

Tutorium Programmieren

Tut Nr.8: FileReader, String API

Michael Friedrich | 17. / 19.12.2013

INSTITUT FÜR THEORETISCHE INFORMATIK



- 1 Blatt3
- 2 Blatt4
- 3 FileReader
- 4 String API
- 5 Quiz

- nur geforderte Dateien hochladen!
- achtet auf die Aufgabenstellung!
 - Modellierung! \Rightarrow Methoden, nicht alles in die main
 - Ausgabe! \Rightarrow Leerzeichen können euch in der Abschlusssaufgabe teuer kosten
 - Aufgabe lesen! zB args \Rightarrow bei Unklarheit fragen!

- nur geforderte Dateien hochladen!
- achtet auf die Aufgabenstellung!
 - Modellierung! \Rightarrow Methoden, nicht alles in die main
 - Ausgabe! \Rightarrow Leerzeichen können euch in der Abschlusssaufgabe teuer kosten
 - Aufgabe lesen! zB args \Rightarrow bei Unklarheit fragen!

- nur geforderte Dateien hochladen!
- achtet auf die Aufgabenstellung!
 - Modellierung! \Rightarrow Methoden, nicht alles in die main
 - Ausgabe! \Rightarrow Leerzeichen können euch in der Abschlusssaufgabe teuer kosten
 - Aufgabe lesen! zB args \Rightarrow bei Unklarheit fragen!

- nur geforderte Dateien hochladen!
- achtet auf die Aufgabenstellung!
 - Modellierung! ⇒ Methoden, nicht alles in die main
 - Ausgabe! ⇒ Leerzeichen können euch in der Abschlusssaufgabe teuer kosten
 - Aufgabe lesen! zB args ⇒ bei Unklarheit fragen!

- nur geforderte Dateien hochladen!
- achtet auf die Aufgabenstellung!
 - Modellierung! \Rightarrow Methoden, nicht alles in die main
 - Ausgabe! \Rightarrow Leerzeichen können euch in der Abschlusssaufgabe teuer kosten
 - Aufgabe lesen! zB args \Rightarrow bei Unklarheit fragen!

- nur geforderte Dateien hochladen!
- achtet auf die Aufgabenstellung!
 - Modellierung! \Rightarrow Methoden, nicht alles in die main
 - Ausgabe! \Rightarrow Leerzeichen können euch in der Abschlusssaufgabe teuer kosten
 - Aufgabe lesen! zB args \Rightarrow bei Unklarheit fragen!

I made a Boo Boo

- Fehler meinerseits...Kein Abzug, nur Anmerkung in der Korrektur. Aber in Zukunft Fehler!
 - Exceptions unbedingt zu vermeiden!

Lücke füllen und damit implizit löschen

```
for(int j = indexFound; j < elements.length; j++) {  
    elements[j] = elements[j+1];  
}  
lastElem--;  
}
```

Führt zu einer Exception! Ähnlich auch in Aufgabenteil C

⇒ Entweder: Nur bis zu $(\text{elements.length} - 1)$ gehen und letztes Element außerhalb der for Schleife manipulieren

⇒ ODER : Indizierung anders wählen: $\text{elements}[j-1] = \text{elements}[j]$ im for-Rumpf

I made a Boo Boo

- Fehler meinerseits...Kein Abzug, nur Anmerkung in der Korrektur. Aber in Zukunft Fehler!
 - Exceptions unbedingt zu vermeiden!

Lücke füllen und damit implizit löschen

```
for(int j = indexFound; j < elements.length; j++) {  
    elements[j] = elements[j+1];  
}  
lastElem--;  
}
```

Führt zu einer Exception! Ähnlich auch in Aufgabenteil C

⇒ Entweder: Nur bis zu $(\text{elements.length} - 1)$ gehen und letztes Element außerhalb der for Schleife manipulieren

⇒ ODER : Indizierung anders wählen: $\text{elements}[j-1] = \text{elements}[j]$ im for-Rumpf

I made a Boo Boo

- Fehler meinerseits...Kein Abzug, nur Anmerkung in der Korrektur. Aber in Zukunft Fehler!
 - Exceptions unbedingt zu vermeiden!

Lücke füllen und damit implizit löschen

```
for(int j = indexFound; j < elements.length; j++) {  
    elements[j] = elements[j+1];  
}  
lastElem--;  
}
```

Führt zu einer Exception! Ähnlich auch in Aufgabenteil C
⇒ Entweder: Nur bis zu $(\text{elements.length} - 1)$ gehen und letztes Element außerhalb der for Schleife manipulieren
⇒ ODER : Indizierung anders wählen: $\text{elements}[j-1] = \text{elements}[j]$ im for-Rumpf

I made a Boo Boo

- Fehler meinerseits...Kein Abzug, nur Anmerkung in der Korrektur. Aber in Zukunft Fehler!
 - Exceptions unbedingt zu vermeiden!

Lücke füllen und damit implizit löschen

```
for(int j = indexFound; j < elements.length; j++) {  
    elements[j] = elements[j+1];  
}  
lastElem--;  
}
```

Führt zu einer Exception! Ähnlich auch in Aufgabenteil C

⇒ Entweder: Nur bis zu $(\text{elements.length} - 1)$ gehen und letztes Element außerhalb der for Schleife manipulieren

⇒ ODER : Indizierung anders wählen: $\text{elements}[j-1] = \text{elements}[j]$ im for-Rumpf

I made a Boo Boo

- Fehler meinerseits...Kein Abzug, nur Anmerkung in der Korrektur. Aber in Zukunft Fehler!
 - Exceptions unbedingt zu vermeiden!

Lücke füllen und damit implizit löschen

```
for(int j = indexFound; j < elements.length; j++) {  
    elements[j] = elements[j+1];  
}  
lastElem--;  
}
```

Führt zu einer Exception! Ähnlich auch in Aufgabenteil C

⇒ Entweder: Nur bis zu $(\text{elements.length} - 1)$ gehen und letztes Element außerhalb der for Schleife manipulieren

⇒ ODER : Indizierung anders wählen: $\text{elements}[j-1] = \text{elements}[j]$ im for-Rumpf

I made a Boo Boo

- Fehler meinerseits...Kein Abzug, nur Anmerkung in der Korrektur. Aber in Zukunft Fehler!
 - Exceptions unbedingt zu vermeiden!

Lücke füllen und damit implizit löschen

```
for(int j = indexFound; j < elements.length; j++) {  
    elements[j] = elements[j+1];  
}  
lastElem--;  
}
```

Führt zu einer Exception! Ähnlich auch in Aufgabenteil C

⇒ Entweder: Nur bis zu $(\text{elements.length} - 1)$ gehen und letztes Element außerhalb der for Schleife manipulieren

⇒ ODER : Indizierung anders wählen: $\text{elements}[j-1] = \text{elements}[j]$ im for-Rumpf

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute private
 - Methoden meistens public (außer Hilfsmethoden)
 - Konstruktor **IMMER public**
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute private
 - Methoden meistens public (außer Hilfsmethoden)
 - Konstruktor **IMMER public**
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute `private`
 - Methoden meistens `public` (außer Hilfsmethoden)
 - Konstruktor **IMMER** `public`
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute `private`
 - Methoden meistens `public` (außer Hilfsmethoden)
 - Konstruktor **IMMER** `public`
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute private
 - Methoden meistens public (außer Hilfsmethoden)
 - Konstruktor **IMMER public**
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute private
 - Methoden meistens public (außer Hilfsmethoden)
 - Konstruktor **IMMER public**
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute private
 - Methoden meistens public (außer Hilfsmethoden)
 - Konstruktor **IMMER public**
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute private
 - Methoden meistens public (außer Hilfsmethoden)
 - Konstruktor **IMMER public**
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

- Kapselung! Jetzt Pflicht...Wiederholungsbedarf?
- „Faustregeln“
 - Attribute private
 - Methoden meistens public (außer Hilfsmethoden)
 - Konstruktor **IMMER public**
- Vergesst nicht, Checkstyle upzudaten
- Listen erstellen wie letztes Tut

```
public static void main(String[] args) {  
  
    if (args.length != 1) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
    FileReader in = null;  
    try {  
        in = new FileReader(args[0]);  
    } catch (FileNotFoundException e) {  
        System.out.println(ERROR_MESSAGE);  
        System.exit(1);  
    }  
    BufferedReader reader = new  
        BufferedReader(in);  
    try {  
        String line = reader.readLine();  
        while (line != null) {  
            // TODO: process line here  
            line = reader.readLine();  
        } catch (IOException e) {  
            System.out.println(ERROR_MESSAGE);  
            System.exit(1);  
        }  
    }  
}
```

- **FileReader** öffnet die Datei
- **BufferedReader** nimmt den Datenstrom aus der Datei entgegen
→ Datei lässt sich Zeilenweise auslesen
- Fehlerbehandlung muss hier geschehen, lernen wir aber erst nächstes Jahr
→ Ihr könnt das nickend annehmen und benutzen
⇒ **TODO** euer Angriffspunkt


```
public static void main(String[] args) {  
  
    if (args.length != 1) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
    FileReader in = null;  
    try {  
        in = new FileReader(args[0]);  
    } catch (FileNotFoundException e) {  
        System.out.println(ERROR_MESSAGE);  
        System.exit(1);  
    }  
    BufferedReader reader = new  
        BufferedReader(in);  
    try {  
        String line = reader.readLine();  
        while (line != null) {  
            // TODO: process line here  
            line = reader.readLine();  
        } catch (IOException e) {  
            System.out.println(ERROR_MESSAGE);  
            System.exit(1);  
        }  
    }  
}
```

- **FileReader** öffnet die Datei
- **BufferedReader** nimmt den Datenstrom aus der Datei entgegen
→ Datei lässt sich Zeilenweise auslesen
- Fehlerbehandlung muss hier geschehen, lernen wir aber erst nächstes Jahr
→ Ihr könnt das nickend annehmen und benutzen
⇒ TODO euer Angriffspunkt

```
public static void main(String[] args) {  
  
    if (args.length != 1) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
    FileReader in = null;  
    try {  
        in = new FileReader(args[0]);  
    } catch (FileNotFoundException e) {  
        System.out.println(ERROR_MESSAGE);  
        System.exit(1);  
    }  
    BufferedReader reader = new  
        BufferedReader(in);  
    try {  
        String line = reader.readLine();  
        while (line != null) {  
            // TODO: process line here  
            line = reader.readLine();  
        } catch (IOException e) {  
            System.out.println(ERROR_MESSAGE);  
            System.exit(1);  
        }  
    }  
}
```

- **FileReader** öffnet die Datei
- **BufferedReader** nimmt den Datenstrom aus der Datei entgegen
→ Datei lässt sich Zeilenweise auslesen
- Fehlerbehandlung muss hier geschehen, lernen wir aber erst nächstes Jahr
→ Ihr könnt das nickend annehmen und benutzen
⇒ TODO euer Angriffspunkt

```
public static void main(String[] args) {  
  
    if (args.length != 1) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
    FileReader in = null;  
    try {  
        in = new FileReader(args[0]);  
    } catch (FileNotFoundException e) {  
        System.out.println(ERROR_MESSAGE);  
        System.exit(1);  
    }  
    BufferedReader reader = new  
        BufferedReader(in);  
    try {  
        String line = reader.readLine();  
        while (line != null) {  
            // TODO: process line here  
            line = reader.readLine();  
        } catch (IOException e) {  
            System.out.println(ERROR_MESSAGE);  
            System.exit(1);  
        }  
    }  
}
```

- **FileReader** öffnet die Datei
- **BufferedReader** nimmt den Datenstrom aus der Datei entgegen
→ Datei lässt sich Zeilenweise auslesen
- Fehlerbehandlung muss hier geschehen, lernen wir aber erst nächstes Jahr
→ Ihr könnt das nickend annehmen und benutzen
⇒ TODO euer Angriffspunkt

```
public static void main(String[] args) {  
  
    if (args.length != 1) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
    FileReader in = null;  
    try {  
        in = new FileReader(args[0]);  
    } catch (FileNotFoundException e) {  
        System.out.println(ERROR_MESSAGE);  
        System.exit(1);  
    }  
    BufferedReader reader = new  
        BufferedReader(in);  
    try {  
        String line = reader.readLine();  
        while (line != null) {  
            // TODO: process line here  
            line = reader.readLine();  
        } catch (IOException e) {  
            System.out.println(ERROR_MESSAGE);  
            System.exit(1);  
        }  
    }  
}
```

- FileReader öffnet die Datei
- BufferedReader nimmt den Datenstrom aus der Datei entgegen
→ Datei lässt sich Zeilenweise auslesen
- Fehlerbehandlung muss hier geschehen, lernen wir aber erst nächstes Jahr
→ Ihr könnt das nickend annehmen und benutzen
⇒ TODO euer Angriffspunkt

```
public static void main(String[] args) {  
  
    if (args.length != 1) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
    FileReader in = null;  
    try {  
        in = new FileReader(args[0]);  
    } catch (FileNotFoundException e) {  
        System.out.println(ERROR_MESSAGE);  
        System.exit(1);  
    }  
    BufferedReader reader = new  
        BufferedReader(in);  
    try {  
        String line = reader.readLine();  
        while (line != null) {  
            // TODO: process line here  
            line = reader.readLine();  
        } catch (IOException e) {  
            System.out.println(ERROR_MESSAGE);  
            System.exit(1);  
        }  
    }  
}
```

- FileReader öffnet die Datei
- BufferedReader nimmt den Datenstrom aus der Datei entgegen
→ Datei lässt sich Zeilenweise auslesen
- Fehlerbehandlung muss hier geschehen, lernen wir aber erst nächstes Jahr
→ Ihr könnt das nickend annehmen und benutzen
⇒ TODO euer Angriffspunkt

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein
 - analog: `String toUpperCase()`

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein
 - analog: `String toUpperCase()`

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein
 - analog: `String toUpperCase()`

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein
 - analog: `String toUpperCase()`

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein
 - analog: `String toUpperCase()`

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein
 - analog: `String toUpperCase()`

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein

- analog: `String toUpperCase()`

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein
 - analog: `String toUpperCase()`

Benötigte Theorie: **REGular EXpressions** (siehe GBI) hier erstmal Trennzeichen, wie z.Bsp. . (Punkt)

Werkzeugkasten, um Strings zu manipulieren

- `String[] split(String regex)` Trennt den String am Trennzeichen und liefert ein Array zurück

```
"boo:and:foo".split(":"); // Ergebnis: {" bo o ","and "," f o o "}
"boo:and:foo".split("o"); // Ergebnis: {" b ","",": and : f "}
```

- `boolean matches(String regex)` Prüft, ob ein String dem regulären Ausdruck entspricht

```
"A1".matches([A-Z][0-9]) // true
"a8".matches([A-Z][0-9]) // false
```

- `String toLowerCase()` Macht jedes Zeichen eines String klein
 - analog: `String toUpperCase()`

Werkzeugkasten, um Strings zu manipulieren

- `String trim()` Entfernt Whitespaces am Anfang und am Ende des Strings
- `int compareTo` Lexikographischer Vergleich, d.h. alphabetisch

```
"Adam".compareTo("Adam"); // Ergebnis == 0  
"Adam".compareTo("Eva"); // Ergebnis < 0  
"Eva".compareTo("Adam"); // Ergebnis > 0  
"Adam".compareTo("adam"); // Ergebnis < 0  
"Eva".compareTo("adam"); // Ergebnis < 0
```

Werkzeugkasten, um Strings zu manipulieren

- `String trim()` Entfernt Whitespaces am Anfang und am Ende des Strings
- `int compareTo` Lexikographischer Vergleich, d.h. alphabetisch

```
"Adam".compareTo("Adam"); // Ergebnis == 0  
"Adam".compareTo("Eva"); // Ergebnis < 0  
"Eva".compareTo("Adam"); // Ergebnis > 0  
"Adam".compareTo("adam"); // Ergebnis < 0  
"Eva".compareTo("adam"); // Ergebnis < 0
```


Werkzeugkasten, um Strings zu manipulieren

- `String trim()` Entfernt Whitespaces am Anfang und am Ende des Strings
- `int compareTo` Lexikographischer Vergleich, d.h. alphabetisch

```
"Adam".compareTo("Adam"); // Ergebnis == 0  
"Adam".compareTo("Eva"); // Ergebnis < 0  
"Eva".compareTo("Adam"); // Ergebnis > 0  
"Adam".compareTo("adam"); // Ergebnis < 0  
"Eva".compareTo("adam"); // Ergebnis < 0
```

Werkzeugkasten, um Strings zu manipulieren

- `String trim()` Entfernt Whitespaces am Anfang und am Ende des Strings
- `int compareTo` Lexikographischer Vergleich, d.h. alphabetisch

```
"Adam".compareTo("Adam"); // Ergebnis == 0  
"Adam".compareTo("Eva"); // Ergebnis < 0  
"Eva".compareTo("Adam"); // Ergebnis > 0  
"Adam".compareTo("adam"); // Ergebnis < 0  
"Eva".compareTo("adam"); // Ergebnis < 0
```

Werkzeugkasten, um Strings zu manipulieren

- `String trim()` Entfernt Whitespaces am Anfang und am Ende des Strings
- `int compareTo` Lexikographischer Vergleich, d.h. alphabetisch

```
"Adam".compareTo("Adam"); // Ergebnis == 0  
"Adam".compareTo("Eva"); // Ergebnis < 0  
"Eva".compareTo("Adam"); // Ergebnis > 0  
"Adam".compareTo("adam"); // Ergebnis < 0  
"Eva".compareTo("adam"); // Ergebnis < 0
```

- siehe pdf

- Oder: Habt ihr noch WehWehchen, die wir uns anschauen sollen?
Zum Beispiel: Listen, Sichtbarkeiten, sonstige Anliegen...?