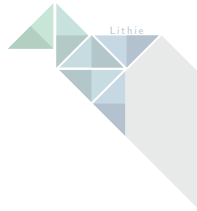
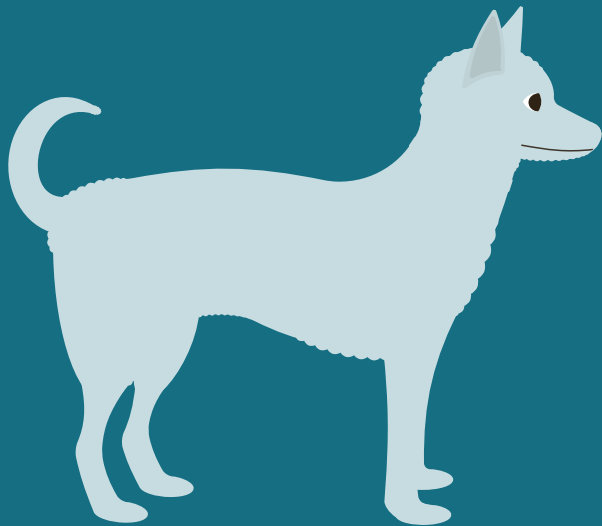


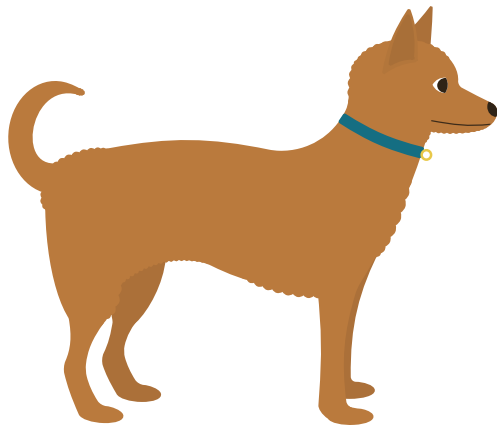
So once, we had holidays

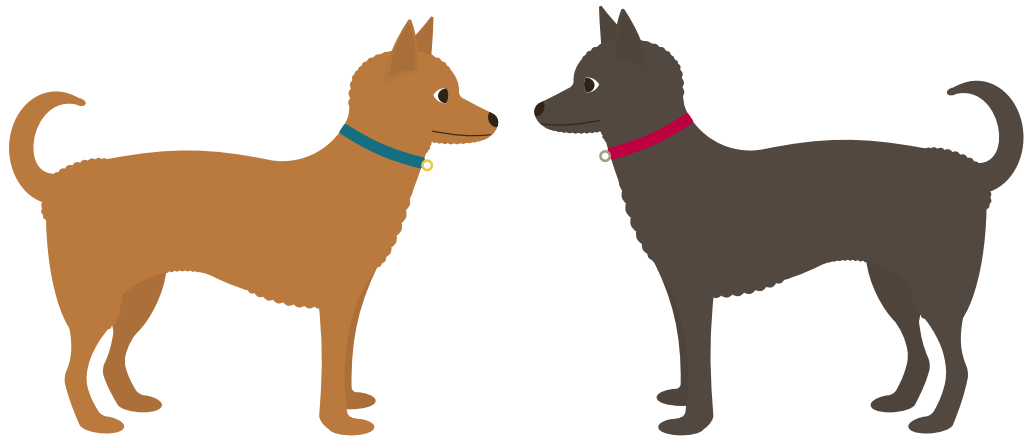
Dreamy Doggos 8

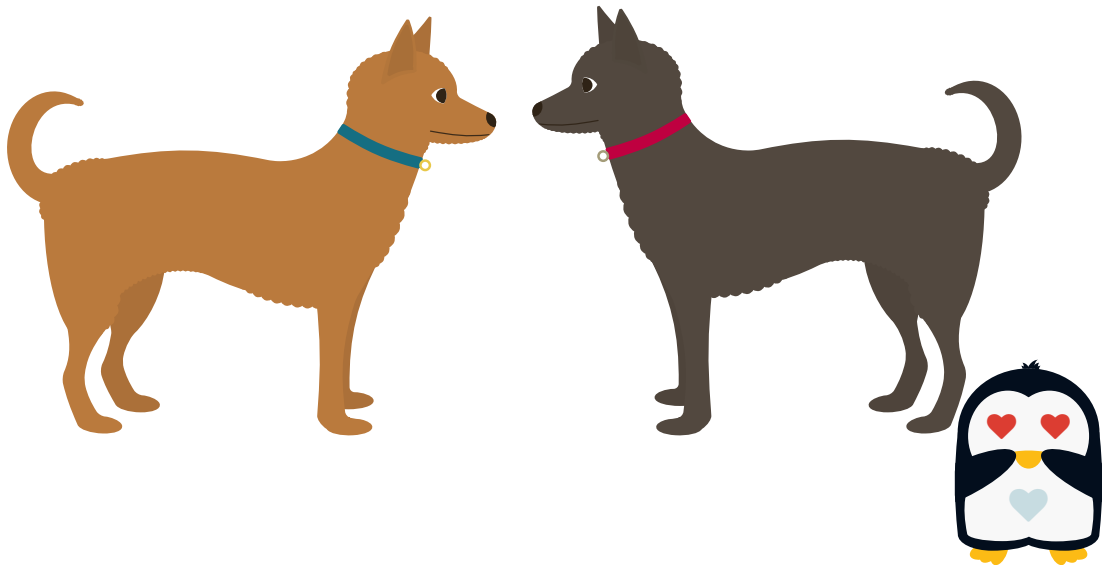
Florian Sihler ◦ KW 2











Kurzes Recap — Weihnachten

Kurzes Recap — Weihnachten

1. Algorithmenkonstruktion

Kurzes Recap — Weihnachten

1. Algorithmenkonstruktion
2. Programmkonstrukte (Namen & Programmfluss)

Kurzes Recap — Weihnachten

1. Algorithmenkonstruktion
2. Programmkonstrukte (Namen & Programmfluss)
3. Arrays und Iterationen

Kurzes Recap — Weihnachten

1. Algorithmenkonstruktion
2. Programmkonstrukte (Namen & Programmfluss)
3. Arrays und Iterationen
4. Unterprogramme

Kurzes Recap — Weihnachten

1. Algorithmenkonstruktion
2. Programmkonstrukte (Namen & Programmfluss)
3. Arrays und Iterationen
4. Unterprogramme
5. Objektorientierte Programmierung

Kurzes Recap — Weihnachten

1. Algorithmenkonstruktion
2. Programmkonstrukte (Namen & Programmfluss)
3. Arrays und Iterationen
4. Unterprogramme
5. Objektorientierte Programmierung
6. Rekursion

Algorithmenkonstruktion

Algorithmenkonstruktion

- Totale Korrektheit

Algorithmenkonstruktion

■ Totale Korrektheit

- *Terminiertheit:*

Endliche Schritte für jede Eingabe.

Algorithmenkonstruktion

■ Totale Korrektheit

- *Terminiertheit:*
- *Partielle Korrektheit:*

Endliche Schritte für jede Eingabe.

Wenn terminiert, dann korrekt.

Algorithmenkonstruktion

■ Totale Korrektheit

- *Terminiertheit:*
- *Partielle Korrektheit:*

Endliche Schritte für jede Eingabe.

Wenn terminiert, dann korrekt.

■ Algorithmusdiskussion

Algorithmenkonstruktion

■ Totale Korrektheit

- *Terminiertheit:*
- *Partielle Korrektheit:*

Endliche Schritte für jede Eingabe.

Wenn terminiert, dann korrekt.

■ Algorithmusdiskussion

- Problemspezifikation

Was meinen Sie mit „schnell“?

Algorithmenkonstruktion

■ Totale Korrektheit

- *Terminiertheit:*
- *Partielle Korrektheit:*

Endliche Schritte für jede Eingabe.

Wenn terminiert, dann korrekt.

■ Algorithmusdiskussion

- Problemspezifikation
- Problemabstraktion

Was meinen Sie mit „schnell“?

Was ist gegeben, was ist gesucht?

Algorithmenkonstruktion

■ Totale Korrektheit

- *Terminiertheit:*
- *Partielle Korrektheit:*

Endliche Schritte für jede Eingabe.

Wenn terminiert, dann korrekt.

■ Algorithmusdiskussion

- Problemspezifikation
- Problemabstraktion
- Algorithmenentwurf

Was meinen Sie mit „schnell“?

Was ist gegeben, was ist gesucht?

Wie kommen wir von gegeben zu gesucht?

Algorithmenkonstruktion

■ Totale Korrektheit

- *Terminiertheit:*
- *Partielle Korrektheit:*

Endliche Schritte für jede Eingabe.

Wenn terminiert, dann korrekt.

■ Algorithmusdiskussion

- Problemspezifikation
- Problemabstraktion
- Algorithmenentwurf
- Korrektheitsnachweis

Was meinen Sie mit „schnell“?

Was ist gegeben, was ist gesucht?

Wie kommen wir von gegeben zu gesucht?

Löst unser Ansatz das Problem?

Algorithmenkonstruktion

■ Totale Korrektheit

- *Terminiertheit:*
- *Partielle Korrektheit:*

Endliche Schritte für jede Eingabe.

Wenn terminiert, dann korrekt.

■ Algorithmusdiskussion

- Problemspezifikation
- Problemabstraktion
- Algorithmenentwurf
- Korrektheitsnachweis
- Aufwandsanalyse

Was meinen Sie mit „schnell“?

Was ist gegeben, was ist gesucht?

Wie kommen wir von gegeben zu gesucht?

Löst unser Ansatz das Problem?

Wie verhält sich der Algorithmus?

Programmkonstrukte

Programmkonstrukte

- Implizite Typkonvertierung

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

- Post vor Prä, sonst wie Arithmetik & Logik.

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

- Post vor Prä, sonst wie Arithmetik & Logik.

■ Default-Werte

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

- Post vor Prä, sonst wie Arithmetik & Logik.

■ Default-Werte

- Nur für Klassen-/Instanzvariablen und Array-Komponenten!

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

- Post vor Prä, sonst wie Arithmetik & Logik.

■ Default-Werte

- Nur für Klassen-/Instanzvariablen und Array-Komponenten!
- Zahlen und Zeichen werden `0`.

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

- Post vor Prä, sonst wie Arithmetik & Logik.

■ Default-Werte

- Nur für Klassen-/Instanzvariablen und Array-Komponenten!
- Zahlen und Zeichen werden `0`.
- Boolean wird `false`.

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

- Post vor Prä, sonst wie Arithmetik & Logik.

■ Default-Werte

- Nur für Klassen-/Instanzvariablen und Array-Komponenten!
- Zahlen und Zeichen werden `0`.
- Boolean wird `false`.
- Der Rest wird `null`.

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

- Post vor Prä, sonst wie Arithmetik & Logik.

■ Default-Werte

- Nur für Klassen-/Instanzvariablen und Array-Komponenten!
- Zahlen und Zeichen werden `0`.
- Boolean wird `false`.
- Der Rest wird `null`.

■ Überschatten

Programmkonstrukte

■ Implizite Typkonvertierung

- Von klein zu groß: `byte` → `short` → `int` → `long` → `float` → `double`.
- Sowie: `char` → `int`.

■ Präzedenzregeln

- Post vor Prä, sonst wie Arithmetik & Logik.

■ Default-Werte

- Nur für Klassen-/Instanzvariablen und Array-Komponenten!
- Zahlen und Zeichen werden `0`.
- Boolean wird `false`.
- Der Rest wird `null`.

■ Überschatten

- Lokaler Bezeichner „überschattet“ die Sichtbarkeit eines globalen.

Arrays und Iterationen

Arrays und Iterationen

- Arrays sind komplexe Datentypen

Arrays und Iterationen

- Arrays sind komplexe Datentypen
- Mehrdimensionale Arrays

Arrays und Iterationen

- Arrays sind komplexe Datentypen
- Mehrdimensionale Arrays
 - „Gibt es an sich nicht.“ `int[][][]` \equiv `((int[])[])[[]]`

Arrays und Iterationen

- Arrays sind komplexe Datentypen
- Mehrdimensionale Arrays
 - „Gibt es an sich nicht.“ `int[][][]` \equiv `((int[])[])[[]]`
 - Sind eindimensionale Arrays von eindimensionalen Arrays von...

Arrays und Iterationen

- Arrays sind komplexe Datentypen
- Mehrdimensionale Arrays
 - „Gibt es an sich nicht.“ `int[][][]` \equiv `((int[])[])[[]]`
 - Sind eindimensionale Arrays von eindimensionalen Arrays von...
- Die drei Schleifenarten sind gleichmächtig

Arrays und Iterationen

- Arrays sind komplexe Datentypen
- Mehrdimensionale Arrays
 - „Gibt es an sich nicht.“ `int[][][]` \equiv `((int[])[])[[]]`
 - Sind eindimensionale Arrays von eindimensionalen Arrays von...
- Die drei Schleifenarten sind gleichmächtig
 - Maximum bekannt: `for`

Arrays und Iterationen

- Arrays sind komplexe Datentypen
- Mehrdimensionale Arrays
 - „Gibt es an sich nicht.“ `int[][][]` \equiv `((int[])[])[[]]`
 - Sind eindimensionale Arrays von eindimensionalen Arrays von...
- Die drei Schleifenarten sind gleichmächtig
 - Maximum bekannt: `for`
 - Mindestens ein mal: `do-while`

Arrays und Iterationen

- Arrays sind komplexe Datentypen
- Mehrdimensionale Arrays
 - „Gibt es an sich nicht.“ `int[][][]` \equiv `((int[][])[])[]`
 - Sind eindimensionale Arrays von eindimensionalen Arrays von...
- Die drei Schleifenarten sind gleichmächtig
 - Maximum bekannt: `for`
 - Mindestens ein mal: `do-while`
 - Sonst: `while`

Unterprogramme

Unterprogramme

- Überladung

Unterprogramme

■ Überladung

- Gleicher Name, andere Signatur.

Unterprogramme

■ Überladung

- Gleicher Name, andere Signatur.
- Signatur: Name & Parametertypliste.

Unterprogramme

■ Überladung

- Gleicher Name, andere Signatur.
- Signatur: Name & Parametertypliste.
- Müssen zudem in selber Klasse sein (später: Vererbung).

Unterprogramme

■ Überladung

- Gleicher Name, andere Signatur.
- Signatur: Name & Parametertypliste.
- Müssen zudem in selber Klasse sein (später: Vererbung).

■ Beim Aufruf macht Java call-by-value

Unterprogramme

■ Überladung

- Gleicher Name, andere Signatur.
- Signatur: Name & Parametertypliste.
- Müssen zudem in selber Klasse sein (später: Vererbung).

■ Beim Aufruf macht Java call-by-value

- Alle Parameter werden kopiert (Stack).

Unterprogramme

■ Überladung

- Gleicher Name, andere Signatur.
- Signatur: Name & Parametertypliste.
- Müssen zudem in selber Klasse sein (später: Vererbung).

■ Beim Aufruf macht Java call-by-value

- Alle Parameter werden kopiert (Stack).
- Bei komplexen Datentypen wird die Referenz kopiert.

Objektorientierte Programmierung

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand
 - *Instanziierung*: Erzeugen eines neuen Objektes.

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand
 - *Instanziierung*: Erzeugen eines neuen Objektes.
 - Wenn keiner: erzeugt Java den leeren Standardkonstruktor.

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand
 - *Instanziierung*: Erzeugen eines neuen Objektes.
 - Wenn keiner: erzeugt Java den leeren Standardkonstruktor.
 - `this` erlaubt Aufruf von Überladungen.

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand
 - *Instanziierung*: Erzeugen eines neuen Objektes.
 - Wenn keiner: erzeugt Java den leeren Standardkonstruktor.
 - `this` erlaubt Aufruf von Überladungen.
- Klassen, Methoden, ...

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand
 - *Instanziierung*: Erzeugen eines neuen Objektes.
 - Wenn keiner: erzeugt Java den leeren Standardkonstruktor.
 - `this` erlaubt Aufruf von Überladungen.
- Klassen, Methoden, ...
 - Haben eine Sichtbarkeit (`public`, ...) unter der sie zugreifbar sind.

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand
 - *Instanziierung*: Erzeugen eines neuen Objektes.
 - Wenn keiner: erzeugt Java den leeren Standardkonstruktor.
 - `this` erlaubt Aufruf von Überladungen.
- Klassen, Methoden, ...
 - Haben eine Sichtbarkeit (`public`, ...) unter der sie zugreifbar sind.
- Daten

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand
 - *Instanziierung*: Erzeugen eines neuen Objektes.
 - Wenn keiner: erzeugt Java den leeren Standardkonstruktor.
 - `this` erlaubt Aufruf von Überladungen.
- Klassen, Methoden, ...
 - Haben eine Sichtbarkeit (`public`, ...) unter der sie zugreifbar sind.
- Daten
 - Haben einen *Gültigkeitsbereich*: Wo die Daten „existieren“.

Objektorientierte Programmierung

- Eine Klasse definiert die Blaupause für Objekte
 - Attribute definieren den Zustand.
 - Methoden definieren den Verhalten.
- Der Konstruktor baut den initialen Zustand
 - *Instanziierung*: Erzeugen eines neuen Objektes.
 - Wenn keiner: erzeugt Java den leeren Standardkonstruktor.
 - `this` erlaubt Aufruf von Überladungen.
- Klassen, Methoden, ...
 - Haben eine Sichtbarkeit (`public`, ...) unter der sie zugreifbar sind.
- Daten
 - Haben einen *Gültigkeitsbereich*: Wo die Daten „existieren“.
 - Ein Datum kann nicht sichtbar, aber dennoch gültig sein (Überschatten).

Präsenzaufgabe

Präsenzaufgabe

1

Teil mich. Nochmal. Nochmal!

Präsenzaufgabe

1

Teil mich. Nochmal. Nochmal!

In dieser Aufgaben sollen Sie mit Hilfe des *euklidischen Algorithmus* den größten gemeinsamen Teiler zweier ganzer Zahlen a und b bestimmen. Implementieren Sie den Algorithmus einmal *iterativ* und einmal *rekursiv*.

Präsenzaufgabe

1

Teil mich. Nochmal. Nochmal!

In dieser Aufgaben sollen Sie mit Hilfe des *euklidischen Algorithmus* den größten gemeinsamen Teiler zweier ganzer Zahlen a und b bestimmen. Implementieren Sie den Algorithmus einmal *iterativ* und einmal *rekursiv*. Welchen jeweiligen Vor- und Nachteil haben die beiden Lösungen?

Präsenzaufgabe

1

Teil mich. Nochmal. Nochmal!

In dieser Aufgaben sollen Sie mit Hilfe des *euklidischen Algorithmus* den größten gemeinsamen Teiler zweier ganzer Zahlen a und b bestimmen. Implementieren Sie den Algorithmus einmal *iterativ* und einmal *rekursiv*. Welchen jeweiligen Vor- und Nachteil haben die beiden Lösungen?

Beispiele:

Präsenzaufgabe

1

Teil mich. Nochmal. Nochmal!

In dieser Aufgaben sollen Sie mit Hilfe des *euklidischen Algorithmus* den größten gemeinsamen Teiler zweier ganzer Zahlen a und b bestimmen. Implementieren Sie den Algorithmus einmal *iterativ* und einmal *rekursiv*. Welchen jeweiligen Vor- und Nachteil haben die beiden Lösungen?

Beispiele:

■ $\text{ggt}(39, 27) \rightarrow 3$

$$39 = 1 \cdot 27 + 12$$

$$27 = 2 \cdot 12 + 3$$

$$12 = 4 \cdot \boxed{3} + 0$$

Präsenzaufgabe

1

Teil mich. Nochmal. Nochmal!

In dieser Aufgaben sollen Sie mit Hilfe des *euklidischen Algorithmus* den größten gemeinsamen Teiler zweier ganzer Zahlen a und b bestimmen. Implementieren Sie den Algorithmus einmal *iterativ* und einmal *rekursiv*. Welchen jeweiligen Vor- und Nachteil haben die beiden Lösungen?

Beispiele:

$$\blacksquare \text{ ggt}(39, 27) \rightarrow 3$$

$$39 = 1 \cdot 27 + 12$$

$$27 = 2 \cdot 12 + 3$$

$$12 = 4 \cdot \boxed{3} + 0$$

$$\blacksquare \text{ ggt}(15, 7) \rightarrow 1$$

$$15 = 2 \cdot 7 + 1$$

$$7 = 7 \cdot \boxed{1} + 0$$

Präsenzaufgabe - Lösung

Präsenzaufgabe - Lösung

- Der größter gemeinsame Teiler also: [Euclid.java](#).

Präsenzaufgabe - Lösung

- Der größter gemeinsame Teiler also: [Euclid.java](#).
- Die Definition:

Präsenzaufgabe - Lösung

- Der größter gemeinsame Teiler also: [Euclid.java](#).
- Die Definition:

$$\text{ggt}(a, b) = \begin{cases} a & \text{wenn } b = 0 \\ \text{ggt}(b, a \bmod b) & \text{sonst.} \end{cases}$$

Präsenzaufgabe - Lösung

- Der größter gemeinsame Teiler also: [Euclid.java](#).
- Die Definition:

$$\text{ggt}(a, b) = \begin{cases} a & \text{wenn } b = 0 \\ \text{ggt}(b, a \bmod b) & \text{sonst.} \end{cases}$$

- In Java? Unfassbar anders natürlich!

Präsenzaufgabe - Lösung

- Der größter gemeinsame Teiler also: [Euclid.java](#).
- Die Definition:

$$\text{ggt}(a, b) = \begin{cases} a & \text{wenn } b = 0 \\ \text{ggt}(b, a \bmod b) & \text{sonst.} \end{cases}$$

- In Java? Unfassbar anders natürlich!

```
public static int ggt(int a, int b) {  
  
  
  
  
  
  
}
```


Präsenzaufgabe - Lösung

- Der größter gemeinsame Teiler also: [Euclid.java](#).
- Die Definition:

$$\text{ggt}(a, b) = \begin{cases} a & \text{wenn } b = 0 \\ \text{ggt}(b, a \bmod b) & \text{sonst.} \end{cases}$$

- In Java? Unfassbar anders natürlich!

```
public static int ggt(int a, int b) {  
    if(b == 0)  
        return a;  
  
}
```

Präsenzaufgabe - Lösung

- Der größter gemeinsame Teiler also: [Euclid.java](#).
- Die Definition:

$$\text{ggt}(a, b) = \begin{cases} a & \text{wenn } b = 0 \\ \text{ggt}(b, a \bmod b) & \text{sonst.} \end{cases}$$

- In Java? Unfassbar anders natürlich!

```
public static int ggt(int a, int b) {  
    if(b == 0)  
        return a;  
    else  
        return ggt(b, a % b);  
}
```

Präsenzaufgabe - Lösung

Präsenzaufgabe - Lösung

- Die iterative Variante:

Präsenzaufgabe - Lösung

- Die iterative Variante:

```
public static int ggt(int a, int b) {
```

}

Präsenzaufgabe - Lösung

- Die iterative Variante:

```
public static int ggt(int a, int b) {  
    while(b != 0) {  
  
    }  
  
}
```

Präsenzaufgabe - Lösung

- Die iterative Variante:

```
public static int ggt(int a, int b) {  
    while(b != 0) {  
        // swap a, b = b, a % b  
  
    }  
  
}
```

Präsenzaufgabe - Lösung

- Die iterative Variante:

```
public static int ggt(int a, int b) {  
    while(b != 0) {  
        // swap a, b = b, a % b  
        int tmp = a % b;  
        a = b;  
        b = tmp;  
    }  
  
}
```


Präsenzaufgabe - Lösung

- Die iterative Variante:

```
public static int ggt(int a, int b) {  
    while(b != 0) {  
        // swap a, b = b, a % b  
        int tmp = a % b;  
        a = b;  
        b = tmp;  
    }  
    return a;  
}
```

Präsenzaufgabe - Iteration vs. Rekursion

Präsenzaufgabe - Iteration vs. Rekursion

- Die *iterative* Lösung ist performanter und braucht weniger Speicher.

Präsenzaufgabe - Iteration vs. Rekursion

- Die *iterative* Lösung ist performanter und braucht weniger Speicher.
- Die *rekursive* Lösung ist kompakter und übersichtlicher.

Übungsblatt 8 - Aufgabe 1a)

Übungsblatt 8 - Aufgabe 1a)

- Taxi nach Heim bittedange ([Taxi.java](#)).

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange ([Taxi.java](#)).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange (`Taxi.java`).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange ([Taxi.java](#)).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange ([Taxi.java](#)).

Implementieren Sie eine Klasse **Taxi** mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {
```

```
}
```

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange ([Taxi.java](#)).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {
```

```
}
```

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange ([Taxi.java](#)).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {  
    double basicCharge;  
    double kilometerPrice;  
    double fuelConsumption;  
    double fueltankCapacity;  
  
    double remainingFuel;  
    double taxometer;  
    double totalEarnings;  
  
}
```

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange (Taxi.java).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {  
    double basicCharge;  
    double kilometerPrice;  
    double fuelConsumption;  
    double fueltankCapacity;  
  
    double remainingFuel;  
    double taxometer;  
    double totalEarnings;  
  
}
```

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange (Taxi.java).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {  
    private final double basicCharge;  
    private final double kilometerPrice;  
    private final double fuelConsumption;  
    private final double fuelTankCapacity;  
  
    double remainingFuel;  
    double taxometer;  
    double totalEarnings;  
  
}
```

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange (Taxi.java).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {  
    private final double basicCharge;  
    private final double kilometerPrice;  
    private final double fuelConsumption;  
    private final double fueltankCapacity;  
  
    double remainingFuel;  
    double taxometer;  
    double totalEarnings;  
  
}
```

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange (Taxi.java).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {  
    private final double basicCharge;  
    private final double kilometerPrice;  
    private final double fuelConsumption;  
    private final double fuelTankCapacity;  
  
    private double remainingFuel;  
    private double taxometer;  
    private double totalEarnings;  
  
}
```


Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange (Taxi.java).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {  
    private final double basicCharge;  
    private final double kilometerPrice;  
    private final double fuelConsumption;  
    private final double fuelTankCapacity;  
  
    private double remainingFuel;  
    private double taxometer;  
    private double totalEarnings;  
  
}
```

Übungsblatt 8 - Aufgabe 1a)

■ Taxi nach Heim bittedange (Taxi.java).

Implementieren Sie eine Klasse `Taxi` mit privaten, unveränderlichen Instanzvariablen für die folgenden Eigenschaften:

- Grundpreis einer Fahrt in Euro
- Den Kilometerpreis in Euro
- Den Kraftstoffverbrauch in $\frac{\text{L}}{100\text{km}}$
- Das Tankvolumen des Taxis in Liter

Fügen Sie weiterhin die folgenden privaten, veränderlichen Variablen hinzu:

- Den Tankinhalt in Litern
- Den Stand des Taxometers in Euro
- Die Gesamteinnahmen in Euro

Erstellen Sie nun einen öffentlichen Konstruktor um diese Variablen sinnvoll zu initialisieren.

```
public class Taxi {
    private final double basicCharge;
    private final double kilometerPrice;
    private final double fuelConsumption;
    private final double fueltankCapacity;

    private double remainingFuel;
    private double taxometer;
    private double totalEarnings;

    public Taxi(...) {
        ...
    }
}
```

Übungsblatt 8 - Aufgabe 1a)

```
public class Taxi {  
    private final double basicCharge;  
    private final double fuelConsumption;  
    private double remainingFuel;  
    private double totalEarnings;  
  
    private final double kilometerPrice;  
    private final double fueltankCapacity;  
    private double taxometer;  
  
}
```

Übungsblatt 8 - Aufgabe 1a)

```
public class Taxi {  
    private final double basicCharge;  
    private final double fuelConsumption;  
    private double remainingFuel;  
    private double totalEarnings;  
  
    private final double kilometerPrice;  
    private final double fueltankCapacity;  
    private double taxometer;  
  
    public Taxi(double basicCharge, double kilometerPrice,  
                double fuelConsumption, double fueltankCapacity) {  
  
    }  
}
```

Übungsblatt 8 - Aufgabe 1a)

```
public class Taxi {  
    private final double basicCharge;  
    private final double fuelConsumption;  
    private double remainingFuel;  
    private double totalEarnings;  
  
    private final double kilometerPrice;  
    private final double fueltankCapacity;  
    private double taxometer;  
  
    public Taxi(double basicCharge, double kilometerPrice,  
                double fuelConsumption, double fueltankCapacity) {  
        this.basicCharge = basicCharge;  
        this.kilometerPrice = kilometerPrice;  
        this.fuelConsumption = fuelConsumption;  
        this.fueltankCapacity = fueltankCapacity;  
  
    }  
}
```

Übungsblatt 8 - Aufgabe 1a)

```
public class Taxi {  
    private final double basicCharge;           private final double kilometerPrice;  
    private final double fuelConsumption;       private final double fueltankCapacity;  
    private double remainingFuel;               private double taxometer;  
    private double totalEarnings;  
  
    public Taxi(double basicCharge, double kilometerPrice,  
                double fuelConsumption, double fueltankCapacity) {  
        this.basicCharge = basicCharge;  
        this.kilometerPrice = kilometerPrice;  
        this.fuelConsumption = fuelConsumption;  
        this.fueltankCapacity = fueltankCapacity;  
  
        this.remainingFuel = fueltankCapacity;  
  
    }  
}
```

Übungsblatt 8 - Aufgabe 1a)

```
public class Taxi {  
    private final double basicCharge;           private final double kilometerPrice;  
    private final double fuelConsumption;       private final double fueltankCapacity;  
    private double remainingFuel;               private double taxometer;  
    private double totalEarnings;  
  
    public Taxi(double basicCharge, double kilometerPrice,  
                double fuelConsumption, double fueltankCapacity) {  
        this.basicCharge = basicCharge;  
        this.kilometerPrice = kilometerPrice;  
        this.fuelConsumption = fuelConsumption;  
        this.fueltankCapacity = fueltankCapacity;  
  
        this.remainingFuel = fueltankCapacity;  
        this.taxometer = 0.0;  
        this.totalEarnings = 0.0;  
    }  
}
```

Übungsblatt 8 - Aufgabe 1b)

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

```
public double drive(double distance) {
```

}

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

```
public double drive(double distance) {  
    double requiredFuel = distance * fuelConsumption / 100;  
  
}
```

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

```
public double drive(double distance) {  
    double requiredFuel = distance * fuelConsumption / 100;  
    if(this.remainingFuel < requiredFuel)  
  
}
```

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

```
public double drive(double distance) {  
    double requiredFuel = distance * fuelConsumption / 100;  
    if(this.remainingFuel < requiredFuel)  
        return -1; // Fahrt nicht möglich  
  
}
```

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

```
public double drive(double distance) {  
    double requiredFuel = distance * fuelConsumption / 100;  
    if(this.remainingFuel < requiredFuel)  
        return -1; // Fahrt nicht möglich  
  
    this.remainingFuel -= requiredFuel;  
  
}
```

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

```
public double drive(double distance) {  
    double requiredFuel = distance * fuelConsumption / 100;  
    if(this.remainingFuel < requiredFuel)  
        return -1; // Fahrt nicht möglich  
  
    this.remainingFuel -= requiredFuel;  
    this.taxometer += basicCharge + distance * kilometerPrice;  
  
}
```

Übungsblatt 8 - Aufgabe 1b)

- Overdriven sie mich home:

```
public double drive(double distance) {  
    double requiredFuel = distance * fuelConsumption / 100;  
    if(this.remainingFuel < requiredFuel)  
        return -1; // Fahrt nicht möglich  
  
    this.remainingFuel -= requiredFuel;  
    this.taxometer += basicCharge + distance * kilometerPrice;  
  
    return taxometer;  
}
```


Übungsblatt 8 - Aufgabe 1c)

Übungsblatt 8 - Aufgabe 1c)

- Money Please:

Übungsblatt 8 - Aufgabe 1c)

- Money Please:

```
public boolean pay(double amount) {
```

}

Übungsblatt 8 - Aufgabe 1c)

■ Money Please:

```
public boolean pay(double amount) {  
    if (amount < taxometer)  
        return false;  
  
}
```

Übungsblatt 8 - Aufgabe 1c)

■ Money Please:

```
public boolean pay(double amount) {  
    if (amount < taxometer)  
        return false;  
    if (amount > taxometer)  
        System.out.format("Received %.2f€ tip%n", amount - taxometer);  
}
```

Übungsblatt 8 - Aufgabe 1c)

■ Money Please:

```
public boolean pay(double amount) {  
    if (amount < taxometer)  
        return false;  
    if (amount > taxometer)  
        System.out.format("Received %.2f€ tip%n", amount - taxometer);  
  
    taxometer = 0.0;  
  
}
```

Übungsblatt 8 - Aufgabe 1c)

■ Money Please:

```
public boolean pay(double amount) {  
    if (amount < taxometer)  
        return false;  
    if (amount > taxometer)  
        System.out.format("Received %.2f€ tip%n", amount - taxometer);  
  
    taxometer = 0.0;  
    totalEarnings += amount;  
  
}
```

Übungsblatt 8 - Aufgabe 1c)

■ Money Please:

```
public boolean pay(double amount) {  
    if (amount < taxometer)  
        return false;  
    if (amount > taxometer)  
        System.out.format("Received %.2f€ tip%n", amount - taxometer);  
  
    taxometer = 0.0;  
    totalEarnings += amount;  
    return true;  
}
```


Übungsblatt 8 - Aufgabe 1c)

■ Money Please:

```
public boolean pay(double amount) {  
    if (amount < taxometer)  
        return false;  
    if (amount > taxometer)  
        System.out.format("Received %.2f€ tip%n", amount - taxometer);  
  
    taxometer = 0.0;  
    totalEarnings += amount;  
    return true;  
}
```

Wir könnten auch `else if` benutzen. So verschmilzt aber ein Guard mit Funktionalität.



Übungsblatt 8 - Aufgabe 1d)

Übungsblatt 8 - Aufgabe 1d)

- Einmal auftankeeen:

Übungsblatt 8 - Aufgabe 1d)

- Einmal auftanken:

```
public void refill(double pricePerLitre) {
```

Übungsblatt 8 - Aufgabe 1d)

- Einmal auftanken:

```
public void refill(double pricePerLitre) {  
    double refill = Math.min(  
        this.totalEarnings / pricePerLitre,  
        this.fuelTankCapacity - this.remainingFuel  
    ); // Können wir finanziell gesehen volltanken?  
  
}
```

Übungsblatt 8 - Aufgabe 1d)

- Einmal auftanken:

```
public void refill(double pricePerLitre) {  
    double refill = Math.min(  
        this.totalEarnings / pricePerLitre,  
        this.fuelTankCapacity - this.remainingFuel  
    ); // Können wir finanziell gesehen volltanken?  
  
    this.remainingFuel += refill;  
  
}
```

Übungsblatt 8 - Aufgabe 1d)

- Einmal auftanken:

```
public void refill(double pricePerLitre) {  
    double refill = Math.min(  
        this.totalEarnings / pricePerLitre,  
        this.fuelTankCapacity - this.remainingFuel  
    ); // Können wir finanziell gesehen volltanken?  
  
    this.remainingFuel += refill;  
    this.totalEarnings -= refill * pricePerLitre;  
}
```

Übungsblatt 8 - Aufgabe 1e)

Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {
```

```
}
```

Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {  
    Taxi taxi = new Taxi(3, 0.5, 10, 50);
```

}

Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {  
    Taxi taxi = new Taxi(3, 0.5, 10, 50);  
    double totalCost = taxi.drive(250);  
  
}
```

Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {  
    Taxi taxi = new Taxi(3, 0.5, 10, 50);  
    double totalCost = taxi.drive(250);  
    if (totalCost == -1) System.out.println("Not enough fuel left");  
    else System.out.println("Trip costs " + totalCost);  
  
}
```

Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {  
    Taxi taxi = new Taxi(3, 0.5, 10, 50);  
    double totalCost = taxi.drive(250);  
    if (totalCost == -1) System.out.println("Not enough fuel left");  
    else System.out.println("Trip costs " + totalCost);  
    // pay  
    if (taxi.pay(totalCost + 3.00)) System.out.println("Tipped 3,00€");  
  
}
```

Übungsblatt 8 - Aufgabe 1e)

■ Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {  
    Taxi taxi = new Taxi(3, 0.5, 10, 50);  
    double totalCost = taxi.drive(250);  
    if (totalCost == -1) System.out.println("Not_enough_fuel_left");  
    else System.out.println("Trip_costs_" + totalCost);  
    // pay  
    if (taxi.pay(totalCost + 3.00)) System.out.println("Tipped_3,00€");  
    // drive  
    totalCost = taxi.drive(450);  
    if (totalCost == -1) System.out.println("Not_enough_fuel_left");  
  
}
```

Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {  
    Taxi taxi = new Taxi(3, 0.5, 10, 50);  
    double totalCost = taxi.drive(250);  
    if (totalCost == -1) System.out.println("Not_enough_fuel_left");  
    else System.out.println("Trip_costs_" + totalCost);  
    // pay  
    if (taxi.pay(totalCost + 3.00)) System.out.println("Tipped_3,00€");  
    // drive  
    totalCost = taxi.drive(450);  
    if (totalCost == -1) System.out.println("Not_enough_fuel_left");  
    // refill & drive  
    taxi.refill(1.7);  
  
}
```


Übungsblatt 8 - Aufgabe 1e)

- Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {  
    Taxi taxi = new Taxi(3, 0.5, 10, 50);  
    double totalCost = taxi.drive(250);  
    if (totalCost == -1) System.out.println("Not enough fuel left");  
    else System.out.println("Trip costs " + totalCost);  
    // pay  
    if (taxi.pay(totalCost + 3.00)) System.out.println("Tipped 3,00€");  
    // drive  
    totalCost = taxi.drive(450);  
    if (totalCost == -1) System.out.println("Not enough fuel left");  
    // refill & drive  
    taxi.refill(1.7);  
    totalCost = taxi.drive(450);  
}
```

Übungsblatt 8 - Aufgabe 1e)

■ Eine beispielhafte (musterlösungsverdächtige) Verwendung:

```
public static void main(String[] args) {  
    Taxi taxi = new Taxi(3, 0.5, 10, 50);  
    double totalCost = taxi.drive(250);  
    if (totalCost == -1) System.out.println("Not_enough_fuel_left");  
    else System.out.println("Trip_costs_" + totalCost);  
    // pay  
    if (taxi.pay(totalCost + 3.00)) System.out.println("Tipped_3,00€");  
    // drive  
    totalCost = taxi.drive(450);  
    if (totalCost == -1) System.out.println("Not_enough_fuel_left");  
    // refill & drive  
    taxi.refill(1.7);  
    totalCost = taxi.drive(450);  
    if (totalCost == -1) System.out.println("Not_enough_fuel_left");  
    else System.out.println("Trip_costs_" + totalCost);  
}
```

Eine Anwendung

Eine Anwendung

- `java` Taxi:

Eine Anwendung

- `java` Taxi:

```
Trip costs 128.0  
Received 3.00€ tip  
Tipped 3,00€  
Not enough fuel left  
Trip costs 228.0
```

Zusatzaufgabe 3/6

Zusatzaufgabe 3/6



Zusatzaufgabe 3/6

Herbert
16.01.2022 16:31



Zusatzaufgabe 3/6

Herbert
16.01.2022 16:31



Josefine
18.01.2022 9:45



Zusatzaufgabe 3/6

Herbert

16.01.2022 16:31



Josefine

18.01.2022 9:45



Ute

25.01.2022 11:05



Zusatzaufgabe 3/6

Herbert

16.01.2022 16:31



Josefine

18.01.2022 9:45



Ute

25.01.2022 11:05



Bernd

01.03.2022 15:25



Zusatzaufgabe 3/6

Herbert

16.01.2022 16:31



Josefine

18.01.2022 9:45



Ute

25.01.2022 11:05



Bernd

01.03.2022 15:25



Reservieren, Bitte.

Zusatzaufgabe 3/6

Zusatzaufgabe 3/6

- Wir werden zuerst betrachten, wie wir mit Zeitpunkten umgehen können.

Zusatzaufgabe 3/6

- Wir werden zuerst betrachten, wie wir mit Zeitpunkten umgehen können.
 - Manuell speichern als `<int> . <byte> . <byte> <byte> : <byte>`

Zusatzaufgabe 3/6

- Wir werden zuerst betrachten, wie wir mit Zeitpunkten umgehen können.
 - Manuell speichern als `<int> . <byte> . <byte> <byte> : <byte>`
 - Speichern als `String`

Zusatzaufgabe 3/6

- Wir werden zuerst betrachten, wie wir mit Zeitpunkten umgehen können.
 - Manuell speichern als `<int>.<byte>.<byte> <byte>:<byte>`
 - Speichern als `String`
 - Speichern als `java.util.Date`

Zusatzaufgabe 3/6

- Wir werden zuerst betrachten, wie wir mit Zeitpunkten umgehen können.
 - Manuell speichern als `<int>.<byte>.<byte> <byte>:<byte>`
 - Speichern als `String`
 - Speichern als `java.util.Date`
- Dann, wie wir Zeitpunkte vergleichen können.

Zusatzaufgabe 3/6

- Wir werden zuerst betrachten, wie wir mit Zeitpunkten umgehen können.
 - Manuell speichern als `<int>.<byte>.<byte> <byte>:<byte>`
 - Speichern als `String`
 - Speichern als `java.util.Date`
- Dann, wie wir Zeitpunkte vergleichen können.
- Anschließend werde ich noch etwas zu `toString` und `equals` verlieren.


```
public class CustomTime {
```

```
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month;  
    public final byte day;  
  
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month;  
    public final byte day;  
  
    public final byte hour;  
    public final byte minute;  
  
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month;  
    public final byte day;  
  
    public final byte hour;  
    public final byte minute;  
  
    public CustomTime(String date, String time) {  
  
    }  
}
```



```
public class CustomTime {  
    public final int year;  
    public final byte month;  
    public final byte day;  
  
    public final byte hour;  
    public final byte minute;  
  
    public CustomTime(String date, String time) {  
        // parse  
    }  
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month, day;  
    public final byte hour, minute;  
  
    public CustomTime(String date, String time) {  
  
  
  
  
  
  
  
  
  
    }  
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month, day;  
    public final byte hour, minute;  
  
    public CustomTime(String date, String time) {  
        this.year = Integer.parseInt(date.substring(0,4));  
  
    }  
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month, day;  
    public final byte hour, minute;  
  
    public CustomTime(String date, String time) {  
        this.year = Integer.parseInt(date.substring(0,4));  
        this.month = Byte.parseByte(date.substring(5,7));  
  
    }  
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month, day;  
    public final byte hour, minute;  
  
    public CustomTime(String date, String time) {  
        this.year = Integer.parseInt(date.substring(0,4));  
        this.month = Byte.parseByte(date.substring(5,7));  
        if(month < 0 || month > 12)  
            throw new IllegalArgumentException("0_<=_month_<=_12");  
    }  
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month, day;  
    public final byte hour, minute;  
  
    public CustomTime(String date, String time) {  
        this.year = Integer.parseInt(date.substring(0,4));  
        this.month = Byte.parseByte(date.substring(5,7));  
        if(month < 0 || month > 12)  
            throw new IllegalArgumentException("0_<=_month_<=_12");  
        // ...  
    }  
}
```

```
public class CustomTime {  
    public final int year;  
    public final byte month, day;  
    public final byte hour, minute;
```

In dieser Aufgaben können wir auch annehmen, dass das Datum „korrekt“ ist. Mit einem kleinen diesbezüglichen Kommentar entfällt damit die Prüfung (ebenso wie die Prüfung von „.“ und „:“).

```
    public CustomTime(String date, String time) {  
        this.year = Integer.parseInt(date.substring(0,4));  
        this.month = Byte.parseByte(date.substring(5,7));  
        if(month < 0 || month > 12)  
            throw new IllegalArgumentException("0_<=_month_<=_12");  
        // ...  
    }  
}
```



```
public class CustomTime {  
    public final int year;  
    public final byte month, day;  
    public final byte hour, minute;
```

In dieser Aufgaben können wir auch annehmen, dass das Datum „korrekt“ ist. Mit einem kleinen diesbezüglichen Kommentar entfällt damit die Prüfung (ebenso wie die Prüfung von „.“ und „:“).

```
    public CustomTime(String date, String time) {  
        this.year = Integer.parseInt(date.substring(0,4));  
        this.month = Byte.parseByte(date.substring(5,7));  
        if(month < 0 || month > 12)  
            throw new IllegalArgumentException("0_<=_month_<=_12");  
        // ...  
    }  
}
```



CustomTime.java

- Das Speichern ist so besonders einfach:

- Das Speichern ist so besonders einfach:

```
public class StringTime {
```

```
}
```

- Das Speichern ist so besonders einfach:

```
public class StringTime {  
    public final String date;  
    public final String time;  
  
}
```

- Das Speichern ist so besonders einfach:

```
public class StringTime {  
    public final String date;  
    public final String time;  
  
    public StringTime(String date, String time) {  
  
    }  
}
```

- Das Speichern ist so besonders einfach:

```
public class StringTime {  
    public final String date;  
    public final String time;  
  
    public StringTime(String date, String time) {  
        this.date = date;  
        this.time = time;  
    }  
}
```

- Das Speichern ist so besonders einfach:

```
public class StringTime {  
    public final String date;  
    public final String time;  
  
    public StringTime(String date, String time) {  
        this.date = date;  
        this.time = time;  
    }  
}
```

- Zur Möglichkeiten der Verrechnung kommen wir da noch.

- Das Speichern ist so besonders einfach:

```
public class StringTime {  
    public final String date;  
    public final String time;  
  
    public StringTime(String date, String time) {  
        this.date = date;  
        this.time = time;  
    }  
}
```

- Zur Möglichkeiten der Verrechnung kommen wir da noch.

StringTime.java

- Was kann das eigentlich?

- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...

- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...
- Suche: „Oracle Date“ *me pls xoxo*

- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...
- Suche: „Oracle Date“ *me pls xoxo*



- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...
- Suche: „Oracle Date“ *me pls xoxo*

- Öhm... Ich muss los.



- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...
- Suche: „Oracle Date“ *me pls xoxo*
- Öhm... Ich muss los.

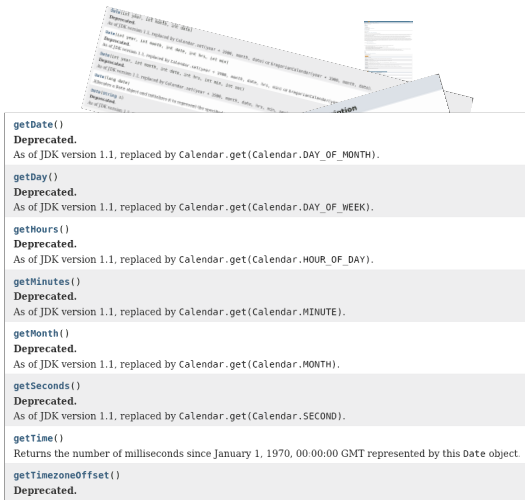


- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...
- Suche: „Oracle Date“ *me pls xoxo*
- Öhm... Ich muss los.

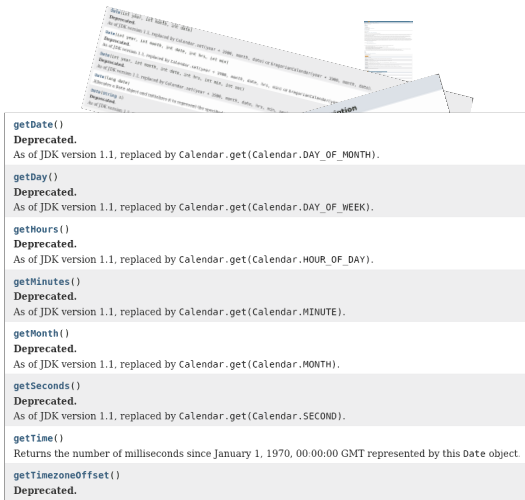


Modifier and Type	Method and Description
boolean	after(Date when) Tests if this date is after the specified date.
boolean	before(Date when) Tests if this date is before the specified date.
Object	clone() Return a copy of this object.
int	compareTo(Date anotherDate) Compares two Dates for ordering.
boolean	equals(Object obj) Compares two dates for equality.

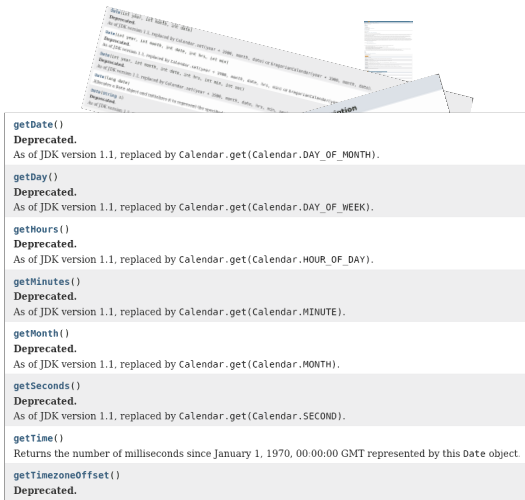
- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...
- Suche: „Oracle Date“ *me pls xoxo*
- Öhm... Ich muss los.



- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...
- Suche: „Oracle Date“ *me pls xoxo*
- Öhm... Ich muss los.
- Quasi alles Deprecated? Schon in **Java 1**?



- Was kann das eigentlich?
- So schlimm kann das ja nicht sein...
- Suche: „Oracle Date“ *me pls xoxo*
- Öhm... Ich muss los.
- Quasi alles Deprecated? Schon in **Java 1**?
- Aiii schnief Looooopf gluk maiiii Laiiiif.



- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen.

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:
`public class` `CalendarDateTime` {

}

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;
```

```
}
```

```
}
```

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;  
  
    public CalendarDateTime(String date, String time) {  
  
  
    }  
}
```

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;  
  
    public CalendarDateTime(String date, String time) {  
        CustomTime custom = new CustomTime(date, time);  
  
    }  
}
```

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;  
  
    public CalendarDateTime(String date, String time) {  
        CustomTime custom = new CustomTime(date, time);  
        GregorianCalendar calendar = new GregorianCalendar();  
  
    }  
}
```

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;  
  
    public CalendarDateTime(String date, String time) {  
        CustomTime custom = new CustomTime(date, time);  
        GregorianCalendar calendar = new GregorianCalendar();  
        calendar.set(custom.year, custom.month - 1, custom.day,  
                    custom.hour, custom.minute);  
    }  
}
```

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;  
  
    public CalendarDateTime(String date, String time) {  
        CustomTime custom = new CustomTime(date, time);  
        GregorianCalendar calendar = new GregorianCalendar();  
        calendar.set(custom.year, custom.month - 1, custom.day,  
            custom.hour, custom.minute);  
        this.date = calendar.getTime();  
    }  
}
```

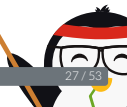
- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;  
  
    public CalendarDateTime(String date, String time) {  
        CustomTime custom = new CustomTime(date, time);  
        GregorianCalendar calendar = new GregorianCalendar();  
        calendar.set(custom.year, custom.month - 1, custom.day,  
                    custom.hour, custom.minute);  
        this.date = calendar.getTime();  
    }  
    // ...  
}
```

- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;  
  
    public CalendarDateTime(String date, String time) {  
        CustomTime custom = new CustomTime(date, time);  
        GregorianCalendar calendar = new GregorianCalendar();  
        calendar.set(custom.year, custom.month - 1, custom.day,  
            custom.hour, custom.minute);  
        this.date = calendar.getTime();  
    }  
    // ...  
}
```

Java beginnt auch bei Monaten in der Regel mit 0!

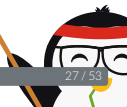


- Kurz gesagt: `Date` hält nur die Sekunden seit „Unix Epoch“ (1. Januar 1970, Mitternacht; fernab der Internationalisierung).
- Wir konstruieren es durch andere Klassen. Mit dem „Gregorianischen Kalender“:

```
public class CalendarDateTime {  
    private final Date date;  
  
    public CalendarDateTime(String date, String time) {  
        CustomTime custom = new CustomTime(date, time);  
        GregorianCalendar calendar = new GregorianCalendar();  
        calendar.set(custom.year, custom.month - 1, custom.day,  
            custom.hour, custom.minute);  
        this.date = calendar.getTime();  
    }  
    // ...  
}
```

CalendarDateTime.java

Java beginnt auch bei Monaten in der Regel mit 0!



Zusatzaufgabe 3/6

Als Java-Datum mit DateFormat

- Java kennt aber auch Datumsformate, wie das `SimpleDateFormat`:

- Java kennt aber auch Datumsformate, wie das `SimpleDateFormat`:

`SimpleDateFormat`



- Java kennt aber auch Datumsformate, wie das SimpleDateFormat:

```
public class DateFormatTime {
```

```
    // ...  
}
```

SimpleDateFormat



- Java kennt aber auch Datumsformate, wie das SimpleDateFormat:

```
public class DateFormatTime {  
    private final Date date;
```

```
    // ...  
}
```

SimpleDateFormat



- Java kennt aber auch Datumsformate, wie das SimpleDateFormat:

```
public class DateFormatTime {  
    private final Date date;
```

```
    public DateFormatTime(String date, String time) {
```

```
    }  
    // ...
```

```
}
```

SimpleDateFormat



- Java kennt aber auch Datumsformate, wie das SimpleDateFormat:

```
public class DateFormatTime {  
    private final Date date;  
  
    public final SimpleDateFormat dateFormat =  
  
    public DateFormatTime(String date, String time) {  
  
    }  
    // ...  
}
```

SimpleDateFormat



- Java kennt aber auch Datumsformate, wie das SimpleDateFormat:

```
public class DateFormatTime {  
    private final Date date;  
  
    public final SimpleDateFormat dateFormat =  
        new SimpleDateFormat("yyyy.MM.dd_HH:mm");  
  
    public DateFormatTime(String date, String time) {  
  
    }  
    // ...  
}
```

SimpleDateFormat



- Java kennt aber auch Datumsformate, wie das SimpleDateFormat:

```
public class DateFormatTime {  
    private final Date date;  
  
    public final SimpleDateFormat dateFormat =  
        new SimpleDateFormat("yyyy.MM.dd_HH:mm");  
  
    public DateFormatTime(String date, String time) {  
        this.date = dateFormat.parse(date + "_" + time);  
    }  
    // ...  
}
```

SimpleDateFormat



- Java kennt aber auch Datumsformate, wie das SimpleDateFormat:

```
public class DateFormatTime {  
    private final Date date;  
  
    public final SimpleDateFormat dateFormat =  
        new SimpleDateFormat("yyyy.MM.dd_HH:mm");  
  
    public DateFormatTime(String date, String time)  
        throws ParseException {  
        this.date = dateFormat.parse(date + "_" + time);  
    }  
    // ...  
}
```

SimpleDateFormat



- Java kennt aber auch Datumsformate, wie das SimpleDateFormat:

```
public class DateFormatTime {  
    private final Date date;  
  
    public final SimpleDateFormat dateFormat =  
        new SimpleDateFormat("yyyy.MM.dd_HH:mm");  
  
    public DateFormatTime(String date, String time)  
        throws ParseException {  
        this.date = dateFormat.parse(date + "_" + time);  
    }  
    // ...  
}
```

DateFormatTime.java

SimpleDateFormat



- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?

- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.

- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.
 - Wie viele Minuten liegen zwischen dem 28. Februar 23:59 und dem 1. März 0:05 im selben Jahr?

- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.
 - Wie viele Minuten liegen zwischen dem 28. Februar 23:59 und dem 1. März 0:05 im selben Jahr?
 - Wir müssen Schaltjahre beachten.

- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.
 - Wie viele Minuten liegen zwischen dem 28. Februar 23:59 und dem 1. März 0:05 im selben Jahr?
 - Wir müssen Schaltjahre beachten.
 - Und Schaltsekunden!

- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.
 - Wie viele Minuten liegen zwischen dem 28. Februar 23:59 und dem 1. März 0:05 im selben Jahr?
 - Wir müssen Schaltjahre beachten.
 - Und Schaltsekunden!
 - Und Säkularjahre...

- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.
 - Wie viele Minuten liegen zwischen dem 28. Februar 23:59 und dem 1. März 0:05 im selben Jahr?
 - Wir müssen Schaltjahre beachten.
 - Und Schaltsekunden!
 - Und Säkularjahre...
- In der Regel dampft man Daten daher auf ihre (Nano-/Milli-/...-)Sekunden seit einem Zeitpunkt herunter.

- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.
 - Wie viele Minuten liegen zwischen dem 28. Februar 23:59 und dem 1. März 0:05 im selben Jahr?
 - Wir müssen Schaltjahre beachten.
 - Und Schaltsekunden!
 - Und Säkularjahre...
- In der Regel dampft man Daten daher auf ihre (Nano-/Milli-/...-)Sekunden seit einem Zeitpunkt herunter.
 - `java.util.Date` tut dies bereits (Millisekunden)!

- Daten vergleichen ist einfach: Sind Jahre, Tage, ... identisch?
- Zu sagen, wie viele Minuten zwischen zwei Daten liegen, ist schwieriger.
 - Wie viele Minuten liegen zwischen dem 28. Februar 23:59 und dem 1. März 0:05 im selben Jahr?
 - Wir müssen Schaltjahre beachten.
 - Und Schaltsekunden!
 - Und Säkularjahre...
- In der Regel dampft man Daten daher auf ihre (Nano-/Milli-/...-)Sekunden seit einem Zeitpunkt herunter.
 - `java.util.Date` tut dies bereits (Millisekunden)!
 - Dies Manuell korrekt zu tun ist *sehr aufwändig* (Die zugehörige Methode `computeTime` in `GregorianCalendar` hat beispielsweise allein 200 Zeilen.)

- Die Minuten zwischen zwei Daten a und b erhalten wir so wie folgt:

- Die Minuten zwischen zwei Daten a und b erhalten wir so wie folgt:

```
long aStamp = a.getTime();
```

```
long bStamp = b.getTime();
```

- Die Minuten zwischen zwei Daten a und b erhalten wir so wie folgt:

```
long aStamp = a.getTime();  
long bStamp = b.getTime();  
double diffInMinutes = (aStamp - bStamp) / (1000 * 60d)
```

- Die Minuten zwischen zwei Daten a und b erhalten wir so wie folgt:

```
long aStamp = a.getTime();  
long bStamp = b.getTime();  
double diffInMinutes = (aStamp - bStamp) / (1000 * 60d)
```

- Magic Numbers:

- Die Minuten zwischen zwei Daten a und b erhalten wir so wie folgt:

```
long aStamp = a.getTime();  
long bStamp = b.getTime();  
double diffInMinutes = (aStamp - bStamp) / (1000 * 60d)
```

- Magic Numbers:

- 1000 Millisekunden bilden eine Sekunde.

- Die Minuten zwischen zwei Daten a und b erhalten wir so wie folgt:

```
long aStamp = a.getTime();  
long bStamp = b.getTime();  
double diffInMinutes = (aStamp - bStamp) / (1000 * 60d)
```

- Magic Numbers:

- 1000 Millisekunden bilden eine Sekunde.
- 60 Sekunden bilden eine Minute.

- Nun eine Variante mit `SimpleDateFormat`.

- Nun eine Variante mit `SimpleDateFormat`.

```
public class Reservation {
```

```
}
```


- Nun eine Variante mit SimpleDateFormat.

```
public class Reservation {  
    private final String name;  
    private final Date resDate;
```

```
}
```

- Nun eine Variante mit SimpleDateFormat.

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
  
    private final SimpleDateFormat dateFormat =  
        new SimpleDateFormat("yyyy.MM.dd_HH:mm");  
  
}
```

- Nun eine Variante mit SimpleDateFormat.

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
  
    private final SimpleDateFormat dateFormat =  
        new SimpleDateFormat("yyyy.MM.dd_HH:mm");  
    public Reservation(String name, String date, String time)  
        throws ParseException {  
  
    }  
}
```

- Nun eine Variante mit SimpleDateFormat.

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
  
    private final SimpleDateFormat dateFormat =  
        new SimpleDateFormat("yyyy.MM.dd_HH:mm");  
    public Reservation(String name, String date, String time)  
        throws ParseException {  
        this.name = name;  
        this.resDate = dateFormat.parse(date + "_" + time);  
    }  
}
```

- Nun eine Variante mit SimpleDateFormat.

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
  
    private final SimpleDateFormat dateFormat =  
        new SimpleDateFormat("yyyy.MM.dd_HH:mm");  
    public Reservation(String name, String date, String time)  
        throws ParseException {  
        this.name = name;  
        this.resDate = dateFormat.parse(date + "_" + time);  
    }  
}
```

Warum nicht **static**? Nun, das hat etwas mit *Threads* zu tun...



Mini-Exkurs: Mit Ausnahmen umgehen

Mini-Exkurs: Mit Ausnahmen umgehen

- `ParseException` ist eine „checked-Exception“ \Rightarrow wir müssen sie behandeln.

Mini-Exkurs: Mit Ausnahmen umgehen

- `ParseException` ist eine „checked-Exception“ \Rightarrow wir müssen sie behandeln.
- In dieser Aufgabe können wir eine korrekte Eingabe annehmen.

Mini-Exkurs: Mit Ausnahmen umgehen

- `ParseException` ist eine „checked-Exception“ \Rightarrow wir müssen sie behandeln.
- In dieser Aufgabe können wir eine korrekte Eingabe annehmen.
- Dies erlaubt ein Unterdrücken des Fehlers:

Mini-Exkurs: Mit Ausnahmen umgehen

- `ParseException` ist eine „checked-Exception“ \Rightarrow wir müssen sie behandeln.
- In dieser Aufgabe können wir eine korrekte Eingabe annehmen.
- Dies erlaubt ein Unterdrücken des Fehlers:

```
try {  
    ...  
} catch (ParseException ex) {  
    ex.printStackTrace();  
}
```

Mini-Exkurs: Mit Ausnahmen umgehen

- `ParseException` ist eine „checked-Exception“ \Rightarrow wir müssen sie behandeln.
- In dieser Aufgabe können wir eine korrekte Eingabe annehmen.
- Dies erlaubt ein Unterdrücken des Fehlers:

```
try {  
    ...  
} catch (ParseException ex) {  
    ex.printStackTrace();  
}
```

- Allgemein ist aber das Weiterreichen sicherer:

Mini-Exkurs: Mit Ausnahmen umgehen

- `ParseException` ist eine „checked-Exception“ \Rightarrow wir müssen sie behandeln.
- In dieser Aufgabe können wir eine korrekte Eingabe annehmen.
- Dies erlaubt ein Unterdrücken des Fehlers:

```
try {  
    ...  
} catch (ParseException ex) {  
    ex.printStackTrace();  
}
```

- Allgemein ist aber das Weiterreichen sicherer:

```
public Reservation(...) throws ParseException {
```


- `toString` ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.

- `toString` ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.
- Mit `String::format` können wir uns aber interessanteren Freuden widmen:

- `toString` ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.
- Mit `String::format` können wir uns aber interessanteren Freuden widmen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
  
}
```


- toString ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.
- Mit `String::format` können wir uns aber interessanteren Freuden widmen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public String toString() {  
  
    }  
}
```

- toString ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.
- Mit `String.format` können wir uns aber interessanteren Freuden widmen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public String toString() {  
        return String.format(Locale.ENGLISH, // March statt März  
                               "  
    };  
}  
}
```

- toString ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.
- Mit `String.format` können wir uns aber interessanteren Freuden widmen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public String toString() {  
        return String.format(Locale.ENGLISH, // March statt März  
            "%s reserved for %2$tB %2$td, %2$tY at %2$tH:%2$tM",  
            name,  
            resDate);  
    }  
}
```

- toString ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.
- Mit `String::format` können wir uns aber interessanteren Freuden widmen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public String toString() {  
        return String.format(Locale.ENGLISH, // March statt März  
            "%s reserved for %2$tB %2$td, %2$tY at %2$tH:%2$tM",  
            this.name, this.resDate  
        );  
    }  
}
```

- toString ist einfach, wenn wir zusätzlich noch die übergebenen Daten speichern.
- Mit `String.format` können wir uns aber interessanteren Freuden widmen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public String toString() {  
        return String.format(Locale.ENGLISH, // March statt März  
            "%s reserved for %2$tB %2$td, %2$tY at %2$tH:%2$tM",  
            this.name, this.resDate  
        );  
    }  
}
```



Format String Syntax

- Aber halt. Es war ja `toString(Reservation)` gefordert.

- Aber halt. Es war ja `toString(Reservation)` gefordert.
- Nun, wir könnten dies wie folgt lösen (das ist aber unüblich):

- Aber halt. Es war ja `toString(Reservation)` gefordert.
- Nun, wir könnten dies wie folgt lösen (das ist aber unüblich):

```
public class Reservation {  
    private final String name;  
    private final Date resDate;
```

```
}
```

- Aber halt. Es war ja `toString(Reservation)` gefordert.
- Nun, wir könnten dies wie folgt lösen (das ist aber unüblich):

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public static String toString(Reservation r) {  
  
    }  
  
}
```

- Aber halt. Es war ja `toString(Reservation)` gefordert.
- Nun, wir könnten dies wie folgt lösen (das ist aber unüblich):

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public static String toString(Reservation r) {  
        return String.format(Locale.ENGLISH, // March statt März  
  
        );  
    }  
  
}
```

- Aber halt. Es war ja `toString(Reservation)` gefordert.
- Nun, wir könnten dies wie folgt lösen (das ist aber unüblich):

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public static String toString(Reservation r) {  
        return String.format(Locale.ENGLISH, // March statt März  
            "%s reserved for %2$tB %2$td, %2$tY at %2$tH:%2$tM",  
            );  
    }  
}
```

- Aber halt. Es war ja `toString(Reservation)` gefordert.
- Nun, wir könnten dies wie folgt lösen (das ist aber unüblich):

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public static String toString(Reservation r) {  
        return String.format(Locale.ENGLISH, // March statt März  
            "%s reserved for %2$tB %2$td, %2$tY at %2$tH:%2$tM",  
            r.name, r.resDate  
        );  
    }  
}
```

- Aber halt. Es war ja `toString(Reservation)` gefordert.
- Nun, wir könnten dies wie folgt lösen (das ist aber unüblich):

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public static String toString(Reservation r) {  
        return String.format(Locale.ENGLISH, // March statt März  
            "%s reserved for %2$tB %2$td, %2$tY at %2$tH:%2$tM",  
            r.name, r.resDate  
        );  
    }  
    public String toString() { toString(this); }  
}
```


- Gibt es Alternativen?

- Gibt es Alternativen?
- Japp. Die gibt es. Beispielsweise mit `Calendar`:

- Gibt es Alternativen?
- Japp. Die gibt es. Beispielsweise mit `Calendar`:

```
public static String toString(Reservation reservation) {
```

```
}
```

- Gibt es Alternativen?
- Japp. Die gibt es. Beispielsweise mit `Calendar`:

```
public static String toString(Reservation reservation) {  
    Calendar calendar = Calendar.getInstance();
```

```
}
```

- Gibt es Alternativen?
- Japp. Die gibt es. Beispielsweise mit `Calendar`:

```
public static String toString(Reservation reservation) {  
    Calendar calendar = Calendar.getInstance();  
    calendar.setTime(reservation.resDate);
```

```
}
```

- Gibt es Alternativen?
- Japp. Die gibt es. Beispielsweise mit `Calendar`:

```
public static String toString(Reservation reservation) {  
    Calendar calendar = Calendar.getInstance();  
    calendar.setTime(reservation.resDate);  
    return "Reservierung_für_" + reservation.name  
  
}
```

- Gibt es Alternativen?
- Japp. Die gibt es. Beispielsweise mit `Calendar`:

```
public static String toString(Reservation reservation) {  
    Calendar calendar = Calendar.getInstance();  
    calendar.setTime(reservation.resDate);  
    return "Reservierung_für_" + reservation.name  
        + "_am_" + calendar.get(Calendar.YEAR) + "."  
        + (calendar.get(Calendar.MONTH) + 1) + "."  
        + calendar.get(Calendar.DAY_OF_MONTH)  
        + "_um_" + calendar.get(Calendar.HOUR_OF_DAY) + ":"  
        + calendar.get(Calendar.MINUTE)  
        + "_Uhr.";  
}
```


- Das `SimpleDateFormat` kann das aber auch!

- Das SimpleDateFormat kann das aber auch!

```
private final String name;  
private final Date resDate;  
private final SimpleDateFormat dateFormat =  
    new SimpleDateFormat("yyyy.MM.dd HH:mm");
```

- Das SimpleDateFormat kann das aber auch!

```
private final String name;  
private final Date resDate;  
private final SimpleDateFormat dateFormat =  
    new SimpleDateFormat("yyyy.MM.dd HH:mm");  
  
public static String toString(Reservation reservation) {  
  
    }  
}
```

- Entscheidet selbst :D

- Das SimpleDateFormat kann das aber auch!

```
private final String name;  
private final Date resDate;  
private final SimpleDateFormat dateFormat =  
    new SimpleDateFormat("yyyy.MM.dd HH:mm");  
  
public static String toString(Reservation reservation) {  
    return "Reservierung_für_" + reservation.name  
        + "_am_" + dateFormat.format(reservation.resDate);  
}
```

- Entscheidet selbst :D

- Felder vergleichen:

■ Felder vergleichen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
  
}
```

- Felder vergleichen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean equals(Reservation reservation) {  
  
    }  
}
```

- Felder vergleichen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean equals(Reservation reservation) {  
        return this == reservation ||  
    }  
}
```


- Felder vergleichen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean equals(Reservation reservation) {  
        return this == reservation ||  
            (Objects.equals(this.resDate, reservation.resDate)  
            }  
}
```

- Felder vergleichen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean equals(Reservation reservation) {  
        return this == reservation ||  
            (Objects.equals(this.resDate, reservation.resDate)  
            && Objects.equals(this.name, reservation.name));  
    }  
}
```

- Felder vergleichen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean equals(Reservation reservation) {  
        return this == reservation ||  
            (Objects.equals(this.resDate, reservation.resDate)  
             && Objects.equals(this.name, reservation.name));  
    }  
}
```

- Der ==-Vergleich ist nicht notwendig, zeigt aber die stärkere Aussage von „identisch“ gegenüber „gleich“.

- Felder vergleichen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean equals(Reservation reservation) {  
        return this == reservation ||  
            (Objects.equals(this.resDate, reservation.resDate)  
            && Objects.equals(this.name, reservation.name));  
    }  
}
```

- Der ==-Vergleich ist nicht notwendig, zeigt aber die stärkere Aussage von „identisch“ gegenüber „gleich“.
- `Object.equals(a, b)` entspricht `a.equals(b)`, kann aber auch mit `a == null` umgehen!

- Eine Form der Verwendung:

■ Eine Form der Verwendung:

```
public class Reservation {  
    public Reservation(String n, String d, String t) throws ParseException;  
    public String toString();  
    public boolean equals(Reservation reservation);
```

```
}  
}
```

■ Eine Form der Verwendung:

```
public class Reservation {  
    public Reservation(String n, String d, String t) throws ParseException;  
    public String toString();  
    public boolean equals(Reservation reservation);  
  
    public static void main(String[] args) throws ParseException {  
  
    }  
}
```


■ Eine Form der Verwendung:

```
public class Reservation {  
    public Reservation(String n, String d, String t) throws ParseException;  
    public String toString();  
    public boolean equals(Reservation reservation);  
  
    public static void main(String[] args) throws ParseException {  
        Reservation r1 = new Reservation("Hugo", "2022.10.15", "14:32");  
        System.out.println(r1);  
  
    }  
}
```

■ Eine Form der Verwendung:

```
public class Reservation {  
    public Reservation(String n, String d, String t) throws ParseException;  
    public String toString();  
    public boolean equals(Reservation reservation);  
  
    public static void main(String[] args) throws ParseException {  
        Reservation r1 = new Reservation("Hugo", "2022.10.15", "14:32");  
        System.out.println(r1);  
        System.out.println("same:_" + r1.equals(r1));  
  
    }  
}
```

■ Eine Form der Verwendung:

```
public class Reservation {  
    public Reservation(String n, String d, String t) throws ParseException;  
    public String toString();  
    public boolean equals(Reservation reservation);  
  
    public static void main(String[] args) throws ParseException {  
        Reservation r1 = new Reservation("Hugo", "2022.10.15", "14:32");  
        System.out.println(r1);  
        System.out.println("same:_" + r1.equals(r1));  
        Reservation other = new Reservation("Hugo", "2022.10.15", "14:32");  
        System.out.println("other:_" + r1.equals(other));  
  
    }  
}
```

■ Eine Form der Verwendung:

```
public class Reservation {  
    public Reservation(String n, String d, String t) throws ParseException;  
    public String toString();  
    public boolean equals(Reservation reservation);  
  
    public static void main(String[] args) throws ParseException {  
        Reservation r1 = new Reservation("Hugo", "2022.10.15", "14:32");  
        System.out.println(r1);  
        System.out.println("same:_" + r1.equals(r1));  
        Reservation other = new Reservation("Hugo", "2022.10.15", "14:32");  
        System.out.println("other:_" + r1.equals(other));  
        Reservation diff = new Reservation("Hugo", "2022.01.15", "14:33");  
        System.out.println("different:_" + r1.equals(diff));  
    }  
}
```


- Beim checkIn können wir das Minutenintervall leicht prüfen:

- Beim checkIn können wir das Minutenintervall leicht prüfen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;
```

```
}
```

- Beim checkIn können wir das Minutenintervall leicht prüfen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean checkIn(Reservation reservation) {  
  
  
  
  
  
  
    }  
}
```


- Beim checkIn können wir das Minutenintervall leicht prüfen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean checkIn(Reservation reservation) {  
        long them = reservation.resDate.getTime();  
        long we = this.resDate.getTime();  
  
    }  
}
```

- Beim checkIn können wir das Minutenintervall leicht prüfen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean checkIn(Reservation reservation) {  
        long them = reservation.resDate.getTime();  
        long we = this.resDate.getTime();  
        double diffInMinutes = (them - we) / (MS_IN_S * S_IN_MS);  
  
    }  
}
```

- Beim checkIn können wir das Minutenintervall leicht prüfen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean checkIn(Reservation reservation) {  
        long them = reservation.resDate.getTime();  
        long we = this.resDate.getTime();  
        double diffInMinutes = (them - we) / (MS_IN_S * S_IN_MS);  
        // is it between 5 and 15 minutes?  
        return diffInMinutes >= -5 && diffInMinutes <= 15 ;  
    }  
}
```

- Beim checkIn können wir das Minutenintervall leicht prüfen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean checkIn(Reservation reservation) {  
        long them = reservation.resDate.getTime();  
        long we = this.resDate.getTime();  
        double diffInMinutes = (them - we) / (MS_IN_S * S_IN_MS);  
        // is it between 5 and 15 minutes?  
        return diffInMinutes >= -5 && diffInMinutes <= 15 ;  
    }  
}
```

- Dabei ist MS_IN_S = 1000 und S_IN_MS = 60.

- Beim checkIn können wir das Minutenintervall leicht prüfen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean checkIn(Reservation reservation) {  
        long them = reservation.resDate.getTime();  
        long we = this.resDate.getTime();  
        double diffInMinutes = (them - we) / (MS_IN_S * S_IN_MS);  
        // is it between 5 and 15 minutes?  
        return diffInMinutes >= -5 && diffInMinutes <= 15 ;  
    }  
}
```

- Dabei ist MS_IN_S = 1000 und S_IN_MS = 60.
- Optional auch Objects.equals(this.name, reservation.name) um den Namen zu prüfen.

- Beim checkIn können wir das Minutenintervall leicht prüfen:

```
public class Reservation {  
    private final String name;  
    private final Date resDate;  
    public boolean checkIn(Reservation reservation) {  
        long them = reservation.resDate.getTime();  
        long we = this.resDate.getTime();  
        double diffInMinutes = (them - we) / (MS_IN_S * S_IN_MS);  
        // is it between 5 and 15 minutes?  
        return diffInMinutes >= -5 && diffInMinutes <= 15 ;  
    }  
}
```

- Dabei ist MS_IN_S = 1000 und S_IN_MS = 60.
- Optional auch Objects.equals(this.name, reservation.name) um den Namen zu prüfen.

Reservation.java

- Was noch funktioniert hätte...

- Was noch funktioniert hätte...
- Anstelle der Millisekunden können wir auch neue Daten konstruieren und diese mit `Date::before` und `Date::after` beziehungsweise `Date::compareTo` vergleichen.

- Was noch funktioniert hätte...
- Anstelle der Millisekunden können wir auch neue Daten konstruieren und diese mit `Date::before` und `Date::after` beziehungsweise `Date::compareTo` vergleichen.
- Es gibt noch viele andere Klassen – wie `LocalTime`, `LocalDateTime`, `LocalDate`, sowie `Duration` – die man benutzen kann!

Präsenzaufgabe

2

Listentausch, ...

Präsenzaufgabe

2

Listentausch, ...

Implementieren Sie eine einfach verkettete Liste, die Integerwerte speichern kann.

2

Listentausch, ...

Implementieren Sie eine einfach verkettete Liste, die Integerwerte speichern kann. Implementieren Sie eine Methode (+ Hilfsmethoden), die zwei Listenelemente anhand ihrer Indizes vertauscht.

2

Listentausch, ...

Implementieren Sie eine einfach verkettete Liste, die Integerwerte speichern kann. Implementieren Sie eine Methode (+ Hilfsmethoden), die zwei Listenelemente anhand ihrer Indizes vertauscht. Die Liste fängt wie ein Array bei 0 zu zählen an. Beachten Sie auch Randbedingungen, wie eine leere Liste.

Präsenzaufgabe - Lösung

Präsenzaufgabe - Lösung

- Wir konstruieren eine Klasse für das Element.

Präsenzaufgabe - Lösung

- Wir konstruieren eine Klasse für das Element.

```
public class Element {
```

```
}
```

Präsenzaufgabe - Lösung

- Wir konstruieren eine Klasse für das Element.

```
public class Element {  
    int value;  
    Element next;  
  
}
```

Präsenzaufgabe - Lösung

- Wir konstruieren eine Klasse für das Element.

```
public class Element {  
    int value;  
    Element next;  
  
    public Element (int value, Element next) {  
  
    }  
  
}
```

Präsenzaufgabe - Lösung

- Wir konstruieren eine Klasse für das Element.

```
public class Element {  
    int value;  
    Element next;  
  
    public Element (int value, Element next) {  
        this.value = value;  
        this.next = next;  
    }  
  
}
```

Präsenzaufgabe - Lösung

- Wir konstruieren eine Klasse für das Element.

```
public class Element {  
    int value;  
    Element next;  
  
    public Element (int value, Element next) {  
        this.value = value;  
        this.next = next;  
    }  
}
```



}

Präsenzaufgabe - Lösung, II

Präsenzaufgabe - Lösung, II

- Wir benötigen eine weitere Methode:

Präsenzaufgabe - Lösung, II

- Wir benötigen eine weitere Methode:

```
public class Element {
```

```
}
```


Präsenzaufgabe - Lösung, II

- Wir benötigen eine weitere Methode:

```
public class Element {
```

```
// ...
```

```
public Element findBefore(int x) {
```

```
}
```

```
}
```

Präsenzaufgabe - Lösung, II

- Wir benötigen eine weitere Methode:

```
public class Element {  
  
    // ...  
    public Element findBefore(int x) {  
        if(x > 1) { // 0 & 1  
  
        } else return this;  
    }  
}
```

Präsenzaufgabe - Lösung, II

- Wir benötigen eine weitere Methode:

```
public class Element {  
  
    // ...  
    public Element findBefore(int x) {  
        if(x > 1) { // 0 & 1  
            if(next != null)  
  
                } else return this;  
        }  
    }
```

Präsenzaufgabe - Lösung, II

- Wir benötigen eine weitere Methode:

```
public class Element {  
  
    // ...  
    public Element findBefore(int x) {  
        if(x > 1) { // 0 & 1  
            if(next != null)  
                return next.findBefore(x-1);  
        } else return this;  
    }  
}
```

Präsenzaufgabe - Lösung, II

- Wir benötigen eine weitere Methode:

```
public class Element {  
  
    // ...  
    public Element findBefore(int x) {  
        if(x > 1) { // 0 & 1  
            if(next != null)  
                return next.findBefore(x-1);  
            else return null; // nicht gefunden  
        } else return this;  
    }  
}
```

Präsenzaufgabe - Lösung, III

Präsenzaufgabe - Lösung, III

- Wir verwalten sie in einer Listenklasse:

Präsenzaufgabe - Lösung, III

- Wir verwalten sie in einer Listenklasse:

```
public class List {
```

```
}
```


Präsenzaufgabe - Lösung, III

- Wir verwalten sie in einer Listenklasse:

```
public class List {  
    Element head;
```

```
}
```

Präsenzaufgabe - Lösung, III

- Wir verwalten sie in einer Listenklasse:

```
public class List {  
    Element head;  
    public List() { head = null;}
```

```
}
```

Präsenzaufgabe - Lösung, III

- Wir verwalten sie in einer Listenklasse:

```
public class List {  
    Element head;  
    public List() { head = null;}  
  
    public Element findBefore(int x){  
  
    }  
  
}
```

Präsenzaufgabe - Lösung, III

- Wir verwalten sie in einer Listenklasse:

```
public class List {  
    Element head;  
    public List() { head = null;}  
  
    public Element findBefore(int x){  
        if(head == null) return null;  
    }  
  
}
```

Präsenzaufgabe - Lösung, III

- Wir verwalten sie in einer Listenklasse:

```
public class List {  
    Element head;  
    public List() { head = null;}  
  
    public Element findBefore(int x){  
        if(head == null) return null;  
        else head.findBefore(x);  
    }  
  
}
```

Präsenzaufgabe - Lösung, III

- Wir verwalten sie in einer Listenklasse:

```
public class List {  
    Element head;  
    public List() { head = null;}  
  
    public Element findBefore(int x){  
        if(head == null) return null;  
        else head.findBefore(x);  
    }  
  
    // ...  
}
```

Präsenzaufgabe - Lösung, IV

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich
```

```
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
  
    }  
  
}
```

```
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
}
```

```
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);
```

```
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);
```

```
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
  
    }  
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
  
        }  
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
    }  
}
```


Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
    }  
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
  
        }  
    }  
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
        }  
    }  
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
        }  
    }  
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
        }  
    }  
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
        }  
    }  
}
```

Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
        }  
    }  
}
```

Präsenzaufgabe - Lösung, IV

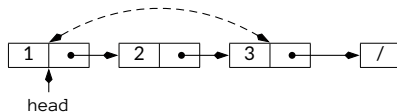
- Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```


Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

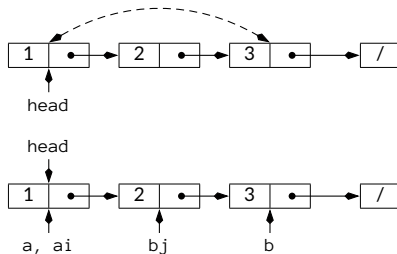
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

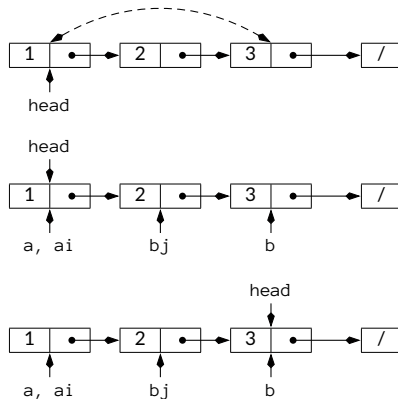
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, IV

- Nun können wir die Tauschoperation implementieren:

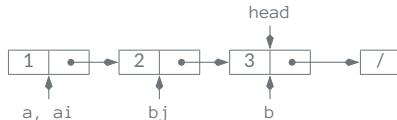
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, V

■ Nun können wir die Tauschoperation implementieren:

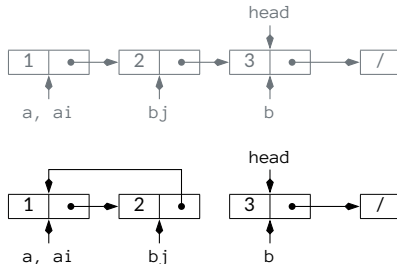
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, V

■ Nun können wir die Tauschoperation implementieren:

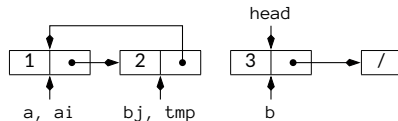
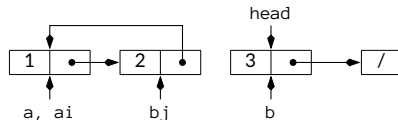
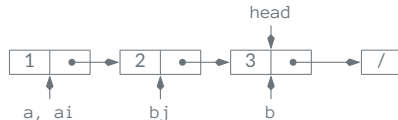
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, V

■ Nun können wir die Tauschoperation implementieren:

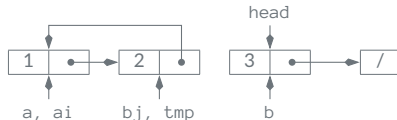
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, VI

■ Nun können wir die Tauschoperation implementieren:

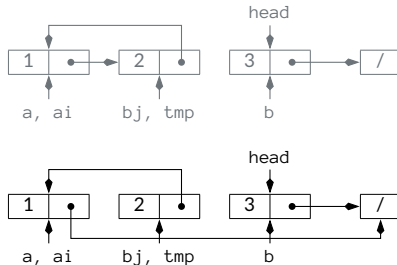
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, VI

■ Nun können wir die Tauschoperation implementieren:

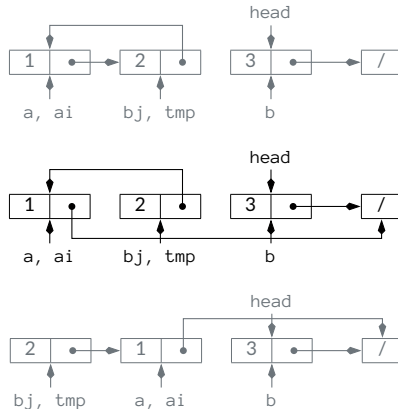
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, VI

■ Nun können wir die Tauschoperation implementieren:

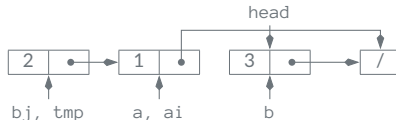
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, VII

■ Nun können wir die Tauschoperation implementieren:

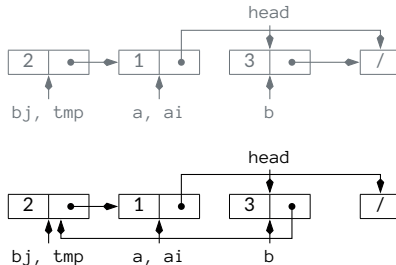
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, VII

■ Nun können wir die Tauschoperation implementieren:

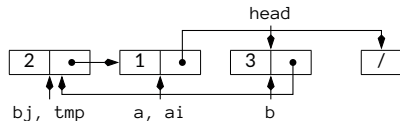
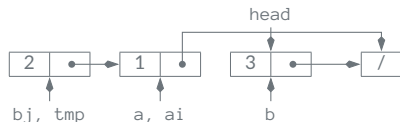
```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, VII

■ Nun können wir die Tauschoperation implementieren:

```
public void swap(int i, int j) {  
    if(i == j) return; // Gleich  
    if(i > j) { // Aufsteigend sortiert:  
        int tmp = i; i = j; j = tmp;  
    }  
    Element ai = findBefore(i);  
    Element bj = findBefore(j);  
    if(ai != null && bj != null) { // In Liste  
        Element b = bj.next, a;  
        if(i == 0) a = head; // Kopf  
        else a = ai.next;  
        if(a != null && b != null) { // Exist.  
            if(i == 0) head = b;  
            else ai.next = b;  
            // Swap  
            bj.next = a;  
            Element tmp = a.next;  
            a.next = b.next; b.next = tmp;  
        }  
    }  
}
```



Präsenzaufgabe - Lösung, VIII

- Übung: versucht mit Hilfsmethoden wie `swapElements(Element, Element)` oder `elementsInList(Element, Element), ...` die Struktur zu vereinfachen.

Präsenzaufgabe, 2

3

It is complex, isn't it?

Präsenzaufgabe, 2

3

It is complex, isn't it?

Sie sehen gleich ein Programm, welches ein Array an Integer-Werten sortiert. Ihre Aufgaben sind die folgenden:

Präsenzaufgabe, 2

3

It is complex, isn't it?

Sie sehen gleich ein Programm, welches ein Array an Integer-Werten sortiert. Ihre Aufgaben sind die folgenden:

1. Schätzen Sie die Laufzeit des folgenden Codes gemäß der \mathcal{O} -Notation ab.

Präsenzaufgabe, 2

3

It is complex, isn't it?

Sie sehen gleich ein Programm, welches ein Array an Integer-Werten sortiert. Ihre Aufgaben sind die folgenden:

1. Schätzen Sie die Laufzeit des folgenden Codes gemäß der \mathcal{O} -Notation ab.
2. Welchen „Trick“ nutzt der Algorithmus?

Präsenzaufgabe, 2

3

It is complex, isn't it?

Sie sehen gleich ein Programm, welches ein Array an Integer-Werten sortiert. Ihre Aufgaben sind die folgenden:

1. Schätzen Sie die Laufzeit des folgenden Codes gemäß der \mathcal{O} -Notation ab.
2. Welchen „Trick“ nutzt der Algorithmus?
3. Welches Wissen über die Domäne macht sich der Algorithmus zu Nutze?
Welche Nachteile bringt das mit sich?

Sie sehen gleich ein Programm, welches ein Array an Integer-Werten sortiert. Ihre Aufgaben sind die folgenden:

1. Schätzen Sie die Laufzeit des folgenden Codes gemäß der \mathcal{O} -Notation ab.
2. Welchen „Trick“ nutzt der Algorithmus?
3. Welches Wissen über die Domäne macht sich der Algorithmus zu Nutze? Welche Nachteile bringt das mit sich?
4. Geben Sie ein Beispiel an, für das dieser konkrete Algorithmus viel mehr Zeit braucht als notwendig!

Präsenzaufgabe, 2

3

It is complex, isn't it?

Sie sehen gleich ein Programm, welches ein Array an Integer-Werten sortiert. Ihre Aufgaben sind die folgenden:

1. Schätzen Sie die Laufzeit des folgenden Codes gemäß der \mathcal{O} -Notation ab.
2. Welchen „Trick“ nutzt der Algorithmus?
3. Welches Wissen über die Domäne macht sich der Algorithmus zu Nutze? Welche Nachteile bringt das mit sich?
4. Geben Sie ein Beispiel an, für das dieser konkrete Algorithmus viel mehr Zeit braucht als notwendig!
5. Für welche Anwendungsfälle ist der Algorithmus gut geeignet?

Präsenzaufgabe, 2

3

It's complex, isn't it?

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {
```

```
}
```

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {  
    if (min > max) throw new IllegalArgumentException("Intervall [min,max]");
```

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {
    if (min > max) throw new IllegalArgumentException("Intervall [min, max]");
    int[] counts = new int[max - min + 1];
}
```


Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {
    if (min > max) throw new IllegalArgumentException("Intervall [min, max]");
    int[] counts = new int[max - min + 1];
    for(int e : arr) {

    }
}
```

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {  
    if (min > max) throw new IllegalArgumentException("Intervall[" + min + ", " + max + "]");  
    int[] counts = new int[max - min + 1];  
    for(int e : arr) {  
        if(!(min <= e && e <= max)) throw new IllegalArgumentException("Element außerhalb Intervall.");  
    }  
  
}
```

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {  
    if (min > max) throw new IllegalArgumentException("Intervall[" + min + ", " + max + "]");  
    int[] counts = new int[max - min + 1];  
    for(int e : arr) {  
        if(!(min <= e && e <= max)) throw new IllegalArgumentException("Element außerhalb Intervall.");  
        counts[e - min]++;  
    }  
  
}
```

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {  
    if (min > max) throw new IllegalArgumentException("Intervall[" + min + ", " + max + "]");  
    int[] counts = new int[max - min + 1];  
    for(int e : arr) {  
        if(!(min <= e && e <= max)) throw new IllegalArgumentException("Element außerhalb Intervall.");  
        counts[e - min]++;  
    }  
  
    int k = 0;  
  
}
```

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {  
    if (min > max) throw new IllegalArgumentException("Intervall[" + min + ", " + max + "]");  
    int[] counts = new int[max - min + 1];  
    for(int e : arr) {  
        if(!(min <= e && e <= max)) throw new IllegalArgumentException("Element außerhalb Intervall.");  
        counts[e - min]++;  
    }  
  
    int k = 0;  
    for(int i = 0; i < arr.length; i++) {  
  
    }  
}
```

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {  
    if (min > max) throw new IllegalArgumentException("Intervall[" + min + ", " + max + "]");  
    int[] counts = new int[max - min + 1];  
    for(int e : arr) {  
        if(!(min <= e && e <= max)) throw new IllegalArgumentException("Element außerhalb Intervall.");  
        counts[e - min]++;  
    }  
  
    int k = 0;  
    for(int i = 0; i < arr.length; i++) {  
        while(counts[k] > 0) k++;  
        arr[i] = k + min;  
        counts[k]--;  
    }  
}
```

Präsenzaufgabe, 2

3

It's complex, isn't it?

```
public static void sort(int min, int max, int[] arr) {  
    if (min > max) throw new IllegalArgumentException("Intervall[" + min + ", " + max + "]");  
    int[] counts = new int[max - min + 1];  
    for(int e : arr) {  
        if(!(min <= e && e <= max)) throw new IllegalArgumentException("Element außerhalb Intervall.");  
        counts[e - min]++;  
    }  
  
    int k = 0;  
    for(int i = 0; i < arr.length; i++) {  
        while(counts[k] <= 0) k++;  
        arr[i] = k + min;  
        counts[k]--;  
    }  
}
```

1. (Worst-Case) Laufzeit in \mathcal{O} -Notation.
2. Welchen „Trick“ nutzt der Algorithmus?
3. Welches Domänenwissen nutzt der Algorithmus? Welche Nachteile hat das?
4. Beispiel, das viel mehr Zeit braucht als notwendig!
5. Für welche Anwendungsfälle ist er gut geeignet?

Präsenzaufgabe, 2 - Lösung

Präsenzaufgabe, 2 - Lösung

1. Die Laufzeit skaliert linear in der Länge des Arrays und der Differenz $\text{max} - \text{min}$ (Traversierung von counts). Damit befinden wir uns in $\mathcal{O}(\max(\text{arr.length}, (\text{max} - \text{min})))$. (Das innere While wird durch k nur genau $\text{max} - \text{min}$ oft durchlaufen.)

Präsenzaufgabe, 2 - Lösung

1. Die Laufzeit skaliert linear in der Länge des Arrays und der Differenz $\max - \min$ (Traversierung von `counts`). Damit befinden wir uns in $\mathcal{O}(\max(\text{arr.length}, (\max - \min)))$. (Das innere While wird durch `k` nur genau $\max - \min$ oft durchlaufen.)
2. Der Algorithmus zählt (in `counts`), wie oft eine Zahl im Intervall vorkommt und reproduziert so das Array (er ist also nicht stabil).

Präsenzaufgabe, 2 - Lösung

1. Die Laufzeit skaliert linear in der Länge des Arrays und der Differenz $\max - \min$ (Traversierung von `counts`). Damit befinden wir uns in $\mathcal{O}(\max(\text{arr.length}, (\max - \min)))$. (Das innere While wird durch `k` nur genau $\max - \min$ oft durchlaufen.)
2. Der Algorithmus zählt (in `counts`), wie oft eine Zahl im Intervall vorkommt und reproduziert so das Array (er ist also nicht stabil).
3. Er nutzt das Wissen über ganze Zahlen \mathbb{Q} , sowie die im Array enthaltenen Extremwerte! Im Zweifelsfall kennen wir z.B. $\max(\text{arr})$ gar nicht.

Präsenzaufgabe, 2 - Lösung

1. Die Laufzeit skaliert linear in der Länge des Arrays und der Differenz $\max - \min$ (Traversierung von `counts`). Damit befinden wir uns in $\mathcal{O}(\max(\text{arr.length}, (\max - \min)))$. (Das innere While wird durch `k` nur genau $\max - \min$ oft durchlaufen.)
2. Der Algorithmus zählt (in `counts`), wie oft eine Zahl im Intervall vorkommt und reproduziert so das Array (er ist also nicht stabil).
3. Er nutzt das Wissen über ganze Zahlen \mathbb{Q} , sowie die im Array enthaltenen Extremwerte! Im Zweifelsfall kennen wir z.B. `max(arr)` gar nicht.
4. Ein Array bei dem `arr.length` \ll $\max - \min$. So wird ein unnötig großes `counts`-Array traversiert. Beispiel: `sort(0, 42_000, new int[] { 3, 0, 40_000 })`.

Präsenzaufgabe, 2 - Lösung

1. Die Laufzeit skaliert linear in der Länge des Arrays und der Differenz $\max - \min$ (Traversierung von `counts`). Damit befinden wir uns in $\mathcal{O}(\max(\text{arr.length}, (\max - \min)))$. (Das innere While wird durch `k` nur genau $\max - \min$ oft durchlaufen.)
2. Der Algorithmus zählt (in `counts`), wie oft eine Zahl im Intervall vorkommt und reproduziert so das Array (er ist also nicht stabil).
3. Er nutzt das Wissen über ganze Zahlen \mathbb{Q} , sowie die im Array enthaltenen Extremwerte! Im Zweifelsfall kennen wir z.B. `max(arr)` gar nicht.
4. Ein Array bei dem `arr.length` \ll $\max - \min$. So wird ein unnötig großes `counts`-Array traversiert. Beispiel: `sort(0, 42_000, new int[] {3, 0, 40_000})`.
5. Arrays mit kleinem Wertebereich, welches zudem möglichst gleichverteilt alle Werte aus diesem wertebereich abdeckt.

