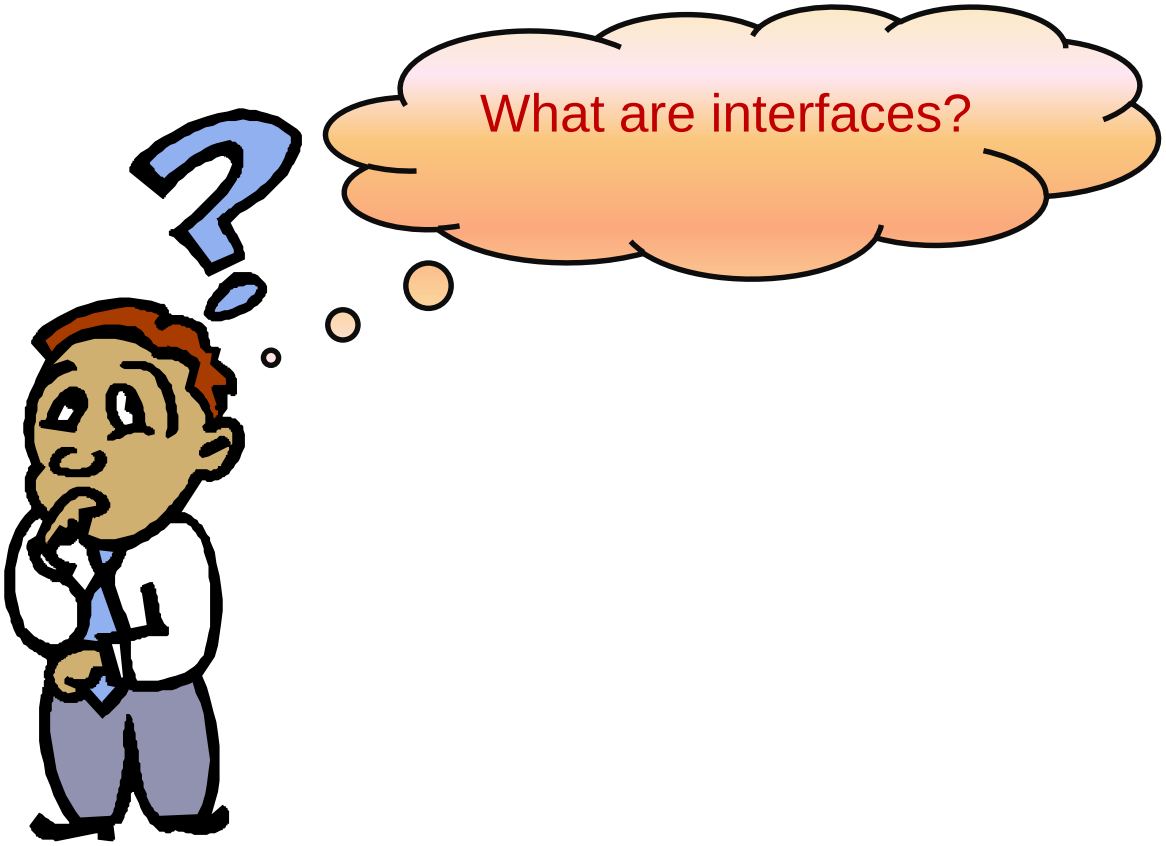
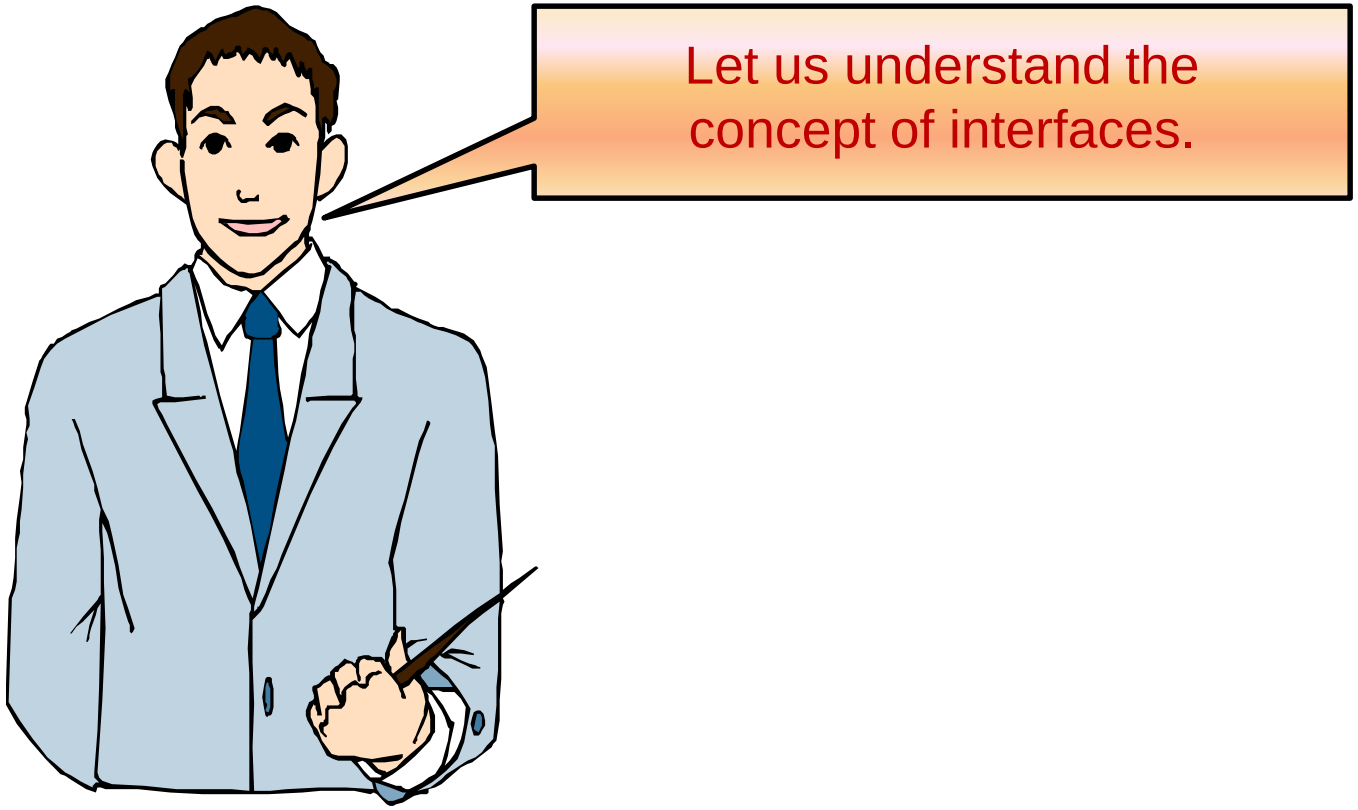
AlmaMate

- In this session, you will learn to:
  - Use interfaces

AlmaMate




Let us understand the concept of interfaces.

AlmaMate

- An interface:
  - Defines properties, methods, and events, which are known as the members of the interface.
  - Looks like a class, but has no implementation.
  - Contains only the declaration of members.
  - Is used when you want a standard structure of methods to be followed by the classes, where classes will implement the functionality.
  - Defines what a class can do, but does not define how the class does it.

- The following points describe the need of using interfaces:
  - Interfaces separate the definition of objects from their implementation.
  - Interfaces allows a class to inherit multiple behaviors from multiple interfaces.
  - Interfaces in C # provide a way to achieve runtime polymorphism.
  - Interface inheritance provides the concept of name hiding. Name hiding is an ability to hide an inherited member name from any code outside the derived class.

- Working with interfaces includes interface declaration and implementation of interfaces by the classes.
- You can declare only methods, functions, and properties in interfaces.
- You cannot declare a variable in interfaces.
- You can declare interfaces using the `interface` keyword.
- The following code snippet shows the declaration of an interface:

```
public interface IMyInterface
{
     //Declare an interface member
     void MethodToImplement();
}
```

AlmaMate

- Interfaces declare methods, which are implemented by classes.
- A class can inherit from a single class but can implement from multiple interfaces.
- The following code shows implementation of an interface:

```
public interface IMyInterface
{
//Declare an interface member
   void MethodToImplement();
}
```

AlmaMate

```
class MyClass : IMyInterface
{
static void Main()
{
InterfaceImplementer iImp = new InterfaceImplementer();
iImp.MethodToImplement();
}
public void MethodToImplement()
{
  Console.WriteLine("MethodToImplement()called.");
}
}
```

AlmaMate

- The following code shows the inheritance of an interface from the existing interface:

```
using System;
interface IParentInterface
 {
     void ParentInterfaceMethod();
 }
 interface IMyInterface:IParentInterface
 {
     void MethodToImplement();
 }
```

AlmaMate

```
class InterfaceImplementer : IMyInterface
{
   static void Main()
   {
     InterfaceImplementer iImp = new
     InterfaceImplementer();
     iImp.MethodToImplement();
     iImp.ParentInterfaceMethod();
   }
   public void MethodToImplement()
   {
     Console.WriteLine("MethodToImplement()called.");
   }
```

```
public void ParentInterfaceMethod()
{
   Console.WriteLine("ParentInterfaceMethod()called.");
}
}
```

- Problem Statement:
  - Furniture and Fittings Company (FFC) manufactures domestic furniture. Customers provide their specifications to the company for the furniture they want. To handle the increased customer’s orders, FFC decides to computerize the order-processing system. The system should accept the choice of the customer related to the attributes of the specified furniture items, such as bookshelves and chairs. You need to develop the system to accept and display the choices of the customer.

- Solution:
  - To create a console-based application for FFC, you need to perform the following tasks:
    1. Create a console-based application.
    2. Build and execute an application.

- In this session, you learned that:
  - An interface defines properties, methods, and events. The properties, methods, and events that are defined in the interface are known as the members of the interface.

AlmaMate

- Solve the following exercises from the Object Oriented Programming Using C# - II book in the Machine Room:
  - Chapter 8: Exercise 1
  - Chapter 8: Exercise 2