



Linnéuniversitetet

Kalmar Vaxjö

Laborationsanvisning

Kassakvitto

Steg 1, laborationsuppgift 2



Författare: Mats Looch

Kurs: ASP.NET Web Forms

Kurskod: 1DV406

Upphovsrätt för detta verk

Detta verk är framtaget i anslutning till kursen ASP.NET Web Forms (1DV406) vid Linnéuniversitetet.

Du får använda detta verk så här:

Allt innehåll i detta verk av Mats Looock, förutom Linnéuniversitetets logotyp, symbol och kopparstick, är licensierad under:



Creative Commons Erkännande-IckeKommersiell-DelaLika 2.5 Sverige licens.

<http://creativecommons.org/licenses/by-nc-sa/2.5/se/>

Det betyder att du i icke-kommersiella syften får:

- kopiera hela eller delar av innehållet
- sprida hela eller delar av innehållet
- visa hela eller delar av innehållet offentligt och digitalt
- konvertera innehållet till annat format
- du får även göra om innehållet

Om du förändrar innehållet så ta inte med Linnéuniversitetets logotyp, symbol och/eller kopparstick i din nya version!

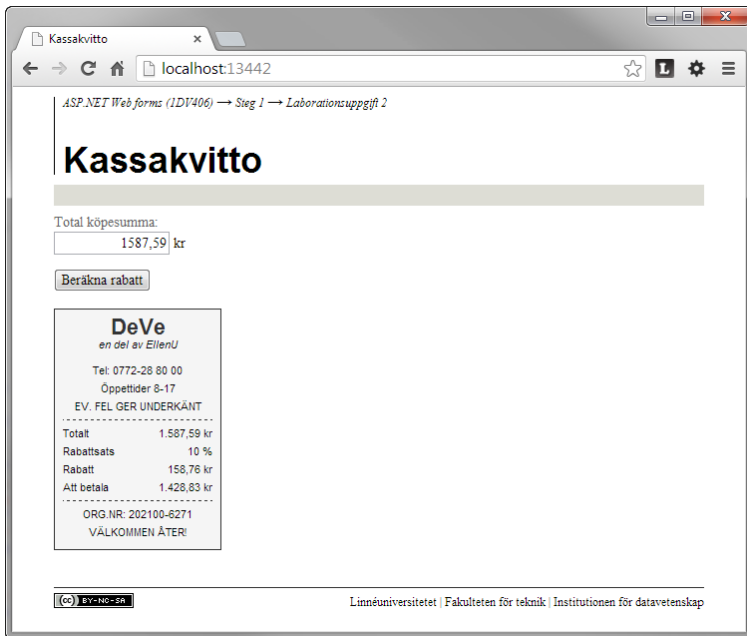
Innehåll

Uppgift	5
Inledning	Fel! Bokmärket är inte definierat.
Inkapsling (<i>Separation of Concern</i>)	Fel! Bokmärket är inte definierat.
Hantering av fel på servern	Fel! Bokmärket är inte definierat.
Mål	Fel! Bokmärket är inte definierat.
Läsvärt	Fel! Bokmärket är inte definierat.

Uppgift

Inledning

Skriv en webbapplikation där användaren matar in en total köpesumma. Beroende på köpesummans storlek ska en bestämd rabattsats tillämpas och beräkning av den erhållna rabatten och summan att betala ska ske och presenteras i form av ett kvitto.



Kassakvitto

ASP.NET Web forms (1DV406) → Steg 1 → Laborationsuppgift 2

Kassakvitto

Total köpesumma:

1587,59 kr

Beräkna rabatt

DeVe
en del av EllenU
Tel: 0772-28 80 00
Öppettider 8-17
EV. FEL GER UNDERKÄNT

Totalt	1 587,59 kr
Rabattsats	10 %
Rabatt	158,76 kr
Att betala	1 428,83 kr

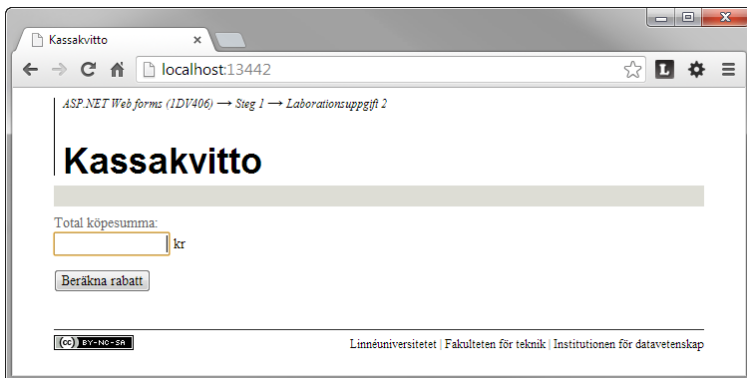
ORG.NR: 202100-8271
VÄLKOMMEN ÅTER!

(cc) BY-NC-SA Linnéuniversitetet | Fakulteten för teknik | Institutionen för datavetenskap

Figur 1. Sida efter att beräkning av rabatt skett.

Presentationslogiklagret

Du kan fritt utforma formuläret som användaren ska mata in uppgifterna i som sedan skickas till webbservern. Data som skickas till servern ska vara validerat.



Kassakvitto

ASP.NET Web forms (1DV406) → Steg 1 → Laborationsuppgift 2

Kassakvitto

Total köpesumma:

kr

Beräkna rabatt

DeVe
en del av EllenU
Tel: 0772-28 80 00
Öppettider 8-17
EV. FEL GER UNDERKÄNT

Totalt	1 587,59 kr
Rabattsats	10 %
Rabatt	158,76 kr
Att betala	1 428,83 kr

ORG.NR: 202100-8271
VÄLKOMMEN ÅTER!

(cc) BY-NC-SA Linnéuniversitetet | Fakulteten för teknik | Institutionen för datavetenskap

Figur 2.

Då det sker en GET av sidan ska inget kvitto visas. Kontrollerna i figur 2 har renderats ut med hjälp av serverkontroller som TextBox och Button.

Genom att använda TextBox och de valideringskontroller som finns är det enkelt att kontrollera om användaren matat in data på rätt sätt. Valideringskontrollerna har du glädje av både på klienten och på server. På klienten sköts valideringen automatiskt med JavaScript. Valideringen ska fungera även om användaren inte tillåter att Javascript körs varför du även alltid måste validera på servern. På servern behöver du bara använda dig av egenskapen IsValid i klassen Page för att undersöka om data som skickats till sidan klarat valideringen eller inte.

Då sidan visas för användaren ska textfältet ha fokus och eventuellt innehåll ska vara markerat. För att lösa detta kommer dina kunskaper i JavaScript till användning eftersom detta krav bara kan lösas på klienten. Du löser kravet på textfältet genom att skriva en Javascript-funktion¹ (som t.ex. använder `focus()` och `select()`) som du ser till att den körs i samband med händelsen `onload` för `body`-elementet.

Kommandoknappen ska vara standardkommandoknapp ("*default button*") och väljas då användaren trycker på Enter-tangenten oavsett vilken kontroll som har fokus. Användaren ska alltså inte behöva klicka på kommandoknappen (eller tabba till den och trycka på mellanslagstangenten) för att skicka formuläret till webbservern, d.v.s. göra en "*postback*". Du behöver inte skriva en egen JavaScript-funktion som gör detta. Använd istället lämpligt attribut som `form`-elementet har och problemet är löst.

Presentationen av resultatet av beräkningen kan du utforma fritt inom ramen av ett kvitto (se figur 1). Tänk på att du kan använda den statiska metoden `String.Format` tillsammans med formatspecificerare som `{0:c}` för valuta och `{0:p0}` för procent utan decimaler och på så sätt skapa strängar med formaterat innehåll. Dessa strängar kan du sedan presentera med hjälp av `Label`-kontroller.

Validering

Innehållet i textfältet ska valideras så att innehållet garanterat kan tolkas som en total köpesumma, d.v.s. ett positivt tal som kan ha decimaler. Valideringen ska utföras i första hand på klienten. Men det är oerhört viktigt att all data valideras på servern oavsett om det validerats på klienten eller inte.

Textfältet måste vara ifyllt på ett korrekt sätt.

- Det får inte vara tomt.
- Texten måste kunna tolkas som flyttal som måste vara större än noll.

Med hjälp av kontrollen `RequiredFieldValidator` kan du säkerställa att ett textfält inte är tomt.

Kontrollen `CompareValidator` använder du till att undersöka om ett textfälts innehåll kan tolkas som ett flyttal eller inte och som dessutom är större än noll. Attributen/egenskaperna `Type`, `ValueToCompare` och `Operator` måste du sätta till lämpliga värden.

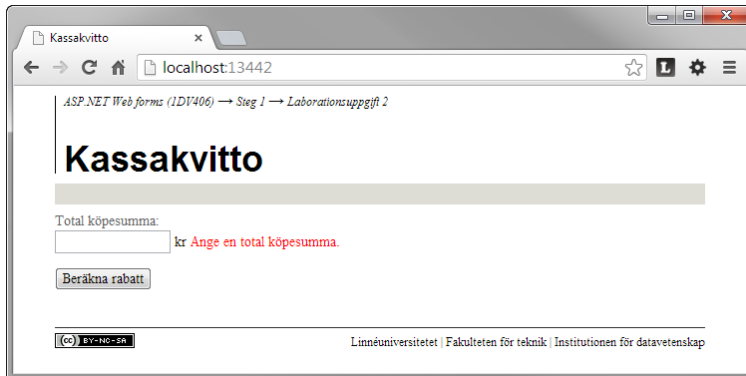
Det finns tre olika sätt ett meddelande kan visas på om användaren inte fyllt i textfältet på ett giltigt sätt.

- Ett meddelande i anslutning till textfältet som innehållet fältet.
- Använd kontrollen `ValidationSummary` och presentera alla meddelanden på ett och samma ställe.
- Använda kontrollen `ValidationSummary` och presentera alla meddelanden i en meddelanderuta.

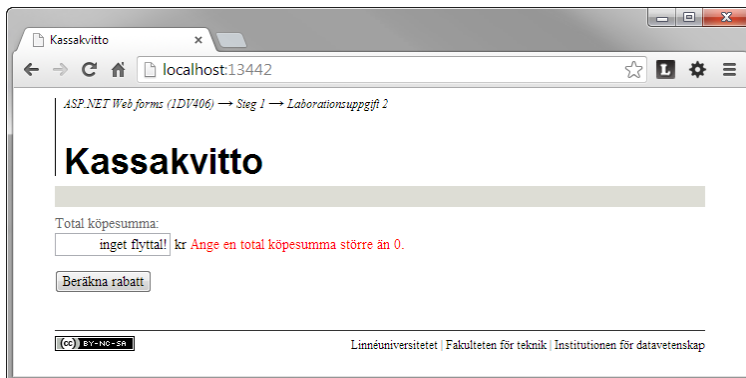
Enklast är att använda är det första alternativet. I denna laborationsuppgift ska du använda dig av det. (Du kommer att få prova på de övriga i kommande laborationsuppgifter.) För att visa ett meddelande i anslutning till textfältet använder du av tidigare nämnda valideringskontroller och sätter egenskapen `ErrorMessage` till det felmeddelande du vill visa.

Du väljer själv om du vill använda dig av "*unobtrusive validation*" eller inte.

¹ Om det inte vore för kravet att innehållet i textfält ska vara markerat hade du istället kunna använda dig av attributet `defaultfocus` `form`-elementet har.




Figur 3. Valideringen misslyckas på grund av att textfältet inte är ifyllt.



Figur 4. Valideringen misslyckas på grund av att texten i textfältet inte kan tolkas som ett flyttal.

Stilmallar

Självklart ska du använda dig av en, eller flera, stilmallar som ger utformningen av dokumentet. Lika självklart är det att dokumentet ska kunna användas även då en klient stängt av stilmallar.



Figur 5. Sida utan stilmall efter att beräkning av rabatt skett.

Affärslogiklagret

I tabell 1 finner du de rabattsatser du ska tillämpa. Är den totala köpesumman t.ex. 789 kr ska rabatten uppgå till 10 %. Är den totala köpesumman mindre än 500 kr utgår ingen rabatt.

Köpesumma	Rabatt
0-499	0 %
500-999	5 %
1000-4999	10 %
5000-	15 %

Tabell 1. Rabattsatser.

Kod som har med beräkning av rabattsats, rabatt och reducerat pris ska du självklart placera i en separat klass i katalogen Model så att den koden inte blandas med kod som har med användargränssnittet att göra.

Du ska följaktligen skapa klassen `Receipt` som ska användas för att just bestämma rabattsatsen, beräkna rabatten i kronor samt det nya priset då rabatten är frändragen. Implementera klassen `Receipt` enligt klassdiagrammet i figur 6. **OBS!** Samtliga medlemmar i klassen måste implementeras.

Klassen Receipt

Klassen `Receipt` ska med utgångspunkt av en total köpesumma beräkna rabatt i procent så väl som kronor och summa att betala efter att rabatten dragits av från den totala köpesumman.

Förutom en konstruktor och en metod har klassen några auto-implementerade egenskaper² och en vanlig egenskap kopplad till det enda privata fältet.

Det privata fältet `_subtotal`, och egenskapen `Subtotal`, representerar den totala köpesumman och sätts via metoden `Calculate`.

De auto-implementerade egenskaperna används till att representera rabatten i procent (`DiscountRate`), rabatten i kronor (`MoneyOff`) och beloppet efter att rabatten dragits från den totala köpesumman (`Total`).

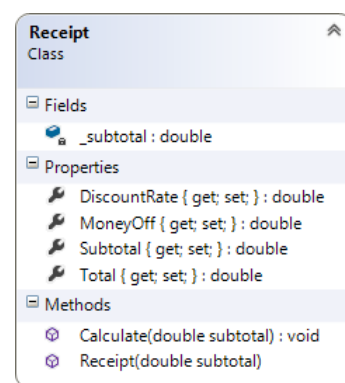
Samtliga egenskaper har publika `get`- och privata `set`-metoder och dess värden sätts i metoden `Calculate`.

Det ska vara helt omöjligt att tilldela fältet `_subtotal` ett värde som är mindre eller lika med 0. Av den anledningen ska du se till att egenskapen `Subtotal` validerar värdet den tilldelas och kasta ett undantag av typen `ArgumentOutOfRangeException` om den totala köpesumman är mindre eller lika med noll.

I metoden `Calculate` placerar du den kod som har med beräkningarna att göra. Här bestäms rabattsatsen, rabatten i kronor och det reducerade priset³. Metoden är den enda medlem som på får något sätt få påverka egenskapernas värden.

Hantering av fel på servern

Skulle ett oväntat fel av något slag inträffa ska användaren slippa se en ”gul-ful” sida med ett automatgenererat felmeddelande. Istället ska du visa en sida med ett användarvänligare(?) meddelande. Figur 7 visar ett exempel på en sådan sida.



Figur 6. Klassen Receipt.

² En auto-implementerad egenskap kräver inget fält eftersom ett dolt fält implementeras automatiskt. Det enda du behöver skriva för att deklarera egenskapen `DiscountRate` är `public double DiscountRate { get; private set; }` vilket är något enklare än att deklarera en vanlig egenskap som knyts till ett privat fält.

³ Det är högst olämpligt att låta egenskaper ha sidoeffekter varför kod som tilldelar flera egenskaper nya värden ska placeras i en metod och inte i en egenskaps `set`- eller `get`-metod.



Figur 7. Anpassad sida som visas vid fel istället för en "gul-ful" sida.

Mål

- Målet med laborationsuppgiften är att du ska skapa en webbapplikation där data valideras på så väl klienten som på servern.
- Efter denna laboration kommer du bildat kunskap gällande hur du implementerar validering med hjälp av några av de valideringskontroller som ASP.NET erbjuder. Under laborationen kommer du arbeta vidare med att skapa egna klasser, få ytterligare insikt i hur "postback" och händelser fungerar och att allt inte kan lösas på serversidan.
- Du ska veta att serverkontrollen `TextBox` renderas ut som ett input-element då egenskapen `TextMode` sätts till `SingleLine`.
- Kunna hantera det en `TextBox`-kontroll innehåller med hjälp av egenskapen `Text`.
- Du ska vara väl förtrogen med hur du validerar innehållet i ett textfält med hjälp av valideringskontrollerna som ASP.NET erbjuder.
- Förstå att då användaren klickar på en knapp sker en "postback" och en händelse skapas som kan tas om hand i "code-behind"-filen.
- Kunna ta hand om händelsen `Click` som inträffar då användaren klickar på en knapp.
- Använda `Label`-kontrollen för att placera ut text.
- Förstå vikten av att, och hur man, presenterar användarvänliga felmeddelanden.

Läsvärt

- Koncentrera dig på en sak i taget. Börja t.ex. med att ordna presentationslogiklagret så att grundläggande krav som validering uppfylls. Fortsätt sedan med hanteringen av fel på servern för att slutligen implementera hantering av "postback" och affärslogik.
- Läs om valideringskontrollerna på sidan "*Types of Validation for ASP.NET Server Controls*", [http://msdn.microsoft.com/en-us/library/bwd43d0x\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/bwd43d0x(v=vs.100).aspx).
- Använd en stilmall för att utforma dokumentet, gör det inte "inline". På sidan *Working with CSS Overview*, <http://msdn.microsoft.com/en-us/library/bb398931.aspx>, hittar du den information du behöver.