TDDC77 - Projekt

Jimbo Resebyrån

Outline

1. **Introduktion**
2. **Funktionella krav**
3. **inmatningar**
4. **Struktur**
5. **Klass-hierarki**
6. **UI**
7. **Felhantering**
8. **Datastrukturer**
9. **Dokumentation**
10. **Högre betyg**

1. Introduktion

Jimbo är en liten resebyrå med 5 anställda som ligger i centrala Linköping. Nu är de i behov av ett bokningssystem som kommer att användas av bara anställda. Tanken är att anställda ska få genomföra alla sina dagliga uppgifter utan att behöva gå till resebyråns hemsida.

Figur 1 - Ennia och Julia använder bokningssystemet för första gången.

2. Funktionella krav

Användarna, anställda, ska kunna genomföra följande uppgifter:

Söka resa

Att söka efter bokade resor baserat på passagerarens resedokument-id. Varje resa ska ha ett unikt nummer som visas för användaren. Sökningen kan ge flera matchningar. Följande ska visas för varje matchning:

- resenummer(unikt)
- Flygnummer
- Avgångsstad
- Avgångsdag
- Ankomststad
- Ankomstdag(ska beräknas baserat på avgångsdag, avgångstid och reselängd)
- Veckonummer
- År
- Avgångstid
- Ankomsttid(ska beräknas baserat på avgångstid och reselängd)

Följande format ska användas:

Resenummer — Flygnummer — Avgångs stad(Avgångs dag) — Ankomst stad(Ankomst dag) — Veckonummer — År — Avgångstid — Ankomsttid

Passagerarens namn och antalet matchningar ska också visas.

Note 1

Exempel:

```
Johan Berglund - 2 matches found.
================================================================================
RQ123 — IP378 — Milan(Wednesday) — Stockholm(Wednesday) — V19 — 2022 — 13:00 — 16:00

RQ778 — IP379 — Stockholm(Wednesday) — Milan(Wednesday) — V40 — 2022 — 15:10 — 18:10
```

```
Sara Bengtsson - no matches found.
================================================================================
```

Resor som redan gjorts eller börjat ska inte visas.

Note 2

För enkelhets skull är alla tider i Linköping tid.

Note 3

Boka resa

Att boka resa genom att ange följande:

- passagerarens förnamn och efternamn.
- Resedokument-id
- Flygnummer
- Vecka
- År
- Uppdateringsbarhet(true eller false)

Ett unikt resenummer genereras och visas för användaren så fort bokningen är klar.

Note 4

Det är inte möjligt att boka resor till datum innan nuvarande datumet.

Note 5

Det blir ett felmeddelande om flyget är fullbokat och det inte finns någon ledig plats.

Note 6

Det blir ett felmeddelande om flygnummret inte är giltigt..

Note 7

Avboka resa

Att avboka resa genom att ange resenummret.

Man ska söka efter resan, skriva ner dess nummer och sedan avboka den genom att använda den här funktionen.

Note 8

Det blir ett felmeddelande om inmatade resenummret inte finns.

Note 9

Updatera resa

Att uppdatera resa genom att mata in resenummret först. Om resan är uppdateringsbar då ska användaren kunna uppdatera bara passagerarens förnamn, efternamn och resedokument-id. Om resan inte är uppdateringsbar då ska ett felmeddelande visas för användaren.

Det blir ett felmeddelande om inmatade resenummret inte finns.

Note 10

Skapa flyg

Att skapa flyg genom att ange följande:

- Avgångsstad
- Avgångstid
- Avgångsdag
- Ankomststad
- Flygbolag
- Flygnummer (unikt)
- Antal platser
- Reselängd

Det förutsätts att alla flyg är rullande och upprepas en gång per vecka.

Note 11

Det är inte möjligt att ha två flyg med samma flygbolag, avgångs stad och ankomst stad. Det blir ett felmeddelande i så fall. Notera gärna att `UIManager` är ansvarig för att skriva ut alla felmeddelanden.

Note 12

Ta bort flyg

Att ta bort flyg genom att ange flygnummret. Alla resor kopplade till flyget ska också tas bort automatiskt. Det blir ett felmeddelande om flyget inte finns.

Avsluta

Programmet avslutas och kontrollen returneras till terminalen.

Autentisering(inte i huvudmenyn)

Autentisering sker genom att använda command-line argument:

```
java Jimbo -u shipa001 -p 3eff43
```

- Det ska inte finnas någon viss ordningen på `-u` och `-p`. Till exempel `java Jimbo -p 3eff43 -u shipa001` ska också fungera lika bra.
- Ett välkomstmeddelande visas för användaren innan huvudmenyn visas.
- I fall användarnamnet och/eller lösenordet inte stämmer ska ett felmeddelande visas för användaren och programmet avslutas automatiskt.
- I fall `-u` och/eller `-p` saknas ska det bli ett felmeddelande.
- I fall `-u` och/eller `-p` saknar värde ska det bli ett felmeddelande.
- I fall det finns fler argument, ogiltiga, än `-u` och `-p` blir det ett felmeddelande.
- Alla användare är hårdkodade i programmet och sparas i en datastruktur. **Det ska finnas en klass som presenterar användare(User).**

Anställd	**Användarnamn**	**Lösenord**
Shima Parsson	`shipa001`	`3eff43`
Ennia Pettersson	`Ennpe001`	`wedr5t`
Julia Noor	`julno001`	`wwr4r3`
Johan Karlsson	`johni001`	`lk9h6d`
Adam Bergqvist	`adabe001`	`0o3u33`

Figur 1 - hårdkodade användare i systemet

bokningssystemet är temrinal-baserat.

Note 13

Några exempel på fel inmatningar vid autentisering

- `java Jimbo -u shipa001 -p`
- `java Jimbo -u -p 3eff43`
- `java Jimbo -u shipa001`
- `java Jimbo -u shipa001 -p 3eff43 -v verbose`
- `Java Jimbo`

3. Inmatningar

Inmatning	Format	Datatyp	tillåtna värden
Resenummer	LL-DDD(L står för bokstav, D står för siffra)	String	—
Flygnummer	LL-DDD(L står för bokstav, D står för siffra)	String	—
Avgångs stad	—	String	—
Avgångs dag	—	String	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Ankomst stad	—	String	—
Ankomst dag	—	String	Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
Veckonummer/vecka	—	int	1-52
År	—	int	—
Avgångstid	HH:MM(H står för timme, M står för minut)	String	—
Reselängd	—	int	0-18 (timmar)
passagerarens förnamn	—	String	—
Passagerarens efternamn	—	String	—
resedokument-id	DDDDDDDD(D står för siffra)	String	—
Uppdateringsbarhet	—	Boolean	True/False
Flygbolag	—	String	Minst 10 olika flygbolag.
Antal platser	—	int	80-380

Figur 2 - Alla inmatningar ska följa ovanstående tabellen

Om inmatade data inte följer ovanstående tabellen då blir det felmeddelande. Användaren kommer dock att få försöka om.

Note 14

4. Struktur

Projektet innehåller minst tre olika paket som följande:

- `se.jimboagency.bookingsystem`
- `se.jimboagency.bookingsystem.ui`
- `se.jimboagency.bookingsystem.logic`

Alla klasser som gäller gränssnitt ligger i `se.jimboagency.bookingsystem.ui` och alla klasser som gäller logik ligger i `se.jimboagency.bookingsystem.logic`. Här följer beskrivning av tre huvudklasser i din implementation fast det kommer att finnas fler klasser:

`Jimbo.java`

`Jimbo.java` ligger i `se.jimboagency.bookingsystem`. Klassen `Jimbo` innehåller inte mycket kod utan main metoden som använder sig av `UIManager`. `Jimbo` är ansvarig för att instansiera både `UIManager` och `LogicManager`. Referensen till `LogikManager` ska skickas till `UIManager` och sedan lagras i `UIManager`.

klassen `Jimbo` ska innehålla så lite kod som möjligt.

Note 15

`UIManager.java`

`UIManager` är en klass som ligger i `ui` paketet och är ansvarig för att kommunicera med användaren, läsa in data och skriva ut resultat/felmeddelanden.

- `UIManager` innehåller bara kod som gäller gränssnitt.
- `UIManager` innehåller inte statiska metoder fast det kan finnas undantag.
- `UIManager` instansieras i klassen `Jimbo`.
- `Jimbo` har `has-a` relation med `UIManager`(composition).

klassen `Jimbo` FÅR INTE anropa metoder som ligger i `logic` paketet direkt. Om behovet finns ska det finnas en metod i klassen `UIManager` som bara vidarebefordrar anropet från `Jimbo` till metoden som ligger i `logic` paketet. Till exempel autentisering.

Note 16

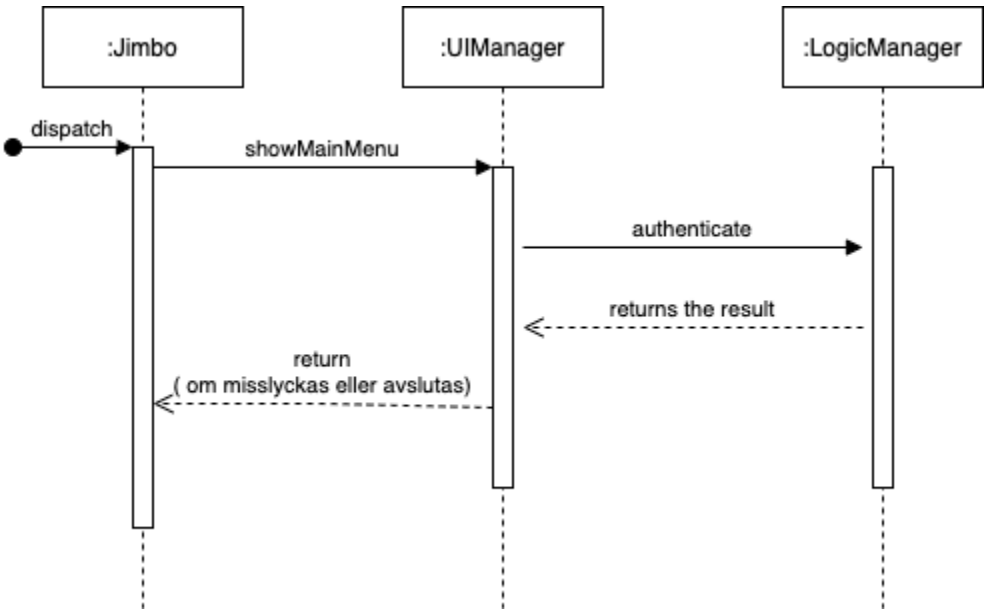
`UIManager` ska inte innehålla kod som INTE gäller gränssnitt. `UIManager` ska vända sig till klasser som ligger i `logic` paketet för att genomföra olika uppgifter. Till exempel datalagring och autentisering.

Note 17

`LogicManager.java`

`LogicManager` är en klass som ligger i `logik` paketet och är ansvarig för att tillhandahålla programmets grundfunktioner, funktionella krav.

- `LogicManager` innehåller bara kod som gäller logic såsom metoden som autentiserar användaren.
- `LogicManager` innehåller inte statiska metoder fast det kan finnas undantag.
- `LogicManager` instansieras i klassen `Jimbo` fast den används av `UIManager`. Referensen till `LogicManager` lagras i `UIManager`.
- `UIManager` har `has-a` relation med `LogicManager`(composition).



Figur 3 - `Jimbo`, `UIManager` och `LogicManager` kommunicerar med varandra.

Jimbo anropar `showMenu` som är ansvarig för att visa huvudmenyn. `showMainMenu` anropar `authenticate` innan att visa huvudmenyn. Om autentisering lyckas då välkomstmeddelandet och huvudmenyn visas annars metoden visar ett felmeddelande och returnerar kontrollen till `main` metoden i `Jimbo`.

5. Klass-hierarki

Skapa din klasshierarki baserat på funktionella kraven och följande klasser. Alla följande klasser ska finnas med i din design:

- `Flight`
- `Booking( abstract )`

- `Airline`
- `Passenger`
- `User`
- `UpdatableBooking`
- `UnUpdatableBooking`
- `LogicManager`
- `UIManager`
- `Jimbo`

Arv och komposition ska användas i designen. Klassdiagrammet för Klass-hierarkin ska visas för din handledare innan att gå vidare. Klassdiagrammet ska också lämnas in med resten av projektet i slutet av kursen.

Note 18

Det går bra med att ha fler klasser än vad som nämnts ovanpå.

Note 19

Alla ovanstående klasser förutom `UIManager` och `Jimbo` tillhör till `logik` paketet.

Note 20

Tänk på följande när du gör din design:

- Det kommer bara att finnas “tur” bokningar/resor. Retur finns ej.
- `Booking` har `has-a` relation med `Flight`.
- `UpdatableBooking` och `UnUpdatableBooking` har `is-a` relation med `Booking`.
- `Booking` får inte instanseras direkt.
- Varje klass ska ha egna konstruktorer för att initiera sina fält. Om klassen ärver fält från en annan klass då ska den anropa konstruktorer som tillhör till superklassen i sina konstruktorer för att kunna initiera ärvda fälten.
- Använd rätt åtkomst.
- Encapsulation och Abstraktion ska appliceras.
- equals() och toString() ska åsidosättas(override) och användas vid behov. Minst en gång.

  Bra läsning

  https://jenkov.com/tutorials/java/constructors.html

6.UI(User Interface)

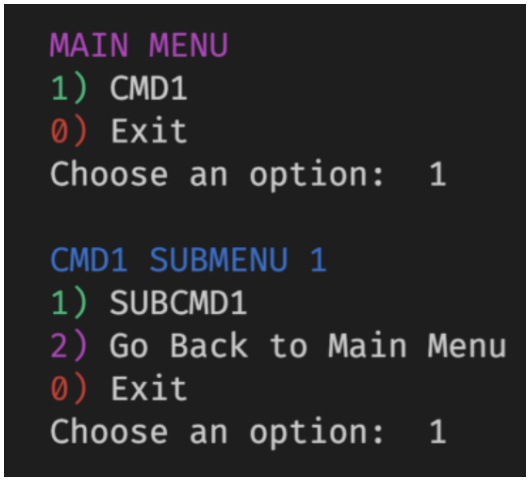
`UIManager` är ansvarig för att tillhandahålla en meny för användaren som kan användas för att göra olika funktioner. Användaren får se en meny i terminalen så fort hen startar programmet.

5.1 Huvudmeny

Så fort programmet startas visas en huvudmeny för användaren, efter ett välkomstmeddelande. Den här huvudmenyn innehåller alla huvudfunktionalitet som användaren kan göra.

5.2 Undermeny

I programmet kan det finnas undermenyer, vid behov, som visas för användaren. undermenyer visas upp efter användaren väljer ett alternativ i huvudmenyn.



Figur 4 - Exempel

5.3 Command-line argument

Command-line argument används för att läsa in användarens användarnamn och lösenord. Det ska finnas två switches:

- `-u` som följs med användarens användarnamn.
- `-p` som följs med användarens lösenord.
- Om det är fel switches då blir det felmeddelande direkt. Till exempel:
    - `Java Jimbo -u annka001 -b wedr5t`
- Om det är något switch som saknas då blir det felmeddelande direkt. Till exempel:
    - `Java Jimbo -u annka001`
- Om det är fel användarnamn och/eller lösenord då blir det felmeddelande direkt. Till exempel:
    - `Java Jimbo -u annka001 -p wedr2s`

7. Felhantering

Det ska finnas åtgärder för alla potentiella exception som kan kastas av de använda metoderna från standard biblioteket.

Alla felmeddelande visas från `UIManager`.

Note 21

Att ha **bara** ett generellt catch-block ska undvikas.

Note 22

Det är viktigt att vända sig till Javadoc för att utreda vilka exception kastas av varje använda metod.

Note 23

Till exempel används `Scanner:nextInt()` i klassen UIManager. Enligt dokumentationen kan metoden kasta tre olika fel.

nextInt

`public int nextInt()`

Scans the next token of the input as an `int`.

An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.

Returns:

the `int` scanned from the input

Throws:

`InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range

`NoSuchElementException` - if input is exhausted

`IllegalStateException` - if this scanner is closed

https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html#nextInt()

Första felet ska hanteras annorlunda jämfört med dem två andra. Om det är fel datatyp då ska användaren få försöka om annars ska alla data lagras och sedan programmet stängs.

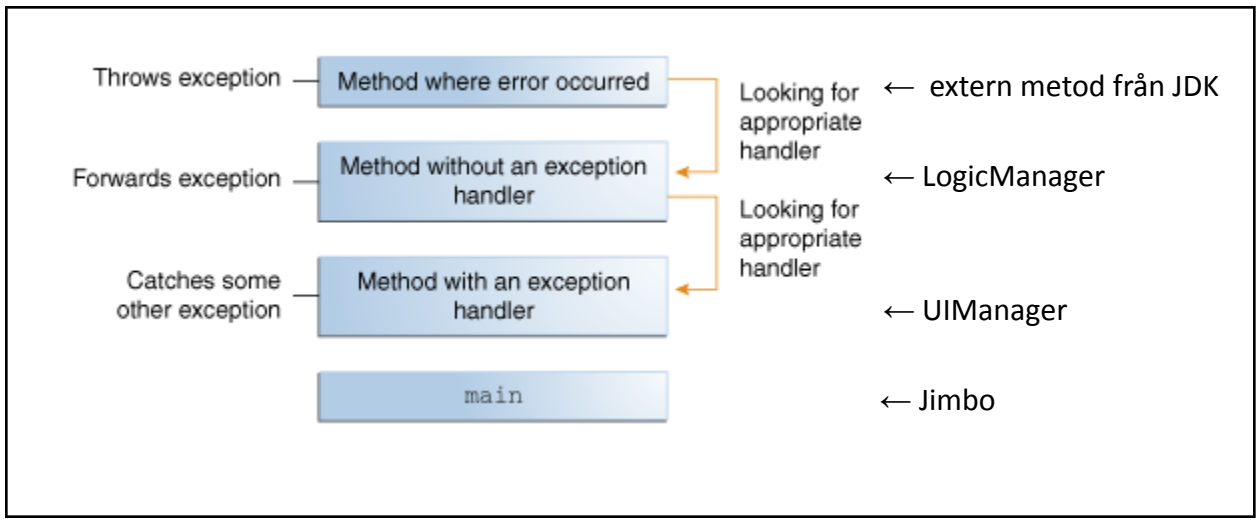
```
try {
    Scanner input = new Scanner(System.in);
    int a = input.nextInt();
}
catch (InputMismatchException e){
    // Let the user know by writing  an error message
    // let the user try again
}
catch (NoSuchElementException e){
    // Let the user know by writing  an error message
```

```
    // store data on the disk
}
catch (IllegalStateException e){
    // Let the user know by writing  an error message
    // Store data on the disk
}
```

Som du har märkt innehåller andra och tredje catch-blocken samma lösning. I det här fallet är det bra att ha ett generellt catch-block i stället. Det här generella catch-blocket ska alltid ligga sist.

```
try {
    Scanner input = new Scanner(System.in);
    int a = input.nextInt();
}
catch (InputMismatchException e){
    // Let the user know by writing  an error message
    // let the user try again
}
catch (Exception e){
    // Let the user know by writing  an error message
    // store data on the disk by calling the store method in LogicManager.
}
```

Om det är något fel i `logik` paketet som inte kan hanteras då ska exceptionet vidarebefordras till `UIManager`. I `UIManager` ska exceptionet hanteras och ett felmeddelande visas för användaren.



Figur 5 - när `LogicManager` kan inte hantera felet och vidarebefordrar det till `UIManager`.

8. Datastrukturer

Använd en eller flera lämpliga datastrukturer för att lagra data så som flyg, resor och användare i primärminnet medan programmet är igång. Se gärna nedan:
https://www.geeksforgeeks.org/data-structures/

Det är upp till dig vilka datastrukturer ska användas. En del data är hårdkodade i programmet så som användare, städer och flygbolag.

> Note: I basversionen kommer data som kan manipuleras av användare, flyg och resor, att förloras så fort programmet avslutas.

Note 24

> Alla datastrukturer ska deklareras i `LogicManager`.

Note 25

9. Dokumentation

Koden ska vara dokumenterad noggrant. Det ska finnas beskrivning för varje eget fält, metod, interface, klass osv.

Exempel:

```
/**
 * The Foo class is a silly example to illustrate documentation
 * comments.
 */
public class Foo {
   /**
    * An integer to keep track of for fun.
    */
          private int count;
         ...
   /**
    * Increment a value by delta and return the new value.
    *
    * @param  delta   the amount the value should be incremented by
    * @return         the post-incremented value
    */
   int increment(int delta) {
       ...
   }
}
```

Bra läsning:
https://www.clear.rice.edu/comp310/JavaResources/comments.html

10. Högre betyg

Om du siktar på högre betyg då ska fler funktioner, som har beskrivits nedan, implementeras. Betygsättning på projektet sker som följande:

Betyg	Funktionalitet
3	Basfunktioner
4	Basfunktioner OCH (Datalagring ELLER GUI)
5	Basfunktioner OCH Datalagring OCH GUI

10.1 Datalagring

Alla flyg och resor ska lagras permanent i json-filer genom att använda GSON biblioteket.

- Alla flyg ska lagras i en fil när användaren avslutar programmet. Filen heter `flights.json` .
- Alla flyg ska laddas från `flights.json` till någon datastruktur så fort programmet startas.
- Alla bokade resor ska lagras i en fil när användaren avslutar programmet. Filen heter `bookings.json` .
- Alla bokade resor ska laddas från `bookings.json` till någon datastruktur så fort programmet startas.
- Alla andra data är hårdkodade i programmet.

Note: genom att använda felhantering ska alla flyg och resor lagras i fall programmet kraschar.

Note 26

Vad är json?

Json är ett format som används för data-exchange och lagring. Se gärna följande för mer information:
https://www.json.org/json-en.html

Vad är GSON?

GSON är ett bibliotek från Google som används för att jobba med json-filer. Se gärna följande för mer information:
https://www.tutorialspoint.com/gson/gson_first_application.htm

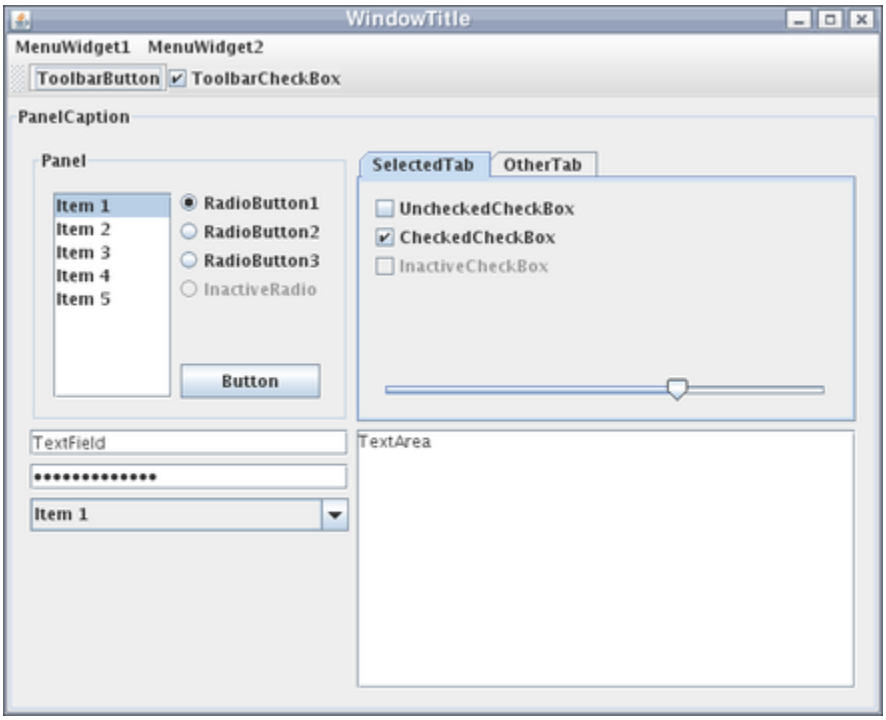
GSON behövs laddas ner som jar-fil och sedan läggas till i Eclipse. Här kan ni ladda ner gson jar-fil:
https://search.maven.org/artifact/com.google.code.gson/gson/2.9.1/jar

10.2 Graphical User Interface

Nu är det dags att lägga till ett nytt funktionalitet baserat på GUI som kan väljas i huvudmenyn. Det nya funktionalitetet handlar om att se statistiken på alla resor under en viss vecka. Användaren ska kunna välja år och vecka och sedan se alla bokningar som gjorts under den valda veckan. GUI:n ska implementeras i Java Swing.

Det är bara det nya funktionalitetet som ska göras i GUI.

Note 27



Figur 6 - Exempel GUI i Java Swing

Implementationen skall vara helt i Java utan att använda något designverktyg.

Note 28

I din implementation använd lämpliga komponenter som passar implementerade funktionalitetet.

Note 29

Alla inmatningar och utmaningar som gäller det här funktionalitetet ska ske genom GUI:n.

Note 30

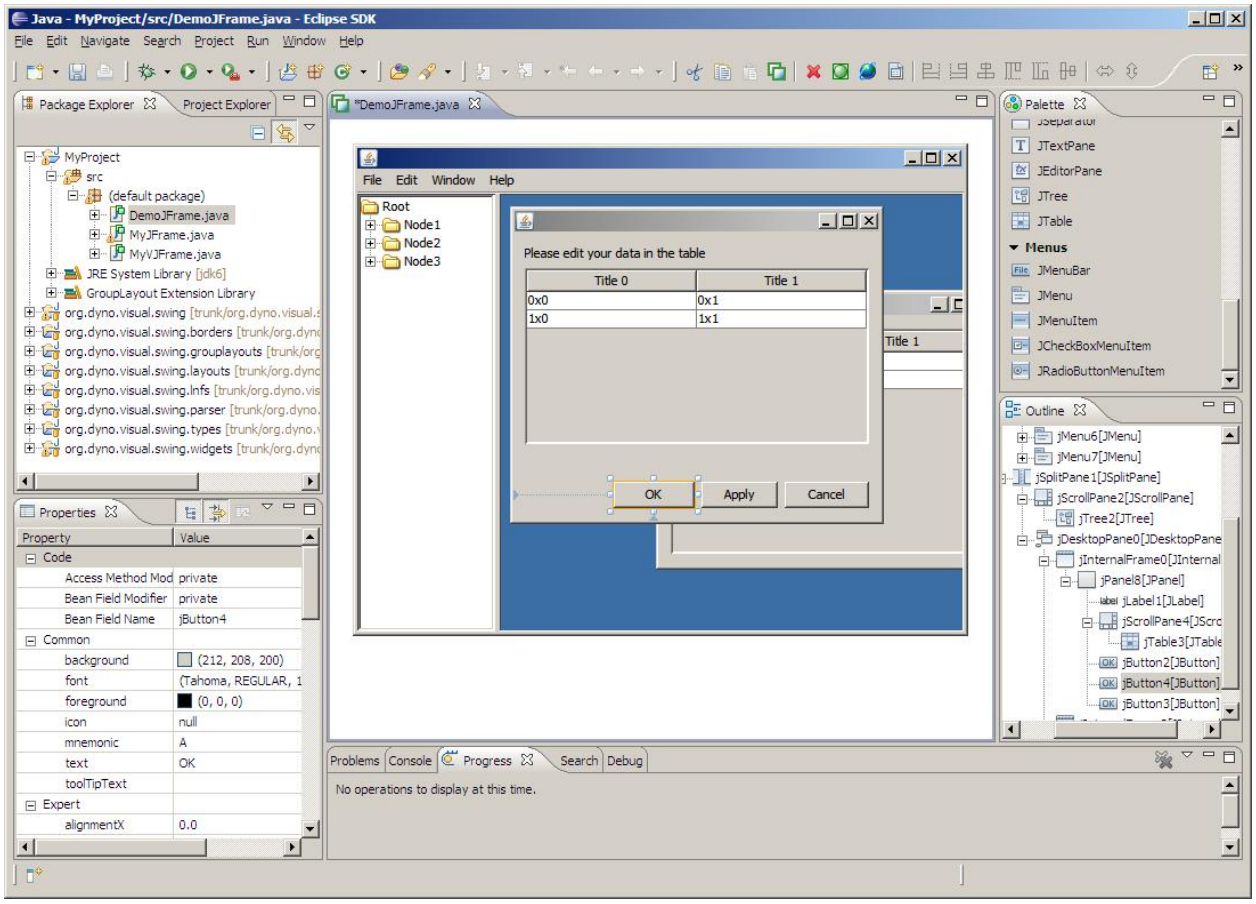


Figure 7 - Exempel på designverktyg som INTE får användas

Swing Tutorial

https://docs.oracle.com/javase/tutorial/uiswing/

Lycka till!

Sahand Sadjadee