Duhok Polytechnic University
Zakho Technical Institute
Department of Information Technology

Object Oriented Programming

2024-2025

Lecture 7: **Inheritance**

Lecturer

Sipan M. Hameed

Inheritance

Inheritance enables programmers to create new classes that reuse, extend, and modify the behavior that is defined in other classes.

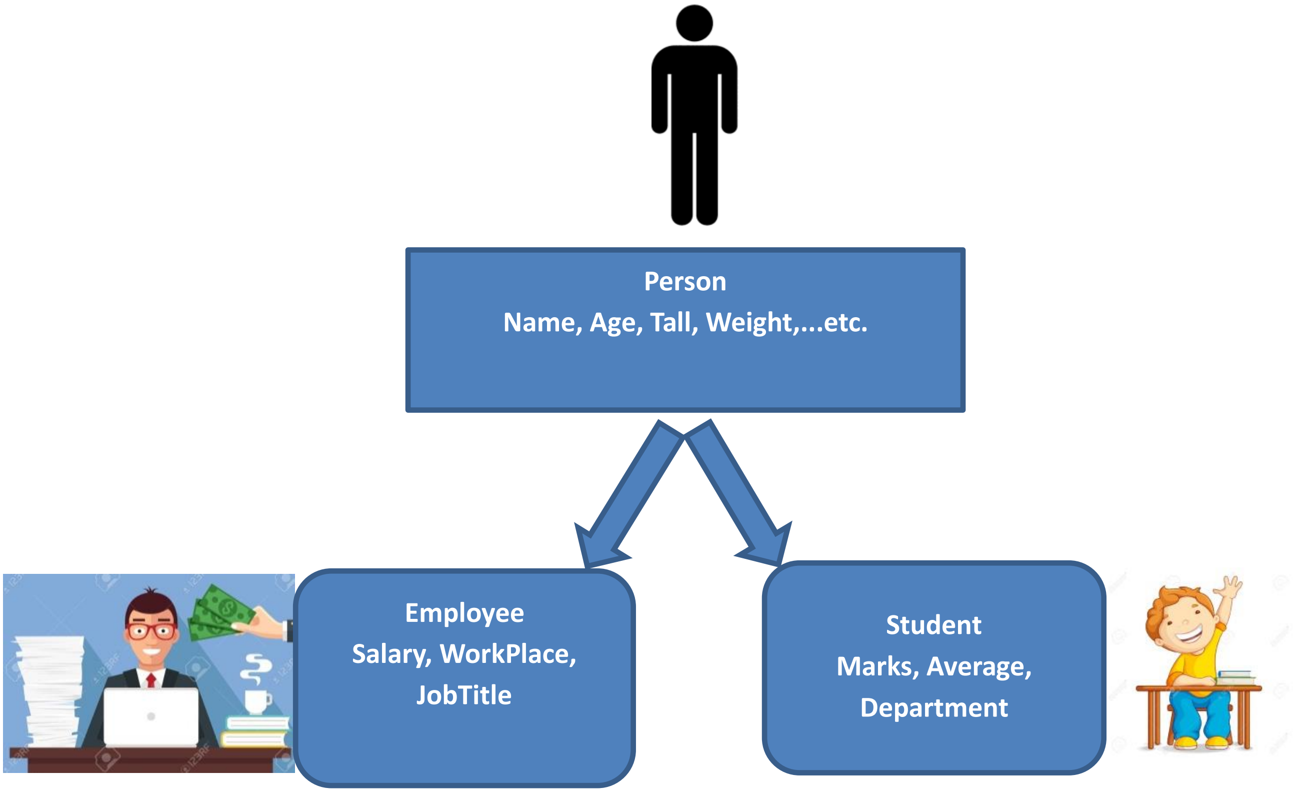
- The class whose members are inherited is called the ***base class (parent or superclass)***,
- and the class that inherits those members is called the ***derived class (child or subclass)***.
- A derived class can have **only one direct base class**.
- The derived class inherits all fields and methods of the base class, except those declared with the `private` access modifier.

However, inheritance is transitive. If Class C is derived from Class B, and Class B is derived from Class A, Class C inherits the members declared in Class B and Class A.

By inheritance, you can reuse the members (fields, methods, properties, etc…) of your parent class. So, there is no need to define the member again. So less code is required in the class.

Conceptually, a derived class is a **specialization** of the base class, For example, if we have a base class **Person**, we might have one derived class that is named **Student** and another derived class that is named **Employee**.

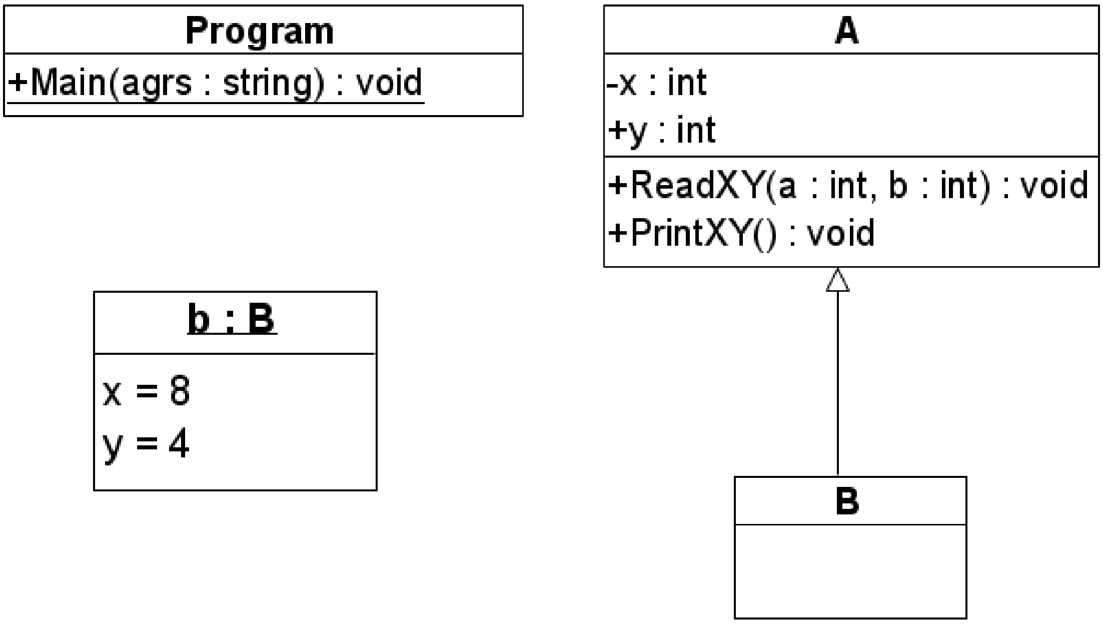
A **Student** is a **Person**, and an **Employee** is a **Person**, but each derived class represents different specializations of the base class


Person
Name, Age, Tall, Weight,...etc.
Employee
Salary, WorkPlace, JobTitle
Student
Marks, Average, Department

In C#, inheritance is performed using the **: symbol**.

Syntax of base and derived classes can be presented bellow

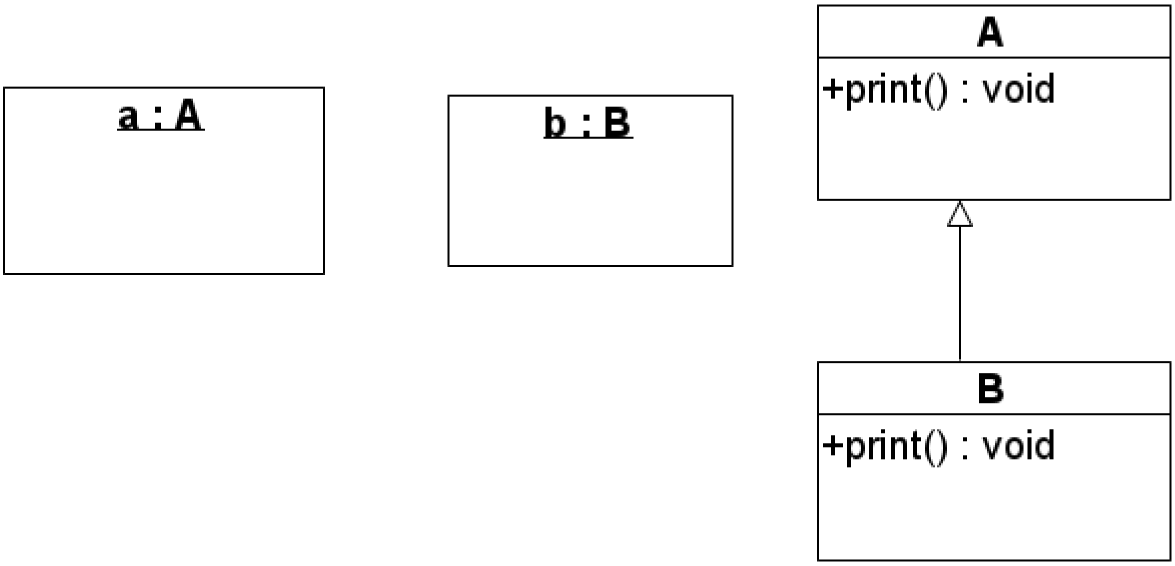
```
// Base class (parent class)
public class BaseClass
{
    // Members (Fields, properties, methods, etc).
}
// Derived class (child class) inheriting from BaseClass
public class DerivedClass : BaseClass
{
    // Members inherited from the base class
    // Additional members. specific to DerivedClass
}
```



Example:

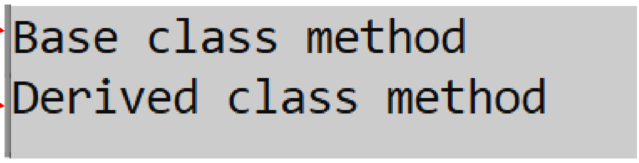
```
class A
{
    private int x;
    public int y;
    public void ReadXY(int a, int b)
    {
        x = a;
        y = b;
    }
    public void PrintXY()
    {
        Console.WriteLine(" X : {0 }  ,  Y : {1} ", x, y);
    }
} //end of class A

class B : A
{
} //end of class B

class Program
{
    static void Main(string[] args)
    {
        B b = new B();
        b.ReadXY(9, 8);
        b.PrintXY();
        b.x = 5;   //error   x is private
        b.y = 4;
    }
}
```

In the example above, we have a class **A** that contains some members. We then created another class, **B**, which inherits some members from the base class (**A**). Creating an object of class B allows you to access all inherited members, except those declared as private.



Example:

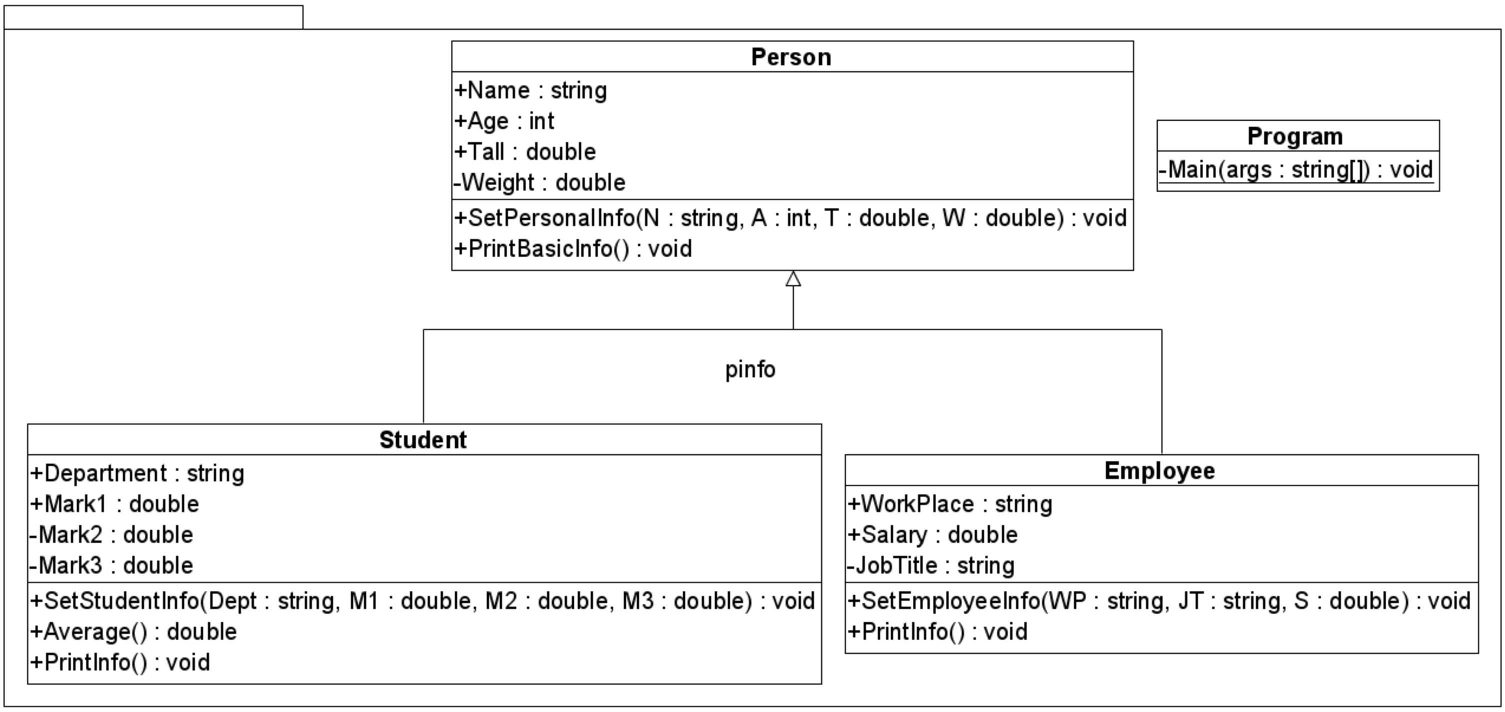
```
class A
{
    public void print()
    {
        Console.WriteLine("Base class method");
    }
}
class B:A
{
    public void print()
    {
        Console.WriteLine("Derived class method");
    }
}
class program
{
    static public void Main()
    {
        A a=new A();
        a.print();
        B b=new B();
        b.print();
    }
}
```

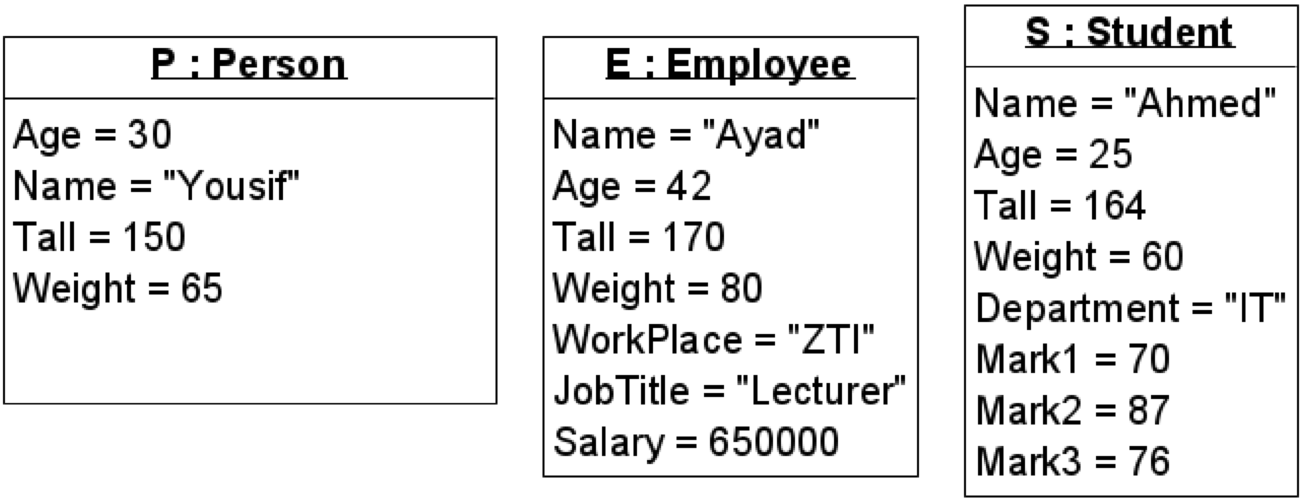


```
Base class method
Derived class method
```

When two methods share the same name, as in the previous example, one defined in the base class and the other in the derived class, the method in the derived class automatically

hides the one in the base class. This means that invoking the method using an object of the derived class will execute the derived class's version, even though both methods have the same name and signature.





Example: in the following example, there are three classes (***Person*** , ***Student*** , ***Employee***) both of ***Student*** and ***Employee*** are derived from ***Person***, so all public members of ***Person*** are inherited.

```
class Person
{
    public string Name;
    public int Age;
    public double Tall, Weight;
    public void SetPersonalInfo(string N, int A, double T, double W)
    {
        Name = N;
        Age = A;
        Tall = T;
        Weight = W;
    }
    public void PrintBasicInfo()
    {
        Console.WriteLine("Name: " + Name + "  Age: " + Age);
        Console.WriteLine("Tall: " + Tall + "  Weight:" + Weight);
    }
} //end of class Person
class Student : Person
{
    public string Department;
    public double Mark1, Mark2, Mark3;
    public void SetStudentInfo(string Dept, double M1, double M2, double M3)
    {
        Department = Dept;
        Mark1 = M1;
        Mark2 = M2;
        Mark3 = M3;
    }
```

This function for setting only information that related to student

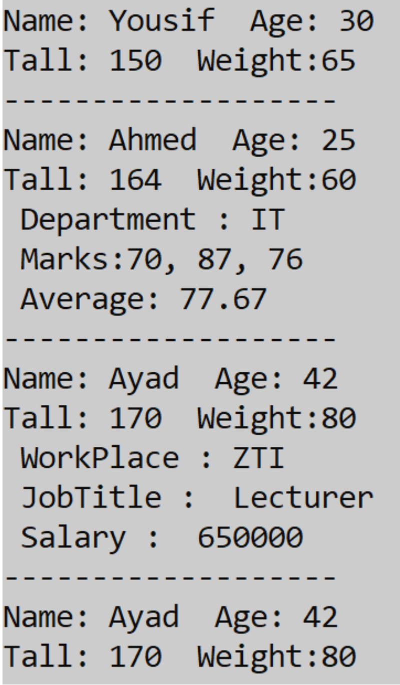
```
    public double Average()
    {
        double AVG = (Mark1 + Mark2 + Mark3) / 3.0;
        return AVG;
    }
    public void PrintInfo()
    {
        PrintBasicInfo();
        Console.WriteLine(" Department : " + Department);
        Console.WriteLine($" Marks:{Mark1}, {Mark2}, {Mark3}");
        Console.WriteLine($" Average: {Average():F2}");
    }
} //end of class Student
```

Here , it first calls PrintBasicInfo which is in base class then continue....

```
class Employee : Person
{
    public string WorkPlace, JobTitle;
    public double Salary;
    public void SetEmployeeInfo(string WP, string JT, double S)
    {
        WorkPlace = WP;
        JobTitle = JT;
        Salary = S;
    }
    public void PrintInfo()
    {
        PrintBasicInfo();
        Console.WriteLine(" WorkPlace : " + WorkPlace);
        Console.WriteLine(" JobTitle :  " + JobTitle);
        Console.WriteLine(" Salary :   " + Salary);
    }
} //end of class Employee

class Program
{
    static void Main(string[] args)
    {
        Person P = new Person();
        P.SetPersonalInfo("Yousif", 30, 150, 65);
        P.PrintBasicInfo();
        Console.WriteLine("-------------------");

        Student S = new Student();
        S.SetPersonalInfo("Ahmed", 25, 164, 60);
        S.SetStudentInfo("IT", 70, 87, 76);
        S.PrintInfo();
        Console.WriteLine("-------------------");

        Employee E = new Employee();
        E.SetPersonalInfo("Ayad", 42, 170, 80);
        E.SetEmployeeInfo("ZTI", "Lecturer", 650000);
        E.PrintInfo();
        Console.WriteLine("-------------------");
        E.PrintBasicInfo();
    }
}
```

Both <u>Student</u> and <u>Employee</u> use `SetPersonalInfo` for setting the general info.

Also can only print the basic information

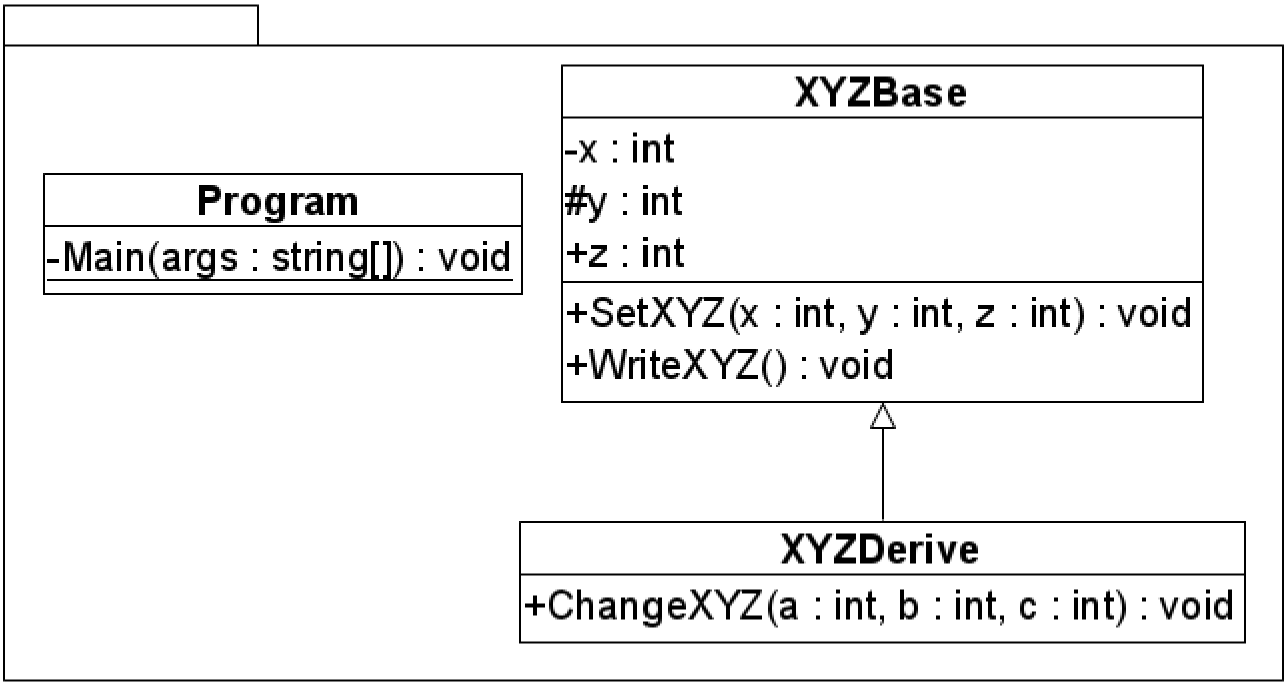


```
Name: Yousif  Age: 30
Tall: 150  Weight:65
-------------------
Name: Ahmed  Age: 25
Tall: 164  Weight:60
 Department : IT
 Marks:70, 87, 76
 Average: 77.67
-------------------
Name: Ayad  Age: 42
Tall: 170  Weight:80
 WorkPlace : ZTI
 JobTitle :  Lecturer
 Salary :  650000
-------------------
Name: Ayad  Age: 42
Tall: 170  Weight:80
```

Protected access modifier

The **protected** keyword is an access modifier (like private & public). A protected member is accessible within its class and by derived class.

- It works as **private** for outside of contain class.
- It works as **public** for derived classes.


XYZBase
-x : int
#y : int
+z : int
+SetXYZ(x : int, y : int, z : int) : void
+WriteXYZ() : void
Program
-Main(args : string[]) : void
XYZDerive
+ChangeXYZ(a : int, b : int, c : int) : void

Example:

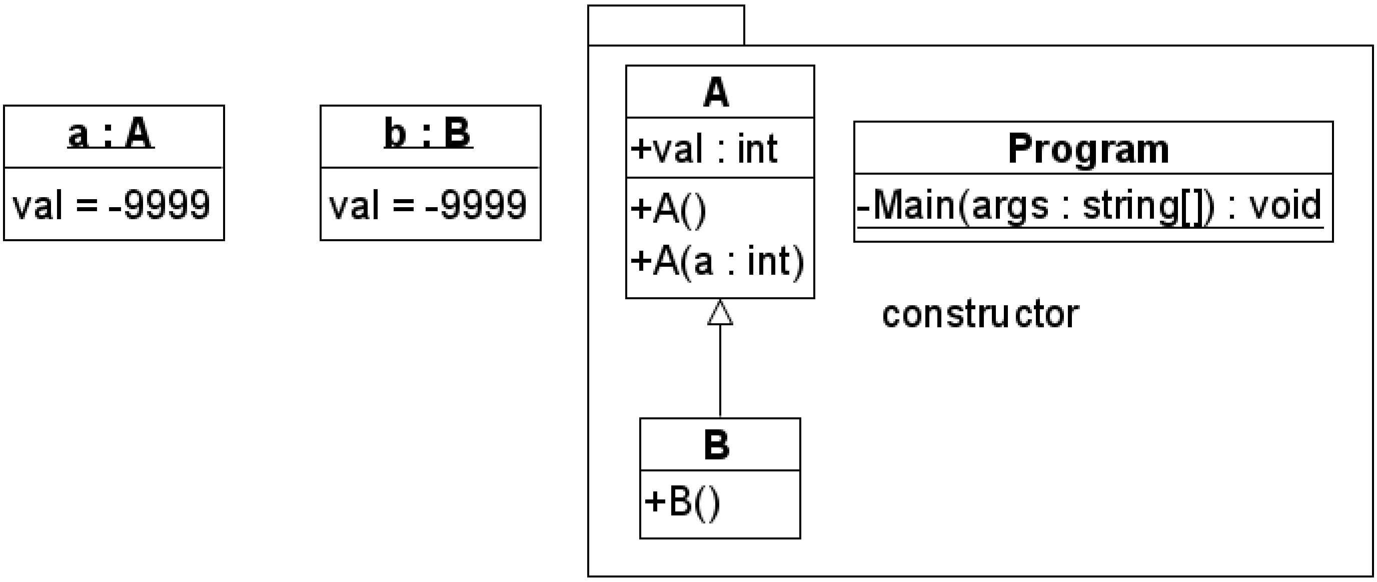
```
class XYZBase
{
    private int x;
    protected int y;
    public int z;
    public void SetXYZ(int x, int y, int z)
    {
        this.x = x; this.y = y; this.z = z;
    }
    public void WriteXYZ()
    {
        Console.WriteLine(" X: " + x + " ,  Y: " + y + " ,  Z: " + z);
    }
} //end of class XYZBase
```

```
class XYZDerive : XYZBase
{
    public void ChangeXYZ(int a, int b, int c)
    {
        x = a;   //error    because x is private
        y = b;   //allowed  because x is protected
        z = c;   //allowed  because x is public
    }

}//end of class XYZDerive
class Program
{
    static void Main(string[] args)
    {
        XYZBase B = new XYZBase();
        B.x = 10;   //error    because x is private
        B.y = 20;   // error   because x is protected
        B.z = 30;   //allowed  because x is public
        Console.ReadLine();
    }
}
```

The protected member can be accessed only from its class (base class) and ferived classes.

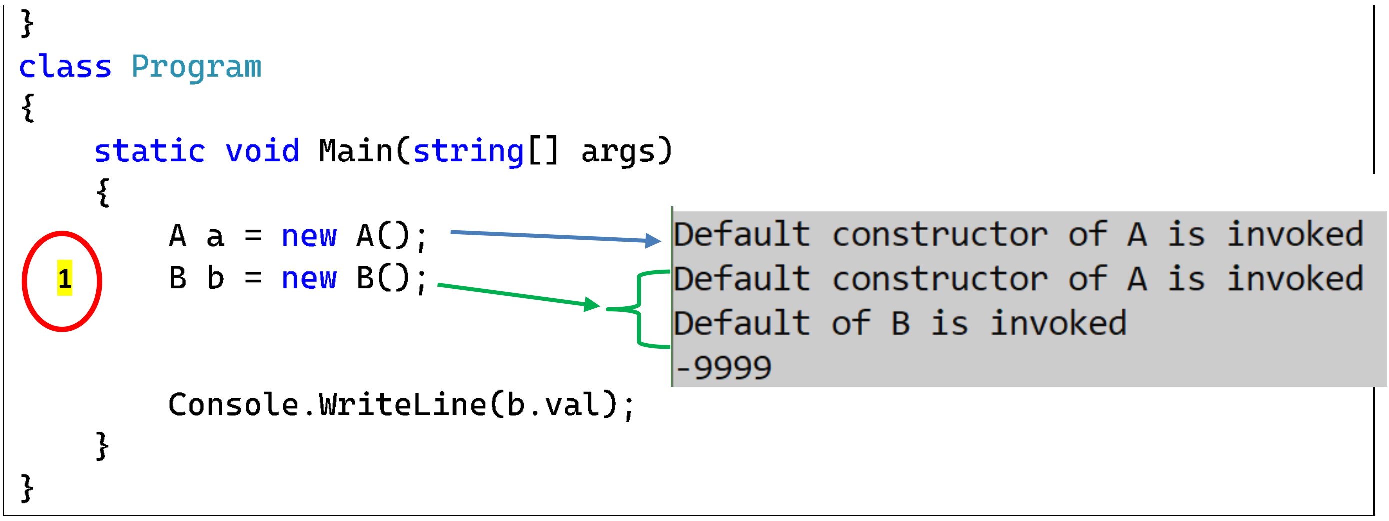
Base-class constructor

by default, when objects of derived class is created, it automatically call the **default construct of base class.**



Example:

```
class A
{
    public int val;
    public A()
    {
        Console.WriteLine("Default constructor of A is invoked");
        val = -9999;
    }
    public A(int a)
    {
        val = a;
    }
}
class B : A
{
    public B()
    {
        Console.WriteLine("Default of B is invoked");
    }
```



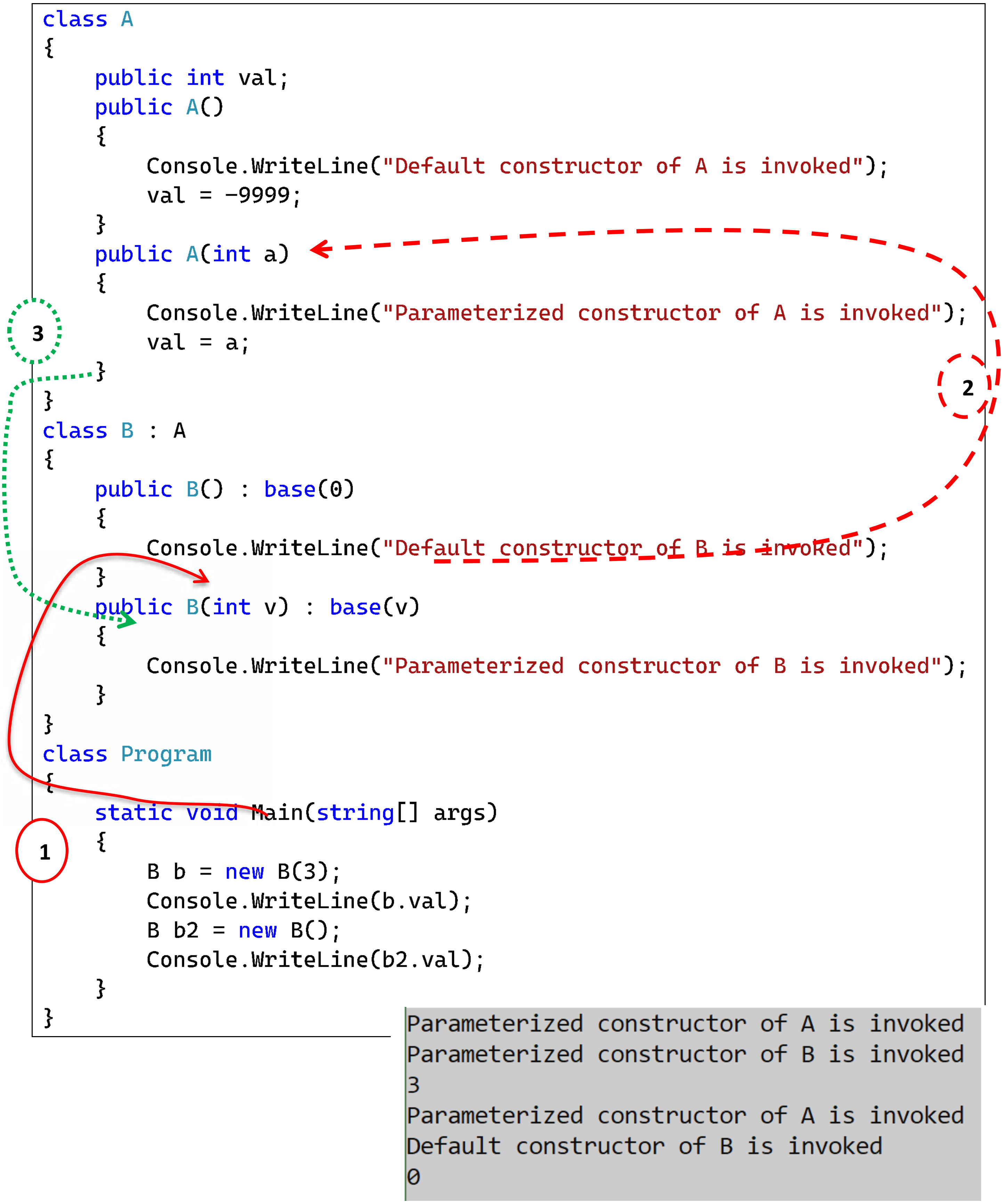
```
}
class Program
{
    static void Main(string[] args)
    {
        A a = new A();
        B b = new B();


        Console.WriteLine(b.val);
    }
}
```

```
Default constructor of A is invoked
Default constructor of A is invoked
Default of B is invoked
-9999
```

Note that (See the output), the constructor for the base class is called before the block for the derived constructor is executed.

To specify which base-class constructor should be called when creating instances of the derived class. A constructor can use the base keyword to call the constructor of a base class.

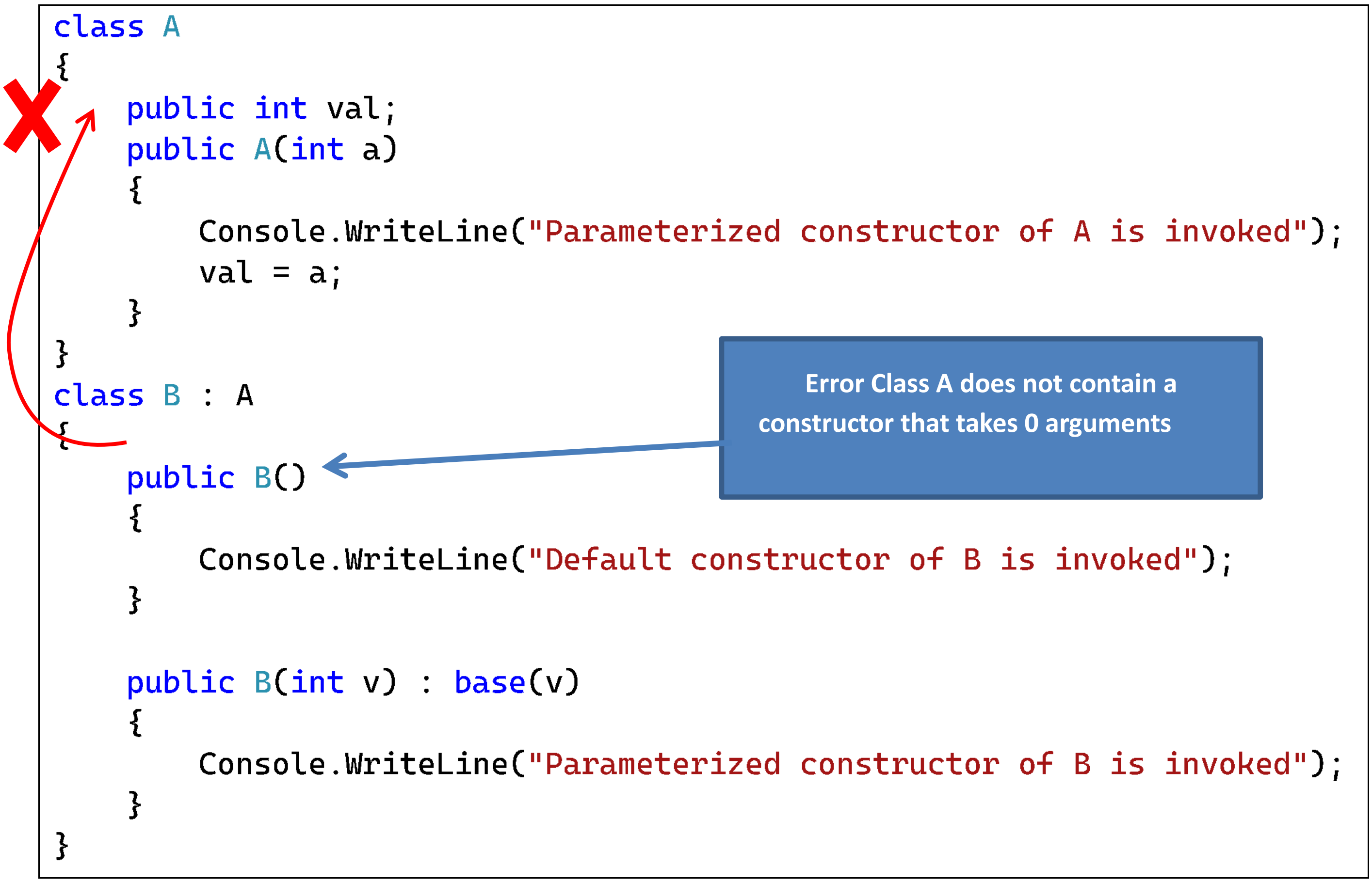
Example:



```
class A
{
    public int val;
    public A()
    {
        Console.WriteLine("Default constructor of A is invoked");
        val = -9999;
    }
    public A(int a)
    {
        Console.WriteLine("Parameterized constructor of A is invoked");
        val = a;
    }
}
class B : A
{
    public B() : base(0)
    {
        Console.WriteLine("Default constructor of B is invoked");
    }
    public B(int v) : base(v)
    {
        Console.WriteLine("Parameterized constructor of B is invoked");
    }
}
class Program
{
    static void Main(string[] args)
    {
        B b = new B(3);
        Console.WriteLine(b.val);
        B b2 = new B();
        Console.WriteLine(b2.val);
    }
}
```

```
Parameterized constructor of A is invoked
Parameterized constructor of B is invoked
3
Parameterized constructor of A is invoked
Default constructor of B is invoked
0
```

Note the following error

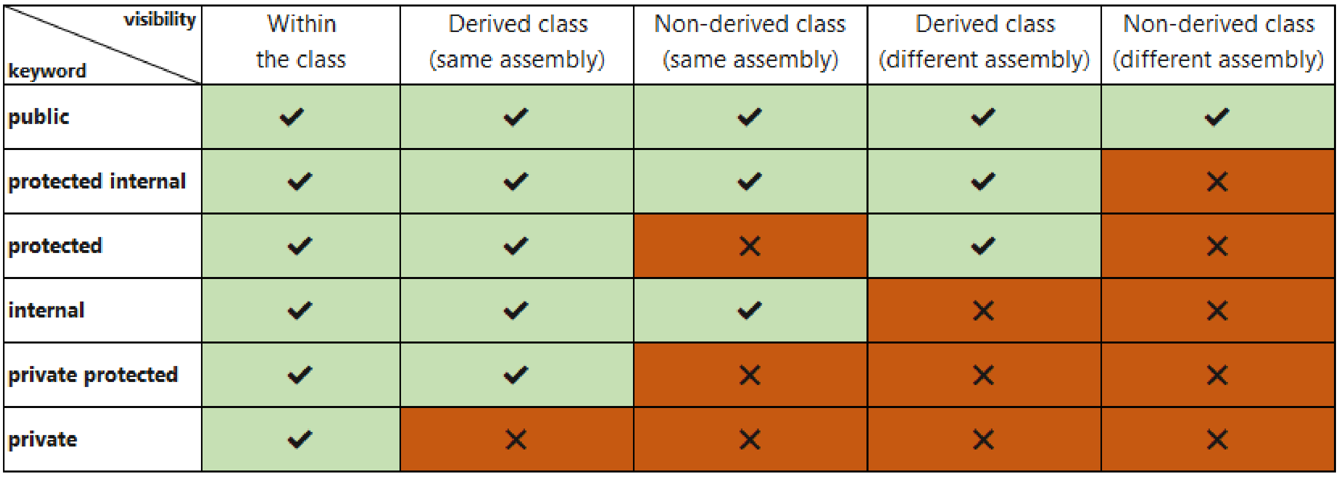


```
class A
{
    public int val;
    public A(int a)
    {
        Console.WriteLine("Parameterized constructor of A is invoked");
        val = a;
    }
}
class B : A
{
    public B()
    {
        Console.WriteLine("Default constructor of B is invoked");
    }

    public B(int v) : base(v)
    {
        Console.WriteLine("Parameterized constructor of B is invoked");
    }
}
```

Error Class A does not contain a constructor that takes 0 arguments

----

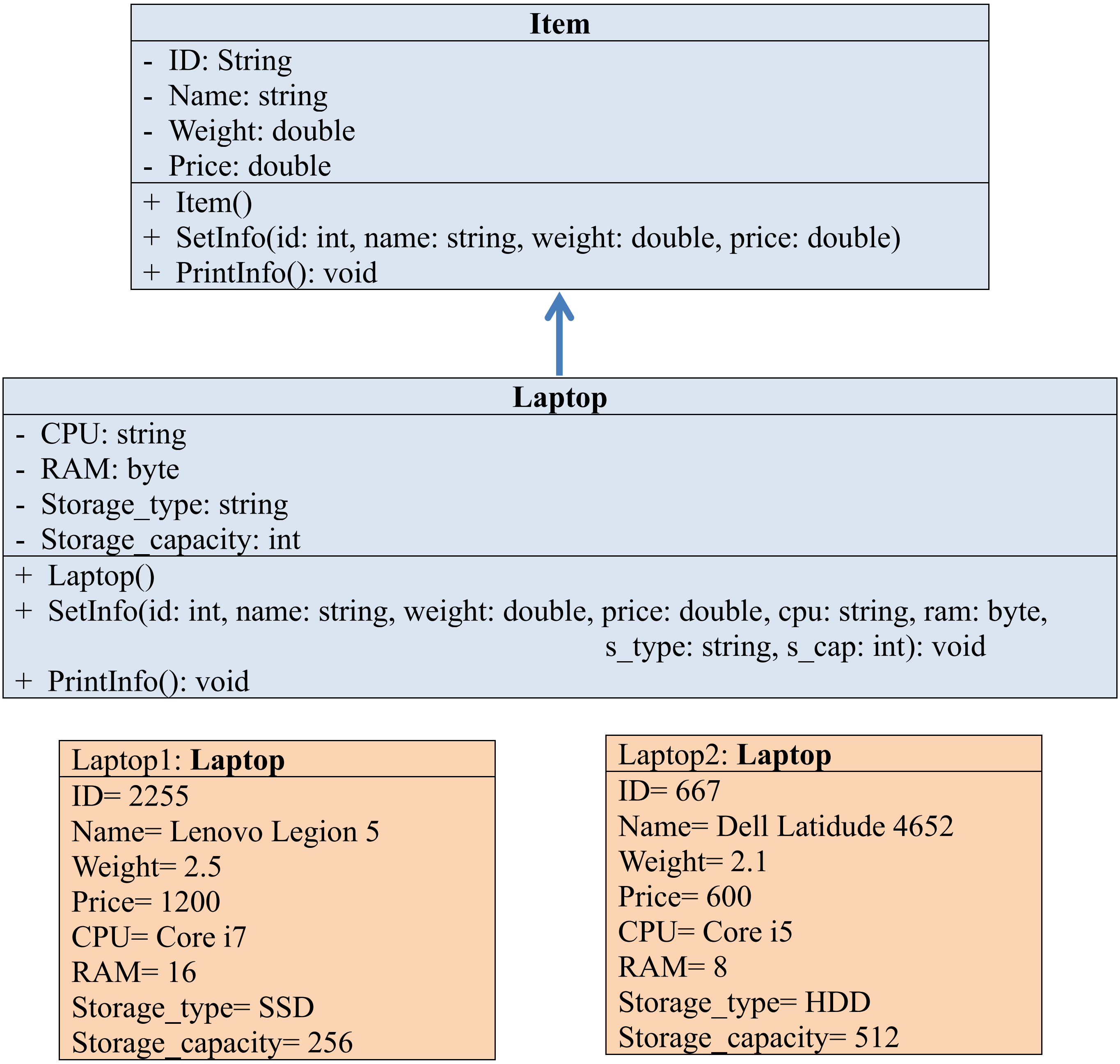
C# Access Modifiers Visibility



keyword \ visibility	Within the class	Derived class (same assembly)	Non-derived class (same assembly)	Derived class (different assembly)	Non-derived class (different assembly)
public	✓	✓	✓	✓	✓
protected internal	✓	✓	✓	✓	✗
protected	✓	✓	✗	✓	✗
internal	✓	✓	✓	✗	✗
private protected	✓	✓	✗	✗	✗
private	✓	✗	✗	✗	✗

Practical Examples

Ex-1: Convert the following UML Diagram to C# code



Item

- ID: String
- Name: string
- Weight: double
- Price: double

+ Item()
+ SetInfo(id: int, name: string, weight: double, price: double)
+ PrintInfo(): void

Laptop

- CPU: string
- RAM: byte
- Storage_type: string
- Storage_capacity: int

+ Laptop()
+ SetInfo(id: int, name: string, weight: double, price: double, cpu: string, ram: byte, s_type: string, s_cap: int): void
+ PrintInfo(): void

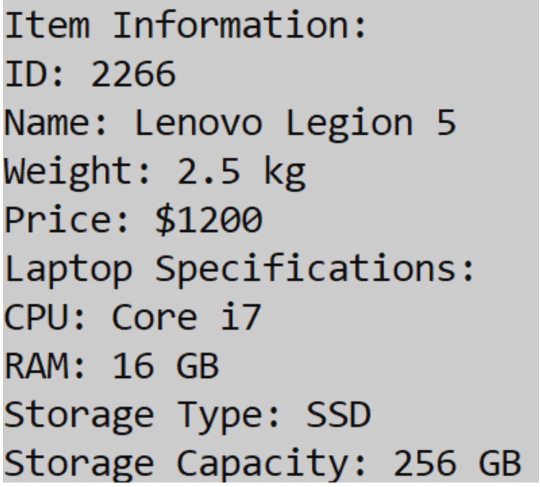
Laptop1: **Laptop**
ID= 2255
Name= Lenovo Legion 5
Weight= 2.5
Price= 1200
CPU= Core i7
RAM= 16
Storage_type= SSD
Storage_capacity= 256

Laptop2: **Laptop**
ID= 667
Name= Dell Latidude 4652
Weight= 2.1
Price= 600
CPU= Core i5
RAM= 8
Storage_type= HDD
Storage_capacity= 512

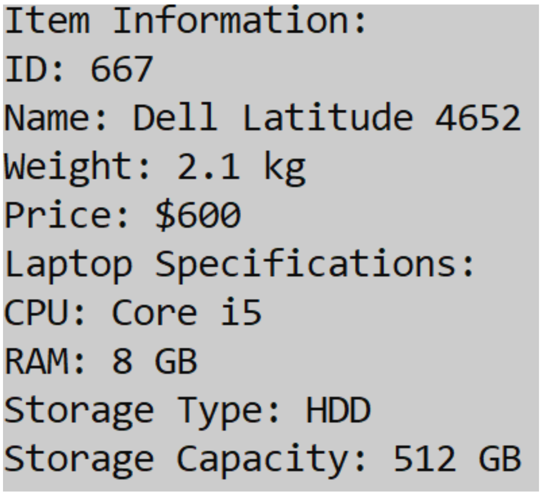
- Define a base class called **Item** with the following members:
    - Data members (Fields):
        - ID
        - Name
        - Weight
        - Price
    - Constructors:
        - Item(): default constructor, to set default value for each field
    - Member methods:
        - SetInfo: to set values to all fields in this class
        - PrintInfo: to display the information of all fields in this class

- Define a derived class called **Laptop** (inheriting from class **Item**) with the following members:
    - Data members:
        - CPU
        - RAM
        - Storage_Type
        - Storage_capacity
    - Constructors:
        - Laptop(): a default constructor to set a default value for each field
    - Member methods:
        - SetInfo: to send values to the base class and also set values to all fields of this class.
        - PrintInfo: to print all information including the field values in the base class and this class as well.
- In the Main() method:
    - Create an object of the class Laptop
    - Call the required method to set and display information.

```
class Item
{
    private int ID;
    private string Name;
    private double Weight;
    private double Price;
    public Item()
    {
        ID = 0;
        Name = "Unknown";
        Weight = 0.0;
        Price = 0.0;
    }
    public void SetInfo(int id, string name, double weight, double price)
    {
        ID = id;
        Name = name;
        Weight = weight;
        Price = price;
    }
    public void PrintInfo()
    {
        Console.WriteLine("Item Information:");
        Console.WriteLine($"ID: {ID}");
        Console.WriteLine($"Name: {Name}");
        Console.WriteLine($"Weight: {Weight} kg");
        Console.WriteLine($"Price: ${Price}");
    }
}


class Laptop : Item
{
    private string CPU;
    private int RAM;
    private string Storage_Type;
    private int Storage_Capacity;
    public Laptop()
    {
        CPU = "Unknown";
        RAM = 0;
        Storage_Type = "Unknown";
        Storage_Capacity = 0;
    }
    public void SetInfo(int id, string name, double weight, double price,
                        string cpu, int ram, string s_type, int s_capacity)
    {
        base.SetInfo(id, name, weight, price);     // Set base class fields

        CPU = cpu;
        RAM = ram;
        Storage_Type = s_type;
        Storage_Capacity = s_capacity;
    }
    public void PrintInfo()
    {
        base.PrintInfo();    // Print base class fields

        Console.WriteLine("Laptop Specifications:");
        Console.WriteLine($"CPU: {CPU}");
        Console.WriteLine($"RAM: {RAM} GB");
        Console.WriteLine($"Storage Type: {Storage_Type}");
        Console.WriteLine($"Storage Capacity: {Storage_Capacity} GB");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Laptop Laptop1 = new Laptop();
        Laptop1.SetInfo(2266, "Lenovo Legion 5", 2.5, 1200,"Core i7", 16, "SSD", 256);
        Laptop1.PrintInfo();

        Console.WriteLine("--------------------------------");
        Laptop Laptop2 = new Laptop();
        Laptop2.SetInfo(667, "Dell Latitude 4652", 2.1, 600,"Core i5", 8, "HDD", 512);
        Laptop2.PrintInfo();
    }
}
```

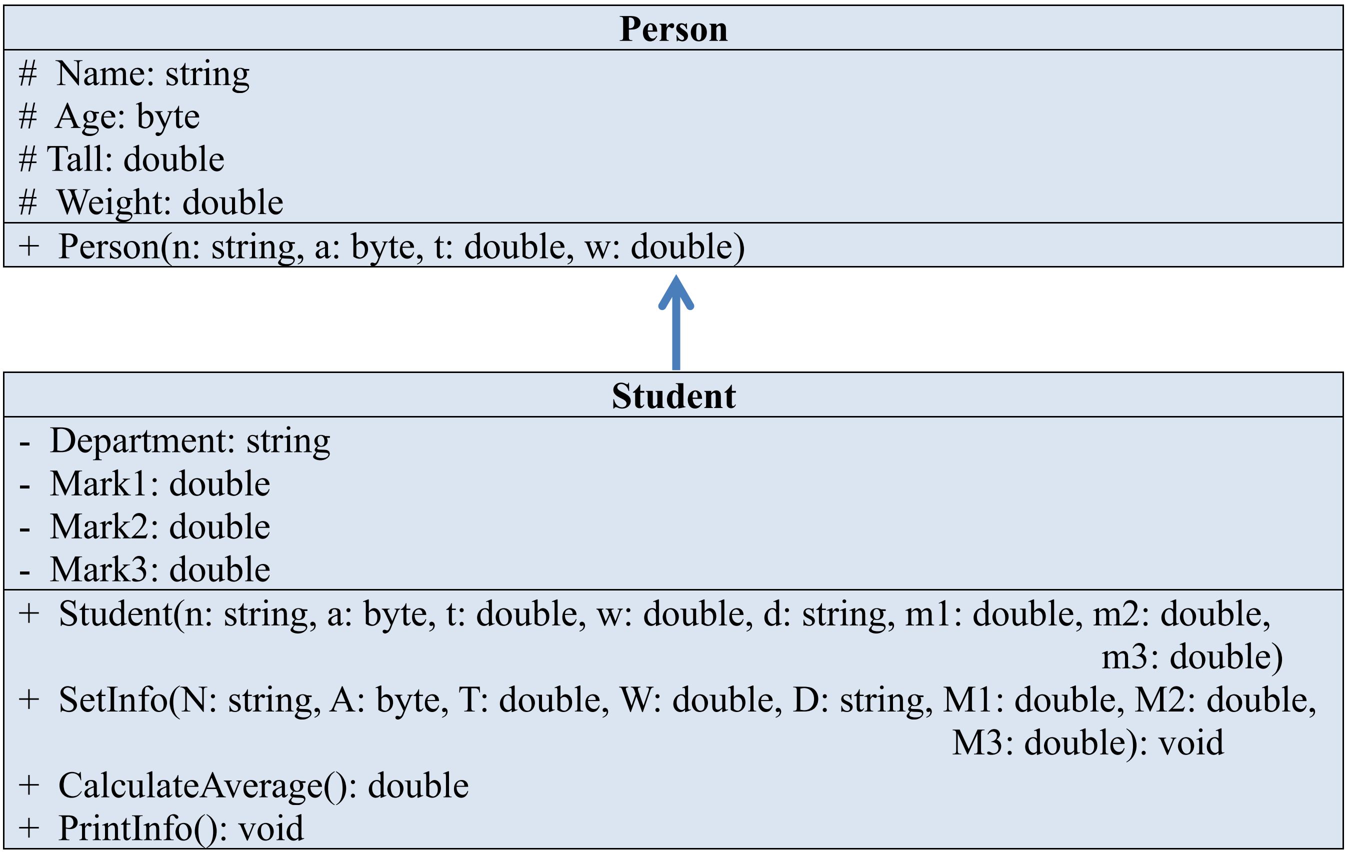


```
Item Information:
ID: 2266
Name: Lenovo Legion 5
Weight: 2.5 kg
Price: $1200
Laptop Specifications:
CPU: Core i7
RAM: 16 GB
Storage Type: SSD
Storage Capacity: 256 GB
```



```
Item Information:
ID: 667
Name: Dell Latitude 4652
Weight: 2.1 kg
Price: $600
Laptop Specifications:
CPU: Core i5
RAM: 8 GB
Storage Type: HDD
Storage Capacity: 512 GB
```

Ex-2: convert the following UML diagram to c# code



Person

# Name: string
# Age: byte
# Tall: double
# Weight: double

+ Person(n: string, a: byte, t: double, w: double)

Student

- Department: string
- Mark1: double
- Mark2: double
- Mark3: double

+ Student(n: string, a: byte, t: double, w: double, d: string, m1: double, m2: double, m3: double)
+ SetInfo(N: string, A: byte, T: double, W: double, D: string, M1: double, M2: double, M3: double): void
+ CalculateAverage(): double
+ PrintInfo(): void

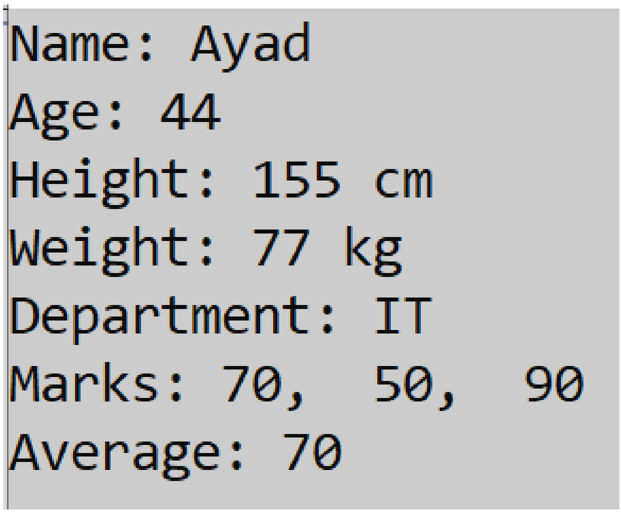
Stud: **Student**

Name= Ayad
Age=44
Tall=155
Weight=77
Department= IT
Mark1= 70
Mark2= 50
Mark3= 90

- Define a base class named **Person** with the following members:
  - Data members:
    - **Name**
    - **Age**
    - **Tall**
    - **Weight**
  - Constructors:
    - **Person**: parameterized constructor to set an initial value for each field.

- Define a derived class **Student** (inheriting from base class **Person**) with the following members:
    - Data members (attributes):
        - **Department**
        - **Mark1**, **Mark2**, **Mark3**
    - Constructors:
        - **Student**: parameterized constructor to give an initial value for each field.
    - Member methods:
        - **SetInfo**: to set a value for each field including the fields inherited from the base class.
        - **CalculateAverag**: to calculate and return the average of student marks.
        - **PrintInfo**: to display all student info including all field values and the average.
- In the **Main()** method:
    - Create an object of the class **Student**
    - Set all information by invoking **SetInfo(…..)** method
    - Display all info by calling **PrintInfo()** method

```
class Person
{
    protected string Name;
    protected byte Age;
    protected double Tall;
    protected double Weight;

    public Person(string n, byte a, double t, double w)
    {
        Name = n;
        Age = a;
        Tall = t;
        Weight = w;
    }
}

class Student : Person
{
    private string Department;
    private double Mark1, Mark2, Mark3;

    public Student(string n, byte a, double t, double w, string d, double m1, double m2,
double m3)  : base(n, a, t, w)
    {
        Department = d;
        Mark1 = m1;
        Mark2 = m2;
        Mark3 = m3;
    }
    public void SetInfo(string N, byte A, double T, double W, string D, double M1, double
M2, double M3)
    {
        // Set base class fields
        Name = N;
        Age = A;
        Tall = T;
        Weight = W;

        // Set derived class fields
        Department = D;
        Mark1 = M1;
        Mark2 = M2;
        Mark3 = M3;
    }

    public double CalculateAverage()
    {
        return (Mark1 + Mark2 + Mark3) / 3;
    }

    public void PrintInfo()
    {
        // Print base class information
        Console.WriteLine($"Name: {Name}");
        Console.WriteLine($"Age: {Age}");
        Console.WriteLine($"Height: {Tall} cm");
        Console.WriteLine($"Weight: {Weight} kg");

        // Print derived class information
        Console.WriteLine($"Department: {Department}");
        Console.WriteLine($"Marks: {Mark1},  {Mark2},  {Mark3}");
        Console.WriteLine($"Average: {CalculateAverage()}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Student stud = new Student("Ahmed", 0, 0, 0, "IT", 0, 0, 0);

        stud.SetInfo("Ayad", 44, 155, 77, "IT", 70, 50, 90);
        stud.PrintInfo();
    }
}
```



```
Name: Ayad
Age: 44
Height: 155 cm
Weight: 77 kg
Department: IT
Marks: 70,  50,  90
Average: 70
```