

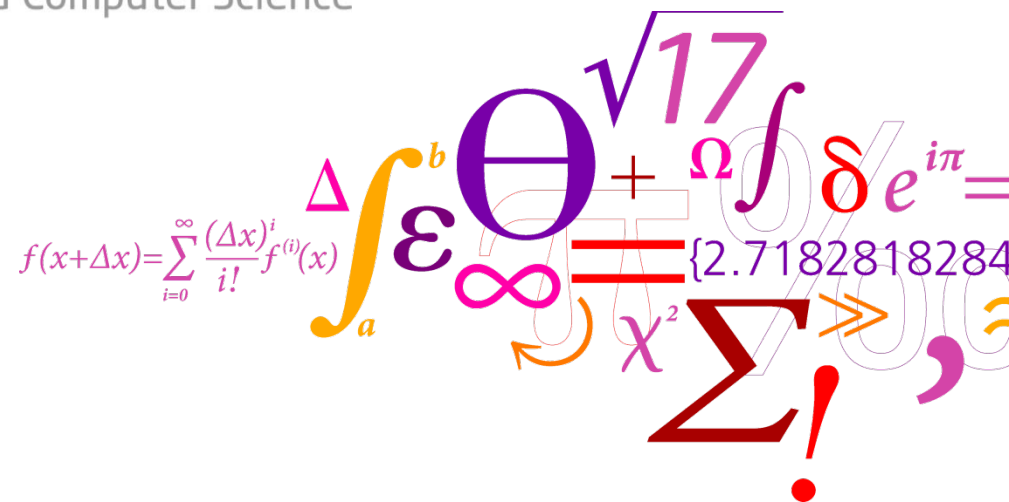
Projekt i software-udvikling (02362)

Forår 2022

Ekkart Kindler

DTU Compute

Department of Applied Mathematics and Computer Science



III. Analyse og konceptuelle modeller

Kort rekapitulation fra
sidste gang og live-
diskussion (igen)

DTU Compute

Department of Applied Mathematics and Computer Science

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$

$$\Delta \int_a^b \varepsilon \Theta + \Omega \int \delta e^{i\pi} = \{2.7182818284\}$$

$$\sqrt{17}$$

$$\infty$$

$$\chi^2$$

$$\Sigma$$

$$!$$

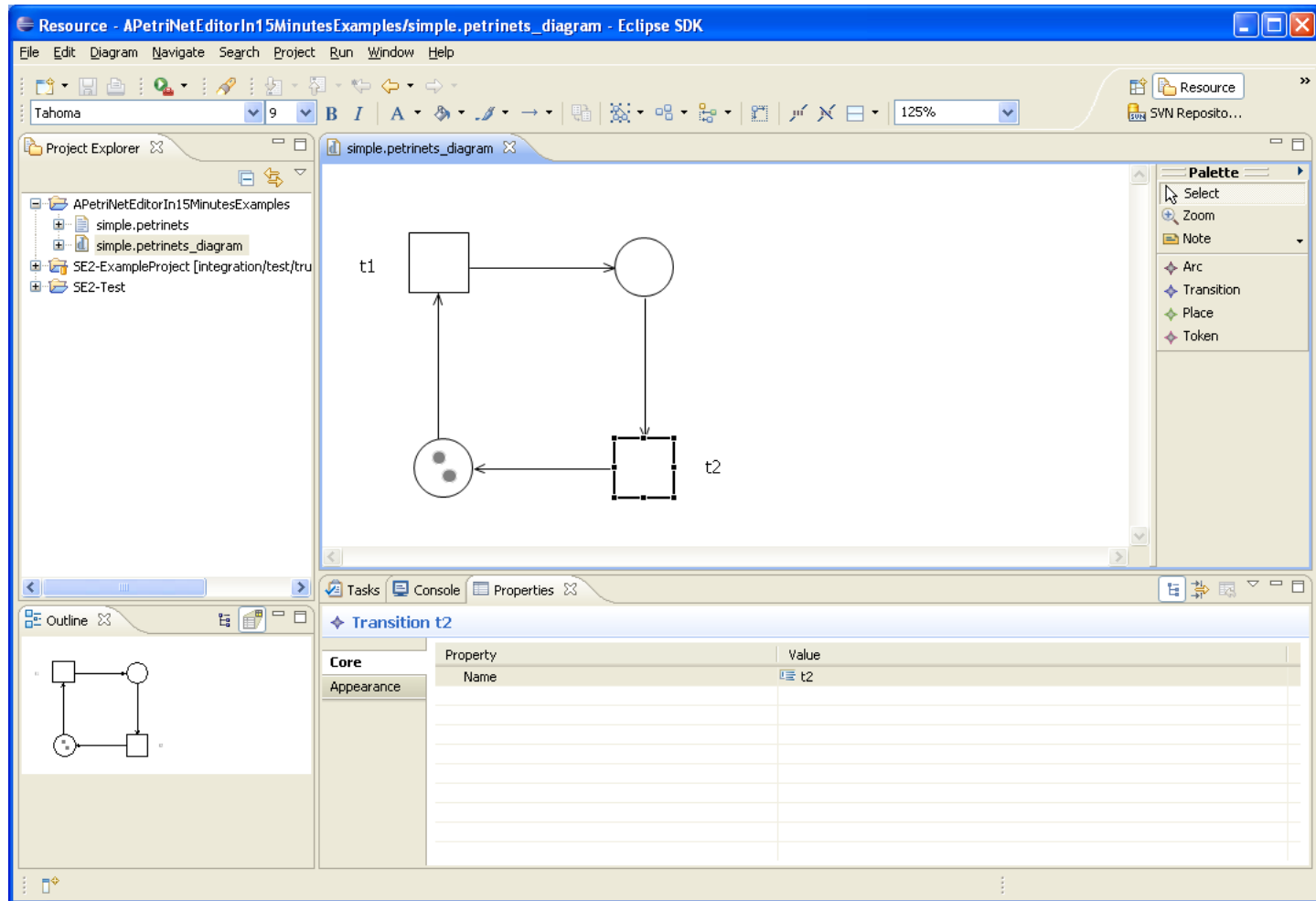
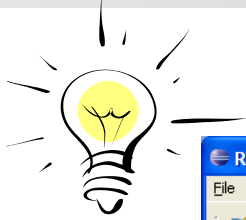
Analysen drejer sig **kun** om
HVAD (engl. **WHAT**) softwaren skal gøre
(ud fra brugerens synsvinkel)

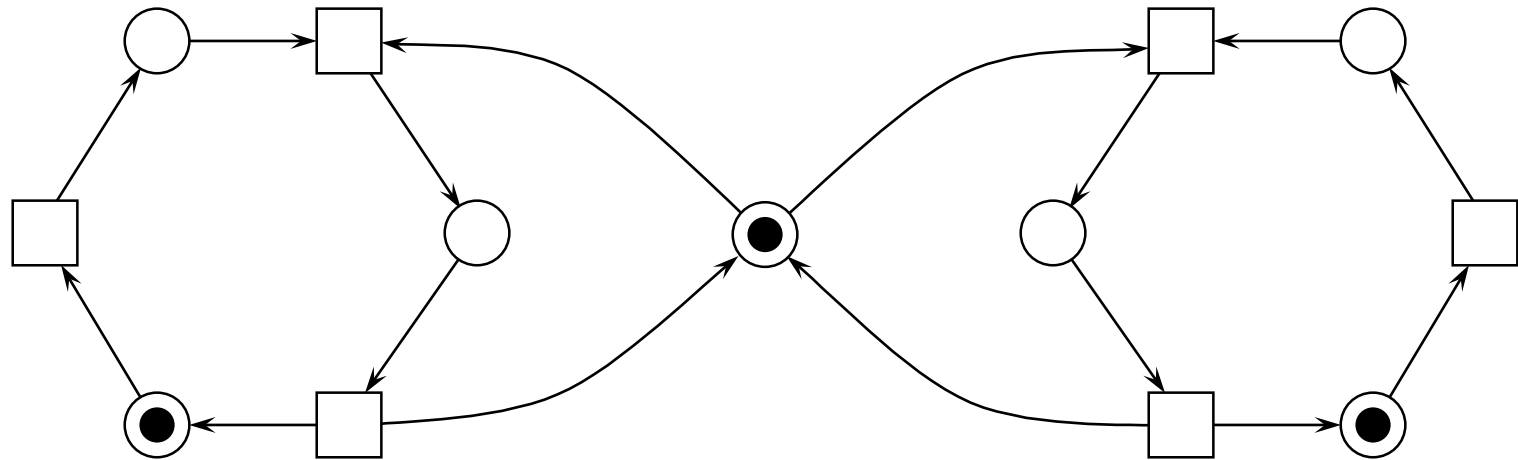
Analysen siger **ikke noget** om
HVORDAN (engl. **HOW**) softwaren skal
realiseres og implementeres

- Find skriftlig dokumentation om området (domænet) og skriv det ud
- Marker alle begreber som synes relevante
- Tal med nogle eksperter fra domænet
- **Taksonomi**: liste af begreber (ord)
- **Glossar**: Beskrivelse af begreber deres egenskaber og hvordan de forholder sig til hinanden
- **Domænemodel**: Begreber, deres attributter og deres relation repræsenteret som klassediagram og beskrivelse af aktiviteter og begrebernes tilstande (aktivitets- og tilstandsdiagrammer)

Disse diagram har ikke noget med programmering at gøre (endnu). Tænk kun på begreber og deres relation!

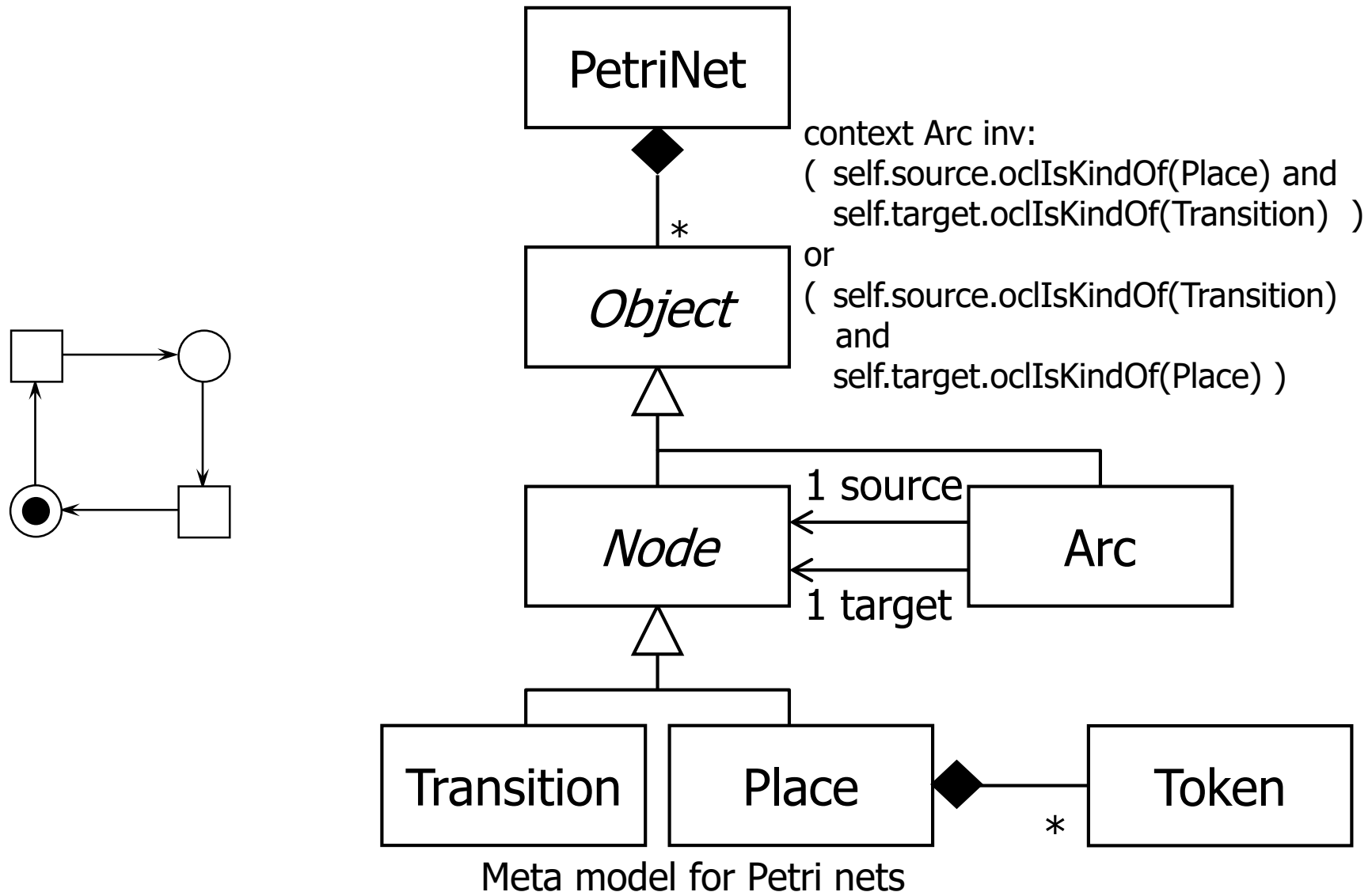
2. Eksempel: Petrinet





- Eksempler
- Taksonomi (på tavlen)
- Glossar
- Model (på tavlen)

Regel: I skal først starte med at lave et UML diagram, hvis I har kigget på nogle eksempler først og har en liste med hovedbegreberne (taksonomi)!



Vi gennemgår processen med hjælp af udtræk af vores RoboRally-projekt

Diskussion i grupper og under live-delen igen.

1. Find beskrivelsen af RoboRally frem
(se projektslides og især links til RoboRallys regler)
2. Taksonomi
3. Glossar
4. Modeller (klassediagrammer, tilstandsdiagrammer, aktivitetsdiagrammer)

Navneord!

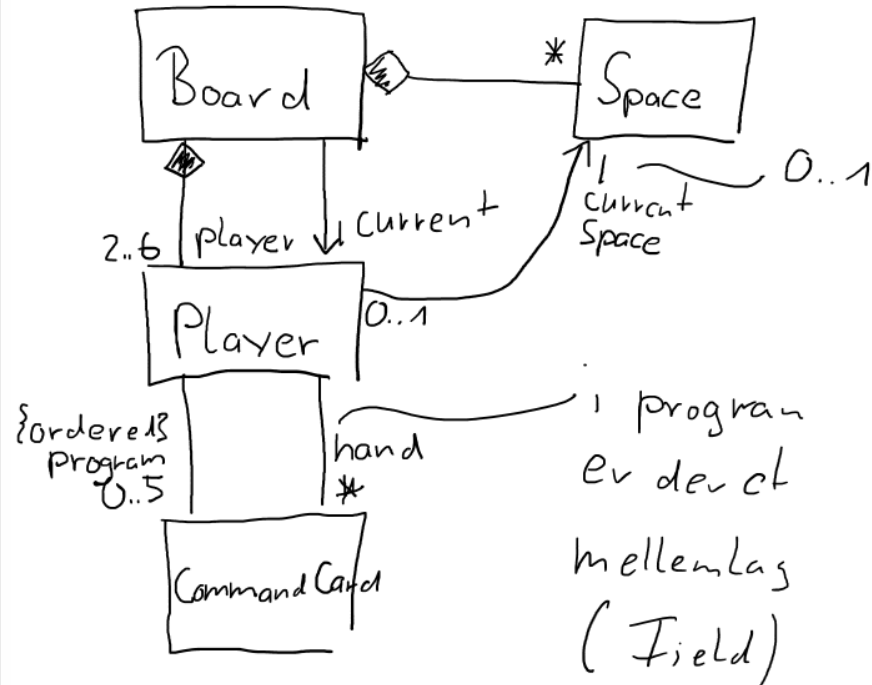
RoboRally: Taksonomi

Hovedord (nouns)

- Robot (player) ✓
- Kommandokort ✓
- Felter (Space) ✓
- Bræt (Board) / Game ✓
- Antenna
- Energy cubes
- Hand card ✓
- Program ✓

Udsagnsord (verbs)

- Start game
- Ryk frem, ... (kommandoer)
- Power-up (upgrade)
- Aktivat programs (robots)
- Program robots



Vi starter med livscyklus af udvalgte objekter, da de er typisk nemmere og tættere på klassediagrammet:

- spil
- spiller (== robot ?)
- felt
- kort
- ...

Diskussion i grupper og på tavlen; vi bruger tilstandsdiagrammer (UML state machines) til det

Faktisk kunne nogle af de der tilstande være et attribut af klasser i klassediagrammet (eller de kan afledes af nogle af objektets andre attributter.

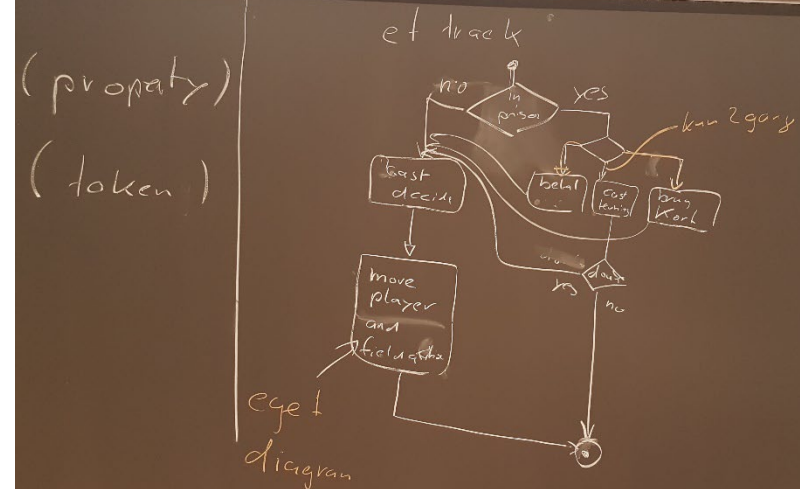
Vi diskuterer udtræk af nogle udvalgte aktiviteter:

- lav et trækk???
- ...

Diskussion i grupper og på tavlen; vi bruger aktivitetsdiagrammer (UML activity diagrams) til det

Bemærk at nogle aktiviteter bliver udført som del af nogle andre aktiviteter; og aktiviteter kan køre også sideløbende med andre aktiviteter

Fra sidste år:
Monopoly/Matador

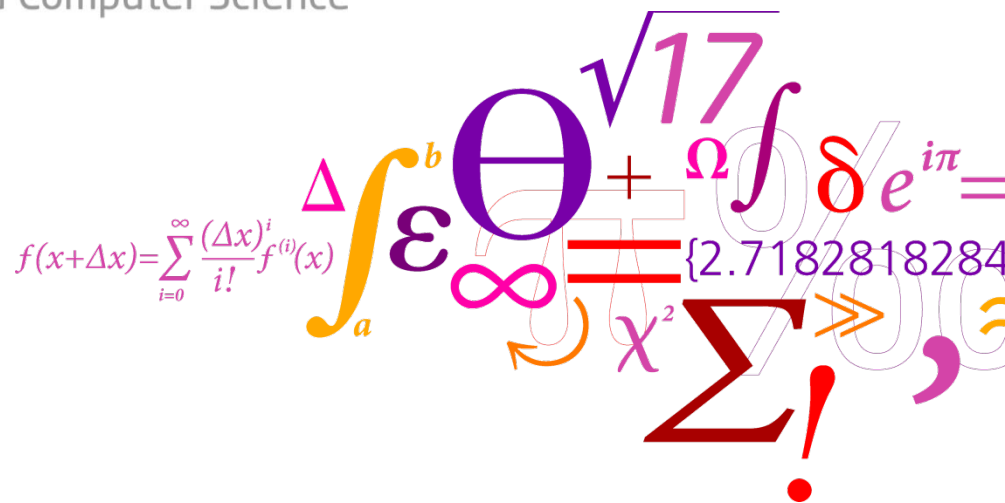


II. (Software) Design

Med nogle flere
design patterns

DTU Compute

Department of Applied Mathematics and Computer Science



A collage of various mathematical symbols and formulas, including:

- $f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$
- $\int_a^b \epsilon \Theta$
- $\sqrt{17}$
- $\Omega \int \delta e^{i\pi} =$
- $\{2.7182818284\}$
- χ^2
- Σ
- ∞
- Δ
- ϵ
- Θ
- Ω
- δ
- $e^{i\pi}$
- χ^2
- Σ
- ∞
- Δ
- ϵ
- Θ
- Ω
- δ
- $e^{i\pi}$
- χ^2
- Σ
- ∞

Analysen drejer sig **kun** om
HVAD (engl. **WHAT**) softwaren skal gøre
(ud fra brugerens synsvinkel)

Analysen siger **ikke noget** om
HVORDAN (engl. **HOW**) softwaren skal
realiseres og implementeres

- **Design** (og implementering) taler om **HVORDAN** (**HOW**) software skal realiseres
- Design taler om den overordnede struktur af softwaren (dens **arkitektur**): hovedkomponenterne, deres interfaces og hvordan de spiller sammen
- Det kaldes på engelsk også for "programming in the large"

Her kan vi kun tale om nogle enkelte aspekter og principper af design og arkitektur!

2. Design patterns

Oprindeligt, kom begrebet
fra: Alexander et al. 1977.

Design patterns (i softwareudvikling) er den destillerede erfaring af softwareudviklings-eksperter hvordan man løser standardproblemer i software-design.

Freeman & Freeman kalder
det “experience reuse”
(genbrug af erfaring)!

De kaldes ofte for “Gang of Four” (GoF / Go4).

- Gamma, Helm, Johnson, Vlissides:
Design Patterns. Addison-Wesley 1995.
- Eric Freeman, Elisabeth Freeman:
Head First Design Patterns. O'Reilly
2004 [FF]
- ...

- Design patterns er et emne i sig selv og kunne være indhold af et separate kursus eller seminar.
- Her gives der bare en kort overblik over ideen bagved og nogle meget vigtige design patterns, som er vigtige at forstå good design.

Name and classification

Observer, object, behavioural

Vi følger
nogenlunde GoF

Intent

”Define a one-to-many dependency between objects so that when an object changes all its dependents are notified and updated automatically” [GoF].

Also known as

Dependents, Publish-Subscribe, Listener

Motivation

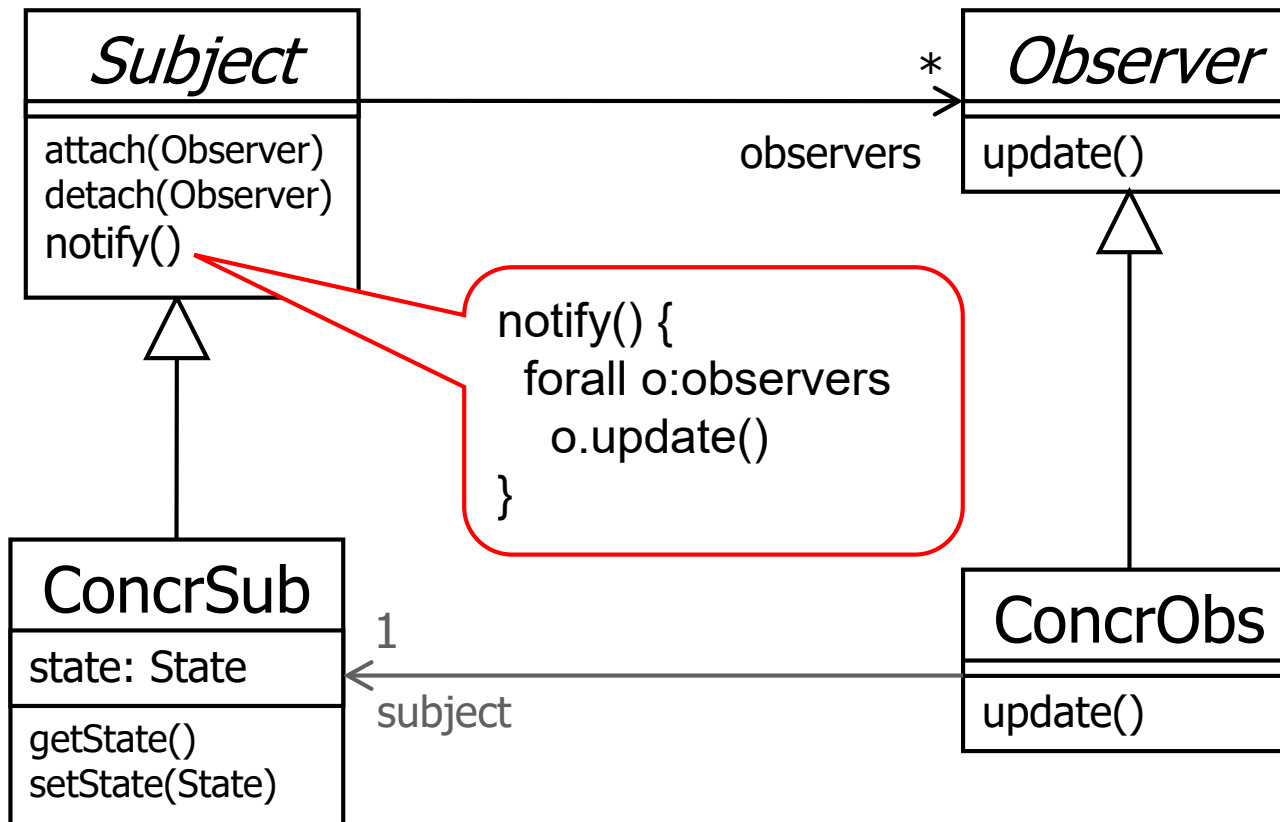
[...] maintain consistency between related objects without introducing tight coupling (which increases reusability) [...]

Typical Example

... update views when the underlying model changes ...

→ MVC!

Structure



Participants (see structure)



■ Subject

- knows its observers
- provides an interface for attaching and detaching Observer objects

■ Observer

- defines the updating interface for being notified

■ ConcreteSubject

- stores the state (of interest)
- sends notifications

■ ConcreteObserver

- Implements the Observer's updating interface to keep its state consistent

Se [GoF] eller [FF]
for detaljer

Name and classification

Singleton, object-based, creational

Intent

Ensure that a class has only one instance, and provide a global point of access to it [GoF]

Motivation

...

Bemærk: Tænk jer godt om, inden I bruger singletons (eller statiske attributter). Meget ofte bliver singletons brugt som "hack"!

Name and classification

Command, behavioural

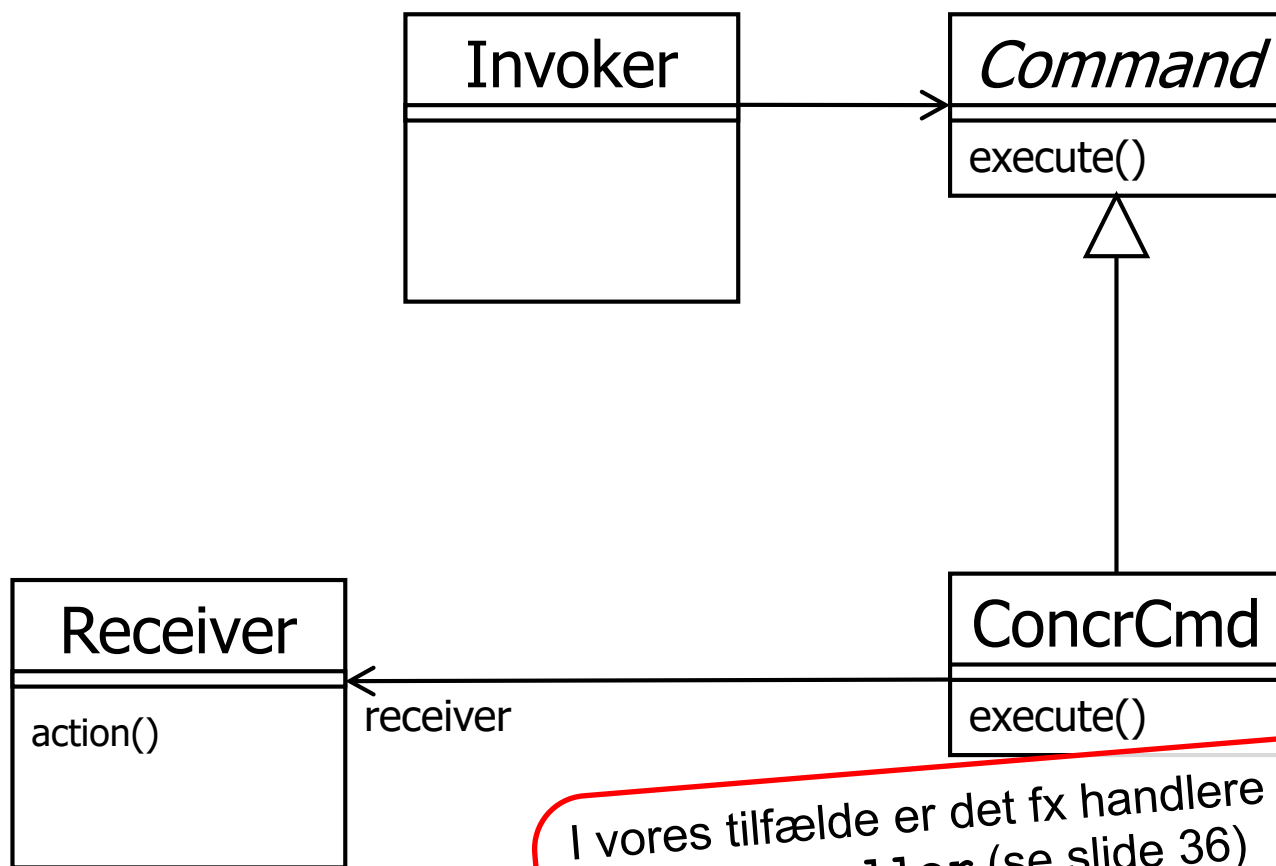
Intent

Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations [GoF]

Motivation

Use of different implementations in different contexts with easy portability ...

Structure



I vores tilfælde er det fx handlere der kalder **GameController** (se slide 36)
Generel er i Java **Runnable** som indkapsler kode som skal eksekveres af andre og dertil kan man bruge søkladte lambda-udtry (det vender vi tilbage til).

- **Variation / extension**

A command can also have **undo()** and **redo()** methods. This can be used to store executed commands on a undo-stack; by executing the **undo()** method in reverse order, earlier commands can be undo (and redone).

Denne slags kommandoer
er kernen i undo/redo-
mekanismen af mange
editorer!

Name and classification

Abstract factory, object, creational

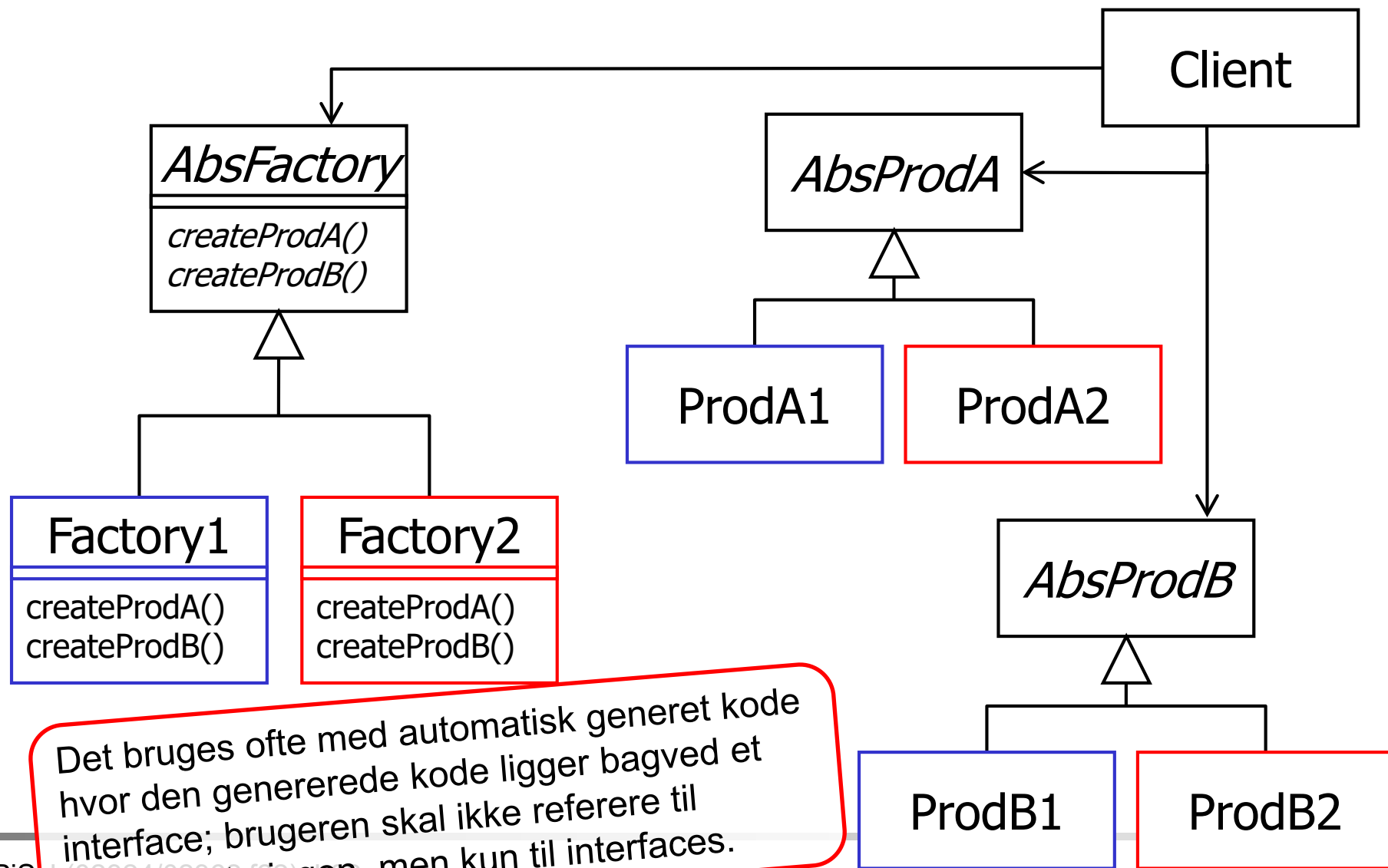
Intent

Provide an interface for creating families of related or dependent objects without specifying their concrete classes [GoF]

Motivation

Use of different implementations in different contexts with easy portability ...

Det vender vi måske tilbage til senere!



- GoF præsenterer 23 design patterns
- Der eksisterer mange flere (og nogle komplekse kombinationer af de enkle mønstre, fx. MVC)
- “Pattern terminologien” bruges ofte til at kommunikere designet! Derfor er det vigtigt at kende design patterns!
- Design patterns hjælper at forstå det overordne design af et stykke software
- Design patterns skal **ikke bruges** alt for skematiske
- Der eksisterer også **anti-patterns** (→ bad smells): hvordan man ikke skal programmeres

IV. Java Praksis

Det vil blive vist med
praktiske eksempler fra
vores projekt i IntelliJ
senere.

Her er der kun nogle
referencer!

DTU Compute

Department of Applied Mathematics and Computer Science

A collage of colorful mathematical symbols and formulas, including a Taylor series, integrals, and various constants.

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\int_a^b \epsilon \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} = \{2.7182818284\}$$
$$\chi^2 \sum!$$

- I jeres endelige projekt skal alle offentlige klasser, metoder og attributter have JavaDoc-kommentarer
- Se fx. pakken **designpatterns** i koden til jeres opgaver!
- Ud fra disse kommentarer kan der genereres HTML dokumentation af jeres kode
- See
[JD:HowTo] *The Javadoc Homepage: How to:*
<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>
[IJ:JavaDoc] *IntelliJ IDEA: Documenting code:*
<https://www.jetbrains.com/help/idea/working-with-code-documentation.html#generate-javadoc>

Eksempel

@xxx kaldes for
JavaDoc tags!

```
/**
 * This is the subject of the observer design pattern roughly
 * following the definition of the GoF.
 *
 * @author Ekkart Kindler, ekki@dtu.dk
 */
public abstract class Subject { ...
```

Java-metoder har typisk ikke
@author tags! Men i jeres
projekt skal I angive autorer til
individuele metoder, hvis ikke
en autor stor for hele klassen.

```
/**
 * This methods allows an observer to register with the subject
 * for update notifications when the subject changes.
 *
 * @param observer the observer who registers
 */
final public void attach(Observer observer) {
```

...

File Edit View History Bookmarks Tools Help

Generated Documentation (Untitled: X) +

file:///C:/Users/ekki/IdeaProjects/PiSD-f20/p ... Search

PREV NEXT FRAMES NO FRAMES

This is an example of a very simple JavaFX application following the MVC-pattern (slightly more strictly as usual).

See: Description

Packages

Package	Description
dk.dtu.compute.se.pisd.designpatterns.observer	This package implements the observer design pattern with the observed subjects represented as an abstract class Subject .

dk.dtu.compute.se.pisd.initial

dk.dtu.compute.se.pisd.initial

dk.dtu.compute.se.pisd.initial

dk.dtu.compute.se.pisd.initial

All Classes

Packages

dk.dtu.compute.se.pisd.designpatterns.c

dk.dtu.compute.se.pisd.initial

dk.dtu.compute.se.pisd.initial.controller

dk.dtu.compute.se.pisd.initial.model

dk.dtu.compute.se.pisd.initial.view

dk.dtu.compute.se.pisd.designpatterns.

Interfaces

Observer

Classes

Subject

This is an example of a very simple JavaFX application following the MVC-pattern (slightly more strictly as usual).

It was made for the course "Project in Software Development".

The purpose is to show and discuss the design patterns and assignments of the course will be based on this project of the course can be based on this.

Copyright (C) 2019, 2020: Ekkart

This project is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, Inc., version 3 of the License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

File Edit View History Bookmarks Tools Help

Generated Documentation (Untitled: X) +

file:///C:/Users/ekki/IdeaProjects/PiSD-f20/p ... Search

All Classes

Packages

dk.dtu.compute.se.pisd.designpatterns.c

dk.dtu.compute.se.pisd.initial

dk.dtu.compute.se.pisd.initial.controller

dk.dtu.compute.se.pisd.initial.model

dk.dtu.compute.se.pisd.initial.view

dk.dtu.compute.se.pisd.designpatterns.

Interfaces

Observer

Classes

Subject

```
public abstract class Subject
extends java.lang.Object
```

This is the subject of the observer design pattern roughly following the definition of the GoF.

Constructor Summary

Constructors

Constructor and Description
Subject ()

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
void	attach (Observer observer)	This methods allows an observer to register with the subject for update notifications when the subject changes.
void	detach (Observer observer)	This methods allows an observer to unregister from the subject again.
protected void	notifyChange ()	This method must be called from methods of concrete subclasses of this subject class whenever its state is changed (in a way relevant for the observer).

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait,

- Til at programmere GUIen bruger vi JavaFX
- Her er der nogle informationer om JavaFX programming:
- JavaFX Applikationen:
 - https://docs.oracle.com/javafx/2/get_started/hello_world.htm
 - https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm
- JavaFX LayoutPanels:
 - https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm#
- JavaFX UI Controls:
 - https://docs.oracle.com/javafx/2/ui_controls/overview.htm#
 - https://docs.oracle.com/javafx/2/ui_controls/text-field.htm#
 - https://docs.oracle.com/javafx/2/ui_controls/button.htm#
- JavaFX Dialogs:
 - <https://examples.javacodegeeks.com/desktop-java/javafx/dialog-javafx/javafx-dialog-example/>

Kreerer et nyt knap-objekt med label "Option 1"

```
Button optionButton = new Button("Option 1");
```

Tilknytter en handling (aktion) til knappen, som bliver eksekveret, når brugeren trykker på knappen.

```
optionButton.setOnAction(  
    e -> gameController.notImplemented() );
```

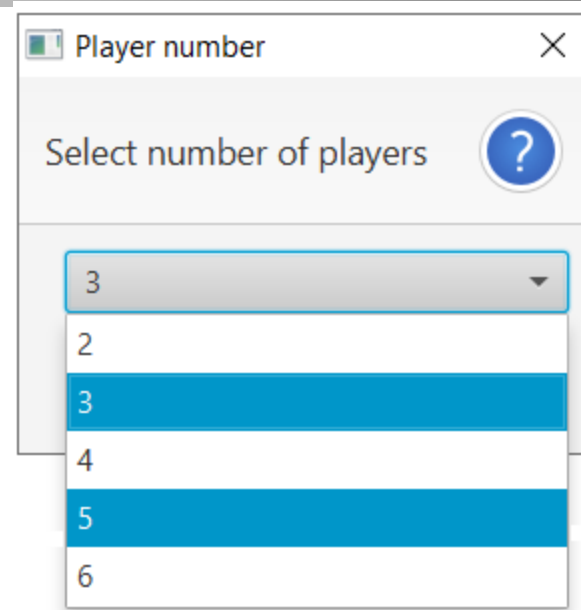
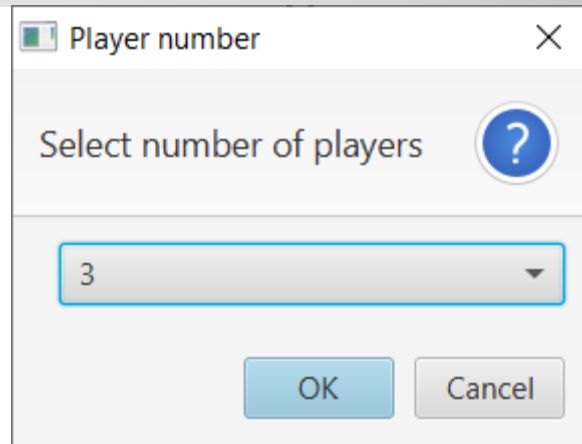
Siger om knappen skal vises disabled eller enabled.

```
optionButton.setDisable(false);
```

```
playerInteractionPanel.getChildren().add(optionButton);
```

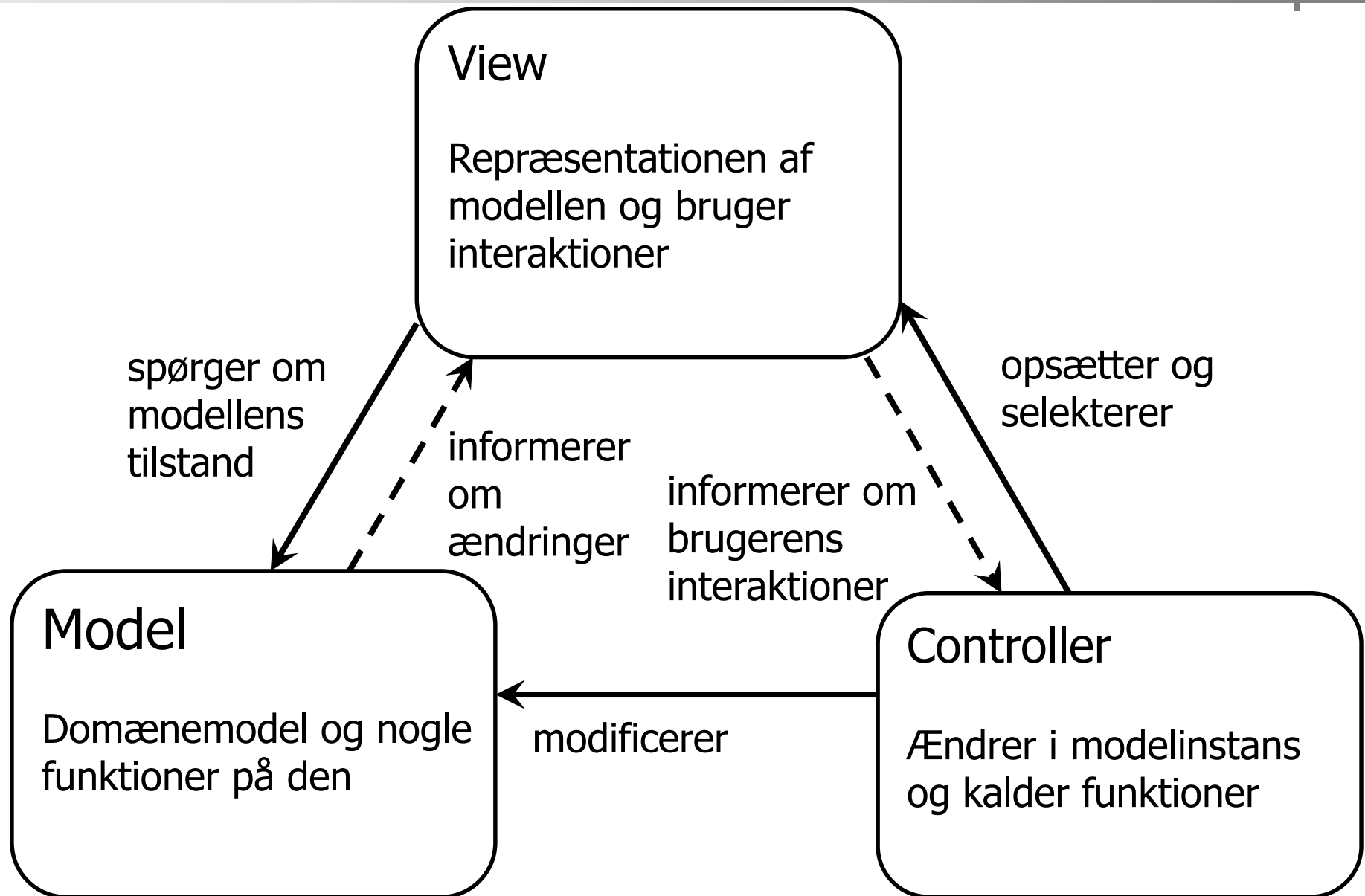
Tilføjer knappen til et andet GUI element (her spillerens panel). Først her bliver knappen synligt (på dette panel)!

Eksempel: ChoiceDialog



- Bemærk at dette er en modal dialog, dvs. brugeren kan først interagere med resten af GUI'en (eller forældre-dialog), når denne her dialog er afsluttet.

I **GameController** skal I helst ikke bruge (modale) dialoger! Se opgave V3, hvordan man kan undgå dette! Men under selve start af spillet (**AppController**) eller under afslutningen er det OK!



```
final private List<Integer> PLAYER_NUMBER_OPTIONS =  
    Arrays.asList(2, 3, 4, 5, 6);
```

...

```
Optional<Integer> result = null;  
do {
```

```
    ChoiceDialog dialog =
```

```
        new ChoiceDialog(PLAYER_NUMBER_OPTIONS.get(0),  
                        PLAYER_NUMBER_OPTIONS);
```

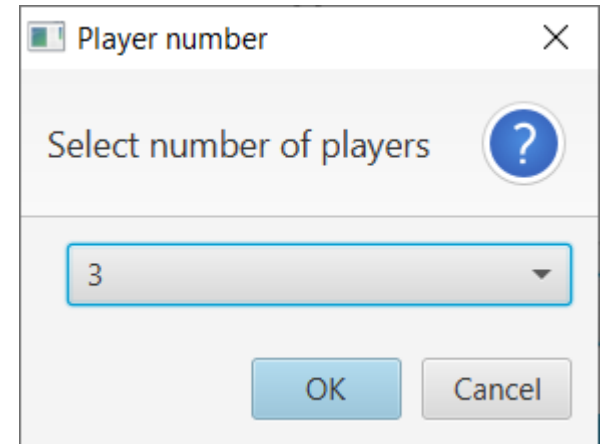
```
    dialog.setTitle("Player number");
```

```
    dialog.setHeaderText("Select number of players");
```

```
    result = dialog.showAndWait();
```

```
} while (!result.isPresent());
```

```
int no = result.get();
```



<http://www2.compute.dtu.dk/courses/02362/f22/opgaver/V03/>

- I får udleveret en udvidet implementering af kommandoer, som også indeholder et *interaktivt kommando*! Det kræver, at spilleren interagerer med softwaren, når sådan et kort bliver eksekveret (for at vælge hvad der skal ske).
- I skal udvide **GameController** og **PlayerView**, så at der vises knapper med de mulige optioner, når kortet bliver eksekveret. Og når spilleren trykker på en af disse knapper, skal **GameController** kaldes og det valgte kommando skal eksekveres for den aktuelle spiller (og eksekvering a robotternes programmer skal fortsættes).
- Derudover skal i dokumentere programmet med JavaDocs!

Detaljerede analyse/konceptuelle modeller til
RoboRally (**afleveringsopgave**):

- Taksonomi (tilstandsbegreber / aktionsbegreber)
- Klassediagrammer
- Tilstandsdiagrammer til relevante klasser (frivilligt)
- Aktivitetsdiagrammer til relevante aktiviteter (frivilligt)