Webbprogrammering

Klasser och metoder.
Subrutiner och funktioner.

Metod

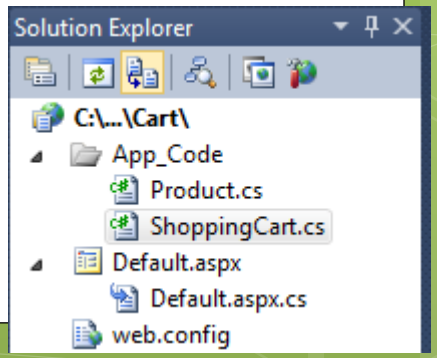
- En metod är ett kodblock som utför något…
- Metoder kan bakas in i klasser för att anropas, tex då en användare klickar på en knapp i gränssnittet
- Tex har ni metoden addProduct() i klassen ShoppingCart:

```
public static void addProduct(Product p)
{
    //Lägger till produkt i kundvagn.
    items.Add(p);
}
```

- Det finns subrutiner som inte returnerar något värde och så finns det funktioner som returnerar värde.

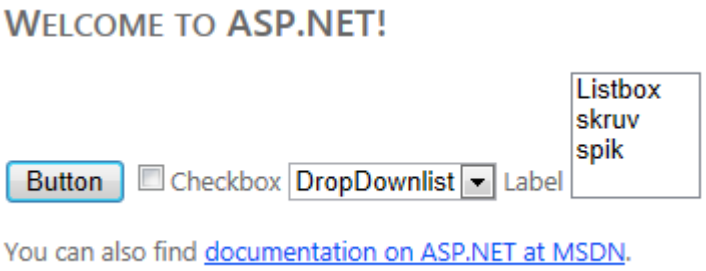
Varför använda kodblock?

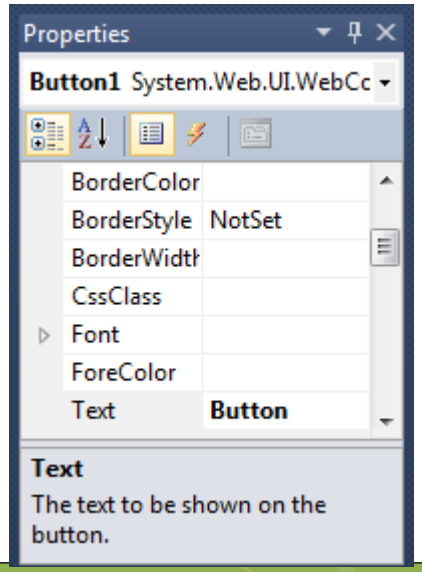
- Vi slipper skriva samma kod på flera ställen
- Vi slipper ändra kod på flera ställen
- Vi har kontroll på vad som skickas till dem och vad som returneras
- Det ger bättre överskådlighet i och med att den är uppdelad i mindre delar
- Det är lättare att felsöka
- Vi kan lättare återanvända delar av koden, både i andra projekt och internt i samma projekt
- Fristående kod läggs i klasser i mappen App_Code



Metoder i objekt

- Objekt baseras på en klass där klassen kan ses som en ritning på hur objektet ska se ut
- Det finns i ramverket .Net jättemånga färdiga klasser tex; button, dropdownlist, checbox, label…
  alla har de färdiga egenskaper och metoder från start



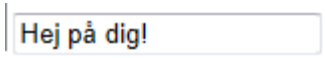


- Metoder har alltid ()
- Vi kan även skapa egna klasser med egna metoder (de kapslas in i klassen)

Ett exempel

- När man klickar på knappen visas texten ”Hej på dig!” i en textruta, vad är objekt och metod?

```
protected void Button1_Click(object sender, EventArgs e)
{
    TextBox1.Text = "Hej på dig!";
}
```



- Alla objektets egenskaper skickas med i objekt sender och felmeddelande fångas upp i EventArgs e.
- En egen metod som vi skapar i en klass - ShoppingCart:

```
private static ArrayList items = new ArrayList();

public static void addProduct(Product p)
{
    //Lägger till produkt i kundvagn.
    items.Add(p);
}
```

- Behöver vi ändra kan vi göra detta på endast ett ställe

Räckvidd

- Om vi vill att våra metoder ska kunna synas utanför våra klasser behöver vi använda public i början av metod
- Vill vi bara använda variabler inne i en klass använder vi ordet private istället.

```
public class Product
{
    //Definierar klassens variablar (egenskaper)
    public String name { get; set; }
    public String price { get; set; }
    public String description { get; set; }
    private String intern { get; set; }

    public Product(String name, String description, String price)
    {
        //Anger värden för variablar (i kontstruktorn, funktionen som skapar klassen)
        this.name = name;
        this.description = description;
        this.price = price;
    }
}
```

Räckvidd 2

- Vi initierar myCalc direkt i codebehind-klassen, då kan vi nå objektet i kodblock, se i switch – case sats

```
public partial class _Default : System.Web.UI.Page
{
    // Deklareras i defaultklassen för sidan (en och samma kalkylator används på sidan)
    Calc myCalc = new Calc();

    protected void Page_Load(object sender, EventArgs e)
    {
        // man kan koda innehåll i operatorlistan med...
        OperatorList.Items.Add("pi");
    }

    protected void CalculateButton_Click(object sender, EventArgs e)
    {
        if (ValueBox1.Text.Length > 0 && ValueBox2.Text.Length > 0)
        {
            double result = 0;
            double value1 = Convert.ToDouble(ValueBox1.Text);
            double value2 = Convert.ToDouble(ValueBox2.Text);

           // operatorlistens värde kallar på rätt funktion
            switch (OperatorList.SelectedValue)
            {
                case "+":
                    result = myCalc.Add(value1, value2);
                    break;
                case "-":
                    result = myCalc.Subtract(value1, value2);
                    break;
                case "*":
                    result = myCalc.Multiply(value1, value2);
```

Anrop

- I er miniräknare var det exempelvis en beräkning:

```
public class Calc
{
    public double Add(double a, double b)
    {
        return b + a;
    }
    public double Subtract(double a, double b)
    {
        return b - a;
    }
    public double Multiply(double a, double b)
    {
        return b * a;
    }
```

```
// operatorlistens värde kallar på rätt funktion
switch (OperatorList.SelectedValue)
{
    case "+":
        result = myCalc.Add(value1, value2);
        break;
    case "-":
        result = myCalc.Subtract(value1, value2);
        break;
    case "*":
        result = myCalc.Multiply(value1, value2);
        break;
```

- Parametrar skickas med (value1, value2). De behöver vara av rätt datatyp:

```
double result = 0;
double value1 = Convert.ToDouble(ValueBox1.Text);
double value2 = Convert.ToDouble(ValueBox2.Text);
```

Laboration 4

- Vi kikar på laboration 4 som ni kör nästa vecka, se utdelade lösblad med labbanvisning och kod.
- Demo av Jesper
- Frågor?