Laborationsrapport

D0018D, Objektorienterad programmering i Java

Laboration: Inlämningsuppgift 4

Salim Daoud

saldao-8@student.ltu.se / salim_daoud@hotmail.com

2019-01-29

Innehållsförteckning

Inledning

Uppgiften går ut på att vidareutveckla bankkontosystemet från inlämningsuppgift 3.
Bankkontosystemet ska nu stödja filhantering och lämplig felhantering skall implementeras.

Genomförande

Jag fortsatte arbeta i den integrerade utvecklingsmiljön Eclipse. Eclipse är väldigt kraftfullt och förenklar skapandet av paket och klasser, upprätthåller och bevara relationer, felsöker m.m. Eclipse har också fiffiga funktioner som att automatiskt ge förslag på möjliga metoder och för importerande av nödvändiga paket. Dessa funktioner var nyttiga nu när man skulle börja använda massor med nya paket och klasser relaterade till filhanteringen.

Jag började mitt arbete med att skapa ett nytt projekt för att sedan kopiera över klasserna från inlämningsuppgift 3 till detta nya projekt. Under utvecklingen så såg jag till att först implementera filhanteringen i metoder i Banklogic för att testa om dessa fungerade för att sedan skapa en egen klass för detta (StorageHandler) för en mer objekorienterad och strukturerad lösning.

Testning av systemet har gjorts kontinuerligt. Varje gång en funktion för filhantering implementerats så har programmets körts för att verifiera att man får rätt beteende och att rätt filer skapas.

För felhanteringen så började jag med att testa programmet med troliga fel en användare kan tänkas göra för att se hur programmet beteer sig, vilka felmeddelanden som uppstår och därmed analysera vilka åtgärder som krävs. Strategin var att först försöka förhindra att programmet hamnar i ett undantagstillstånd snarare än att i första hand försöka hantera undantag. I och med denna strategi ämnade jag att få en lättare, säkrare och snabbare kod/program.

Jag fortsatte även att arbeta med versionshanteringsverktyget ”Git” och webbhotellet ”Github” för att hålla koll på mina ändringar och för att lagra min kod i ett säkert ställe.

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 koncentrerat mig på att koda så strukturerat och tydligt som möjligt så att koden snarare blir självförklarande. Därmed kommenterar jag endast där det kan anses vara behövligt eller där det är otyligt.

Systembeskrivning

Systemet är utvidgat med en ny definitionsklass:

StorageHandler klassen implementerar en klass som hanterar sparandet och uppladdning av data till och från filer. Med hjälp av denna klass kan systemet spara undan information om alla kunder, deras konton och deras transaktioner för att i ett senare skede ladda upp och presentera denna information för användaren. Klassen hanterar även alla undantagssituationer som kan uppstå vid hantering av filer. Klassen har skrivits på ett sådant sätt som tillåter den iframtiden att refaktoriseras för att t.ex. byta till databashantering utan att gränssnittet eller övriga klassers anrop behöver ändras.

Diskussion

StorageHandler

- Valet av namn var för att ge systemet flexibiliteten att gömma vilken metod och resurser som användes för att spara och ladda data. I och med detta namn kan man sedan inom denna klass välja om man skall använda fil- eller databashantering. Enligt ovan nämnt.
- Jag hade först en idé att skapa denna klass som en statisk klass för att sedan – utan att behöva skapa och skicka referenser av klassen runt om i programmet – kunna anropa dess metoder överallt i koden där denna behövdes. Jag ansåg att systemet ej behövde objekt av denna klass och att klassen i sig inte hanterade några tillstånd utan att spara och ladda metoderna endast kunde tjäna som – jag kallar det – ”direkta och raka” anrop. Jag valde trots detta – utan någon större anledning förutom osäkerhet - att skapa ett objekt av denna klass i BankLogic och att det endast var BankLogic som kommunicerade med StorageHandler. I och med detta så skickas inga referenser runt i programmet till denna klass och det blir renare och tydligare ansvarsområden.
- Egentligen är denna klass endast ämnat för filhantering och inte att hantera något som är relaterat till GUI:t, trots detta så skriver denna klass ut felmeddelanden till användaren via dialogrutor. Anledningen till detta var att jag ville ge användaren mer specifika felmeddelanden när dessa undantag skedde. Jag övervägde att endast hantera undantagen i den anropande metoden i TransactionDialog klassen eller att passera vidare undantag (samma eller egna definierade) till den anropande metoden och klassen. Jag ville dock urskilja mellan fel som uppstod pga läsning/skrivning och fel relaterade till stängning och ville samtidigt lokalisera problemet nära dess uppkomst så jag slipper propagera och deklarera/fånga undantag i flera ställen/klasser i koden. Därmed så hoppades jag på att få en enklare och snabbare kod (obs. IO undantagen kunde dock inte undvikas). I och med detta hanterade jag endast undantagssituationerna i denna klass vilket ledde till att jag använde mig utav dessa

dialogrutor för mer detaljerad information om felen i denna klass. I efterhand tror jag att jag skulle ha skapat egna klasser för undantag med rätt felmeddelanden och propagerat vidare dessa för att sedan låtaTransactionDialog klassen presentera denna information med hjälp av dialogrutor. (Galjic, ”Programmeringsprinciper i Java”, 2013, 423 & 426)

- Jag funderade även på hur jag skulle strukturera min try-catch metod med tanke på att jag ville vara säker på att jag alltid stängde strömmarna oavsett om filhanteringen gick bra eller ej. Detta för att stänga förbindelsen på ett korrekt vis och att tvinga all data som skall skrivas till filen att nå sin destination (”flush”-mekanismen). I och med detta placerade jag stängningen i ett finally-block. Då kommer stängingen att exekveras oavsett resultat av try-catch blocken. För att i sin tur vara säker på att stängningen gått rätt till så placerade jag även denna i en try-catch block inom finally-blocket. Jag tyckte dock inte – speciellt vid läsning – att en misslyckad stängning skulle vara så allvarlig så att man alltid behövde skriva ett felmeddelande till användaren. I efterhand så skulle jag nog ha använt en annan stängingsmodell sk. ”try-with-resources” i och med att mina använda klasser alla implementerar gränssnittet ”AutoClosable”. I detta fall anropas metoden close automatiskt och koden blir mindre och tydligare. (Galjic, ”Programmeringsprinciper i Java”, 2013, 466-468)
- En annan liten utmaning jag stötte på var att skapa nya rader när jag skrev till en fil med hjälp av klassen FileWriter. Först lade jag endast till ”\n” efter de strängar jag ville skriva ned till filen med förhoppningen att detta skulle skapa nya rader, men detta hjälpte inte. Jag försökte då efter tips från nätet att skapa nya rader med hjälp av operativsystemets tecken för detta vilket visade sig vara framgångsrikt. (https://stackoverflow.com/a/18549788)

TransactionHistoryDialog

- För att ge användaren återkoppling över hur sparandet av transaktioner har gått så valde jag att deaktivera spara-knappen när denna har utförts framgångsrikt. Detta då det är viktigt med ett användarvänligt program som ständigt via olika funktioner ger tydlig återkoppling till användaren.

OverviewWin:

- För att förhindra att en användare råkar ladda upp samma kunder två gånger från systemet och dämed skapa massor med kopior i listorna så deaktiverades meny alternativet för laddning efter en uppladdning. Detta även för att slippa ha logik för att jämföra om kunden redan har återskapats eller ej. Jag funderade även på hur systemet egentligen är tänkt och specificerat när det gäller uppladdningar av kunder när kunder redan har skapats. Det enda kravet som nämndes i uppgiften var att nya kunder som har skapats efter uppladdning av kunder ska följa den ordningsföljd som anges av filen och inget mer. Därmed sparade jag helt enkelt undan denna räknare till filen som en int och läste sedan denna vid uppladdning och satte värdet av räknaren till systemet. Jag introducerade därmed metoder för att stödja sättandet av räknarens värde till systemet

(Account). Om jag hade fått välja så skulle jag ha implementerat denna funktion så att programmet automatiskt sparade och laddade upp information om systemets kunder vid öppning och stängning av programmet. Ser ingen större nytta med att en användare ska manuellt behöva ladda upp denna information då det är underförstått att denne vill börja där systemet avslutades och att det är ett och samma samma banksystem som avses. Men i och med att inlämningsuppgift 3 och 4 indikerar på att man ska ha denna valmöjlighet så implementerades detta.

- Inmatning från textfälten för namn och personummer hanteras i systemet endast som strängar. Därmed så kan programmet inte fallera på inmatning av tecken som ej normalt ingår i ett namn eller personnummer. T.ex. namnet ”Sa1lim%” med personnumret ”1981elva13$” var helt accepterat och kunde ej få programmet att krascha. Dock tyckte jag att detta var fult och därmed ville vägleda användaren till en mer korrekt och förståelig inmatning som i sin tur ger användaren en fin och professionell upplevelse av programmet. Pga detta så implementerade jag – med hjälp av tips från nätet – kontroller som såg till att inmatning av namn endast innehåller bokstäver och att inmatning av personnummer endast innehåller siffror och följer ett bestämt format (som inkluderar ett bindestreck). För att förhindra oförklarliga felmeddelanden eller andra underliga beteenden så säkerställde jag även att alla ”tomrum” - som en användare kan ha råkat lägga till - filtrerades bort från strängarna.(https://stackoverflow.com/a/3059373, https://stackoverflow.com/a/40097058 & https://stackoverflow.com/a/16035109)
- Det är även värt att nämna att programmet tidigare har implementerat andra smarta funktioner för att förhindra att oväntade fel och krascher ska uppstå. T.ex. som att man endast kan markera en kund och konto i taget, att man mer eller mindre alltid måste ha markerat en kund och ett konto för att få utföra en funktion m.m.

TransactionDialog

- I denna klass och dialogfönster kan användaren få för sig eller råka skriva in andra tecken än siffror för önskade transaktioner. Detta ledde i sin tur till en krasch pga en sk. NumberFormatException. I och med detta så implementerade jag en try-catch struktur som skulle fånga upp denna felaktiga inmatning och sedan meddela användaren om att endast ange transaktionen med siffror.