**Polymorphism In C# Programming**

* Polymorphism is one of the four pillars of Object Oriented Programming.
* Polymorphism in C# is a concept by which we can perform a single action by different ways.
* Polymorphism is derived from 2 Greek words: **POLY** and **MORPHS**.
* The word **"poly" means many** and **"morphs" means forms**.
* So polymorphism means **many forms**.

**Real World Example**

**Single person have multiple forms, explained in figure below**



**Same Person is a father of someone, son of someone, husband of someone and teacher by profession.**

**There Are Two Types Of Polymorphism**

1. Static Polymorphism (Compile Time Polymorphism)
2. Dynamic Polymorphism (Run Time Polymorphism)

**Static Polymorphism (Compile Time Polymorphism) In C#**

* The mechanism of linking a function with an object during compile time is called static polymorphism or early binding.
* It is also called static binding.

**C# provides two techniques to implement static polymorphism. They are −**

* Method Or Function Overloading
* Operator Overloading

**Method Or Function Overloading**

* You can have multiple definitions for the same function name in the same scope.
* The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.
* You cannot overload function declarations that differ only by return type.

**In object-oriented programming, 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.
* While declaring a method, 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.

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



**The following code overloads the Square() method to calculate the square of the given int and float values:**

using System;

class MethodOverloadExample

{

static void Main(string[] args)

{

Console.WriteLine(“Square of integer value “ + Square(5));

Console.WriteLine(“Square of float value “ + Square(2.5F));

}

static int Square(intnum)

{

return num \* num;

}

static float Square(float num)

{

return num \* num;

}

}

**In Above Code,**

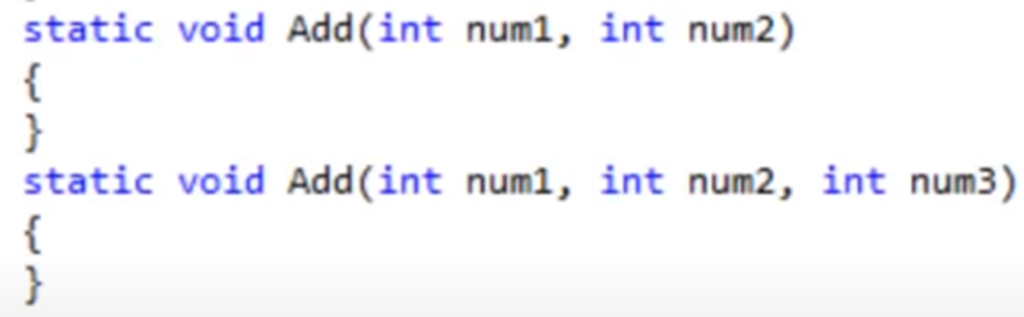
* Two methods with the same name but with different parameters are declared in the class.
* The two Square() methods take in parameters of int type and float type respectively.
* Within the Main() method, depending on the type of value passed, the appropriate method is invoked and the square of the specified number is displayed in the console window.

**Output**

Square of integer value 25  
Square of float value 6.25

**Guidelines and Restrictions**

* The methods to be overloaded should perform the same task.
* The signatures of the overloaded methods must be unique.
* When overloading methods, the return type of the methods can be the same as it is not a part of the signature.
* The **ref** and **out** parameters can be included as a part of the signature in overloaded methods.



**Source Code Of Method Overloading In C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace METHOD\_OVERLOADING

{

class Program

{

//public void Add()

//{

// int a = 20;

// int b = 30;

// int c = a + b;

// Console.WriteLine(c);

//}

public void Add(int a, int b)

{

int c = a + b;

Console.WriteLine(c);

}

public int Add(int a, int b)

{

int c = a + b;

return c;

}

//public void Add(string a, string b)

//{

// string c = a + " " + b;

// Console.WriteLine(c);

//}

//public void Add(float a, float b)

//{

// float c = a + b;

// Console.WriteLine(c);

//}

static void Main(string[] args)

{

Program p = new Program();

//p.Add();

// p.Add(2.5f, 1.5f);

//p.Add(10, 5);

p.Add("Adil", "Mehmood");

Console.ReadLine();

}

}

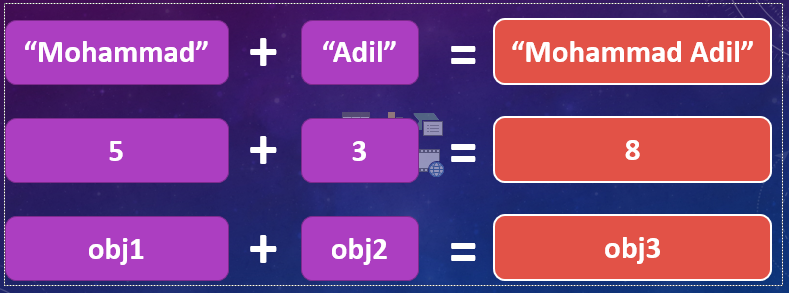
}

**Operator Overloading In C#**

* This concept resides in Polymorphism.
* 1. Static polymorphism (compile time polymorphism)
  + Method Overloading
  + Operator Overloading
* 2. Dynamic polymorphism (run time polymorphism)
  + Method Overriding

**Operator Overloading**

* The concept of overloading a function can also be applied to operators.
* Operator overloading gives the ability to use the same operator to do various operations.
* It provides additional capabilities to C# operators when they are applied to user-defined data types.
* It enables to make user-defined implementations of various operations where one or both of the operands are of a user-defined class.
* Only the predefined set of C# operators can be overloaded.



**Example**

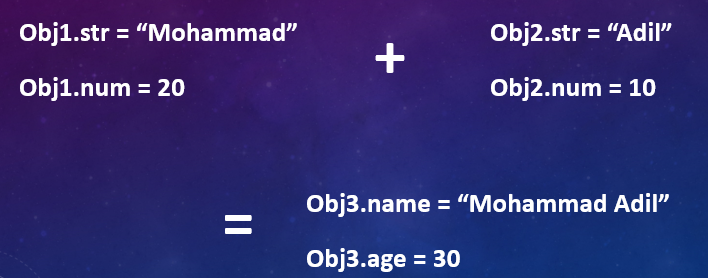
Class NewClass

{

public string str;

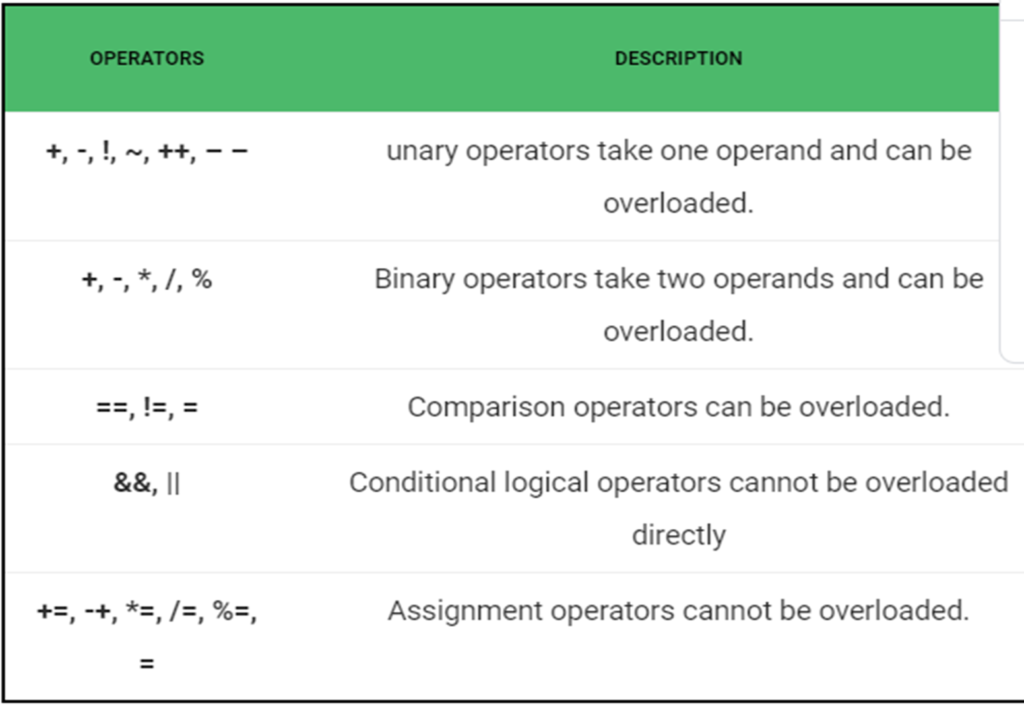
public int num;

}



**Operator Overloading**

* To make operations on a user-defined data type is not as simple as the operations on a built-in data type.
* To use operators with user-defined data types, they need to be overloaded according to a programmer’s requirement.
* An operator can be overloaded by defining a function to it.
* The function of the operator is declared by using the operator keyword.
* Operators may be considered as functions internal to the compiler.



**Source Code Of Operator Overloading**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OperatorOverloading

{

class NewClass

{

public string str;

public int num;

public static NewClass operator +(NewClass obj1, NewClass obj2)

{

NewClass obj3 = new NewClass();

obj3.str = obj1.str + " " + obj2.str;

obj3.num = obj1.num + obj2.num;

return obj3;

}

}

class Program

{

static void Main(string[] args)

{

NewClass obj1 = new NewClass();

obj1.str = "Mohammad";

obj1.num = 20;

NewClass obj2 = new NewClass();

obj2.str = "Adil";

obj2.num = 30;

NewClass obj3 = new NewClass();

obj3 = obj1 + obj2;

Console.WriteLine(obj3.str);

Console.WriteLine(obj3.num);

Console.ReadLine();

}

}

}

**Dynamic Or Runtime Polymorphism In C#**

* Run time polymorphism is achieved by method overriding.
* Method overriding allows us to have virtual and abstract methods in the base using derived classes with the same name and the same parameter.

**C# Method Overriding**

* If derived class defines same method as defined in its base class, it is known as method overriding in C#.
* It is used to achieve runtime polymorphism.
* It enables you to provide specific implementation of the method in child class which is already provided by its base class.
* To perform method overriding in C#, you need to use virtual keyword with base class method and override keyword with derived class method.
* A method declared using the virtual keyword is referred to as a virtual method.
* In the derived class, you need to declare the inherited virtual method using the override keyword.
* In the derived class, you need to declare the inherited virtual method using the override keyword which is mandatory for any virtual method that is inherited in the derived class.
* The override keyword overrides the base class method in the derived class.

**Source Code Of Method Overriding**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace METHOD\_OVERRIDING

{

class parent

{

public virtual void print()

{

Console.WriteLine("This a method of PARENT class");

}

}

class child : parent

{

public override void print()

{

Console.WriteLine("This a method of CHILD class");

}

}

class Program

{

static void Main(string[] args)

{

parent p = new child();

p.print();

Console.ReadLine();

}

}

}