Laborationsrapport

D0018D, Objektorienterad programmering i Java

Laboration: Inlämningsuppgift 3

Salim Daoud

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

2018-01-24

Innehållsförteckning

Inledning

Uppgiften går ut på att skapa ett grafiskt användargränssnitt (GUI) som använder min tidigare logikklass med all funktionalitet implementerad i samband med inlämningsuppgift 2.

Genomförande

Jag fortsatte arbeta i den integrerade utvecklingsmiljön Eclipse. Eclipse är väldigt kraftfullt som förenklar skapandet av paket och klasser, upprätthålla och bevara relationerna, felsökning 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 man i vanliga fall inte är så bekant med.

Jag började mitt arbete med att skapa ett nytt projekt för att sedan kopiera över klasserna från inlämningsuppgift 2 till detta nya projekt. Under utvecklingen så såg jag till att först ha en klar bild på hur jag vill att GUI:t skulle vara uppbyggt och strukturerat för att sedan labba med olika layout managers och förstå hur de fungerar för att enklare använda mig utav dessa. Jag satt även med papper och penna för att rita upp rutsystem som kunde vara till hjälp vid användandet av dessa layout managers. Sedan utvecklades dessa GUI och kopplades till respektive logik iterativt.

Testning av systemet har gjorts kontinuerligt. Varje gång GUI:t utökats med ett block komponenter tillhörandes en funktion så har programmets körts och testats för detta ändamål. Genom att kontinuerligt testa programmet ger detta mig en möjlighet till att snabbare och enklare upptäcka och korrigera eventuella fel och brister.

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. Även angivit källor till de sidor jag har fått inspiration och tips ifrån.

Systembeskrivning

Systemet är utvidgat med fem nya definitionsklasser – alla bygger upp och hanterar GUI:t. Alla dessa klasser ärver från antingen JFrame eller JDialog. De bygger sedan upp sina egna utseenden. För att huvudfönstret inte skulle bli alltför grötig och därmed försvåra för användaren att använda programmet så skapades dialog klasser för att sköta majoriteten av inmatning av information från användaren.

MainWindow klassen implementerar en definitionsklass för det grafiska gränssnittet som fungerar som en snabbåtkomst till övriga funktioner och fönster som användaren kan tänka sig önska. Denna klass innehåller även main metoden.

OverviewLogicWin klassen implementerar en definitionsklass för det grafiska gränssnittet som representerar översikten av systemet och är även den klass som sköter kommunikationen med bankens logik klass. Därmed är denna klass huvud GUI:t för hela programmet. Denna klass använder sig som sagt av klassen BankLogic för att realisera användarens önskemål och för programmets logik. Huvudsyftet med denna klass och GUI är att göra det enkelt för användaren att få en överblick av systemet, att använda programmet och att snabbt och enkelt utföra sina administrativa uppgifter.

AddAccountDialog klassen implementerar ett dialogfönster (grafiskt gränssnitt) för att skapa ett konto. Fönstret tillåter användaren att välja en kontotyp via en drop-down lista.

TransactionDialog implementerar ett dialogfönster för att utföra en transaktion. Klassen är generisk i den mening att den består av samma grafiska komponenter oavsett om det handlar om en insättning eller uttag och tar därmed emot ett argument som specificerar typen av transaktion.

TransactionDialog implementerar ett dialogfönster för att presentera alla transaktioner som har utförts för ett specifikt konto.

Diskussion

MainWindow

- Mitt mål var att försöka mig på flera olika layout managers för att få en bredare förståelse av dessa verktygs styrkor och möjligheter. (Jag ville självklart också satsa på ett högre betyg). I detta GUI ville jag ha en design där relaterade funktioner är grupperade ihop i samma rad och samtidigt att knapparna för dessa funktioner skulle vara centrerade i respektive rad. Jag valde att realisera detta genom att skapa paneler för varje rad med FlowLayout som layout manager och att sedan stapla dessa på varandra genom att placera alla dessa i en panel med BoxLayout som layout manager och ange att dessa ska placeras vertikalt dvs uppifrån och ned.

- Detta och alla andra fönster/GUI:n använder sedan i grunden layout managern BorderLayout för att placera menyn högst upp, hela det specifika huvudinnehållet för detta fönster centralt och i vissa fall knappar längst ned i fönstret. I och med att man endast kan ha en komponent synlig centralt i fönstret (JFrame) via denna layout manager så placeras en panel i denna sektion som fungerar som en behållare av alla andra önskade komponenter som ska bygga upp det specifika huvudinnehållet för detta fönster. (Fahlman, 2014)
- En annan utmaning eller fundering jag mötte var hur man bygger upp ett GUI med önskade tomma utrymmen mellan dess komponenter för att få till en fin och användarvänlig design. Jag fann att man kunde utnyttja olika metoder för detta (jag har använt mig av de första tre metoderna):
    - Användningen av ”tomma ramar” där man kan ange hur mkt tomt utrymme man vill ha runt en komponent i alla dimensioner. Denna metod fann jag vara mkt kraftfull och användbar. (Oracle Java Documentation, u.å)
    - Layout managers som FlowLayout har stöd för detta (som t.ex. dess konstruktor).
    - Lägga in osynliga komponenter. (Fahlman, 2014)
    - I layout managers som BoxLayout kan man skapa och ange storlek av ett utrymme (rigid area) eller ange var ett överflödigt utrymme ska hamna. (Fahlman, 2014)
- En annan metod jag använde mig av för att förhindra att designen för mina fönster skulle se dålig ut var att förhindra (via metoden resizable(false)) att användaren själv kan dra i fönstret för att ändra stolek på denna. I och med att jag ej har tagit hänsyn till sk. resposiv design så leder detta annars till en ful design och intryck av användaren.
- Jag har använt mig av inre klasser för lyssnarklasserna och detta främst för att jag fann detta sätt att vara mer strukturerat och enklare att följa koden då man grupperar lyssnare ämnade för vissa relaterade händelser, funktioner eller sektioner av GUI:t tillsammans. Detta var ett tips jag fann i en av Fahlmans föreläsningar (”Grafiska användargränssnitt”, 2014). Dock var en nackdel med detta att man ej kan utnyttja ”this” referensen för själva fönstret vid behov. Jag fick gå runt detta med att skapa en medlemsvariabel för fönstrets klass och att sätta denna referens i konstruktorn till ”this” referensen. På detta sätt kan man sedan få tillgång till denna medlemsvariabel i den inre klassen.
- I och med det ”höga” antalet knappar och att samma aktioner skulle göras på dessa knappar (sätta samma storlek för finare design och koppla dessa till lyssnare) så lades knapparna in i en lista. Listan kunde sedan skickas som argument till metoderna som utförde de önskade aktionerna. Syftet var att få mindre och mer strukturerad kod.
- Jag ville även att användaren skulle ha möjlighet att avsluta programmet via menyn och funderade på hur man på ett kontrollerat sätt utförde detta. Jag fann att man i flera ställen på nätet rekommenderade likt Midany (”Java Exit Button from a MenuItem”, 2013) att anropa metoden System.exit(0).

OverviewWin

- Jag funderade på hur jag på bäst sätt kunde presentera och påverka systemets översikt i ett och samma fönster. Jag hade en del idéer och fann sedan inspiration från Dodds

(”Simple Bank in Java”, 2011). Jag tyckte att denna struktur var klockren och en utmaning att försöka efterlikna och programmera. Denna design gav mig även möjligheten att variera mina komponenter och använda mig utav JList som var en möjlighet för högre betyg.

- För detta GUI använde jag mig av layout managern GridLayout för att skapa ett rutsystem på 2x2 där varje ruta är lika stor. I varje ruta som representerar en sektion placeras en JPanel som sedan ska innehålla komponenter.
    - För de övre två rutorna skall endast listorna (JList) vara komponenten. I och med att en JPanel använder per default FlowLayout så fyllde listan inte hela denna sektion vilket jag önskade. Jag fann då tipset av dic19 (”How to make a JList filling JPanel”, 2013) att man för sin panel kan använda layout managern BorderLayout för att en komponent ska fylla hela dess behållare.
    - För de två nedre sektionerna/rutorna för ”Customer”- och ”Account Information” har jag använt mig av GridLayout för att bygga upp respektive rutsystem och sedan placerat dess komponenter (text fält, etiketter och knappar). Jag fann detta kraftfullt då man kan få till ett fint symmetriskt utseende där alla komponenter är lika stora och korrekt placerade. Jag använde här tekniken att placera tomma komponenter – i mitt fall - tomma etiketter i de rutor där innehåll skulle saknas. En annan intressant detalj var att jag ansåg att ett kreditkonto bör presentera sin kreditgräns men att komponenterna för dessa inte behövs vid fallet då ett sparkonto presenteras. I och med detta ville jag endast visualisera dessa två komponenter när ett kreditkonto presenteras och gömma dessa då ett sparkonto presenteras. Jag löste detta genom att alltid ha dessa två komponenter placerade i GUI:t men att gömma och visa dessa visuellt beroende på vad det är för konto som ska presenteras (via setVisible metoden för en komponent).
- Jag upptäckte även att grundkoden för att bygga upp utseendet för fönstrets menyer mer eller mindre var samma som den kod som användes i välkomstfönstret. Idag skulle jag möjligen ha valt att ha ett och samma fönster för de båda men att bygga upp två separata paneler med respektive innehåll och sedan växla mellan dessa beroende på användarens val (likt exemplet ”CarRegister Demo” i föreläsningen ”Uppbyggnad av ett GUI”). Med denna lösning skulle jag behålla gemensamma komponenter som menyerna men å andra sidan möjligen öppna upp för mycket mer kod i en och samma klass. Funderade även på om man skulle kunna placera koden för menyerna separat och sedan importera/inkludera denna på ett smart sätt i båda klasserna. Hade jag haft mer tid skulle jag lagt ned denna på att forska på vad bästa ”common practise” är för detta.
- En annan förbättring skulle vara att placera komponenterna för listorna i en JScrollPane komponent. Därmed förstörs eller förändras inte GUI:t beroende på hur många element som listorna innehåller. Jag gjorde inte detta då man i denna övning troligen inte behöver lägga till så pass många kunder/konton i listorna. Dock implementerades och praktiserades detta i dialogrutan som presenterar kontons transaktioner.
- Jag fann att det blev en hel del medlemsvariabler för dessa GUI baserade klasser. Jag försökte sedan banta ned till de som endast behövde vara medlemsvariabler då de användes i olika metoder och/eller inre klasser. Detta ledde även till att jag valde att skapa knapparna i respektive metod och sedan i dess lyssnarklass extrahera

knapparnas texter för att urskilja vilken knapp som har tryckts. Detta snarare än att deklarera knapparna som medlemsvariabler. (Fahlman, 2014)

- För att minska risken för buggar och faktumet att systemet är uppbyggt på att man endast skall välja en kund och sedan presentera dennes konton så behövde jag begränsa möjligheten för användaren att välja fler kunder än ett i taget. Fann i samband med detta ett tips från java2s.com (”Setting the Selection Mode of a JList Component”, u.å.) på att man kan ställa in detta via metoden setSelectionMode. En annan funktion jag implementerade för att öka användarvänligheten var att text fälten automatiskt rensades då man t.ex. lägger till kunder så att användaren snabbt och enkelt kan utföra nästa önskade aktion. Interaktionen förbättrades även avsevärt genom användningen av Java Swings JOptionPane dialogrutor. Dessa var mycket enkla att använda och ger ett otroligt värdefullt informationsutbyte mellan programmet och användaren.
- En annan utmaning och problem jag stötte på var hur jag uppdaterade [JList] listorna med de [ArrayList] listor som returnerades då jag hämtade alla kunder och konton. Speciellt när [JList] listorna redan hade skapats. Jag fann att man via metoden setListData kunde uppdatera [JList] listorna i efterhand men denna metod kunde ej ta en ArrayList som argument. Jag fann senare ett tips från O’Dea (”Java ArrayList into JList”, 2010) där han nämner att man via toArray metoden kan ”packa upp” [ArrayList] listan och via detta returnera en lista av typen Objects vilket setListData-metoden tenderade att acceptera.
- I och med att jag använder mig av flera dialogrutor för att bl.a. samla input från användaren och att dessa klasser sedan behöver uppdatera bank systemets logik så började jag att fundera över hur detta skulle gå till. Då jag inte har använt mig utav någon MVC arkitektur så övervägde jag alternativen att antingen skicka med en referens till denna OverviewWin klass eller att skicka med en referens till BankLogic klassen då dessa dialogrutor skapades och anropades. Jag ville dock inte skicka med en referens till själva bank systemets logik då detta är känslig data och data jag inte vill utsätta för någon manipulation. Jag ville som sagt hålla denna information och logik så säker och felfri som möjlig. För att kontrollera detta valde jag att skapa publika metoder i denna OverviewWin klass som dessa övriga dialog klasser kan anropa för att påverka och ändra systemet och dess information på ett kontrollerat och säkert sätt. Förövrigt så skickades redan denna OverviewWin referens i samband med att dialogrutorna skapades då dessa behöver veta vilket förälderfönster de tillhör, ska centreras runt och blockera (dvs modaliteten).
- För bättre läslighet så har även privata metoder skapats och används för att återanvända och upprätthålla en strukturerad kod.

AddAccountDialog:

- Som tidigare nämnt ville jag använda mig utav dialogrutor av diverse anledningar. I och med att jag ville ha en egen design på dessa så lät jag mina egna klasser ärva av JDialog för att sedan specialisera dessa enligt mina egna önskemål. Jag testade faktiskt först att skapa dessa sk. pop-up fönster baserade på JFrame men fann en större utmaning främst när jag ville ställa in en modalitet som möjliggjorde att det första fönstret skulle blockeras. Det slutade därmed med att skapa dessa fönster baserade

på JDialog och det förenklade som sagt övriga funktioner som att blockera förälderfönstret och centrera dialogrutan.
- För att förenkla valet av kontotyp och för möjligheten att använda varierande komponenter valde jag att använda JComboBox. Även denna komponent gör att användaren snabbare och smidigare kan interagera med programmet och bli presenterad till de valmöjligheter som finns.
- Jag funderade senare på hur man på bästa vis stängde ned fönstret om man av någon anledning inte ville fullfölja sin aktion. Fick tips av Aslam (”Button for closing a JDialog”, 2011) att anropa metoden dispose som även frigör alla resurser kopplade till fönstret.

TransactionDialog

- Jag upptäckte tidigt att detta fönster kan återanvändas för både insättningar och uttag. Jag ville därmed skapa en mer generisk klass där programmet själv kunde ange och avgöra vad det var för transaktion som skall ske och därmed påverka dess smått varierande utseende. I och med detta skapades de publika konstanterna DEPOSIT och WITHDRAW i OverviewWin klassen och som sedan skickades med som argument till dennes konstruktor. Detta medförde även en mer läsvänlig kod. Beroende på vilken transaktion som angetts så ändrades titeln för fönstret och sedan anropades respektive metod för att utföra transaktionen.

Ändringar av tidigare skapta klasser:

- BankLogic:
    - Metoden getAccountIds lades till för att förenkla funktionen i OverviewWin som uppdaterade konto listan med kundens existerande konton.
    - I den privata metoden findCustomer så jämfördes tidigare kunders personnummer via operatorn ”==”. I och med att denna operator i fallet av två olika referenser till två olika objekt faktiskt jämför referensernas adresser och ej deras innehåll så gav det mig oväntade resultat vid körning. I och med detta och efter tips av Panjak (”Java String Compare”, u.å.) så ändrades detta till att använda metoden equals för att jämföra innehållet.
    - Då information av stända konton nu presenterades av en dialogruta så tog jag bort skärmuskriften.
- Customer:
    - I och med att jag ville presentera kreditgränsen i mitt GUI så såg jag till att lägga till denna information i strängen som returneras av metoden getAccount.
- CreditAccount:
    - Lade till metoden getCreditLimit av ovan nämnda anledning.
    - Namnet på kontona (även för klassen SavingsAccount) ändrades från svenska till engelska t.ex kreditkonto till Credit account då programmet parsar denna information och skriver ut denna i GUI:t som är på engelska.

Referenser

Fahlman, Susanne. 2014. *”Grafiska användargränssnitt”*. [Elektronisk]. Tillgänglig: https://ltu.instructure.com/courses/5339/files/807085/download [2019-01-28]

Oracle Java Documentation (u.å). *”The Java™ Tutorials - How to Use Borders”*. [Elektronisk]. Tillgänglig: https://docs.oracle.com/javase/tutorial/uiswing/components/border.html [2019-01-28].

A.Midany. 2013. *“Java Exit Button from a MenuItem”*. [Elektronisk]. Tillgänglig: https://stackoverflow.com/a/19764841 [2019-01-28].

Dodds, Kent C. 2011. “*Simple Bank in Java”.* [Elektronisk]. Tillgänglig: https://www.google.se/search?q=java+gui+for+simple+bank+system&rlz=1C1GCEU_svSE820SE821&source=lnms&tbm=isch&sa=X&ved=0ahUKEwj0pcTDqJbfAhWFCCwKHU3UC_kQ_AUIDigB&biw=1187&bih=618#imgrc=pcaBZZeDE5ShtM: [2019-01-28]

Dic19, 2013. *”How to make a JList filling JPanel”*. [Elektronisk]. Tillgänglig: https://stackoverflow.com/a/20359885 [2019-01-28]

java2s.com. u.å. *“Setting the Selection Mode of a JList Component”*. [Elektronisk]. Tillgänglig: http://www.java2s.com/Tutorial/Java/0240__Swing/SettingtheSelectionModeofaJListComponent.htm [2019-01-28].

O’Dea, Alain. 2010. *“Java ArrayList into JList”*. [Elektronisk]. Tillgänglig: https://stackoverflow.com/a/3269560 [2019-01-28].

Aslam, Mohammed. 2011. *“Button for closing a JDialog”*. [Elektronisk]. Tillgänglig: https://stackoverflow.com/a/6970105 [2019-01-28].

Panjak. u.å. *“Java String Compare”*. [Elektronisk]. Tillgänglig: https://www.journaldev.com/18009/java-string-compare#java-string-comparison-using-operator [2019-01-28].