Kassa System AB

Introduktion till testing av it-system 2015-10-26

|  |  |
| --- | --- |
| Grupp 7:  Jacob Nienhuysen | jani1342 |
| Erik Holmström | erho4567 |
| Paulina Palmé | papa6605 |
| Leo Oxfält | leox9811 |

# Introduktion

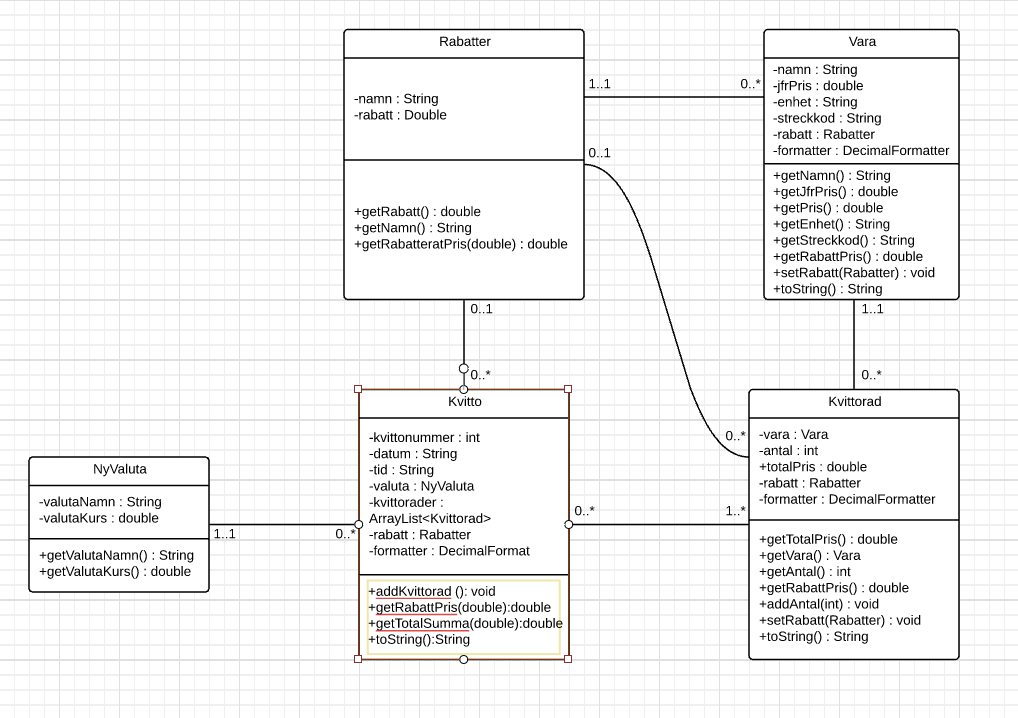
Den projektuppgift som valdes var Kvitton och Rabatter som går ut på att gruppen ska implementera klasser som skulle kunna användas i ett kassasystem för att representera kvitton och rabatter.

**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

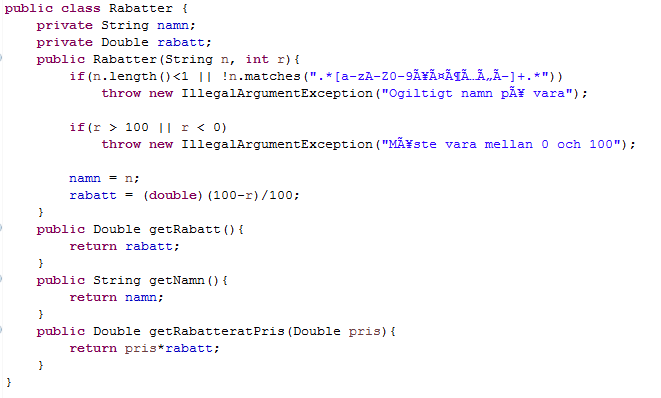
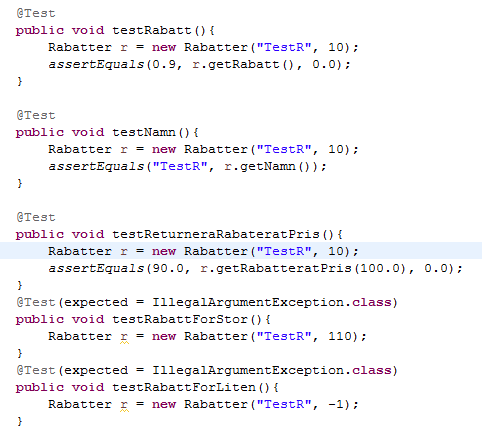
# Slutlig design

**

# 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.

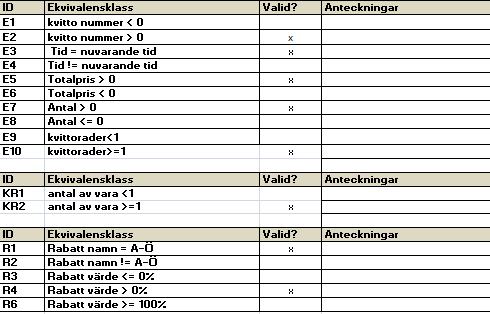
En annan erfarenhet som vi fått av att använda oss av TDD är att när man hittar fel i koden och gör en förändring, så finns testen där för att ge en direkt feedback på om man förstört något annat när man gjorde sin förändring.

# Ekvivalensklassuppdelning – Hela systemet

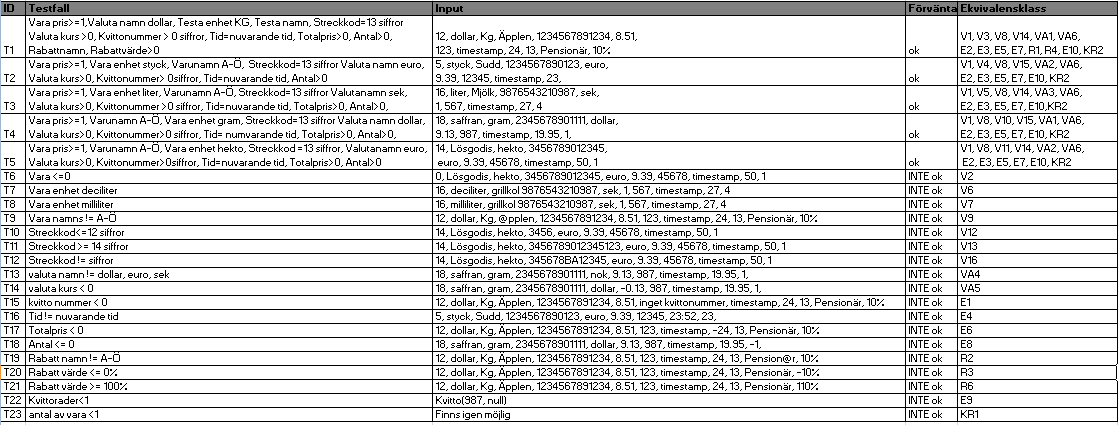
Då flera av klasserna i systemet kräver input i form av objekt av andra klasser i systemet, valde vi att utföra ekvivalensklassuppdelning på alla delar samtidigt. Exempelvis går det inte att skapa ett kvitto utan att man först skapat både ett valutaobjekt och en kvittorad som i sin tur kräver att man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

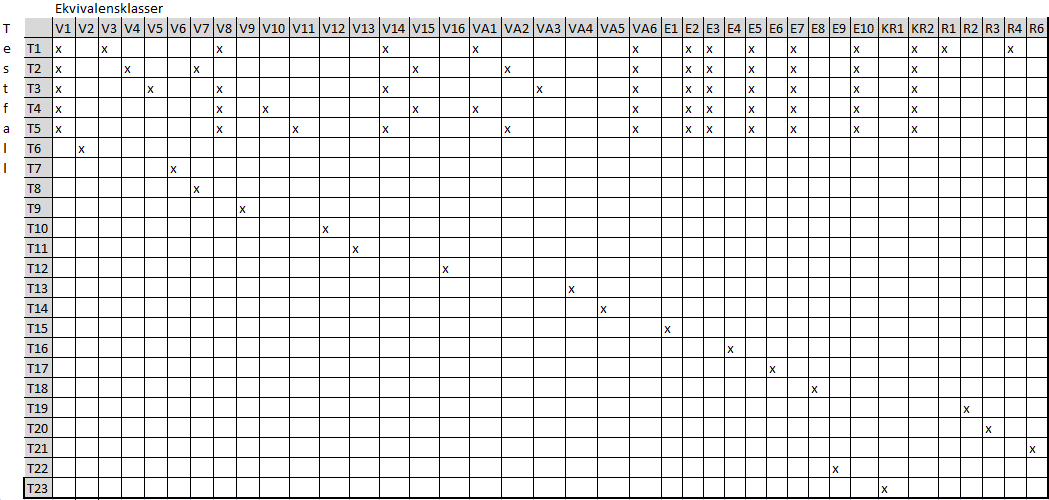
# Ekvivalensklasser – Hela systemet



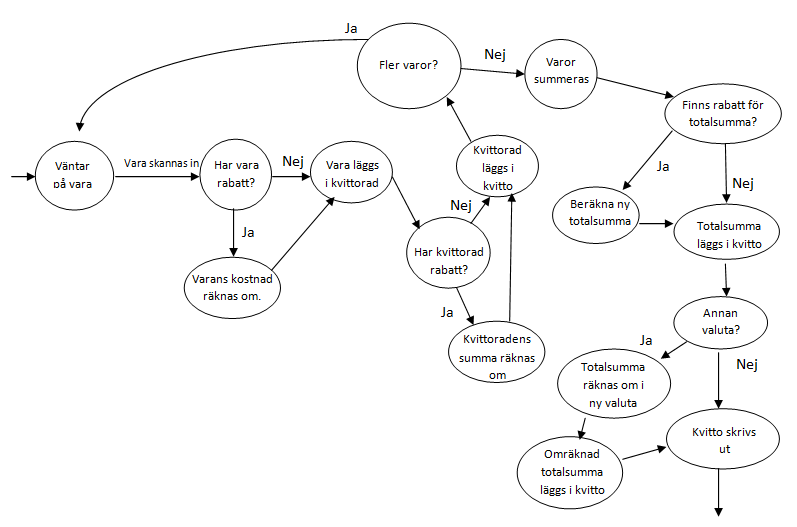
# Testfall – Hela systemet

**

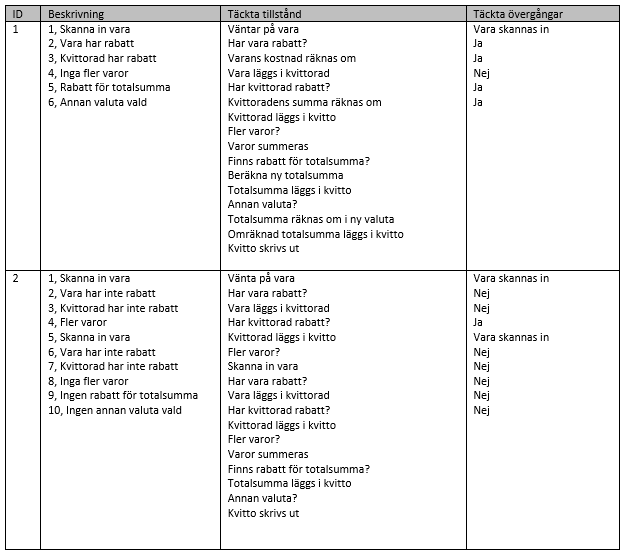
# Testmatris – Hela systemet



# 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

Vi valde att göra en formell granskning av hela systemet. Eftersom projektet är så pass litet valde vi att inte genomföra gransknigen fullt så formellt som den beskrivs i boken, utan genom en enklare inspektion av varandras skrivna klasser. Vid funderingar frågades författaren av klassens kod om hur författaren tänkt och varför. För att underlätta granskningen och se till att alla delar vart granskade valde vi att använda oss av checklistan från workshopen: (<https://ilearn2.dsv.su.se/pluginfile.php/51707/mod_resource/content/0/checklist.pdf>).  
Vi valde att använda denna checklista eftersom att vi använt den tidigare under workshopen och upplevde att den var enklare att följa en de som togs upp under föreläsningen.

Eftersom att koden inte är så stor och när vi kodat har vi varit väldigt noga upptäcktes inte så många fel under granskningen. De fel som kom upp under granskningen dokumenterades i en tabell som kan ses nedan.

# Granskningsrapport

|  |  |  |
| --- | --- | --- |
| Påträffat fel | Beskrivning | Allvarlighet (1-5) |
| ÅÄÖ i kod | Visas inte korrekt pågrund av skillnaden i datorernas språk inställningar | 2  I metod och klassnamn ställer det till besvär. Annars rent estetiska problem |
| double deklarerat med stort D | I deklarationen av variabeln double är den skriven med stor D | 1  Inte konsekvent med resten av koden. |
|  |  |  |

# Granskning – erfarenheter

Efter att ha granskat varandras kod insåg vi att det är lätt att missa fel eftersom att man ofta har väldigt stor tilltro till sig själv och sin egen kod. Vi upplevde även att det ibland kunde kännas svårt att rapportera ett fel som man har upptäckt eftersom att det kan bli dålig stämning i gruppen. Men med vetskapen i ryggen att det är viktigt att rapportera alla fel som man hittar även fast författaren av koden kan ta illa vid sig upplevde vi att det kändes lättare.

En viktig del av hur man rapporterar fel är hur man lägger fram det och vi märkte tidigt att det bästa sättet var att öppet säga och dokumentera ett fel utan att ifrågasätta vem som gjort felet eller varför.

När vi granskade efter att vi hade kört FindBug insåg vi att det är lätt att missa mindre uppenbara fel, som till exempel variabler som inte används eller metoder som anropar variabler som riskerar att inte ha blivit initierade.

Under granskning av någon annans kod är det lätt att man hakar upp sig på hur andra programmerare strukturerar sin kod eftersom det skiljer sig ifrån vad man själv tycker är korrekt.

# Kodkritiksystem

Som kodkritiksystem valde vi att använda oss av BugFind på hela systemet. Fel hittades bara i klassen Kvittorad och klassen Vara.  
  
**Fel som hittats före den formella granskningen:**

|  |  |  |
| --- | --- | --- |
| Påträffat fel | Beskrivning | Allvarlighet (1-5) |
| I klassen Vara: Use of non –localized String.toUpperCase() or String.to.LowerCase() | När vi inte angav någon språkkodning reagerade den eftersom att det finns en risk att den inte konverterar internationella tecknen rätt. | 3 |
| I klassen Kvittorad: Unused public or protected filed… | Variabeln totalpris används inte men eftersom att den är deklarerad i public misstänker den att den kan komma att användas utifrån. | 1  Egentligen ingen större fara att den är public eftersom den inte används i koden överhuvudtaget. |
| I klassen Kvittorad och i klassen Vara: Vara rabatt is not initialized | Variablen rabatt initieras inte i konstruktorn, men kan ändå anropas i en metod. | 2  Eftersom rabatt inte initieras men ändå kan anropas i metoden utan någon kontroll finns risken att man får null i returvärde. |

# Kodkritiksystem

**Fel som hittats med BugFind efter den formella granskningen och rättningen:**

När vi jämförde vad vi hittade under granskningen med resultatet från BugFind-körningen insåg vi att vi missade ett par fel som BugFind upptäckte. Detta skulle kunna vara ett tecken på att vi var dåliga på granska, men om man kollar på de fel som BugFind faktiskt upptäckte kan man konstatera att det var fel som är svåra för en granskare att upptäcka när koden gås igenom.

# Statiska mått

Vi valde att utföra statiska mått med hjälp av programmet Metrix 1.3.6 på hela systemet inklusive testklasserna.  
Resultatet kan ses i tabellen nedan

|  |  |  |
| --- | --- | --- |
| Statiskt mått | Resultat | Diskussion |
| Antal rader kod | 233st |  |
| Flest rader kod | Klassen kvitto (81st) |  |
| Antal klasser | 5st |  |
| McCabe Medel | 1,722st |  |
| McCabe Max i klassen Vara | 11st | Väldigt intressant att det fanns så många vägar i klassen Vara vilket visade på att klassen är komplexare än vad vi trodde. |
| Attribut totalt | 22st | Intressant att se att måtten stämmer överens med det klassdiagram som vi ritade upp i designstadiet av programmet. Detta visar på att vi har byggt det system som vi från början tänkt bygga. |
| Attribut i klassen Kvitto | 7st |
| Attribut i klassen Vara | 6st |
| Attribut i klassen Kvittorad | 5st |
| Attribut i klassen Rabatter | 2st |
| Attribut i klassen Valuta | 2st |
| Afferent Coupling | 20 | Antal klasser som är beroende av andra klasser i samma paket. Detta är intressant eftersom det är 5 klasser men hela 20 kopplingar vilket tyder på att dessa klasser är väldigt beroende av varandra för att fungera. |

# Täckningsgrad

Vi valde att använda oss av programmet EclEmma för att mäta vår täckningsgrad. Våra testfall uppnår 100% täckningsgrad för själva produktionskoden men inte på vår testkod. Detta beror på att testfallen förväntar sig ett fel och när de får ett fel i anropet avbryts testet. När systemet avbrutit ett testfall räknas det som att det inte är kört även fast det är det.

# Profiler

Vi använde oss av NetBeans för att testa koden med profiler. Vi använde den inbyggda standardprofilern och testade både hur mycket minne och processorkraft som gick åt vid körning. Därefter gjorde vi en analys av resultaten som vi fick fram.

Vid körningen av minnes-profilern kunde vi se att systemet främst arbetade med objekt av typen int och char. Att integers användes i stor utsträckning var ganska väntat, eftersom vi nästan uteslutande gör matematiska uträkningar. Chars används lite mer till följd av att vi i klassen Vara anropar metoden toLowerCase, för att försäkra att enheten alltid lagras i gemener även om användaren råkar mata in versaler.

Ur CPU-profilern fick vi nästan inte ut några värden alls eftersom projektet är så pass litet. Man skulle kanske kunna tvinga fram ett resultat som vara större om man skrev ett längre testfall som gjorde mer och om koden innehöll fler komplicerade och/eller minneskrävande beräkningar.

# Byggscript

Vi valde att använda oss av programmet ANT för att bygga vårt byggscript. Slutliga utformning:



# Övrigt

**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.