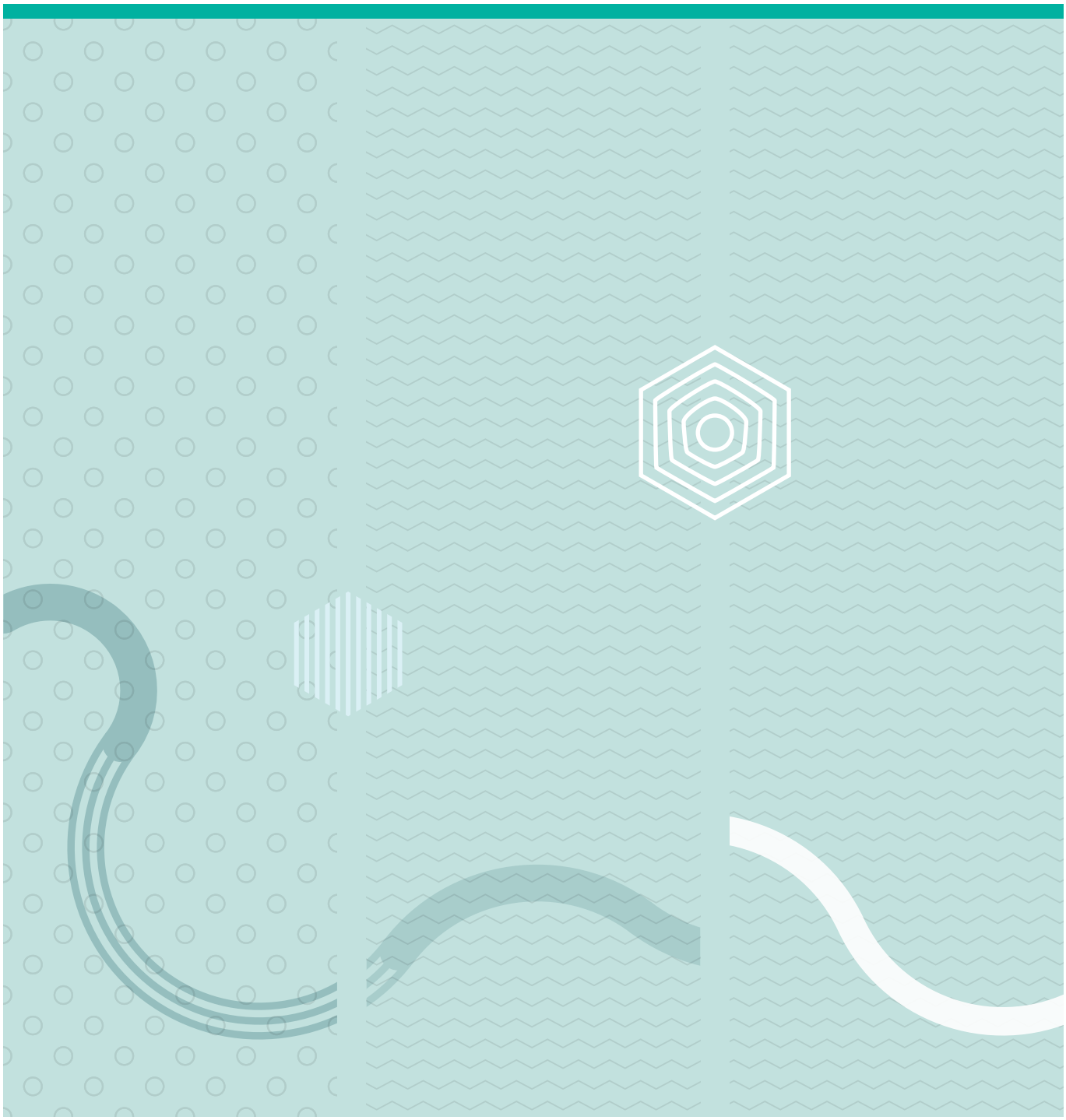
Universitetet i Sørøst-Norge
—
Skriftserien nr. 41
2020

JavaFX

Med Eclipse og Scene Builder

Trond R. Braadand



Trond R. Braadland

JavaFX

Med Eclipse og Scene Builder

Universitetet i Sørøst-Norge
Hønefoss, 2020

Skriftserien fra Universitetet i Sørøst-Norge nr. 41

ISSN: 2535-5325 (Online)
ISBN: 978-82-7860-432-8 (Online)

Forord

Dette kompendiet er skrevet for studenter på kurset OBJ2100 – objektorientert programmering 2 – ved Universitetet i Sørøst-Norge, studiested Ringerike. Kurset er på 7,5 studiepoeng. Studentene har tidligere hatt 15 studiepoeng men grunnleggende programmering med Python, og 7,5 studiepoeng med objektorientert programmering med Java i kurset OBJ2000 – objektorientert programmering 1. Parallelt med OBJ2100 går kurset OAD2000 – objektorientert analyse og design. Dette kurset er først og fremst et kurs i bruk av UML. Siden samme lærer er ansvarlig for begge kursene på Ringerike, har de til en viss grad blitt flettet sammen.

I begge kursene i objektorientert programmering brukes Eclipse som utviklingsmiljø. Kompendiet har derfor som utgangspunkt at det er denne plattformen som brukes. Overføringsverdien til andre plattformer, som IntelliJ, burde være stor.

Bakgrunnen for at jeg har laget dette kompendiet er at lærebøker i Java, inkludert Y. Daniel Liangs lærebok[1], som er pensum i kurset hos oss, ikke går langt nok i presentasjonen av JavaFX. Dette gjelder spesielt bruken av tabeller for presentasjon av data, men også bruk av flere scener i samme applikasjon. Videre har jeg ønsket å presentere Scene Builder som en ekstra plattform for design av grafiske brukergrensesnitt. Dette er heller ikke behandlet i lærebøkene, iallfall ikke i det omfang jeg ser som ønskelig.

Selv om bruken av tabeller og Scene Builder er sentrale i kompendiet, har jeg også med en del elementære eksempler. I tillegg er mange av grensesnittene illustrert med UML klassediagrammer.

Trond R. Braadland,

Førstelektor USN

[1] Y. Daniel Liang: Introduction to Java Programming and Data Structures. 11th edition, Pearson 2019

Innhold

Introduksjon til JavaFX

JavaFX er foreløpig (mai 2020) ikke med i Java SDK SE. Den er derfor heller ikke inkludert i Eclipse. Det finnes imidlertid en plug-in for Eclipse, kalt e(fx)clipse. Denne er enkel å installere. Mer om det senere.

JavaFX er et bibliotek for å lage grafiske brukergrensesnitt. Det er det biblioteket Oracle (som eier Java) vil satse på fremover. JavaFX er imidlertid ett av flere grafiske biblioteker for Java. Det vil være nyttig å begynne med å se på disse.

Grafiske biblioteker i Java

Programmeringsspråket Java ble utviklet av Sun Microsystems og lansert i 1995. Siden det er Java overtatt av Oracle. Tidligere grafiske biblioteker for Java er disse:

- AWT: Forkortelse for Application Window Toolkit. Dette biblioteket ble lansert med java versjon 1.1 i 1997 (versjonsnummereringen av de første utgavene av Java er litt forvirrende. De fem første versjonene ble nummerert 1.0, 1.1, 1.2, 1.3 og 1.4, deretter ble de nummerert Java 5, Java 6 og så videre). Den aller første java-versjonen, 1.0, hadde et nokså rudimentært grafisk bibliotek. AWT er basert på bruk av operativsystemets API. Dette betyr at hvis man programmerer en avkryssingsboks i Java (ved bruk av AWT-klassen CheckBox), vil programmet ved kjøring kalle operativsystemets innebygde rutine for å lage en avkryssingsboks (det er disse rutinene som utgjør operativsystemets API). Siden disse rutinene er forskjellige for eksempel for Windows og MacOS, vil grensesnittet se forskjellig ut avhengig av operativsystemet.
- Swing: introdusert med Java versjon 1.2 i 1997. Fortsatt er det operativsystemets API som brukes, men nå med et «lag» over som gjør at en grafisk applikasjon ser likeens ut uavhengig av operativsystemet. Faktisk er mange av klassene i Swing subklasser av klasser i AWT:
- Java FX: Dette biblioteket har vært under utvikling siden 2008. Det har ikke vært en del av en standard Java installasjon før Java versjon 11. FX skal erstatte Swing, i den forstand at Swing ikke vil bli videreutviklet. Swing vil fortsatt være en del av Java, akkurat som AWT fortsatt er det.

Elementer i et grafisk bibliotek

Uansett programmeringsspråk er det mange fellestrekk når det gjelder programmering av grafiske brukergrensesnitt:

Man må kunne kontrollere plasseringen av de forskjellige komponentene i brukergrensesnittet.

Programmet må fange opp hendelser forårsaket av brukeren, for eksempel et museklikk på en knapp.

Man må kunne kontrollere egenskaper som farger og fonter.

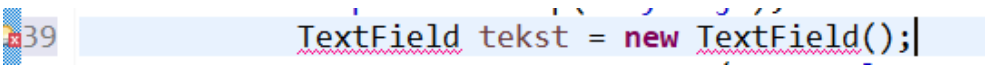
Man må ha tilgang til en rekke standard komponenter som brukes til å bygge opp brukergrensesnittet:

- Knapper
- Menyer
- Tekstfelt
- Tabeller
- Kombobokser
- Avkrysningsfelter og radioknapper
- Rullefelt
- Og mye mer

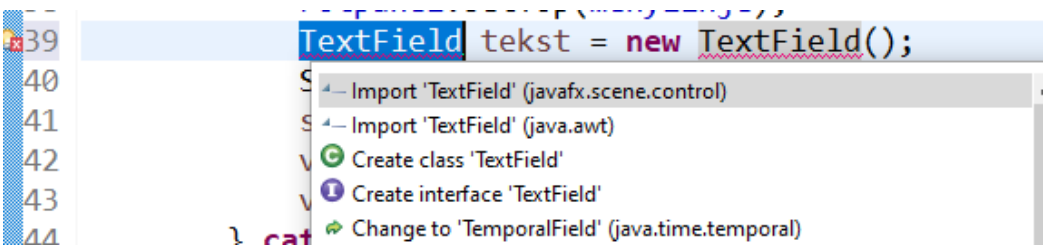
Vær oppmerksom på at noen klasser har samme navn i JavaFX som i awt. Klassene er imidlertid forskjellige, og må ikke blandes. Det er derfor viktig å kontrollere import-setningene Eclipse genererer. Her er et eksempel:

```
 5 import javafx.application.Application;
 6 import javafx.collections.FXCollections;
 7 import javafx.collections.ObservableList;
 8 import javafx.scene.Scene;
 9 import javafx.scene.control.Button;
10 import javafx.scene.control.Label;
11 import javafx.scene.control.TableColumn;
12 import javafx.scene.control.TableView;
13 import javafx.scene.control.TextField;
14 import javafx.scene.control.cell.PropertyValueFactory;
15 import javafx.scene.layout.BorderPane;
16 import javafx.scene.layout.GridPane;
17 import javafx.stage.Stage;
18 import oblig_del1.Kontroll;
```

Eclipse varsler når en klasse må importeres ved å vise en rød «dot» foran linjenummeret:



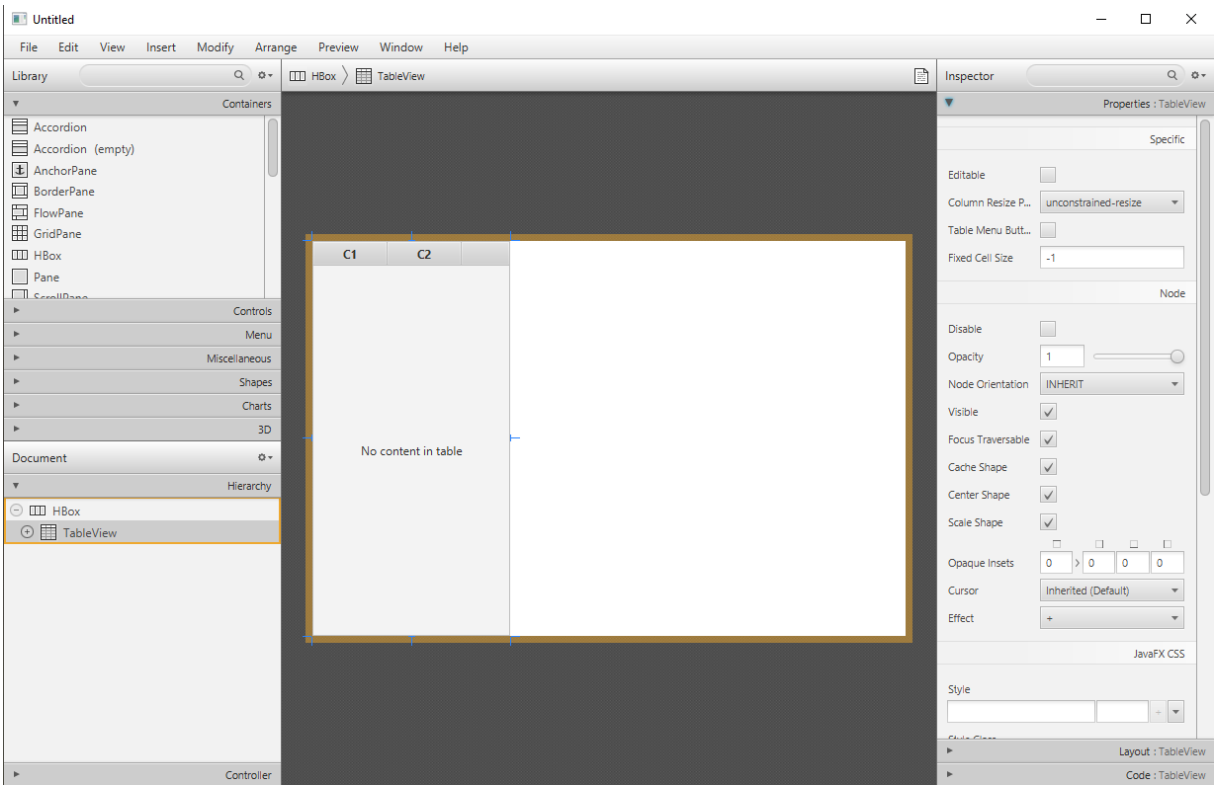
Når vi klikker på «dotten», foreslår Eclipse importen:



Her ser vi at det finnes en klasse TextField både i JavaFX (javafx.scene.contol) og i awt (java.awt). I en JavaFX applikasjon er det alltid den fra JavaFX vi velger.

Styring av brukergrensesnittets layout med FXML

Med moderne grafiske utviklingsverktøy kan vi bygge opp et grensesnitt ved å velge komponenter fra en verktøykasse og deretter plassere dem i et vindu. Nedenfor er vist en skjermdump fra Scene Builder, som er Oracles IDE for JavaFX:

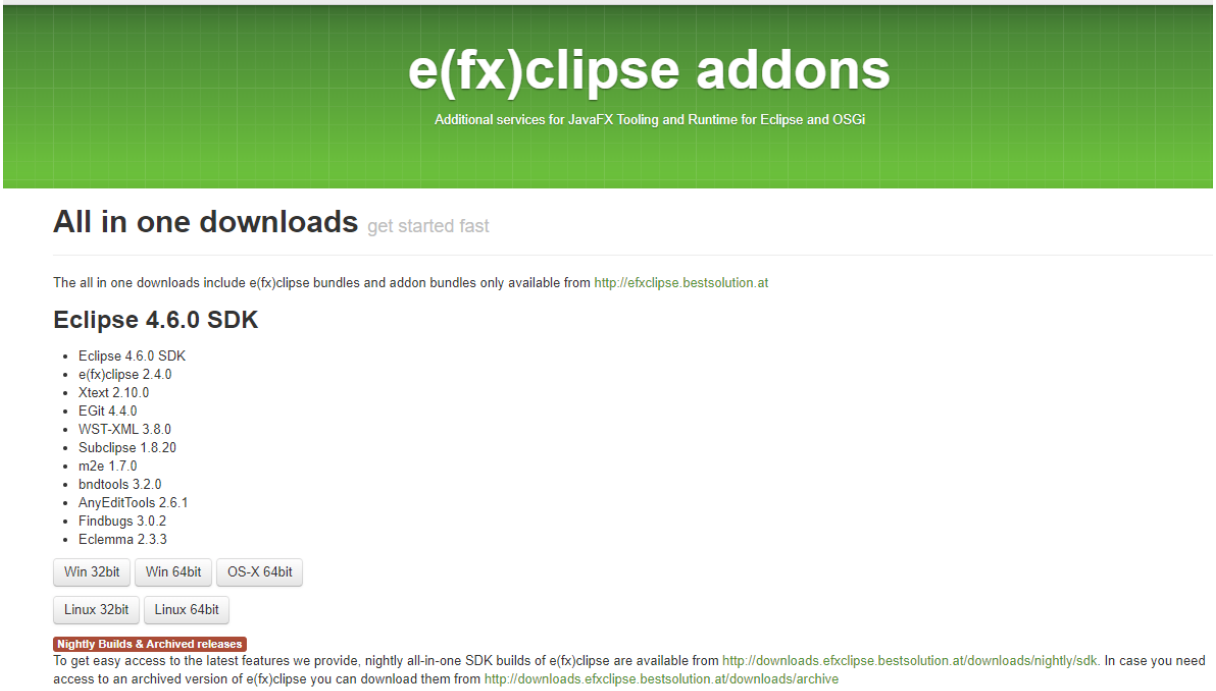


Scene Builder kan brukes med både Eclipse, IntelliJ, NetBeans og JDeveloper. Scene Builder kan også kjøres som et stand-alone utviklingsverktøy. Med Scene Builder lages grensesnittet med FXML, som er en variant av XML, i stedet for hardkoding med JavaFX.

Installere JavaFX i Eclipse

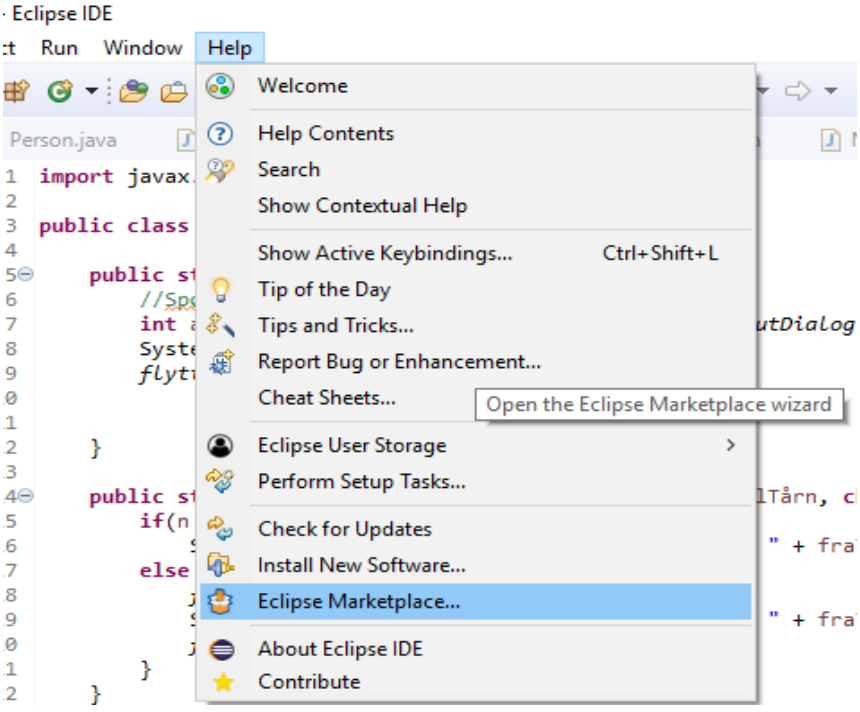
Det enkleste er å laste ned en komplett pakke med Eclipse og e(fx)clipse ferdig installert. Du finner den på denne adressen: https://efxclipse.bestsolution.at/install.html#all-in-one

Det er ikke den nyeste versjonen av Eclipse, men det spiller liten rolle. Her er hva som omfattes i pakken:

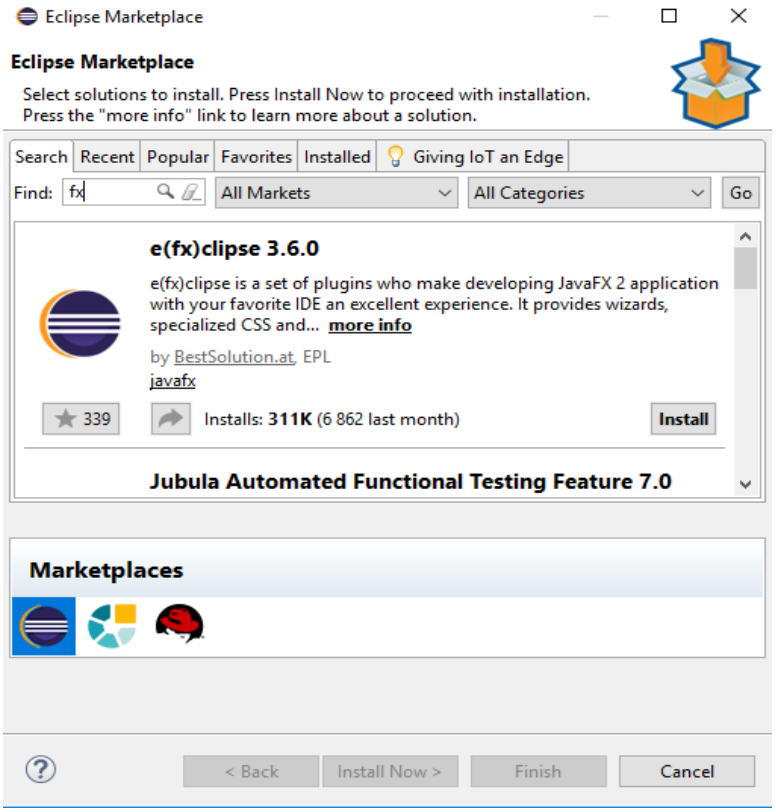


Ønsker du å bruke din eksisterende Eclipse, kan du installere en plug-in kalt e(fc)clipse.

Gå til Help og velg Marketplace eller Install new Software:



Søk på FX, og Eclipse plug-in for Java FX kommer opp:



Klikk Install.

Etter installasjonen (omstart av Eclipse kreves) kan vi nå velge å starte et JavaFX-prosjekt.

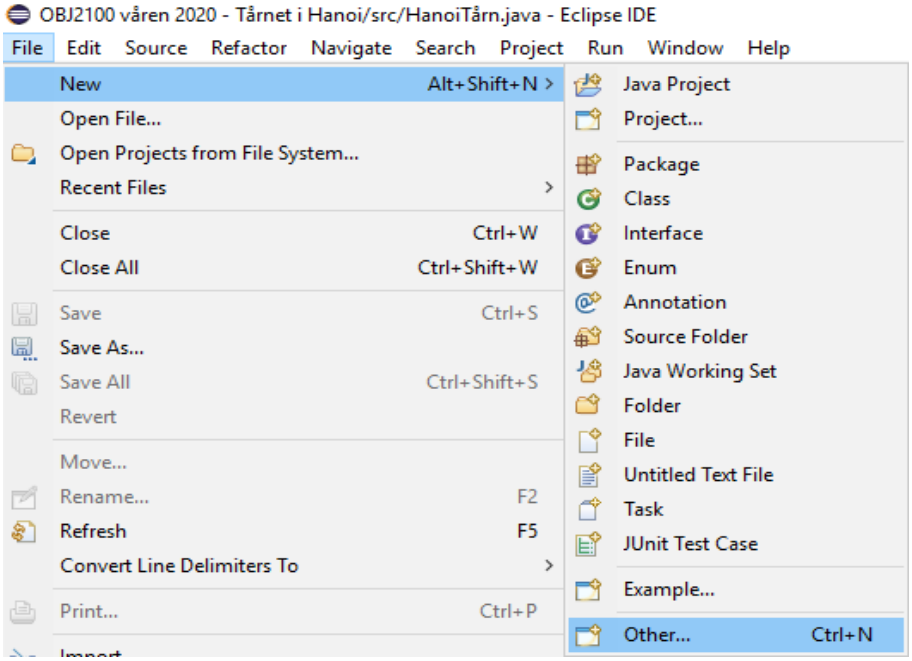
Med denne måten å gjøre det på, har du alltid siste versjon av programmene.

Et alternativ er å installere den offisielle JavaFX separat. Dette gir ikke den samme integrasjonen med Eclipse som e(fx)clipse gjør, men kan være et alternativ hvis det er et problem med sistnevnte. Se her for beskrivelse:
https://stackoverflow.com/questions/52144931/how-to-add-javafx-runtime-to-eclipse-in-java-11

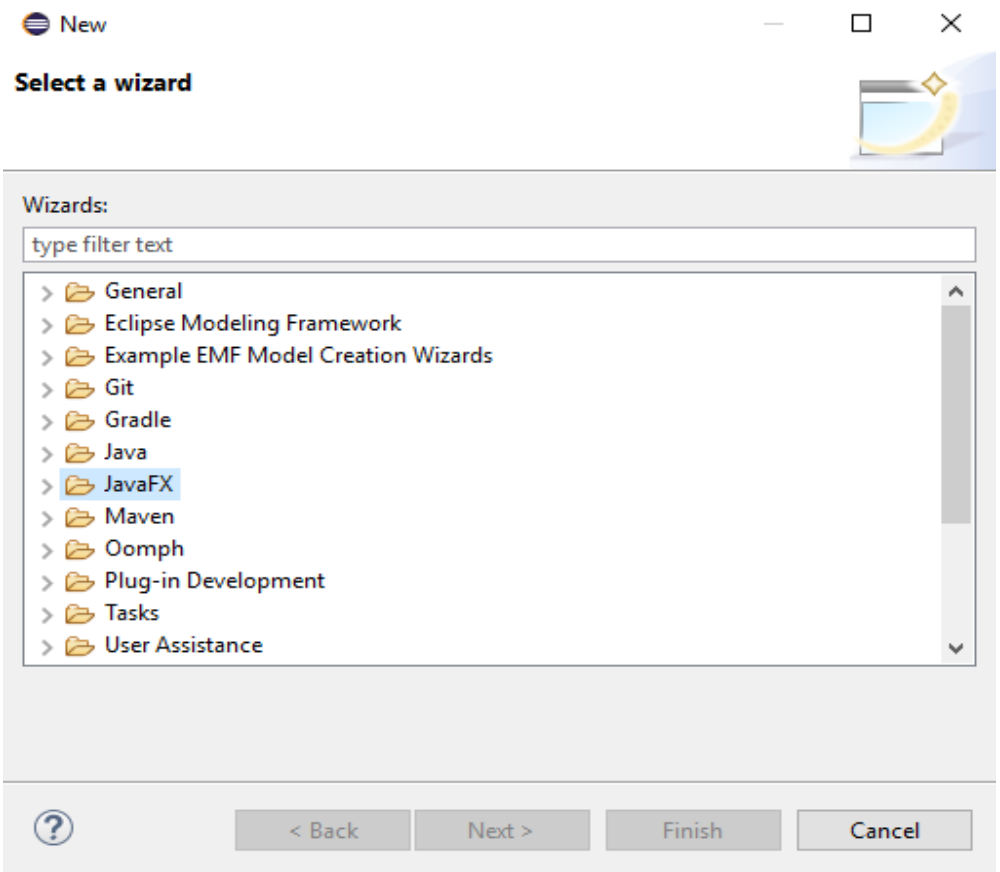
Det er også mulig å laste ned en Eclipse-versjon der e(fx)clipse allerede er installert. Dette vil imidlertid okke være den nyeste utgaven. Versjonen kan lastes ned her:
https://efxclipse.bestsolution.at/install.html#all-in-one

Opprette et JavaFX-prosjekt

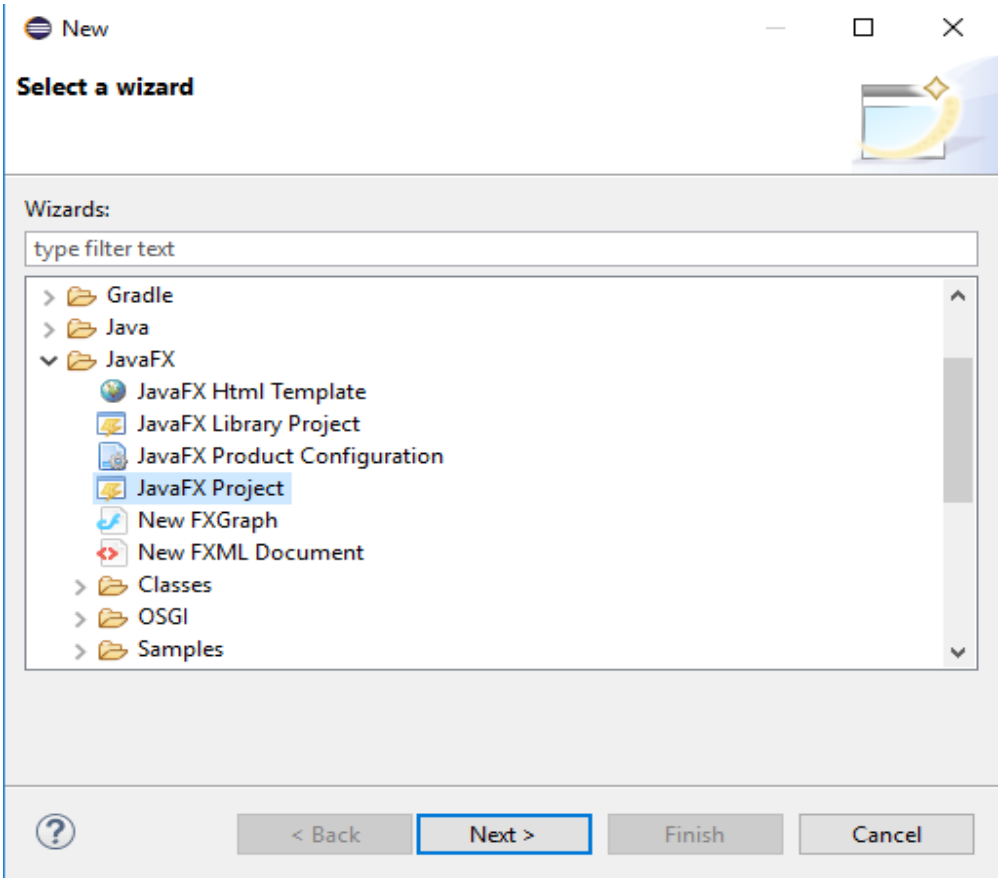
I stedet for å opprette et Java prosjekt, velger vi Other:



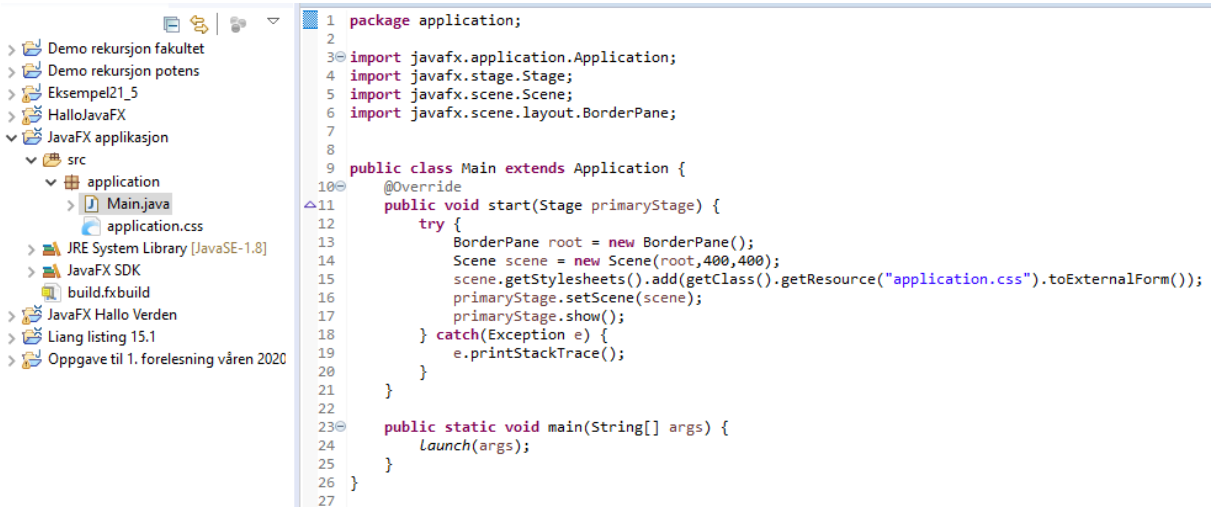
Vi ser at JavaFX er lagt til:



Velg JavaFX Project:



Når prosjektet er opprettet, vil også en grafisk applikasjon med navnet Main være opprettet. Den ser slik ut:



Prosjektet er her kalt JavaFX applikasjon. Vi ser at under scr er en pakke kalt application (automatisk generert), og at denne inneholder en klasse Main og en CSS-fil kalt application.css. Den siste skal vi foreløpig ikke bry oss om. Vi skal se nærmere på klassen Main.

Her er hva denne koden betyr:

Linje 1: pakken application opprettes automatisk.

Linjene 3 til og med 6: importsetningene genereres automatisk.

Linje 9: Et vindu er en subklasse av FX-klassen Application.

Linje 11: metoden start() genereres, med et objekt av klassen Stage («oppsetning»)som parameter.

Linje 12: en try..catch blokk settes opp.

Linje 13: Her lages en layoutmanager for å kontrollere plassering av elementer.

Linje 14: Et objekt av klassen Scene («scene») opprettes.

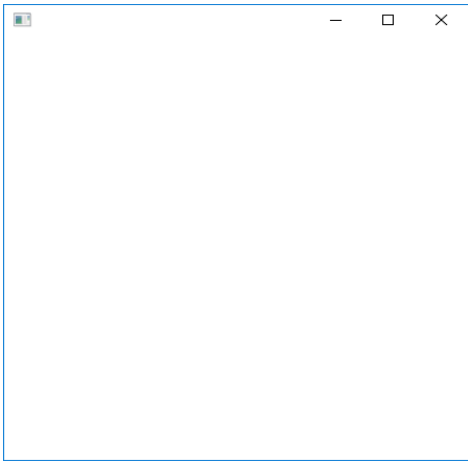
Linje 15: Her henter Scene-objektet en referanse til CSS.

Linje 16: oppsetningen setter scenen.

Linje 16: Oppsetningen vises.

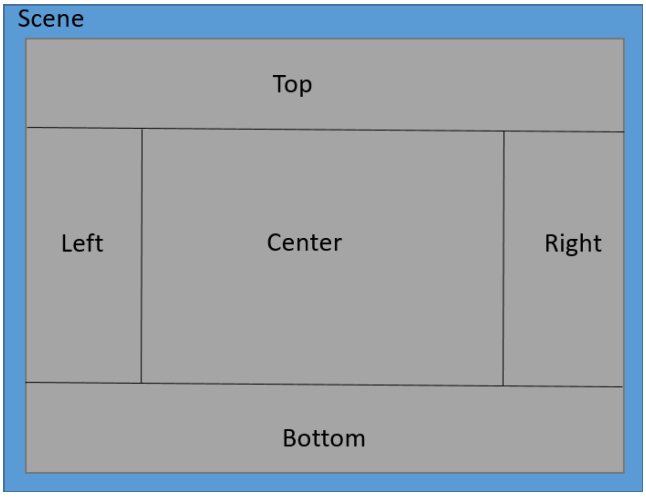
Linjene 23 – 25: main()-metoden kaller metoden launch(), som igjen kaller srart().

Hvis vi kjører dette programmet, lages et vindu som ser slik ut:



Det er foreløpig ingen synlige elementer i vinduet. Det skal vi forandre på.

Vi skal utvide programmet til å bli det tradisjonelle «Hello World». For å få til det skal vi legge til en Label med tekst. Vi må imidlertid også fortelle programmet hvor denne skal plasseres. Default layout for JavaFX kales BorderPane. Når den er lagt til en scene, ser den slik ut (det vil si, vi ser den ikke, men det er slik den er organisert internt):



Nå kan vi utvide koden fra JavaFX applikasjon til å skrive en hilsen:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.stage.Stage;
5  import javafx.scene.Scene;
6  import javafx.scene.control.Label;
7  import javafx.scene.layout.BorderPane;
8
9
10 public class Main extends Application {
11     @Override
12     public void start(Stage primaryStage) {
13         try {
14             BorderPane root = new BorderPane();
15             Scene scene = new Scene(root,400,400);
16             Label hilsen = new Label();
17             hilsen.setText("Hallo der!");
18             root.setCenter(hilsen);
19             scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
20             primaryStage.setScene(scene);
21             primaryStage.show();
22         } catch(Exception e) {
23             e.printStackTrace();
24         }
25     }
26
27     public static void main(String[] args) {
28         launch(args);
29     }
30 }
```

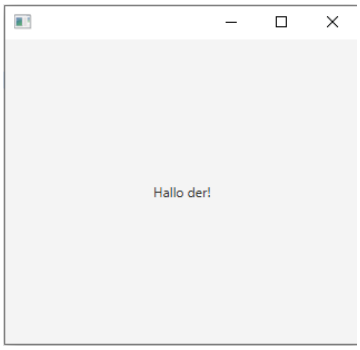
Utvidelsene av programmet:

Linje 16: Et objekt av klassen Label opprettes.

Linje 17: Teksten til objektet settes.

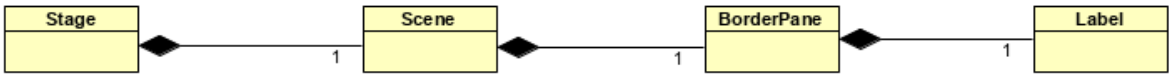
Linje 18: Label'en legges til i center i BorderPane.

Vinduet ser nå slik ut:



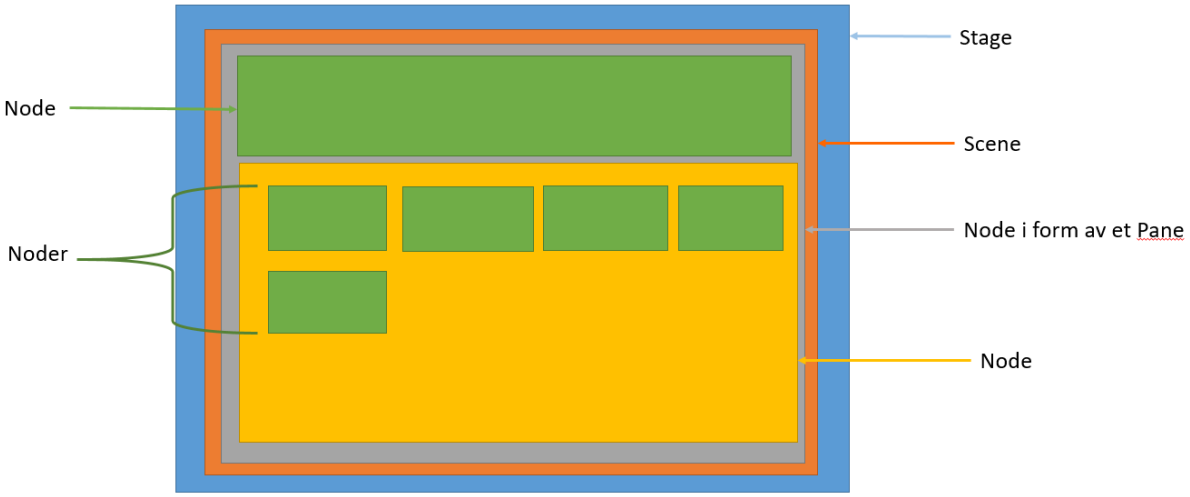
Label'en med teksten er en node i vinduet.

Et klassediagram for programmet ser slik ut:



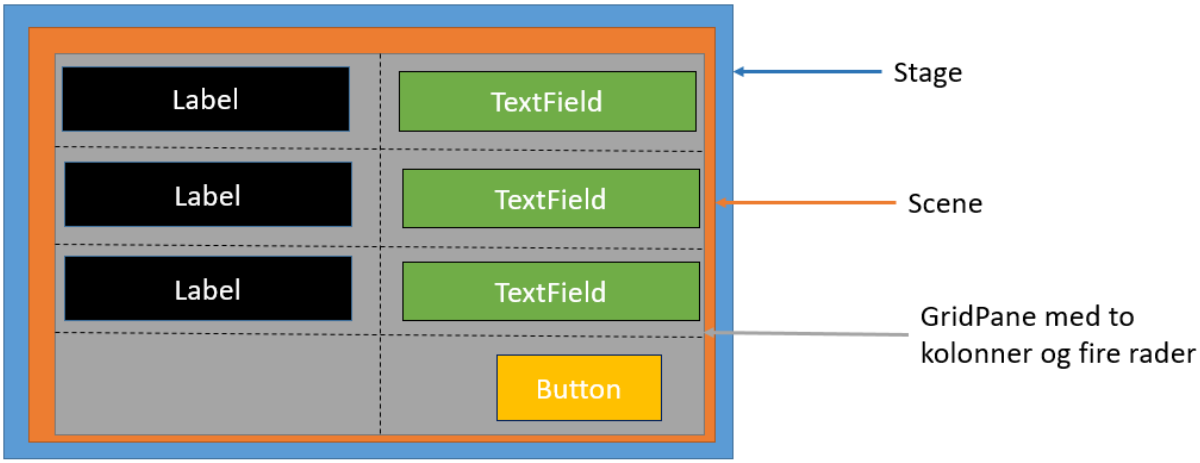
Oppbygningen av en JavaFX vindu

JavaFX bruker en teater-metafor for oppbygningen av vinduer. Et vindu består av en Stage (vi kan kanskje kalle det en «oppsetning» på norsk; som oppsetningen av et teaterstykke). I en oppsetning plasserer vi en scene, og i denne igjen kan vi plassere noder. Dette kan vises slik:



Nodene er alle elementene vi kan plassere i vinduet.

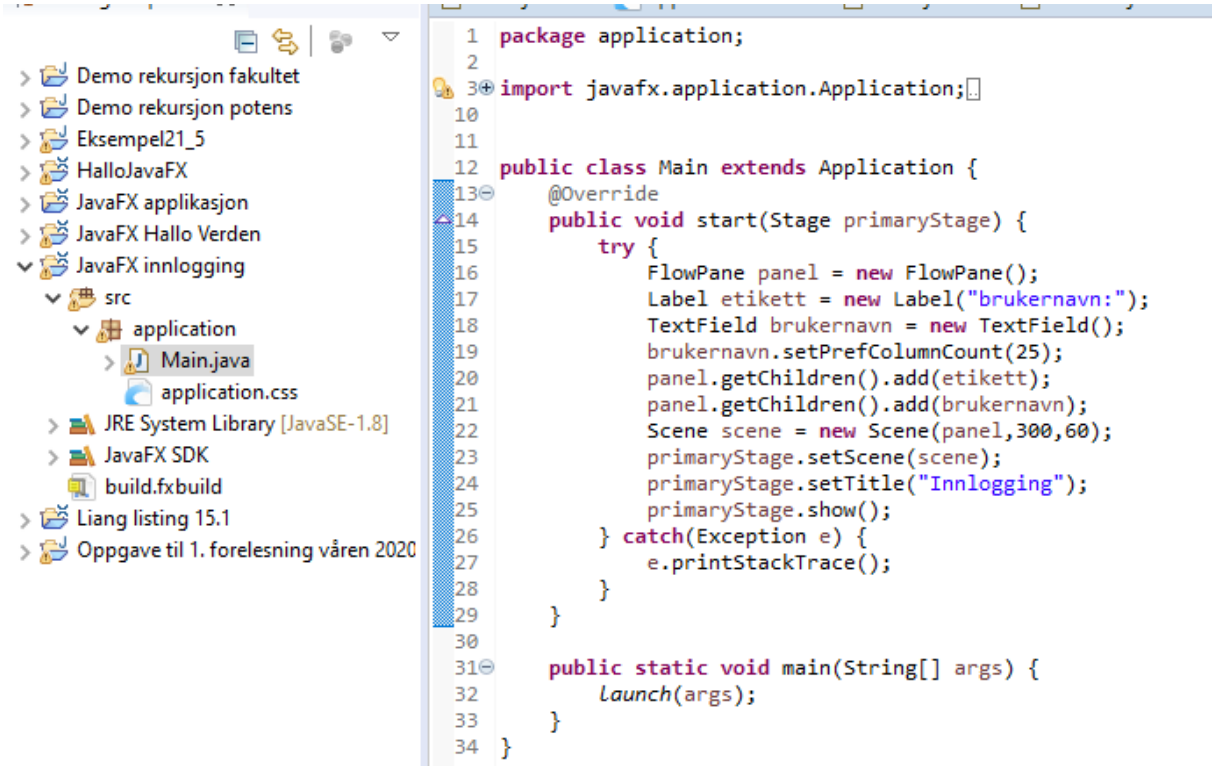
Nedenfor er vist oppbygningen av et vindu med noder, bestående av en node i form av et objekt av GridPane, som igjen inneholder noder i form av objekter av Label, TextField og Button:



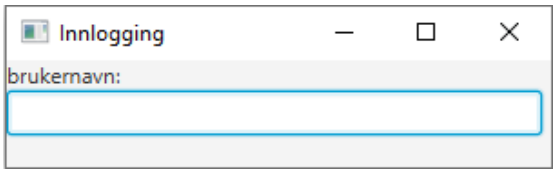
Legg merke til noden GridPane. Denne styrer oppsettet (layouten) av vinduet, i dette tilfellet et rutenett. De såkalte layoutpanelene er viktig for å forstå oppbygningen av vinduer.

FlowPane

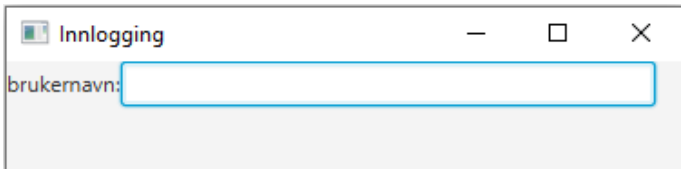
Layoutpanelet FlowPane organiserer nodene slik at de legges inn etter hverandre. Nedenfor er et enkelt eksempel der default-manageren BorderPane er byttet ut med FlowPane, og vi legger til en label og et tekstfelt:



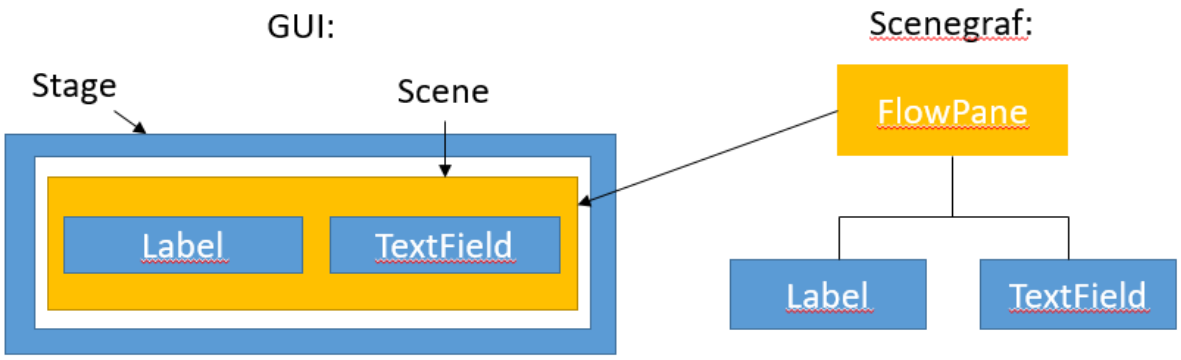
Når vi kjører dette programmet, får vi dette vinduet:



Drar vi i kanten av vinduet for å øke bredden, faller nodene på plass slik:



GUI'et er bygget opp på denne måten:



Til venstre vises GUI'et med sine komponenter. Til høyre vises en scenegraf med nodene som er lagt inn i scenen. Vi ser at nodene her representerer et tre.

FlowPane-objektet utgjør roten i det treet som er scenegrafen. Som vi kan se av programeksemplene, skal konstruktøren til klassen Scene ha et layoutpanel som parameter.

Styring av oppsett

Verktøy Delphi og .NET er laget spesifikt for Windows, og det genereres automatisk kode for plasseringen av komponentene i vinduet; kode vi ikke kan se. Siden Java er plattformuavhengig, må vi i utgangspunktet selv programmere plasseringen. Dette gjør vi ved å bruke klasser som kalles Layout Managers. Java har en rekke av disse klassene. En layout manager kommuniserer igjen med det underliggende operativsystemets API.

Default i Java og JavaFX er en layout manager som kalles BorderLayout i Swing og BorderPane i JavaFX. Denne deler opp vinduet som vist nedenfor:

Vi kan nå plassere komponenter i vinduet ved å spesifisere i hvilken av de fem delene en komponent skal være. Mer om dette senere.

Andre layout managers i JavaFX er disse:BorderPane

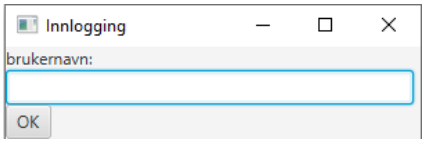
- HBox
- VBox
- StackPane
- GridPane
- FlowPane
- TilePane
- AnchorPane

Vi skal se på noen av disse senere.

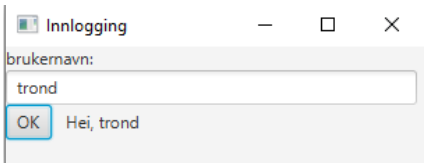
Håndtering av hendelser

En hendelse er for eksempel et klikk på en knapp. Vi skal utvide programeksemplet Innlogging med en knapp og en ekstra label. Når vi har skrevet inn navnet vårt og klikker på knappen, skal programmet skrive ut en hilsen. Programmet ser slik ut:

Før vi gjør noe:



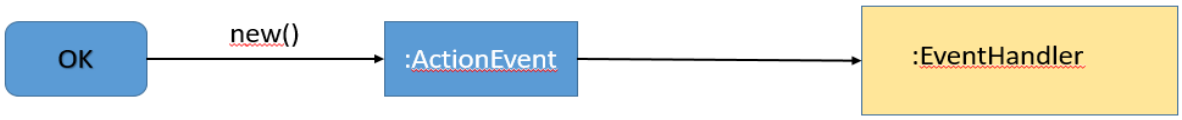
Etter at navn er skrevet inn og knappen klikket:



Koden for dette programmet ser slik ut:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.event.ActionEvent;
5  import javafx.stage.Stage;
6  import javafx.scene.Scene;
7  import javafx.scene.control.Button;
8  import javafx.scene.control.Label;
9  import javafx.scene.control.TextField;
10 import javafx.scene.layout.BorderPane;
11 import javafx.scene.layout.FlowPane;
12
13
14 public class Main extends Application {
15     private Button knapp;
16     private Label etikett;
17     private TextField brukernavn;
18     private Label hilsen;
19     @Override
20     public void start(Stage primaryStage) {
21         try {
22             FlowPane panel = new FlowPane();
23             panel.setHgap(10);
24             knapp = new Button("OK");
25             knapp.setOnAction(e -> behandleKlikk(e));
26             etikett = new Label("brukernavn:");
27             brukernavn = new TextField();
28             hilsen = new Label();
29             brukernavn.setPrefColumnCount(25);
30             panel.getChildren().add(etikett);
31             panel.getChildren().add(brukernavn);
32             panel.getChildren().add(knapp);
33             panel.getChildren().add(hilsen);
34             Scene scene = new Scene(panel,300,60);
35             primaryStage.setScene(scene);
36             primaryStage.setTitle("Innlogging");
37             primaryStage.show();
38         } catch(Exception e) {
39             e.printStackTrace();
40         }
41     }
42
43     public void behandleKlikk(ActionEvent event) {
44         hilsen.setText("Hei, " + brukernavn.getText());
45     }
46
47     public static void main(String[] args) {
48         launch(args);
49     }
50 }
```

Det som skjer når vi klikker på knappen, er at denne lager et hendelsesobjekt. For en knapp er dette objektet av klassen ActionEvent. Dette objektet «fanges opp» av en «hendelseshåndterer», dersom denne er knyttet til knappen. Hendelseshåndtereren implementerer interfacet EventHandler. Vi kan illustrere denne prosessen slik:



I programkoden realiseres dette på denne måten:

- Vi skriver en hendelsesmetode med et ActionEvent-objekt som parameter. Denne metoden skal kjøres når et ActionEvent-objekt lages av knappen
- For at hendelsesmetoden skal fange opp ActionEvent-objektet, må den knyttes til knappen.

Her er et eksempel på en hendelseshåndterer:

```
//Hendelseshåndterer:
public void behandleKlikk(ActionEvent e) {
    System.out.println("Du registrerte: " + txtNavn.getText() + ", " + txtAdresse.getText() + ", " + txtTelefon.getText());
}
```

Hendelseshåndtereren knyttes til knappen med denne koden i start():

```
btnOk = new Button("OK");
//Knytter knappen til hendelseshåndtereren:
btnOk.setOnAction(e -> behandleKlikk(e));
```

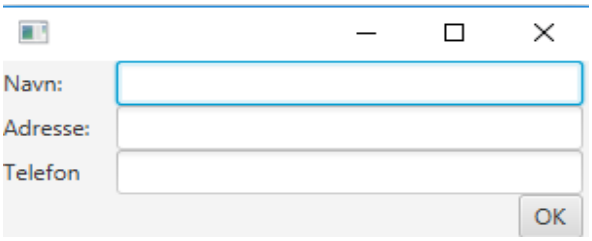
Parameteren til setOnAction er et eksempel på lambda-kode, der e står for hendelsesobjektet. I nyere utgaver av JavaFX trenger vi ikke ha med hendelsesobjektet som parameter i lytteren:

```
public void behandleNy() {
    Person person = new Person(txtNavn.getText(), txtAdresse.getText(), txtTelefon.getText());
    persondata.setText(person.toString());
    primaryStage.setScene(scene1);
}
```

```
Button button2 = new Button("Registrer");
button2.setOnAction(e -> behandleNy());
```

Eksempel med bruk av GridPane:

Vi skal lage et enkelt vindu for innlesing av persondata. En knapp skal, når den blir klikket på, lese dataene og gjøre noe med dem. Slik ser vinduet ut:



Nedenfor er vist programmet for dette:

```
1  package application;
2
3  import javafx.event.ActionEvent;
4
5  import javafx.application.Application;
6  import javafx.geometry.HPos;
7  import javafx.stage.Stage;
8  import javafx.scene.Scene;
9  import javafx.scene.control.Button;
10 import javafx.scene.control.Label;
11 import javafx.scene.control.TextField;
12 import javafx.scene.layout.GridPane;
13
14
15 public class Main extends Application {
16     //Elementene i grensesnittet er her deklarert globalt
17     //label'ene trenger man egentlig ikke deklarere globalt
18     Label lblNavn;
19     Label lblAdresse;
20     Label lblTelefon;
21     TextField txtNavn;
22     TextField txtAdresse;
23     TextField txtTelefon;
24     Button btnOk;
25
26     @Override
27     public void start(Stage primaryStage) {
28         try {
29             //Oppretter et GridPane:
30             GridPane rotpanel = new GridPane();
31             rotpanel.setHgap(10);
32             lblNavn = new Label("Navn: ");
33             lblAdresse = new Label("Adresse: ");
34             lblTelefon = new Label("Telefon");
35             txtNavn = new TextField();
36             txtAdresse = new TextField();
37             txtTelefon = new TextField();
38             txtNavn.setPrefColumnCount(20);
39             txtAdresse.setPrefColumnCount(20);
40             txtTelefon.setPrefColumnCount(20);
41             btnOk = new Button("OK");
42             //Knytter knappen til hendelseshåndvereren:
43             btnOk.setOnAction(e -> behandleKlikk(e));
```

```
            //Legger til elementene i grid'en:
            rotpanel.add(lblNavn, 0, 0);
            rotpanel.add(lblAdresse, 0, 1);
            rotpanel.add(lblTelefon, 0, 2);
            rotpanel.add(txtNavn, 1, 0);
            rotpanel.add(txtAdresse, 1, 1);
            rotpanel.add(txtTelefon, 1, 2);
            rotpanel.add(btnOk, 1, 3);
            //Høyrejusterer knappen:
            GridPane.setHalignment(btnOk, HPos.RIGHT);
            Scene scene = new Scene(rotpanel,300,100);
            //Standard, lagt til av Eclipse:
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    //Hendelseshåndterer:
    public void behandleKlikk(ActionEvent e) {
        System.out.println("Du registrerte: " + txtNavn.getText() + ", " + txtAdresse.getText() + ", " + txtTelefon.getText());
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

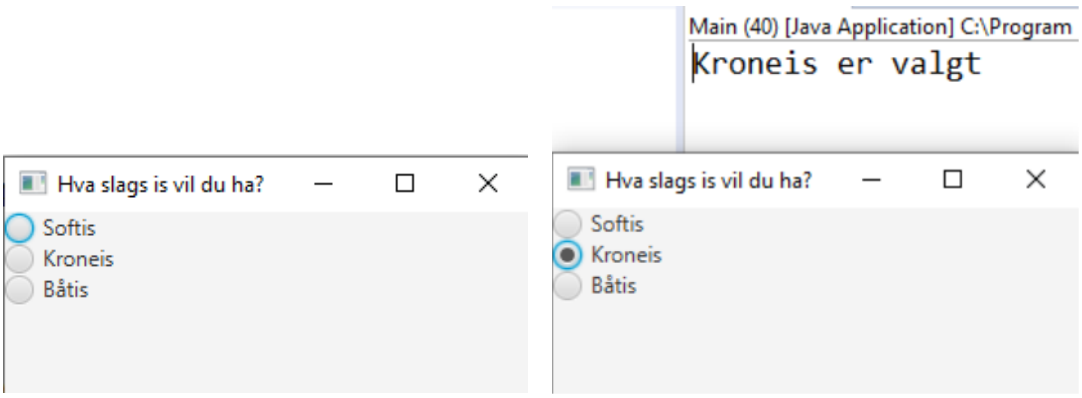
Radioknapper

En radioknapp kan være valgt eller ikke valgt (tilstander). Typisk kombineres radioknapper i grupper, slik at bare én knapp av gangen kan være valgt.

Her er et lite program som bruker radioknapper. Her er også vist en annen måte å lage hendelseshåndtering på:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.stage.Stage;
5  import javafx.scene.Scene;
6  import javafx.scene.control.RadioButton;
7  import javafx.scene.control.ToggleGroup;
8  import javafx.scene.layout.VBox;
9
10 public class Main extends Application {
11
12     final ToggleGroup gruppe = new ToggleGroup();
13     @Override
14     public void start(Stage primaryStage) {
15         try {
16             VBox rotpanel = new VBox();
17             RadioButton rb1 = new RadioButton("Softis");
18             rb1.setToggleGroup(gruppe);
19             RadioButton rb2 = new RadioButton("Kroneis");
20             rb2.setToggleGroup(gruppe);
21             RadioButton rb3 = new RadioButton("Båtis");
22             rb3.setToggleGroup(gruppe);
23             //Legger til i rotpanelet:
24             rotpanel.getChildren().addAll(rb1, rb2, rb3);
25             rb1.setOnAction(e -> {if(rb1.isSelected())System.out.println("Softis er valgt");});
26             rb2.setOnAction(e -> {if(rb2.isSelected())System.out.println("Kroneis er valgt");});
27             rb3.setOnAction(e -> {if(rb3.isSelected())System.out.println("Båtis er valgt");});
28             Scene scene = new Scene(rotpanel,200,100);
29             primaryStage.setTitle("Hva slags is vil du ha?");
30             primaryStage.setScene(scene);
31             primaryStage.show();
32         } catch(Exception e) {
33             e.printStackTrace();
34         }
35     }
36
37     public static void main(String[] args) {
38         launch(args);
39     }
40 }
```

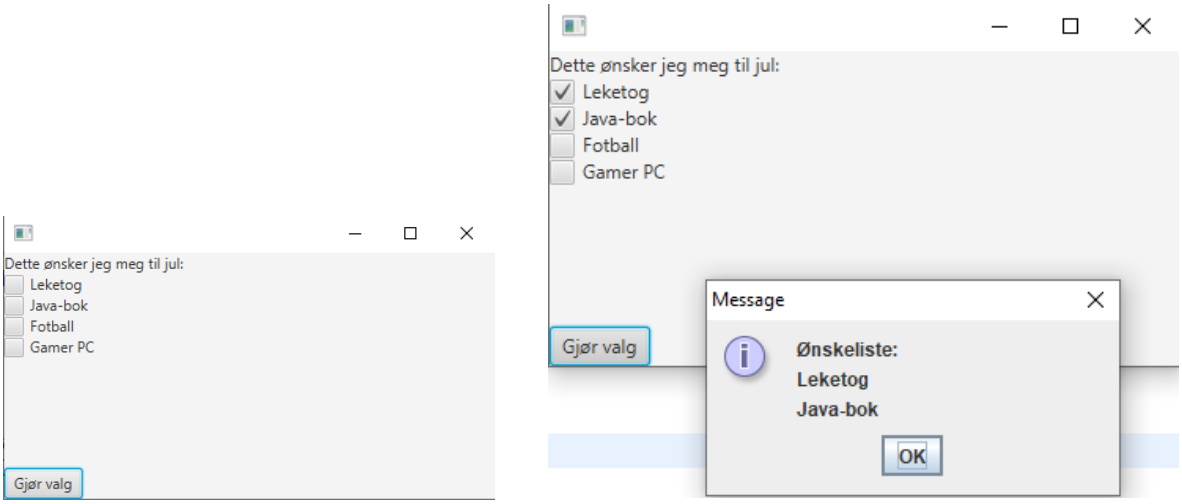
Når vi kjører programmet, ser det slik ut:



Valgene skrives ut i konsollvinduet.

Checkbokser

Med checkbokser kan vi, i motsetning med radiaknapper, krysse av flere på en gang. Nedenfor er vist et program som benytter checkbokser:



Til venstre ser vi vinduet før valg er tatt. Til høyre ser vi resultatet av å ha gjort valg og så klikket Gjør valg. En ønskeliste skrives ut i en JOptionPane meldingsboks (fra Java swing).

Programmet ser slik ut:

```
15 public class Main extends Application {
16     private CheckBox tog, bok, ball, pc;
17     @Override
18     public void start(Stage primaryStage) {
19         try {
20             BorderPane root = new BorderPane();
21             root.setTop(new Label("Dette ønsker jeg meg til jul:"));
22             VBox krysspanel = new VBox();
23             tog = new CheckBox("Leketog");
24             bok = new CheckBox("Java-bok");
25             ball = new CheckBox("Fotball");
26             pc = new CheckBox("Gamer PC");
27             Button valg = new Button("Gjør valg");
28             valg.setOnAction(e -> behandleValg());
29             root.setBottom(valg);
30             krysspanel.getChildren().addAll(tog, bok, ball, pc);
31             root.setCenter(krysspanel);
32             Scene scene = new Scene(root,400,200);
33             primaryStage.setScene(scene);
34             primaryStage.show();
35         } catch(Exception e) {
36             e.printStackTrace();
37         }
38     }
```

Håndteringen av klikk på «Gjør valg» er det denne metoden som tar seg av:

```
40     public void behandleValg() {
41         String ønskeliste = "";
42         if(tog.isSelected()) ønskeliste+="Leketog" + "\n";
43         if(bok.isSelected()) ønskeliste+="Java-bok" + "\n";
44         if(ball.isSelected()) ønskeliste+="Fotball" + "\n";
45         if(pc.isSelected()) ønskeliste+="Gamer PC";
46         tog.setSelected(false);
47         bok.setSelected(false);
48         ball.setSelected(false);
49         pc.setSelected(false);
50         JOptionPane.showMessageDialog(null, "Ønskeliste:" + "\n" + ønskeliste);
51     }
```

Forklaring:

Linje 16: checkBox'ene deklareres globalt for at lytteren skal vite om dem.

Linje 20: Et BorderPane opprettes som layout for scenen.

Linje 21: En overskrift legges inn i toppen av panelet.

Linje 22: En VBox layout opprettes. Avkrysningsboksene skal legges inn i dette.

Linje 23 – 26: avkrysningsbokser opprettes.

Linje 27: En Button opprettes.

Linje 28: Knappen knyttes til lytteren.

Linje 29: Knappen legges inn i bunnen av rotpanelet.

Linje 30: CheckBox'ene legges inn i VBox-panelet.

Linje 31: VBox-panelet legges i sentrum av rotpanelet.

Linje 32: Scenen opprettes med rotpanelet som parameter.

Lytteren:

Linje 41: En tom String opprettes. Denne skal bygges opp av valgene som er gjort.

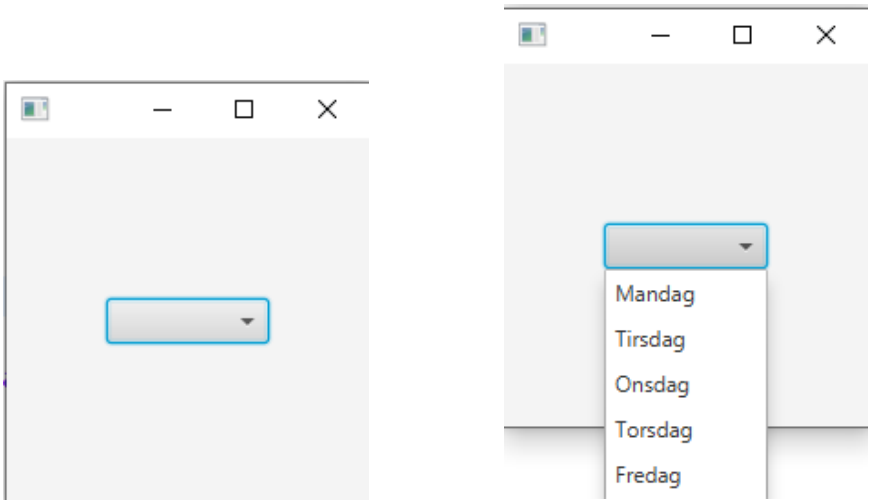
Linje 42 – 45: Tester på om en avkrysningsboks er valgt.

Linje 46 – 50: «Avkrysser» boksene, klar for neste jul.

Linje 51: Ønskelisten skrives ut i med javax.swing.JOptionPane.showMessageDialog().

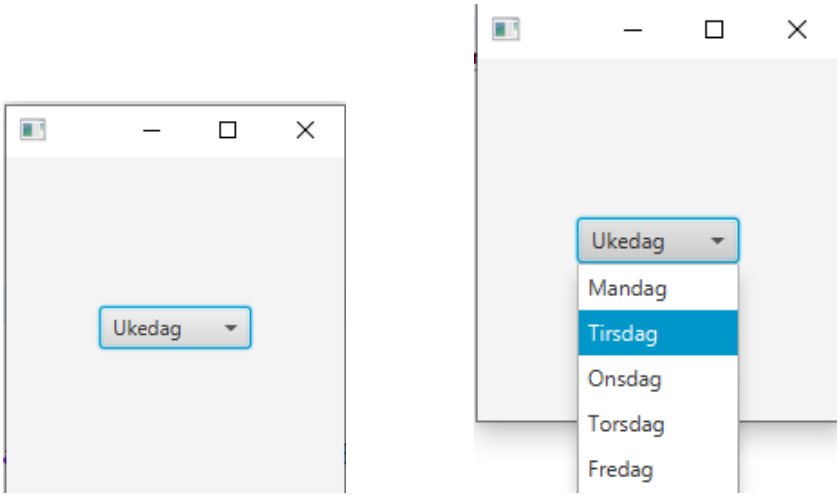
Comboboxer

ComboBoxer, eller nedtrekkslister på norsk, er en kompakt måte å vise flere alternativer på. Vi skal lage et lite program som lager følgende boks:



Til venstre er boksen lukket, til høyre er den åpnet.

Vi kan også sette enten en defaultverdi eller en tittel for selve boksen. Da ser det slik ut, før og etter åpning:



Når vi skal finne hvilket alternativ som er valgt, henter vi ut dette med metoden getValue(), som returnerer en String.

Hendelseshåndtereren er her temmelig enkel; den skal bare skrive ut valget i konsollet:

```
30    public void behandleValg() {
31        String valg = liste.getValue();
32        System.out.println(valg);
33    }
```

Så knytter vi listen til håndteringsmetoden slik i start()-metoden:

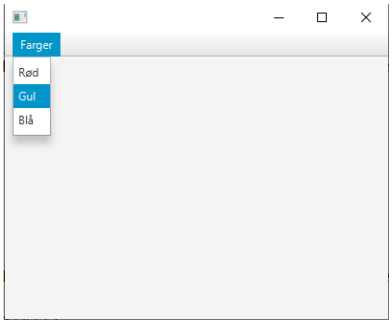
```
liste.setOnAction(e -> behandleValg());
```

Hele programmet ser slik ut:

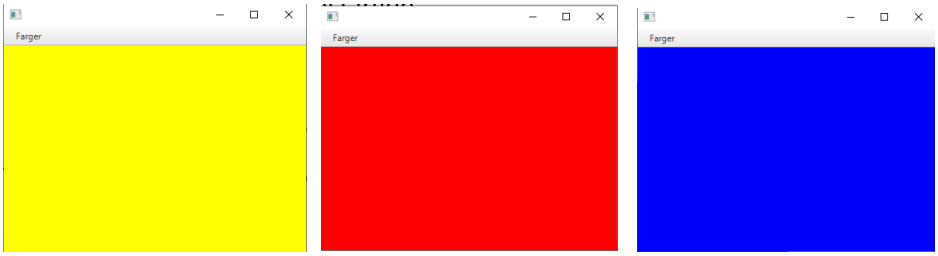
```
10 public class Main extends Application {
11     ComboBox<String> liste;
12     String[] alternativer = {"Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag"};
13     @Override
14     public void start(Stage vindu) {
15         try {
16             BorderPane rotpanel = new BorderPane();
17             liste = new ComboBox<String>();
18             liste.getItems().addAll(alternativer);
19             liste.setValue("Ukedag");
20             liste.setOnAction(e -> behandleValg());
21             rotpanel.setCenter(liste);
22             Scene scene = new Scene(rotpanel,200,200);
23             vindu.setScene(scene);
24             vindu.show();
25         } catch(Exception e) {
26             e.printStackTrace();
27         }
28     }
29 
30     public void behandleValg() {
31         String valg = liste.getValue();
32         System.out.println(valg);
33     }
34 
35     public static void main(String[] args) {
36         launch(args);
37     }
38 }
```

Menyer

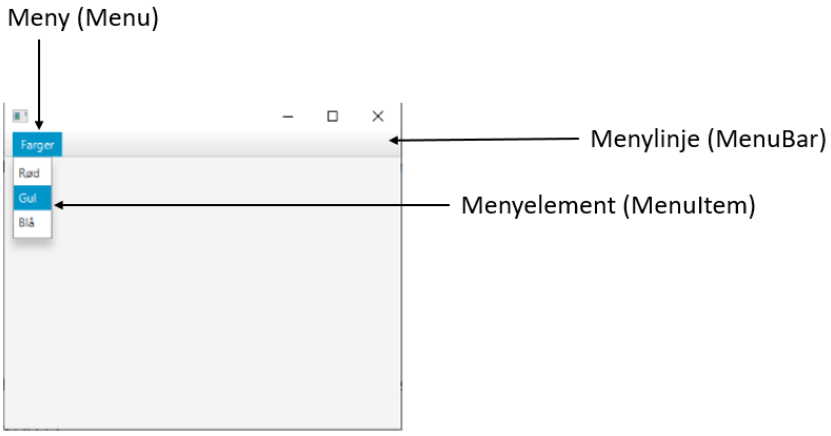
Vi skal lage et program som setter opp et vindu med en meny. Menyvalgene skal være forskjellige bakgrunnsfarger for vinduet. Her er vinduet før vi har gjort noen valg:



Når vi gjør et valg, skifter bakgrunnsfargen:



Måten en meny i JavaFX (og også Swing) er bygget opp på:



Vi må først opprette et objekt av klassen MenuBar. Til denne kan vi «feste» objekter av klassen Menu. Hvert av disse igjen kan ha flere MenuItems.

Et klikk på et menyelement trigger et ActionEvent-objekt. I Swing kunne vi enkelt lese ut hvilket element som ble valgt, ved rett og slett å hente ut teksten. Dette går ikke i JavaFX, så her anbefales det å lage en lyttermetode for hvert menyelement. Med mange valg burde dette bli mer oversiktlig enn å ha en felles metode.

Nedenfor er vist programmet med lyttere:

```
19
20 public class Main extends Application {
21     private MenuBar menylinje = new MenuBar();
22     private Menu filmeny = new Menu("Farger");
23     private MenuItem rød = new MenuItem("Rød");
24     private MenuItem gul = new MenuItem("Gul");
25     private MenuItem blå = new MenuItem("Blå");
26     private MenuItem avslutt = new MenuItem("Avslutt");
27     BorderPane rotpanel = new BorderPane();
28
29     @Override
30     public void start(Stage vindu) {
31         try {
32
33             filmeny.getItems().addAll(rød, gul, blå);
34             menylinje.getMenus().addAll(filmeny);
35             rød.setOnAction(e -> behandleRød(e));
36             gul.setOnAction(e -> behandleGul(e));
37             blå.setOnAction(e -> behandleBlå(e));
38             rotpanel.setTop(menylinje);
39             Scene scene = new Scene(rotpanel,400,300);
40             scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
41             vindu.setScene(scene);
42             vindu.show();
43         } catch(Exception e) {
44             e.printStackTrace();
45         }
46     }
```

Det er tre lyttere, en for hvert menyalternativ:

```
48     public void behandleRød(ActionEvent e) {
49         // create a background fill
50         BackgroundFill bakgrunnsfyll = new BackgroundFill(Color.RED, CornerRadii.EMPTY, Insets.EMPTY);
51         // create Background
52         Background bakgrunn = new Background(bakgrunnsfyll);
53         rotpanel.setBackground(bakgrunn);
54     }
55
56     public void behandleGul(ActionEvent e) {
57         // create a background fill
58         BackgroundFill bakgrunnsfyll = new BackgroundFill(Color.YELLOW, CornerRadii.EMPTY, Insets.EMPTY);
59         // create Background
60         Background bakgrunn = new Background(bakgrunnsfyll);
61         rotpanel.setBackground(bakgrunn);
62     }
63
64     public void behandleBlå(ActionEvent e) {
65         // create a background fill
66         BackgroundFill bakgrunnsfyll = new BackgroundFill(Color.BLUE, CornerRadii.EMPTY, Insets.EMPTY);
67         // create Background
68         Background bakgrunn = new Background(bakgrunnsfyll);
69         rotpanel.setBackground(bakgrunn);
70     }
```

Listebokser med innhold

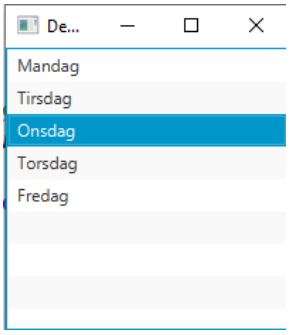
Listebokser gjør mye det samme som combobokser, men her kan vi gjøre flere valg samtidig. Vi skal i dette avsnittet også introdusere datamodellen for en grafisk komponent. Datamodellen er en egen klasse som skal håndtere innholdet i listen. Dette er basert på Model-View-Control (MVC) modellen. Hvis vi skal lage en listeboks med alternativer, vil denne bestå av tre bestanddeler:

- Et objekt av klassen ListView, som presenterer data
- Et objekt av klassen ObservableList. Denne klassen er generisk og kontrollerer elementene i listen
- Objektene som skal vises, som regel av klassen String

Et enkelt program med en listeboks er vist nedenfor:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.collections.FXCollections;
5  import javafx.collections.ObservableList;
6  import javafx.stage.Stage;
7  import javafx.scene.Scene;
8  import javafx.scene.control.ListView;
9  import javafx.scene.layout.BorderPane;
10
11 public class Main extends Application {
12     @Override
13     public void start(Stage vindu) {
14         try {
15             vindu.setTitle("Demo av liste");
16             //Oppretter et ListView:
17             ListView liste = new ListView();
18             //Oppretter datamodellen. Denne skal inneholde elementene i listen:
19             ObservableList<String> innhold = FXCollections.observableArrayList();
20             //Gir innhold innhold:
21             innhold.addAll("Mandag", "Tirsdag", "Onsdag", "Torsdag", "Fredag");
22             //Knytter innholdet til listen:
23             liste.setItems(innhold);
24             BorderPane rotpanel = new BorderPane();
25             Scene scene = new Scene(rotpanel,200,200);
26             rotpanel.setCenter(liste);
27             scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
28             vindu.setScene(scene);
29             vindu.show();
30         } catch(Exception e) {
31             e.printStackTrace();
32         }
33     }
34
35     public static void main(String[] args) {
36         launch(args);
37     }
38 }
```

Kjører vi programmet, blir resultatet dette vinduet:

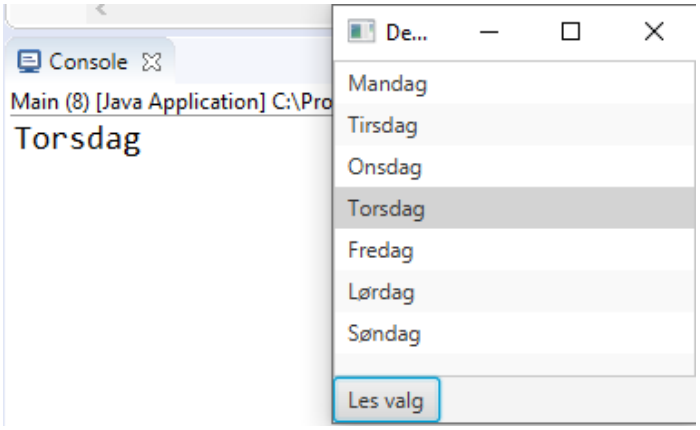


Programmet gjør foreløpig ikke noe annet enn å presentere listeboksen. Vi skal nå utruste det med en lytter. Denne skal fange opp hvilket element i listeboksen det er klikket på.

Met en listeboks kan vi velge flere elementer samtidig ved å holde nedt Ctrl og klikke på elementer. Default er imidlertid at vi bare kan velge ett element. Metoden for å hente ut valgte elementer gir et objekt av klassen ObservableList som resultat. Denne vil inneholde indeksene til de valgte elementene i datamodellen for listen. Siden vi bare skal bruke single selection, vil vi finne indeksen vi er ute etter på plass 0 i ObservableList'en.

Det kan være nok å hente ut indeksene for valg for å kunne teste i en lytter. I dette eksempelet skal vi imidlertid skrive ut valget som tekst. For at vi skal kunne gjøre dette, må vi ha en ekstra datastruktur med navnene på valgmulighetene. Jeg har valgt å bruke en ArrayList til å legge inn ukedagene i. Nå er klassen ObservableList laget slik at en av konstruktørene skal ha nettopp en liste som parameter. Dermed vil det være samsvar mellom indeksene i ArrayList'en og indeksene i ObservableList'en som er datamodell for listeboksen. Den ObservableList'en vi får når vi leser valget vil ha indeksen til valget i posisjon 0.

Programmet ser slik ut når vi kjører det med single selection:



Nedenfor er vist klassene som er i bruk for å lage dette programmet:

```
3  import java.util.ArrayList;
4
5  import javafx.application.Application;
6  import javafx.collections.FXCollections;
7  import javafx.collections.ObservableList;
8  import javafx.stage.Stage;
9  import javafx.scene.Scene;
10 import javafx.scene.control.Button;
11 import javafx.scene.control.ListView;
12 import javafx.scene.control.SelectionMode;
13 import javafx.scene.layout.BorderPane;
```

Hele programmet er vist på neste side.

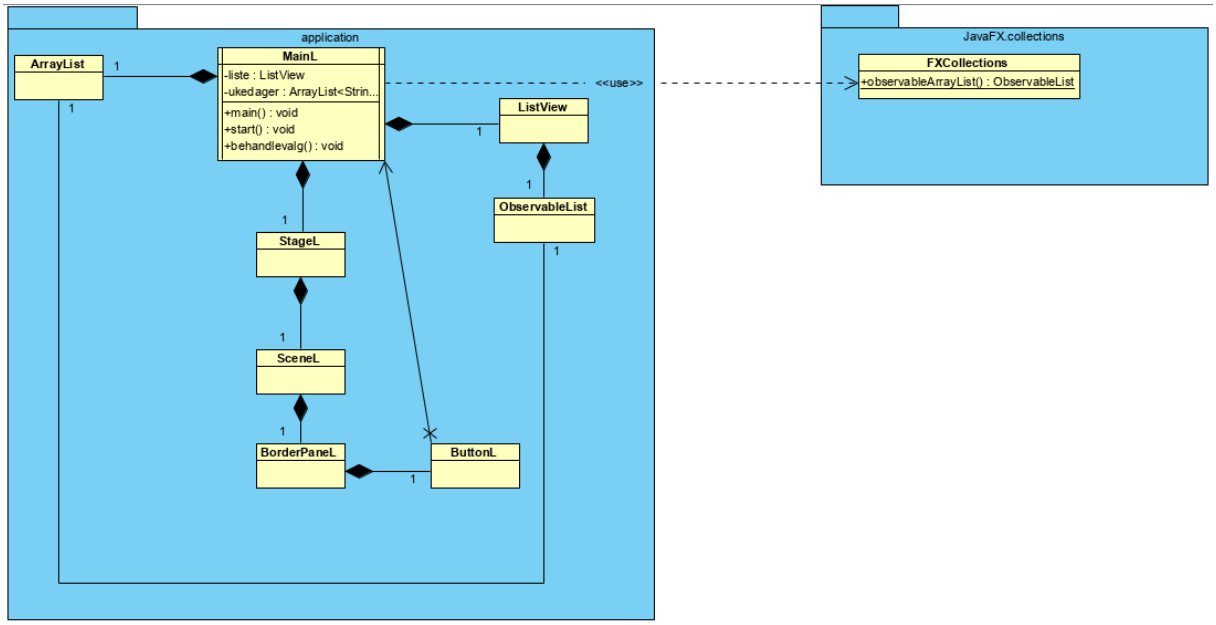
Metoden start(), som setter opp vinduet:

```
15 public class Main extends Application {
16     private ListView liste;
17     private ArrayList<String> ukedager = new ArrayList<>();
18     @Override
19     public void start(Stage vindu) {
20         try {
21             vindu.setTitle("Demo av liste");
22             //Oppretter et ListView:
23             liste = new ListView();
24             //Oppretter datamodellen. Denne skal inneholde elementene i listen:
25             ObservableList<String> innhold = FXCollections.observableArrayList();
26             //Gir ukedager innhold:
27             ukedager.add("Mandag");
28             ukedager.add("Tirsdag");
29             ukedager.add("Onsdag");
30             ukedager.add("Torsdag");
31             ukedager.add("Fredag");
32             ukedager.add("Lørdag");
33             ukedager.add("Søndag");
34             innhold.addAll(ukedager);
35             liste.setItems(innhold);
36             Button knapp = new Button("Les valg");
37             knapp.setOnAction(e -> behandleValg());
38             BorderPane rotpanel = new BorderPane();
39             Scene scene = new Scene(rotpanel,200,200);
40             rotpanel.setCenter(liste);
41             rotpanel.setBottom(knapp);
42             scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
43             vindu.setScene(scene);
44             vindu.show();
45         } catch(Exception e) {
46             e.printStackTrace();
47         }
48     }
```

Lytteren, som behandler valg. Merk at det er et klikk på knappen som er hendelsen:

```
50     public void behandleValg() {
51         //Henter ut de valgte elementene i listen. ObservableList'en vil inneholde indeksene, ikke tekstene:
52         ObservableList valgteElementer = liste.getSelectionModel().getSelectedIndices();
53         //Finner indeksen til valgt element i listen. Siden vi har single selection, vil vi finne
54         //indeksen på plass 0 i ObservableList'en:
55         int indeks = (int)valgteElementer.get(0);
56         //Indeksen i ArrayList'en med ukenavn er den samme:
57         System.out.println(ukedager.get(indeks));
58     }
```

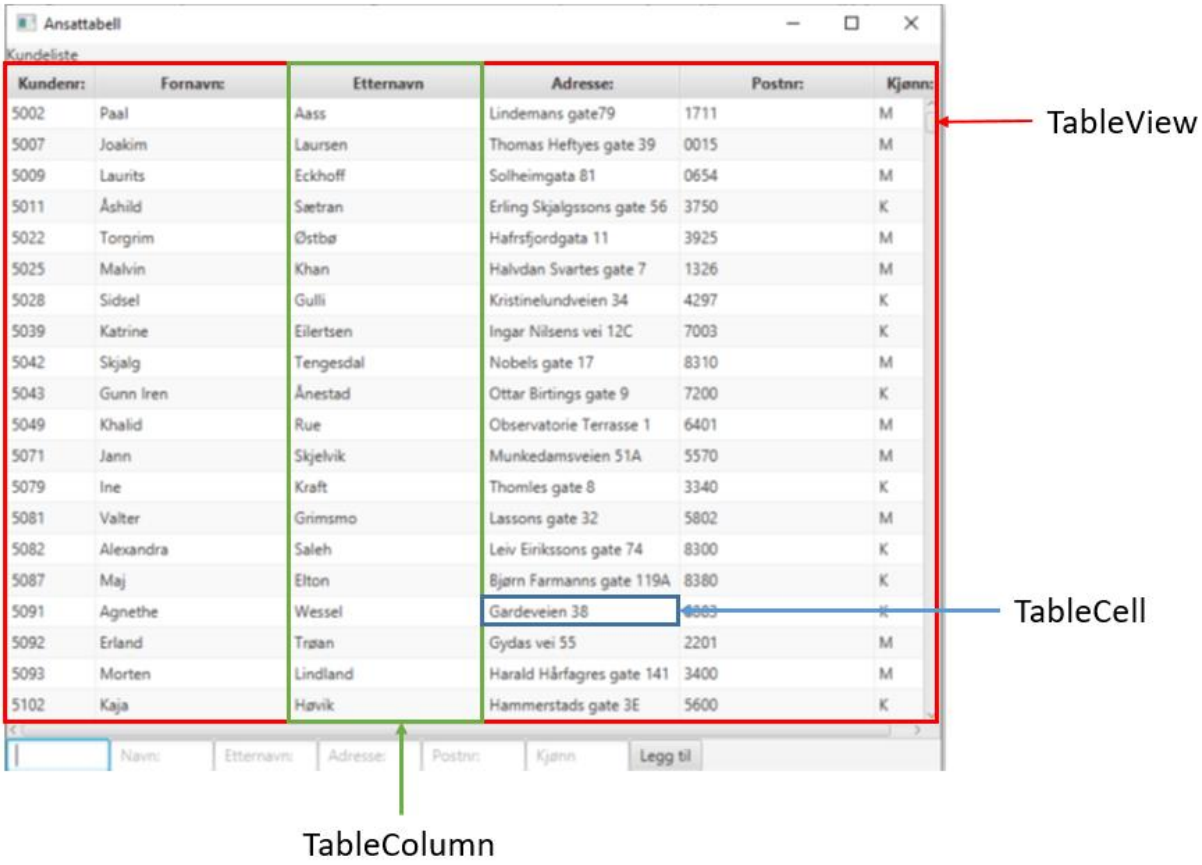
Et klassediagram kan være en hjelp til å forstå programmet:


application
ArrayList
MainL
-liste : ListView
-ukedager : ArrayList<Strin...
+main() : void
+start() : void
+behandlevalg() : void
<<use>>
JavaFX.collections
FXCollections
+observableArrayList() : ObservableList
ListView
ObservableList
StageL
SceneL
BorderPaneL
ButtonL
1

Tabeller

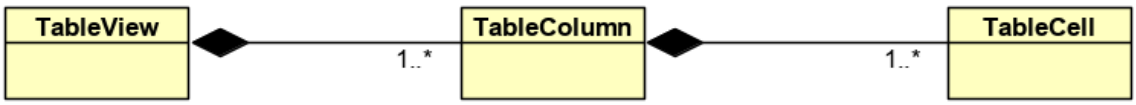
Presentasjon av data i tabeller er svært vanlig. Tenk på en faktura med sine fakturalinjer. I JavaFx er en tabell et objekt av klassen TableView. Vi kan lage overskrifter for kolonnene ved å lagre objekter av klassen TableColumn.

Oppbygningen av en tabell i JavaFX kan illustreres slik:



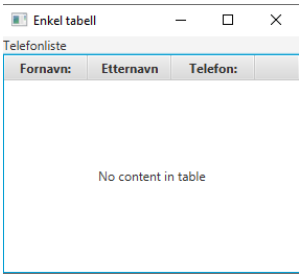
I et objekt av klassen TableView kan vi legge flere objekter av klassen TableColumn. Hver av disse vil igjen inneholde flere objekter av klassen TableCell. I tillegg vil hver kolonne ha en overskrift.

Et UML klassediagram for oppbyggingen ser slik ut:



Objekter av klassen TableCell trenger vi normalt ikke bry oss om. De opprettes automatisk når vi knytter tabellen til data.

Vi skal lage et enkelt vindu med en tabell. I utgangspunktet er selve tabellen tom. Vinduet ser slik ut:



Koden for vinduet ser slik ut:

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.stage.Stage;
5  import javafx.scene.Scene;
6  import javafx.scene.control.Label;
7  import javafx.scene.control.TableColumn;
8  import javafx.scene.control.TableView;
9  import javafx.scene.layout.BorderPane;
10
11 public class Main extends Application {
12     private TableView tabell = new TableView();
13     @Override
14     public void start(Stage vindu) {
15         try {
16             BorderPane rotpanel = new BorderPane();
17             Scene scene = new Scene(rotpanel,400,400);
18             vindu.setTitle("Enkel tabell");
19             vindu.setWidth(300);
20             vindu.setHeight(400);
21             TableColumn fornavn = new TableColumn("Fornavn:");
22             TableColumn etternavn = new TableColumn("Etternavn");
23             TableColumn telefon = new TableColumn("Telefon:");
24             tabell.getColumns().addAll(fornavn, etternavn, telefon);
25             Label overskrift = new Label("Telefonliste");
26             rotpanel.setTop(overskrift);
27             rotpanel.setCenter(tabell);
28             vindu.setScene(scene);
29             vindu.show();
30         } catch(Exception e) {
31             e.printStackTrace();
32         }
33     }
34
35     public static void main(String[] args) {
36         launch(args);
37     }
38 }
```

Forklaring på programmet:

Linje 12: oppretter et objekt av klassen TableView.

Linje 18: setter en tittel på vinduet

Linje 21 til 23: setter inn kolonner med overskrifter

Linje 24: legger kolonneoverskriftene til tabellen

Linje 25: lager en overskrift til tabellen

Linje 26: legger overskriften inn i rotpanelets toppfelt

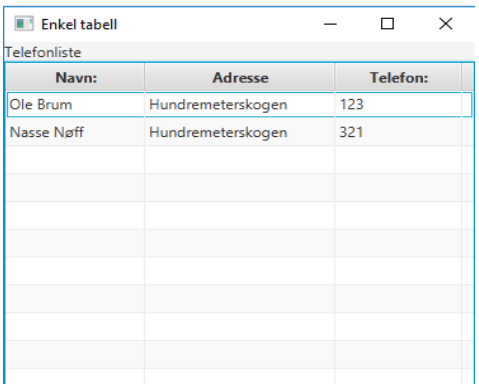
Linje 27: legger tabellen inn i rotpanelets midtfelt

TableView med data

For å kunne legge inn data i en tabell, trenger vi først en datamodell. Det er denne som håndterer selve innholdet i tabellen, mens tabellen selv presenterer dem.

Vi trenger et objekt av klassen ObservableList som datamodell. Denne får vi ved å kalle klassemetoden observableArrayList() fra klassen FXCollections, altså på samme måte som med en liste.

Her er tabellen vår med data om personer:



Programkoden er vist nedenfor. Til prosjektet hører også en klasse Person, vist på neste side.

```
15 public class Main extends Application {
16     //Lager tabellen
17     private TableView tabell = new TableView();
18     //deklarerer en datamodell for tabellen med innhold:
19     private ObservableList<Person> data = FXCollections.observableArrayList(new Person("Ole Brum", "Hundremeterskogen", "123"),
20             new Person("Nasse Nøff", "Hundremeterskogen", "321"));
21     @Override
22     public void start(Stage vindu) {
23         try {
24             BorderPane rotpanel = new BorderPane();
25             Scene scene = new Scene(rotpanel,600,600);
26             vindu.setTitle("Enkel tabell");
27             vindu.setWidth(300);
28             vindu.setHeight(400);
29             TableColumn navn = new TableColumn("Navn:");
30             navn.setMinWidth(100);
31             navn.setCellValueFactory(new PropertyValueFactory<Person, String>("navn"));
32             TableColumn adresse = new TableColumn("Adresse");
33             adresse.setMinWidth(100);
34             adresse.setCellValueFactory(new PropertyValueFactory<Person, String>("adresse"));
35             TableColumn telefon = new TableColumn("Telefon:");
36             telefon.setMinWidth(100);
37             telefon.setCellValueFactory(new PropertyValueFactory<Person, String>("telefon"));
38             tabell.getColumns().addAll(navn, adresse, telefon);
39             //innhold.add(rad);
40             //Legger inn data i datamodellen:
41             tabell.setItems(data);
42             //tabell.refresh();
43             Label overskrift = new Label("Telefonliste");
44             rotpanel.setTop(overskrift);
45             rotpanel.setCenter(tabell);
46             vindu.setScene(scene);
47             vindu.show();
48         } catch(Exception e) {
49             e.printStackTrace();
50         }
51     }
```

Forklaring:

Linje 17: Oppretter et objekt av klassen TableView.

Linje 18: Oppretter datamodellen som skal inneholde elementene som skal vises i tabellen.

Selve datamodellen er et objekt av klassen ObservableList og opprettes ved å kalle

klassemetoden FXCollections.observableArrayList(). I programmet gjøres dette

kallet med to Person-objekter som parametre, slik at datamodellen får innhold.

Linje 29: Oppretter en kolonne med overskrift:

```
TableColumn navn = new TableColumn("Navn:");
```

Linje 30: Setter minste bredde på kolonnen

Linje 31: Her legges til et objekt som skal vise attributtverdiene for en person, et objekt av klassen PropertyValueFactory.

Linje 38: Kolonnene legges til tabellen.

Linje 41: datamodellen legges til tabellen.

Linje 44 – 45: Tabellen og overskriften legges til i rotpanelet.

En TableColumn må ha et objekt som kan sette celleverdier knyttet til seg. Slike objekter kalles factory-objekter. Deres oppgave er å hente ut verdier og presentere dem i celler i tabellen. I programmet over er brukt objekter av klassen PropertyValueFactory. Objektene av denne knyttes til attributtene i klassen som definerer disse. I vårt tilfelle har vi klassen Person med attributtene navn, adresse og telefon. I programkoden står dette:

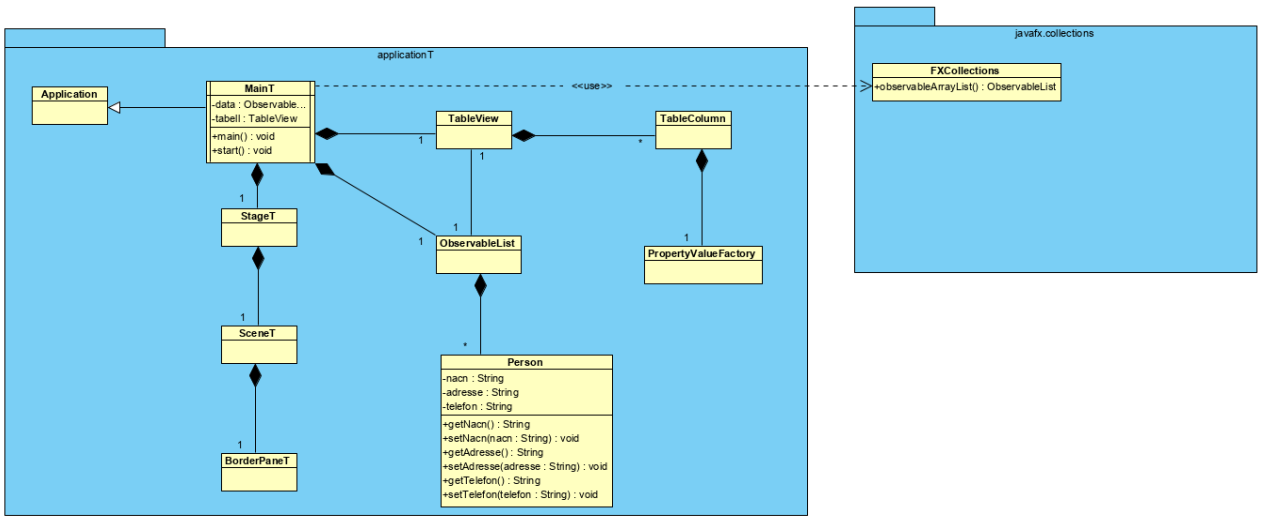
```
33          TableColumn navn = new TableColumn("Navn:");
34          navn.setMinWidth(100);
35          navn.setCellValueFactory(new PropertyValueFactory<Person, String>("navn"));
```

Parameteren til konstruktøren til TableColumn er overskriften for kolonnen. Parameter til konstruktøren til PropertyValueFactory er navnet på attributtet i klassen Person. Factory-objektet vil da kalle getter-metoden for attributtet (iallfall hvis den følger standardnavngivning og heter getNavn()).

Klassen Person er vist her:

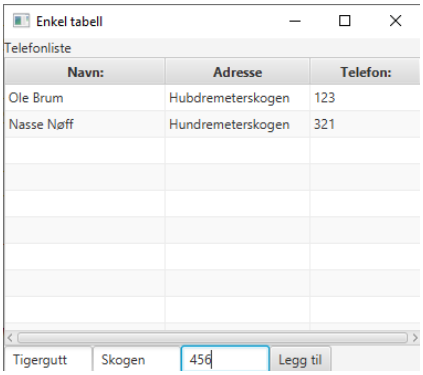
```java
public class Person {
    private String navn;
    private String adresse;
    private String telefon;

    public Person(String navn, String adresse, String telefon) {
        this.navn = navn;
        this.adresse = adresse;
        this.telefon = telefon;
    }

    public String getNavn() {
        return navn;
    }

    public void setNavn(String navn) {
        this.navn = navn;
    }

    public String getAdresse() {
        return adresse;
    }

    public void setAdresse(String adresse) {
        this.adresse = adresse;
    }

    public String getTelefon() {
        return telefon;
    }

    public void setTelefon(String telefon) {
        this.telefon = telefon;
    }

}
```

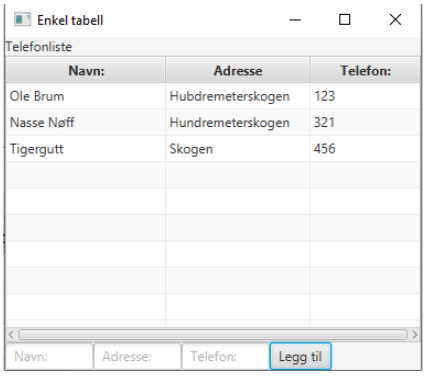
Et klassediagram for programmet:



Tabell med oppdatering

Hvis vi ønsker å legge til nye personer i tabellen, er dette ganske enkelt. Nedenfor er vist vinduet før og etter oppdatering av tabellen:





Tekstfeltene for data om nye person er lagt til i Bottom-delen av rotpanelet, sammen med en knapp for å utføre registreringen. Disse nodene er ikke lagt direkte til rotpanelet. Først er et FlowPane lagt til i Bottom, deretter legges nodene inn i dette.

En lytter er knyttet til knappen. For at lytteren skal vite om feltene for nytt navn, ny adresse og ny telefon, må disse være deklarert globalt i klassen:

```
18 public class Main extends Application {
19     //Lager tabellen
20         private TableView tabell = new TableView<>();
21         //deklarerer en datamodell for tabellen med innhold:
22         private ObservableList<Person> data = FXCollections.observableArrayList(new Person("Ole Brum", "Hubdremeterskogen", "123"),
23                 new Person("Nasse Nøff", "Hundremeterskogen", "321"));
24         TextField nyttnavn, nyadresse, nytelefon;
```

De nye linjene i programmet er vist her:

```
48              FlowPane registreringspanel = new FlowPane();
49              nyttnavn = new TextField();
50              nyttnavn.setPromptText("Navn: ");
51              nyttnavn.setMaxWidth(navn.getPrefWidth());
52              nyadresse = new TextField();
53              nyadresse.setPromptText("Adresse: ");
54              nyadresse.setMaxWidth(adresse.getPrefWidth());
55              nytelefon = new TextField();
56              nytelefon.setPromptText("Telefon: ");
57              nytelefon.setMaxWidth(telefon.getPrefWidth());
58              Button nyknapp = new Button("Legg til");
59              nyknapp.setOnAction(e -> behandleNy());
60              rotpanel.setTop(overskrift);
61              rotpanel.setCenter(tabell);
62              rotpanel.setBottom(registreringspanel);
63              registreringspanel.getChildren().addAll(nyttnavn, nyadresse, nytelefon, nyknapp);
```

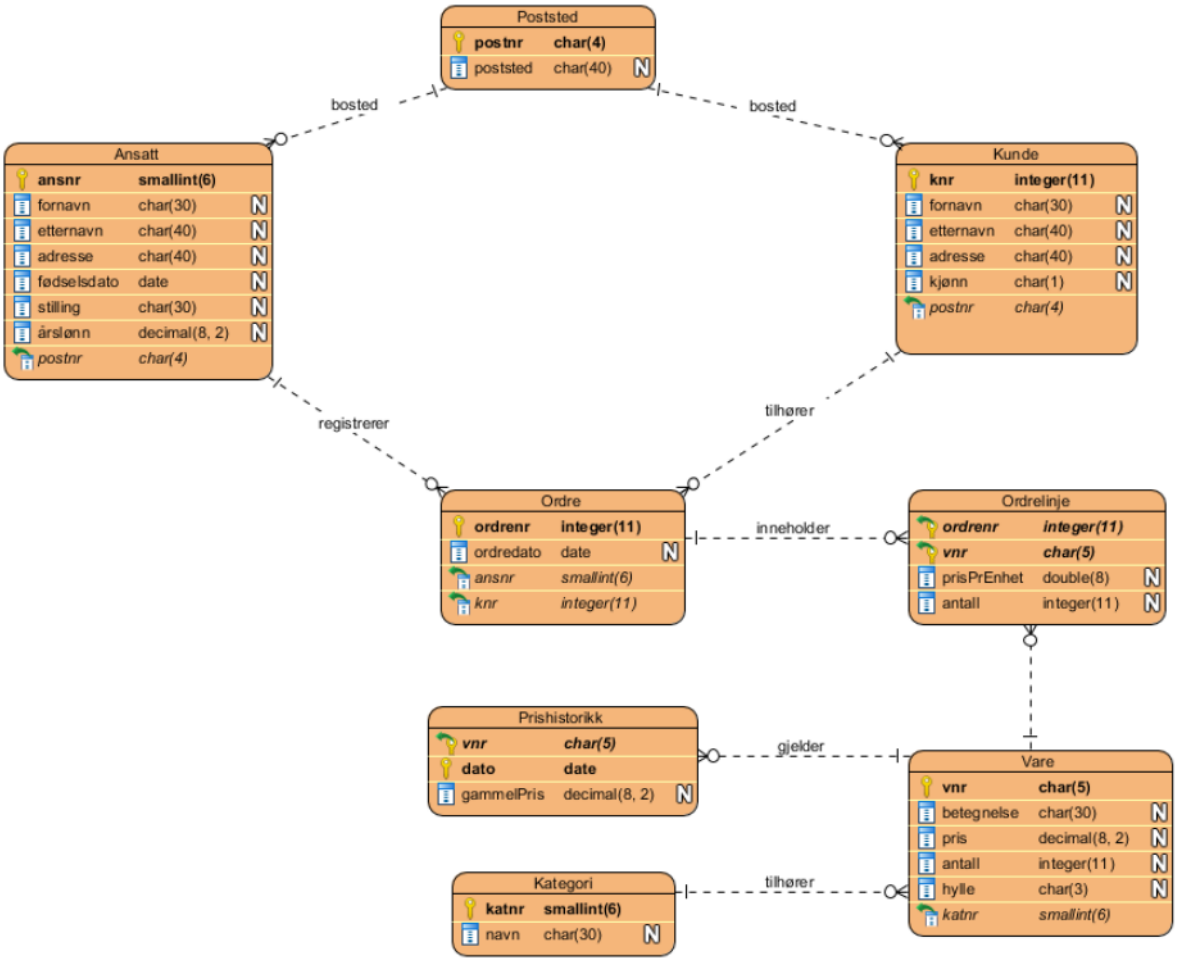
Lytteren ser slik ut:

```
71      public void behandleNy() {
72          data.add(new Person(nyttnavn.getText(), nyadresse.getText(), nytelefon.getText()));
73          nyttnavn.clear();
74          nyadresse.clear();
75          nytelefon.clear();
76      }
```

Når en ny post legges til i datamodellen, oppdateres tabellen automatisk.

Programmering mot relasjonsdatabase

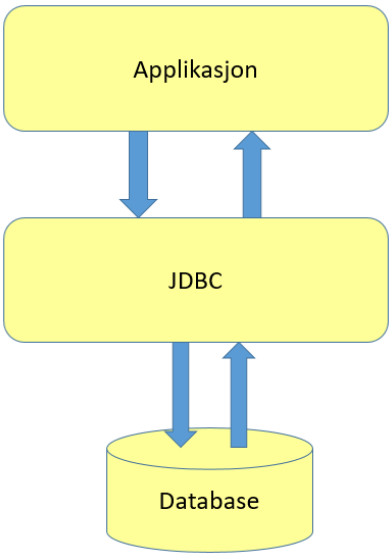
Vi skal i dette programeksempelet programmere mot databasen Hobbyhuset, som dere kjenner fra før (Bjørn Kristoffersen: Databasesystemer). Datamodellen for databasen er vist nedenfor:



Vi skal først lage en spørring mot tabellen Kunde, med presentasjon i et TableView. Deretter skal vi sette inn nye poster i databasetabellen.

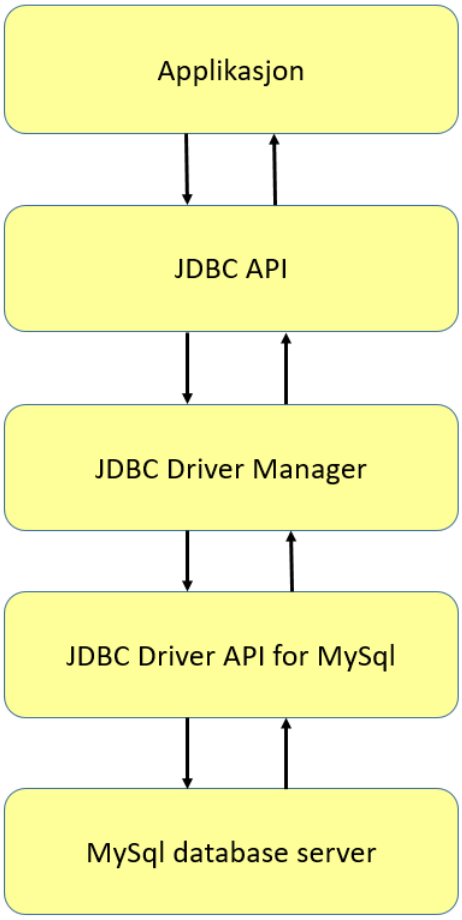
Vi skal nå også gjenoppta bruken av en Kontroll-klasse. Klassen Main skal fra nå av bare inneholde det som har med kommunikasjonen med brukeren å gjøre. Klassen Kontroll skal ta seg av all kommunikasjon med databasesystemet.

Kommunikasjon med et databasesystem krever en mellomvare, dvs et program som overfører signaler mellom vår applikasjon og databasesystemet. Med Java-applikasjoner er grunnlaget for mellomvaren en samling grensesnitt som heter JDBC. En enkel fremstilling av dette ser ut som følger:



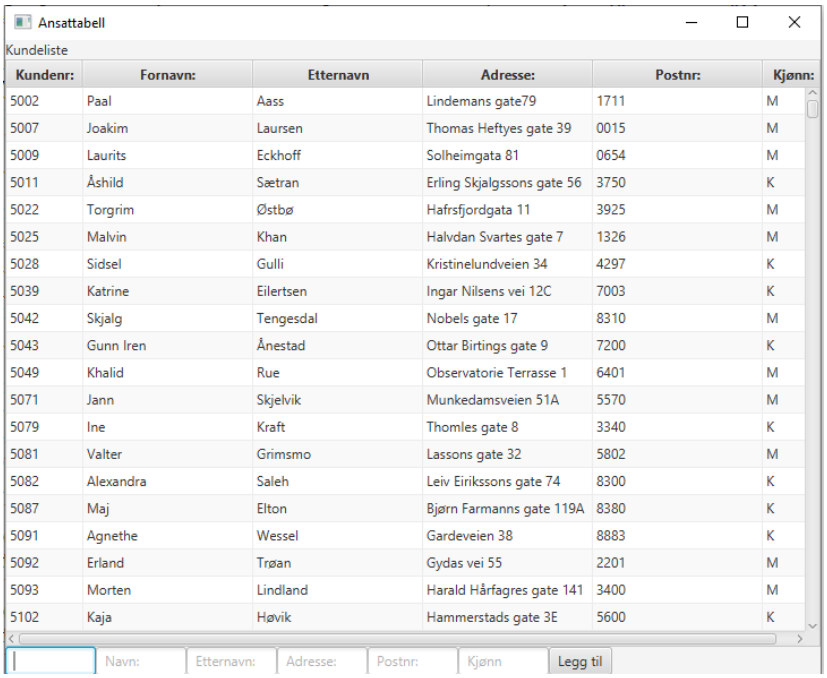
Grensesnittene (interface) som er definert i JDBC er implementert som databasedrivere for det enkelte databasesystem. For MySql heter denne driveren Connector/J og kan lastes ned fra denne adressen: https://dev.mysql.com/downloads/connector/j/

En mer detaljert fremstilling av koblingen til databasen ser slik ut:

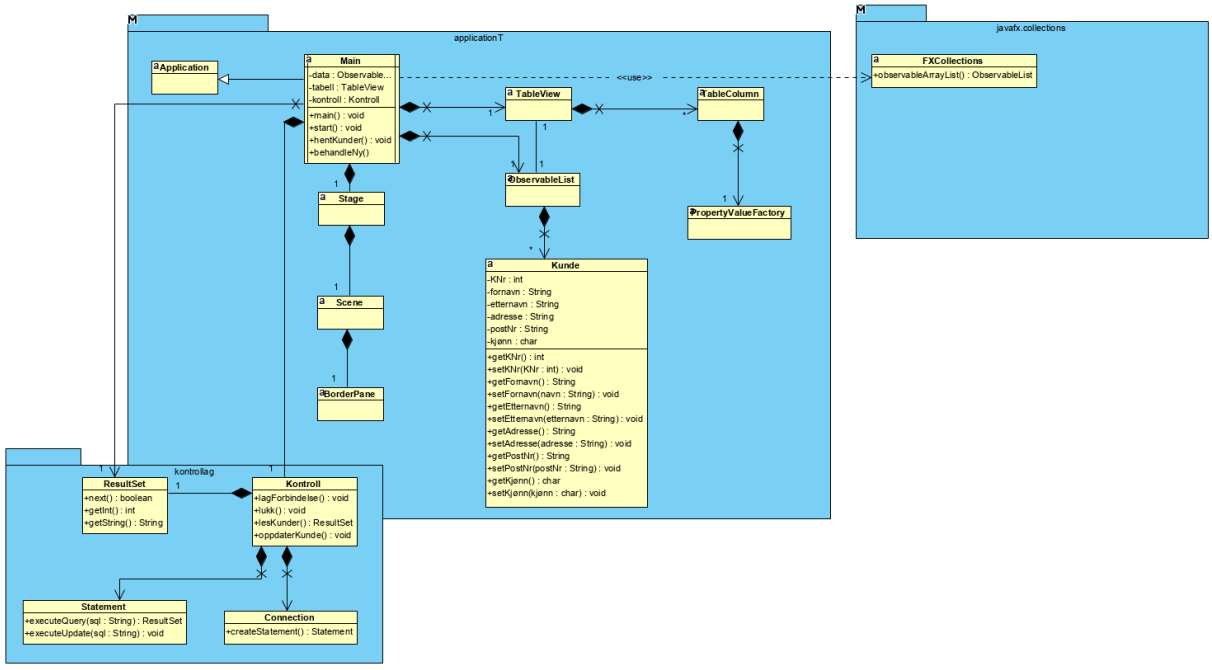


For installasjon og tilknytning til den enkelte applikasjon; se PowerPointen «OBJ2100 – JavaFX lister og tabeller».

Tabellvisningen i JavaFX ser slik ut, inkludert et panel for registrering av nye kunder:



Et UML klassediagram for applikasjonen ser slik ut:



Kontrollklassen med de nødvendige klassene for å kommunisere med databasen er her lagt i en egen pakke.

Kontrollklassen

I kontrollklassen trenger vi noen globale deklarasjoner. Disse er vist nedenfor, sammen med forklaringer:

```
import java.sql.Statement; //Sjekk denne! Eclipse kan importere fra beans!
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;

public class Kontroll {
    private String databasenavn = "jdbc:mysql://localhost:3306/hobbyhuset";
    private String databasedriver = "com.mysql.jdbc.Driver";
    private Connection forbindelse;
    private ResultSet resultat;
    private Statement utsagn;
```

Databasenavn og drivernavn

Lager forbindelsen til databasen

Resultatet av en SQL-spørring legges i et ResultSet

Et Statement-objekt sender spørringen til databasen

I kontrollklassen opprettes først kontakt med databasen ved hjelp av databasedriveren for MySql:

Metoden kaster eventuelt unntak videre

Her bruker dere deres egen bruker og passord

```
public void lagForbindelse() throws Exception {
try {
        forbindelse = DriverManager.getConnection(databasenavn,"root","OBJ2100");
    } catch(Exception e) {
        throw new Exception("Kan ikke oppnå kontakt med databasen");
    }
}
```

Kaster et nytt unntaksobjekt med feilmelding

Det anbefales å også ha en metode for å lukke forbindelsen til databasen og gjerne også de andre objektene:

```
public void lukk() throws Exception {
    try {
        if(forbindelse != null) {
            forbindelse.close();
            resultat.close();
            utsagn.close();
        }
    }catch(Exception e) {
        throw new Exception("Kan ikke lukke databaseforbindelse");
    }
}
```

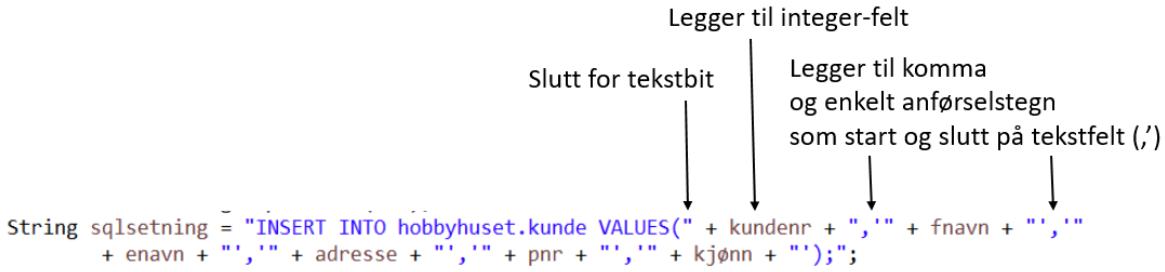
En spørring på kundetabellen kan gjøres enkelt med en SELECT * FROM Kunde. Vi kan plukke ut de feltene vi vil ha fra ResultSet'et senere. Dermed er vi ikke så avhengige av å ha høy kompetanse på SQL:

```
35    public ResultSet lesKunder() throws Exception {
36        ResultSet resultat = null;
37        String sql = "SELECT * FROM Kunde";
38        try {
39            utsagn = forbindelse.createStatement();
40            resultat = utsagn.executeQuery(sql);
41        }catch(Exception e) {throw new Exception("Kan ikke åpne databasetabell");}
42        return resultat;
43    }
```

Med innsetting av en ny kunde må vi passe på flere ting. Her er metoden for å sette inn en ny kunde:

```
45    public void oppdaterKunde(String knr, String fnavn, String enavn, String adresse, String pnr, String kjønn) throws Exception {
46        int kundenr = Integer.parseInt(knr);
47        String sqlsetning = "INSERT INTO hobbyhuset.kunde VALUES(" + kundenr + ",'" + fnavn + "','"
48                + enavn + "','" + adresse + "','" + pnr + "','" + kjønn + "');";
49        System.out.println(sqlsetning);
50        try {
51            Statement utsagn = forbindelse.createStatement();
52            utsagn.executeUpdate(sqlsetning);
53        }catch(Exception e) {throw new Exception("Kan ikke lagre data");}
54    }
```

Den største vanskeligheten her er å holde styr på doble og enkle anførselstegn i sql-setningen. Husk at i sql skal det være enkle anførselstegn rundt tekstvariabler som skal settes inn, og ingen anførselstegn rundt tallvariabler. Videre brukes doble anførselstegn for å bryte opp selve teksten. Vi kan se nærmere på en del av INSERT-setningen:



Som vist i kodeeksempelet har jeg lagt til en System.out.println(sqlsetning) for å kunne kontrollere at vi har fått det til.

Det neste vi må passe på, er at MySql sjekker referanseintegritet. Det betyr at når vi skal sette inn en ny kunde, må postnummeret vi vi sette inn allerede finnes i Poststed-tabellen.

Main-klassen

I Main-klassen har vi som tidligere all kommunikasjon med brukeren. Det betyr at tabellpresentasjonene lages her, og data for nye kunder leses inn her. Feilmeldinger skal også presenteres her.

La oss starte med oppsettet for tabellpresentasjonen. Her er også vist globale deklarasjoner:

```
20 public class Main extends Application {
21     Kontroll kontroll = new Kontroll();
22     //Lager tabellen
23     private TableView tabell = new TableView<>();
24     //deklarerer en datamodell for tabellen med innhold:
25     private ObservableList<Kunde> data = FXCollections.observableArrayList();
26     TextField nyttKnr, nyttFnavn, nyttEnavn, nyadresse, nyttPnr, nyttKjønn;
27     @Override
28     public void start(Stage vindu) {
29         try {
30             kontroll.lagForbindelse();
31             BorderPane rotpanel = new BorderPane();
32             Scene scene = new Scene(rotpanel,900,600);
33             vindu.setTitle("Ansattabell");
34             vindu.setWidth(900);
35             vindu.setHeight(600);
36             TableColumn kundenr = new TableColumn("Kundenr:");
37             kundenr.setMinWidth(10);
38             kundenr.setCellValueFactory(new PropertyValueFactory<Kunde, String>("kundenr"));
39             TableColumn fornavn = new TableColumn("Fornavn:");
40             fornavn.setMinWidth(150);
41             fornavn.setCellValueFactory(new PropertyValueFactory<Kunde, String>("fornavn"));
42             TableColumn etternavn = new TableColumn("Etternavn");
43             etternavn.setMinWidth(150);
44             etternavn.setCellValueFactory(new PropertyValueFactory<Kunde, String>("etternavn"));
45             TableColumn adresse = new TableColumn("Adresse:");
46             adresse.setMinWidth(150);
47             adresse.setCellValueFactory(new PropertyValueFactory<Kunde, String>("adresse"));
48             TableColumn postnr = new TableColumn("Postnr:");
49             postnr.setMinWidth(150);
50             postnr.setCellValueFactory(new PropertyValueFactory<Kunde, String>("postnr"));
51             TableColumn kjønn = new TableColumn("Kjønn:");
52             kjønn.setMinWidth(10);
53             kjønn.setCellValueFactory(new PropertyValueFactory<Kunde, String>("kjønn"));
54             tabell.getColumns().addAll(kundenr, fornavn, etternavn, adresse, postnr, kjønn);
55             //Legger inn data i datamodellen:
56             tabell.setItems(data);
```

Programmet skiller seg ikke så mye fra tidligere eksempler med Person-data. Denne gangen brukes objekter av klassen Kunde for å presentere data i tabellen.

I Bottom-delen av rotpanelet legges det til et FlowPane for feltverdiene til en ny kunde:

```
58             FlowPane registreringspanel = new FlowPane();
59             nyttKnr = new TextField();
60             nyttKnr.setPromptText("Kundenr:");
61             nyttKnr.setMaxWidth(kundenr.getPrefWidth());
62             nyttFnavn = new TextField();
63             nyttFnavn.setPromptText("Navn: ");
64             nyttFnavn.setMaxWidth(fornavn.getPrefWidth());
65             nyttEnavn = new TextField();
66             nyttEnavn.setPromptText("Etternavn: ");
67             nyttEnavn.setMaxWidth(etternavn.getPrefWidth());
68             nyadresse = new TextField();
69             nyadresse.setPromptText("Adresse: ");
70             nyadresse.setMaxWidth(adresse.getPrefWidth());
71             nyttPnr = new TextField();
72             nyttPnr.setPromptText("Postnr:");
73             nyttPnr.setMaxWidth(postnr.getPrefWidth());
74             nyttKjønn = new TextField();
75             nyttKjønn.setPromptText("Kjønn");
76             nyttKjønn.setMaxWidth(kjønn.getPrefWidth());
```

```
77            //Oppretter en knapp:
78            Button nyknapp = new Button("Legg til");
79            nyknapp.setOnAction(e -> behandleNy());
80            rotpanel.setTop(overskrift);
81            rotpanel.setCenter(tabell);
82            rotpanel.setBottom(registreringspanel);
83            registreringspanel.getChildren().addAll(nyttKnr, nyttFnavn, nyttEnavn, nyadresse, nyttPnr, nyttKjønn, nyknapp);
84            hentKunder();
85            vindu.setScene(scene);
86            vindu.show();
87        } catch(Exception e) {
88            e.printStackTrace();
89        }
90    }
```

start()-metoden kaller en metode for å hente data fra databasen og persentere dem i tabellform (linje 84). Metoden som gjør dette ser slik ut:

```
106    public void hentKunder() {
107        data.clear();
108        try {
109            ResultSet resultat = kontroll.lesKunder();
110            while(resultat.next()) {
111                ObservableList rad = FXCollections.observableArrayList();
112                int kundenr = resultat.getInt(1);//I ResultSet starter indeksene på 1
113                String fnavn = resultat.getString(2);
114                String enavn = resultat.getString(3);
115                String adr = resultat.getString(4);
116                String postnr = resultat.getString(5);
117                String kjønn = resultat.getString(6);
118                data.add(new Kunde(kundenr, fnavn, enavn, adr, postnr, kjønn));
119            }
120        }catch(Exception e) {System.out.println(e.getMessage());}
121    }
```

Metoden kaller lesKunder() i Kontroll og får returnert et ResultSet med resultatet av spørringene. ResultSet har metoden next(), som kan brukes til å kontrollere en løkke. next() returnerer true så lenge det er flere poster igjen i ResultSet'et. Vær oppmerksom på at i et ResultSet starter nummereringen av felten på 1, ikke 0 som vi ellers er vant til. Resultset har også forskjellige metoder for uthenting avhengig av hva slags datatype vi vil ha. Her ser vi bruk av getInt() og getString().

Når alle data er hentet ut, oppretter vi et nytt objekt av klassen Kunde og legger det til ObservableList'en data. Data for objektet vil da automatisk bli oppdatert i tabellpresentasjonen.

I vinduet har vi også en knapp som skal klikkes når vi har lagt inn data for en ny kunde. Lytteren for knappen ser slik ut:

```
94    public void behandleNy() {
95        try {
96            kontroll.oppdaterKunde(nyttKnr.getText(), nyttFnavn.getText(), nyttEnavn.getText(),
97                    nyadresse.getText(), nyttPnr.getText(), nyttKjønn.getText());
98            hentKunder();
99        }catch(Exception e) {System.out.println(e.getMessage());}
100       nyttKnr.clear();
101       nyttFnavn.clear();
102       nyttEnavn.clear();
103       nyadresse.clear();
104       nyttPnr.clear();
105       nyttKjønn.clear();
106   }
```

Oppdaterer tabellvisning
Etter at ny kunde er lagt inn

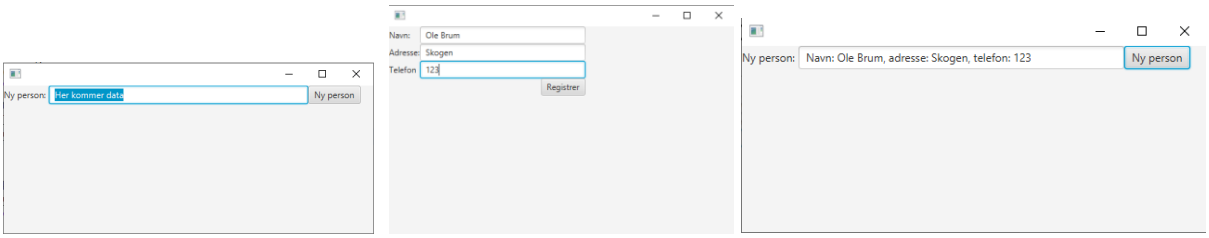
Sceneskifte

I en skikkelig grafisk applikasjon vil vi gjerne ha muligheten for å åpne forskjellige vinduer med hver sine oppgaver. I swing har vi klassen JDialog, som gjør at vi kan lage dialogvinduer som åpnes fra hovedvinduet (som er en subklasse av JFrame). JavaFX mangler denne klassen. Det finnes riktig nok en klasse som heter Dialog, men denne brukes mer som meldingsboksene til JOptionPane i swing. Dette betyr imidlertid ikke at vi ikke kan lage dialoger i JavaFX.

I JavaFX lager vi dialoger ved å lage flere scener for en stage. På mange måter er dette enklere inn med JDialog i swing. Vi skal se på et enkelt eksempel.

Enkel bruk av to scener

Nedenfor ser vi to scener i samme stage i en JavaFX applikasjon. Den første scenen er vist før og etter registrering av en ny person i den andre scenen:



Forklaring:

Til venstre ser vi scene1, med en Label for ledetekst, en TextArea for utskrift og en Button. Et klikk på knappen åpner scene2, som har et GridPane med Label'er, TextArea'er og en Button. Når vi har skrevet inn navn, adresse og telefon for en ny person, skjer følgende:

- Data leses fra TextArea'ene.
- Ei objekt av klassen Person lages.
- Scene1 vises
- Person-objektets toString()-metode kalles for utskrift i tekstfeltet i scene1.

Siden begge scenene lages i, og aktiveres fra, den samme stage'en, er Person-objektet tilgjengelig for scene1. Med dialogvinduer i swing er dette litt mer komplisert.

Programkoden er vist nedenfor:

```
16 public class Main extends Application {
17     Scene scene1, scene2;
18     TextField txtNavn, txtAdresse, txtTelefon, persondata;
19     Stage primaryStage;
20     @Override
21     public void start(Stage primaryStage) {
22         try {
23             //Scene 1:
24             Button button1= new Button("Ny person");
25             this.primaryStage = primaryStage;
26             button1.setOnAction(e -> primaryStage.setScene(scene2));
27             FlowPane layout1 = new FlowPane();
28             Label ledetekst = new Label("Ny person: ");
29             persondata = new TextField("Her kommer data");
30             persondata.setPrefColumnCount(30);
31             layout1.getChildren().addAll(ledetekst, persondata, button1);
32             scene1= new Scene(layout1, 500, 200);
33             //Scene 2:
34 
35             GridPane layout2 = new GridPane();
36             scene2= new Scene(layout2, 500, 300);
37             Button button2= new Button("Registrer");
38             button2.setOnAction(e -> behandleNy());
39             Label lblNavn = new Label("Navn:");
40             Label lblAdresse = new Label("Adresse:");
41             Label lblTelefon = new Label("Telefon");
42             txtNavn = new TextField();
43             txtAdresse = new TextField();
44             txtTelefon = new TextField();
45             txtNavn.setPrefColumnCount(20);
46             txtAdresse.setPrefColumnCount(20);
47             txtTelefon.setPrefColumnCount(20);
48             layout2.add(lblNavn, 0, 0);
49             layout2.add(lblAdresse, 0, 1);
50             layout2.add(lblTelefon, 0, 2);
51             layout2.add(txtNavn, 1, 0);
52             layout2.add(txtAdresse, 1, 1);
53             layout2.add(txtTelefon, 1, 2);
54             layout2.add(button2, 1, 3);
55             GridPane.setHalignment(button2, HPos.RIGHT);
56             primaryStage.setScene(scene1);
57             primaryStage.show();
58         } catch(Exception e) {
59             e.printStackTrace();
60         }
61 
62     }
63 
64     public void behandleNy() {
65         primaryStage.setScene(scene2);
66         Person person = new Person(txtNavn.getText(), txtAdresse.getText(), txtTelefon.getText());
67         persondata.setText(person.toString());
68         primaryStage.setScene(scene1);
69     }
70 
71     public static void main(String[] args) {
72         launch(args);
73     }
74 }
```

Forklaring:

Linje 17 – 19: Her deklareres de attributter som må være globalt tilgjengelige i klassen.

Linje 24: Her opprettes knappen for scene1.

Linje 25: Programklassens attributt primaryScene settes lik parameteren primaryScene i start().

Linje 26: her settes handlingen for klikk på button1. Det eneste denne skal gjøre er å skifte fokus til scene2.

Linje 27 – 31: Her opprettes et FlowPane og elementene som skal vises i dette i scene1. Deretter legges disse inn i FlowPane.

Linje 32: Her opprettes scene1 med FlowPanet (layout1) som parameter.

Linje 35: Layout for scene2 opprettes som en GridPane.

Linje 36: scene2 opprettes med layout2 som parameter.

Linje 37: Knappen button2 opprettes.

Linje 38: button2 knyttes til lytteren behandleKlikk() som vi finner på linjene 64 – 69.

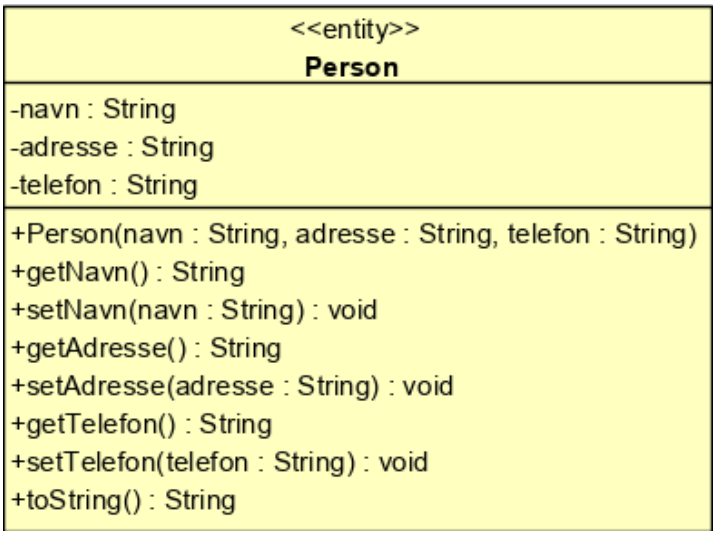
Linje 39 – 54: Elementene i layout2 opprettes og legges inn.

Linje 55: Her høyrejusteres knappen button2 i griden.

Linje 56: Her settes scene1 til «åpningsscene».

Linje 64 – 69: Lytteren for button2 setter først visningen til scene2. Deretter opprettes et objekt av klassen Person der attributtverdier leses fra tekstfeltene i griden. Teksten i tekstfeltet persondata i scene1 settes ved å kalle toString()-metoden til Person. Endelig settes visningen til scene1.

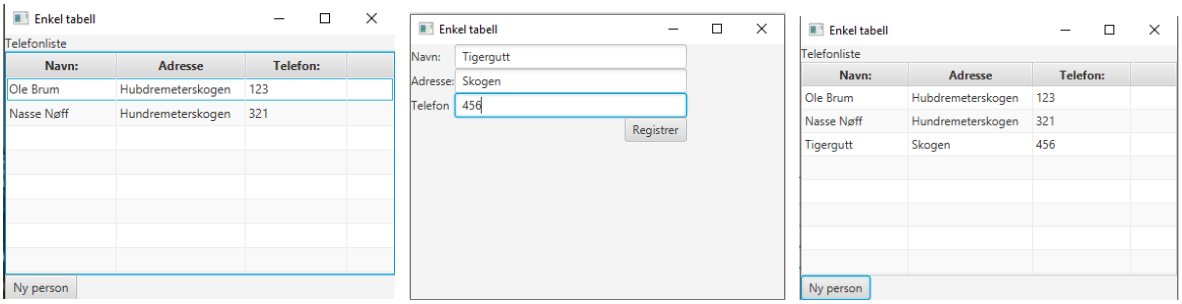
Nedenfor er et klassediagram for klassen Person:



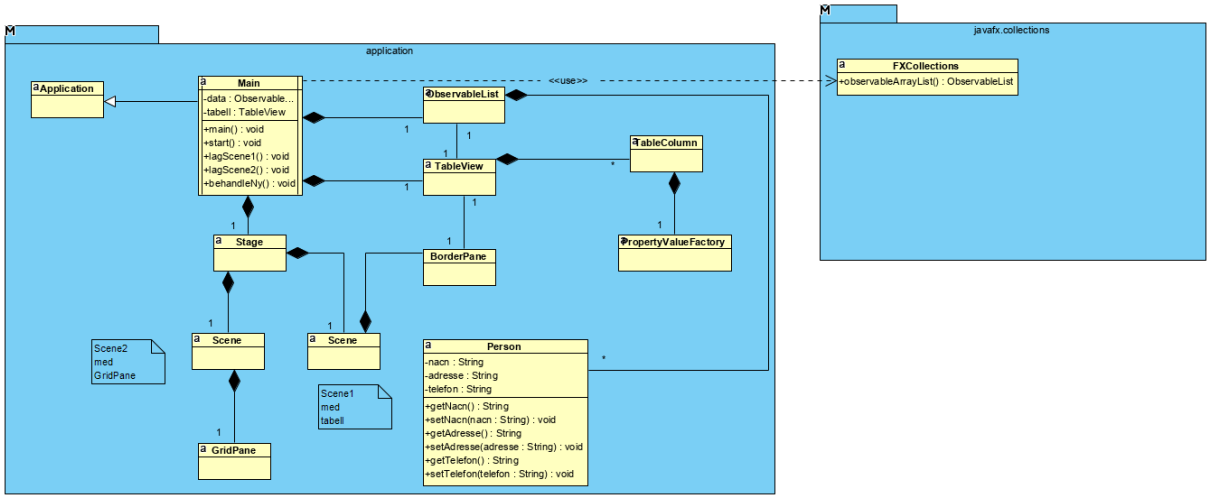
Egen scene for oppdatering av tabell

Vi skal gå tilbake til vårt tidligere eksempel med en tabell som kan oppdateres. Denne gangen skal registrering av et nytt objekt skje i et eget vindu.

Når programmet kjører, ser vi scenene nedenfor. Scene1 er vist før oppdatering til venstre, og etter oppdatering til høyre. I midten er scenen for registrering av en ny person:



Vi kan begynne med å vise et UML klassediagram for applikasjonen:



Det er viktig at attributter som skal være tilgjengelige for alle metoder er deklarert globalt i programklassen:

```
19 public class Main extends Application {
20     //Lager tabellen
21         private TableView tabell = new TableView<>();
22         //deklarerer en datamodell for tabellen med innhold:
23         private ObservableList<Person> data = FXCollections.observableArrayList(new Person("Ole Brum", "Hubdremeterskogen", "123"),
24                     new Person("Nasse Nøff", "Hundremeterskogen", "321"));
25         Scene scene1, scene2;
26         TextField txtNavn, txtAdresse, txtTelefon, persondata;
27         Stage vindu;
28
```

I Main-klassen er det egne metoder som setter opp scenene:

- lagScene1(): Setter opp en scene med en tabell i et BorderPane. Tabellen har tre kolonner. I tillegg en knapp for å registrere en ny person, som skifter fokus til scene2.
- lagScene2(): Setter opp en scene med et GridPane for tekstfelt med ledetekst, samt en Button for oppretting av et nytt Person-objekt som så legges til i ObservableList'en som er grunnlag for visning i tabellen. Fokus settes til scene1.

I tillegg er det en lytter-metode som utløses av klikk på knappen i scene2.

```
39      public void lagScene1() {
40          BorderPane rotpanel = new BorderPane();
41          scene1 = new Scene(rotpanel,400,300);
42          vindu.setTitle("Enkel tabell");
43          vindu.setWidth(400);
44          vindu.setHeight(300);
45          TableColumn navn = new TableColumn("Navn:");
46          navn.setMinWidth(100);
47          navn.setCellValueFactory(new PropertyValueFactory<Person, String>("navn"));
48          TableColumn adresse = new TableColumn("Adresse");
49          adresse.setMinWidth(100);
50          adresse.setCellValueFactory(new PropertyValueFactory<Person, String>("adresse"));
51          TableColumn telefon = new TableColumn("Telefon:");
52          telefon.setMinWidth(100);
53          telefon.setCellValueFactory(new PropertyValueFactory<Person, String>("telefon"));
54          lagScene2();
55          Button ny = new Button("Ny person");
56          ny.setOnAction(e -> vindu.setScene(scene2));
57          tabell.getColumns().addAll(navn, adresse, telefon);
58          //Legger inn data i datamodellen:
59          tabell.setItems(data);
60          Label overskrift = new Label("Telefonliste");
61          rotpanel.setTop(overskrift);
62          rotpanel.setCenter(tabell);
63          rotpanel.setBottom(ny);
64          vindu.setScene(scene1);
65          vindu.show();
66      }
```

Forklaring:

Linje 40: Her opprettes et BorderPane kalt rotpanel, som skal brukes av scenen.

Linje 41: Scene1 opprettes med rotpanelet som parameter.

Linje 45: Et TableView er allerede opprettet i linje 21. Nå lages den første kolonnen til.

Linje 46: Minimum bredde for kolonnen settes.

Linje 47: Et PropertyValueFactory-objekt opprettes og knyttes til kolonnen. PropertyValueFactory er generisk, og dette objektet skal behandle Person-objekter, og kolonnen navn. Oppgaven til dette objektet er å kalle metoden getNavn() fra et Person-objekt. PropertyValueFactory-objektet forutsetter at Person inneholder get-metoder.

Linje 48 – 53: De to neste kolonnene for adresse og telefon legges til.

Linje 54: Metoden lagScene2() kalles slik at scene2 opprettes.

Linje 55: En Button for å registrere en ny person lages.

Linje 56: En handling knyttes til knappen, i dette tilfelle settes fokus til scene2.

Linje 57: Kolonnene legges til tabellen.

Linje 58: Innholdet i tabellen settes til ObservableList'en data.

Så til metoden lagScene1:

```
66      public void lagScene2() {
67          GridPane layout2 = new GridPane();
68          scene2= new Scene(layout2, 500, 300);
69          Button button2= new Button("Registrer");
70          button2.setOnAction(e -> behandleNy());
71          Label lblNavn = new Label("Navn:");
72          Label lblAdresse = new Label("Adresse:");
73          Label lblTelefon = new Label("Telefon");
74          txtNavn = new TextField();
75          txtAdresse = new TextField();
76          txtTelefon = new TextField();
77          txtNavn.setPrefColumnCount(20);
78          txtAdresse.setPrefColumnCount(20);
79          txtTelefon.setPrefColumnCount(20);
80          layout2.add(lblNavn, 0, 0);
81          layout2.add(lblAdresse, 0, 1);
82          layout2.add(lblTelefon, 0, 2);
83          layout2.add(txtNavn, 1, 0);
84          layout2.add(txtAdresse, 1, 1);
85          layout2.add(txtTelefon, 1, 2);
86          layout2.add(button2, 1, 3);
87          GridPane.setHalignment(button2, HPos.RIGHT);
88      }
```

Forklaring:

Linje 67: Her settes layout til et GridPane som kalles layout2.

Linje 68: Her opprettes et scene-objekt kalt scene2 med layout2 som parameter.

Linje 69: En knapp for registrering opprettes.

Linje 70: Knappen får en hendelse knyttet til seg. Denne kaller lytteren behandleNy(), som vi skal se på senere.

Linje 71 – 87: Disse linjene er de samme som i eksempelet side 44, linjene 39 – 55.'

Lytteren for knappen button2 ser slik ut:

```
89      public void behandleNy() {
90          Person person = new Person(txtNavn.getText(), txtAdresse.getText(), txtTelefon.getText());
91          data.add(person);
92          vindu.setScene(scene1);
93      }
```

Linje 90: Her opprettes et objekt av klassen Person ved å lese verdiene i tekstfeltene i scene2.

Linje 91: Objektet legges til ObservableList'en data.

Linje 92: Fokus settes til scene1.

Data for det nye objektet vil nå vises i tabellen i scene1.

Vi skal nå se på start()-metoden:

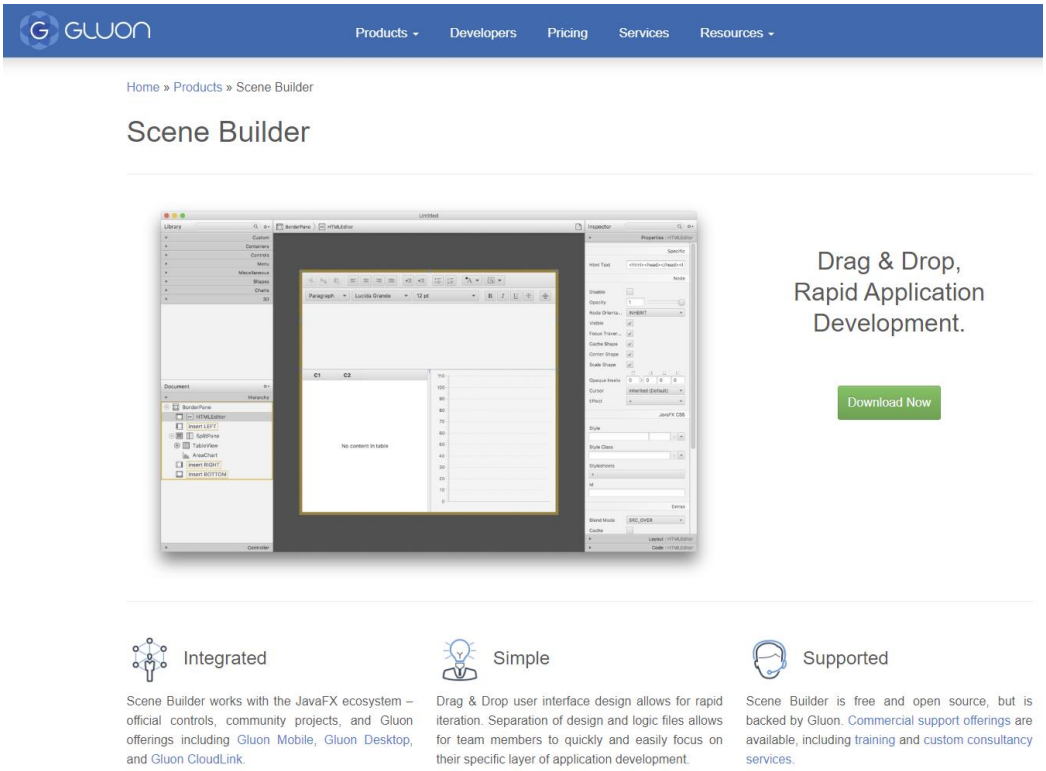
```
30      public void start(Stage vindu) {
31          try {
32              this.vindu = vindu;
33              lagScene1();
34          } catch(Exception e) {
35              e.printStackTrace();
36          }
37      } //start()
```

Denne metoden er i det vesentlige lik start()-metoden i eksempelet side 34, bortsett fra at vi nå oppretter en Button (linje 48) som legges inn i bunndelen av rotpanelet (linje 56). Knappen får en hendelse som rett og slett åpner scene2 (linje 49).

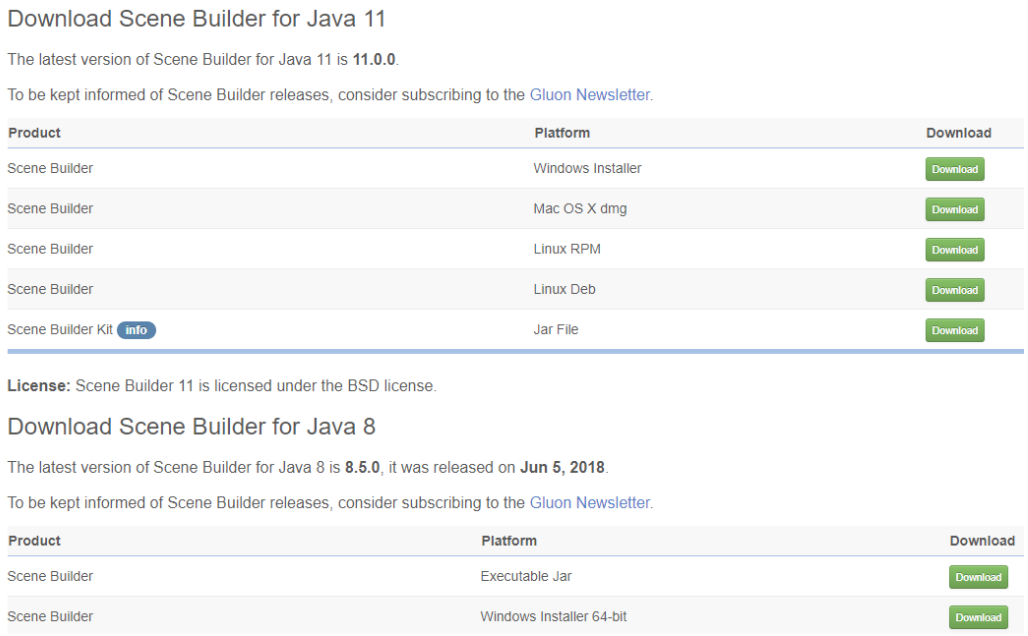
Ut fra dette eksempelet skal det nå være mulig å gå videre med å arbeide mot databaser.

Installere Scene Builder

For nedlasting, gå til denne siden: https://gluonhq.com/products/scene-builder/



Last ned Scene Builder for Java 11 hvis du har den versjonen. Hvis ikke, laster du ned Scene Builder for Java 8. Velg riktig plattform:

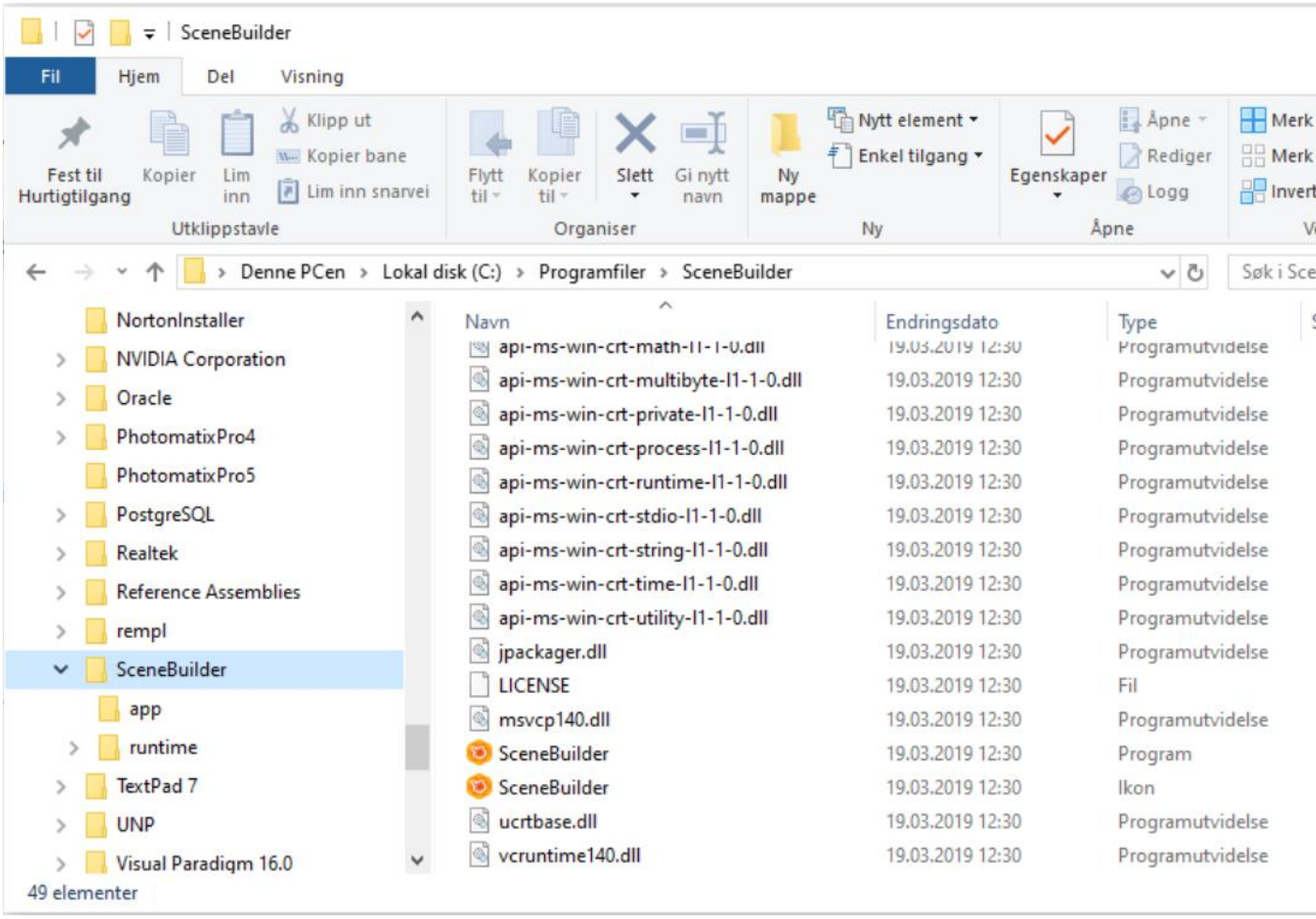


Bruk av Scene Builder med Eclipse

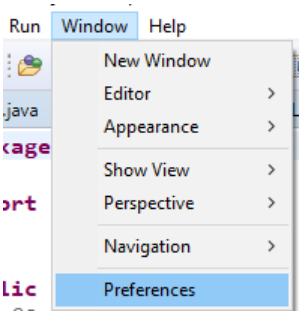
Scene Builder for JavaFX er et visuelt verktøy for å designe JavaFX brukergrensesnitt uten koding. Komponenter i brukergrensesnittet kan plasseres med «drag and drop» fra en «verktøykasse». Deretter kan egenskaper for den enkelte komponent settes. Definisjonen av brukergrensesnittet lagres som FXML-kode, en variant av XML. FXML-filen knyttes så til prosjektet. Vi skal se på flere eksempler på dette.

Når Scene Builder er installert, kan den integreres med Eclipse.

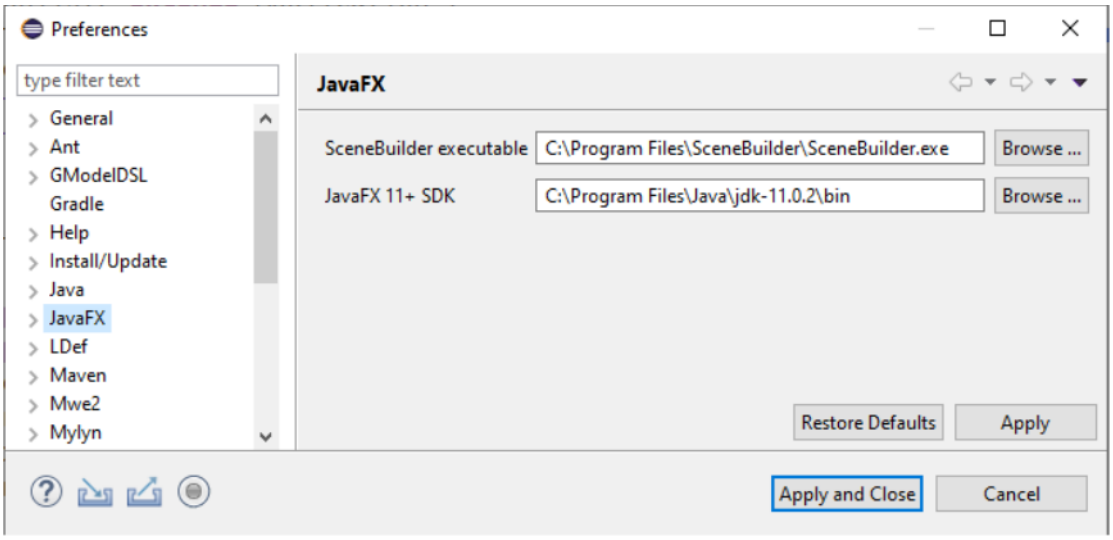
Installasjonsprogrammet installerer Scene Builder i en mappe under Programfiler. Selve programfilen ligger etter alle dll-filene:



For å koble Scene Builder til Eclipse, går vi inn på Windows-menyen i Eclipse og velger Preferences:



I vinduet vi da får opp, angir vi stien til Scene Builder:

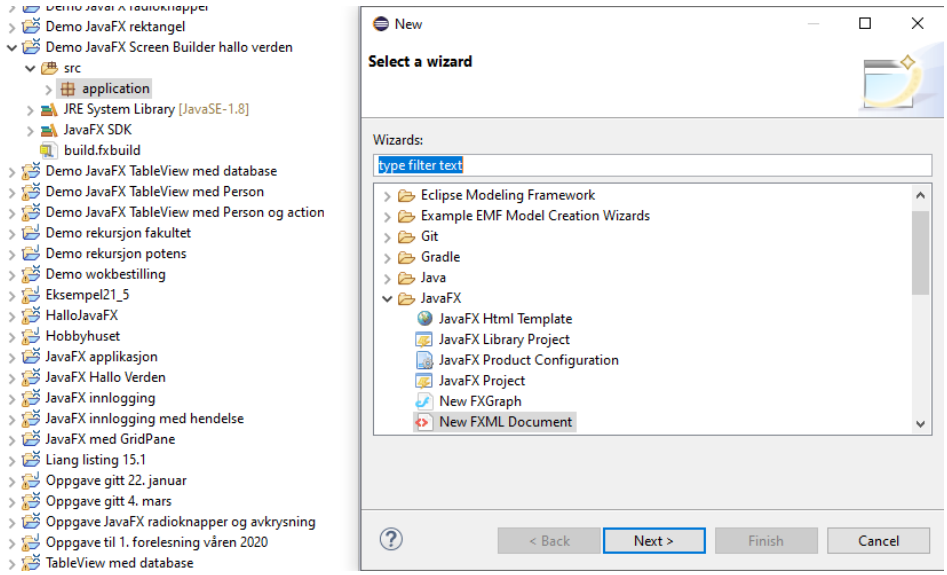


En første applikasjon med Scene Builder: Hallo verden

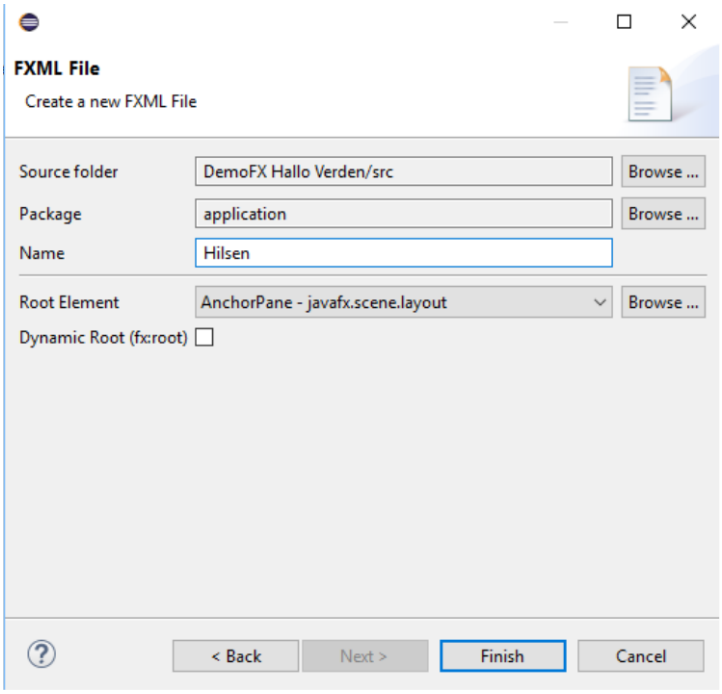
Som vanlig når man skal lære et nytt programmeringsverktøy, starter vi med en «Hallo verden» applikasjon.

Først oppretter vi et JavaFX prosjekt på vanlig måte. Her har jeg kalt det Demo JavaFX Scene Builder hallo verden.

Scene Builder bruker FXML til å definere grensesnittet (et subset av XML). Når vi nå skal lage grensesnittet med Scene Builder, åpner vi prosjektmappen, markerer application, høyreklikker og velger New | Other. Denne gang velger vi New FXML Document:



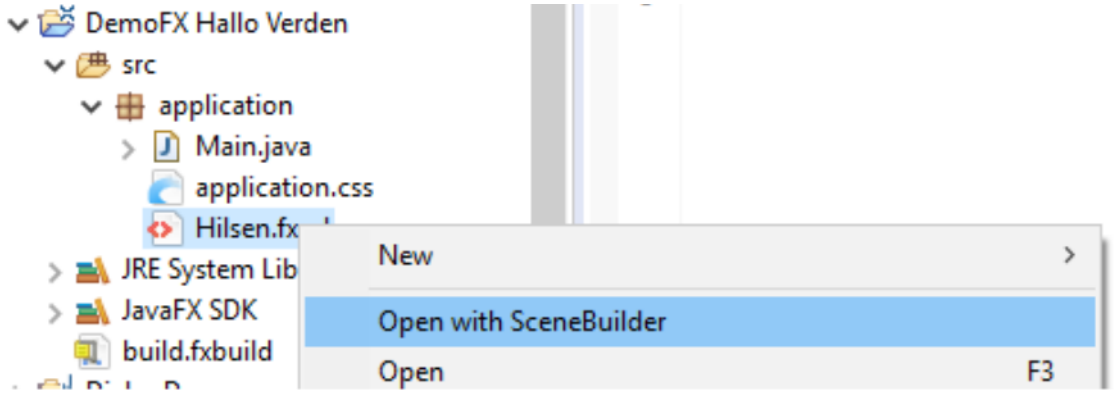
Vi gir dokumentet et navn:



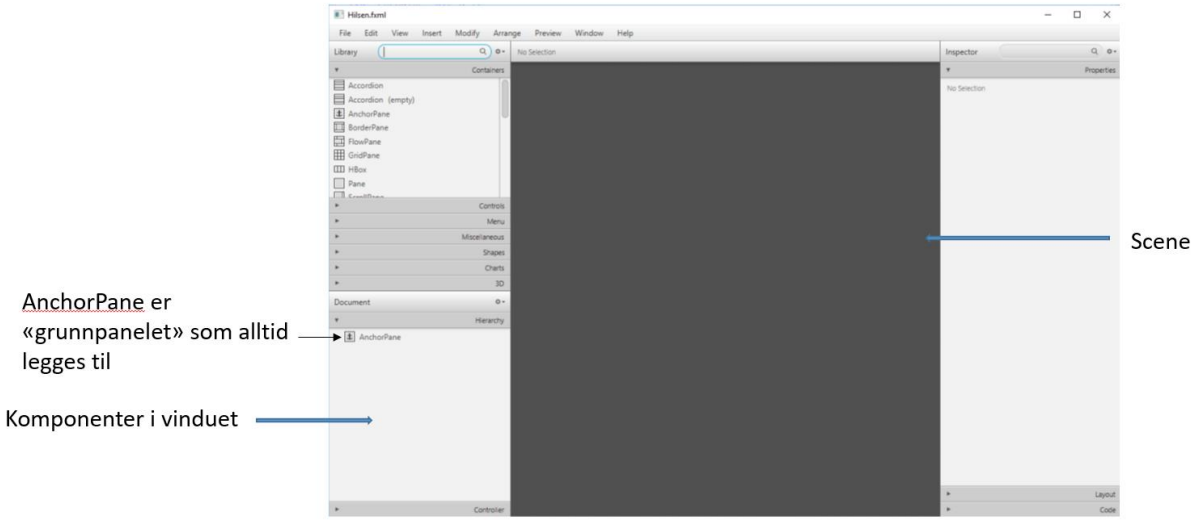
Dette genererer et FXML-dokument:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.scene.layout.AnchorPane?>
4
5  <AnchorPane xmlns:fx="http://javafx.com/fxml/1">
6      <!-- TODO Add Nodes -->
7  </AnchorPane>
8
9
```

Vi høyreklikker på dokumentet og velger Open with Scene Builder:

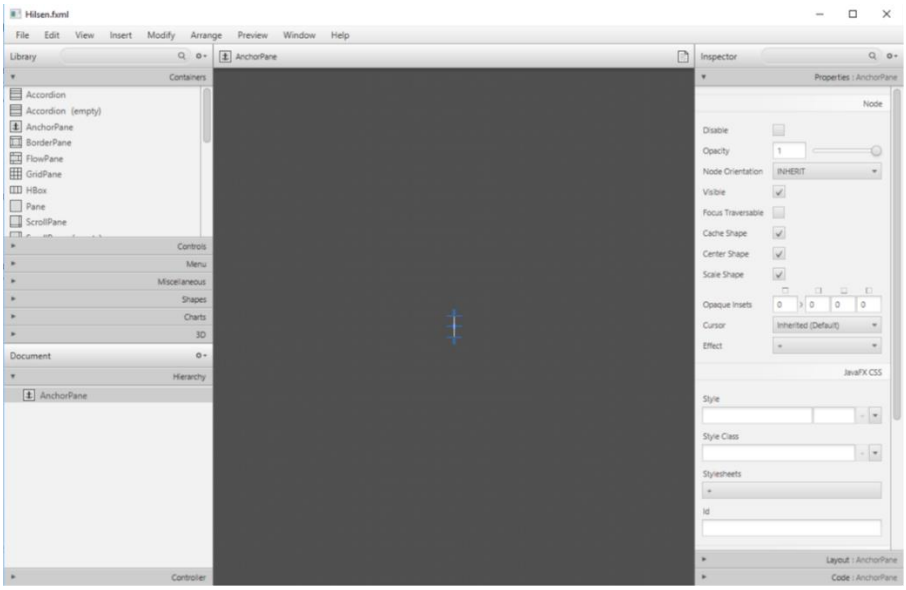


Scene Builder åpner seg slik:

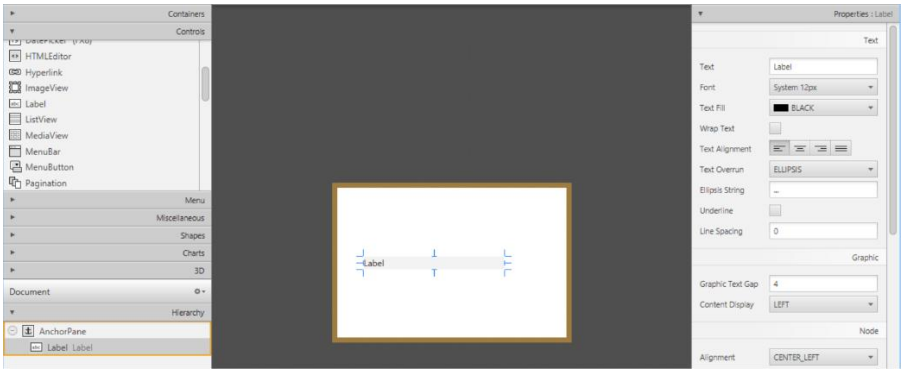


Vi kan nå bruke «drag and drop» på elementer fra verktøykassen til venstre i Scene Builder. Alt vi gjør oppdateres i FXML-dokumentet.

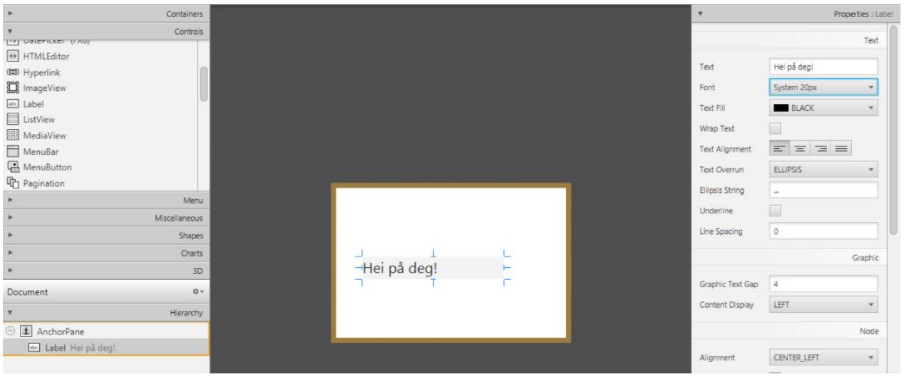
Klikker vi på AnchorPane nederst til venstre, blir et minimert panel markert med blått i scenen. Vi kan utvide panelet ved å dra i kantene:



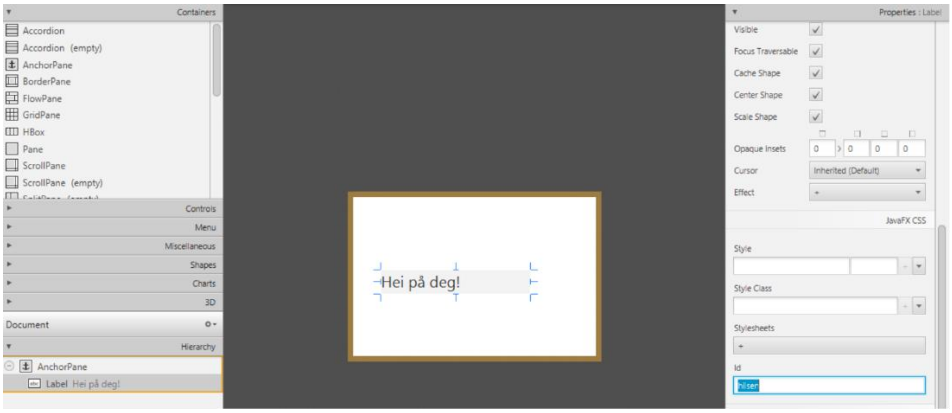
Vi kan nå legge til en Label i AnchorPane ved å dra den ut fra verktøykassen. I panelet til høyre ser vi at teksten foreløpig er «Label» (øverst til høyre):



Vi kan nå sette teksten i panelet til høyre:

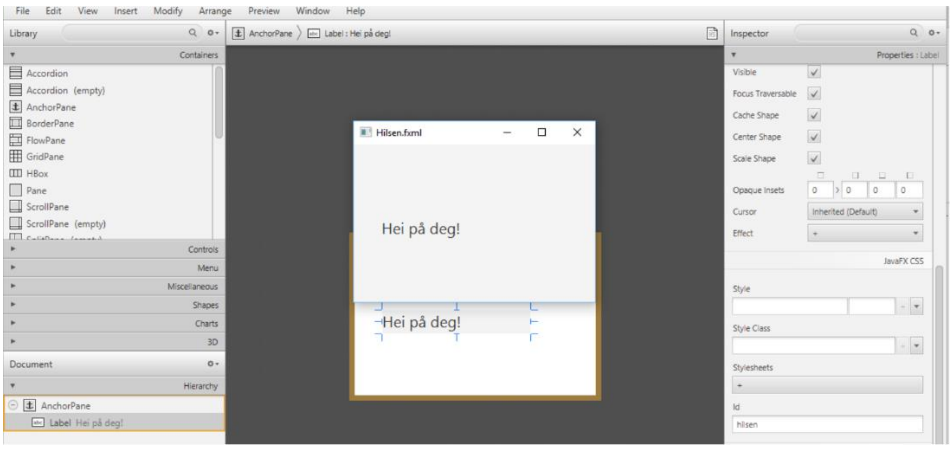


Videre gir vi labelen en ID (nederst til høyre):

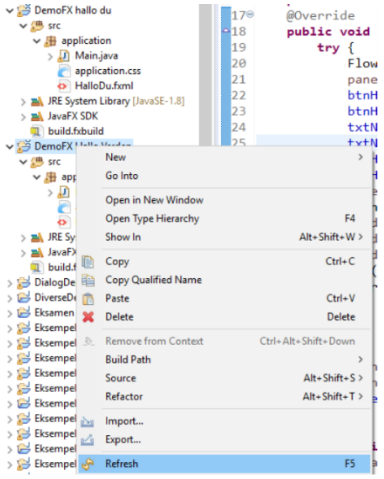


Her er id «hilsen».

Vi kan nå kjøre et Preview av applikasjonen:



Vi lagrer og velger Refresh for prosjektet. FXML-dokumentet blir da oppdatert:

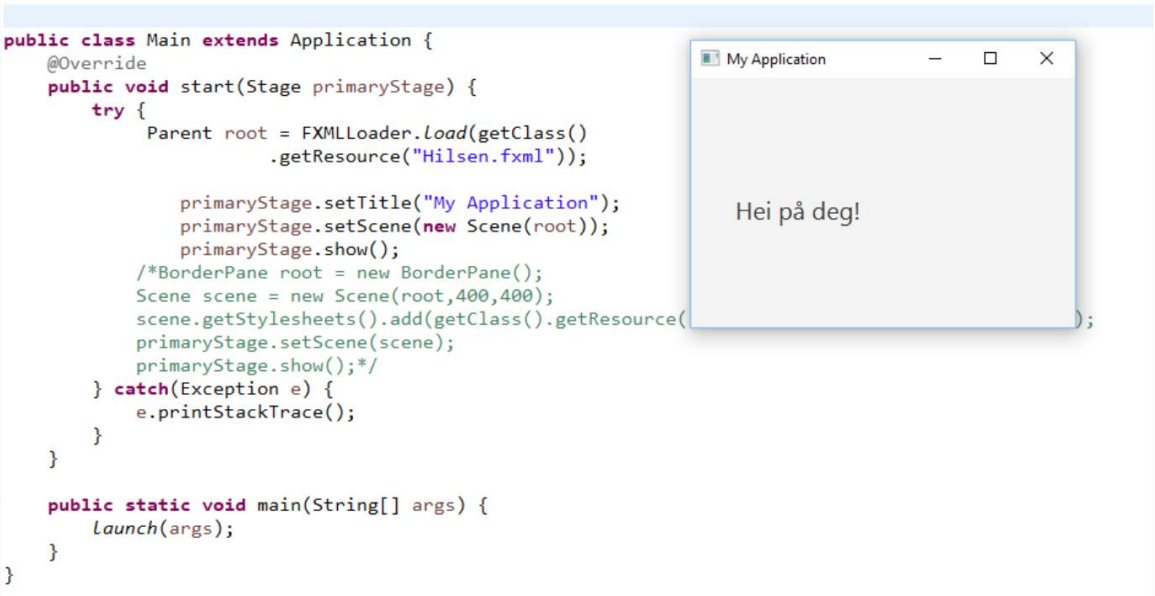



```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>

<AnchorPane prefHeight="204.0" prefWidth="313.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Label id="hilsen" layoutX="36.0" layoutY="94.0" prefHeight="17.0" prefWidth="192.0" text="Hei på deg!">
         <font>
            <Font size="20.0" />
         </font>
      </Label>
   </children>
</AnchorPane>

```

Vi går nå til Main-klassen og legger inn koden nedenfor i start(). Jeg har her bare kommentert bort den opprinnelige koden generert av Eclipse:

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
             Parent root = FXMLLoader.load(getClass()
                        .getResource("Hilsen.fxml"));

                primaryStage.setTitle("My Application");
                primaryStage.setScene(new Scene(root));
                primaryStage.show();
            /*BorderPane root = new BorderPane();
            Scene scene = new Scene(root,400,400);
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.show();*/
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

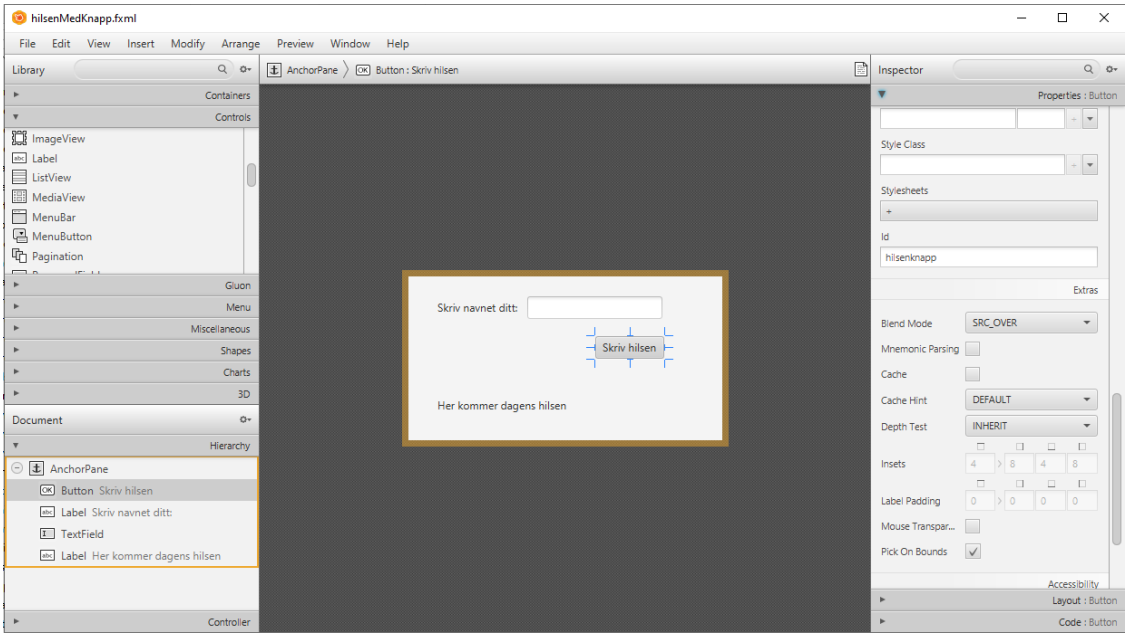
Til slutt kjører vi applikasjonen på vanlig måte:



Scene Builder: håndtere hendelser

Brukergrensesnitt er ikke mye verdt hvis de ikke kan reagere på handlinger fra brukeren. Tidligere har vi hardkodet lyttere i JavaFX. Vi skal nå introdusere FXML til å håndtere hendelser.

Vi oppretter et JavaFX-prosjekt i Eclipse som før, og oppretter en FXML-fil. Denne åpner vi så i Scene Builder og bygger grensesnittet med «drag and drop»:



De fire nodene i ankerpanelet ser vi som et hierarki under ankerpanelet (vinduet nederst til venstre). I vinduet til høyre er egenskaper (Properties) for Button'en vist. Her må vi gi hver node en id. I tilfellet Button er denne gitt id'en «hilsenknapp» (se i øvre halvdel av vinduet til høyre). De tre andre nodene har også fått sine id'er: label med teksten «Skriv navnet ditt:»: ledetekst. TextField: navn. Label med tekst «Her kommer dagens hilsen»: hilsen.

Main-klassen for prosjektet skal se slik ut:

```
11 public class Main extends Application {
12     @Override
13     public void start(Stage primaryStage) {
14         try {
15             try {
16                 Parent root = FXMLLoader.load(getClass().getResource("hilsenMedKnapp.fxml"));
17                 primaryStage.setTitle("Dagens hilsen");
18                 primaryStage.setScene(new Scene(root));
19                 primaryStage.show();
20             } catch(Exception e) {
21                 e.printStackTrace();
22             }
23         } catch(Exception e) {
24             e.printStackTrace();
25         }
26     }
27
28     public static void main(String[] args) {
29         launch(args);
30     }
31 }
```

Vi må nå lage en Controller-klasse for grensesnittet. **En slik klasse må ikke forveksles med klassene vi har kalt Kontroll tidligere** (derfor skriver jeg den med C). Controllerklassen skal kontrollere oppsettet av vinduet, mens de klassene vi har kalt Kontroll utgjør applikasjonslaget i programmene våre.

En Controllerklasse tagger FXML-elementer, som attributter om metoder. Taggingen angir at FXML-lasteren (FXML loader) kan laste nodene selv om de er deklarert som private. Taggingen kan ikke brukes på konstruktører eller klasser. Kontrollerklassen for prosjektet ser slik ut:

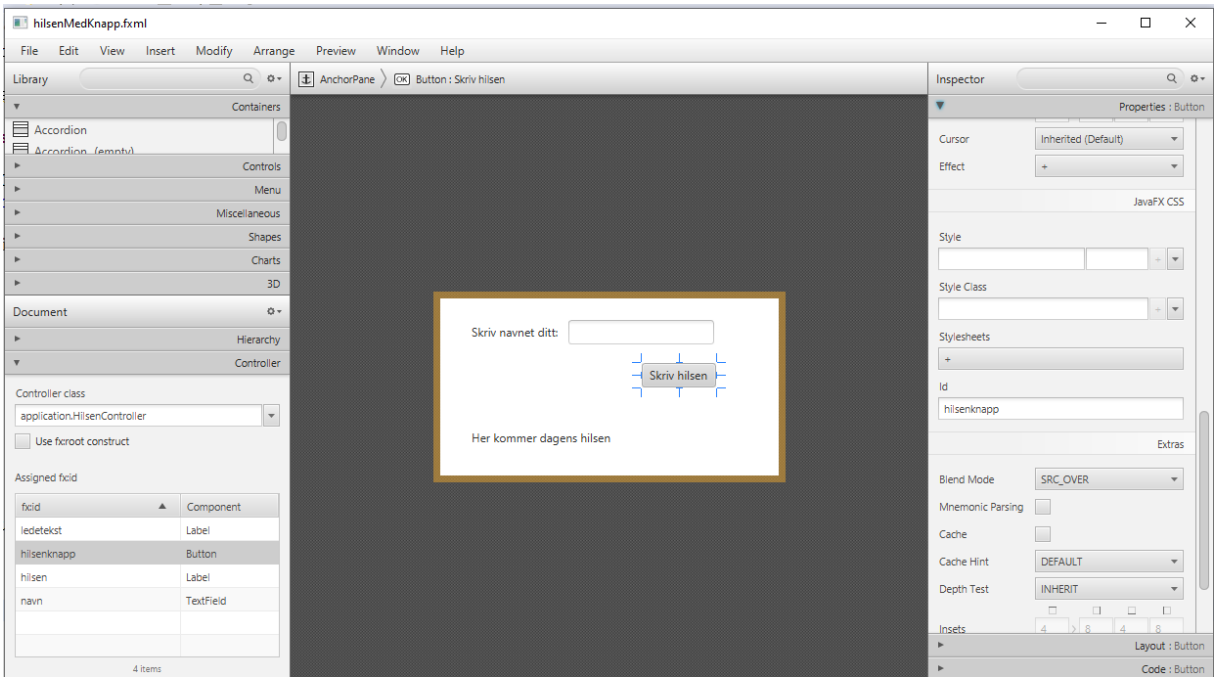
```
1  package application;
2
3  import javafx.fxml.FXML;
4  import javafx.scene.control.Button;
5  import javafx.scene.control.Label;
6  import javafx.scene.control.TextField;
7
8  public class HilsenController {
9      @FXML
10     private Button hilsenknapp;
11
12     @FXML
13     private Label ledetekst;
14
15     @FXML
16     private Label hilsen;
17
18     @FXML
19     private TextField navn;
20
21     public HilsenController() {
22         //Nødvendig for instansiering
23     }
24
25     @FXML
26     public void behandleKlikk() {
27         hilsen.setText("Hallo, " + navn.getText() + "!");
28     }
29
30     @FXML
31     public void initialize() {
32
33     }
34 }
```

Controllerklassen angir navnene på elementene vi skal ha i brukergrensesnittet, samt lytteren.

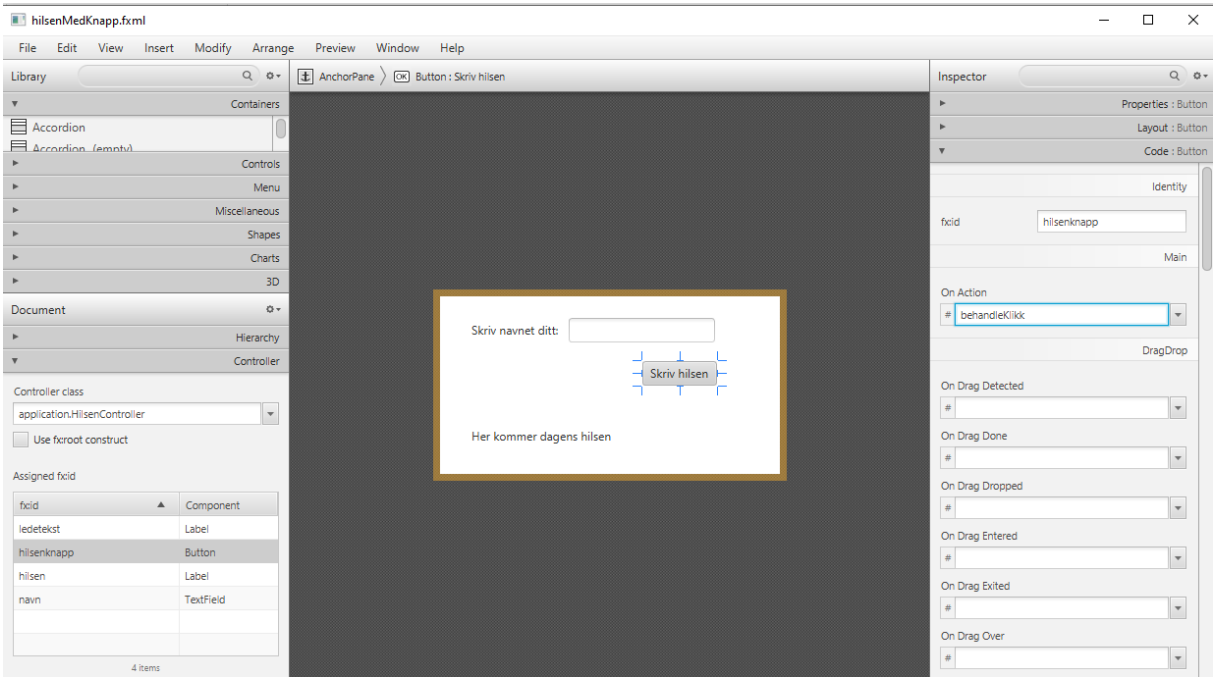
En Controller-klasse må ha en konstruktør, selv om den er tom. Uten konstruktøren kan ikke FXML-lasteren lage et objekt av klassen.

Controller-klassen kan også ha en metode kalt initialize(). Denne vil kalles av programmet etter at lastingen av FXML-dokumentet er ferdig.

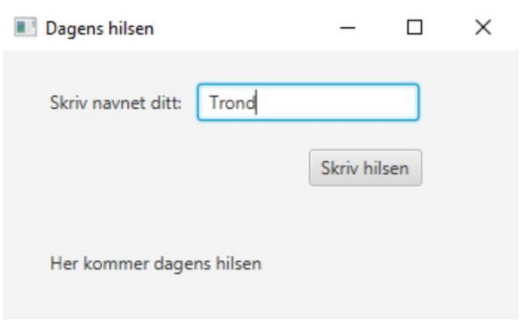
Det er FXML-koden som definerer selve grensesnittet. Det vi gjør i Scene Builder legges inn som XML-kode i FXML-filen. Vi kan nå åpne Scene Builder:

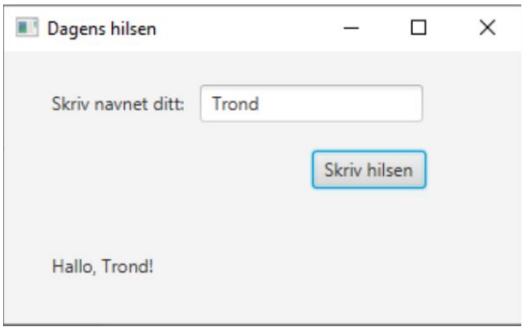


Ser vi i vinduet nederst til venstre (Controller), ser vi at her er Controller class angitt. Denne skal vi se i feltet for klassen, og kan velge den. Til høyre ser vi at id for knappen er hilsenknapp. Klikker vi på Code (nederst til høyre), kan vi angi lytteren for knappen. Denne (behandleKlikk) skal også komme opp i feltet. Pass på å ikke skrive parenteser bak metodenavnet:



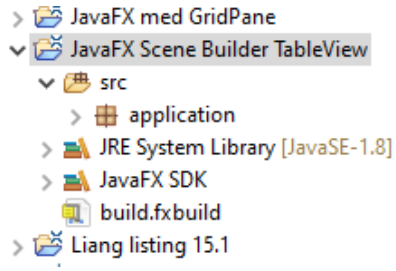
Også de andre elementene i grensesnittet må få sine id'er, som er de samme som er brukt i Controllerklassen. Når vi er ferdige med dette, lagrer vi grensesnittet. I prosjektet bør vi kjøre en Refresh (F5) for å være sikker på at FXML-filen er oppdatert. Resultatet når vi kjører programmet er dette:


Dagens hilsen
Skriv navnet ditt:
Trond
Skriv hilsen
Her kommer dagens hilsen

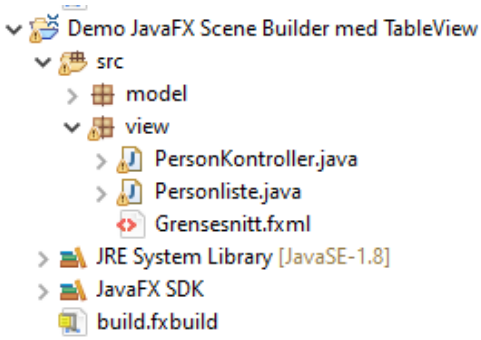

Dagens hilsen
Skriv navnet ditt:
Trond
Skriv hilsen
Hallo, Trond!

Scene Builder: lage TableView

Vi skal nå bruke Scene Builder til å lage et vindu med et TableView. Vi skal nå bygge applikasjonen vår på MVC-prinsippet (Model – View – Control), og skal ha en pakke for hver av de tre delene av programmet. Det betyr at vi sletter pakken application som Eclipse genererer:

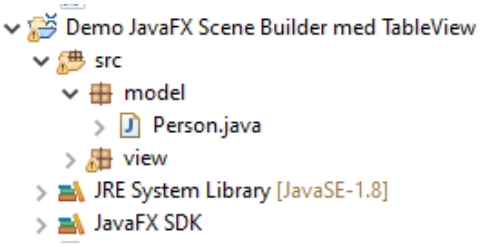


I stedet oppretter vi denne pakkestrukturen:



Pakkene er opprettet på vanlig måte med File | New | Package.

I pakken model skal vi ha klassen Person, som skal håndtere dataene om personer. Siden vi har brukt denne klassen tidligere, kopierer vi den bare inn i pakken:



Klassen Person

Før vi gjør noe annet, må vi gjøre endringer i klassen Person. JavaFX TableView er laget slik at vi må ha objekter av en klasse som skal presenteres i tabellen. Da vi programmerte JavaFX direkte, kunne vi bruke vanlige variabler og gettere. Med Scene Builder må vi imidlertid gjøre det annerledes. Her bruker vi Property-variabler, som StringProperty (det anbefales for åvrig å gjøre det også med direkte programmering med JavaFX. Person-klassen ser etter endringene slik ut:

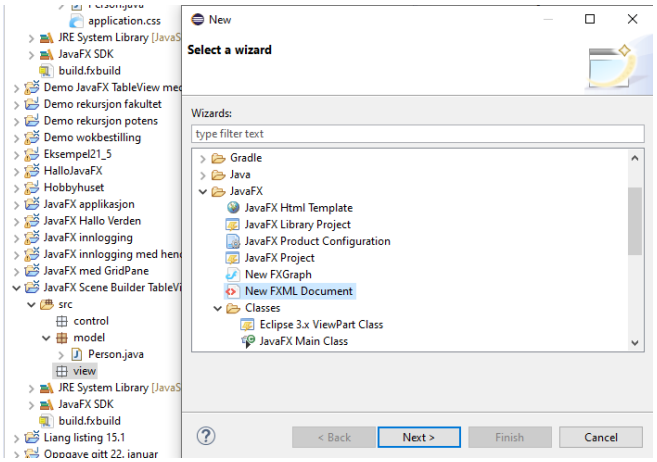
```java
public class Person {
    private final StringProperty navn;
    private final StringProperty adresse;
    private final StringProperty telefon;

    public Person(String navn, String adresse, String telefon) {
        this.navn = new SimpleStringProperty(navn);
        this.adresse = new SimpleStringProperty(adresse);
        this.telefon = new SimpleStringProperty(telefon);
    }

    public String getNavn() {
        return navn.get();
    }

    public StringProperty navnProperty() {
        return navn;
    }

    public String getAdresse() {
        return adresse.get();
    }

    public StringProperty adresseProperty() {
        return adresse;
    }

    public String getTelefon() {
        return telefon.get();
    }

    public StringProperty telefonProperty() {
        return telefon;
    }

}
```

Grunnleggende oppsett uten data i tabellen

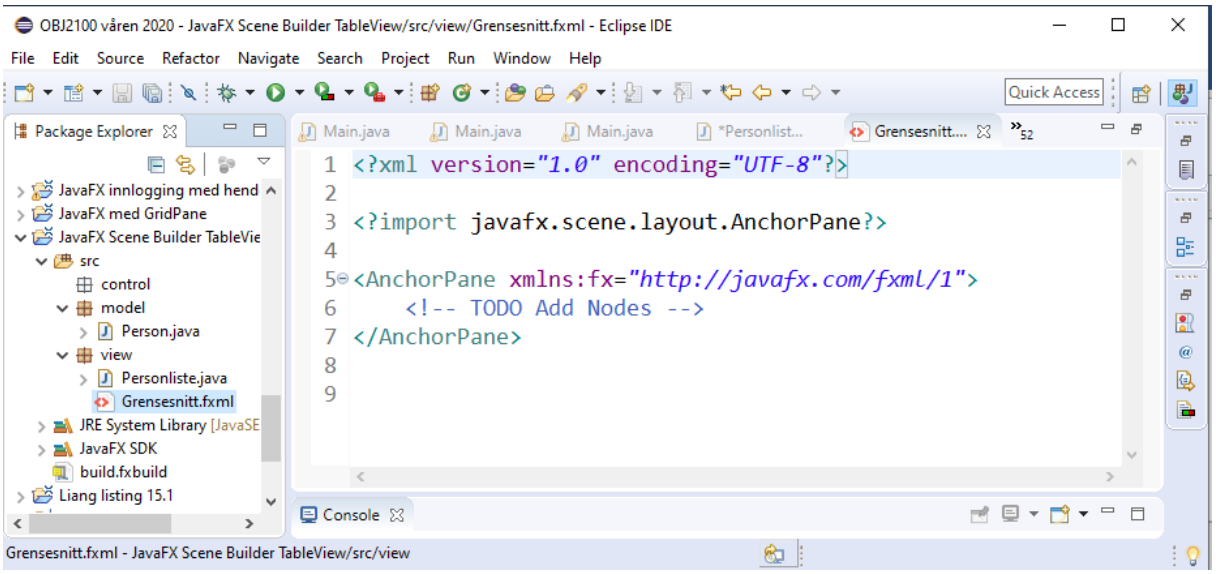
Applikasjonen skal i første omgang vise dette vinduet:

Første trinn er å opprette et FXML-dokument. Det er to måter å lage grafiske grensesnitt på med JavaFX: programmere fra bunnen av slik vi har gjort tidligere, eller bruke XML. Scene Builder er basert på at grensesnittet er spesifisert i XML.

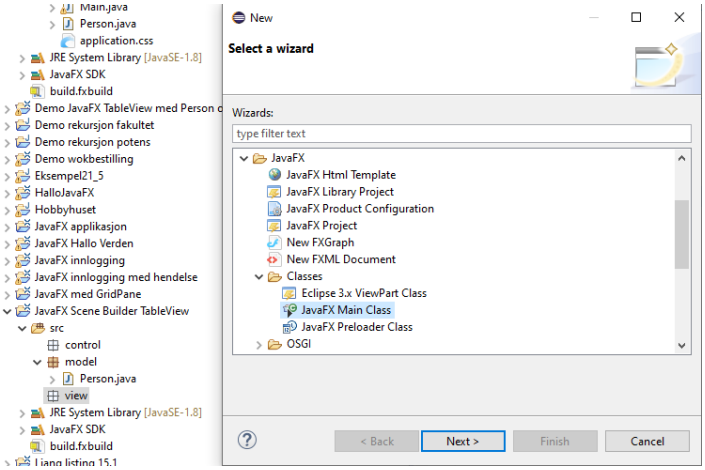
Vi høyreklikker på pakken view og velger New | Other | New FXML Document:



Vi kaller dokumentet Grensesnitt:



I pakken view skal vi også ha Main-klassen. FXML-dokumentet som brukes av Scene Builder definerer bare utseendet til grensesnittet. Selve opprettelsen av det må vi ha Main for å gjøre. Vi går frem som for FXML-dokumentet og velger JavaFX Main Class:



Her kan vi også velge å gi klassen et annet navn enn Main (men den er fortsatt en Main-klasse, noe som betyr at det er den som inneholder main()-metoden). Main-klassen opprettes, men med en litt mer minimalistisk kode enn den vi får når vi oppretter et nytt prosjekt:

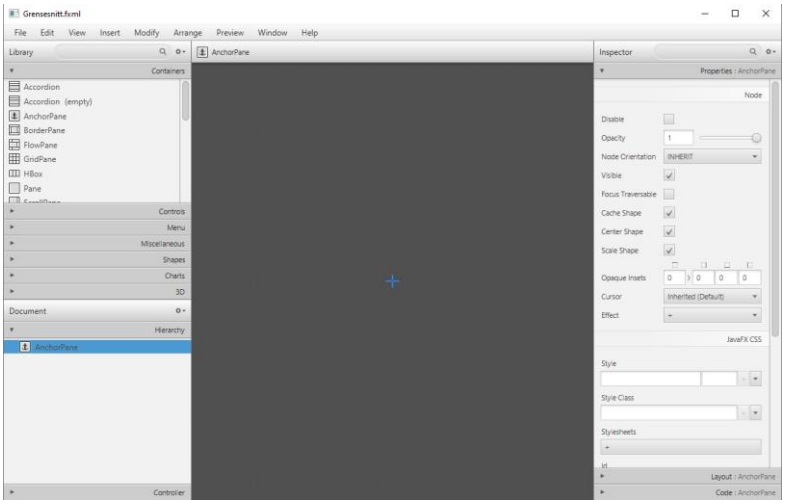
```
1  package view;
2
3  import javafx.application.Application;
5
6  public class Personliste extends Application {
7
8      @Override
9      public void start(Stage primaryStage) {
10
11     }
12
13     public static void main(String[] args) {
14         launch(args);
15     }
16 }
```

Vi legger inn den samme koden som i «hallo verden» applikasjonen:

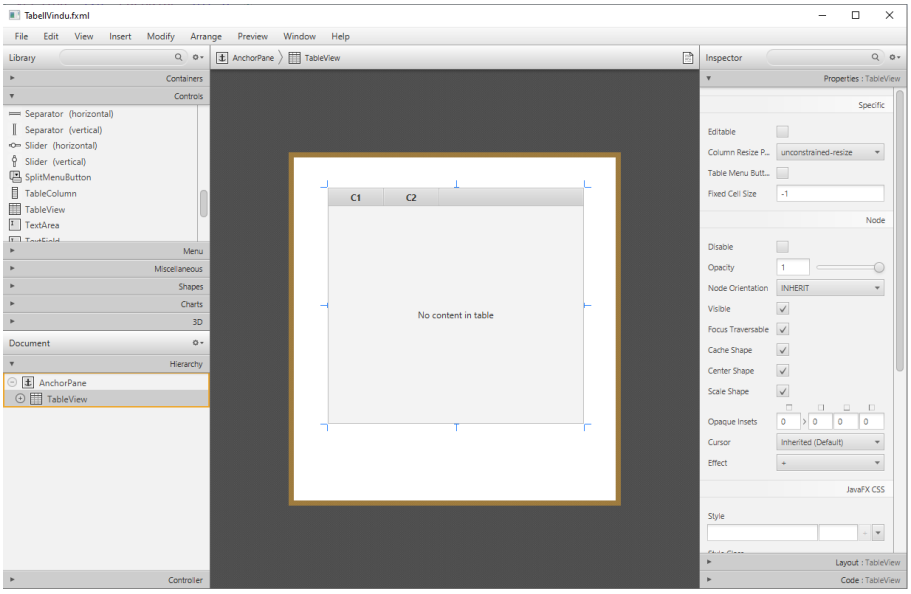
```
9  public class Personliste extends Application {
10
11     @Override
12     public void start(Stage primaryStage) {
13         try {
14             Parent root = FXMLLoader.load(getClass().getResource("Grensesnitt.fxml"));
15             primaryStage.setTitle("Personliste");
16             primaryStage.setScene(new Scene(root));
17             primaryStage.show();
18         } catch(Exception e) {
19             e.printStackTrace();
20         }
21     }
22
23     public static void main(String[] args) {
24         launch(args);
25     }
26 }
```

Strengt tatt er ikke try .. catch-blokken nødvendig.

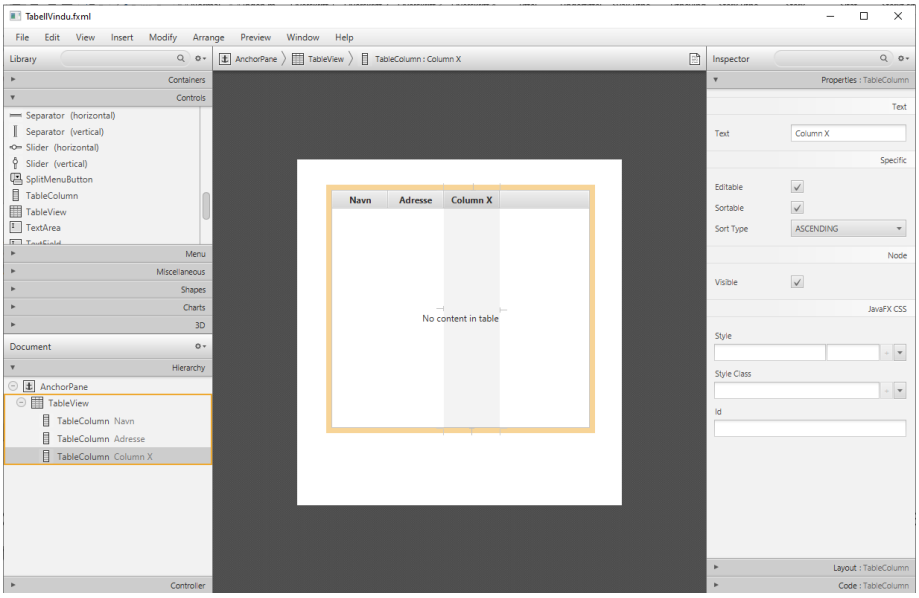
Ny kan vi åpne FXML-dokumentet i Scene Builder. Høyreklikk på dokumentet og velg Open with Scene Builder. Det mørkegrå feltet markerer scenen:



Scene Builder bruker et såkalt ankerpanel (AnchorPane) som rotpanel. Når vi markerer AnchorPane i vinduet nederst til venstre, vises det som et blått kors i scenen. Utvid det til ønsket størrelse. Deretter drar vi et TableView ut i ankerpanelet. Nå skal Scene Builder se slik ut:

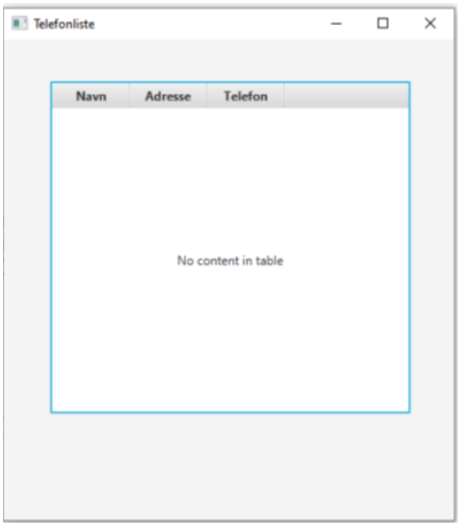


Vi ser at tabellen opprettes med to kolonner. Vi kan selvsagt legge til nye kolonner, og forandre overskriftene. Vi skal i første omgang lage en telefonliste identisk med den vi har laget tidligere:



Her er C1 og C2 omdøpt til navn og adresse, og en ny kolonne er lagt til (dra en TableColumn ut fra verktøykassen). Scene Builder har gitt denne navnet Column X. Vi forandrer dette til Telefon.

Når vi kjører programmet, får vi dette vinduet:



Fylle tabellen med data

Før vi fortsetter, må vi tilbake til main-klassen (Personliste). Som i et tidligere eksempel legger vi til en ObservableList med data:

```
14 public class Personliste extends Application {
15     ObservableList<Person> data = FXCollections.observableArrayList(new Person(
16             "Ole Brum", "Hubdremeterskogen", "123"),
17             new Person("Nasse Nøff", "Hundremeterskogen", "321"));
18     Stage primærStage;
19     AnchorPane rotlayout;
```

Her er også Stage'en (nå kalt primærStage) og den grafiske applikasjonens AnchorPane deklarert. Dette fordi de skal brukes av metoder vi skal skrive senere.

I pakken view skal vi nå lage en kontroller (vi må ha den i pakken view for at Scene Builder skal finne den). En kontroller i denne sammenheng må ille forveksles med kontrollklasser som inneholder programlogikk. I denne sammenhengen vil et Controller-objekt kontrollere grensesnittet.

```
3  import javafx.fxml.FXML;
9
10 public class PersonKontroller {
11     @FXML
12     private TableView<Person> persontabell;
13
14     @FXML
15     private TableColumn<Person, String> navneKolonne;
16
17     @FXML
18     private TableColumn<Person, String> adresseKolonne;
19
20     @FXML
21     private TableColumn<Person, String> telefonKolonne;
22
23     @FXML
24     private Label navneLabel;
25
26     @FXML
27     private Label adresseLabel;
28
29     @FXML
30     private Label telefonLabel;
31
32     private Personliste personliste;
33
34     public PersonKontroller() {
35
36     }
37
38     @FXML
39     private void initialize() {
40         navneKolonne.setCellValueFactory(cellData -> cellData.getValue().navnProperty());
41         adresseKolonne.setCellValueFactory(cellData -> cellData.getValue().adresseProperty());
42         telefonKolonne.setCellValueFactory(cellData -> cellData.getValue().telefonProperty());
43     }
44
45     public void setPersonliste(Personliste personliste) {
46         persontabell.setItems(personliste.getData());
47     }
48
49 }
```

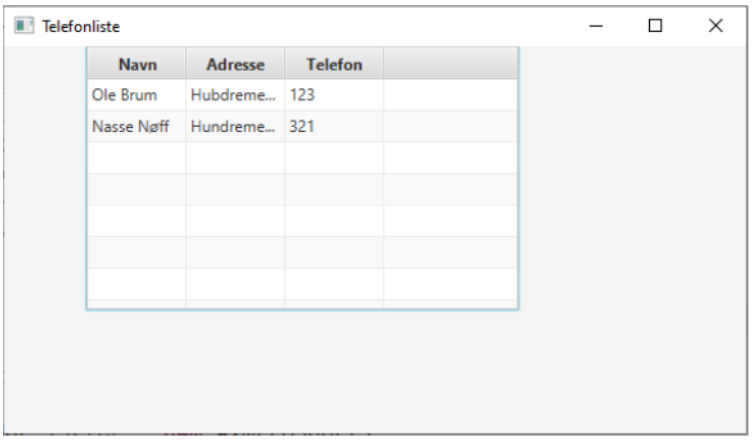
Forklaring:

- Elementene i grensesnittet må navngis her som tagger.
- Metoden initialize() kalles automatisk, og setter verdiene i kolonnene.
- Metoden setPersonliste() får en referanse til hovedprogrammet (her kalt Personliste i stedet for Main) og kaller metoden getData() for å få en referanse til hovedprogrammets ObservableList. Denne legges så til persontabellen.

Hovedprogrammet (som altså heter Personliste i stedet for Main) ser slik ut:

```
14 public class Personliste extends Application {
15     ObservableList<Person> data = FXCollections.observableArrayList(new Person(
16             "Ole Brum", "Hubdremeterskogen", "123"),
17             new Person("Nasse Nøff", "Hundremeterskogen", "321"));
18     Stage primærStage;
19     AnchorPane rotlayout;
20 
21     @Override
22     public void start(Stage primaryStage) {
23         try {
24             this.primærStage = primaryStage;
25             this.primærStage.setTitle("Telefonliste");
26             initierRotlayout();
27             visPersonKontroller();
28         } catch(Exception e) {
29             e.printStackTrace();
30         }
31 
32     }
33 
34     public void initierRotlayout() {
35         try {
36             FXMLLoader laster = new FXMLLoader();
37             laster.setLocation(Personliste.class.getResource("Grensesnitt.fxml"));
38             rotlayout = (AnchorPane) laster.load();
39             Scene scene = new Scene(rotlayout);
40             primærStage.setScene(scene);
41             primærStage.show();
42         }catch(Exception e) {e.printStackTrace();}
43     }
44 
45     public ObservableList<Person> getData()     {
46         return data;
47     }
48 
49     public Stage getPrimærStage() {
50         return primærStage;
51     }
52 
53     public void visPersonKontroller() {
54         try {
55             FXMLLoader laster = new FXMLLoader();
56             laster.setLocation(Personliste.class.getResource("Grensesnitt.fxml"));
57             AnchorPane personoversikt = (AnchorPane)laster.load();
58             rotlayout.getChildren().addAll(personoversikt);
59             PersonKontroller kontroll = laster.getController();
60             kontroll.setPersonliste(this);
61         }catch(Exception e) {}
62     }
63 
64     public static void main(String[] args) {
65         launch(args);
66     }
67 }
```

Nå kan vi kjøre programmet, og det ser da slik ut:



Litteratur

Dette kompendiet er basert på egne programmer og forelesninger. Eksemplene er egenutviklede, men flere kilder er brukt til oppslag. Nedenfor er vist noen av disse.

Bøker

Istad, Roy M. og Bjørn Kristoffersen: Forstå programmering med Java, 2. utgave. Universitetsforlaget 2019.

Kristoffersen, Bjørn: Databasesystemer, 5. Utgave. Universitetsforlaget 2020.

Liang, Y. Daniel: Introduction to Java Programming and Data Structures. 11th edition, Pearson Education 2019.

Nettressurser

O7planning: JavaFX Programming Tutorials

https://o7planning.org/en/11009/javafx

Oracle: JavaFX Documentation Home

https://docs.oracle.com/javafx/2/

Pomarolli, Andreas: JavaFX Scene Builder Tutorial.

https://examples.javacodegeeks.com/desktop-java/javafx/scene/javafx-scene-builder-tutorial/

Skriftserien nr. 41
2020
—

JavaFX
Med Eclipse og Scene Builder
—

Forfattere: Trond R. Braadland
—

ISBN 978-82-7860-432-8
ISSN 2535-5325

usn.no