Events

Events are based on delegates and offer a publish/subscribe mechanism to delegates. You can find events everywhere across the framework.
In Windows applications, the `Button` class offers the `Click` event. This type of event is a delegate.

In the code example shown in this section, events are used to connect the CarDealer and Consumer classes. The CarDealer class offers an event when a new car arrives. The Consumer class subscribes to the event to be informed when a new car arrives.

Event Publisher

You start with a CarDealer class that offers a subscription based on events. CarDealer defines the event named NewCarInfo of type EventHandler<CarInfoEventArgs> with the event keyword.

```
using static System.Console;
using System;

namespace ProCSharp.Delegates
{
  public class CarInfoEventArgs: EventArgs
  {
    public CarInfoEventArgs(string car)
    {
       Car = car;
    }

    public string Car { get; }
  }

  public class CarDealer
  {
    public event EventHandler<CarInfoEventArgs> NewCarInfo;

    public void NewCar(string car)
    {
      WriteLine($"CarDealer, new car {car}");

      NewCarInfo?.Invoke(this, new CarInfoEventArgs(car));
    }
  }
}
```

The class CarDealer offers the event NewCarInfo of type eventHandler<CarInfoEventArgs>.

As a convention, events typically use methods with two parameters; the first parameter is an object and contains the sender of the event, and the second parameter provides information about the event.

With EventHandler<TEventArgs>, the first parameter needs to be of type object, and the second parameter is of type T. EventHandler<TEventArgs> also defines a constraint on T; it must derive from the base class EventArgs.

The class CarDealer fires the event by calling the Invoke method of the delegate. This invokes all the handlers that are subscribed to the event.

```
NewCarInfo?.Invoke(this, new CarInfoEventArgs(car));
```

Firing the event is just a one-liner. However, this is only with C# 6.

Here is the same functionality implemented before C# 6.

```
EventHandler<CarInfoEventArgs> newCarInfo = NewCarInfo;
if (newCarInfo != null)
{
  newCarInfo(this, new CarInfoEventArgs(car));
}
```

With C# 6, all this could be replaced by using null propagation, with a single code line as you've seen earlier.

Before firing the event, it is necessary to check whether the delegate NewCarInfo is not null. If no one subscribed, the delegate is null:

```
protected virtual void RaiseNewCarInfo(string car)
{
  NewCarInfo?.Invoke(this, new CarInfoEventArgs(car));
}
```

Event Listener

The class Consumer is used as the event listener. This class subscribes to the event of the CarDealer and defines the method NewCarIsHere that in turn fulfills the requirements of the EventHandler<CarInfoEventArgs> delegate with parameters of type object and CarInfoEventArgs .

```
using static System.Console;

namespace ProCSharp.Delegates
{
  public class Consumer
  {
    private string _name;

    public Consumer(string name)
    {
      _name = name;
    }

    public void NewCarIsHere(object sender, CarInfoEventArgs e)
    {
      WriteLine($"{_name}: car {e.Car} is new");
    }
  }
}
```

Now the event publisher and subscriber need to connect. This is done by using the NewCarInfo event of the CarDealer to create a subscription with +=. The consumer Michael subscribes to the event, then the consumer Sebastian, and next Michael unsubscribes with -=.

Now the event publisher and subscriber need to connect. This is done by using the NewCarInfo event of the CarDealer to create a subscription with +=. The consumer Michael subscribes to the event, then the consumer Sebastian, and next Michael unsubscribes with -=.

```
namespace ProCSharp.Delegates
{
  class Program
  {
    static void Main()
    {
      var dealer = new CarDealer();

      var daniel = new Consumer("Daniel");
      dealer.NewCarInfo += michael.NewCarIsHere;

      dealer.NewCar("Mercedes");

      var sebastian = new Consumer("Sebastian");
      dealer.NewCarInfo += sebastian.NewCarIsHere;

      dealer.NewCar("Ferrari");
```

```
      dealer.NewCarInfo -= sebastian.NewCarIsHere;

      dealer.NewCar("Red Bull Racing");
    }
  }
}
```

Running the application, a Mercedes arrived and Daniel was informed. After that, Sebastian registers for the subscription as well, both Daniel and Sebastian are informed about the new Ferrari. Then Sebastian unsubscribes and only Daniel is informed about the Red Bull:

```
CarDealer, new car Mercedes
Daniel: car Mercedes is new
CarDealer, new car Ferrari
Daniel: car Ferrari is new
Sebastian: car Ferrari is new
CarDealer, new car Red Bull Racing
Daniel: car Red Bull is new
```