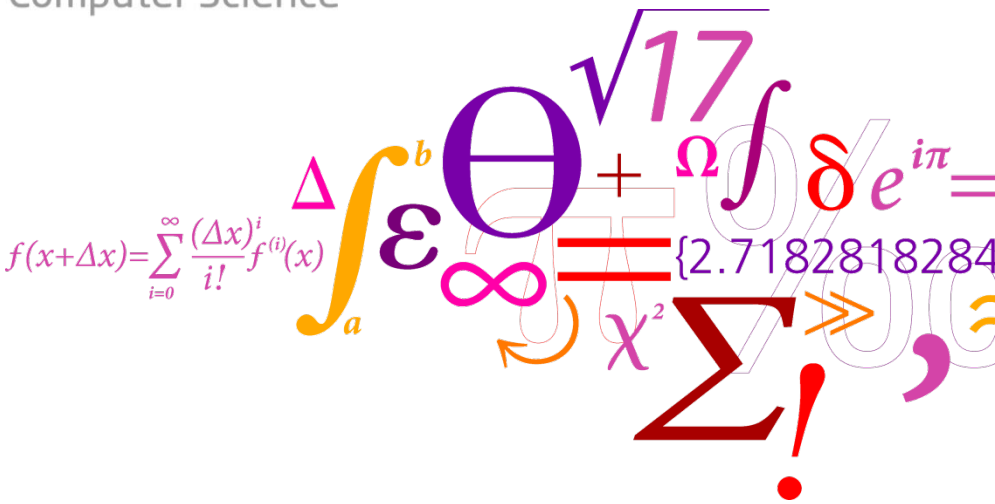
Projekt i software-udvikling (02362)
Forår 2022

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science



DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Motivation

- Implementeringer i programmer er ofte meget teknisk
- Der er mange data 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

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Vi har brug for mechanismer som hjælper os med at

- fokussere på de vigtige ting:

  Abstraktion

- Gemmer implementeringsdetailjer

  Hiding

Eksempel: Interface `Stack`

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

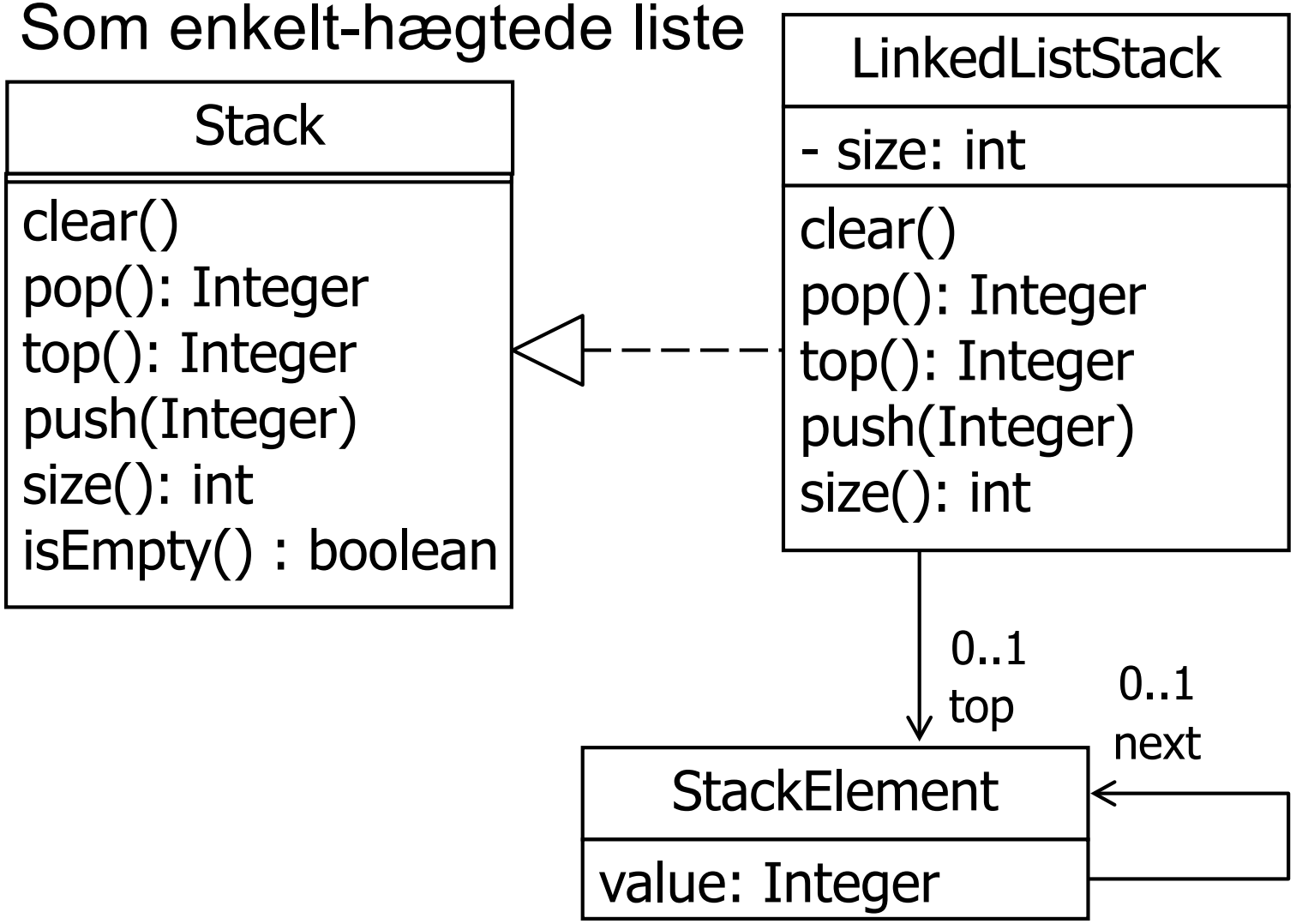
```
public interface Stack {

  void clear();

  Integer pop();

  Integer top();

  void push(Integer value);

  int size();

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

En stak af heltals-objekter (`Integer`)

Implementering

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Som enkelt-hægtede liste



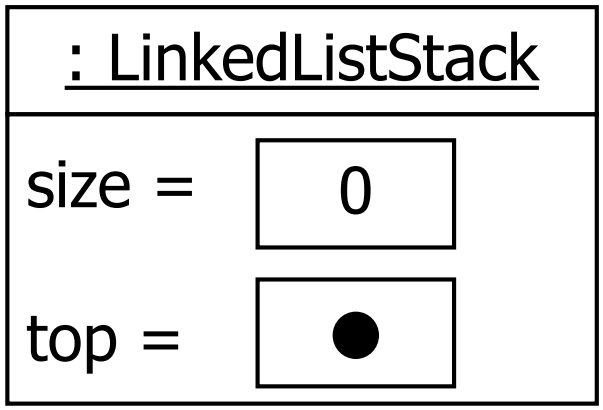
```
public class LinkedListStack implements Stack {

   private StackElement top = null;

   private int size = 0;

   public void clear() {
      top = null;
      size = 0;
   }

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

      return null;
   }
```

```
public Integer top() {
    if (top != null) {
        return top.value;
    }

    return null;
}

public void push(Integer value) {
    StackElement newElement = new StackElement(value, top);
    size++;
    top = newElement;
}

public int size() {
    return size;
}
```

```
    private class StackElement {

        private Integer value;

        private StackElement next;

        private StackElement(Integer value, StackElement next) {
            this.value = value;
            this.next = next;
        }
    }

}
```

Instans (”Objektdiagram”)

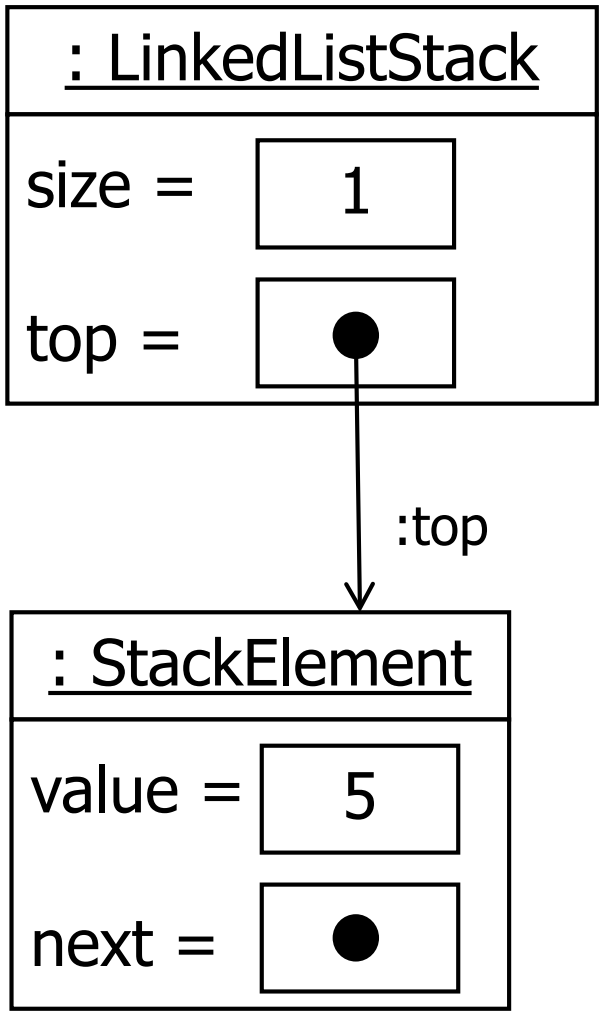
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Tom stak

Instance ("Objectdiagram")

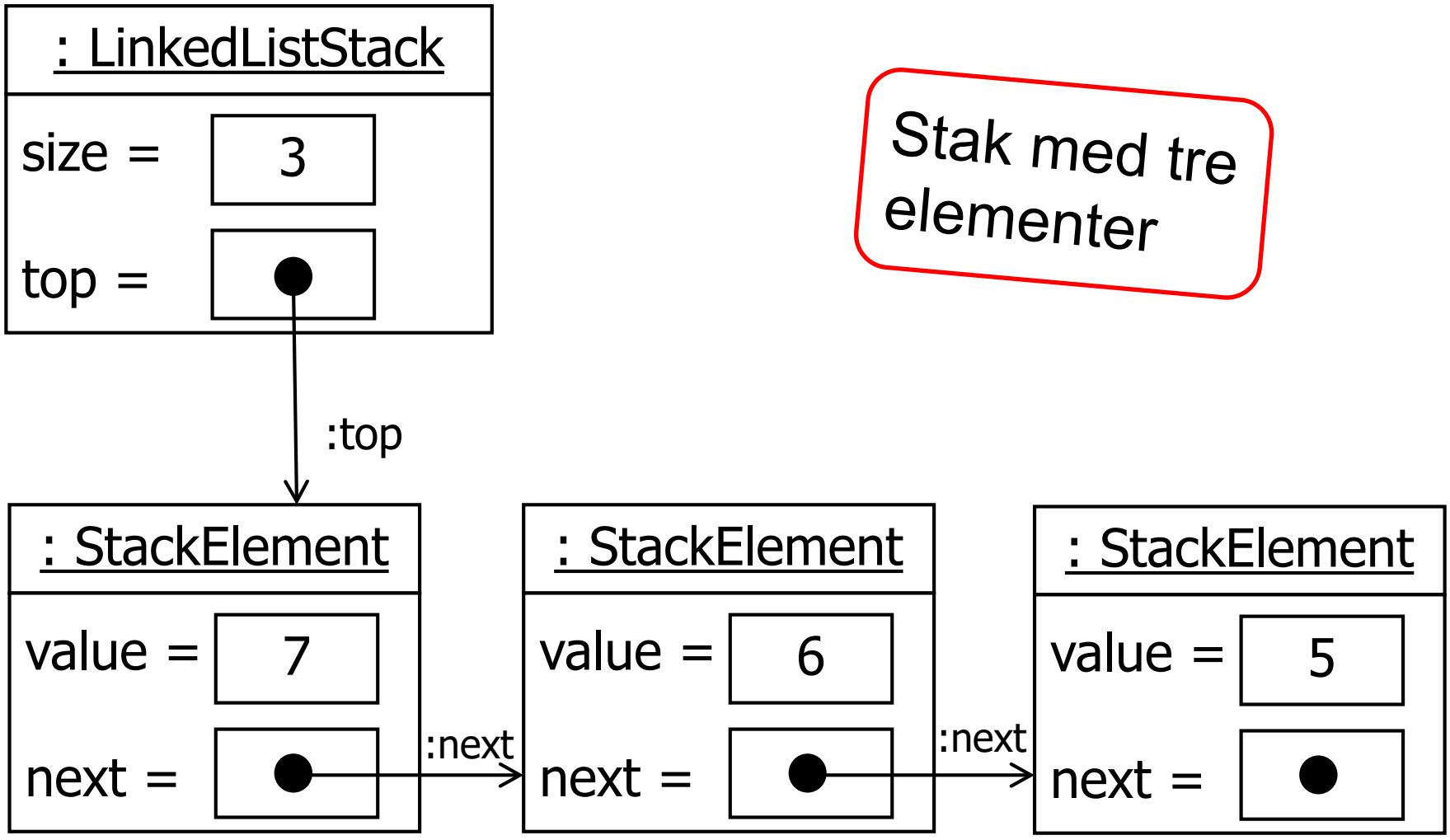
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Stak med et element

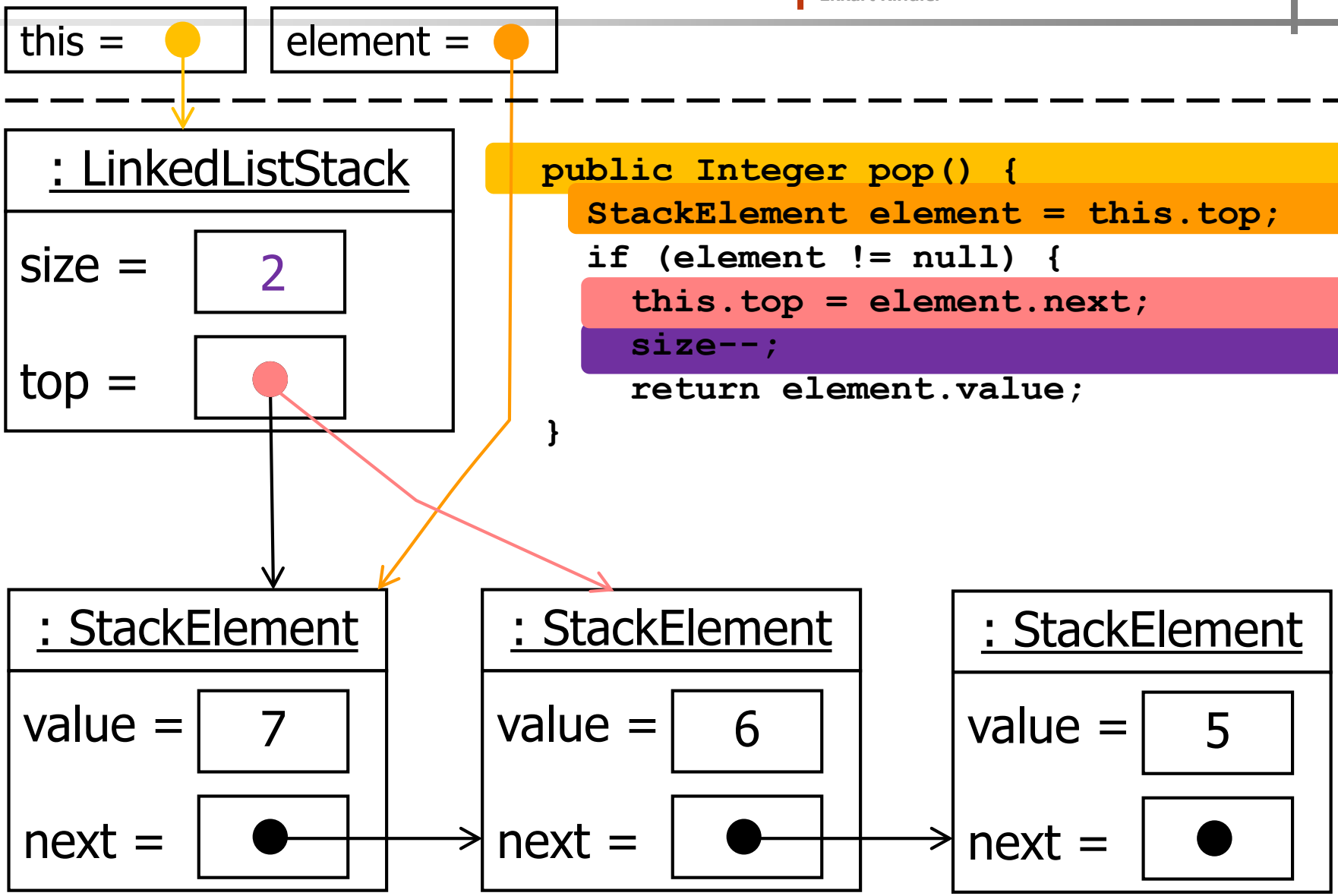
Instance (”Objectdiagram”)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



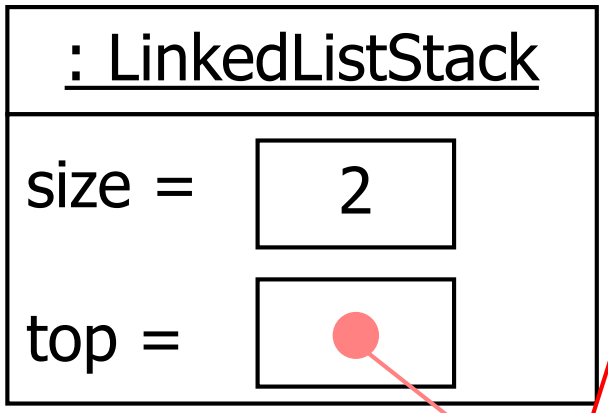
pop()

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

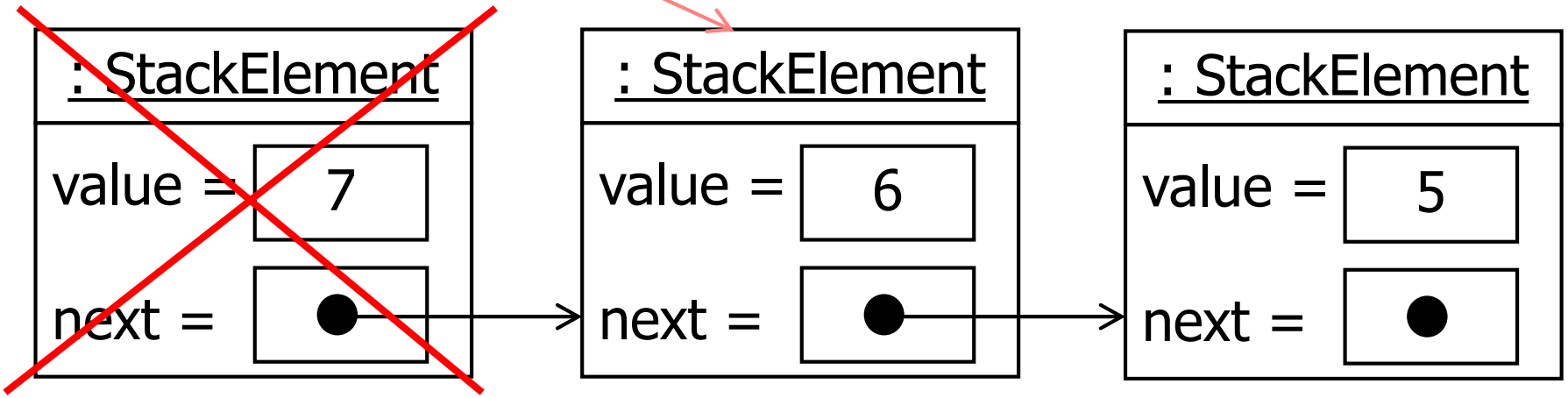


pop()

Dette objekt vil blive "garbage collected" senere

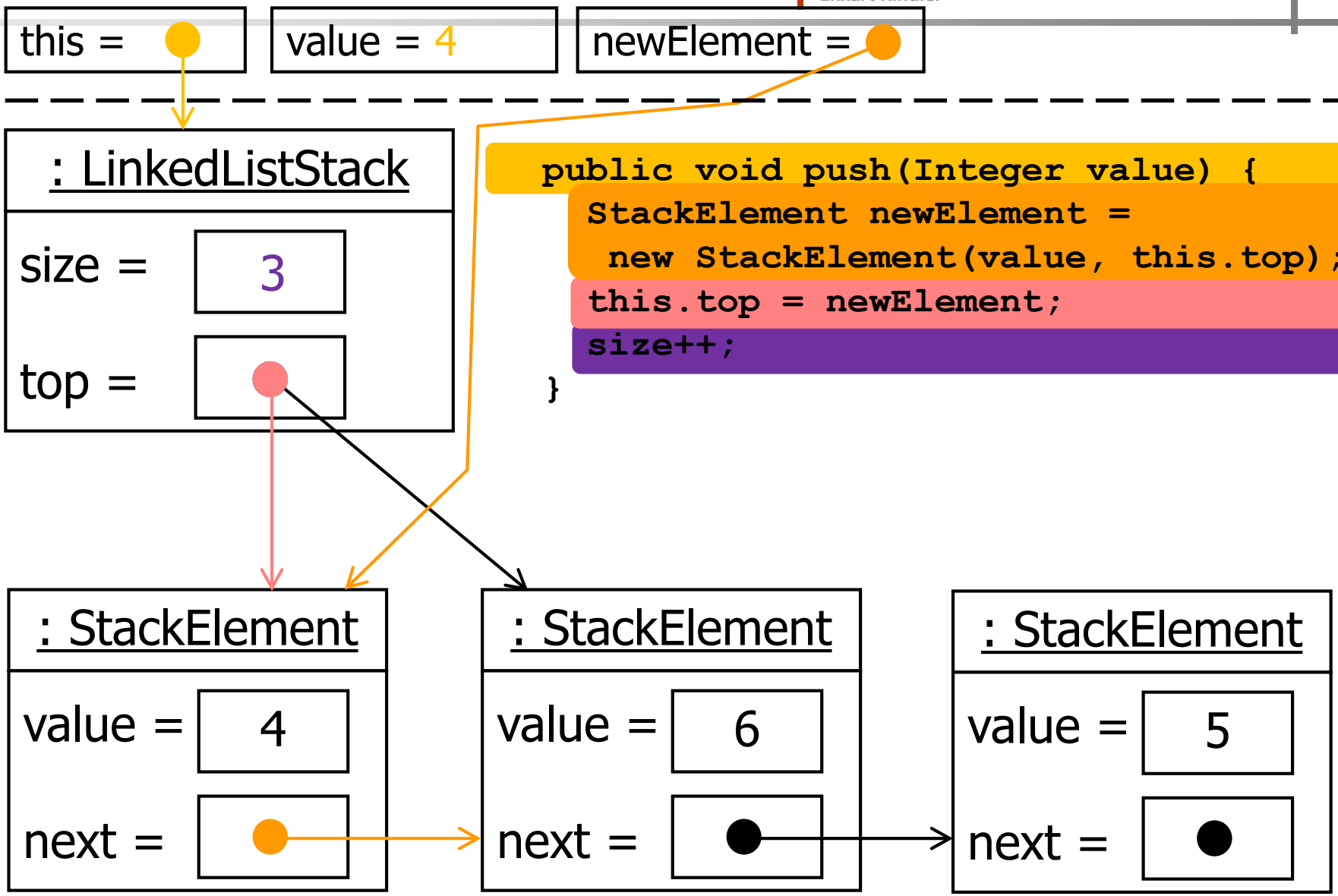



```
lic Integer pop() {
StackElement element = this.top;
if (element != null) {
   this.top = element.next;
   size--;
   return element.value;
}
```



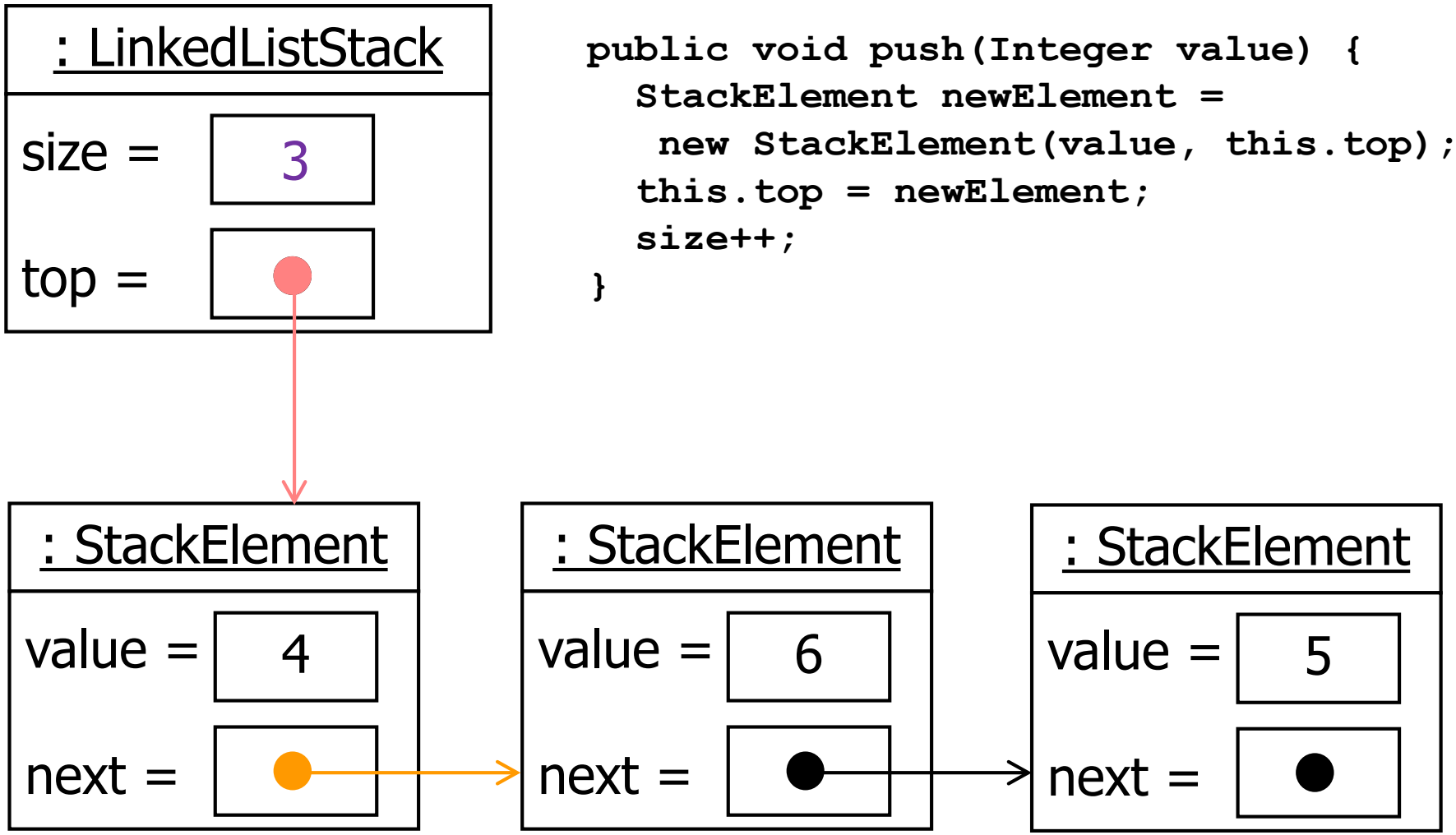
push()

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



push()

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler




```
public void push(Integer value) {
   StackElement newElement =
    new StackElement(value, this.top);
   this.top = newElement;
   size++;
}
```

Generisk `Stack`

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();

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

Lige så nemt kunne man definere og realisere en stak med elementer af hvilken som helst type `E`: generisk stak: Bare erstat `Integer` med `E`.

Spørgsmål

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Hvordan kan man adskille ægte resultater fra resultater, som er en slags fejlmeldinger?

Svar: Exceptions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Metoder kan ”kaste” (engl. *throw*) undtagelser (engl. *exceptions)*.

Dem som har kaldt metoden som kaster (engl. *throw*) en *exception*, kan enten

- fange (engl. catch) denne exception og fortsætte med beregningen eller
- ikke fange den, så ryger exception op et niveau højere op (i dette tilfælde kan metoden ikke fortsætte selv – undtaget **`finally`**, som vi diskuterer senere)

Hvis ingen fanger exception, så stopper hele programmet/beregning med denne exception.

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();

}
```

`throws` betyder kun: metoden "kan kaste", men ikke at metoden nødvendigvis gør det!

Vi definerer her at de forskellige metoder kan kaste forskellige exceptions. De exceptions vi bruger her er Javas indbyggede exceptions. Men man kan også definere egne exceptions (se vores eksempel fra sidste uge).

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<E> 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 IllegalArgumentException {
        StackElement<E> newElement =
              new StackElement<>(value, top);
        size++;
        top = newElement;
}
```

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

Eks.: Stak (implement.)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
...
private class StackElement<E> {
        private E value;
        private StackElement next;

        private StackElement<E>(
                        E value,
                        StackElement<E> next)
                    throws IllegalArgumentException {
            if (value == null) {
              throw new IllegalArgumentException();
            }
            this.value = value;
            this.next = next;
} } }
```

Eks.: Stak (brug)

Vi bruger `Integer` nu for type-parameter `E`; så har vi en stak til heltal objekter igen.

```
public void test() {
  Stack<Integer> stack = new LinkedListStack<Integer>();
  ...
  stack.clear();
  stack.pop();
}
```

`pop()` udløser (kaster) en `IllegalStateException`, men da den ikke bliver fanget ryger den op til den som kaldte `test()` metoden.

Eks.: Stak (brug)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public void test() {
   Stack<Integer> stack = new LinkedListStack<Integer>();
   ...
   stack.clear();
   try {
      stack.pop();
   } catch (IllegalStateException e) {
      // do something to fix things
   }
}
```

I det her eksempel ville det selvfølgelig være bedre at tjekke om stakken er tom (`isEmpty()`) inden man kalder `pop()`.

`pop()` udløser (kaster) en `IllegalStateException`, den bliver fanget af catch blokken. Derfor ryger den ikke videre op. I catch blokken kan man prøve at oprette om problemet. Man kan også tilgå exception ved brug af variable `e`,

Eks.: Stak (brug)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
public void test() {
  Stack<Integer> stack = new LinkedListStack<Integer>();
  ...

  try {
    stack.pop();
  } catch (IllegalStateException e) {
    // do something to fix things
  } finally {
    System.out.println("This will always be printed!");
  }
}
```

`finally` blokken bliver altid gennemført, hvis try-blokken bliver startet – uanset om der bliver fanget en exception eller ej. Det kan man bruge til at ryde op.

Specificere Exceptions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Indtil videre har vi specificeret alle exceptions som en metode kan kaste
- Nogle gange bliver det for meget at specificere alle exceptions, som en metode kan kaste. Det gælder især exceptions som man ikke kan gøre noget ved
- Derfor er der to slags exceptions i Java:
  - *Overvågede* (checked) exceptions
  - *Uovervågede* (unchecked) exceptions

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 (kald den med de rigtige parameter), eller IOException (prøv igen) og tilgang til database (→ se DB-kode fra sidste gang)

- skal specificeres i metodens deklaration eller fanges i metodens implementering

Uovervågede exceptions

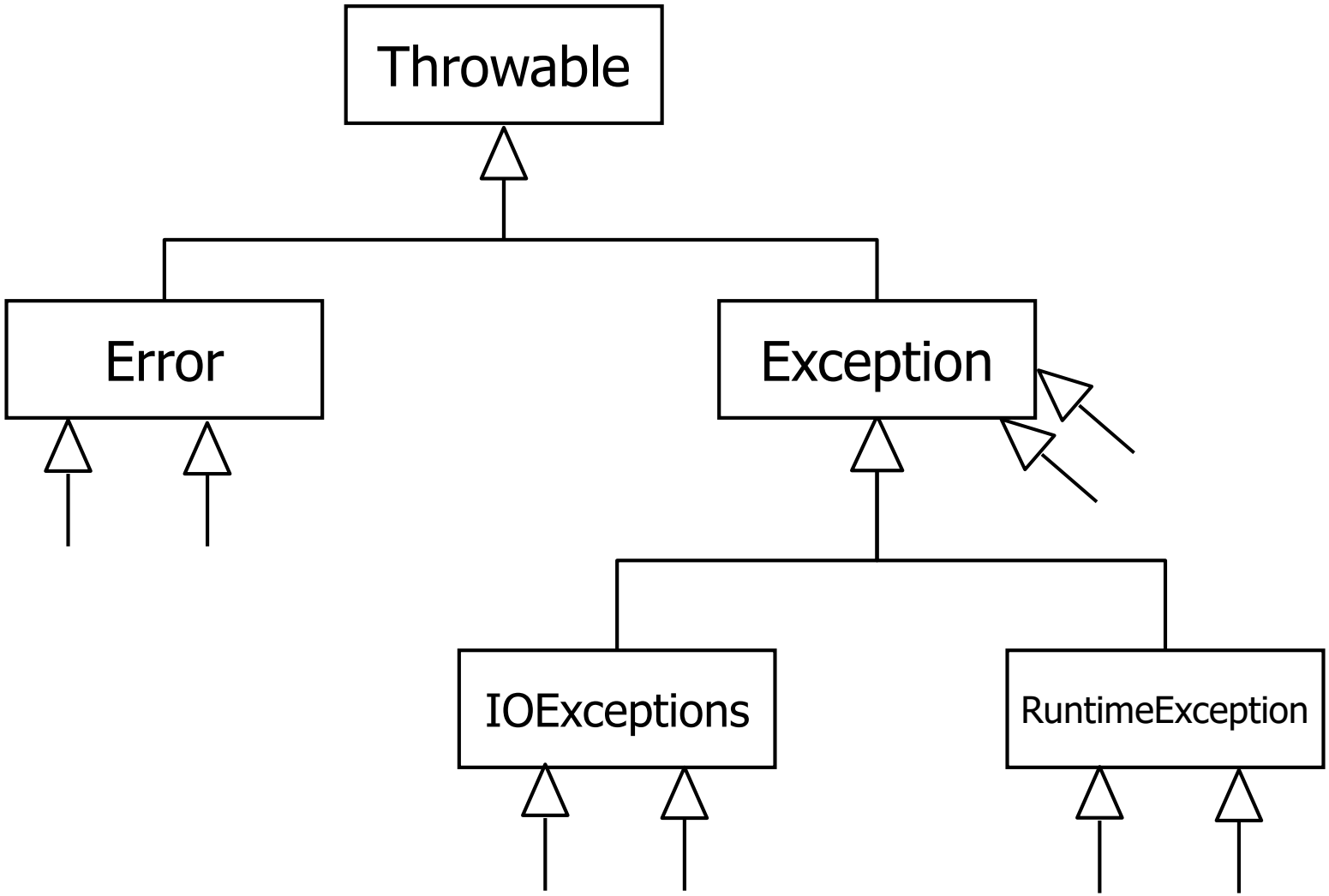
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- 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

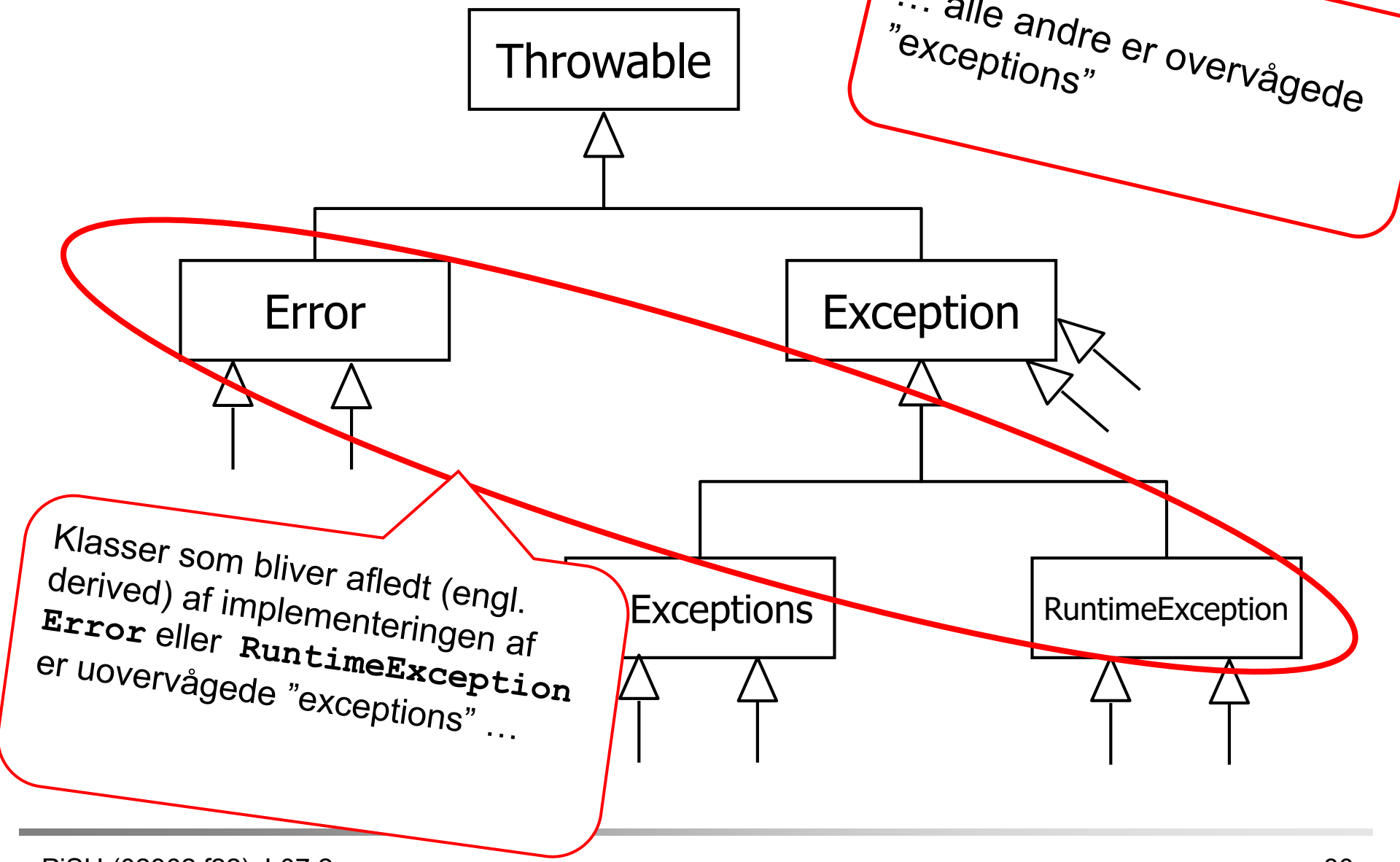
Exception: Typehierarki

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



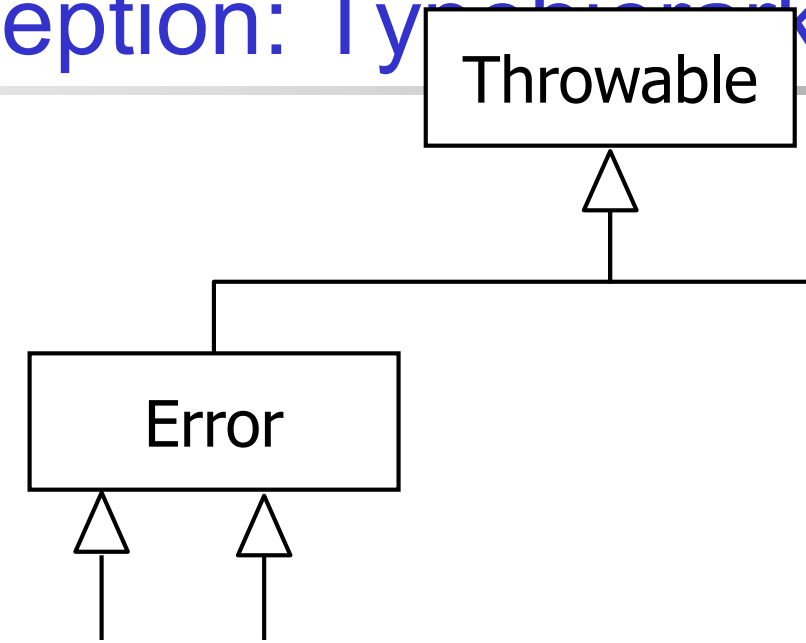
Exception: Typehierarki

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

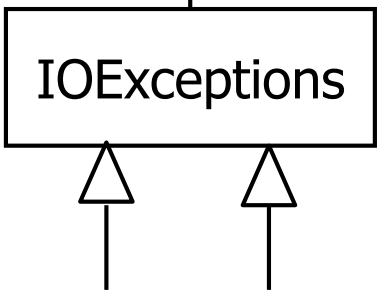


Exception: Typehierarki

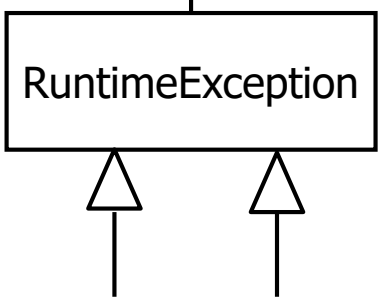
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



- IOError
- StackOverflowError
- OutOfMemoryError
- …



- FileNotFoundException
- EOFException
- …



- ClassCastException
- IllegalArgumentException
- NullPointerException
- IndexOutOfBoundsException
- …

- Reflective-Operation-Exception
- SQLException
- …

Egne Exceptions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Man kan selv definere exceptions:
  man afleder (nøgleord extends) en klasse fra Java's **`Exception`** klasse

- Man skal så overveje om man afleder af
  - Error,
  - RuntimeException,
  - Exception eller
  - IOException

Det bliver så uovervågede (unchecked) exceptions, …

… alle andre er overvågede (checked) exceptions

Kommentarer

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- I nogle tilfælde giver det mening at man specificerer også uovervågede exceptions i metoden, så at APIen er eksplicit om mulige exceptions (se fx. vore eksempler IllegalArgumentException)!

- Man skal **ikke** definere alle sine exceptions som RuntimeExceptions bare fordi man ikke er nødt til at specificere dem! Javas dokumentation siger:

  ”If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.”

Kommentarer

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Hvis man fanger en exception med **`catch`**, kan man kaste (**`throw`**) den samme exception igen (hvis man ikke kan behandle den selv eller kun delvis) lige som man kaster en exception som man har kreeret selv. Hvis den er overvåget, så skal man selvfølgelig specificere det.

```
try {
   // do something, which might cause an
   // exception
} catch (Exception e)  {
   // do some cleanup
   throw e;
}
```

Kommentarer

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Man kan også kreere en ny exception, med den oprindelige som parameter. Det hedder *chaining of exceptions:*

```
try {
   // do something, which might cause an
   // exception
} catch (Exception e)  {
   // do some cleanup
   throw new MyOwnException("My message", e);
}
```

Positive Exceptions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Exceptions kan også være ”positive”: fx. når en robot når sin sidste checkpoint.
- Hvis det sker midt i en længere følge af metodeopkald. Kan man nemt komme tilbage til den første opkald, med at kaste en **`GameEnded`** exception (som så evtl. skal fanges af den yderste opkald).

Men hvis I bygger jeres software op rigtigt, så er det ikke nødvendigt.

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

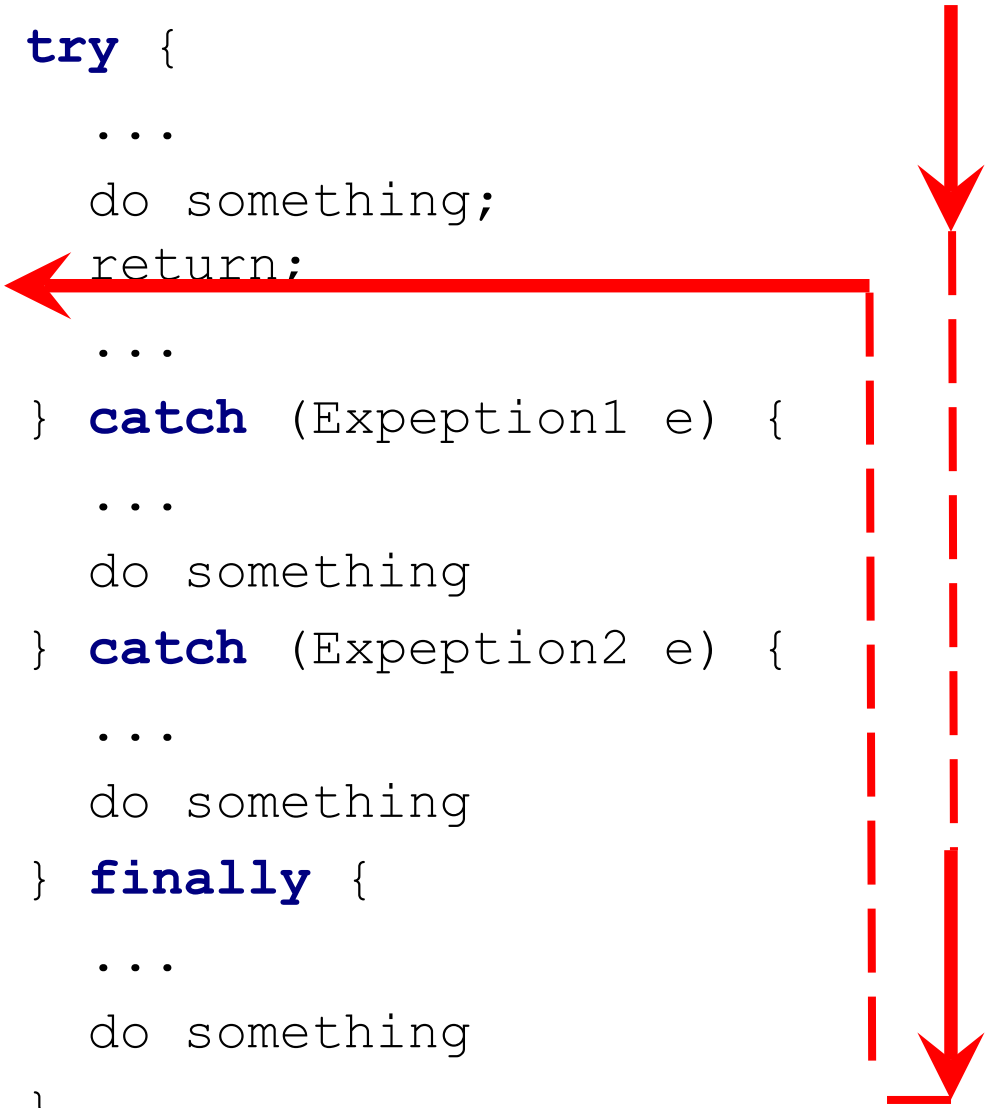
og den første catch-blok som matcher (er super klasse af) den udløste exception eksekveret (men kun en!)

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).

Exceptions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Mere information

Men I kan gerne kigge selv allerede nu:
https://docs.oracle.com/javase/tutorial/essential/exceptions/

- Flere exceptions i koden vedr. databasetilknytning (jf. demo)

Opgave A4 (til 19. 4.)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Anden **prototype** med **fokus** på:

- Finpudsning af database-tilknytning (V4a) og loade spilleplader fra filer (V4b) med vægge og aktioner

- Spillelogik:
  - Man skal kunne afslutte (vinde) et spil (”checkpoints” og bogføring af hvilke ”checkpoints” en spiller kom forbi allerede)
  - Flere forskellige aktioner som er tilknyttet forskellige felter
  - Evtl. robotaktioner imod andre (laser)