Curs11

Evenimente C#

Ce este un eveniment?

Este o notificare trimisă de către un obiect care declanşeanză executarea unei / unor acțiuni!

În C# implementarea evenimentelor s-a făcut după șablonul `Observer` (design pattern `Observer`). Obiectul observat trimite notificările către obiectele observabile fără să știe nimic despre acestea. Mai mult, evenimentele C# sunt implementate cu ajutorul tipului `delegate`

Cum se construiește un eveniment?

1. Se declară un delegat care indică semnătura acţiunilor de executat.
2. Se declară evenimentul: o variabilă de tip delegatul anterior, în plus aparând şi cuvântul cheie `event`

Aplicaţie: Să se implementeze o listă de obiecte (colecţie) înzestrată cu notificări la adăugarea / eliminarea obiectelor. Adică să putem adăuga acţiuni de executat pentru evenimente de tipul: `OnAdded` şi `OnRemoved`.

```
namespace EventApplication {
  // 1. Declararea delegatului
  public delegate void Notify<R> (R elem);

  public class ObservableList<T> {
    public List<T> Elements { get; private set; }
    // 2. Declararea evenimentelor
    public event Notify<T> Added;
    public event Notify<T> Removed;

    public ObservableList() {
      Elements = new List<T> ();
    }

    public void Add(T newElement) {
      Elements.Add(newElement);
      OnAdded(newElement);
    }

    public void Remove(T element) {
      if (Elements.Contains(element))
        Elements.Remove(element);

      OnRemoved(element);
    }

    protected virtual void OnAdded(T elem) {
      Added?.Invoke(elem);
```

```
      //if(Added != null) {
      //    Added.Invoke();
      //}
    }

    protected virtual void OnRemoved(T elem) {
      Removed?.Invoke(elem);
    }
  }
}
```

Evenimente predefinite: `EventHandler`, `EventHandler<EventArgs>`

- `EventHandler`: pentru evenimente fără transmitere de date
- `EventHandler<EventArgs>`: pentru evenimente cu transmitere de date

```
namespace EventApplication {
  // 1. Declararea delegatului
  // public delegate void Notify<R>(R elem);
  public class ObservableList<T> {
    public List<T>Elements { get; private set; }

    // 2. Declararea evenimentelor
    public EventHandler Added;
    public EventHandler Removed;

    public ObservableList() {
      Elements = new List<T>();
    }

    public void Add(T newElement) {
      Elements.Add(newElement);
      OnAdded(this, EventArgs.Empty);
    }

    public void Remove(T element) {
      if (Elements.Contains(element))
        Elements.Remove(element);
      OnRemoved(this, EventArgs.Empty);
    }

    protected virtual void OnAdded(object sender, EventArgs args) {
      Added?.Invoke(sender, args);
      //if(Added != null) {
      //    Added.Invoke();
      //}
    }

    protected virtual void OnRemoved(object sender, EventArgs args) {
      Removed?.Invoke(sender, args);
    }
  }
}
```

Varianta cu transmitere de date către eveniment:

```
namespace EventApplication {
  // 1. Declararea delegatului
  // public delegate void Notify<R>(R elem);
  public class ObservableList<T> {
    public List<T>Elements { get; private set; }

    // 2. Declararea evenimentelor
    public EventHandler<T> Added;
    public EventHandler<T> Removed;

    public ObservableList() {
      Elements = new List<T>();
    }

    public void Add(T newElement) {
      Elements.Add(newElement);
      OnAdded(this, newElement);
    }

    public void Remove(T element) {
      if (Elements.Contains(element))
        Elements.Remove(element);
      OnRemoved(this, element);
    }

    protected virtual void OnAdded(object sender, T args) {
      Added?.Invoke(sender, args);
      //if(Added != null) {
      //    Added.Invoke();
      //}
    }

    protected virtual void OnRemoved(object sender, T args) {
      Removed?.Invoke(sender, args);
    }
  }
}
```

Se pot transmite mai multe informaţii `custom` către evenimente prin implementarea unei clase derivate din `EventArgs` şi încapsularea datelor în aceasta.

```
namespace EventApplication {
  // 1. Declararea delegatului
  // public delegate void Notify<R>(R elem);
  public class CustomEventArgs<T>: EventArgs {
    public T elem { get; set; }
    public DateTime Date { get; set; }
  }

  public class ObservableList<T> {
    public List<T>Elements { get; private set; }

    // 2. Declararea evenimentelor
    public EventHandler<CustomEventArgs<T>> Added;
    public EventHandler<CustomEventArgs<T>> Removed;

    public ObservableList() {
```

```
      Elements = new List<T>();
    }

    public void Add(T newElement) {
      Elements.Add(newElement);
      OnAdded(this, new CustomEventArgs<T> { elem = newElement, Date =
DateTime.Now });
    }

    public void Remove(T element) {
      if (Elements.Contains(element))
        Elements.Remove(element);
      OnRemoved(this, new CustomEventArgs<T> { elem = element, Date =
DateTime.Now });
    }

    protected virtual void OnAdded(object sender, CustomEventArgs<T> args) {
      Added?.Invoke(sender, args);
      //if(Added != null) {
      //    Added.Invoke();
      //}
    }

    protected virtual void OnRemoved(object sender, CustomEventArgs<T> args) {
      Removed?.Invoke(sender, args);
    }
  }
}
```