الجامعة المستنصرية
كليـــــــة العـــــلوم
قسم علوم الحاسبات

البرمجة الكيانية / المرحلة الثانية / حسن قاسم محمد

Object Oriented Programming

Abstraction

Abstract Classes and Methods

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either **abstract classes** or **interfaces**

The abstract keyword is used for classes and methods:

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).

An abstract class can have both abstract and regular methods:

```
abstract class Animal
{
  public abstract void animalSound();
  public void sleep()
  {
    Console.WriteLine("ZZZ");
  }
}
```

```
Animal myObj = new Animal();
```

X

it is not possible to create an object of the Animal class:

```
Will generate an error (Cannot
create an instance of the abstract
class or interface 'Animal')
```

To access the abstract class, it must be inherited from another class.

we use the `override` keyword to override the base class method.

```
// Abstract class

abstract class Animal

{

  // Abstract method (does not have a
body)

  public abstract void animalSound();

  // Regular method

  public void sleep()

  {

    Console.WriteLine("ZZZ");

  }

}
```

```
class Cat : Animal          // Derived class
                           (inherit from Animal)

{  public override void animalSound()

   {    // The body of animalSound() is provided here

     Console.WriteLine("The cat says: meao");

   }

}

class Program

{       static void Main(string[] args)

   {   Cat myCat = new Cat(); // Create a Cat object

     myCat.animalSound();  // Call the abstract method

     myCat.sleep();  // Call the regular method

   }   }
```

Why To Use Abstract Classes and Methods?

- To achieve security
- hide certain details
- show the important details of an object.

Abstraction can also be achieved with **Interfaces**

Interface

Another way to achieve abstraction

An `interface` is a completely "**abstract class**", which can only contain abstract methods and properties (with empty bodies):

Example

```
// interface

interface Animal

{

  void animalSound(); // interface method (does not have a body)

  void run(); // interface method (does not have a body)

}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class.

To implement an interface, use the `:` symbol (just like with inheritance).

The body of the interface method is provided by the "implement" class.

Note that you do not have to use the `override` keyword when implementing an interface:

```
// Interface

interface IAnimal

{   void animalSound(); // interface method (does
not have a body)

}

// Cat "implements" the IAnimal interface

class Cat : IAnimal

{  public void animalSound()

  {

    // The body of animalSound() is provided here

    Console.WriteLine("The Cat says: meao");

  }  }
```

```
 class Program

{

  static void Main(string[] args)

  {

    Cat myCat = new Cat();  // Create
a Cat object

    myCat.animalSound();

  }

}
```

Notes on Interfaces:

- Like **abstract classes**, interfaces cannot be used to create objects
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface members are by default `abstract` and `public`
- An interface cannot contain a constructor (as it cannot be used to create objects)

Note: To implement multiple interfaces, separate them with a comma

Multiple Interfaces

To implement multiple interfaces, separate them with a comma:

```
interface IFirstInterface

{  void myMethod(); // interface method

}

 interface ISecondInterface

{

  void myOtherMethod(); // interface
method

}
```

```
// Implement multiple interfaces

class DemoClass : IFirstInterface, ISecondInterface

{  public void myMethod()

  {  Console.WriteLine("Some text..");   }

  public void myOtherMethod()

  {  Console.WriteLine("Some other text..."); }  }

 class Program

{  static void Main(string[] args)

  { DemoClass myObj = new DemoClass();

    myObj.myMethod();

    myObj.myOtherMethod();

  } }
```

Interfaces summary

An interface has the following properties:

- An interface is typically like an abstract base class with only abstract members. Any class or struct that implements the interface must implement all its members. Optionally, an interface may define default implementations for some or all of its members.
- An interface can't be instantiated directly. Its members are implemented by any class or struct that implements the interface.
- A class or struct can implement multiple interfaces. A class can inherit a base class and also implement one or more interfaces.

Thank’s