6

Sichtbar ungültige Seiteneffekte!

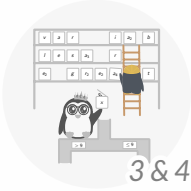
Tutorium sechs

Theorie

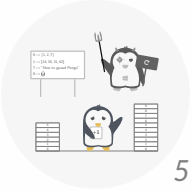
2

13

Grundlagen



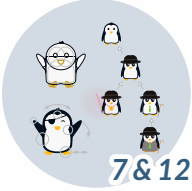
3 & 4



5

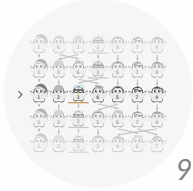
6

Vertiefungen

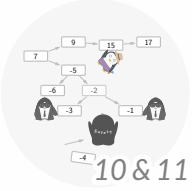


7 & 12

8



9



10 & 11

Algorithmen

- Totale Korrektheit
  - *Terminiertheit:* Endliche Schritte für jede Eingabe
  - *Partielle Korrektheit:* Wenn terminiert, dann korrekt
- Weitere Eigenschaften
  - *Determiniertheit:* Gleiche Eingabe → Gleiche Ausgabe
  - *Determinismus:* Gleiche Eingabe → Gleiche Zustandsfolge

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

Konstrukte

- *Implizit*: byte → short → int → long → float → double
  Zahlen von klein zu groß, sowie: char → int

- *Präzedenzregeln*:
  Post vor Prä, sonst wie Arithmetik & Logik

- *Default-Werte*: Zahlen und Zeichen `0`, Boolean `false`, Rest **`null`**
  Nur bei: Arrays, Instanz- und Klassenvariablen (JLS17 4.12.5)

- *Überschatten*:
  Lokaler Bezeichner überdeckt Gültigkeit des globalen

Arrays & Iteration

- Arrays sind komplexe Datentypen
- Mehrdimensionale Arrays sind eindimensionale Arrays von eindimensionalen Arrays von…
- Die drei Schleifenarten sind gleich mächtig
  - Maximum bekannt: `for`
  - Mindestens ein mal: `do-while`
  - Sonst: `while`

Unterprogramme

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

- Beim Aufruf macht Java call-by-value:
  - Alle Parameter werden kopiert (Stack)

- `void` gibt als Keyword an, dass die Methode keinen Rückgabetyp hat

Objektorientierte Progammierung

- Eine Klasse definiert die Blaupause für Objekte
  - Attribute definieren den Zustand
  - Methoden definieren den Verhalten
  - Statische Elemente sind nicht Teil der Blaupause (sie gehören der Klasse!)
- Der Konstruktor baut den initialen Zustand
  - *Instanziierung*: Erzeugen eines neuen Objektes
  - Wenn keiner: erzeugt Java den leeren Standardkonstruktor
  - `this` erlaubt Aufruf von überladenen Konstruktoren
- Klassen, Methoden, …: *Sichtbarkeit* (`public`, …)
- *Gültigkeit*sbereich: Wo die Variablen „deklariert sind“ (Überschatten, …)

Ich glaub es bedarf weiterer Animationen.

— Break-Down-Flo

Präsenzaufgabe

Präsenzaufgabe

1 Meine Kleine, du hast Palindrom!

In dieser Aufgabe sollen Sie eine rekursive Methode implementieren, die überprüfen soll, ob es sich bei dem als Parameter übergebenen String um ein Palindrom handelt. Ein Palindrom ist ein Wort, das vorwärts und rückwärts gelesen identisch ist. Die Überprüfung soll nicht case-sensitive sein, d.h. das Wort „Kajak“ soll zum Beispiel ein gültiges Palindrom sein.

Beantworten Sie zudem folgende Fragen bezüglich Ihrer Implementierung:

a) Ist Ihre Implementierung ein linear-rekursiver Algorithmus? Warum?

b) Ist Ihre Implementierung ein end- oder kopf-rekursiver Algorithmus? Warum?

Eine erste Idee

In dieser Aufgabe sollen Sie eine rekursive Methode implementieren, die überprüfen soll, ob es sich bei dem als Parameter übergebenen String um ein Palindrom handelt. Ein Palindrom ist ein Wort, das vorwärts und rückwärts gelesen identisch ist. Die Überprüfung soll nicht case-sensitive sein, d.h. das Wort „Kajak“ soll zum Beispiel ein gültiges Palindrom sein.

Idee:

- Prüfe für i = 0 bis i = $\lfloor \texttt{s.length}/2 \rfloor$, ob „`s[i] == s[s.length - i - 1]`“.
- `String::toLowerCase()` entweder für jeden Vergleich oder einmal per Hilfsmethode.
- Noch einfacher: Anstelle `i` zu inkrementieren, löschen wir das erste und letzte Zeichen nach dem Vergleich (via `String::substring(int, int)`—das Ende ist exklusiv)!

Von Palindromen

Palindrome.java

```
public static boolean isPalindrome(String s) {  die „Hilfsmethode“
   return isPalindromeRecursive(s.toLowerCase());
}


private static boolean isPalindromeRecursive(String s) {
   if(s.length() < 2)  Basisfall: weniger als zwei Zeichen
      return true;
   else if(s.charAt(0) != s.charAt(s.length() - 1))  Basisfall: kein Palindrom!
      return false;
   else  Rekursionsfall
      return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```

Sie Simulantario Sie!

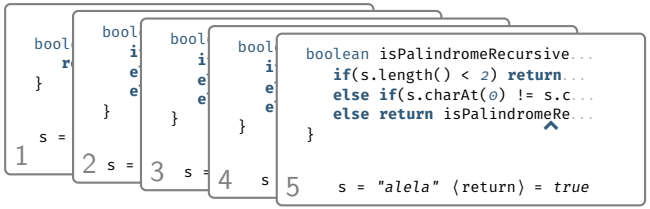
```
private static boolean isPalindrome(String s) {
   return isPalindromeRecursive(s.toLowerCase());
}
private static boolean isPalindromeRecursive(String s) {
   if(s.length() < 2) return true;
   else if(s.charAt(0) != s.charAt(s.length() - 1)) return false;
   else return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```

„RegaleIaGEr“

(Wer sieht auch ein e, dass die Arme hochwirft?)

Sie Simulantario Sie!

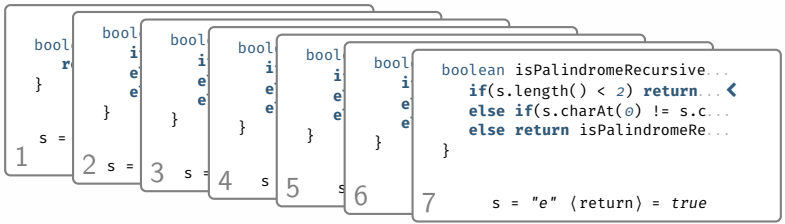
```
private static boolean isPalindrome(String s) { s="RegalelaGEr"
   return isPalindromeRecursive(s.toLowerCase()); "regalelager"
}
private static boolean isPalindromeRecursive(String s) { s="alela"
   if(s.length() < 2) return true;
 › else if(s.charAt(0) != s.charAt(s.length() - 1)) return false; 'a'!='a'
   else return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```



Sie Simulantario Sie!

```
private static boolean isPalindrome(String s) {
   return isPalindromeRecursive(s.toLowerCase());
}
private static boolean isPalindromeRecursive(String s) {
   if(s.length() < 2) return true;
   else if(s.charAt(0) != s.charAt(s.length() - 1)) return false;
   else return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```

s="RegalelaGEr"
"regalelager"
s="e"



Sie Simulantario Sie!

```
private static boolean isPalindrome(String s) { s="RegalelaGEr"
   return isPalindromeRecursive(s.toLowerCase()); < true
}
private static boolean isPalindromeRecursive(String s) {
   if(s.length() < 2) return true;
   else if(s.charAt(0) != s.charAt(s.length() - 1)) return false;
   else return isPalindromeRecursive(s.substring(1, s.length() - 1));
}
```

```
boolean isPalindrome(String s) {
   return isPalindromeRecurs...
}
                    ^

s = "RegalelaGEr" ⟨return⟩ = true
1
```

Mit ⟨return⟩ haben wir hier die zurückzugebenden anonymen Variablen referenziert. Generell ist hier das „Speichern“ der Position für den Aufstieg der Rekursion informal dargestellt.

Iterativer Ansatz

- Diese Aufgabe lässt sich auch iterativ prüfen.
- Für das Palindrom schauen wir für jedes Zeichen i der „linken Hälfte“ ob es mit dem gespiegelten `length` – i – 1 der „rechten Hälfte“ übereinstimmt:

```
public static boolean isPalindromeIterative(String s) {
   s = s.toLowerCase();
   for(int i = 0; i < s.length() / 2; i++) {
      if(s.charAt(i) != s.charAt(s.length() - i - 1))
         return false;
   }
   return true;
}
```

Iterativer Ansatz, II

› Dies können wir als Tail-Rekursion umschreiben:

PalindromeIterative.java

```java
public static boolean isPalindrome(String s) {
   return helper(s.toLowerCase(), 0);
}

private static boolean helper(String s, int i) {
   if (i >= s.length() / 2)
      return true;
   if (s.charAt(i) != s.charAt(s.length() - i - 1))
      return false;
   return helper(s, i + 1);
}
```

```java
s = s.toLowerCase();
for(int i = 0; i < s.length() / 2; i++) {
   if(s.charAt(i) != s.charAt(s.length() - i - 1))
      return false;
}
return true;
```

Ja was isses nun?

a) *Ist Ihre Implementierung ein linear-rekursiver Algorithmus? Warum?*

Unser Verfahren ruft sich *höchstens ein mal selbst auf*, damit ist er linear-rekursiv!

b) *Ist Ihre Implementierung ein end- oder kopf-rekursiver Algorithmus? Warum?*

Der Algorithmus ist End-Rekursiv, da die letzte im Rekursionsfall ausgeführte Operation die Rekursion selbst ist. Damit geschieht im Aufstieg nichts mehr.

Übungsblatt 6

Aufgabe 1: Klassenentwurf in Java

- *Erstellen Sie eine Klasse Kreis. Diese soll folgende Eigenschaften und Methoden implementieren:*
  - *Privaten Instanzvariablen für den Radius, Flächeninhalt und Umfang.*
  - *einen öffentlichen Konstruktor, der den Radius des Kreises als Parameter übernimmt, die restlichen Eigenschaften daraus ableitet, und die entsprechenden Variablen initialisiert.*
  - *get- und set-Methoden für die Instanzvariablen. Dabei soll es nur für den Radius eine set-Methode geben und Flächeninhalt und Umfang sollen daraus abgeleitet werden. Stellen Sie sicher, dass nur gültige Werte für den Radius ($r \geq 0$) akzeptiert werden.*
  - *Eine Instanzmethode um die Eigenschaften des Kreises auf der Kommandozeile auszugeben*
  - *Eine Instanzmethode die einen Kreisradius als Parameter übernimmt, den entsprechenden Flächeninhalt berechnet, das zugehörige Attribut aktualisiert und dieses zurückgibt.*
  - *Eine Instanzmethode die einen Kreisradius als Parameter übernimmt, den entsprechenden Umfang berechnet, das zugehörige Attribut aktualisiert und dieses zurückgibt.*
- *Erstellen Sie eine weitere Klasse `KreisMain`, die den Programmeinstiegspunkt implementiert. Lesen Sie über die Kommandozeilenparameter einen Radius ein und instanziieren Sie innerhalb der `main` Methode ein Objekt der Klasse `Kreis` mit dem eingelesenen Radius. Lassen Sie sich die Eigenschaften dieser Instanz anzeigen.*

Eine runde Sache!

1. Private Instanzvariablen für Radius, Flächeninhalt und Umfang.

2. öffentlicher Konstruktor, der den Radius des Kreises als Parameter übernimmt, die restlichen Eigenschaften daraus ableitet, und die entsprechenden Variablen initialisiert.

```
public class Kreis {
  private double radius;
  private double flaeche;
  private double umfang;

  public Kreis(double radius) {
     this.radius = radius;
     this.flaeche = Math.PI * radius * radius;
     this.umfang = 2 * Math.PI * radius;
  }
  ...
}
```

Eine runde Sache!

3. get- und set-Methoden für die Instanzvariablen. Dabei soll es nur für den Radius eine set-Methode geben und Flächeninhalt und Umfang sollen daraus abgeleitet werden. Stellen Sie sicher, dass nur gültige Werte für den Radius ($r \geq 0$) akzeptiert werden.

```
public class Kreis {
   private double radius, flaeche, umfang;
   private Kreis(double radius) { ... }

   public double getRadius() {
      return this.radius;
   }
   public double getFlaeche() {
      return this.flaeche;
   }
   public double getUmfang() {
      return this.umfang;
   }
}
```

Eine runde Sache!

3. get- und set-Methoden für die Instanzvariablen. Dabei soll es nur für den Radius eine set-Methode geben und Flächeninhalt und Umfang sollen daraus abgeleitet werden. Stellen Sie sicher, dass nur gültige Werte für den Radius ($r \geq 0$) akzeptiert werden.

```
public class Kreis {
  private double radius, flaeche, umfang;
  ...
  public void setRadius(double r) {
     if (r < 0) return;
     this.radius = r;
     this.flaeche = Math.PI * radius * radius;
     this.umfang = 2 * Math.PI * radius;
  }
  public void setUmfang(double umfang) {
     this.umfang = umfang;
  }
  public void setFlaeche(double flaeche) {
     this.flaeche = flaeche;
  }
}
```

Eine runde Sache!

4. Eine Instanzmethode um die Eigenschaften des Kreises auf der Kommandozeile auszugeben.

5. Eine Instanzmethode die einen Kreisradius als Parameter übernimmt, den entsprechenden Flächeninhalt berechnet, das zugehörige Attribut aktualisiert und dieses zurückgibt.

```
public class Kreis {
  private double radius, flaeche, umfang;
  ...

  public void print() {
    System.out.println("Radius: " + radius + "cm" +
        + "\n Fläche: " + flaeche + "cm^2"
        + "\n Umfang: " + umfang + "cm");
  }

  public double berechneFlaeche(double radius) {
     this.flaeche = Math.PI * radius * radius;
     return this.flaeche;
  }
}
```

Eine runde Sache!

6. Eine Instanzmethode die einen Kreisradius als Parameter übernimmt, den entsprechenden Umfang berechnet, das zugehörige Attribut aktualisiert und dieses zurückgibt.

```
public class Kreis {
   private double radius, flaeche, umfang;
   ...

   public double berechneUmfang(double radius) {
      this.umfang = 2 * Math.PI * radius;
      return this.umfang;
   }
}
```

Eine runde Sache?

```java
public class Kreis {
    private double radius;
    private double flaeche;
    private double umfang;
    public Kreis(double radius) {
        this.radius = radius;
        this.flaeche = Math.PI * radius * radius;
        this.umfang = 2 * Math.PI * radius;
    }
    public double getRadius() {
        return this.radius;
    }
    public double getFlaeche() {
        return this.flaeche;
    }
    public double getUmfang() {
        return this.umfang;
    }
    public void setRadius(double r) {
        if (r < 0) return;
        this.radius = r;
        this.flaeche = Math.PI * radius * radius;
        this.umfang = 2 * Math.PI * radius;
    }
    public void setUmfang(double umfang) {
        this.umfang = umfang;
    }
    public void setFlaeche(double flaeche) {
        this.flaeche = flaeche;
    }
    public void print() {
        System.out.println("Radius: " + radius + "cm" +
            + "\n Fläche: " + flaeche + "cm^2"
            + "\n Umfang: " + umfang + "cm");
    }
    public double berechneFlaeche(double radius) {
        this.flaeche = Math.PI * radius * radius;
        return this.flaeche;
    }
    public double berechneUmfang(double radius) {
        this.umfang = 2 * Math.PI * radius;
        return this.umfang;
    }
}
```

Eine runde Sache?

```java
public class Kreis {
    private double radius;
    private double flaeche;
    private double umfang;
    public Kreis(double radius) {
        this.radius = radius;
        berechneFlaeche(radius);
        berechneUmfang(radius);
    }
    public double getRadius() {
        return this.radius;
    }
    public double getFlaeche() {
        return this.flaeche;
    }
    public double getUmfang() {
        return this.umfang;
    }
    public void setRadius(double r) {
        if (r < 0) return;
        this.radius = r;
        berechneFlaeche(r);
        berechneUmfang(r);
    }
    public void setUmfang(double umfang) {
        this.umfang = umfang;
    }
    public void setFlaeche(double flaeche) {
        this.flaeche = flaeche;
    }
    public void print() {
        System.out.println("Radius: " + radius + "cm" +
            + "\n Fläche: " + flaeche + "cm^2"
            + "\n Umfang: " + umfang + "cm");
    }
    public double berechneFlaeche(double radius) {
        this.flaeche = Math.PI * radius * radius;
        return this.flaeche;
    }
    public double berechneUmfang(double radius) {
        this.umfang = 2 * Math.PI * radius;
        return this.umfang;
    }
}
```

Eine runde Sache?

Kreis.java

```
public class Kreis {
   private double radius;
   private double flaeche;
   private double umfang;
   public Kreis(double radius) {
      setRadius(radius);
   }
   public double getRadius() {
      return this.radius;
   }
   public double getFlaeche() {
      return this.flaeche;
   }
   public double getUmfang() {
      return this.umfang;
   }
   public void setRadius(double r) {
      if (r < 0) return;
      this.radius = r;
      berechneFlaeche(r);
      berechneUmfang(r);
   }
   public void setUmfang(double umfang) {
      this.umfang = umfang;
   }
   public void setFlaeche(double flaeche) {
      this.flaeche = flaeche;
   }
   public void print() {
      System.out.println("Radius: " + radius + "cm" +
         + "\n Fläche: " + flaeche + "cm^2"
         + "\n Umfang: " + umfang + "cm");
   }
   public double berechneFlaeche(double radius) {
      this.flaeche = Math.PI * radius * radius;
      return this.flaeche;
   }
   public double berechneUmfang(double radius) {
      this.umfang = 2 * Math.PI * radius;
      return this.umfang;
   }
}
```

Ja dreht es sich nun?

- So haben wir noch unzählige Unsauberkeiten:
  - Der `Kreis(dobule)` prüft nun auch, ob der Radius mindestens 0 ist (ist das gewollt?)
  - Mit `setUmfang(double)` und `setFlaeche(double)` können wir die abgeleiteten Eigenschaften zerstören
  - Analog mit `berechneFlaeche(double)` und `berechneUmfang(double)`
  - Wir nutzen ein eigenes `print` und nicht Javas `toString`-Mechanismus
- Hier kam dies aber einfach durch die Aufgabe…
- Die lief ja auch recht gut!

Oh halt! Die `KreisMain.java`!

KreisMain.java

```java
public class KreisMain {
  public static void main(String[] args) {
    double radius = Double.parseDouble(args[0]);
    Kreis k = new Kreis(radius);
    k.print();
  }
}
```

Aufgabe 2: Gültigkeitsbereiche von Variablen

a) Betrachten Sie folgende Klassendefinition:

```
public class DemoA {
   static int x = 0;
   int y = 1;
   public static int methode(int z){
      return this.y + z;
   }
}
```

this bezieht sich auf das Objekt auf dem die Methode aufgerufen wird. Statische Methoden werden aber auf der Klasse aufgerufen! Das gilt natürlich auch ohne das this, da es sich dann immernoch auf eine Instanzvariable bezieht.

Handelt es sich hierbei um eine gültige Java Klassendefinition? Begründen Sie Ihre Antwort kurz.

Nein eine statische Methode kann nicht auf Instanzattribute (hier: y) zugreifen.

Und nun Ausgaben!

b) Betrachten Sie folgende Klassendefinition:

```
public class DemoB {
    int x = 0;
    public static void methode(int x){
        System.out.println(x);
    }
    public static void methode2(){
        methode(1);
    }
    public static void main(String[] args){
        methode2();
    }
}
```

Welche Ausgabe erzeugt das Programm? Begründen Sie Ihre Antwort kurz.

Ausgabe: „1\n“. Die `methode2()` ruft `methode(int)` auf, wobei der Parameter `x = 1` die Instanzvariable `x = 0` überschattet.

Um an die Instanzvariable zu kommen, bräuchten wir in `methode(int)` den Ausdruck `this.x`!

Und nochmal Ausgaben!

c) *Betrachten Sie folgende Klassendefinition:*

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

Welche Ausgabe erzeugt das Programm? Begründen Sie Ihre Antwort kurz.

Ausgabe: „0\n1\n2\n3“. Die Klassenvariable erhält den Default-Wert `0`. Der erste Aufruf von `methode()` erzeugt „0\n1\n2\n3“. Da `i` statisch ist, verbleibt `i = 4` als Seiteneffekt. Die for-Schleife iteriert kein weiteres mal.

Aufgabe 3: Seiteneffekte

Person.java

```
class Person {
  private String name;
  private ValuePair v;
  public Person(String name, int x, int y) {
    this.name = name;
    v = new ValuePair(x,y);
  }
  public String getName() {
    return name;
  }
  public ValuePair getValuePair() {
    return v;
  }
}
```

ValuePair.java

```
class ValuePair {
  int x; int y;
  public ValuePair(int x, int y) {
    this.x = x; this.y = y;
  }
}
```

TestPerson.java

```
class TestPerson {
  ▷ display in browser
  public static void main(String[] args) {
    Person p = new Person("Heinz", 11, 22);
    String n1 = p.getName();
    ValuePair xy1 = p.getValuePair();
    System.out.println(n1 + " " + xy1.x + " " + xy1.y);
    n1 = "Hugo";
    String n2 = p.getName();
    xy1.x = 33;
    xy1.y = 44;
    ValuePair xy2 = p.getValuePair();
    System.out.println(n2 + " " + xy2.x + " " + xy2.y);
  }
}
```

Die Animationen finden sich im Animationsfoliensatz!

Und wer-wo-wie-was ist das jetzt?

a) *Beschreiben Sie kurz, warum der Seiteneffekt auftritt.*

Die Methode `Person::getValuePair` gibt nur eine Referenz auf das `ValuePair` v-Attribut zurück. Änderungen an dieser Referenz betreffen das selbe Objekt wie das, auf welches das v zeigt.

Und dafür hat der *Depp* von Autor über sechs Stunden lang nur an der einen Animationsfolie gearbeitet??

Ja!

Wie kriegen wir das weg?

b) *In einigen Fällen sind Seiteneffekte erwünscht oder zumindest toleriert. Hier gehen wir davon aus, dass dies nicht der Fall ist und Sie sollen die Dateien* `Person.java`, `ValuePair.java` *und* `TestPerson.java` *so anpassen, dass der Seiteneffekt aus a) nicht mehr auftritt. Schreiben Sie die Begründung für die vorgenommene Änderung als Kommentar in die Datei. Achten Sie darauf, dass die Datei kompilierbar bleibt und auch zur Laufzeit nicht abstürzt, sowie dass die Funktionalität der Klassen erhalten bleibt.*

Zunächst: es reicht nicht einfach die `main`-Methode zu ändern. So wird das Auftreten des prinzipiell Seiteneffekts ja nicht verhindert sondern nur kaschiert. Wir können aber einfach `Person::getValuePair` eine Kopie zurückgeben lassen:

Person.java

```
class Person {
  ...
  public ValuePair getValuePair() {
    return v;
    return new ValuePair(v.x, v.y);
  }
}
```

Jetzt liefert `Person::getValuePair` eine neue Referenz auf ein neues Objekt zurück. Änderungen lassen das Attribut unberührt.

Und das hat warum geholfen?

Person.java

```
class Person {
  private String name;
  private ValuePair v;
  public Person(String name, int x, int y) {
    this.name = name;
    v = new ValuePair(x,y);
  }
  public String getName() {
    return name;
  }
  public ValuePair getValuePair() {
    return new ValuePair(v.x, v.y);
  }
}
```

ValuePair.java

```
class ValuePair {
  int x; int y;
  public ValuePair(int x, int y) {
    this.x = x; this.y = y;
  }
}
```

TestPerson.java

```
class TestPerson {
    ▷ display in browser
  public static void main(String[] args) {
    Person p = new Person("Heinz", 11, 22);
    String n1 = p.getName();
    ValuePair xy1 = p.getValuePair();
    System.out.println(n1 + " " + xy1.x + " " + xy1.y);
    n1 = "Hugo";
    String n2 = p.getName();
    xy1.x = 33;
    xy1.y = 44;
    ValuePair xy2 = p.getValuePair();
    System.out.println(n2 + " " + xy2.x + " " + xy2.y);
  }
}
```

Die Animationen finden sich im Animationsfoliensatz!

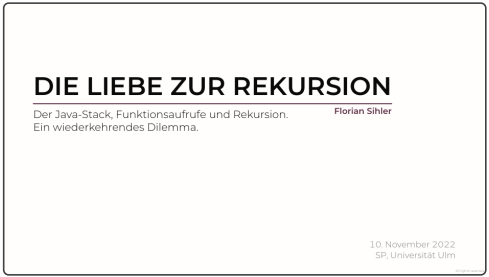
Noch ein paar weitere Worte

- Man könnte auch ein neues Array übergeben
- Auch das modifizieren der `ValuePair`-Attribute mit **`final`** ist denkbar
- Für Kopien bastelt man sich meistens einen Kopier-Konstruktor
  - Dieser erhält ein Objekt des gleichen Typs: `ValuePair(ValuePair other)`
  - Dabei initialisiert er das Objekt mit dem Zustand des übergebenen Objektes
- Theoretisch wäre es auch möglich `getValuePair()` in `getValuePairX()` und `getValuePairY()` zu teilen
  - So geben geben wir nur noch primitive Datentypen nach außen
  - Das kann aber als Veränderung der Funktionalität verstanden werden

Aussicht: Übungsblatt 7

Rekursion!

- Ein erneuter Blick auf die Präsenzaufgabe kann sicherlich hilfreich sein!
- Mathematische Definitionen lassen sich *quasi* 1:1 übersetzen.
- Als Vorgeschmack kann man sich die zugehörige Episode ansehen:



Aber wie das Erhabene von Dämmerung und Nacht, wo sich die Gestalten vereinigen, gar leicht erzeugt wird, so wird es dagegen vom Tage verscheucht, der alles sondert und trennt, und so muss es auch durch jede wachsende Bildung vernichtet werden, wenn es nicht glücklich genug ist, sich zu dem Schönen zu flüchten und sich innig mit ihm zu vereinigen, wodurch denn beide gleich unsterblich und unverwüstlich sind.
— Johann W.v. Göthe [Goe12]

Abschließendes

Zusammenfassend

[Goe12] Johann Wolfgang Goethe. *Aus meinem Leben. Dichtung und Wahrheit*. 1812

- Rekursive Methoden rufen sich selbst (direkt oder indirekt) wieder auf
  - Geschieht dies maximal ein mal, so ist die Rekursion linear
  - Ist der rekursive Aufruf das letzte Statement, haben wir eine Tail-Rekursion
- Variablen haben Gültigkeits- und Sichtbarkeitsbereiche
  - Die Gültigkeit hängt an der Deklaration (Überschattung, …)
  - Die Sichtbarkeit hängt an **`public`**, **`private`**, …
- Seiteneffekte sind potentiell ein großes Problem!

Es ist 4:04 Uhr…