GUI («Graphical User Interface») del 2

- Interaksjon med brukeren
- Hendelsesdrevet programmering
- Tråder i GUI
- Et spill basert på klikkbare ruter

Se også på

- Infoskrivet https://www.uio.no/studier/emner/matnat/ifi/IN1010/v19/notater/fx-intro-in1010.pdf
- Programkoden i https://www.uio.no/studier/emner/matnat/ifi/IN1010/v19/programmer/GUI/

Det enkleste eksempelet på brukerinteraksjon

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.control.Button;
import javafx.event.*;

public class Stopp extends Application {
    class StoppBehandler implements EventHandler<ActionEvent> {
        @Override
        public void handle(ActionEvent e) {
            Platform.exit();
        }
    }

    @Override
    public void start(Stage teater) {
        StoppBehandler stopp = new StoppBehandler();
        Button stoppKnapp = new Button("Stopp");
        stoppKnapp.setOnAction(stopp);

        Pane kulisser = new Pane();
        kulisser.getChildren().add(stoppKnapp);
        Scene scene = new Scene(kulisser);
        teater.setTitle("Klikk for å stoppe");
        teater.setScene(scene);
        teater.show();
    }
```

Dette er nytt:

- Vi definerer **StoppBehandler** som i metoden handle angir håndteringen av trykk.
- En **Button** er en trykknapp som brukeren kan trykke på. Metoden setOnAction angir hvem som skal ta seg av trykket.
- Metoden Platform.exit er den beste måte å avslutte et GUI-program på.

Programmeringsparadigmer

Det finnes mange **programmeringsparadigmer**, for eksempel

- **Imperativ programmering** der utførelsen følger programflyten angitt av programmereren.
  - **Objektorientert programmering** er en undergruppe der operasjonene er knyttet til objekter.
- **Hendelsesdrevet programmering** der brukerens handlinger styrer programflyten.

Hendelsesdrevet programmering

Her ligger programmet passivt og venter på at noe skal skje. Dette noe kan være

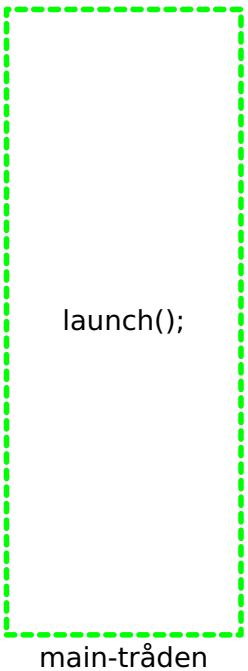
- brukeren trykker på en knapp på skjermen
- brukeren flytter musen
- brukeren trykker på en mustast
- brukeren trykker på en tast på tastaturet
- brukeren slipper opp en tast
- brukeren endrer størrelsen på vinduet
- en alarm utløses

. . . og mye annet.

Oppstart

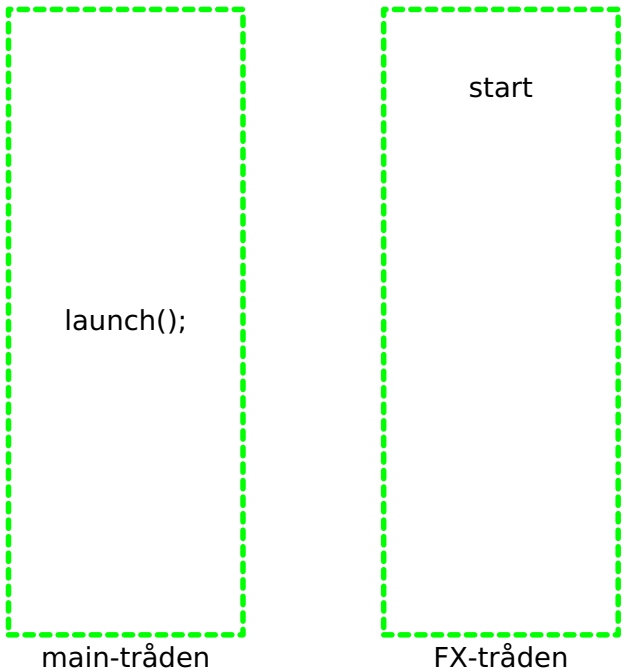
Programmet startes i main-tråden.

Det kaller launch().



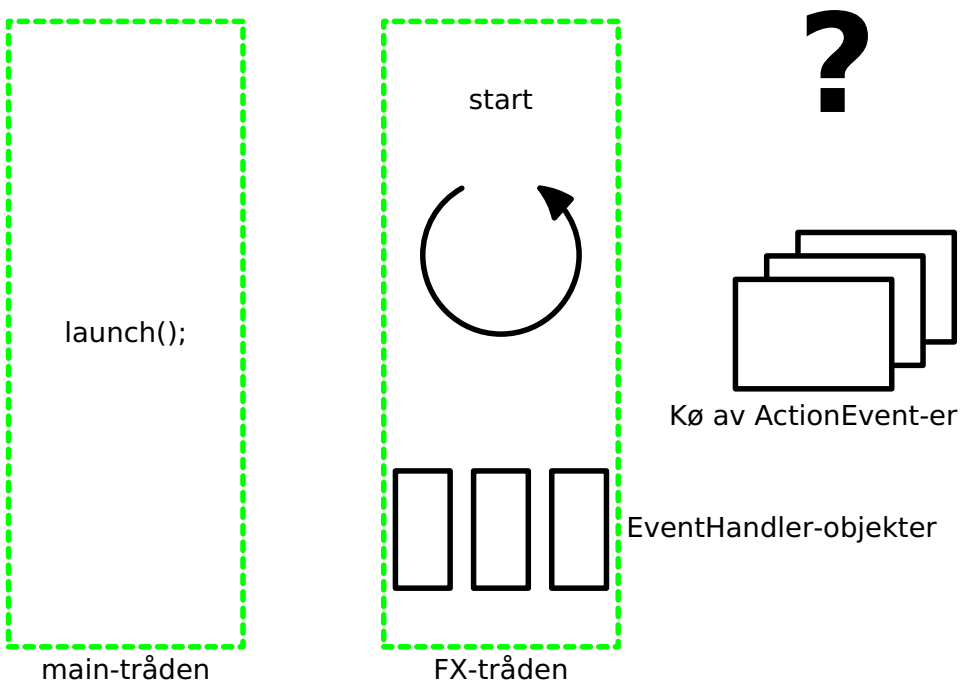
Oppstart av JavaFX

JavaFX kjører metoden start i FX-tråden for å sette opp scenebildet.



Vente på hendelser

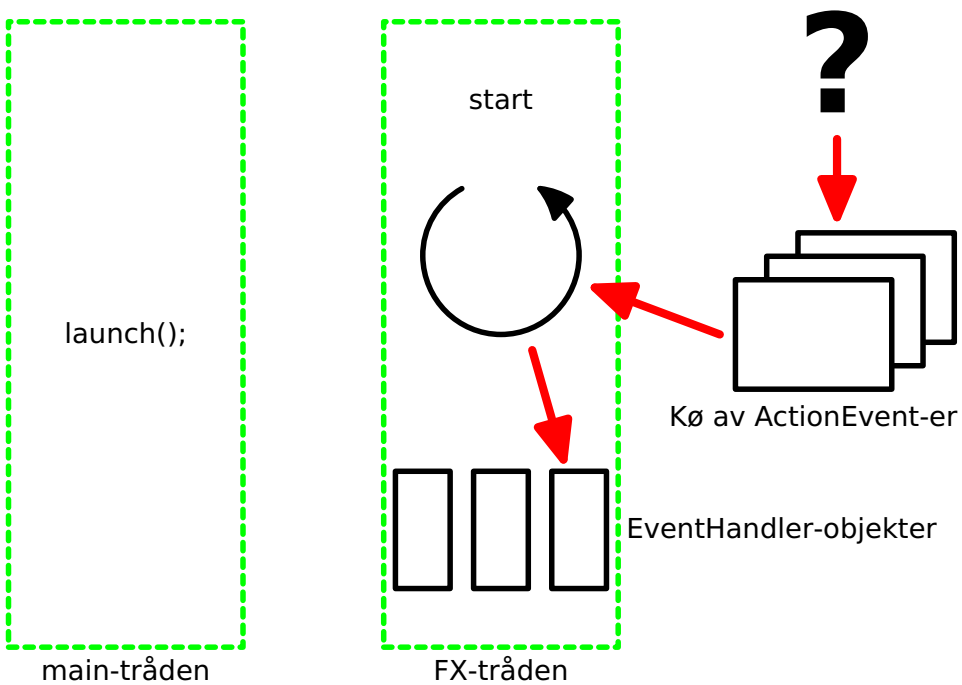
Nå ligger FX-tråden og venter på at noe skal skje.



Håndtere hendelser

Hver hendelse som inntreffer, resulterer i et ActionEvent-objekt i køen.

Hendelsesløkken i FX-tråden vil ta ActionEvent-ene etter tur, og den korrekte EventHandler-en vil bli kalt.



Hvorfor trenger vi en kø av ActionEvent-er?

FX-tråden kan bare ta seg av én hendelse av gangen, men noen ganger kan flere hendelser inntreffe omtrent samtidig.

Da trenger vi køen for å ta vare på de hendelsene som venter på å bli tatt hånd om.

Eksempel: Et telleverk

Dette vinduet inneholder:

- telleren (en Text)
- trykknappen Øk teller (en Button)
- trykknappen Nullstill (en Button)
- trykknappen Stopp (en Button)

Hvordan bygge opp et GUI-program

1. Finn ut nøyaktig hva programmet skal gjøre.
2. Lag en håndtegning av alle elementene.
3. Skriv kode som genererer elementene og plasserer dem på rett plass.
4. Lag EventHandler-klassene.

Del 1: Import og hovedstruktur

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.Pane;
import javafx.scene.text.*;
import javafx.event.*;

public class Telleverk extends Application {
    int teller = 0;
    Text tellerSomText = new Text("0");

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

Del 2: Deklarasjon av EventHandler-ne

```
class NulleBehandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        teller = 0;
        tellerSomText.setText("0");
    }
}

class TelleBehandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        teller++;
        tellerSomText.setText(""+teller);
    }
}

class StoppBehandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        Platform.exit();
    }
}
```

Del 3: Oppsettet av vinduselementene

```
@Override
public void start(Stage teater) {
    tellerSomText.setFont(new Font(25));
    tellerSomText.setX(30);  tellerSomText.setY(25);

    Button telleknapp = new Button("Øk teller");
    telleknapp.setLayoutX(10);  telleknapp.setLayoutY(50);
    TelleBehandler tell = new TelleBehandler();
    telleknapp.setOnAction(tell);

    Button nulleknapp = new Button("Nullstill");
    nulleknapp.setLayoutX(14);  nulleknapp.setLayoutY(80);
    NulleBehandler nulle = new NulleBehandler();
    nulleknapp.setOnAction(nulle);

    Button stoppknapp = new Button("Stopp");
    stoppknapp.setLayoutX(20);  stoppknapp.setLayoutY(110);
    StoppBehandler stopp = new StoppBehandler();
    stoppknapp.setOnAction(stopp);
```

Del 4: Avsluttende initiering

```
        Pane kulisser = new Pane();
        kulisser.setPrefSize(90,150);
        kulisser.getChildren().add(tellerSomText);
        kulisser.getChildren().add(telleknapp);
        kulisser.getChildren().add(nulleknapp);
        kulisser.getChildren().add(stoppknapp);

        Scene scene = new Scene(kulisser);
        teater.setTitle("Teller");
        teater.setScene(scene);
        teater.show();
    }
}
```

Advarsel: Ting tar tid

I JavaFX finnes det i utgangspunktet kun én tråd som både skal oppdatere vinduet og håndtere alle EventHandler-ne. Det innebærer at

- mens en EventHandler jobber, er resten av GUI-systemet dødt
- hvis en EventHandler får en for stor oppgave (for eksempel beregne et sjakktrekk), må det opprettes en egen tråd til det.

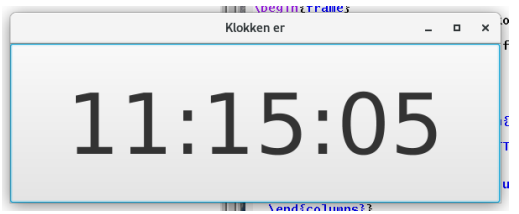
Et eksempel på dårlig koding

```
static void ventAktivt(int n) {
    // Kjør kode som tar omtrent n sekunder.
    for (int i = 1;  i <= n;  i++) {
        long x = 1;
        for (long j = 1;  j <= 250*1000*1000;  j++) {
            x = (3+j)*x % (1000*1000);
        }
    }
}

class NulleBehandler implements EventHandler<ActionEvent> {
    @Override public void handle(ActionEvent e) {
        teller = 0;  tellerSomText.setText("0");
        ventAktivt(3);
    }
}

class TelleBehandler implements EventHandler<ActionEvent> {
    @Override public void handle(ActionEvent e) {
        teller++;  tellerSomText.setText(""+teller);
        ventAktivt(3);
    }
}

class StoppBehandler implements EventHandler<ActionEvent> {
    @Override public void handle(ActionEvent e) {
        Platform.exit();
    }
}
```

En klokke

Vi ønsker å lage en klokke:

- Klokken skal vise timer, minutter og sekunder.
- Den skal oppdateres hvert sekund.
- Den kan stoppes ved å klikke på den.

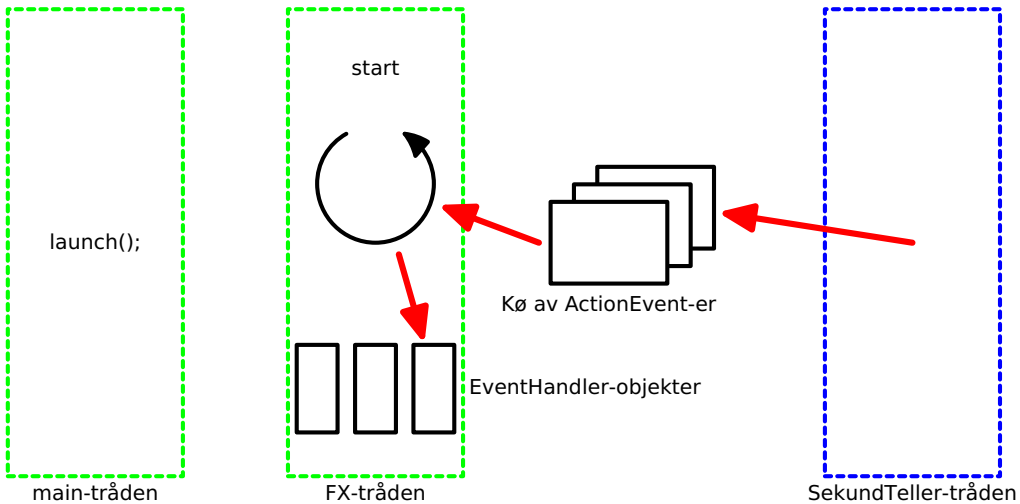


Hva bør vi tenke på?

- Vi får klokkeslettet fra datamaskinen.
- Oppdateringen besørges av en egen tråd som går i løkke: den sover ett sekund og oppdaterer så klokkeslettet.
- Selve klokken må være en button slik at vi kan klikke på den.

Siden oppdateringen gjøres utenfor FX-tråden, kan vi ikke kalle setText direkte. Vi må i stedet legge en hendelse i køen; dette gjøres med et kall på

Platform.runLater(Runnable r)



Del 1: Import og hovedstruktur

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.Pane;
import javafx.scene.text.Font;
import javafx.event.*;

import java.time.LocalTime;

public class Klokke extends Application {
    Button klokken = new Button(naa());
    boolean slutt = false;

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

    private static String naa() {
        // Hva er klokken nå? Svaret er på formen "12:34:56".
        LocalTime t = LocalTime.now();
        return String.format("%02d:%02d:%02d",
            t.getHour(), t.getMinute(), t.getSecond());
    }
```

Del 2: EventHandler-e

```
class StoppBehandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        slutt = true;  Platform.exit();
    }
}
```

Del 3: Sekundtellertråden

```
class SekundTeller extends Thread {
    // En egen tråd som oppdaterer klokkeslettet hvert sekund.

    @Override
    public void run() {
        while (! slutt) {
            try {
                sleep(1000);
            } catch (InterruptedException e) { Platform.exit(); }

            Platform.runLater(new VisTid());
        }
    }
}

class VisTid implements Runnable {
    @Override
    public void run() {
        klokken.setText(naa());
    }
}
```

Del 4: Resten av initieringen

```
    @Override
    public void start(Stage teater) {
        klokken.setFont(new Font(100));
        klokken.setOnAction(new StoppBehandler());

        Pane kulisser = new Pane();
        kulisser.getChildren().add(klokken);

        Scene scene = new Scene(kulisser);
        teater.setTitle("Klokken er");
        teater.setScene(scene);
        teater.show();

        new SekundTeller().start();
    }
}
```

Tripp-trapp-tresko med GUI

I vår løsning skal vi ha dette:

- Et vindu med
  - et spillebrett på 3×3 ruter
  - et tekst med info til brukeren
  - en stoppknapp
- Hver rute på 3×3-brettet skal være klikkbar.
- Ved hvert klikk skal programmet
  1. sjekke om trekket er lovlig
  2. vise brukerens trekk
  3. sjekke om brukeren har vunnet
  4. beregne og vise maskinens trekk
  5. sjekke om maskinen har vunnet

Definisjon av de klikkbare rutene

Vi lager en egen subklasse Rute fordi vi ønsker å lagre et merke i hver rute. De lovlige verdiene skal være:

- `' '` når ruten er ledig
- `'X'` når maskinen har valgt ruten
- `'O'` når spilleren har valgt ruten.

```
class Rute extends Button {
    char merke = ' ';

    Rute() {
        super(" ");
        setFont(new Font(50));
        setPrefSize(120, 120);
    }

    void settMerke(char c) {
        setText(""+c);  merke = c;
    }
}
```

Sett opp rutenettet

```
brett = new Rute[9+1];
Klikkbehandler klikk = new Klikkbehandler();
for (int i = 1;  i <= 9;  i++) {
    brett[i] = new Rute();
    brett[i].setOnAction(klikk);
}

GridPane rutenett = new GridPane();
rutenett.setGridLinesVisible(true);
rutenett.add(brett[1], 0, 0);
rutenett.add(brett[2], 1, 0);
rutenett.add(brett[3], 2, 0);
rutenett.add(brett[4], 0, 1);
rutenett.add(brett[5], 1, 1);
rutenett.add(brett[6], 2, 1);
rutenett.add(brett[7], 0, 2);
rutenett.add(brett[8], 1, 2);
rutenett.add(brett[9], 2, 2);
rutenett.setLayoutX(10);  rutenett.setLayoutY(10);
```

Håndtering av klikk i en rute

Metoden getSource gir oss hvilken rute det ble klikket i.

```
class Klikkbehandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        if (! ferdig)
            spillO((Rute)e.getSource());
    }
}
```

Håndter brukerens valg

```
void spillO(Rute r) {
    if (r.merke != ' ') {
        statusinfo.setText("Ruten er opptatt; velg en annen");
        return;
    } else {
        statusinfo.setText("Velg en rute");
    }

    r.settMerke('O');
    if (harVunnet('O')) utropVinner('O');

    if (! ferdig) spillX();
}
```

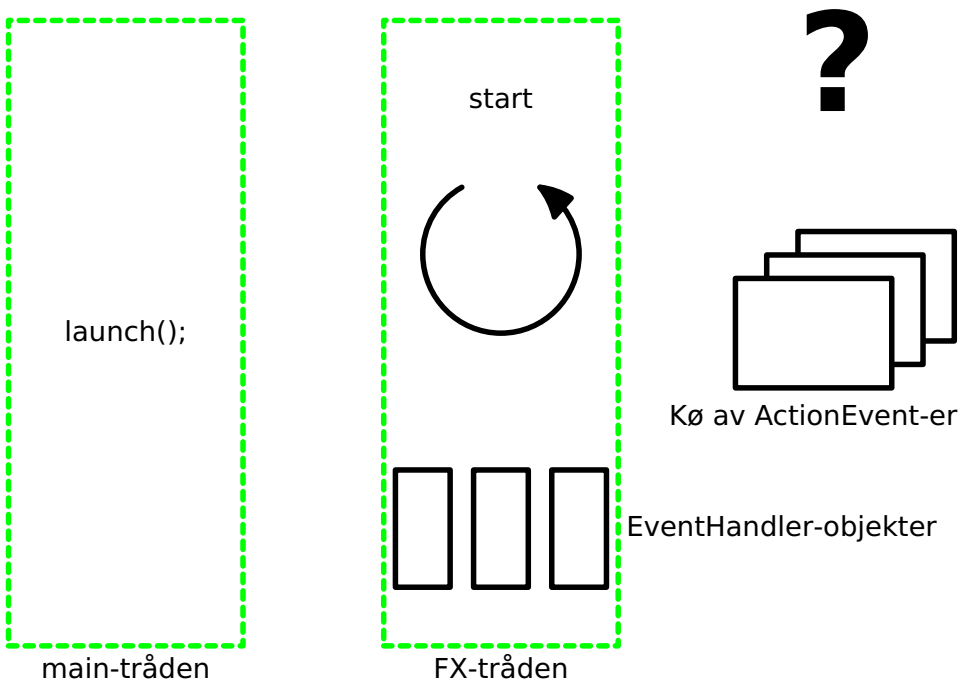
La datamaskinen velge sitt neste trekk

Datamaskinen gjør sitt neste trekk helt tilfeldig.

```
Random tilfeldig = new Random();
void spillX() {
    int p;
    do {
        p = tilfeldig.nextInt(9)+1;
    } while (brett[p].merke != ' ');
    brett[p].settMerke('X');

    if (harVunnet('X')) utropVinner('X');
    else if (erUavgjort()) utropUavgjort();
}
```

De to trådene

- De to trådene kan jobbe i fred for hverandre.
- De kan kommunisere gjennom felles variable.
- Men: main-tråden bør ikke kalle GUI-metoder; det går ikke alltid bra!
- I stedet kan main-tråden legge inn nye Event-er i køen ved å kalle Platform.runLater(r) der r implementerer Runnable (se eksemplet Klokke.java).



Noen gode råd

Før du prøver å lage ditt første GUI-program:

- Hent et program fra IN1010-sidene.
- «Lek» litt med programmet. Brøv å endre noe eller legge noe til, og se hva som skjer.