

Common keywords and features of C#



- ❖ Research about common keywords and features in C#.
- ❖ Know how to use keywords in C# such as partial, params, ref, out .
- ❖ Know how to use some features such as extension method, optional arguments.

Project introduction

- A system manages student information:
 - ❖ Student information: Student Id, Name, Age, Class Name, School Year, Couse Name, Mark
 - ❖ Allow to add/modify student information by Student Id
 - ❖ Allow to print information
 - ❖ Allow to get average mark of group students

start

- ❖ C# class is too long, difficult to maintain
- ❖ There are 2 or more developers working together
- ❖ Class has multiple purposes, and can be categorized them

partial (C# 2.0)

Rules for partial classes

- ❖ We must use the partial keyword in each partial classes.
- ❖ The partial class must have the same level of access (public, protected, private ...).
- ❖ The partial class must be in the same assembly or the same module (.exe or .dll).
- ❖ If a part is abstract or sealed, the other parts must also be declared abstract or sealed.
- ❖ If a part has inherited from another class or interfaces, other parts also understand that inheriting from that class or interfaces, so you don't need to inherit it.

In Student.cs file

```
public partial class Student
{
    public Student() { }

    public string StudentId { get; set; }

    public string Name { get; set; }

    public int Age { get; set; }

    public string ClassName { get; set; }

    public int SchoolYear { get; set; }

    public string CourseName { get; set; }

    public decimal Mark { get; set; }
}
```


In StudentPresent.cs file

```
public partial class Student
{
    public void PrintInformation()
    {
        Console.WriteLine("*****");
        Console.WriteLine("Name: {0}", this.Name);
        Console.WriteLine("Age: {0}", this.Age);
        Console.WriteLine("Class Name: {0}", this.ClassName);
        Console.WriteLine("School Year: {0}", this.SchoolYear);
        Console.WriteLine("Course Name: {0}", this.CourseName);
        Console.WriteLine("Mard: {0}", this.Mark);
        Console.WriteLine("*****");
    }
}
```

next

- ❖ Create method to get average mark of some student
 - ❖ Create an array
 - ❖ Push students into the array
 - ❖ Loop the array

```
Student tony = new Student("Tony", 21, "Microsoft technology", 1, "C#", 9);  
Student oliver = new Student("Oliver", 21, "Microsoft technology", 1, "C#", 6.5m);  
Student jack = new Student("Jack", 21, "Microsoft technology", 1, "C#", 4.2m);  
Student harry = new Student("Harry", 21, "Microsoft technology", 1, "C#", 5.9m);  
Student jacob = new Student("Jacob", 21, "Microsoft technology", 1, "C#", 7.9m);  
Student charlie = new Student("Charlie", 21, "Microsoft technology", 1, "C#", 8.0m);  
Student thomas = new Student("Thomas", 21, "Microsoft technology", 1, "C#", 3.6m);  
Student george = new Student("George", 21, "Microsoft technology", 1, "C#", 9.6m);  
Student james = new Student("James", 21, "Microsoft technology", 1, "C#", 9);
```

```
Student[] students = new Student[10];  
students[0] = tony;  
students[1] = oliver;  
students[2] = jack;  
students[3] = harry;  
students[4] = jacob;  
students[5] = charlie;  
students[6] = thomas;  
students[7] = george;  
students[8] = james;  
  
var avgMark = GetAverageMark(students);
```

Questions

- ❖ Why do we need create an array?
- ❖ How is the length of the array?
- ❖ Can we make it shorter?

params examples

```
public static decimal GetAverageMark(params Student[] students)
{
    var avgMark = 0;
    //// your code to return average mark
    return avgMark;
}
```

```
var avgMark = GetAverageMark(tony, oliver, jack, harry, jacob, charlie, thomas, george, james);
```

```
var avgMark = GetAverageMark(tony, oliver, jack, harry, jacob);
```

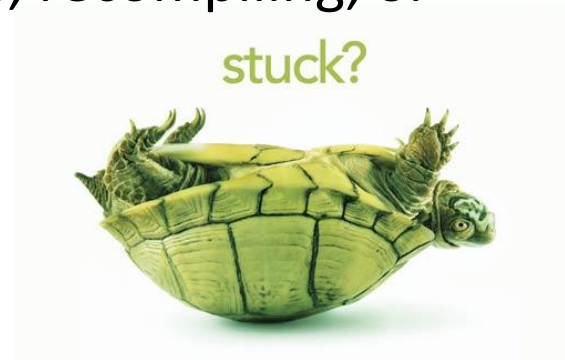


Difference number of arguments

- ❖ It is used as a parameter which can take the variable number of arguments
- ❖ It is useful when programmer don't have any prior knowledge about the number of parameters to be used.
- ❖ Only one params keyword is allowed in method.
- ❖ The length of params will be zero if no arguments will be passed

next

- ❖ You are using data type from .net Framework or 3rd party.
- ❖ You want to add methods to existing types
- ❖ BUT, cannot creating a new derived type, recompiling, or otherwise modifying the original type



Extension Method

from C# 3.0

- ❖ Extension methods allow existing classes to be extended without relying on inheritance or having to change the class's source code.
- ❖ If the class is sealed then there is no concept of extending its functionality. For this a new concept is introduced, in other words extension methods.
- ❖ This feature is important for all developers, especially if you would like to use the dynamism of the C# enhancements in your class's design.

Steps to create extension method

- ❖ Step 1: Create new **static** class. There is no such rule for the name of class. Suggestion name should be **[DataType]+Extension**
E.g. you want to extend class Student => StudentExtensions
- ❖ Step 2: Add a **static** method with name of method you want
- ❖ Step 3: Put instance of original data as a first argument of the method, and comes along with **this** keyword


Steps to create extension method

```
public static class StudentExtension
{
    public static string ToEvaluate(this Student student)
    {
        if (student.Mark >= 9)
        {
            return "Excellent";
        }

        //// ...
        return "False";
    }
}
```



```
Student student = new Student();
//// some code here
student.ToEvaluate();
```

 (extension) string Student.ToEvaluate()

- ❖ An extension method must be defined in a top-level static class.
- ❖ An extension method with the same name and signature as an instance method will not be called.
- ❖ Extension methods cannot be used to override existing methods.
- ❖ The concept of extension methods cannot be applied to fields, properties or events.
- ❖ Overuse of extension methods is not a good style of programming.

❖ Create extension method to print currency in word

- ❖ Data type to extend: decimal
- ❖ Method name: PrintCurrency
- ❖ Example: 123,45 should be printed:

“One hundred twenty three dollar and forty five cents”

next

Check the code

```
Student tony = new Student("Tony", 21, "Microsoft technology", 1);  
Student oliver = new Student("Oliver", 21, "Microsoft technology", 1);  
Student jack = new Student("Jack", 21, "Microsoft technology", 1);  
Student harry = new Student("Harry", 21, "Microsoft technology", 1);  
Student jacob = new Student("Jacob", 21, "Microsoft technology", 1);  
Student charlie = new Student("Charlie", 21, "Microsoft technology", 1);  
Student thomas = new Student("Thomas", 21, "Microsoft technology", 1);  
Student george = new Student("George", 21, "Microsoft technology", 1);  
Student james = new Student("James", 21, "Microsoft technology", 1);  
Student william = new Student("William", 23, "Microsoft technology", 1);
```

User said: Almost students are 21, in “Microsoft technology” class, and in the first year.

I don't want to re-type them for many times

Solution 1: Create new overloading method.

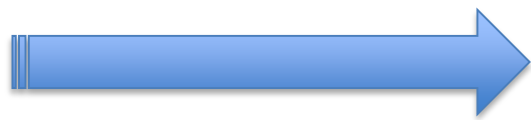
Evaluation: It's OK. But, is there anyway better?

Solution 2: From C# 4.0, use Optional Arguments
if arguments are not passed, they will use default value

- ❖ Each Optional Parameter has a default value as its part of the definition.
- ❖ If no argument is sent for the Optional Parameter default is being used.
- ❖ Default value of the Optional Parameter must be constant
- ❖ Optional Parameter must define at the end of the any required parameter.
- ❖ Optional Parameter could be applied on Constructor, Method, and Indexer
- ...

Implement Optional Parameters

```
public Student(string name, int age = 21, string className = "Microsoft technology", int schoolYear = 1)
{
    this.Name = name;
    this.Age = age;
    this.ClassName = className;
    this.SchoolYear = schoolYear;
}
```



```
Student tony = new Student("Tony");
Student oliver = new Student("Oliver");
Student jack = new Student("Jack");
Student harry = new Student("Harry");
Student jacob = new Student("Jacob");
Student charlie = new Student("Charlie");
Student thomas = new Student("Thomas");
Student george = new Student("George");
Student james = new Student("James");
Student william = new Student("William", 23);
```

next

User want to create a student name Peter, 21 years old, in “Microsoft technology” class, but, in 2nd year.

Should I use this code?

```
Student peter = new Student("Peter", 2);
```

Named Arguments

```
Student peter = new Student(name: "Peter", schoolYear: 2);
```

Named arguments free you from the need to remember or to look up the order of parameters in the parameter lists of called methods. The parameter for each argument can be specified by parameter name.

If you do not remember the order of the parameters but know their names, you can send the arguments in any order.

Named arguments also improve the readability of your code by identifying what each argument represents.

- ✓ Know about common features and keyword in C#
- ✓ Understand the mechanism and reasons for use features and keyword such as partial, params, ref, out ...
- ✓ Know how to apply the actual features and keyword

Thank you

