Informatik: Klassen und Objekte

Ziele

- Klasse erstellen, daraus Objekt anlegen und mit Objekt arbeiten können
- Unterschied zwischen Klasse und Objekte verstehen
- Mit Membervariablen in der Klasse selbst aber auch von anderen Klassen aus arbeiten können
- Membervariablen anhand von Getter- und Settermethoden verbergen können
- Den `this`-Zeiger verstehen und richtig verwenden können
- Private und öffentliche Methoden in Klassen richtig einsetzen können
- Die Bedeutung der Methoden `clone`, `equals`, `compareTo` und `toString` verstehen
- In Arrays Objekte verwalten können

Klassen

Bruch.java

```
public class Bruch
{
  // Datenelemente, Membervariablen
  private int zaehler = 0;
  private int nenner = 1;

  // Methoden
  public String toString() {
    String ret = zaehler + "/" + nenner;
    return ret;
  }
}
```

- definieren neue Typen
- bestehen aus *Variablen* (*Datenelementen*, *Membervariablen*)
- enthalten *Methoden*
- Klassendefinition in eigener Datei

Primitive Datentypen	**`int`**, **`double`**, **`boolean`**, **`char`**, …
Referenztypen	`Bruch`, `String`, Arrays, …

ACHTUNG: Klasse bzw. Typ definiert nur „Bauplan“. Objekt muss aus Klasse erst erstellt (instanziiert) werden, damit man mit Objekt arbeiten kann

Primitive Datentypen	*Referenztypen*
// Deklaration `int i = 0;` i | 0	*// Deklaration* `Bruch b = null;` b | null
// Entfällt	*// Instanziierung* `b = new Bruch();` b → zaehler | 0 nenner | 1 Objekt
// Verwendung `i = 30;` `System.out.println(i);`	*// Verwendung* `b.nenner = 1;` `b.zaehler = 2;` `System.out.println(b.toString());`

- Aus Klassen können beliebig viele Objekte erzeugt werden
- Jedes Objekt erhält bei seiner Instanziierung seine eigenen Datenelemente

Instanziieren und Verwenden von Objekten im Hauptprogramm

Hauptprogramm.java

```
public class Hauptprogramm
{
  public static void main(String[] args) {
    Bruch b1 = new Bruch();
    Bruch b2 = new Bruch();
    b1.nenner = 1;
    b1.zaehler = 2;
    b2.nenner = 2;
    b2.zaehler = 3;
    System.out.println(b1.toString());
    System.out.println(b2.toString());
  }
}
```

- b1 und b2 sind *lokale Variablen* der Methode `main`
- b1 und b2 sind *Instanzen* der Klasse `Bruch`
- `b1.nenner`, `b1.zaehler` sind *Membervariablen* des Objekts b1

Zugriff auf Datenelemente

Hauptprogramm.java

```
public class Hauptprogramm
{
  public static void main(String[]
    Bruch b1 = new Bruch();
    Bruch b2 = new Bruch();
    b1.nenner = 1;
    b1.zaehler = 2;
    b2.nenner = 2;
    b2.zaehler = 3;
    System.out.println(b1.toString(
    System.out.println(b2.toString(
  }
}
```

Bruch.java

```
public class Bruch
{
  // Datenelemente, Membervariablen
  private int zaehler = 0;
  private int nenner = 1;

  // Methoden
  public String toString() {
    String ret = zaehler + "/" + nenner;
    return ret;
  }
}
```

Außerhalb der Klasse:

Über die Objektvariablen
`b1.nenner = 1;`

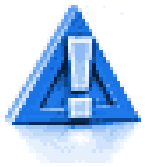
Da b1 lokale Variable ist, ist sie nur innerhalb von `main` gültig

Innerhalb der Klasse:

Direkt
`nenner = nenner + 2;`

Ein Datenelement (`nenner`, `zaehler`) kann in der gesamten Klasse verwendet werden

Geheimhaltungsprinzip und this



- Datenelemente in Klasse müssen von der Umwelt verborgen werden
- Zugriff muss über *Methoden* (**Getter-** und **Settermethoden**) erfolgen

Bruch.java

```
public class Bruch
{
  private int zaehler = 0;
  private int nenner = 1;

  public int getNenner() {
    return nenner;
  }
  public void setNenner(int nenner) {
    if (nenner != 0)
      this.nenner = nenner;
  }
  public int getZaehler() {
    return zaehler;
  }
  public void setZaehler(int zaehler) {
    this.zaehler = zaehler;
  }
```

Hauptprogramm.java

```
public class Hauptprogramm
{
  public static void main(String[] args) {
    Bruch b1 = new Bruch();
    Bruch b2 = b1;
    b1.setNenner(1);
    b1.setZaehler(2);
    System.out.println(b2.toString());
  }
```

- Datenelemente müssen mit **`private`** definiert werden
- Die Benennung der Setter- und Gettermethoden erfolgt nach klar definierten Regeln
- Damit in der Klasse Parameter (nenner) von Membervariable (**`this.`**nenner) unterschieden werden kann, verwendet man **`this`**
- **`this`** zeigt auf das Objekt selbst

Wieso Geheimhaltung?

- Zuweisung unerwünschter Werte an die Membervariablen wird unterbunden
- Einfachere Fehlersuche, weil der Zugriff auf die Membervariable in der Settermethode überwacht werden kann (wann wird mit welchem Wert zugegriffen. Wäre Membervariable öffentlich, so kann von vielen Stellen aus unkontrolliert auf sie zugegriffen werden)
- Eigenschaften können durch Getter- und Settermethoden vorgespiegelt werden
- Getter- und Settermethoden verbergen interne Optimierungen (z. B. das Kürzen eines Bruches wird nur beim Getter aufgerufen)

Öffentliche und private Methoden

Bruch.java

```
public class Bruch
{
  private int zaehler = 0;
  private int nenner = 1;

  public void set(int zaehler, int nenner) {
    this.zaehler = zaehler;
    this.nenner = nenner;
  }

  public void kuerzen() {
    int ggt = ggT(this.zaehler, this.nenner);
    this.zaehler = this.zaehler / ggt;
    this.nenner = this.nenner / ggt;
  }

  public void multiplizieren(Bruch b) {
    this.zaehler = this.zaehler * b.getZaehler();
    this.nenner = this.nenner * b.getNenner();
    kuerzen();
  }

  private int ggT(int m, int n) {
    int ret = 0;
```

Hauptprogramm.java

```
public class Hauptprogramm
{
  public static void main(String[] args) {
    Bruch b1 = new Bruch();
    Bruch b2 = new Bruch();
    b1.set(2, 3);
    b2.set(3, 4);
    System.out.println(b1.toString());
    System.out.println(b2.toString());
    b1.multiplizieren(b2);
    System.out.println(b1.toString());
    System.out.println(b2.toString());
  }
}
```

- Eine durch **`private`** definierte Methode kann nur in der Klasse selbst verwendet werden

Von Klassen häufig bereitgestellte Methoden

`clone`	Dupliziert ein Objekt
`equals`	Vergleicht zwei Objekte ob diese gleich sind
`compareTo`	Vergleicht ebenfalls und liefert zurück, welches Objekt kleiner/größer ist
`toString`	Gibt den Inhalt eines Objektes als String aus

Bruch.java

```
public class Bruch
 {
   private int zaehler = 0;
   private int nenner = 1;

   public boolean equals(Bruch b) {
     boolean ret = this.zaehler * b.getNenner() == this.nenner * b.getZaehler();
     return ret;
   }

   public int compareTo(Bruch b) {
     int ret = this.zaehler * b.getNenner() - this.nenner * b.getZaehler();
     return ret;
   }

   public Bruch clone() {
     Bruch ret = new Bruch();
     ret.set(this.zaehler, this.nenner);
     return ret;
   }
```

```
Bruch b1 = new Bruch();
b1.set(2, 3);
Bruch b2 = b1.clone();
b2.setZaehler(4);
System.out.println(b1.equals(b2));          // ergibt false
System.out.println(b1.compareTo(b2));       // ergibt -6
System.out.println(b1.toString());          // ergibt 2/3
```

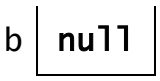
Alle Klassen besitzen bereits Methoden `equals` und `toString`. In Klasse `Bruch` wurden diese neu umdefiniert.

- Standard-`equals` angewendet auf die zwei Brüche `1/1` und `1/1` würde **`false`** ergeben, weil hier nur Zeiger verglichen werden, `Bruch.equals` ergibt **`true`**
- Standard-`toString` angewendet auf den Bruch `1/1` würde den String "`Bruch@923e30`" ergeben, `Bruch.toString()` ergibt "`1/1`"

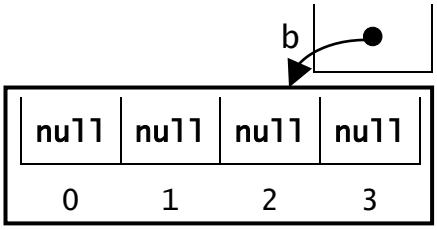
Arrays und Objekte

Es soll ein Array angelegt werden, das mehrere Bruch-Objekte in sich abspeichert

```
// Deklaration
Bruch[] b = null;
```



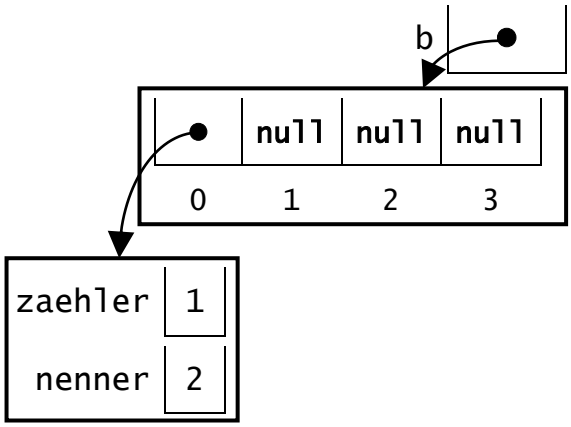

```
// Instanziierung des Arrays
b = new Bruch[4];
```



ACHTUNG: Legt nur das Array-Objekt an, legt aber keine Bruch-Objekte an

```
// Instanziierung des ersten Bruches
b[0] = new Bruch();
b[0].set(1, 2);
```

Legt Bruch-Objekt an und gibt Zeiger des Objektes nach b[0]




```
b[3] = new Bruch();
b[3].set(3, 4);
b[1] = b[0].clone();
b[1].multiplizieren(b[3]);
for (int i = 0; i < b.length; i = i + 1)
   if (b[i] != null)
      System.out.println(b[i].toString());
```