App-programmering med FXML

God måte å få inn objektorientert tankegang

Oversikt over forelesningen

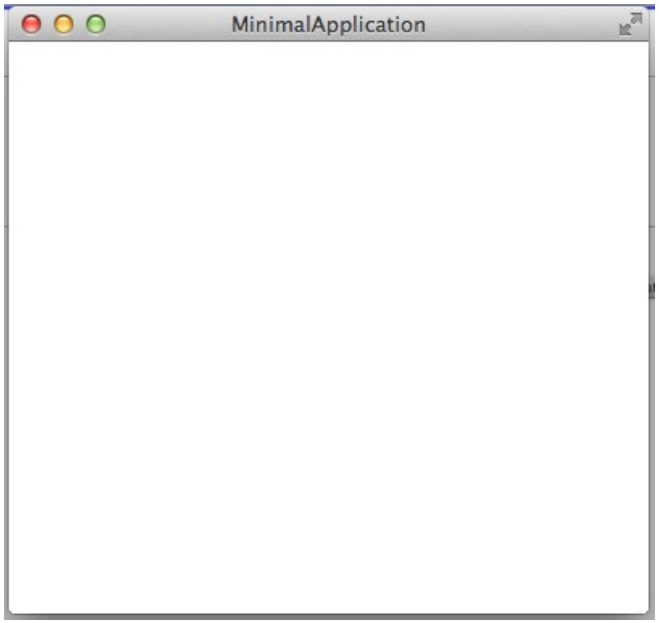
- Kjøring/oppstart av JavaFX-applikasjoner
- Strukturer av grafiske og interaktive elementer
  - Container-klasser og layout
  - Noder og nodehiarki
- Interaktivitet
  - lyttere og hendelser
- Definisjonsfiler for JavaFX grensesnitt: fxml-filer og controller-klasser

Oppsett JavaFX etc

- https://www.ntnu.no/wiki/display/tdt4100/Oppsett+av+Scenebuilder+i+VSCode

Oppstart/kjøring av FX-applikasjoner

```
public class MinimalApplication extends Application {

  @Override
  public void start(Stage stage) throws Exception {

    Pane root = new Pane(); // Root of the scene graph
    Scene scene = new Scene(root, 500, 500);

    stage.setScene(scene);
    stage.setTitle("MinimalApplication");
    stage.show();
  }

  public static void main(String[] args) {
    launch(MinimalApplication.class, args);
  }
}
```



MinimalApplication

Denne gjør ikke noe annet enn hvordan en JavaFX-app startes. Vi skal så

1. Utvide appen med grafiske elementer og tilhørende interaksjon
2. Hvordan vi skal organisere koden når vi skal lage mer kompliserte apper

Nøkkelideen for punkt 3 er å skille forskjellige aspekter av koden fra hverandre med løs kobling:

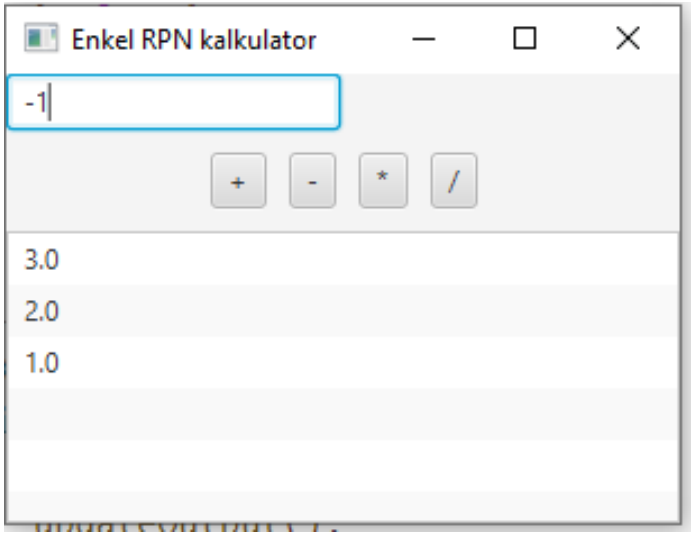
- selve applikasjonsinnhold og logikk (modell)
- den grafiske presentasjon (view)
- Interaksjon (kontroller)

Eksempel med innhold: Enkel RPN kalkis

Vi ser først på en implementasjon som har alt i en fil, SimpleRpn.java

Vi har her eksempler på

- Kontrollene TextField, Button, ListView
- Layout ved gruppering horisontalt og vertikalt (hiarkisk)
- Interaksjon med bruker via hendelser (events): Klikke på knapper, trykke enter-tasten.
- Interaksjon mellom elementene I GUI'et: Enter og knappe-trykk utfører oppgaver og oppdaterer view'et.



JavaFX grafisk oppbygging

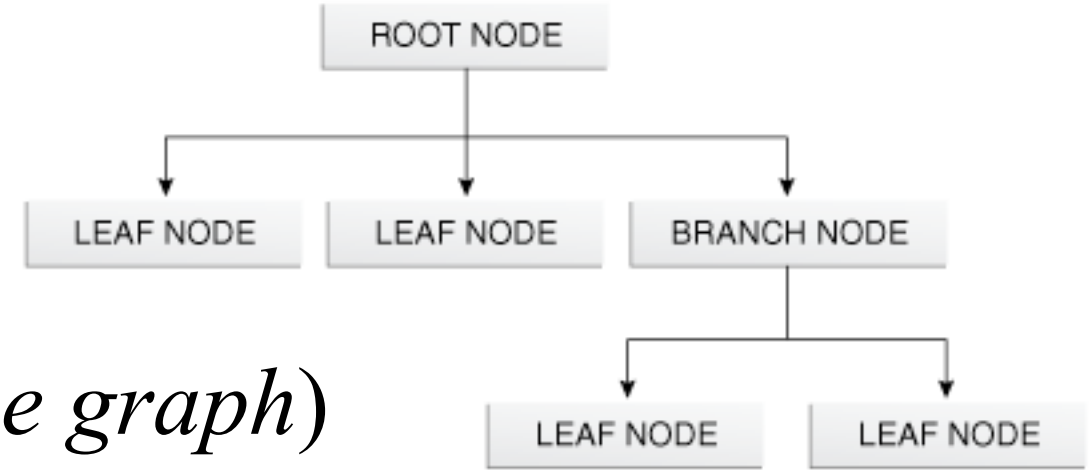
Den grafiske delen består av:

- Stages, som svarer til et vindu
- Som inneholder nøyaktig ett Scene-objekt
- Som inneholder en Node som er rot-noden til et tre av noder kalt for Scene grafen.

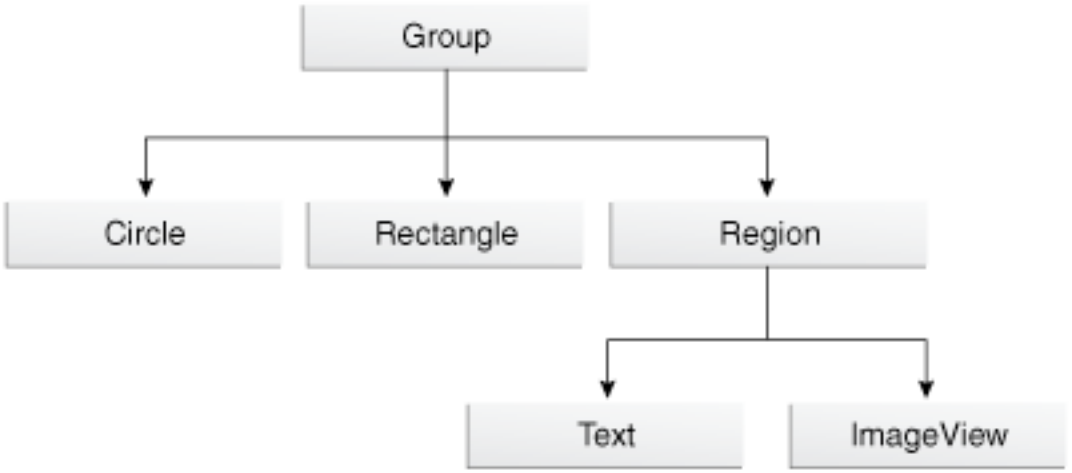
Det er Node-objektene som utgjør de grafiske elementene.

Strukturer av grafiske og interaktive elementer

generell struktur (såkalt *scene graph*)

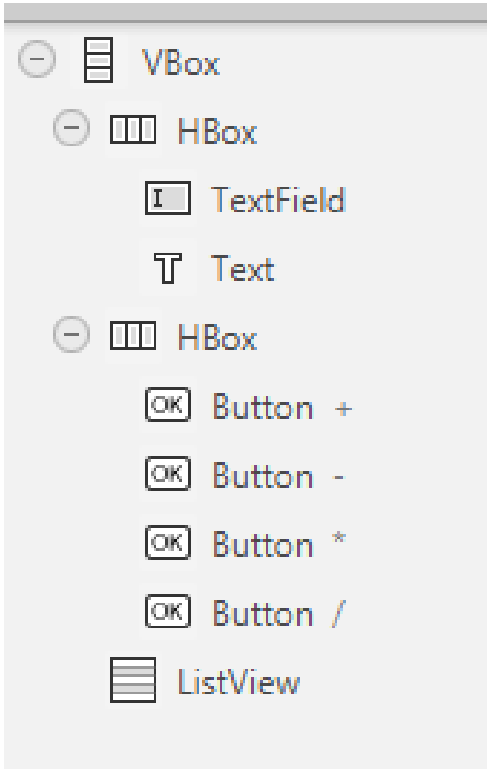


eksempel-struktur



Oppbygging av Rpn

- Hver linje er en node
- Noen noder har barn, dvs er Parent-noder



Noder

- Alle noder har en del felles egenskaper
  - id – navn som kan knyttes til noden og brukes til søk og CSS (styling av GUI'et)
  - x,y-posisjon og størrelse – settes som en kombinasjon av ulike typer transformasjon, som translasjon, rotasjon, skalering
  - forskyving – ekstra dx, dy-par

Eksempler på Noder

LAYOUTS:
organiserer andre elementer (inne i seg)

- Group
- Region
- Pane
- HBox
- VBox
- FlowPane
- BorderPane
- BorderPane
- StackPane
- TilePane
- GridPane
- AnchorPane
- TextFlow

CONTROLS:

- Button
- CheckBox
- ChoiceBox
- ColorPicker
- ComboBox
- DatePicker
- Label
- ListView
- Menu
- MenuBar
- ProgressBar
- RadioButton
- Slider
- SplitPane
- TableView
- TabPane
- TextArea
- TextField
- ToggleButton
- ToolBar
- TreeTableView

Dette er ikke komplett liste. Det finnes også digrammer/grafe-elementer, grafikk, lyd, video, webleser …

Noen noder er av type Parent og inneholder andre Node-objekter.

Interaktivitet

(http://docs.oracle.com/javafx/2/ui_controls/jfxpub-ui_controls.htm)

- Interaktive komponenter reagerer på brukerinput og brukes til å
  - styre applikasjonen
  - redigerer applikasjonsdata
- Det finnes et vell av disse, f.eks.
  - knapper (trykk, avkrysning, valg)
  - tekst (felt, paneler og editorer)
  - nedtrekksmenyer
  - lister og tabeller
  - web-innhold

Containere og layout

- Plassering av noder bestemmes av et samspill mellom container og node
  - containeren kan ha egen layout-logikk
  - noden kan ha en grafisk transformasjon som forskyver, roterer, skalerer og generelt regner ut hvor den skal være
- Container-objekter velges typisk ut fra hvilken layout-logikk de har innebygget

Nyttige containere

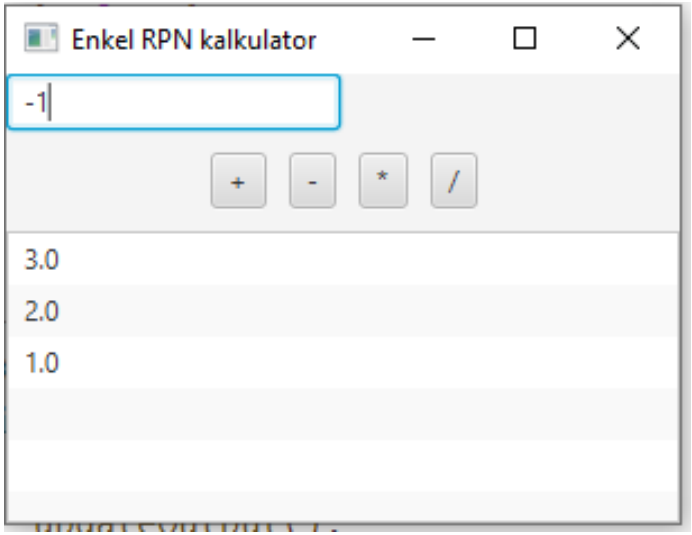
(http://docs.oracle.com/javafx/2/layout/jfxpub-layout.htm)

- HBox og VBox – horisontal eller vertikal layout
- GridPane og TilePane – rutenett-layout
- BorderLayout – fire regioner rundt midten
- AnchorLayout – elementer knyttes til en eller flere kanter
- Pane – nøytral/generell (ingen layout)

Omskriving av SimpleRpn

Nøkkelord: Løs kobling

All dataene stacken er bare lagret i viewet, og regneoperasjonene opererer direkte på disse. Det vil fort bli veldig komplisert/rotat, men også begrensende, fordi GUI-elementene har begrenset støtte for operasjoner.

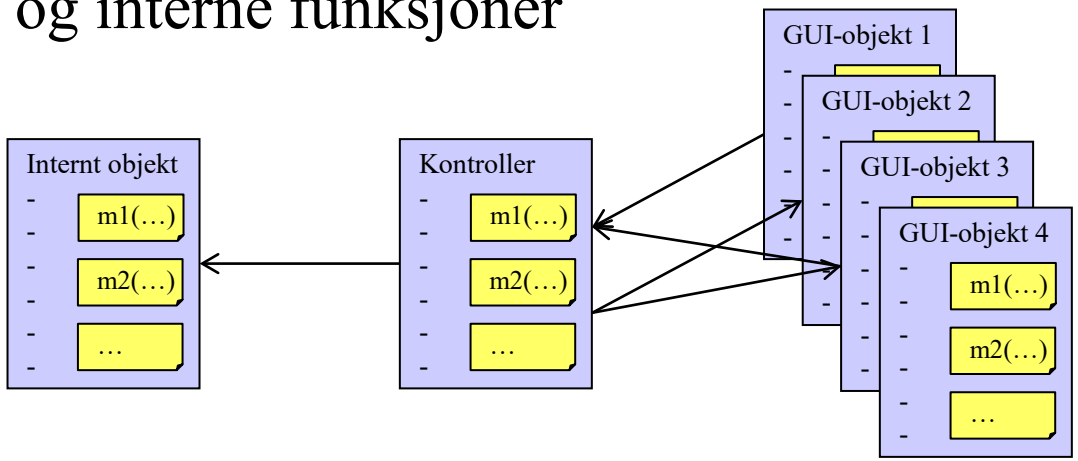


Vi vil derfor lage en implementasjon av SimpleRpn uavhengig av GUI'et, som vi kaller **modellen**.

GUI'et skal ha et minimal kjennskap til detaljer i modellen.

App-er

- (G)UI (ratt, girspak, dashbord, ...)
  - GUI-objekter – interaktive og rent grafiske
  - kontroller-objekter - koordinerer (G)UI-objekter og interne objekter
- Intern tilstand (motor)
  - objekter med app-logikk og app-data, altså intern tilstand og interne funksjoner



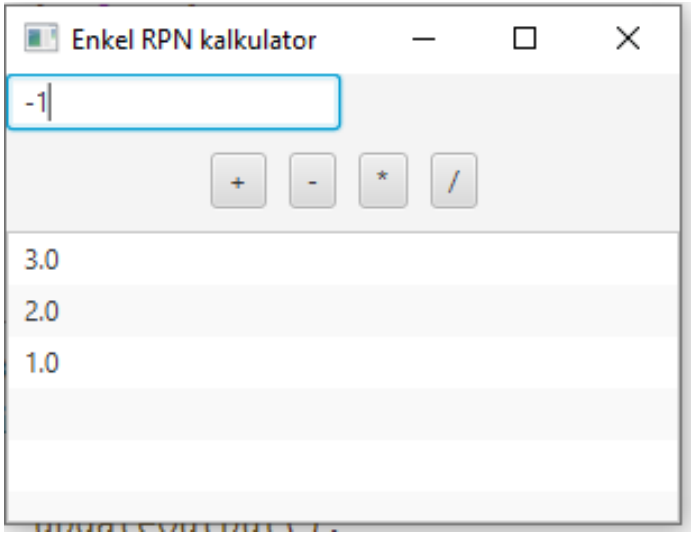
RpnModel

Vi definerer derfor en modell for kalkulatoren vår.

Den skal håndtere selvstendig alt som skal til for å ha en fungerende kalkulator, men uten et brukergrensesnitt.

Se RpnModel.java.

Vi skal nå kunne bruke RpnModel enkelt uten GUI'et, f.eks. Ved direkte kall eller tekst/terminalbasert.



FXML

- Strukturer av grafiske og interaktive elementer (noder) kan beskrives av såkalte *brukergrensesnittdefinisjonsfiler*
- Dette er XML-filer av typen FXML med generell syntaks for å
  - bygge opp et hierarki av instanser
  - sette attributter
- Forenkler koden betraktelig:
  - Skiller kontrol-logikk fra node-hierarkiet
  - Gjør det mulig å endre mye av grensesnittet uten å endre koden
- Grensesnittet initialiseres ved å:
  - laste inn FXML-fila og bygge node-hierarkiet
  - søke frem relevante elementer og legge på lyttere

GUI-elementer og FXML

- GUI-elementer
  - ren grafikk – tekst, streker, rektangler og andre figurer
  - interaktive elementer – tekstfelt, knapper, lister osv.
  - grupperingselement – layout, f.eks. horisontalt, rutenett
- FXML
  - opprettes kankje enklest ved å kopiere inn en eksisterende…
  - redigeres som tekst i VS Code
  - grafisk med SceneBuilder
  - kan kobles sammen, men ingen integrasjon…

Java vs. FXML

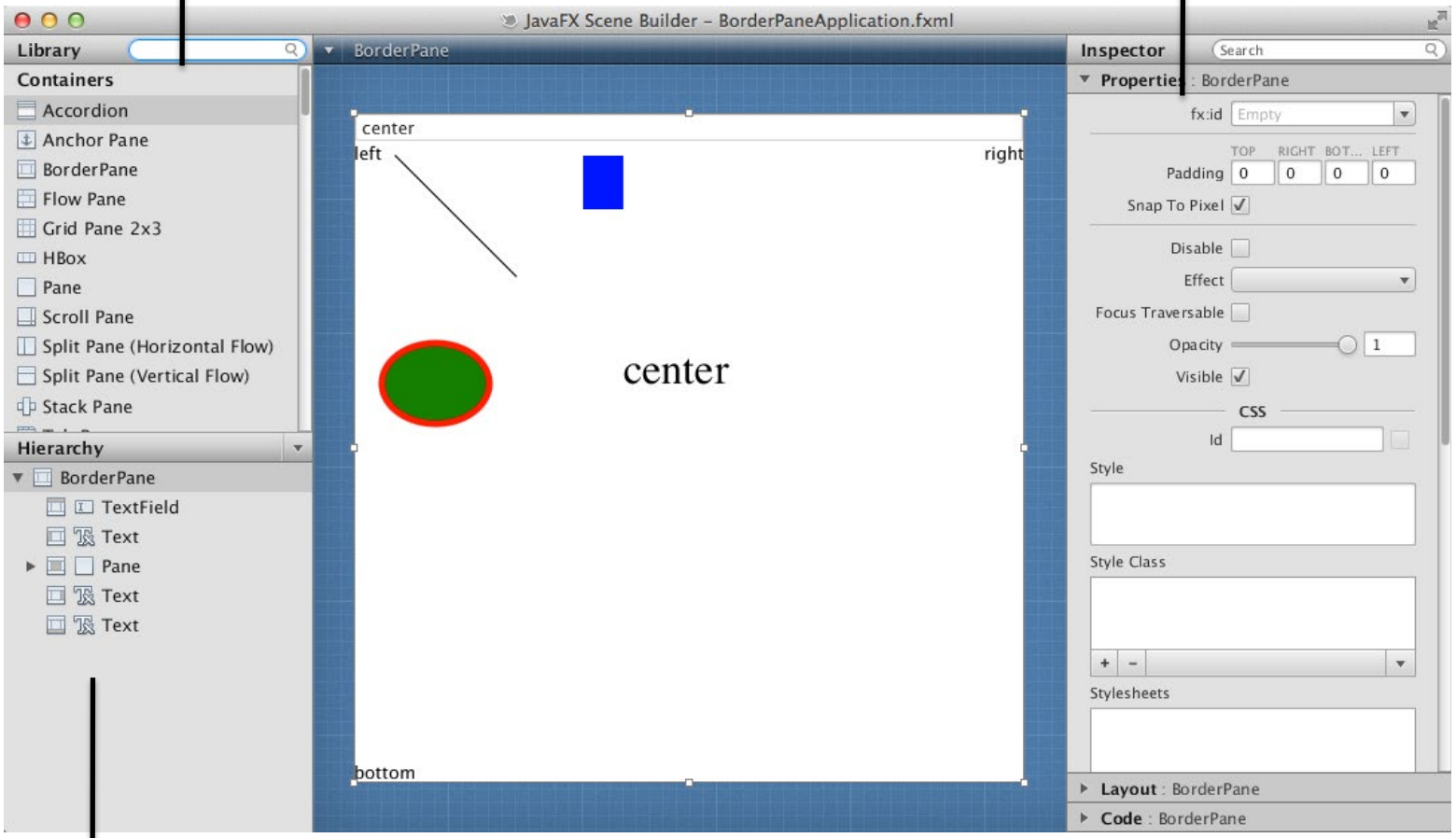
```
Pane root = new Pane(); // Root of the scene graph
Rectangle rect1 = new Rectangle(50, 40, 20, 30);
Line line1 = new Line(10, 10, 50, 40);
root.getChildren().add(line1);
root.getChildren().add(rect1);

Pane pane = new Pane(); // Root of the scene graph
Rectangle rect2 = new Rectangle(50, 40, 20, 30);
Line line2 = new Line(10, 10, 50, 40);
pane.getChildren().add(line2);
pane.getChildren().add(rect2);

pane.setLayoutX(200);
pane.setLayoutY(100);
pane.setScaleX(2);
root.getChildren().add(pane);
```

```
<Pane xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8">
        <Line endX="50" endY="40" startX="10" startY="10" />
        <Rectangle height="30" layoutX="50" layoutY="40" width="20" />
        <Pane layoutX="200" layoutY="100" scaleX="2">
                <Line endX="50" endY="40" startX="10" startY="10" />
                <Rectangle height="30" layoutX="50" layoutY="40" width="20" />
        </Pane>
</Pane>
```

SceneBuilder – leser og skriver FXML

klassepalett

properties



node-hierarki

FXML

- FXML lastes inn og oversettes til scene graph vha. en FXMLLoader:

```
@Override
public void start(Stage primaryStage) throws IOException {
  FXMLLoader fxmlLoader = new FXMLLoader();
  Parent root = (Parent) fxmlLoader.load(this.getClass().getResourceAsStream("Example.fxml"));
  primaryStage.setScene(new Scene(root));
  primaryStage.show();
}
```

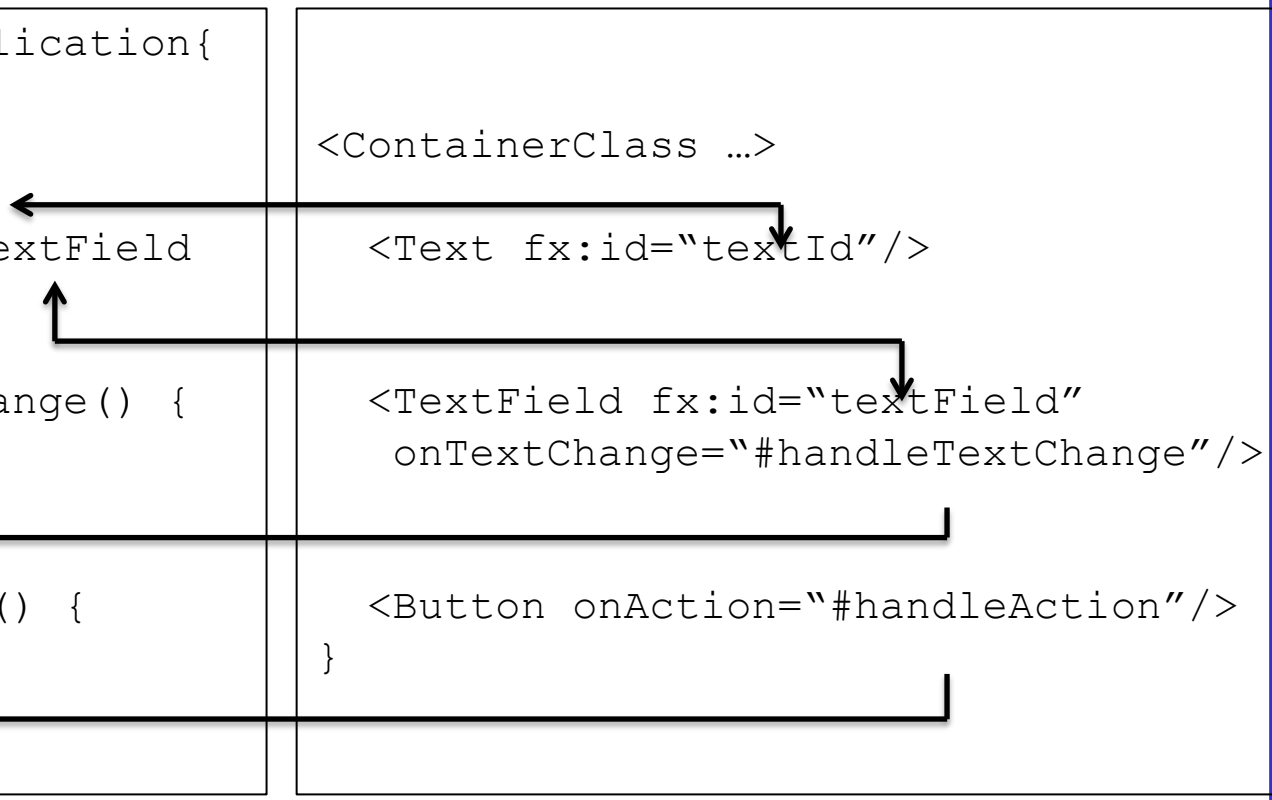
UI-kontrollere

- App-en må kunne reagere på input og endre UI-et
- Objektet som gjør det kalles en “Controller”
  - controlleren kan angis i app-klassen med FXMLLoader.setController(this)
- FXMLLoader-en kobler sammen view og controller
  - setter referanser i controller til view-objekter
  -
    ```
    <Text fx:id="textId" ... />
    @FXML Text textId;
    ```
  - knytter controller-metoder til UI-hendelser
  -
    ```
    <TextField ... onTextChange="#handleTextChange"/>
    @FXML handleTextChange(Observable...) { ... }

    <Button ... onAction="#handleAction"/>
    @FXML handleAction() { ... }
    ```

Controller og FXML

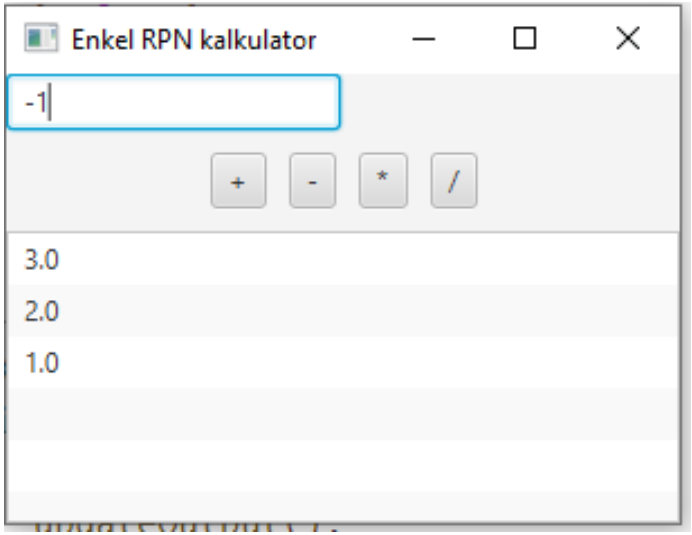
```
class … extends Application{


  @FXML Text textId
  @FXML TextField textField

  @FXML
  void handleTextChange() {
    ...
  }
 @FXML
  void handleAction() {
    ...
  }
}
```

```
<ContainerClass …>

  <Text fx:id="textId"/>

  <TextField fx:id="textField"
   onTextChange="#handleTextChange"/>

  <Button onAction="#handleAction"/>
}
```



FXML for Rpn

Vi skal nå se på hvordan vi kan bruke en FXML definisjonsfil for å generere GUI-komponentene til Rpn-appen vår.



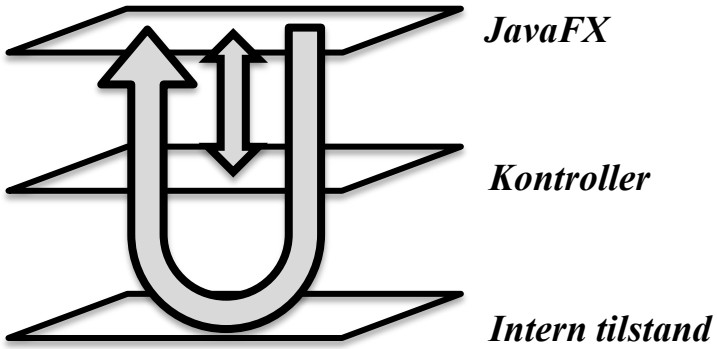
Se

- Rpn.fxml
- RpnController.java
- RpnApp.java

Kontroller og FXML

- Kjøring av FXML
  - det opprettes JavaFX-objekter tilsvarende hvert element (tag)
  - hvert JavaFX-objekt sin oppførsel styres av attributtene
  - men hva med koblingen til den app-spesifikke interne tilstanden? det er kontrolleren sin oppgave!
- Kontrolleren må kunne
  - reagere på brukerinteraksjon, f.eks. knappetrykk
  - oppdatere GUI-objektene, og trenger derfor referanser til dem (altså de som skal kunne oppdateres)
  - FXML har spesielle koder for rigging av kontrolleren

Kontroller-logikk



- Reagere på brukerinteraksjon
  - typisk handleXYZ-metoder
  - henter ut innfylte data med get-metoder, konverterer evt. til ønskede typer
  - kaller metoder på interne objekter
- Oppdatere
  - leser ut ny tilstand med get-metoder
  - oppdaterer GUI-objekter vha. set-metoder

Rigging av kontroller

- Kontroller-klasse
  - **fx:controller**-attributt angir kontroller-klasse
  - kontroller-objekt opprettes automagisk
- Trigging ved brukerinteraksjon:
  - FXML-attributt på element: **onXYZ="#metode"**
  - Java-metode i kontroller: **@FXML void metode() { ... }**
- Referanse til JavaFX-objekt
  - FXML-attributt på element: **fx:id="attributt"**
  - Java-attributt i kontroller: **@FXML Type attributt;**
  - settes automagisk etter opprettelse av kontroller
- Initialisering
  - Java-metode i kontroller: **@FXML void initialize() { ... }**
  - kalles automagisk etter at referansene er satt

Initialisering av intern tilstand

- Konstruktør
  - (tom) konstruktør brukes når kontroller-objekt opprettes
  - kalles for tidlig til at GUI-objektene kan nås og brukes
  - kan brukes til initialisering av intern tilstand
- **@FXML void initialize() { ... }**
  - kalles automagisk etter at GUI er satt opp, merk at dette er spesifikt for FXML-applikasjoner
  - kan brukes til initialisering av intern tilstand
  - brukes til å oppdatere GUI-objektene, slik at de stemmer med den initielle interne tilstand
- Vi kan med andre ord initialisere objektene, men ikke vise tilstanden i GUI uten initialize