Kassa system AB

Introduktion till testing av it-system 2015-10-23

|  |  |
| --- | --- |
| Jacob Nienhuysen | jani1342 |
| Erik Holmström | erho4567 |
| Paulina Palmé | papa6605 |
| Diana Delta | leox9811 |

Obs! Ändra inte på formatet, det ska vara liggande A5.

# Introduktion

Den projektuppgift som valdes var Kvitton och Rabatter som går ut på att gruppen ska implementera klasser som skulle kunna användas i ett kassasystem för att representera kvitton och rabatter.

**Verktyg:  
Versionshanterare:**  
Vi valde att använda oss av GitHub:  
<https://github.com/jacobnienhuysen/INTEGrupp7.git>

**Byggserver**:  
Eclipse  
NetBeans  
  
**Byggscript**  
ANT

**Enhetstestramverk:**  
JUnit  
  
**Övriga verktyg**  
Notepad ++  
Office paketet  
Facebook

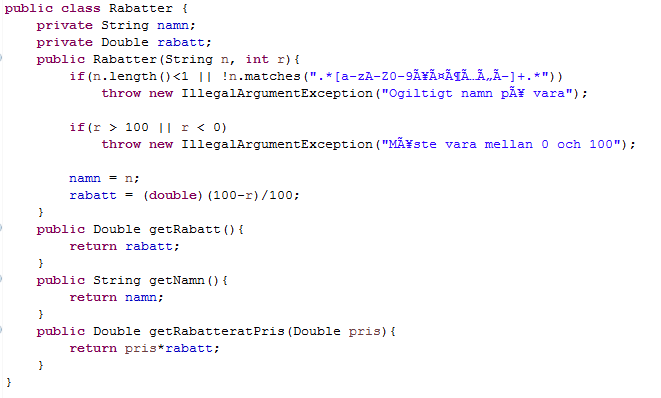
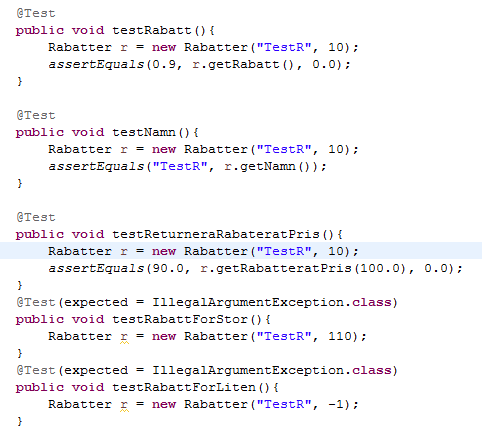
# Slutlig design

*En övergripande modell över systemet. Lämpligt format är ett eller flera klassdiagram, plus eventuella andra modeller som behövs för att förstå hur systemet är uppbyggt. Diagrammen ska vara läsbara. Det är dock fullständigt okej att de är detaljerade, bara det går att zooma in ordentligt på dem. Ett tips är att börja med ett översiktligt diagram som inte innehåller mer än paket och klassnamn, och att sedan lägga till mer detaljerade diagram efter det.*

# Testdriven utveckling – process

Efter att ha ritat upp ett klassdiagram över systemet valde vi att dela upp klasserna mellan oss. Därefter skrev vi ett JUnit test och sedan skrev vi den minsta mängden nödvändig kod för att testet skulle lyckas. Detta gjorde vi fram tills att klassen var klar och då gick vi vidare till nästa klass.

Exempel på TDD från systemet (Klass Rabatter):



# Testdriven utveckling – erfarenheter

Vi upplevde att projektet i sig är så pass litet att vi kände att det eventuellt hade varit enklare att skapa systemet utan TDD. Men vi förstår det praktiska med att använda sig av TDD i större projekt som är mer komplexa. Eftersom man har ett program som kompilerar vet man att man har kod som fungerar och inte behöver testas.

En annan erfarenhet som vi fått av att använda oss av TDD är att när man hittar fel i koden och gör en förändring, så finns testen där för att ge en direkt feedback på om man förstört något annat när man gjorde sin förändring.

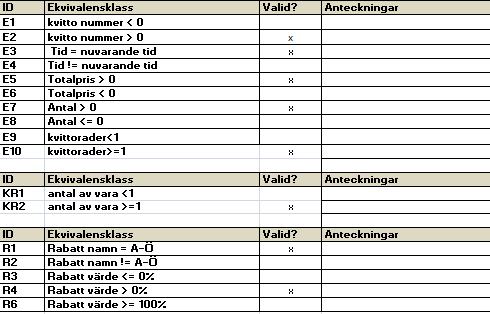
# Ekvivalensklassuppdelning – Hela systemet

*En kort presentation av vad ni valt ut för att tillämpa ekvivalensklassuppdelning på. Ni ska kort motivera valet, och ge tillräckligt med information för att det ska gå att bedöma er. Detta avsnitt och de tre föjande (till och med testmatrisen) ska finnas för samtliga delar ni tillämpat ekvivalensklassuppdelning på.*

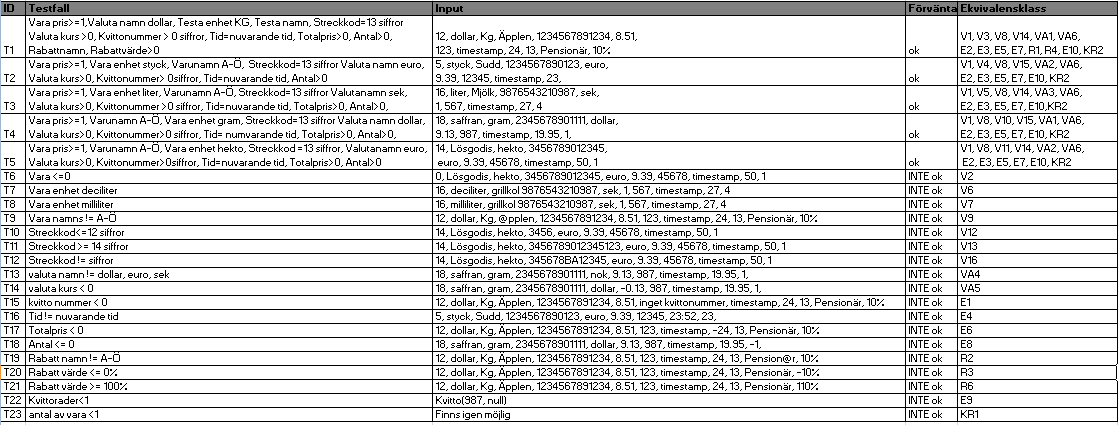
Då flera av klasserna i systemet kräver input i form av objekt av andra klasser i systemet, valde vi att utföra ekvivalensklassuppdelning på alla delar samtidigt. Exempelvis går det inte att skapa ett kvitto utan att man först skapat både ett valutaobjekt och en kvittorad som i sin tur kräver att man matar in ett objekt av klassen Vara.

Det ledde även till att flera av de valida ekvivalensklasserna överlappas i testfallen, för att testerna skulle generera godkända resultat.

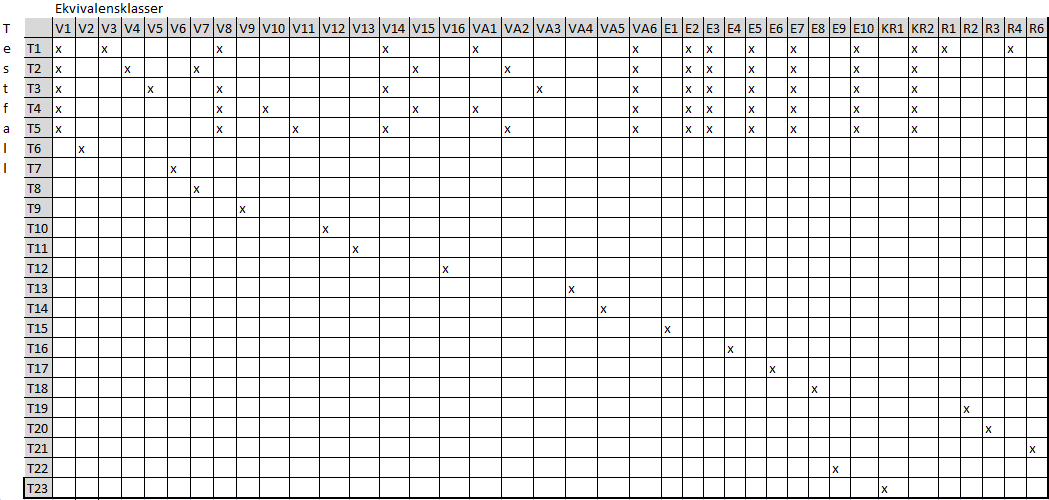
# Ekvivalensklasser – Hela systemet



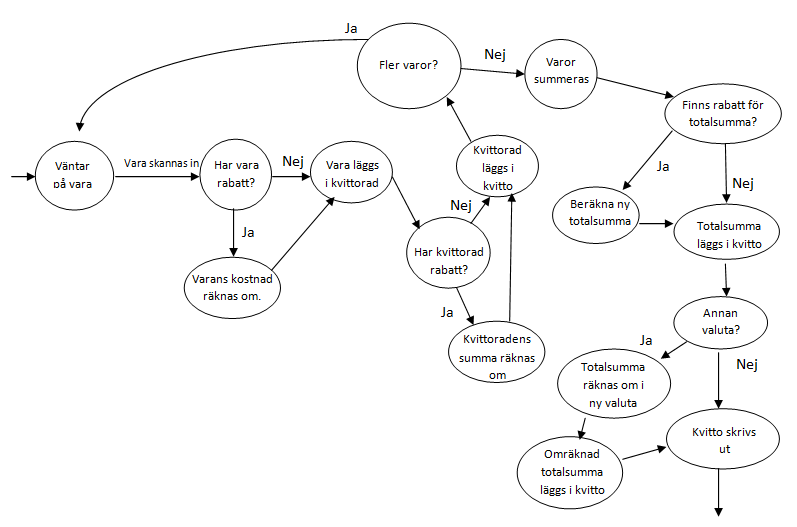
# Testfall – Hela systemet

**

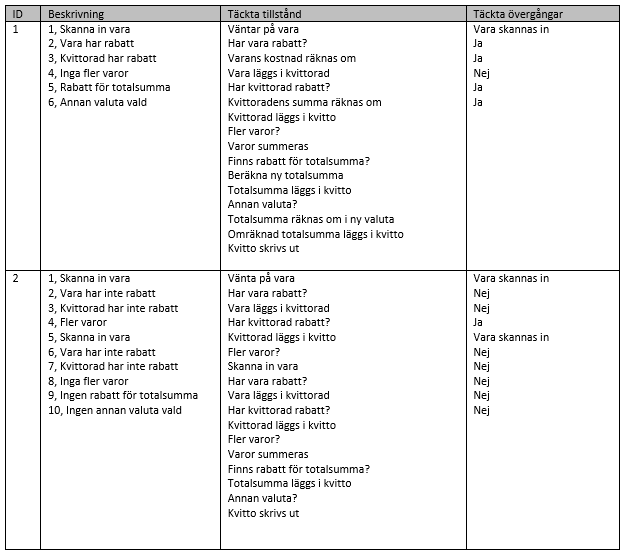
# Testmatris – Hela systemet



# Tillståndsbaserad testning

Eftersom vi valde att göra ekvivalensklasser för hela systemet valde vi att även gör en tillståndsmaskin för hela systemet. Vi har valt att använda oss av täckningskriteriumet statement coverage. Valet av täckningsgrad baseras på att vi dels vill täcka alla övergångar(bransch coverage) vilket gör att vi samtidigt täcker alla tillstånd och uppnår då statement coverage.

# Testfall för tillståndsbaserad testning



# Granskning

*En kort presentation av den del av koden ni valt ut för att göra en formell granskning av och processen ni använt er av inklusive eventuella checklistor, scenarier, edyl. Ni ska kort motivera valen, och ge tillräckligt med information för att det ska gå att bedöma er.*

# Granskningsrapport

*En lista över de påträffade felen och hur pass allvarliga ni bedömer dem.*

# Granskning – erfarenheter

*En diskussion om vilka era erfarenheter ni dragit av att tillämpa granskning. Det finns inget rätt eller fel här. Enda sättet att bli underkända är att bara fuska över punkten och bara säga något pliktskyldigt. Ni förväntas förhålla er till såväl kursboken som utdelat material och IEEE Std 1028.*

# Kodkritiksystem

*En presentation av de problem som hittats med hjälp av verktyg för statisk analys och en diskussion av dem enligt anvisningarna. Det räcker alltså inte med att bara lista problemen, ni måste förhålla er till dem också. Tänk också på att ni ska göra detta både på koden som den såg ut före granskningen och på koden efter att ni rättat det som kommit fram under granskningen*.

# Statiska mått

*En presentation och diskussion kring ett antal lämpliga statiska mått på koden. Att vi inte specificerar exakt vilka mått som ska tas upp beror på att olika verktyg har olika uppsättningar, men vi förväntar oss fler och mer intressanta mått än bara rena storleksmått som LOC, #klasser, #metoder, etc. Även här är det viktigt att förhållas sig till måtten, inte bara lista dem.*

# Täckningsgrad

*En översikt över vilken täckningsgrad era testfall uppnått. Denna kan antagligen tas rakt av från verktyget ni använt för att mäta den. Om ni inte uppnått fullständig täckning så ska detta förklaras och motiveras.*

# Profiler

*En kort presentation av hur ni gått tillväga för att testa koden med en profiler och vilka resultat ni fick fram.*

# Byggscript

*Byggscriptets första (seriösa) version, och den slutliga.*

# Övrigt

*Här kan ni ta upp övrigt av relevans för bedömningen av ert arbete. Om avsnittet inte behövs kan det plockas bort.*

**Internationalisering**  
Vi upptäckte att vårt kassasystem hanterade punkt och komma i pris på olika sätt beroende på om det kördes på en dator med operativsystemet inställt på svenska eller engelska. Detta eftersom att java hanterar tecknen olika. Vi valde att bortse från detta i vårt system eftersom att det skulle bli för omfattande för den här kursen.

**ANT tycker inte om svenska**  
Vi upptäckte att ANT inte alls tycker om Å, Ä, Ö vilket blev ett litet problem eftersom vissa klasser hade dessa svenska tecken i sig. Följden av att ANT inte tyckte om svenska tecken vart att ANT inte kunde köras.