

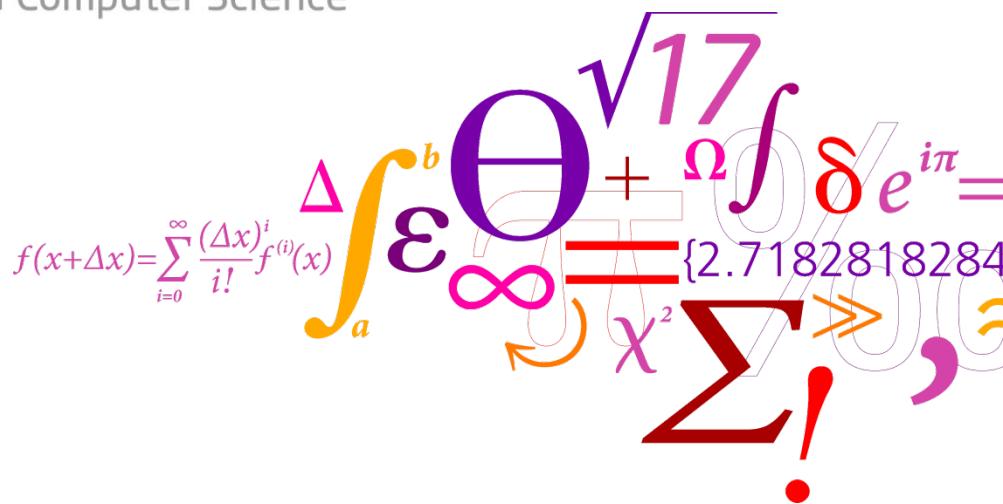
# Projekt i software-udvikling (02362)

## Forår 2022

Ekkart Kindler

**DTU Compute**

Department of Applied Mathematics and Computer Science

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


# I. Introduktion

Meget kort rekapitulation  
af uge 1

**DTU Compute**

Department of Applied Mathematics and Computer Science

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\Theta^{\sqrt{17}} + \Omega \int_a^b \delta e^{i\pi} =$$
$$\Sigma! \gg,$$
$$\infty = \{2.7182818284590452353602874713526624977572470636231870679821480865132521609611781595891565953352076143444147$$

# Opfriskning (eksempel)

```
class Pos {  
    int x;  
    int y;  
}  
  
class PosName {  
    String name;  
    Pos pos;  
    public PosName(String name, Pos pos) {  
        this.name = name;  
        this.pos = pos;  
    } }
```

Denne kode er ikke påen!

Kun et teknisk eksempel,  
som opfriskning af nogle  
begreber og tænkemåder!

# Opfriskning (eksempel)

```
class Test {  
    public static void main(String[] args) {  
        Pos pos1 = new Pos();  
        pos1.x = 1;  
        pos1.y = 2;  
  
        Pos pos2 = pos1;  
        pos2.x = 3;  
        pos2.y = 4;
```

Hvilke værdier har  
`pos1.x`, `pos2.x`,  
`pos1.y` og `pos2.y`  
når programmet når her?  
Og hvorfor?

→ Se diskussion på tavlen!

# Opfriskning (eksempel)

```
PosName posName1 = new PosName("position 1", pos1);  
PosName posName2 = new PosName("position 2", pos2);
```

```
Pos[] posArray = new Pos[10];  
for (int i = 0; i < posArray.length; i++) {  
    posArray[i].x = i;  
    posArray[i].y = i;  
}
```

Det ville udløse en  
NullPointerException

```
posArray[1] = pos1;  
posArray[2] = pos2;
```

# Opfriskning (eksempel)

```
PosName posName1 = new PosName("position 1", pos1);  
PosName posName2 = new PosName("position 2", pos2);
```

```
Pos[] posArray = new Pos[10];  
for (int i = 0; i < posArray.length; i++) {  
    posArray[i] = new Pos();  
    posArray[i].x = i;  
    posArray[i].y = i;  
}
```

Hvordan ville  
værdierne se ud her?

```
posArray[1] = pos1;  
posArray[2] = pos2;
```

→ Se diskussion på  
tavlen!

# Model View Controller (MVC)

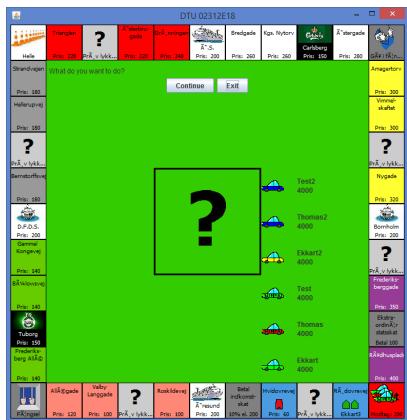
DTU Compute

Department of Applied Mathematics and Computer Science

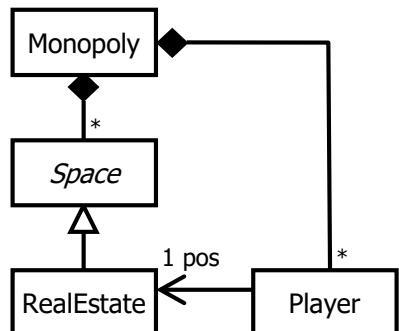
Ekkart Kindler



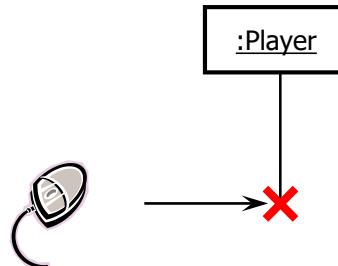
## View



## Model

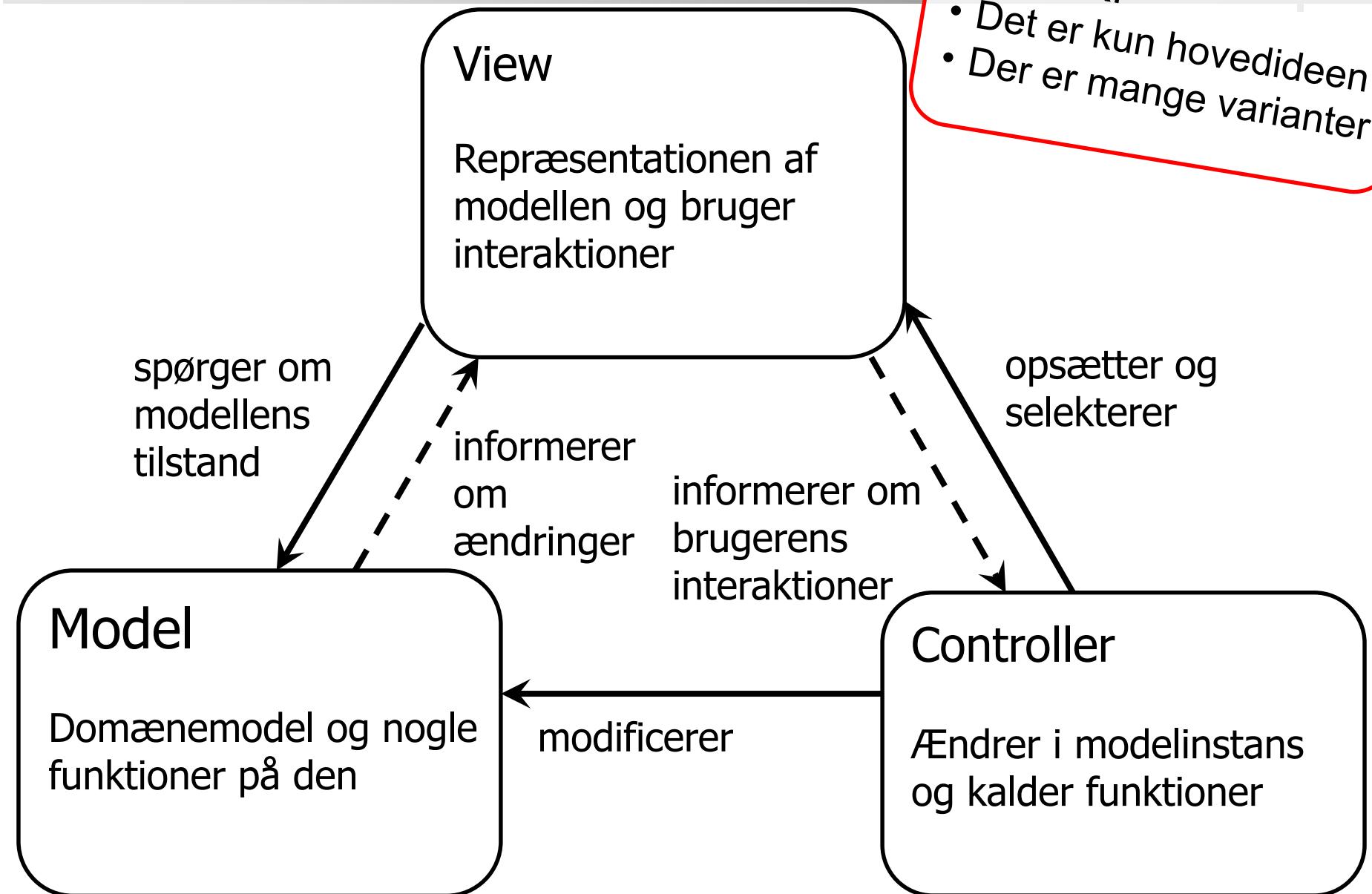


## Controller

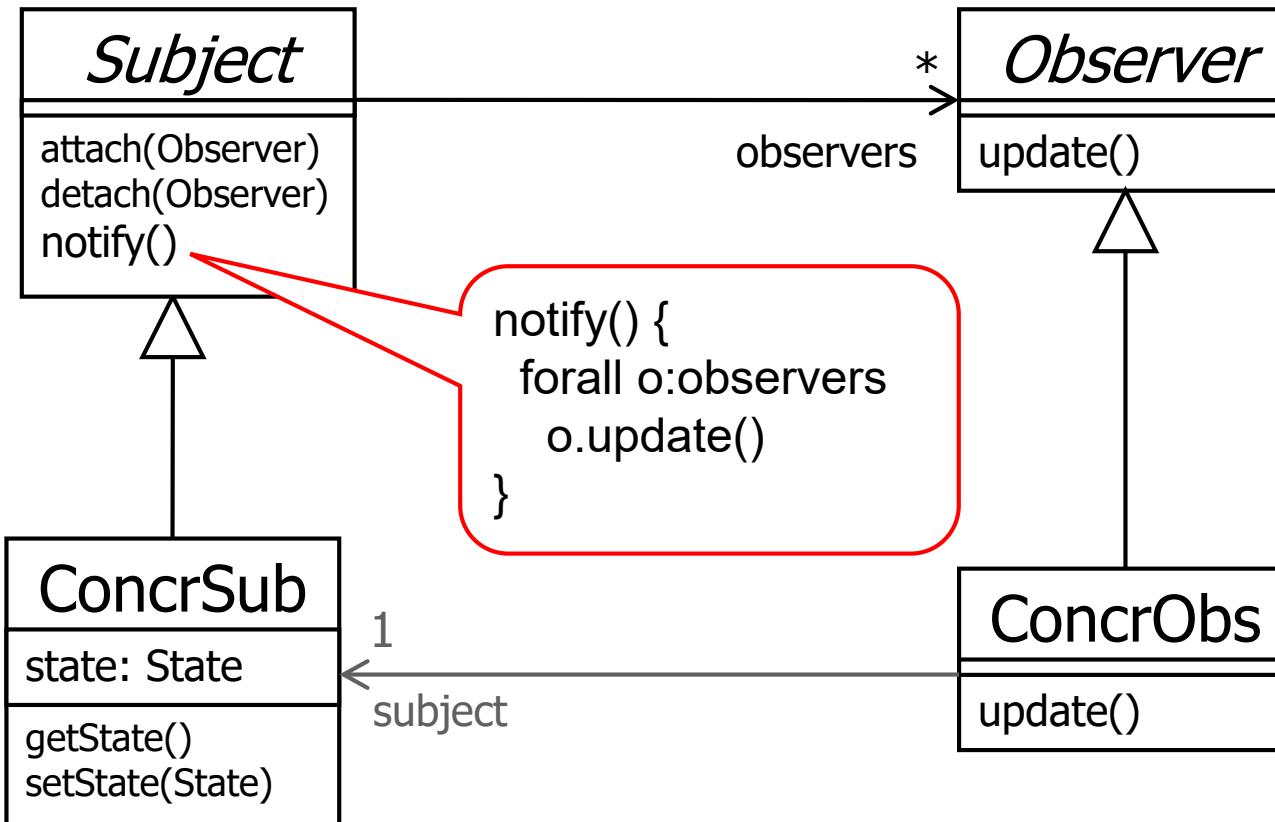


## Bemærk:

- Det er kun hovedideen
- Der er mange varianter



## Structure



# Diskussion: Eksempel

The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "rob orally". It contains a "src" directory with "main" and "test" packages. "main.java" contains "dk.dtu.compute.se.pisd" and "rob orally" packages, which further contain "controller" and "GameController" classes. "test.java" contains "dk.dtu.compute.se.pisd.rob orally.controller" package with "GameControllerTest" class.
- Code Editor:** The "GameController.java" file is open. The code implements a controller for the RoboRally game, handling player turns and programming phases. A red callout bubble in the bottom right corner contains the text "RoboRally 1.1".
- Toolbars and Menus:** Standard IDE menus like File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, DB Navigator, Help are visible at the top.
- Status Bar:** The status bar at the bottom shows "Build completed successfully in 9 sec, 493 ms (8 minutes ago)".
- Right Panel:** A "Maven" panel is visible on the right side of the interface.

```
private void continuePrograms() {
    do {
        executeNextStep();
    } while (board.getPhase() == Phase.ACTIVATION && !board.isStepMode());
}

// XXX: V2
private void executeNextStep() {
    Player currentPlayer = board.getCurrentPlayer();
    if (board.getPhase() == Phase.ACTIVATION && currentPlayer != null) {
        int step = board.getStep();
        if (step >= 0 && step < Player.NO_REGISTERS) {
            CommandCard card = currentPlayer.getProgramField(step).getCard();
            if (card != null) {
                Command command = card.command;
                executeCommand(currentPlayer, command);
            }
            int nextPlayerNumber = board.getPlayerNumber(currentPlayer) + 1;
            if (nextPlayerNumber < board.getPlayersNumber()) {
                board.setCurrentPlayer(board.getPlayer(nextPlayerNumber));
            } else {
                step++;
                if (step < Player.NO_REGISTERS) {
                    makeProgramFieldsVisible(step);
                    board.setStep(step);
                    board.setCurrentPlayer(board.getPlayer(0));
                } else {
                    startProgrammingPhase();
                }
            }
        } else {
            // this should not happen
            assert false;
        }
    } else {
        // this should not happen
        assert false;
    }
}
```

Designkapitlet (II)  
fortsætter vi med  
sener!

### III. Analyse og konceptuelle modeller

Nu starter vi forfra igen  
med analysen eller "hvad"  
softwaren skal gøre!

DTU Compute

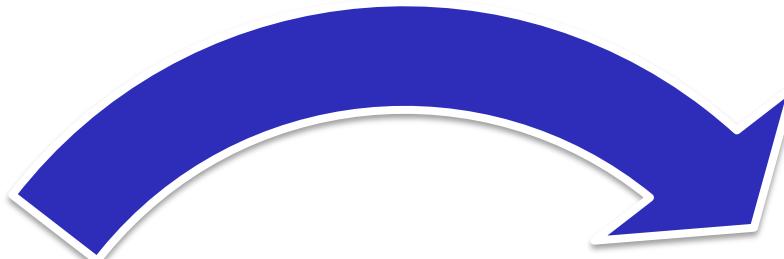
Department of Applied Mathematics and Computer Science

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\Theta^{\sqrt{17}} + \Omega \int_a^b \delta e^{i\pi} =$$
$$\Sigma! \gg \chi^2 = \{2.71828182845904523536028747135266249775724706362351902559337106152382364590457250463291140773189550980354628566180360135110243375075388782998940378951936$$

**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

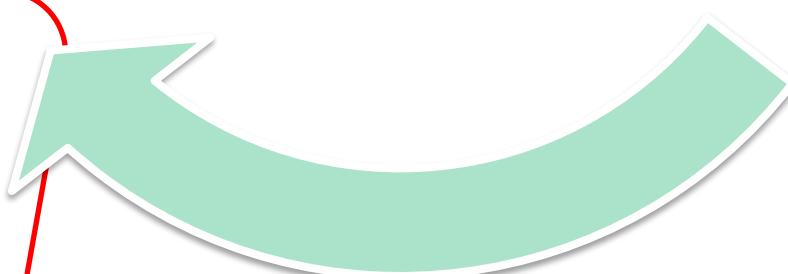
“Hvad” skal  
softwaren  
gøre?



# WHAT

# HOW

Realiteten er mere  
kompliceret, men at  
adskille “what” og  
“how” i jeres hoved er  
et første.



“Hvordan” er  
softwaren  
realiseret?

Inden man kan implementere et stykke software er man nødt til af finde ud af **hvad** selve software skal gøre og formulere det fra brugerens synsvinkel:

- Hvilke brugere er der?
- Hvilke begreber (engl. concepts) spiller en rolle
- Hvordan hænger begreberne sammen?
- Hvilke funktioner (use-cases/aktiviteter) er der?  
Hvad indebærer de og hvornår kan man gennemføre dem?

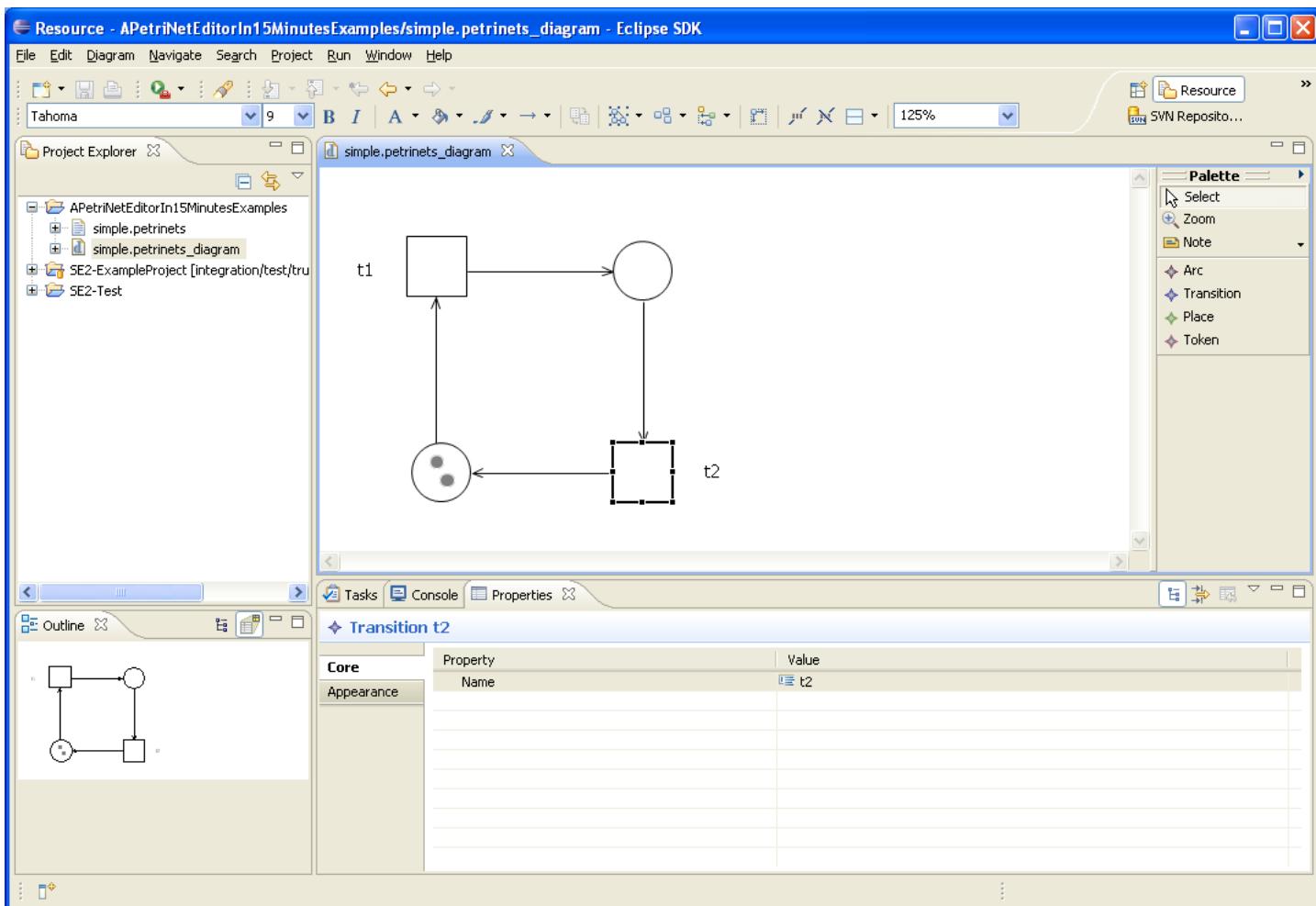
I vores projekt er det mest af alt: RoboRally's regler

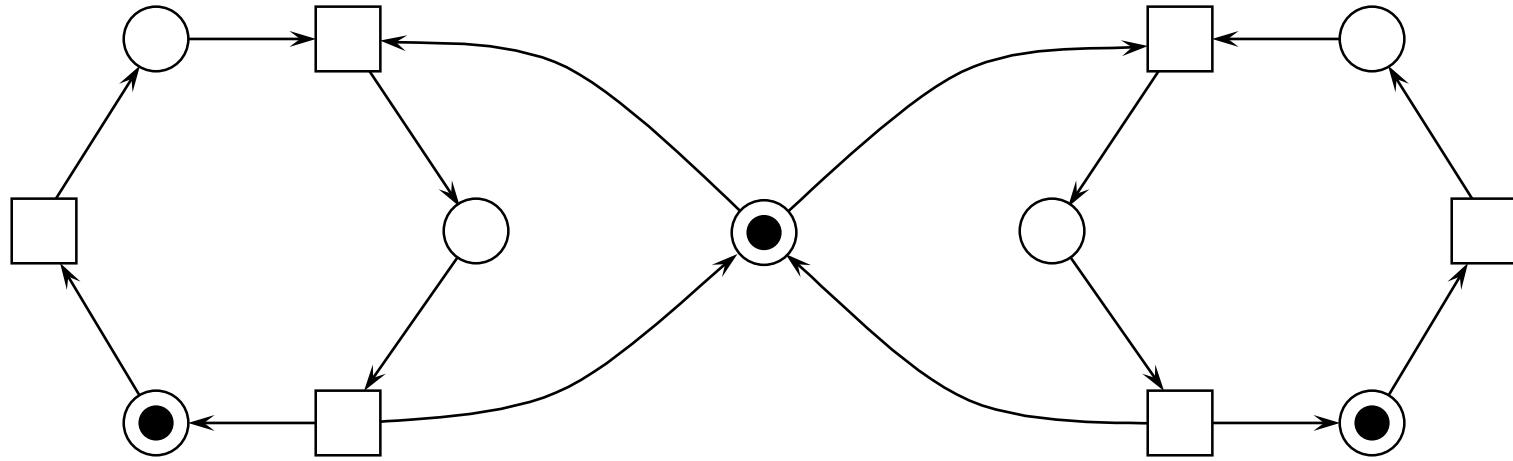
- Find skriftlig dokumentation om området (domænet) og skriv det ud
- Marker alle begreber som synes relevante
- Tal med nogle eksperter fra domænet
- Diskuter indbyrdes og saml informationerne op og strukturer den

- For at samle informationerne op give dem struktur:
  - **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

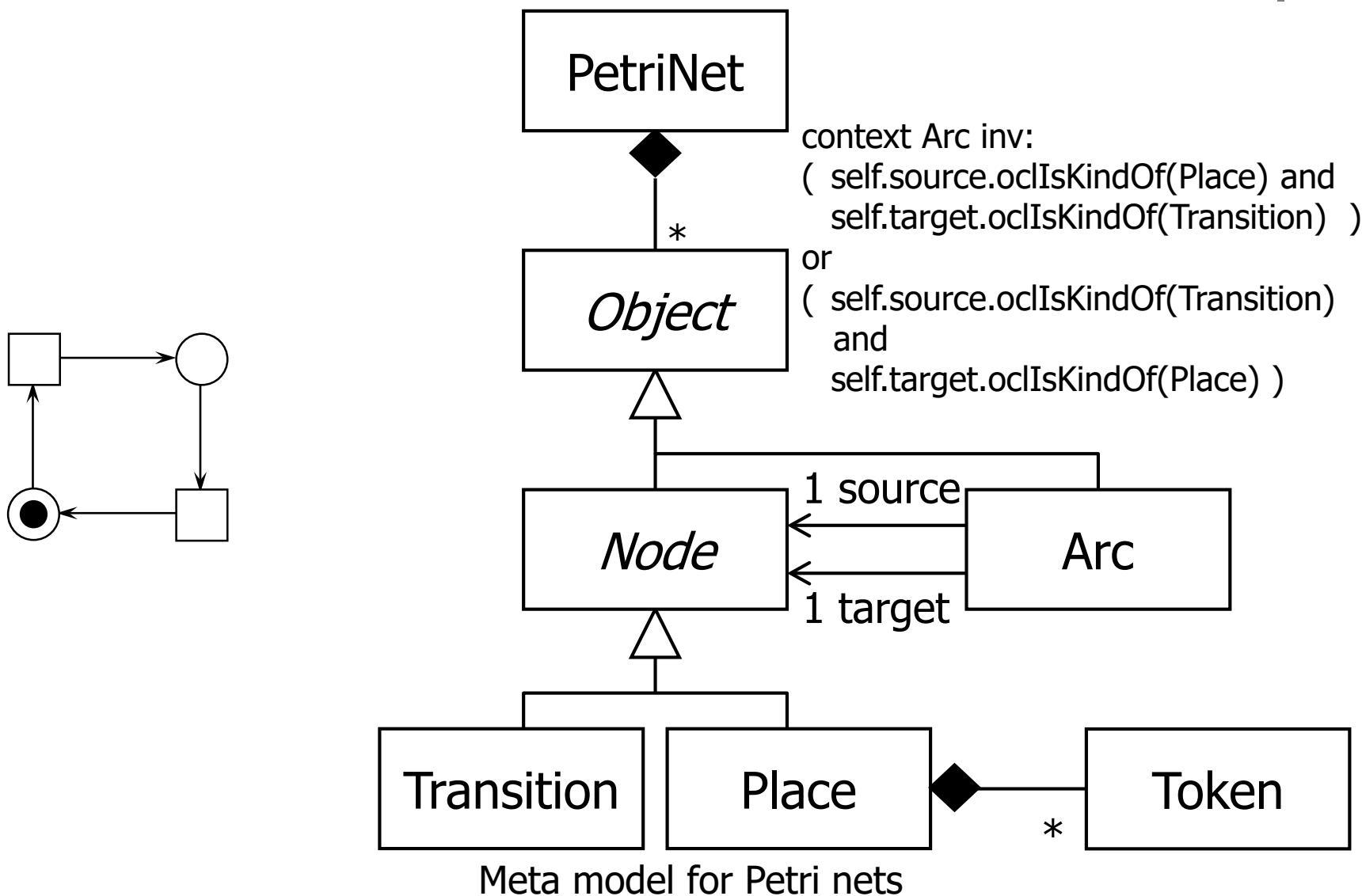




- Eksempler
- Taksonomi (på tavlen)
- Glossar
- Model (på tavlen)

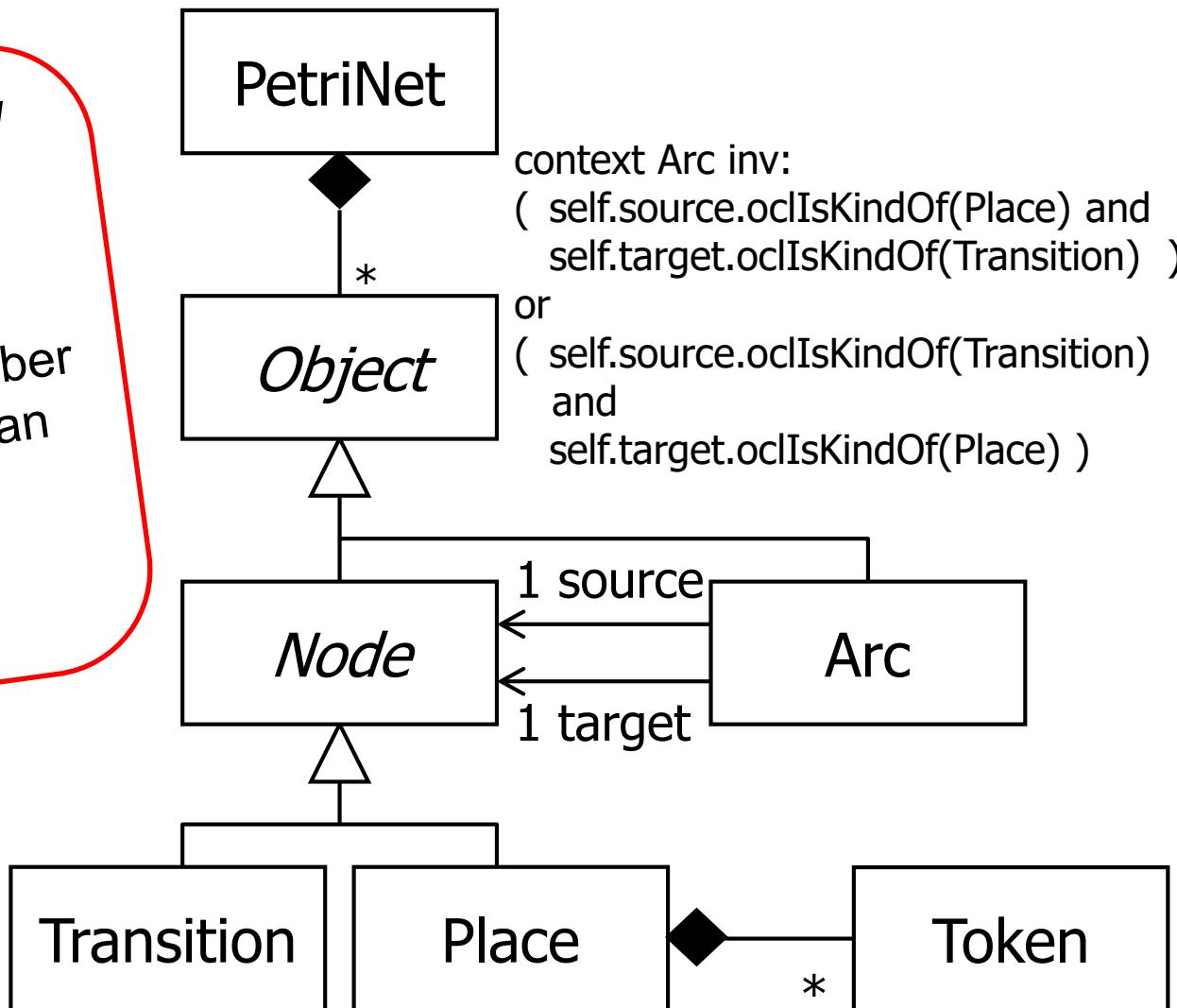
**Regel:** I skal først starte med at lave et UML diagram, hvis I har kigget på nogle eksempler først og har en liste med hovedbegreberne (taksonomi)!

# Model (Meta Model)

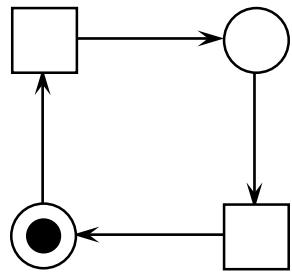


# Model (Meta Model)

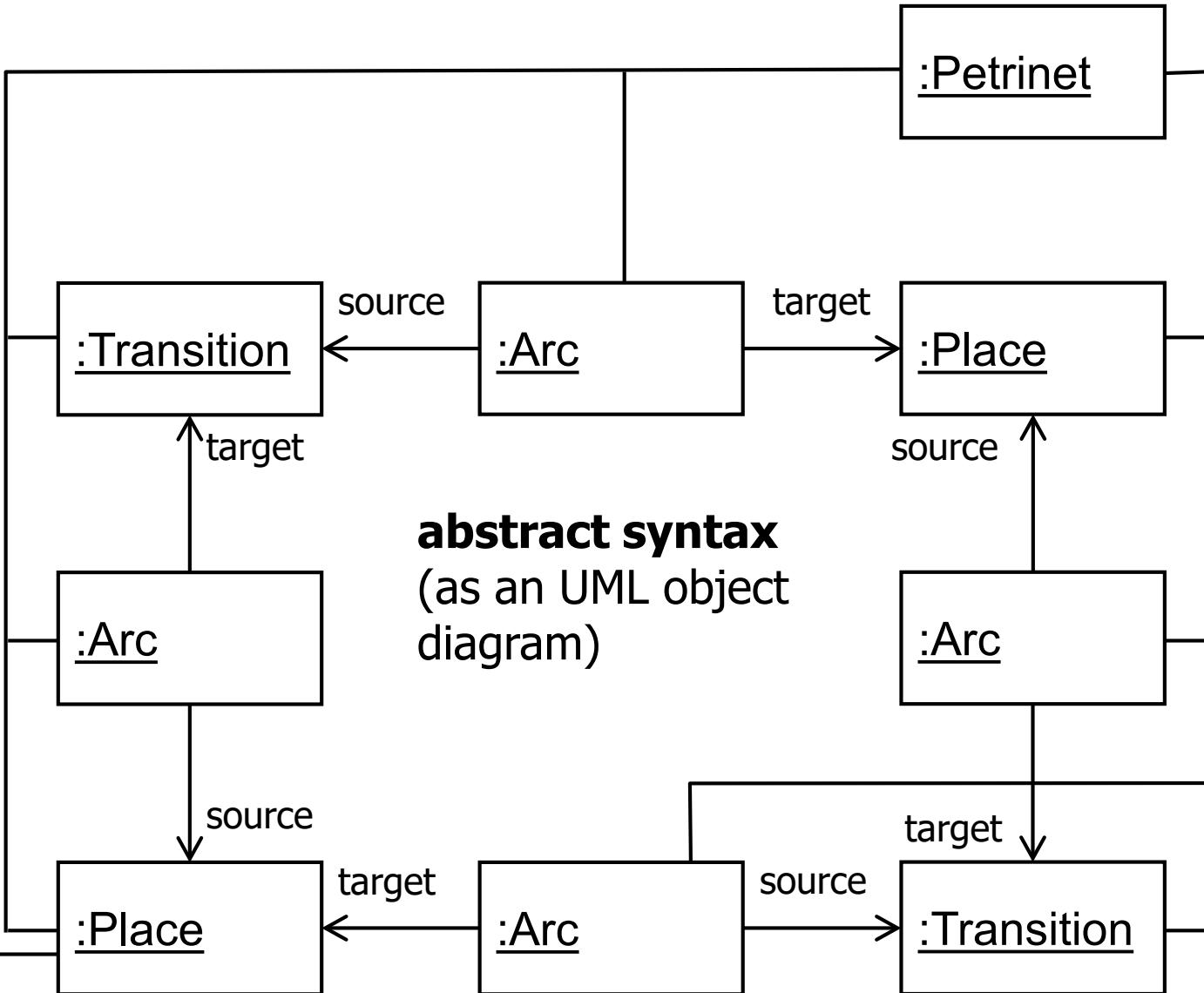
**Regel:** Under analysen / konceptuel modellering, tænk ikke på programmering eller Java! Det er kun begreber fra domænet og hvordan de relater til hinanden (Petrinet i vores eksempel)!

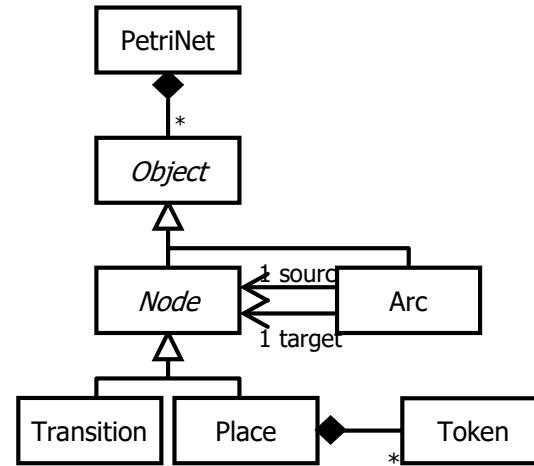
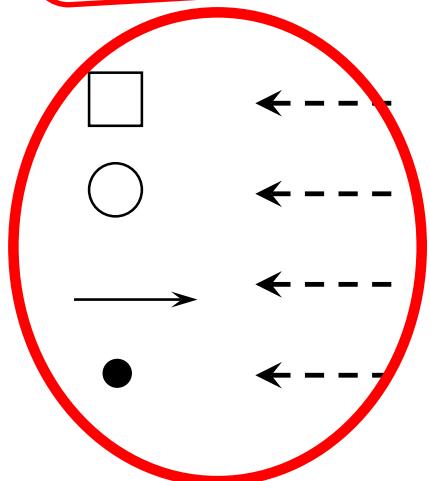


# Syntaks (abstrakt and konkret)

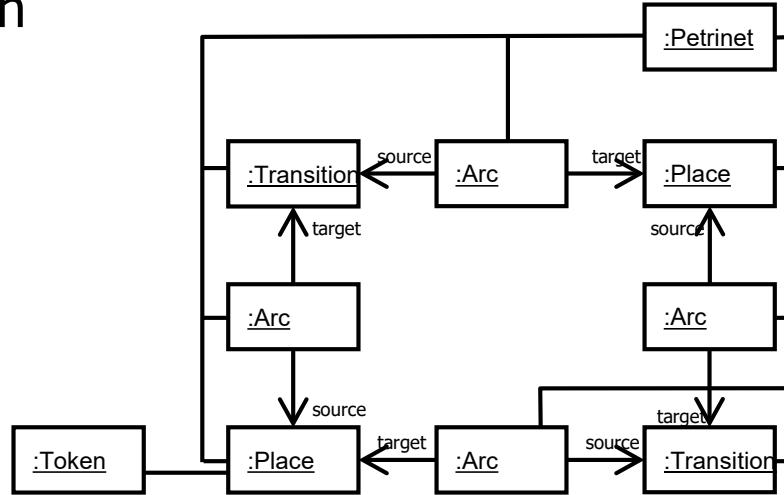
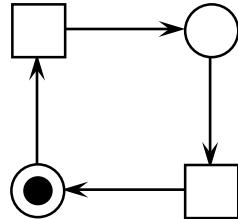


graphical /  
**concrete**  
**syntax**





generate an editor

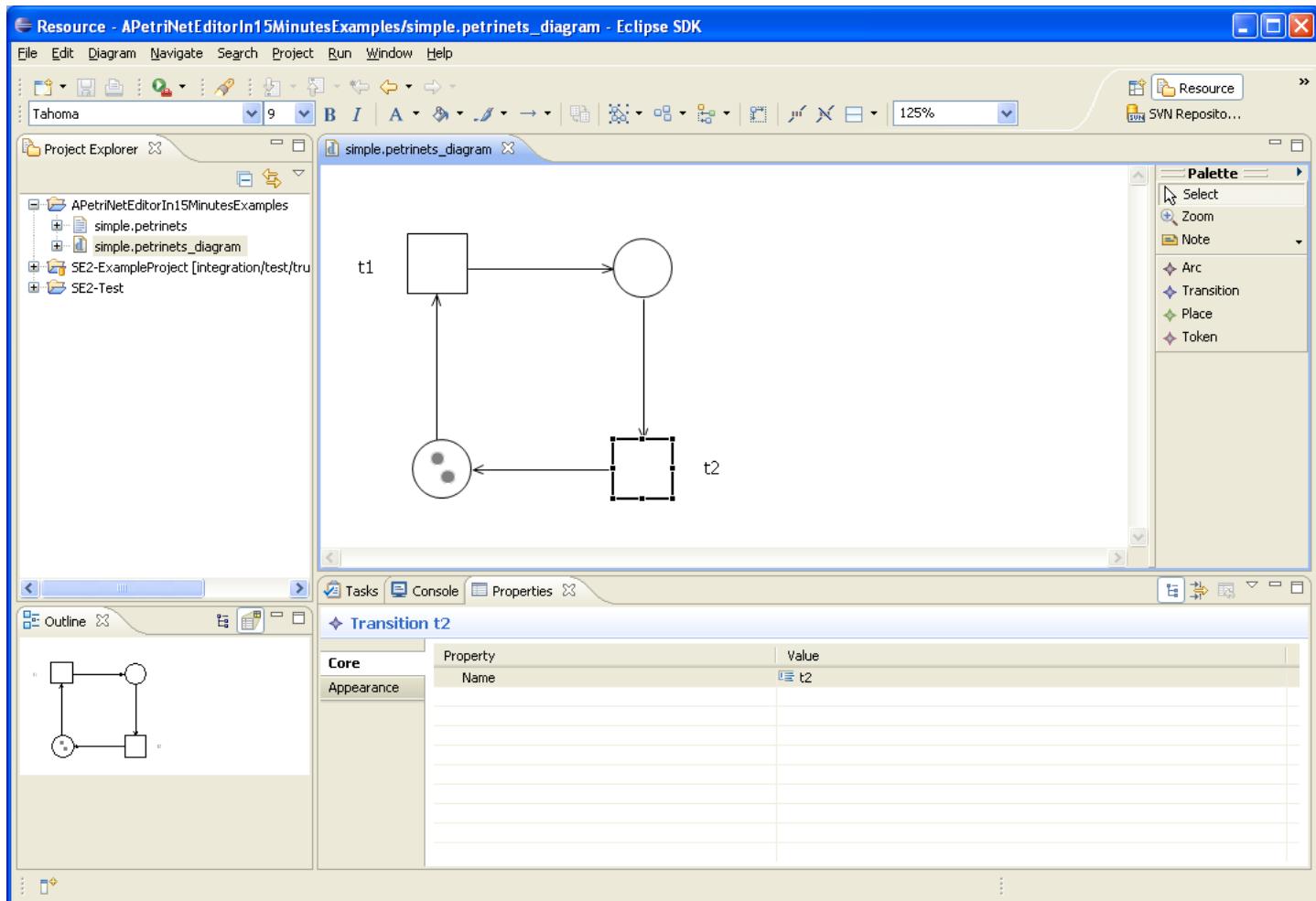


# Resultat

DTU Compute

Department of Applied Mathematics and Computer Science

Ekkart Kindler



# 3. Eksempel: RoboRally

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

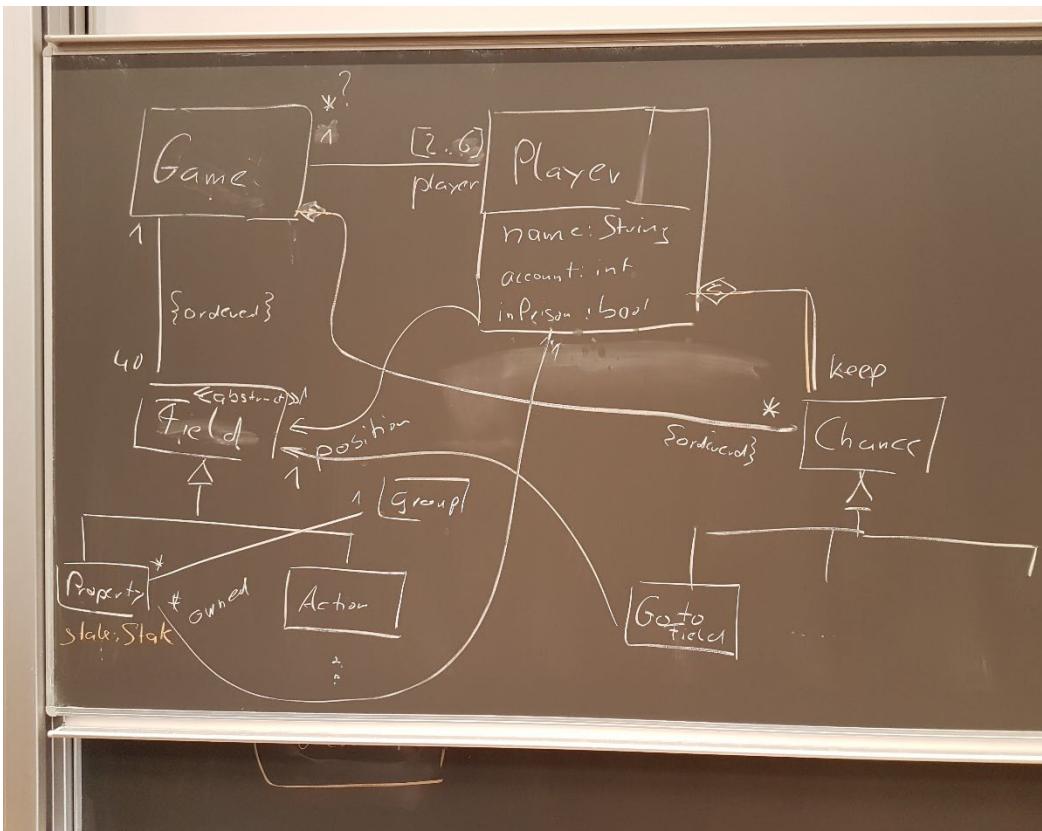
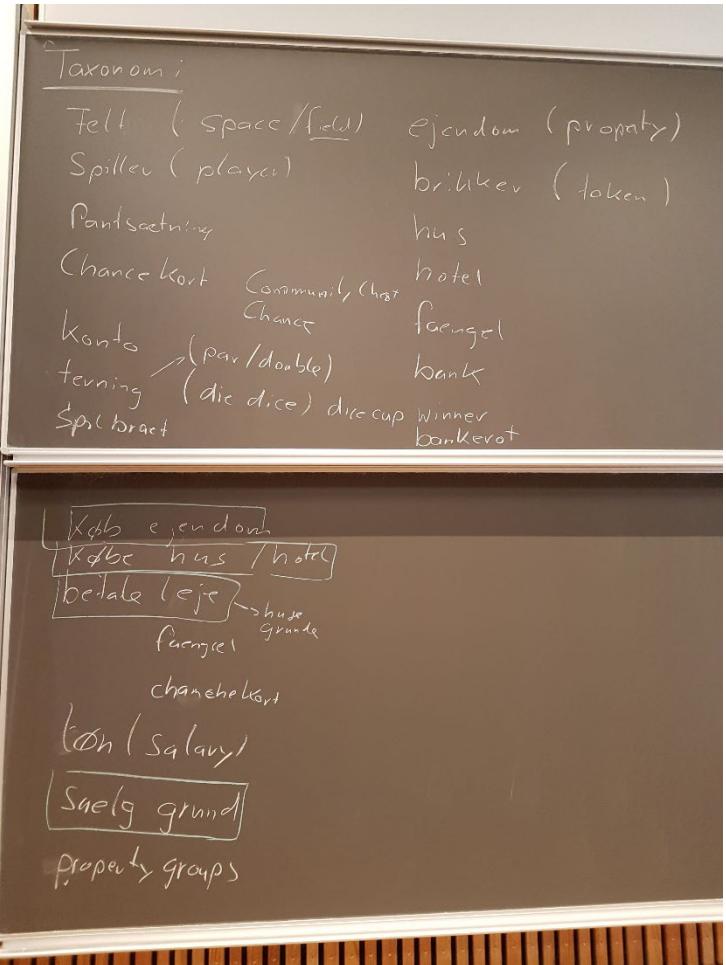
1. Find beskrivelsen af RoboRally frem (se projektslides og især links til RoboRally's regler)
2. Taksonomi
3. Glossar
4. Modeller (klassediagrammer, tilstandsdiagrammer, aktivitetsdiagrammer)

Det springer vi over her,  
men det kan gerne være  
en del af jeres rapport!

Diskuter kort i jeres i grupper ...

... diskussion med hele klassen.

# Fra de tidelige år: Monopoly/Matador



Vi starter med livscyklus af udvalgte objekter, da de er typisk nemmere og tætter 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 tilstår være et attribut af klasser i klassediagrammet (eller de kan afledes af nogle af objektets andre attributter).

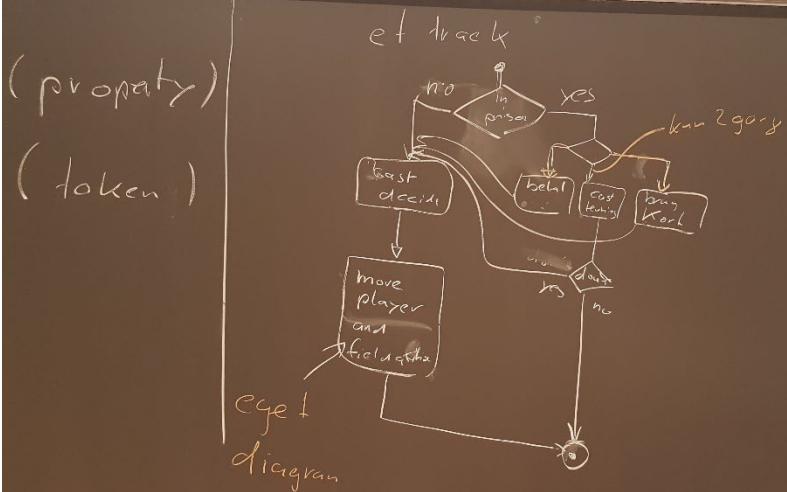
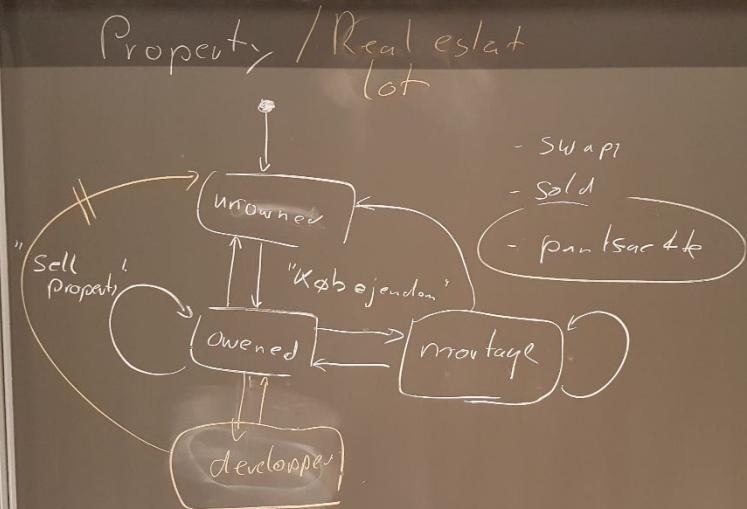
## Vi diskuterer udtræk af nogle udvalgte aktiviteter:

- lav et træk???
- ...

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 de tidelige år: Monopoly/Matador



Ud over taksonomien og diagrammerne, skal analysen også indeholde

- Tekst som beskriver diagrammerne
- Diskussion af krav, som ikke er beskrevet igennem taksonomien og diagrammerne (ikke-funktionelle krav):
  - brugbarhed
  - performance
  - vedligeholdbarhed
  - ...
- GUI

I vores projekt kommer hoveddelen af GULen fra mig og det er ikke hovedfokus! Derfor kan dens beskrivelse være meget kort (udover nogle særlige features fra jer).

<http://www2.compute.dtu.dk/courses/02362/f22/opgaver/V02/>

Bemærk at opgave A1 beskrives sammen med opgave V2!

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

- Taksonomi (tilstandsbegreber / aktionsbegreber)
- Klassediagrammer
  - De er først påkrævet med aflevering A2: Projektdefinition
- Tilstandsdigrammer til relevante klasser (frivilligt)
- Aktivitetsdiagrammer til relevante aktiviteter (frivilligt)

# RoboRally 1.1.0 → ???

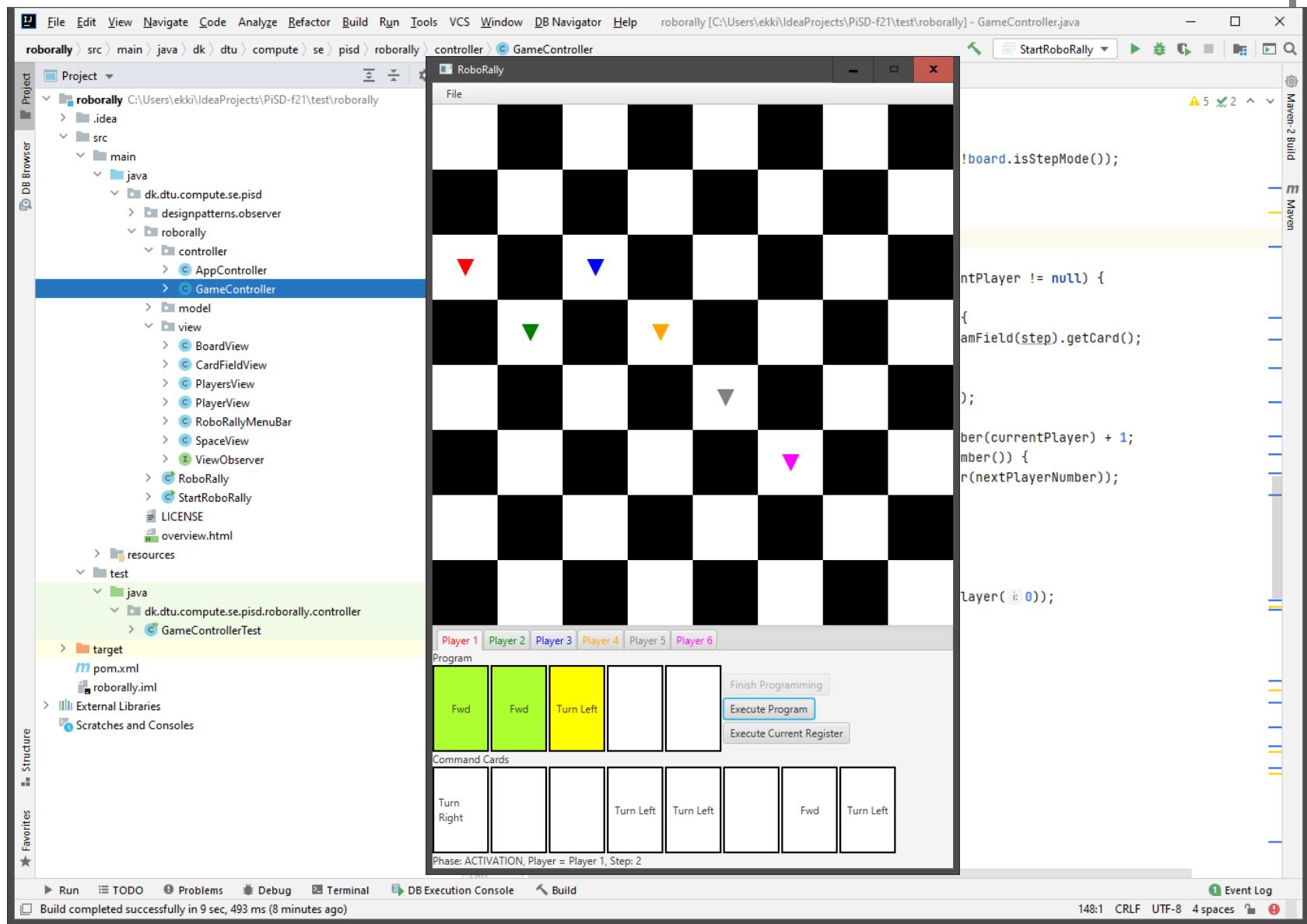
The screenshot shows an IDE interface with the following details:

- Project Structure:** The project is named "rob orally". It contains a "src" directory with "main" and "test" packages. "main.java" contains "dk.dtu.compute.se.pisd" and "rob orally" packages, which further contain "controller" and "view" packages. "controller" contains "AppController" and "GameController" classes, both of which are selected in the Project tree.
- Code Editor:** The main editor window displays the "GameController.java" code. The code implements methods for continuing programs and executing the next step based on the current phase and player's turn.
- Toolbars and Menus:** Standard IDE menus like File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, DB Navigator, and Help are visible at the top.
- Bottom Status Bar:** The status bar shows the build log: "Build completed successfully in 9 sec, 493 ms (8 minutes ago)".
- Right Sidebar:** A sidebar on the right shows build logs for Maven 2.2.1 and Maven 2.2.2.

```
private void continuePrograms() {
    do {
        executeNextStep();
    } while (board.getPhase() == Phase.ACTIVATION && !board.isStepMode());
}

// XXX: V2
private void executeNextStep() {
    Player currentPlayer = board.getCurrentPlayer();
    if (board.getPhase() == Phase.ACTIVATION && currentPlayer != null) {
        int step = board.getStep();
        if (step >= 0 && step < Player.NO_REGISTERS) {
            CommandCard card = currentPlayer.getProgramField(step).getCard();
            if (card != null) {
                Command command = card.command;
                executeCommand(currentPlayer, command);
            }
            int nextPlayerNumber = board.getPlayerNumber(currentPlayer) + 1;
            if (nextPlayerNumber < board.getPlayersNumber()) {
                board.setCurrentPlayer(board.getPlayer(nextPlayerNumber));
            } else {
                step++;
                if (step < Player.NO_REGISTERS) {
                    makeProgramFieldsVisible(step);
                    board.setStep(step);
                    board.setCurrentPlayer(board.getPlayer(0));
                } else {
                    startProgrammingPhase();
                }
            }
        } else {
            // this should not happen
            assert false;
        }
    } else {
        // this should not happen
        assert false;
    }
}
```

# RoboRally 1.1.0 → ???



<http://www2.compute.dtu.dk/courses/02362/f22/opgaver/V02/>

I fik udleveret en enkel implementering af RoboRally-spillet (som del af Opgave V1):

- Tilknyt knapperne (i viewer) til de rigtige metoder i GameController
- Implementer robottens kommandoer i GameController
- Tilføj test til de ny-implementerede metoder i GameController

To test findes der allerede i kommentarer i test klassen (dem kan i bruge); men der skal tilføjes flere.

Flere detaljer om opgaven og løsningen i en separat præsentation