 - c. Only if they are static
 - d. Only if they are read only
-

Question 4:

In C#, can an interface inherit from another interface?

- a. No, interfaces cannot inherit from each other
 - b. Yes, interfaces can inherit from multiple interfaces
 - c. Yes, but only if they have the same methods
 - d. Only if the interfaces are in the same namespace
-

Question 5:

Which keyword is used to implement an interface in a class in C#?

- a. inherit
- b. use

- c. extends
 - d. implements
-

Question 6:

Can an interface contain static methods in C#?

- a. Yes
 - b. No
 - c. Only if the interface is sealed
 - d. Only if the methods are private
-

Question 7:

In C#, can an interface have explicit access modifiers for its members?

- a. Yes, for all members
 - b. No, all members are implicitly public
 - c. Yes, but only for abstract members
 - d. Only if the interface is sealed
-

Question 8:

What is the purpose of an explicit interface implementation in C#?

- a. To hide the interface members from outside access
 - b. To provide a clear separation between interface and class members
 - c. To allow multiple classes to implement the same interface
 - d. To speed up method resolution
-

Question 9:

In C#, can an interface have a constructor?

- a. Yes, but it must be private
 - b. No, interfaces cannot have constructors
 - c. Yes, but only if the interface is sealed
 - d. Only if the constructor is static
-

Question 10:

How can a C# class implement multiple interfaces?

- a. By using the "implements" keyword
 - b. By using the "extends" keyword
 - c. By separating interface names with commas
 - d. A class cannot implement multiple interfaces
-

Part 02

Question 01:

Define an interface named `IShape` with a property `Area` and a method `DisplayShapeInfo`.

Create two interfaces, `ICircle` and `IRectangle`, that inherit from `IShape`.

Implement these interfaces in classes `Circle` and `Rectangle`.

Test your implementation by creating instances of both classes and displaying their shape information.

Question 02:

In this example:

- Define the `IAuthenticationService` interface with two methods: `AuthenticateUser` and `AuthorizeUser`.
- The `BasicAuthenticationService` class implements this interface and provides the specific implementation for these methods.
- In the `BasicAuthenticationService` class:
 - The `AuthenticateUser` method compares the provided username and password with stored credentials.
 - It returns `true` if the user is authenticated and `false` otherwise.
 - The `AuthorizeUser` method checks if the user with the given username has the specified role.
 - It returns `true` if the user is authorized and `false` otherwise.
- In the `Main` method:
 - Create an instance of the `BasicAuthenticationService` class and assign it to the `authService` variable of type `IAuthenticationService`.
 - Call the `AuthenticateUser` and `AuthorizeUser` methods using this interface reference.

This implementation allows you to switch the authentication service implementation easily by:

- Creating a new class that implements the `IAuthenticationService` interface.
 - Providing the desired logic for authentication and authorization.
-

Question 03:

Define the `INotificationService` interface with a method `SendNotification` that takes a recipient and a message as parameters.

- Create three classes:
 - `EmailNotificationService`
 - `SmsNotificationService`
 - `PushNotificationService`

Each class implements the `INotificationService` interface:

- The `EmailNotificationService` class simulates sending an email by outputting a message to the console.

- The `SmsNotificationService` class simulates sending an SMS by outputting a message to the console.
- The `PushNotificationService` class simulates sending a push notification by outputting a message to the console.

In the `Main` method:

- Create instances of each notification service class.
- Call the `SendNotification` method with sample recipient and message values.

This implementation allows you to easily switch between different notification channels by:

- Creating new classes that implement the `INotificationService` interface.
- Providing the specific logic for each channel.