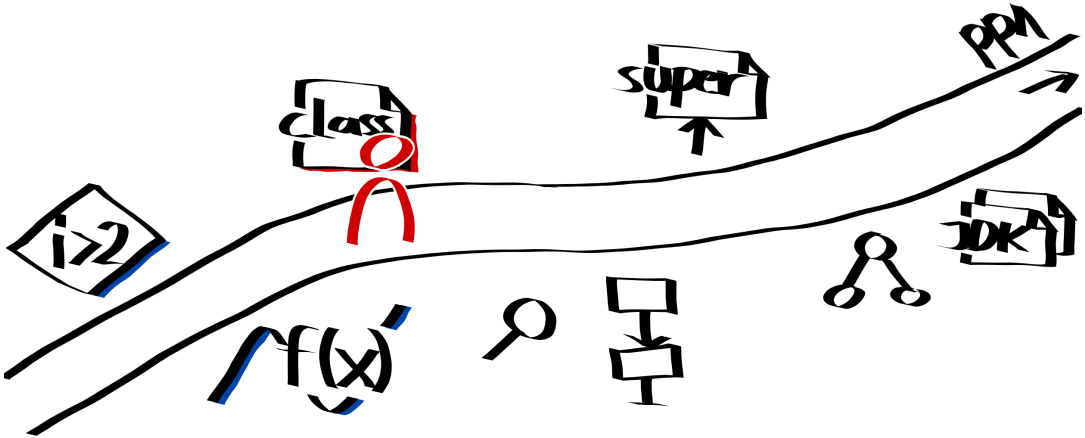
Kapitel 3: Objekte, Speicher & Klassen

VL 11: Schreiben eigener Klassen

Wintersemester 2023/24

Stand 14. November 2023, 15:41 Uhr

Wo stehen wir gerade?



Vorlesungsfeedback

- Zeit: ? (n=)
- Tempo/Schwierigkeit: schwieriger, keine Fragen

Motivation

Heinrich Heine Universität Düsseldorf

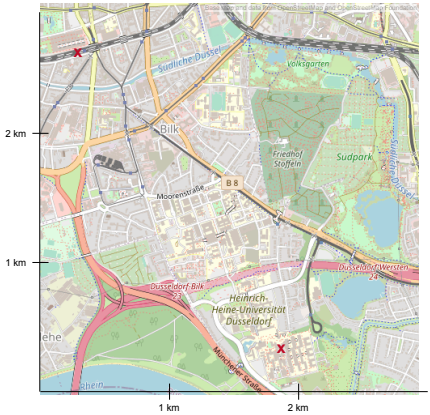
Datentypen bilden Objekte der „echten Welt“ nach

- Farben (`Color`)
- Bilder (`Picture`)

wollen aber mehr:

- Briefe
- Mitarbeiter:innen
- komplexe Zahlen
- Fahrräder
- . . .

⇒ Definition eigener Datentypen



Klasse

Heinrich Heine
Universität Düsseldorf

Definition

Eine **Klasse** definiert einen Datentyp mit Zustand (Variablen) und Verhalten (Methoden). `new` erzeugt mithilfe des *Konstruktors* eine *Instanz der Klasse* (oder *Objekt*). Jede Instanz hat einen eigenen Zustand (Instanz-Variablen).

`String`, `Color` usw. waren auch (fremde) Klassen

In der realen Welt:
Mithilfe einer Fahrrad-Bauanleitung (Klasse) kann ich mehrere Fahrräder (Objekte/Instanzen) bauen.

Bestandteile einer Klasse: Deklaration

Heinrich Heine Universität Düsseldorf

Point.java

```java
1 public class Point {
```

- `public`: Klasse darf von jeder anderen Klasse benutzt werden
- in der Regel jede Klasse in eigene Datei *Klassenname*.java

Bestandteile einer Klasse: Instanz-Variablen

Heinrich Heine Universität Düsseldorf

Point.java

```java
3     private double x;
4     private double y;
```

- speichern „Zustand“ eines Objekts
- Instanzvariablen liegen als Objekt-Eigenschaften im Heap
  → jede Instanz hat eigene Werte für Instanz-Variablen
- Standardwerte (wenn nicht selbst gesetzt): `0`, `false` bzw. `null`

Bestandteile einer Klasse: Konstruktor

Heinrich Heine Universität Düsseldorf

Point.java

```java
6       public Point(double x, double y) {
7           this.x = x;
8           this.y = y;
9       }
```

- selber Name wie Klasse
- kein Rückgabetyp, kein explizites `return`
- definiert, wie Objekt-Eigenschaften initial aussehen
- stellt typischerweise Gültigkeit der Eigenschaften sicher
- wird mit `new` aufgerufen

Default-Konstruktor

Heinrich Heine
Universität Düsseldorf

- falls kein Konstruktor selbst definiert: Default-Konstruktor angelegt
  - keine Argumente
  - Instanzvariablen mit Standardwerten

Bestandteile einer Klasse: Instanz-Methoden

Heinrich Heine Universität Düsseldorf

Point.java

```java
    public void subtract(Point point2) {
        this.x = this.x - point2.x;
        this.y = this.y - point2.y;
    }
```

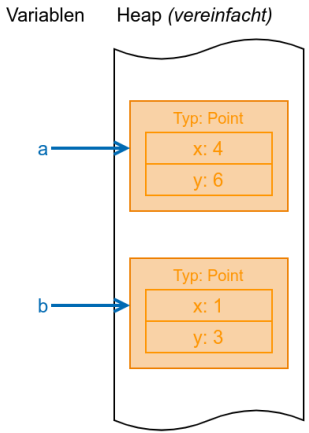
- definieren Verhalten von Objekten
- haben Zugriff auf alle Instanz-Variablen (und -Methoden) derselben Klasse
- impliziter Parameter `this`: bezieht sich auf das Objekt, auf dem die Methode aufgerufen wurde
- fremde Instanzmethoden in Vergangenheit schon benutzt:
  - `System.out.println()`
  - `"hi".toUpperCase()`

`this` = Objekt, auf dem Instanz-Methode aufgerufen

Heinrich Heine Universität Düsseldorf

```
1  Point a = new Point(4, 6);
2  Point b = new Point(1, 3);
3  a.subtract(b);
```

```
1  public class Point {
3      private double x;
4      private double y;
28     public void subtract(Point point2) {
29         this.x = this.x - point2.x;
30         this.y = this.y - point2.y;
31     }
```

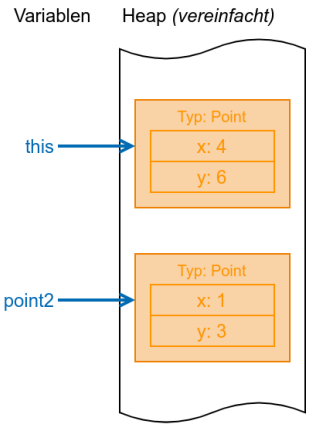


`this` = Objekt, auf dem Instanz-Methode aufgerufen

Heinrich Heine Universität Düsseldorf

```
1  Point a = new Point(4, 6);
2  Point b = new Point(1, 3);
3  a.subtract(b);
```

```
1  public class Point {
3      private double x;
4      private double y;
28     public void subtract(Point point2) {
29         this.x = this.x - point2.x;
30         this.y = this.y - point2.y;
31     }
```



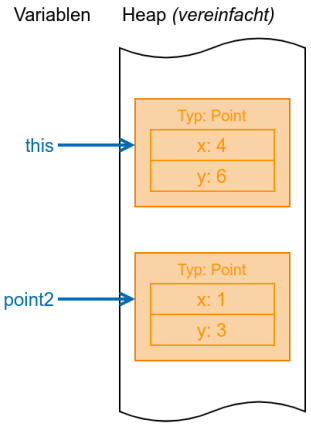
`this` = Objekt, auf dem Instanz-Methode aufgerufen

Heinrich Heine Universität Düsseldorf

```
1  Point a = new Point(4, 6);
2  Point b = new Point(1, 3);
3  a.subtract(b);
```

```
1  public class Point {
3      private double x;
4      private double y;
28     public void subtract(Point point2) {
29         this.x = this.x - point2.x;
30         this.y = this.y - point2.y;
31     }
```

- `this.` darf weggelassen werden, wenn es keine Mehrdeutigkeit gibt



Getter und Setter

Heinrich Heine Universität Düsseldorf

Point.java

```java
    public void setX(double newX) {
        if(newX == Double.POSITIVE_INFINITY || newX == Double.NEGATIVE_INFINITY
          ↪   || Double.isNaN(newX)) {
            System.out.println(newX + " verboten");
            return;
        }
        this.x = newX;
    }

    public double getX() {
        return this.x;
    }
```

- gesteuerter Zugriff auf Instanz-Methoden
- klar definierte Schnittstelle ($\rightarrow$ API)
- Schutz vor versehentlicher Veränderung durch fremden Code

$\Rightarrow$ Information Hiding Principle ($\rightarrow$ Programmierpraktikum)

private vs. public

Heinrich Heine Universität Düsseldorf

private

Was `private` ist, kann nur innerhalb der Klasse selbst gesehen und verändert[1] werden. (minimale Sichtbarkeit)

public

Was `public` ist, kann von jeder Klasse aus gesehen und verändert[1] werden. (maximale Sichtbarkeit)

Faustregel: Was nicht public sein muss, wird private gemacht (minimale Sichtbarkeit).

[1] wenn nicht final, siehe gleich

equals-Methode

- zur Erinnerung: `==` vergleicht bei Objekt-Typen „nur“ Referenzen
- `equals` vergleicht standardmäßig auch Referenzen
- Klasse kann `equals` anders (sinnvoll) definieren

Point.java

```java
    public boolean equals(Point that) {
        // Danger: Rundungsfehler bei Doubles beachten!
        return this.x == that.x && this.y == that.y;
    }
```

toString-Methode

- gibt String-Repräsentation des Objekts zurück
- `System.out.println()` ruft bei Objekten `toString` auf
- Standardverhalten von `toString` JRE-abhängig
- Klasse kann `toString` anders (sinnvoll) definieren

Point.java

```java
33      public String toString() {
34          return x + " " + y;
35      }
```

Klassen-Variablen[2]

Heinrich Heine Universität Düsseldorf

Product.java

```java
3     private static final double TAX_RATE = .19;
```

- `static` → existiert einmal pro Klasse (nicht pro Objekt)
  - ⇒ liegt nur einmal im Heap
- typischerweise für unveränderliche (`final`) Werte
  - ! final ist nur der Wert in der Variablen, im Fall von finalen Variablen mit Objekttyp aber nicht der Heap-Inhalt
- statische Variablen anderer Klassen früher schon verwendet:
  - `Math.PI`
  - `System.out`
  - `System.in`

[2] statische Variablen

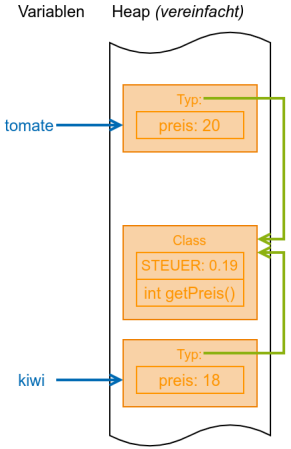
Statische Variablen im Heap

```java
public class Produkt {
    private int preis;
    private static final double STEUER = 0.19;

    public Produkt(int preisInCent) {
        this.preis = preisInCent;
    }

    public int getPreis() {
        return (int)(preis * (1 + STEUER));
    }

    public static void main(String[] args) {
        Produkt tomate = new Produkt(20);
        Produkt kiwi = new Produkt(18);
    }
}
```



Warnung

Heinrich Heine Universität Düsseldorf

statische Variablen nicht als „globale“ Variablen verwenden!

- Code mit globalen Variablen schwieriger verständlich, da Variable von potentiell allen Codezeilen veränderbar
- Code mit statischen Variablen schwieriger zu testen $\rightarrow$Propra
  - auch für automatische Tests der Übungsblätter
  - nur **finale**, statische Variablen in Übungsabgaben verwenden
- besser: Informationen mithilfe von Rückgabewerten und Parametern durchreichen

Klassen-Methoden[3]

Heinrich Heine Universität Düsseldorf

Product.java

```java
    public static double getTaxRate() {
        return TAX_RATE;
    }
```

- `static` → können ohne Objektreferenz direkt über Klassenname aufgerufen werden
- keinen impliziten Parameter `this`
- statische Methoden anderer Klassen früher schon verwendet:
  - `Math.sqrt`
  - `Integer.parseInt`
  - unsere bisher selbst geschriebenen Methoden

[3] statische Methoden, class methods, static methods

Instanz-Methoden vs. Klassen-Methoden

Heinrich Heine Universität Düsseldorf

Instanz-Methoden ...

- haben Zugriff auf:
  - alle Klassen-Variablen derselben Klasse
  - alle Klassen-Methoden derselben Klasse
  - alle Instanz-Variablen derselben Klasse
  - alle Instanz-Methoden derselben Klasse
  - `this`
- sind aufrufbar auf:
  - Instanz der Klasse

Klassen-Methoden ...

- haben Zugriff auf:
  - alle Klassen-Variablen derselben Klasse
  - alle Klassen-Methoden derselben Klasse
- sind aufrufbar auf:
  - Klasse
  - Instanz der Klasse (untypisch)

Hello-World revisited: Klasse

Heinrich Heine Universität Düsseldorf

```
public class HelloWorld
```

- `public`: von überall (auch außerhalb der Klasse) sichtbar
- `class`: definiert eine Klasse
- `HelloWorld`: Klassenname (muss mit Dateinamen übereinstimmen)

Hello-World revisited: main-Methode

Heinrich Heine Universität Düsseldorf

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

- `public`: von überall (auch außerhalb der Klasse) aufrufbar
- `static`: ohne Objekt der Klasse aufrufbar
- `void`: kein Rückgabewert
- `main`: Name der Methode (Einstiegsmethode für java)
- `String[] args`: ein Parameter
  - `String[]`: Typ: String-Array
  - `args`: (beliebiger) Name

Hello-World revisited: Ausgabe

```
System.out.println("Hello World");
```

- `System`: Klasse (in Java eingebaut)
- `out`: statische Variable in `System`
- `println`: Methode auf dem Objekt `out`
- `"Hello World"`: String-Parameter für `println`

Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche . . .

- eigene Klassen mit Instanz-Methoden und -Variablen **definieren**.
- Getter und Setter für Instanz-Variablen **schreiben**.
- die String-Repräsentation eines Objekts mit toString **definieren**
- Konstanten als Klassen-Variablen **definieren**.

Wichtige Begriffe

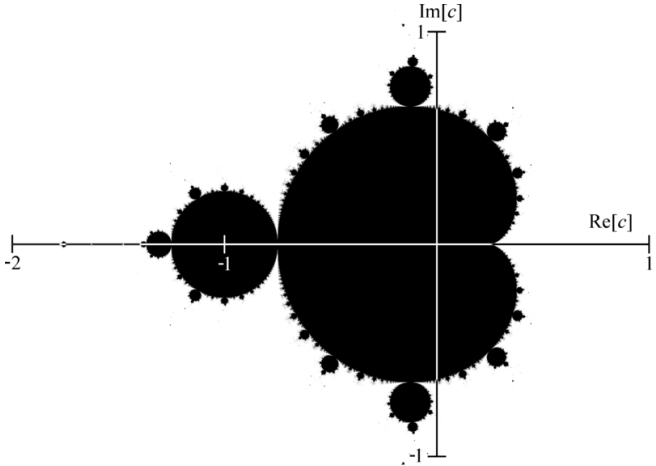
Heinrich Heine Universität Düsseldorf

Klasse
Instanz-Methode
this
private
final

Instanz-Variable
this
Getter
static
Klassen-Methode

Default-Konstruktor
Instanz-Methode
Setter
Klassen-Variable

Mandelbrotmenge



Konstruktion der Menge

In der Mandelbrot-Menge sind alle komplexen Zahlen $c = x + yi$ enthalten (= schwarz markiert), für die die Folge $z_{k+1} = z_k^2 + c$ mit $z_0 = 0$ immer $|z_n| \leq 2$ bleibt.

Rechenregeln für komplexe Zahlen $c_1 = x_1 + y_1 i$ und $c_2 = x_2 + y_2 i$:

- $c_1 + c_2 = (x_1 + x_2) + (y_1 + y_2)i$
- $c_1 \cdot c_2 = (x_1 x_2 - y_1 y_2) + (x_1 y_2 + y_1 x_2)i$
- $|c_1| = \sqrt{x^2 + y^2}$