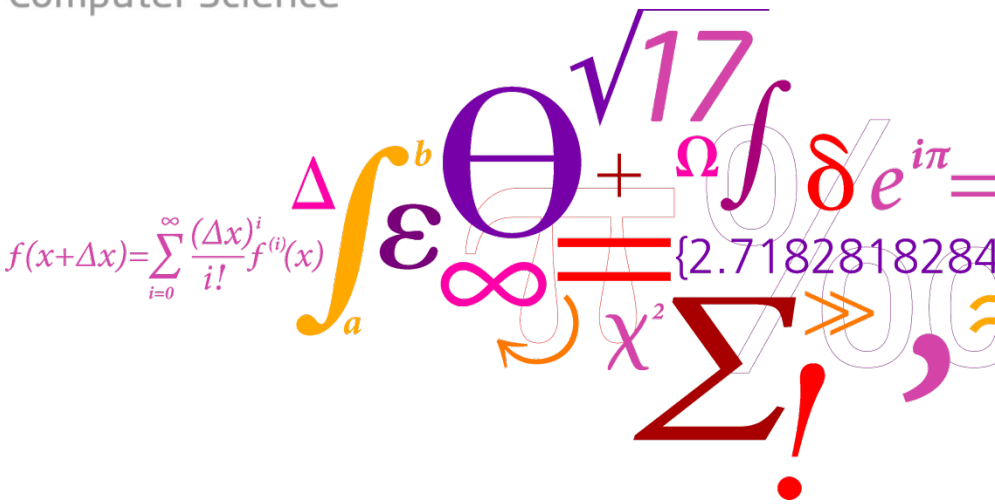
# Projekt i software-udvikling (02362)
## Forår 2022

Ekkart Kindler

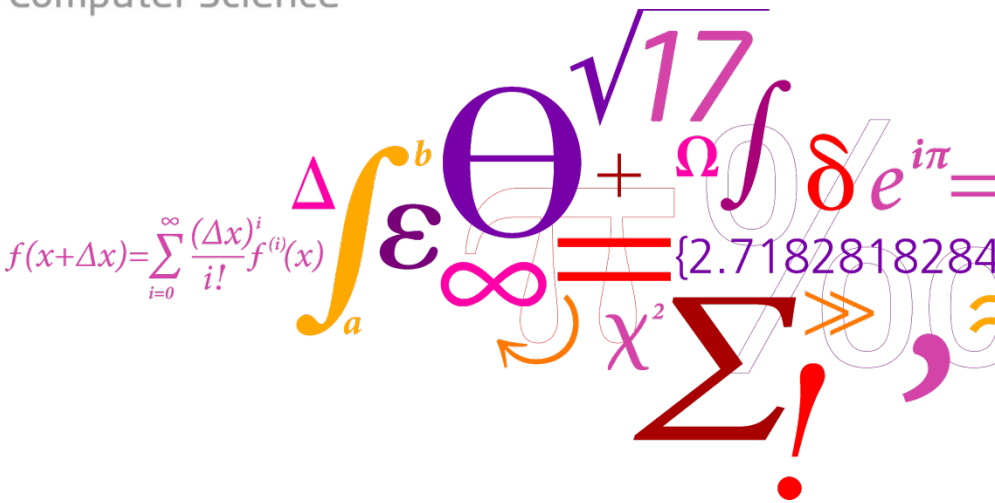
DTU Compute
Department of Applied Mathematics and Computer Science



# XI. Afslutning

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

**DTU Compute**
Department of Applied Mathematics and Computer Science



# Kursus: Kort sagt

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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

# Emner

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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

# Emner

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Modellering (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)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

# 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

# Interface `Stack`

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public interface Stack {

  void clear();

  Integer pop();

  Integer top();

  void push(Integer value);

  int size();

  default boolean isEmpty() {
    return size() == 0;
  }
}
```

# Implementering (eks.)

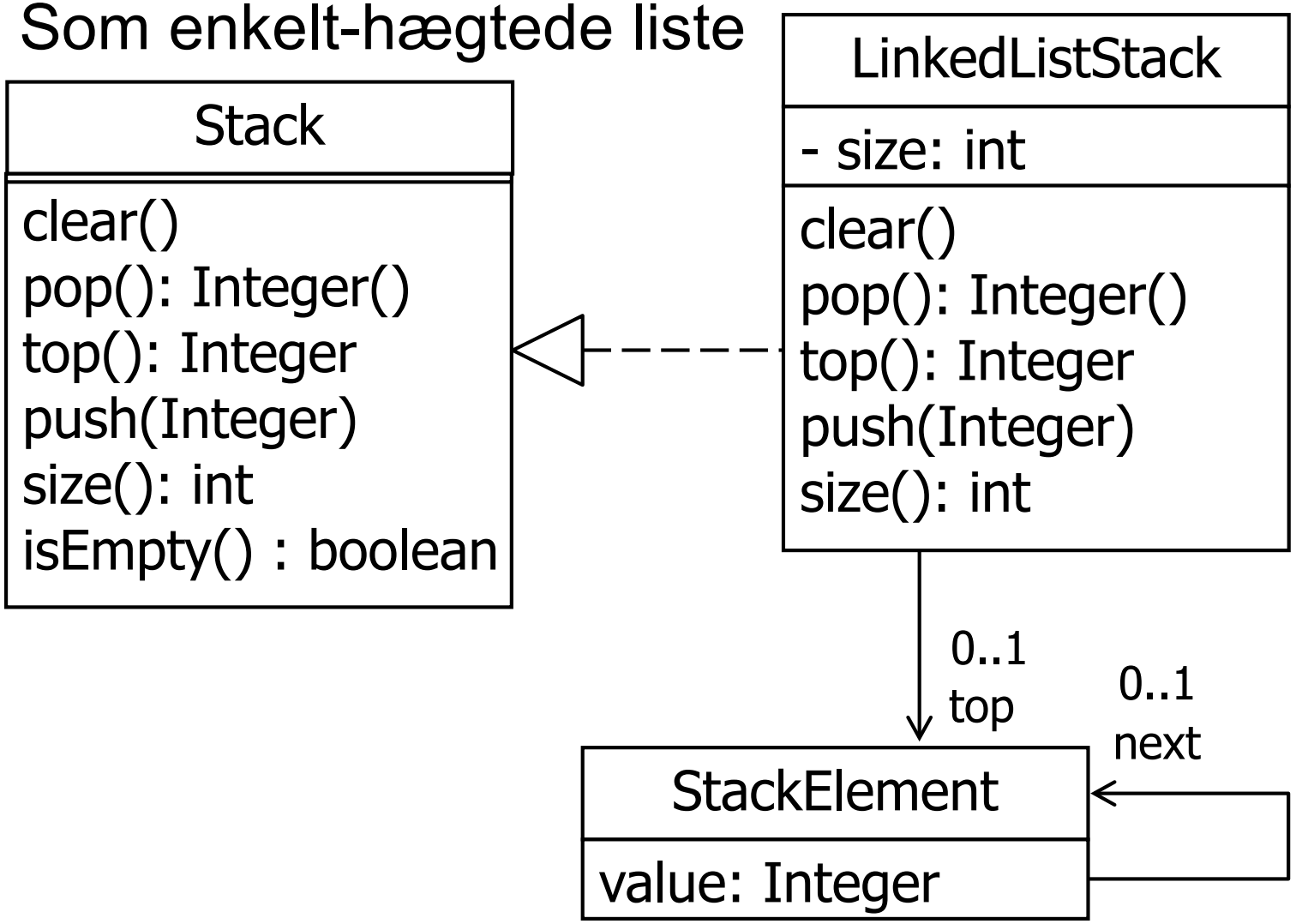
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public class LinkedListStack implements Stack {

   @Override
   public void clear() {
      ...
   }

   @Override
   public Integer pop() {
      ...;
   }
   ...
}
```

# Implementeringer

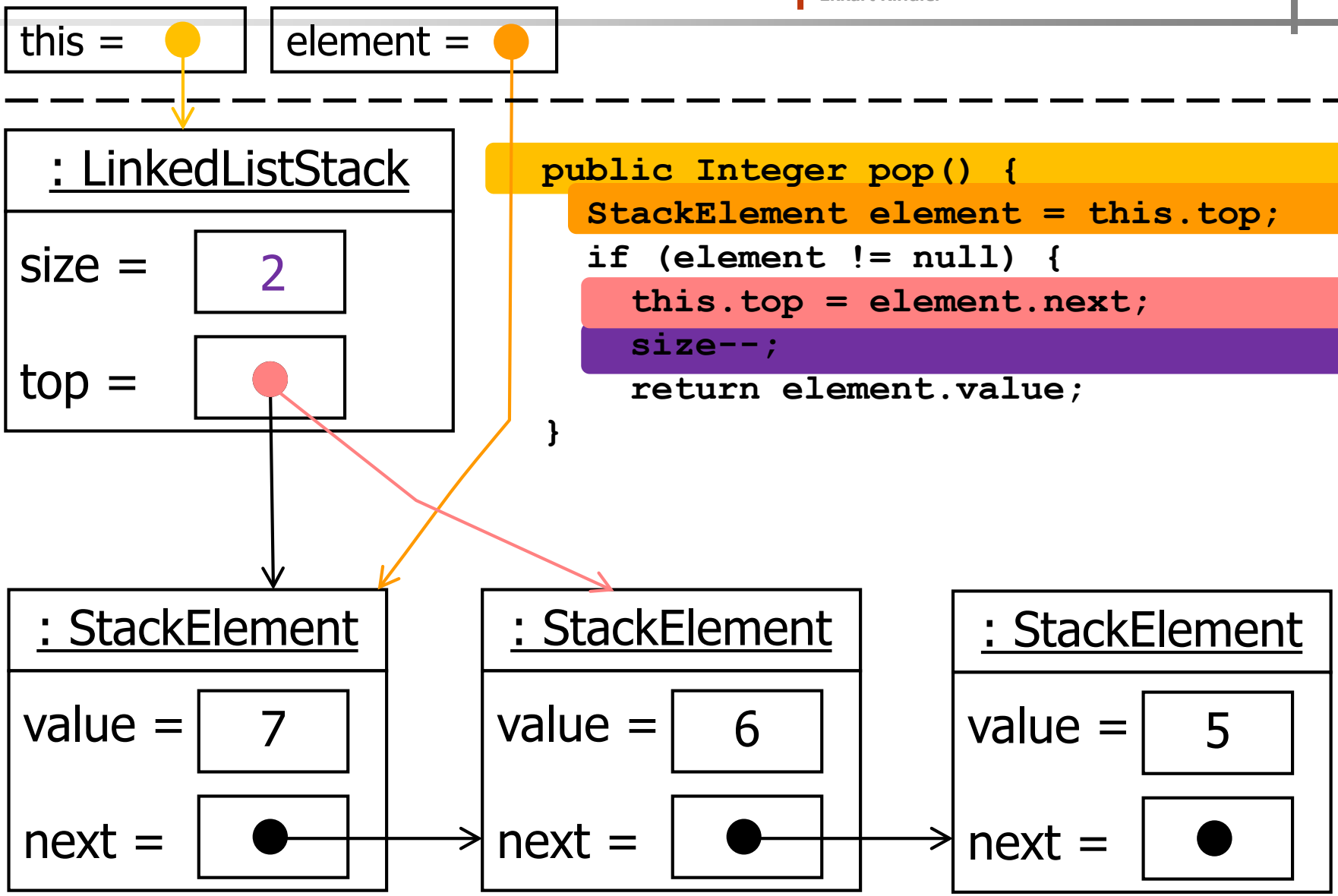
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Som enkelt-hægtede liste



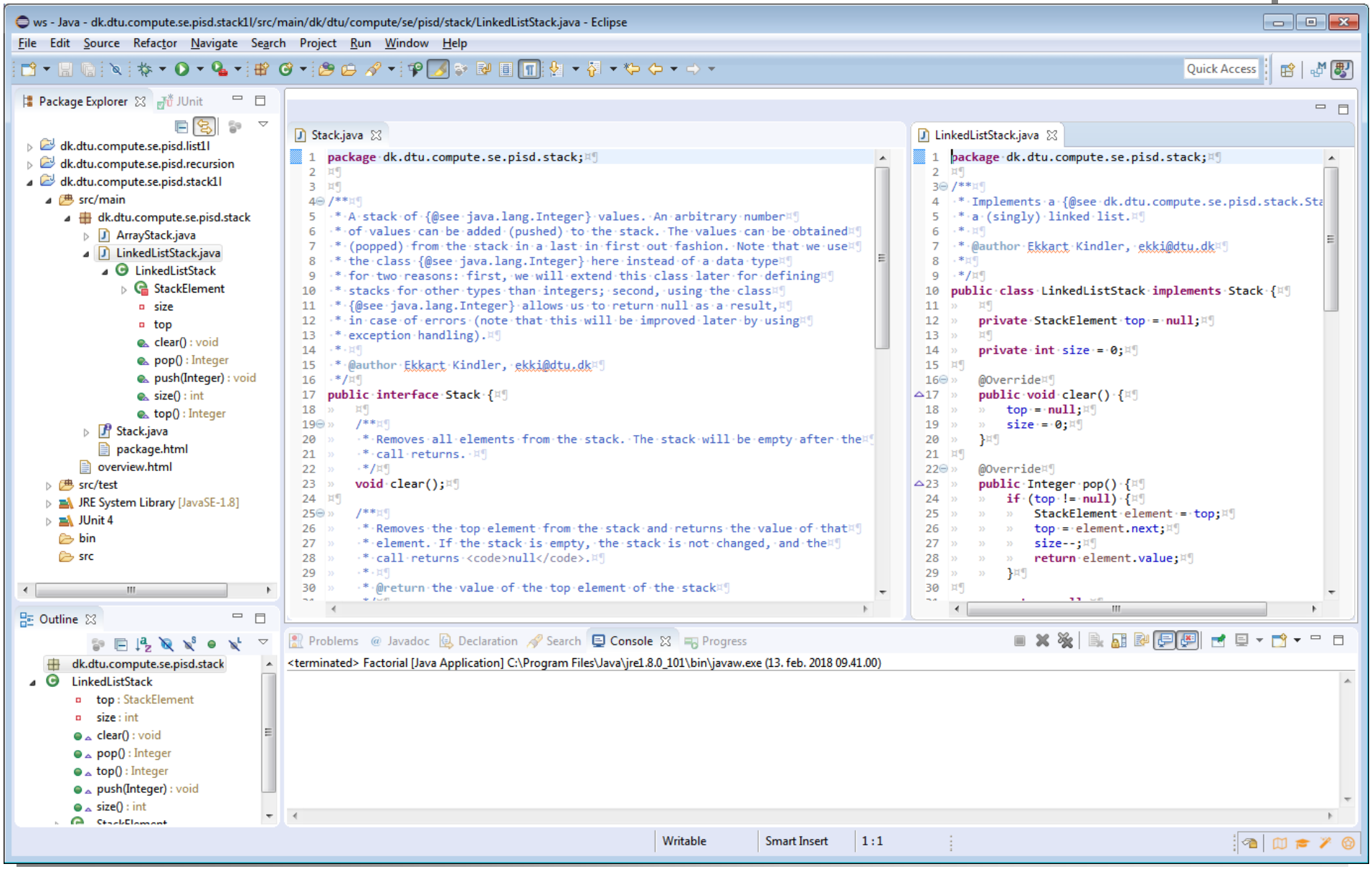
# pop()

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



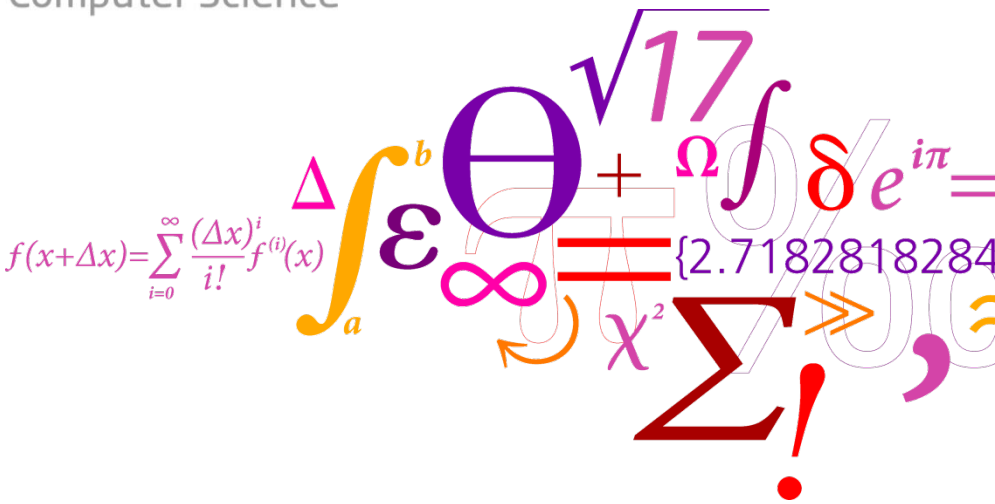
# JavaDoc

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



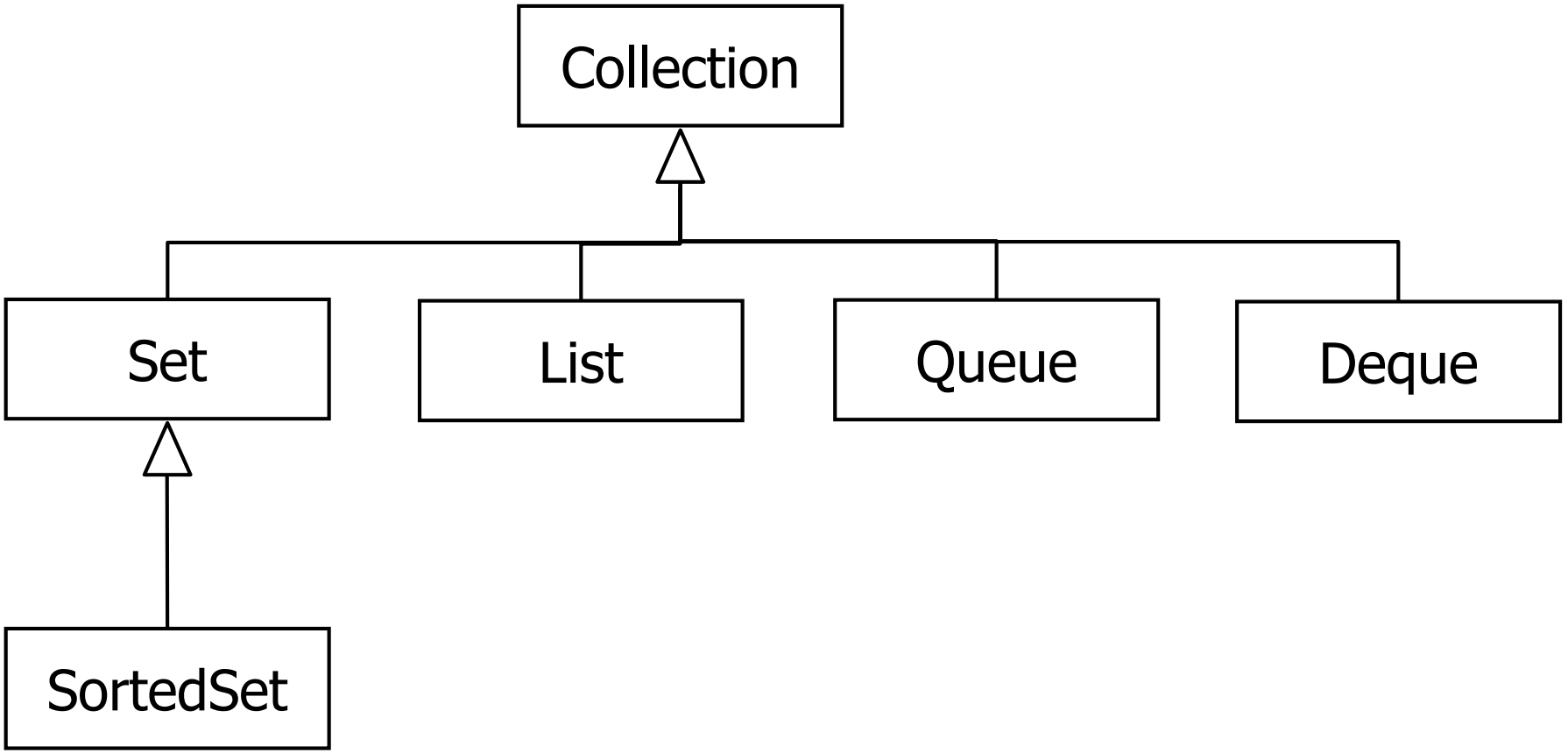
# Javas indbyggede datastrukturer

**DTU Compute**
Department of Applied Mathematics and Computer Science



# Collections overview

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



# Comparable

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public interface Comparable<T> {

    public int  compareTo(T o);

}
```

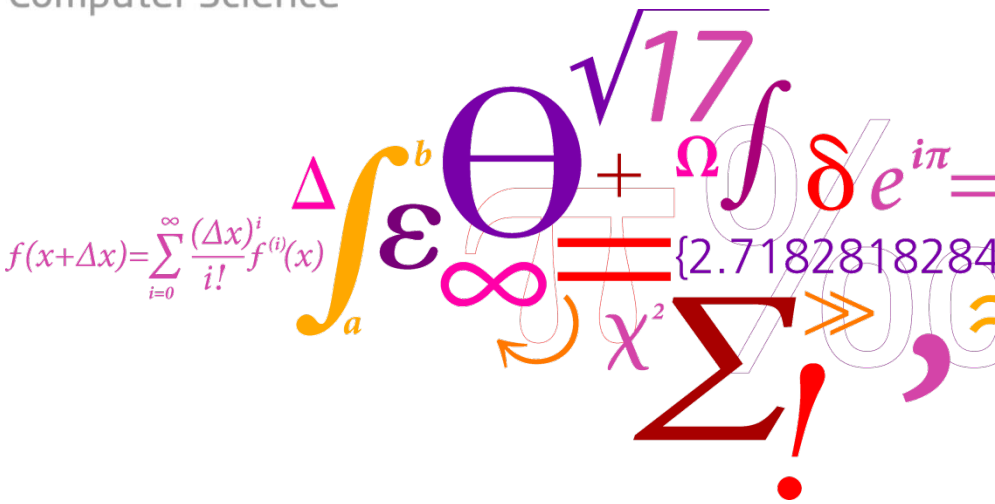
`o1.compareTo(o2) < 0` hvis object `o1` er mindre end object `o2`

`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



# Bubble sort

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Vi antager at listen ligger i et array

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

og

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

er den aktuelle længde af listen

# Bubble sort (1st version)

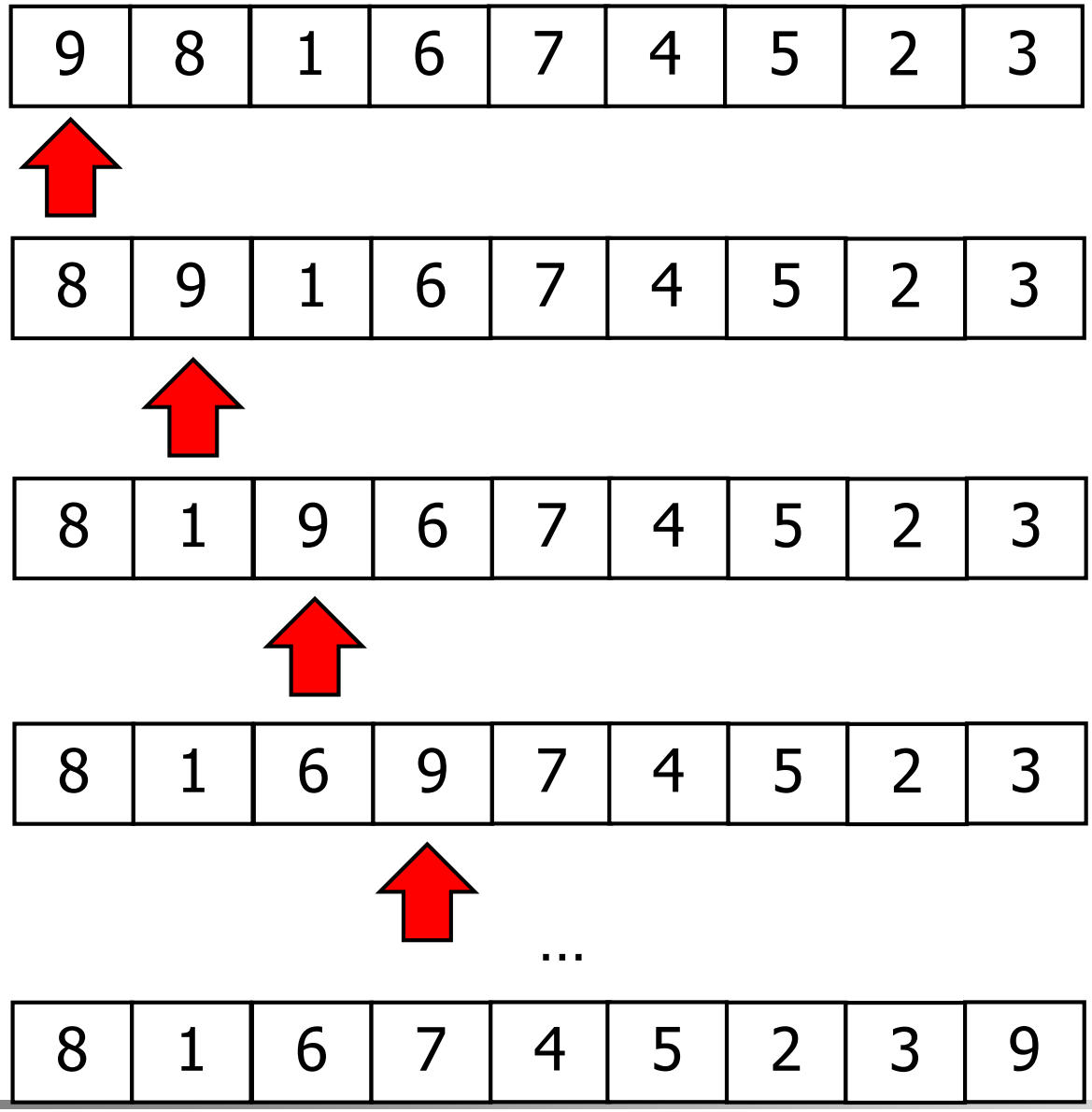
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
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.

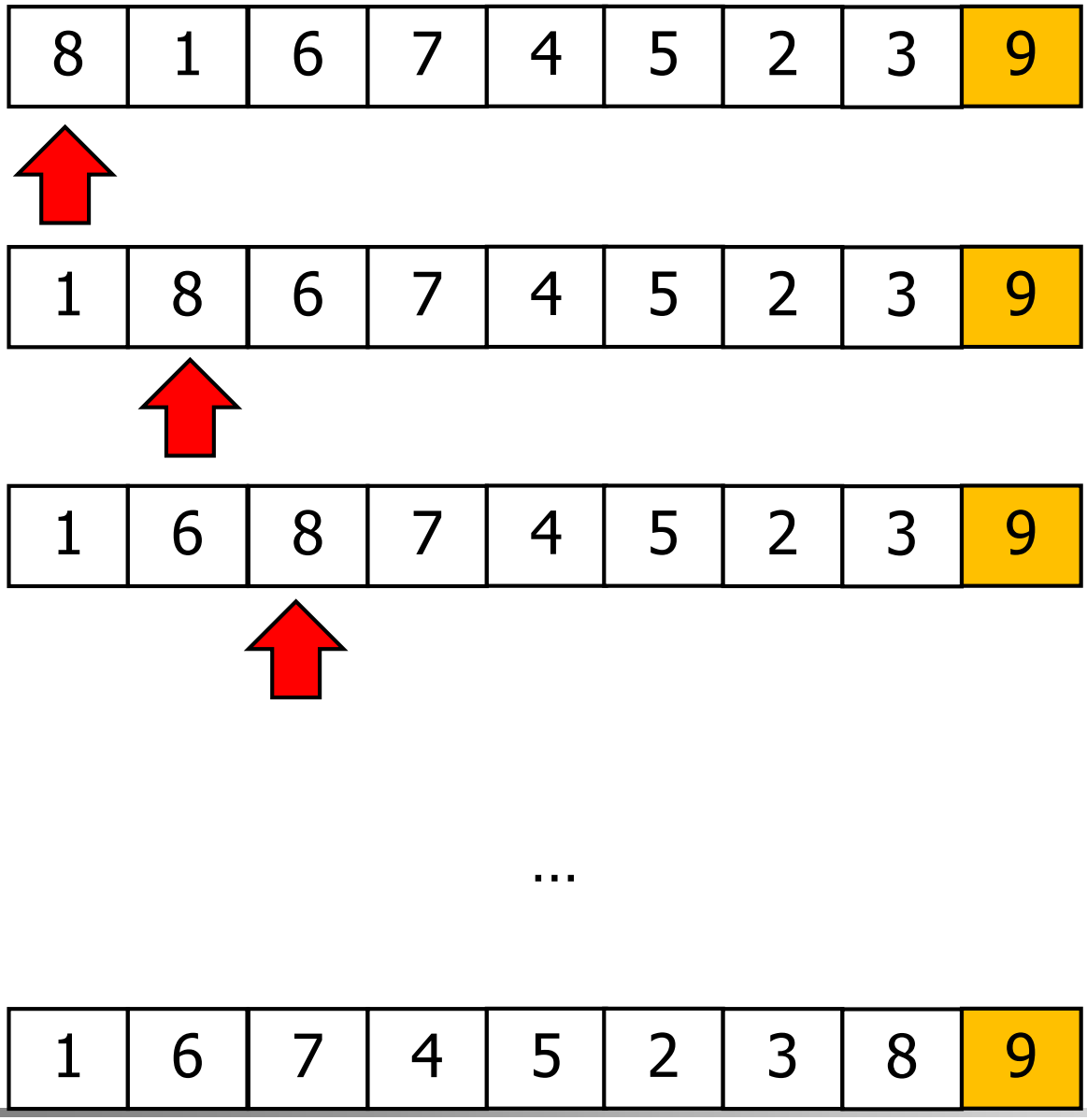
# Første iteration

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



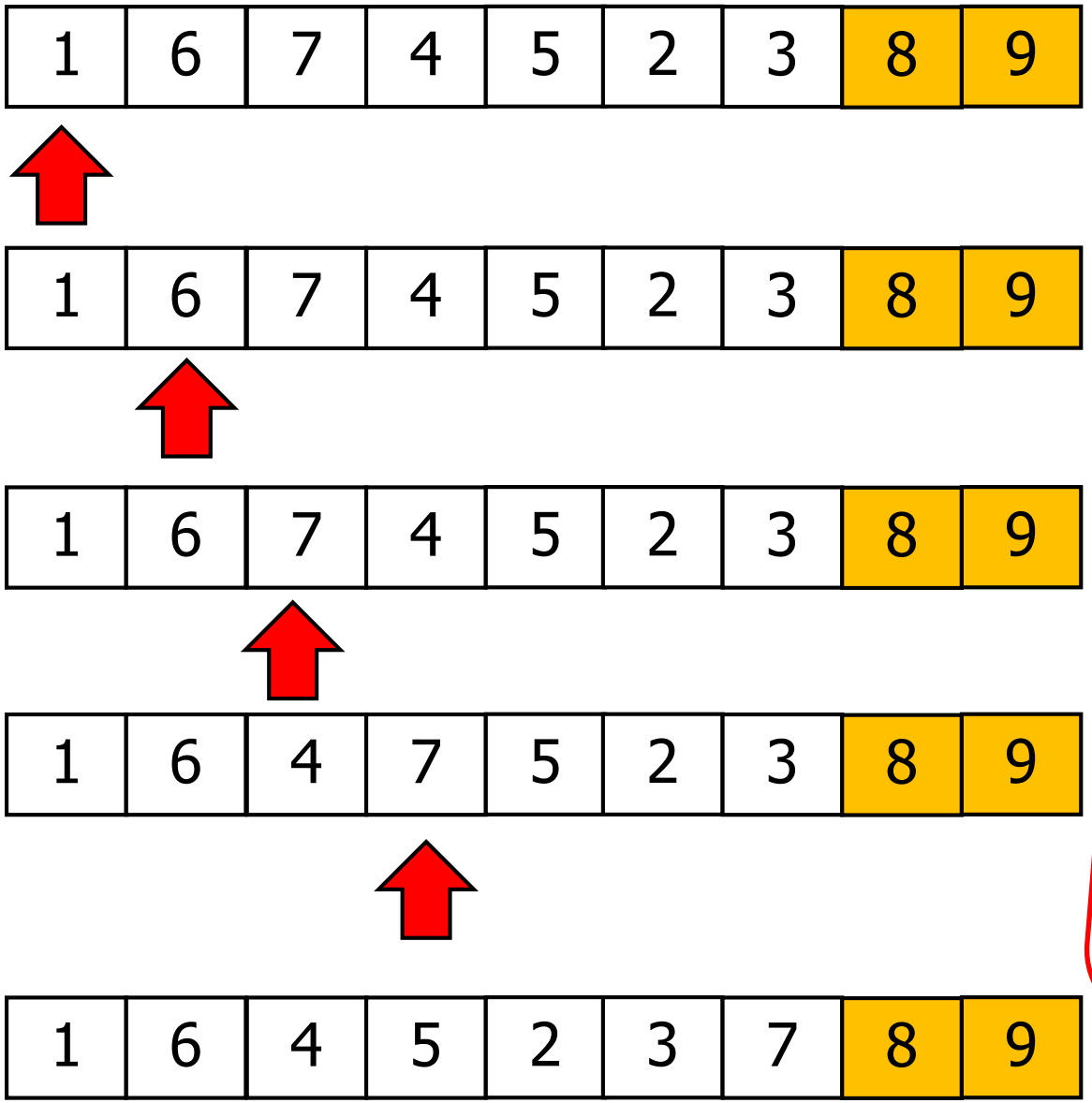
# Anden iteration

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



# Tredje iteration

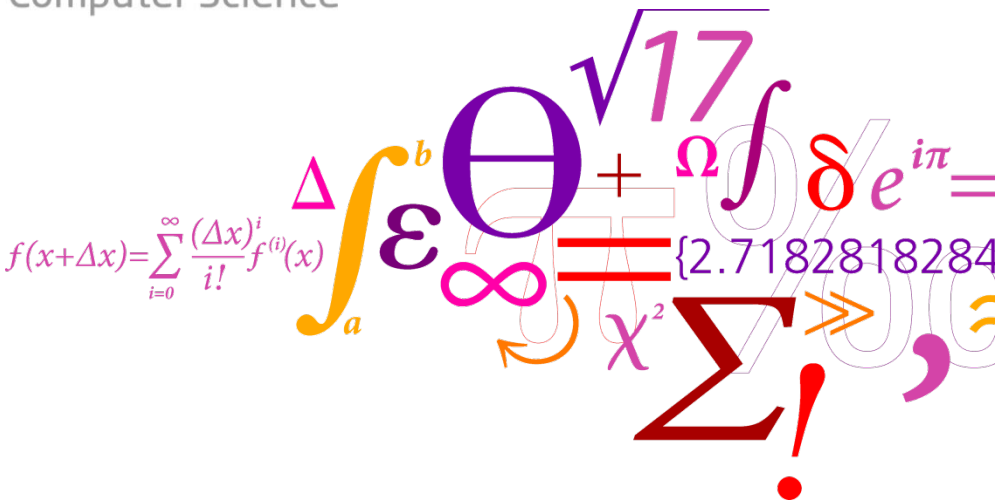
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

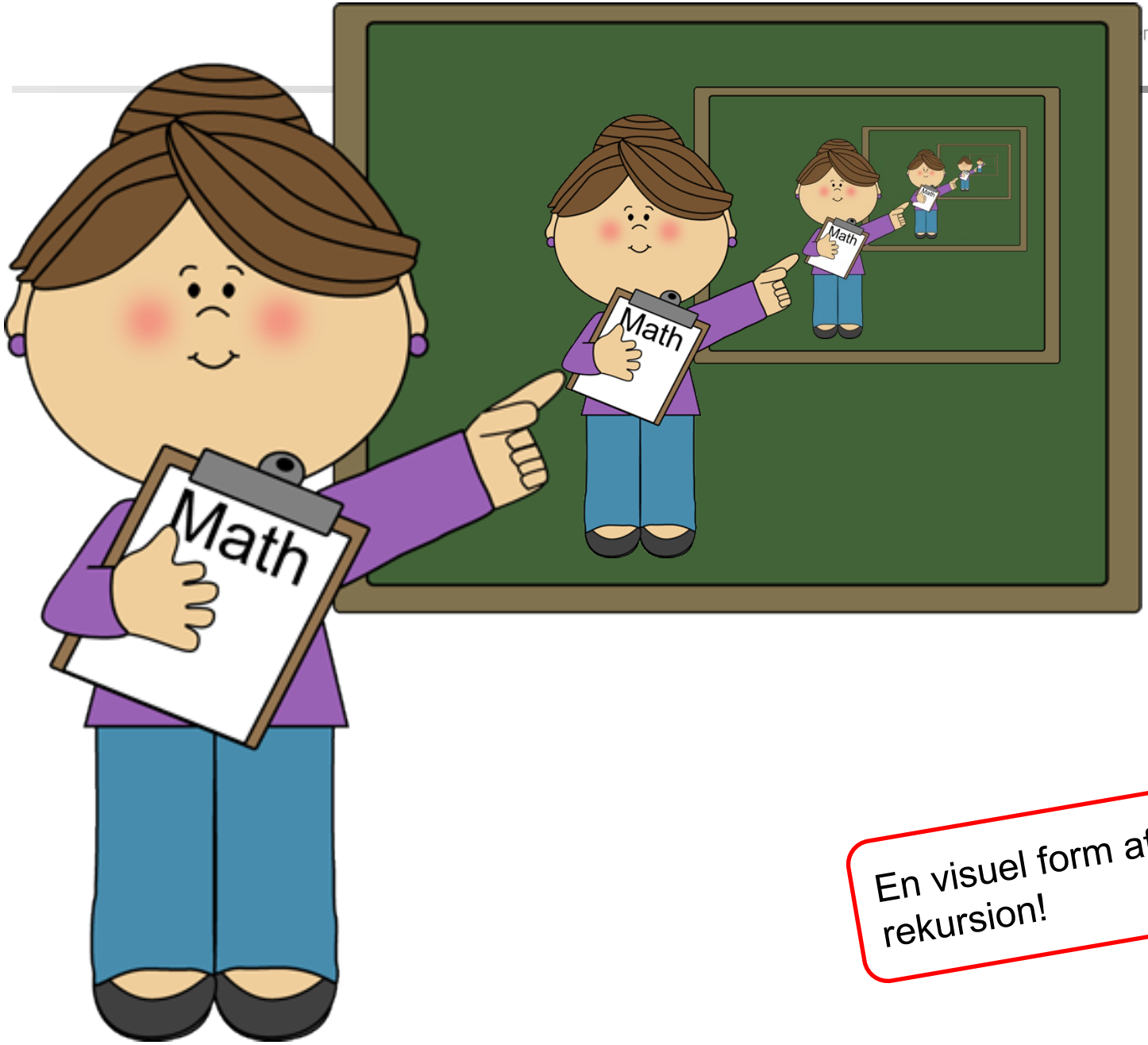



*I hver iteration kan vi stoppe en position tidligere!*

# Rekursion

**DTU Compute**
Department of Applied Mathematics and Computer Science






En visuel form af en rekursion!

# Rekursiv definition *n!*

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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

Her er der **rekursion** igen

# Fakultet-funktion i Java

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
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?

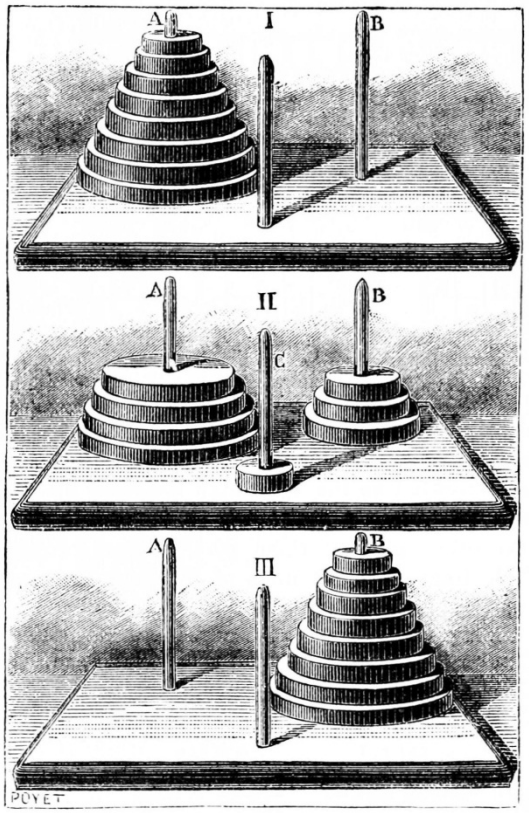
# Andet Eksempel



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

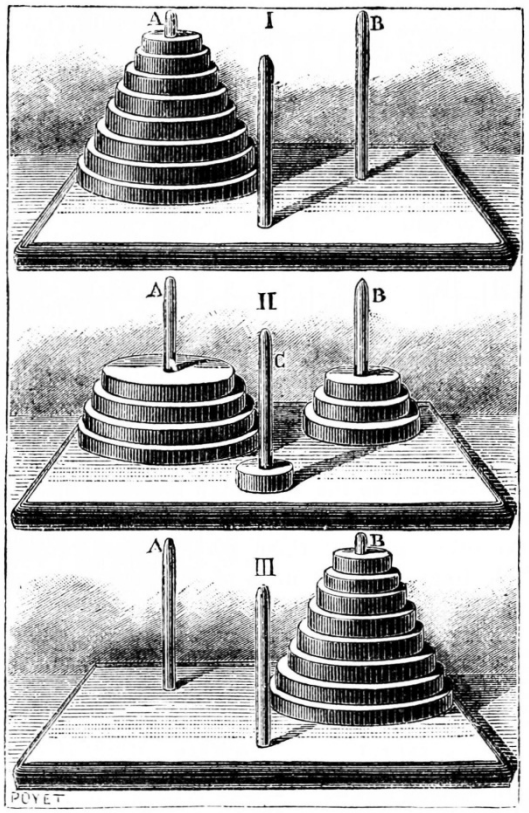


# Løsningsidé:

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Flyt *n* skiver fra *A* til *B*

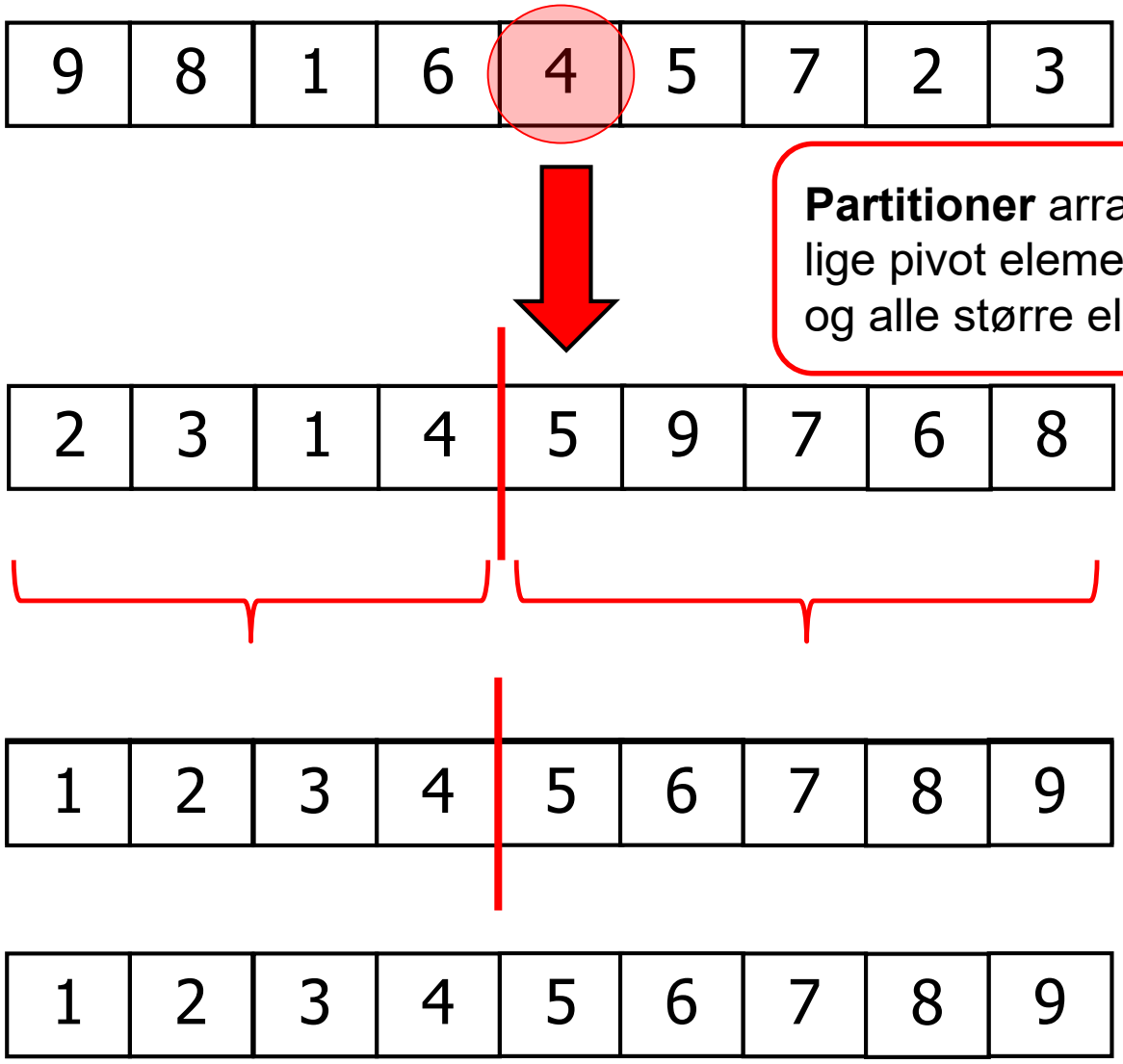
1. Flyt de øverste *n-1* skiver fra *A* til *C*
   (iføge reglerne)
2. Så fly den nedereste (størrste)
   skive *n* fra *A* til *B*
   (som er OK ifølge reglerne)
3. Nu flyt de *n-1* skiver fra *C* til *B*




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

# Quicksort (→ Hoare 1959)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler




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

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

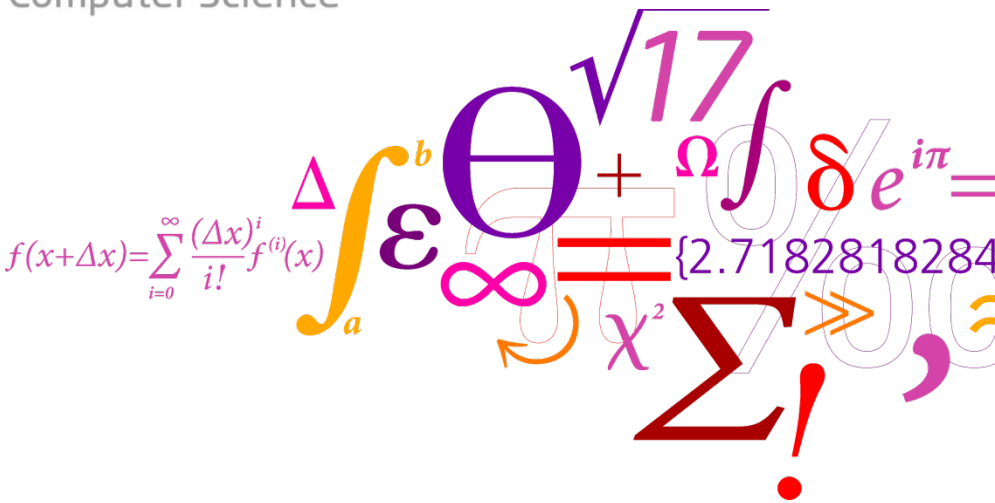
```
private void quicksort(int lower, int upper) {
  if (lower >= upper) {
    return;
  }

  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



# Eks.: Stak (interface)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
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)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
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.)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
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.)

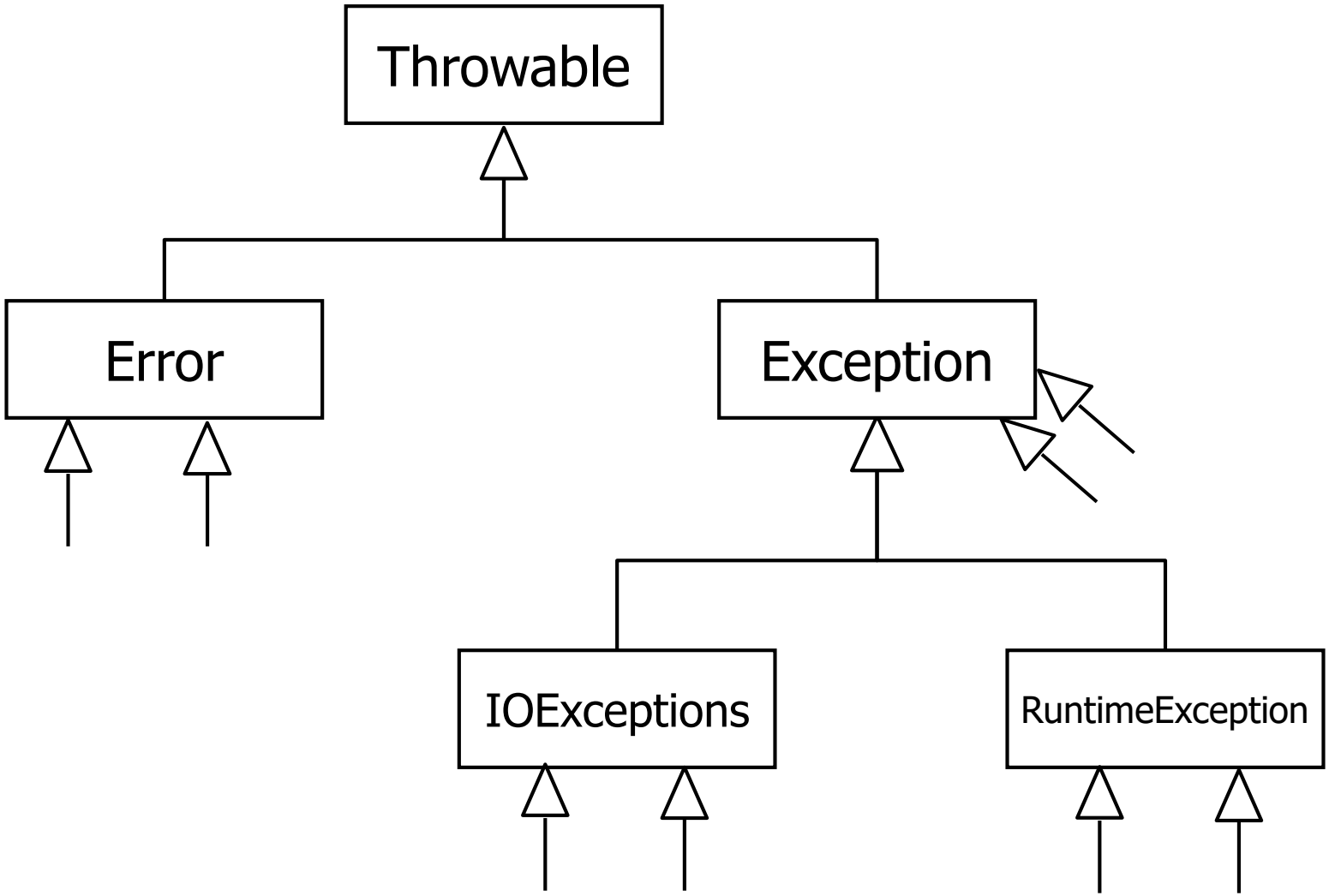
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
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

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



# *Overvågede* exceptions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- 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

# *Uovervågede* exceptions

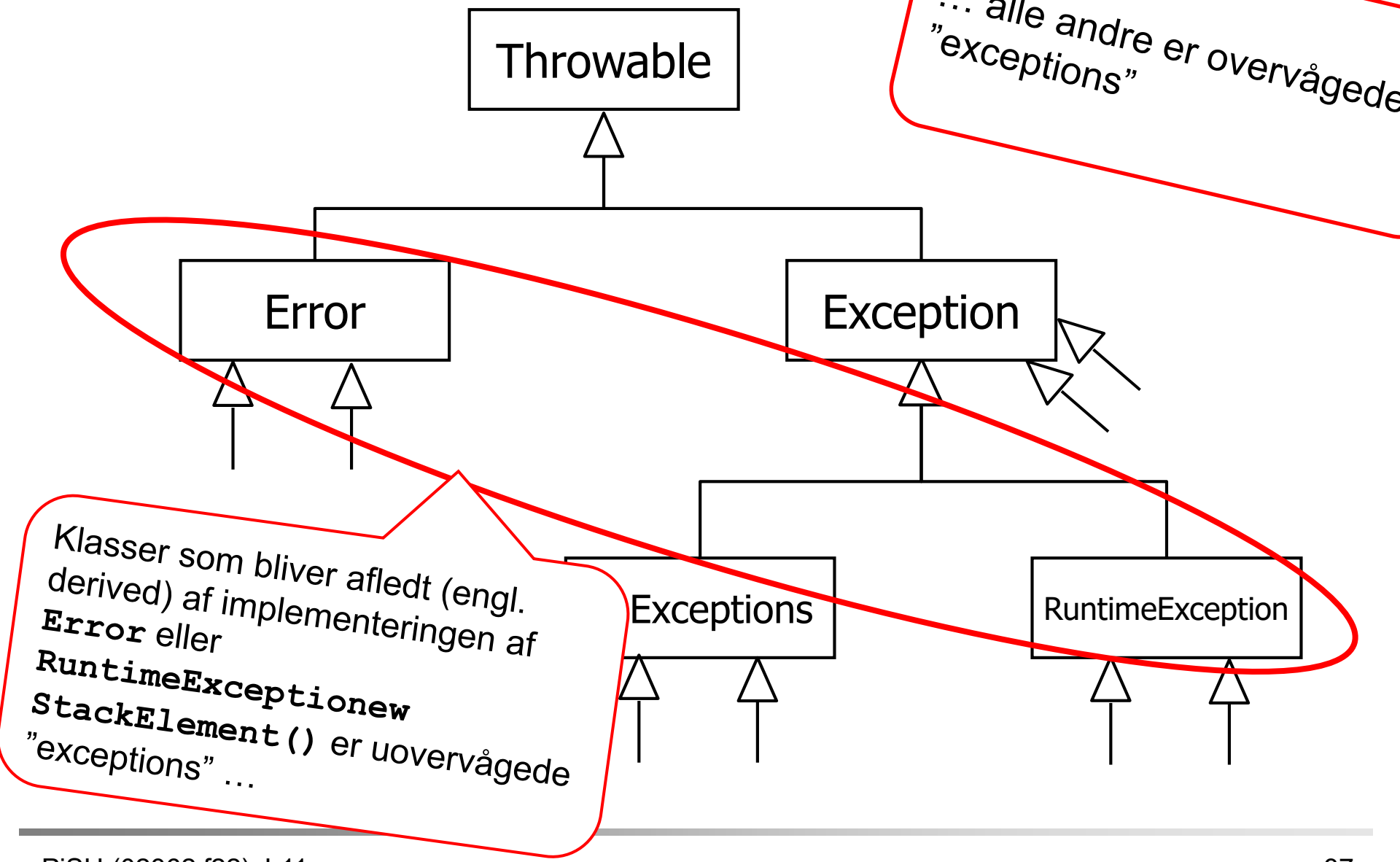
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- er exceptions, hvor den, som kalder metoden (*engl. caller*), ikke kan gøre noget ved
  typisk: programmeringsfejl i implementering
  (NullPointerException, ClassCastException, ArrayIndexOutOfBounds, 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

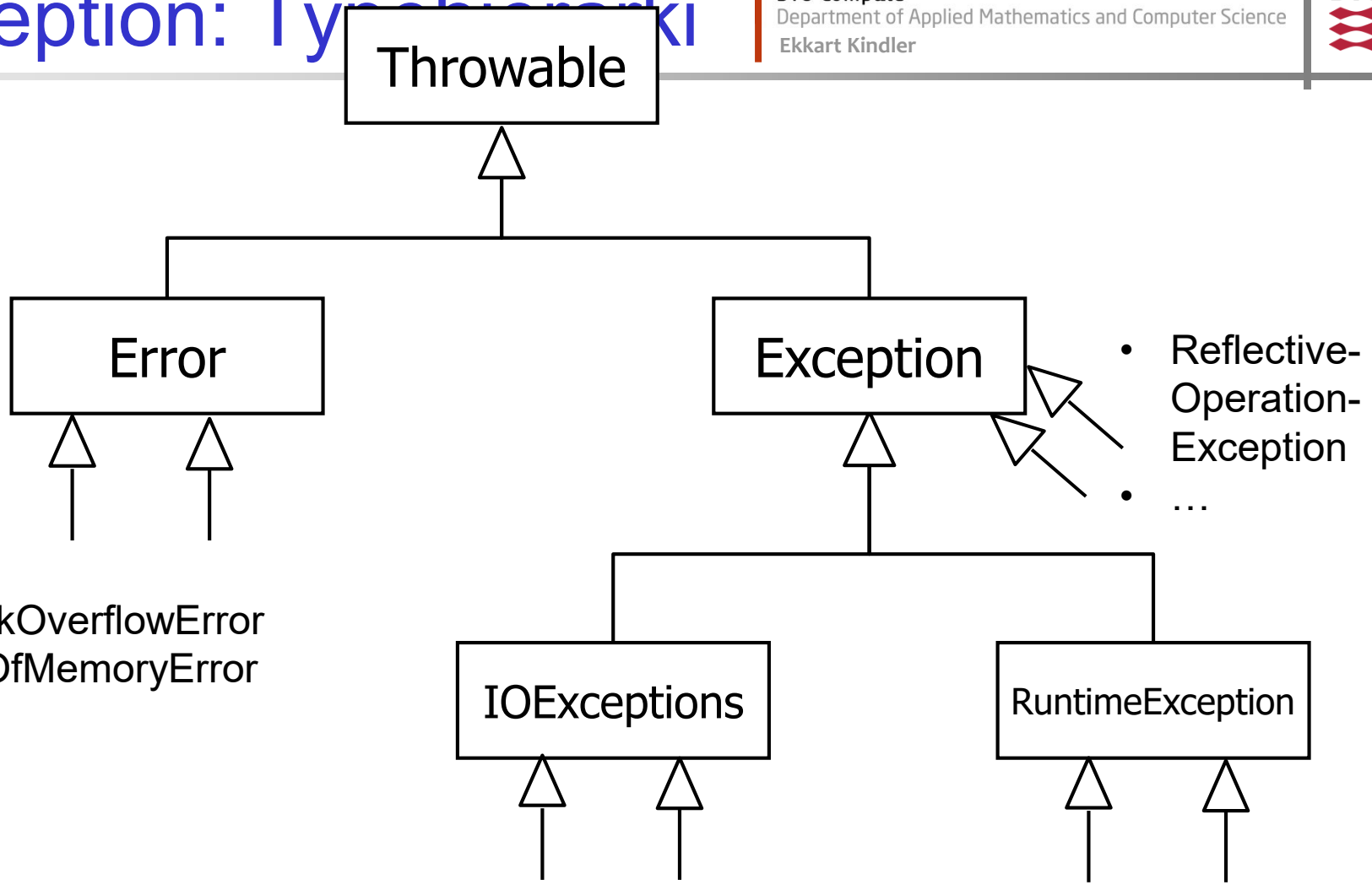
# Exception: Typehierarki





# Exception: Typhierarki

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler




- StackOverflowError
- OutOfMemoryError
- …

- Reflective-Operation-Exception
- …

- FileNotFoundException
- EOFException
- …

- ClassCastException
- IllegalArgumentException
- NullPointerException
- IndexOutOfBoundsException
- …

# Kontrolflow Exceptions



```
try {
   ...
   do something;
   callSomeone();
   ...
} catch (Expeption1 e) {
   ...
   do something
} catch (Expeption2 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

# Kontrolflow Exceptions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
try {
   ...
   do something;
   callSomeone();
   ...
} catch (Exeption1 e) {
   ...
   do something
} catch (Exeption2 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

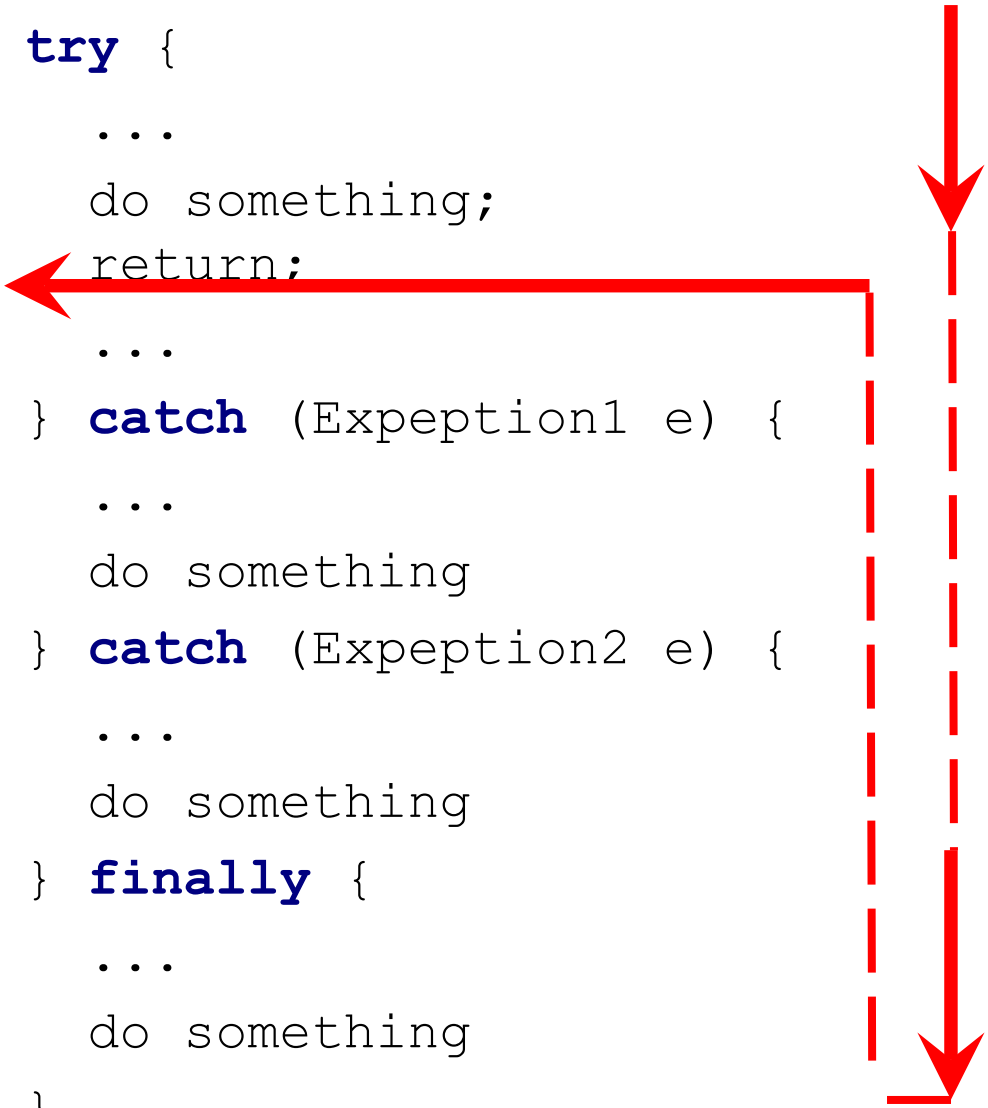
og den første catch-blok som matcher (er super klasse af) den udløste exception eksekveret (men kun en!)

og finally-bokken (hvis den er der) bliver også ekseveret!

# En krølle: finally/return

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

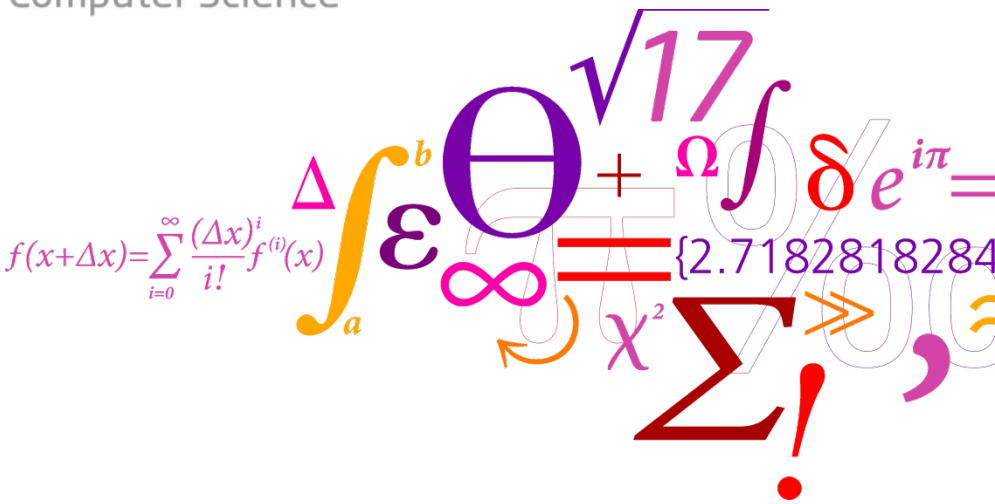
```
try {
   ...
   do something;
   return;
   ...
} catch (Exeption1 e) {
   ...
   do something
} catch (Exeption2 e) {
   ...
   do something
} finally {
   ...
   do something
}
```



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



# Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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

# Hvordan gør man det

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

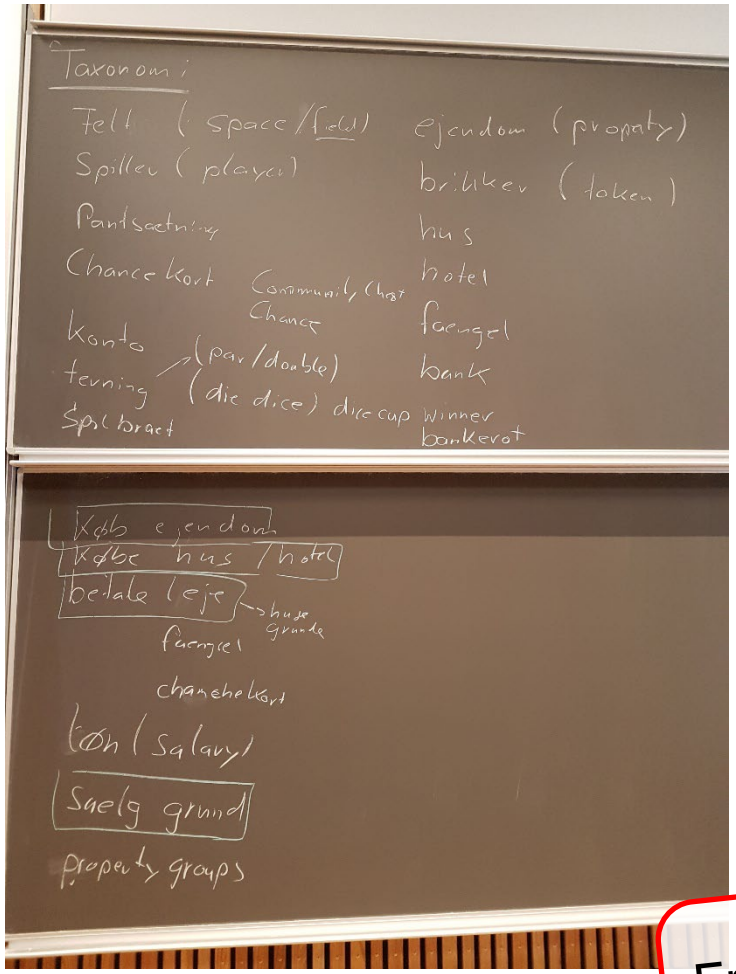
- Find skriftlig dokumentation om området (domænet) og skriv det ud
- Marker alle begreber som synes relevante
- Tal med nogle eksperter fra domænet

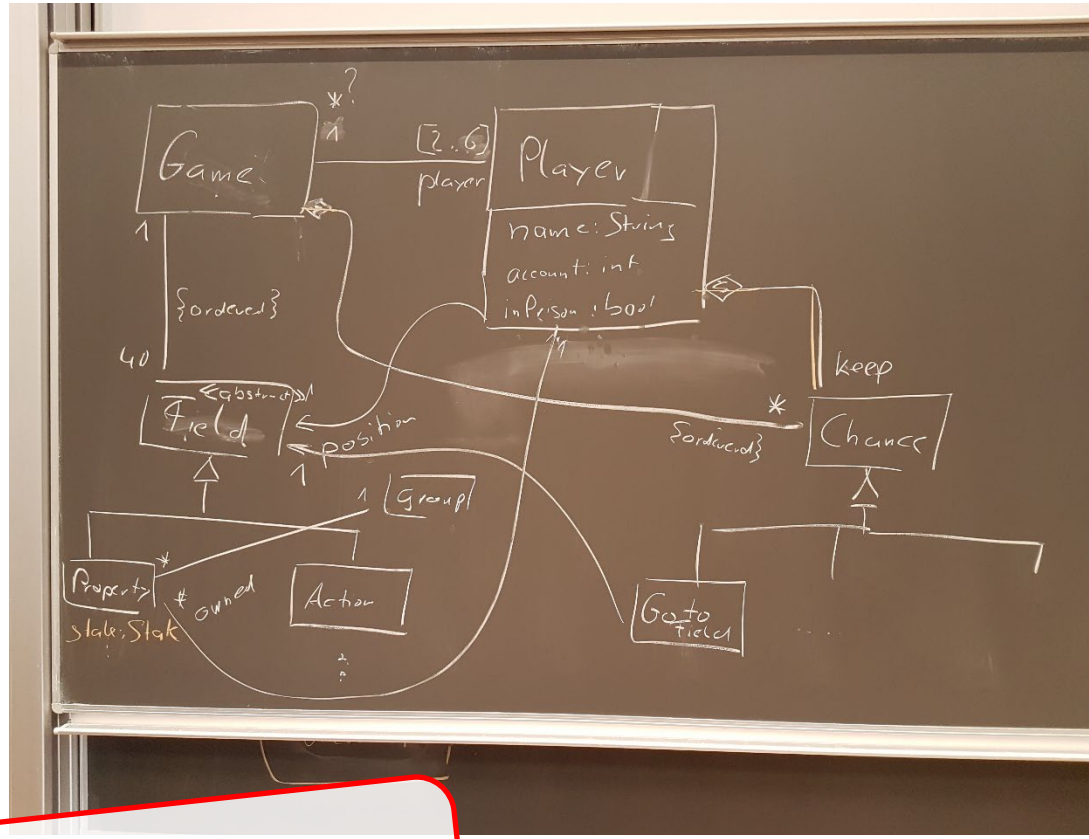
→ **Taksonomi**: liste af begreber (ord)

→ **Glossar**: Beskrivelse af begreber deres egenskaber og hvordan de forholder sig til hinanden

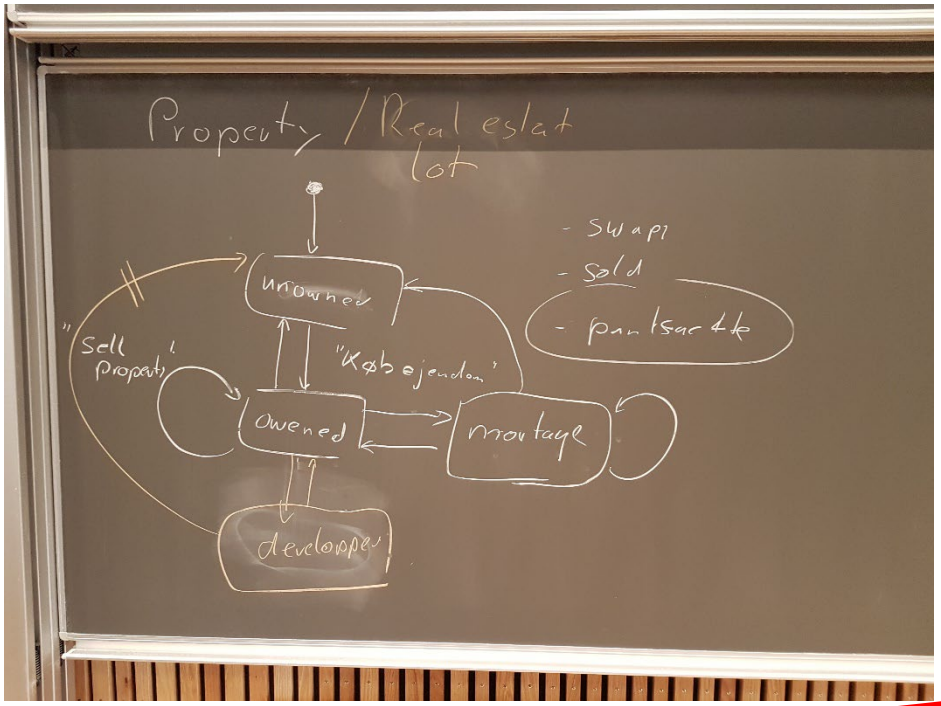
→ **Domænemodel**: Begreber, deres attributter og deres relation repræsenteret som klassediagram

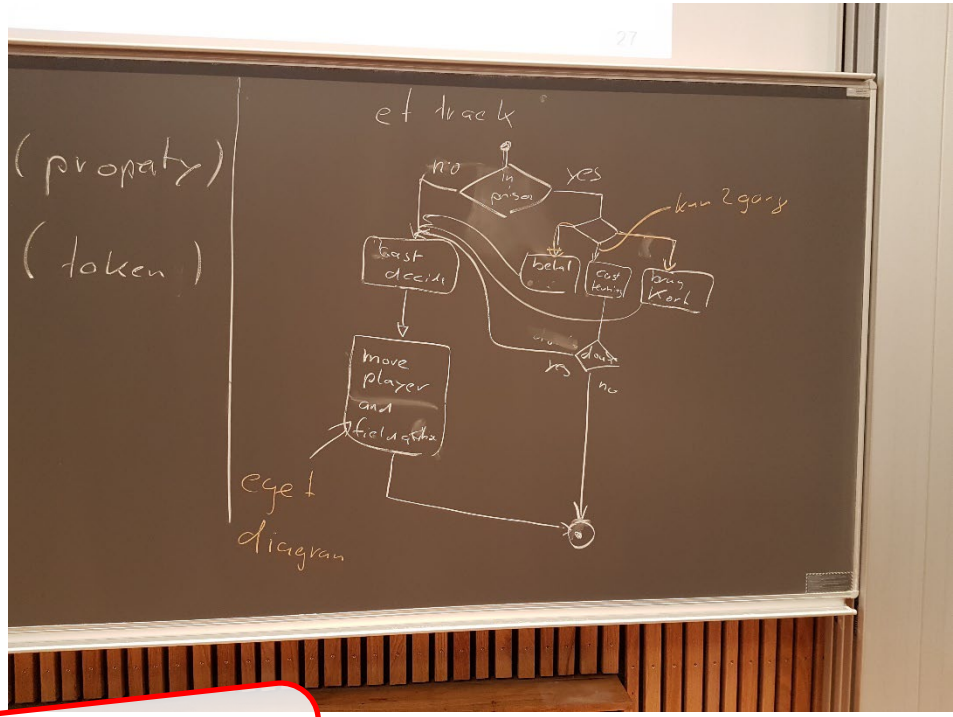
Dette klassediagram har ikke noget med programmering at gøre (endnu). Tænk kun på begreber og deres relation!




Fra 2019:
Monopoly/Matador!



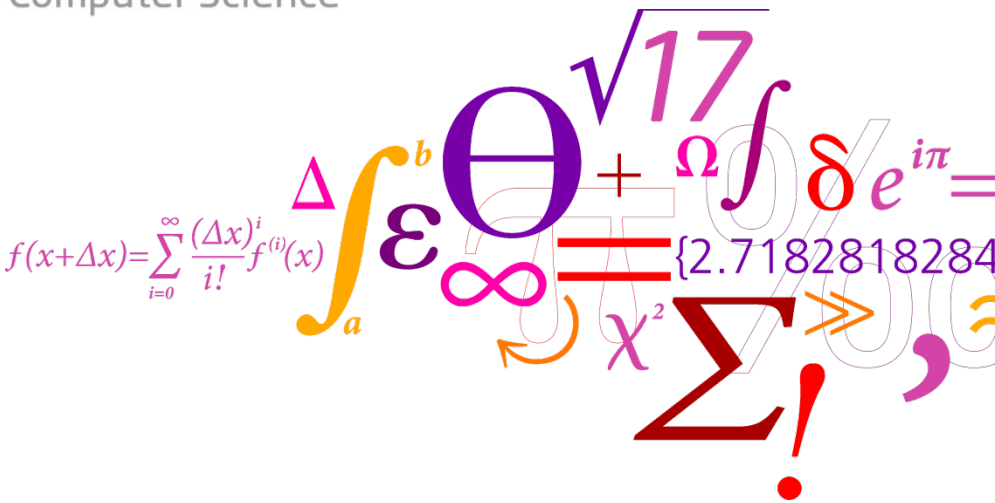




Fra 2019:
Monopoly/Matador!

# (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



# 1. Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

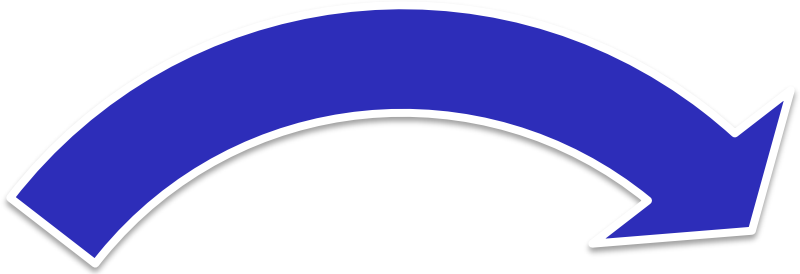
**Analysen** drejer sig **kun** om **HVAD** (engl. **WHAT**) softwaren skal gøre (ud fra brugerens synsvinkel)

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

# Co-evolution

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

"**Hvad**" skal softwaren gøre?



WHAT HOW

"**Hvordan**" er softwaren realiseret?

# Design patterns

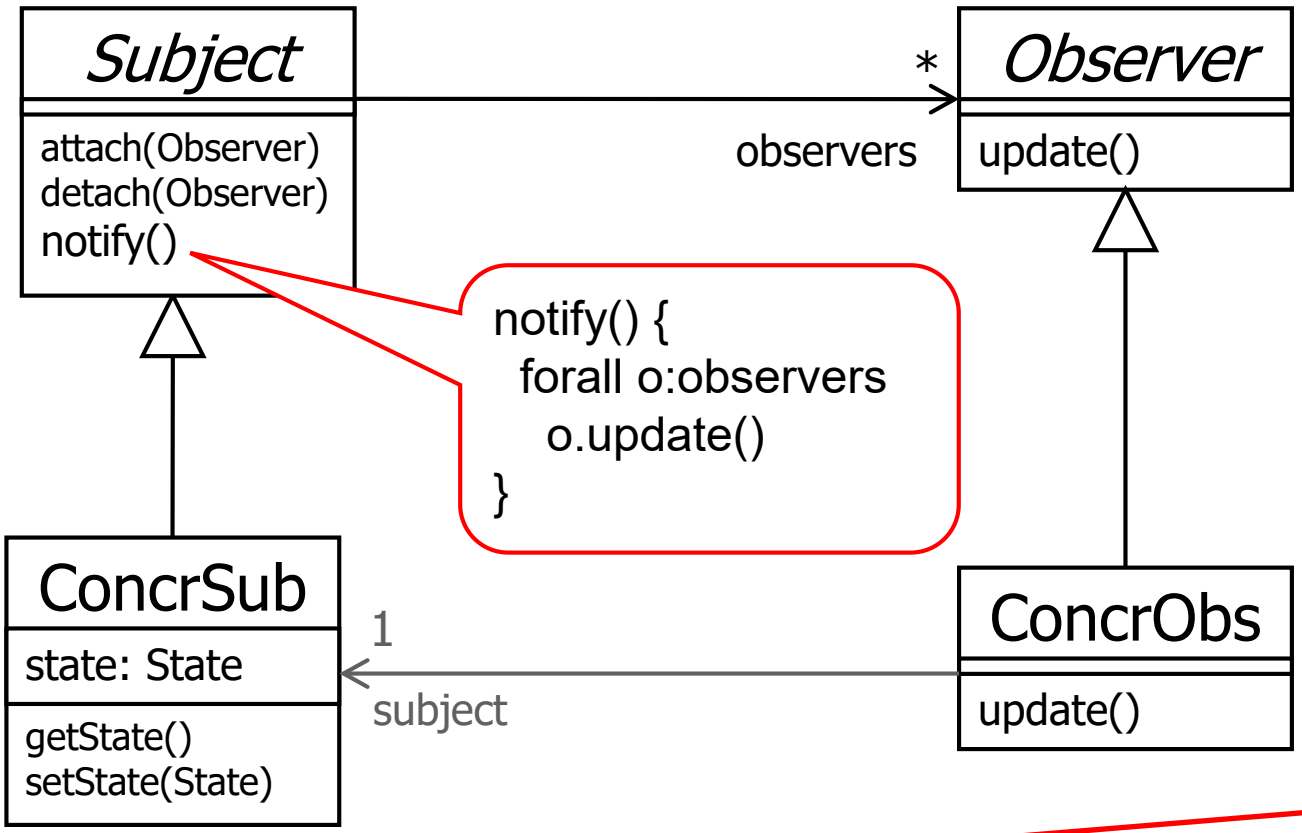
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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.

# Eksempel: Observer

## Structure




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

# Model View Controller (MVC)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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:

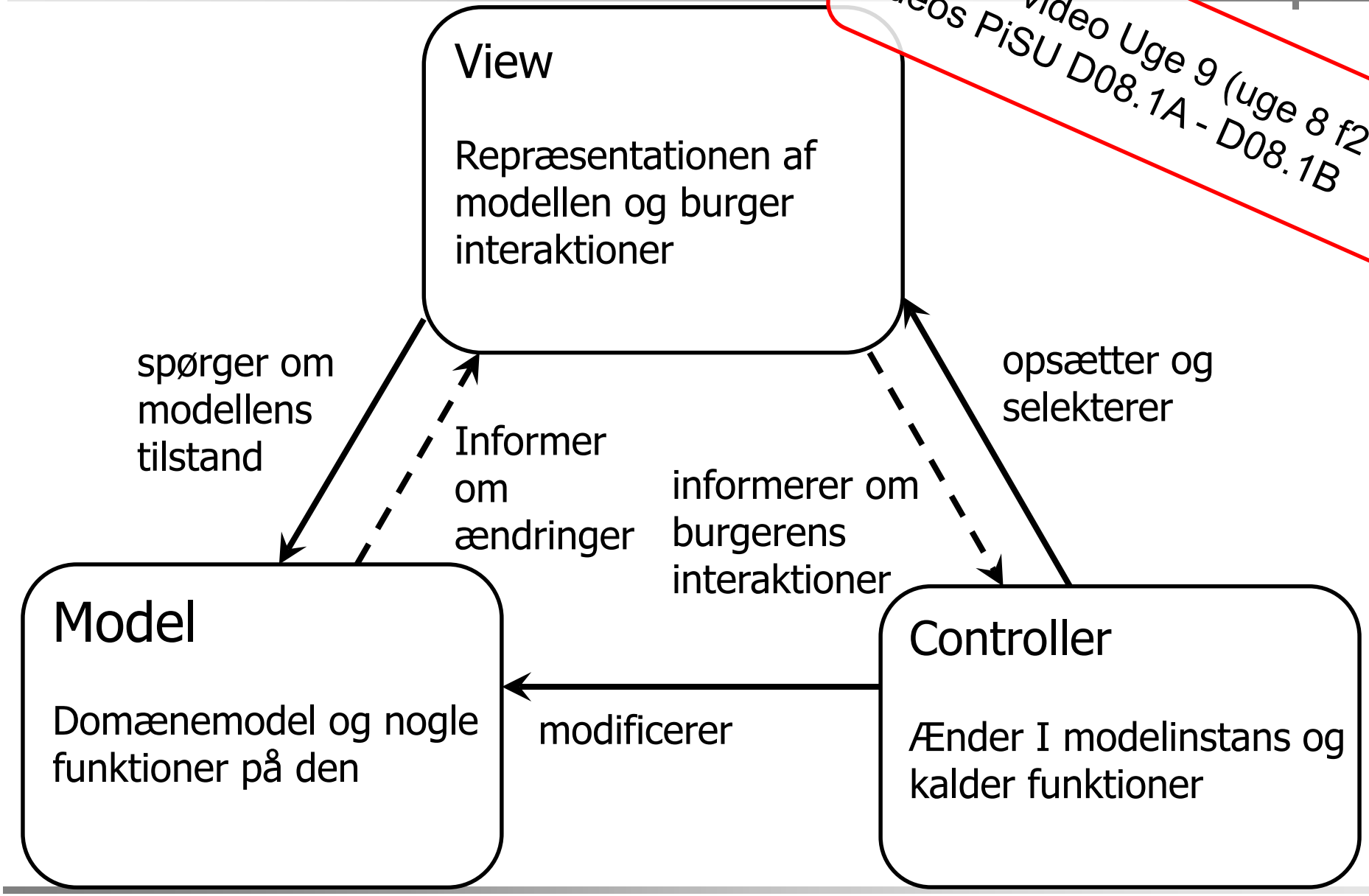
- Delen som viser modellens information til brugeren (**view**)
- Delen som koordinerer med brugeren og implementerer logikken bagved (**controller**)

**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".

# MVC



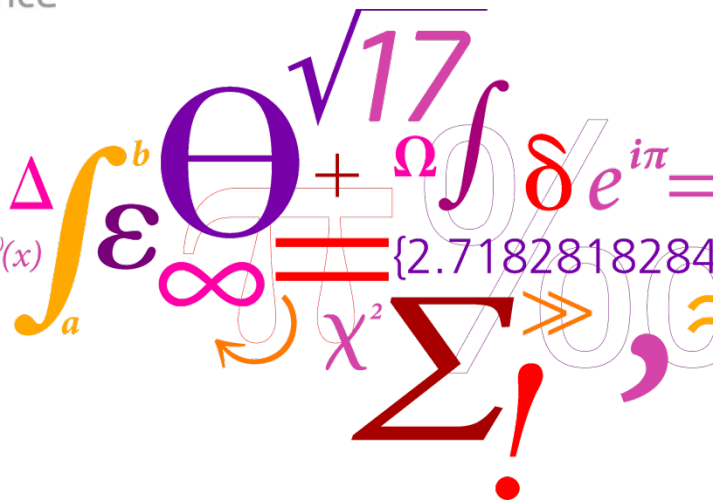
Se også video Uge 9 (uge 8 f21)
videos PiSU D08.1A - D08.1B



# Læse og skrive filer, Gson

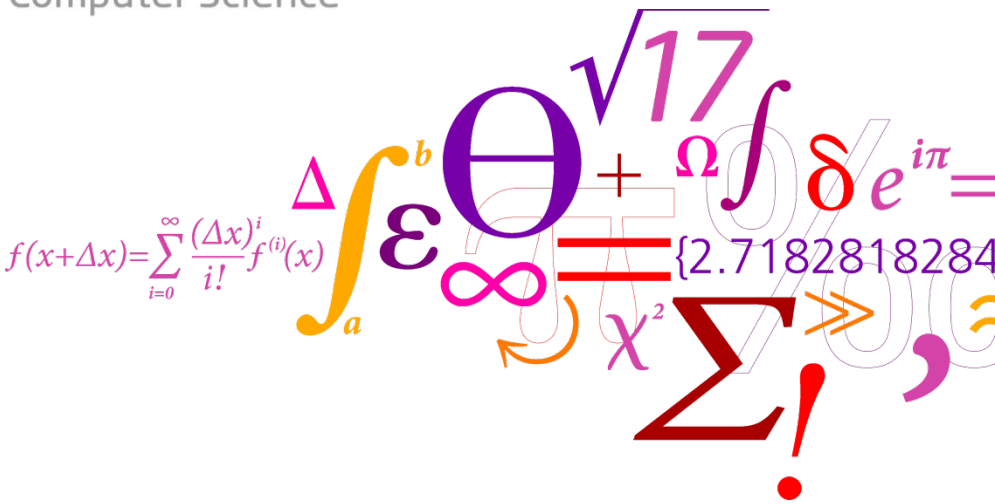
**DTU Compute**
Department of Applied Mathematics and Computer Science

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



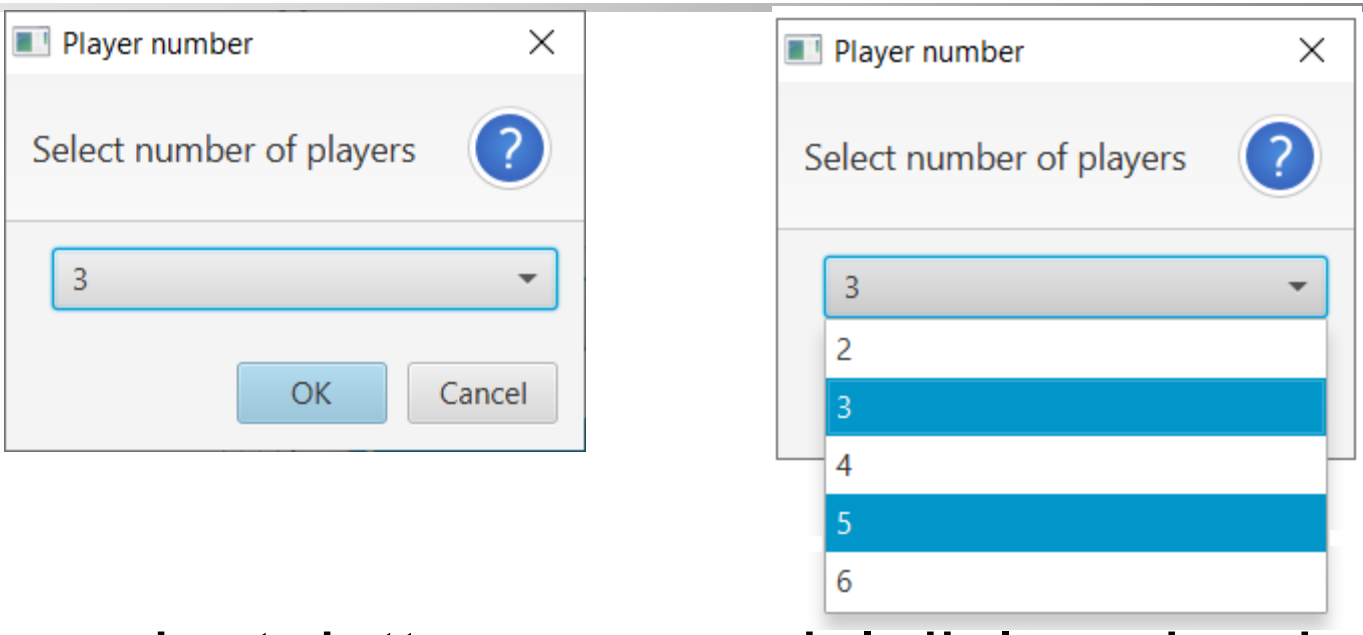
# Dialoger og inputvalidering

**DTU Compute**
Department of Applied Mathematics and Computer Science



# Eksempel: ChoiceDialog

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler




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

# Regulære Udtryk

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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

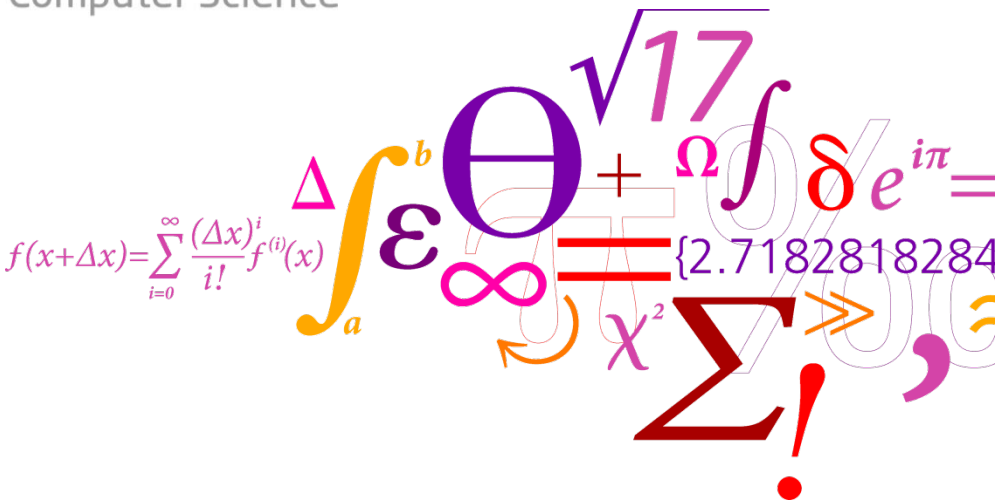
- Adresser
- Email-addresser
- Telefonnumre
- Datoer

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



```
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`