Obserververbar

INF101 forelesning 26. april 2022

Torstein Strømme

Stikkord: funksjonelle grensesnitt

Fremover

- I dag: siste ordinære forelesning.
  Nytt forsøk på å forstå observerbare variabler.

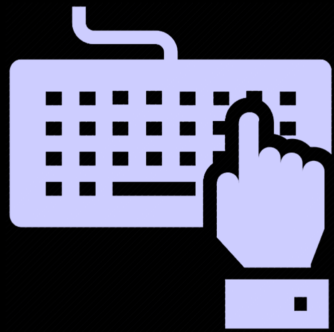
- Fredag: deadline for semesteroppgave 2
- Neste uke: frivillig prøveeksamen onsdag 12-17 i Egget (Studentsenteret)

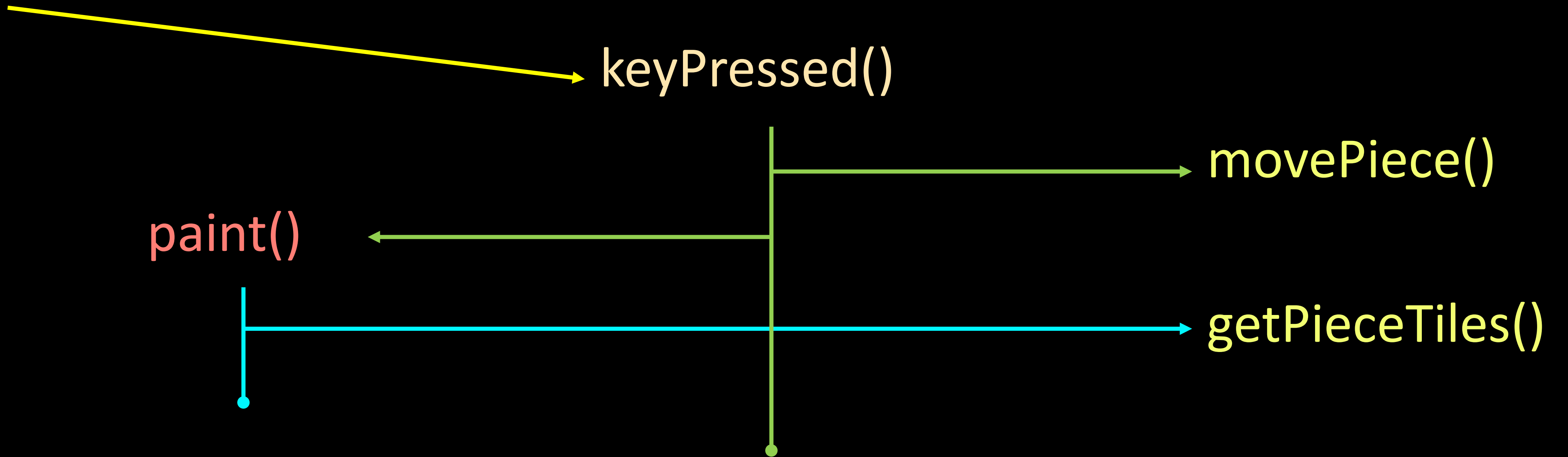
- Fokusgrupper om semesteroppgave 2 (gratis kinobilletter)

Tetris: model-view-controller

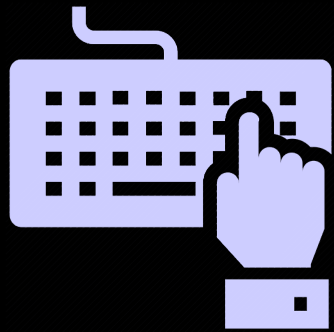
- Modellen
  - Lagrer informasjon
  - Regler for modifisering av informasjon
- Kontroller
  - Bestemmer hvordan input skal påvirke modellen
- Visning
  - Viser modellen
  - Henter visse typer input (tastetrykk, musebevegelser, klikk)

Tetris: model-view-controller





Tetris: model-view-controller



TetrisView | *TetrisController* | *TetrisModel*

addKeyListener() ← view.addKeyListener(this)

keyPressed()

movePiece()

paint()

getPieceTiles()

Er det egentlig bra at *kontrollen* kaller paint/repaint?

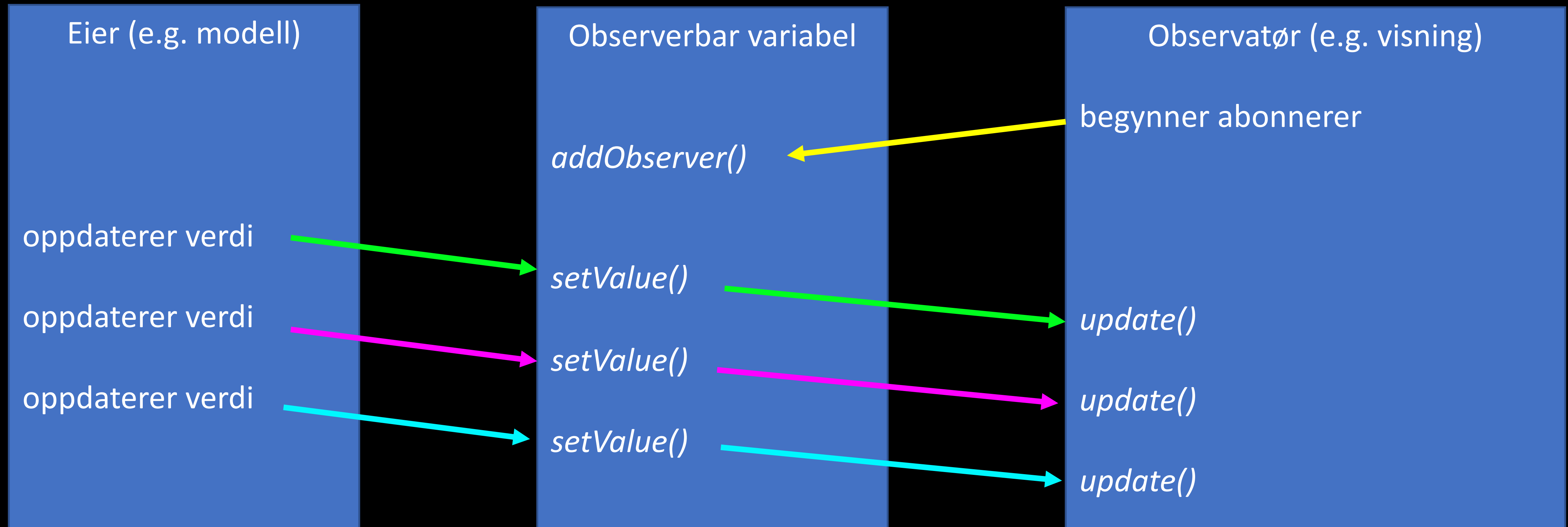
Bør *modellen* kalle repaint?

- Modellen *bør* kalle repaint fordi:
  - Modellen vet best om den har endret seg
  - Kontrollen bør slippe å tenke på visningen
  - Uansett hvilken mekansime som endret modellen, blir view oppdatert

- Modellen bør *ikke* kalle repaint
  - Modellen bør slippe å tenke på visningen
  - Modellen vet ikke hvilke deler av modellen som faktisk vises

Løsning: observerbare variabler

- En observerbar variabel er en variabel man kan "abonnere på endringer ved"



Observerbare variabler

```
interface Observable <E> {

    void addObserver(Observer observer);
    void setValue(E newValue);
    E getValue();

}
```

```
interface Observer {

    void update();

}
```

Observerbare variabler: testing og live-koding

```
Observable<String> observable = new Observable<>("Hello");
assertEquals("Hello", observable.getValue());

TestObserver observer = new TestObserver();
observable.addObserver(observer);
assertFalse(observer.flag);
observable.setValue("Goodbye!");
assertTrue(observer.flag);

assertEquals("Goodbye!", observable.getValue());
```

```
class TestObserver implements Observer {
    boolean flag = false;
    public void update() { this.flag = true; }
}
```

Observerbare variabler: testing og live-koding

```java
Observable<String> observable = new Observable<>("Hello");
assertEquals("Hello", observable.getValue());

TestObserver observer = new TestObserver();
observable.addObserver(observer);
assertFalse(observer.flag);
observable.setValue("Goodbye!");
assertTrue(observer.flag);

assertEquals("Goodbye!", observable.getValue());
```

```java
class TestObserver implements Observer {
    boolean flag = false;
    public void update() { this.flag = true; }
}
```

Funksjonelle grensesnitt

- Et grensesnitt er *funksjonelt* dersom det har *én* metode
- Eksempler:
  - Observer `void update();`
  - Runnable `void run();`
  - ActionListener `void actionPerformed(ActionEvent e);`
  - Function<T, R> `R apply(T t);`

- Funksjonelle grensesnitt kan bruke :: syntax for å opprette et objekt fra en metode

Funksjonelle grensesnitt

- Funksjonelle grensesnitt kan bruke :: syntax for å opprette et objekt fra en metode

- Eksempel:

```
void setFlagToTrue() {
    this.flag = true;
}

observable.addObserver(this::setFlagToTrue);
```

Funksjonelle grensesnitt

- Funksjonelle grensesnitt kan bruke () - > {} syntax for å opprette et objekt fra en metode

- Eksempel:

```
observable.addObserver(() -> { this.flag = true; });
```

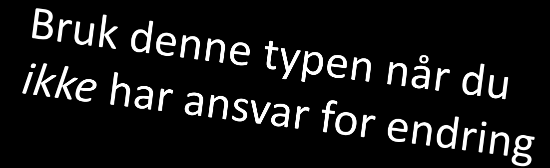
Funksjonelle grensesnitt

- Funksjonelle grensesnitt kan bruke () - > {} syntax for å opprette et objekt fra en metode

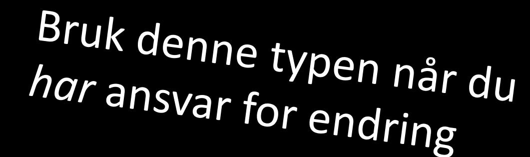
- Eksempel:

```
timer.addActionListener((e) -> { this.model.tick(); });
```

Observerbare variabler *med innkapsling*

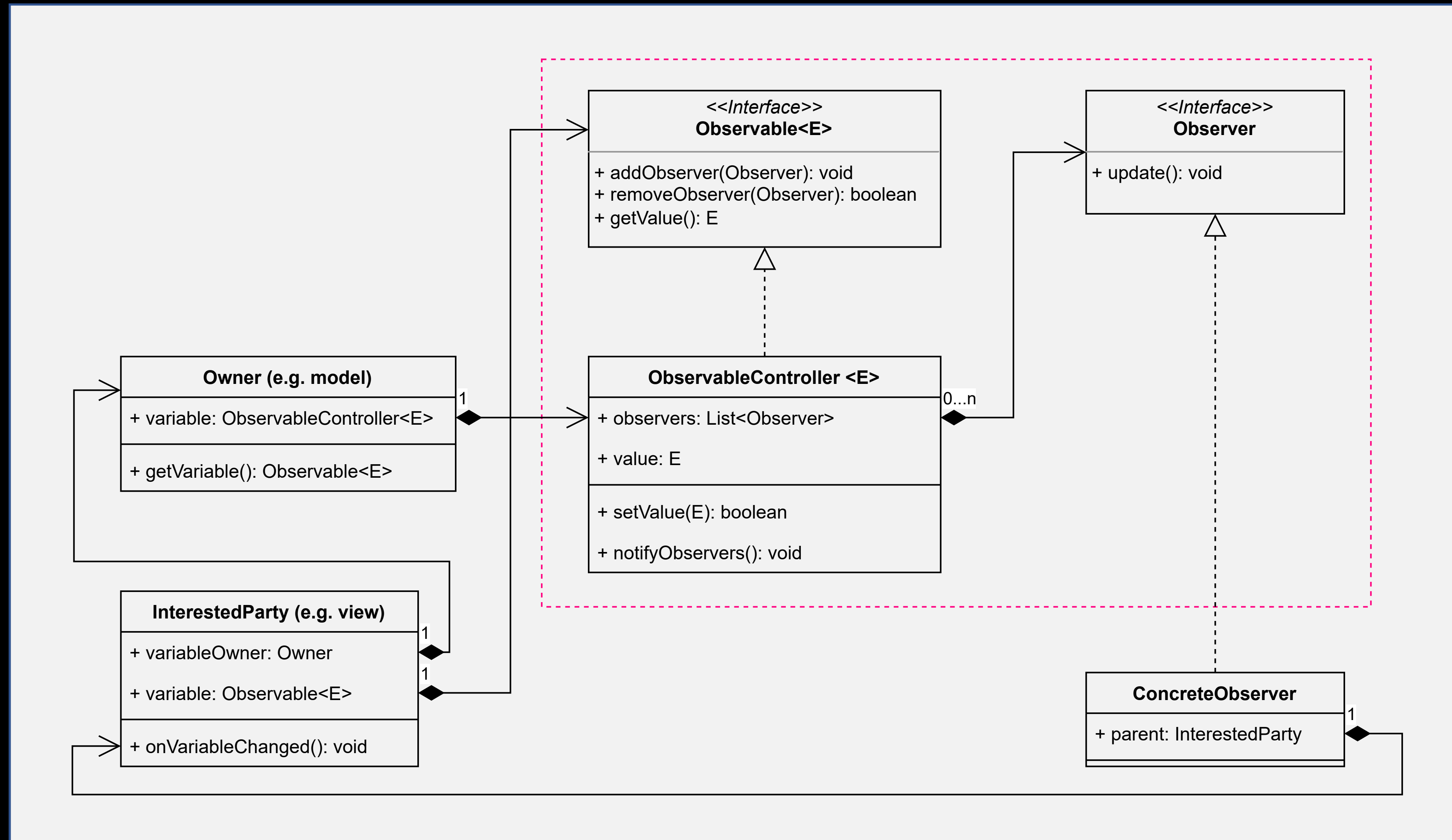



```
interface Observable <E> {

    void addObserver(Observer observer);
    E getValue();

}
```




```
interface ControlObservable <E> extends Observable <E> {
    void setValue(E newValue);
}
```

Observerbar UML



Observerbare variabler: oppsummering

- En wrapper/boks rundt en variabel
- Man kan abonnere på endringer/få utført et metodekall etter eget ønske hver gang variabelen endrer seg
- I model-view-controller kan visningen ha rolle som observatør/abonnent, mens modellen har rolle som eier.
  - Visningen kan lytte til endringer i variabelen den tegner, og blir da ansvarlig for kall til repaint på egen hånd: uten hjelp av verken modell eller kontroll.
- Metoden som skal kalles er ofte ukjent for den som "eier" variabelen
- Idéen er mye brukt, og har mange navn: ActionListener, EventListener, Subject-Observer, Attachable, Bindable, Subscribable, PropertyChangeListener, etc.
- Koden blir mindre omstendelig med :: -syntax og () -> {} –syntax
- Kan innkapsles som uforanderlig ved bruk av restriktive grensesnitt

Live-koding

- micro:bit
  - Har to knapper vi bruker som "sensorer"
  - sender en melding hver gang knappene trykkes ned eller går opp

- vår oppgave: vise tilstanden til de to sensorene i en Java GUI