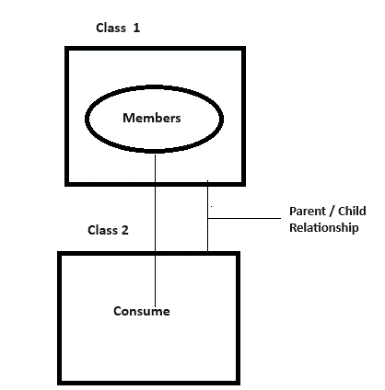
# Object-oriented programming (OOP) Lecture 3: Inheritance

**Inheritance**

* It a mechanism of consuming the members of one class from another class by establishing **parent/child** relationship between classes which provide reusability.



* In the above figure class 1 is called the **parent** class or **base** class or **super** class whereas class 2 is called **child** class or **derived** class or **sub** class

**Note**: child class can consume the members of its parent class as if it’s the owner of those members **except private** members which are accessed by the owner class only.

**Rules of Inheritance**

While working with inheritance we need to follow/consider a set of rules. `

**Rule 1**: Whenever you create an instance of the child class, then the constructor of this child class will implicitly call the constructor of the parent class. Hence, **the constructor of the parent class must be accessible from the child class**. See the code example below.

class Class1

{

Class1() // Constructor of Class1 declared as private by default

{

Console.WriteLine("Class1 constructor is called");

}

public void Test1()

{

Console.WriteLine("Test1() Method");

}

public void Test2()

{

Console.WriteLine("Test2() Method");

}

}

class Class2 : Class1

{

public void Test3()

{

Console.WriteLine("Test3() Method");

}

static void Main()

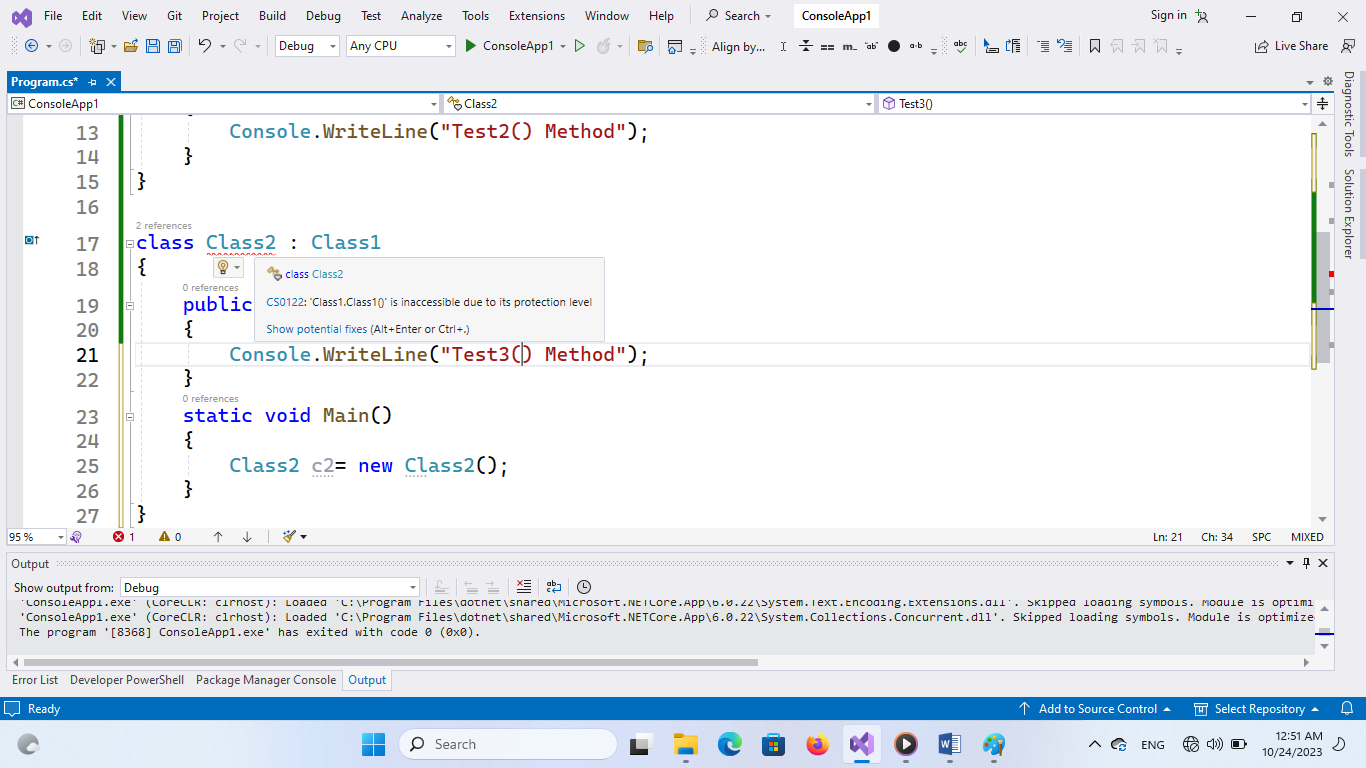
{

Class2 c2= new Class2();

}

}

We are going to get an error as shown in the below figure



Now if we create a constructor in Class2 and make the constructor of Class1 as public then the constructor of Class2 will implicitly call the constructor of Class1 first and then the constructor of the Class2 will be executed. Observe the code below

class Class1

{

public Class1() // Constructor of Class1

{

Console.WriteLine("Class1 constructor is called");

}

public void Test1()

{

Console.WriteLine("Test1() Method");

}

public void Test2()

{

Console.WriteLine("Test2() Method");

}

}

class Class2 : Class1

{

public Class2() // constructor of classs2 will first call Class1 constructor

{

Console.WriteLine("Constructor of Class2 is called")

}

public void Test3()

{

Console.WriteLine("Test3() Method");

}

static void Main()

{

Class2 c2= new Class2();

c2.Test1();

c2.Test2();

c2.Test3();

}

}

**Rule 2:** In inheritance, child class can access the members of parent class, but **parent class can not access** the members of child class that are purely defined in the child class.

**Rule 3:** we can initialize **parent class variable** by using the **child class instance** to make it as a reference. (a reference is a point to an instance which has no memory allocation) see the code example below

class Class1

{

public Class1() // Constructor of Class1

{

Console.WriteLine("Class1 constructor is called");

}

public void Test1()

{

Console.WriteLine("Test1() Method");

}

public void Test2()

{

Console.WriteLine("Test2() Method");

}

}

class Class2 : Class1

{

public Class2() // constructor of classs2

{

Console.WriteLine("Constructor of Class2 is called");

}

public void Test3()

{

Console.WriteLine("Test3() Method");

}

static void Main()

{

Class1 c1; // variable of Class1

Class2 c2= new Class2(); // instance of Class2

c1 = c2; // c1 is a reference of parent class created by using child class instance

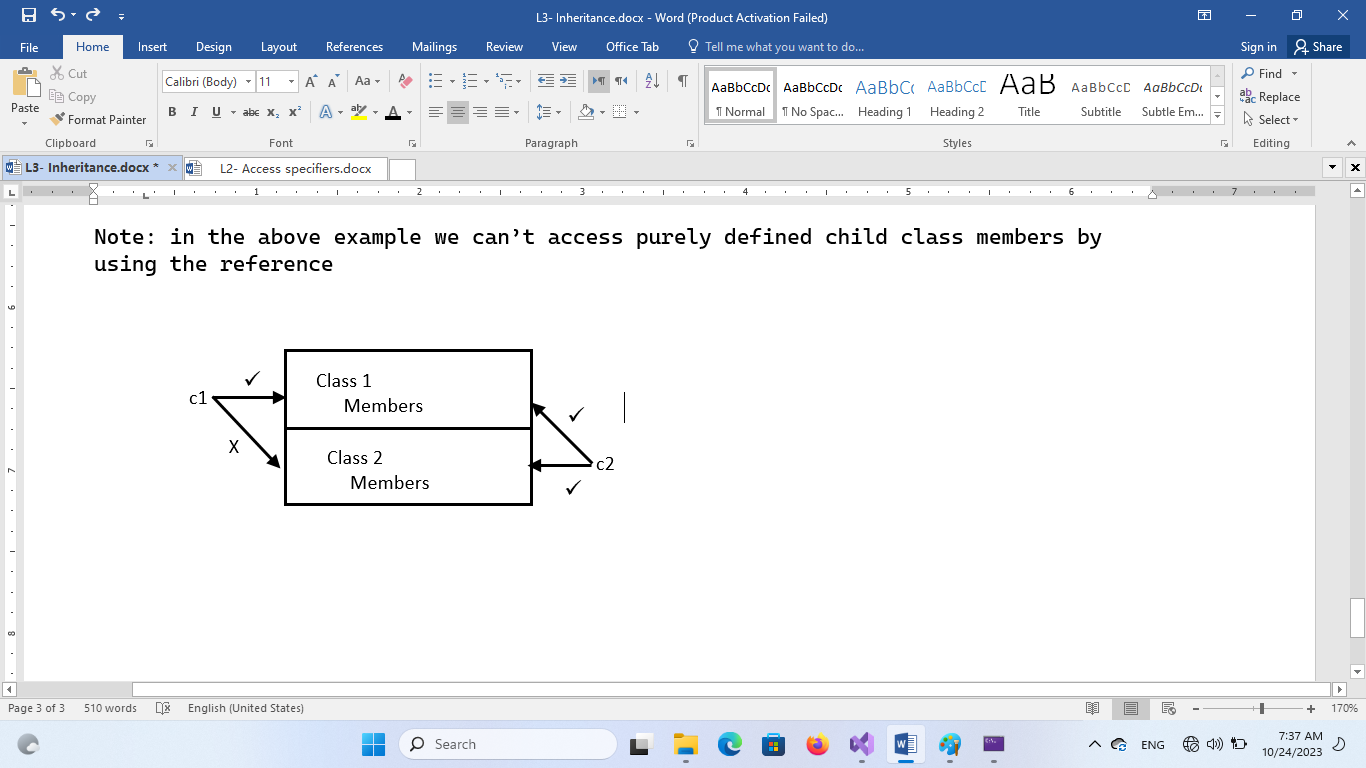
c1.Test1();

c1.Test2();

}

}

Note: in the above example **we can’t access purely defined child class members by using the reference**



**Rule 4:** Every class that is defined by us or pre-defined in the libraries of the language has a **default parent class which is Object class of system namespace**. Because of this any class can access the four important members of the Object class i.e (Equals(), GetHashCode(), GetType(), and ToString()).

## Types of Inheritance

* This talks about the **number of parent classes a class can have**. as per the standards of OOP we have two types of inheritance

1. single inheritance
2. multiple inheritance

* If a class has one immediate parent class to it, we call it as single inheritance. and if at all a class has more than one immediate parent class to it, we call it as multiple inheritance

|  |  |
| --- | --- |
| **Single Inheritance** | **Multiple Inheritance** |
| Class3  Class1  Class2 | Class3  Class2  Class1 |

**Rule 5:** In .NET and java languages **there is no support for multiple inheritance through classes**. What **they support is only single inheritance**, because multiple inheritance suffers from **ambiguity problem**.As we can see form the figure below if two parent classes (Class1 and Class2) has the same method (Test()) then the ambiguity problem will occur.

Test()

Test()

Class3

Class1

Class200000000000000000000000000000000000000000000000000000000

**Rule 6:** In the first rule of inheritance we have learned that whenever child class instance is created, child class constructor implicitly calls its parent class constructor for execution. But this implicit calling will be possible **only if the parent class constructor is parameterless**. whereas if it's parameterized constructor then it's the responsibility of the programmer to explicitly call the parent class constructor from the child class constructor by using "**base**" keyword and pass all the required values as parameters to the parent class constructor.

We can test this through the following the same example of rule 1 and make the following changes

public Class1(int i) // Constructor of Class1

{

Console.WriteLine("Class1 constructor is called"+i);

}

* Now if you run the example you will get an error stating that the parent class constructor requires a parameter value for execution.
* To overcome the above problem rewrite the constructor of child class i.e Calss2 as following

public Class2(int x):base(x) // constructor of classs2 will first call Class1 constructor with required parameters

{

Console.WriteLine("Constructor of Class2 is called")

}

Now rewrite the code in the main method as following

Class2 c2= new Class2(20);

In the above case the value that we are sending to Class2 constructor by creating the instance will first go to Class2 constructor and from their it goes to Class1 constructor for execution.

**How to implement inheritance in our applications?**

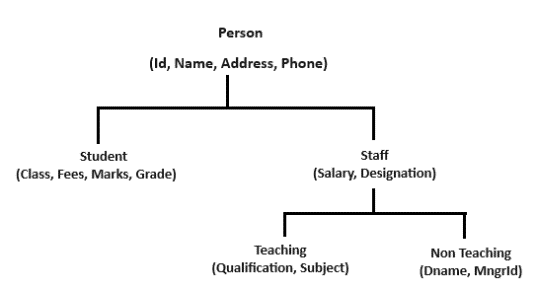
* As we discussed earlier when we study the meaning of OOP, we said that we should consider our application as a collection of objects these are also called **Entities.** So if we want to implement the concept of inheritance in our application we can follow the following steps

1. Identify the entities that are associated with the application. for example if the application is a school management system, then the entities will be as following

**Student**, **Teaching Staff**, **Non Teaching Staff**.

1. Now identify the **attributes** of each entity

|  |  |  |
| --- | --- | --- |
| **Student** | **Teaching Staff** | **Non Teaching Staff** |
| * Id * Name * Address * Phone * Class * Fees * Marks * Grade | * Id * Name * Address * Phone * Designation * Salary * Qualification * Subject | * Id * Name * Address * Phone * Designation * Salary * Dname * MngrId |

1. Now identify the **common attributes** between all entities and then put them in a hierarchical order as following
2. After that create the classes for

each entity as we learn that.