FortuneCloud
We Understand Technology

C# Microsoft .NET

What is C# ?

- C# is a programming language of .Net Framework.
- It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.
- we can develop different types of secured and robust applications:
- Window applications
- Web applications
- Web service applications

- C# is designed for CLI (Common Language Infrastructure).
- CLI is a specification that describes executable code and runtime environment.

C# History

- ❖ **Anders Hejlsberg** is known as the **founder of C# language.**
- ❖ C# has evolved much since their first release in the year **2002**.
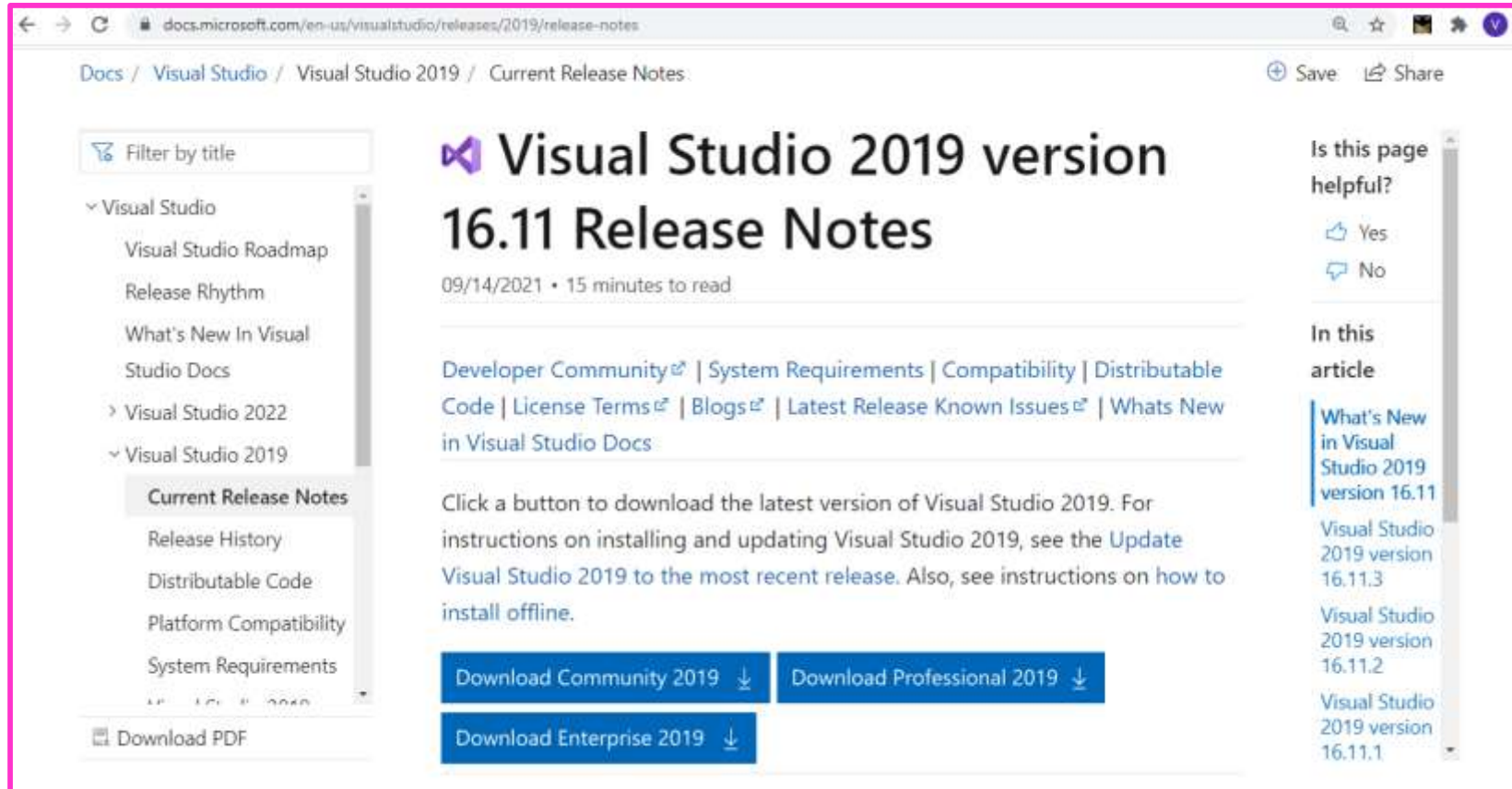- ❖ It was introduced with **.NET Framework 1.0**

C# Version

Version	Date	.Net
C# 1.0	January 2002	.NET Framework 1.0
C# 2.0	November 2005	.NET Framework 2.0
C# 3.0	November 2007	.NET Framework 3.0
C# 4.0	April 2010	.NET Framework 4
C# 5.0	August 2012	.NET Framework 4.5
C# 6.0	July 2015	.NET Framework 4.6 .NET Core 1.0 .NET Core 1.1
C# 7.0	March 2017	.NET Framework 4.7
C# 8.0	September 2019	.NET Core 3.0 .NET Core 3.1
C# 9.0	September 2020	.NET 5.0

Applications Developed In c#

> Video Games. C# is extremely popular in many sectors of the gaming industry.

> Anti-Hacking Software.

> Windows Apps (i.e.: Microsoft Office, Skype, Photoshop) .

> Mobile Apps.

> Windows Store Apps.

> cloud-based services.
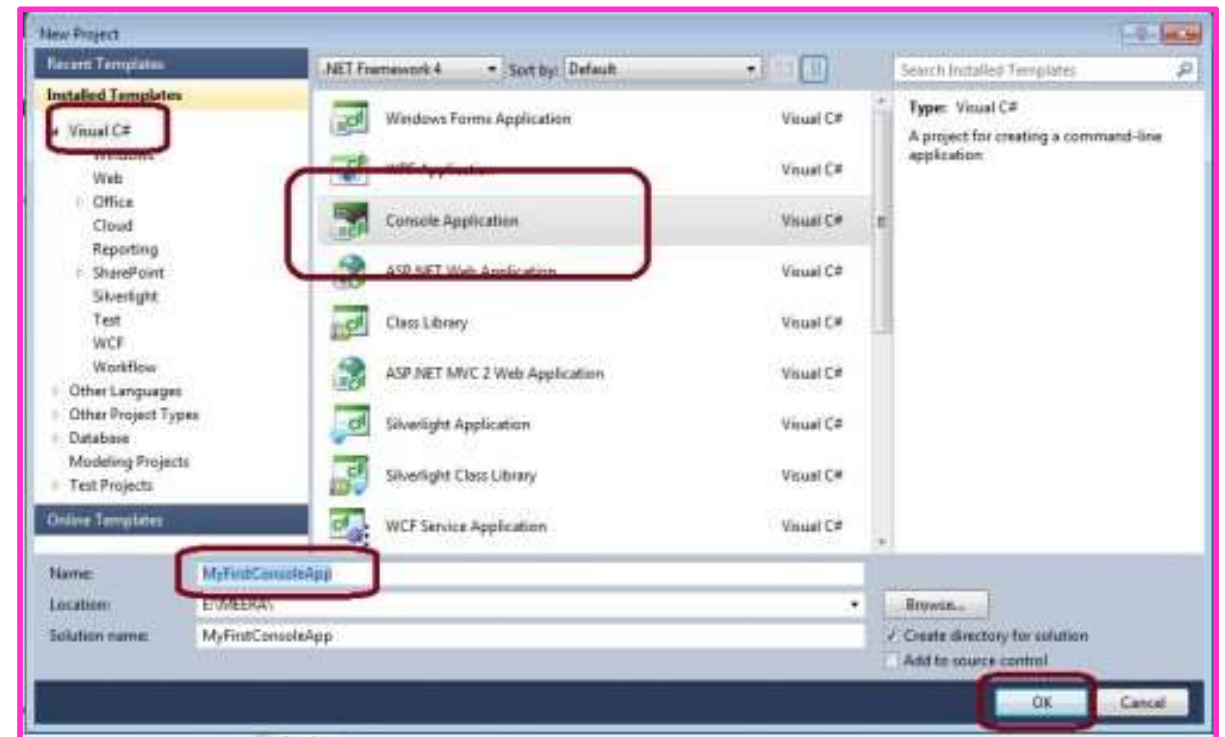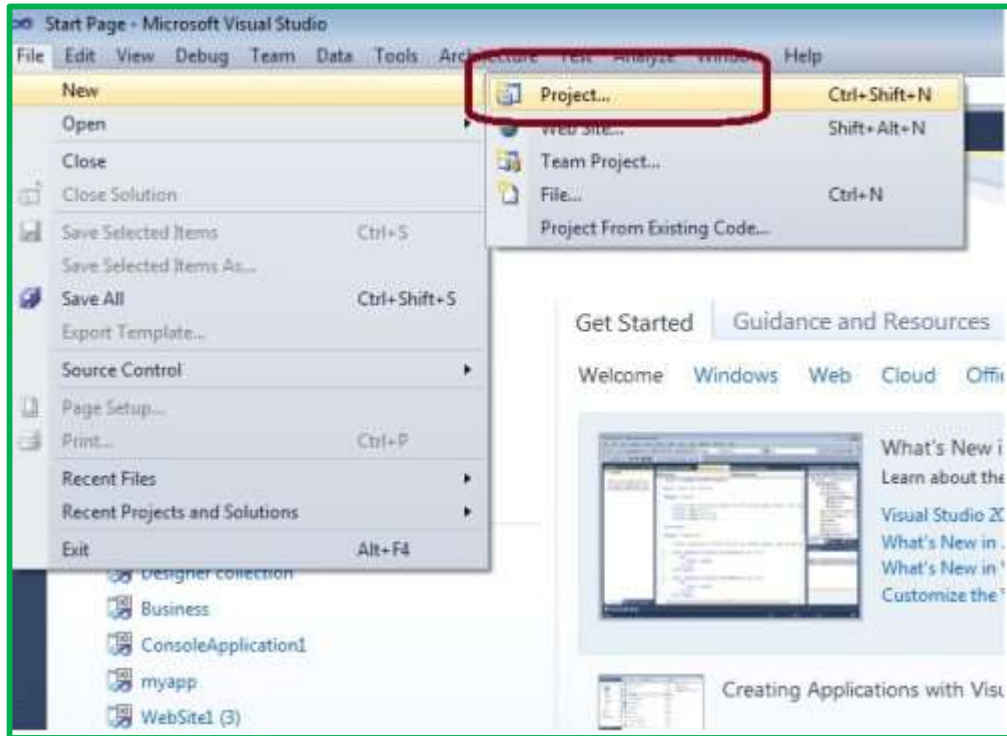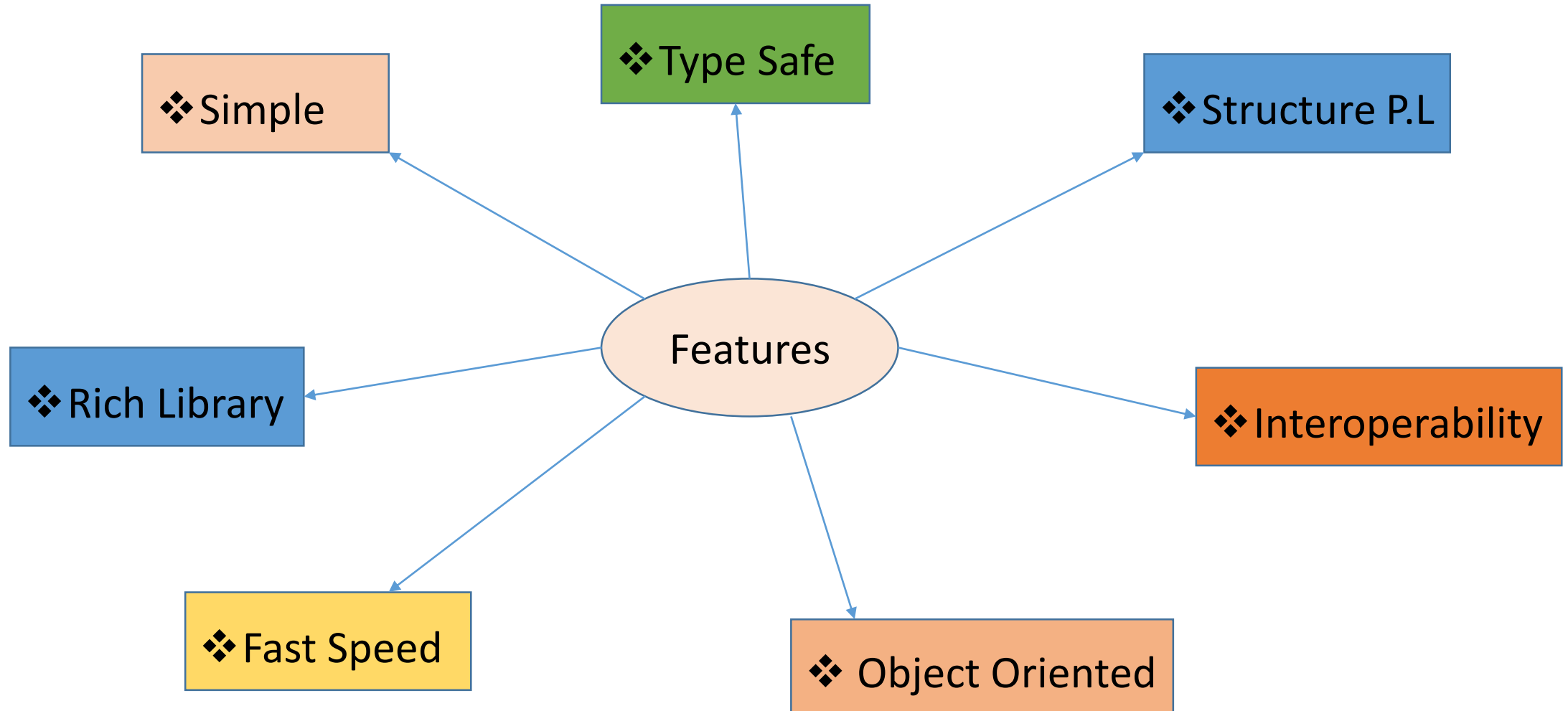
> websites, enterprise software

V S Installation

https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=community&rel=16&utm_medium=microsoft&utm_source=docs.microsoft.com&utm_campaign=download+from+relnotes&utm_content=vs2019ga+button

Create New Project

First Project

FortuneCloud
We Understand Technology

Keyword

Class name

Main Method

```
class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
```

WriteLine() is the static method of Console class which is used to write the text on the console.

CLI and CLR

C# code → Compiler
VB.NET code → Compiler
F# code → Compiler

Common Language Infrastructure

Common Intermediate Language

.NET compatible languages compile to a second platform-neutral language called Common Intermediate Language (CIL).

Common Language Runtime

The platform-specific Common Language Runtime (CLR) compiles CIL to machine-readable code that can be executed on the current platform.

010011001010111101010110100110

FortuneCloud
We Understand Technology

Data Type

Variable

> It is used to store data. Its value can be changed and it can be reused many times.

❖ String message="WELCOME ";
   Console.WriteLine(message);

❖ int val=30;
   Console.Write(val);

> String Concatenate:-

string a = "WELCOME";
string b = " TO INDIA";
Console.WriteLine(a+b);

Example

```
Console.WriteLine("Enter the value");

int a =Convert.ToInt32(Console.ReadLine());

Console.WriteLine(a);
```

```
Console.WriteLine("Enter the value");

float a =Convert.ToSingle(Console.ReadLine());

Console.WriteLine(a);
```

Error:-Cannot implicitly convert type 'string' to 'int'

Operator

Unary Operator	⟶ ++, --	Unary Operator
Ternary Operator	⟶ ?:	Ternary or Conditional Operator

Binary Operator	+, -, *, /, %	Arithmetic Operators
	<,<=, >, >=, ==, !=	Relational Operators
	&&, ||, !	Logical Operators
	&, |, <<, >>, ~, ^	Bitwise Operators
	=, +=, -=,*=, /=, %=	Assignment Operators

Example

Relational Operator:-

- int x = 5, y = 10;
- float f = 5.3f;

- Console.WriteLine(3 < 2);
- Console.WriteLine(x < y);
- Console.WriteLine(x < f);
- Console.WriteLine(x == y);
- Console.WriteLine(x != y);

Logical Operator :-

- Console.WriteLine(!(5 > 3));
- Console.WriteLine((5 > 2) && (10 < 5));
- Console.WriteLine((5 > 2) || (10 < 5));

FortuneCloud
We Understand Technology

Type Casting

✓ When the variable of one data type is changed to another data type is known as the **Type Casting**.
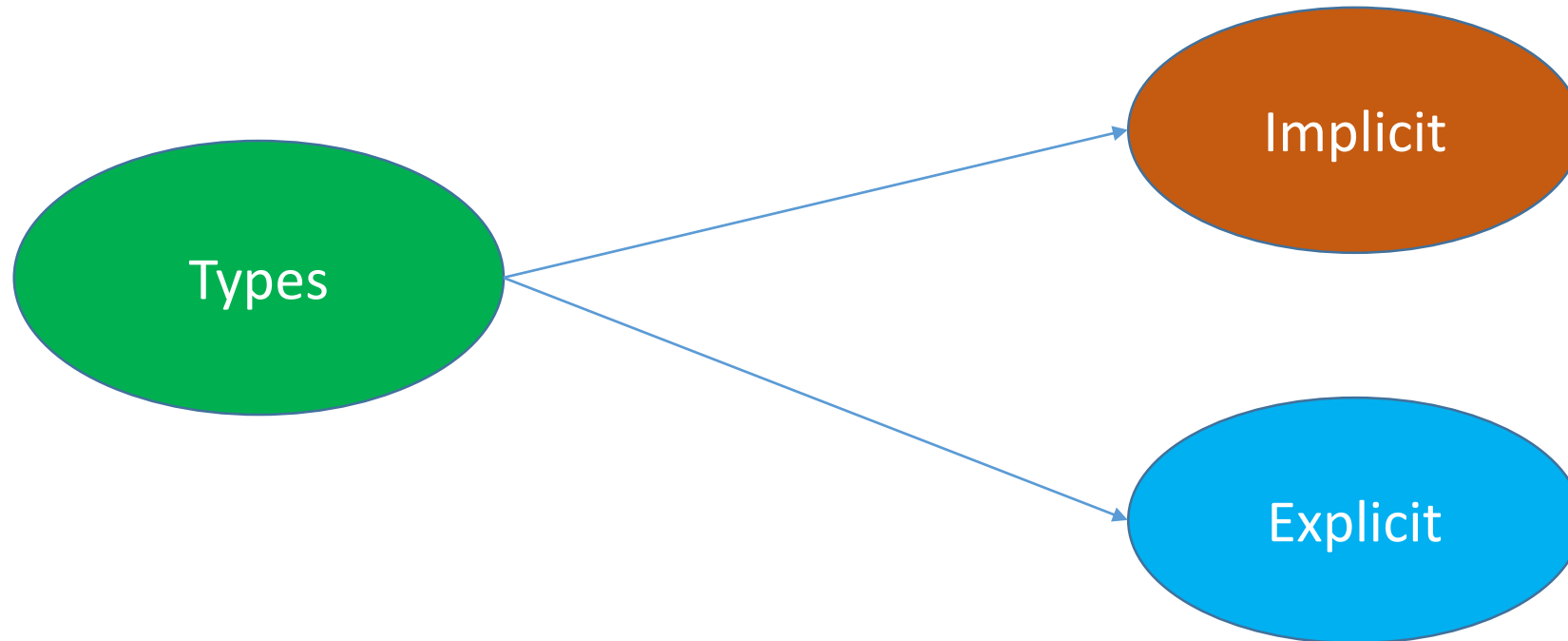
Implicit Casting

FortuneCloud
We Understand Technology

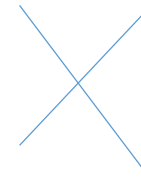- ✓ **Implicit Casting (automatically)** - converting a smaller type to a larger type size
- ✓ **char -> int -> long -> float -> double**

```
 int val = 10;
float b = val;
Console.WriteLine(b);
```

```
Double val = 10;
float b = val;
```

Explicit Casting (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char

```
Double val = 10;
float b = (float)val;
```

Keywords

Abstract	As	Base	Bool	Break	Ulong	Unit
Byte	Case	Catch	Char	Event	Value	While
Explicit	Extern	False	Finally	Fixed	Set	Readonly
Float	For	Foreach	Namespace	New	Short	Ref
Null	Object	Operator	Out	Override	Unchecked	Return
Params	Private	Static	String	Struct	Virtual	Sbyte
Switch	This	Throw	True	Try	Sizeof	Sealed
Typeof	Checked	Class	Const	Continue	Unsafe	Using
Decimal	Default	Delegate	Double	Do	Volatile	
Else	Enum	Get	Goto	If	Stackalloc	
Implicit	In	Int	Interface	Internal	Ushort	
Is	Lock	Long	Protected	Public	Void	

Control Statements

Control
Statements

Decision
Making

if
else if
switch

Loops

for
while
do-while
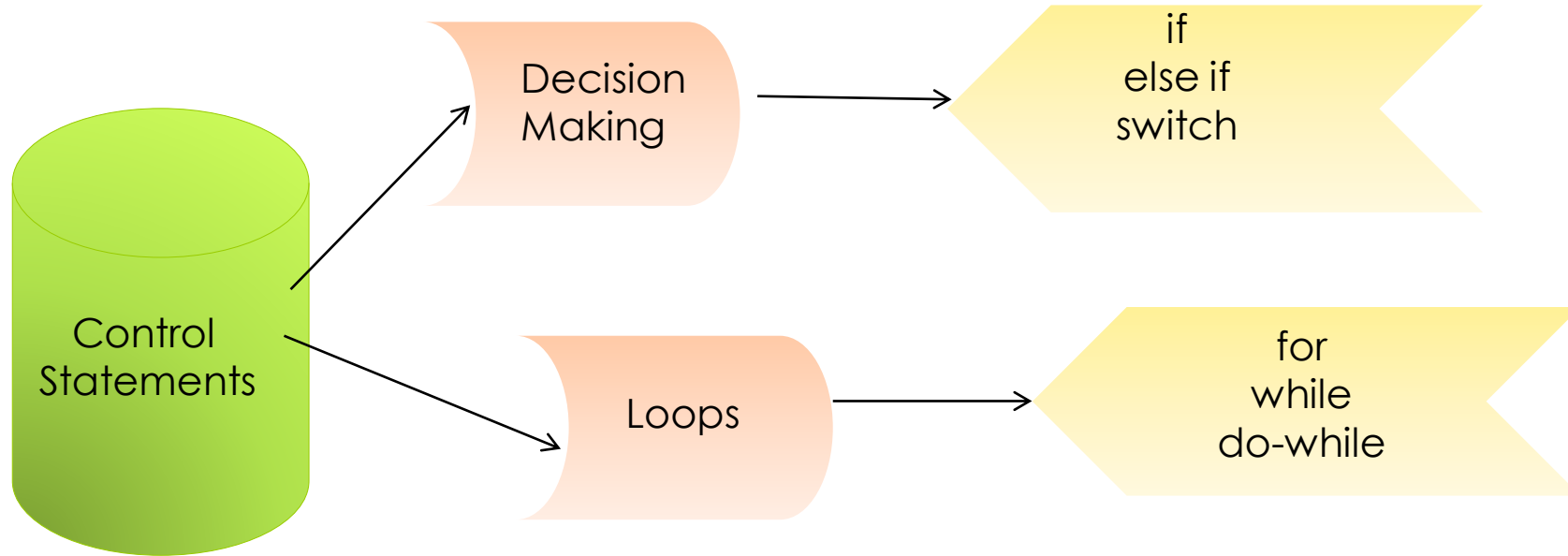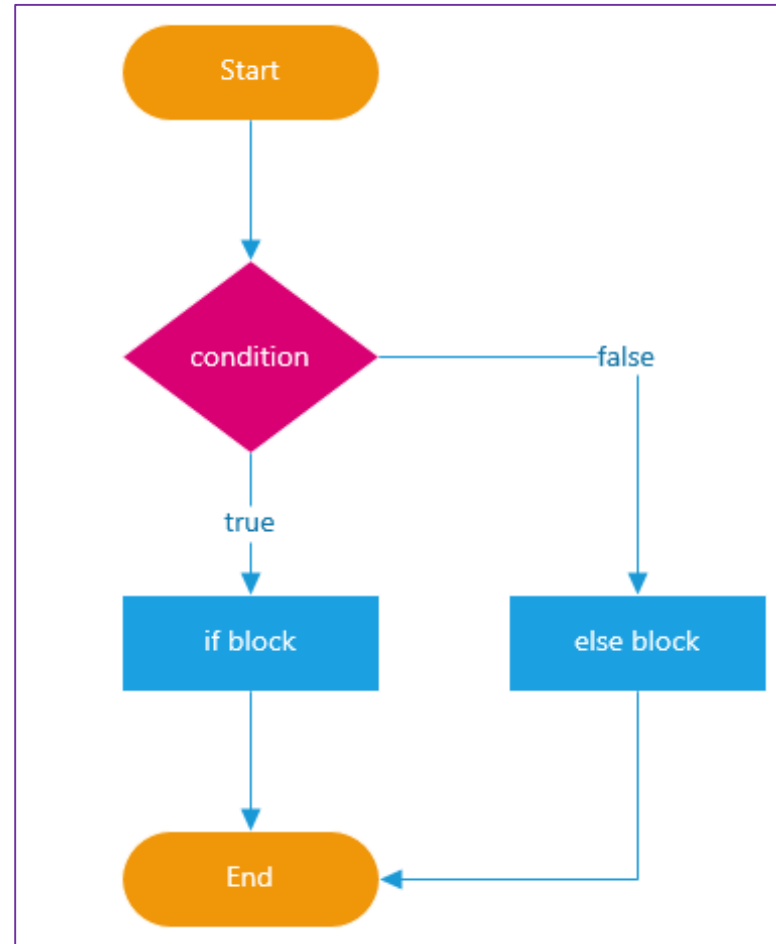
if-else

Example

```csharp
int num = 11;
if (num % 2 == 0)
{
    Console.WriteLine("It is even number");
}
else
{
    Console.WriteLine("It is odd number");
}
```

IF-else-if ladder Statement:-

```
if(condition1){

 //code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false


}
```

- ✓ When we have only one condition to test, if-then and if-then-else statement works fine.
- ✓ But what if we have a multiple condition to test and execute one of the many block of code.

```
Console.WriteLine("Enter a number to check grade:");
        int num = Convert.ToInt32(Console.ReadLine());


        if (num <0 || num >100)
        {
            Console.WriteLine("wrong number");
        }
        else if (num >= 50 && num < 60)
        {
            Console.WriteLine("First class");
        }

        else if (num >= 60 && num < 75)
        {
            Console.WriteLine("Distinction");
        }
     else{
    Console.WriteLine("Condition not true");  }
```

FortuneCloud
We Understand Technology

```
switch(expression/Variable){
case value1:
 //code to be executed;
 break;

case value2:
 //code to be executed;
 break;
......

default:
 //code to be executed if all cases are not matched;
 break;
}
```

✓ it finds the matching value, the statements inside that case are executed.

Example:-

```
        char ch;
        Console.WriteLine("Enter an
alphabet");
        ch =
Convert.ToChar(Console.ReadLine());

        switch(Char.ToLower(ch))
        {
            case 'a':
                Console.WriteLine("Vowel");
                break;
            case 'e':
                Console.WriteLine("Vowel");
                break;
            case 'i':
                Console.WriteLine("Vowel");
                break;
            case 'o':
                Console.WriteLine("Vowel");
                break;
            case 'u':
                Console.WriteLine("Vowel");
                break;
            default:
                Console.WriteLine("Not a
vowel");
                break;
        }
```

```
for(initialization; condition; incr/decr){
//code to be executed
}
```

✓ execute certain block of statements for a specified number of times.

Ex.
compute sum of first n natural numbers

```
public static void Main(string[] args)
{
        int n = 5,sum = 0;

        for (int i=1; i<=n; i++)
        {
                // sum = sum + i;
                sum += i;
        }


Console.WriteLine("Sum of first {0} natural numbers = {1}", n, sum);}
```

✓ execute a block of code as long as the specified condition returns false.

```
                    int i = 0; // initialization

        while (i < 10) // condition
        {
                    Console.WriteLine("i = {0}", i);

                    i++; // increment
        }
```

Do While

Syntax:-

```
do
{
    //code block

} while(condition);
```
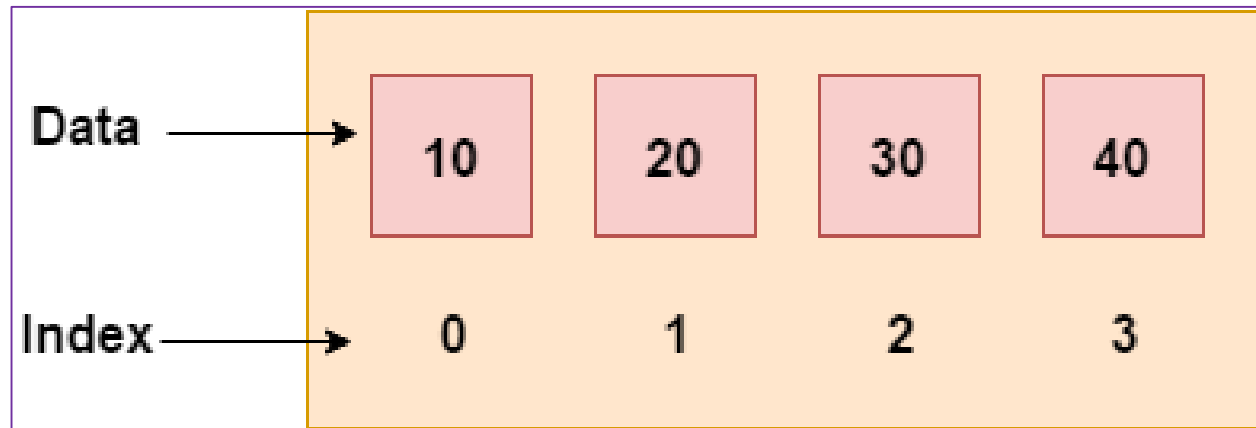
Example:-

```
int i = 0;

do
{
    Console.WriteLine(  i);
    i++;

} while (i < 5);
```

Array:-

> An array stores a fixed-size sequential collection of elements of the same type.

> An array is used to store a collection of data.

> stored at contiguous memory locations.

Data	10	20	30	40
Index	0	1	2	3
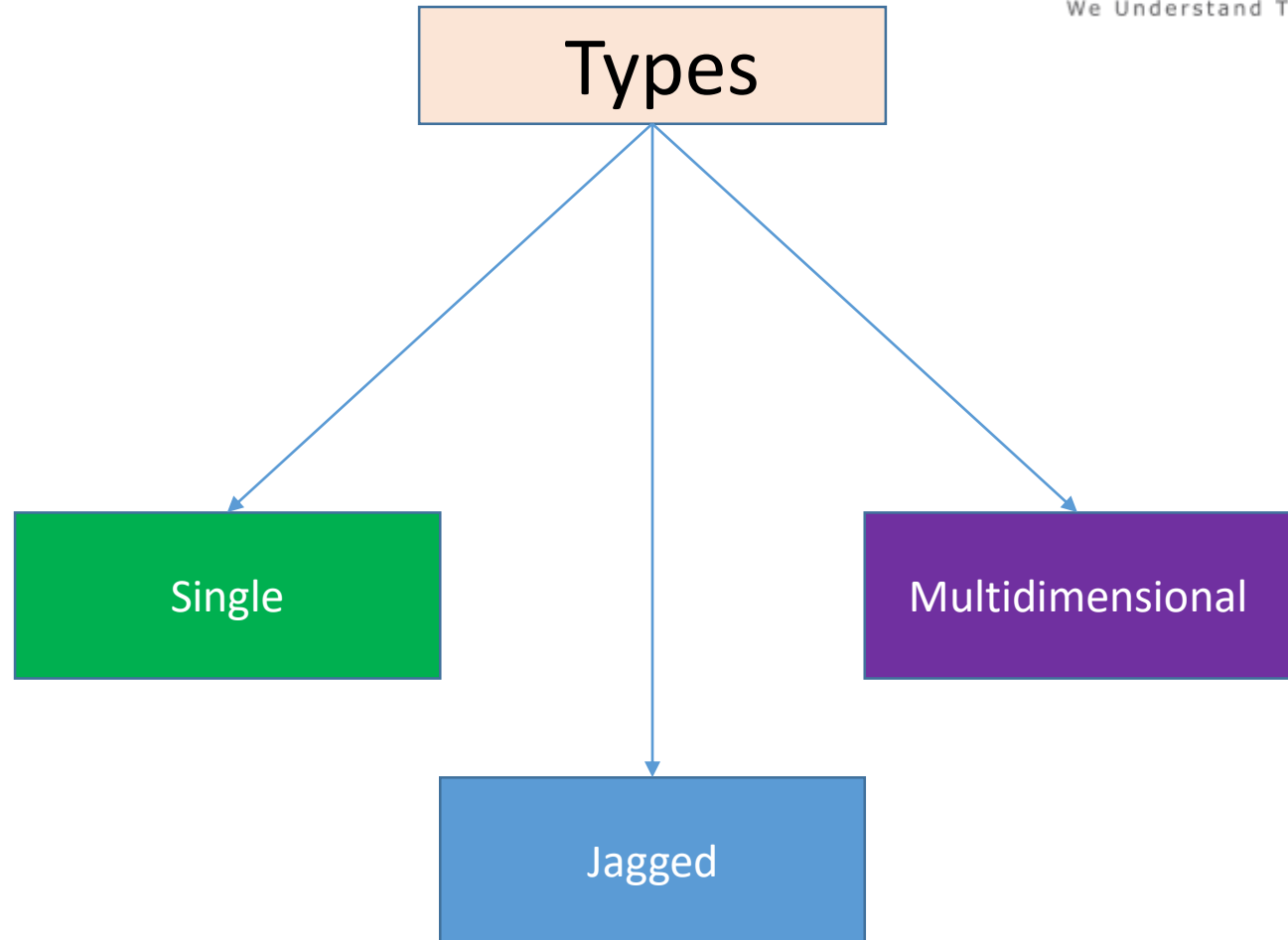
Single Dimensional Array

There are 3 ways to initialize array at the time of declaration.

int[] arr = **new int**[5]; //creating array

int[] arr = **new int**[5]{ 10, 20, 30, 40, 50 };

int[] arr = **new int**[]{ 10, 20, 30, 40, 50 };

int[] arr = { 10, 20, 30, 40, 50 };

```
namespace DemoApplication {
    class Program
    {
        static void Main(string[] args)
        {

            Int32[] values;
                 1      2          3

        }
    }
}
```

Variable name

Data type

Shows this is an array

```
        static void Main(string[] args)
        {

            Int32[] values;

            values = new Int32[3];   1

            values[0] = 1;
            values[1] = 2;   2
            values[2] = 3;

        }
    }
```

Assigning values to the elements of the array

Array is going to hold 3 elements

```
values = new Int32[3];

values[0] = 1;
values[1] = 2;
values[2] = 3;

Console.WriteLine(values[0]);
Console.WriteLine(values[1]);
Console.WriteLine(values[2]);

Console.ReadKey();
```

Using Console.WriteLine to send each element value to the console

```
public static void Main(string[] args)
  {
      int[] arr = new int[5];//creating array
      arr[0] = 10;//initializing array
      arr[2] = 20;
      arr[4] = 30;

      //traversing array
      for (int i = 0; i < arr.Length; i++)
      {
          Console.WriteLine(arr[i]);
      }
  }
```

Multidimensional Arrays

➤ The multidimensional array is also known as rectangular arrays in C#.

➤ The data is stored in tabular form (row * column) which is also known as matrix.

➤ To create multidimensional array, we need to use comma inside the square brackets.

```
int[,] arr=new int[3,3];//declaration of 2D array
int[,,] arr=new int[3,3,3];//declaration of 3D array
```

```
int[,] arr = new int[3,3]= { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

```
int[,] arr = new int[,]{ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

```
int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

```
public static void Main(string[] args)
  {
     int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };//decl and init

     //traversal
     for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
           Console.Write(arr[i,j]+" ");
        }
        Console.WriteLine();//new line at each row
     }
```

Jagged Arrays

✓ In C#, jagged array is also known as "array of arrays" because its elements are arrays.
✓ The element size of jagged array can be different.

Syntax:-

int[][] arr = **new int**[2][];

Size of an Array

dimensions of the array

Example

```
public static void Main()
  {
     int[][] arr = new int[2][];// Declare the array

     arr[0] = new int[] { 11, 21, 56, 78 };// Initialize the array
     arr[1] = new int[] { 42, 61, 37, 41, 59, 63 };

     // Traverse array elements
     for (int i = 0; i < arr.Length; i++)
     {
        for (int j = 0; j < arr[i].Length; j++)
        {
           System.Console.Write(arr[i][j]+" ");
        }
        System.Console.WriteLine();
     }
```
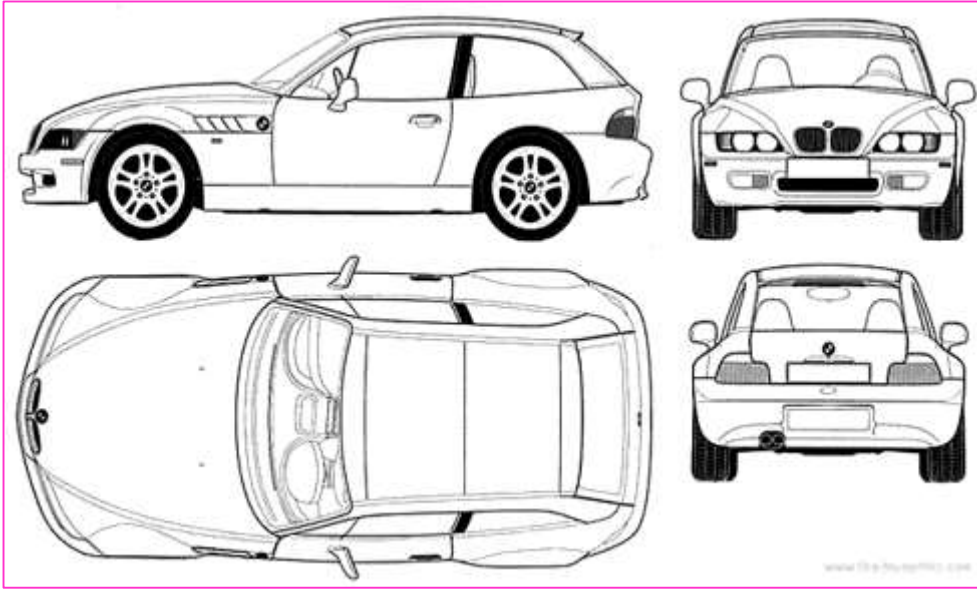
Class

❖ **Class is nothing but the collection of object**

❖ Concept of class in real time belong to programming world in object oriented language.

❖ **A class is a template or blueprint which defines the state and behavior of all object belonging to that class.**

❖ **A Class is composed of field/attributes/variables and methods collectively called as data member of class.**

❖ **Ex.student ,Employee**

❖ **Syntax to declare class :**

```
    class <classname>
{
  // data members of class
}
```

Object

- ❖ Any entity that has identity, state and behaviour.

- ❖ Identity of object distinguishes it from other object of same type.

- ❖ State of object refer to its characteristics or attributes.

- ❖ Behavior of object comprises its action

- ❖ The object stores its identity and state in a variable / attribute and exposes its behavior through method.

- ❖ Examples: a car,a persons, etc.

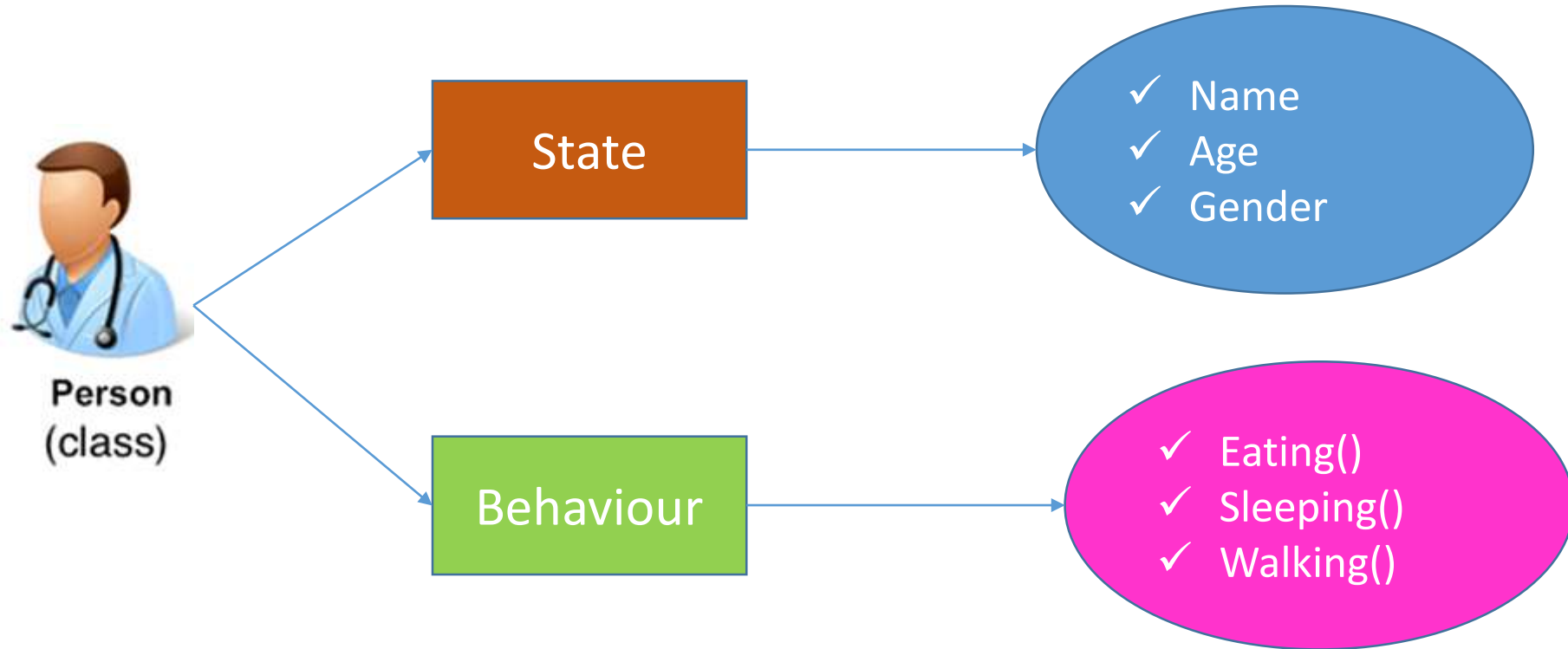- ❖ Syntax:-<classname> <objectname> = new <classname>();

Car (Blueprint)

Object(Real Car)

```
public class Student
  {
      int id; //data member (also instance variable)
      String name; //data member(also instance variable)

  public static void Main(string[] args)
    {
        Student s1 = new Student() ;//creating an object of Student
        s1.id = 101;
        s1.name = "KARAN";
        Console.WriteLine(s1.id);
        Console.WriteLine(s1.name);


    }
  }
```

✓ Function is a block of code that has a signature.

✓ Function is used to execute statements specified in the code block.

```
<access-specifier><return-type>FunctionName(<parameters>)
{
// function body
// return statement
}
```

```csharp
class Program
   {

      public void Show() // No Parameter
      {
         Console.WriteLine("This is non parameterized function");
            }


      // Main function, execution entry point of the program
      static void Main(string[] args)
      {
         Program program = new Program(); // Creating Object
         program.Show(); // Calling Function
      }
   }
```

❖ A special method of the class that will be automatically invoked when an instance of the class is created is called a constructor.

❖ constructor has the same name as class name.

❖ When you have not created a constructor in the class, the compiler will automatically create a default constructor in the class.

❖ NOTE:-**Constructor has not a return type because implicit return type of constructor is class itself.**

Rules for Constructor

✓ A class can have any number of constructors.

✓ A constructor doesn't have any return type, not even void.

✓ A static constructor can not be a parameterized constructor.

✓ Within a class you can create only one static constructor.

✓ **For constructor default access modifier is public. We can change it to private.**
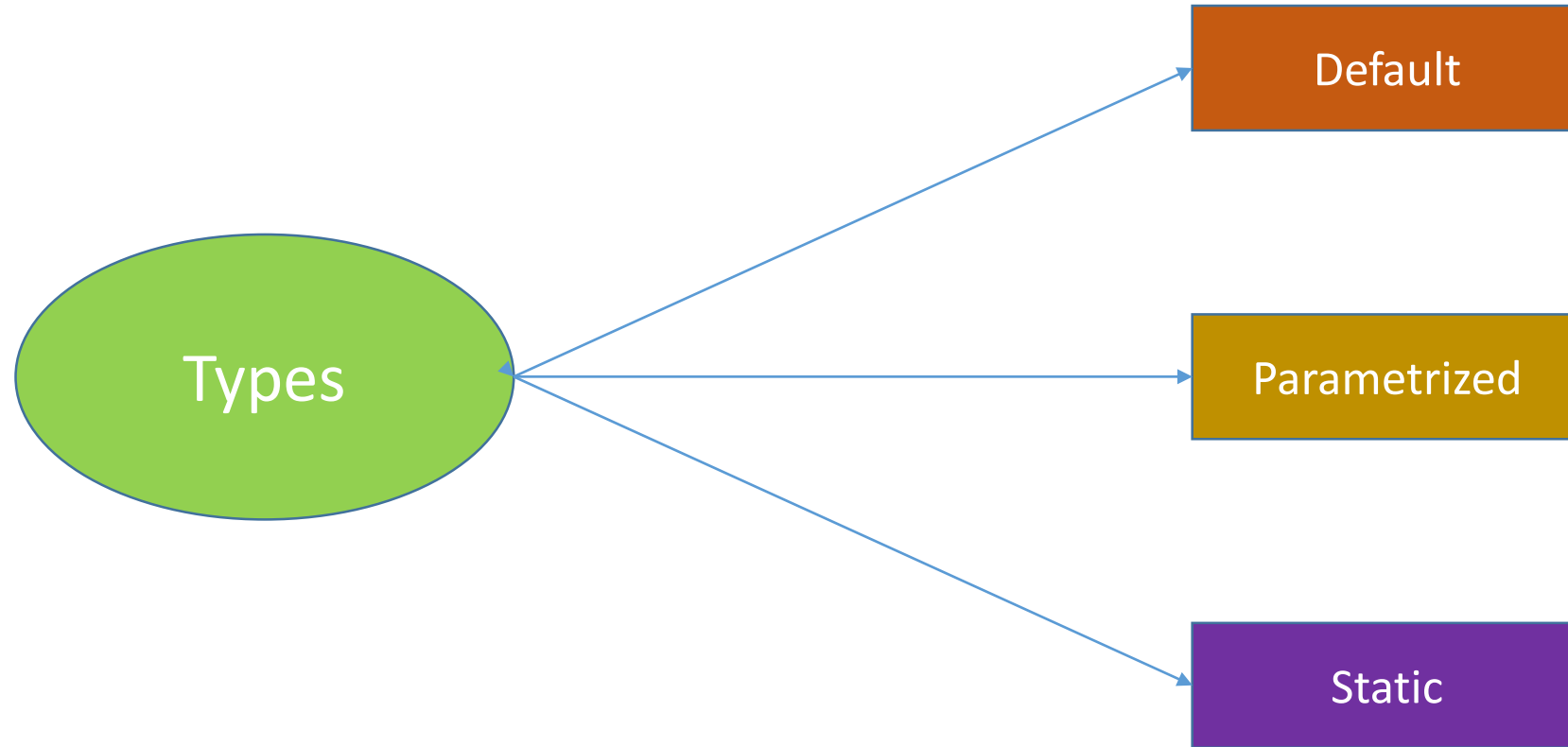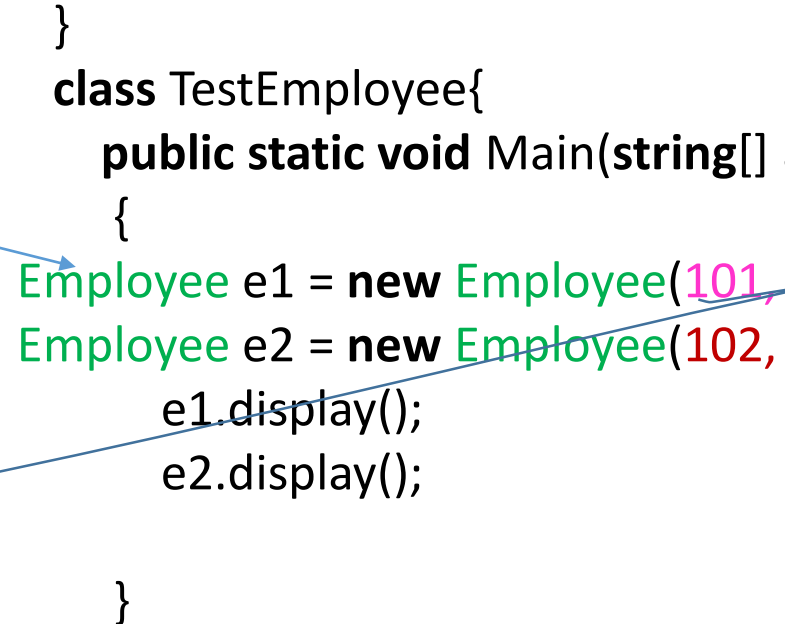
Default Constructor Example

```csharp
public class Employee
  {
     public Employee()
     {
Console.WriteLine("Default Constructor Invoked" );
     }
  }
  class TestEmployee{
     public static void Main(string[] args)
     {
         Employee e1 = new Employee();
         Employee e2 = new Employee();
     }
}
```

Parameterized Constructor Example

```
public class Employee
{
    public int id;
    public String name;
    public float salary;
    public Employee(int i, String n,float s)
    {
        id = i;
        name = n;
        salary = s;
    }
    public void display()
    {
        Console.WriteLine(id + " " + name+" "+salary);
    }
}
class TestEmployee{
    public static void Main(string[] args)
    {
        Employee e1 = new Employee(101, "Sonoo", 8900f);
        Employee e2 = new Employee(102, "Mahesh", 4900f);
            e1.display();
            e2.display();

    }
}
```

Destructor

✓ A destructor works opposite to constructor.
✓ It can be defined only once in a class. Like constructors, it is invoked automatically.
✓ Destructors are usually **used to deallocate memory and do other** clean-up for a class object and its class members when the object is destroyed.
✓ A destructor has the same name as the class, used by a tilde (~).

Note:-C# destructor cannot have parameters. modifiers can't be applied on destructors.

Examples

```csharp
public class Employee
{
    public Employee()
    {
        Console.WriteLine("Constructor Invoked");
    }
    ~Employee()
    {
        Console.WriteLine("Destructor Invoked");
    }
}
class TestEmployee{
    public static void Main(string[] args)
    {
        Employee e1 = new Employee();

    } }
```

✓ It can be used **to refer current class instance variable**.

✓ It can be used **to pass value as a parameter to another method**.
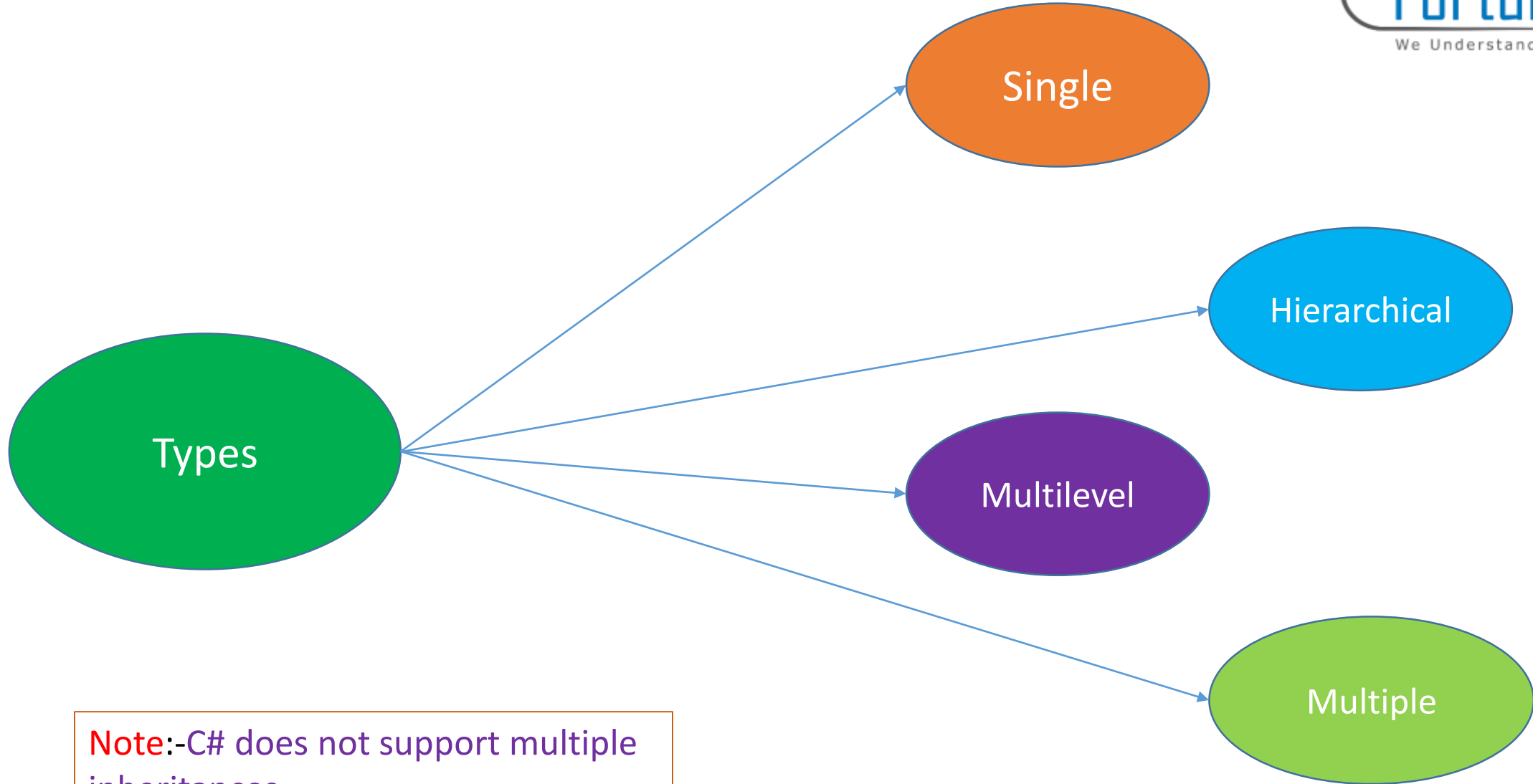
Example

```
public class Employee
  {
     public int id;
     public String name;
     public float salary;
     public Employee(int id, String name,float salary)
     {
        this.id = id;
        this.name = name;
        this.salary = salary;
     }
     public void display()
     {
        Console.WriteLine(id + " " + name+" "+salary);
     }
  }
```

```
class TestEmployee{
    public static void Main(string[] args)
    {
        Employee e1 = new Employee(101, "Karan", 8900f);
        Employee e2 = new Employee(102, "Mahesh", 4900f);

        e1.display();
        e2.display();

    }
}
```
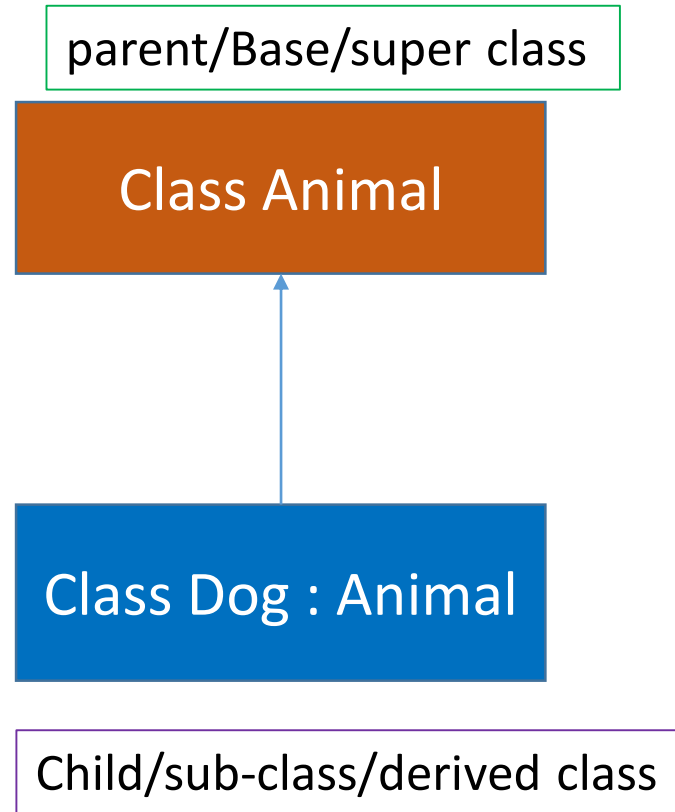
Inheritance

✓ It is the process of acquiring the methods and attributes of the base class.

✓ The new class is called as derived class.

✓ It is the process of creating new class from existing class.

✓ The reason behind OOP programming is to promote the reusability of code and to reduce complexity in code and it is possible by using inheritance.

✓ Acquiring (taking) the properties of one class into another class is called inheritance.

✓ Inheritance provides reusability by allowing us to extend an existing class.

Types of Inheritance:-

Types

Single

Hierarchical

Multilevel

Multiple

Note:-C# does not support multiple inheritances

Single Inheritance Example

parent/Base/super class

Class Animal

Class Dog : Animal
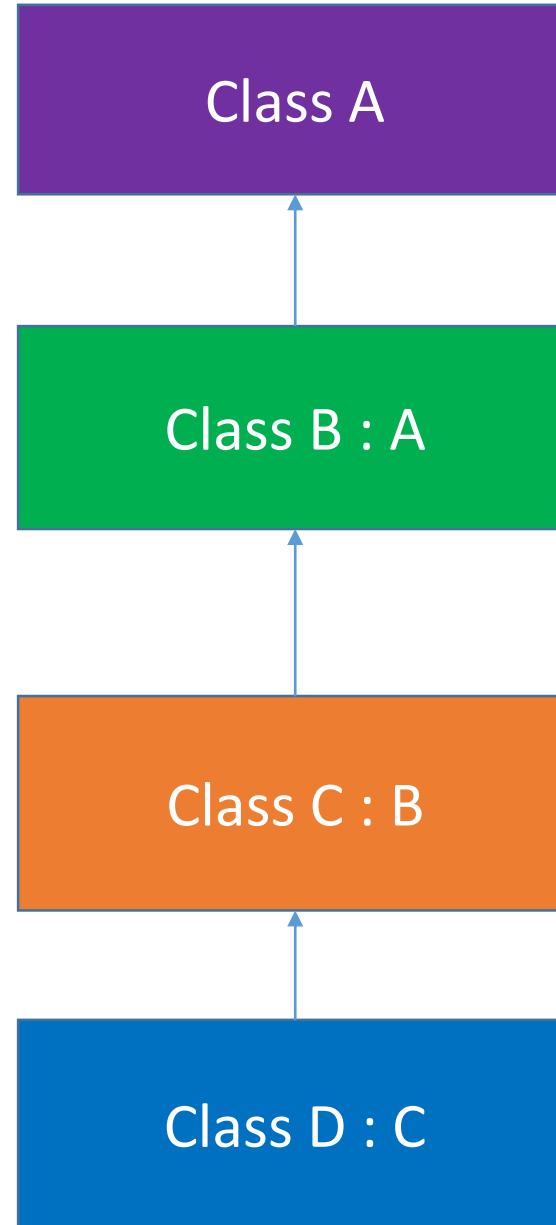
Child/sub-class/derived class

```
public class Animal
 {
     public void eat() { Console.WriteLine("Eating..."); }
}
public class Dog: Animal
{
     public void bark() { Console.WriteLine("Barking..."); }
}
class TestInheritance2{
     public static void Main(string[] args)
     {
         Dog d1 = new Dog();
         d1.eat();
         d1.bark();
     }
}
```
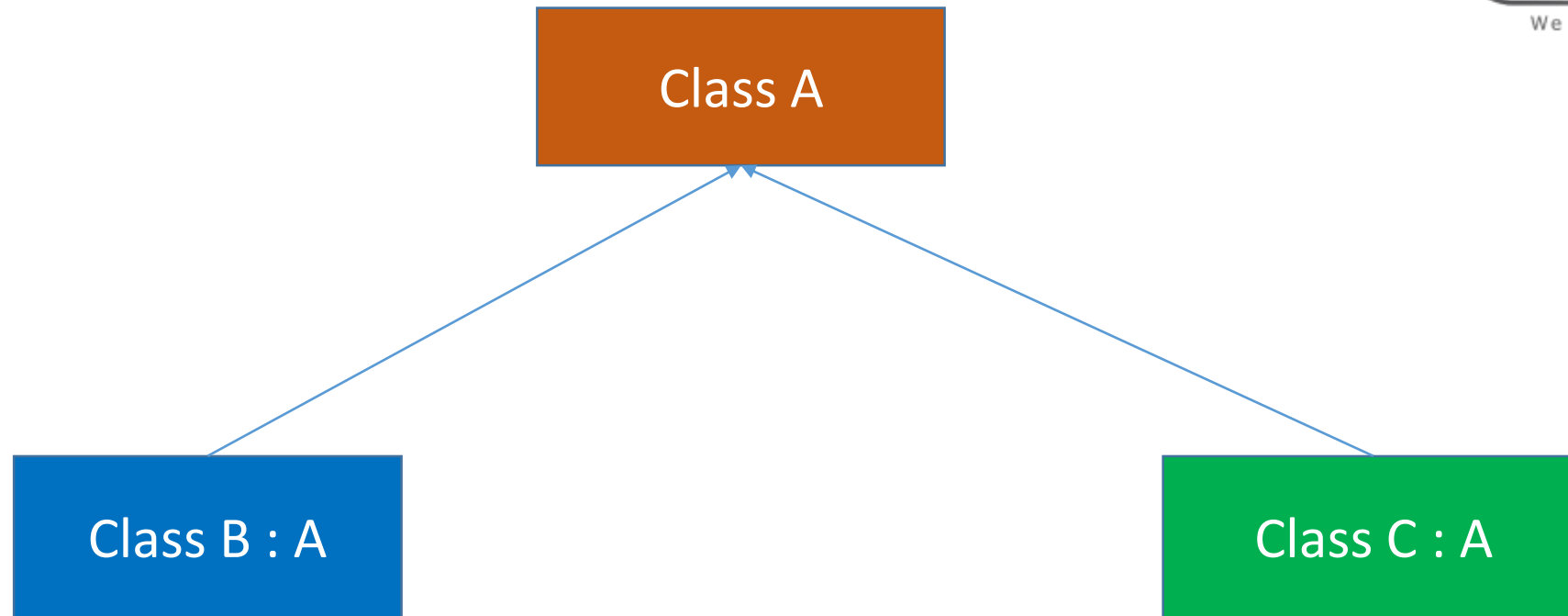
Aggregation(HAS-A) Relationship

- ➢ Aggregation is a process in which one class defines another class as any entity reference.

- ➢ It is another way to reuse the class.

- ➢ It is a form of association that represents HAS-A relationship.

HAS-A

Loosely Coupled

Driver

```
public class Driver
  {
    public int id;
     public string name;

    public Driver(int id, string name)
    {
        this.id = id;
        this.name = name;
    }
  }
```

```
public class car
  {

    Driver d;
    public string car_name;

    public car(string car_name,Driver d)
    {
        this.car_name = car_name;
        this.d = d;
    }
    public void display()
    {

        Console.WriteLine("car name:-" + car_name);
        Console.WriteLine("id:-"+d.id+"\nname:-"+d.name);

    }
  }
```

```
class Myclass
    {
        static void Main(string[] args)
        {
            Driver d = new Driver(1, "RAJ");
            car c = new car("BMW",d);
            c.display();
            Console.ReadKey();


        }
    }
```
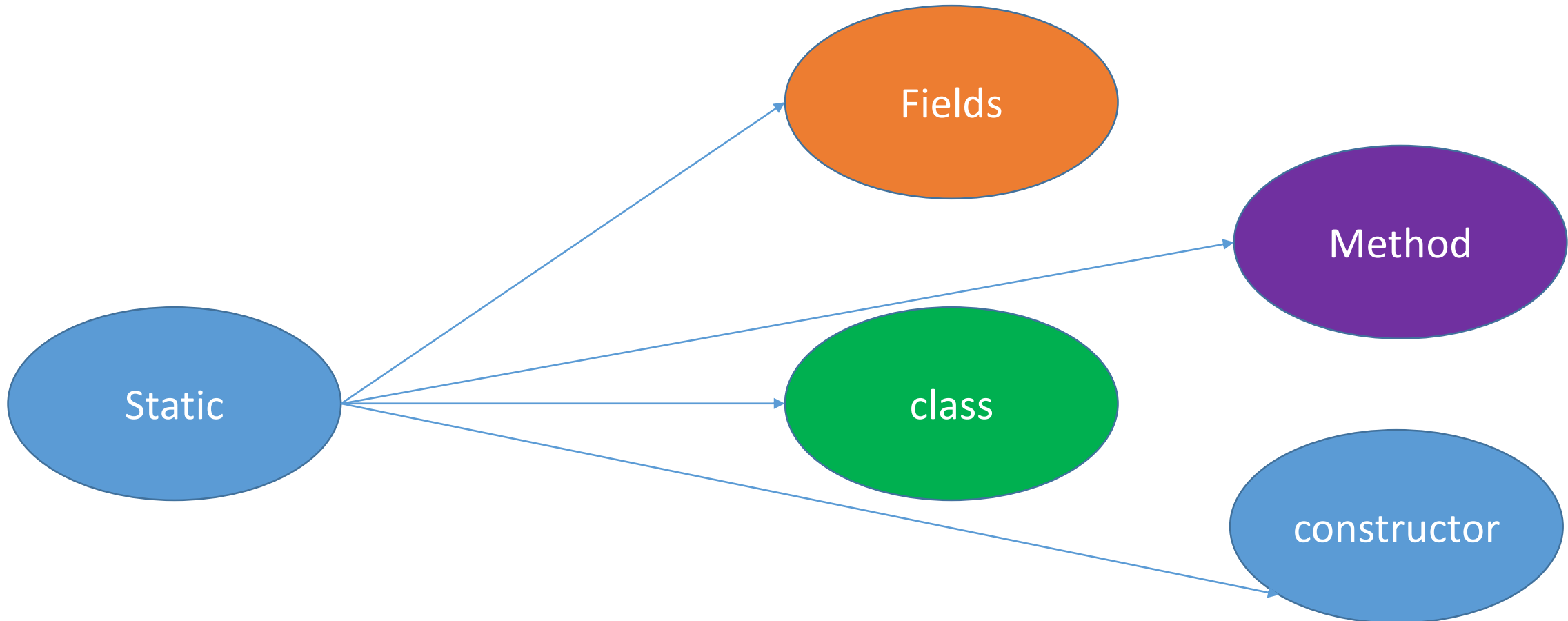
Fields

✓ Static field gets memory only once at the time of class loading.
✓ The common property among all the objects is referred by static field such as company name in case of employees, college name in case of students.

```
class Myclass
  {
      int a = 330;//non-static fields
      static void Main(string[] args)
      {
        Console.WriteLine(a);
        Console.ReadKey();

      }
  }
```

Make it static

```csharp
class Student
  {
    int id;
    string course, name;
    static string college_name = "DY PATIL";
    public Student(int id, string name, string course)
    {
      this.id = id;
      this.name = name;
      this.course = course;
    }
    public void show()
    {
      Console.WriteLine(id + " name= " + name + " course=
" + course + " college_name= " + college_name);
    }
    static void Main(string[] args)
    {
      Student s1 = new Student(101, "Ravi Dubey",
"M.C.A.");
      Student s2 = new Student(102, "KARAN Malik", "B.
Tech.");
      s1.show();
      s2.show();

      Console.ReadKey();
    }
```

Static Method

- ✓ No required object u can directly called method.
- ✓ Non-static method can't give the access of inside static method

```
class HelloWorld
{
   static  void display()
  {
     Console.WriteLine("display method");
  }
  static void Main() {
         display();
      Console.WriteLine("Hello World");
 }
}
```

Static class

✓ static class is like the normal class but it cannot be instantiated. It can have only static members.

```
static class Demo{
public static int a=10;
  public static void show()
{
  Console.WriteLine("show");
  }
}
class HelloWorld {
static void Main() {
 Console.WriteLine(Demo.a);
  Demo.show();
   }
}
```

Static Constructor

✓ C# static constructor is used to initialize static fields.
✓ C# static constructor cannot have any modifier or parameter.
✓ C# static constructor is invoked implicitly. It can't be called explicitly.

```csharp
public class Account
  {
      public int id;
      public String name;
      public static float rateOfInterest;
      public Account(int id, String name)
      {
          this.id = id;
          this.name = name;
      }
      static Account()
      {
          rateOfInterest = 9.5f;
      }
      public void display()
      {
          Console.WriteLine(id + " " + name+"
"+rateOfInterest);
      }
  }
  class TestEmployee{
      public static void Main(string[] args)
      {
          Account a1 = new Account(133,
"Karan");
          Account a2 = new Account(199,
"Mahesh");
          a1.display();
          a2.display();

      }
  }
```

Public:-

- ✓ C# Access modifiers or specifiers are the keywords that are used to specify accessibility or scope of variables and functions.
- ✓ The public keyword is an access modifier for types and type members. Public access is the most permissive access level.
- ✓ There are no restrictions on accessing public members.

private

✓ Private access is the least permissive access level.
✓ Private members are accessible only within the body of the class or the struct in which they are declared.

Accessibility
✓ Cannot be accessed by object.
✓ Cannot be accessed by derived classes.

✓ A protected member is accessible from within the class in which it is declared, and from within any class derived from the class that declared this member.

✓ A protected member of a base class is accessible in a derived class only if the access takes place through the derived class type.

✓ The internal keyword is an access modifier for types and type members.

✓ We can declare a class as internal or its member as internal. Internal members are accessible only within files in the same assembly (.dll).

✓ In other words, access is limited exclusively to classes defined within the current project assembly.

- ✓ The protected internal accessibility means protected OR internal, not protected AND internal.

- ✓ In other words, a protected internal member is accessible from any class in the same assembly, including derived classes.
- ✓ The protected internal access modifier seems to be a confusing but is a union of protected and internal in terms of providing access but not restricting. It allows:

Aggregation(HAS-A)

- ✓ In C#, aggregation is a process in which one class defines another class as any entity reference.
- ✓ It is another way to reuse the class.
- ✓ It is a form of association that represents HAS-A relationship.

```
public class Address
{

    public string addressLine, city, state;
    public Address(string addressLine, string city, string state)
    {

        this.addressLine = addressLine;
        this.city = city;
        this.state = state;

    }
}
```

```csharp
public class Employee
  {
    public int id;
    public string name;
    public Address address;//Employee HAS-A Address
    public Employee(int id, string name, Address address)
    {
      this.id = id;
      this.name = name;
      this.address = address;
    }
    public void display()
    {
      Console.WriteLine(id + " " + name + " " +
        address.addressLine + " " + address.city + " " + address.state);
    }
  }
```

```
public class TestAggregation
  {
     public static void Main(string[] args)
     {
        Address a1=new Address("IT PARK
SECTOR","pune","UP");
        Employee e1 = new Employee(1,"Sonoo",a1);
        e1.display();
     }
  }
```

➢ The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a greek word.

➢ **Polymorphism allows us to perform a single action in different ways**.

➢ There are two types of polymorphism in C#: **compile time polymorphism and runtime polymorphism**.

➢ Compile time polymorphism is achieved by method overloading and operator overloading in C#.

```csharp
public class Animal{
    public virtual void eat(){
        Console.WriteLine("eating...");
    }
}
public class Dog: Animal
{

    public override void eat()
    {

        Console.WriteLine("eating bread...");
    }


}
public class TestPolymorphism
{
    public static void Main()
    {

        Animal a= new Dog();
        a.eat();
    }
}
```
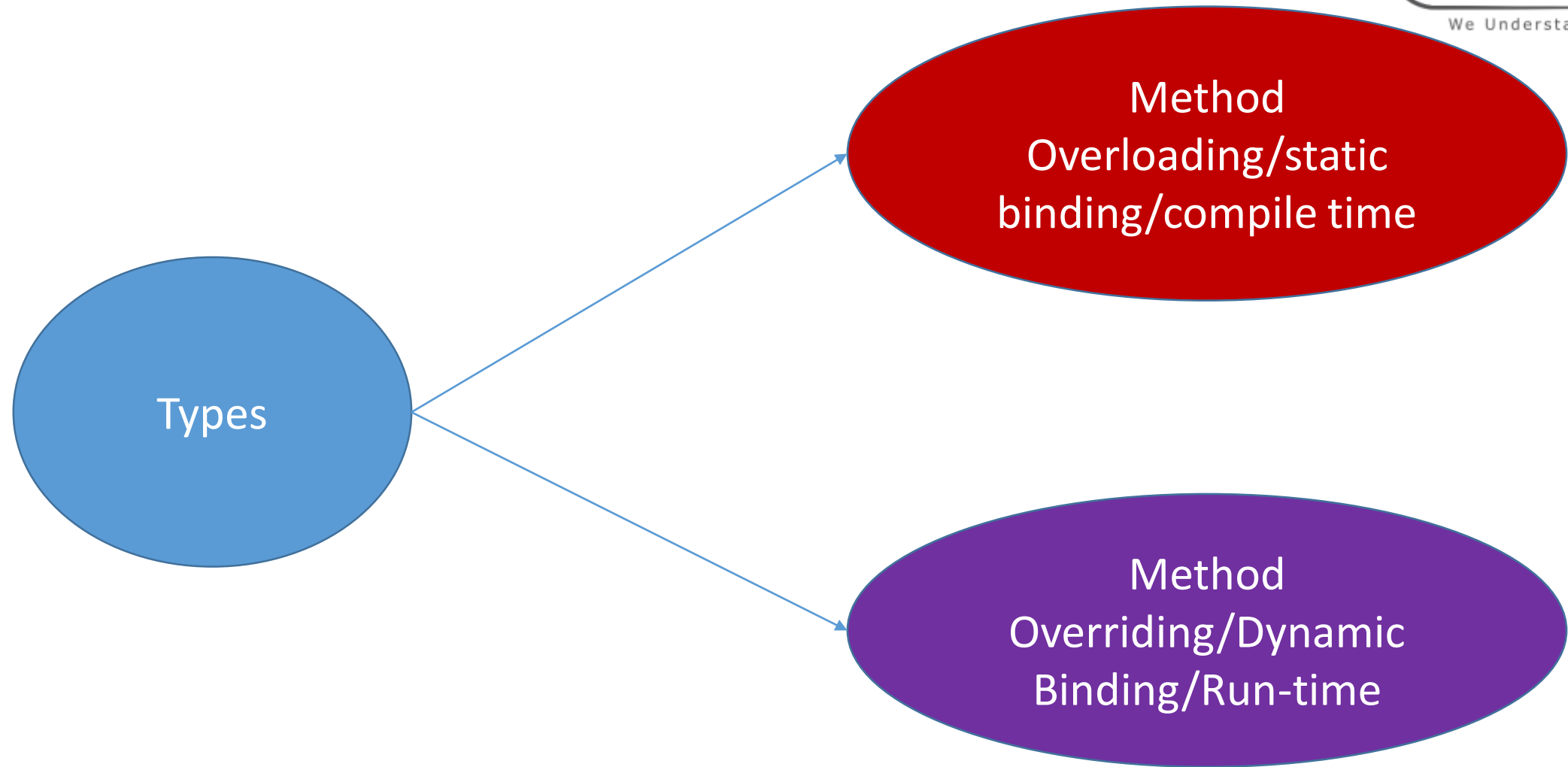
Method Overloading

✓ If we create two or more members having same name but different in number or type of parameter, it is known as member overloading.

In c# we can overload:
✓ Methods
✓ Constructors
✓ indexed properties

❖ The **advantage** of method overloading is that it increases the readability of the program because you don't need to use different names for same action.

You can perform method overloading in C# by two ways:
- ✓ By changing number of arguments
- ✓ By changing data type of the arguments

Example

```
public class Cal{
    public static int add(int a,int b){
        return a + b;
    }
    public static int add(int a, int b, int c)
    {

        return a + b + c;
    }
}
public class TestMemberOverloading
{

    public static void Main()
    {
        Console.WriteLine(Cal.add(12, 23));
        Console.WriteLine(Cal.add(12, 23, 25));
    }
}
```

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.

- To perform method overriding in C#, you need to use **virtual** keyword with base class method and **override** keyword with derived class method.

```
public class Animal{
    public virtual void eat(){
        Console.WriteLine("Eating...");
    }
}
public class Dog: Animal
{

    public override void eat()
    {
        Console.WriteLine("Eating bread...");
    }
}
public class TestOverriding
{

    public static void Main()
    {
        Dog d = new Dog();
        d.eat();
    }
}
```