2

Diese Worte wusch ich habe

Tutorium Zwei

2

Diese Worte [illegible]uscht getau[illegible] ich habe

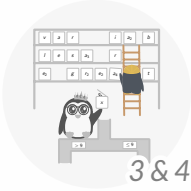
Tutorium Zwei

Theorie

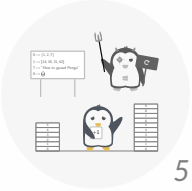
2

13

Grundlagen



3 & 4



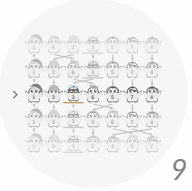
5

6

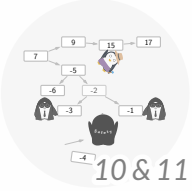
Vertiefungen

7 & 12

8



9



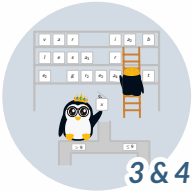
10 & 11

Theorie

2

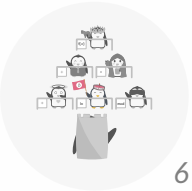
13

Grundlagen



3 & 4

5

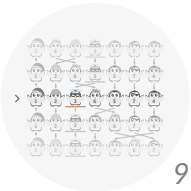


6

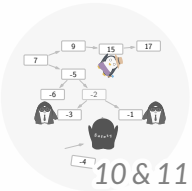
Vertiefungen

7 & 12

8



9



10 & 11

Kurzwiederholung

Kurzwiederholung

- Algorithmenkonstruktion

Kurzwiederholung

- Algorithmenkonstruktion
  - Spezifikation
    - Begriffe mit Problemrelevanz

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation › Abstraktion

  Begriffe mit Problemrelevanz — Gegeben & Gesucht

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation › Abstraktion › Entwurf

  Spezifikation: Begriffe mit Problemrelevanz
  Abstraktion: Gegeben & Gesucht
  Entwurf: Algorithmus

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation › Abstraktion › Entwurf › Verifikation

  Spezifikation: Begriffe mit Problemrelevanz
  Abstraktion: Gegeben & Gesucht
  Entwurf: Algorithmus
  Verifikation: Termination & partielle Korrektheit

Kurzwiederholung

- Algorithmenkonstruktion

Spezifikation › Abstraktion › Entwurf › Verifikation › Aufwandsanalyse

Spezifikation: Begriffe mit Problemrelevanz
Abstraktion: Gegeben & Gesucht
Entwurf: Algorithmus
Verifikation: Termination & partielle Korrektheit
Aufwandsanalyse: Laufzeitverhalten

Kurzwiederholung

- Algorithmenkonstruktion

Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

- Kerneigenschaften eines Algorithmus:

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

- Kerneigenschaften eines Algorithmus:
  - Von einem Prozessor (Mensch, …) in endlich vielen Schritten Ausführ- und Reproduzierbar

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

- Kerneigenschaften eines Algorithmus:
  - Von einem Prozessor (Mensch, …) in endlich vielen Schritten Ausführ- und Reproduzierbar
  - Eine endliche Beschreibung in Elementaroperationen

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

- Kerneigenschaften eines Algorithmus:
  - Von einem Prozessor (Mensch, …) in endlich vielen Schritten Ausführ- und Reproduzierbar
  - Eine endliche Beschreibung in Elementaroperationen

- Abstrakte Variablenoperationen:

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

- Kerneigenschaften eines Algorithmus:
  - Von einem Prozessor (Mensch, …) in endlich vielen Schritten Ausführ- und Reproduzierbar
  - Eine endliche Beschreibung in Elementaroperationen

- Abstrakte Variablenoperationen:

```
long x;
x = 21;
x = 42;
```

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

- Kerneigenschaften eines Algorithmus:
  - Von einem Prozessor (Mensch, …) in endlich vielen Schritten Ausführ- und Reproduzierbar
  - Eine endliche Beschreibung in Elementaroperationen

- Abstrakte Variablenoperationen:

```
long x;     Deklaration — Reservieren von Namen mit Typ long
x = 21;
x = 42;
```

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

- Kerneigenschaften eines Algorithmus:
  - Von einem Prozessor (Mensch, …) in endlich vielen Schritten Ausführ- und Reproduzierbar
  - Eine endliche Beschreibung in Elementaroperationen

- Abstrakte Variablenoperationen:

```
long x;     Deklaration — Reservieren von Namen mit Typ long
x = 21;     Initialisierung — Erste Wertzuweisung, macht die Variable verwendbar
x = 42;
```

Kurzwiederholung

- Algorithmenkonstruktion

  Spezifikation (Begriffe mit Problemrelevanz) › Abstraktion (Gegeben & Gesucht) › **Entwurf** (Algorithmus) › Verifikation (Termination & partielle Korrektheit) › Aufwandsanalyse (Laufzeitverhalten)

- Kerneigenschaften eines Algorithmus:
  - Von einem Prozessor (Mensch, …) in endlich vielen Schritten Ausführ- und Reproduzierbar
  - Eine endliche Beschreibung in Elementaroperationen

- Abstrakte Variablenoperationen:

```
long x;     Deklaration — Reservieren von Namen mit Typ long
x = 21;     Initialisierung — Erste Wertzuweisung, macht die Variable verwendbar
x = 42;     Wertzuweisung — Überschreiben/Ändern des vorherigen Wertes
```

„Mach's farbig“
— Isabell Bannweg

Exkurs: Kommandozeilenparameter

M-Array me und andere Zukunftsträume

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit `[ ]`:

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit `[]`:

```
int[] arr = {42, -7, 13};
```

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit `[]`:

```
int[] arr = {42, -7, 13};
```

Ein Array an Integern

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit `[]`:

```
int[] arr = {42, -7, 13};
```

Ein Array an Integern

```
System.out.println(arr[1]);
```

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit `[]`:

```
int[] arr = {42, -7, 13};
          Ein Array an Integern
System.out.println(arr[1]);
```

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit `[]`:

```
int[] arr = {42, -7, 13};
```

Ein Array an Integern

```
System.out.println(arr[1]); // → -7
```

Beginnt bei 0

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit [ ]:

```
int[] arr = {42, -7, 13};
```
Ein Array an Integern

```
System.out.println(arr[1]); // → -7
```
Beginnt bei 0

```
System.out.println(arr.length);
```

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit [ ]:

```
int[] arr = {42, -7, 13};
```

Ein Array an Integern

```
System.out.println(arr[1]); // → -7
```

Beginnt bei 0

```
System.out.println(arr.length);
```

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit [ ]:

```
int[] arr = {42, -7, 13};
```

Ein Array an Integern

```
System.out.println(arr[1]); // → -7
```

Beginnt bei 0

```
System.out.println(arr.length); // → 3
```

Liefert die Array-Größe

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit [ ]:

```
int[] arr = {42, -7, 13};
```

Ein Array an Integern

```
System.out.println(arr[1]); // → -7
```

Beginnt bei 0

```
System.out.println(arr.length); // → 3
```

Liefert die Array-Größe

```
System.out.println(arr[arr.length - 1]);
```

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit [ ]:

```
int[] arr = {42, -7, 13};
```
Ein Array an Integern

```
System.out.println(arr[1]); // → -7
```
Beginnt bei 0

```
System.out.println(arr.length); // → 3
```
Liefert die Array-Größe

```
System.out.println(arr[arr.length - 1]);
```

M-Array me und andere Zukunftsträume

- Arrays werden später genauer behandelt
- Arrays sind Listen fester Größe die Elemente des gleichen Datentyps beinhalten
- Wir notieren sie mit [ ]:

```
int[] arr = {42, -7, 13};
```

Ein Array an Integern

```
System.out.println(arr[1]); // → -7
```

Beginnt bei 0

```
System.out.println(arr.length); // → 3
```

Liefert die Array-Größe

```
System.out.println(arr[arr.length - 1]); // → 13
```

Die Sache mit dem `String[] args`

Die Sache mit dem `String[] args`

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

    }
}
```

Die Sache mit dem `String[] args`

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

    }
}
```

Die Sache mit dem `String[] args`

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

    }
}
```

- Hier haben wir ein Array von Strings

Die Sache mit dem `String[] args`

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

    }
}
```

- Hier haben wir ein Array von Strings
- Dieses wird von der Java Virtual Machine gefüllt

Die Sache mit dem `String[] args`

```
public class Example {
    public static void main(String[] args) {
        System.out.println(args.length);
    }
}
```

- Hier haben wir ein Array von Strings
- Dieses wird von der Java Virtual Machine gefüllt

Die Sache mit dem `String[] args`

```
public class Example {
    public static void main(String[] args) {
        System.out.println(args.length);
    }
}
```

- Hier haben wir ein Array von Strings
- Dieses wird von der Java Virtual Machine gefüllt
- Beim Programmstart mit `java Example`, können wir Argumente übergeben

Ein wunderbares Beispiel

Ein wunderbares Beispiel

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




    }
}
```

Ein wunderbares Beispiel

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

        for(int i = 0; i < args.length; i++) {

        }

    }
}
```

Ein wunderbares Beispiel

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

      for(int i = 0; i < args.length; i++) {
         System.out.println(args[i]);
      }

   }
}
```

Ein wunderbares Beispiel

java Example Hermeline mögen Mäuse

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

      for(int i = 0; i < args.length; i++) {
         System.out.println(args[i]);
      }

   }
}
```

Ein wunderbares Beispiel

```
java Example Hermeline mögen Mäuse
```

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

      for(int i = 0; i < args.length; i++) {
         System.out.println(args[i]);
      }

   }
}
```

Ein wunderbares Beispiel

```
java Example Hermeline mögen Mäuse
```

```
public class Example {
    public static void main(String[] args) {
                                        args = { "Hermeline", "mögen", "Mäuse"}
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }

    }
}
```

Ein wunderbares Beispiel

```
java Example Hermeline mögen Mäuse
```

```
public class Example {
    public static void main(String[] args) {
                                    3          args = { "Hermeline", "mögen", "Mäuse"}
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }

    }
}
```

Ein wunderbares Beispiel

```
java Example Hermeline mögen Mäuse
```

```
public class Example {
    public static void main(String[] args) {
                                    3          args = { "Hermeline", "mögen", "Mäuse"}
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }

    }
}
```

Ein wunderbares Beispiel

```
java Example Hermeline mögen Mäuse
```

```
public class Example {
    public static void main(String[] args) {
                                    3
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }

    }
}
```

args = { "Hermeline", "mögen", "Mäuse" }

Ein wunderbares Beispiel

```
java Example Hermeline mögen Mäuse
```

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

        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }

    }
}
```

3

args = { "Hermeline", "mögen", "Mäuse"}

i = 0

Hermeline

Ein wunderbares Beispiel

```
java Example Hermeline mögen Mäuse
```

```
public class Example {
    public static void main(String[] args) {
                                    3
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```

args = { "Hermeline", "mögen", "Mäuse" }

i = 0 → Hermeline

i = 1 → mögen

Ein wunderbares Beispiel

```
java Example Hermeline mögen Mäuse
```

```
public class Example {
    public static void main(String[] args) {
        for(int i = 0; i < args.length; i++) {
            System.out.println(args[i]);
        }
    }
}
```

3

args = { "Hermeline", "mögen", "Mäuse" }

i = 0 → Hermeline

i = 1 → mögen

i = 2 → Mäuse


Zahlen als Parameter
werden leider auch
zu String.


Zahlen als Parameter werden leider auch zu String.


Wir machen einen eigenen Zahlenkonverter.


Zahlen als Parameter werden leider auch zu String.
Wir machen einen eigenen Zahlenkonverter.

Leider nein.

Leider nein. Wir greifen „nur“ vor…

Leider nein. Wir greifen „nur“ vor…

`int`? Wie primitiv!

`int`? Wie primitiv!

- Für jeden primitiven Datentyp existiert eine „Wrapper-Klasse“

`int`? Wie primitiv!

- Für jeden primitiven Datentyp existiert eine „Wrapper-Klasse“

KLASSE??

`int`? Wie primitiv!

- Für jeden primitiven Datentyp existiert eine „Wrapper-Klasse“ KLASSE??
- Wie `System`, welches uns `out.println` liefert, gibt es `Integer` für `int`

`int`? Wie primitiv!

- Für jeden primitiven Datentyp existiert eine „Wrapper-Klasse“ KLASSE??
- Wie `System`, welches uns `out.println` liefert, gibt es `Integer` für `int`
  - `Integer.parseInt(...)`, `Double.parseDouble(...)`, … konvertieren `"42"` zu `42`

`int`? Wie primitiv!

- Für jeden primitiven Datentyp existiert eine „Wrapper-Klasse“ KLASSE??
- Wie `System`, welches uns `out.println` liefert, gibt es `Integer` für `int`
  - `Integer.parseInt(...)`, `Double.parseDouble(...)`, … konvertieren `"42"` zu `42`
  - Genauer liefert `Integer.parseInt("42")` die Zahl `42` zurück

`int`? Wie primitiv!

- Für jeden primitiven Datentyp existiert eine „Wrapper-Klasse“ KLASSE??
- Wie `System`, welches uns `out.println` liefert, gibt es `Integer` für `int`
  - `Integer.parseInt(...)`, `Double.parseDouble(...)`, … konvertieren `"42"` zu `42`
  - Genauer liefert `Integer.parseInt("42")` die Zahl `42` zurück
  - Wir gehen hier zunächst nur von gültigen Eingaben (Zahlen) aus

`int`? Wie primitiv!

- Für jeden primitiven Datentyp existiert eine „Wrapper-Klasse“ KLASSE??
- Wie `System`, welches uns `out.println` liefert, gibt es `Integer` für `int`
  - `Integer.parseInt(...)`, `Double.parseDouble(...)`, … konvertieren `"42"` zu `42`
  - Genauer liefert `Integer.parseInt("42")` die Zahl `42` zurück
  - Wir gehen hier zunächst nur von gültigen Eingaben (Zahlen) aus
- Nun sind wir gewappnet für die Präsenzaufgabe!

Es gibt keine if-Schleifen, sondern nur if-Abfragen!
— if-schleife.de

Präsenzaufgabe

Präsenzaufgabe

1

Wenn… Ja wenn nur die Schleife nicht wär…

Präsenzaufgabe

1 **Wenn… Ja wenn nur die Schleife nicht wär…**

Legen Sie eine Java Datei namens `ErsteSchleife.java` an (oder bearbeiten Sie die Aufgabe auf einem Blatt Papier).

Präsenzaufgabe

1 **Wenn… Ja wenn nur die Schleife nicht wär…**

Legen Sie eine Java Datei namens `ErsteSchleife.java` an (oder bearbeiten Sie die Aufgabe auf einem Blatt Papier). Lesen Sie in der main Methode eine `int` Variable namens n über die Kommandozeilenparameter ein und implementieren Sie eine for Schleife, die jede dritte Zahl von 1 bis einschließlich n ausgibt.

Präsenzaufgabe

1 **Wenn… Ja wenn nur die Schleife nicht wär…**

Legen Sie eine Java Datei namens `ErsteSchleife.java` an (oder bearbeiten Sie die Aufgabe auf einem Blatt Papier). Lesen Sie in der main Methode eine `int` Variable namens n über die Kommandozeilenparameter ein und implementieren Sie eine for Schleife, die jede dritte Zahl von 1 bis einschließlich n ausgibt. Beispiel:

```
n = 13
```

Präsenzaufgabe

1 **Wenn… Ja wenn nur die Schleife nicht wär…**

Legen Sie eine Java Datei namens `ErsteSchleife.java` an (oder bearbeiten Sie die Aufgabe auf einem Blatt Papier). Lesen Sie in der main Methode eine `int` Variable namens n über die Kommandozeilenparameter ein und implementieren Sie eine for Schleife, die jede dritte Zahl von 1 bis einschließlich n ausgibt. Beispiel:

```
n = 13 // → 1 4 7 10 13
```

Ein Schleifenzähler

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {




}
```

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

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




    }
}
```

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);



   }
}
```

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for(int i = 1; i <= n; i = i + 3) {

      }
   }
}
```

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for(int i = 1; i <= n; i = i + 3) {
         System.out.println(i);
      }
   }
}
```

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for(int i = 1; i <= n; i = i + 3) {
         System.out.println(i);
      }
   }
}
```

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for(int i = 1; i <= n; i = i + 3) {
         System.out.println(i);
      }
   }
}
```

Das Beispiel der Aufgabe hatte keine neuen Zeilen!

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for(int i = 1; i <= n; i = i + 3) {
         System.out.println(i);
      }
   }
}
```

Das Beispiel der Aufgabe hatte keine neuen Zeilen!
Dafür gibt es „print“ anstelle von „print **lin**e“.

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for(int i = 1; i <= n; i = i + 3) {
         System.out.println(i);
      }
   }
}
```

Was, wenn es keine Argumente gibt? Hier hilft eine if-Abfrage.

Das Beispiel der Aufgabe hatte keine neuen Zeilen! Dafür gibt es „print“ anstelle von „print **l**in**e**“.

Ein Schleifenzähler

- Mit dem Vorwissen ist es an sich nur Einsetzen:

```
public class ErsteSchleife {
   public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);
      for(int i = 1; i <= n; i = i + 3) {
         System.out.println(i);
      }
   }
}
```

Was, wenn es keine Argumente gibt? Hier hilft eine if-Abfrage.

Das Beispiel der Aufgabe hatte keine neuen Zeilen! Dafür gibt es „print“ anstelle von „print **l**in**e**“.

ErsteSchleife.java

In the Mean-While

In the Mean-While

- Wir können dies aber auch mit einer `while`-Schleife machen (links)

In the Mean-While

› Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```
int i = 1;
```

In the Mean-While

› Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```
int i = 1;
while(i <= n) {


}
```

In the Mean-While

› Wir können dies aber auch mit einer **while**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");

}
```

In the Mean-While

› Wir können dies aber auch mit einer **while**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

In the Mean-While

- Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

- Oder mit einer **`do-while`**-Schleife (rechts)

In the Mean-While

- Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

```
int i = 1;
```

- Oder mit einer **`do-while`**-Schleife (rechts)

In the Mean-While

- Wir können dies aber auch mit einer **while**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

```
int i = 1;
if(n > 0) {




}
```

- Oder mit einer **do-while**-Schleife (rechts)

In the Mean-While

- Wir können dies aber auch mit einer **while**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

```
int i = 1;
if(n > 0) {
   do {


   } while (i <= n);
}
```

- Oder mit einer **do-while**-Schleife (rechts)

In the Mean-While

- Wir können dies aber auch mit einer **while**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

```
int i = 1;
if(n > 0) {
   do {
      System.out.print(i + " ");

   } while (i <= n);
}
```

- Oder mit einer **do-while**-Schleife (rechts)

In the Mean-While

- Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

```
int i = 1;
if(n > 0) {
   do {
      System.out.print(i + " ");
      i += 3;
   } while (i <= n);
}
```

- Oder mit einer **`do-while`**-Schleife (rechts)

In the Mean-While

- Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```java
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

```java
int i = 1;
if(n > 0) {
   do {
      System.out.print(i + " ");
      i += 3;
   } while (i <= n); Das ; nicht vergessen!
}
```

- Oder mit einer **`do-while`**-Schleife (rechts)

In the Mean-While

- Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```
int i = 1;
while(i <= n) {
   System.out.print(i + " ");
   i += 3;
}
```

```
int i = 1;
if(n > 0) {                Soll die Schleife überhaupt
   do {                    einmal laufen?
      System.out.print(i + " ");
      i += 3;
   } while (i <= n);       Das ; nicht vergessen!
}
```

- Oder mit einer **`do-while`**-Schleife (rechts)

In the Mean-While

- Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```java
int i = 1;
while(i <= n) {
    System.out.print(i + " ");
    i += 3;
}
```

```java
int i = 1;
if(n > 0) { // Soll die Schleife überhaupt
    do {    //      einmal laufen?
        System.out.print(i + " ");
        i += 3;
    } while (i <= n); // Das ; nicht vergessen!
}
```

- Oder mit einer **`do-while`**-Schleife (rechts)
- „**`java`** `ErsteSchleife` *`12`*“ liefert die Ausgabe „*`1 4 7 10`*“

In the Mean-While

- Wir können dies aber auch mit einer **`while`**-Schleife machen (links)

```java
int i = 1;
while(i <= n) {
    System.out.print(i + " ");
    i += 3;
}
```

```java
int i = 1;
if(n > 0) { // Soll die Schleife überhaupt
    do {    //      einmal laufen?
        System.out.print(i + " ");
        i += 3;
    } while (i <= n); // Das ; nicht vergessen!
}
```

- Oder mit einer **`do-while`**-Schleife (rechts)
- „**`java`** `ErsteSchleife` *`12`*“ liefert die Ausgabe „*`1 4 7 10`*“

ErsteSchleifeWhile.java, ErsteSchleifeDoWhile.java

Essen für die Reste!

Essen für die Reste!

- Anstelle von $i \leftarrow i + 3$ können wir beispielsweise auch Modulo-Rechnen:

Essen für die Reste!

- Anstelle von $i \leftarrow i + 3$ können wir beispielsweise auch Modulo-Rechnen:

```
for (int i = 0; i <= n; i++) {


}
```

Essen für die Reste!

- Anstelle von $i \leftarrow i + 3$ können wir beispielsweise auch Modulo-Rechnen:

```
for (int i = 0; i <= n; i++) {
   if(i % 3 == 1)

}
```

Essen für die Reste!

- Anstelle von $i \leftarrow i + 3$ können wir beispielsweise auch Modulo-Rechnen:

```
for (int i = 0; i <= n; i++) {
   if(i % 3 == 1)
      System.out.print(i + " ");
}
```

Essen für die Reste!

- Anstelle von $i \leftarrow i + 3$ können wir beispielsweise auch Modulo-Rechnen:

```
for (int i = 0; i <= n; i++) {
   if(i % 3 == 1)
      System.out.print(i + " ");
}
```

- Es gibt eigentlich immer etliche Möglichkeiten…

Essen für die Reste!

- Anstelle von $i \leftarrow i + 3$ können wir beispielsweise auch Modulo-Rechnen:

```java
for (int i = 0; i <= n; i++) {
    if(i % 3 == 1)
        System.out.print(i + " ");
}
```

- Es gibt eigentlich immer etliche Möglichkeiten…

ErsteSchleifeModulo.java

Übungsblatt 2

Aufgabe 1: Algorithmen und Unteralgorithmen

Aufgabe 1: Algorithmen und Unteralgorithmen

Oft ist es zur besseren Strukturierung notwendig komplexere Algorithmen aufzuteilen. Dies ist beispielsweise der Fall, wenn Code-Elemente häufig an verschiedenen Stellen eingesetzt werden oder wenn es der Übersichtlichkeit dienen soll. Im Folgenden sollen Sie einen einfachen Sortieralgorithmus in zwei Teilalgorithmen umsetzen. Zusätzlich sollen Sie lernen die Eingaben (sog. Parameter oder Argumente) des Aufrufs auf Gültigkeit zu überprüfen. Für zukünftige Aufgaben sollen Sie dies selbstständig erkennen und umsetzen.

Aufgabe 1: Algorithmen und Unteralgorithmen

Oft ist es zur besseren Strukturierung notwendig komplexere Algorithmen aufzuteilen. Dies ist beispielsweise der Fall, wenn Code-Elemente häufig an verschiedenen Stellen eingesetzt werden oder wenn es der Übersichtlichkeit dienen soll. Im Folgenden sollen Sie einen einfachen Sortieralgorithmus in zwei Teilalgorithmen umsetzen. Zusätzlich sollen Sie lernen die Eingaben (sog. Parameter oder Argumente) des Aufrufs auf Gültigkeit zu überprüfen. Für zukünftige Aufgaben sollen Sie dies selbstständig erkennen und umsetzen.

Gehen Sie zur Sortierung nach folgendem Schema vor.

Aufgabe 1: Algorithmen und Unteralgorithmen

Oft ist es zur besseren Strukturierung notwendig komplexere Algorithmen aufzuteilen. Dies ist beispielsweise der Fall, wenn Code-Elemente häufig an verschiedenen Stellen eingesetzt werden oder wenn es der Übersichtlichkeit dienen soll. Im Folgenden sollen Sie einen einfachen Sortieralgorithmus in zwei Teilalgorithmen umsetzen. Zusätzlich sollen Sie lernen die Eingaben (sog. Parameter oder Argumente) des Aufrufs auf Gültigkeit zu überprüfen. Für zukünftige Aufgaben sollen Sie dies selbstständig erkennen und umsetzen.

Gehen Sie zur Sortierung nach folgendem Schema vor.

Seien $l_1, \ldots, l_N$ *die Werte einer Liste* L *der Länge* N. *Für jedes* $i \in \{1, \ldots, N-1\}$ *betrachte der Reihe nach jedes* l_j *mit* $j \in \{i+1, \ldots, N\}$ *und vergleiche* l_i *mit* l_j. *Falls* $l_i > l_j$ *vertausche* l_i *und* l_j *und fahre mit dem nächsten* j *fort.*

Aufgabe 1: Algorithmen und Unteralgorithmen

Oft ist es zur besseren Strukturierung notwendig komplexere Algorithmen aufzuteilen. Dies ist beispielsweise der Fall, wenn Code-Elemente häufig an verschiedenen Stellen eingesetzt werden oder wenn es der Übersichtlichkeit dienen soll. Im Folgenden sollen Sie einen einfachen Sortieralgorithmus in zwei Teilalgorithmen umsetzen. Zusätzlich sollen Sie lernen die Eingaben (sog. Parameter oder Argumente) des Aufrufs auf Gültigkeit zu überprüfen. Für zukünftige Aufgaben sollen Sie dies selbstständig erkennen und umsetzen.

Indizes		*Liste*				*Vertauschen?*
i	*j*	l_1	l_2	l_3	l_4	
1	2	▸ **5**	▸ **8**	4	0	*Nein, denn* $5 \not> 8$
1	3	▸ **5**	8	▸ **5**	0	*Ja, denn* $5 > 4$
1	4	▸ **4**	8	5	▸ **0**	*Ja, denn* $4 > 0$
Runde vorbei, alle $j \in \{i+1, \ldots, N\}$ *betrachtet!*						
2	3	0	▸ **8**	▸ **5**	5	*Ja, denn* $8 > 5$
2	4	0	▸ **5**	8	▸ **4**	*Ja, denn* $5 > 4$
Runde vorbei, alle $j \in \{i+1, \ldots, N\}$ *betrachtet!*						
3	4	0	4	▸ **8**	▸ **5**	*Ja, denn* $8 > 5$
Fertig, alle $i \in \{1, \ldots, N-1\}$ *betrachtet!*						
	→	0	4	5	8	

Gehen Sie zur Sortierung nach folgendem Schema vor.

Seien $l_1, \ldots, l_N$ *die Werte einer Liste* L *der Länge* N. *Für jedes* $i \in \{1, \ldots, N-1\}$ *betrachte der Reihe nach jedes* l_j *mit* $j \in \{i+1, \ldots, N\}$ *und vergleiche* l_i *mit* l_j. *Falls* $l_i > l_j$ *vertausche* l_i *und* l_j *und fahre mit dem nächsten* j *fort.*

Aufgabe 1: Algorithmen und Unteralgorithmen

Oft ist es zur besseren Strukturierung notwendig komplexere Algorithmen aufzuteilen. Dies ist beispielsweise der Fall, wenn Code-Elemente häufig an verschiedenen Stellen eingesetzt werden oder wenn es der Übersichtlichkeit dienen soll. Im Folgenden sollen Sie einen einfachen Sortieralgorithmus in zwei Teilalgorithmen umsetzen. Zusätzlich sollen Sie lernen die Eingaben (sog. Parameter oder Argumente) des Aufrufs auf Gültigkeit zu überprüfen. Für zukünftige Aufgaben sollen Sie dies selbstständig erkennen und umsetzen.

Indizes		*Liste*				*Vertauschen?*
i	*j*	l_1	l_2	l_3	l_4	
1	2	▸ ***5***	▸ ***8***	4	0	*Nein, denn* $5 \not> 8$
1	3	▸ ***5***	8	▸ ***5***	0	*Ja, denn* $5 > 4$
1	4	▸ ***4***	8	5	▸ ***0***	*Ja, denn* $4 > 0$
Runde vorbei, alle $j \in \{i+1, \ldots, N\}$ *betrachtet!*						
2	3	0	▸ ***8***	▸ ***5***	5	*Ja, denn* $8 > 5$
2	4	0	▸ ***5***	8	▸ ***4***	*Ja, denn* $5 > 4$
Runde vorbei, alle $j \in \{i+1, \ldots, N\}$ *betrachtet!*						
3	4	0	4	▸ ***8***	▸ ***5***	*Ja, denn* $8 > 5$
Fertig, alle $i \in \{1, \ldots, N-1\}$ *betrachtet!*						
→		0	4	5	8	

Gehen Sie zur Sortierung nach folgendem Schema vor.

Seien $l_1, \ldots, l_N$ *die Werte einer Liste* L *der Länge* N. *Für jedes* $i \in \{1, \ldots, N-1\}$ *betrachte der Reihe nach jedes* l_j *mit* $j \in \{i+1, \ldots, N\}$ *und vergleiche* l_i *mit* l_j. *Falls* $l_i > l_j$ *vertausche* l_i *und* l_j *und fahre mit dem nächsten* j *fort.*

So sortiert der beschriebene Algorithmus die Liste $(5, 8, 4, 0)$ *aufsteigend zu* $(0, 4, 5, 8)$. *Die beiden* ▸ ***Markierungen*** *je Zeile geben an, welche Elemente in dem jeweiligen Schritt verglichen werden.*

Elemente vertauschen

Elemente vertauschen

a) *Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes* i *und* j *gültig sind, d.h. dass sie größer als* 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Elemente vertauschen

a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i *und* j *gültig sind, d.h. dass sie größer als* 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

Elemente vertauschen

a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i und j *gültig sind, d.h. dass sie größer als* 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

1 **if** N < 2 **or** i < 1 **or** j < 1 **or** i > N **or** j > N **then**

Elemente vertauschen

a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i und j *gültig sind, d.h. dass sie größer als* 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2 |   print „Eingabe ungültig!“ to output device;
3 |   stop program;
```

Elemente vertauschen

a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i und j *gültig sind, d.h. dass sie größer als* 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!“ to output device;
3     stop program;
4 make new t have value l_i;
```

Elemente vertauschen

a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i und j *gültig sind, d.h. dass sie größer als* 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!“ to output device;
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
```

Elemente vertauschen

a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i und j *gültig sind, d.h. dass sie größer als* 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!“ to output device;
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
6 make l_j have value t;
```

Elemente vertauschen

a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i und j *gültig sind, d.h. dass sie* ***größer als*** 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!“ to output device;
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
6 make l_j have value t;
```

Elemente vertauschen

a) Entwerfen Sie einen Algorithmus, der gegeben einer Liste und zwei Indizes die entsprechenden Elemente vertauscht. Stellen Sie sicher, dass die Indizes i *und* j *gültig sind, d.h. dass sie* ***größer als*** 1 *und kleiner als die Länge der Liste sind. Stellen Sie zudem sicher, dass die Länge der Liste mindestens* 2 *beträgt. Der Algorithmus soll für den Fall der Ungültigkeit mit einer Fehlermeldung abbrechen.*

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!“ to output device;
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
6 make l_j have value t;
```

In der Aufgabe stand etwas von „größer als 1“, aber wie wir Listen indizieren ist nicht von uns standardisiert.

Sich den Tag schön-sortieren

Sich den Tag schön-sortieren

b) *Entwerfen Sie einen Algorithmus, der genau nach dem angegebenen Schema eine Liste sortiert. Stellen Sie auch hier sicher, dass die Liste mindestens zwei Elemente hat. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass der Algorithmus* `vertausche(L, i, j)` *zur Verfügung steht.*

Sich den Tag schön-sortieren

b) *Entwerfen Sie einen Algorithmus, der genau nach dem angegebenen Schema eine Liste sortiert. Stellen Sie auch hier sicher, dass die Liste mindestens zwei Elemente hat. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass der Algorithmus* `vertausche(L, i, j)` *zur Verfügung steht.*

Input : List L = $(l_1, \ldots, l_N)$

Sich den Tag schön-sortieren

b) *Entwerfen Sie einen Algorithmus, der genau nach dem angegebenen Schema eine Liste sortiert. Stellen Sie auch hier sicher, dass die Liste mindestens zwei Elemente hat. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass der Algorithmus* `vertausche(L, i, j)` *zur Verfügung steht.*

Input : List L = $(l_1, \ldots, l_N)$

1 **if** N < 2 **then**

Sich den Tag schön-sortieren

b) Entwerfen Sie einen Algorithmus, der genau nach dem angegebenen Schema eine Liste sortiert. Stellen Sie auch hier sicher, dass die Liste mindestens zwei Elemente hat. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass der Algorithmus `vertausche(L, i, j)` zur Verfügung steht.

Input : List L = $(l_1, \ldots, l_N)$

```
1 if N < 2 then
2     print „Eingabe ungültig!“ to output device and stop program;
```

Sich den Tag schön-sortieren

b) *Entwerfen Sie einen Algorithmus, der genau nach dem angegebenen Schema eine Liste sortiert. Stellen Sie auch hier sicher, dass die Liste mindestens zwei Elemente hat. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass der Algorithmus* `vertausche(L, i, j)` *zur Verfügung steht.*

Input : List L = $(l_1, \ldots, l_N)$

```
1 if N < 2 then
2     print „Eingabe ungültig!“ to output device and stop program;
3 Make new i have value 1;
4 while i < N do
  |
  |
  |
  |
  |
9 |   Increment i by 1;
```

Sich den Tag schön-sortieren

b) Entwerfen Sie einen Algorithmus, der genau nach dem angegebenen Schema eine Liste sortiert. Stellen Sie auch hier sicher, dass die Liste mindestens zwei Elemente hat. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass der Algorithmus `vertausche(L, i, j)` zur Verfügung steht.

Input : List L = $(l_1, \ldots, l_N)$

```
1 if N < 2 then
2     print „Eingabe ungültig!“ to output device and stop program;
3 Make new i have value 1;
4 while i < N do
5     Make new j have the value i + 1;
6     while j ≤ N do
8         Increment j by 1;
9     Increment i by 1;
```

Sich den Tag schön-sortieren

b) Entwerfen Sie einen Algorithmus, der genau nach dem angegebenen Schema eine Liste sortiert. Stellen Sie auch hier sicher, dass die Liste mindestens zwei Elemente hat. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass der Algorithmus `vertausche(L, i, j)` zur Verfügung steht.

Input : List L = $(l_1, \ldots, l_N)$

```
1 if N < 2 then
2     print „Eingabe ungültig!“ to output device and stop program;
3 Make new i have value 1;
4 while i < N do
5     Make new j have the value i + 1;
6     while j ≤ N do
7         if l_i > l_j then call vertausche(L, i, j);
8         Increment j by 1;
9     Increment i by 1;
```

Du kriegsts' nicht raus? Better Call Gauß!

Du kriegsts' nicht raus? Better Call Gauß!

c) *Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.*

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!" to output device;
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
6 make l_j have value t;
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!" to output device;
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
6 make l_j have value t;
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

1

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!" to output device;
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
6 make l_j have value t;
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = ($l_1, \ldots, l_N$), Index i, and Index j

```
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2      print „Eingabe ungültig!" to output device;
3      stop program;
4  make new t have value l_i;
5  make l_i have value l_j;
6  make l_j have value t;
```

1 *1*

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
                1          1          1
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!" to output device;
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
6 make l_j have value t;
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2      print „Eingabe ungültig!" to output device;
3      stop program;
4  make new t have value l_i;
5  make l_i have value l_j;
6  make l_j have value t;
```

1 *1* *1* *1*

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = ($l_1, \ldots, l_N$), Index i, and Index j

```
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2      print „Eingabe ungültig!" to output device;
3      stop program;
4  make new t have value l_i;
5  make l_i have value l_j;
6  make l_j have value t;
```

1 *1* *1* *1* *1*

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
          1        1        1        1        1
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then
2     print „Eingabe ungültig!" to output device;    1
3     stop program;
4 make new t have value l_i;
5 make l_i have value l_j;
6 make l_j have value t;
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = ($l_1, \ldots, l_N$), Index i, and Index j

```
        1          1          1          1          1
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2      print „Eingabe ungültig!" to output device;     1
3      stop program;     1
4  make new t have value l_i;
5  make l_i have value l_j;
6  make l_j have value t;
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = ($l_1, \ldots, l_N$), Index i, and Index j

```
        1          1          1          1          1
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2  |   print „Eingabe ungültig!" to output device;     1
3  |   stop program;     1
4  make new t have value l_i;     1
5  make l_i have value l_j;
6  make l_j have value t;
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
        1          1          1          1          1
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2      print „Eingabe ungültig!" to output device;    1
3      stop program;    1
4  make new t have value l_i;    1
5  make l_i have value l_j;    1
6  make l_j have value t;
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = $(l_1, \ldots, l_N)$, Index i, and Index j

```
        1        1        1        1        1
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2      print „Eingabe ungültig!" to output device;    1
3      stop program;    1
4  make new t have value l_i;    1
5  make l_i have value l_j;    1
6  make l_j have value t;    1
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = ($l_1, \ldots, l_N$), Index i, and Index j

```
                1          1          1          1          1
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2  |   print „Eingabe ungültig!" to output device;    1      } 2
3  |   stop program;    1                                    }
4  make new t have value l_i;    1
5  make l_i have value l_j;    1
6  make l_j have value t;    1
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = ($l_1, \ldots, l_N$), Index i, and Index j

```
                1          1          1          1          1
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2  |   print „Eingabe ungültig!" to output device;    1      } 2
3  |   stop program;    1                                    }
4  make new t have value l_i;    1                          }
5  make l_i have value l_j;      1                          } 3
6  make l_j have value t;        1                          }
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = ($l_1, \ldots, l_N$), Index i, and Index j

```
                1          1          1          1          1
1  if N < 2 or i < 1 or j < 1 or i > N or j > N then
2  |   print „Eingabe ungültig!" to output device;   1
3  |   stop program;   1                                          } 2 (Nur schlechtester Pfad)
4  make new t have value l_i;   1
5  make l_i have value l_j;     1                                 } 3
6  make l_j have value t;       1
```

Du kriegsts' nicht raus? Better Call Gauß!

c) Geben Sie eine worst-case Laufzeitabschätzung für Ihren Algorithmus aus Teilaufgabe b) an. Falls Sie Teilaufgabe a) nicht bearbeitet haben, können Sie annehmen, dass das Vertauschen (inkl. aller Überprüfungen höchstens) neun elementare Operationen benötigt. Begründen Sie Ihre Antwort.

„Wir" haben die a) bearbeitet. Also…Lets begin!

Input : List L = ($l_1, \ldots, l_N$), Index i, and Index j

```
1 if N < 2 or i < 1 or j < 1 or i > N or j > N then        1 1 1 1 1
2 |   print „Eingabe ungültig!" to output device;          1
3 |   stop program;                                        1
4 make new t have value l_i;                               1
5 make l_i have value l_j;                                 1
6 make l_j have value t;                                   1
```

2 (Nur schlechtester Pfad)

3

$\rightarrow \quad 1E + 1E + 1E + 1E + 1E + 3E = 8E$

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then
2      print „Eingabe ungültig!“ to output device and stop program;
3  Make new i have value 1;
4  while i < N do
5      Make new j have the value i + 1;
6      while j ≤ N do
7          if l_i > l_j then
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then
2      print „Eingabe ungültig!“ to output device and stop program;
3  Make new i have value 1;
4  while i < N do
5      Make new j have the value i + 1;
6      while j ≤ N do
7          if l_i > l_j then
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Input : List L = $(l_1, \ldots, l_N)$

1

```
1  if N < 2 then
2      print „Eingabe ungültig!“ to output device and stop program;
3  Make new i have value 1;
4  while i < N do
5      Make new j have the value i + 1;
6      while j ≤ N do
7          if l_i > l_j then
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                              (1)
2      print „Eingabe ungültig!“ to output device and stop program;   (2)
3  Make new i have value 1;
4  while i < N do
5      Make new j have the value i + 1;
6      while j ≤ N do
7          if l_i > l_j then
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                              1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                                   1
4  while i < N do
5      Make new j have the value i + 1;
6      while j ≤ N do
7          if l_i > l_j then
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                          (1)
2      print „Eingabe ungültig!“ to output device and stop program;   (2)
3  Make new i have value 1;                               (1)
4  while i < N do                                         (1)
5      Make new j have the value i + 1;
6      while j ≤ N do
7          if l_i > l_j then
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                          (1)
2      print „Eingabe ungültig!“ to output device and stop program;   (2)
3  Make new i have value 1;                               (1)
4  while i < N do                                         (1)
5      Make new j have the value i + 1;                   (2)
6      while j ≤ N do
7          if l_i > l_j then
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                          (1)
2      print „Eingabe ungültig!“ to output device and stop program;   (2)
3  Make new i have value 1;                               (1)
4  while i < N do                                         (1)
5      Make new j have the value i + 1;                   (2)
6      while j ≤ N do                                     (1)
7          if l_i > l_j then
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   (1)
2      print „Eingabe ungültig!“ to output device and stop program;   (2)
3  Make new i have value 1;                        (1)
4  while i < N do                                  (1)
5      Make new j have the value i + 1;            (2)
6      while j ≤ N do                              (1)
7          if l_i > l_j then                       (1)
8              call vertausche(L, i, j);
9          Increment j by 1;
10     Increment i by 1;
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   (1)
2      print „Eingabe ungültig!“ to output device and stop program;   (2)
3  Make new i have value 1;                        (1)
4  while i < N do                                  (1)
5      Make new j have the value i + 1;            (2)
6      while j ≤ N do                              (1)
7          if l_i > l_j then                       (1)
8              call vertausche(L, i, j);           (8)
9          Increment j by 1;
10     Increment i by 1;
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   (1)
2      print „Eingabe ungültig!“ to output device and stop program;   (2)
3  Make new i have value 1;                        (1)
4  while i < N do                                  (1)
5      Make new j have the value i + 1;            (2)
6      while j ≤ N do                              (1)
7          if l_i > l_j then                       (1)
8              call vertausche(L, i, j);           (8)
9          Increment j by 1;                       (1)
10     Increment i by 1;
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              1
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  1        N – 1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              1
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  1        N – 1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              1        ?
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  1        N – 1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              1        ?
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Angenommen N *sei* 5*, dann gilt für die Anzahl innerer Schleifendurchläufe:*

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  1        N − 1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              1        ?
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Angenommen N *sei* 5*, dann gilt für die Anzahl innerer Schleifendurchläufe:*

i	*#*
1	*4*
2	*3*
3	*2*
4	*1*

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                              (1)
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                                   1
4  while i < N do                                             (1)  N − 1
5      Make new j have the value i + 1;                       2
6      while j ≤ N do                                         (1)  ?
7          if l_i > l_j then                                  (1)
8              call vertausche(L, i, j);                      8
9          Increment j by 1;                                  1
10     Increment i by 1;                                      1
```

Angenommen N *sei* 5*, dann gilt für die Anzahl innerer Schleifendurchläufe:*

i	#
1	4
2	3
3	2
4	1

Wir summieren also von 1 *bis* N − 1*. Nach Gauß:* $\frac{N^2-N}{2}$ *Gesamtdurchläufe.*

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  1        N − 1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              1        ? = (N²−N)/2
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Angenommen N *sei* 5*, dann gilt für die Anzahl innerer Schleifendurchläufe:*

i	#
1	4
2	3
3	2
4	1

Wir summieren also von 1 *bis* N − 1*. Nach Gauß:* $\frac{N^2-N}{2}$ *Gesamtdurchläufe.*

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  1        N − 1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              1        ? = (N²−N)/2
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Angenommen N *sei* 5*, dann gilt für die Anzahl innerer Schleifendurchläufe:*

i	#
1	4
2	3
3	2
4	1

Wir summieren also von 1 *bis* N − 1*. Nach Gauß:* $\frac{N^2-N}{2}$ *Gesamtdurchläufe.*

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                              1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                                   1
4  while i < N do                                             N − 1
5      Make new j have the value i + 1;                       2
6      while j ≤ N do                                         ? = (N²−N)/2
7          if l_i > l_j then                                  1
8              call vertausche(L, i, j);                      8
9          Increment j by 1;                                  1
10     Increment i by 1;                                      1
```

Angenommen N *sei* 5*, dann gilt für die Anzahl innerer Schleifendurchläufe:*

i	#
1	4
2	3
3	2
4	1

Wir summieren also von 1 *bis* N − 1*. Nach Gauß:* $\frac{N^2-N}{2}$ *Gesamtdurchläufe.*

$$\rightarrow 1\,E + 1\,E + (N-1)\cdot(2\,E + 1\,E) + \frac{N^2-N}{2}(1\,E + 8\,E + 1\,E)$$

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  N − 1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              ? = (N²−N)/2
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Angenommen N *sei* 5*, dann gilt für die Anzahl innerer Schleifendurchläufe:*

i	#
1	4
2	3
3	2
4	1

Wir summieren also von 1 *bis* N − 1*. Nach Gauß:* $\frac{N^2-N}{2}$ *Gesamtdurchläufe.*

$$\rightarrow 1\,E + 1\,E + (N-1)\cdot(2\,E + 1\,E) + \frac{N^2-N}{2}(1\,E + 8\,E + 1\,E)$$

$$= -1\,E + 3\cdot N\,E + 5\cdot N^2\,E - 5\cdot N\,E$$

Du kriegsts' nicht raus? Better Call Gauß! The Sequel!

Input : List L = $(l_1, \ldots, l_N)$

```
1  if N < 2 then                                   1
2      print „Eingabe ungültig!“ to output device and stop program;   2
3  Make new i have value 1;                        1
4  while i < N do                                  N − 1
5      Make new j have the value i + 1;            2
6      while j ≤ N do                              ? = (N²−N)/2
7          if l_i > l_j then                       1
8              call vertausche(L, i, j);           8
9          Increment j by 1;                       1
10     Increment i by 1;                           1
```

Angenommen N *sei* 5*, dann gilt für die Anzahl innerer Schleifendurchläufe:*

i	#
1	4
2	3
3	2
4	1

Wir summieren also von 1 *bis* N − 1*. Nach Gauß:* $\frac{N^2-N}{2}$ *Gesamtdurchläufe.*

$$\begin{aligned}
&\to 1\,E + 1\,E + (N-1)\cdot(2\,E + 1\,E) + \frac{N^2-N}{2}(1\,E + 8\,E + 1\,E) \\
&= -1\,E + 3\cdot N\,E + 5\cdot N^2\,E - 5\cdot N\,E \\
&= 5\cdot N^2\,E - 2\cdot N\,E - 1\,E \quad \in \mathcal{O}(N^2)
\end{aligned}$$

Unterprogramm -dadamm

Unterprogramm -dadamm

- `System.out.print`, `Integer.parseInt`, … sind keine Elementaroperation

Unterprogramm -dadamm

- `System.out.print`, `Integer.parseInt`, ... sind keine Elementaroperation
- Es sind implementierte Algorithmen, die eine Aufgabe erledigen

Unterprogramm -dadamm

- `System.out.print`, `Integer.parseInt`, … sind keine Elementaroperation
- Es sind implementierte Algorithmen, die eine Aufgabe erledigen
- Wenn wir bestehende Algorithmen wiederverwenden, heißen sie „Unteralgorithmen“

Unterprogramm -dadamm

- `System.out.print`, `Integer.parseInt`, … sind keine Elementaroperation
- Es sind implementierte Algorithmen, die eine Aufgabe erledigen
- Wenn wir bestehende Algorithmen wiederverwenden, heißen sie „Unteralgorithmen“
  - Sie erlauben einen ersten Abstraktionsmechanismus um Teilprobleme auszulagern

Unterprogramm -dadamm

- `System.out.print`, `Integer.parseInt`, … sind keine Elementaroperation
- Es sind implementierte Algorithmen, die eine Aufgabe erledigen
- Wenn wir bestehende Algorithmen wiederverwenden, heißen sie „Unteralgorithmen“
  - Sie erlauben einen ersten Abstraktionsmechanismus um Teilprobleme auszulagern
  - Ziel ist dann die Komposition aus immer abstrakteren Bausteinen

Unterprogramm -dadamm

- `System.out.print`, `Integer.parseInt`, … sind keine Elementaroperation
- Es sind implementierte Algorithmen, die eine Aufgabe erledigen
- Wenn wir bestehende Algorithmen wiederverwenden, heißen sie „Unteralgorithmen“
  - Sie erlauben einen ersten Abstraktionsmechanismus um Teilprobleme auszulagern
  - Ziel ist dann die Komposition aus immer abstrakteren Bausteinen
- (Unter-)Algorithmen sind ein wichtiger und elementarer Teil der Programmierung

Unterprogramm -dadamm

- `System.out.print`, `Integer.parseInt`, … sind keine Elementaroperation
- Es sind implementierte Algorithmen, die eine Aufgabe erledigen
- Wenn wir bestehende Algorithmen wiederverwenden, heißen sie „Unteralgorithmen“
  - Sie erlauben einen ersten Abstraktionsmechanismus um Teilprobleme auszulagern
  - Ziel ist dann die Komposition aus immer abstrakteren Bausteinen
- (Unter-)Algorithmen sind ein wichtiger und elementarer Teil der Programmierung
- Die Umsetzung in Java werden wir später betrachten.

Aufgabe 2: Datentypen

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? Begründen Sie Ihre Antwort!

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? ***Begründen Sie Ihre Antwort!***

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? ***Begründen Sie Ihre Antwort!***

a) Die Email-Adresse einer Person

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? ***Begründen Sie Ihre Antwort!***

a) Die Email-Adresse einer Person

Email-Adressen sind lange Zeichenketten welche auch Sonderzeichen enthalten. Daher ist `String` naheliegend.

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? ***Begründen Sie Ihre Antwort!***

a) *Die Email-Adresse einer Person*

Email-Adressen sind lange Zeichenketten welche auch Sonderzeichen enthalten. Daher ist `String` naheliegend.

b) *Den Akkustand eines Smartphones*

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? ***Begründen Sie Ihre Antwort!***

a) *Die Email-Adresse einer Person*

Email-Adressen sind lange Zeichenketten welche auch Sonderzeichen enthalten. Daher ist `String` naheliegend.

b) *Den Akkustand eines Smartphones*

Ein Akkustand wird für gewöhnlich als Ganzzahl angegeben (→ `byte`, `short`, `int` und `long`). Der Akkustand reicht dabei meist von 0 % bis 100 % → `byte`.

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? ***Begründen Sie Ihre Antwort!***

a) *Die Email-Adresse einer Person*

Email-Adressen sind lange Zeichenketten welche auch Sonderzeichen enthalten. Daher ist `String` naheliegend.

b) *Den Akkustand eines Smartphones*

Ein Akkustand wird für gewöhnlich als Ganzzahl angegeben (→ `byte`, `short`, `int` und `long`). Der Akkustand reicht dabei meist von 0 % bis 100 % → `byte`.

c) *Den Notendurchschnitt einer Prüfung*

Aufgabe 2: Datentypen

Welchen Datentyp würden Sie wählen um folgende Daten zu speichern? ***Begründen Sie Ihre Antwort!***

a) *Die Email-Adresse einer Person*

Email-Adressen sind lange Zeichenketten welche auch Sonderzeichen enthalten. Daher ist `String` naheliegend.

b) *Den Akkustand eines Smartphones*

Ein Akkustand wird für gewöhnlich als Ganzzahl angegeben (→ `byte`, `short`, `int` und `long`). Der Akkustand reicht dabei meist von 0 % bis 100 % → `byte`.

c) *Den Notendurchschnitt einer Prüfung*

Die Noten einer Prüfung ist meist eine Fließkommazahl (→ `float`, `double`), deren Wertebereich klein mit geringer Auflösung an Nachkommastellen ist (z.B. 1–6 mit einer Nachkommastelle). Deswegen wählen wir hier `float`.

Aufgabe 2: Datentypen

II

Aufgabe 2: Datentypen II

d) Ein Satzzeichen

Aufgabe 2: Datentypen II

d) Ein Satzzeichen

Wir gehen hier von einem Satzzeichen aus, welches durch eine einzige UTF-16 Einheit dargestellt werden kann. In diesem Fall reicht ein `char` (sonst `String` das ist aber schon wieder Overkill).

Aufgabe 2: Datentypen

II

d) Ein Satzzeichen

Wir gehen hier von einem Satzzeichen aus, welches durch eine einzige UTF-16 Einheit dargestellt werden kann. In diesem Fall reicht ein `char` (sonst `String` das ist aber schon wieder Overkill).

› Eine eindeutig richtige Antwort gibt es nicht (Begründungen!).

Aufgabe 2: Datentypen II

d) Ein Satzzeichen

Wir gehen hier von einem Satzzeichen aus, welches durch eine einzige UTF-16 Einheit dargestellt werden kann. In diesem Fall reicht ein `char` (sonst `String` das ist aber schon wieder Overkill).

- Eine eindeutig richtige Antwort gibt es nicht (Begründungen!).
- Begründungen wie „weniger Speicherplatz“ sind problematisch(er)

Aufgabe 2: Datentypen II

d) Ein Satzzeichen

Wir gehen hier von einem Satzzeichen aus, welches durch eine einzige UTF-16 Einheit dargestellt werden kann. In diesem Fall reicht ein `char` (sonst `String` das ist aber schon wieder Overkill).

- Eine eindeutig richtige Antwort gibt es nicht (Begründungen!).
- Begründungen wie „weniger Speicherplatz“ sind problematisch(er)
  - Java kann für ein `byte` auch 32 bit oder mehr (64 bit, …) reservieren

Aufgabe 2: Datentypen II

d) Ein Satzzeichen

Wir gehen hier von einem Satzzeichen aus, welches durch eine einzige UTF-16 Einheit dargestellt werden kann. In diesem Fall reicht ein `char` (sonst `String` das ist aber schon wieder Overkill).

- Eine eindeutig richtige Antwort gibt es nicht (Begründungen!).
- Begründungen wie „weniger Speicherplatz“ sind problematisch(er)
  - Java kann für ein `byte` auch 32 bit oder mehr (64 bit, …) reservieren
  - Analog wird `boolean` oft als `byte` umgesetzt (JVMS17 2.3.4)

Aufgabe 2: Datentypen II

d) Ein Satzzeichen

Wir gehen hier von einem Satzzeichen aus, welches durch eine einzige UTF-16 Einheit dargestellt werden kann. In diesem Fall reicht ein `char` (sonst `String` das ist aber schon wieder Overkill).

- Eine eindeutig richtige Antwort gibt es nicht (Begründungen!).
- Begründungen wie „weniger Speicherplatz“ sind problematisch(er)
  - Java kann für ein `byte` auch 32 bit oder mehr (64 bit, …) reservieren
  - Analog wird `boolean` oft als `byte` umgesetzt (JVMS17 2.3.4)
  - Das hat mit Geschwindigkeit, Parallelisierung, Sicherheit, … zu tun

Aufgabe 2: Datentypen II

d) Ein Satzzeichen

Wir gehen hier von einem Satzzeichen aus, welches durch eine einzige UTF-16 Einheit dargestellt werden kann. In diesem Fall reicht ein `char` (sonst `String` das ist aber schon wieder Overkill).

- Eine eindeutig richtige Antwort gibt es nicht (Begründungen!).
- Begründungen wie „weniger Speicherplatz“ sind problematisch(er)
  - Java kann für ein `byte` auch 32 bit oder mehr (64 bit, …) reservieren
  - Analog wird `boolean` oft als `byte` umgesetzt (JVMS17 2.3.4)
  - Das hat mit Geschwindigkeit, Parallelisierung, Sicherheit, … zu tun
  - Für uns in EidI sind die Details nicht wichtig

Ein wenig mehr über primitive Datentypen

Ein wenig mehr über primitive Datentypen

`boolean`

Ein wenig mehr über primitive Datentypen

`boolean`

`byte` `short` `int` `long` `float` `double`

Ein wenig mehr über primitive Datentypen

`boolean`

`byte` `short` `int` `long` `float` `double`

`char`

Ein wenig mehr über primitive Datentypen

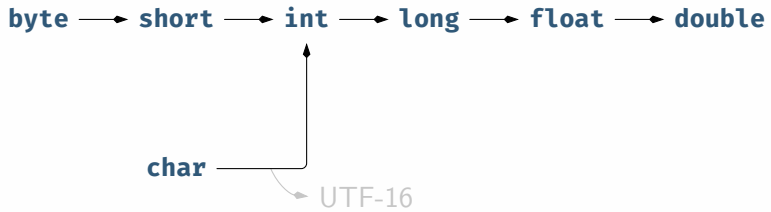
`boolean`

`byte` → `short` → `int` → `long` → `float` → `double`

`char`

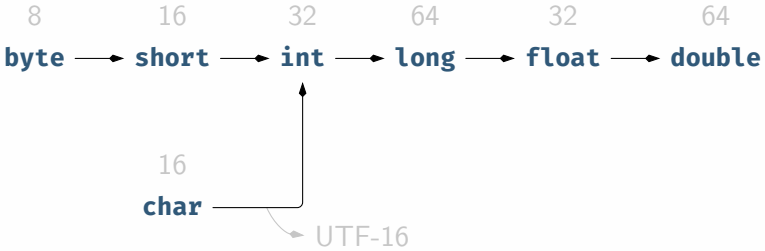
Ein wenig mehr über primitive Datentypen

`boolean`

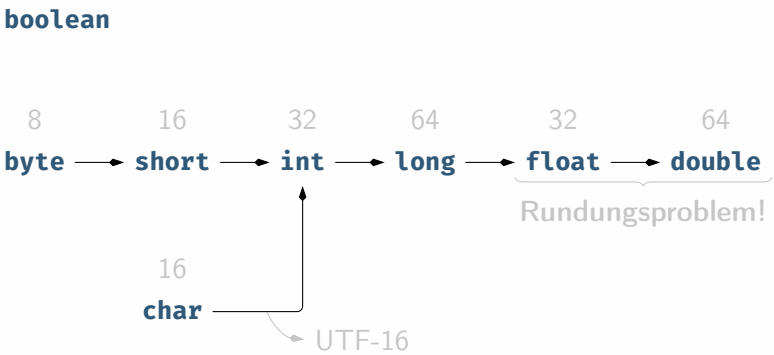


Ein wenig mehr über primitive Datentypen

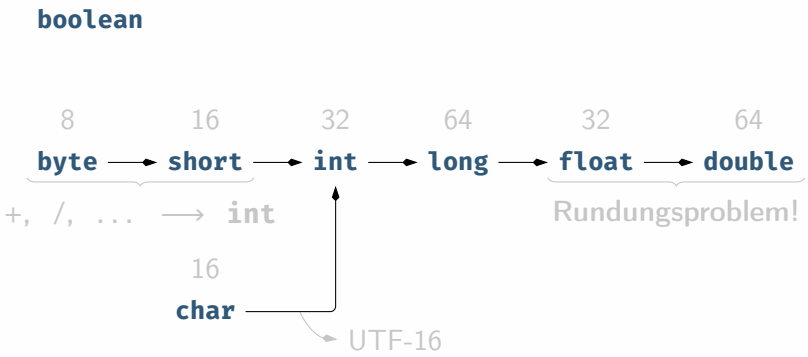
boolean



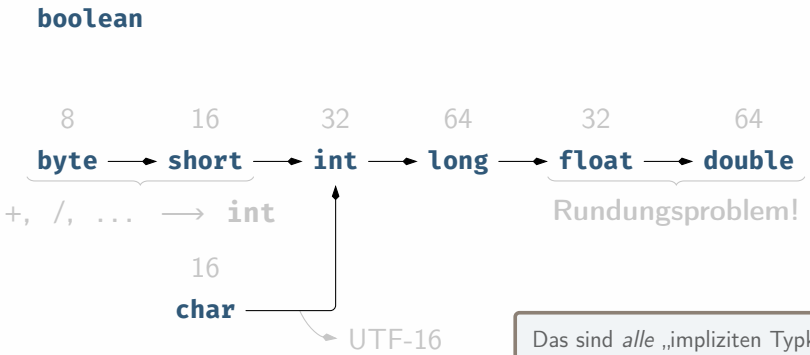
Ein wenig mehr über primitive Datentypen



Ein wenig mehr über primitive Datentypen



Ein wenig mehr über primitive Datentypen



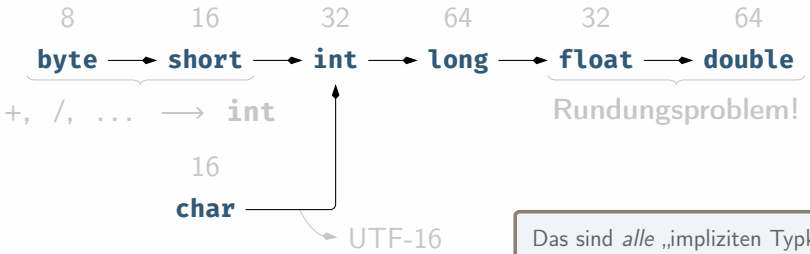
Das sind *alle* „impliziten Typkonvertierungen“ in Java.

String hat da nichts zu suchen.

Ein wenig mehr über primitive Datentypen

void ist in Java kein Datentyp!
JLS17 4.2 & 14.8

boolean



Das sind *alle* „impliziten Typkonvertierungen“ in Java.

String hat da nichts zu suchen.

- `byte`

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$

  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.

- `short`

- `byte` fasst $[\overbrace{-128}^{-2^7} \text{ bis } \overbrace{127}^{2^7-1}]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[\overbrace{-32\,768}^{-2^{15}} \text{ bis } \overbrace{32\,767}^{2^{15}-1}]$

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[-2^{15} = -32\,768 \text{ bis } 2^{15} - 1 = 32\,767]$
  32 767 Sekunden sind ≈ 9,1 Stunden.

- `byte` fasst $[-2^7 = -128$ bis $2^7 - 1 = 127]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[-2^{15} = -32\,768$ bis $2^{15} - 1 = 32\,767]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char`

Primitives Fassungsvermögen

Ganzzahlwerte

- `byte` fasst $[\overset{-2^7}{-128}$ bis $\overset{2^7-1}{127}]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[\overset{-2^{15}}{-32\,768}$ bis $\overset{2^{15}-1}{32\,767}]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[\overset{0}{0}$ bis $\overset{2^{16}-1}{65\,535}]$

- `byte` fasst $[\overset{-2^7}{-128}$ bis $\overset{2^7-1}{127}]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[\overset{-2^{15}}{-32\,768}$ bis $\overset{2^{15}-1}{32\,767}]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[\overset{0}{0}$ bis $\overset{2^{16}-1}{65\,535}]$
  Das reicht für alle Code Units von UTF-16!

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[-2^{15} = -32\,768 \text{ bis } 2^{15} - 1 = 32\,767]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[0 \text{ bis } 2^{16} - 1 = 65\,535]$
  Das reicht für alle Code Units von UTF-16!
- `int`

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[-2^{15} = -32\,768 \text{ bis } 2^{15} - 1 = 32\,767]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[0 \text{ bis } 2^{16} - 1 = 65\,535]$
  Das reicht für alle Code Units von UTF-16!
- `int` fasst $[-2^{31} = -2\,147\,483\,648 \text{ bis } 2^{31} - 1 = 2\,147\,483\,647]$

Primitives Fassungsvermögen

Ganzzahlwerte

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[-2^{15} = -32\,768 \text{ bis } 2^{15} - 1 = 32\,767]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[0 \text{ bis } 2^{16} - 1 = 65\,535]$
  Das reicht für alle Code Units von UTF-16!
- `int` fasst $[-2^{31} = -2\,147\,483\,648 \text{ bis } 2^{31} - 1 = 2\,147\,483\,647]$
  2 147 483 647 Sekunden sind ≈ 68,05 Jahre.

- `byte` fasst $[\overset{-2^7}{-128}$ bis $\overset{2^7-1}{127}]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[\overset{-2^{15}}{-32\,768}$ bis $\overset{2^{15}-1}{32\,767}]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[\overset{0}{0}$ bis $\overset{2^{16}-1}{65\,535}]$
  Das reicht für alle Code Units von UTF-16!
- `int` fasst $[\overset{-2^{31}}{-2\,147\,483\,648}$ bis $\overset{2^{31}-1}{2\,147\,483\,647}]$
  2 147 483 647 Sekunden sind ≈ 68,05 Jahre.
- `long`

- `byte` fasst $[\overset{-2^7}{-128}\text{ bis }\overset{2^7-1}{127}]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[\overset{-2^{15}}{-32\,768}\text{ bis }\overset{2^{15}-1}{32\,767}]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[\overset{0}{0}\text{ bis }\overset{2^{16}-1}{65\,535}]$
  Das reicht für alle Code Units von UTF-16!
- `int` fasst $[\overset{-2^{31}}{-2\,147\,483\,648}\text{ bis }\overset{2^{31}-1}{2\,147\,483\,647}]$
  2 147 483 647 Sekunden sind ≈ 68,05 Jahre.
- `long` fasst $[\overset{-2^{63}}{-9\,223\,372\,036\,854\,775\,808}\text{ bis }\overset{2^{63}-1}{9\,223\,372\,036\,854\,775\,807}]$

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[-2^{15} = -32\,768 \text{ bis } 2^{15} - 1 = 32\,767]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[0 \text{ bis } 2^{16} - 1 = 65\,535]$
  Das reicht für alle Code Units von UTF-16!
- `int` fasst $[-2^{31} = -2\,147\,483\,648 \text{ bis } 2^{31} - 1 = 2\,147\,483\,647]$
  2 147 483 647 Sekunden sind ≈ 68,05 Jahre.
- `long` fasst $[-2^{63} = -9\,223\,372\,036\,854\,775\,808 \text{ bis } 2^{63} - 1 = 9\,223\,372\,036\,854\,775\,807]$
  9 223 372 036 854 775 807 Sekunden sind ≈ 292 271 023 045,3 Jahre ≈ 658 Millionen mal „älter“, als das Universum.

- `byte` fasst $[-2^7 = -128 \text{ bis } 2^7 - 1 = 127]$
  Das reicht aus, um alle Primzahlen zwischen 2 000 und 3 000 zu zählen. 127 Sekunden sind ≈ 0,035 Stunden.
- `short` fasst $[-2^{15} = -32\,768 \text{ bis } 2^{15} - 1 = 32\,767]$
  32 767 Sekunden sind ≈ 9,1 Stunden.
- `char` fasst $[0 \text{ bis } 2^{16} - 1 = 65\,535]$
  Das reicht für alle Code Units von UTF-16!
- `int` fasst $[-2^{31} = -2\,147\,483\,648 \text{ bis } 2^{31} - 1 = 2\,147\,483\,647]$
  2 147 483 647 Sekunden sind ≈ 68,05 Jahre.
- `long` fasst $[-2^{63} = -9\,223\,372\,036\,854\,775\,808 \text{ bis } 2^{63} - 1 = 9\,223\,372\,036\,854\,775\,807]$
  9 223 372 036 854 775 807 Sekunden sind ≈ 292 271 023 045,3 Jahre ≈ 658 Millionen mal „älter“, als das Universum.

JLS17 4.2.1, JVMS17 2.3.1

- `float`

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$

  Joa… Das sind mehr als $5 \cdot 10^{30}$ Jahre.

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$
  Joa… Das sind mehr als $5 \cdot 10^{30}$ Jahre.

- `double`

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$
  Joa… Das sind mehr als $5 \cdot 10^{30}$ Jahre.
- `double` schweift zwischen $\pm 2^{-1\,022}$ bis zu $(2 - 2^{-52}) \cdot 2^{1\,023}$

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$
  Joa… Das sind mehr als $5 \cdot 10^{30}$ Jahre.
- `double` schweift zwischen $\pm 2^{-1\,022}$ bis zu $(2 - 2^{-52}) \cdot 2^{1\,023}$
  So zählen sie sich leicht, die Atome des Universums!

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$
  Joa… Das sind mehr als $5 \cdot 10^{30}$ Jahre.
- `double` schweift zwischen $\pm 2^{-1\,022}$ bis zu $(2 - 2^{-52}) \cdot 2^{1\,023}$
  So zählen sie sich leicht, die Atome des Universums!

- Dabei sollten wir die Rundungsprobleme aber nicht außer Acht lassen

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$
  Joa… Das sind mehr als $5 \cdot 10^{30}$ Jahre.
- `double` schweift zwischen $\pm 2^{-1\,022}$ bis zu $(2 - 2^{-52}) \cdot 2^{1\,023}$
  So zählen sie sich leicht, die Atome des Universums!

- Dabei sollten wir die Rundungsprobleme aber nicht außer Acht lassen
- Jenseits des Primitiven liefert Java natürlich mehr

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$
  Joa… Das sind mehr als $5 \cdot 10^{30}$ Jahre.
- `double` schweift zwischen $\pm 2^{-1\,022}$ bis zu $(2 - 2^{-52}) \cdot 2^{1\,023}$
  So zählen sie sich leicht, die Atome des Universums!

- Dabei sollten wir die Rundungsprobleme aber nicht außer Acht lassen
- Jenseits des Primitiven liefert Java natürlich mehr
  - `BigInteger` begrenzt die Anzahl Bits durch einen `int`

- `float` schweift zwischen $\pm 2^{-126}$ bis zu $(2 - 2^{-23}) \cdot 2^{127}$
  Joa… Das sind mehr als $5 \cdot 10^{30}$ Jahre.
- `double` schweift zwischen $\pm 2^{-1\,022}$ bis zu $(2 - 2^{-52}) \cdot 2^{1\,023}$
  So zählen sie sich leicht, die Atome des Universums!

- Dabei sollten wir die Rundungsprobleme aber nicht außer Acht lassen
- Jenseits des Primitiven liefert Java natürlich mehr
  - `BigInteger` begrenzt die Anzahl Bits durch einen `int`
  - `BigDecimal` erlaubt Skalieren durch `int` und volle Kontrolle übers Runden

Aufgabe 3: Boolesche Ausdrücke

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen `b` an.

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen `b` an.

```
boolean b = true;
```

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen `b` an.

```
boolean b = true;

b = !(true || false) || b;
```

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;

b = !(true && false);
```

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;

b = !(true && false);

b = (!(true && true) || (false || false));
```

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen `b` an.

```
boolean b = true;

b = !(true || false) || b;

b = !(true && false);

b = (!(true && true) || (false || false));

b = !b;
```

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;

b = !(true && false);

b = (!(true && true) || (false || false));

b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a – b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen `b` an.

```
boolean b = true;

b = !(true || false) || b;
      ^^^^

b = !(true && false);

b = (!(true && true) || (false || false));

b = !b;
```

`true`

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a − b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen `b` an.

```
boolean b = true;

b = !(true || false) || b;
      (true)
      (false)

b = !(true && false);

b = (!(true && true) || (false || false));

b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a − b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;
      true   false    true

b = !(true && false);

b = (!(true && true) || (false || false));

b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a − b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;
```

true (für `true || false`); false (für `!(true || false)`); true (für `b`)

b ist ***true***

```
b = !(true && false);

b = (!(true && true) || (false || false));

b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a – b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;
b = !(true && false);

b = (!(true && true) || (false || false));

b = !b;
```

true / false / false / true

b ist ***true***

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a – b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;
```

true; false; false; true

b ist ***true***

```
b = !(true && false);
```

true

```
b = (!(true && true) || (false || false));

b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a − b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;
```

true; false; false; true

b ist ***true***

```
b = !(true && false);
```

false; true

b ist ***true***

```
b = (!(true && true) || (false || false));

b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a − b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;
```
true, false, true

b ist ***true***

```
b = !(true && false);
```
false, false

b ist ***true***

```
b = (!(true && true) || (false || false));
```
true, true

```
b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a – b ⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b ⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;      // b ist true
b = !(true && false);           // b ist true
b = (!(true && true) || (false || false));
b = !b;
```

true, false, false, true

true, false, true

false

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a – b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;      // b ist true
b = !(true && false);           // b ist true
b = (!(true && true) || (false || false));
b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a – b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;
```

```
b = !(true || false) || b;
```
true; false; false; false; true

b ist ***true***

```
b = !(true && false);
```
true; false

b ist ***true***

```
b = (!(true && true) || (false || false));
```
true; true; false; false

b ist ***false***

```
b = !b;
```

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a – b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;

b = !(true || false) || b;
b = !(true && false);
b = (!(true && true) || (false || false));
b = !b;
```

true
false false true
b ist **true**

false false true
b ist **true**

false true true false
b ist **false**

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a – b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;
```

```
b = !(true || false) || b;
```
true; false; false; false; true

b ist ***true***

```
b = !(true && false);
```
true; false

b ist ***true***

```
b = (!(true && true) || (false || false));
```
true; true; false; false; false

b ist ***false***

```
b = !b;
```
false; true

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a − b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 3: Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Werten Sie die folgenden Ausdrücke in der Reihenfolge, in der sie ausgeführt werden (d.h. von oben nach unten), aus. Geben Sie dazu für jede Zeile (beginnend mit der zweiten Zeile) den neuen Wert der Variablen b an.

```
boolean b = true;
```

```
b = !(true || false) || b;
```
true, false, false, true — *b ist* **true**

```
b = !(true && false);
```
true, true, false — *b ist* **true**

```
b = (!(true && true) || (false || false));
```
false, false, false — *b ist* **false**

```
b = !b;
```
true — *b ist* **true**

a++, a-- ⟶ !a, −a, ++a, --a ⟶ a ∗ b, a / b, a % b ⟶ a + b, a − b
⟶ a == b, a < b, . . . ⟶ a ^ b ⟶ a && b
⟶ a || b ⟶ . . .

Aufgabe 4: Varablen anlegen und verarbeiten

Aufgabe 4: Varablen anlegen und verarbeiten

In der Vorlesung haben Sie gelernt, wie in Java Variablen deklariert und initialisiert werden. Nun sollen Sie in einem kleinen Beispiel diese Konzepte selbst umsetzen. Betrachten Sie die `.java`*-Datei namens* `Variablen.java` *mit folgendem Inhalt:*

Aufgabe 4: Varablen anlegen und verarbeiten

In der Vorlesung haben Sie gelernt, wie in Java Variablen deklariert und initialisiert werden. Nun sollen Sie in einem kleinen Beispiel diese Konzepte selbst umsetzen. Betrachten Sie die `.java`*-Datei namens* `Variablen.java` *mit folgendem Inhalt:*

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

    }
}
```

Aufgabe 4: Varablen anlegen und verarbeiten

In der Vorlesung haben Sie gelernt, wie in Java Variablen deklariert und initialisiert werden. Nun sollen Sie in einem kleinen Beispiel diese Konzepte selbst umsetzen. Betrachten Sie die `.java`*-Datei namens* `Variablen.java` *mit folgendem Inhalt:*

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


   }
}
```

a) *Erweitern Sie den Code so, dass zwei int Variablen deklariert werden. Initialisieren Sie diese anschließend mit einem beliebigen gültigen Wert und addieren Sie sie. Das Ergebnis dieser Addition soll in eine neue dritte int Variable gespeichert werden. Geben Sie das Ergebnis dann über den Aufruf System.out.println(name) aus, wobei Sie name entsprechend ersetzen.*

Aufgabe 4: Varablen anlegen und verarbeiten

In der Vorlesung haben Sie gelernt, wie in Java Variablen deklariert und initialisiert werden. Nun sollen Sie in einem kleinen Beispiel diese Konzepte selbst umsetzen. Betrachten Sie die `.java`*-Datei namens* `Variablen.java` *mit folgendem Inhalt:*

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


   }
}
```

a) *Erweitern Sie den Code so, dass zwei int Variablen deklariert werden. Initialisieren Sie diese anschließend mit einem beliebigen gültigen Wert und addieren Sie sie. Das Ergebnis dieser Addition soll in eine neue dritte int Variable gespeichert werden. Geben Sie das Ergebnis dann über den Aufruf System.out.println(name) aus, wobei Sie name entsprechend ersetzen.*

b) *Geben Sie als Kommentar im Java Code aus Teilaufgabe a) jeweils an, ob es ich bei dieser Zeile um eine Deklaration, Initialisierung oder Zuweisung handelt.*

Aufgabe 4: Varablen anlegen und verarbeiten

In der Vorlesung haben Sie gelernt, wie in Java Variablen deklariert und initialisiert werden. Nun sollen Sie in einem kleinen Beispiel diese Konzepte selbst umsetzen. Betrachten Sie die `.java`*-Datei namens* `Variablen.java` *mit folgendem Inhalt:*

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


   }
}
```

a) *Erweitern Sie den Code so, dass zwei int Variablen deklariert werden. Initialisieren Sie diese* ***anschließend*** *mit einem beliebigen gültigen Wert und addieren Sie sie. Das Ergebnis dieser Addition soll in eine neue dritte int Variable gespeichert werden. Geben Sie das Ergebnis dann über den Aufruf System.out.println(name) aus, wobei Sie name entsprechend ersetzen.*

b) *Geben Sie als Kommentar im Java Code aus Teilaufgabe a) jeweils an, ob es ich bei dieser Zeile um eine Deklaration, Initialisierung oder Zuweisung handelt.*

Simple Adds and Stuff

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

```
class Variablen {




}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

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




   }
}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

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




   }
}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;




   }
}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;
      zahl1 = 40;
      zahl2 = 2;


   }
}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;
      zahl1 = 40;
      zahl2 = 2;
      ergebnis = zahl1 + zahl2;

   }
}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;
      zahl1 = 40;
      zahl2 = 2;
      ergebnis = zahl1 + zahl2;
      System.out.println(ergebnis);
   }
}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

b) Geben Sie an, wo Deklarationen, Initialisierungen und Zuweisungen stattfinden.

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;
      zahl1 = 40;
      zahl2 = 2;
      ergebnis = zahl1 + zahl2;
      System.out.println(ergebnis);
   }
}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

b) Geben Sie an, wo Deklarationen, Initialisierungen und Zuweisungen stattfinden.

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;
      zahl1 = 40;
      zahl2 = 2;
      ergebnis = zahl1 + zahl2;
      System.out.println(ergebnis);
   }
}
```

*Alle drei Variablen werden **deklariert***

Simple Adds and Stuff

a) *Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)*

b) *Geben Sie an, wo Deklarationen, Initialisierungen und Zuweisungen stattfinden.*

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;  Alle drei Variablen werden deklariert
      zahl1 = 40;  Eine Initialisierung mit Wert 40
      zahl2 = 2;
      ergebnis = zahl1 + zahl2;
      System.out.println(ergebnis);
   }
}
```

Simple Adds and Stuff

a) *Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)*

b) *Geben Sie an, wo Deklarationen, Initialisierungen und Zuweisungen stattfinden.*

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;   Alle drei Variablen werden deklariert
      zahl1 = 40;   Eine Initialisierung mit Wert 40
      zahl2 = 2;   Eine Initialisierung mit Wert 2
      ergebnis = zahl1 + zahl2;
      System.out.println(ergebnis);
   }
}
```

Simple Adds and Stuff

a) *Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)*

b) *Geben Sie an, wo Deklarationen, Initialisierungen und Zuweisungen stattfinden.*

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;   Alle drei Variablen werden deklariert
      zahl1 = 40;   Eine Initialisierung mit Wert 40
      zahl2 = 2;   Eine Initialisierung mit Wert 2
      ergebnis = zahl1 + zahl2;   Eine Initialisierung mit dem Ergebnis der Summe
      System.out.println(ergebnis);
   }
}
```

Simple Adds and Stuff

a) *Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)*

b) *Geben Sie an, wo Deklarationen, Initialisierungen und Zuweisungen stattfinden.*

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;   Alle drei Variablen werden deklariert
      zahl1 = 40;   Eine Initialisierung mit Wert 40
      zahl2 = 2;   Eine Initialisierung mit Wert 2
      ergebnis = zahl1 + zahl2;   Eine Initialisierung mit dem Ergebnis der Summe
      System.out.println(ergebnis);   Eine Ausgabe (für die b irrelevant)
   }
}
```

Simple Adds and Stuff

a) Erweitern Sie den Code (Deklarieren, Initialisieren, Addieren und Ausgeben)

b) Geben Sie an, wo Deklarationen, Initialisierungen und Zuweisungen stattfinden.

```
class Variablen {
   public static void main(String[] args) {
      int zahl1, zahl2, ergebnis;   Alle drei Variablen werden deklariert
      zahl1 = 40;   Eine Initialisierung mit Wert 40
      zahl2 = 2;   Eine Initialisierung mit Wert 2
      ergebnis = zahl1 + zahl2;   Eine Initialisierung mit dem Ergebnis der Summe
      System.out.println(ergebnis);   Eine Ausgabe (für die b irrelevant)
   }
}
```

Ist es wichtig, dass die letzte Initialisierung mit einer Summe ist? Nö. Es ist einfach nur der erste Wert, den `Ergebnis` erhält.

Aussicht: Übungsblatt 3

Ein kleiner Schwenk

Ein kleiner Schwenk

- Wenn wir Java-Code möchten, steht das da (ab jetzt wird das der Standardfall sein).

Ein kleiner Schwenk

- Wenn wir Java-Code möchten, steht das da (ab jetzt wird das der Standardfall sein).
- In Aufgabe 1 suchen wir *einen* booleschen Ausdruck der das Programm abbildet.

Ein kleiner Schwenk

- Wenn wir Java-Code möchten, steht das da (ab jetzt wird das der Standardfall sein).
- In Aufgabe 1 suchen wir *einen* booleschen Ausdruck der das Programm abbildet.
  Vergleiche hierzu die Präsenzaufgabe von letzter Woche.

Ein kleiner Schwenk

- Wenn wir Java-Code möchten, steht das da (ab jetzt wird das der Standardfall sein).
- In Aufgabe 1 suchen wir *einen* booleschen Ausdruck der das Programm abbildet.
  Vergleiche hierzu die Präsenzaufgabe von letzter Woche.
- Für die Schleifen der 2. Aufgabe, kann man sich die Alternativen dieser Präsenzaufgabe nochmal ansehen.

Ein kleiner Schwenk

- Wenn wir Java-Code möchten, steht das da (ab jetzt wird das der Standardfall sein).
- In Aufgabe 1 suchen wir *einen* booleschen Ausdruck der das Programm abbildet.
  Vergleiche hierzu die Präsenzaufgabe von letzter Woche.
- Für die Schleifen der 2. Aufgabe, kann man sich die Alternativen dieser Präsenzaufgabe nochmal ansehen.
- Allgemein gilt: Wenn Pseudocode oder ein Ansatz gegeben ist, soll der Algorithmus auch diesen Verwenden oder darauf aufbauen!

Aufgabe 3: Algorithmen in Java implementieren

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.
- Viele einzelne Punkte werden nun kollektiv Wichtig

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.
- Viele einzelne Punkte werden nun kollektiv Wichtig
  - Einhalten der Java-Syntax

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.
- Viele einzelne Punkte werden nun kollektiv Wichtig
  - Einhalten der Java-Syntax
  - Welcher Datentyp passt am besten?

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.
- Viele einzelne Punkte werden nun kollektiv Wichtig
  - Einhalten der Java-Syntax
  - Welcher Datentyp passt am besten?
  - Welche Art von Schleife, Fallunterscheidung, … eignet sich?

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.
- Viele einzelne Punkte werden nun kollektiv Wichtig
  - Einhalten der Java-Syntax
  - Welcher Datentyp passt am besten?
  - Welche Art von Schleife, Fallunterscheidung, … eignet sich?
  - Was drückt der Pseudocode implizit aus, ist aber in Java explizit notwendig?

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.
- Viele einzelne Punkte werden nun kollektiv Wichtig
  - Einhalten der Java-Syntax
  - Welcher Datentyp passt am besten?
  - Welche Art von Schleife, Fallunterscheidung, … eignet sich?
  - Was drückt der Pseudocode implizit aus, ist aber in Java explizit notwendig?
  - Und umgekehrt: was müssen wir in Java nicht schreiben?

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.
- Viele einzelne Punkte werden nun kollektiv Wichtig
  - Einhalten der Java-Syntax
  - Welcher Datentyp passt am besten?
  - Welche Art von Schleife, Fallunterscheidung, … eignet sich?
  - Was drückt der Pseudocode implizit aus, ist aber in Java explizit notwendig?
  - Und umgekehrt: was müssen wir in Java nicht schreiben?
  - …

Aufgabe 3: Algorithmen in Java implementieren

- Eine abstrakte Notation wird konkret.
- Viele einzelne Punkte werden nun kollektiv Wichtig
  - Einhalten der Java-Syntax
  - Welcher Datentyp passt am besten?
  - Welche Art von Schleife, Fallunterscheidung, … eignet sich?
  - Was drückt der Pseudocode implizit aus, ist aber in Java explizit notwendig?
  - Und umgekehrt: was müssen wir in Java nicht schreiben?
  - …
- Kommentare empfehlen sich da, wo der Gedanke des Algorithmus hinter Java-Gebrabbel verschwindet. (If you are interested, consider looking up literate programming [Knu84])

Aufgabe 4: Iterative Wurzelberechnung

Aufgabe 4: Iterative Wurzelberechnung

- Für die Fließkommazahl gibt es `Float.parseFloat` oder `Double.parseDouble`

Aufgabe 4: Iterative Wurzelberechnung

- Für die Fließkommazahl gibt es `Float.parseFloat` oder `Double.parseDouble`
- Verwendet das Heron-Verfahren wie angegeben (keine alternative Wurzelberechnung)

Aufgabe 4: Iterative Wurzelberechnung

- Für die Fließkommazahl gibt es `Float.parseFloat` oder `Double.parseDouble`
- Verwendet das Heron-Verfahren wie angegeben (keine alternative Wurzelberechnung)
- Ihr dürft gerne eure Entscheidungen in den Kommentaren begründen.

Aufgabe 4: Iterative Wurzelberechnung

- Für die Fließkommazahl gibt es `Float.parseFloat` oder `Double.parseDouble`
- Verwendet das Heron-Verfahren wie angegeben (keine alternative Wurzelberechnung)
- Ihr dürft gerne eure Entscheidungen in den Kommentaren begründen.
- An welcher Stelle kommen Unteralgorithmen zum Einsatz?
  Und… Wo könnten sie noch hilfreich sein?

Der zeitliche Abstand […] läßt den wahren Sinn, der in einer Sache liegt, erst voll herauskommen. Die Ausschöpfung des wahren Sinnes aber, der in einem Text oder einer künstlerischen Schöpfung gelegen ist, kommt nicht irgendwo zum Abschluß, sondern ist in Wahrheit ein unendlicher Prozeß.
— Hans-Georg Gadamer [Gad60, unverändert, p. 282]

Abschließendes

Zusammenfassend

Zusammenfassend

[Gad60] Hans-Georg Gadamer. „Wahrheit und Methode“. 1960

[Knu84] Donald Ervin Knuth. „Literate programming“. 1984

Zusammenfassend

[Gad60] Hans-Georg Gadamer. „Wahrheit und Methode“. 1960

[Knu84] Donald Ervin Knuth. „Literate programming“. 1984

- Kommandozeilenparameter landen im `args`-Array (als `String`)

Zusammenfassend

[Gad60] Hans-Georg Gadamer. „Wahrheit und Methode“. 1960

[Knu84] Donald Ervin Knuth. „Literate programming“. 1984

- Kommandozeilenparameter landen im `args`-Array (als `String`)
- `Integer.parseInt("13")` liefert die vom `String` beschriebene Ganzzahl (*13*)

Zusammenfassend

[Gad60] Hans-Georg Gadamer. „Wahrheit und Methode“. 1960

[Knu84] Donald Ervin Knuth. „Literate programming“. 1984

- Kommandozeilenparameter landen im `args`-Array (als `String`)
- `Integer.parseInt("13")` liefert die vom `String` beschriebene Ganzzahl (*13*)
- Unteralgorithmen sind ein elementarer Programmiermechanismus

Zusammenfassend

[Gad60] Hans-Georg Gadamer. „Wahrheit und Methode“. 1960

[Knu84] Donald Ervin Knuth. „Literate programming“. 1984

- Kommandozeilenparameter landen im `args`-Array (als `String`)
- `Integer.parseInt("13")` liefert die vom `String` beschriebene Ganzzahl (*13*)
- Unteralgorithmen sind ein elementarer Programmiermechanismus
- Javas implizite Typkonvertierungen existieren nur für primitive Datentypen

Zusammenfassend

[Gad60] Hans-Georg Gadamer. „Wahrheit und Methode“. 1960

[Knu84] Donald Ervin Knuth. „Literate programming“. 1984

- Kommandozeilenparameter landen im `args`-Array (als `String`)
- `Integer.parseInt("13")` liefert die vom `String` beschriebene Ganzzahl (*13*)
- Unteralgorithmen sind ein elementarer Programmiermechanismus
- Javas implizite Typkonvertierungen existieren nur für primitive Datentypen
  - `byte` → `short` → `int` → `long` → `double` und `char` → `int`

Zusammenfassend

[Gad60] Hans-Georg Gadamer. „Wahrheit und Methode“. 1960

[Knu84] Donald Ervin Knuth. „Literate programming“. 1984

- Kommandozeilenparameter landen im `args`-Array (als `String`)
- `Integer.parseInt("13")` liefert die vom `String` beschriebene Ganzzahl (*13*)
- Unteralgorithmen sind ein elementarer Programmiermechanismus
- Javas implizite Typkonvertierungen existieren nur für primitive Datentypen
  - `byte` → `short` → `int` → `long` → `double` und `char` → `int`
  - „Zahlen von klein nach groß, `char` zu `int`“