Objekte

Softwareentwicklung I (IB)

Prof. Dr. Oliver Braun

Letzte Änderung: 28.01.2020 17:32

Neue Typen durch Klassen

- `int`, `double`, `boolean` sind vordefinierte Typen
- **Klassen** definieren neue Typen
- Beispiel in diesem Kapitel: Brüche, wie zum Beispiel
  $\frac{1}{2}$, $\frac{43}{937}$, $\frac{0}{1330}$, $\frac{-11}{4}$

Klassennamen

- Klassen sind mit eindeutigen Identifiern benannt
- In der Regel englische Substantive, erster Buchstabe groß
- Syntax:

```
class Classname {
   ...
}
```

- Beispiel: Klasse `Rational` für Brüche

```
class Rational {
   ...
}
```

- Jede Klassendefinition in einer eigenen Quelltextdatei
- Dateiname = Klassenname `.java`

Objektvariable

- Bestandteile eines Bruchs: Zähler, Nenner (engl. numerator, denominator)
- Komponenten in der Klassendefinition auflisten:

```
class Rational {
  int numer;
  int denom;
}
```

- Einzelne Komponente = **Objektvariable** (auch „Instanzvariable“, „Datenelement“, „Feld“, engl. field)
- Anzahl und Reihenfolge der Objektvariablen beliebig

Objektvariablen als Variablen

- Objektvariablen sind Variablen, ebenso wie bisher verwendete Variablen
- Zur begrifflichen Abgrenzung: Bisher benutzte Variablen = **lokale Variablen**
- Gleich: Syntax der Definition von Objektvariablen und lokalen Variablen
- Unterschiedlich: Ort der Definition:

Objektvariablen	... Elemente von Klassen
Lokale Variablen	... Anweisungen in Methoden

- Benennung von Objektvariablen:
  wie lokale Variablen eindeutig innerhalb einer Klasse

Primitive Typen vs. Referenztypen

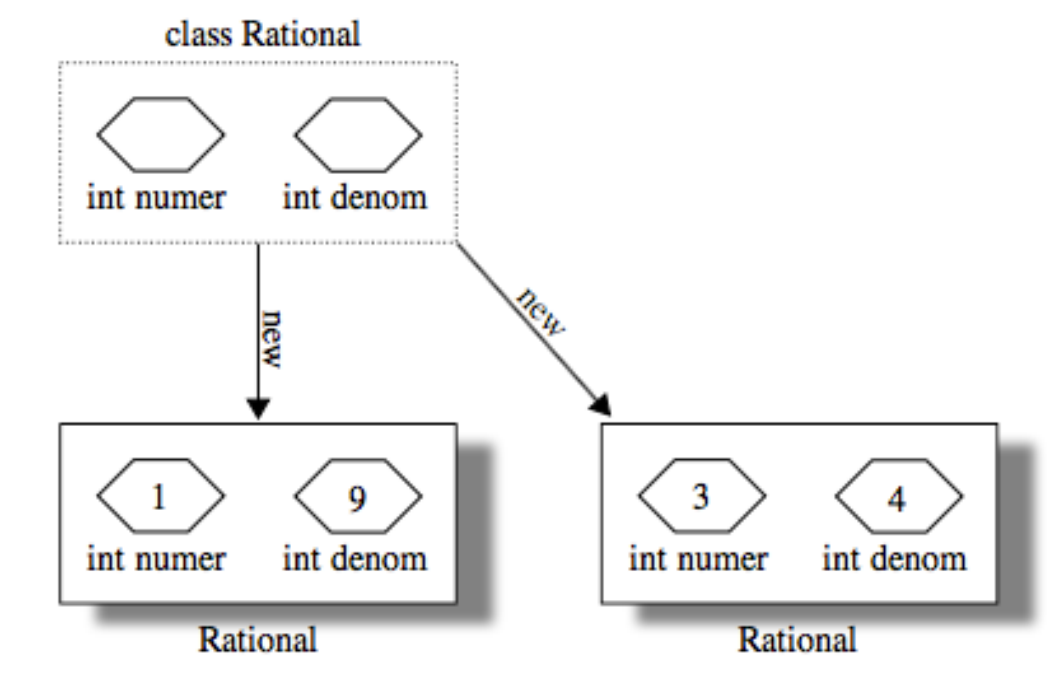
- `Rational` = neuer Typ, steht gleichberechtigt neben `int`, `double`, `boolean`
- Unterscheidung (Java-spezifisch):
  - `int`, `double`, `boolean` sind **primitive Typen**: atomar, innerer Aufbau eines Wertes spielt keine Rolle
  - `Rational` ist ein **Referenztyp**: Enthält isolierte Bestandteile (Objektvariablen), diese können einzeln angesprochen und verarbeitet werden
- Alle Klassen definieren Referenztypen
- Sortiment primitiver Typen liegt fest, können nicht neu definiert werden
- Erster Nutzen von Klassen: bündeln ihre Bestandteile

Objekte (1/2)

- Klassendefinition $\equiv$ Bauplan, Konstruktionsvorschrift, Blaupause
- Objekte der Klasse müssen explizit geschaffen werden, entstehen nicht von alleine
- **Objekt** = Exemplar, Instanz
- eine Klassendefinition $\Rightarrow$ beliebig viele Objekte

Objekte (2/2)

- Beispiel: Zwei `Rational`-Objekte mit den Werten $\frac{1}{9}$ und $\frac{3}{4}$:



Operator **new**

- Erzeugen eines neuen Objektes = **instanziieren** (auch „konstruieren“, „allokieren“)
- Syntaktisch mit Operator **new**. Beispiel:

```
new Rational()
```

- **new** produziert aus einer Klassendefinition ein einzelnes, neues Objekt dieser Klasse
- Mehrere Objekte $\Rightarrow$ mehrere Aufrufe von **new**

Klasse und Anwendung (1/2)

- Javaprogramm mit Klassen besteht aus:
  - Definition der Klasse
  - Anwendung der Klasse
- Definition und Anwendung (meist) in verschiedenen Quelltextdateien
- Klassendefinition wie oben:

```
class Rational {
   ...
}
```

Klasse und Anwendung (2/2)

- Anwendung beispielsweise in `main`:

```
class Application {
    public static void main(String[] args) {
      ... new Rational() ...
    }
}
```

- Formal: Auch `main` steckt in einer Klassendefinition; diese dient aber nur als Rahmen für `main`

new als Operator

- `new` = unärer Operator
- Priorität sehr hoch, wie andere unäre Operatoren
- Operand von `new`: Klassenname + leere, runde Klammern

```
new       Rational()
↑         ↑
Operator  Operand
```

- Wert des `new`-Ausdrucks: neu geschaffenes Objekt

Referenzvariablen

- Definition von Variablen für alle Javatypen, einschließlich Referenztypen
- Variablen von Referenztyp = **Referenzvariable**
- Beispiel:

```
Rational r;
```

- Gegensatz: „primitive Variable“ = Variable von primitivem Typ

Wertzuweisung an Referenzvariable

- Fundamentaler Unterschied:

  Primitive Variable: Variable und Speicherplatz für Wert fallen zusammen. Beispiel:

  ```
  int n = 23;
  ```

  Referenzvariable Variable und Wert (= Objekt) existieren unabhängig und getrennt.

- Definition einer Referenzvariablen schafft nur Variable, kein Objekt!

Einzelschritte beim Initialisieren einer Referenzvariablen

1. Referenzvariable definieren:

```
Rational r;
```

2. Neues Objekt mit `new` allokieren:

```
new Rational()
```

3. Objekt an Variable zuweisen:

```
r = new Rational();
```

- Variable referenziert Objekt $\Rightarrow$ Bezeichnung Referenzvariable
- Alle Schritte zusammen: Initialisieren einer Referenzvariablen bei der Definition

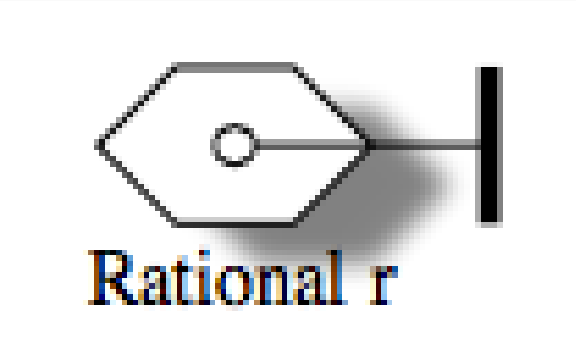
```
Rational r = new Rational();
```

`null`-Referenz (1/2)

- `null` steht für *kein Objekt*
- `null` kann an jede Referenzvariable zugewiesen werden:

```
Rational r = null;
```

- Skizze:



- Achtung: `null` ist für den Moment ganz okay, aber etwas *old-fashioned*
  - wir lernen später `Optional` kennen!

`null`-Referenz (2/2)

- `null` = wohldefinierter Wert, kann verglichen werden. Beispiel:

```
if (r == null) {
  System.out.println("no object");
}
```

- Neu definierte lokale Variablen sind nicht initialisiert, unabhängig vom Typ
- Wert `null` heisst **nicht** nicht initialisiert
- Weisen Sie einer Referenzvariablen immer `null` zu, wenn kein Objekt zur Hand ist

Zugriff auf Objektvariablen

- Jedes Objekt enthält die Objektvariablen, die in der Klassendefinition festgelegt sind
- Objektvariablen eines Objektes können einzeln angesprochen werden: Elementzugriff
- Objekt an das sich ein Elementzugriff richtet: **Zielobjekt**
- Syntax: `Zielobjekt.Objektvariable`

Beispiel

- Erzeugen eines `Rational`-Objekts mit Wert $\frac{1}{9}$:

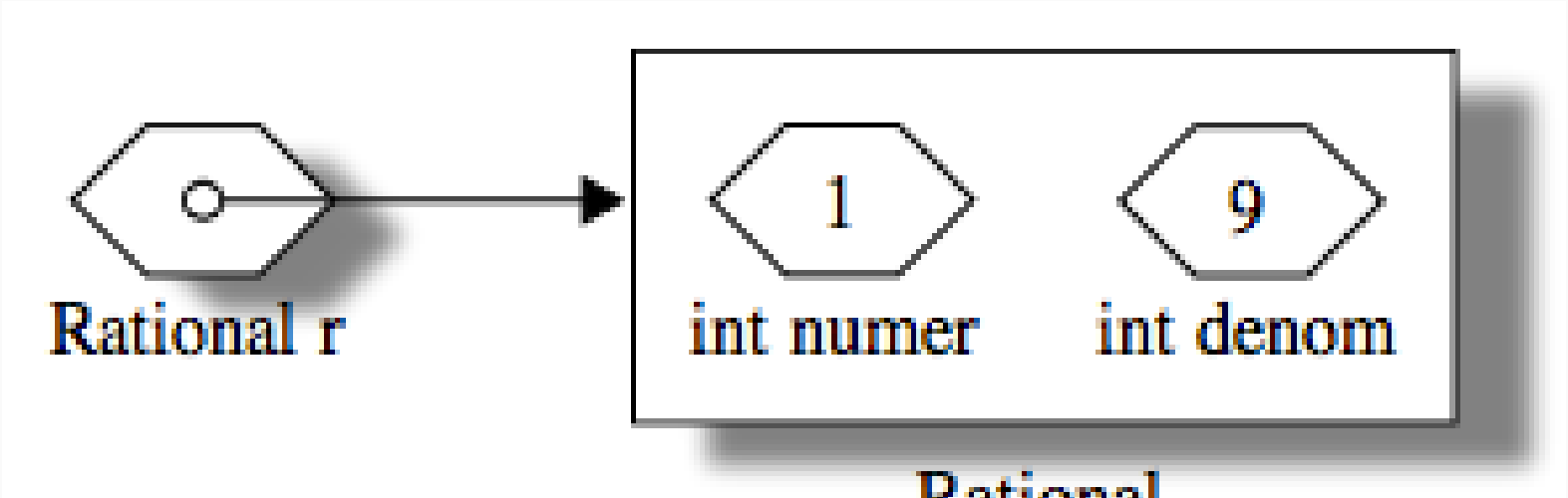
  1. `Rational`-Objekt erzeugen und an Variable zuweisen:

     ```
     Rational r = new Rational();
     ```

  2. Zähler und Nenner des Zielobjektes `r` einzeln zuweisen:

     ```
     r.numer = 1;
     r.denom = 9;
     ```

Anschließend: `r` ist mit $\frac{1}{9}$ initialisiert:



Umgang mit Objektvariablen

- Elementzugriff spricht Objektvariablen innerhalb eines Objektes an
- Gleiche Verwendung wie lokale Variablen
- Nur Zugriffssyntax zeigt Unterschied zwischen Objektvariablen und lokalen Variablen

Beispiel

- Zähler oder Nenner eines `Rational`-Objektes ...
  - in einem Ausdruck verwenden:

```
int i = 5 - r.numer * 3;
```

  - mit Operatorzuweisung und Inkrementoperator modifizieren:

```
r.numer *= 10;
r.numer++;
```

  - vergleichen:

```
if (r.denom ≠ 0) ...
```

  - usw...

Objektvariablen unterschiedlicher Objekten (1/3)

- Jedes Objekt hat eigene Objektvariablen
- Elementzugriff richtet sich an eine Objektvariable innerhalb eines Objektes (des Zielobjektes)
- Andere Objektvariablen des Zielobjektes und Objektvariablen anderer Objekte unberührt

Objektvariablen unterschiedlicher Objekten (2/3)

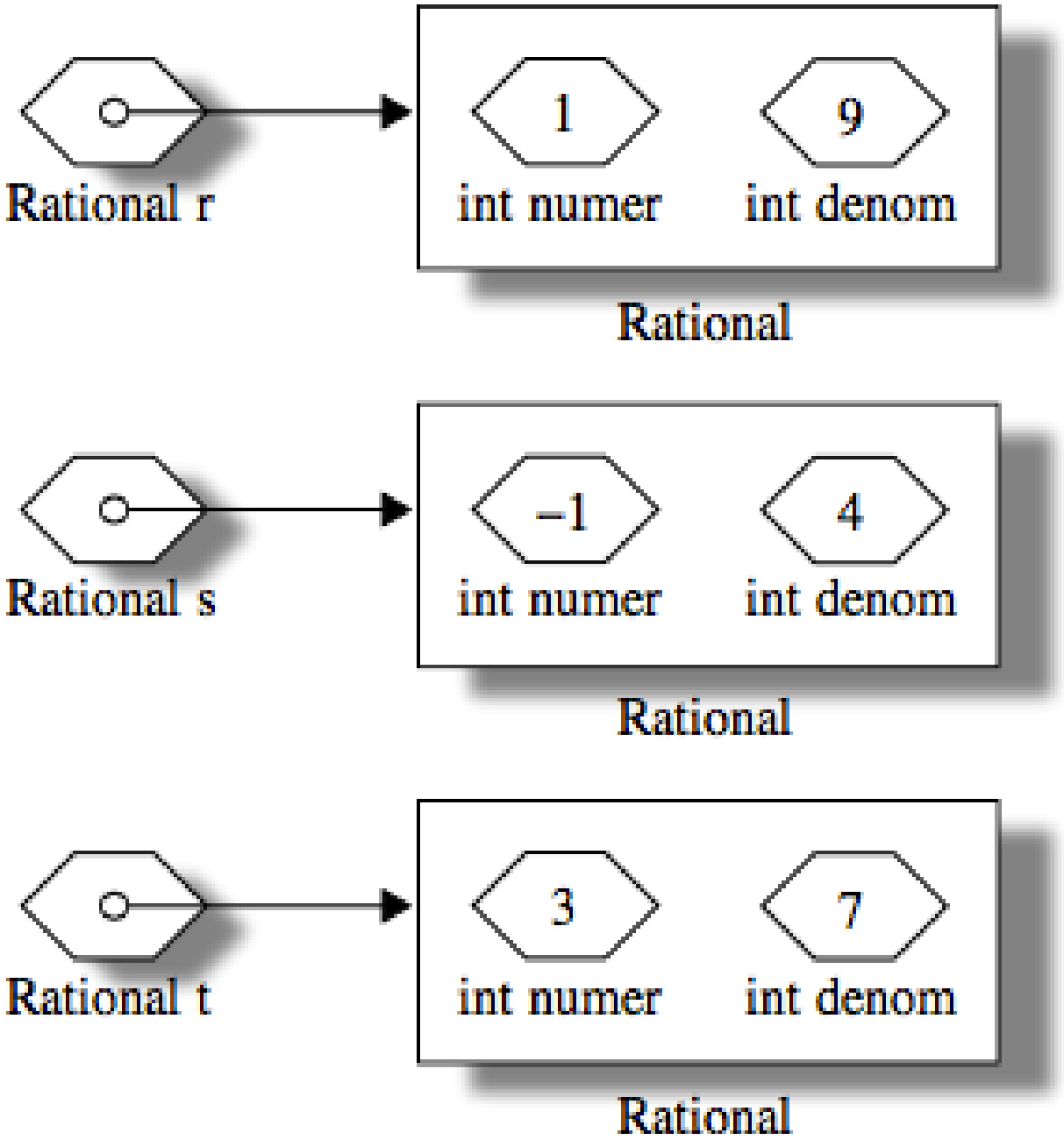
- Beispiel: Drei Brüche erzeugen, mit unterschiedlichen Werten initialisieren:

```
Rational r = new Rational();
r.numer = 1;
r.denom = 9;                          // r = 1/9

Rational s = new Rational();
s.numer = -1;
s.denom = 4;                          // s = -1/4

Rational t = new Rational();
t.numer = 3*r.numer;
t.denom = r.denom + 2 - s.denom;      // t = 3/7
```

Objektvariablen unterschiedlicher Objekten (3/3)



Wertzuweisungen bei primitiven Typen

- Wertzuweisung primitiver Typen kopiert den Wert

```
int a = 1;
int b = a;
```

Beide Variablen haben Wert 1:

- Änderungen einer Variablen ohne Auswirkungen auf die andere:

```
...
b++;
System.out.println(a);          // gibt 1 aus
```

`b` inkrementiert auf 2, `a` immer noch 1.

Wertzuweisungen bei Referenztypen (1/2)

- Anders bei Referenzvariablen: Wertzuweisung bei Referenztypen dupliziert Referenz, nicht das Objekt!

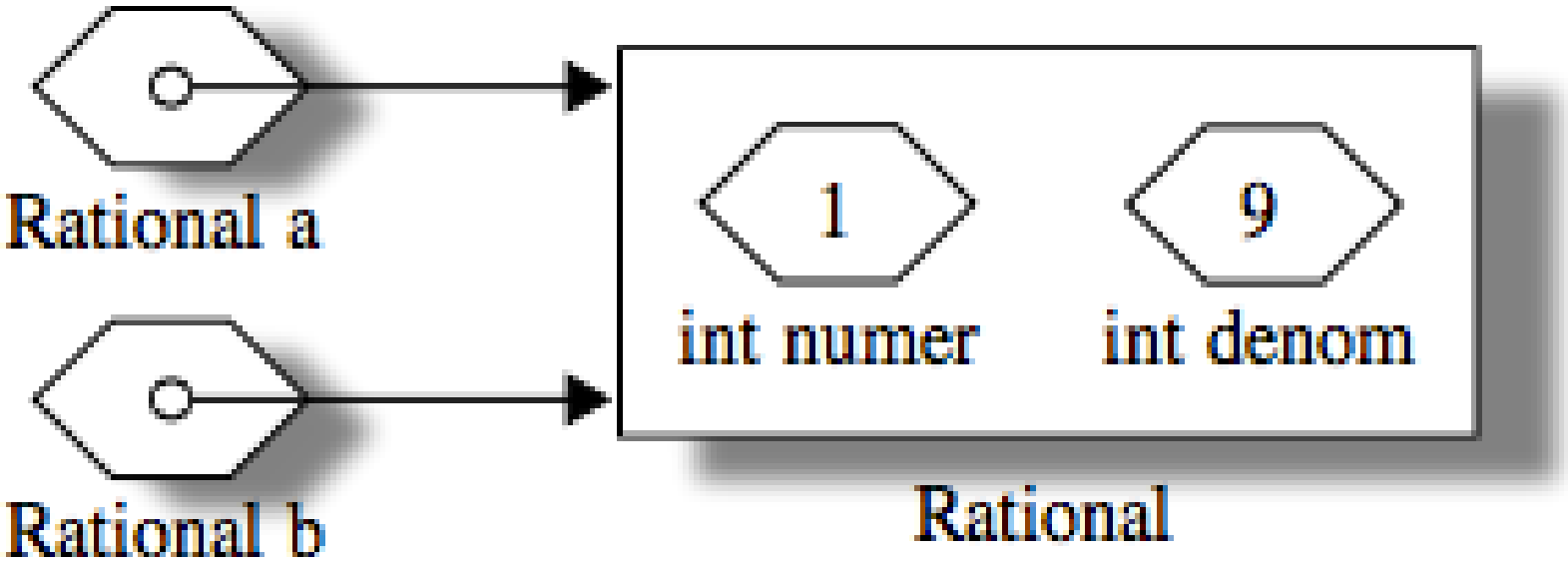
```
Rational a = new Rational();
a.numer = 1;
a.denom = 9;
Rational b = a;
```

  Beide Variablen referenzieren dasselbe Objekt mit Wert $\frac{1}{9}$

- **Aliasing** (dt. etwa: „Überlagerung“)

- Aliasing in der Praxis kontrollierbar, sogar wünschenswert

Wertzuweisungen bei Referenztypen (2/2)



- Änderungen eines Objektes in beiden Variablen sichtbar

```
...
b.numer++;
System.out.println(a.numer);     // gibt 2 aus!
```

Vergleich von Referenztypen (1/4)

- Vergleich von Objekten mit `==` prüft die Identität:
- `true`: wenn beide Operanden **ein und das selbe** Objekt sind
- `false`: wenn die Operanden verschiedene Objekte sind
- Vergleich mit `==` ignoriert den Inhalt der Objekte

Vergleich von Referenztypen (2/4)

- Identität aus Sicht der Programmlogik meist wenig interessant.

  Beispiel:

```
Rational a = new Rational();
a.numer = 1;
a.denom = 9;
Rational b = new Rational();
b.numer = 1;
b.denom = 9;

if(a == b) // false
    ...
```

Vergleich von Referenztypen (3/4)

- Dagegen:

```
Rational a = ...;
Rational b = a;

if (a == b) // true
    ...
```

Vergleich von Referenztypen (4/4)

- Inhaltlicher Vergleich von Objekten: Objektvariablen paarweise vergleichen

```
Rational a = ...;
Rational b = ...;

if (a.numer == b.numer  &&  a.denom == b.denom)
    ...
```

- Frage des Objektvergleichs wird später wieder aufgegriffen

Lebensdauer

- Lokale Variablen werden ...

geschaffen	wenn die Definition erreicht wird
freigegeben	wenn der Block der Definition verlassen wird

- Objektvariablen werden ...

geschaffen	sobald ein Objekt erzeugt wird (mit **new**)
freigegeben	wenn das Objekt nicht mehr erreichbar ist

- Lebensdauer von Objekten einschließlich der darin enthaltenen Objektvariablen unabhängig von lokalen Variablen