Session: 6

Classes and Methods

Objectives

- Explain classes and objects
- Define and describe methods
- List the access modifiers
- Explain method overloading
- Define and describe constructors and destructors

Object-Oriented Programming

- Programming languages are based on two fundamental concepts: data and ways to manipulate data.
- Traditional languages such as Pascal and C used the procedural approach which focused more on ways to manipulate data rather than on the data itself.
- This approach has several drawbacks such as lack of re-use and lack of maintainability.
- To overcome these difficulties, OOP was introduced, which focused on data.
- The object-oriented approach defines objects as entities having a defined set of values and a defined set of operations that can be performed on these values.

Features of OOP

Abstraction

- **Extracting only the required information from objects.**

  For example, consider a television, it has a manual stating how to use the tv. However, this manual does not show all the technical details of the television.

Encapsulation

- **Visible only some specific information of a class.**

  Encapsulation also called data hiding.

  Both abstraction and encapsulation are complementary to each other.

Inheritance

- **Creating a new class based on an existing class.**
  The existing class: base class.
  The new class: derived class.

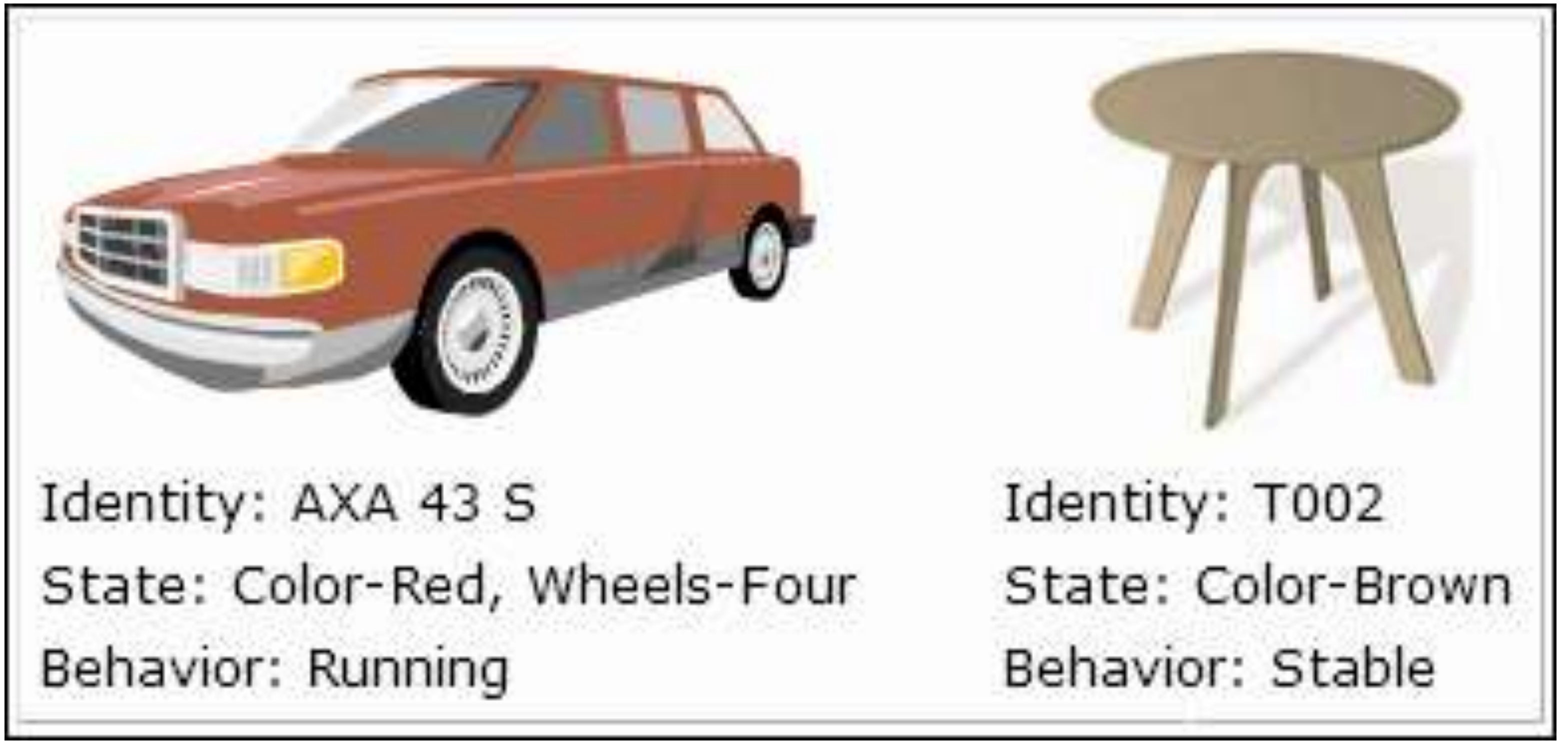
  This is a very important concept of OOP as it helps to reuse the inherited attributes and methods.

Polymorphism

- **The ability to behave differently in different situations.**

  It is seen in programs where you have multiple methods declared with the same name but with different parameters and different behavior.

Objects



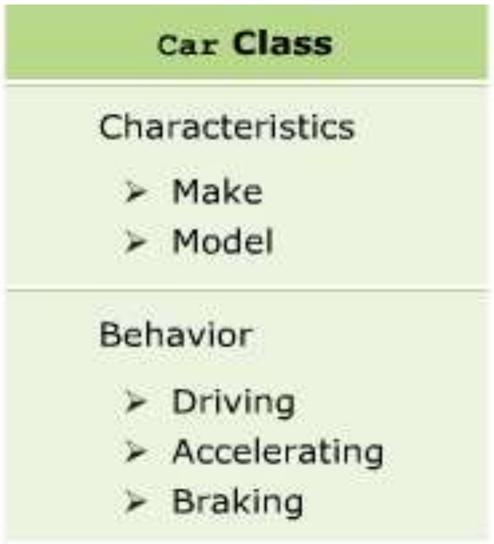
- An object stores its identity and state in fields (also called variables) and exposes its behavior through methods.

Classes

- Several objects have a common state and behavior and thus, can be grouped under a single class.

Example

- A Ford Mustang, a Volkswagen Beetle, and a Toyota Camry can be grouped together under the class Car. Here, Car is the class whereas Ford, Volkswagen, and Toyota are objects of the class Car.
- The following figure displays the class Car:



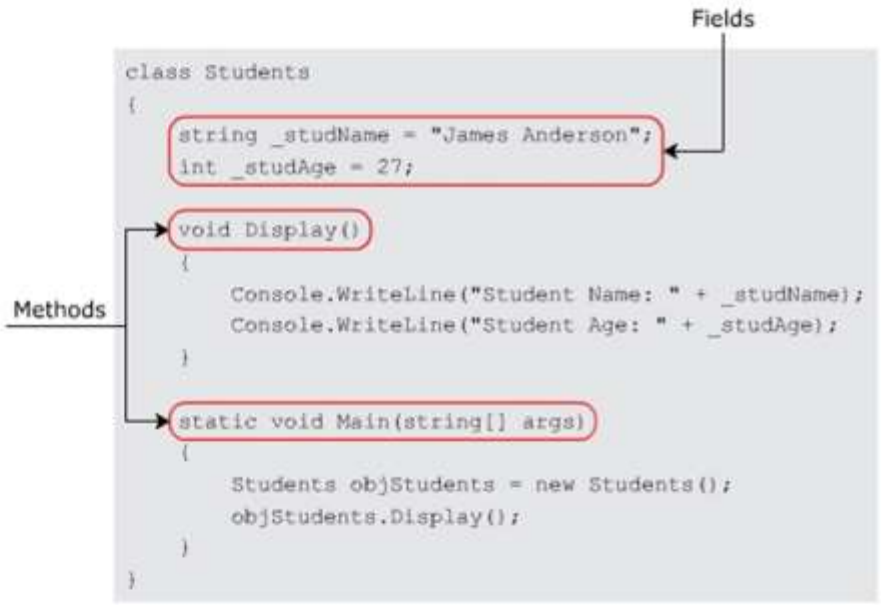
Car Class

Characteristics

- Make
- Model

Behavior

- Driving
- Accelerating
- Braking



```
class Students
{
    string _studName = "James Anderson";
    int _studAge = 27;

    void Display()
    {
        Console.WriteLine("Student Name: " + _studName);
        Console.WriteLine("Student Age: " + _studAge);
    }

    static void Main(string[] args)
    {
        Students objStudents = new Students();
        objStudents.Display();
    }
}
```

Instantiating Objects

- To access the variables and methods defined within class, using objects.
- An object is instantiated using the **new** keyword.
  On encountering the new keyword, JIT compiler allocates memory for the object and returns a reference of that allocated memory.

Syntax

```
<ClassName> <objectName> = new <ClassName>();
```

Creating Methods

- Conventions to be followed for naming methods :

Cannot be a C# keyword, cannot contain spaces, and cannot begin with a digit

Can begin with a letter, underscore, or the "@" character

Example

Invalid method names include **5Add**, **AddSum()**, **@int()**.

Syntax

```
<access_modifier> <return_type> <MethodName> ([parameters])
{
     // body of the method
}
```

Method Parameters and Arguments

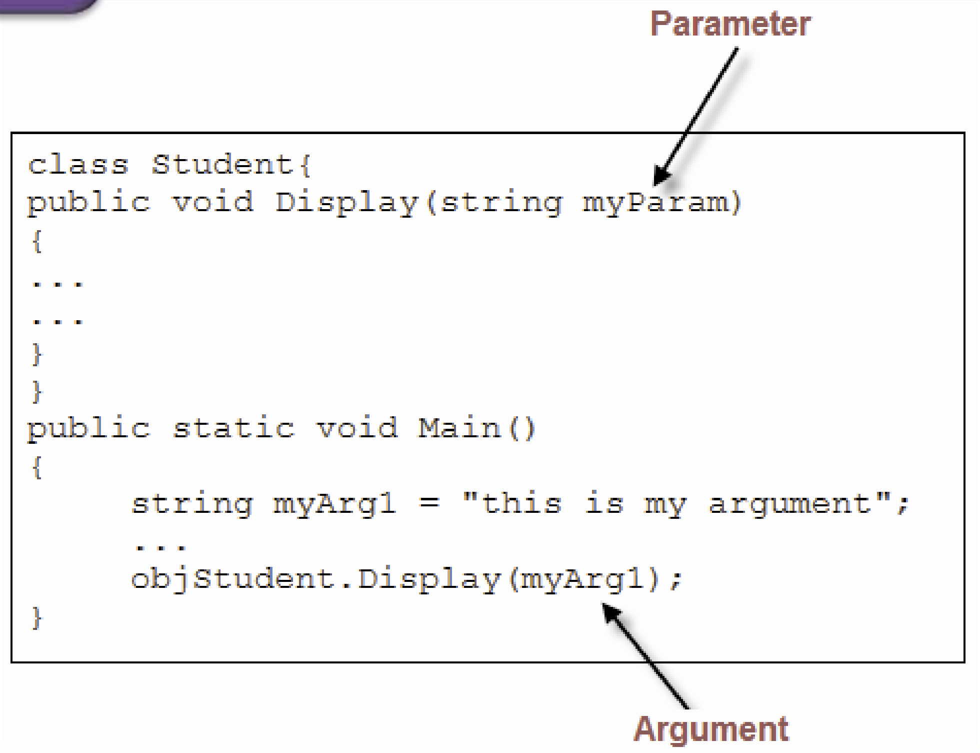
Parameters

- The variables included in a method definition are called parameters.

Argument

- When the method is called, the data send into the method's parameters are called arguments.

- Example



```
class Student{
public void Display(string myParam)
{
...
...
}
}
public static void Main()
{
     string myArg1 = "this is my argument";
     ...
     objStudent.Display(myArg1);
}
```

Named and Optional Arguments

- The following code shows how to use optional arguments:

Snippet

```
using System;
class OptParameterExample {
   void printMessage(String message = "Hello user!") {
         Console.WriteLine("{0}", message);
   }
   static void Main(string[] args) {
     OptParameterExample o = new OptParameterExample();
     o.printMessage("Welcome User!");
     o.printMessage();
   }
}
```

Static Classes

- cannot be inherited, consists of static data members and static methods only.
  The **`static`** keyword is used before the class name
- cannot create an instance of a static class using the **`new`** keyword. However, static constructors can be defined to initialize the static members.
- Since there is no need to create objects of the static class to call the required methods, the implementation is simpler and faster.

Static Methods

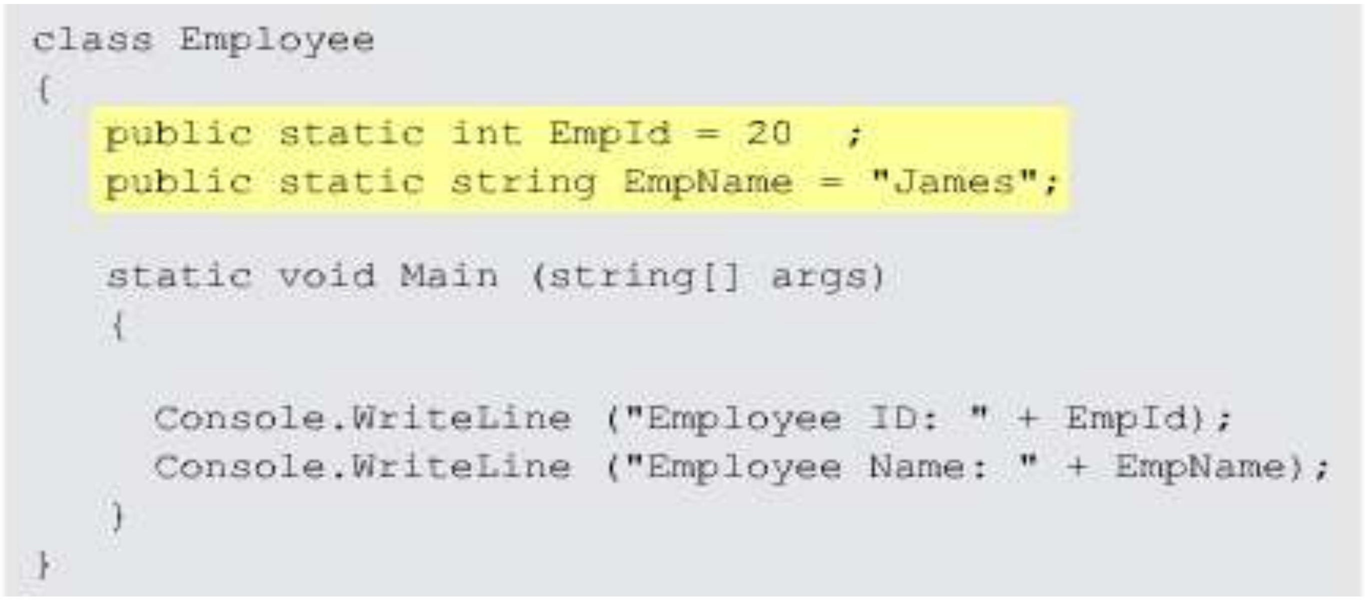
- declared by using the `static` keyword.
  For example, the `Main()` method is a static method and it does not require any instance of the class for it to be invoked.
- can directly refer only to static variables and methods of the class.
- also can refer to non-static methods and variables by using an instance of the class.

Syntax

```
Static <return_type> <MethodName>()
{
    // body of the method
}
```

Static Variables

- is a special variable, accessed without using an object of class.
- declared by the **static** keyword. When a static variable is created, it is automatically initialized before it is accessed.
- Only one copy of a static variable is shared by all the objects of the class.
- Therefore, a change in the value of such a variable is reflected by all the objects of the class.



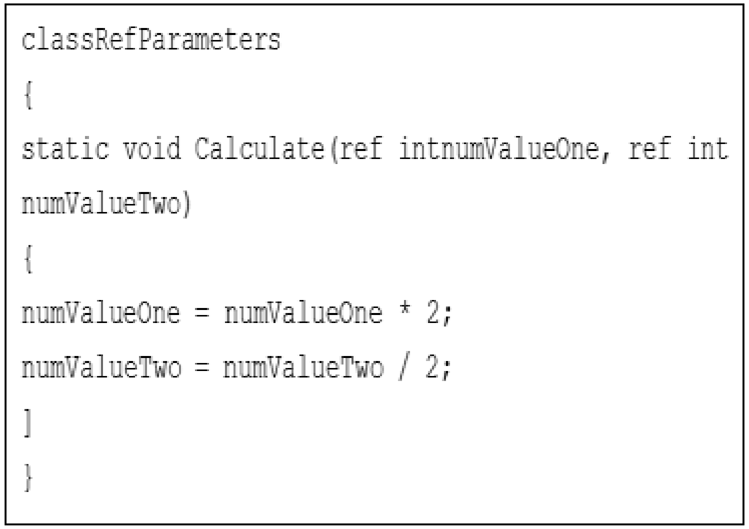
```
class Employee
{
    public static int EmpId = 20  ;
    public static string EmpName = "James";

    static void Main (string[] args)
    {

      Console.WriteLine ("Employee ID: " + EmpId);
      Console.WriteLine ("Employee Name: " + EmpName);
    }
}
```

- In C#, there are four commonly used access modifiers.

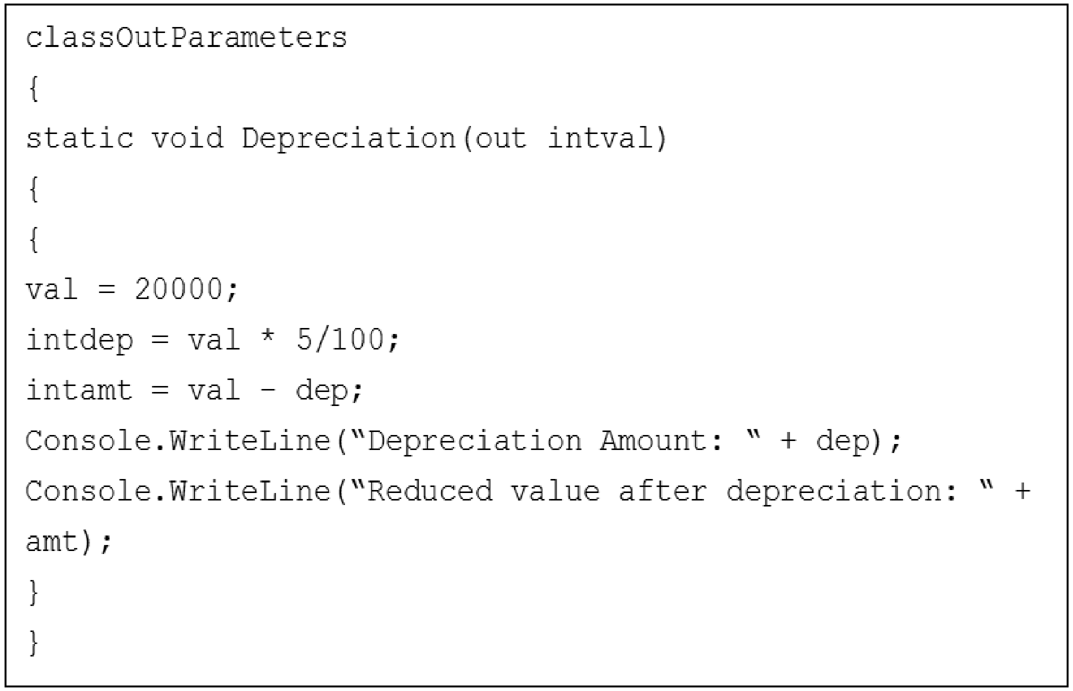
- **public**:
  provides the most permissive access level.
- **private:**
  provides the least permissive access level. Private members are accessible only within the class in which they are declared.
- **protected:**
  Protected the class members are accessible within the class as well as within the derived classes.
- **internal:**
  Internal members are accessible only within the classes of the same assembly.

`ref` and `out` Keywords

- Causes arguments to be passed to called method by reference.
- In call by reference, the called method changes the value of the arguments passed to it.
- Both the called method and the calling method must explicitly specify the `ref,out` keyword before the required parameters.
- `out` keyword does **not require** the arguments to be initialized.



```
classRefParameters
{
static void Calculate(ref intnumValueOne, ref int
numValueTwo)
{
numValueOne = numValueOne * 2;
numValueTwo = numValueTwo / 2;
}
}
```



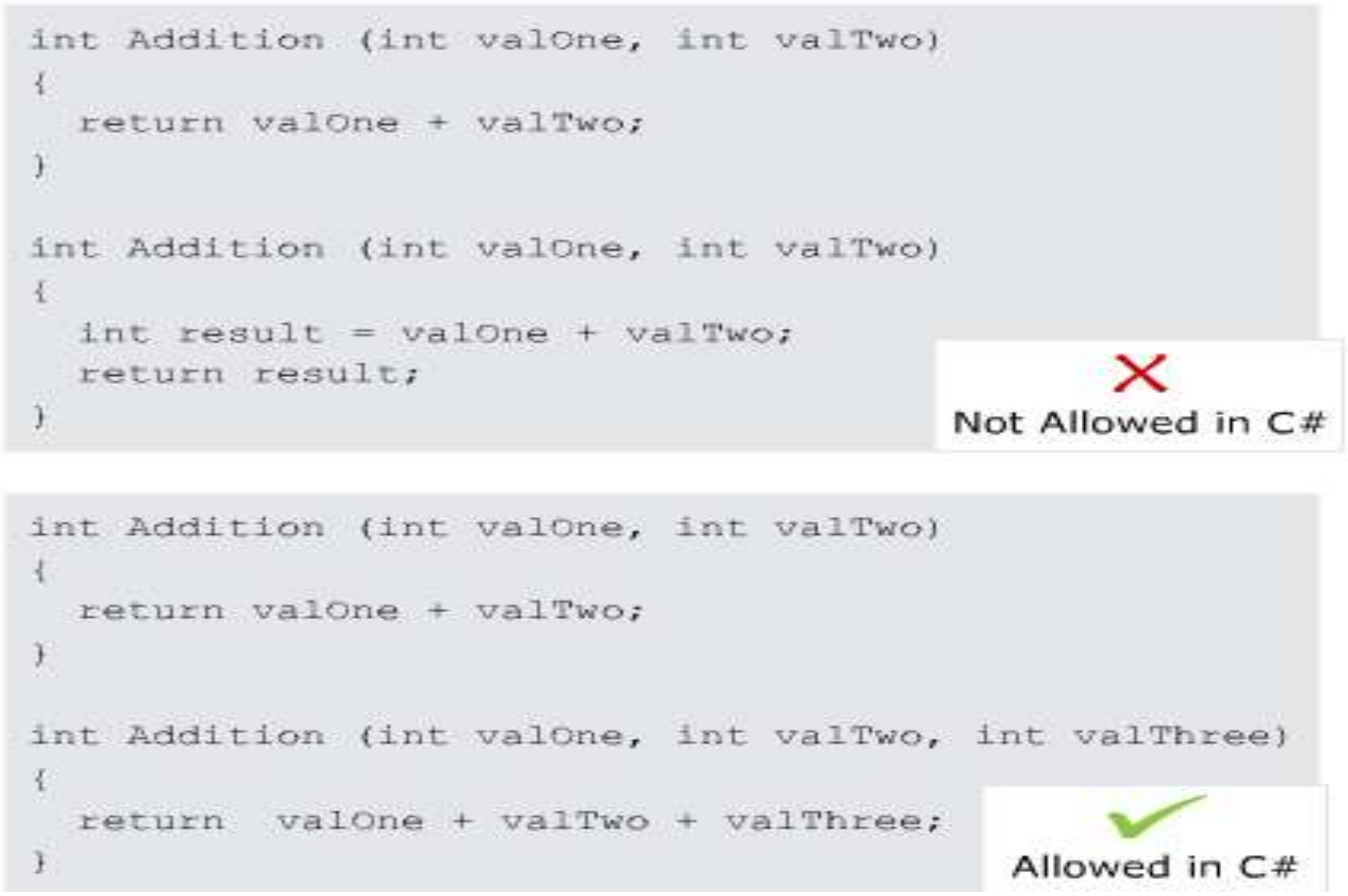
```
classOutParameters
{
static void Depreciation(out intval)
{
{
val = 20000;
intdep = val * 5/100;
intamt = val - dep;
Console.WriteLine("Depreciation Amount: " + dep);
Console.WriteLine("Reduced value after depreciation: " +
amt);
}
}
```

Method Overloading in C# 1-2

- In OOP, every method has a signature which includes:
  - The number of parameters passed to the method, the data types of parameters and the order in which the parameters are written.
  - the signature of the method is written in parentheses next to the method name.
  - No class is allowed to contain two methods with the same name and same signature, but it is possible for a class to have two methods having the same name but different signatures.
  - The concept of declaring more than one method with the same method name but different signatures is called method overloading.

Method Overloading in C# 2-2

- The following figure displays the concept of method overloading using an example:



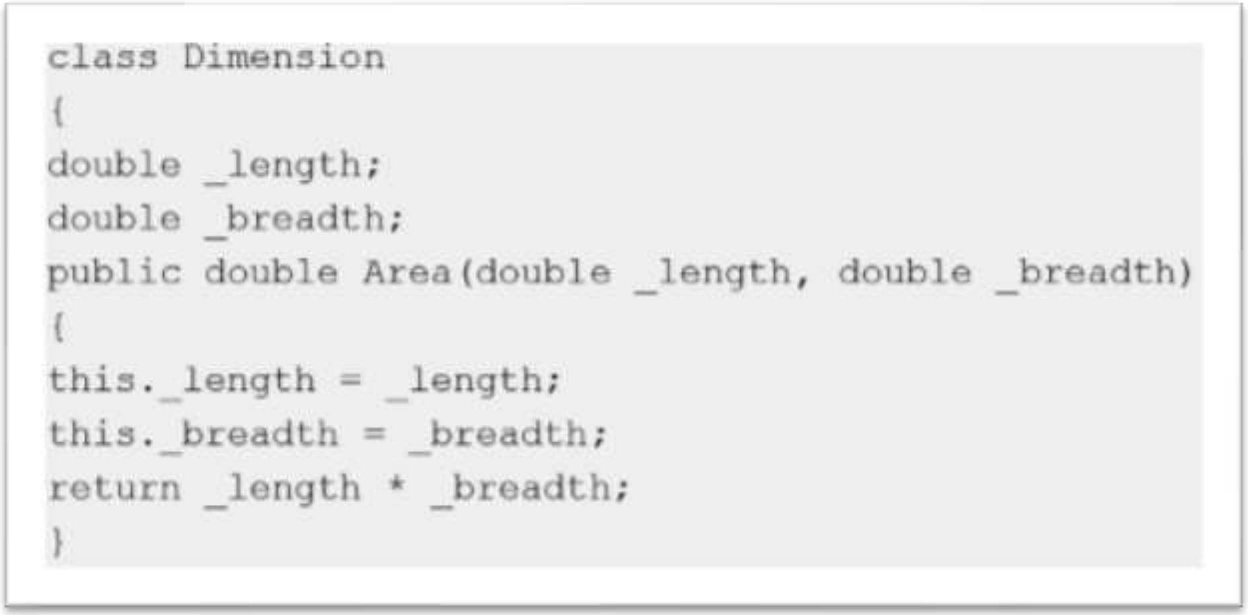
```
int Addition (int valOne, int valTwo)
{
   return valOne + valTwo;
}

int Addition (int valOne, int valTwo)
{
   int result = valOne + valTwo;
   return result;
}
```

```
int Addition (int valOne, int valTwo)
{
   return valOne + valTwo;
}

int Addition (int valOne, int valTwo, int valThree)
{
   return  valOne + valTwo + valThree;
}
```

The this keyword

- The **this** keyword is used to refer to the current object of the class to resolve conflicts between variables having same names and to pass the current object as a parameter.

- You cannot use the **this** keyword with static variables and methods.



```
class Dimension
{
double _length;
double _breadth;
public double Area(double _length, double _breadth)
{
this._length = _length;
this._breadth = _breadth;
return _length * _breadth;
}
```

Constructors and Destructors

- A C# class can define constructors and destructors as follows:

Constructors

- A C# class can contain one or more special member functions having the same name as the class, called constructors.
- A constructor is a method having the same name as that of the class.

Destructors

- A C# class can also have a destructor (only one is allowed per class), which is a special method and also has the same name as the class but prefixed with a special symbol ~.
- A destructor of an object is executed when the object is no longer required in order to de-allocate memory of the object.

Default and Static Constructors

Default Constructors

C# creates a default constructor for a class if no constructor is specified within the class.

The default constructor automatically initializes all the numeric data type instance variables to zero.

If you define a constructor in the class, the default constructor is no longer used.

Static Constructors

A static constructor is used to initialize static variables and to perform a particular action only once.

It is invoked before any static member of the class is accessed.

A static constructor does not take any parameters and does not use any access modifiers because it is invoked directly by the CLR instead of the object.

Constructor Overloading

- The following code demonstrates the use of constructor overloading:

```
using System;
public class Rectangle  {
    double _length;
    double _breadth;
    public Rectangle() {
        _length = 13.5;
        _breadth = 20.5;
    }
    public Rectangle(double len, double wide) {
       _length = len;
       _breadth = wide;
    }

    public double Area() {
       return _length * _breadth;
    }
    static void Main(string[] args) {
      Rectangle objRect1 = new Rectangle();
      Console.WriteLine("Area of rectangle = " + objRect1.Area());
      Rectangle objRect2 = new Rectangle(2.5, 6.9);
      Console.WriteLine("Area of rectangle = " + objRect2.Area());
    }
}
```

Summary

- The programming model that uses objects to design a software application is termed as OOP.
- A method is defined as the actual implementation of an action on an object and can be declared by specifying the return type, the name and the parameters passed to the method.
- It is possible to call a method without creating instances by declaring the method as static.
- Access modifiers determine the scope of access for classes and their members.
- The four types of access modifiers in C# are public, private, protected and internal.
- Methods with same name but different signatures are referred to as overloaded methods.
- In C#, a constructor is typically used to initialize the variables of a class.