Informatik: Konstruktoren und statische Elemente

Ziele

- Wissen wann Konstruktoren eingesetzt werden sollen
- Default- und Custom-Konstruktoren definieren und ineinander verketten können
- Statische Datenelemente gezielt einsetzen können
- Statische Konstanten definieren und verwenden können
- Statische Methoden programmieren können und mit diesen auf die Datenelemente richtig zugreifen können
- Verstehen wieso über statische Methoden nicht auf Membervariablen zugegriffen werden kann
- Verstehen was die VM beim Starten des Programms mit der Klasse macht, welche die `main`-Methode beinhaltet
- Einfache grafische Benutzeroberflächen mit Swing programmieren können

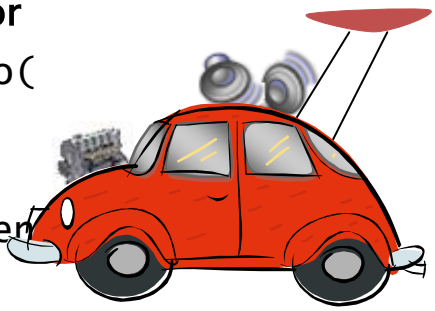
Konstruktoren

Standardkonstruktor

```
Auto a = new Auto();
```

Individualkonstruktor

```
Auto a = new Auto(
   "Rot", 500,
   "Spoiler
   getönte Scheiben
   HiFi");
```



Jedes Objekt muss vom *Konstruktor* vor dem ersten Gebrauch in einen sinnvollen *Startzustand* gebracht werden

```
public class Auto
{
    // Standardkonstruktor oder Default-Konstruktor
    public Auto() {
        ▪ Fülle Benzin, Öl und Wasser nach
        ▪ Kontrolliere Batterie
        …
    }

    // Individualkonstruktor oder Custom-Konstruktor
    public Auto(String farbe, int ps, String optionals) {
        this();
        ▪ Setze Farbe
        ▪ Motorleistung soll sein 500 PS
        ▪ Montiere Optionals
        …
    }
}
```

- Konstruktor hat denselben *Namen* wie die Klasse
- Konstruktordefinition beginnt ohne **`void`** und *Rückgabetyp*
- Wird vom Programmierer der Klasse kein Konstruktor definiert, wird vom Compiler *Default-Konstruktor* mit leerem Rumpf eingefügt

  ABER: Definiert Programmierer eigenen Konstruktor, wird vom Compiler *kein* Default-Konstruktor mehr eingefügt
- **`this`**() ruft den parameterlosen Konstruktor der Klasse auf

Beispiel Bruch

Bruch.java

```
public class Bruch
{
  public int zaehler = 0;
  public int nenner = 0;

  public Bruch() {
    this(0, 1);
  }

  public Bruch(int zaehler) {
    this(zaehler, 1);
  }

  public Bruch(int zaehler, int nenner) {
    if (nenner == 0)
      nenner = 1;
    this.zaehler = zaehler;
    this.nenner = nenner;
  }

  public Bruch(Bruch b) {
    this(b.zaehler, b.nenner);
  }
}
```

Default-Konstruktor

Custom-Konstruktor

Custom-Konstruktor

Kopierkonstruktor

- Ein *verketteter Konstruktoraufruf* mit **`this`** muss als erste Anweisung im Konstruktorrumpf stehen

```
Bruch b1 = new Bruch();
Bruch b2 = new Bruch(3);
Bruch b3 = new Bruch(1, 2);
Bruch b4 = new Bruch(b3);
```

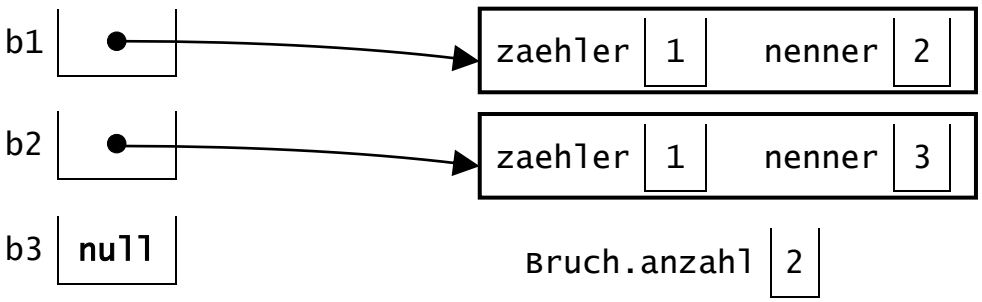
Statische Datenelemente oder Klassenvariablen

Bruch.java

```
public class Bruch
{
  public static int anzahl = 0;
  public int zaehler = 0;
  public int nenner = 0;

  public Bruch() {
    this(0, 1);
  }

  public Bruch(int zaehler) {
    this(zaehler, 1);
  }

  public Bruch(int zaehler, int nenner) {
    anzahl = anzahl + 1;
    if (nenner == 0)
      nenner = 1;
    this.zaehler = zaehler;
    this.nenner = nenner;
  }

  public Bruch(Bruch b) {
    this(b.zaehler, b.nenner);
  }
}
```

- anzahl ist *Klassenvariable*
- gehört zur Klasse
- existiert bereits bevor Objekt angelegt wird
- existiert unabhängig von einem Objekt
- existiert nur einmal
- Zugriff über Klassenname oder Objektvariable die auch null sein kann

```
System.out.println(Bruch.anzahl);       // ergibt 0
Bruch b1 = new Bruch(1, 2);
System.out.println(b1.anzahl);          // ergibt 1
Bruch b2 = new Bruch(b1);
b2.nenner = 3;
System.out.println(b2.anzahl);          // ergibt 2
Bruch b3 = null;
System.out.println(b3.anzahl);          // ergibt 2
```



Statische Konstanten

```
public class Math {
   public static final double PI = 3.141592653589793;
   …
}
```

Beispiel Seriennummern für Objekte

Auto.java

```
public class Auto
{
  private static int naechsteSeriennummer = 0;
  private final int seriennummer;
  private int ps = 0;

  public Auto(int ps) {
    this.seriennummer = naechsteSeriennummer;
    this.ps = ps;
    naechsteSeriennummer = naechsteSeriennummer + 1;
  }

  public static int getNaechsteSeriennummer() {
    return naechsteSeriennummer;
  }
  public int getSeriennummer() {
    return this.seriennummer;
  }
}
```

ACHTUNG: Memberkonstante seriennummer wird esrt im Konstruktor initialisiert

```
System.out.println(Auto.getNaechsteSeriennummer());  // ergibt 0
Auto a1 = new Auto(500);
System.out.println(a1.getSeriennummer());            // ergibt 0
System.out.println(Auto.getNaechsteSeriennummer());  // ergibt 1
Auto a2 = new Auto(200);
System.out.println(a2.getSeriennummer());            // ergibt 1
System.out.println(Auto.getNaechsteSeriennummer());  // ergibt 2

Auto a3 = null;
System.out.println(a3.getSeriennummer());            // Fehler (1)
System.out.println(Auto.getSeriennummer());          // Fehler (2)
```

Statische Methoden oder Klassenmethoden

- wie statische Datenelemente mit **`static`** deklariert
- sind unabhängig von Objekten und gehören der ganzen Klasse
- können schon über *Klasse* aufgerufen werden
- können aber auch über Objektvariablen gestartet werden

Zum Unterschied dazu können *normale Methoden*

- nur über angelegtes Objekt gestartet werden(1)
- nicht über die Klasse gestartet werden(2)

Beispiel Klasse String

```
public final class String
{
   public static String valueOf(double d) { … }
   public static String valueOf(int i) { … }
   …
   public char charAt(int i) { … }
   public String compareTo(String s) { … }
   public boolean equals(Strings s) { … }
   public int indexOf(char c) { … }
   public int length() { … }
   public String substring(int b) { … }
   public String substring(int b, int e) { … }
   public String toString() { … }
   public String trim() { … }
   …
}
```

Klassen-methoden

Methoden

```
String s = String.valueOf(-315);                    // ergibt "-315"
System.out.println(s.substring(1) + s.charAt(0));   // ergibt "315-"
```

Klassenmethoden

Aufruf über die Klasse `String` oder die Objektvariable s

Normale Methoden

Aufruf über die Objektvariable s

Achtung

Autoverwaltung.java

```
public class Autoverwaltung
{
  private Auto[] a = null;

  public static void main(String[] args) {
    System.out.println("Autoverwaltung");
    System.out.println("==============");
    a = new Auto[10];
```

Cannot make a static reference to the non-static field a

In einer *Klassenmethode* kann nicht auf eine *Membervariable* zugegriffen werden

a muss ebenfalls als *Klassenvariable* definiert werden

Wenn VM Java-Programm startet, wird von der Klasse (`Autoverwaltung`) *kein* Objekt angelegt

Grafische Benutzerschnittstellen (GUI) mit Swing

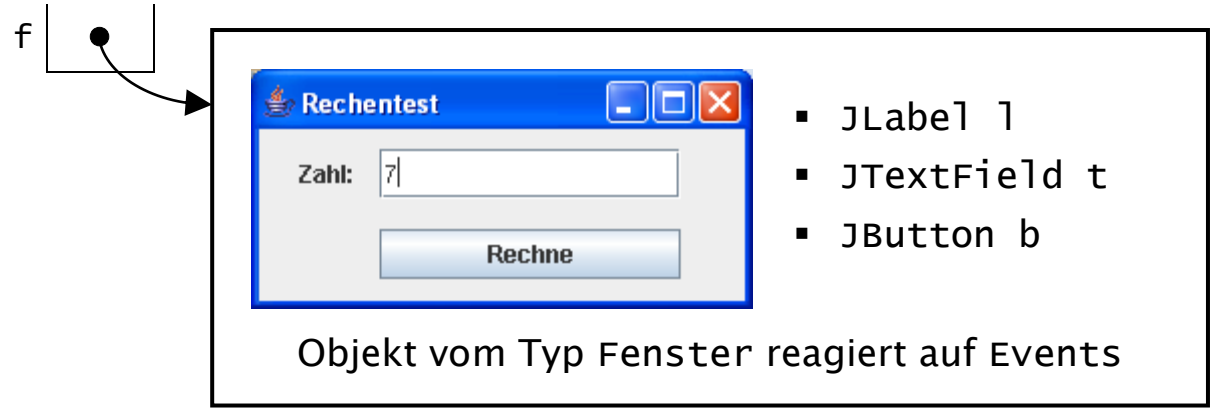
Klasse `FensterHauptprogramm`

Erstellt in der `main`-Methode das Fenster

Klasse `Fenster`

- *Konstruktor* legt Fenster mit seinen *Komponenten* an
- Definiert *Ereignisbehand-lungsmethoden*

```
public class FensterHauptprogramm
{
  public static void main(String[] args) {
    Fenster f = new Fenster();
  }
}
```



Im Konstruktor des Fensters werden

- *Fenstereigenschaften* gesetzt
- Fensterkomponenten (`JLabel`, `JTextField`, `JButton`) *angelegt*
- Fensterkomponenten zum *Fenster hinzugefügt*
- *Ereignisbehandlung* (Knopf gedrückt) programmiert

```
import javax.swing.*; // JFrame, JTextField, JButton, JLabel
import java.awt.*;    // Container
import java.awt.event.*;   // Adapter, Listener, Events
public class Fenster extends JFrame
{
  private JLabel l = null          // Private Komponenten des Fensters;
  private JTextField t = null;
  private JButton b = null;

  public Fenster() {               // Konstruktor des Fensters
    setTitle("Rechentest");        // Fenstereigenschaften
    setBounds(50,60,250,120);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    l = new JLabel("Zahl:");       // Komponenten anlegen
    l.setBounds(20,10,50,25);
    t = new JTextField();
    t.setBounds(60,10,150,25);
    b = new JButton();
    b.setText("Rechne");
    b.setBounds(60,50,150,25);

    Container contentPane =        // Komponenten zum Fenster fügen
      getContentPane();
    contentPane.setLayout(null);
    contentPane.add(l);
    contentPane.add(t);
    contentPane.add(b);

    setVisible(true);              // Fenster sichtbar machen

    b.addActionListener(           // Ereignis: Knopf drücken
      new ActionListener() {

        public void actionPerformed(ActionEvent ev) {
          try {
            double d = Double.parseDouble(t.getText());
            d = Math.sqrt(d);
            t.setText(String.valueOf(d));
          } catch (NumberFormatException e) {
            t.setText("Fehler");
          }
        }

      }
    );

  }                                // Ende des Konstruktors
}
```

Konvertierung von `JTextField`-Inhalten: String ➔ Zahl

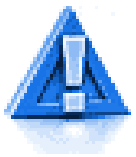
```
try {
   double d = Double.parseDouble("3.14");
   int i = Integer.parseInt("7");
} catch (NumberFormatException e) {
   …
}
```

werfen `NumberFormatException`.

Konvertierung: Zahl ➔ String

```
Strings s = String.valueOf(d);
```

Programmierregeln: Reihenfolge der Klasseninhalte



```
public class Klassenname
{
```

1. öffentliche Konstanten
2. private Klassenvariablen
3. private Membervariablen
4. Konstruktoren
5. Getter- und Settermethoden
6. private und öffentliche Methoden

```
}
```