Programming Language - C#

Events & Delegates

Delegates

- The delegate is a reference type data type that defines the method signature.

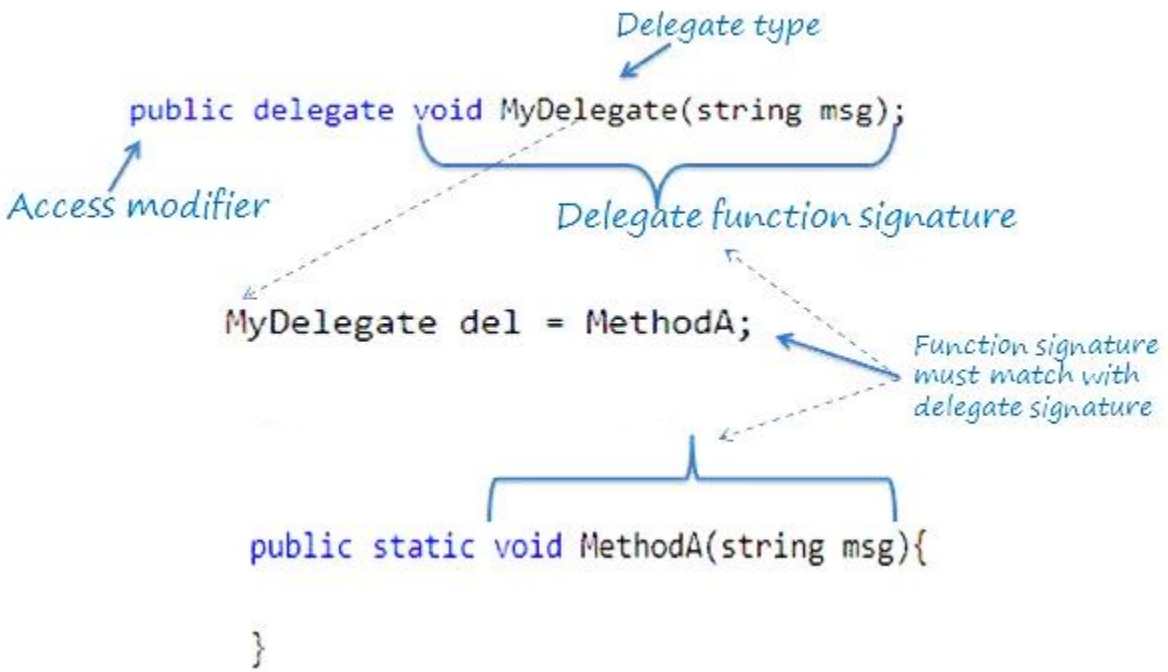
- You can define variables of delegate, just like other data type, that can refer to any method with the same signature as the delegate.

- There are three steps involved while working with delegates:
  - Declare a delegate
  - Set a target method
  - Invoke a delegate

Delegates

Delegate Syntax

```
[access modifier] delegate [return type] [delegate name]([parameters])
```



Multicast Delegate

- The delegate can point to multiple methods.
- A delegate that points multiple methods is called a multicast delegate.
- The "+" or "+=" operator adds a function to the invocation list, and the "-" and "-=" operator removes it.

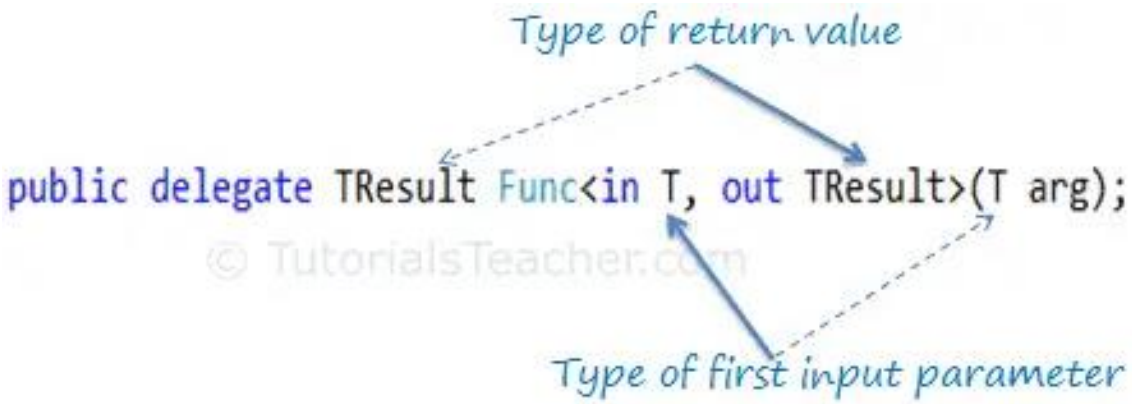
Points to Remember

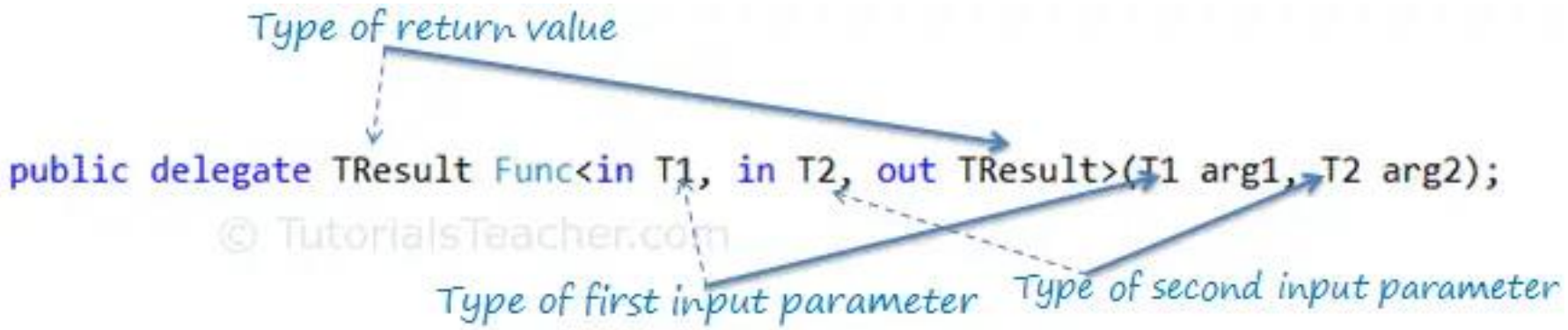
- Delegate is the reference type data type that defines the signature.
- Delegate type variable can refer to any method with the same signature as the delegate.
- Syntax: [access modifier] delegate [return type] [delegate name]([parameters])
- A target method's signature must match with delegate signature.
- Delegates can be invoked like a normal function or Invoke() method.
- Multiple methods can be assigned to the delegate using "+" or "+=" operator and removed using "-" or "-=" operator. It is called multicast delegate.
- If a multicast delegate returns a value, then it returns the value from the last assigned target method.
- Delegate is used to declare an event and anonymous methods in C#.

Func Delegate

- C# includes built-in generic delegate types Func and Action, so that you don't need to define custom delegates manually in most cases.
- Func is a generic delegate included in the System namespace.
- It has zero or more input parameters and one out parameter.
- The last parameter is considered as an out parameter.

Func Delegate





Points to Remember

- Func is built-in delegate type.
- Func delegate type must return a value.
- Func delegate type can have zero to 16 input parameters.
- Func delegate does not allow ref and out parameters.
- Func delegate type can be used with an anonymous method or lambda expression.

Action Delegate

- Action is a delegate type defined in the System namespace.
- An Action type delegate is the same as Func delegate except that the Action delegate doesn't return a value.
- In other words, an Action delegate can be used with a method that has a void return type.

Points to Remember

- Action delegate is same as func delegate except that it does not return anything.
- Return type must be void.
- Action delegate can have 0 to 16 input parameters.
- Action delegate can be used with anonymous methods or lambda expressions.

Advantages of Action and Func Delegates

- Easy and quick to define delegates.
- Makes code short.
- Compatible type throughout the application.

Predicate Delegate

- Predicate is the delegate like Func and Action delegates.
- It represents a method containing a set of criteria and checks whether the passed parameter meets those criteria.
- A predicate delegate methods must take one input parameter and return a boolean - true or false.
- Points to Remember:
  - Predicate delegate takes one input parameter and boolean return type.
  - Anonymous method and Lambda expression can be assigned to the predicate delegate.

Anonymous Method

- As the name suggests, an anonymous method is a method without a name.
- Anonymous methods in C# can be defined using the delegate keyword and can be assigned to a variable of delegate type.
- Anonymous Method Limitations
    - It cannot contain jump statement like goto, break or continue.
    - It cannot access ref or out parameter of an outer method.
    - It cannot have or access unsafe code.
    - It cannot be used on the left side of the is operator.

Points to Remember

- Anonymous method can be defined using the delegate keyword
- Anonymous method must be assigned to a delegate.
- Anonymous method can access outer variables or functions.
- Anonymous method can be passed as a parameter.
- Anonymous method can be used as event handlers.

Events

- An event is a notification sent by an object to signal the occurrence of an action.
- Events in .NET follow the observer design pattern.
- The class who raises events is called Publisher, and the class who receives the notification is called Subscriber.
- There can be multiple subscribers of a single event.

Points to Remember

- An event is a wrapper around a delegate. It depends on the delegate.
- Use "event" keyword with delegate type variable to declare an event.
- Use built-in delegate EventHandler or EventHandler<TEventArgs> for common events.
- The publisher class raises an event, and the subscriber class registers for an event and provides the event-handler method.
- Name the method which raises an event prefixed with "On" with the event name.
- The signature of the handler method must match the delegate signature.

Points to Remember

- Register with an event using the += operator. Unsubscribe it using the -= operator. Cannot use the = operator.
- Pass event data using EventHandler<TEventArgs>.
- Derive EventArgs base class to create custom event data class.
- Events can be declared static, virtual, sealed, and abstract.
- An Interface can include the event as a member.
- Event handlers are invoked synchronously if there are multiple subscribers.

Covariance and Contravariance

- Covariance and contravariance allow us to be flexible when dealing with class hierarchy.
- Covariance enables you to pass a derived type where a base type is expected.
- Co-variance is like variance of the same kind.
- The base class and other derived classes are the same kind of class that adds extra functionalities to the base type.
- So, covariance allows you to use a derived class where a base class is expected.

Contravariance

- Contravariance is applied to parameters.
- Contravariance allows a method with the parameter of a base class to be assigned to a delegate that expects the parameter of a derived class.

Thank you

Innovative Services

Passionate Employees

Delighted Customers