Delegates, Lambdas, Events

Vladyslav Khrapov

Delegates

Delegate – is a type that represents references to methods with a particular parameter list and return type.

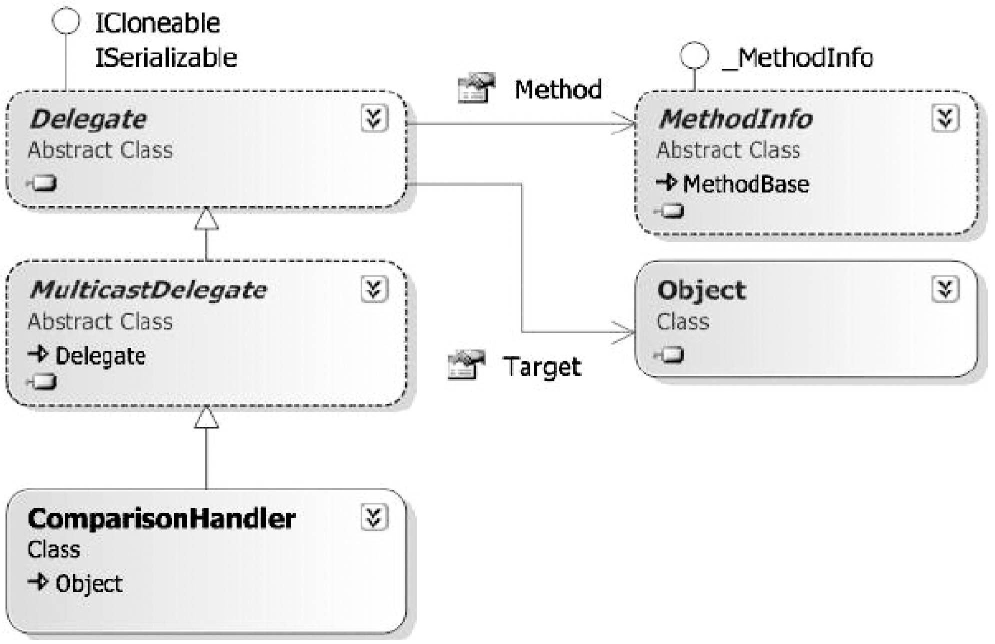
When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance.

Delegates are used to pass methods as arguments to other methods

Microsoft:
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/

Delegates

All delegates inherit base abstract class **MulticastDelegate** which inherit base abstract class Delegate.



Delegates signature

Any **delegate can point** to any **static** or **non static method** if the method has the **same signature** and **return type**.

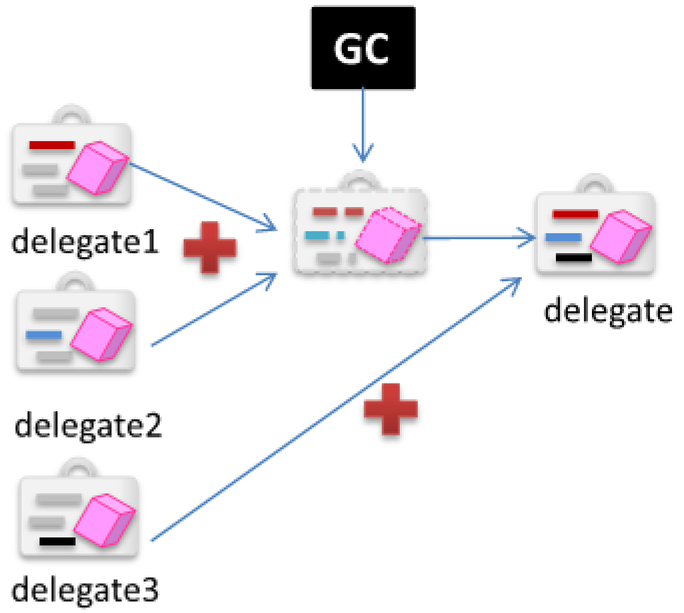
```
public delegate void MyDelegate();

class Program
{
	static void Main()
	{
		MyDelegate del = new MyDelegate([pass class method here]);
		del.Invoke();
		del();
	}
}
```

Combined delegates

A useful property of delegate objects is that multiple objects can be assigned to one delegate instance by using the **+** operator.

The multicast delegate contains a list of the assigned delegates. When the multicast delegate is called, it invokes the delegates in the list, in order. Only delegates of the same type can be combined.

The **-** operator can be used to remove a component delegate from a multicast delegate.



Anonymous delegates (anonymous functions)

In a situation where creating a new method is unwanted overhead, C# enables you to **instantiate a delegate and immediately specify a code block** that the delegate will process when it is called.

The block can contain either a **lambda expression** or an **anonymous method**.

In versions of **C# before 2.0**, **the only way** to declare a delegate was to use **named methods**. **C# 2.0 introduced anonymous methods** and in **C# 3.0 and later**, **lambda expressions** supersede anonymous methods as the **preferred way** to write inline code.

Delegates usage

This ability to refer to a method as a parameter makes **delegates ideal for defining callback methods**.

For example, **a reference to a method** that compares two objects **could be passed as an argument** to a sort algorithm.

Because the comparison code is in a separate procedure, the sort algorithm **code can be written in a more general way**.

Delegates usage

- LINQ (expressions)
- Win Forms (event handlers)
- ASP.NET (event handlers)
- ASP.NET Core (OWIN/Katana and Middlewares)

Delegates built into .NET

- **Action** - Encapsulates a method that has no parameters and does not return a value.

(**public delegate void Action();**)

- **Func** - Encapsulates a method that has one parameter and returns a value of the type specified by the **TResult** parameter.

(**public delegate TResult Func<in T, out TResult>(T arg);**)

- **Comparison<T>** - Represents the method that compares two objects of the same type.

(**public delegate int Comparison<in T>(T x, T y);**)

Events

Events - are, like delegates, a *late binding* mechanism. In fact, events are built on the language support for delegates.

Events are a way for an object to broadcast (to all interested components in the system) that something has happened. Any other component can subscribe to the event, and be notified when an event is raised.