Laborationsrapport

D0018D, Objektorienterad programmering i Java

Laboration: Inlämningsuppgift 1

**Salim Daoud**

[**saldao-8@student.ltu.se**](mailto:saldao-8@student.ltu.se) **/** [**salim\_daoud@hotmail.com**](mailto:salim_daoud@hotmail.com)

2018-11-06

# Innehållsförteckning

Inledning 1

Genomförande 1

Systembeskrivning 1

Diskussion 2

Referenser 5

# Inledning

Uppgiften går ut på att skapa ett enkelt banksystem som hanterar kunder och deras sparkonton.

# Genomförande

För att börja programmera denna uppgift så har jag följt anvisningarna för att ladda ned Java via JDK (Java development Kit) och den integrerade utvecklingsmiljön Eclipse. Eclipse är väldigt kraftfullt som förenklar skapandet av paket och klasser, upprätthålla och bevara relationerna, felsökning m.m

Jag började mitt arbete med att skapa ett projekt för att sedan skapa min första klass och paket. Under utvecklingen så såg jag till att först skapa de variabler jag misstänkte att jag behövde, funderade på deras tillgängligheter utanför klassen, för att sedan skapa dess metoder. Under denna process såg jag till att skapa en temporär main klass för varje klass jag utvecklade där jag stegvis skapade objekt av klassen och testade dess publika gränssnitt och beteende. Detta för att slutligen när alla klasser och grundarbetet för hela systemet var färdigt testas med testklassen vi har blivit försedda med. När jag sedan arbetade med att finslipa och vidareutveckla systemet så såg jag till att ständigt testa systemet med testklassen för att säkerställa att jag inte har förstört ngt och att systemet fortfarande är intakt och fungerar som det ska.

Jag valde även att arbeta med versionshanteringsverktyget ”Git” och webbhotellet ”Github” för att hålla koll på mina ändringar och lagra min kod i ett säkert ställe. I och med detta så kan även felsökningen förenklas då man lättare kan hoppa tillbaks till en tidigare fungerande version och även se exakta modifieringar som har gjorts mellan två olika versioner.

Kommentering av klasser och metoder har gjorts med hjälp av Javadoc. När det gäller annan kommentering av koden och det som sker där så har jag försökt att praktisera filosofin ”clean code” och därmed koncentrera mig på koda så strukturerat och tydligt som möjligt så att koden blir självförklarande. Därmed kommenterar jag endast där det kan anses vara behövligt eller otyligt.

# Systembeskrivning

Systemet är uppbyggd av smarta objekt av tre olika definitionsklasser; en klass som representerar en kund, en klass som representerar ett sparkonto och en klass som är bankens logik och system. Tanken med skapandet av smarta objekt och instanser är att dessa enkelt modulariserar systemet, inkapslar data och strukturer, för att sedan erbjuda ett antal publika tjänster via sitt gränssnitt som andra delar av systemet kan aktivera för att styra dess utseende och beteende.

SavingsAccount klassen implementerar en definitionsklass för sparkonton. Denna klass hanterar all information och tjänster som är relaterade till ett sparkonto som saldo, räntesatser, utföra transaktioner etc.

Customer klassen implementerar en definitionsklass för kunder. Denna klass hanterar all information och tjänster som är relaterade till en kund som namn och konton som kunden äger.

BankLogic klassen implementerar en definitionsklass som hanterar bank systemets logik. Denna klass hanterar alla kunder och deras konton.

# Diskussion

SavingsAccount:

* Då det kändes naturligt att kontotypen för klassen är ett ”sparkonto” och att detta aldrig ska behöva ändras för denna klass så föll valet på att detta skulle representeras som en konstant. Faktumet att detta är gemensamt för alla instanser för denna klass ledde till att jag valde att deklarera denna variabel som final static. Nyckelordet ”final” för kontant och ”static” att detta är en klassvariabel snarare än en instansvariabel. (Galjic 2013, 526).
* Kontonumret för ett konto/instans skulle enligt anvisningar vara unika för hela bank systemet och därmed föll valet även här för att använda mig utav en klassvariabel via nyckelordet ”static”. I och med detta så kommer varje instans tilldelas ett unikt nummer då denna variabel akumuleras i klassen efter varje tilldelning. Därmed definieras även startvärdet i klassvariabeln i samband med att denna deklareras.
* I denna klass och i andra klasser valde jag att förkorta satser och kod där det var uppenbart och tydligt vad jag ville åstadkomma, men valde att ej göra detta där det skulle bli otydligt och möjligen komplicerat eller svårare att följa vad som sker i koden.
* I metoden ”withdraw” demonstreras fördelen med att kapsla in data och sedan endast kunna manipulera dessa via kontrollerade publika gränssnitt. Genom att använda denna metod och inte direkt variabeln ”balance” så kan definitionsklassen implementera logik för att kontrollera att saldot ej understiger 0. Om detta är fallet så kan man returnera att försöket misslyckades och bibehålla rätt tillåtna värden och tillstånd för klassens variabler. Detta är en fördel med objektorienterad programmering och leder till lättare och säkrare system. (Galjic 2013, 517).
* Alla variabler är inkapslade och endast – om möjligt – accessbara via publika metoder.
* Presentationsdata presenteras via tjänsten ”toString”. Genom att överlagra denna metod som ärvs av Javas förälderklass ”Object” så kan t.ex. andra metoder i Java som ”println” skriva ut vald information om klassen via denna metod. I detta bank system kan andra klasser använda denna information i andra metoder för att sedan presentera värdefull information till användare. (Galjic 2013, 508).
* Denna klass - likt övriga klasser – kommenteras via Javadocs standard. I och med detta blir det enkelt för en utomstående – och för en själv när man senare ska titta tillbaks på koden för rättning, utveckling och annan underhållning – att sätta sig in i vad syftet med olika metoder är, förklaring på dess parameterlistor och vad metoden är tänkt att returnera (för senare använding). En annan fiffig sak med detta är att man kan generera dokumentationen i HTML format. (Tutorialspoint, u.å.)

Customer:

* Jag skapade en variabel för denna klass i form av en lista som hanterar alla konton/objekt som en kund äger. Jag fick då en tankeställare då jag skulle bestämma mig för var jag skulle definiera och initiera denna variabel av typen ArrayList<SavingsAccount>. Mina val låg i att antingen göra detta i samband med deklareringen av denna variabel eller i konstruktorn som vanligtvis brukar hantera initieringen av variabler för en klass. Efter lite sökande på nätet och tips av Hughes (”Best practice for initializing an ArrayList field in Java”, 2016) så föll valet på att göra detta i samband med deklareringen av denna variabel. Jag tyckte att detta skulle vara säkrare då jag säkerställer att jag inte missar att senare initiera denna som i sin tur skulle kunna resultera i en referens som har värdet null och som skulle krasha systemet/programmet vid körning när denna referens senare skulle användas och då kasta en ”NullPointerException”. Observera att listan ”accounts” i själva verket är en referens som pekar på en lista med referenser av typen ”SavingsAccount” som var och en pekar på objekt av typen ”SavingsAccount”.
* Jag funderade även på om personnumret skulle vara av typen String eller int/long. Upptäckte ganska tidigt att max värdet för int inte skulle kunna representera ett tolvsiffrigt personnummer utan att valet snarare var mellan String och long. I och med att användningen av personnumret i programmet endast skulle vara i sammanhang av presentation så kändes det naturligt att representera detta med typen String. Därmed slipper jag även att typomvandla personnumret varje gång jag behöver presentera informationen en sträng.
* Java tillhandahåller och förser instanser/objekt med en referens till sig själv dvs till instansen/objektet i fråga som används. Denna mekanism är väldigt användbar då jag i klassens konstruktor och andra metoder tydligt kan klargöra att det är instansvariabeln som ska tilldelas värdet av metodens argument. Detta speciellt i fallet då instansvariabler och metoders argument delar samma namn. Anledningen till att jag väljer att ha samma namn är att namnen på variablerna/argumenten är självförklarande och tydliga då de representerar samma sak och att jag därmed inte vill lägga ned tid på att hitta på olika namn för samma representationer. (Galjic 2013, 500).
* I denna klass - likt övriga klasser - kan vi se exempel på s.k. set och get metoder som är ett vanligt sätt att manipulera och erhålla variabel värden i objektorienterad programmering. (Galjic 2013, 575).
* I metoden ”createSavingsAccount” kan vi se exempel på att vi skapar en referens till ett SavingsAccount objekt och samtidigt skapar ett objekt av typen Savingsaccount via operatorn ”new”. I och med att operatorn ”new” endast skapar ett namnlöst objekt så skapar vi en referens till denne som vi sedan använder i programmet för att utföra olika aktioner. (Galjic 2013, 308).
* I metoden ”deleteAccounts” kan vi se exempel på:
  1. Användningen av for-loopar för listor med objekt. Jag valde denna typ av for- loop då denna dels var renare och tydligare i mitt fall och dels för att jag inte inte direkt behövde indexerna i fallet av for-loopar med index.
  2. Metoden clear() som tar bort alla element/referenser i listan vilket leder till att inga referenser existerar för objekten av typen ”SavingsAccount” vilket gör att dessa blir föremål för Javas skräpinsamlare.
* Denna klass - likt övriga klasser – implementerar även kontroller i metoder som kontrollerar att man använder referenser som verkligen pekar på ett objekt och inte är en null pekare. Detta för att hindra att systemet/programmet kraschar vid körning.
* I denna klass valde jag att implementera en hjälpklass ”findAccount” för att hitta och returnera det aktuella kontot som systemet vill anropa och utföra olika operationer på. Detta för att undvika att duplicera samma kod i flera ställen i klassen så att det blir mindre kod, enklare och renare implementering och lättare logik att följa. I och med att denna metod inte behöver användas utanför denna klass och därmed endast ska förenkla implementeringen av denna definitionsklass så fick denna metod deklareras som ”private”.
* Även i denna klass valde jag att presentera data via tjänsten ”toString”. Funderade på om denna klass endast skulle returnera information om själva kunden (namn och personnummer) eller om den även skulle returnera uppgifter för kundens alla konton. I och med att endast en sträng ska returneras och att man i programmet senare vill skapa en lista av strängar där kundens namn och personnummer ska vara ett element och sedan information om varje konto skall vara element för sig själva, så valde jag att endast presentera och returnera information om kundens namn och personnummer. På detta vis så slipper man senare försöka inspektera en väldigt lång sträng av tokens för att extrahera tokens som ska tillhöra olika konton och element. Programmet kommer då – enligt min känsla – ha en enklare logik att följa.

BankLogic:

* I och med att den enda variabeln denna klass har initieras vid deklareringen (av tidigare nämnda anledningar) så behöver jag egentligen inte definiera en konstruktor. Jag skulle lika gärna ha utelämnat denna och låtit kompilatorn skapa en förvald konstruktor. Jag valde ändå skapa en tom konstruktor av två anledningar:
  1. För tydlighetens skull; att visa att detta verkligen är meningen och avsikten.
  2. Om jag senare skulle välja att skapa specifik konstruktor med argument och dylikt så kommer kompilatorn inte att skapa en förvald konstruktor och jag skulle då explicit behöva skapa en förvald konstruktor utan argument. (Galjic 2013, 503).
* Skapade även här en hjälpklass ”findCustomer” av samma anledningar som tidigare nämnt för hjälpklassen ”findAccount” i klassen ”Customer”.
* Den enda funderingen jag har för att förbättra programmet är att man i anvisningarna nämner: ”Du kommer troligtvis att skapa ett antal hjälpmetoder, privata metoder”. Jag kom ”bara” fram till att jag behövde en hjälpmetod. Mina tankar florerade annars i om jag kunde fytta ned en del logik för uppsamling av data till klassen ”Customer” och erhålla denna information via metoder i den klassen. T.ex. att metoden ”getCustomer” endast kallar en metod i klassen ”Customer” som sedan bygger upp information och returnerar listan av strängar. Men detta kanske är mer av en smaksak.

# Referenser

Galjic, Fadil. 2013. *Programmeringsprinciper i Java*. : Studentlitteratur AB.

Tutorialspoint (u.å). *”Java - Documentation Comments”*. [Elektronisk]. Tillgänglig: <https://www.tutorialspoint.com/java/java_documentation.htm> [2018-11-06].

Hughes, Nathan. 2016. *Best practice for initializing an ArrayList field in Java*. [Elektronisk]. Tillgänglig: <https://stackoverflow.com/a/38125288> [2018-11-06].