Einführung in die Programmierung

10. Klassen und Objekte

Prof. Dr. Marcel Luis

Westf. Hochschule

WS 2022/2023

Motivation und Einführung

- Bisher kennen wir keine Möglichkeit, strukturierte Daten (z. B. Rechtecke) als eine *Einheit* zu behandeln.
- Einzige Möglichkeit: unmittelbar auf den „Einzelkomponenten“ strukturierter Daten (z. B. den Koordinaten der Rechtecke) operieren.
- Besser wäre es jedoch, einmalig zu beschreiben, wie ein Rechteck zusammengesetzt ist und fortan nur noch mit Rechtecken zu operieren.
- Die Möglichkeit dazu bieten *Klassen* und *Objekte*.

Klasse als „Bauplan“ für Objekte

```
public class Rechteck {

    /** Ursprung, d.h. linke obere Ecke dieses Rechtecks. */
    private Punkt ursprung;

    /** Breite dieses Rechtecks. */
    private double breite;

    /** Höhe dieses Rechtecks. */
    private double hoehe;

    ...
}
```

Klasse als Typ

Mit dieser Definition der Klasse ist es nun möglich, `Rechteck` als *Typ* in Variablendeklarationen zu verwenden, denn mit der *Klasse* `Rechteck` wird gleichzeitig der *Klassen-Typ* `Rechteck` definiert.
Man kann nun z. B. lokale Variablen deklarieren, deren Typ `Rechteck` ist

```
Rechteck einRechteck;
```

oder Methoden definieren, deren formale Parameter oder Rückgabe vom Typ `Rechteck` sind.

Klasse als Typ

Beispiel

```
/**
 * Erzeugt umschließendes Rechteck für r1 und r2.
 *
 * @param r1  ein Rechteck
 * @param r2  zweites Rechteck
 * @return kleinstes Rechteck, das r1 und r2
 *         umschließt
 */
public static Rechteck erzeugeHuelle(Rechteck r1,
                                     Rechteck r2) {

  ...
}
```

Strukturierte Daten

- Der Umgang mit strukturierten Daten und die Möglichkeit, die Struktur dieser Daten zu beschreiben, ist ein wesentliches Element der Software-Entwicklung.
- In nahezu jeder praktischen Anwendung hat man es nicht ausschließlich mit *elementaren* Werten zu tun, sondern mit vielfältig *strukturierten* Daten, die oftmals reale Dinge dieser Welt repräsentieren.
- Alle höheren Programmiersprachen bieten Unterstützung für den Umgang mit strukturierten Daten.
- Es gibt aber Unterschiede hinsichtlich des *Blickwinkels*, aus dem diese Daten betrachtet werden.

Zwei Sichtweisen auf Algorithmus

Beispiel

Problem

Suche in einem Bücherkatalog anhand eines Suchbegriffs und Aufbereitung der Ergebnisliste zur Anzeige im Browser (HTML-Format).

Algorithmus (prozedurale Sicht)

1. Es werden alle Bücher des Katalogs durchlaufen.
2. Je Buch wird geprüft, ob es zum Suchbegriff passt.
3. Wenn ja, wird es der Ergebnisliste hinzugefügt.
4. Aus der Ergebnisliste wird eine HTML-Darstellung erzeugt.

Zwei Sichtweisen auf Algorithmus

Beispiel

Algorithmus (objekt-orientierte Sicht)

1. Der *Katalog durchläuft* seine Bücher.
2. Jedes *Buch prüft*, ob es zum Suchbegriff passt.
3. Wenn ja, *nimmt* es die *Ergebnisliste* bei sich *auf*.
4. Der „*HTML-Redakteur*“ *schreibt* die Ergebnisliste in seinem Format.

Prozedural vs. objekt-orientiert

Prozedural	Objekt-orientiert
Es wird nur gesagt, *was* gemacht wird. Es wird nicht gesagt, *wer* etwas macht.	Es wird auch gesagt, *wer* etwas macht.
Der Algorithmus ist im Passiv formuliert.	Der Algorithmus ist im Aktiv formuliert.
Die Daten werden bearbeitet.	Die Daten *selbst* führen die Tätigkeiten aus. Sie sind aktiv. Sie sind die *Akteure* des Algorithmus.

Prozedural vs. objekt-orientiert

Prozedural	Objekt-orientiert
Es wird nur gesagt, *was* gemacht wird. Es wird nicht gesagt, *wer* etwas macht.	Es wird auch gesagt, *wer* etwas macht.
Der Algorithmus ist im Passiv formuliert.	Der Algorithmus ist im Aktiv formuliert.
Die Daten werden bearbeitet.	Die Daten *selbst* führen die Tätigkeiten aus. Sie sind aktiv. Sie sind die *Akteure* des Algorithmus.

Wie definiert man, wozu *Objekte* (die strukturierten Daten) zu agieren in der Lage sind?

Klasse als Beschreibung der Fähigkeiten ihrer Objekte

```
public class Rechteck {

    /** Ursprung, d.h. linke obere Ecke dieses Rechtecks. */
    private Punkt ursprung;

    ...

    /**
     * Liefert die Fläche dieses Rechtecks.
     * @return Fläche dieses Rechtecks
     */
    public double gibFlaeche() {

        return breite * hoehe;
    }
}
```

Zusammenfassung der Einleitung

Was enthält die Klasse `Rechteck`?

- Definition der *Struktur* von `Rechteck`-Objekten
- *Methoden*, die Rechtecke (selbst!) ausführen können

Zusammenfassung der Einleitung

Was enthält die Klasse `Rechteck`?

- Definition der *Struktur* von `Rechteck`-Objekten
- *Methoden*, die Rechtecke (selbst!) ausführen können

Wichtig

- Diese Methoden werden ohne den Modifikator `static` definiert.
- Methoden, die von Objekten ausgeführt werden, sind *keine* statischen Methoden.

Erzeugen von Objekten

- Objekte existieren nicht einfach per se, sondern müssen *erzeugt* werden.
- Das geschieht durch sogenannte *Konstruktoren*. Dies sind spezielle Methoden, deren einzige Aufgabe es ist, Objekte zu erzeugen und ggf. mit Werten zu initialisieren.

Konstruktoren

Die Syntax für Konstruktoren ist:

```
Modifikatoren Klassenname(Parameterliste) {

    /* Rumpf des Konstruktors */
    ...
}
```

- Konstruktoren heißen stets so wie die Klassen, in denen sie stehen.
- Eine Klasse kann beliebig viele Konstruktoren enthalten, sofern diese sich hinsichtlich ihrer Parametertypen unterscheiden (vgl. Überladen von Methoden).

Konstruktoren

Beispiele

```
/**
 * Erzeugt ein Rechteck mit den angegebenen Koordinaten.
 *
 * @param x  horizontaler Ursprung dieses Rechtecks
 * @param y  vertikaler Ursprung dieses Rechtecks
 * @param breite  Breite
 * @param hoehe  Höhe
 */
public Rechteck(double x, double y,
                double breite, double hoehe) {

    ursprung = new Punkt(x, y);
    this.breite = breite;
    this.hoehe = hoehe;
}
```

Konstruktoren

Beispiele

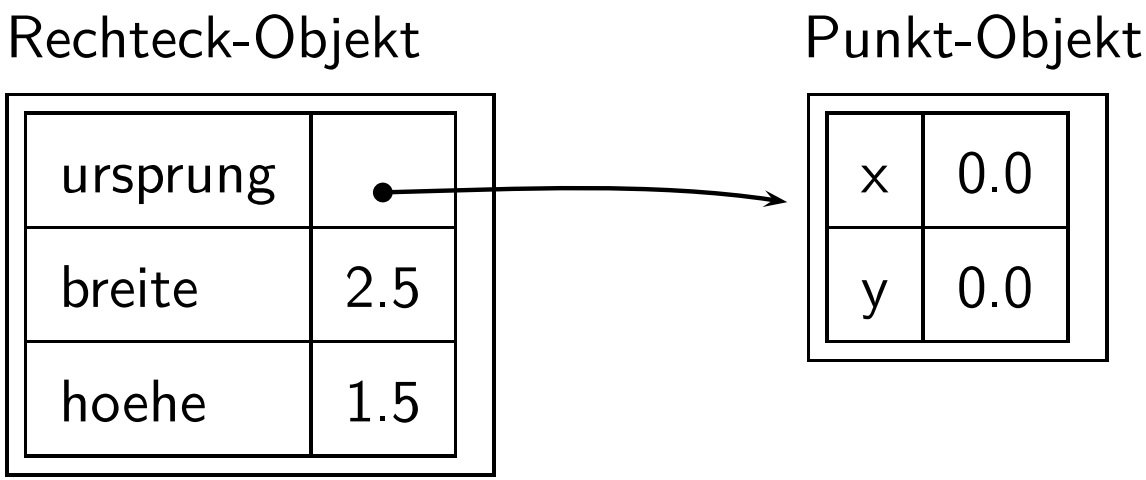
```
/**
 * Erzeugt ein Rechteck mit der angegebenen Breite und Höhe.
 * Der Ursprung liegt bei (0, 0).
 *
 * @param breite  Breite
 * @param hoehe  Höhe
 */
public Rechteck(double breite, double hoehe) {

    this.ursprung = new Punkt(0, 0);
    this.breite = breite;
    this.hoehe = hoehe;
}
```

Aufruf von Konstruktoren

Beispiel für Klasse `Rechteck`

- Durch die Auswertung des Konstruktor-Aufrufs
  `new Rechteck(2.5, 1.5)` wird ein Objekt der Klasse `Rechteck` erzeugt, das (solange es existiert) genau dieser Klasse angehört.
- Die Struktur des Objekts entspricht den in der Klasse `Rechteck` aufgeführten Variablen `ursprung`, `breite` und `hoehe`.



Konstruktor-Aufruf ist ein Ausdruck

- Der Aufruf eines Konstruktors ist ein Ausdruck, der innerhalb anderer Ausdrücke verwendet werden kann.
- Der Typ des Ausdrucks ist die Klasse, von der das Objekt erzeugt wird.
- `new Rechteck()` ist also ein Ausdruck des (Klassen-) Typs `Rechteck`.
- Bei Zuweisungen `Variable = Ausdruck`, in denen der Typ der Variablen ein Klassen-Typ ist, muss der Typ des Ausdrucks mit dem Typ der Variablen übereinstimmen. (In der Vorlesung OPR wird diese Regel noch modifiziert.)
- Dies gilt gleichermaßen für die Übergabe von aktuellen Parametern an formale Parameter beim Aufruf von Methoden.

Instanzmethoden

Instanzmethoden einer Klasse werden von Objekten dieser Klasse ausgeführt.

Anwendungsbeispiel

```
Punkt p = new Punkt(2, 5);
Rechteck r = new Rechteck(1, 2, 3, 4);
System.out.println(r.gibFlaeche());
System.out.println(r.enthaeltPunkt(p));
```

Instanzmethoden

Definition der Methode `gibFlaeche` in `Rechteck`

```
/**
 * Liefert die Fläche dieses Rechtecks.
 *
 * @return Fläche dieses Rechtecks
 */
public double gibFlaeche() {

    return breite * hoehe;
}
```

Instanzmethoden

Definition der Methode `enthaeltPunkt` in `Rechteck`

```
/**
 * Prüft, ob der übergebene Punkt innerhalb dieses Rechtecks
 * oder auf dessen Begrenzungslinien liegt.
 *
 * @param punkt  Punkt
 * @return <code>true</code> genau dann, wenn der Punkt innerhalb
 *         dieses Rechtecks oder auf seinen Begrenzungslinien liegt
 */
public boolean enthaeltPunkt(Punkt punkt) {

    return (ursprung.gibX() <= punkt.gibX()
            && punkt.gibX() <= ursprung.gibX() + breite
            && ursprung.gibY() <= punkt.gibY()
            && punkt.gibY() <= ursprung.gibY() + hoehe);
}
```

Instanzmethoden

- Instanzmethoden sind Methoden, die von Objekten (Instanzen) ausgeführt werden. Sie werden *ohne* den Modifikator `static` deklariert.
- Alle Objekte einer Klasse können dieselben Methoden ausführen, und zwar diejenigen, die in der Klasse definiert sind.
- Die Formulierung „ein Objekt führt eine Methode aus“ drückt die Vorstellung aus, dass Objekte *aktive* Daten sind.
- Eine Instanzmethode operiert auf den Instanzvariablen des Objekts (im Beispiel `ursprung`, `breite` und `hoehe`) und den Parametern der Methode (im Beispiel `punkt`).

Aufruf von Instanzmethoden

- Der Aufruf einer Instanzmethode erfolgt durch

  $$\mathit{Objektausdruck}\,.\,\mathit{Methodenname}(\mathit{Ausdruck}_1,\ \ldots,\ \mathit{Ausdruck}_n)$$

- Der Objektausdruck vor dem Punktoperator (`.`) ist ein Ausdruck, dessen Wert ein Objekt ist.
- Die Methode muss in der Klasse T definiert sein, wenn T der Typ des Objektausdrucks ist.

Instanzmethoden ohne Rückgabe

- Eine Instanzmethode operiert auf den Instanzvariablen ihres Objekts. Der Zustand des Objekts kann durch Zuweisungen an die Instanzvariablen verändert werden.
- Ist es Aufgabe einer Methode, nur den Zustand eines Objekts zu verändern und keinen Ergebniswert zu liefern, so wird die Methode mit Rückgabetyp `void` deklariert (engl. *void* = leer, ohne, unbesetzt).
- Der Aufruf einer Methode, deren Rückgabetyp `void` ist, ist kein Ausdruck, sondern eine Anweisung. Die Ausführung der Methode liefert *keinen* Wert.

Instanzmethoden ohne Rückgabe

Beispiel

```
/**
 * Vergrößert dieses Rechteck um den angegebenen Faktor.
 *
 * @param faktor Faktor, um den das Rechteck vergrößert
 * wird
 */
public void vergroessere(double faktor) {

    breite = faktor * breite;
    hoehe = faktor * hoehe;
}
```

Identität von Objekten

- Jedes Objekt besitzt nach seinem Erzeugen eine eigene Identität, einen einzigartigen „Fingerabdruck“. (Letztendlich handelt es sich bei der Identität um die Adresse des Objekts im Speicher der virtuellen Maschine.)
- Die Operatoren == und != vergleichen die *Identität* von Objekten.

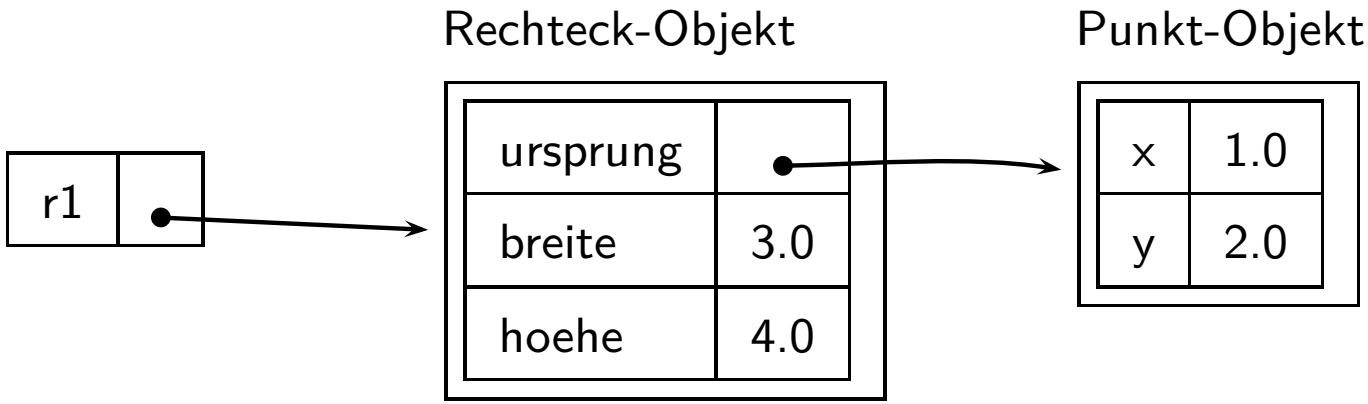
Identität von Objekten

Beispiel

```
Rechteck r1 = new Rechteck(1, 2, 3, 4);
Rechteck r2 = new Rechteck(1, 2, 3, 4);
System.out.println(r1 == r2);
r2 = r1;
System.out.println(r1 == r2);
```

Identität von Objekten

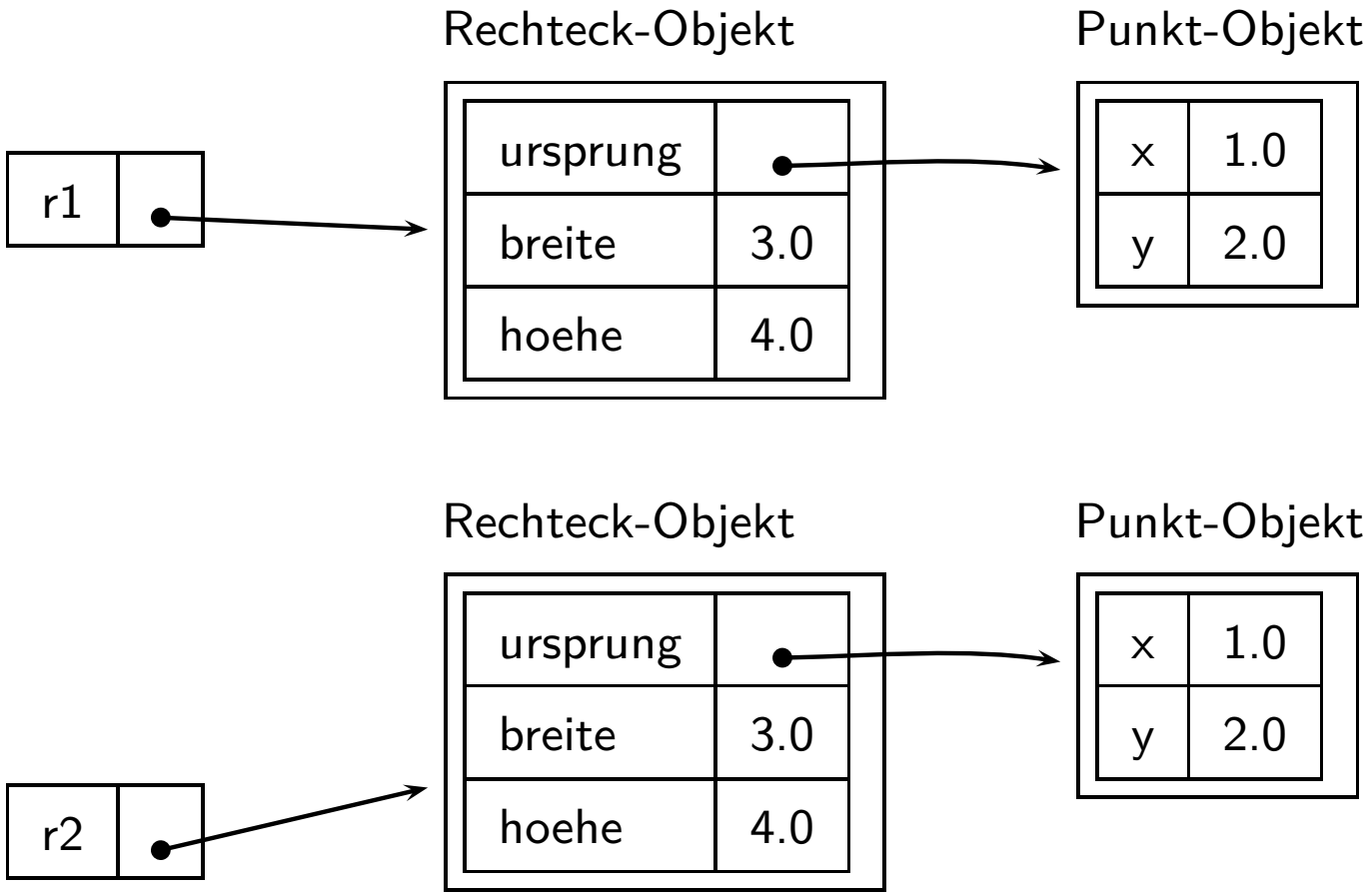
Fortsetzung des Beispiels




```
Rechteck r1 = new Rechteck(1, 2, 3, 4);
```

Identität von Objekten

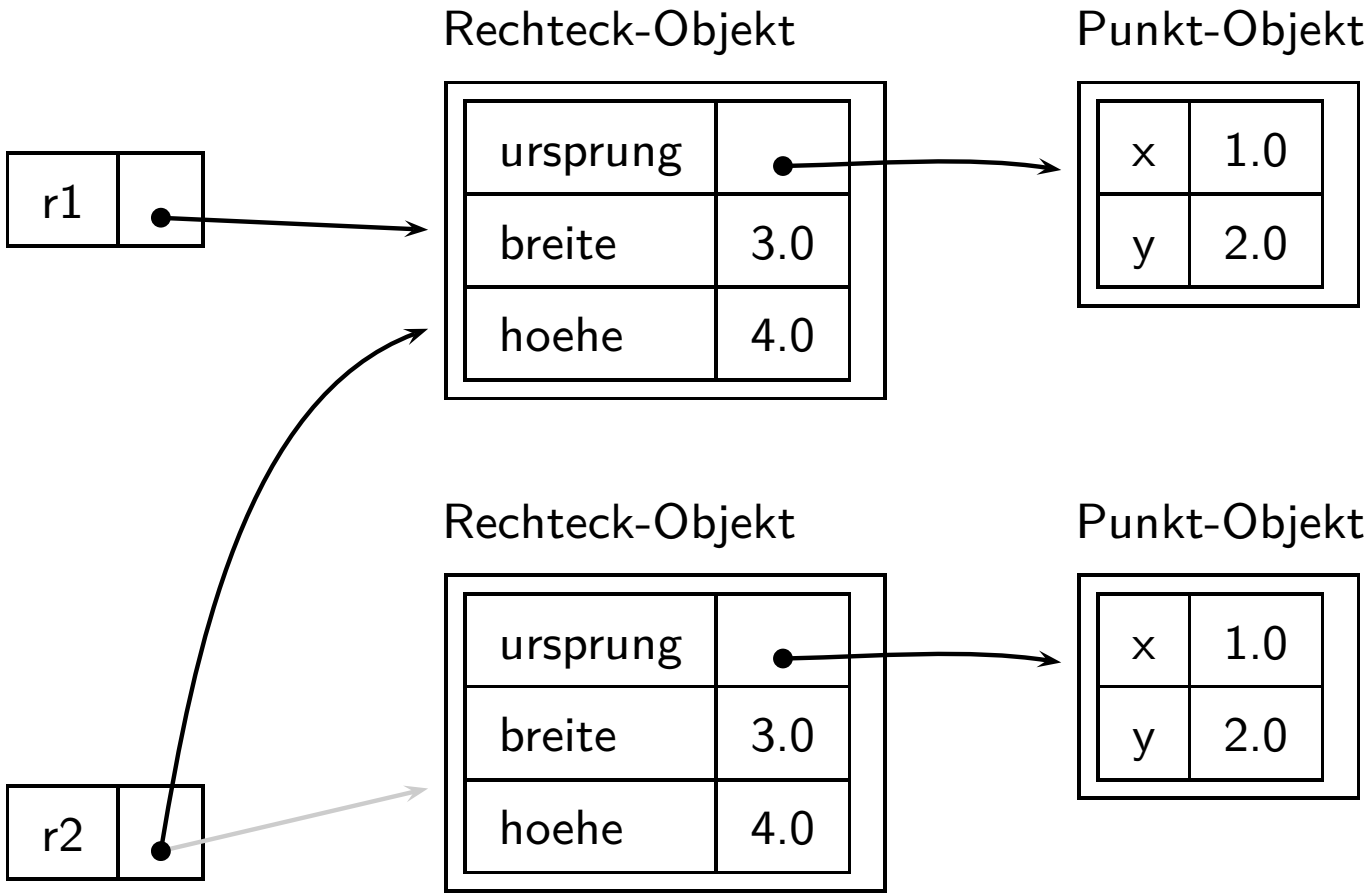
Fortsetzung des Beispiels




```
Rechteck r2 = new Rechteck(1, 2, 3, 4);
System.out.println(r1 == r2); //false
```

Identität von Objekten

Fortsetzung des Beispiels




```
r2 = r1;
System.out.println(r1 == r2); //true
```

Instanzvariablen

- In der Literatur werden die Begriffe *Objekt* und *Instanz* weitgehend synonym verwendet. Die Variablen eines Objekts werden deshalb *Instanzvariablen* genannt.
- Welche Instanzvariablen ein Objekt besitzt, ergibt sich aus dem „Bauplan“ der Klasse, durch deren Konstruktor das Objekt erzeugt wurde.
- Instanzvariablen werden in der Regel – und in dieser Veranstaltung *immer* – mit dem Modifikator `private` deklariert.

Zugriff auf Instanzvariable

Eine Instanzvariable mit Modifikator `private` kann nur in der Klasse verwendet werden, in der sie deklariert ist.

Syntax für Zugriff auf Instanzvariable

Objektausdruck . *Variablenname*

- Durch *Objektausdruck* wird das Objekt angegeben, auf dessen Instanzvariable zugegriffen wird.
- Häufig ist der Objektausdruck einfach eine Variable. Es ist jedoch jeder Ausdruck des geeigneten Typs erlaubt.
- Allgemein: Wenn der Objektausdruck den Typ T hat, wobei T eine Klasse ist, dann muss die Instanzvariable in dieser Klasse definiert sein.

Zugriff auf Instanzvariable

```
/**
 * Liefert das kleinste Rechteck, das dieses und das übergebene
 * Rechteck umhüllt. Beide Rechtecke bleiben unverändert.
 * @param r  ein Rechteck
 * @return kleinstes Rechteck um beide Rechtecke
 */
public Rechteck erzeugeHuelle(Rechteck r) {

    /* Minimale und maximale horizontale und vertikale
     * Koordinaten beider Rechtecke bestimmen.
     */
    double xMin = Math.min(ursprung.gibX(), r.ursprung.gibX());
    double yMin = Math.min(ursprung.gibY(), r.ursprung.gibY());

    double xMax = Math.max(ursprung.gibX() + breite,
                           r.ursprung.gibX() + r.breite);
    double yMax = Math.max(ursprung.gibY() + hoehe,
                           r.ursprung.gibY() + r.hoehe);

    return new Rechteck(xMin, yMin, xMax - xMin, yMax - yMin);
}
```

Zugriff auf Instanzvariable

- Durch `r.ursprung` erfolgt der Zugriff auf die Instanzvariable `ursprung` des übergebenen Rechtecks in der Variablen `r`.
- Durch `ursprung` erfolgt der Zugriff auf die Instanzvariable des Rechteck-Objekts, das die Methode ausführt.
- Auch in diesem Fall ist die Notation *Objektausdruck*.*Variablenname* möglich: `this.ursprung`.
- `this` ist eine *Pseudovariable* und enthält als Wert stets dasjenige Objekt, das die Methode gerade ausführt.

Zustand eines Objekts

- Jedes Objekt einer Klasse besitzt seine *eigenen* Instanzvariablen.
- Diese sind zwar *namentlich gleich* mit den Instanzvariablen aller anderen Objekte dieser Klasse, ihre Werte sind aber *unabhängig* voneinander.
- Die Werte der Instanzvariablen repräsentieren den *Zustand* des Objekts.

Zustand eines Objekts

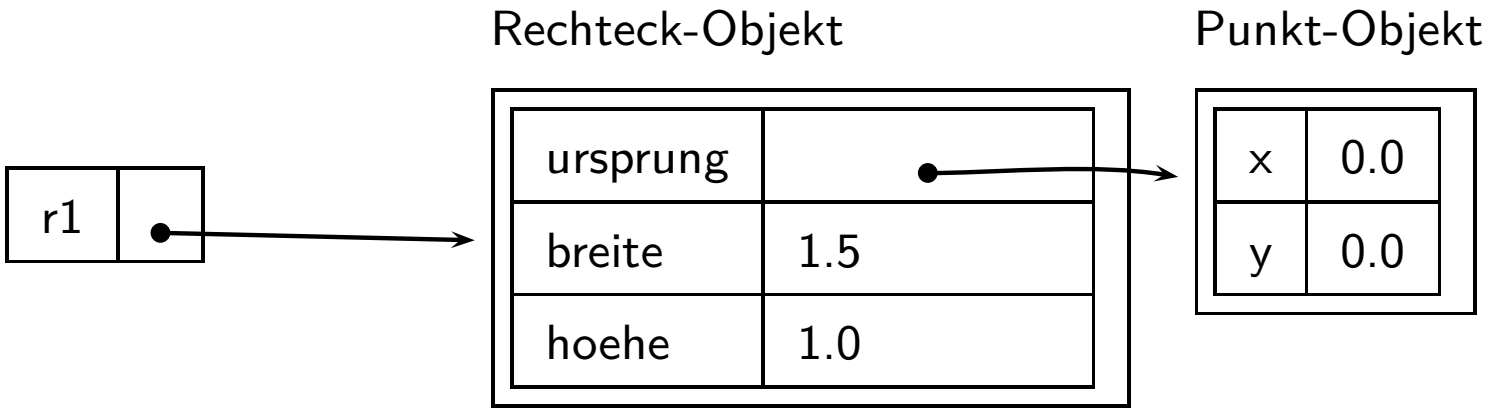
Beispiel

```
Rechteck r1 = new Rechteck(1.5, 1.0);
Rechteck r2 = new Rechteck(2.5, 2.0);
r1.vergroessere(2);
r2 = r1;
r2.vergroessere(3);
```

Was ist der Zustands der Objekte in `r1` und `r2`?

Zustand eines Objekts

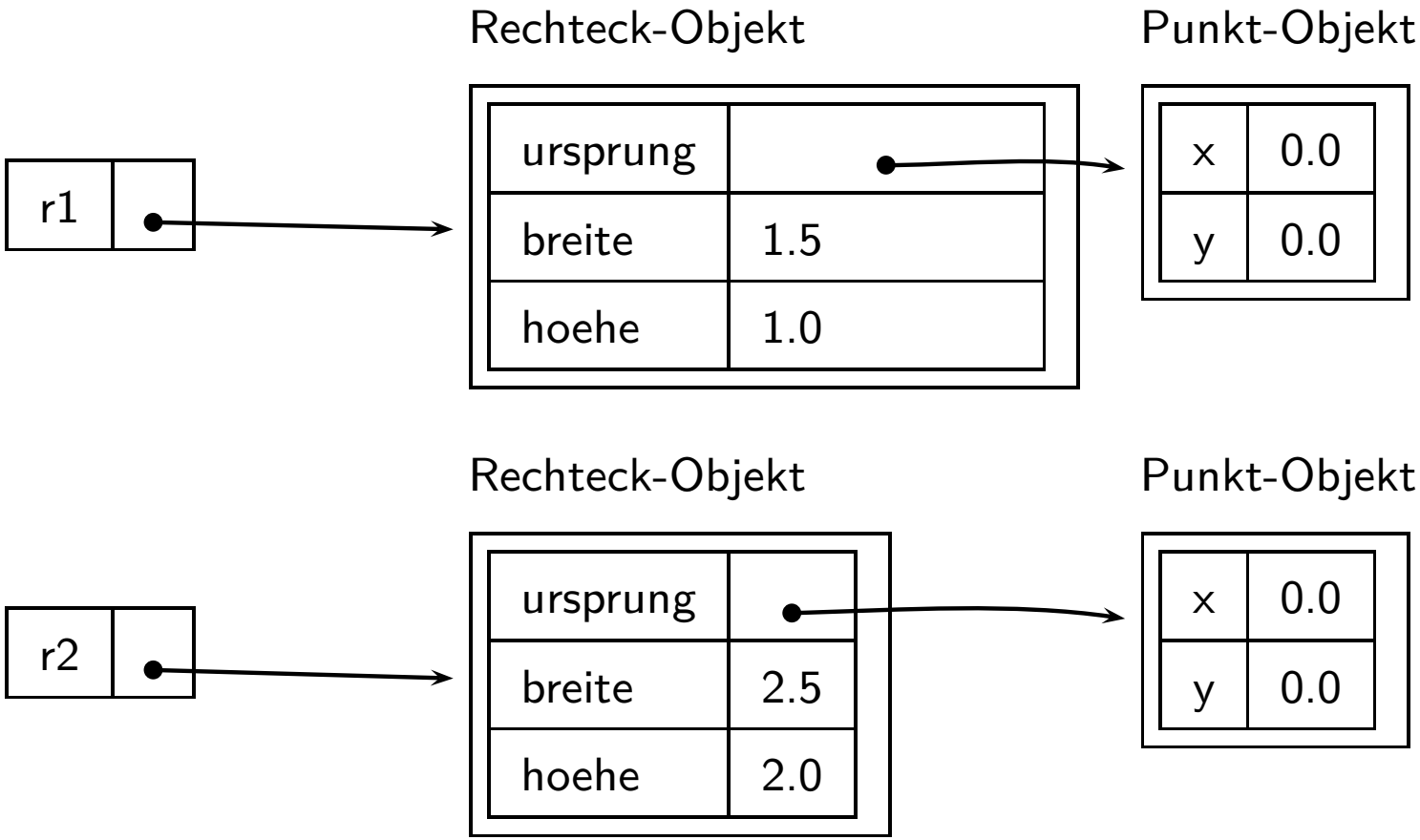
Fortsetzung des Beispiels




```
Rechteck r1 = new Rechteck(1.5, 1.0);
```

Zustand eines Objekts

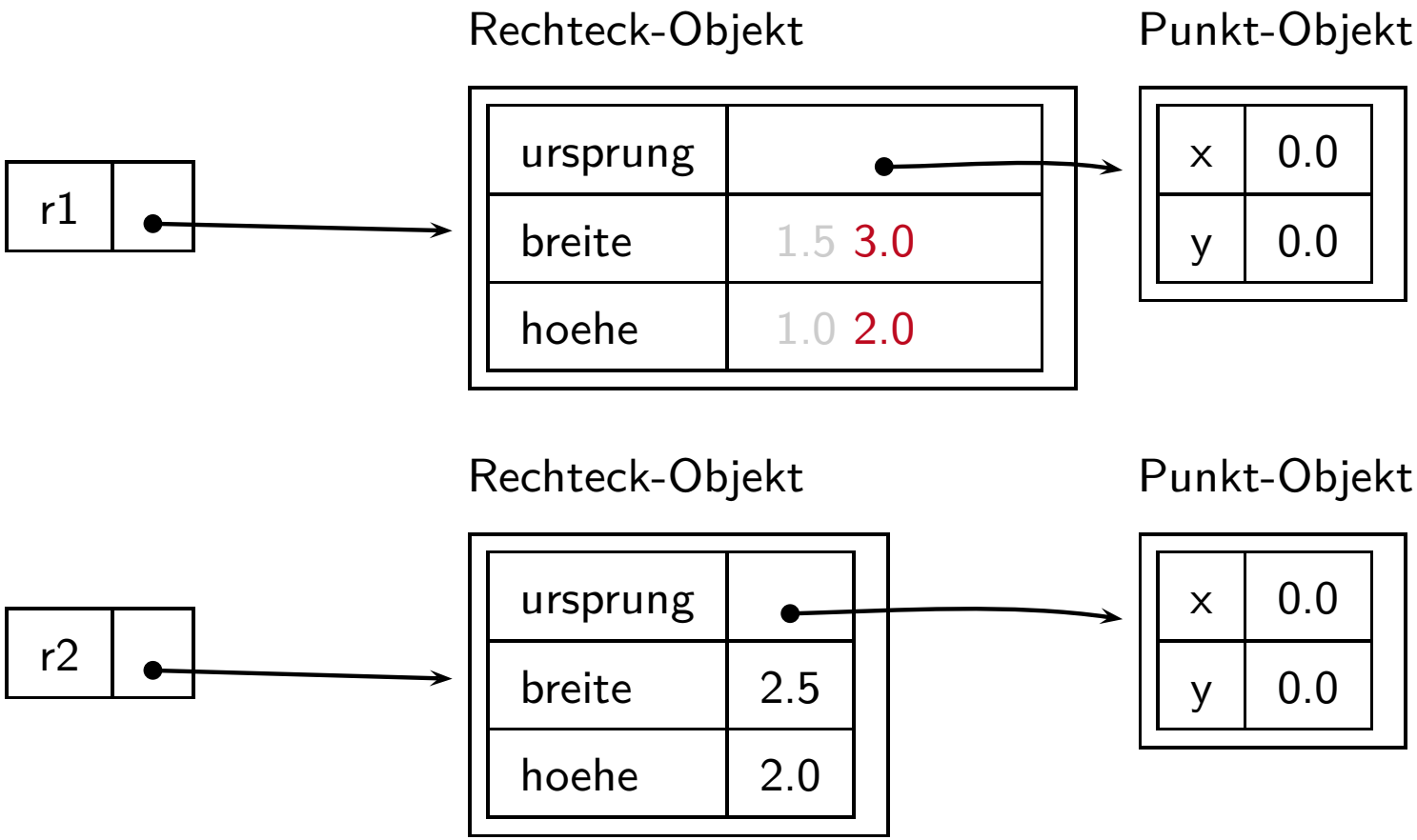
Fortsetzung des Beispiels




```
Rechteck r2 = new Rechteck(2.5, 2.0);
```

Zustand eines Objekts

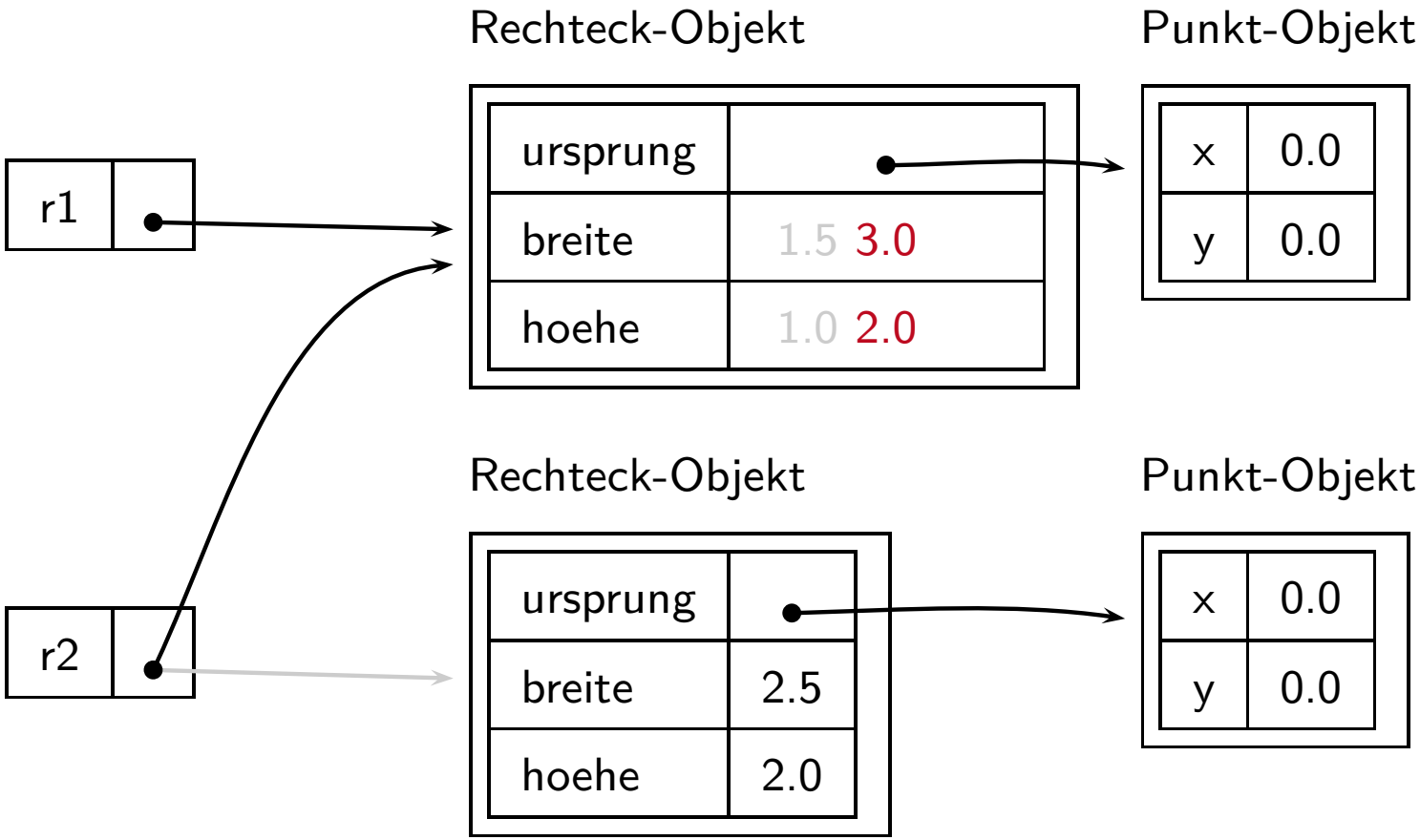
Fortsetzung des Beispiels




```
r1.vergroessere(2);
```

Zustand eines Objekts

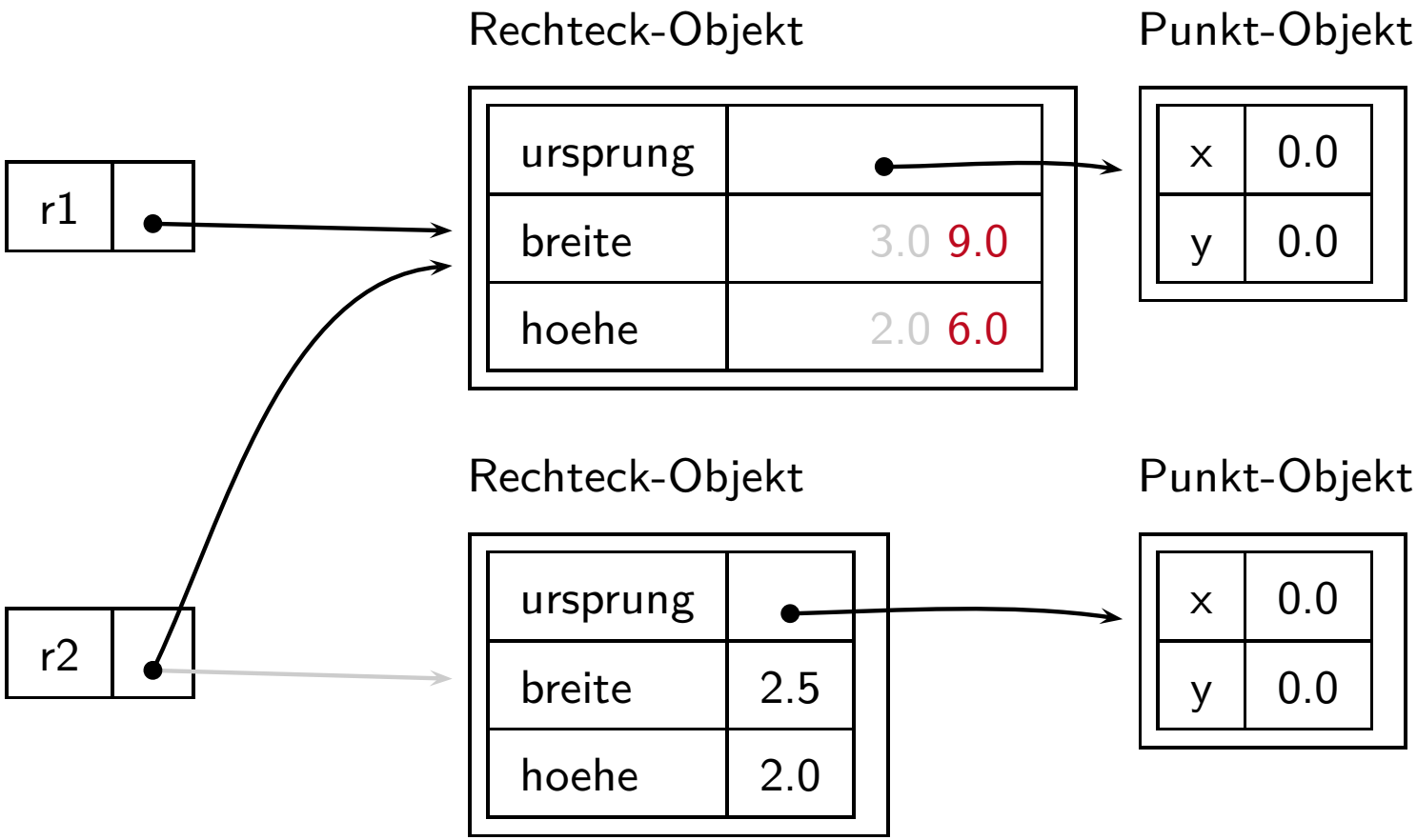
Fortsetzung des Beispiels




```
r2 = r1;
```

Zustand eines Objekts

Fortsetzung des Beispiels




```
r2.vergroessere(3);
```

Standardwerte von Instanzvariablen

- Instanzvariablen besitzen *Standardwerte* (z. B. 0 bei `int`-Variablen), sofern ihnen im Konstruktor keine Werte explizit zugewiesen werden.
- Dies unterscheidet Instanzvariablen von lokalen Variablen innerhalb von Methoden, die nach der Deklaration *keinen* definierten Wert enthalten.
- Guter Stil: Benötigt eine Instanzvariable einen bestimmten Anfangswert, weist man ihn im Konstruktor explizit zu, auch wenn er mit dem Standardwert übereinstimmt.

Standardwerte bei Referenz-Typen

- Der Standardwert von Instanzvariablen mit Referenz-Typ ist `null`.
- `null` zeigt an, dass eine Referenzvariable kein Objekt enthält. Der Wert ist nur für Referenzvariablen vorgesehen und kann nicht in einen primitiven Typ wie z. B. die ganze Zahl 0 umgewandelt werden.
- Die `null`-Referenz ist typenlos und kann jeder Referenzvariable zugewiesen werden. Beispiele:

```
Rechteck r = null;
Punkt p = null;
```

- Der Test, ob eine Referenzvariable `null` enthält, erfolgt mit dem Operator ==.

Modifikator `final`

Eine Instanzvariable kann mit dem Modifikator `final` deklariert werden.

Beispiel (Deklaration einer Instanzvariablen als `final`)

```
/** Ursprung, d.h. linke obere Ecke dieses Rechtecks. */
private final Punkt ursprung;
```

- Einer mit `final` deklarierten Instanzvariablen muss (guter Stil: im Konstruktor) ein Wert zugewiesen werden. Der Mechanismus der Standardwerte „greift nicht“.
- Eine zweite Zuweisung an eine finale Instanzvariable führt zu einem Compilefehler.
- Wenn es algorithmische Absicht ist, einer Variablen nur einmal einen Wert zuzuweisen, erhöht die Verwendung von `final` die Programmiersicherheit.

Klassenvariablen

- Neben *Instanzvariablen*, die sich auf einzelne Instanzen von Klassen beziehen und *einmal pro Instanz* existieren, erlaubt Java die Definition von Variablen, die sich auf die ganze Klasse und alle ihre Objekte gemeinsam beziehen.
- Die sogenannten *Klassenvariablen* werden mit dem Modifikator `static` deklariert (statische Variablen).
- Sie sind *nicht* Teil von Objekten, sondern Teil der *Klasse* und existieren genau *einmal*.
- Der Zugriff auf Klassenvariablen ist ohne die Existenz eines Objekts einer Klasse möglich.

Zugriff auf Klassenvariablen

- Der Zugriff auf eine Klassenvariable einer Klasse K erfolgt von außerhalb der Klasse K durch Qualifizierung mit dem Klassennamen.
- Innerhalb der Klasse K ist die Qualifizierung zulässig, aber nicht erforderlich.

Beispiele

```
System.out

Math.PI

Sparkonto.zinssatz  // der einheitliche Zinssatz
                    // aller Sparkonten
```

Symbolische Konstanten

Symbolische Konstanten sind *spezielle statische Variablen*, die zusätzlich mit dem Modifikator `final` deklariert sind.

Beispiele

```
public static final double PI = 3.1415927;
public static final int MAX_ANZAHL = 50;
```

- Der Modifikator `final` bewirkt, dass der Variablen *höchstens einmal* ein Wert zugewiesen werden kann. Dadurch ist der Wert der Variablen tatsächlich *nicht* variabel, sondern nach erstmaliger Wertzuweisung *konstant*.
- Bezeichnungen für Konstanten werden per Konvention durchgehend groß geschrieben mit _ als Worttrenner.

Verwendung symbolischer Konstanten

- Die Verwendung symbolischer Konstanten ist sinnvoll, um
  - die Bedeutung von Konstanten zu verdeutlichen,
  - Konsistenzprobleme zu vermeiden, wenn eine Konstante an mehreren Stellen im Quellcode verwendet wird.
- Symbolische Konstanten werden per Konvention (mit nur wenigen Ausnahmen) für alle konstanten Werte eines Programms definiert.
- Die Definition am Beginn einer Klasse zeigt auf einen Blick die Konstanten, von denen die Klasse abhängt.
- In *Testklassen* dürfen Konstanten direkt im Quellcode verwendet werden.

Klassenmethoden

- Klassenmethoden sind Methoden einer Klasse. Sie werden durch den Modifikator `static` deklariert. Statische Methoden (s. Kapitel 7) sind Klassenmethoden.
- Klassenmethoden beschreiben Funktionen, die nicht spezifisch für die individuellen Objekte einer Klasse sind.
- Der Zugriff auf Klassenmethoden ist ohne die Existenz eines Objekts einer Klasse möglich.
- Der Zugriff auf eine Klassenmethode einer Klasse K erfolgt von außerhalb der Klasse K durch Qualifizierung mit dem Klassennamen.
- Innerhalb der Klasse K ist die Qualifizierung zulässig, aber nicht erforderlich.
- Auch Klassenmethoden (statische Methoden) können den Rückgabetyp `void` besitzen.

Klassenmethoden und Instanzvariablen

- Eine Klassenmethode kann nicht direkt auf eine Instanzvariable der gleichen Klasse zugreifen.
- Instanzvariablen beschreiben die Struktur (den „Bauplan“) eines Objekts der Klasse. Klassenmethoden sind *unabhängig* von Objekten und haben deshalb keinen Zugriff auf diese Instanzvariablen.
- Der Versuch des Zugriffs führt zu einem Fehler beim Compilieren.

Instanz- und Klassenvariable

	Instanzvariable v der Klasse K	Klassenvariable v der Klasse K
Bedeutung	Beschreibt eine Eigenschaft der individuellen Objekte der Klasse K.	Beschreibt eine Eigenschaft der Klasse selbst; ist nicht Teil der Objekte der Klasse K.
Definition	ohne Modifikator `static`	mit Modifikator `static`
Wert	In jedem Objekt der Klasse K hat v einen individuellen Wert.	v hat genau einen Wert.
Zugriff	$o.v$; o ist Objektausdruck vom Typ K; zur Laufzeit ist Wert des Ausdrucks ein Objekt der Klasse K.	$K.v$, wenn Zugriff aus Methode einer anderen Klasse erfolgt.
	`this.`v oder v, wenn ein Objekt in einer Instanzmethode der Klasse K auf seine Variable v zugreift.	v, wenn Zugriff aus Methode derselben Klasse erfolgt (aber hier ist auch Qualifizierung mit K möglich).

Instanz- und Klassenmethode

	Instanzmethode m der Klasse K	Klassenmethode m der Klasse K
Bedeutung	Beschreibt Fähigkeit, die Objekte der Klasse K besitzen.	Beschreibt Funktion, die nicht spezifisch für Objekte der Klasse K ist.
Definition	ohne Modifikator `static`	mit Modifikator `static`
Aufruf	o`.`m`(`a_1`, ...,`a_n`)`; o ist Objektausdruck vom Typ K; zur Laufzeit ist Wert des Ausdrucks ein Objekt der Klasse K.	K`.`m`(`a_1`, ...,`a_n`)`, wenn Aufruf aus Methode einer anderen Klasse erfolgt.
	`this.`m`(`a_1`, ...,`a_n`)` oder `m(a1, ..., aN)`, wenn Objekt in Instanzmethode der Klasse K seine Methode m aufruft.	m`(`a_1`, ...,`a_n`)`, wenn Aufruf aus Methode derselben Klasse erfolgt (aber hier ist auch Qualifizierung des Aufrufs mit K möglich).
Bemerkung	In Methode m ist Zugriff auf Klassenvariablen der Klasse K möglich.	In Methode m ist Zugriff auf Instanzvariablen der Klasse K nicht möglich.
	In Methode m ist Aufruf von Klassenmethoden der Klasse K möglich.	In Methode m ist Aufruf `this.`m`(`a_1`, ...,`a_n`)` von Instanzmethoden der Klasse K nicht möglich.