

KLASSENMODELLIERUNG 1: GRUNDLAGEN

Eigene Klassen mit Variablen und Methoden modellieren, erstellen und verwenden

Struktur und Inhalt des Kurses wurden 2012 von Markus Heckner entwickelt. Im Anschluss haben Alexander Bazo und Christian Wolff Änderungen am Material vorgenommen. Die aktuellen Folien wurden von Alexander Bazo erstellt und können unter der [MIT-Lizenz](#) verwendet werden.

AKTUELLER SEMESTERFORTSCHRITT (WOCHE 6)

Kursabschnitt	Themen		
Grundlagen	Einführung	Einfache Programme erstellen	Variablen, Klassen & Objekte
	Kontrollstrukturen & Methoden	Arrays & komplexe Schleife	
Klassenmodellierung	Grundlagen der Klassenmodellierung		
Interaktive Anwendungen	Event-basierten Programmierung		
Datenstrukturen	Listen, Maps & die Collections		
Software Engineering	Planhaftes Vorgehen bei der Softwareentwicklung		
	Qualitätsaspekte von Quellcode		

PINGO-QUIZ



<https://pingo.coactum.de/368355> 

DAS PROGRAMM FÜR HEUTE

- 5-Minuten-Thema: Texte in Variablen speichern (*Strings*)
- Das Prinzip des *Information Hidings*
- Klassen modellieren, erstellen, verwenden und dokumentieren
- Objekte als Parameter verwenden

TEXT IN VARIABLEN SPEICHERN: STRINGS

BEKANNTEN DATENTYPEN

Bereich	Datentypen	Beispiele Wertebereich	Code-Beispiele (Variablen)
Ganzzahlen	int	-2,-1,0,1,2	int x = 42;
Fließkommazahlen	double, float	-2.1,-1.2,0.0,1.7,2.2	double x = 13.37;
Einzelne Zeichen	char	a,-,!	char x = x;
Wahrheitswerte	boolean	true, false	boolean x = false;
(Graphische) Objekte	jeweiliger Klassenname	Circle, Label,...	Circle x = new Circle(50,50,100,Colors.RED);

Häufig wollen wir aber auch Text(e) in unseren Anwendung verwenden!

TEXTE MIT *STRINGS* ABBILDEN

```
String myName = "Alexander Bazo";
```

Strings sind Objekte, können aber - unter Verwendung von Anführungsstrichen - als sogenanntes Literal und ohne new-Schlüsselwort erzeugt werden.

Fun Fact: Intern wird für die Speicherung des Textes in der String-Klasse ein Array aus einzelnen char-Elementen verwendet.

Hinweis: In Java (und vielen anderen Programmiersprachen) ist ein Literal ein unveränderbarer Ausdruck (en. *literally* ~ *wortwörtlich*). Jeder fest codierte numerische Wert, true & false sowie jedes im Code notierte Zeichen bzw. jede Zeichenkette ist ein solches, unveränderbares Literal mit einer festen Bedeutung.

GRUNDLAGEN DER ARBEIT MIT *STRINGS*

```
1 // Strings erzeugen und in Variablen speichern
2 String myName = "Alexander Bazo";
3
4 // Länge (Anzahl der Zeichen) eines Strings bestimmten
5 int numberOfCharacters = myName.length();
6
7 // Strings ausgeben
8 System.out.println(myName)
9
10 // Strings verbinden
11 String firstName = "Alexander";
12 String lastName = "Bazo";
13 String fullName = firstName + " " + lastName;
14
15 // Einzelne Bestandteile des Textes auslesen
16 for (int i = 0; i < myName.length(); i++) {
17     char singleChar = myName.charAt(i);
18     System.out.println("[ " + i + "]: " + singleChar);
19 }
```

STRINGS EINLESEN

Strings können - in Kommandozeilen-basierten Anwendungen - über die Scanner-Klasse eingelesen werden:

```
1 // Erzeugt einen Scanner, der von der Standardeingabe (in der Regel die Kommandozeile) liest
2 Scanner scanner = new Scanner(System.in);
3 /**
4  * Gibt die nächste Eingabe des Nutzenden als String zurück, d.h. Sie erhaltenen alle
5  * eingegebenen Zeichen bis zum nächsten Zeilenumbruch (Enter-Taste).
6  * Achtung: Ihr Programm hängt hier fest, bis der Nutzende die Eingabe beendet hat
7  * (Enter-Taste gedrückt hat). Erst dann wird die nächste Codezeile ausgeführt.
8 */
9 String input = scanner.nextLine();
```

Hinweis: Die Scanner-Klasse bietet weitere Methoden an, über die auch Zahlen oder Wahrheitswerte aus der ausgewählten Eingabequelle eingelesen werden können (Vgl. [Orcacles Java-Dokumentation](#)).

EINE WIEDERVERWENDBARE METHODE FÜR DAS EINLESEN VON TEXT

```
1
2 /**
3  * Die Methode gibt den als Parameter übergebenen Text als Hinweis an
4  * an den Nutzenden aus und liefert dessen Eingabe/Antwort zurück.
5 */
6 private String readString(String prompt) {
7     System.out.print(prompt);
8     Scanner scanner = new Scanner(System.in);
9     return scanner.nextLine();
10 }
11
12 private void readName() {
13     String myName = readString("Bitte geben Sie Ihren Namen ein: ");
14 }
```

VORGEGEBENE KLASSEN VERWENDEN

CLIENT VS. IMPLEMENTOR

Hinweis: Im Rahmen der **Bereitstellung und Verwendung** von Klasse unterscheiden wir zwischen den Personen, die Klassen modellieren und entwickeln und denjenigen, die diese Klassen in eigenen Programmen verwenden.

Clients nutzen bestehende Klassen in eigenen Programmen. Wir haben als NutzerInnen die vorgegebenen Klassen der *GraphicsApp*-Umgebung, z.B. *Circle*, *Label* oder *Rectangle* verwendet.

Implementors stellen Klassen für *Clients* bereit. Die von uns verwendeten Klassen der *GraphicsApp*-Umgebung wurden für uns bereitgestellt.

Die Rollen sind nicht trennscharf, in der Regel sind wir *Client* und *Implementor*, da wir auf der Grundlage vorgegebener Klassen eigene Programme/Klassen entwerfen. Die Klassifizierung dient eher der Beschreibung der Autorenschaft einer einzelnen Klasse.

INFORMATION HIDING

Im Verhältnis zwischen *Client* und *Implementor* spielt das Prinzip des *Information Hiding* eine wesentliche Rolle:

- Als *Client* interessiert es uns nicht, wie `Math.random()` funktioniert oder was in `Ellipse.draw()` passiert. Wir brauchen/sehen nur das Ergebnis.
- Die eigentliche Implementierung (*Was passiert?*) wird (bewusst) vor dem *Client* versteckt. Dieser muss nur die Methodensignatur, d.h. Parameter & Rückgabewert kennen um zu wissen, was möglich ist.
- Der *Implementor* muss diese relevanten Informationen zugänglich dokumentieren (Vgl. Dokumentation der `GraphicsApp`

INFORMATION HIDING IN DER RANDOM-KLASSE (1/2)

Die Klasse Random stellt für uns als *Clients* eine *black box* dar. Die internen Abläufe sind nicht relevant, solange die bekannten Schnittstellen (öffentliche Methoden) die erwarteten Ergebnisse zurückgeben:

Input	Zufallsgenerator	Rückgabe
Methoden für die gewünschten Datentypen, z.B. nextInt()	Von uns erstellte Instanz der Random-Klasse	Zufälliger Rückgabewert vom jeweils angefragten Datentypen.

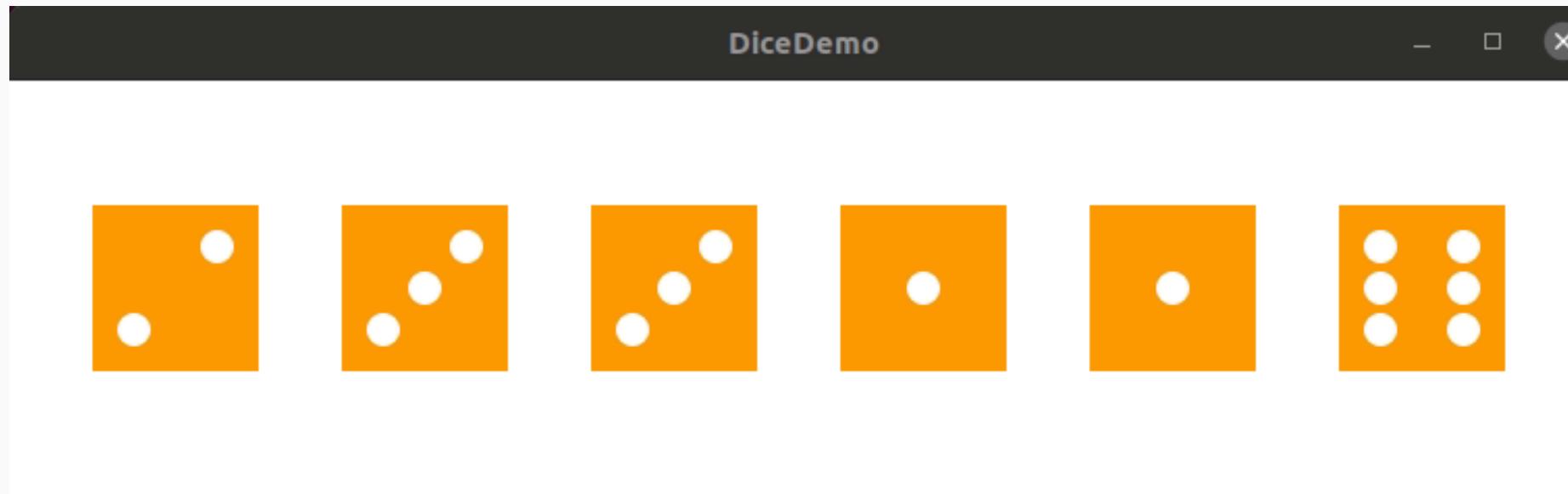
INFORMATION HIDING IN DER RANDOM-KLASSE (2/2)

Einige Methoden der Random-Klasse:

Rückgabetyp	Methode	Beschreibung
boolean	nextBoolean()	Returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.
double	nextDouble()	Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.
int	nextInt(int bound)	Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

Quelle: Java 10 Dokumentation, Oracle

BEISPIEL: EIN ZUFALLSGENERATOR ZUM WÜRFELN



Hinweis: Wie verwenden als *Client* die Methoden der Random-Klasse um sechs zufällige Werte zwischen 1 und 6 zu berechnen, die in Form von *Würfeln* auf dem Bildschirm dargestellt werden.

BEISPIEL: SIMULATION VON N-WÜRFEN

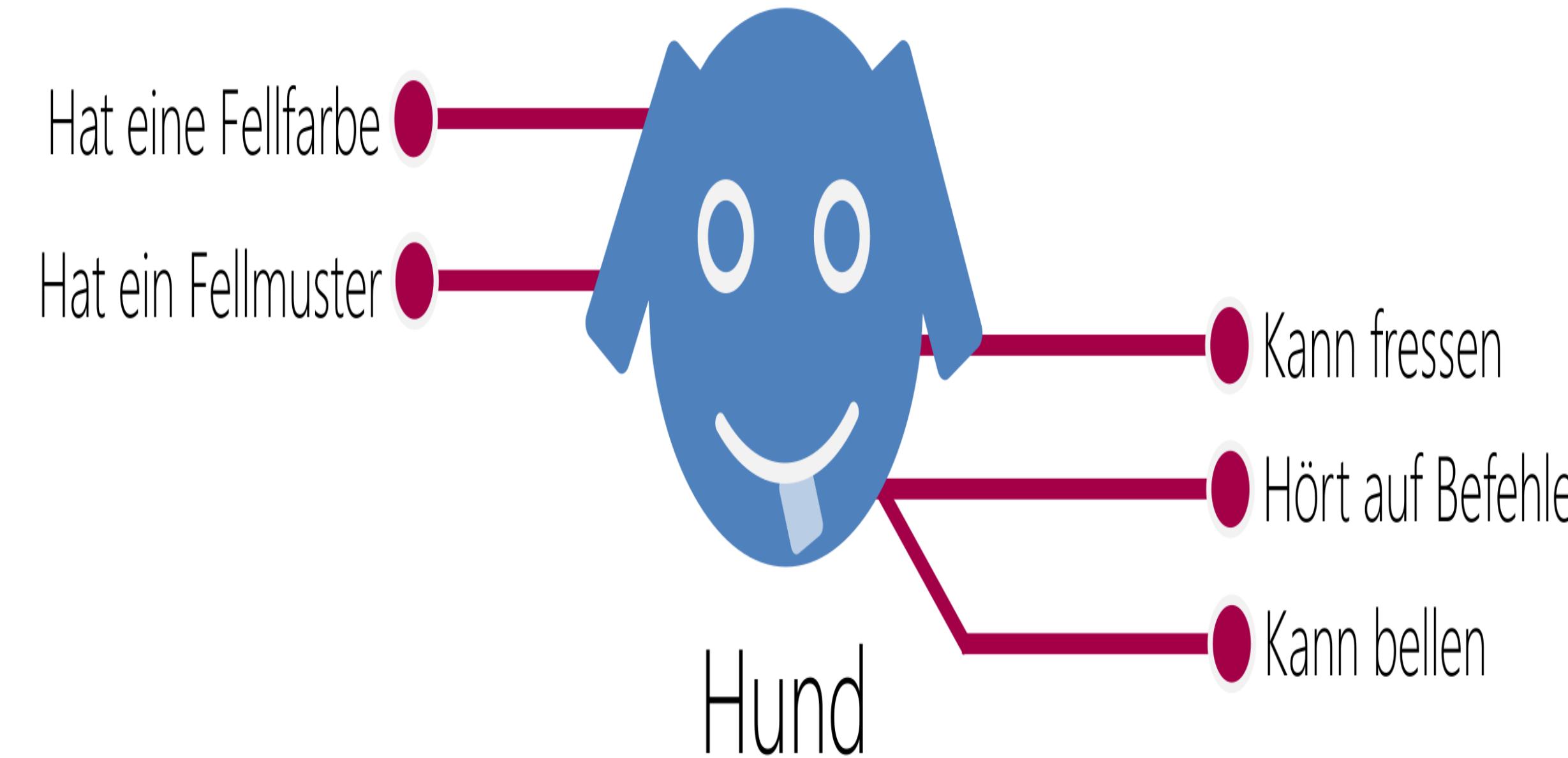
```
1 private void rollDice() {  
2     Random random = new Random();  
3     int numberOfDice = readInt("Mit wie vielen Würfeln soll ich würfeln? ");  
4     int result = 0;  
5     int maxResult = numberOfDice * MAX_DICE_VALUE;  
6     int numberofTries = 0;  
7     while (result != maxResult) {  
8         result = 0;  
9         for(int i = 0; i < numberOfDice; i++) {  
10             int randomDiceValue = MIN_DICE_VALUE + random.nextInt(MAX_DICE_VALUE);  
11             result += randomDiceValue;  
12         }  
13         numberofTries++;  
14     }  
15     System.out.println("Das maximale Ergebnis (" + maxResult + ") wurde nach " + \  
16     numberofTries + " Versuchen erreicht!");  
17 }
```

EIGENE KLASSEN ENTWERFEN UND VERWENDEN

WIEDERHOLUNG: OBJEKTE UND KLASSEN (1/3)

- Ein Objekt ist eine Datenstruktur mit einem bestimmten Zustand und Verhalten.
- Eine Klasse ist eine Vorlage, die eine gemeinsame Struktur für alle Objekte (Instanzen) dieser Klasse definiert.
- Jedes Objekt ist eine Instanz einer bestimmten Klasse, eine Klasse kann als Vorlage für viele Instanzen dienen.
- Klassen sind in Hierarchien aus Subklassen und Superklassen organisiert.

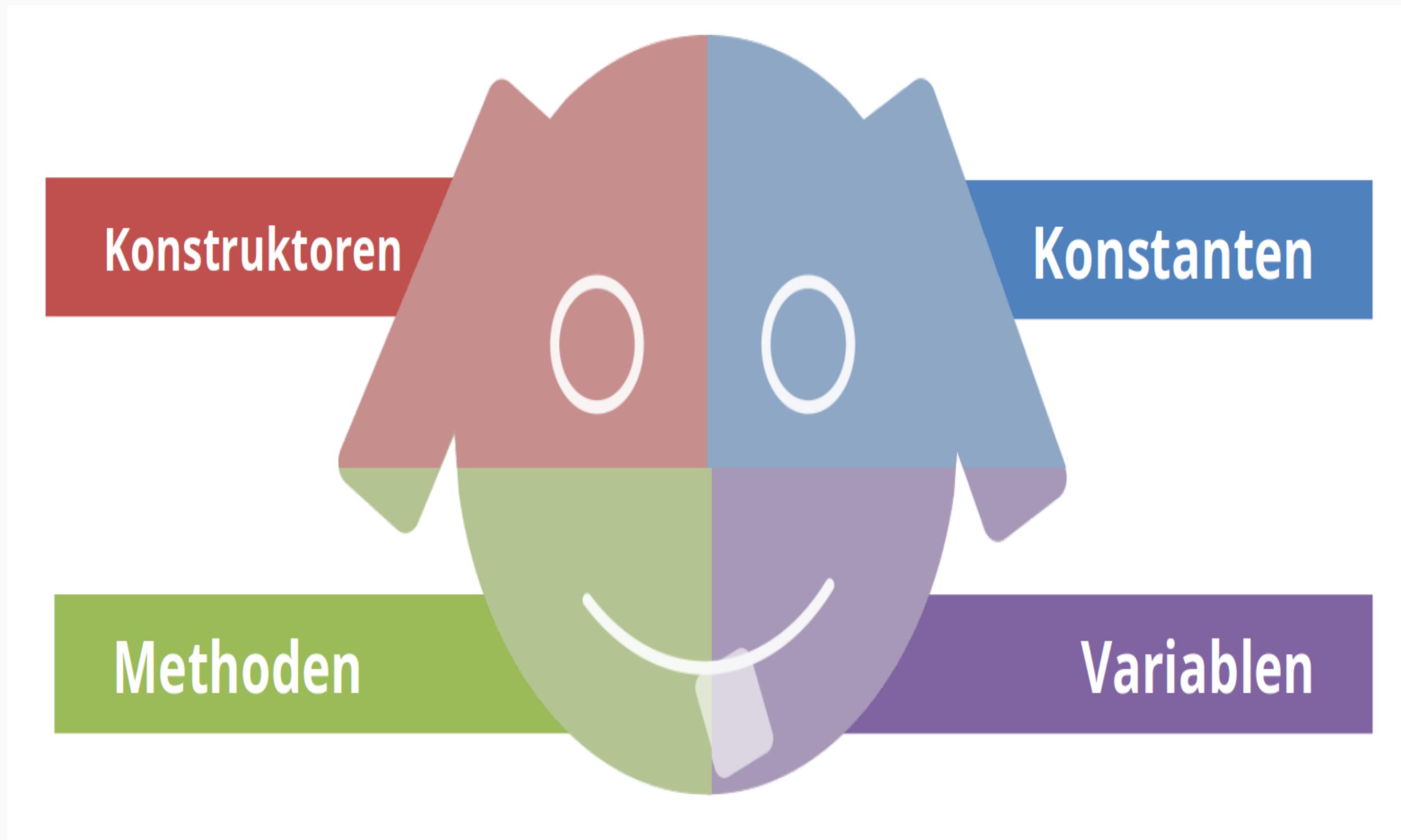
WIEDERHOLUNG: OBJEKTE UND KLASSEN (2/3)



WIEDERHOLUNG: OBJEKTE UND KLASSEN (3/3)



ZENTRALE BESTANDTEILE EINER KLASSE



EIGENE KLASSEN DEFINIEREN (1/2)

Objekterzeugung (Konstruktor): Die individuelle Fellfarbe des Hundes wird festgelegt.

Verhalten (Methoden): (Öffentliche) Methoden, die das Verhalten aller Hunde definieren (*Fressen, Bellen, Apportieren*).

Zustand (Instanzvariablen): (Öffentliche) Variablen, die individuelle Zustände für Eigenschaften abbilden, die alle Hunde haben (*Name, Hunger*).

Kurzfristige Zustände (Lokale Variablen in Methoden): Variablen, die zwischenzeitliche Ergebnisse/Zustände abbilden, z.B. ob ein Hund beim Apportieren einen Stock im Mund hat.

EIGENE KLASSEN DEFINIEREN (2/2)

Hinweis: Klassen werden in separaten Dateien definiert, deren Name identisch mit dem Namen der Klasse ist. Die Dateien haben die Endung .java.

```
1 // Dieser Code greift den folgenden Folien vorweg!
2 public class Dog {
3     private Color furColor;
4     private boolean isHungry;
5
6     public Dog(Color furColor) {
7         this.furColor = furColor;
8         isHungry = true;
9     }
10
11    public void eat() {
12        if(isHungry == true) {
13            isHungry = false;
14        }
15    }
16 }
```

SICHTBARKEIT VON KLASSENBESTANDTEILEN

Die Sichtbarkeit bzw. die Zugriffsmöglichkeiten aller Bestandteile einer Klasse können über Zugriffsmodifikatoren spezifiziert werden. Diese geben an, von wo auf die entsprechenden Elemente zugegriffen werden kann. Die beiden wichtigsten Modifikatoren sind:

- **public:** Andere Teile des Programms (Klassen) haben Zugriff auf die Methoden oder Variablen
- **private:** Die Methoden und Variablen sind nur innerhalb der Klasse zugänglich

```
1 Random random = new Random();
2 /**
3  * nextInt muss eine öffentliche (public) Methode sein, da wir von Außen (aus unserer Klasse)
4  * auf die Methode der erzeugten Instanz zugreifen können.
5 */
6 int x = random.nextInt();
```

INSTANZVARIABLEN

- Anders als lokale Variablen werden Instanzvariablen zu Beginn der Klasse (und nicht in Methoden) definiert.
- Instanzvariablen sind innerhalb des gesamten Objekts sichtbar und *leben* solange das Objekt *lebt*. Sie sind innerhalb der ganzen Instanz verwendbar.
- Instanzvariablen repräsentieren die Eigenschaften bzw. den Zustand des Objekts. Auch wenn die Eigenschaften nur einmal in der Klasse definiert werden, verfügt jedes Objekt über sein eigenes Set an Instanzvariablen, d.h. jede Dog-Instanz hat eine eigene Variable (`furColor`) für die Fellfarbe.

KONSTRUKTOREN (1/2)

- Konstruktoren sind besondere Methoden, über die Objekte/Instanzen einer Klasse erzeugt werden
- Über Konstruktoren werden in der Regel initiale Werte für die Eigenschaften eines neu zu erstellenden Objekts gesetzt

KONSTRUKTOREN (2/2)

Der Name des Konstruktors ist identisch zum Klassennamen. Bei Konstruktoren wird kein expliziter Rückgabewert (auch kein void) angegeben. Ein neues Objekt wird über den Aufruf eines Konstruktors mit dem new-Schlüsselwort erzeugt. Innerhalb einer Klasse können unterschiedliche Konstruktoren mit verschiedenen Parametern definiert werden (Vgl. graphische Objekte in der *GraphicsApp*).

```
1 public class Dog {  
2     private Color furColor;  
3     private boolean hungry;  
4  
5     public Dog(Color color) {  
6         furColor = color;  
7         hungry = true;  
8     }  
9 }
```

Aus unserer Dog-Klasse kann über den Aufruf des Konstruktors `new Dog(Colors.BLACK)` eine konkrete Dog-Instanz erzeugt werden.

DAS SCHLÜSSELWORT **this**

Innerhalb einer Klasse kann an jeder Stelle über das Schlüsselwort **this** auf die aktuelle Instanz verwiesen werden.

Oft wird vom **this**-Zeiger gesprochen, was technisch aber nicht völlig korrekt ist, da Java kein explizites Zeiger-Konzept unterstützt. Präzise müsste es **this**-Referenz heißen. Die **this**-Referenz können wir z.B. nutzen, um bei Namensgleichheit Parameter von Instanzvariablen zu unterscheiden:

```
1 private Color furColor;
2 private boolean hungry;
3
4 public Dog(Color furColor) {
5     this.furColor = furColor;
6     hungry = true;
7 }
```

STATISCHE VARIABLEN UND KONSTANTEN

- Statische Variablen (Klassenvariablen) werden von allen Instanzen einer Klasse geteilt (!=Instanzvariablen):
`private static boolean dogCatcherSpotted`
- Konstanten werden ebenfalls als Klassenvariablen angelegt und sind darüber hinaus unveränderbar:
`private static final String BEST_FRIEND = "Human";`
- Öffentliche, statische Variablen und Methoden können auch ohne Instanz – direkt auf der Klasse – aufgerufen werden:
`Colors.getRandomColor();`

ÖFFENTLICHE UND PRIVATE METHODEN (2/2)

Methoden beschreiben das interne Verhalten von Klassen (private) und bilden die Schnittstelle zu anderen Programmbestandteilen (public).

Innerhalb von Methoden kann auf die Instanzvariablen und andere Methoden zugegriffen werden, und es können lokale Variablen definiert werden. Eine öffentliche Methode kann private/interne Methoden aufrufen.

Dekomposition ist also auch hier möglich und notwendig.

Auf der nächsten Folie: Methoden für unsere Dog-Klasse.

ÖFFENTLICHE UND PRIVATE METHODEN (2/2)

```
1 public void fetch() {  
2     boolean hasStick = false;  
3     runToStick();  
4     while(!hasStick) {  
5         hasStick = tryToPickUpStick();  
6     }  
7     runToOwner();  
8 }  
9  
10 private void runToStick() {  
11     // ...  
12 }  
13  
14 private void runToOwner() {  
15     // ...  
16 }  
17  
18 private boolean tryToPickUpStick() {  
19     return RandomGenerator.getInstance().nextBoolean();  
20 }
```

BESONDERE METHODEN: GETTER & SETTER

- Instanzvariablen sollten in der Regel versteckt (`private`) sein.
Dadurch wird das Prinzip des *Information Hiding* gewahrt.
- Wenn doch ein lesender/schreibender Zugriff von Außen nötig sein sollte, geschieht diese über Methoden, die den eigentlichen Zugriff auf die Daten kapseln: *Getter & Setter*.

```
1 public Color getFurColor() {  
2     return furColor;  
3 }  
4  
5 public void setOwner(String owner) {  
6     if(!owner.equals("Cruella de Ville")) {  
7         this.owner = owner;  
8     }  
9 }
```

PRAKTISCHE BEISPIELE

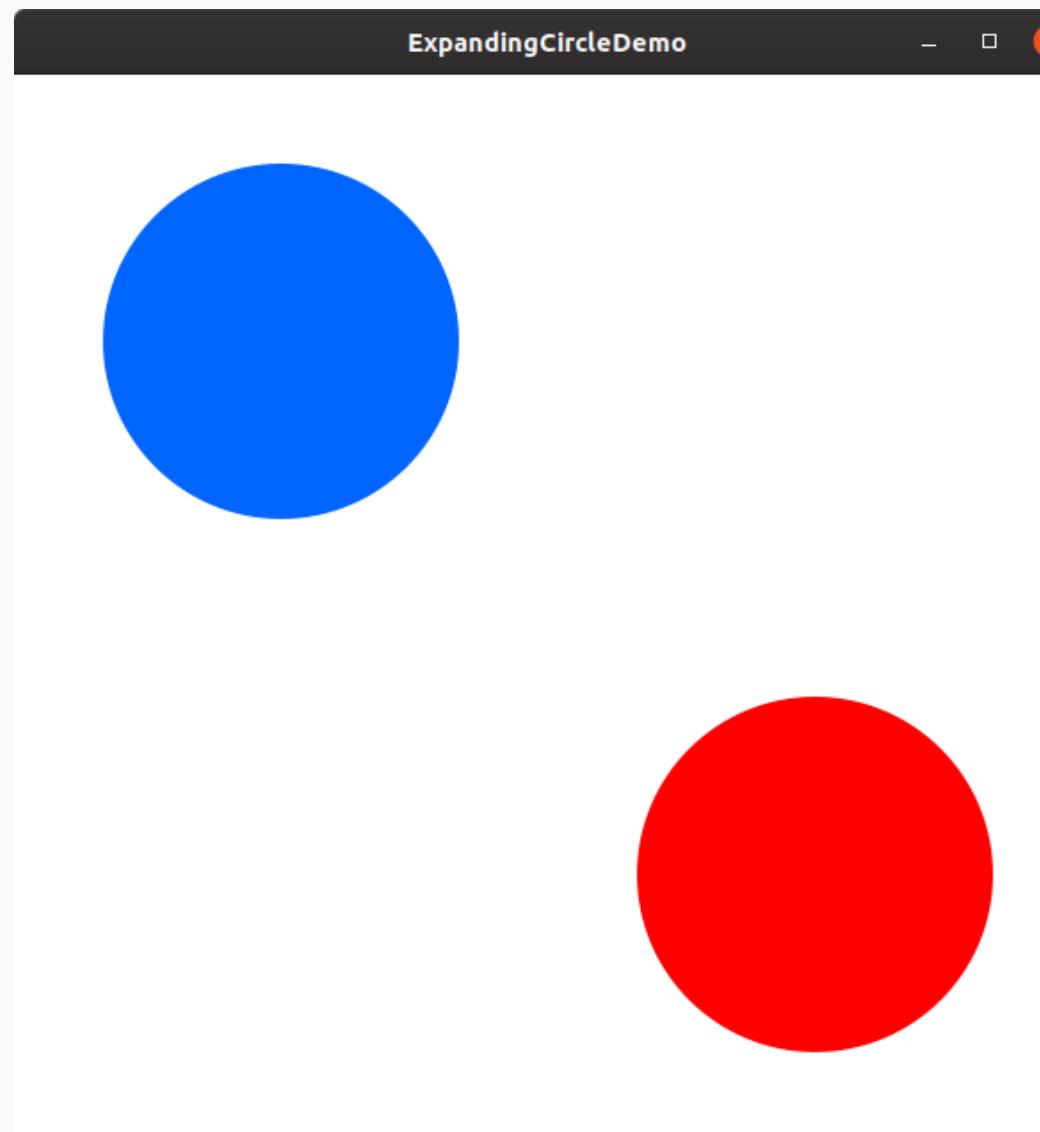
BEISPIEL: STUDENT & STUDENTSAPP

```
Run: StudentsAppDemo
/opt/intellij/idea-IU-192.6817.14/jbr/bin/java -javaagent:/opt/int
Bitte gebe die Informationen für den nächsten Studierenden ein.
Name: A Student
Note: 2,3
Bitte gebe die Informationen für den nächsten Studierenden ein.
Name: B Student
Note: 1,7
Bitte gebe die Informationen für den nächsten Studierenden ein.
Name: C Student
Note: 3,0
#1: A Student (2.3)
#2: B Student (1.7)
#3: C Student (3.0)

Process finished with exit code 0
```

Eine Klasse Student mit Name, Leistungspunkten und Note. Jeder Studierende verfügt über eine fortlaufend vergebene ID. Die Eigenschaften werden über den Konstruktor gesetzt. *Getter*-Methoden erlauben den lesenden Zugriff. LPs können über eine öffentliche Methode ergänzt werden. Die Anwendung speichert die durch Eingabe der Nutzenden erzeugten Studierenden in einem Array.

BEISPIEL: EXPANDING CIRCLE



Eine Klasse `ExpandingCircle` kapselt ein sich ausdehnendes `Circle`-Objekt. Im Konstruktor werden Position, initiale Größe und die Farbe des Kreises übergeben. Über die öffentlichen Methoden `update` und `draw` kann der Kreis vergrößert und gezeichnet werden. Die Anpassung der Größe erfolgt schrittweise bis zu einem definierten Maximalwert. In einer *GraphicsApp* werden zwei Instanzen der `ExpandingCircle`-Klasse erstellt und verwendet.

OBJEKTE ALS PARAMETER

CAVEAT: OBJEKTE ALS PARAMETER

Objekte verhalten sich anders als primitive Datentypen, wenn Sie als (Übergabe-)Parameter eingesetzt werden:

```
1 private void modifyStudent(Student student, int lp) {  
2     System.out.println("Modifying Students LPs");  
3     student.addLP(lp);  
4 }
```

Hinweis: Objekte werden **nicht** als Kopien weitergegeben sondern als Verweise (Referenzen) auf die *Originale*. Wenn wir ein Objekt (Parameter) in einer Methode ändern, ändern wir das Original. Technisch gesehen wird auhc hier konkreter Wert, nämlich die "Adresse" des übergebenen, Java arbeitet immer mit dem *Call by Value*-Prinzip.

KONSEQUENZEN DES *CALL BY VALUE*-ANSATZES IN JAVA (1/2)

- Wird ein Objekt an eine Methode übergeben, befindet sich in der Parameter-Variable die Adresse (im Speicher) an der das Objekt bzw. seine Methoden (nicht wirklich) und Eigenschaften gespeichert (die schon) werden.
- Wenn wir die Eigenschaften des Objekts über die Parameter-Variable ändern, ändern wir das ursprüngliche Objekt.
- Wenn wir die Parameter-Variable überschreiben, bleibt das ursprüngliche unberührt (wir ändern dann nur einen von mehreren der Orte, an denen die Adresse des Objekts gespeichert ist).

KONSEQUENZEN DES *CALL BY VALUE*-ANSATZES IN JAVA (2/2)

```
1 /**
2  * LPs werden für das äußere Objekt gesetzt, die Initialisierung des neuen Objekts
3  * ist aber auf den Scope beschränkt.
4 */
5 private void modifyStudent(Student student, int lp) {
6     System.out.println("Modifying Students LPs");
7     student.addLP(lp);
8     student = new Student("A new User", 4.0);
9 }
```

Hinweis: Wir greifen dieses Thema im späteren Verlauf der Vorlesung beim Thema *Speicherverwaltung* noch einmal auf.

KLASSEN MIT JAVADOC KOMMENTIEREN UND DOKUMENTIEREN

KOMMENTARE IN JAVA

Kommentare dienen der zusätzlichen Beschreibung von Quelltext. Sie haben keine funktionale sondern eine dokumentierende Bedeutung:

```
1
2 /**
3  * Diese Methode erzeugt ein zufälliges Circle-Objekte (zufällige Position und Farbe) und
4  * gibt die Instanz als Rückgabe wert zurück.
5 */
6 public Circle createRandomCircle() {
7     // createRandomNumber gibt eine zufällige Ganzzahl zwischen 0 und dem übergebenen
8     // Parameter zurück
9     int xPos = createRandomNumber(SCREEN_WIDTH);
10    int yPos = createRandomNumber(SCREEN_HEIGHT);
11    Color circleColor = Colors.getRandomcolors();
12    // Erzeugen des Circle-Instanz (CIRCLE_RADIUS ist eine Konstante auf Klassenebene)
13    Circle circle = new Circle(xPos, yPos, CIRCLE_RADIUS, circleColor);
14    return circle;
15 }
```

KOMMENTARE MIT JAVADOC STRUKTURIEREN

```
1 /**
2  * Diese Methode erzeugt ein zufälliges Circle-Objekt innerhalb der
3  * Zeichenfläche, dessen Radius im übergebenen Wertebereich liegt.
4  *
5  * @param minRadius Minimaler Radius des zu erstellenden Kreises (inklusive)
6  * @param maxRadius Minimaler Radius des zu erstellenden Kreises (exklusive)
7  * @return Die erzeugte Circle-Instanz
8 */
9 public Circle createRandomCircleWithColor(int minRadius, int maxRadius) {
10     int xPos = createRandomNumber(SCREEN_WIDTH);
11     int yPos = createRandomNumber(SCREEN_HEIGHT);
12     int radius = minRadius + createRandomNumber(maxRadius - minRadius);
13     return new Circle(xPos, yPos, radius, Colors.getRandomcolors());
14 }
```

Hinweis: Das **JavaDoc-Format** erlaubt die gezielte Beschreibung einzelner Code-Bestandteile (z.B. Parameter oder Rückgabewerte). Auf Basis der Kommentare kann eine Übersicht (Dokumentation) des Programms erstellt werden kann (Vgl.: **GraphicsApp-Dokumentation** .

ZUSAMMENFASSUNG

- Strings sind Zeichenketten und können mit anderen Variablen konkateniert (= zusammengefügt, lat. catena = Kette) werden.
- Aus Sicht des Client-Programmierers ist es unerheblich, wie eine Klasse oder Methode implementiert wird.
- Klassen beinhalten typischerweise die folgenden Komponenten: Instanzvariablen, Konstanten, Konstruktoren, Methoden.
- Methoden und Variablen sind als private zu definieren, außer man wünscht explizit Zugriff von außerhalb des Objekts.
- Eigene Klassen sollten gut dokumentiert werden. JavaDoc ermöglicht automatisierte Erstellung von html-Dokumentation eigener Klassen.
- Objekte verhalten sich anders als primitive Datentypen, wenn Sie als Parameter von Methoden verwendet werden.

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT. WENN SIE MÖCHTEN, SEHEN WIR UNS IM ANSCHLUSS IN DER ZENTRALÜBUNG!

Fragen oder Probleme? In allgemeinen Angelegenheiten und bei Fragen zur Vorlesung wenden Sie sich bitte an Alexander Bazo (alexander.bazo@ur.de). Bei organisatorischen Fragen zu den Studienleistungen und Übungsgruppen schreiben Sie bitte Florin Schwappach (florin.schwappach@ur.de). Bei inhaltlichen Fragen zu den Übungen, Beispielen und Studienleistungen schreiben Sie uns unter mioop@mailman.uni-regensburg.de.

QUELLEN

Eric S. Roberts, *The art and science of Java: an introduction to computer science, New international Edition*, 1. Ausgabe, Pearson, Harlow, UK, 2014.