Script generated by TTT

Title: Seidl: Informatik_1 (19.11.2012)

Date: Mon Nov 19 18:14:45 CET 2012

Duration: 87:53 min

Pages: 33

10 Klassen und Objekte

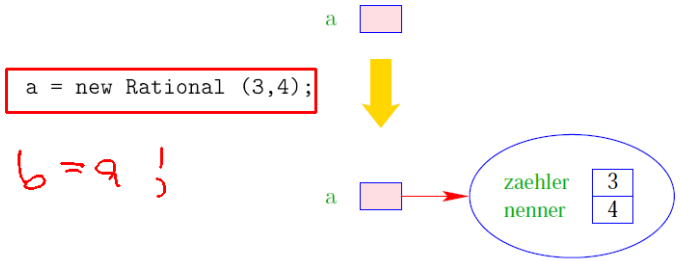
Datentyp	=	Spezifikation von Datenstrukturen
Klasse	=	Datentyp + Operationen
Objekt	=	konkrete Datenstruktur

Beispiel: Rationale Zahlen

- Eine rationale Zahl $q \in \mathbb{Q}$ hat die Form $q = \frac{x}{y}$, wobei $x, y \in \mathbb{Z}$.
- x und y heißen Zähler und Nenner von q.
- Ein Objekt vom Typ `Rational` sollte deshalb als Komponenten `int`-Variablen `zaehler` und `nenner` enthalten:

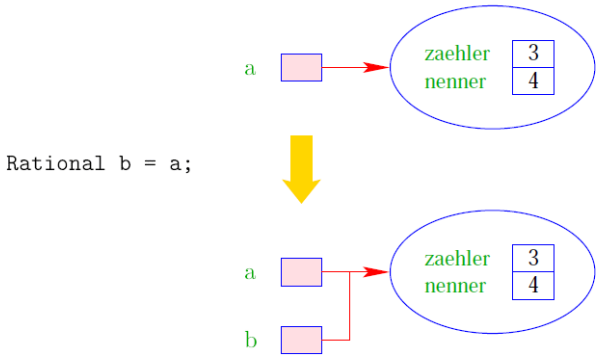
- Die Daten-Komponenten eines Objekts heißen Instanz-Variablen oder Attribute.

- `Rational name ;` deklariert eine Variable für Objekte der Klasse `Rational`.
- Das Kommando `new Rational(...)` legt das Objekt an, ruft einen Konstruktor für dieses Objekt auf und liefert das neue Objekt zurück:

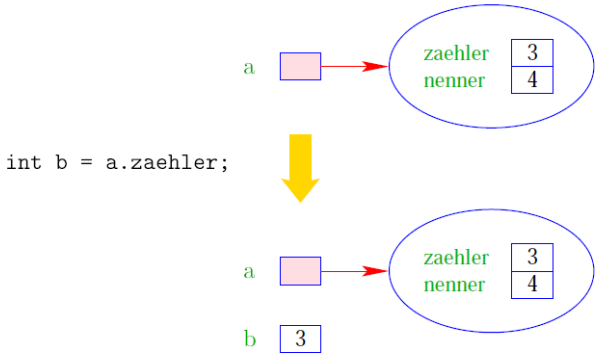


- Der Konstruktor ist eine Prozedur, die die Attribute des neuen Objekts initialisieren kann.

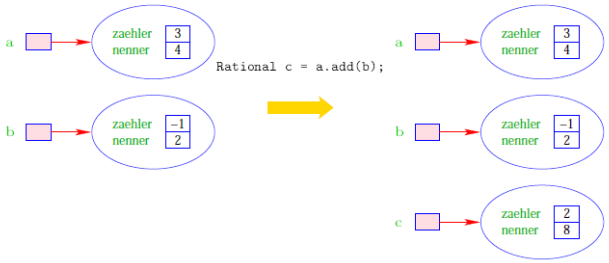
- Der Wert einer `Rational`-Variable ist ein Verweis auf einen Speicherbereich.
- `Rational b = a;` kopiert den Verweis aus `a` in die Variable `b`:

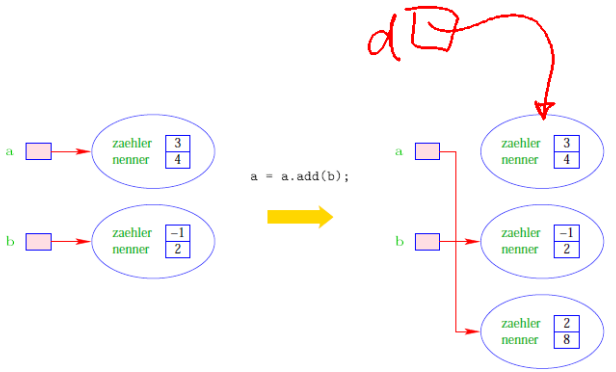


- `a.zaehler` liefert den Wert des Attributs `zaehler` des Objekts `a`:



- `a.add(b)` ruft die Operation `add` für `a` mit dem zusätzlichen aktuellen Parameter `b` auf:





- Die Operationen auf Objekten einer Klasse heißen auch Methoden, genauer: Objekt-Methoden.

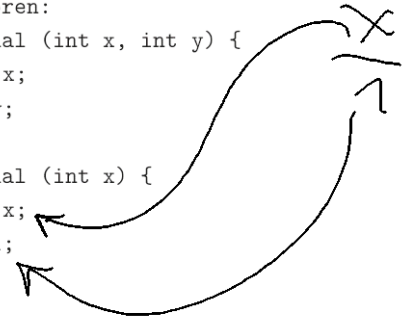
Zusammenfassung:

Eine Klassen-Deklaration besteht folglich aus Deklarationen von:

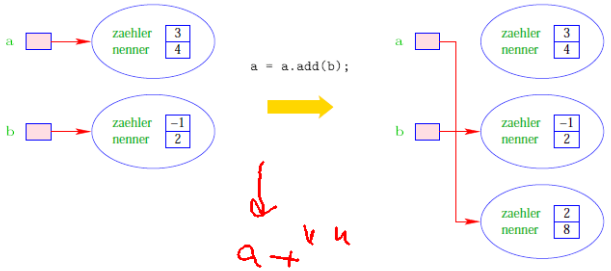
- Attributen für die verschiedenen Wert-Komponenten der Objekte;
- Konstruktoren zur Initialisierung der Objekte;
- Methoden, d.h. Operationen auf Objekten.

Diese Elemente heißen auch Members der Klasse.


```
public class Rational {
   // Attribute:
   private int zaehler, nenner;
   // Konstruktoren:
   public Rational (int x, int y) {
      zaehler = x;
      nenner = y;
      }
   public Rational (int x) {
      zaehler = x;
      nenner = 1;
      }
       ...
```




```
// Objekt-Methoden:
public Rational add (Rational r) {
   int x = zaehler * r.nenner +r.zaehler * nenner;
   int y = nenner * r.nenner;
   return new Rational (x,y);
   }
public boolean equals (Rational r) {
   return (zaehler * r.nenner == r.zaehler * nenner);
   }
public String toString() {
   if (nenner == 1) return "" + zaehler;
   if (nenner > 0) return zaehler +"/"+ nenner;
   return (-zaehler) +"/"+ (-nenner);
   }
}  // end of class Rational
```

- Die Operationen auf Objekten einer Klasse heißen auch Methoden, genauer: Objekt-Methoden.

```
// Objekt-Methoden:
public Rational add (Rational r) {
   int x = zaehler * r.nenner +r.zaehler * nenner;
   int y = nenner * r.nenner;
   return new Rational (x,y);
   }
public boolean equals (Rational r) {
   return (zaehler * r.nenner == r.zaehler * nenner);
   }
public String toString() {
   if (nenner == 1) return "" + zaehler;
   if (nenner > 0) return zaehler +"/"+ nenner;
   return (-zaehler) +"/"+ (-nenner);
   }
}  // end of class Rational
```


Bemerkungen:

- Jede Klasse sollte in einer separaten Datei des entsprechenden Namens stehen.
- Die Schlüsselworte `private` bzw. `public` klassifizieren, für wen die entsprechenden Members sichtbar, d.h. zugänglich sind.
- `private` heißt: nur für Members der gleichen Klasse sichtbar.
- `public` heißt: innerhalb des gesamten Programms sichtbar.
- Nicht klassifizierte Members sind nur innerhalb des aktuellen ↑Package sichtbar.

- Konstruktoren haben den gleichen Namen wie die Klasse.
- Es kann mehrere geben, sofern sie sich im Typ ihrer Argumente unterscheiden.
- Konstruktoren haben keine Rückgabewerte und darum auch keinen Rückgabetyp.
- Methoden haben dagegen stets einen Rückgabe-Typ, evt. `void`.

```
public void inc (int b) {
   zaehler = zaehler + b * nenner;
   }
```



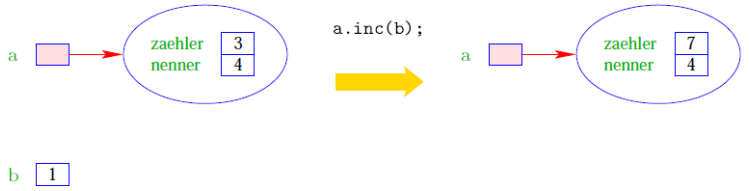
```
public class Rational {
   // Attribute:
   private int zaehler, nenner;
   // Konstruktoren:
   public Rational (int x, int y) {
      zaehler = x;
      nenner = y;
      }
   public Rational (int x) {
      zaehler = x;
      nenner = 1;
      }
       ...
```

$\frac{3}{4} + 5 = \frac{3}{4} + \frac{20}{4} = \frac{23}{4}$

- Konstruktoren haben den gleichen Namen wie die Klasse.
- Es kann mehrere geben, sofern sie sich im Typ ihrer Argumente unterscheiden.
- Konstruktoren haben keine Rückgabewerte und darum auch keinen Rückgabetyp.
- Methoden haben dagegen stets einen Rückgabe-Typ, evt. `void`.

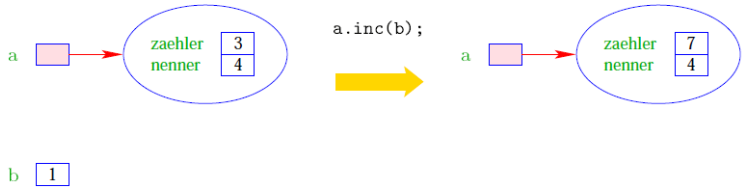
```
public void inc (int b) {
   zaehler = zaehler + b * nenner;
   }
```


- Die Objekt-Methode `inc()` modifiziert das Objekt, für das sie aufgerufen wurde.



- Die Objekt-Methode `equals()` ist nötig, da der Operator "==" bei Objekten die Identität der Objekte testet, d.h. die Gleichheit der Referenz !!!
- Die Objekt-Methode `toString()` liefert eine `String`-Darstellung des Objekts.
- Sie wird implizit aufgerufen, wenn das Objekt als Argument für die Konkatenation "+" auftaucht.
- Innerhalb einer Objekt-Methode/eines Konstruktors kann auf die Attribute des Objekts direkt zugegriffen werden.
- `private`-Klassifizierung bezieht sich auf die Klasse nicht das Objekt: die Attribute aller `Rational`-Objekte sind für `add` sichtbar !!

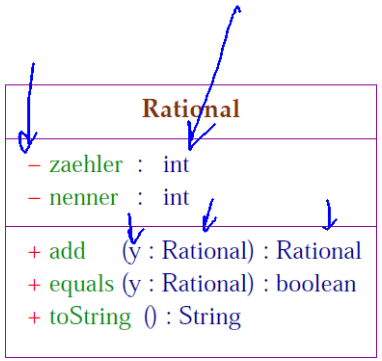
- Die Objekt-Methode `inc()` modifiziert das Objekt, für das sie aufgerufen wurde.



- Konstruktoren haben den gleichen Namen wie die Klasse.
- Es kann mehrere geben, sofern sie sich im Typ ihrer Argumente unterscheiden.
- Konstruktoren haben keine Rückgabewerte und darum auch keinen Rückgabetyp.
- Methoden haben dagegen stets einen Rückgabe-Typ, evt. `void`.

```
public void inc (int b) {
   zaehler = zaehler + b * nenner;
   }
```


Eine graphische Visualisierung der Klasse Rational, die nur die wesentliche Funktionalität berücksichtigt, könnte so aussehen:



Diskussion und Ausblick:

- Solche Diagramme werden von der UML, d.h. der Unified Modelling Language bereitgestellt, um Software-Systeme zu entwerfen (↑Software Engineering)
- Für eine einzelne Klasse lohnen sich ein solches Diagramm nicht wirklich.
- Besteht ein System aber aus sehr vielen Klassen, kann man damit die Beziehungen zwischen verschiedenen Klassen verdeutlichen.

Diskussion und Ausblick:

- Solche Diagramme werden von der UML, d.h. der Unified Modelling Language bereitgestellt, um Software-Systeme zu entwerfen (↑Software Engineering)
- Für eine einzelne Klasse lohnen sich ein solches Diagramm nicht wirklich.
- Besteht ein System aber aus sehr vielen Klassen, kann man damit die Beziehungen zwischen verschiedenen Klassen verdeutlichen.

Diskussion und Ausblick:

- Solche Diagramme werden von der UML, d.h. der Unified Modelling Language bereitgestellt, um Software-Systeme zu entwerfen (↑Software Engineering)
- Für eine einzelne Klasse lohnen sich ein solches Diagramm nicht wirklich.
- Besteht ein System aber aus sehr vielen Klassen, kann man damit die Beziehungen zwischen verschiedenen Klassen verdeutlichen.

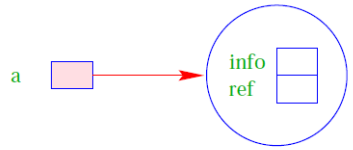
Achtung:

UML wurde nicht speziell für Java entwickelt. Darum werden Typen abweichend notiert. Auch lassen sich manche Ideen nicht oder nur schlecht modellieren.

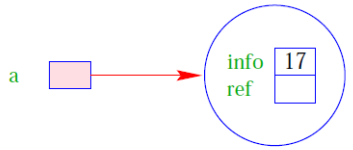
10.1 Selbst-Referenzen

```
public class Cyclic {
   private int info;
   private Cyclic ref;
   // Konstruktor
   public Cyclic() {
      info = 17;
      ref = this;
      }
      ...
} // end of class Cyclic
```


Für `Cyclic a = new Cyclic();` ergibt das:



Für `Cyclic a = new Cyclic();` ergibt das:

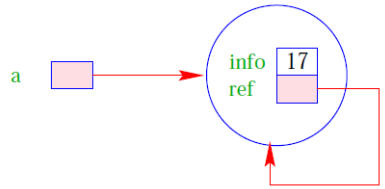


10.1 Selbst-Referenzen

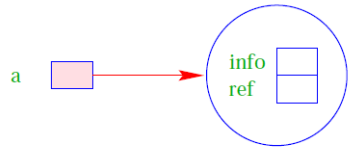
```
public class Cyclic {
   private int info;
   private Cyclic ref;
   // Konstruktor
   public Cyclic() {
      info = 17;
      ref = this;
      }
      ...
   } // end of class Cyclic
```

Innerhalb eines Members kann man mithilfe von `this` auf das aktuelle Objekt selbst zugreifen !

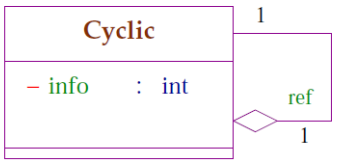
Für `Cyclic a = new Cyclic();` ergibt das:



Für `Cyclic a = new Cyclic();` ergibt das:



Modellierung einer Selbst-Referenz:



Die Rauten-Verbindung heißt auch Aggregation.

Das Klassen-Diagramm vermerkt, dass jedes Objekt der Klasse `Cyclic` einen Verweis mit dem Namen ref auf ein weiteres Objekt der Klasse `Cyclic` enthält.

10.2 Klassen-Attribute

- Objekt-Attribute werden für jedes Objekt neu angelegt,
- Klassen-Attribute einmal für die gesamte Klasse.
- Klassen-Attribute erhalten die Qualifizierung `static`.

```
public class Count {
   private static int count = 0;
   private int info;
   // Konstruktor
   public Count() {
      info = count; count++;
      } ...
   } // end of class Count
```


count 0

```
Count a = new Count();
```