3

Crossover Zahnbehandlungen

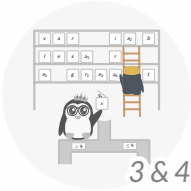
Wurzelziehen Drei

Theorie

2

13

Grundlagen



3 & 4

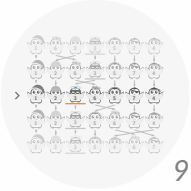
5

6

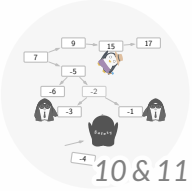
Vertiefungen

7 & 12

8



9



10 & 11

Kurzwiederholung

- Algorithmen können wir auf diverse Eigenschaften untersuchen:
  - *Determiniertheit:* die gleiche Eingabe erzeugt immer die gleiche Ausgabe
  - *Determinismus:* bei gleicher Eingabe folgt stets die gleiche Schrittfolge (⇒ Determiniert)
  - *Totale Korrektheit:* Terminiert und partiell korrekt (Terminiert ⟹ gewünschte Ausgabe)

- Manche von Javas primitiven Datentypen können implizit konvertiert werden
  - `byte` → `short` → `int` → `long` → `float` → `double` und `char` → `int`
- Javas Präzedenzregeln geben an, wie Operatoren implizit geklammert werden
  - (stark) a`++` ⟶ `!`a, `++`a ⟶ a `/` b, a `*` b ⟶ a `+` b, a `-` b ⟶ a `==` b ⟶ a `&&` b ⟶ a `||` b (schwach)
  - Bei Ganzzahldivisionen werden Nachkommastelle abgeschnitten (`5`/`2` ergibt `2`)
  - Bei unterschiedlichen Typen (`5.0`/`2`) wird der „kleinere“ implizit „angehoben“ (`5.0`/`2.0`)
  - `byte` oder `short` werden dabei immer mindestens zu `int` („promoted“/widening, JLS17 5.6)
- Wenn verschachtelte Schleifen voneinander abhängen, hilft Gauß ($\sum_{i=1}^{N} i = \frac{N \cdot (N+1)}{2}$)

Zum Warmwerden

1 Ich bin der Klammeraffen Typ

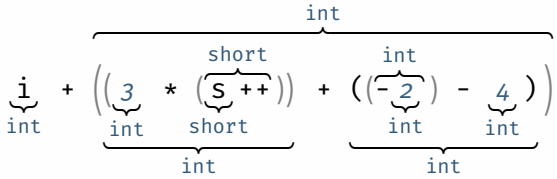
Drücken Sie im Folgenden die Präzedenzregeln *explizit* durch Klammern aus, so dass die unten stehenden Ausdrücke auch ohne Präzedenzregeln in der gleichen Reihenfolge ausgeführt werden. Geben Sie weiter den Typ des gesamten Ausdrucks an (`int i`, `short s` und `float f`). Werten Sie die Ausdrücke *nicht* aus.

1. `i + 3 * s++ + (-2 - 4)`
2. `true && 3 > 4d || 2 - f != 5`

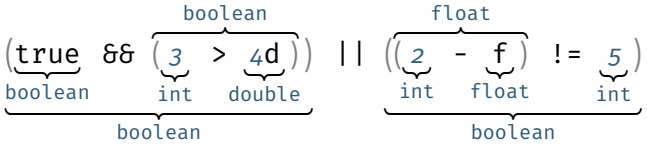
Klammeraffenfreude

Interaktiv kann dies in der JShell ausprobiert werden. Definiert einfach eine Methode wie `void give(Object o) System.out.println(o.getClass());`. Dann gibt z.B. `give(3 + 2d);` den Typ des Ausdrucks aus.

1. Wir klammern und erhalten den Typ `int`:



2. Wir klammern erneut und erhalten den Typ `boolean`:



My volition shrinks from the painful task of recalling my humiliation; yet, like a second Prometheus, I will endure this and worse, if by any means I may arouse in the interiors of Plane and Solid Humanity a spirit of rebellion against the Conceit which would limit our Dimensions to Two or Three or any number short of Infinity.
— Edwin A. Abbott [Abb87]

Präsenzaufgabe

Präsenzaufgabe

2

Von doppelten Methoden

Mehrdimensionale Arrays, also solche Arrays, bei denen die Elemente selbst wiederum Arrays sein können, können wir als Matrizen interpretieren, beispielsweise ist `array[1][3]` das Element der zweiten Zeile in der vierten Spalte. Legen Sie eine Java Datei namens `PositiveEintraege.java` an und implementieren Sie folgende Teilaufgaben innerhalb dieser Datei (oder bearbeiten Sie die Aufgabe auf einem Blatt Papier).

1. Initialisieren Sie ein zweidimensionales Array mit 3×3 Elementen vom Typ `double` mit gültigen Werten in der `main`-Methode.
2. Implementieren Sie eine Methode `public static int anzahlPositive (double[][] matrix)`, die die Anzahl der positiven Eintrage (> 0) bestimmt und zurückgibt. Für ungültige Matrizen soll die Methode `-1.0` zurückgeben.
3. Testen Sie die Implementierung aus b) mit ihrem Array aus a).

An Array a day

- Wir basteln uns eine Klasse und Initialisieren froh umher.

```
public class PositiveEintraege {
   public static void main(String[] args) {
      double[][] mat = {
           { -1.0,  21.0,  3.0 },
           {  1.0,  42.0, -3.0 },
           {  1.0, -84.0,  3.0 }
       };
    }
}
```

mat[0] → {-1.0, 21.0, 3.0}

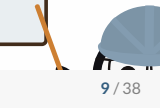
mat[1][2] → -3.0

Hey guy's, i did some methods

*Implementieren Sie eine Methode `anzahlPositive(double[][])`, die die Anzahl der positiven Eintrage (> 0) bestimmt und zurückgibt. Für **ungültige** Matrizen soll die Methode `-1.0` zurückgeben.*

```
public class PositiveEintraege {
   public static void main(String[] args) { ... }
   public static int anzahlPositive(double[][] matrix) {
      int anzahl = 0;
      for(int row = 0; row < matrix.length; row++) {
         for(int col = 0; col < matrix[row].length; col++) {
            if(matrix[row][col] > 0) anzahl = anzahl + 1;
         }
      }
      return anzahl;
   }
}
```

Was heißt „ungültig?“ Das werden wir voerst zurückstellen!



Complex Guardians

- Arrays sind in Java *komplexe* Datentypen (vs. Javas primitiver Datentypen)
- Neben einem „gültigen“ Wert, können diese auch den Wert `null` haben
  - Mit `double[][] matrix = null;` liefert `matrix[1]` eine `NullPointerException`
  - Prüfen können wir dies beispielsweise mit `matrix == null`
- Allgemein sollte nie `null` übergeben oder zurückgegeben werden
- Neben `null`, könnte man auch noch weitere Dinge abprüfen
  - Gibt es überhaupt eine Zeile `matrix.length == 0`?
  - Gibt es überhaupt eine Spalte `matrix[row].length == 0`?
  - Und viele mehr… In jedem Fall sollte man diese mit einem Kommentar absichern
- Mit komplexen Datentypen werden wir später noch viel Freude haben!

Die Suche nach der positiven Anzahl

```java
public static int anzahlPositive(double[][] matrix) {
   if (matrix == null || matrix.length == 0) {
      System.out.println("Matrix ungültig!");
      return -1;
   }

   int anzahl = 0;
   for(int row = 0; row < matrix.length; row++) {
      for(int col = 0; col < matrix[row].length; col++) {
         if (matrix[row][col] > 0)
            anzahl = anzahl + 1;
      }
   }
   return anzahl;
}
```

Testen des Algorithmus

- Wir kehren in die `main`-Methode zurück:

```
public class PositiveEintraege {
    public static void main(String[] args) {
        double[][] mat = {{-1, 21, 3}, {1, 42, -3}, {1, -84, 3}};
        int anzahl = anzahlPositive(mat);
        System.out.println("erwartet: 6, erhalten: " + anzahl);
    }

    public static int anzahlPositive(double[][] matrix) { ... }
}
```

PositiveEintraege.java

Assertions und Tests

- Natürlich können wir einfach mit `if` prüfen ob wir das geforderte Ergebnis erhalten.
- Von Haus aus liefert Java `assert` ⟨`Bedingung`⟩
  - Dieses liefert einen Fehler, wenn die Bedingung nicht erfüllt ist:

    ```
    int i = 0;
    assert i > 0; // → Fehler!
    ```

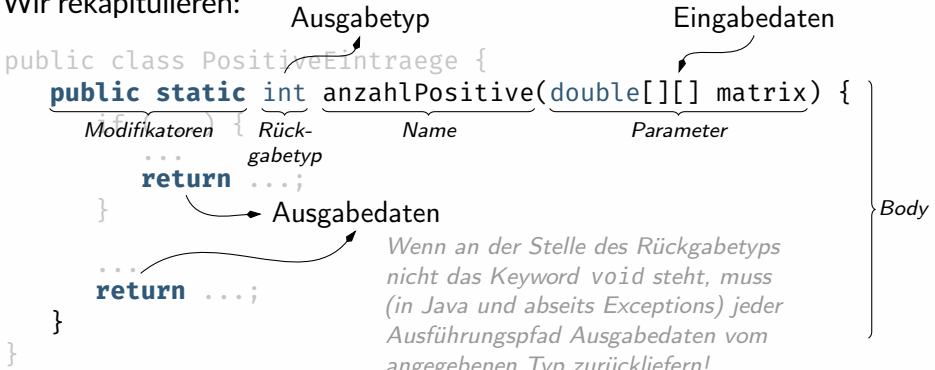
  - Damit Assertions ausgeführt werden, benötigt java `-ea` als Argument: `java -ea Example`
    Das `-ea` steht hier für „enable assertions“
  - Assertions sollten immer nur zusätzlich zu Bedingungen verwendet werden.
- Für „richtige“ Unit-Tests gibt es Framework wie JUnit (das obliegt weiteren Veranstaltungen)
- Allgemein ist Testen ein aufwändiger und wichtiger Teil der Programmierarbeit!

Wait. What was that?

- Wir haben eine eigene Methode erschaffen. Oder gar eine Funktioooon?

psssssst

- Wir rekapitulieren:



Das ist nicht gut, bis ich es nicht fein-Schleif

- Werfen wir noch einmal einen Blick auf die Hauptschleife:

```
for(int row = 0; row < matrix.length; row++) {
    for(int col = 0; col < matrix[row].length; col++) {
        if (matrix[row][col] > 0) ...
    }
}
```

- Über diverse Daten in Java (Arrays, …), können wir „iterieren“
- Dafür bietet java ein wenig *syntactic-sugar* mit „for-each“!

Zucker und weitere tolle Dinge

- „for-each“ ist kein Keyword, sondern eine Alternative **`for`**-Schleife:

```
for(double[] row : matrix) {
    for(double cell : row) {
        if (cell > 0) ...
    }
}
```

- Ein Nachteil? Wir verlieren die Information, das wie-vielte Element das ist.
  Weiter können wir durch `cell` die Matrix nicht mehr verändern. Das kommt später.
- Dafür ist es meist (viel) kompakter!

```
for(int row = 0; row < matrix.length; row++) {
    for(int col = 0; col < matrix[row].length; col++) {
        if (matrix[row][col] > 0) ...
    }
}
```

Übungsblatt 3

Aufgabe 1: Bedingte Anweisungen und Boolesche Ausdrücke

Betrachten Sie den folgenden Java Code-Ausschnitt. Vereinfachen Sie die verschachtelten `if/else` Blöcke so, dass die korrekte Zuweisung der Variablen `z` ohne Kontrollstrukturen (`if`, `else`, `switch`, `while`, `do-while`, `for`, ternärer Operator) in einer Zeile verarbeitet wird, d.h. direkt über eine Zuweisung mithilfe eines Booleschen Ausdrucks. Gehen Sie davon aus, dass die Variablen `x` und `y` vom Typ `int` und `z` vom Typ `boolean` sind und alle Variablen gültige Werte besitzen.

```
if(x >= 0){ • •
    if(y <= 0) { • •
        z = true;
    } else {   •
        if(y == 1) { z = true; }
    }
} else {
    z = false;
}
```

z ist nur *true*, wenn

- `x >= 0 && y <= 0`

Oder:

- `x >= 0 && !(y <= 0) && y == 1`

Kombiniert kommen wir auf:

```
z = (x >= 0 && y <= 0) || (x >= 0 &&↩
     !(y <= 0) && y == 1);
```

Ausdrücke vereinfachen

- Wir hatten einen recht langen Ausdruck:

```
z = (x >= 0 && y <= 0) || (x >= 0 && !(y <= 0) && y == 1);
```

- Zunächst ist `!(y <= 0)` redundant, wenn `y == 1` gilt:

```
z = (x >= 0 && y <= 0) || (x >= 0 && y == 1);
```

- In beiden Fällen muss `x >= 0` gelten:

```
z = (x >= 0) && (y <= 0 || y == 1);
```

- Nun ist `y` ein `int`, zwischen `0` und `1` ist nichts:

```
z = (x >= 0) && (y <= 1);
```

Welche dieser Klammern sind optional?

Ein Mini-Kommentar über die Korrektheit

› Das hatte quasi jeder und die Mehrheit hat Recht!

```
if(x >= 0){
   if(y <= 0) {
      z = true;
   } else {
      if(y == 1) {
         z = true;
      }
   }
} else {
   z = false;
}
```

Sei $x \geq 0$ und $y > 1$. Welchen Wert hat z?

- Na ja, den Alten!

  ```
  z = x >= 0 && (y <= 1 || z);
  ```

- Das Problem? Was, wenn z davor nicht initialisiert wurde?
- Dann ist das ohne Kontrollstrukturen unmöglich
- Korrekt wäre: **die Aufgabe ist** (mit EidI-Wissen) **unmöglich**

Genau genommen kann man für eine spezifische Version und Programmstruktur. . .

We love that. Don’t we.

Aufgabe 2: Schleifen

Erstellen Sie die Java Datei `Schleifen.java` und implementieren Sie die folgenden Teilaufgaben innerhalb der `main` Methode (sog. Programmeinstiegspunkt) dieser Datei.

a) *Lesen Sie über die Kommandozeilenparameter eine Variable vom Typ `int` ein, die wir im Folgenden mit `n` bezeichnen. Implementieren Sie anschließend die folgenden Schleifen, deren Verhalten von `n` abhängen soll.*

b) *Implementieren Sie eine `for`-Schleife, die alle ganzen Zahlen von `n` bis `1` durchläuft und diese ausgibt.* Beispiel: *`n = 5` gibt `5 4 3 2 1` aus.*

c) *Implementieren Sie eine `do-while-Schleife`, die alle ungeraden ganzen Zahlen von `1` bis einschließlich `n` ausgibt.* Beispiel: *`n = 14` gibt `1 3 5 7 9 11 13` aus und `n = 5` gibt `1 3 5` aus.*

Making Loops

- Wir haben dies bereits letzte Woche analysiert, deswegen hier ein wenig schneller.

```java
int n = Integer.parseInt(args[0]);

for (int i = n; i >= 1; i--) {
   System.out.println(i); // oder System.out.print(...)
}

if(n >= 1) {
   int j = 1;
   do {
      if (j % 2 == 1) // ungerade
         System.out.println(j);
      j = j + 1;
   } while (j <= n);
}
```

Schleifen.java

Aufgabe 3: Algorithmen in Java implementieren

In dieser Aufgabe sollen Sie einen vorgegebenen Algorithmus in ein identisches Java Programm umwandeln. Achten Sie insbesondere auf eine passende Auswahl der Datentypen der Variablen und die Art der Schleife. Legen Sie dazu eine Java Datei namens `Quersumme.java` an und implementieren Sie den folgenden Algorithmus in der `main` Methode.

Eingabe: *Ganze Zahl* zahl

```
falls zahl < 0 :
    Gebe „Eingabe ungültig!“ aus
    beende Algorithmus vorzeitig
Setze quersumme = 0
Setze divRest = 0
solange zahl > 0 :
    divRest = zahl % 10
    quersumme = quersumme + divRest
    zahl = ganzzahligAbrunden(zahl / 10)
Gebe quersumme aus
ende
```

- Zunächst das Grundgerüst
- Doch welche Datentypen?
- Hier genügt `int` für alles.
- Abrunden? `int` is 'nuff

```
public static void main(String[] args) {
   int zahl = Integer.parseInt(args[0]);
}
```

Eine Implementation übernehmen

› Abseits der Datentypentscheidung ist die Java-Syntax das größte Hindernis:

```
   int zahl = Integer.parseInt(args[0]);
01: if (zahl < 0) {
02:     System.out.println("Eingabe ungültig!");
03:     return;
   }

04: int quersumme = 0;
05: int divRest = 0;
06: while (zahl > 0) {
07:     divRest = zahl % 10;
08:     quersumme = quersumme + divRest;
09:     zahl = zahl / 10;
   }
10: System.out.println(quersumme);
```

Alternativ auch `System.err.println`

Initialisierung notwendig? Ja, wegen „08“

Initialisierung notwendig? Nein, da „07“ überschreibt

Ganzzahldivision schneidet Nachkommastellen ab

Eingabe: Ganze Zahl zahl
01: **falls** zahl < 0 **:**
02: Gebe „Eingabe ungültig!“ aus
03: **beende Algorithmus vorzeitig**
04: Setze quersumme = 0
05: Setze divRest = 0
06: **solange** zahl > 0 **:**
07: divRest = zahl % 10
08: quersumme = quersumme + divRest
09: zahl = ganzzahligAbrunden(zahl / 10)
10: Gebe quersumme aus
11: **ende**

Quersumme.java

Ein paar Kommentare

- Je nach Pseudocode-„Stil“ und Problem ist die Implementation in Java problematisch.
  - Manchmal hat eine unsaubere Übernahme von Algorithmen katastrophale Folgen
    mal was sehr Java-aktuelles: CVE, Blog, Computerphile. Auch wenn hier von C++ portiert wurde.
  - Ein formale Analyse bringt wenig, bei einer problematischen Implementation
- Mit weiteren Konzepte werden wir Java weiter erkunden.
- Funktioniert Javas „%“ wie das mathematische Modulo?
  Nicht ganz, einfach mal mit negativen, sowie Fließkommazahlen spielen (JLS17 15.17.3)

Aufgabe 4: Iterative Wurzelberechnung

In dieser Aufgabe sollen Sie einen Algorithmus der als Berechnungsvorschrift gegeben ist implementieren. Im Gegensatz zu Aufgabe 3 ist der Algorithmus also nicht vollständig entwickelt sondern nur in der einfachsten Form angegeben. Das Heron-Verfahren ist ein numerisches Verfahren zur Approximation der k*-ten (*$k \in \mathbb{N}, k > 1$*) Wurzel einer Zahl* $\sqrt[k]{x} \in \mathbb{R}$, $x > 0$, *d.h. wir suchen* $y = \sqrt[k]{x}$.
Wir beginnen mit dem Startwert

$$y_0 = x$$

und fahren für $n \geq 1$ *nach der folgenden Iterationsvorschrift fort:*

$$y_{n+1} = \frac{(k-1)y_n^k + x}{k \cdot y_n^{k-1}}$$

wobei y_n *die Approximation nach* n *Berechnungsschritten sei. Da einige Wurzelzahlen (z.B.* $\sqrt{2}$*) nur näherungsweise dargestellt werden können, brechen wir ab, wenn* $|y_{n+1} - y_n| < 10^{-8}$ *erfüllt ist.*
Erstellen Sie eine Java Datei namens `Heron.java` *und implementieren Sie dieses Verfahren wie oben beschrieben. Lesen Sie die Werte für* k *und* x *über die Kommandozeilenparameter ein.*

Iterative Berechnung

```
public static void main(String[] args) {
    double x = Double.parseDouble(args[0]);
    int k = Integer.parseInt(args[1]);

    if (x <= 0 || k <= 1) {
        System.err.println("Eingaben ungültig!");
        return;
    }

    double y = x;
    ...
}
```

Eingabe: $k \in \mathbb{N}$, $k > 1$ *und* $x \in \mathbb{R}$, $x > 0$

Mit $y_0 = x$, $y_{n+1} = \frac{(k-1)y_n^k + x}{k \cdot y_n^{k-1}}$

Abbruch wenn $|y_{n+1} - y_n| < 10^{-8}$

Da wir für die Abbruchbedingung ein y_{n+1} *benötigen, eignet sich do-while*

Iterative Berechnung

Eingabe: $k \in \mathbb{N}$, $k > 1$ *und* $x \in \mathbb{R}$, $x > 0$

Mit $y_0 = x$, $y_{n+1} = \frac{(k-1)y_n^k + x}{k \cdot y_n^{k-1}}$

Abbruch wenn $|y_{n+1} - y_n| < 10^{-8}$

Da wir für die Abbruchbedingung ein y_{n+1} *benötigen, eignet sich do-while*

```
public static void main(String[] args) {
    double x; int k; ...
    double y = x;

    double oldy;
    do {
        oldY = y;
        y = ((k-1) * Math.pow(y, k) + x) / (k * Math.pow(y, k-1));
    } while (Math.abs(oldY - y) > 1E-8);

    System.out.println(y);
}
```

Vorbedingungen, Nachbedingungen und Ausgaben

- Jeder (Unter-)Algorithmus wird Vor- und Nachbedingungen haben
  - Große Teile werden bereits vom Typsystem abgedeckt
  - Dennoch lohnt es sich *immer*, über die Bedingungen nachzudenken
  - Oftmals gibt es eine Menge abzuwägen
  - Ein (cooles) dies verinnerlichendes Konzept ist Design by contract [Mey86; Mey92]

- Ausgaben können in Java auch mit `printf/format` formatiert werden
  - Hier gibt es eine eigene Formatter-Syntax
  - Formatierungshilfen natürlich auch in vielen anderen Sprachen (auch in komfortabler)
  - Beispielsweise:

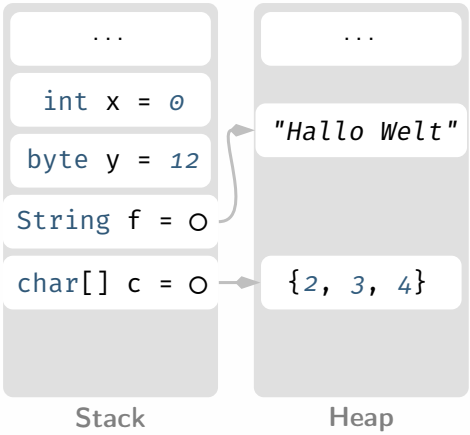
    ```
    System.out.printf("Die %d-te Wurzel von %.3f ist %.3f\n", k, x, y);
    ```

  - Hier besteht auch Unterstützung für Zeiten! (String-Interpolation kann aber mehr)

Aussicht: Übungsblatt 4

Javas' Speicherverwaltung

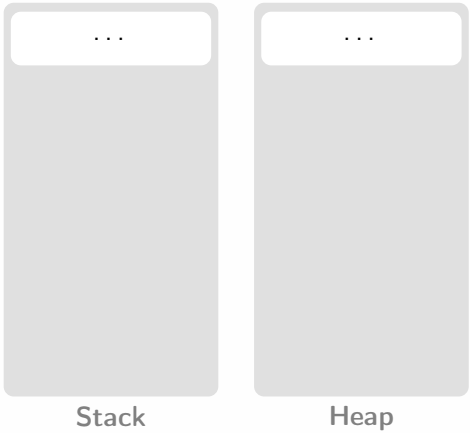
```
int x = 0;
byte y = 12;
String f = "Hallo Welt";
char[] c = {2, 3, 4};
```



Stack-Frames

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```

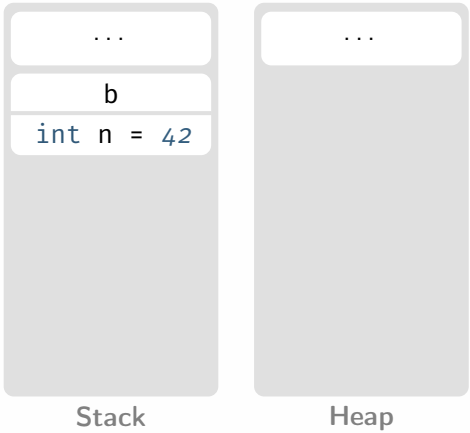


Stack-Frames

Wir werden Rückgabemechanismen sträflich vereinfachen!

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```

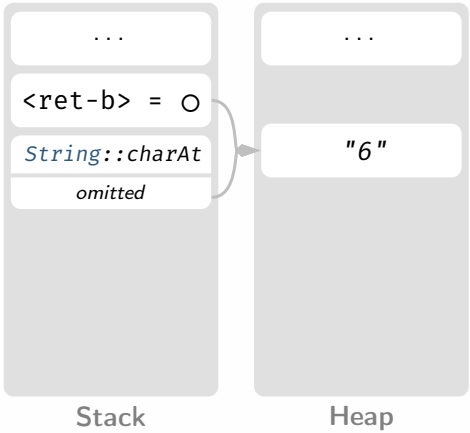


Stack-Frames

Wir werden Rückgabemechanismen sträflich vereinfachen!

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
          ^
```

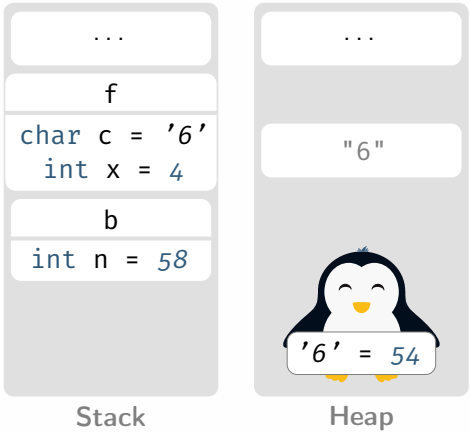


Stack-Frames

Wir werden Rückgabemechanismen sträflich vereinfachen!

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```

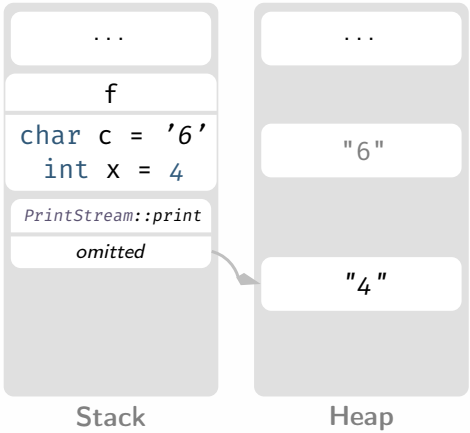


Stack-Frames

Wir werden Rückgabemechanismen sträflich vereinfachen!

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```



Stack-Frames

Wir werden Rückgabemechanismen sträflich vereinfachen!

```
void f(char c) {
  int x = 4;
  System.out.print(b(c+x));
}
String b(int n) {
  return "" + (n % 9);
}

f(b(42).charAt(0));
```

Ein paar Kommentare

- Referenzdatentypen werden wir noch einige Male begegnen
- Die Animationen hier entstammen meiner Präsentation über Rekursion.
- Allgemein gibt es aber ein paar Dinge festzuhalten:
  - Heap und Stack haben erstmal nichts mit den gleichnamigen Datenstrukturen zu tun
  - Dies Speichermodell hilft bei Lebenszyklen, `final`, `==`, …
  - Ich werde öfters auf Heap und Stack zurückgreifen
- Referenzdaten sind `null`, wenn sie auf dem Stack liegen, aber auf kein Element „zeigen“.
- Wir sprechen bei Java von Referenzen, nicht von Zeigern!

Höhere Dimensionen und Methoden

- Arrays von Arrays sind nichts besonderes — wenn man das versteht, sind sie zahm
- Hier hilft ein Blick auf die Präsenzaufgabe
- Wo wir eben noch keine Methoden hatten, kommen nun unzählige
  - Unteralgorithmen sind omnipräsent
  - Wir versuchen alle sinnvoll auslagerbaren Schritte auszulagern
  - Dies verbessert auch die Lesbarkeit (`a + b` vs. `addVisualPadding(a, b)`)
  - Weiter erlaubt es dies, Unteralgorithmen wiederzuverwenden!
- Probiert euch gerne aus und formuliert eure Annahmen!

There is no real ending. It's just the place where you stop the story.
— Frank Herbert online

Abschließendes

Zusammenfassend

[Abb87] Edwin A. Abbott. *Flatland: a romance of many dimensions*. 1987

[Mey86] Bertrand Meyer. *Design by Contract*. 1986

[Mey92] Bertrand Meyer. „Applying ‚design by contract'". 1992

- **Der nächste Donnerstag ist ein Feiertag!**
- Arrays sind komplexe Datentypen, die auf dem Heap verwaltet werden.
  - Damit können sie `null` sein (keine Referenz auf den Heap)
  - Übergeben wir ein Array einer Methode, wird die Referenz (Stack) kopiert (→ selbes Array)
- Es gibt viele kleine Tricks und Kniffe, die man primär durch (selbst) ausprobieren lernt
- Eure Lösungen und Ansätze werden sich wahrscheinlich immer mehr unterscheiden
  - Je mehr Mühe ihr euch gebt, desto ausführlicher das individuelle Feedback
  - Formuliert gerne auch Fragen/Ideen/… und kommentiert fleißig