Methoden

(„Funktionen“)

Methoden: Syntax

```
[Modifizierer] Typ Methodenname([Parameter])
 {
  Anweisungen
 }
```

- *`Methodenname`* ist der Name der Methode
- Jede Methode hat einen Rückgabe-*`Typ`*
- gibt eine Methode kein Ergebnis zurück, muß die Methode den Typ `void` haben
- Ein Ergebnis wird mit `return` *`Ausdruck;`* zurückgegeben

Methoden: Syntax

```
[Modifizierer] Typ Methodenname([Parameter])
 {
  Anweisungen
 }
```

- Ein `return` beendet die Methode
- `void`-Methoden können mit `return;` enden
- *`Parameter`*: durch Kommas getrennte Übergabewerte – jeweils durch Datentyp und Name spezifiziert

Methoden: Syntax

```
[Modifizierer] Typ Methodenname([Parameter])
 {
  Anweisungen
 }
```

- Parameter werden beim Methoden-Aufruf kopiert ⇒

  Variable, die an Methoden übergeben werden, bleiben unverändert

- Ein Programm startet mit Aufruf der Methode `main()`
- Erste Anwendungen: Stellen Sie den *`Modifizierer`* `static` voran

Methoden

Beispiel:

numerische Ableitung

```
// Datei: Beispiel5.java

public class Beispiel5 {
        /* Die Funktion, die wir auswerten wollen */
        static double f(double x) {
          return(Math.sin(x));
         }

        static final double deltaX = 1e-6; // kleiner Abstand

        /* 1. Ableitung */
        static double ableitung(double x) {
          return((f(x+deltaX)-f(x-deltaX))/(2*deltaX));
         }

        /* Hauptprogramm */
        public static void main(String[] args) {
         for(double x=0; x<=2*Math.PI; x+=0.1) {
           System.out.println("sin'(" + x + ") = " + ableitung(x) );
           System.out.println(" cos(" + x + ") = " + Math.cos(x) );
          }
    }
}
```

Methoden – fortgesetzt

- Variablen können innerhalb der `class`-Konstruktion auch außerhalb von Methoden definiert werden. Auf solche Variablen können alle nachfolgenden Methoden zugreifen
- `final`-Variable: **Konstante**, deren Wert nach der Zuweisung nicht veränderbar ist
- Lokale Variablen werden innerhalb der Methode deklariert und gelten nur in der Methode (analog: Block)
- Rekursion: Selbst-Aufruf einer Methode
- Überladen: Für verschiedene *`Parameter`*-Listen können mehrere gleichnamige Methoden existieren

Methoden

Beispiel:

Wieviele Möglich-keiten gibt es, daß 10 ganze Zahlen zwischen 0 und 3 die Summe 10 besitzen ?

```
// Datei: Beispiel6.java

public class Beispiel6 {
        static final int zielSumme = 10;   // gesuchte Summe
        static final int Plaetze = 10;     // 10 Plaetze
        static final int Bereich = 3;      // Groesster Zahlenwert
        static int Anzahl = 0;          // Anzahl

        static void test_summe(int summe, int platz) {
          int i;                    // Lokale Variable
          if(summe > zielSumme)
             return;                  // Schneller Ruecksprung
          if(platz == Plaetze)
           if(summe == zielSumme)      // Passt Summe ?
             Anzahl++;
          if(platz < Plaetze)
           for(i=0; i<=Bereich; i++)
             test_summe(summe+i, platz+1); // Rekursion
         }

        /* Hauptprogramm */
        public static void main(String[] args) {
          Anzahl = 0;
          test_summe(0, 0);
          System.out.println(Anzahl + " Moeglichkeiten");
    }
}
```

Objektorientierte Programmierung

Klassen

- Java fasst Attribute (Variablen) und Methoden in Klassen zusammen
- Allgemeiner Aufbau einer Klasse:

```
[Modifizierer] class Klassenname
  [extends Basisklasse]
  [implements Schnittstellen] {
  Attributdeklarationen
  Methodendeklarationen
 }
```

- „Bauanweisung“

Klassen

- Java fasst Attribute (Variablen) und Methoden in Klassen zusammen
- Allgemeiner Aufbau einer Klasse:

```
[Modifizierer] class Klassenname
  [extends Basisklasse]
  [implements Schnittstellen] {
  Attributdeklarationen
  Methodendeklarationen
 }
```

- „Bauanweisung“

Klassen

- Ausprägung einer Klasse: „Objekt“ oder„Instanz“
- *`Attribute`* (Variablen) beschreiben Zustand des Objekts
- Jede Quelldatei
  - ★ sollte nur eine Klasse enthalten
  - ★ muß den Namen *`Klassenname`*`.java` besitzen
  - ★ mehrere Klassen: nur eine darf `public` sein; diese bestimmt den Dateinamen
- Jede Klasse definiert einen eigenen Datentyp *`Klassenname`*

Beispiel-Klasse: Rational

```
// Datei: Rational.java

public class Rational {
    private final int Zaehler;  // Zaehler und
    private final int Nenner;   // Nenner
```

- *`Modifizierer`* `public`: Die Klasse ist von außen sichtbar
- *`Attribute`* der Klasse: Zähler und Nenner
  - ★ `private`: von außen nicht sichtbar
  - ★ `final`: nach Initialisierung nicht veränderbar

Beispiel-Klasse: Rational

```
public static int gcd(long a, long b)
 {
   if(a == 0)            // Sonderbehandlung fuer Nullen
     return(1);
   if(b == 0)
     return(1);

   if(a < 0)             // Betrag bilden
     a=-a;
   if(b < 0)
     b=-b;

   if(a == 1)            // Sonderbehandlung fuer Einsen
     return(1);
   if(b == 1)
     return(1);

   /* Das ist der Euklidische Algorithmus */
   long remainder = a % b;
   while(remainder != 0)
    {
      a=b;
      b=remainder;
      remainder = a % b;
    }           // Nun steht der GGT in b
   return((int) b);
}
```

Größter Gemeinsamer Teiler (GGT):

- ★ existiert unabhängig von Instanz (`static`)
- ★ von außen sichtbar (`public`)

Beispiel-Klasse: Rational

```
public Rational(int z, int n) {
  int g = gcd(z,n); // Erst kuerzen
  if(n <= 0)
    System.err.println("### Achtung: Nenner " + n + " ist nicht positiv");
  if(z == 0)     // Sonderbehandlung fuer 0
    g = n;
  Zaehler = z/g;
  Nenner = n/g;
}
```

- Konstruktor: Eine Methode, deren Name mit dem Klassennamen identisch ist

★ erzeugt Objekt und initialisiert dessen Inhalt (hier: Zähler und Nenner)

Beispiel-Klasse: Rational

```
// Zugriff von ausserhalb auf Zaehler
public int Zaehler()  {
 return Zaehler;
}

// Zugriff von ausserhalb auf Nenner
public int Nenner()  {
 return Nenner;
}
```

- Zugriff von außerhalb auf `private`-Attribute: über `public`-Methoden

Beispiel-Klasse: Rational

```
// Erzeuge ein neues Objekt mit Wert (this + b)
public Rational plus(Rational b)
 {
   int z2 = b.Zaehler;
   int n2 = b.Nenner;
   int gcd = gcd(Nenner, n2);  // Auf den Hauptnenner bringen
   int f1 = n2/gcd;
   int f2 = Nenner/gcd;
   return new Rational(Zaehler*f1+z2*f2, Nenner*f1);
}
```

- `new` erzeugt ein neues Objekt (Instanz) der Klasse
- Objekte werden als Referenz an Methoden übergeben ⇒ Referenz liegt als Kopie vor,

  Attribute des Objekts können aber von der Methode verändert werden

Beispiel-Klasse: Rational

```
// Erzeuge ein neues Objekt mit Wert (this - b)
public Rational minus(Rational b)
 {
   int z2 = b.Zaehler;
   int n2 = b.Nenner;
   int gcd = gcd(Nenner, n2);  // Auf den Hauptnenner bringen
   int f1 = n2/gcd;
   int f2 = Nenner/gcd;
   return new Rational(Zaehler*f1-z2*f2, Nenner*f1);
}
```

- Punktnotation:

  Zugriff auf Attribute oder Methoden eines Objekts:

  `Referenz.Attribut` oder *`Referenz.Methode`*

Beispiel-Klasse: Rational

der Vollständigkeit halber: Multiplikation und Division

```
// Erzeuge ein neues Objekt mit Wert (this * b)
public Rational times(Rational b)
 {
   int z1 = Zaehler;
   int n1 = Nenner;
   int z2 = b.Zaehler;
   int n2 = b.Nenner;
   int gcd = gcd(z1, n2);     // kuerzen
   z1 /= gcd;
   n2 /= gcd;
   gcd = gcd(n1, z2);    // kuerzen
   n1 /= gcd;
   z2 /= gcd;
   return new Rational(z1*z2, n1*n2);
 }
```

```
// Erzeuge ein neues Objekt mit Wert (this / b)
public Rational divide(Rational b)
 {
   int z1 = Zaehler;
   int n1 = Nenner;
   int z2 = b.Zaehler;
   int n2 = b.Nenner;
   int gcd = gcd(z1, z2); // kuerzen: Zaehler
   z1 /= gcd;
   z2 /= gcd;
   gcd = gcd(n1, n2);    // und Nenner
   n1 /= gcd;
   n2 /= gcd;
   if(z2 == 0)
   System.err.println("### Achtung: Division durch Null");
   if(z2 > 0)        // Fallunterscheidung fuer b<0
     return new Rational(z1*n2, n1*z2);
   else
     return new Rational(-z1*n2, -n1*z2);
 }
```

+ geeignete Anzahl von „}“ um Klasse zu schließen

Beispiel-Klasse: Rational

Anwendungsbeispiel:

```
Rational a = new Rational(8, 6);
System.out.println("a = " + a);      // 4/3
Rational b = new Rational(13, 2);
System.out.println("b = " + b);      // 13/2
Rational c = a.times(b);
System.out.println("c = a*d = " + c);  // 26/3
```

verwendet eine nicht weiter
beschriebene Methode zur Ausgabe