

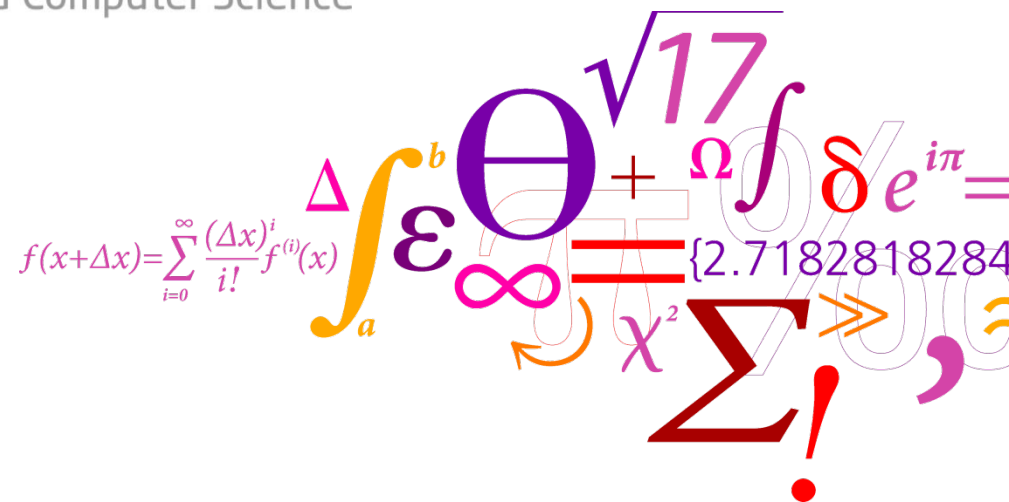
# Projekt i software-udvikling (02362)

## Forår 2022

Ekkart Kindler

DTU Compute

Department of Applied Mathematics and Computer Science



# XI. Afslutning

**Bemærk:** Der er ingen nye slides (nogle har nye kommentarer)!

DTU Compute

Department of Applied Mathematics and Computer Science

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

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

$$\infty \chi^2 \sum!$$

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

- Modelling (domæne og designmodeller)
- God stil og skik i programmering
- Java docs
- Design pattern (→ observer)
- Model-View-Controller-princippet (MVC)
- Brugerdialog og inputvalidering (regulære udtryk)
- Filer (især JSON/Gson)
- Threads
- Tilknytning til database (→ 02327 og 02362 uge 6, 10 og 11)

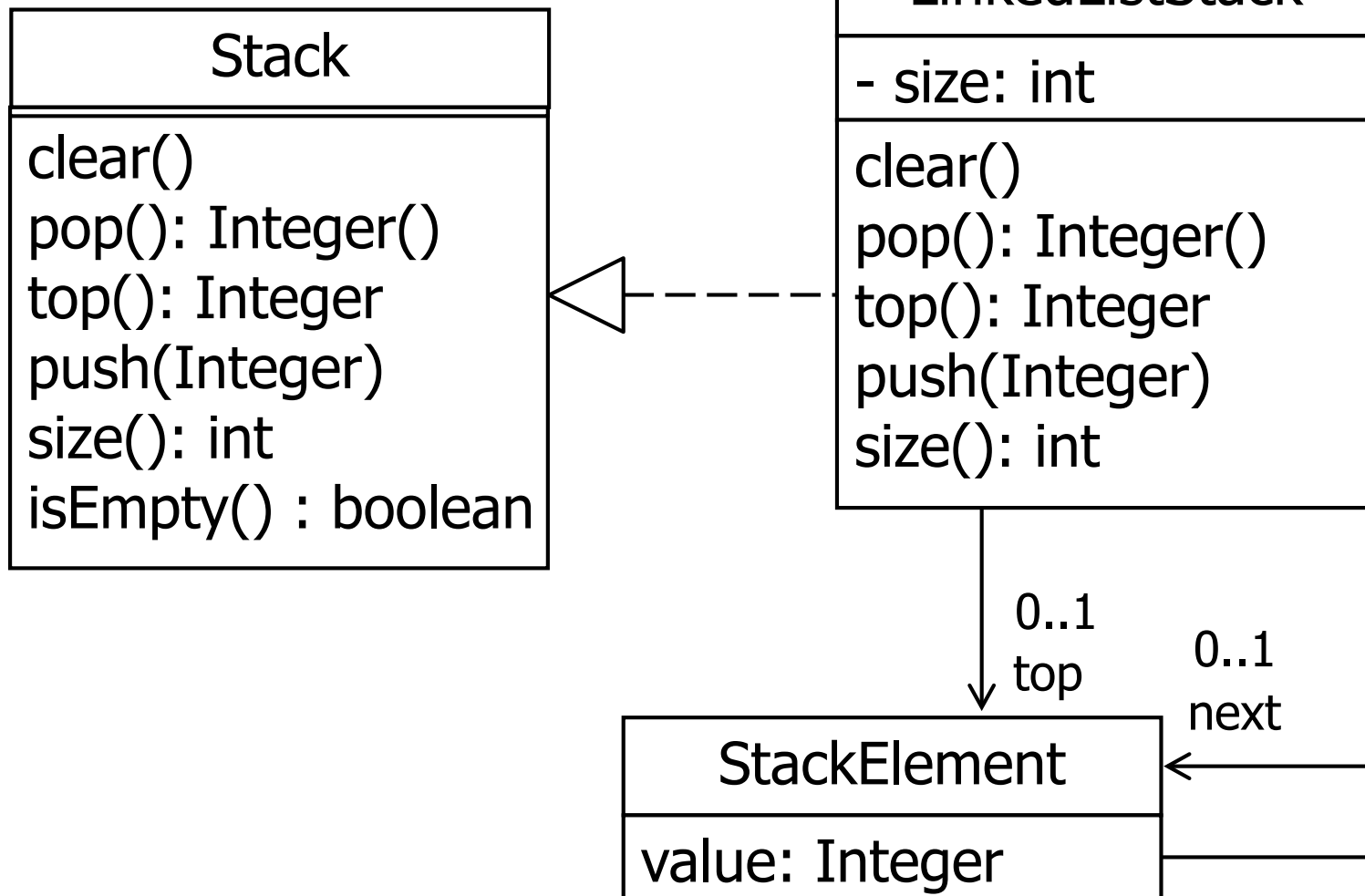
## Motivation

- 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;  
    }  
}
```

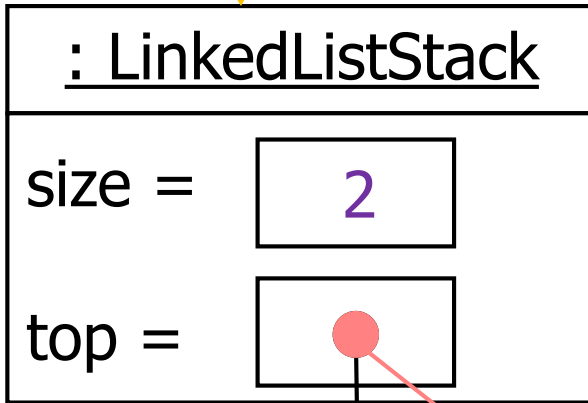
```
public class LinkedListStack implements Stack {  
  
    @Override  
    public void clear() {  
        ...  
    }  
  
    @Override  
    public Integer pop() {  
        ...;  
    }  
  
    ...  
}
```

- Som enkelt-hægtede liste

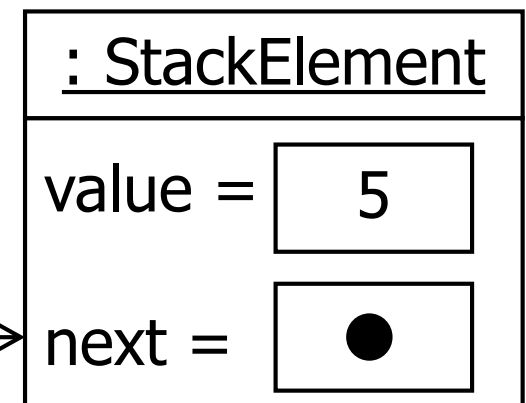
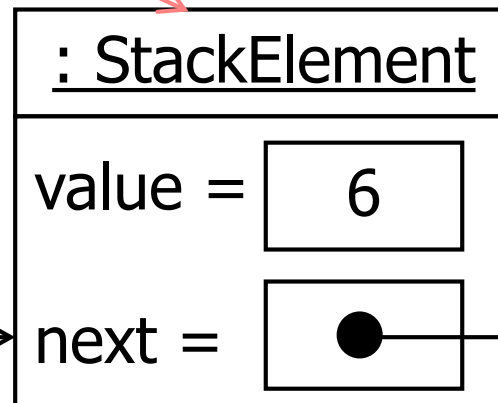
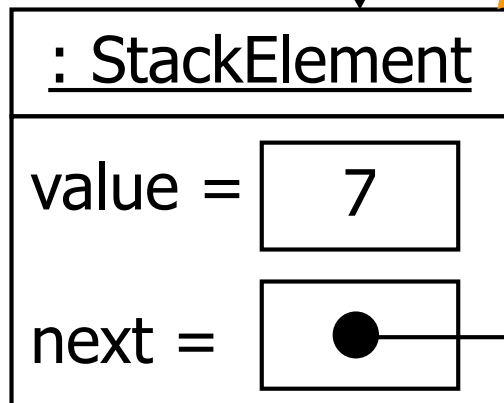


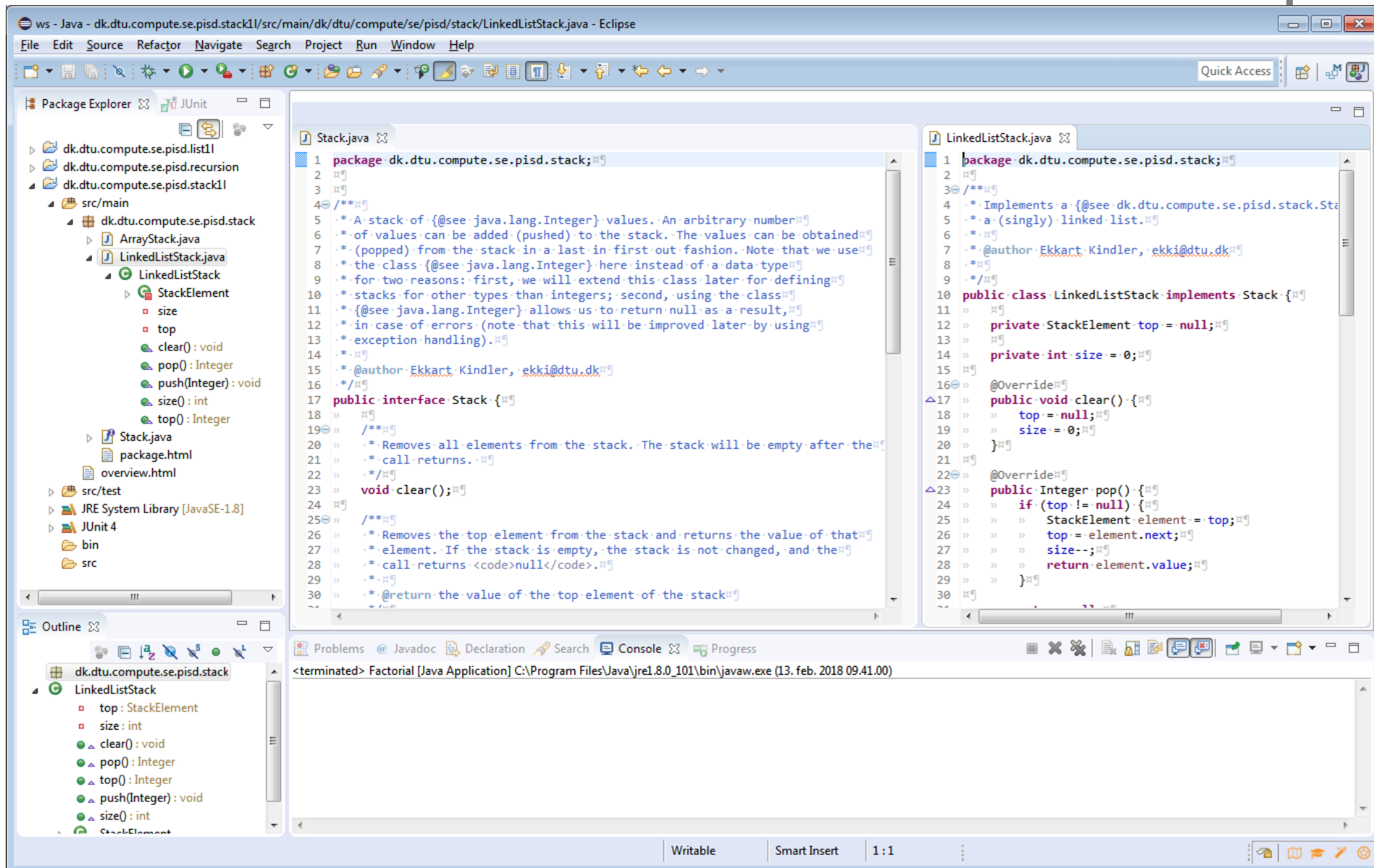
# pop()

this = ● element = ●



```
public Integer pop() {  
    StackElement element = this.top;  
    if (element != null) {  
        this.top = element.next;  
        size--;  
    }  
    return element.value;  
}
```





ws - Java - dk.dtu.compute.se.pisd.stack11/src/main/dk/dtu/compute/se/pisd/stack/LinkedListStack.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- dk.dtu.compute.se.pisd.list11
- dk.dtu.compute.se.pisd.recursion
- dk.dtu.compute.se.pisd.stack11
  - src/main
    - dk.dtu.compute.se.pisd.stack
      - ArrayStack.java
      - LinkedListStack.java
        - LinkedListStack
          - 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 @author Ekkart Kindler, ekki@dtu.dk
16 */
17 public interface Stack {
18     //
19     /**
20      * Removes all elements from the stack. The stack will be empty after the
21      * call returns.
22      */
23     void clear();
24
25     /**
26      * Removes the top element from the stack and returns the value of that
27      * element. If the stack is empty, the stack is not changed, and the
28      * call returns <code>null</code>.
29      */
30     //
31     // @return the value of the top element of the stack
32 
```

LinkedListStack.java

```

1 package dk.dtu.compute.se.pisd.stack;
2
3 /**
4  * Implements a {@see dk.dtu.compute.se.pisd.stack.Stack}
5  * a (singly) linked list.
6  */
7 @author Ekkart Kindler, ekki@dtu.dk
8
9 /**
10  *
11  */
12 public class LinkedListStack implements Stack {
13     //
14     private StackElement top = null;
15     private int size = 0;
16
17     @Override
18     public void clear() {
19         top = null;
20         size = 0;
21     }
22
23     @Override
24     public Integer pop() {
25         if (top != null) {
26             StackElement element = top;
27             top = element.next;
28             size--;
29             return element.value;
30         }
31     }
32 
```

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

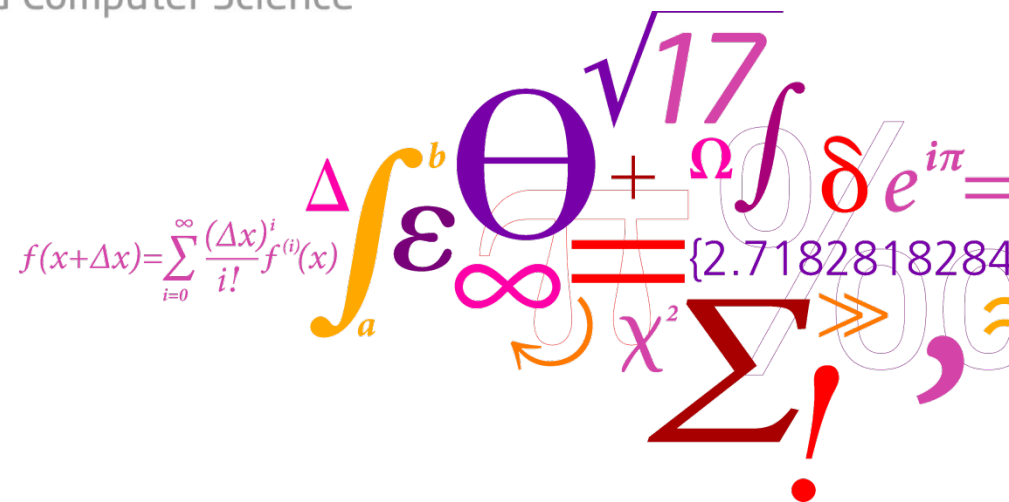
<terminated> Factorial [Java Application] C:\Program Files\Java\jre1.8.0\_101\bin\javaw.exe (13. feb. 2018 09.41.00)

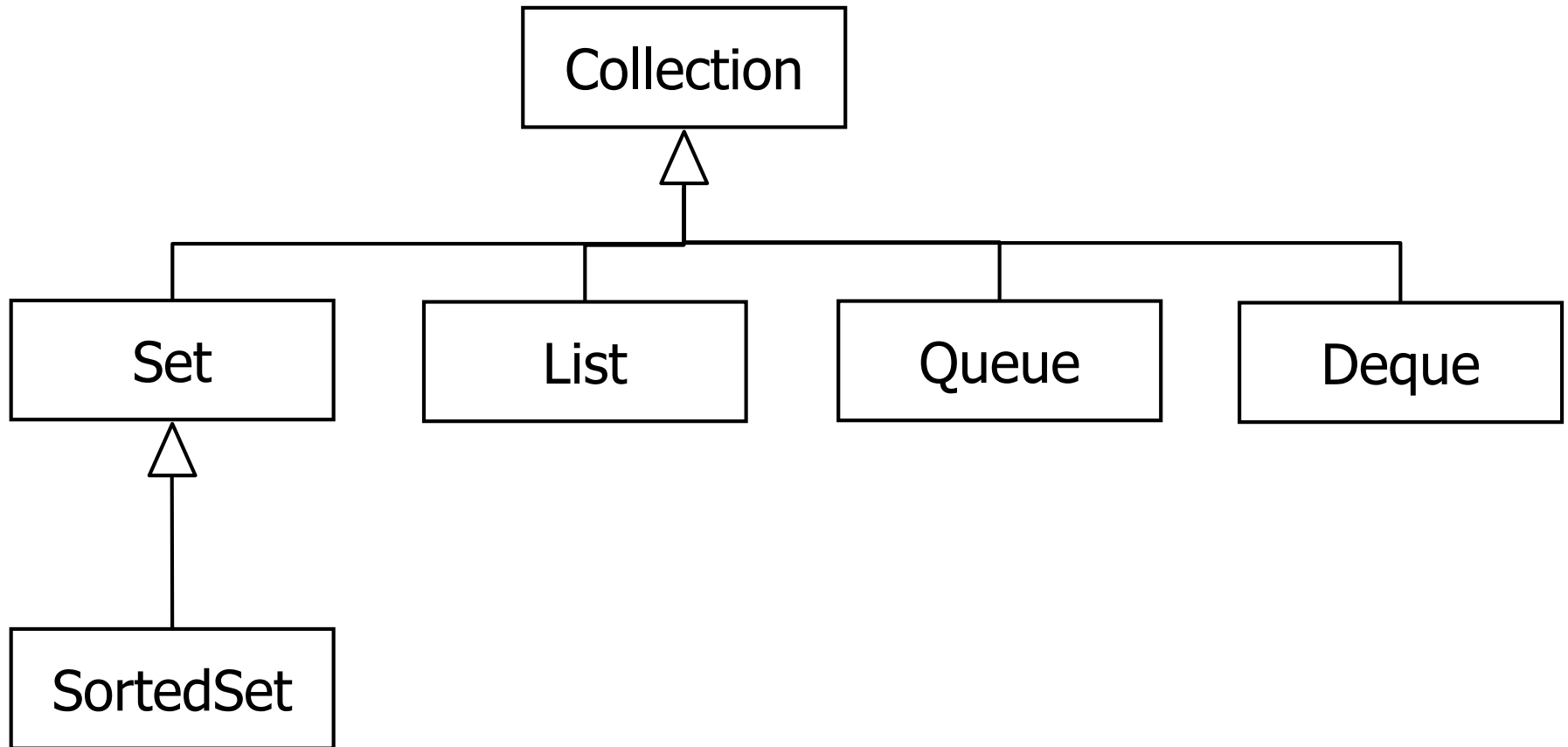
Writable Smart Insert 1:1

# Java's indbyggede datastrukturer

**DTU Compute**

Department of Applied Mathematics and Computer Science





```
public interface Comparable<T> {
```

```
    public int compareTo(T o);
```

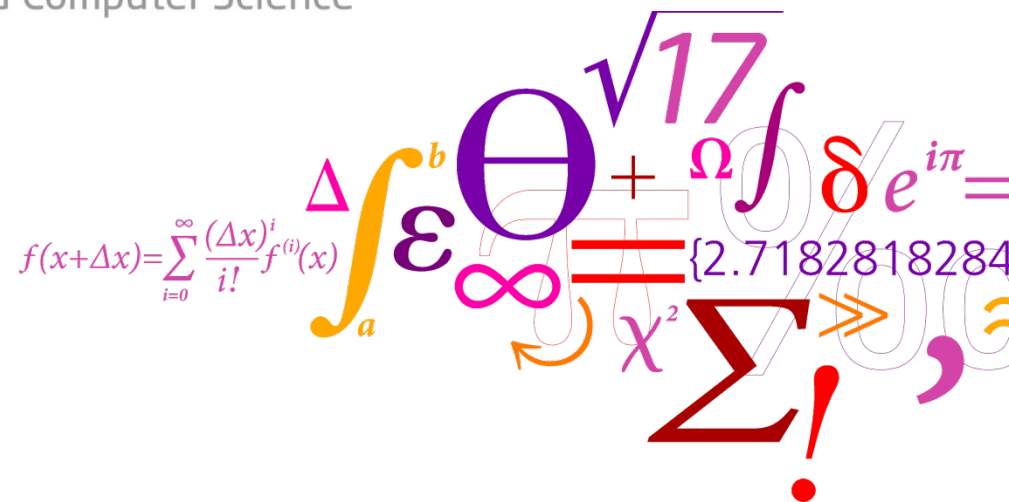
```
}
```

*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



Vi antager at listen ligger i et array

```
int[] list = ...
```

og

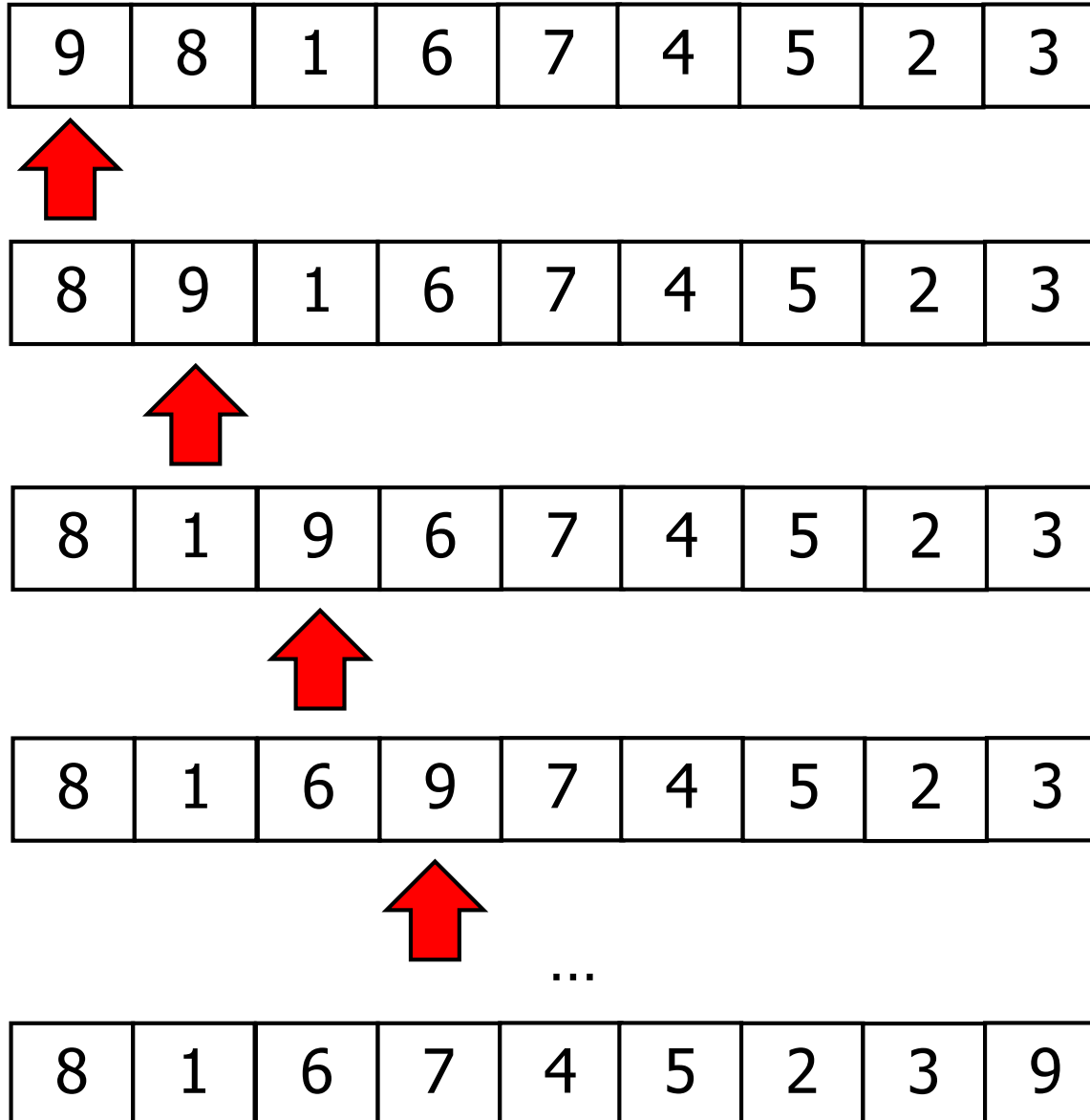
```
int size = ...
```

er den aktuelle længde af listen

```
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



# 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 |
|---|---|---|---|---|---|---|---|---|



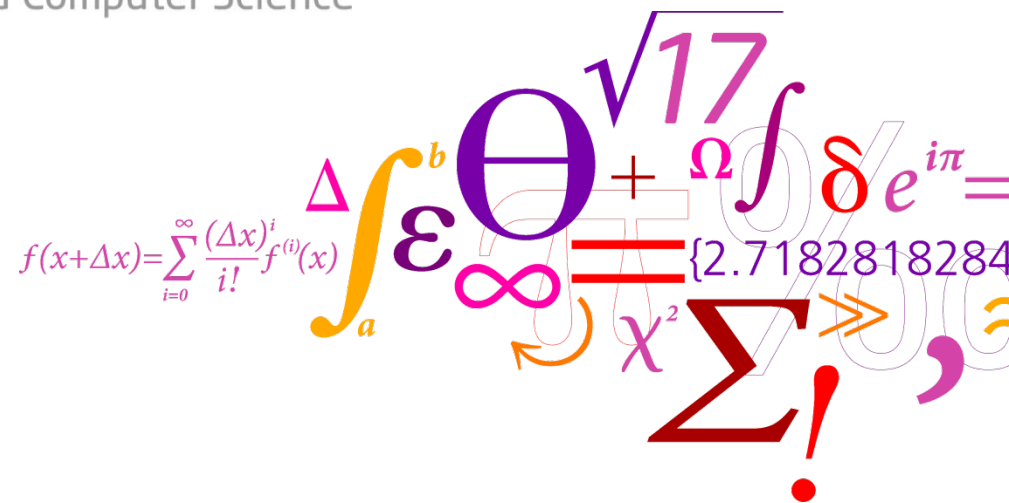
|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 4 | 5 | 2 | 3 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|

I hver iteration kan vi  
stoppe en position  
tidligere!

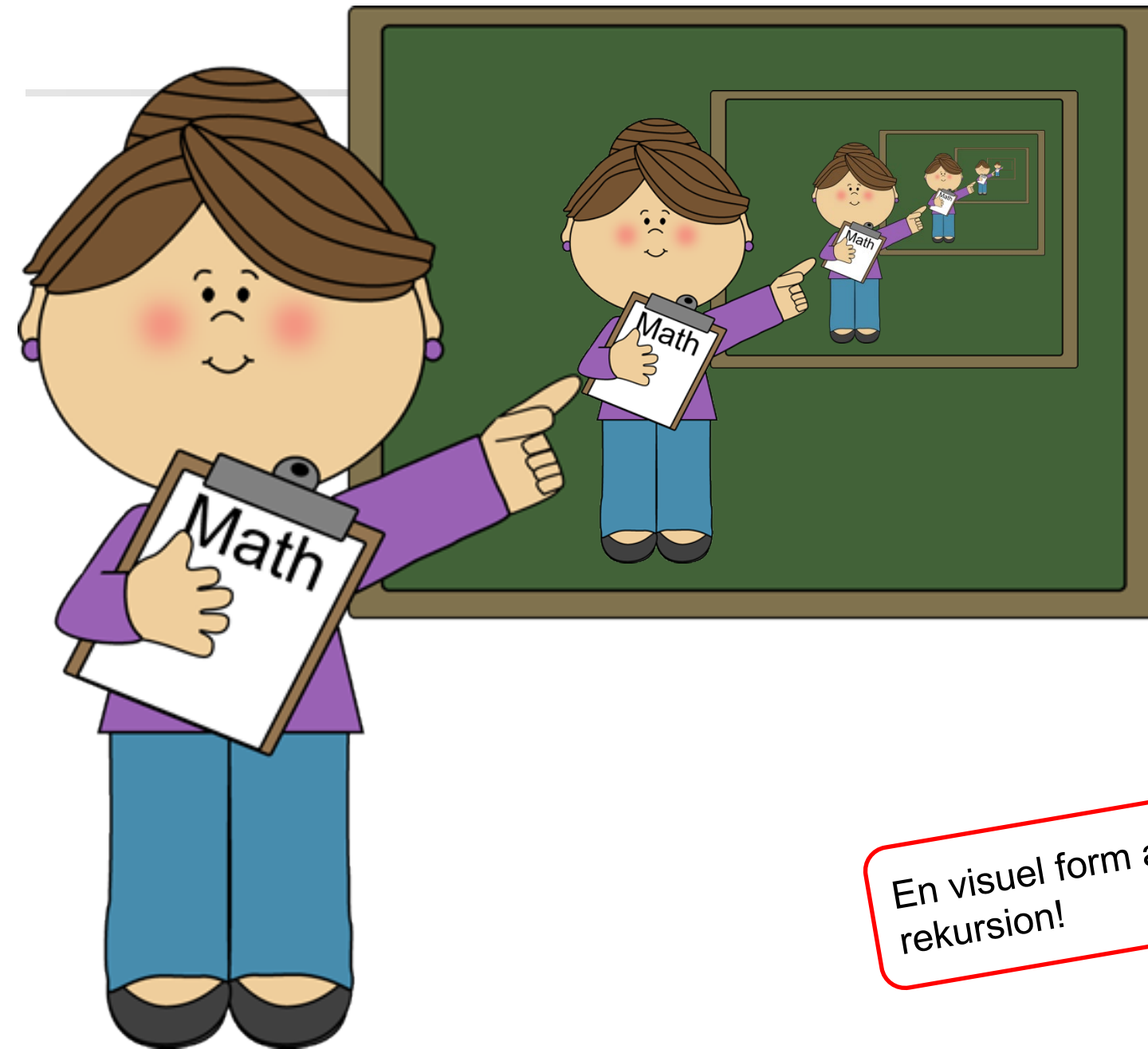
# Rekursion

**DTU Compute**

Department of Applied Mathematics and Computer Science



A collage of colorful mathematical symbols and expressions. It includes a Taylor series expansion  $f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$ , a definite integral  $\int_a^b \epsilon$ , a large purple  $\Theta$ , a square root  $\sqrt{17}$ , a plus sign  $+$ , a Greek letter  $\Omega$ , a delta function  $\delta$ , an exponential  $e^{i\pi}$ , an equals sign  $=$ , a set of curly braces  $\{2.7182818284\}$ , a Greek letter  $\chi^2$ , a summation symbol  $\Sigma$ , a greater-than sign  $>$ , a comma  $,$ , and a red exclamation mark  $!$ .



En visuel form af en  
rekursion!

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

Her er der **rekursion** igen

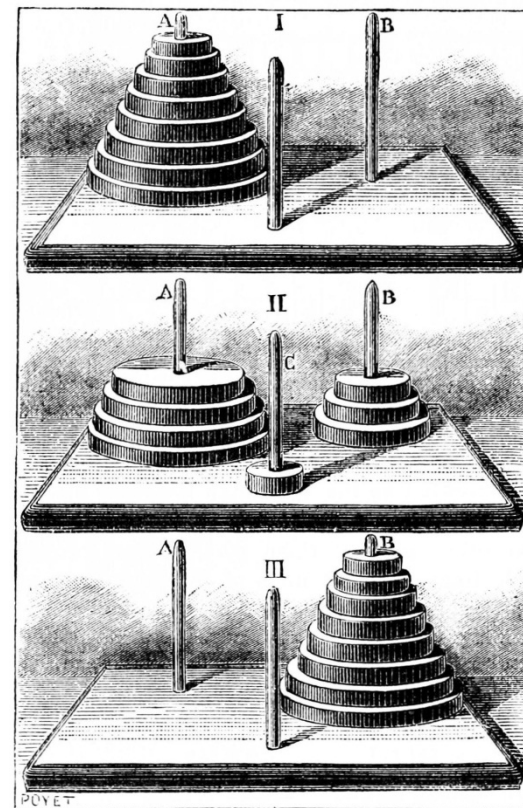
```
private static int factorial(int n) {  
    if (n == 0) {  
        return 1;  
    } else if (n > 0) {  
        return n * factorial(n-1);  
    } else {  
        throw IllegalArgumentException(  
            "f(" + n + ")");  
    }  
}
```

Må vi det? Metoden deklarerer ikke  
*IllegalArgumentException*?  
Hvordan kan vi finde ud af det?

## Engl. Towers of Hanoi

### Hanois tårne: Matematisk "puslespil":

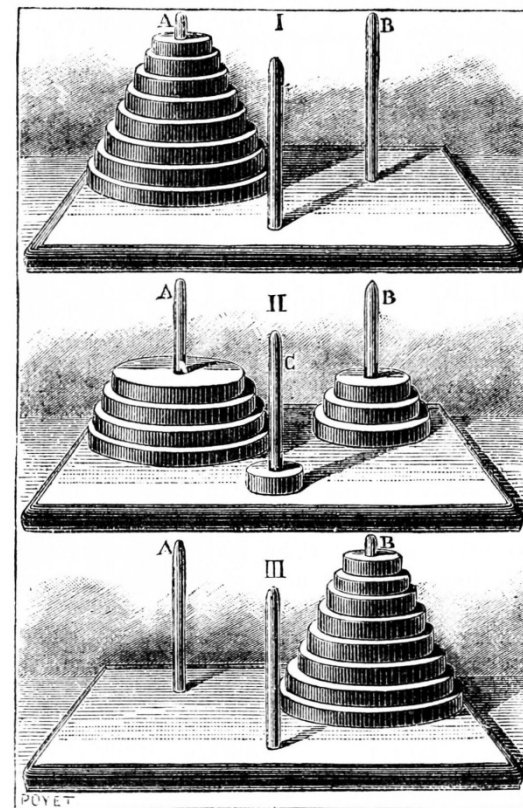
- Stativ med tre pinde ( $A$ ,  $B$  og  $C$ )
- På pind  $A$  er der  $n$  skiver med aftagende størrelse (set nedfra)
- **Opgave:**  
Flyt alle  $n$  skiver fra  $A$  til  $B$ , men
  - Kun en skive ad gangen
  - En større skive må aldrig ligge på en skive som er mindre



Flyt  $n$  skiver fra  $A$  til  $B$

1. Flyt de øverste  $n-1$  skiver fra  $A$  til  $C$   
(ifølge reglerne)
2. Så fly den nederste (størreste) 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)

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

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

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



**Sorter** partitionerne uafhængigt af hinanden **rekursiv**

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

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

Så er hele arrayet sorteret

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

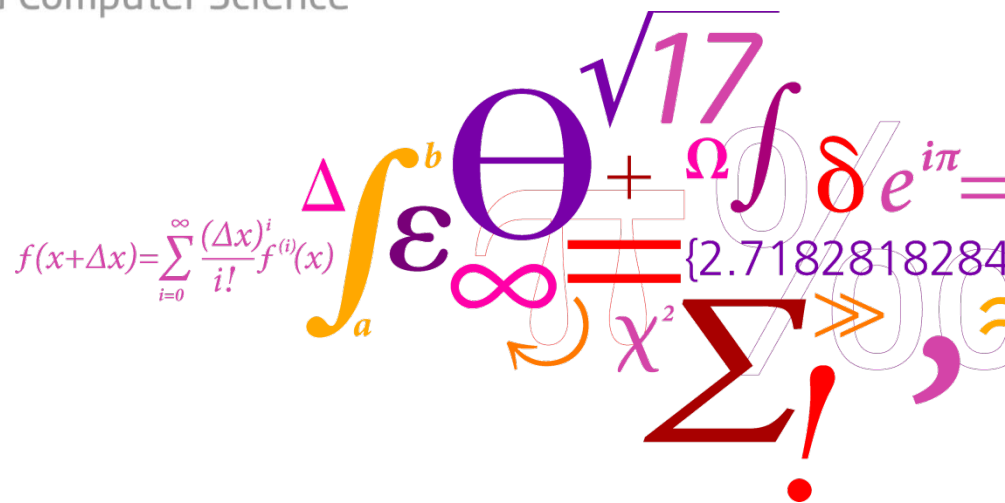
Bemærk at det er vigtigt at begge partitioner bliver mindre! Begge skal have mindst et element

# API Design

Indeholder også tekniske  
emner som generiske typer  
og exceptions!

**DTU Compute**

Department of Applied Mathematics and Computer Science



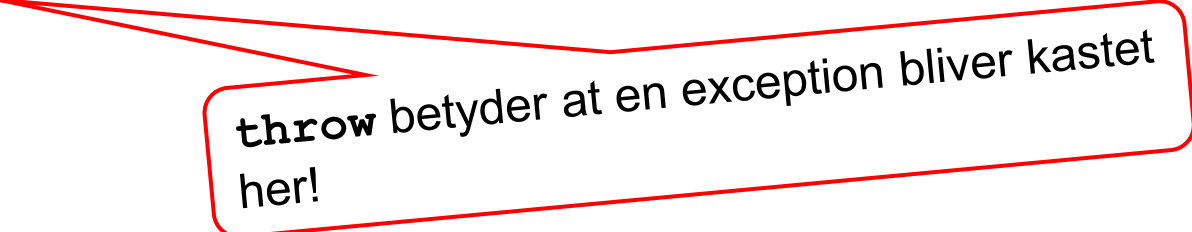
A collage of colorful mathematical symbols and formulas. Visible elements include: a Taylor series formula  $f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$ ; a definite integral  $\int_a^b \epsilon$ ; a large purple  $\Theta$ ; a square root  $\sqrt{17}$ ; a plus sign  $+$ ; a pink infinity symbol  $\infty$ ; a red  $\delta$ ; an exponential term  $e^{i\pi}$ ; a set notation  $\{2.7182818284\}$ ; a chi-squared symbol  $\chi^2$ ; a large red summation symbol  $\Sigma$ ; and a red exclamation mark  $!$ .

```
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**

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

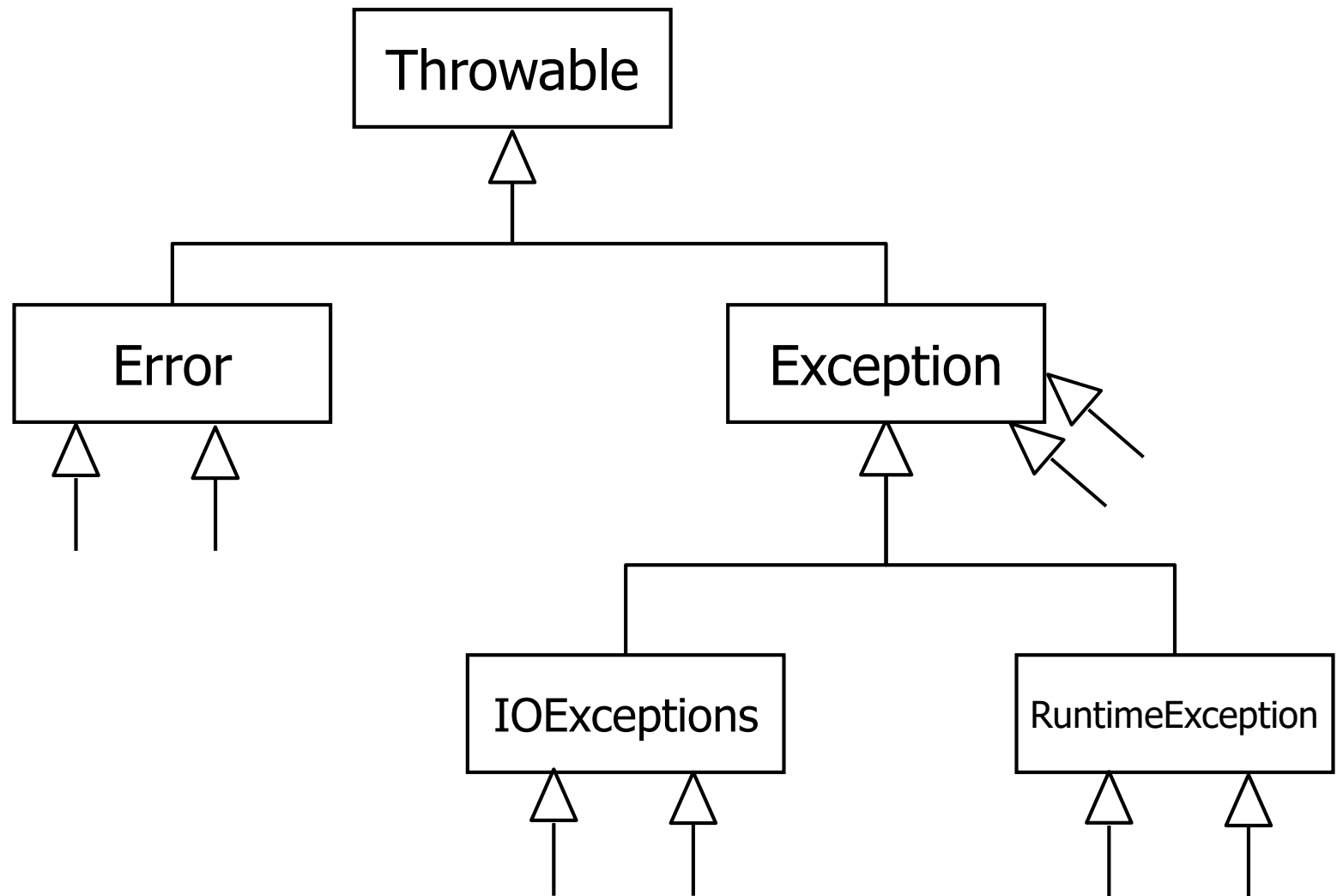
```
public class LinkedListStack<E> implements Stack<E> {  
  
    ...  
  
    public E pop() throws IllegalStateException {  
        if (top != null) {  
            StackElement element = top;  
            top = element.next;  
            size--;  
            return element.value;  
        }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



throw betyder at en exception bliver kastet her!

```
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()`.



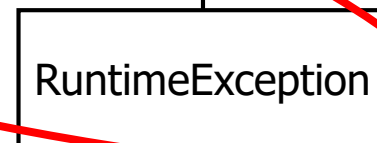
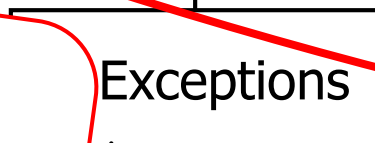
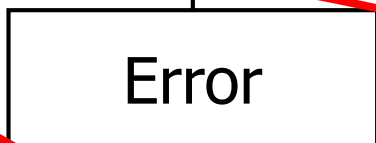
- er exceptions, hvor den, som kalder metoden (*engl. caller*), kan gøre noget ved
- typisk: forkert brug af metoden (kalde med de rigtige parameter), IOException (prøve igen),
- skal specificeres i metodens deklaration eller fanges i metodens implementering

- 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

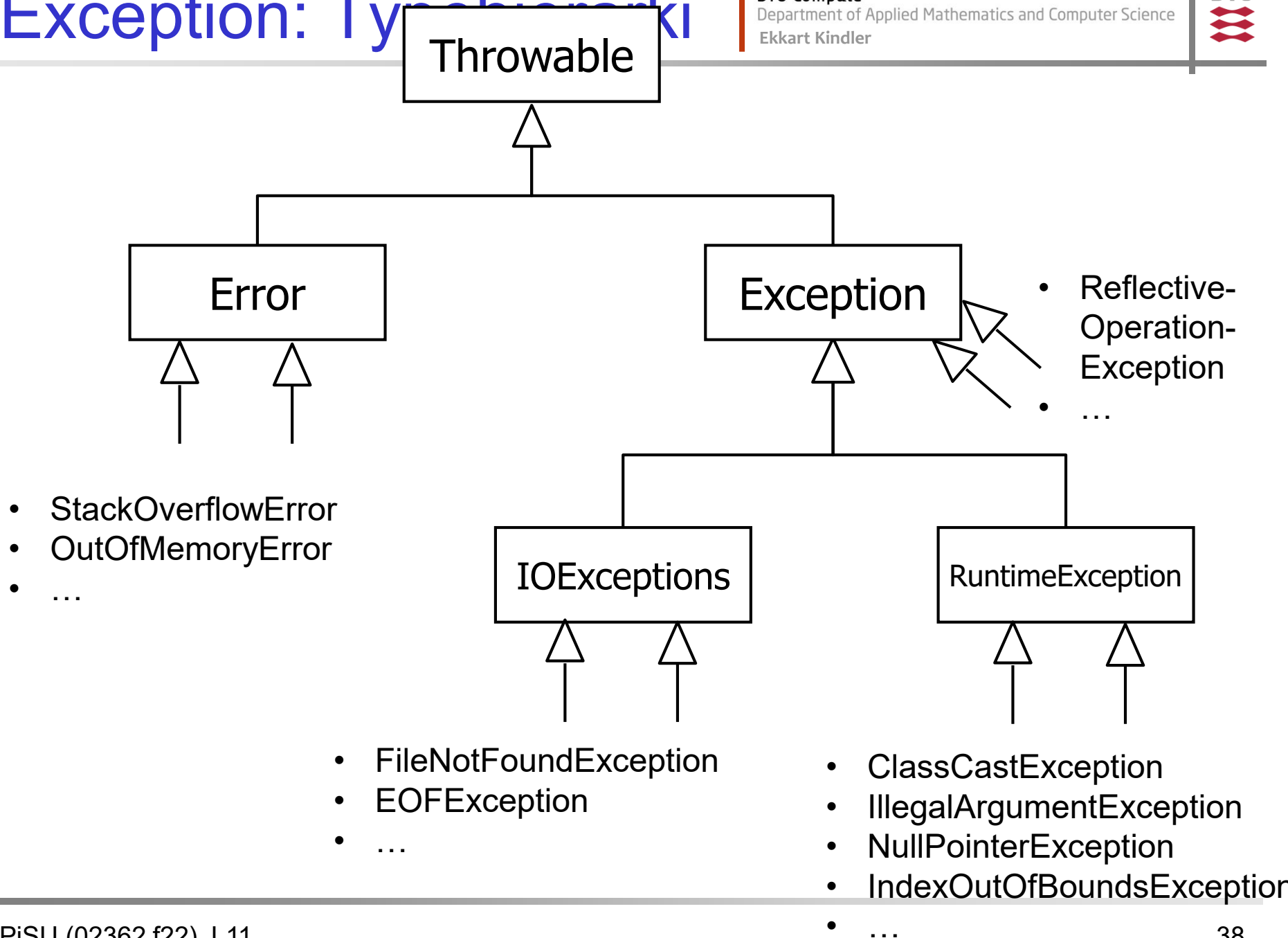
# Exception: Typehierarki



... alle andre er overvågede "exceptions"



Klasser som bliver afledt (engl. derived) af implementeringen af **Error** eller **RuntimeException** er uovervågede "exceptions" ...



# Kontrolflow Exceptions

```
try {  
    ...  
    do something;  
    callSomeone();  
    ...  
} catch (Exception1 e) {  
    ...  
    do something  
} catch (Exception2 e) {  
    ...  
    do something  
} finally {  
    ...  
    do something  
}
```



Hvis der ikke sker en exception i try-blokken,

Så bliver catch-blokkene sprunget over!

Men finally-bokken (hvis den er der) bliver ekseveret

```
try {  
    ...  
    do something;  
    callSomeone();  
    ...  
} catch (Exception1 e) {  
    ...  
    do something  
} catch (Exception2 e) {  
    ...  
    do something  
} finally {  
    ...  
    do something  
}
```



Når der sker en exception i try-blokken (som ikke bliver fanget indeni),

bliver resten af koden i try-blokken sprunget over

og den første catch-blok som matcher (er super klasse af) den udløste exception eksekveret (men kun en!)

og finally-blokken (hvis den er der) bliver også eksekveret!

# En krølle: finally/return

```
try {  
    ...  
    do something;  
    return;  
    ...  
} catch (Exception1 e) {  
    ...  
    do something  
} catch (Exception2 e) {  
    ...  
    do something  
} finally {  
    ...  
    do something  
}
```

Finally-blokken bliver **altid** eksekveret når man forlader en try/blok (også når man forlader den med return).

# Analyse og konceptuelle modeller

**DTU Compute**

Department of Applied Mathematics and Computer Science

$$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$$
$$\int_a^b \varepsilon \Theta + \Omega \int \delta e^{i\pi} = \{2.7182818284\}$$
$$\chi^2 \Sigma ! > ,$$

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

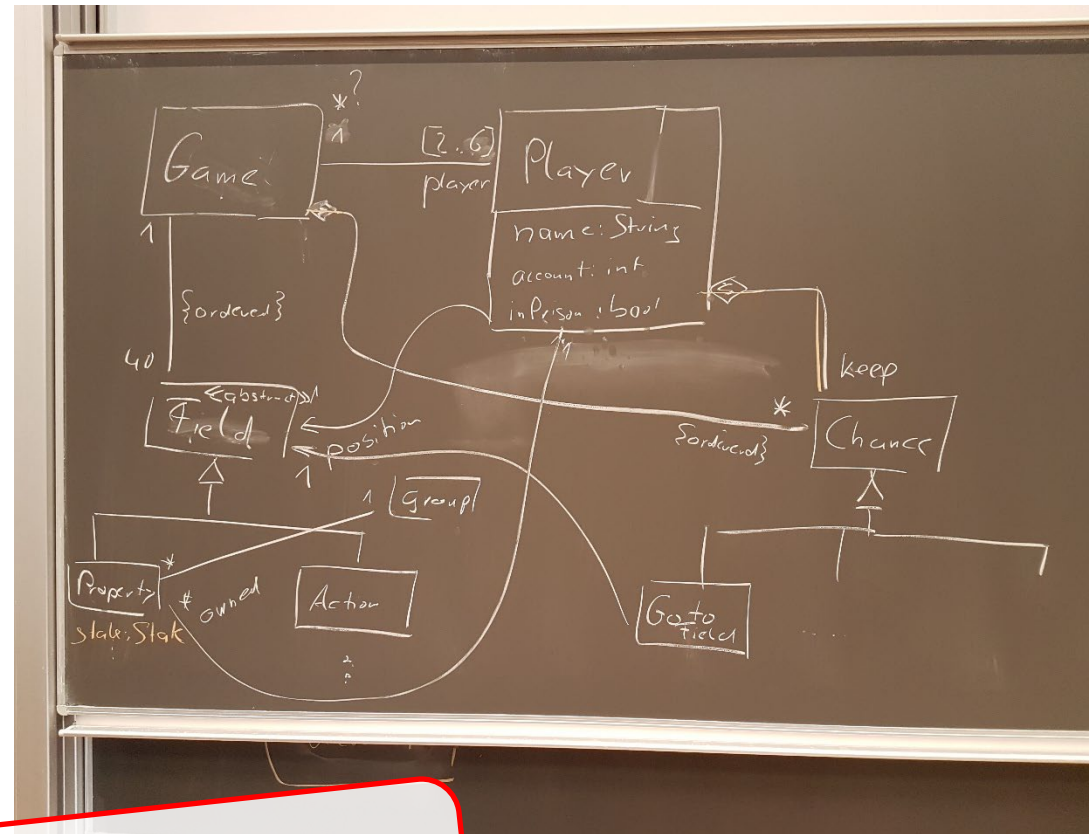
- Find skriftlig dokumentation om området (domænet) og skriv det ud
- Marker alle begreber som synes relevante
- Tal med nogle eksperter fra domænet
- **Taksonomi**: liste af begreber (ord)
- **Glossar**: Beskrivelse af begreber deres egenskaber og hvordan de forholder sig til hinanden
- **Domænemodel**: 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!

### Taxonomi

|                    |                    |
|--------------------|--------------------|
| Telt (space/field) | ejendom (property) |
| Spiller (player)   | brille (token)     |
| Plantsætning       | hus                |
| Chancekort         | hotel              |
| Chance             | fængsel            |
| Konto (par/double) | bank               |
| terning (die dice) | winner             |
| Spilbræt           | bankrot            |

Køb ejendom  
Køb hus/hotel  
betale leje → hus  
fængsel  
chancekort  
Løn (salary)  
Sælg grund  
property group



Fra 2019:  
Monopoly/Matador!

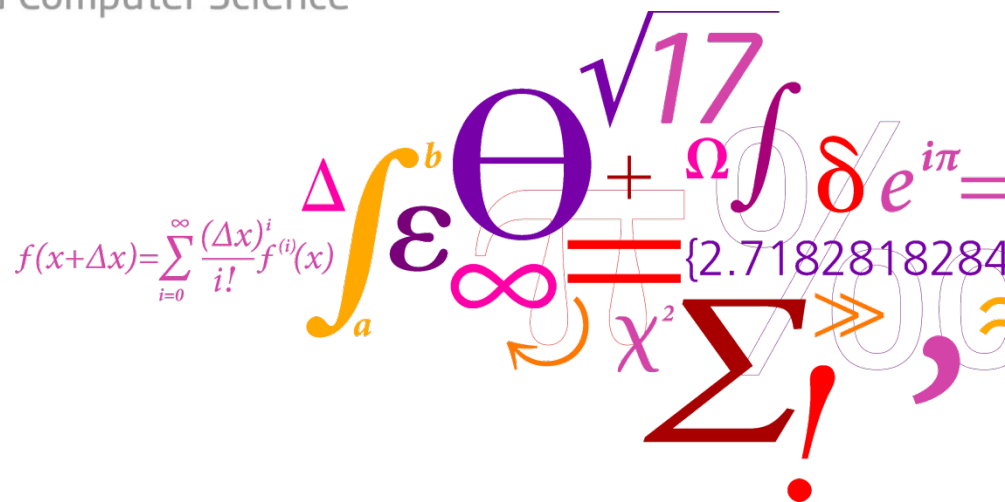
Fra 2019:  
Monopoly/Matador!

# (Software) Design

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

**DTU Compute**

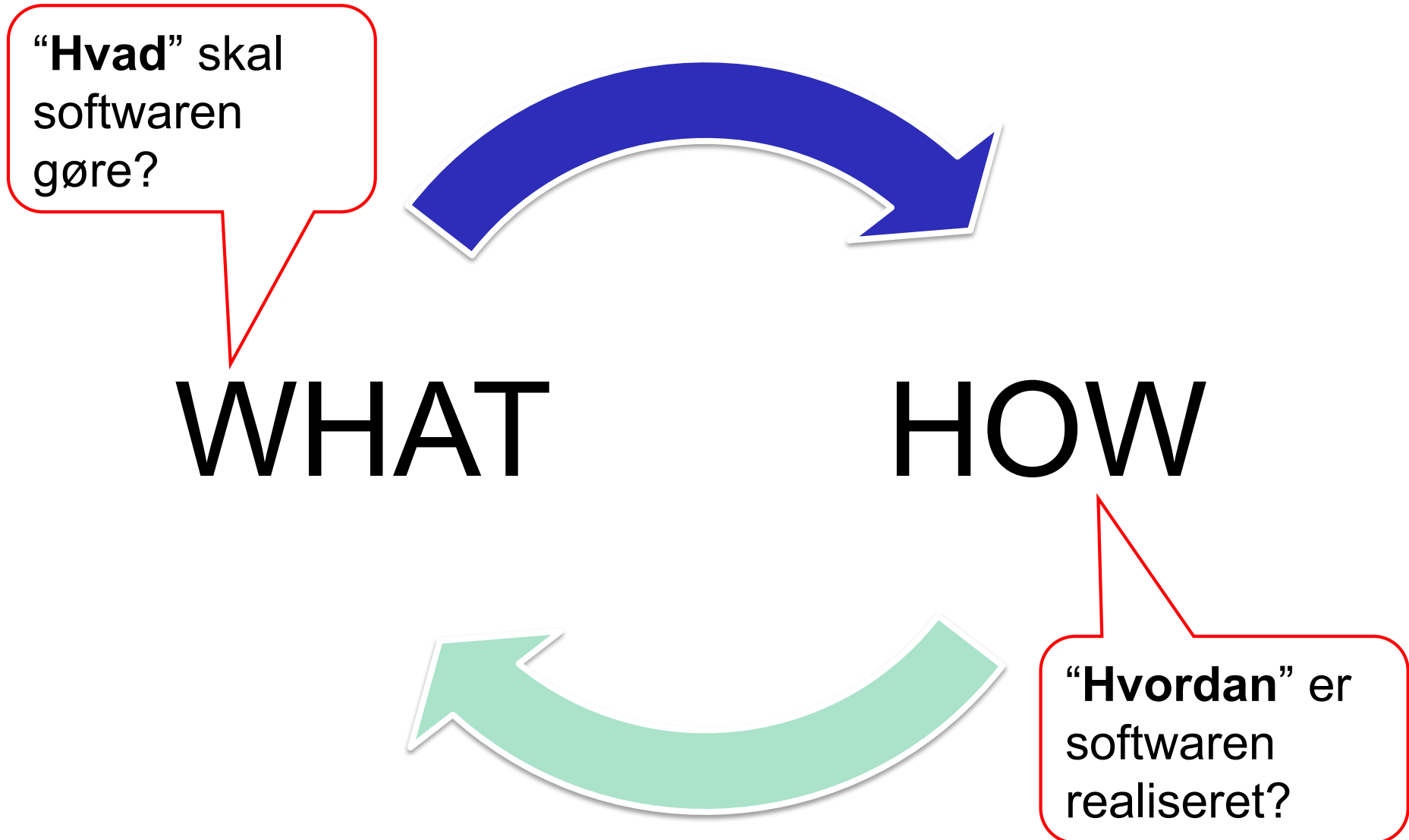
Department of Applied Mathematics and Computer Science



A collage of various mathematical symbols and formulas in different colors (purple, yellow, red, pink, orange). The symbols include:  $\Delta$ ,  $\int_a^b$ ,  $\epsilon$ ,  $\Theta$ ,  $\sqrt{17}$ ,  $+$ ,  $\Omega$ ,  $\int$ ,  $\delta$ ,  $e^{i\pi}$ ,  $=$ ,  $\{2.7182818284\}$ ,  $\infty$ ,  $\chi^2$ ,  $\Sigma$ ,  $!$ ,  $>$ , and  $\approx$ . A formula  $f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$  is also visible.

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

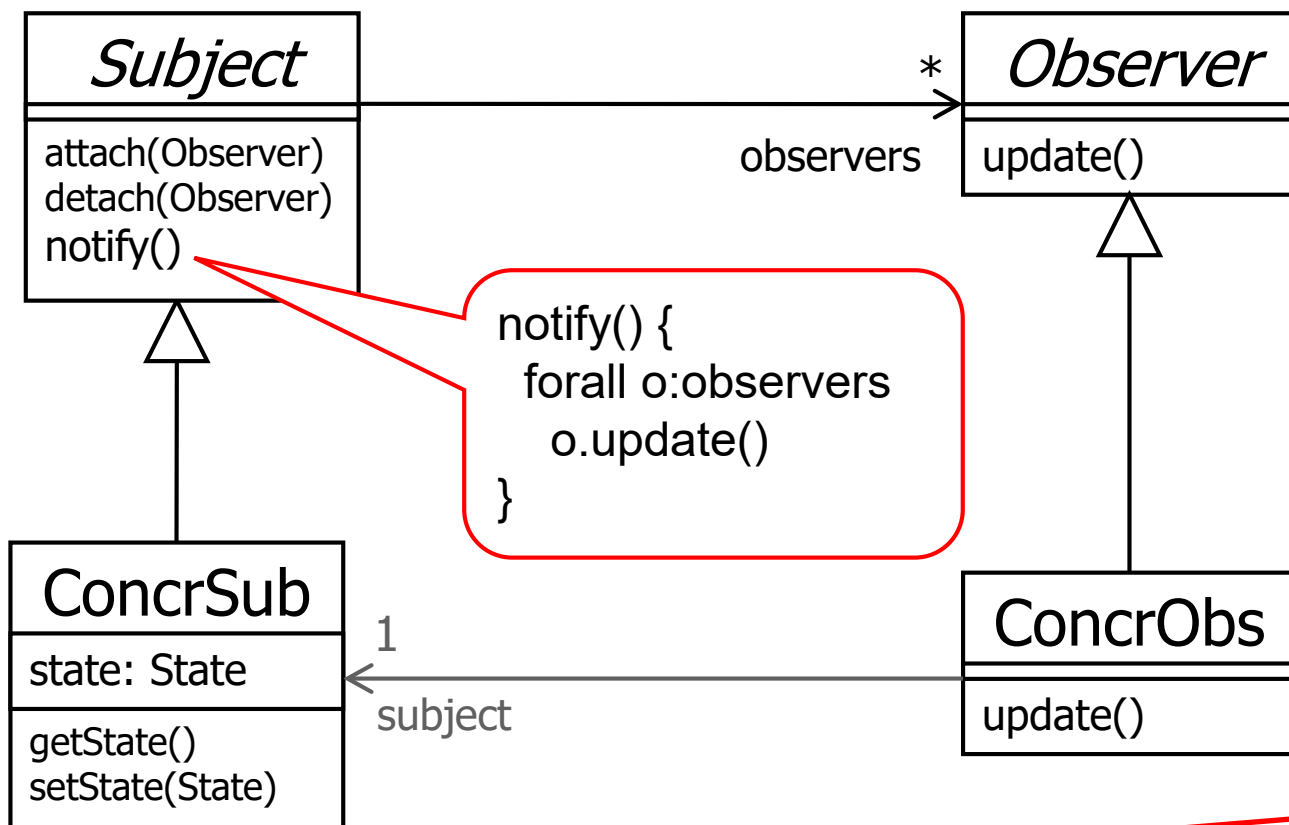
**Analysen** siger **ikke noget** om  
**HVORDAN** (engl. **HOW**) softwaren skal  
realiseres og implementeres



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.

## Structure



Hvor og hvordan bliver det brugt i jeres software? Det gælder også de dele, som kommer fra Ekkart!

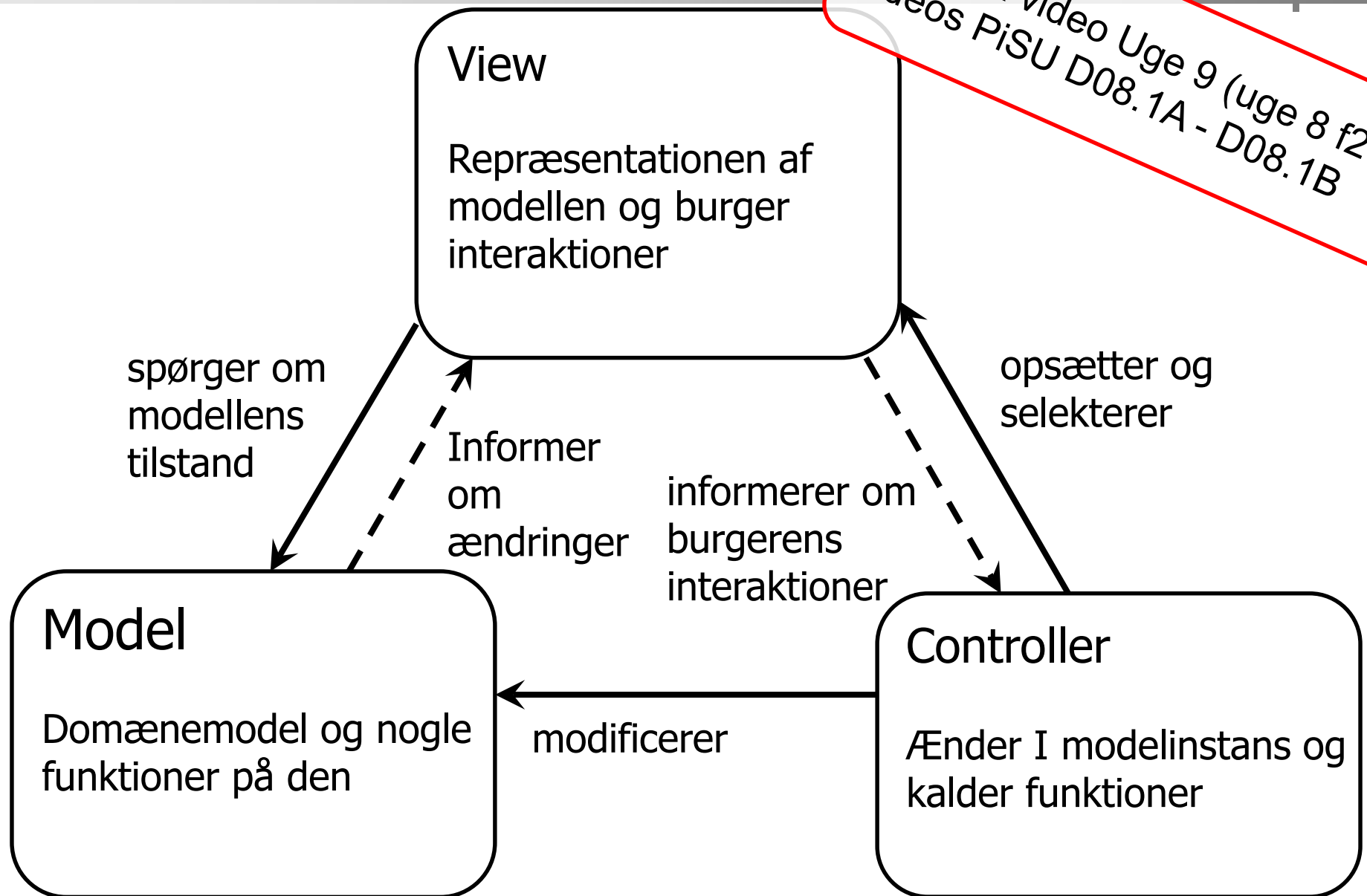
Hvis man gøre det rigtigt er **domæne modeller** (**model**) og deres implementeringer en fundamental del af den udviklede software

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".

Se også video Uge 9 (uge 8 f21)  
videos PiSU D08.1A - D08.1B

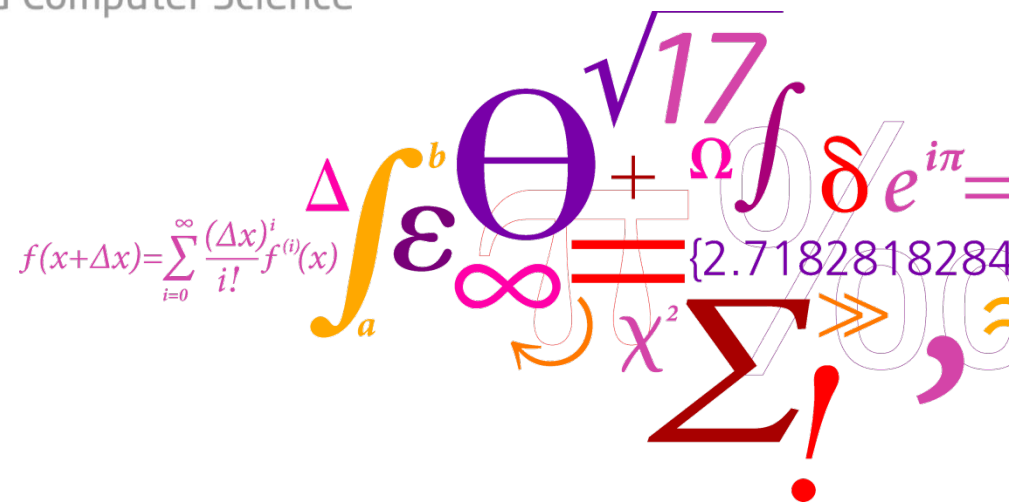


# Læse og skrive filer, Gson

# Dialoger og inputvalidering

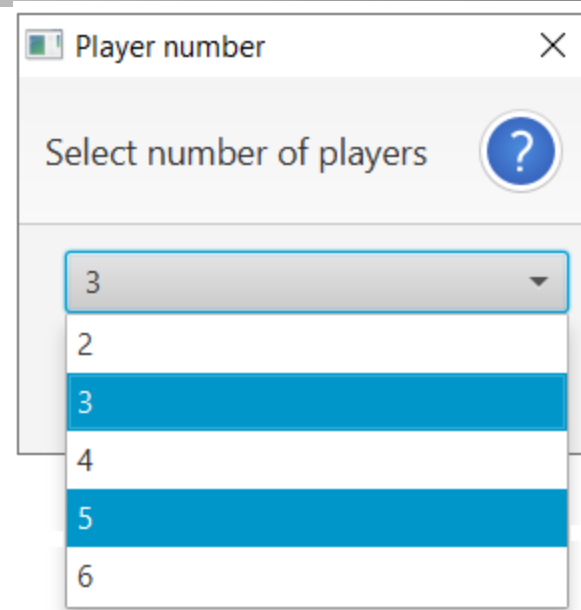
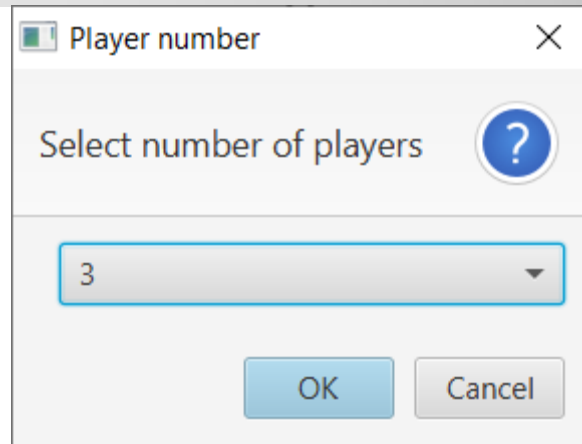
**DTU Compute**

Department of Applied Mathematics and Computer Science



A collage of various mathematical symbols and expressions in different colors (purple, orange, red, pink, yellow). The symbols include:  $\int_a^b$ ,  $\Delta$ ,  $\epsilon$ ,  $\Theta$ ,  $\sqrt{17}$ ,  $\Omega$ ,  $\delta$ ,  $e^{i\pi}$ ,  $\infty$ ,  $\chi^2$ ,  $\Sigma$ ,  $!$ ,  $\{2.7182818284\}$ , and  $\gg$ . Some symbols are overlaid on others, creating a dense, artistic representation of mathematics.

# 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!**

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

- Adresser
- Email-adresser
- Telefonnumre
- Datoer

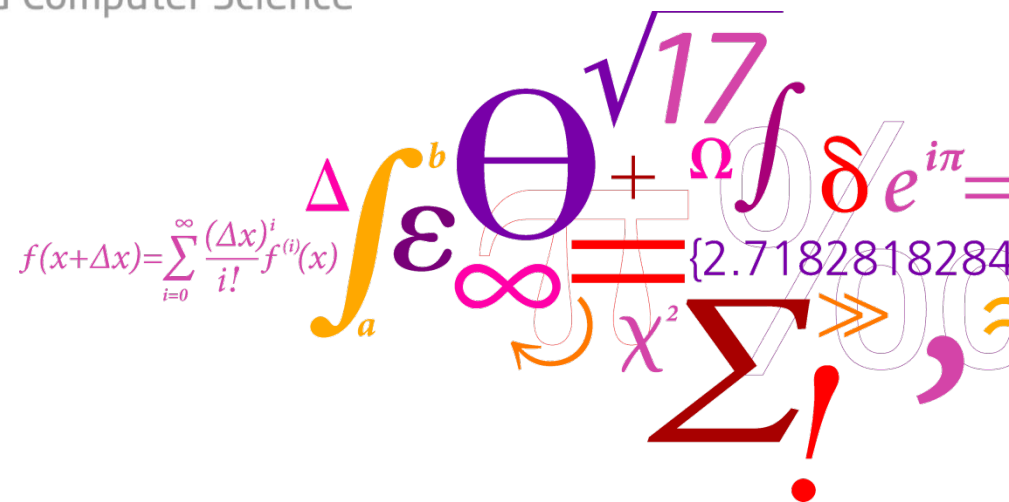
Regulære udtryk er et meget gammelt koncept i datalogi – meget ældre en Java. Men her bruger vi især notationen fra Java (\*, +, ?, | er i generel brug også i andre kontekster)  
→ Uge 8 video PiSU L08.A og L08.B

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 and expressions. It includes the Taylor series expansion  $f(x+\Delta x) = \sum_{i=0}^{\infty} \frac{(\Delta x)^i}{i!} f^{(i)}(x)$ , an integral  $\int_a^b \epsilon \Theta$ , a square root  $\sqrt{17}$ , a plus sign  $+$ , a Greek letter  $\Omega$ , a delta function  $\delta$ , an exponential  $e^{i\pi}$ , an equals sign  $=$ , a set of numbers  $\{2.7182818284\}$ , an infinity symbol  $\infty$ , a chi-squared symbol  $\chi^2$ , a summation symbol  $\Sigma$ , a greater-than symbol  $>$ , and an exclamation mark  $!$ .

```
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