

Projekt i software-udvikling (02362)

Forår 2021

Ekkart Kindler

DTU Compute

Department of Applied Mathematics and Computer Science

Bemærk: Kun slides 2-3 og
10-15 er nye. Alle andre slides
blev præsenter allerede!

A vibrant collage of mathematical symbols and numbers. It features a large purple integral symbol with a yellow boundary, a red summation symbol with a red exclamation mark, a purple infinity symbol, a red theta symbol with a purple circle, a red delta symbol with a red plus sign, and a red e to the power of i pi symbol. The background is white with faint, semi-transparent mathematical drawings like a coordinate system and a bell curve.

X. Projekt: Aflevering og Eksamens

DTU Compute

Department of Applied Mathematics and Computer Science

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\Theta^{\sqrt{17}} + \Omega \int_a^b \delta e^{i\pi} =$$
$$\infty = \{2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138214808681319529359027997266146389132$$
$$\Sigma! \gg,$$

Info og Tjeklist

Projekt i software-udvikling (02362): Aflevering — Mozilla Firefox

Projekt i software-udvikling (02362) +

www2.compute.dtu.dk/courses/02362/f21/aflevering.shtml

Search

DTU Compute

Department of Applied Mathematics and Computer Science

DTU

02362: PROJEKT I SOFTWARE-UDVIKLING (F21)

Informationer om den endelige aflevering

Det endelige projekt skal afleveres i grupper igennem DTU Learn (aflevering "Endelig Projektaflevering"). Bemærk at der skal afleveres selve software (som et eksporteret IntelliJ-projekt) og en rapport (som PDF-fil), hvor alle autorerne af de forskellige bidrag af gruppens medlemmer er markeret (i koden og rapporten).

Afleveringsfrist er **søndag, den 16. maj 2021, kl. 23⁵⁹**.

Bemærk at aflevering af software skal indeholde alt hvad man har brug for til at starte selve software fra IntelliJ, og softwaren skal kunne bygges automatisk med Maven. Aflevering skal også indeholde instrukserne hvordan man skal opsætte og konfigurere databasen og selve software, så at de kører sammen.

Den endelige aflevering skal indeholde:

1. Al kode og alle nødvendige konfigurations-filer, så at man kan importere og starte IntelliJ-projektet på en anden computer end jeres egen.
2. Autor-tags på metode-niveau, som markerer hvem der har bidraget med hvad til implementeringen. Hvis et autor-tag ikke er studienummeret (som det helst skal være) eller den studerendes fulde navn, skal der inkluderes en liste med hvilke tags der svarer til hvilke studerende.
3. Unit-tests, som I har brugt til at teste jeres software.
4. Instrukser hvordan man skal installere, konfigurere og starte selve software; det gælder især opsætning og konfiguration af databasen.
5. Selve rapport, som indeholder al information, som vi har diskuteret før: se [PiSU-projekt.pdf](#) og [PiSU-projekt-2.pdf](#)
6. I rapporten skal der markeres, hvem der har skrevet hvad (på underafsnitsniveau).

Tjekliste:

1. Er alle nødvendige filer med i afleveringen (det gælder også filer som er nødvendige til at opsætte og databasen)?
2. Kan softwaren kompileres, installeres (for Java 8 eller Java 15 med Maven) og startes på en anden computer, når man følger jeres instrukser og kun bruger jeres afleverede filer. I må antage, at brugeren har installeret Java 8, Java 15 og IntelliJ IDEA (Community Edition) allerede, og at en MySQL-server kører lokalt på samme computer (localhost). Afnør om I kan installere og køre softwaren på en anden computer, ud fra det projekt I afleverer.
3. Er referencer til gammel eller irrelevant kode slettet ("Optimize Imports"). Er der ikke længere fejl eller advarsler i projektet?
4. Er der Javadoc til al relevant kode (husk dem som I skulle tilføje i starten af projektet)?
5. Er der Java @authz tags i koden? Og er de opdateret?
6. Er rapporten komplet og forståelig?
7. Er alle bidrag af alle gruppens medlemmer markeret i rapporten?

Se <http://www2.compute.dtu.dk/courses/02362/f21/aflevering.shtml>

Rapport (indhold)

Se PiSU-projekt.pdf og også
eksemplet (materiale på
DTU Learn uge 13)!

■ Introduktion

- Kort overblik over projektet og hovedpunkter, som opgaven indebærer
- Overblik over selve rapporten og hvordan den skal læses

■ Problembeskrivelse

- Kontekst og baggrund
- Overblik over hovedfunktioner (i grupper) med prioritering og argumentation for denne (kan fx bruge MoSCoW-kategorier: Must, Should, Could, Won't)
- Husk også at nævne, hvilke data, der skal gemmes permanent (persistence)
- Hvad kan I bygge videre på (fx GUI og kode i fik fra Ekkart v1.0, v1.1, v1.4)

- Kravspecifikation (Analyse)
 - Beskrivelse af krav fra brugerens perspektiv
 - Beskriv funktioner og use-cases (som brugeren opfatter dem)
 - Beskriv de relevante informationer som domænemodel:
 - Klassediagrammer
 - Aktivitetsdiagrammer
 - evt. tilstandsdiagrammer
 - Beskriv ikke-funktionelle krav: fx brugbarhed (usability), vedligeholdbarhed (maintainability), ...
 - Evtl. UC-diagram med beskrivelse af de vigtigste use-cases

Husk: Alle diagrammer skal forklares i teksten!

■ Database- og softwaredesign

Overblik over de vigtigste komponenter, som inkluderer, også tilkobling a databasen: arkitektur

- Software design:
hovedkomponenter af software og
deres sammenspil
 - Klassediagrammer (indeholder især de vigtigste model-, kontroller- og view-klasser)
 - Sekvensdiagrammer som viser samspil mellem vigtige klasser/komponenter
- Database design → se DB kursus

→ Især MVC, DAL, ...

- Implementering
 - Hvordan er designet bliver implementeret (men kun nogle interessante udtræk, ikke hele kodden)
 - Dette omfatter software og database
 - Herunder kan I også diskutere evt. mangler af jeres implementering
- Udviklingsproces og test
 - Udviklingsproces og brugte værktøjer
 - Diskussion af JUnit-test (automatisk)
 - Diskussion af acceptance-test (manuelle test, fx use-cases)

- Håndbog
 - Hvordan installerer man selve software
 - Hvordan starter man softwareen
 - Hvordan bruger man software (den eksisterende GUI er I ikke nødt til at diskutere med alle detaljer); evt. med nogle screenshots
- Konklusion
 - Opsamling af hele resultatet
 - Nogle afsluttende bemærkninger

Det skal være muligt
at bruge softwaren
uden anden
information!

- Referencer
 - Litteratur og
 - Websider I har bugt
- Appendiks (evt.)
 - Nogle relevante diagrammer som ikke kunne være med i hoveddelen
 - Komplet ordliste / glossar

Hus at sige: Hvem
der har skrevet hvad!

15 minutter

Den røde tråd: Why, What, How, Demo

- Why: Indledning & Motivation
- What: Hvad har i udviklet fra brugerens synsvinkel (Analyse)
 - Hvilke hoved-usecases og -funktioner omfatter jeres software
 - GUI (hvis der er noget specielt hos jer)
 - Tekniske perspektiv af det: domænemodel (klassediagrammer, tilstandsdiagrammer, aktivitetsdiagrammer)

og måske "what" på meget højt abstraktionsniveau.

Why, what, how:
gerne med nogle
Powerpoint-slides

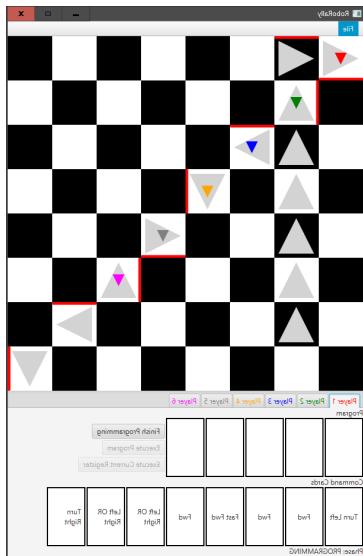
- How: Hvordan er softwaren realiseret

- Arkitektur & Design
- Nogle få implementeringsdetaljer

Især: MVC og DAL

Et kort overblik af jeres IntelliJ-projekt og måske database (MySQL workbench, etc.)

- Demo: Vis hvad jeres software kan



Tip: Gem et spil i en spænende situation, så at I kan vise situationer, som kun sker senere hen i spillet (checkpoint, vinde, skubbe, pit, ...)

Hvert foredrag skal have en indledning og afslutning (afrunding)

I skal tale til

- en med ikke så meget teknisk forstand (CEO), som skal forstå hvad softwaren kan gøre
- og end med mere teknisk forstand (CTO), som I skal overbevise at det I gøre giver mening og I har styr på jeres design

så at begge to får noget ud af det

10-15 minutter

I skulle kunne svare på, hvordan de forskellige emner fra forelæsningen kan findes i jeres analyse, design og implementering.

Det gælder især:

- Domænemodel og diagrammer
- MVC
- Detaljer af jeres
 - controllere: app- og spillelogik
 - DAO / filer
- Exceptions
- Generics

Hver studerende skal
bringe en computer
med softwareprojektet
installeret i IntelliJ, så
at I kan pege på
tingene i
implementeringen i
jeres IDE.

I skal også have
rapporten på jeres
computere, så at vi fx.
kan spørge om jeres
diagrammer i rapporten

Tidsplan

Se <http://www2.compute.dtu.dk/courses/02362/f21/eksamen.shtml>



DTU Compute

Department of Applied Mathematics and Computer Science

Eksamener her

Mandag, 31. maj

8⁰⁰ - 10⁰⁰: Group 1

10¹⁵ - 12¹⁵: Group 2

13⁰⁰ - 15⁰⁰: Group 4

...

Onsdag, 2. juni

8⁰⁰ - 10⁰⁰: Group 3

10¹⁵ - 12¹⁵: Group 5

13⁰⁰ - 15⁰⁰: Group 6

...

The screenshot shows a web browser window with the URL www2.compute.dtu.dk/courses/02362/f21/eksamen.shtml. The page is titled "02362: PROJEKT I SOFTWARE-UDVIKLING (F21)" and has a sub-section "Eksamener". It contains text about the exam format, requirements, and a list of items needed for the exam. Below this, it discusses the use of Zoom and provides information about the exam plan for Monday, May 31st.

02362: PROJEKT I SOFTWARE-UDVIKLING (F21)

Eksamener

Eksaminer til kursus "Projekt i software-udvikling" (02362) finder, som bekendt, sted **mandag, den 31. maj og onsdag, den 2. juni 2021!** Eksaminer foregår på Zoom – linket til Zoom-mødet bliver bekendtgjort via en besked på DTU Learn.

Formatet er at der først ville være en gruppepræsentation på max. 15 minutter med en Powerpoint-præsentation og en live-demo af projektet (hvor alle gruppemedlemmer skal være en aktiv del af). Bagefter er der en eksamination af de enkelte gruppemedlemmer (på 10-15 minutter hvert). Hver studerende skal have gruppens udviklede software (samt databasen) installeret på dens egen computer og skulle kunne køre og vise den i IntelliJ.

Der vil også være lidt mere information om eksamen under sidste forelæsning i uge 13 .

For at gennemføre eksamenen skal hver studerende have installeret

- Zoom, samt et web-kamera og et mikrofon,
- IntelliJ med en kørbart version af gruppens software (som inkluderer også en kørende database) og mulighed for at vise en præsentation (enten Powerpoint eller PDF) og gruppens rapport på dens computer.

Desuden skal hver studerende vise billedelegitimation (studiekort eller pas) på en måde, så at bedømmerne kan se det. Endvidere skal den studerende med sit web-kamera kunne vise rummet vedkommende sidder i, for at bedømmerne kan sikre sig, at den studerende er alene i rummet. Desuden henviser DTU på [DTU's æreskodeks](#) og nogle generelle ting vedr. online-eksamen på DTU Inside [her](#).

Zoom-mødet til eksamenen bliver konfigureret med et "venteværelse", hvor alle studerende af en gruppe skal være logget på imens gruppens eksamen foregår. Alle studerende skal logges på i god tid op til eksamenen. På deres tidspunkter bliver de forskellige studerende så "kaldt ind" i mødelokalet (i starten er det hele gruppen til gruppepræsentation).

For at afprøve og teste opsætningen og udstyret afholdes der gruppemødrene i uge 13 (torsdag, den 4. maj) med samme opsætning og på samme Zoom-mødet, som også eksamenen ville foregå. Tidsplanen til prøveksamen/gruppemødrene bliver bekendtgjort på websiden med materiale og informationerne om uge 13 på kursets websider.

Tidsplanen til eksaminer til de forskellige grupper er:

Mandag, den 31. maj 2021:

- 8⁰⁰ - ca. 10⁰⁰: Gruppe 1
- 10¹⁵ - ca. 12¹⁵: Gruppe 2
- 13⁰⁰ - ca. 15⁰⁰: Gruppe 4
- 15¹⁵: Evtl. re-eksamen

XI. Afslutning

DTU Compute

Department of Applied Mathematics and Computer Science

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

$\Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$

$\epsilon^b \Theta = \{2.71828182845904523536028747135266249$

$\infty = \sum \gg,$

χ^2

Indtil nu (1. semester):

- **Kendskab** til nogle programmerkonstrukter
- Basale **færdigheder** i programmering

Efter dette kursus

- **Programmererfaring** med et lidt større projekt
- **Kendskab** til nogle yderlige programmerkonstrukter
- **Erfaring** med god programmeringsstil
- **Erfaring** med at debugge projekter
- **Lidt erfaring** med databaser

- (Java) Interfaces
- Datatyper
 - egne
 - brug af indbyggede datatyper i Java
- Rekursion
- Exceptions
- Generics
- Tests

- Modellering (domæne og designmodeller)
- God stil og skik i programmering
- Java docs
- Design pattern
- Model-View-Controller-princippet (MVC)
- Brugerdialog og inputvalidering
(regulære udtryk)
- Filer
- Threads
- Tilknytning til database
(→ 02327 og forelæsning uge 6, 7 og 10)

Motivation

- Implementeringer i programmer er ofte meget teknisk
- Der er mange dataer som kan ødelægges, når man ikke bruger dem rigtigt
- Man ved ikke hvad der er vigtigt og ikke så vigtigt

- Programmet er svært at forstå
- Programmet kan nemt ødelægges

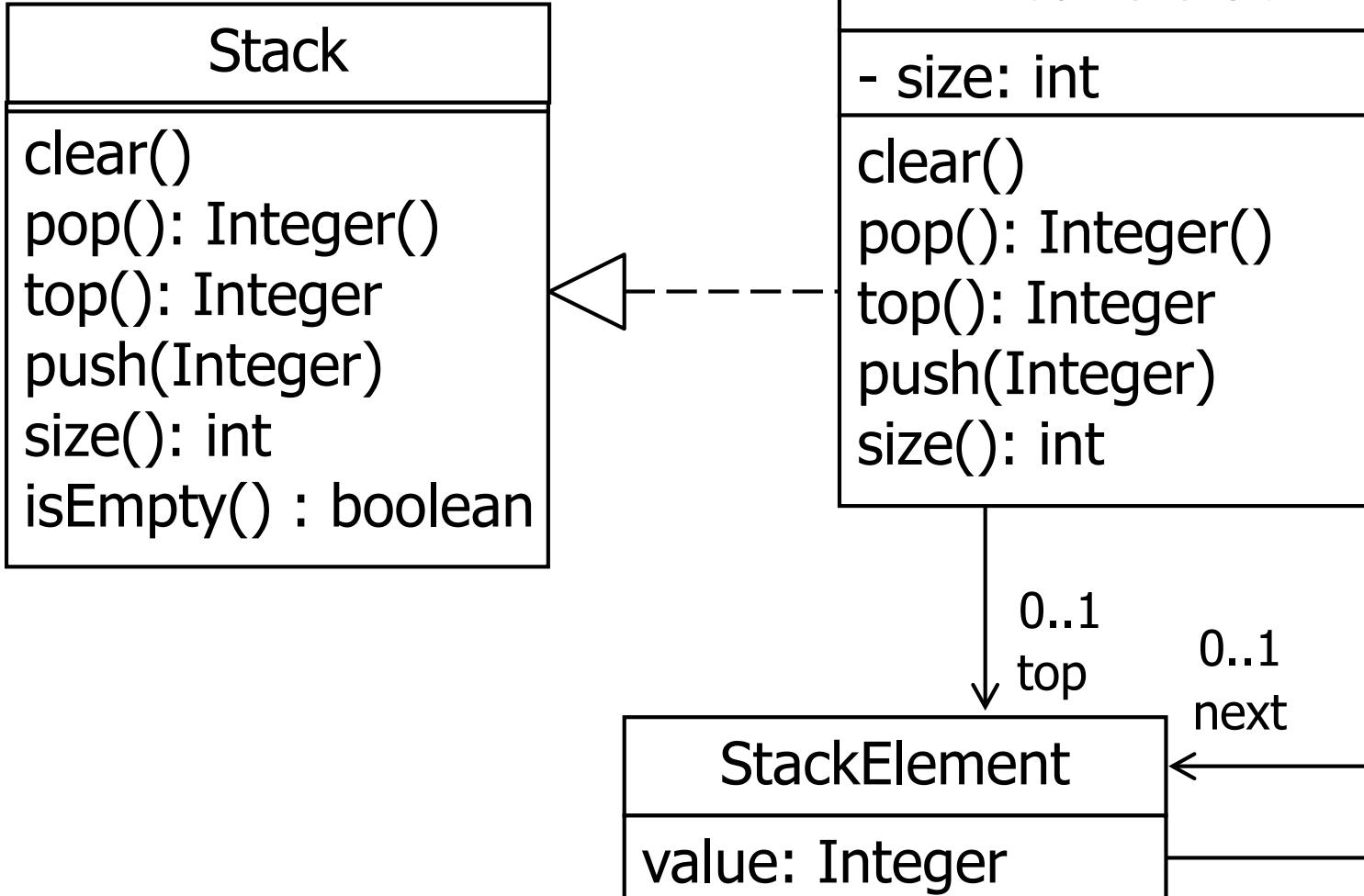
```
public interface Stack {  
  
    void clear();  
  
    Integer pop();  
  
    Integer top();  
  
    void push(Integer value);  
  
    int size();  
  
    default boolean isEmpty() {  
        return size() == 0;  
    }  
}
```

Implementering (eks.)

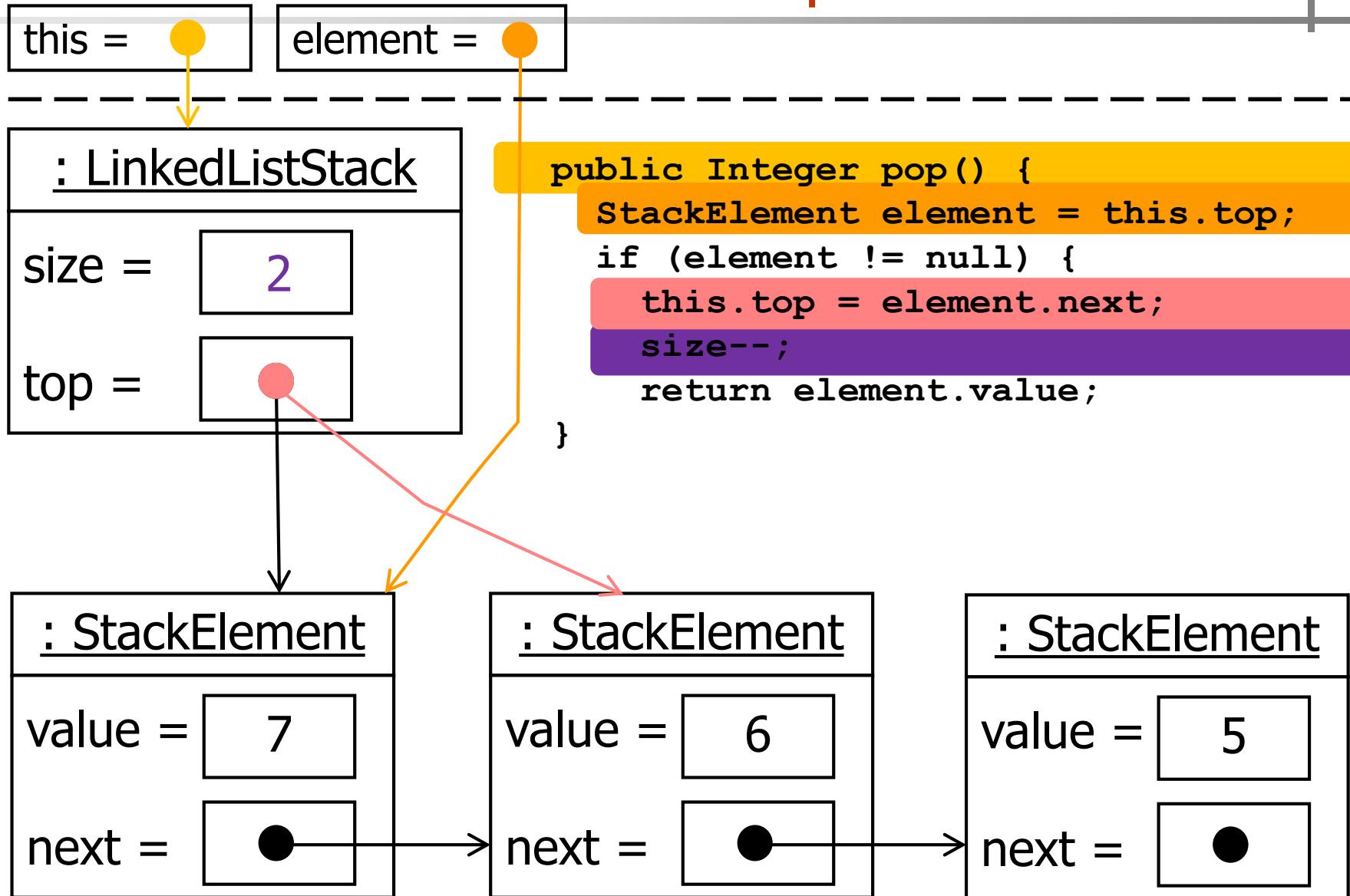
```
public class LinkedListStack implements Stack {  
  
    @Override  
    public void clear() {  
        ...  
    }  
  
    @Override  
    public Integer pop() {  
        ...;  
    }  
    ...  
}
```

Implementeringer

- Som enkelt-hægtede liste



pop()



ws - Java - dk.dtu.compute.se.pisd.stack1/src/main/dk/dtu/compute/se/pisd/stack/LinkedListStack.java - Eclipse

File Edit Source Refactor Navigate Project Run Window Help

Quick Access

Package Explorer

- dk.dtu.compute.se.pisd.list1
- dk.dtu.compute.se.pisd.recursion
- dk.dtu.compute.se.pisd.stack1
 - src/main
 - dk.dtu.compute.se.pisd.stack
 - ArrayStack.java
 - LinkedListStack.java
 - StackElement
 - size
 - top
 - clear(): void
 - pop(): Integer
 - push(Integer): void
 - size(): int
 - top(): Integer
 - Stack.java
 - package.html
 - overview.html
 - src/test
 - JRE System Library [JavaSE-1.8]
 - JUnit 4
 - bin
 - src

Stack.java

```

1 package dk.dtu.compute.se.pisd.stack;
2
3
4 /**
5 * A stack of {@see java.lang.Integer} values. An arbitrary number
6 * of values can be added (pushed) to the stack. The values can be obtained
7 * (popped) from the stack in a last-in-first-out fashion. Note that we use
8 * the class {@see java.lang.Integer} here instead of a data type
9 * for two reasons: first, we will extend this class later for defining
10 * stacks for other types than integers; second, using the class
11 * {@see java.lang.Integer} allows us to return null as a result,
12 * in case of errors (note that this will be improved later by using
13 * exception handling).
14 */
15
16 /**
17 * @author Ekkart Kindler, eKKi@dtu.dk
18 */
19 public interface Stack {
20
21     /**
22     * Removes all elements from the stack. The stack will be empty after the
23     * call returns.
24     */
25     void clear();
26
27     /**
28     * Removes the top element from the stack and returns the value of that
29     * element. If the stack is empty, the stack is not changed, and the
30     * call returns <code>null</code>.
31
32     * @return the value of the top element of the stack
33 }

```

LinkedListStack.java

```

1 package dk.dtu.compute.se.pisd.stack;
2
3 /**
4 * Implements a (singly) linked list.
5 *
6 * @author Ekkart Kindler, eKKi@dtu.dk
7 */
8
9
10 public class LinkedListStack implements Stack {
11
12     private StackElement top = null;
13
14     private int size = 0;
15
16     @Override
17     public void clear() {
18         top = null;
19         size = 0;
20     }
21
22     @Override
23     public Integer pop() {
24         if (top != null) {
25             StackElement element = top;
26             top = element.next;
27             size--;
28             return element.value;
29         }
30     }

```

Outline

- dk.dtu.compute.se.pisd.stack
 - LinkedListStack
 - top: StackElement
 - size: int
 - clear(): void
 - pop(): Integer
 - top(): Integer
 - push(Integer): void
 - size(): int
 - StackElement

Problems @ Javadoc Declaration Search Console Progress

<terminated> Factorial [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (13. feb. 2018 09:41:00)

Writable Smart Insert 1:1

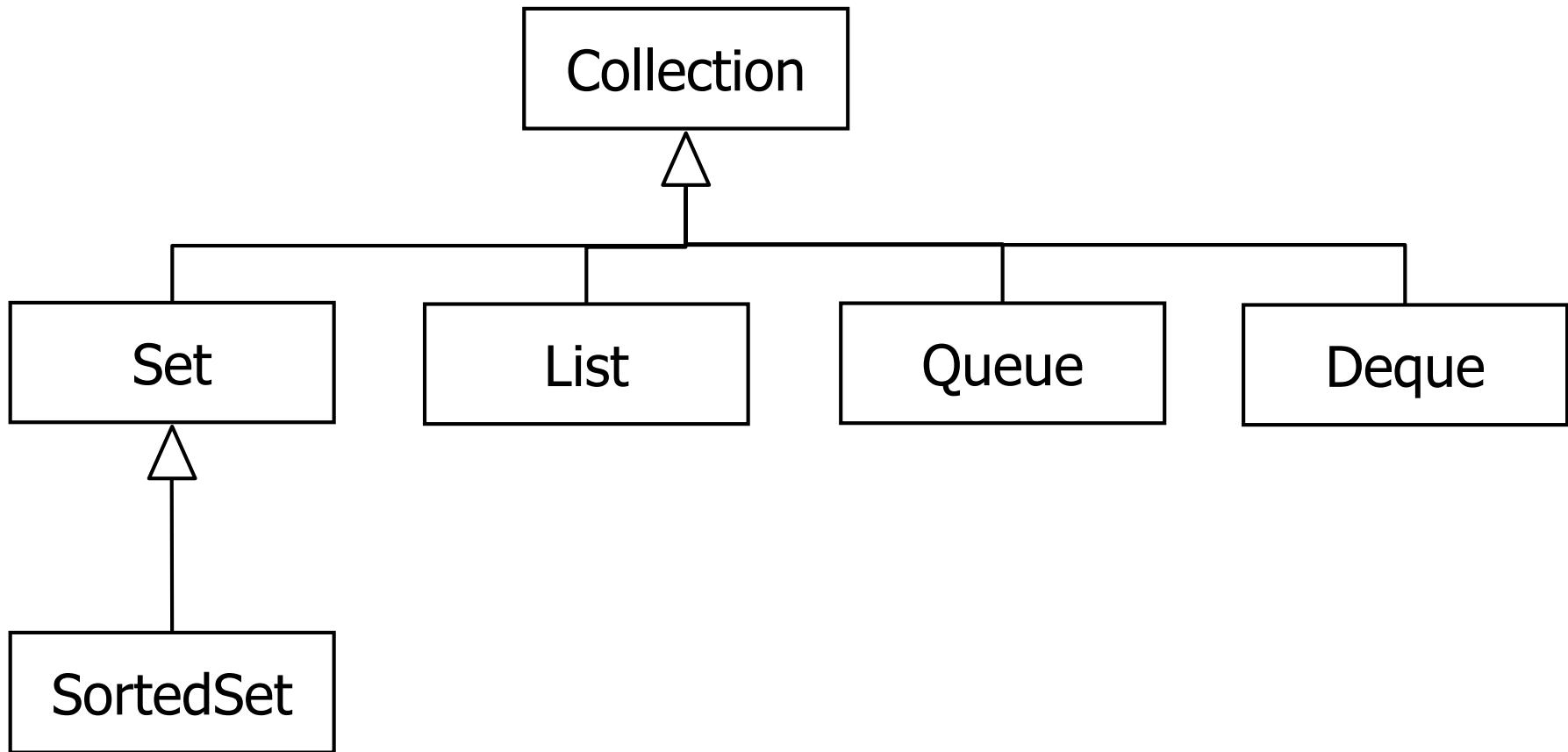
Javas indbyggede datastrukturer

DTU Compute

Department of Applied Mathematics and Computer Science

A collage of mathematical symbols including integrals, summation, infinity, and various Greek letters like Delta, Epsilon, Theta, Omega, and Chi.

Collections overview



Comparable

```
public interface Comparable<T> {  
  
    public int compareTo(T o);  
  
}
```

o1.compareTo(o2) < 0 hvis object o1 er mindre end object o2
o1.compareTo(o2) == 0 hvis object o1 og object o2 er lige
o1.compareTo(o2) > 0 hvis object o1 er større end object o2

Lister, Sortering, og Søgning

DTU Compute

Department of Applied Mathematics and Computer Science

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

$\Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$

$\epsilon^b \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$

$\infty = \{2.71828182845904523536028747135266249$

$\chi^2 \Sigma^{\gg},$

$\sum!$

Lidt mere systematisk:

Vi antager at listen ligger i et array

int[] list = ...

og

int size = ...

er den aktuelle længde af listen

Bubble sort (1st version)

```
boolean swapped;  
do {  
    swapped = false;  
    for (int i=0; i+1<size; i++) {  
        if (values[i] > values[i+1]) {  
            int v = values[i];  
            values[i] = values[i+1];  
            values[i+1] = v;  
            swapped = true;  
        }  
    }  
} while (swapped);
```

Byt systematisk, så længe der er elementer ved siden af hinanden, som ikke er sorteret.

Første iteration

9	8	1	6	7	4	5	2	3
---	---	---	---	---	---	---	---	---



8	9	1	6	7	4	5	2	3
---	---	---	---	---	---	---	---	---



8	1	9	6	7	4	5	2	3
---	---	---	---	---	---	---	---	---



8	1	6	9	7	4	5	2	3
---	---	---	---	---	---	---	---	---



...

8	1	6	7	4	5	2	3	9
---	---	---	---	---	---	---	---	---

Anden iteration

8	1	6	7	4	5	2	3	9
---	---	---	---	---	---	---	---	---



1	8	6	7	4	5	2	3	9
---	---	---	---	---	---	---	---	---



1	6	8	7	4	5	2	3	9
---	---	---	---	---	---	---	---	---



...

1	6	7	4	5	2	3	8	9
---	---	---	---	---	---	---	---	---

Tredje iteration

1	6	7	4	5	2	3	8	9
---	---	---	---	---	---	---	---	---



1	6	7	4	5	2	3	8	9
---	---	---	---	---	---	---	---	---



1	6	7	4	5	2	3	8	9
---	---	---	---	---	---	---	---	---



1	6	4	7	5	2	3	8	9
---	---	---	---	---	---	---	---	---



1	6	4	5	2	3	7	8	9
---	---	---	---	---	---	---	---	---

I hver iteration kan vi stoppe en position tidligere!

Rekursion

DTU Compute

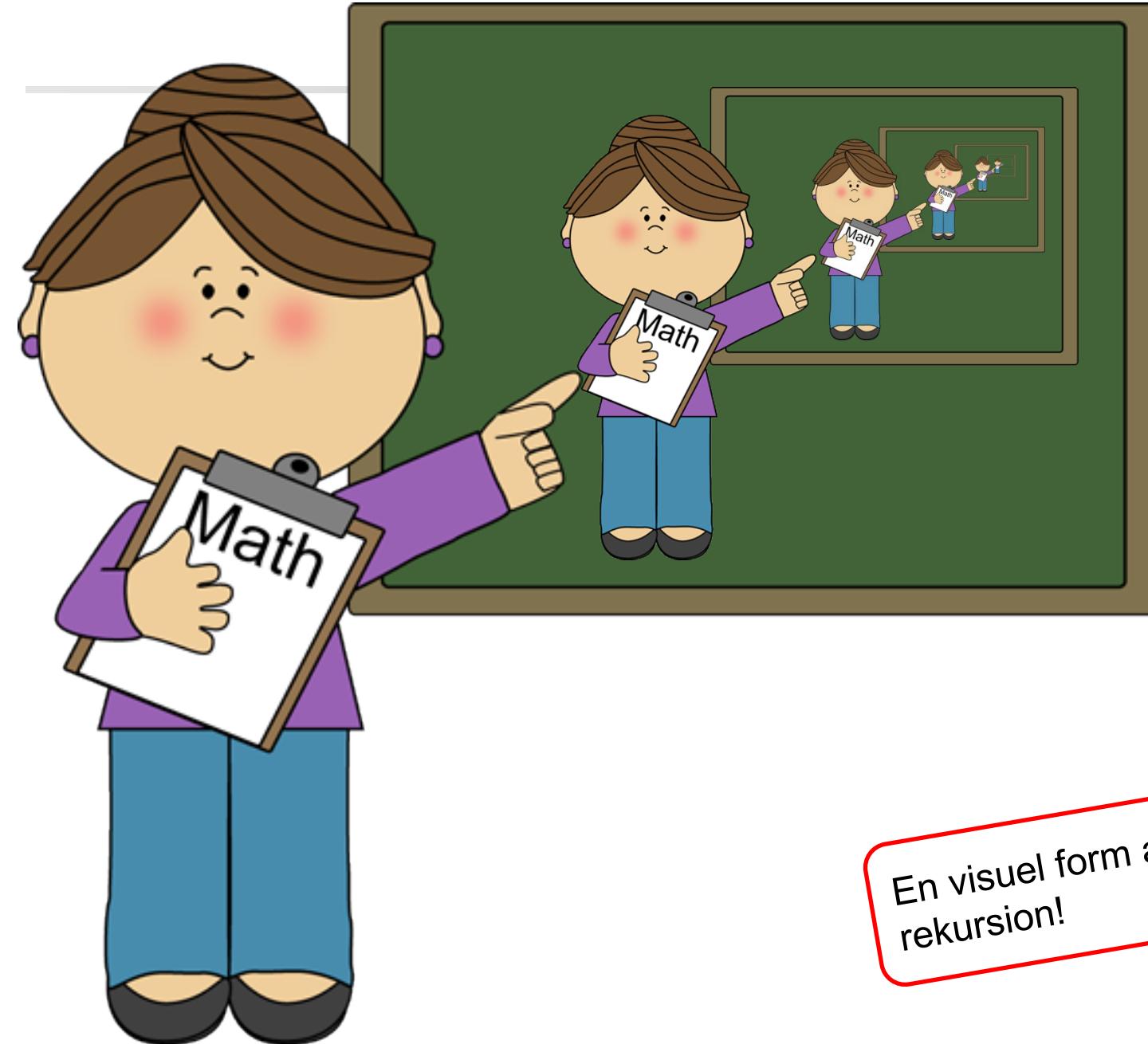
Department of Applied Mathematics and Computer Science

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

$\Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$

$\epsilon^b - \infty = \{2.71828182845904523536028747135266249$

Σ^{\gg}, χ^2



En visuel form af en
rekursion!

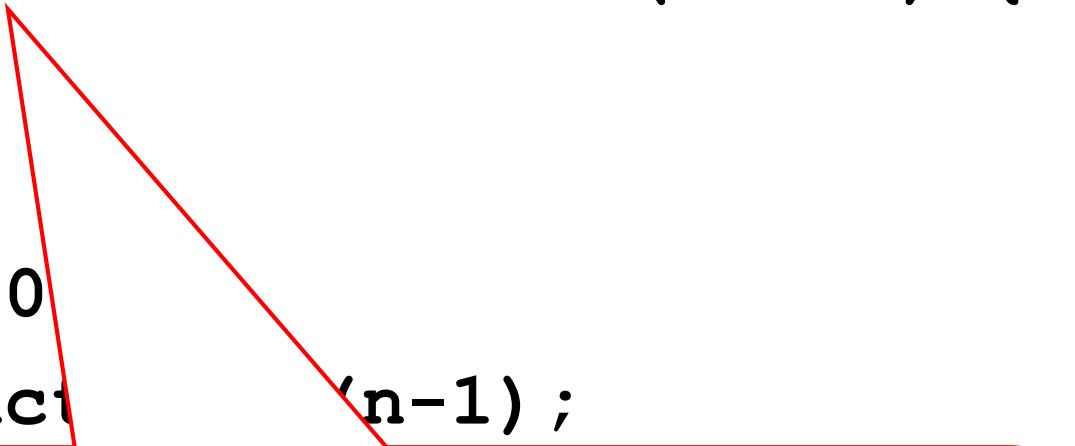
Rekursiv definition $n!$

$$n! = \begin{cases} 1 & \text{hvis } n=0 \\ n * (n-1)! & \text{hvis } n>0 \end{cases}$$

Her er der **rekursion** igen

```
private static int factorial(int n) {  
    if (n == 0) {  
        return 1;  
    } else if (n > 0) {  
        return n * factorial(n-1);  
    } else {  
        throw IllegalArgumentException(  
            "f(" + n + ")" );  
    }  
}
```

```
private static double factorial(int n) {  
    if (n == 0) {  
        return 1;  
    } else if (n > 0)  
        return n * fact  
    } else {  
        throw Ille  
    "f(" + n +  
    }  
}
```



Nogle detaljer om **primitive datatyper** i Java:
int og **long**: når tal bliver for store/små, bliver de forkerte (uden advarsel)!

Det starter allerede, at gå galt med **int** på **factorial(14)**!

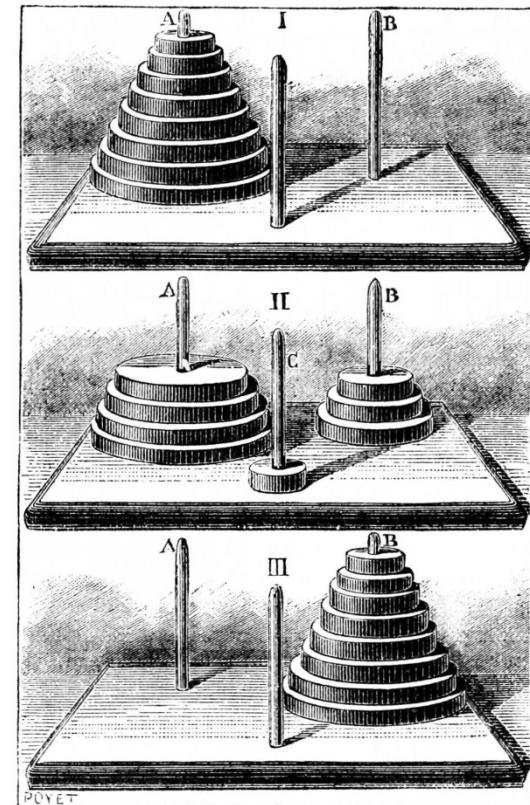
Når man bruger **double** og **float** eksisterer værdier som **Infinity** og **NaN** (Not a Number), som viser, at resultatet er for stort eller forkert.

Andet Eksempel

Engl. Towers of Hanoi

Hanois tårne: Matematisk "puslespil":

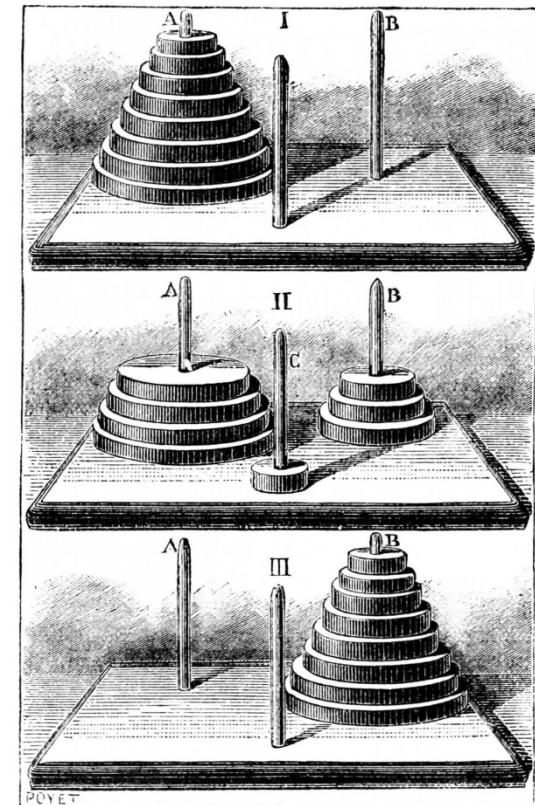
- Stativ med tre pinde (A, B og C)
- På pind A er der n skiver med aftagende størrelse (set nedfra)
- **Opgave:**
Flyt alle n skiver fra A til B, men
 - Kun en skive ad gangen
 - En større skive må aldrig ligge på en skive som er mindre



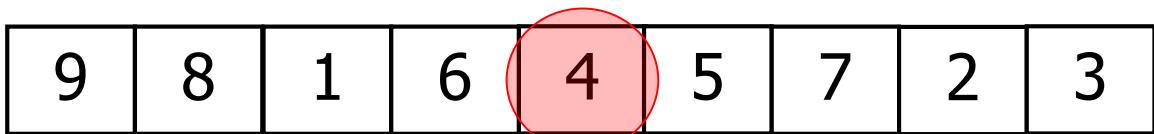
Flyt n skiver fra A til B

1. Flyt de øverste $n-1$ skiver fra A til C
(iføge reglerne)
2. Så fly den nedereste (største) skive n fra A til B
(som er OK ifølge reglerne)
3. Nu flyt de $n-1$ skiver fra C til B

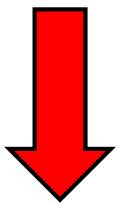
Sammen med rekursion er det løsningen fordi 1. og 3. er samme problemstilling bare for $(n-1)$ skiver



Quicksort (→ Hoare 1959)



Vælg et tilfældigt **pivot** element



Partitioner arrayet, så at alle tal lavere eller lige pivot elementet ligger på venstre side og alle større eller lige på højre side



Sorter partitionerne uafhængigt af hinanden **rekursiv**



Så er hele arrayet sorteret

Quicksort

```
private void quicksort(int lower, int upper) {  
    if (lower >= upper) {  
        return;  
    }  
  
    int partition = partition(lower, upper);  
    quicksort(lower, partition);  
    quicksort(partition+1, upper);  
}
```

Bemærk at det er
vigtigt at begge
partitioner bliver
mindre! Begge skal
have mindst et
element

API Design

Indeholder også tekniske
emner som generiske typer
og exceptions!

DTU Compute

Department of Applied Mathematics and Computer Science

A vibrant collage of mathematical symbols and numbers. It features a large purple integral symbol with a yellow boundary, a red summation symbol with a red exclamation mark, a blue infinity symbol with an orange arrow, a green theta symbol with a purple circle, a pink delta symbol with a red 'i' and 'pi', and a blue omega symbol with a blue circle. The background is white with faint, semi-transparent mathematical drawings like a coordinate system and a bell curve.

Eks.: Stak (interface)

```
public interface Stack<E> {  
  
    void clear();  
  
    E pop();  
  
    E top();  
  
    void push(E value);  
  
    ...  
  
    int size();  
  
}
```

E er en parameter, som er den type stakken skal senere have: generisk datatype

Eks.: Stak (interface)

```
public interface Stack<E> {  
  
    void clear();  
  
    E pop() throws IllegalStateException;  
  
    E top() throws IllegalStateException;  
  
    void push(E value) throws IllegalArgumentException;  
  
    ...  
  
    int size();  
  
}
```

Eks.: Stak (implement.)

```
public class LinkedListStack<E> implements Stack<E> {  
  
    ...  
  
    public E pop() throws IllegalStateException {  
        if (top != null) {  
            StackElement element = top;  
            top = element.next;  
            size--;  
            return element.value;  
        }  
        throw new IllegalStateException();  
    }  
    ...
```

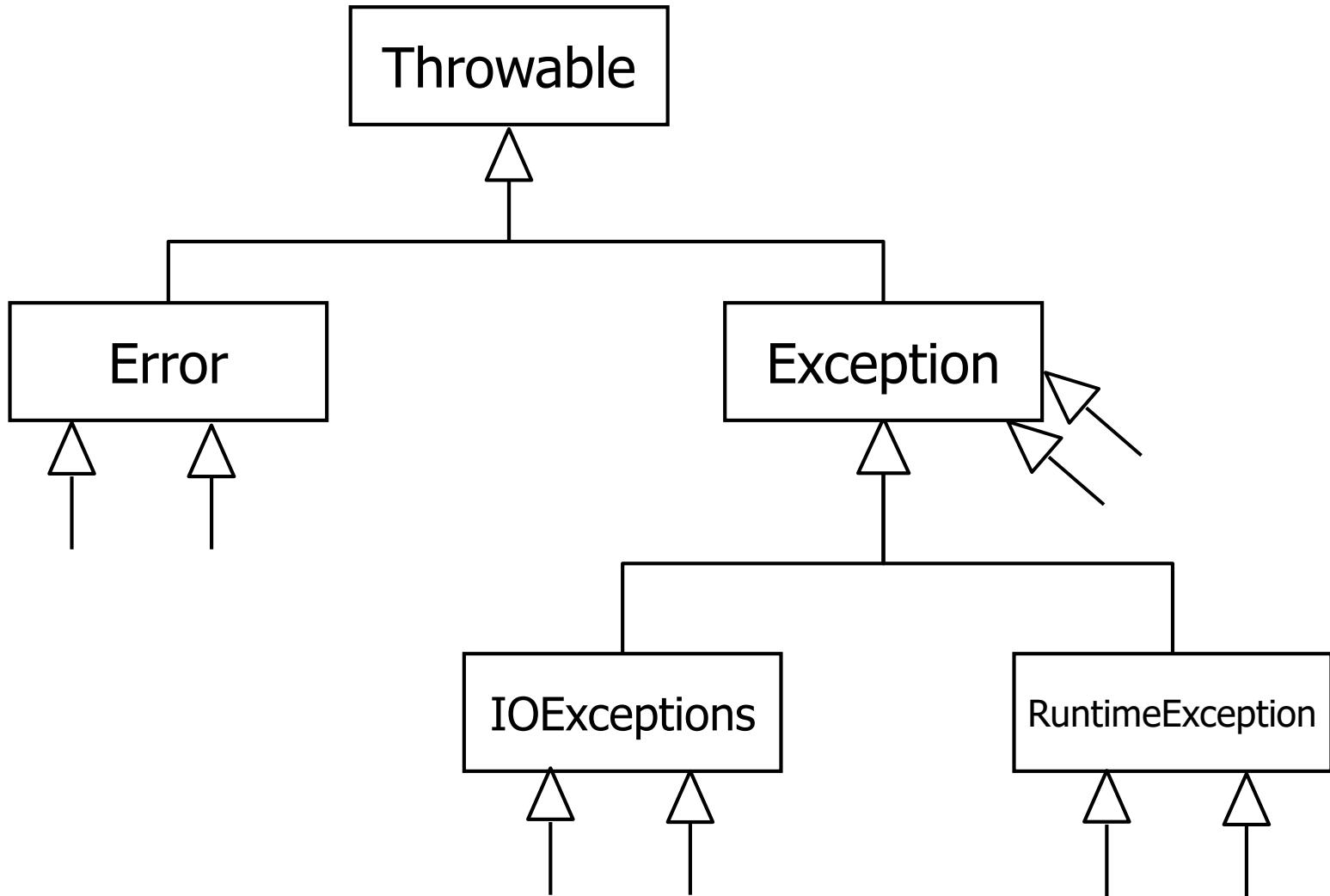
throw betyder at en exception bliver kastet her!

Eks.: Stak (implement.)

```
public void push(E value)
    throws IllegalStateException {
    StackElement newElement =
        new StackElement(value, top);
    size++;
    top = newElement;
}
```

Implementeringen af `push()`
kaster ikke en exception, men `new`
`StackElement()` gør; og da
`push()` ikke fanger den, ryger den
videre op til kalderen af `push()`.

Exception: Typehierarki



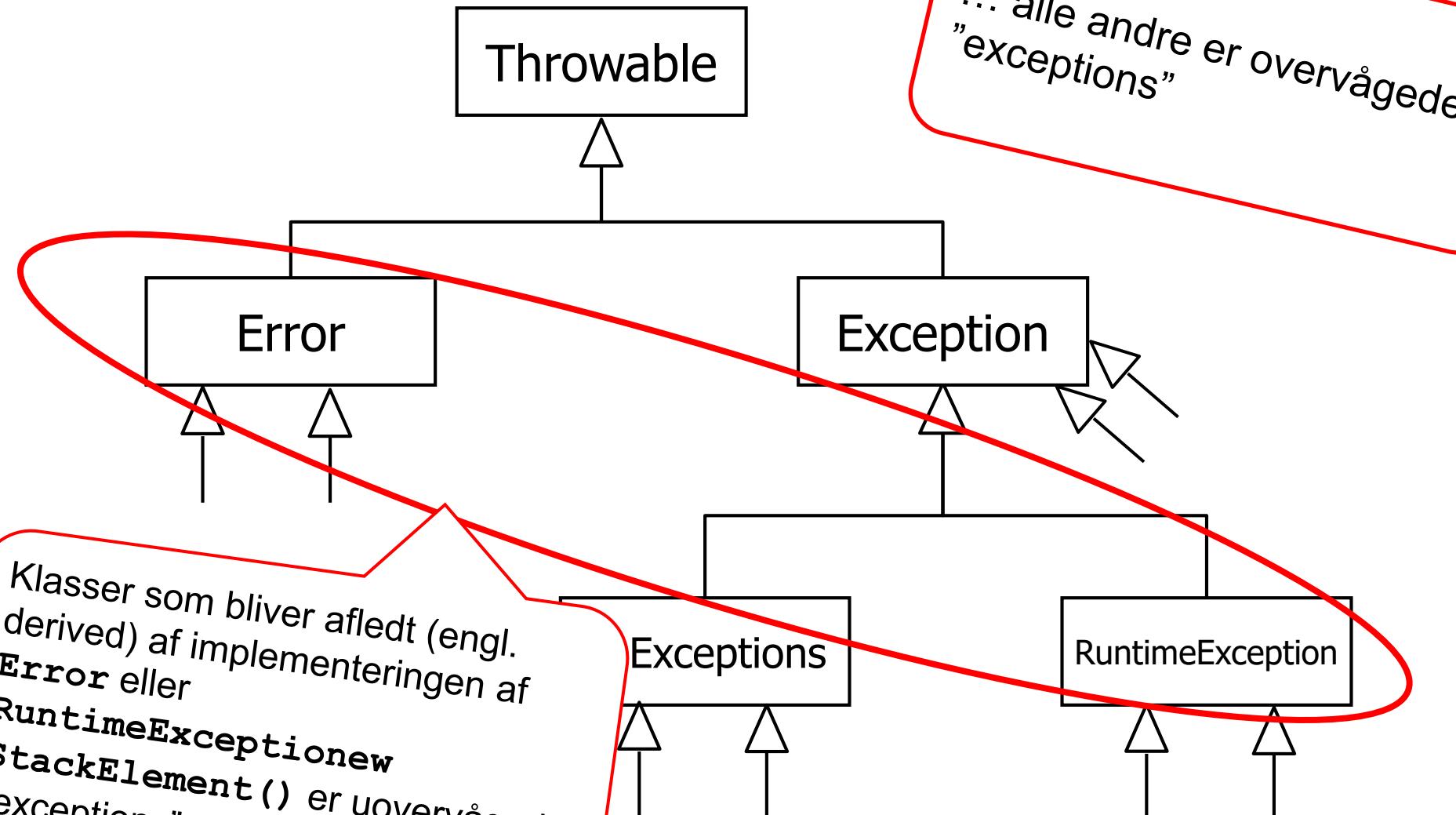
- er exceptions, hvor den, som kalder metoden (*engl. caller*), kan gøre noget ved

typisk: forkert brug af metoden (kalde med de rigtige parameter), IOException (prøve igen),

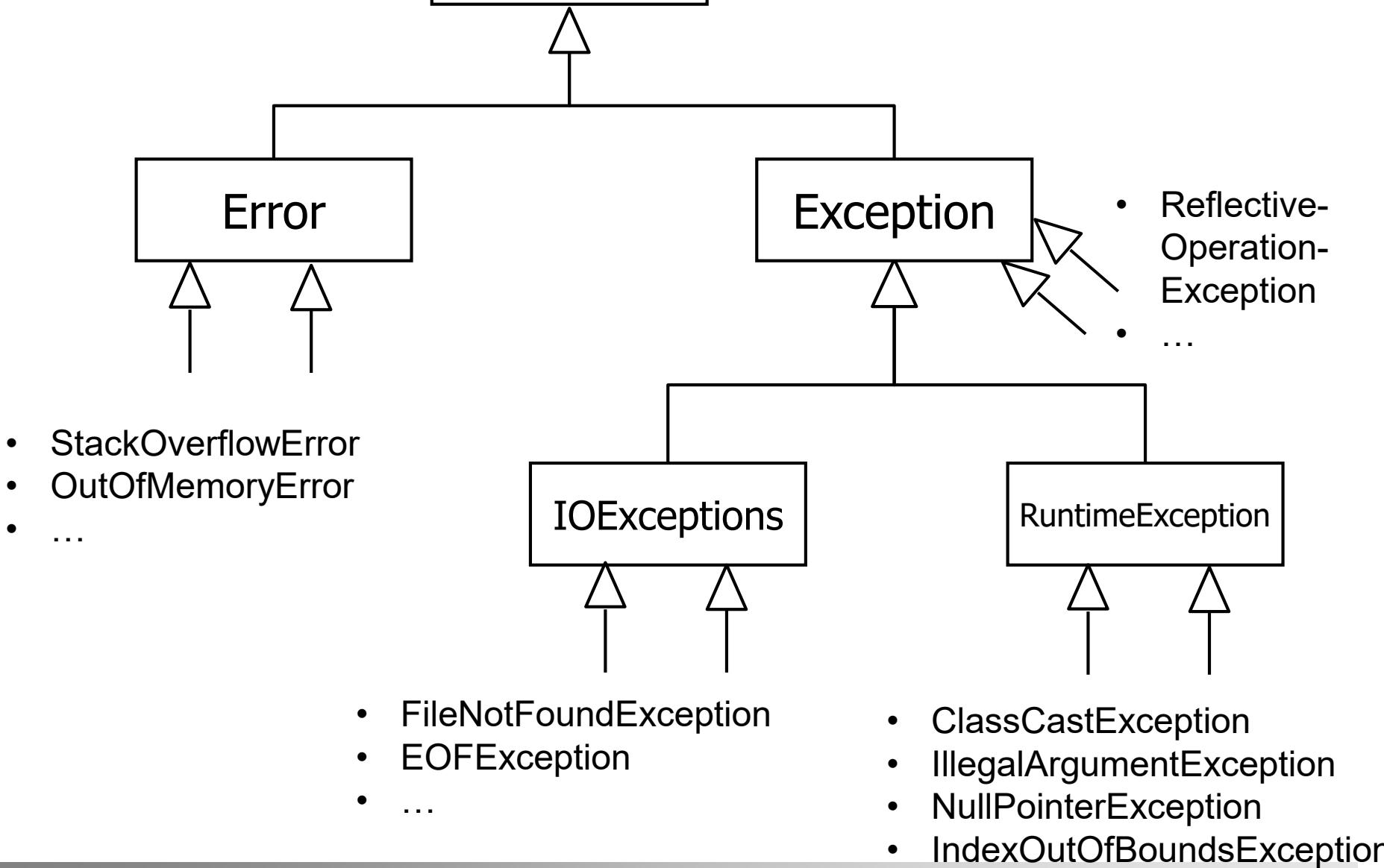
- skal specificeres I metodens deklaration eller fanges I metodens implementering

- er exceptions, hvor den, som kalder metoden (*engl. caller*), ikke kan gøre noget ved typisk: programmeringsfejl i implementering (NullPointerException, ClassCastException, ArrayIndexOutOfBoundsException, StackOverflowError, IOError, ...)
- er man ikke nødt til at specificere i metodens deklaration, selvom de bliver kastet eller ikke fanget i metodens implementering
- ryger typisk op til main-tråden og stopper hele programmet, når de optræder

Exception: Typehierarki



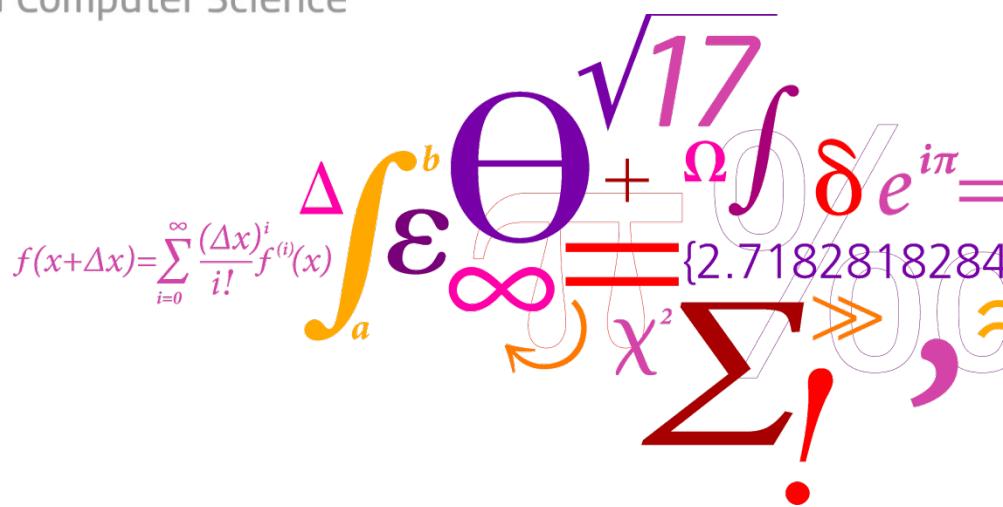
Exception: Typo schlimm



Analyse og konceptuelle modeller

DTU Compute

Department of Applied Mathematics and Computer Science

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


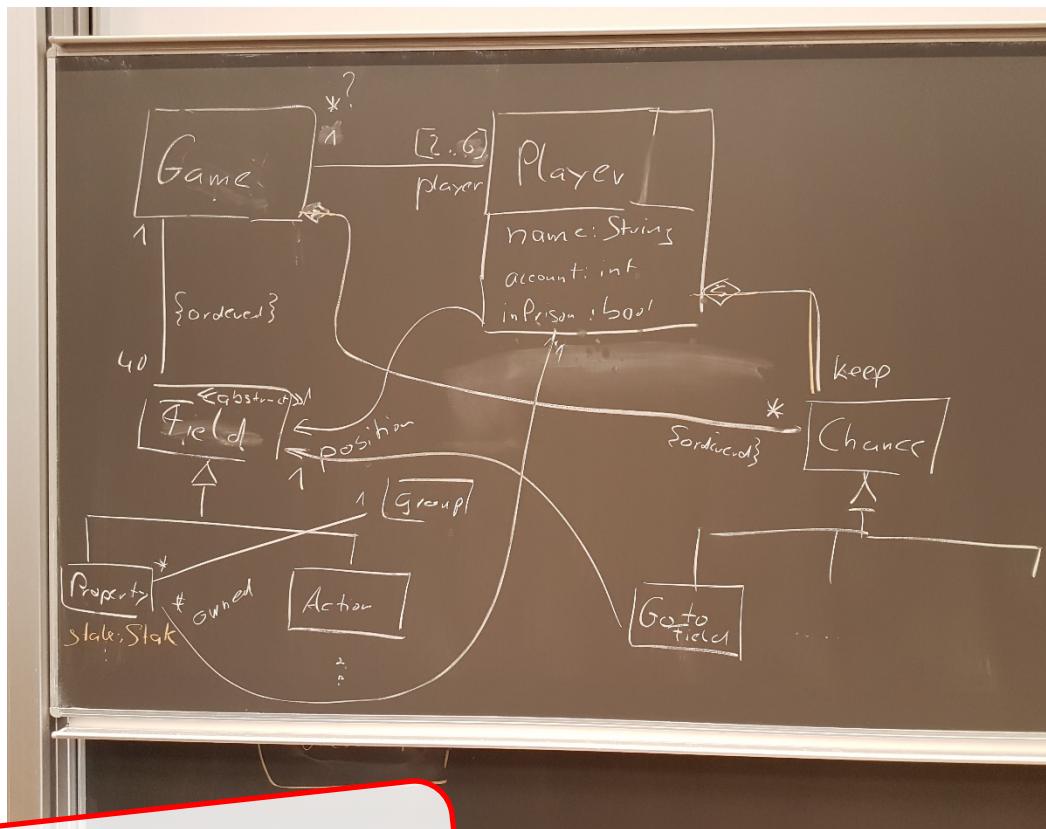
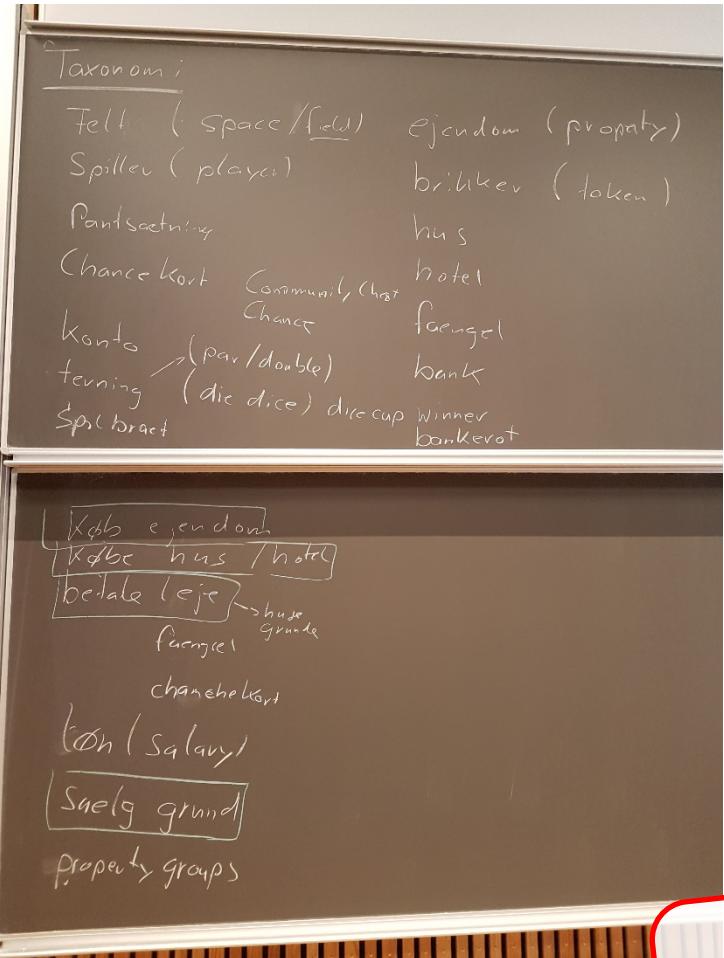
Inden man kan implementere et stykke software er man nødt til af finde ud af **hvad** selve software skal gøre og formulere det fra brugerens synsvinkel:

- 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?

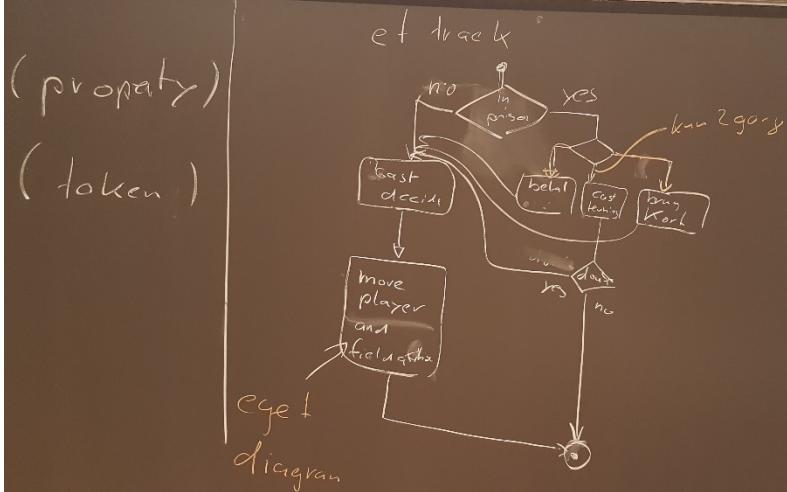
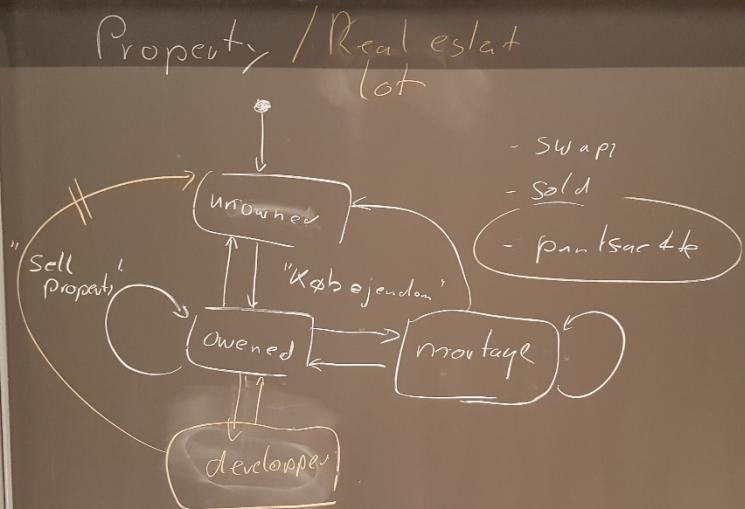
I vores projekt er det mest af alt: RoboRally-spillets regler

- 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

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



Fra 2019:
Monopoly/Matador!



Fra 2019:
Monopoly/Matador!

(Software) Design

Bare for at være sikker på at vi ikke taler om GUI layout design.

DTU Compute

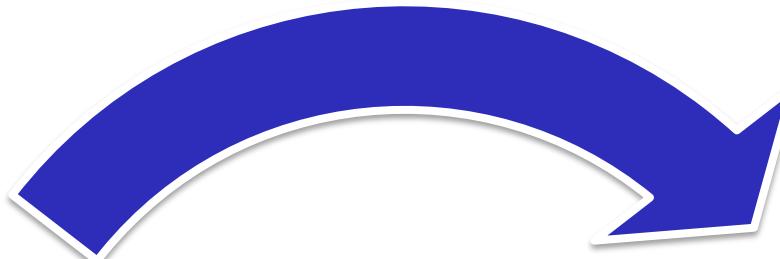
Department of Applied Mathematics and Computer Science

$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$
$$\Theta^{\sqrt{17}} + \Omega \int_a^b \delta e^{i\pi} =$$
$$\Sigma! \gg \chi^2 = \{2.71828182845904523536028747135266249\}$$

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

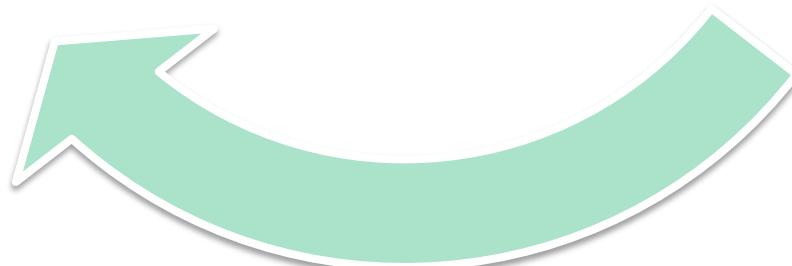
Analysen siger **ikke noget** om
HVORDAN (engl. **HOW**) softwaren skal
realiseres og implementeres

“Hvad” skal
softwaren
gøre?



WHAT

HOW

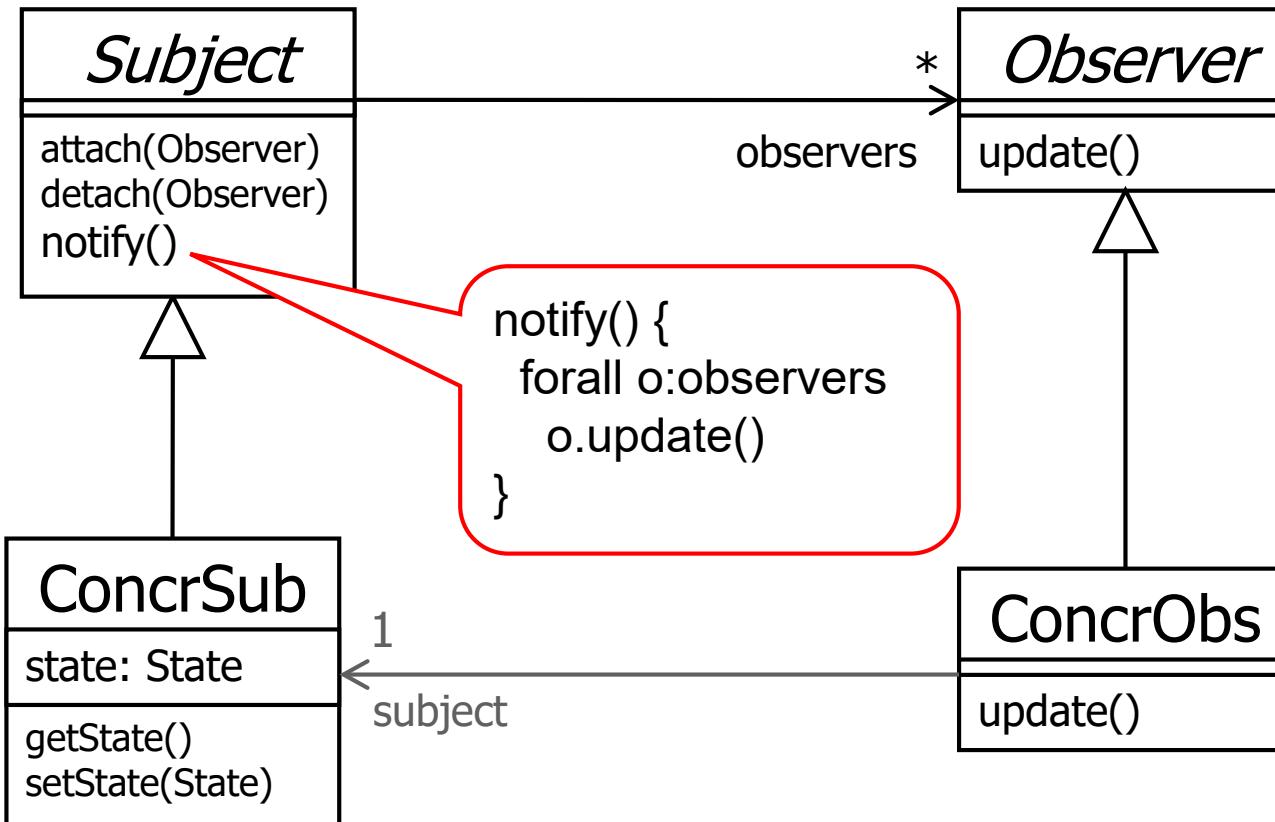


“Hvordan” er
softwaren
realiseret?

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

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

Structure

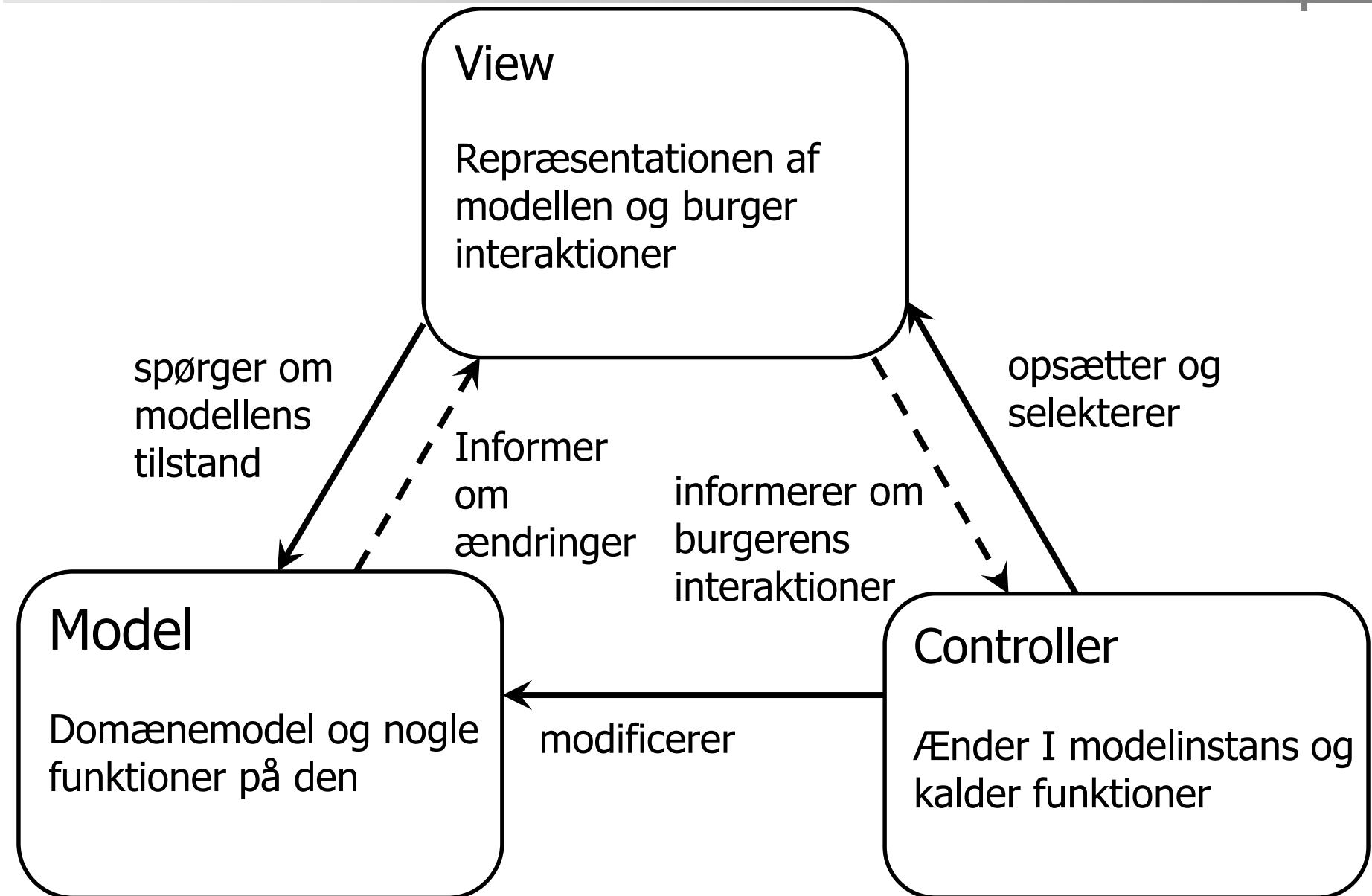


Hvis man gøre det rigtigt er **domæne modeller (model)** og deres implementeringer en fundamental del af den udviklede software

Men der ville være yderlige dele software:

- Delen som viser modellens information til brugeren (**view**)
- Delen som koordinerer med brugeren og implementerer logikken bagved (**controller**)

Bemærk: Disse dele af softwaren kan også modelleres, men de skal ikke forveksles med model forstand af domæne modellen (**hvad**). De kaldes for “software modeller” (**hvordan**) eller ”design modeller”.



Læse og skrive filer, Gson

DTU Compute

Department of Applied Mathematics and Computer Science

Uge 7, video PiSU 07.A - 07.C

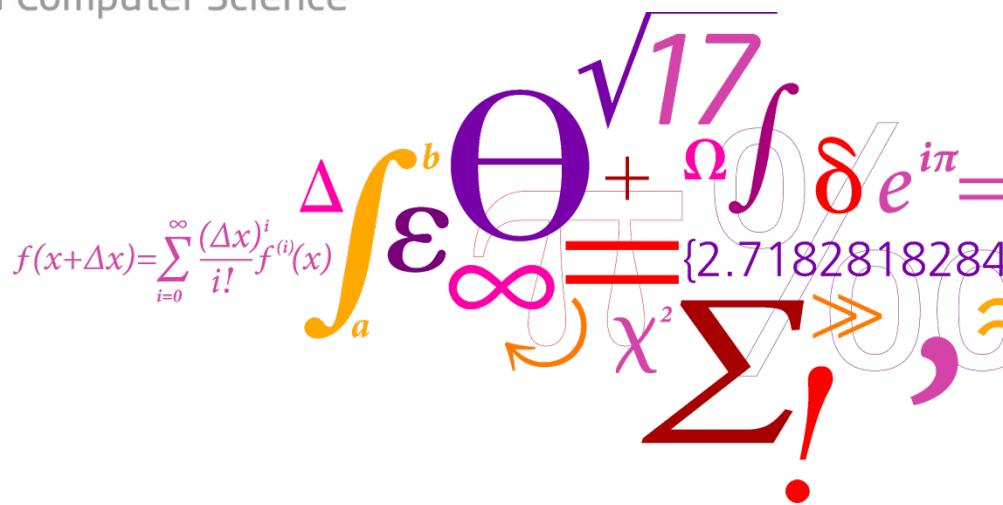
$$f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$$

$$\Theta^{\sqrt{17}} + \Omega \int_a^b \delta e^{i\pi} = \{2.71828182845904523536028747135266249 \dots\}$$
$$\Sigma!$$

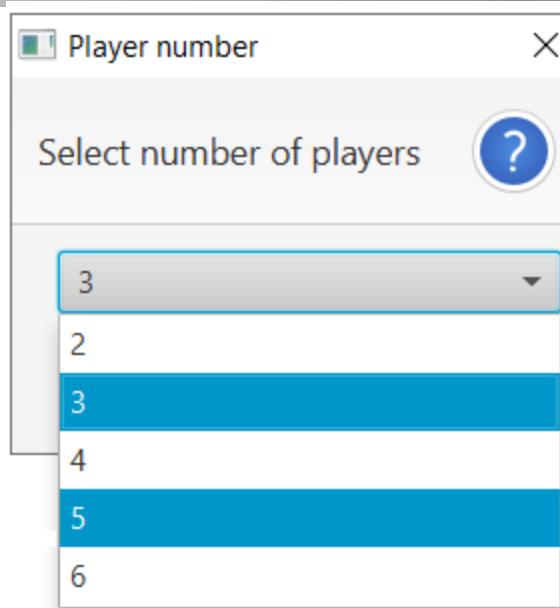
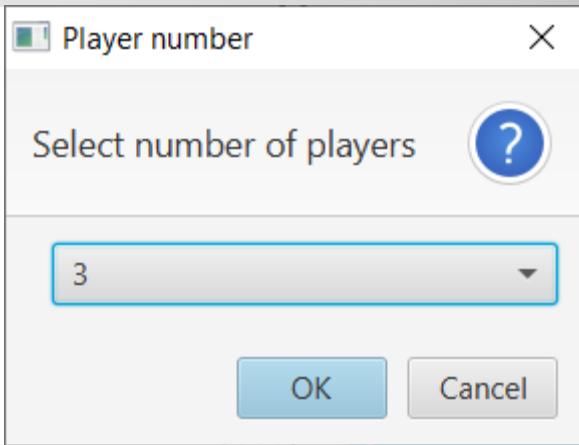
Dialoger og inputvalidering

DTU Compute

Department of Applied Mathematics and Computer Science



Eksempel: ChoiceDialog



- Bemærk at dette er en modal dialog, dvs. brugeren kan først interagere med resten af GULen (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!

I nogle situationer vil man sikre sig, at brugeren indtaster informationer, som overholder en hvis struktur:

- Adresser
- Email-addresser
- Telefonnumre
- Datoer

Regulære udtryk er et meget gammelt koncept i datalogi – meget ældre en Java. Men her bruger vi især notationen fra Java (*, +, ?, | er i generel brug også i andre kontekster)
→ Uge 8 video PiSU L08.A og L08.B

Mange af disse strukturer kan defineres med såkaldte **regulære udtryk**

DAL & DAO (V4/A4)

DTU Compute

Department of Applied Mathematics and Computer Science

A collage of mathematical symbols including integrals, summations, and Greek letters in various colors.

```
public interface IRepository {  
  
    boolean createGame(Board game);  
  
    boolean updateGame(Board game);  
  
    Board loadGame(int id);  
  
    List<GameInDB> getGameIds();  
  
}
```

- Prepared statements

→ Uge 6 video PiSU L06.5 og L06.6

- Brug af opdaterbare ResultSets

- Transaktioner:

- `connection.setAutoCommit(false)`
- `connection.commit() // explicit commit`
- `connection.rollback() // explicit rollback`