Händelser och delegater

Event/Delegate fördjupning

Sammanfattning

Idag?

- ‘Delegate’
- ‘Event’
- ‘EventHandler’
- ‘Observer’ designmönstret

s.30-38 och s.133-158

Delegat - repetition

typ deklaration: public delegate int Calculate (int n1, int n2);

...

void add(int n1, int n2) {Console.Write(“Add:”n1 + n2 + “ ”);}

void sub(int n1, int n2) {Console.Write(“Sub:”n1 - n2);}

...

instans: Calculate calc=myclass.add;

calc+=myclass.sub;

invokation: calc(2, 3); -> Add:5 Sub:-1

Delegat

- Använder egen Invoke metod
- + eller += är egentligen Delegate.Combine metoden
- Man kan också ta bort metoder (som calc -= myclass. sub) som är Delegate.Remove metoden
- är ‘oföränderlig’ som strängar och säkert att skicka och manipulera utan att orsakar problem för originalet

*Hinder till ‘garbarge collection’ av klass/metod om delegat instans kan inte tas bort

Delegat

- Delegate.GetInvocationList tar fram listan av instanser
- De exekveras i ordning
- I [a,b,c] om b har en Exception, c aldrig exekveras

Event - repetition

public delegate void TickHandler(this, EventArgs e);

…

public event TickHandler Tick;

this.Tick += new TickHandler(MinTickMetod);

delegate TickHandler nu regera på händelsen....

Tick(this, new EventArgs());

med kod i MinTickMetod {...}

Händelse - ‘Event’

- En Event (händelse) är inte riktigt en delegat själv - det är mer som en egenskap är till en instansvariabel, dvs dörren till en delegat

Delegat - ‘EventHandler’

Däremot är EventHandler en delegat (och kan användas lite som en Event).

```
public EventHandler<MyEventArgs> Tick;
…
Tick += MinTickMetod;
…
Tick(this, MyEventArgs); //när Tick invokationen händer
-> MinTickMetod exekveras
```

Delegat - ‘EventHandler’

public void MinTickMetod(object sender, MyEventArgs e)
{...}

Ser det bekant ut? Som…

public void MyButton_Click(object sender, EventArgs e) {...}

EventHandler används väldigt mycket i UI.

Delegat - ‘EventHandler’

‘Contravariance’ (s.138,139)

```
static void LogEvent(object sender, EventArgs e) {
Console.WriteLine("Händelse loggad!"); }
```

```
button.KeyPress += LogEvent;
button.MouseClick += LogEvent;
```

LogEvent tar emot alla trots att de egentligen har KeyPressEventArgs och MouseEventArgs

Delegat - ‘EventHandler’

‘Covariance’ (s.140)

```
delegate Stream StreamFactory();

static MemoryStream GenerateSampleData {...}

StreamFactory factory = GenerateSampleData;
//Fungerar för att delegaten tar MemoryStream som en giltig Stream (Stream är superklassen)
```

Observer designmönstret - koncept

Exempel - Labyrint

- MazeLogic räknar ut vilka vägg som borde förstöras (händelse - Square Wall Removed) och vilka rutar som tillhör lösningen (händelse - Path Segment Drawn).

- Metoder RemoveLine och FillPathSquare i LabyrinthMazeGridForm tittar på och reagera med att ändra om labyrinten.

Exempel - Labyrint (cont)

Exempel:

https://github.com/robinos/LabyrinthWindowsForms

Observer användningsområden

- UI-händelser med kod reaktion
- Kodberäkningar/uppdatering till UI
- Mellan klasser (ie. reagera när x)
- En viktigt del av olika andra designmönster

Andra Observer implementationer

IObserver<T>

En klass som implementera IObserver<T> skickar en kopia av sig själv till en klass som implementera IObservable<T> för att Subscribe (prenumerera) till den.

Det kräver att man implementera metoder för att ta emot meddelanden med OnNext (ny data), OnError (fel) och OnCompleted (ingen ny data).

Andra Observable implementationer

IObservable<T>

En klass som implementera IObservable<T> måste ha metoden Subscribe för att kom ihåg alla Observer objekt som tittar på och meddela om ny data, meddela vid fel och meddela när ingen ny data finns.

Delegat - anonymous metoder

Anonyma metoder : delegate void Action<T>(T obj)

```
Action<string> printReverse = delegate(string text) {
char[] chars = text.ToCharArray();
Array.Reverse(chars);
Console.WriteLine(new string(chars)); };

printReverse("Hello world");
```

Delegat - anonyma metoder

Anonyma metoder kan inte använda ‘this’ om metoden är i en värdetyp (som en struct)

```
List<int> minLista = new List<int>();
minLista.Add(5);
minLista.Add(10);                    //ForEach tar en delegat
minLista.ForEach( delegate(int n) {
    Console.WriteLine(Math.Sqrt(n)); } );
```

Delegat - anonyma metoder

För att få en returvärde kan man använda Predicate<T>

public delegate bool Predicate<T>(T obj)

```
Predicate<int> isEven=delegate(int x) { return x % 2 == 0; };
Console.WriteLine(isEven(1));
Console.WriteLine(isEven(4));
```

Delegat - anonyma metoder

Ignorera parameter

```
button.Click+=delegate{Console.WriteLine("LogPlain"); };
```

istället för:

```
button.Click+=delegate(object sender, EventArgs e) { ... };
```

Delegat - anonyma metoder

Anonyma metoder kan använda sig av variabler i yttremetoden

```
{
string capturedVariable = "captured";
...
MethodInvoker x = delegate() {
string anonLocal = "local to anonymous method";
Console.WriteLine(capturedVariable + anonLocal); };
}
```

Delegat - anonyma metoder

Det är variabeln som blir fångad, inte bara värdet!

En bra exempel av konsekvenser syns på s.150-158 (variabeln lever lika länge som delegatinstansen)

Exempel användningsområde:

```
List<Person> FindAllYoungerThan(List<Person> people, int limit) {
    return people.FindAll( delegate (Person person)
    { return person.Age < limit; } ); }
```