**========Interface Points**

The point of interfaces is not to help you remember what method to implement, it is here to define a contract. In foreach , IEnumerable defines a contract between foreach and any enumerable thing. It says : "Whoever you are, as long as you stick to the contract (implement IEnumerable), I promise you I'll iterate over all your elements". And, that is great (for a non dynamic language).

**========When to choose abstract class and interface in c#**

Interface:

–> If your child classes should all implement a certain group of methods/functionalities but each of the child classes is free to provide its own implementation then use interfaces.

–> If you want your child classes to implement multiple unrelated functionalities in short multiple inheritance use interfaces

Abstract Classes

–> When you have a requirement where your base class should provide default implementation of certain methods whereas other methods should be open to being overridden by child classes use abstract classes.

–> The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share.

<https://stackoverflow.com/questions/10914802/why-i-should-go-for-interfaces-in-c-sharp-when-i-can-implement-the-methods-direc>

<http://www.c-sharpcorner.com/article/when-to-use-abstract-class-and-interface-in-real-time-projects/>

If you have some kind of default functionality to share across classes in the hierarchy, you can use an abstract class. But if you don't have any default implementation to share and just need to define contracts for derived classes to follow; interface is the most preferred choice. It is a standard rule when using an interface, be sure you have done it right the first time. Once the interface is implemented by derived classes, it is difficult to update or modify the interface since everyone else's code breaks

OOPS Concept Where Used Abstract Class Ex. (Discuss With Pinu)

3 Types of User - One Common functionality for all user implemented in abstract class all other methods are override

Admin User, Client User, Employee User

Functions -Login(), GetUser(), Permissions()

**===**[**Why can an abstract class have constructor?**](https://stackoverflow.com/questions/19944644/why-can-an-abstract-class-have-constructor)

<https://stackoverflow.com/questions/19944644/why-can-an-abstract-class-have-constructor>

It's there to enforce some initialization logic required by all implementations of your abstract class, or any methods you have implemented on your abstract class (not all the methods on your abstract class have to be abstract, some can be implemented).

Any class which inherits from your abstract base class will be obliged to call the base constructor.

You usually want abstract classes when you need to have different strategies for some particular cases, so it makes sense to be able to do everything else in the abstract class. And it's a good practice to make the constructor protected.

Because there might be a standard way you want to instantiate data in the abstract class. That way you can have classes that inherit from that class call the base constructor.

Abstract class can not be instantiated directly, but whenever a class derived from the abstract is instantiated the Abstract class constructor is invoked, this can be used for various initialization purposes like to initialize properties methods and all.