UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

Applikationsutveckling i Java 7.5 p
5DV135

Obligatorisk uppgift nr

2

Namn	Rafael Da Alesandro	
E-post	c14rdo	@cs.umu.se
Datum	14-1-2016	
Handledare	Victor Zamanian Niklas Fries Adam Dahlgren Lindström	

Innehållsförteckning

Problemspecifikation

Syftet med denna laboration var att öva på XML-Parsning,skapa ett trådsäkert program och ytterligare träning utav MVC designmönstret. Programmet ska fungera som ett sista-minuten bokning som en användare kan använda för att hitta en resa. Programmet visar information som Stadnamn, hotellnamn, hotellbetyg, destination, resedatum, pris för resan och slutligen en bild på hotellet. Programmet ska kunna spara de senaste inställningarna från användaren och man ska kunna söka på specifika städer, som t.ex. Gran Canaria. Programmet ska vara uppkopplat mot fritidsresor.se där användaren kan ställa in en uppdateringsfrekvens för hur ofta programmet ska hämta information från hemsidan så den innehåller aktuell data.

Åtkomst och användarhandledning

Den körbara jar-filen finns i c14rdo/edu/appjava/ou2l med de kompilerade filer och programkoden. För att köra programmet så kan man starta det med kommandot: java -jar TravelInfo.jar Eller klicka på den exekverbara filen.(notera det står travel-Info, stort i, inte l). Den javaversion som krävs för att programmet ska fungera är 1.7 och nyare. I programmet så finns det fyra synliga knappar som alltid är framme. Dessa är Show Details som visar mer information om en resa då man klickat på den i tabellen. Upp och ned pilknappar som är till för att ändra uppdateringsfrekvensen utav programmet och slutligen Search som är till för att söka efter någon destination att resa till. I menyn Options så finns det Refresh som uppdaterar tabellen med ny information från hemsidan fritidsresor. Help skapar en pop-up där användaren kan läsa kort om hur man använder programmet och slutligen Exit som stänger av programmet.

Systembeskrivning

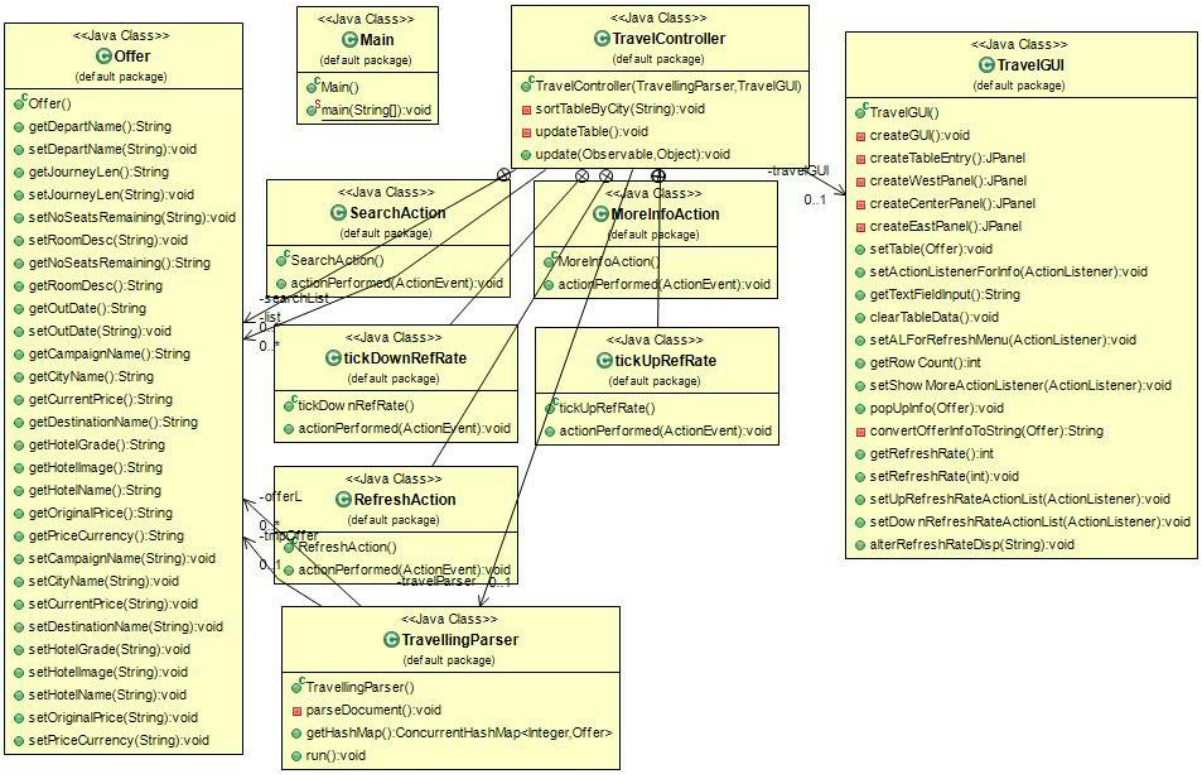
Programmet TravelInfo är designat efter design mönstret MVC, som står för ***model view controller***. Det som gör denna design så bra är att modellen och vyn ska vara så oberoende utav varandra som möjligt. Vilket tilllåter kontrollen att använda modellen och vyn på många olika sätt utan att behöva ändra på koden i modellen/vyn. Detta designmönster fungerar eftersom modellen, som utför alla beräkningar ***inte*** behöver hjälp från någon annan klass för att fungera korrek. Vyn behöver inte hjälp från modellen för att visa upp det grafiska gränssnittet, men kan kräva att vissa ActionListeners skapas i kontrollen för att sedan lägga till i de komponenter som finns i vyn. För denna uppgift så krävdes dessutom ännu ett design mönster, nämligen Observer-Observable.

Varför det design mönstret valdes att använda var för att det mest effiktiva sättet att bygga upp tabellen i programmet (den som visar datat om resor) uppdaterades endast efter att listan var färdiggjort. Vilket gjorde det så att programmet satt stilla i onödan ett tag tills listan var klar. En tråd användes för inläsningen av datat från Fritidsresor.se för tabell informationen och den listan var inte klar förän tråden var klar. Men tack vare Observer-Observable så gick det att notifiera från modellen till kontrollern att ett element vart till lagd i listan och gick att användas.
På så sätt var inte tabellen inte tom i onödan utan kunde ladda samtidigt som listan skapades.

TravellingParser är modellen som sköter all inläsning av datat från fritidsresor. TravellingParser extendar Observer och implementerar Runnable. Eftersom modellen kommer endast köras via trådar så implementerar den Runnable och eftersom det krävs en viss kommunikation mellan Kontrollen och Modellen så utökar modellen Observer för att notifiera kontrollern att ett nytt element finns i listan att plocka ut och visa i tabellen.

TravelGUI är vy klassen som visar upp all information på ett fönster med knappar som Show Details, Search och två pilknappar som ska föreställa ökning och säkning av uppdateringsfrekvensen. Det finns även en meny där refresh, help och exit är alternativ som går att använda. Show Details visar ytterligare information om en resa när man markerar den i tabellen, search är till för att leta efter destinationer att resa till. I menyn finns help som ger användaren mer information om hur programmet fungerar och om hur sökningen fungerar. Refresh hämtar ny data från fritidsresor. Exit stänger av programmet.

TravelController som är kontrollen för MVC metoden har främst som ansvar att se till att modellen och vyn fungerar ihop och så att de enda kopplingarna mellan modellen och vyn kommer från kontrollen. I denna klass så skapas 5 innre klasser som implementerar klassen ActionListener. Det är för att vissa knappars funktioner inte kan definieras innuti vyn eftersom det krävs någon information från modellen. Det bästa sättet att skapa dessa Actions för knapparna och menyerna är genom att skapa dessa inre klasser innuti kontrollen och sätta dessa komponenters ActionListeners via kontrollen.



Lösningens begränsningar

Den största begränsningen med programmet är nog att man inte kan boka en resa direkt via programmet. Dessutom så saknas extra funktionalitet som pris sökning, prisbegränsning och programmet visar inte vars ifrån resorna startar. Med det så menas till exempel att det saknas information som kan berätta till användaren att en resa till Grekland går från Umeå. Man kan inte heller sätta krav på hotellbetyget som exempelvis söka på hotell som har minst 3 i betyg.
Inga JUnit-test skapades eftersom det verkade för konstigt att skapa JUnit tester till parsern, för onödigt att göra till Offer, då den endast hade getters och setters och vyn samt kontrollern samma anldedning som för parsern.

Problem och reflektioner

Det jobbigaste med denna labb var i början när xml-parsern skrevs. Eftersom jag inte hade någon tidigare erfarenhet så tog det ett bra tag innan den fungerade. Dessutom så stötte jag på några fel som jag inte förstod som höll mig fast på parsningen alldeles för länge. Uppgiften kändes tråkig och jobbig då man inte kom någon vart, men så fort parsningen rullade så vart den genast mycket roligare. Det jobbigaste med parsern var nog att jag inte förstod hur den fungera förutom vad metoderna som skötte parsningen gjorde och var till för.

Varje gång jag gör en lapp så vill jag helst förstå allt som händer i det program jag skrivit och jag kände inte riktigt det för denna labb. Även fast programmet inte har några JUnittester så tycker jag att den vart rätt så felfri. Nu kan det vara så det finns logiska fel som är svårare att hitta som oftast förekommer då man har byggt sitt program kring bra tester, men denna gång så verkar det inte

Testkörningar

Inga riktiga test skapades under labbens gång, men flera tester utfördes genom byggandet av programmet. T.ex. parsern räckte med att se att ett av alla element sparades och gick att skriva ut eller använda. Med det så menas att klassen Offer som element kunde skriva ut dens attribut. Sedan så var det för det mesta GUI test som felrisk vid knapptryckningar som skapades. Alla test som användes för att hitta fel och säkerställa sig om att programmet gör som det ska gjordes under körtid utav programmet. När Show Detail knappen implementerades så testades den för fel genom att inte markera ett element och försöka visa mer information om den. Det som hände då som resultat var att programmet krashade på grund utav att getRow metoden returnerar -1 om ingen row är markerad. -1 i den listan som användes gav out of bounds som krashade programmet.