

Ambientes Virtuais de Execução

Eventos

Eventos

- ▶ A definição de um membro do tipo `Evento` permite a um determinado tipo notificar outros objectos que algo aconteceu.
 - ▶ Ex: a classe `Button` tem um evento designado por `Click`
- ▶ Todas as classes que representam um evento têm de derivar da classe `System.EventArgs`

```
public class EventArgs{  
    public static readonly EventArgs Empty = new EventArgs();  
    public EventArgs(){ }  
}
```

- ▶ Quando não é necessário passar dados aquando da notificação de eventos, não é necessário definir um novo tipo de evento
 - ▶ Pode passar-se `EventArgs.Empty` ou `null`.

TemperatureChangedEventArgs

```
public class TemperatureChangedEventArgs : EventArgs{

    private readonly int oldTemperature, newTemperature;

    public TemperatureChangedEventArgs(int oldTemp, int newTemp){
        oldTemperature = oldTemp;
        newTemperature = newTemp;
    }

    public int OldTemperature{ get { return oldTemperature; } }

    public int NewTemperature{ get { return newTemperature; } }
}
```

Modelo de Eventos

- ▶ O modelo de eventos na framework .Net é baseado na existência de um *delegate* que conecta um evento com o seu *handler*.
- ▶ O tipo *delegate* define o conjunto de argumentos que são passados ao método que manipula o evento
 - ▶ Todos os que recebem a notificação do evento têm de fornecer um método *callback* que esteja de acordo com o *delegate* definido

EventHandler

- ▶ EventHandler é um *delegate* pré-definido
 - ▶ para quando o evento não gera dados

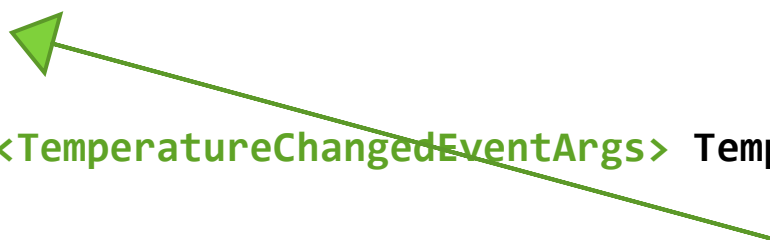
```
public delegate void EventHandler(Object sender, EventArgs e)
```

- ▶ EventHandler<TEventArgs> é um *delegate* pré-definido
 - ▶ neste caso é indiferente se o evento gera dados ou não.

```
public delegate void  
EventHandler<TEventArgs>(Object sender, TEventArgs e)  
                        where TEventArgs : EventArgs
```

Thermostat

```
public class Thermostat{  
    private int temperature;  
    public event EventHandler<TemperatureChangedEventArgs> TemperatureChanged;  
  
    virtual protected void  
    OnTemperatureChanged(TemperatureChangedEventArgs args){  
        if(TemperatureChanged!=null)  
            TemperatureChanged(this,args);  
    }  
  
    public int Temperature{  
        get { return temperature; }  
        set {  
            TemperatureChangedEventArgs args;  
            args=new TemperatureChangedEventArgs(temperature, value);  
            temperature = value;  
            OnTemperatureChanged(args);  
        }  
    }  
}
```



**Tipo produtor de
acontecimentos
(Observable)**

TemperatureChangeObserver

```
public class TemperatureChangeObserver{

    public TemperatureChangeObserver(Thermostat t){
        t.TemperatureChanged += this.TemperatureChanged;
    }

    public void
    TemperatureChanged(Object sender, TemperatureChangedEventArgs temp){
        Console.WriteLine("ChangeObserver: Old={0}, New={1}, Change={2}",
            temp.OldTemperature ,
            temp.NewTemperature,
            temp.NewTemperature-temp.OldTemperature
        );
    }
}
```

TemperatureAverageObserver

```
public class TemperatureAverageObserver{
    private int sum=0, count=0;
    public TemperatureAverageObserver(Thermostat t){
        t.TemperatureChanged += this.TemperatureChanged;
    }

    public void
    TemperatureChanged(Object sender, TemperatureChangedEventArgs temp){
        count++;
        sum+=temp.NewTemperature;
        Console.WriteLine("AverageObserver: Average={0:F},
                           (double)sum/(double)count);
    }
}
```


Testing...

```
static void Main(string[] args){  
    Thermostat thermo=new Thermostat();  
    TemperatureChangeObserver tc = new TemperatureChangeObserver( thermo);  
    TemperatureAverageObserver ta = new TemperatureAverageObserver( thermo);  
    thermo.Temperature=50;//set  
    thermo.Temperature=20;//set  
    thermo.Temperature=40;//set  
}
```

```
ChangeObserver: Old=0, New=50, Change=50  
AverageObserver: Average=50,00  
ChangeObserver: Old=50, New=20, Change=-30  
AverageObserver: Average=35,00  
ChangeObserver: Old=20, New=40, Change=20  
AverageObserver: Average=36,67
```

Pseudo IL

- ▶ Cada membro evento é suportado por um par de métodos e um campo do tipo delegate correspondente

```
.field private class
    [mscorlib]System.EventHandler`1<class TemperatureChangedEventArgs> TemperatureChanged
.method public hidebysig specialname instance void
    add_TemperatureChanged(class [mscorlib]System.EventHandler`1<class
TemperatureChangedEventArgs> 'value') cil managed{
// . . .
IL_000b: call class [mscorlib]System.Delegate::Combine(class
                [mscorlib]System.Delegate, class [mscorlib]System.Delegate)
// . . .
}
.method public hidebysig specialname instance void
    remove_TemperatureChanged(class [mscorlib]System.EventHandler`1<class
TemperatureChangedEventArgs> 'value') cil managed
{
//...
    IL_000b: call class [mscorlib]System.Delegate::Remove(class
[mscorlib]System.Delegate, class [mscorlib]System.Delegate)
//...
}
```

Controlar explicitamente o registo de eventos

```
public class Thermostat{
    private int temperature;
    private readonly Object eventLock=new Object();
    private event EventHandler<TemperatureChangedEventArgs> _TemperatureChanged;
    public event EventHandler<TemperatureChangedEventArgs> TemperatureChanged{
        add{lock(eventLock){_TemperatureChanged+= value;}}
        remove{lock(eventLock){_TemperatureChanged -= value;}}
    }

    virtual protected void
    OnTemperatureChanged(TemperatureChangedEventArgs args){
        EventHandler<TemperatureChangedEventArgs> temp = _TemperatureChanged;
        if(temp!=null)
            temp(this,args);
    }

    public int Temperature{
        get { return temperature; }
        set {
            TemperatureChangedEventArgs args;
            args=new TemperatureChangedEventArgs(temperature, value);
            temperature = value;
            OnTemperatureChanged(args);
        }
    }
}
```

Padrão Observer

- ▶ O padrão **Observer** é um padrão de software no qual um objecto, designado por sujeito (*Observable*), mantém uma lista de dependentes, designados por observadores (*Observers*) e notifica-os automaticamente de qualquer mudança de estado.
- ▶ O padrão Observer pode ser implementado em C# recorrendo ao modelo de Eventos
 - ▶ Thermostat → Observable
 - ▶ TemperatureChangeObserver
 - ▶ TemperatureAverageObserver

} Observers

.Net Reactive Framework (Rx)

- ▶ Contém as interfaces `IObserver/IObservable`
 - ▶ Capturam a essência do padrão `Observer`

```
interface IObservable <out T>{  
    IDisposable subscribe(IObserver o);  
}
```

```
interface IDisposable{  
    void Dispose( );  
}
```

```
interface IObservable <out T>{  
    void OnCompleted();  
    void OnNext(T v);  
    void OnError(Exception e);  
}
```

.Net Reactive Framework (Rx)

Temperature

```
public class Temperature{

private int oldTemperature, newTemperature;

public Temperature(int oldTemp, int newTemp){
    oldTemperature=oldTemp;
    newTemperature=newTemp;
}

public int OldTemperature{get{ return oldTemperature;}}
public int NewTemperature{get{return newTemperature;}}

}
```

.Net Reactive Framework (Rx)

Thermostat

```
public class Thermostat : IObservable<Temperature>{
    private int temperature =0;  private List< IObservable< Temperature > > observers;
    public Thermostat() { observers = new List< IObservable< Temperature > > (); }
    public IDisposable Subscribe(IObservable<Temperature> observer) {
        if (! observers.Contains(observer))      observers.Add(observer);
        return new Unsubscriber(observers, observer);}
    public int Temperature{
        get { return temperature;}
        set {
            Temperature t=new Temperature(temperature,value);
            temperature=value;
            foreach (var observer in observers) {  observer.OnNext(t); } }
    }

    private class Unsubscriber : IDisposable{
        private List<IObservable<Temperature>>_observers;
        private IObservable<Temperature> _observer;
        public Unsubscriber(List<IObservable<Temperature>> obsers, IObservable<Temperature> obser){
            this._observers = obsers;
            this._observer = obs;}
        public void Dispose() {
            if(_observer != null && _observers.Contains(_observer)) _observers.Remove(_observer);}
    }
}
```

.Net Reactive Framework (Rx)

TemperatureChangeObserver

```
public class TemperatureChangeObserver : IObservable<Temperature>{
    private IDisposable unsubscriber;
    public virtual void Subscribe(IObservable<Temperature> provider){
        if (provider != null) unsubscriber = provider.Subscribe(this);
    }
    public virtual void OnCompleted() {
        Console.WriteLine("ChangeObserver:Completed");
        this.Unsubscribe();
    }
    public virtual void OnError(Exception e){
        Console.WriteLine("ChangeObserver:Can not be completed");
    }
    public virtual void OnNext(Temperature value) {
        Console.WriteLine("ChangeObserver: Old={0}, New={1}, Changed={2}",
            value.OldTemperature, value.NewTemperature,
            value.NewTemperature-value.OldTemperature);
    }

    public virtual void Unsubscribe() { unsubscriber.Dispose(); }
}
```



.Net Reactive Framework (Rx)

TemperatureAverageObserver

```
public class TemperatureAverageObserver : IObservable<Temperature>{
    private IDisposable unsubscriber;
    private int sum=0,count=0;
    public virtual void Subscribe(IObservable<Temperature> provider){
        if (provider != null)
            unsubscriber = provider.Subscribe(this);
    }
    public virtual void OnCompleted() {
        Console.WriteLine("AverageObserver: Completed");
        this.Unsubscribe();
    }
    public virtual void OnError(Exception e){
        Console.WriteLine("AverageObserver: Can not be completed");
    }
    public virtual void OnNext(Temperature value)
    {
        sum=sum+value.NewTemperature; count++;
        Console.WriteLine("AverageObserver: Average={0:F}", (double)sum/(double)count);
    }
    public virtual void Unsubscribe(){ unsubscriber.Dispose(); }
}
```

