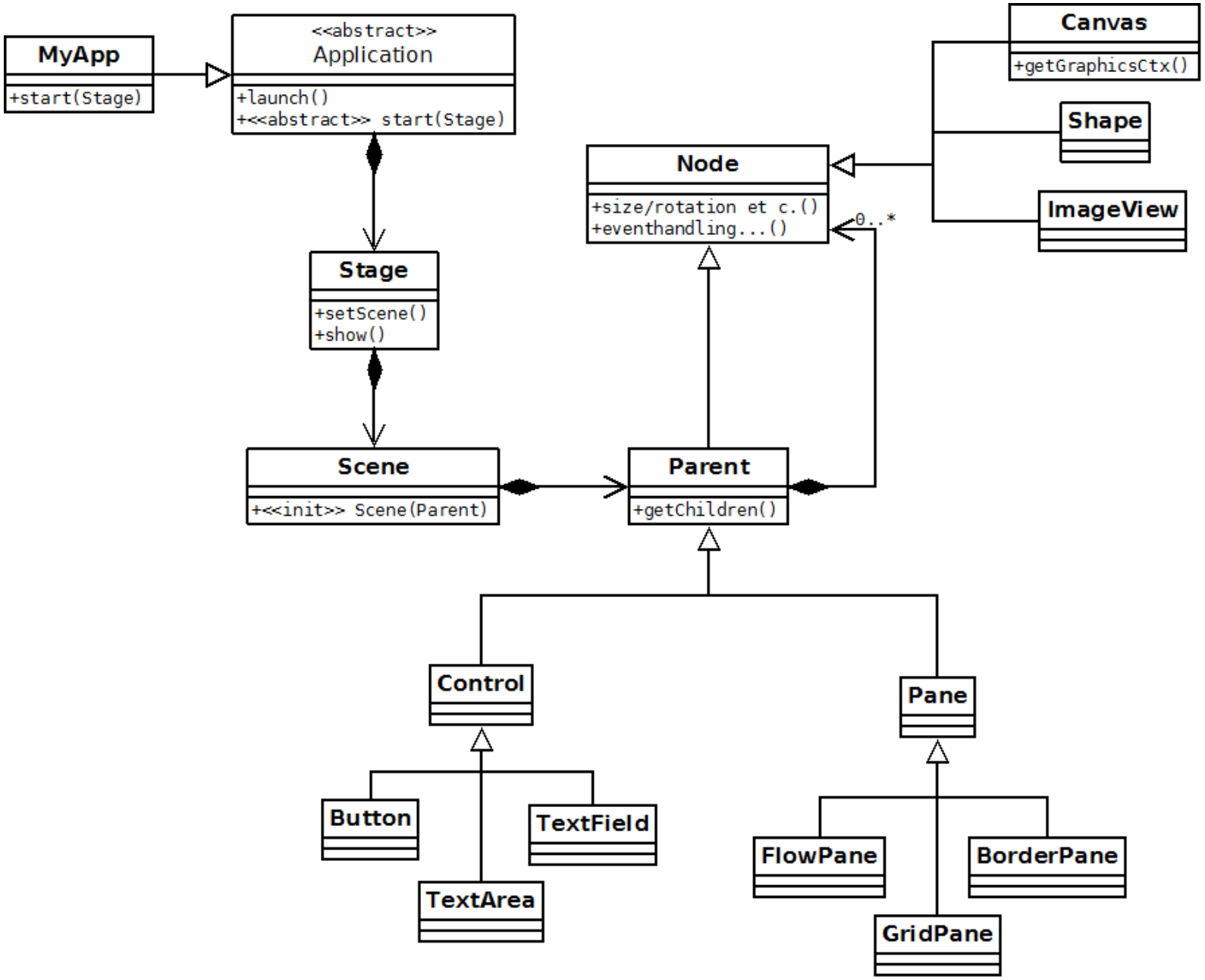
Grafiska användargränssitt och händelsehantering

…i Java

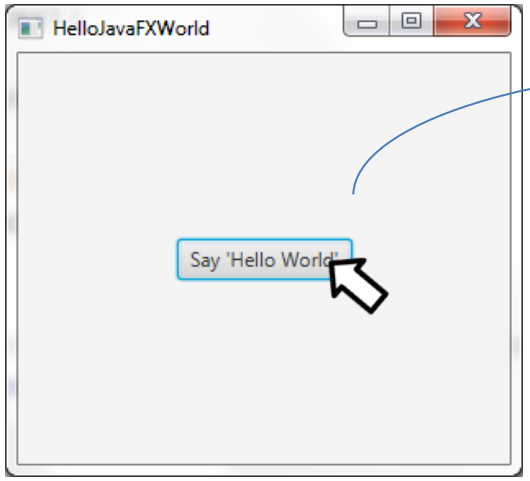
Del 2

Dagens föreläsning

- Repetition på GUI och händelsehantering
- Model-View-Controller
- Subject-Observer
- FXML


MyApp
+start(Stage)
<<abstract>>
Application
+launch()
+<<abstract>> start(Stage)
Canvas
+getGraphicsCtx()
Shape
ImageView
Node
+size/rotation et c.()
+eventhandling...()
0..*
Stage
+setScene()
+show()
Scene
+<<init>> Scene(Parent)
Parent
+getChildren()
Control
Button
TextField
TextArea
Pane
FlowPane
BorderPane
GridPane

Händelsehantering



Event-objekt

EventHandler-object

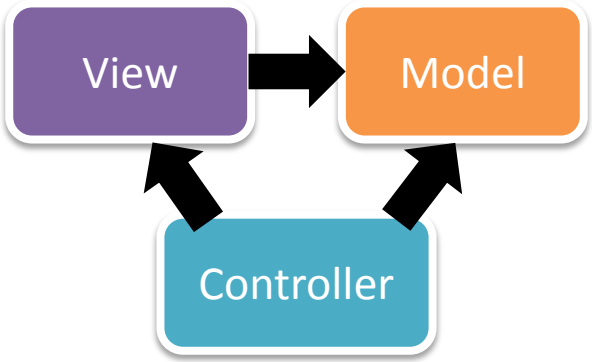
```
class ButtonHandler implements
   EventHandler<ActionEvent> {

   @Override
   public void handle(ActionEvent me) {
      // Take some action…
   }
}
```

- Händelsekälla, ex knapp eller inmatningsfält
- Händelse, ex knapptryck eller ENTER i inmatningsfält – ett Event-objekt skapas
- Händelsehanterare – implementerar `EventHandler.handle(Event)` – definierar vad som ska göras

Model-View-Controller (MVC)

- Applikation med användargränssnitt

Presenterar data från modellen och visar användargränssnittet

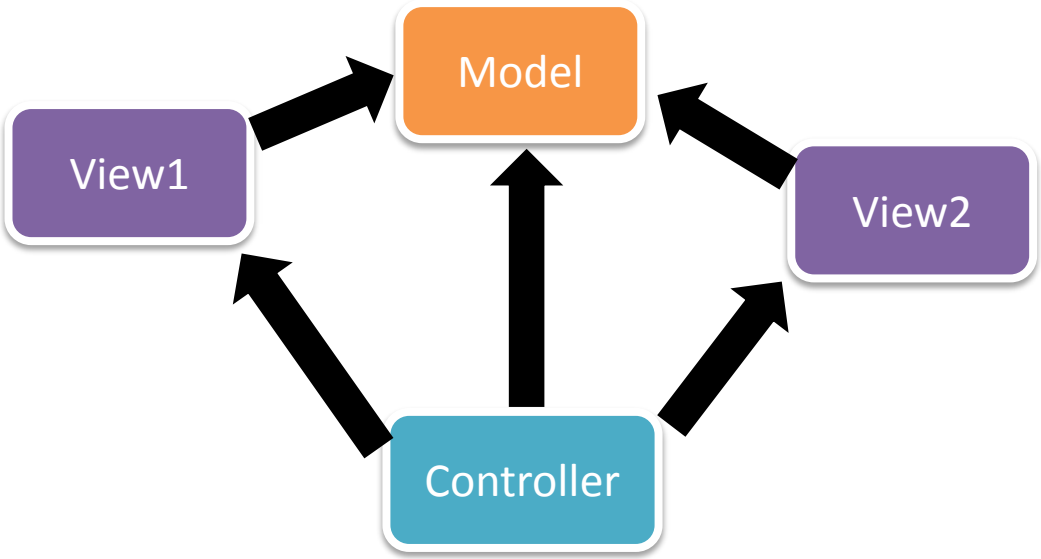


Hanterar beräkningar eller tillhandahålla data som applikationen använders sig av

Hanterar användarinput och initierar uppdateringar av data i modellen samt uppdatering av användargränssnittet

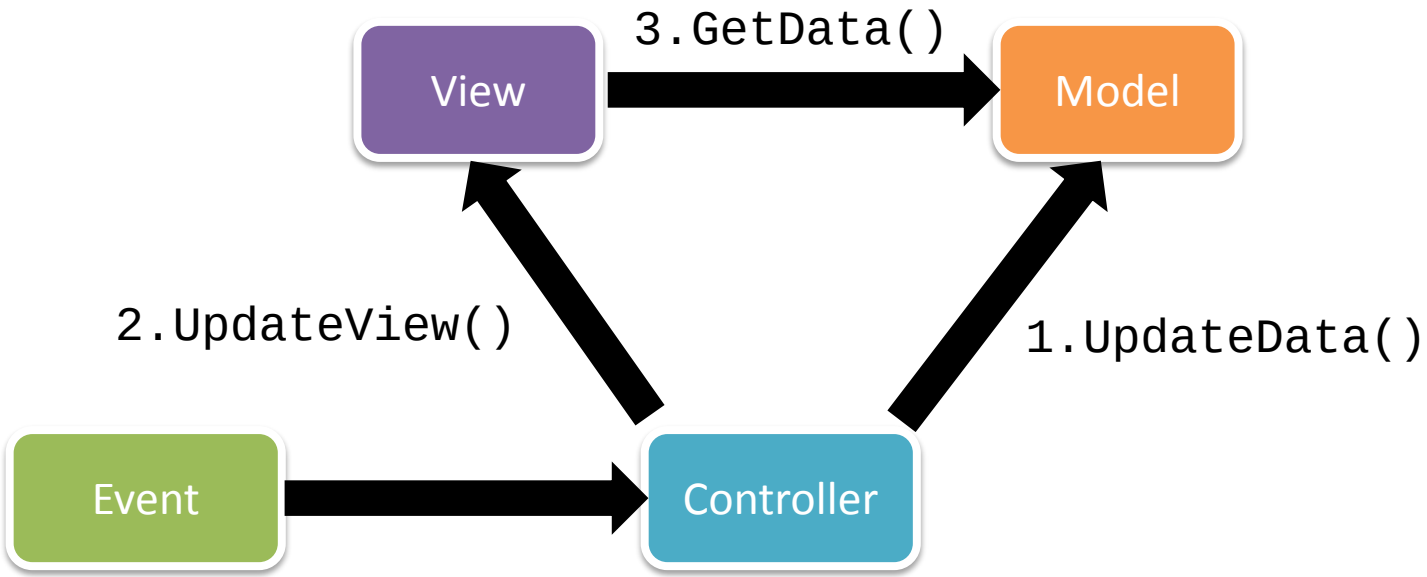
Model-View-Controller

- Fördelar:
  - Uppfyller hög kohesion, låg koppling
  - Gör det möjligt att testa modellen för sig
  - Gör det möjligt återanvända modellen med andra vyer



Model-View-Controller

- I praktiken:

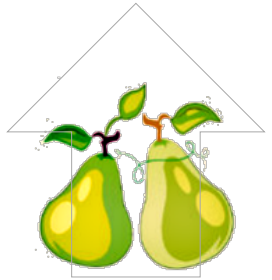


- NB- Både View och Model består normalt av en uppsättning klasser, i var sitt paket

Model-View-Controller

- Black Jack – vad är modellen?

- Flera vyer – Word eller NetBeans
  Vad är modelldatat i NetBeans

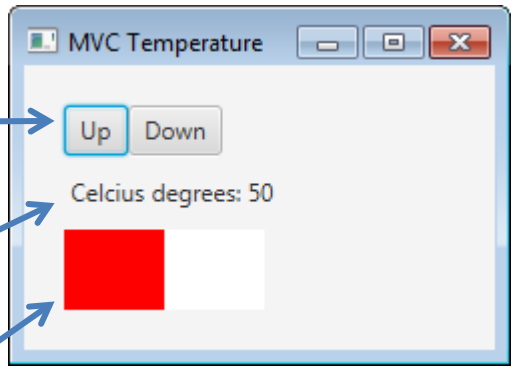


Exempel på MVC

Button

TextLabel extends Label

TempBar extends Pane



Subject-Observer

- Klassen `Observable` representerar ett observerbart objekt (subject), t.ex. variabler i modell i MVC
- Ett observerbart objekt kan ha en eller flera observerare (`Observer`) som vill veta när det har blivit ändringar i modellen
- Ett meddelande skickas till dess observerare när det en ändring har skett

Subject-Observer

- `Observable` (`java.util.Observable`) - superklass
  - `addObserver(Observer o)`
    Lägger till en observer till listan med observers om den inte redan finns i listan.
  - `setChanged()`
    Markerar att det har skett en förändring.
  - `notifyObservers()`
    Om det har skett en förändring, meddela alla observers

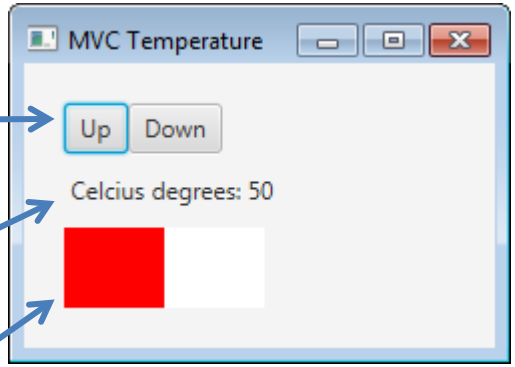
- `Observer` (`java.util.Observer`) - interface
  - `update(Observer o, Object arg)`
    Anropas när det observerade objektet ändras

Exempel på Subjective Observer

Button

TextLabel extends Label

TempBar extends Pane

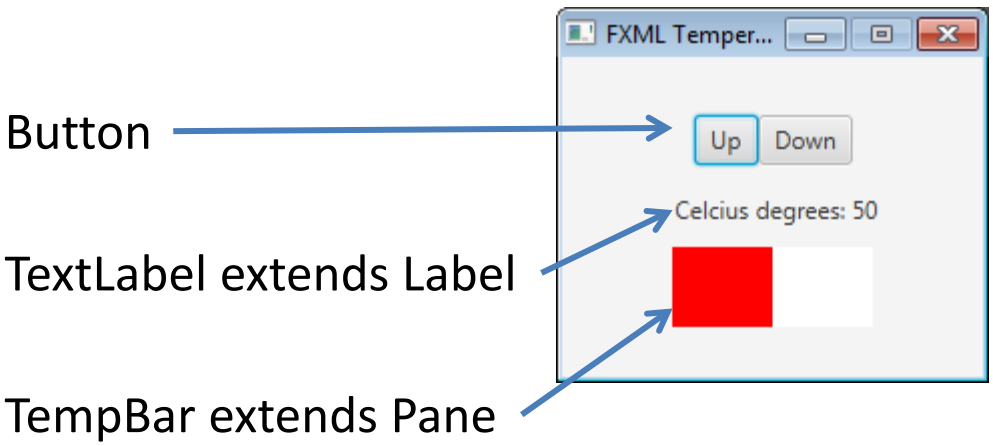


FXML

- Ett annat sätt att definiera användargränssnittet
- Man kan använda SceneBuilder för att bygga upp scenen
- Själva fxml-filen är view-delen som inte behöver kompilera om, om man vill göra ändringar

- http://docs.oracle.com/javase/8/scene-builder-2/work-with-java-ides/sb-with-nb.htm#CHEEHIDG

Exempel på FXML



Frågor?