Projektbeskrivning

<Bookie>

2017-03-06

Projektmedlemmar:
Johan Can <johca907@student.liu.se>
Yousef Hashem <youha847@student.liu.se>

Handledare:
Dennis Ljung <denlj069@student.liu.se>

Table of Contents

Planering

1. **Introduktion till projektet**

Vi har tänkt skapa ett kalendersystem där man kan boka, avboka i sin egen kalender samt kolla på andras scheman och kalendrar. Tanken är att varje kalender ska bete sig som en inloggning och att man ska behöva ett lösenord (om användaren vill) för att få tillgång till kalendern. Om man råkar boka en tid samtidigt som en befintlig tid i en annan kalender och det finns minst ett nyckelord som matchar ska programmet fråga om din händelse är förknippad med andra personens händelse och i så fall koppla de två händelserna till varandra. Allt ska representeras med ett användarvänligt gränssnitt likt den Google Calendar använder sig av.

Varje kalender ska vara oberoende av de andra kalendrarna men ska kunna titta på de andra kalendrarna och kolla efter matchande eller liknande mönster som det som finns i kalendern själv. Frekvent använda ord eller händelser kommer ges som förslag vid liknande event och tider. Vid bokning av någon tid ska man kunna välja ifall man vill att den här tiden/eventet skall upprepas exempelvis dagligen eller varje vecka.

2. Ytterligare bakgrundsinformation

Man ska kunna boka evenemang och avboka fram tills fem minuter innan tiden är inne. Samma person ska inte kunna boka två saker under samma tid, och vid överlappning ska klienten fråga om användaren verkligen vill skriva över ett evenemang med ett annat. Man ska inte kunna boka in något på en tid som redan är förbi, och man ska inte heller kunna boka in någon annans schema från sitt utan tillåtelse. Varje användare ska därför kunna ge tillåtelse till andra användare så att de kan boka in tider på sitt schema.

3. Milstolpar

#	Beskrivning
1	Skapa lämpliga klasser.
2	Rita ut ett fönster med relevanta knappar.
3	Koppla fönstret till själva programmet.
4	Lägg till interaktion med mus och tangentbord.
5	Skapa en kalender.
6	Lägg till flera användare.
7	Koppla en kalender till varje användare.
8	Det finns datatyper för kalendrar samt ett sätt att boka tider på.
9	Restriktioner och regler implementeras. (Tider får inte överlappa, osv)
10	Spara/ladda kalendrar så att den inte skapas på nytt vid varje uppstart.
11	Upprepade bokningar.

12	Påminnelser på användarens begäran.
13	Lämpliga programgjorda påminnelser (exempel: dags att åka, förberedelser)
14	Tillåta andra användare att göra bokningar åt dig.
15	Delade kalendrar.
16	Implementera inloggningar och lösenord för varje användares kalender.
17	Ge lämpliga förslag vid bokning (Exempelvis om två bokningar någorlunda matchar så ges förslag om personerna i fråga ska gå tillsammans och uppdaterar informationen därefter).
18	Skapa en databas som lagrar varje användares preferenser, vanor/mönster.
19	Koppla kalendern till en GPS.
20	Meddela om transporttider osv.
21	Få reda på vilken dag det är.

4. Övriga implementationsförberedelser

Alla större åtskiljande funktionalitet kommer delas upp i olika klasser, såsom bokningar, användare, inlärning, databas, interface för kalendrar etc.

Book: Boka, avboka, restriktioner.

Calendar: Nytt objekt för varje kalender. Kalendermall, vem som är ägare etc.

User: Namn, kontakt- och allmän information, relationer(familj, vänner) etc.

Frame: Knappar, grafisk representation av kalendern etc.

UserDatabase: Lagrar information om en användares vanor, mönster etc.

UserPredictions: Ger förslag utifrån användarens vanor, mönster, etc.

Main: Startar upp programmet.

Tanken är att varje klass ska ha hand om just en funktionalitet och att klasserna sedan kopplas samman och beror på varandra för att programmet ska fungera i helhet.

5. Utveckling och samarbete

Vi arbetar genom att båda skriver på samma del till och börja med, och diskuterar medan vi arbetar för maximal inlärning. Ungefär fram till milstolpe 10 kommer vi arbeta tillsammans. Efter det kommer vi arbeta mer och mer individuellt och visa upp vad vi har gjort för varandra. Målet är att båda ska lära sig lika mycket och skriva ungefär lika mycket kod så att inte en lär sig allting och förklarar för den andra. Arbetstiderna kommer främst vara på eftermiddagar och kvällar, och på helger ifall det krävs. Vi har båda ambitioner att nå betyg 5 i kursen.

Slutinlämning

6. Implementationsbeskrivning

6.1. Milstolpar

Färgkodning av milstolparna ovan har implementerats där grön markering visar att milstolpen är helt genomförd. Inga av milstolparna blev delvis genomförda och de återstående ickemarkerade milstolparna är de som inte har implementerats.

Milstolpen som är markerad med gul färg började vi smått på men kom snabbt fram till att det krävdes mer arbete och tid än vad som fanns kvar och gavs därför upp för att finslipa koden.

6.2. Dokumentation för programkod, inklusive UML-diagram

6.2.1 Övergripande programstruktur

Hela programmet styrs av ett grafiskt gränssnitt där varje knapp på gränssnittet är kopplad till en ActionListener. Det är alla lyssnare som driver programmet framåt. Vid varje knapptryck anropas en Action som skapar en pop-up frame med komponenter som hör till den funktionen. Med hjälp av en metod har vi gjort att varje ny frame som skapas har ett standardutseende med exempelvis en ”cancel” och en ”confirm” knapp. Confirmknappen är kopplad till en speciell Action (beroende på vilken typ av pop-up det är) som läser in alla inmatningar och använder dem som parametrar för att utföra uppgiften. Cancelknappen stänger helt enkelt ner framen utan att ta hänsyn till inmatningarna.

Exceptions hanteras genom att ett try-catch block där det kan ske vissa typer av felinmatningar, exempelvis ett passerat datum. Vissa exceptions, exempelvis NullPointerExceptions, ska inte kunna förekomma överhuvudtaget och är därför hanterade med if-satser. Pop-upen kommer inte upp om inte allt som ska finnas existerar, exempelvis så skriver programmet ut ett meddelande ifall man försöker byta användare utan att några användare finns.

Skärmen uppdateras successivt vid varje anrop till ActionListeners och på så sätt undviks implementationen av en tick-funktion.

6.2.2 Övergripande programstruktur

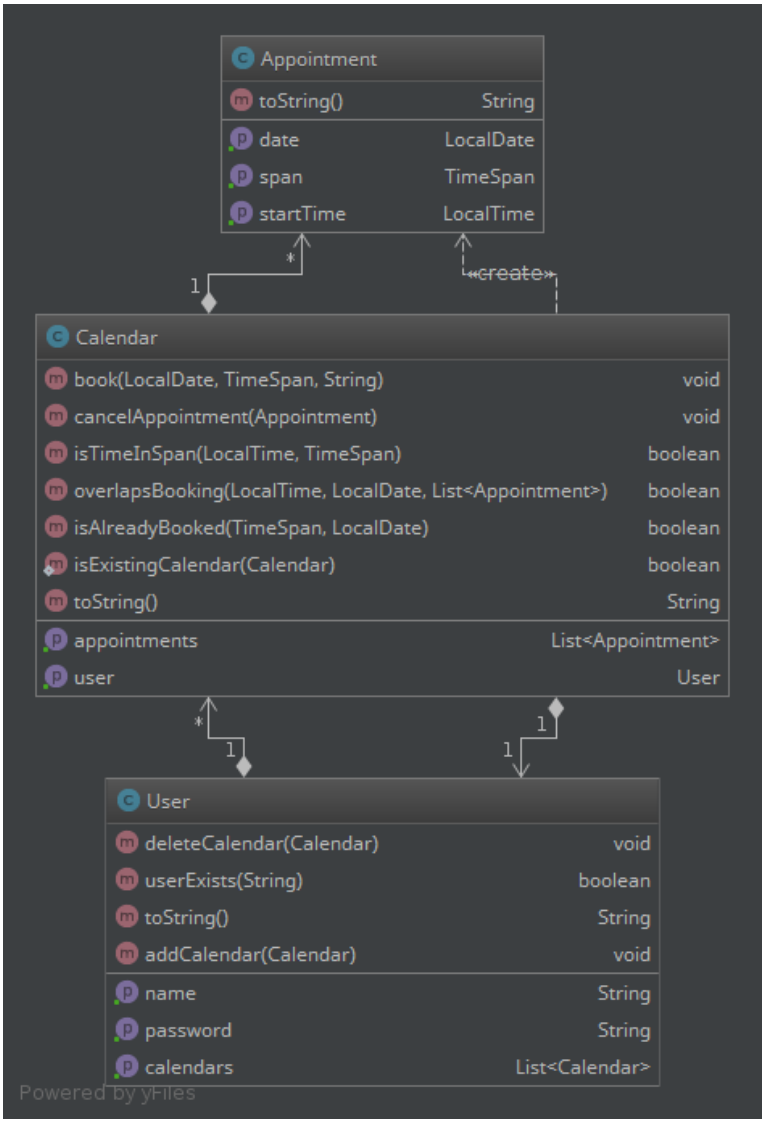
De tre viktigaste klasserna i programmet är User, Calendar och Appointment:

- Varje User-objekt har ett namn, en lista av kalendrar och ett valfritt lösenord. Implementation av en lista av Calendars är för att varje User ska kunna förknippas med olika Calendar-objek. I User klassen finns metoder för att lägga till och radera objektets egna Calendars. I konstruktorn kollas även om ett User-objekt med de inmatade parametrarna redan finns, för att förhindra dubbletter av Users.
- Varje Calendar-objekt innehåller en lista med Appointments, för att hålla alla appointments som hör till den kalendern samlade. Ett User-objekt används som inparameter för att förknippa varje Calendar med en User . Klassen hanterar regler och restriktioner såsom att det inte är möjligt att boka två saker på samma tid och kollar i konstruktorn om ett Calendar-objekt redan finns för att förhindra dubletter. Varje objekt har möjlighet att boka och avboka appointments genom att förändra listan av Appointments som varje objekt har.

- Varje Appointment-objekt skapas med hjälp av ett datum, ett tidsintervall och ett ämne. Klassen innehåller inga interna restriktioner utan sköter bara skapandet av Appointment-objekt. Det enda som krävs för att objektet ska skapas är att det uppfyller reglerna för inparametrarna samt att ämnet inte är en tom sträng.

De tre klasserna är kopplade till varandra så att Calendar-klassen tar ett User-objekt som inparameter och att varje Calendar-objekt innehåller en lista av appointments. Vid bokning av en appointment körs ett antal kontrollfall för att förhindra felbokning. Detta är även anledningen till varför Appointment-konstruktorn inte har några interna restriktioner och kontroller. Varje User-objekt hanterar även en lista av Calendars. Detta ger en direkt relation mellan User- och Calendar-klasserna och Calendar- och Appointment-klasserna och en indirekt relation mellan User- och Appointment-klasserna.

En visualisering av detta finns i figur 1.



Figur 1

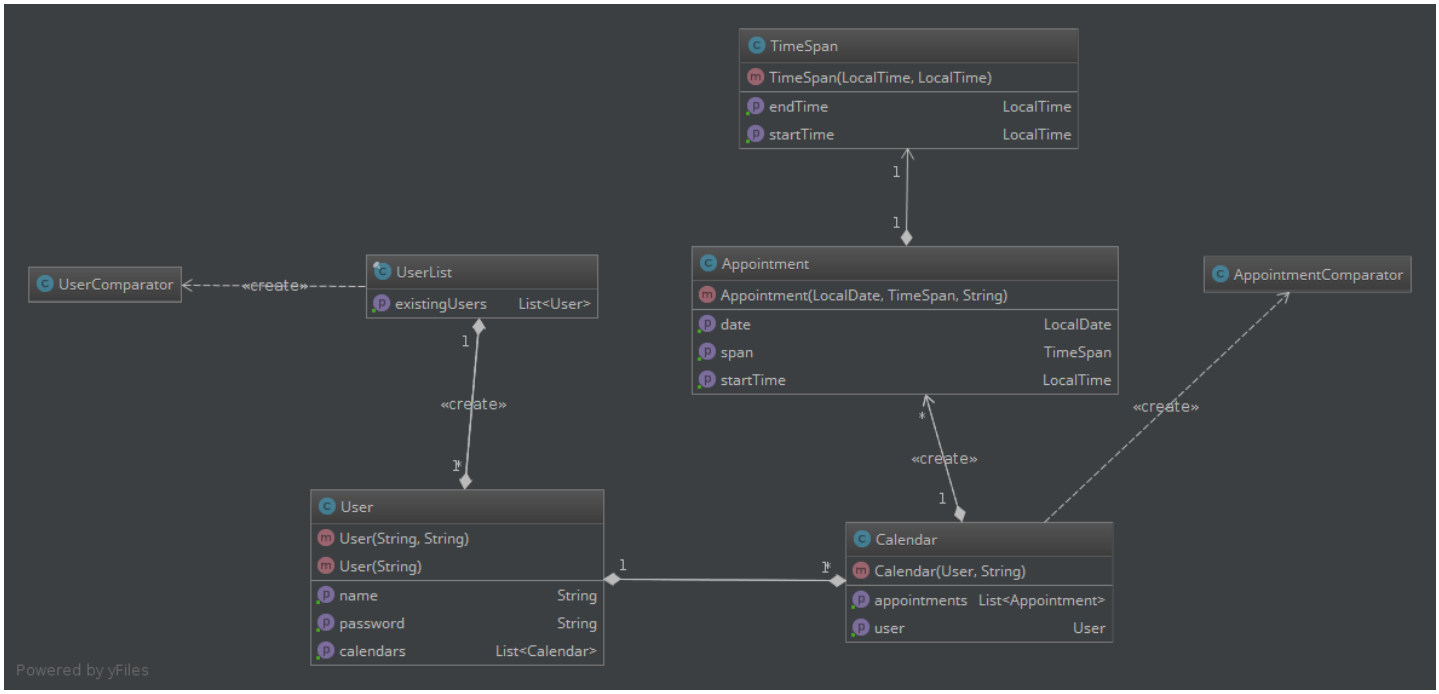
Utöver User, Calendar och Appointment finns flera mindre klasser. Utöver vanliga klasser finns även comparators och en singleton.

- Comparator-klassen UserComparator används för att jämföra User-objekt med avseende på första bokstaven i ”name”. Detta för att man lätt ska kunna hitta en användare.
- Comparator-klassen AppointmentComparartor har samma funktionalitet som UserComparator men används för att jämföra AppointNär ni pratar om klasser och metoder **ska deras namn anges tydligt** (inte bara ”vår timerklass” eller

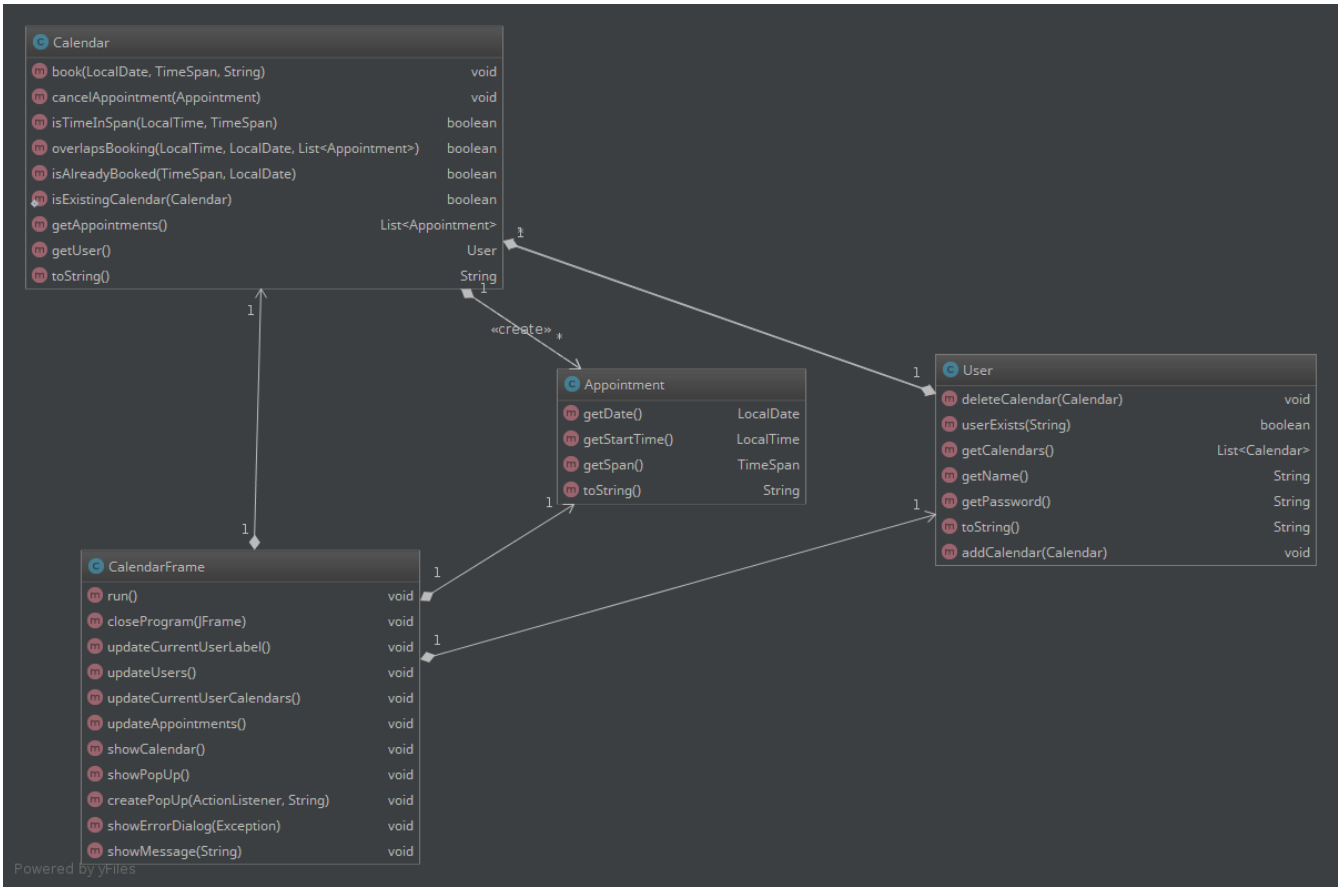
”utritningsmetoden”).

- Framhäv gärna det ni själva tycker är bra lösningar eller annat som handledaren borde titta på vid bedömningen.ment-objekt. Den jämför främst med avseende på starttiden.
- Singleton-klassen UserList hanterar listan med alla användare och ser till att det endast finns en lista av användare. Den använder sig av UserComparator för att sortera bland User-objekten som finns i listan.
- TimeSpan tar in två LocalTime objekt, en start- och en sluttid, och skapar ett tidsintervall istället för två individuella tider. I klassen görs även en kontroll om starttiden kommer efter sluttiden, annars godkänns inte inparametrarna.

I figur 2 visas hur de mindre klaserna är kopplade till de tre större klasserna i programmet.



Figur 2



Figur 3

CalendarFrame hör ihop med de tre större klasserna på ett unikt sätt. Det är den stora klassen i programmet och den använder sig av Calendar-, User- och Appointment-klasserna för att göra det användaren vill. Det är även den grafiska representationen av hela programmet och sköter bildandet och visningen av pop-ups. I figur 3 ovan ser man att CalendarFrame har många diverse metoder medan de andra tre klasserna har metoder som är mer riktade åt ett område.

6.2.3 Nämnvärda lösningar

Något som vi tycker är bra med programmet är feedbacken användare får vid användning av programmets funktioner. Om något fält saknas eller en inmatning är ofullständig I något steg kommer ett fönster upp som säger precis vad som saknas istället för att bara ignorera eller signalera fel. Detta tog majoriteten av vår tid och krävde att många fall var tvungna att täckas.

6.3. Användning av fritt material

Det enda bibliotek som använts som inte ingår i Java 8 är MigLayout. Detta för att kunna få bättre struktur på alla komponenter I alla fönster. Exempelvis kan man I MigLayout skapa ett rutsystem och sedan positionera komponenter med koordinater vilket är mer flexibelt än BorderLayout. Dessutom valde vi just MigLayout över andra layout-typer eftersom det var den layouten föreläsaren i kursen rekommendarade.

6.4. Användning av objektorientering

1. **Definiera en klass som final**: Klassen UserList definierades som final eftersom det är inte menat att den ska kunna ärvas eller ha några subklasser. Vi är medvetna om att detta stänger av en väldigt kraftfull och flexibel funktion av språket för just den delen av koden. Anledningen vi valde att göra detta är eftersom klassen är en singleton-klass som inte ska kunna tas efter och med deklarationen final kan klassen fortfarande ärva andra klasser ifall det skulle behövas.
2. Som nämnt i stycket ovan är klassen UserList även en singleton-klass. UserList är en klass som ska hantera en lista av alla befintliga användare och behöver endast ett objekt i vår implementation. Objektet ska kunna nås av de som behöver det och detta uppnås genom att skapa en singleton klass och skicka ut en instans av det objektet till de som frågar efter det. Med singleton-klasser kan man behandla alla dessa problem på ett bra sätt. Man garanterar att endast ett objekt av den klassen skapas, ger en offentlig åtkomstpunkt till objektet och man kan skapa fler instanser av objektet ifall det behövs utan att påverka användningen av det nuvarande objektet.

   Det är svårt för oss att veta exakt hur man ska göra för att lösa detta problem i ett språk som inte har den här egenskapen då vi endast programmerat i Python och Java, men en av tankarna är att man hårdkodar in skapandet av ett objekt och kallar till det på samma sätt som vi frågar efter instansen i Java. Det är dock inte en fin lösning då hårdkodning är svårt att ändra på utan att påverka hela programmet och svårare att förstå om man läser koden utan att ha arbetat på programmet förut.
3. Vi har även valt att skapa två komparatorer, AppointmentComparator och UserComparator, för att lättare kunna sortera listor av Appointments och Users på det sättet vi vill. Vi blev introducerade till komparatorer i tetrislabben och tyckte det var ett bra sätt att jämföra objekt på för att sortera en lista. Det tar bort behovet för att skapa en funktion som går igenom alla element i listan manuellt och jämför med elementet innan tills allt ser rätt ut som vi gjorde i Python. Dessutom går det att ändra sättet objekten jämförs på utan att behöva strukturera om en hel funktion. Detta har en tydlig struktur och är lätt att implementera för det ändamålet.

6.5. Motiverade designbeslut med alternativ

1. Ett designdilemma som uppstod tidigt var vilken layout som skulle användas. Valet stod mellan MigLayout, GridBagLayout, SpringLayout och BorderLayout. Valet föll på MigLayout på grund av dess flexibilitet, där man skapa ett rutnät och positionera komponenter med koordinater som i GridBagLayout. Dessutom kan man vid behov positionera komponenter med koordinater i pixlar precis som i SpringLayout. Detta är krångligare än att använda BorderLayout men ändå simplare och mer flexibelt än de andra två layout typerna och ger dessutom större flexibilitet än alla tre ovannämnda layouter.

   Hade vi istället använt oss av BorderLayout hade vi behövt skapa flera olika JPanels för att nå önskat resultat. Detta kan ses som mer modulärt men samtidigt onödigt. GridBagLayout och SpringLayout var de två vi funderade på att använda först men kom sedan fram till att MigLayout har funktionaliteten av båda och vi släppte snabbt iden att använda någon av dem. Därför användes MigLayout istället.

2. När en funktion I programmet används skapas en frames för att visa hur funktionen används. Eftersom det finns många olika funktioner så behövdes det många olika frames med olika innehåll.

   Vi valde då mellan att skapa flera olika fördefinierade frames med alla komponenter eller skapa en basframe med bara standard funktioner (confirmknapp och cancelknapp) och sedan lägga till de komponenter som behövs för respektive funktion. Vi valde att skapa en basframe eftersom det valet minskar mängden redundant kod.

3. Först när ett User-objekt skapades lades detta objekt till i listan ExistingUsers som fanns I User klassen. Detta för att dels kunna visa alla User objekt som skapats och för att kunna kolla om en använda försöker skapa ett User objekt som redan finns.

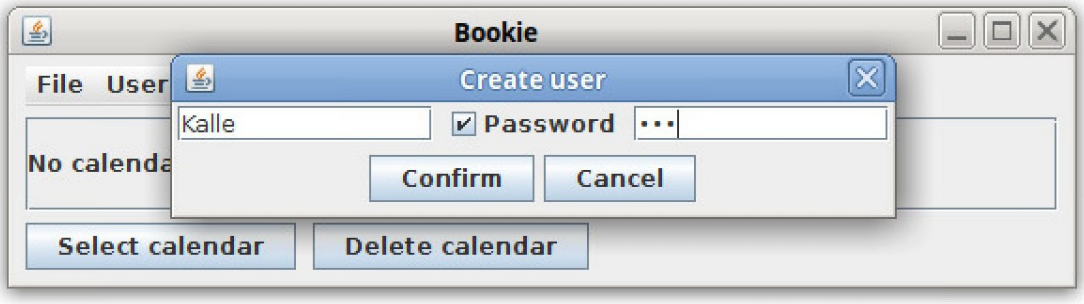
   Men senare implementerades en UserList som innehåller att User objekt. Detta för få mer struktur, där allt som har med listan att göra hanteras där istället för att blanda funktionalitet av ett User objekt och en lista av dessa I en och samma klass.

4. Alla funktioner beror på dels en Action (popUpAction) som gör en frame och en Action som utför själva uppgiften(ConfirmAction). Det blev många olika Actions som lades I CalendarFrame och vi tyckte att det kunde varit bättre om dessa lades I en separat utility class. På sätt kan man anropa respektive Action från CalendarFrame och få mer strukturerad kod samt hålla olik funktionalitet separata. Dock fungerade detta inte på grund av att varje action beror på komponenter I CalendarFrame och måste därmed implementeras I den klassen och vi valde därför att ha med alla Actions i CalendarFrame.

5. Ännu ett av dilemmorna vi hade var ifall vi skulle ha med en lista av alla Calendars och en lista av alla Users vid varje pop-up för att den som ska använda programmet ska markera en Calendar och en User som ska användas eller ifall vi skulle hålla koll på vilken User och Calendar som är i fokus just nu och använda den som val för bokning och avbokning. Vi valde självklart andra alternativet eftersom det är lättare att implementera och mindre rörigt för den som använder programmet. Dessutom minskar det rörigheten i koden då många av metoderna hade behövt onödiga inparametrar ifall vi hade valt det första alternativet istället för det andra.

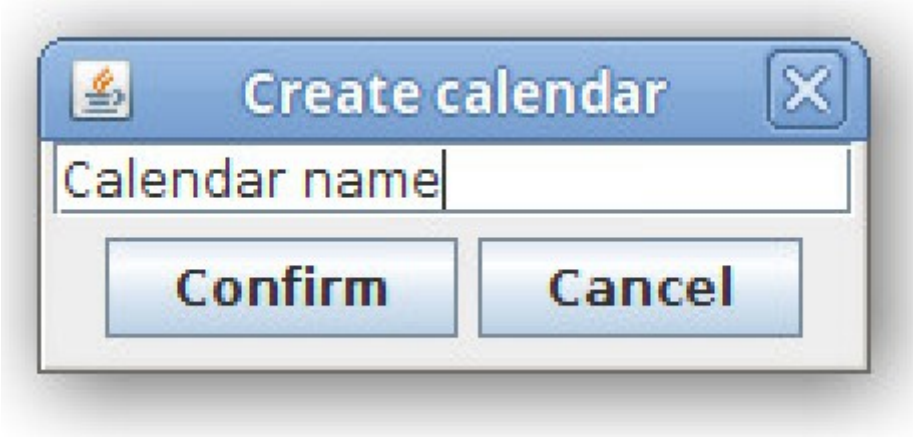
7. Användarmanual

När man kör koden kommer det först upp ett fönster med en menyrad längst upp till vänster och två knappar längst ner. Dessa knappar är allt som användaren kan interagera med, I menyraden finns tre fält – ”File”, ”User” och ”System”. Det första man måste göra är att skapa en användare och byta till den. Detta gör man genom att trycka på ”User” och sedan på ”New user”. Då kommer en ny ruta fram och det ser ut såhär:



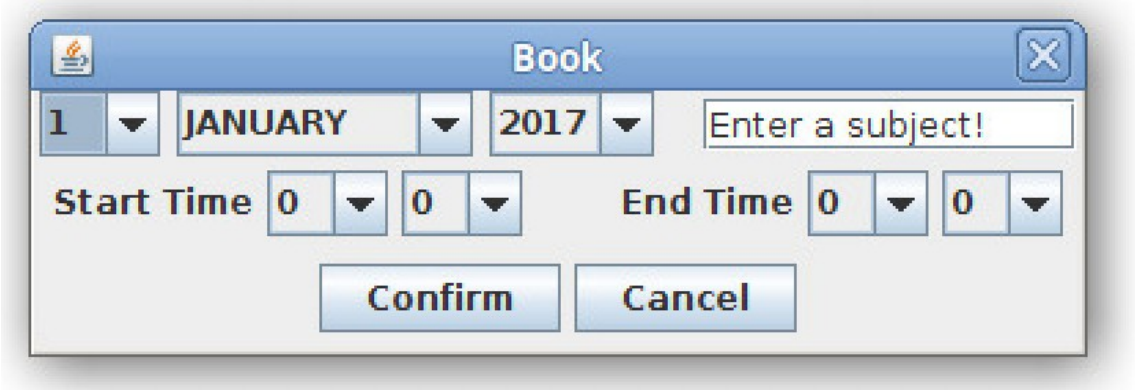
Efter att man skapat sina användare måste man välja en användare som ska stå i fokus. Detta gör man genom att på ”User” i menyraden och sedan på "Change user”. Då kommer en ny ruta upp med en lista på alla nuvarande användare och ett fält att skriva in sitt lösenord i. Har man skapat användaren utan ett lösenord så behöver man inte skriva in något i fältet.

Efter det är det dags att skapa en kalender. För att göra det navigerar man sig in till ”File” och sedan ”Create calendar”. Då kommer en ruta upp med bara ett skrivfält. I detta fältet skriver man det önskade namnet på kalendern och trycker sedan på konfirm.



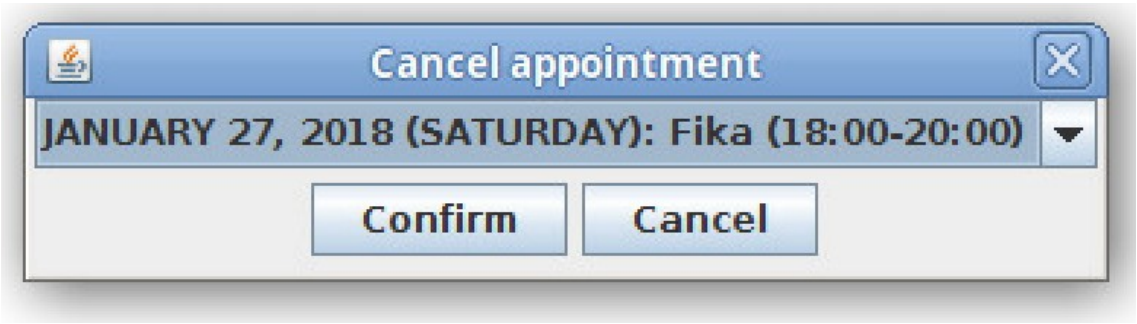
Nu när man har skapat en kalender kan man börja boka, men innan man kan boka någon måste man välja en kalender genom att trycka på ”Select calendar” och sedan välja vilken kalender man vill boka i. Detta gör man i rutan som kommer upp efter att man tryck på knappen.

För att sedan få upp bokningsrutan trycker man på ”File” och sedan på ”Book”. En ruta med många fält och knappar kommer upp. På denna rutan finns all information man behöver för att boka en tid.

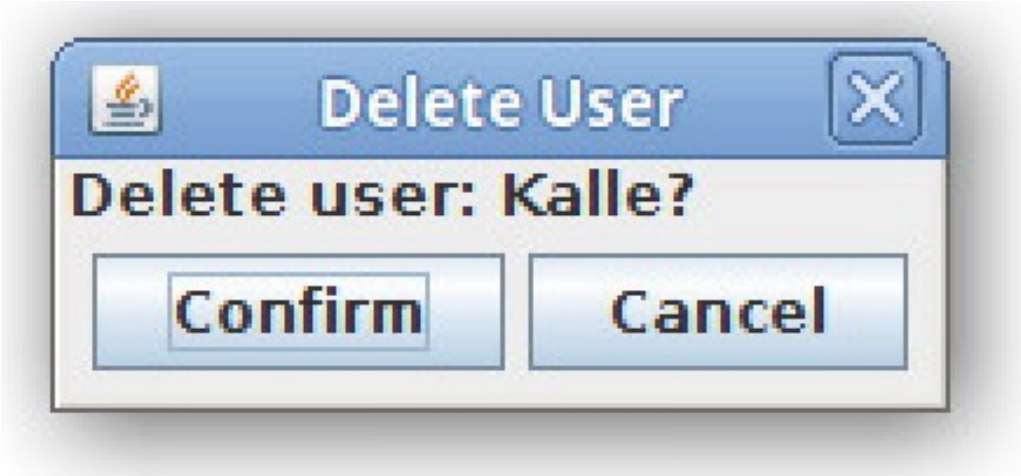


Om man skulle vilja avboka en tid eller ta bort en användare/kalender så navigerar man sig in i ”File” respektive ”User” eller trycker på knappen ”Delete calendar”.

Väljer man att avboka en tid kommer en ruta upp med en lista på alla bokningar man har gjort i den nuvarande kalendern. Efter att man valt vilken tid man vill avboka trycker man på ”Confirm”.



Väljer man att ta bort en användare eller en kalender kommer det bara upp en ruta som frågar efter bekräftelse och sedan tas den bort.



8. Slutgiltiga betygsambitioner

Nu i efterhand siktar vi på en 4a i det här projektet då vi underskattade hur mycket tid det skulle krävas för att få en 5a.

9. Utvärdering och erfarenheter

- ***Vad gick bra? Mindre bra?***

- ***Vilket material och vilken hjälp har ni använt er av? Har ni gått på föreläsningar? Läst boken? Letat på nätet? Gått på handledda labbar? Ställt många frågor? Vad har ”hjälpt” bäst? Vi vill gärna veta för att kunna vidareutveckla kurs och kursmaterial åt rätt håll!***

  Vi har gått på majoriteten av alla handledda labbar och föreläsningar, men nätet har varit till mest hjälp. Vi ställde även frågor till labbassistenterna.
- ***Har ni lagt ned för mycket/lite tid?***

  Vi har lagt ner för lite tid för att nå vårt preliminära mål men ändå tillräckligt mycket för att vara nöjda med arbetstiden.
- ***Var arbetsfördelningen jämn? Om inte: Vad hade ni kunnat göra för att förbättra den?***

  Arbetsfördelningen har varit väldigt jämn. Vi försökte hela tiden dela upp ett jämnt arbete så att inte ena parten gör allt inom ett område. Vi gick regelbundet igenom vad vi gjorde och demonstrerade för varandra så att båda

parterna alltid hängde med i vad som hände.

- ***Har ni haft någon nytta av projektbeskrivningen? Vad har varit mest användbart med den? Minst?***

  Vi har haft nytta av den. Milstolparna var mest användbara med marginal. Det var också bra med feedback från handledaren vid inlämning av beskrivningen. Vi använde oss också av arbetsmetodiken vi hade gjort i projektbeskrivningen. Allt var ungefär lika användbart om man inte räknar med milstolparna, och vi skulle nog säga att det inte fanns någon som var minst användbart.

- ***Har arbetet fungerat som ni tänkt er? Har ni följt "arbetsmetodiken"? Något som skiljer sig? Till det bättre? Till det sämre?***

  Det har gått som vi tänkt oss. Det skedde inga större problem under arbetsgången och vi följde arbetsmetodiken med vissa ändringar till det bättre då vi kom på bättre lösningar allt eftersom.

- ***Vad har varit mest problematiskt, om man utesluter den programmeringstekniska delen? Alltså saker runt omkring, som att hitta ledig tid eller plats att vara på.***

  Eftersom en av parterna är bosatt i Norrköping fanns det vissa svårigheter när det gäller ledig tid eftersom man inte kunde arbeta tillsammans under sena kvällar. Plats att vara på fann vi alltid.

- ***Vilka tips skulle ni vilja ge till studenter i nästa års kurs?***

  Börja så snart som möjligt och se verkligen till att lägga tid åt projektet varje vecka. Viktigt är också att hitta någon på ungefär samma kunskapsnivå som en själv för att maximera inlärningen.

- ***Har ni saknat något i kursen som hade underlättat projektet?***

  Vi fick allt vi hade kunnat önska oss.

- ***Har ni saknat något i kursen som hade underlättat er egen inlärning?***

  Fler labbar som tetris. Första 4 labbarna borde varit mindre instruktioner och mer eget tänkande.