«Organization»
Skellefteå

Laborationsrapport

D0018D, Objektorienterad programmering i Java

Laboration 1

Iana Kalinichenko

iankal-7@student.ltu.se

2018-03-08

Innehållsförteckning

Inledning 1
Genomförande 1
Systembeskrivning 2
Diskussion 3
Referenser 3

Inledning

I den första inlämningsuppgiften skulle man skapat en bas för hantering av bankens kunder – klassen BankLogic.java som byggs på klasser Customer.java (vilken hanterar allt som berör någon specifik kund inkluderande listan av alla sparkonton som kunden har) och SavingsAccount.java (vilken hanterar allt som berör något specifikt konto).

Genomförande

Jag har genomfört arbetet genom att först skapa SavingsAccount.java, efter det – Customer.java och sist – BankLogic.java. Jag följde tillgängliga kravspecifikationer och skapade först det som framgick från uppgiftens instruktioner.

I varje klass följde jag samma rutin: 1) skapa instansvariabler; 2) skapa klassvariabler vid behöv; 3) bestämma vilka variabler skulle vara konstanter 4) strukturera listor av alla variabler 5) skapa konstruktörer 6) skapa setters 7) skapa getters 8) skapa övriga metoder som nämn i instruktionerna.

När jag inte kunde skriva färdig någon metod på grund av att det saknades nödvändiga metoder i en annan klass, så skrev jag färdig den ursprungliga metoden färdig som att det redan fanns de där metoderna som jag saknade. Men när metoden var färdig så återvände jag till den där klassen där metoderna saknades. På så sätt kunde jag enkelt se vilket värde behövde returneras och vad skulle skickas som parametrar. Utifrån det började jag skriva den metoden som saknades.

Jag skrev allt i Oxygen.1a Release (4.7.1a) och använde mjukvarans stödfunktioner som genererar kommentarsmallar, visar fel i koden, markerar variabler som ej används, eller föreslår att uppdatera variablers namn som inte stämmer.

Jag är inte helt nöjd med det hur jag hade själva arbetsgång, eftersom jag började utan övergripande planering. Om jag kunde göra annorlunda, så skulle jag hellre rita ett schema över hur klasser, deras variabler, metoder och relationer ser ut. I stället så började jag ta en instruktion efter en annan och implementera det som stod i kravspecifikationer. Det går att göra så när man skriver väldigt små program, men det kan leda till riktiga problem när man skriver mer omfattande kod eller samarbetare med andra. Då är det extra viktigt att se programmets ”skeletten” först och bli överens över vad som skickas som input/output mellan olika delar av koden.

En annan sak som jag skulle ha gjort annorlunda var att genomföra små tester av enstaka metoder direkt efter att jag blev färdig med att skriva någon metod. På så sätt skulle jag kunna upptäcka eventuella fel på tidiga stadier och fixa dem innan koden blev för komplicerat.

Istället litade jag på TestBank.java och skrev hela uppgiften färdig innan jag testkörde för första gången. Jag hade tur att det kom bara en litet fel som jag kunde enkelt hitta och åtgärda.

Vid felsökning kollade jag i err.txt vad felet/en kallades och på vilken rad i vilken klass det har uppstått. Efter det så genomförde jag en webbsökning för att säkerställa att jag förstår vad som felkoden innebär.

Det enda fel som jag fick var java.util.ConcurrentModificationException i Customer.java-metoden `public String closeAccount(int accountId)`. Sökning på StackOverflow ("How to avoid java.util.ConcurrentModificationException when iterating through and removing elements from an ArrayList - Stack Overflow," 2012) hjälpte mig att snabbt förstå att jag försökte ta bort elementet från ArrayList med att jag itererade genom listan, så jag flyttade borttagningen utanför loopen.

Efter det gick alla tester som de ska utan någon felrapportering och med returvärden som i testexemplar.

Systembeskrivning

Systemet består än så länge av tre klasser – BankLogic.java, Customer.java och SavingsAccount.java.

SavingsAccount.java är en basklass som inte har någon beroende på någon annan klass. Den klassen hanterar ett sparkonto som en typ av bankens konto. Typ och räntesats i den klassen är variabler som är konstanta och gemensamma för alla objekt av den klassen. Klassen håller på vilket nummer et visst konto har och hur mycket pengar det finns på. Varje kontonummer är unikt i hela banken därför finns det en gemensam klassvariabel som håller koll på vilket var det sista kontonumret som tilldelades. Den här klassvariabeln uppdateras varje gång när ett nytt objekt av SavingsAccount-klass skapas. Klassen ger möjlighet att jobba med summan som ligger på ett visst konto: se hur mycket pengar det finns, lägga en hel ny summa, lägga pengar till en befintlig summa, ta ut pengar och räkna ut räntan.

Customer.java är en klass som är beroende av SavingsAccount-klass. Customer.java hanterar en kund som en typ av personen involverade i bankens verksamhet. Förutom personliga data (namn och personnummer), så har varje kund en lista av bankkonto med alla sparkonton som de har. Med användning av publika metoder från SavingsAccount.java, så hjälper Customer.java att hantera kundens konto: lägga till eller ta bort ett konto, sätta på eller ta ut pengar från någon specifik konto samt få information om: alla data på alla konton, bara kontonummer på alla konton, alla data på ett specifikt konto.

BankLogic.java är en klass som är beroende av Customer.java-klass. BankLogic.java hanterar alla kunder som banken har. Här kan man lägga till eller ta bort en kund samt genomföra alla viktiga transaktioner för en viss kund på ett visst konto. Klassen får information om vilken kund (beroende på personnummer) och vilket konto (beroende på kontonummer) som ska hanteras och efter baskontrol (t.ex. att kundlistan inte är tom) skickar uppgifter vidare till Customer.java som vid behov skickar uppgifter vidare till SavingsAccount.java.

Diskussion

Jag försökte följa de här riktlinjerna när jag genomförde laborationen:

- ändring och hantering av något objekt skal genomföras bara av klassen som ansvarar för den här typen av objekt. Alla andra klasser ska ske arbetet med objektet som en ”black box” där uppgifterna skickas till för att få en viss output utan sätta sig in i hur allt det inre fungerar;
- alla variablers namn ska tala för sig själva och beskriva kodens logic på ett strukturerat sätt;
- användning av smarta for-loppar skall föredras;
- indata skall kontrolleras innan metoden utförs.

Jag tycker att jag har lyckats med dem mesta av de här riktlinjerna dock kanske inte på sättet som är effektiv eller lämplig, speciell om man skulle jobba med större projekt.

Som jag har redan nämnt skulle jag dedikerat mer tid till planering och testkörande. Min läsning följer dock objektorienterade principer av enkapsulering och abstraktion: 1) Alla instansvariabler är privata och kan nås bara via specifika publika metoder; 2) Systemet består av enstaka komponenter som bygger på varandra (“4 major principles of Object-Oriented Programming | Raymond Lewallen,” 2005).

Referenser

4 major principles of Object-Oriented Programming | Raymond Lewallen. (2005). Retrieved March 8, 2018, from http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/

How to avoid java.util.ConcurrentModificationException when iterating through and removing elements from an ArrayList - Stack Overflow. (2012). Retrieved March 8, 2018, from https://stackoverflow.com/questions/8104692/how-to-avoid-java-util-concurrentmodificationexception-when-iterating-through-an