**Note:** Basics are written on hard copy. We are starting from Inheritance here.

# Inheritance

* + The process of creating a new class from an existing class such that the new class acquires all the properties and behaviours of the existing class is called inheritance.
  + The properties (or behaviours) are transferred from which class is called the superclass or parent class or base class whereas the class which derives the properties or behaviours from the superclass is known as a subclass or child class or derived class.
  + Inheritance is the concept that is used for code reusability.
  + Types of inheritance -

1. Single Inheritance: When a class is derived from a single base class then the inheritance is called single inheritance.
2. Multilevel Inheritance: When a derived class is created from another derived class, then that type of inheritance is called multilevel inheritance.
3. Hierarchical Inheritance: When more than one derived class is created from a single base class then it is called Hierarchical inheritance.
4. Hybrid Inheritance: Hybrid Inheritance is the inheritance that is the combination of any single, hierarchical, and multilevel inheritances.
5. 5) Multiple Inheritance: When a derived class is created from more than one base class then such type of inheritance is called multiple inheritances. But multiple inheritances are not supported by .net using classes and can be done using interfaces. Multiple inheritance introduces much more complexity into a class hierarchy.
   * Rules to be considered while working with inheritance in C# -

**Rule1:** In inheritance, the constructor of the parent class must be accessible to its child class otherwise the inheritance will not possible because when we create the child class object first it goes and calls the parent class constructor so that the parent class variable will be initialized and we can consume them under the child class.

**NOTE:** The reason why a child class internally calls its parent class constructor is to initialize parent class and can consume them under child class.

**Rule2:** In inheritance, the child classes can consume the parent class members but the parent class does not consume child class members that are purely defined in the child class.

* If you don’t want to give accessibility of the base class members to the non-derived class (in this case class Program) and would like to give derived class (Employee) then we need to use protected to the members.

# Abstraction

* Abstraction is a process of hiding the implementation and showing only the functionality to the user. Ex-we send SMS where we type msg and send the msg, we don't know the internal processing about the message delivery. There are 2 ways to achieve abstraction- abstract class(0-100%) and interface(100%).
* **Abstraction vs Encapsulation**

|  |  |
| --- | --- |
| **Abstraction** | **Encapsulation** |
| It is the process of gaining the information. | It is the process of contain the information. |
| It hides unwanted information. | It hides the data in a single unit along with a method to  Protect the info from outside. |
| We can implement abstraction using abstract classes or  Interfaces. | It can be implemented by access modifiers. |

* A class that is declared with abstract keyword is called abstract class. It is also called partially implemented class.
* An abstract class may or may not have abstract method but if a class contains abstract methods, then it must be declared

With abstract keyword.

* We cannot instantiate abstract class.
* Abstract method can be declare with abstract keyword and it does not contain definition.
* It can contain both static and instance variables.
* An abstract method or class cannot be declared as sealed.
* A class that provides implementation of abstract class or interface is called concrete class.
  + **When do you choose interface over an abstract class or vice versa?**

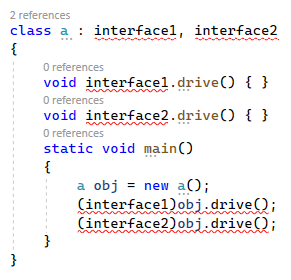
If we want some implementation that will be the same for all the derived classes, then it is better to go for an abstract class instead of an interface. With the interface, we can move our implementation to any class that implements the interface. With the abstract class, we can share the implementation for all the derived classes in one central place, and thus avoid code duplication in the derived classes.

* **If a class inherits an interface, what are the 2 options available for that class?**

**Option 1:** Provide Implementation for all the members inherited from the interface.

**Option 2:** If the class does not wish to provide Implementation for all the members inherited from the interface, then the class has to be marked as abstract.

* **A class inherits from 2 interfaces and both the interfaces have the same method name as shown below. How should the class implement the drive method for both Car and Bus interfaces?**



* **Why should the method have an abstract keyword if it does not have a body in C#?**

In a class, we are allowed only to define a class with the body. Since we are changing its default behavior (which means removing its body) it must have the abstract keyword in its prototype.

* **When to use the abstract method in C#?**

Abstract methods are usually declared where two or more subclasses are expected to fulfil a similar role in a different manner.

* **Why abstract class cannot be instantiated?**

If compiler allows us to create the object for an abstract class, we can invoke the abstract method using that object which cannot be executed by CLR at runtime.

* **Can we declare an abstract method as sealed?**

No, because it should be allowed to override in subclasses.

* **Can we declare abstract method as private?**

No, because it should be inherited in subclasses.

* **What type of member can we define in an abstract class?**

We can define all static and non-static members including properties, fields, indexes, and also abstract methods.

* **Will abstract class members be created when a subclass object is created?**

Yes**,** its non-static members get memory when its concrete sub-class object is created.

* **How can we execute static and non-static concrete members of the abstract class?**

Static members can be executed directly from its main method and its non-static members are executed by using its concrete sub-class object.

* **Can we declare an abstract method as static?**

No, we are not allowed to declare an abstract method as static.

* **Overriding method vs Abstract method?**

In Method Overriding, the child class re-implementing the method is optional but in Abstract method, the child class implementing the method is mandatory.

* Interface
* Interface is a blueprint of a class. It is a mechanism to achieve abstraction and multiple inheritance and loose coupling.
* What is the need for an interface when we have the abstract class to define abstract methods?

.NET doesn’t support multiple inheritances with classes. We have to use interface to develop abstraction for supporting multiple inheritances. If we define an abstract class in place of an interface, a service provider cannot implement multiple specifications so that the service provider cannot have multiple businesses.

* + By using the keyword interface, we can declare an interface.
  + By default, the members of an interface are public and abstract. They don’t allow explicit access modifiers. An interface can contain

1. Abstract methods
2. Properties
3. Indexes
4. Events
5. Delegates

but not contains - non-abstract functions, constructors, data fields and destructors.

* + Interface contains private non-abstract methods.
  + Interface cannot implement an abstract class. But Abstract class implements interface.
  + A interface cannot be declared with sealed. It will cause compilation error.
  + Is more than one interface are allowed to implement a class?

Yes, a class can implement multiple interfaces; this is an effective way to achieve multiple inheritances in .NET. But a class can extend only one superclass.

* + Is it necessary to implement all interface methods?

It is not necessary for a class that implements an interface to implement all its methods, but in this case, the class must be declared as abstract.

* + How interface is different from a class in C#?

1. We cannot instantiate an interface.
2. An interface does not contain any constructor or data fields or destructor, etc.
3. All of the methods of an interface are abstract and public by default.
4. An interface is not extended by a class; it is implemented by a class.
5. An interface can extend multiple interfaces.
   * What are the similarities between the interface and abstract class in C#?
6. Both interface and the abstract class cannot be instantiated means we cannot create the object.
7. But we can create a reference variable for both interface and abstract class.
8. The subclass should implement all abstract methods.
9. Both cannot be declared as sealed.

##### What are the differences between the interface and abstract class in C#?

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| It is a partially implemented class. It allows us to define both concrete and abstract methods. | It is a fully un-implemented class. It allows us to define only abstract methods. |
| It should be declared as abstract by using the abstract keyword, abstract methods should also contain the abstract keyword. | It should be created by using the keyword interface. Declaring its methods as abstract is optional because by default the methods of an interface are abstract. The compiler places abstract keywords at the time of program compilation. |
| A class that contains one or more abstract functions is called abstract class. | The class which contains all the abstract functions is known as an interface. |
| Its member’s default accessibility modifier is private and can be changed to any of the other accessibility modifiers. | Its member’s default accessibility modifier is public and cannot be changed. |
| It is possible to declare data fields in an abstract class. | But it is not possible to declare any data fields in an interface. |
| An abstract class can contain the non-abstract function. | An interface cannot contain non-abstract functions. |
| An abstract class can inherit from another abstract class or from an interface. | An interface can inherit from only other interfaces but cannot inherits from the abstract class. |
| It can have inner classes | It can also have inner classes. |
| An abstract class cannot be used to implement multiple inheritances. | An interface can be used to implement multiple inheritances. |
| Abstract class members can have access modifiers. | Interface members cannot have access modifiers. |

* Polymorphism
* It means many forms.
* When a function shows different behaviours when we passed different type of input, then it is called polymorphism. Ex- Vehicle has various forms like 2 wheelers, 4 wheelers etc.
* Polymorphism is of 2 types –

1. Static Polymorphism / Compile-Time Polymorphism / Early Binding
2. Dynamic Polymorphism / Run-Time Polymorphism / Late Binding

* Polymorphism can be implemented by –

1. Function Overloading
2. Function Overriding
3. Function Hiding

* **Compile Time Polymorphism**

1. If the function is going to be executed from the same bounded class at runtime, then it is called Compile time polymorphism.
2. This happens in the case of Method overloading.
3. Method Overloading – Multiple methods with same name but different parameters.
4. It is not possible to overload a method based on return type and params modifier.

* **When should we overload methods?**

If we want to execute the same logic with different types and number of arguments.

* **What are the advantages of using Method Overloading?**

If we overload the methods, then the user of our application gets comfort feeling in using the method with an impression that he/she calling one method by passing different types of values. The best example for us is the system-defined “WriteLine()” method.

* **Can we overload method in the same class?**

Yes, we can. Method can be overloaded in the same, sub or super class because they are different methods.

But we can’t override a method In the same class because they are same methods with different implementation.

* **What is Inheritance-Based Overloading in C#?**

A method that is defined in the parent class can also be overloaded under its child class. It is called Inheritance-Based Overloading in C#.

* **Runtime Polymorphism**

1. If the function is going to be executed from the different class at runtime rather than the class bounded at compilation- time, then it is called runtime polymorphism.
2. This happens in the case of Method Overriding.