

# VARIABLEN UND OBJEKTE

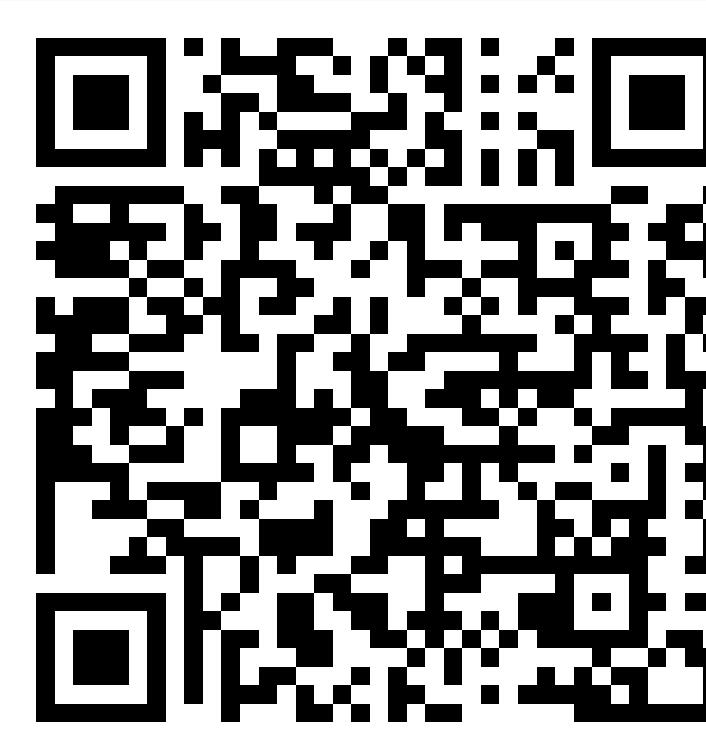
**Speichern und manipulieren von primitiven Datentypen und die  
Verwendung vorgegebener Klassen**

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 3)

Kursabschnitt	Themen		
Grundlagen	Einführung	Einfache Programme erstellen	
	Variablen, Klassen & Objekte	Kontrollstrukturen & Methoden	
	Arrays & komplexe Schleife		
Klassenmodellierung	Grundlagen der Klassenmodellierung	Vererbung & Sichtbarkeit	
Interaktive Anwendungen	Event-basierten Programmierung	String- & Textverarbeitung	
Datenstrukturen	Listen, Maps & die Collections	Speicherverwaltung	
	Umgang mit Dateien		
Software Engineering	Planhaftes Vorgehen bei der Softwareentwicklung		

# PINGO-QUIZ



<https://pingo.coactum.de/454414> ↗

## RÜCKBLICK

- Java ist eine plattformunabhängig Programmiersprache: Zum Ausführen von Java-Programmen wird eine plattformspezifische Laufzeitumgebung benötigt
- Programme unterscheiden sich in der Entwicklungsphase (Quellcode) und der Ausführungsphase
- Klassen sind das zentrale Strukturelement eines Java-Programms und bei Programmstart, wird immer die essentielle `main`-Methode ausgeführt
- *Decomposition* und die *Top Down*-Analyse sind zentrale Konzepte, um Programmierprobleme strukturiert zu lösen und übersichtlichen und wartbaren Code zu erstellen

# DAS PROGRAMM FÜR HEUTE

- Einführung in die Verwendung von Variablen und mathematischen Operationen
- Erste Informationen zu Klassen und Objekte (das wird in der nächsten Woche fortgesetzt)
- Zentrale Funktionen und grundlegende Verwendung der *Graphics App*

# (ZWISCHEN-) SPEICHERN VON WERTEN IN VARIABLEN

## EINIGE DINGE, DIE WIR AKTUELL NOCH NICHT KÖNNEN

- Beim Arbeiten mit *Bouncer*, die Häufigkeit bestimmter Ereignisse, z.B. die Anzahl *roter* Felder auf einer Straße bestimmen
- Mathematische Berechnungen durchführen und deren Ergebnisse im späteren Programmverlauf wieder verwerten

Für diese oder ähnliche Aufgaben müssen wir uns innerhalb unserer Programme **Dinge merken**. In den meisten Programmiersprachen werden für diesen Zweck **Variablen** eingesetzt. So auch in Java!

# VARIABLEN

Eine Variable ist ein Ort (oder Schachtel, *Label*, Flasche, ...), an dem ein Programm Informationen für die spätere Nutzung speichern kann – es handelt sich um einen benannten, typisierten Speicherplatz, in dem Sie Werte abspeichern und abrufen können. In Java bestehen Variablen immer aus drei zusammengehörigen Informationseinheiten:

TYP	NAME	WERT
Welche Art von Information wird in der Variable abgelegt?	Über welchen Namen wird die Variable im Code angesprochen?	Welchen Wert hat die Variable zu einem bestimmten Zeitpunkt?

Im Quellcode schaut das dann z.B. so aus:

```
int numPlayers = 11;
```

int steht für den Typ (hier eine Ganzzahl), numPlayers für den vergebenen Namen und 11 ist der Wert, der in der erstellten Variable gespeichert wird.

## VARIABLEN: TYPEN UND WERTE (1/2)

In Java hat jede Variable einen bestimmten Typ, der vorgibt, welche Art von Information gespeichert werden kann. Es existieren unterschiedliche Typen für unterschiedliche Informationsarten. Dabei wird zwischen primitiven und komplexen Typen (oder Referenzdatentypen) unterschieden.

**Typ und Wert einer Variable müssen zusammen passen. Wenn nicht, passieren im besten Fall ungewollte Dinge oder im schlimmsten Fall stürzt Ihr Programm ab!**

## VARIABLEN: TYPEN UND WERTE (2/2)

Datentyp	Mögliche Werte	Anwendungsbeispiel
int	Ganzzahlen: Natürliche Zahlen (1,2,3,...), 0 und negative Werte	<code>int numPlayers = 11</code>
float	Gleitkommazahlen, also <i>int mit Komma</i> , daher genauer!	<code>float averageGoals = 0.2</code>
double	Wie float aber kleinere bzw. größere Zahlen möglich	<code>double winProbability = 3.14159265359</code>
char	Zeichen: Buchstaben, Interpunktionszeichen, etc.	<code>char keyPlayerOne = 'x'</code>
boolean	Wahrheitswert: ja oder nein bzw. true oder false	<code>boolean playing = true</code>

## VARIABLEN DEKLARIEREN UND MANIPULIEREN (1/2)

Sie können in Java die bekannten mathematischen Operatoren (u.a. +, -, / oder \*) verwenden, um numerische Werte zu manipulieren. Mathematische Ausdrücke können direkt im Code notiert werden. Ausgewertet werden diese unter Verwendung der bekannten mathematischen (Klammer-) Regeln.

```
1 int result;  
2  
3 result = 42 + 1337; // In der Variable result ist nun der Wert 1379 gespeichert  
4 result = 42 - 1337; // In der Variable result ist nun der Wert -1295 gespeichert  
5 result = 42 * 1337; // In der Variable result ist nun der Wert 56154 gespeichert  
6 result = 1337 / 42; // In der Variable result ist nun der Wert 31 gespeichert
```

Achtung: Rationale Zahlen können nur in float- oder double-Variablen gespeichert werden. Beim *Verpacken* in eine int-Variable geht der Teil nach dem Dezimalzeichen verloren.

## VARIABLEN DEKLARIEREN UND MANIPULIEREN (2/2)

```
1 public void run() {  
2     double myDouble = 4.2;  
3     int myInt = 42;  
4     int anotherInt;  
5     anotherInt = 24;  
6     myInt = 1337;  
7     myInt = 1337 + 42;  
8     anotherInt = myInt;  
9     myInt = 101;  
10 }
```

Methode run ausführen

Code Zeile -  
myDouble -  
myInt -  
anotherInt -

# NAMENSKONVENTIONEN FÜR VARIABLEN (1/3)

Zulässige Variablennamen ...

- ... beginnen mit einem Buchstaben oder Unterstrich (\_)
- ... bestehen nur aus Buchstaben, Zahlen und Unterstrichen
- ... und sind keine reservierten Wörter (*Keywords*) in Java

## Java Language Keywords

Here is a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords const and goto are reserved, even though they are not currently used. true, false, and null might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

abstract	continue	for	new	switch
assert**	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum***	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

## NAMENSKONVENTIONEN FÜR VARIABLEN (2/3)

Nur einige dieser Namen können für Variablen verwendet werden:

x

WIDTH

~~2endPlayer~~ Beginnt nicht mit Buchstaben oder Unterstrich.

\_numPlayers

numberOfPlayers

number\_of\_players

~~Number of Players~~ Enthält Leerzeichen.

~~number\*Players~~ Enthält andere Zeichen als Buchstaben, Zahlen und den Unterstrich.

~~else~~ Ist ein reserviertes Wort.

## NAMENSKONVENTIONEN FÜR VARIABLEN (3/3)

- Variablen die den genannten Konventionen entsprechen sind syntaktisch korrekt: *Ihr Programm funktioniert*
- Zusätzlich gibt es weitere Konventionen, die beschreiben, wie Ihr Code, unabhängig seiner syntaktischen Korrektheit, formatiert sein sollte: *Ihr Code ist lesbar gestaltet*
- Für alle Bezeichner gilt: Vermeiden Sie das Mischen von englischer und deutscher Sprache
- Für Variablen gilt: Namen beginnen mit einem Kleinbuchstaben und werden (wenn nötig) in der CamelCase-Notation geschrieben: d.h. *alleBuchstabenEinesNeuenWortesWerdenGrossGeschrieben*

# BLÖCKE

Ein Block ist eine abgetrennte Menge von Anweisungen  
(Statements) zwischen geschweiften Klammern:

```
1 {
2     int x = 5;
3     double y = readDouble("Zahl: ");
4     System.out.println("y ist: " + y);
5 }
```

Blöcke können notwendiger Teil eines Sprachelements z.B. einer Schleife oder einer if-Abfrage sein, oder als optionale Strukturelemente innerhalb einer Methode (oder bedingt auch einer Klasse) existieren. Im Bezug auf Variablen definieren Blöcke deren *Scope*.

# SCOPES IN JAVA

Der Scope (Anwendungsbereich) einer Variable beschreibt den Bereich in dem sie verwendet werden kann und ist auf den Block beschränkt, in dem diese deklariert wurde:

Klasse	Methode	Kontrollstrukturen
Variablen, die auf Klassenebene deklariert wurden, können von jeder Stelle der umschließenden Klasse genutzt werden.	Variablen, die in einer Methode deklariert werden, sind nur dort, nicht aber in der umschließenden Klasse nutzbar.	Variablen, die innerhalb einer Kontrollstruktur deklariert werden, sind nur dort und nicht in der umschließenden Methode nutzbar.

## BEISPIEL: VARIABLE-SCOPE

```
1 public void printNum() {  
2     int x = 5;  
3     double y = readDouble("Zahl: ");  
4     System.out.println("y ist: " + y);  
5 }  
6  
7 public void printAnotherNum() {  
8     // BUG: Das funktioniert nicht, da y im aktuellen Scope nicht definiert wurde  
9     System.out.println("y ist: " + y);  
10 }
```

Die Deklaration der Variable `y` aus `printNum` ist in `printAnotherNum` nicht mehr gültig. Der Fehler wird vom Compiler erkannt und in *IntelliJ* sogar schon vor dem Ausführen automatisch durch hervorheben der entsprechenden Code-Zeile markiert.

# ABKÜRZUNGEN FÜR HÄUFIGE MATHEMATISCHE OPERATIONEN (1/3)

Beim Programmieren stoßen Sie häufig auf solche Konstruktionen:

w = w + 2; x = x \* 3; y = y / 4; z = z - 5;

bzw.

variable = variable [OPERATOR] value

Was passiert hier? Auf der rechten Seite der Operationen (des *Ausdrucks*) wird der aktuelle Wert der Variable modifiziert. Anschließend wird dieser neue Wert in der Variable auf der linken Seite gespeichert.

# ABKÜRZUNGEN FÜR HÄUFIGE MATHEMATISCHE OPERATIONEN (2/3)

Für diese, sehr häufige, Art der Variablenmanipulation können Sie eine Kurzschreibweise benutztten, die Modifikation und Zuweisung verbindet:

variable [OPERATOR]= value;

bzw.

w += 2; x \*= 3; y /= 4; z -= 5;

# ABKÜRZUNGEN FÜR HÄUFIGE MATHEMATISCHE OPERATIONEN (3/3)

Besonders häufig ist die Addition bzw. Subtraktion um genau 1, z.B. zum Zählen von durchgeführten Schleifendurchgängen. Dafür gibt es eine noch kürzere Schreibweise:

```
1 int i = 42;
2 i++; // Ersetzt i += 1: Die Variable i hat jetzt den Wert 43
3 i--; // Ersetzt i -= 1: Die Variable i hat jetzt wieder den Wert 42;
```

## EIN BLICK ZURÜCK AUF DIE `for`-SCHLEIFE

Wir haben bereits in den ersten zwei Wochen Variablen verwendet, u.a. in den `for`-Schleifen:

```
1 for(int i=0; i < 3; i++) {  
2     bouncer.turnLeft();  
3 }
```

In der sogenannte *Zählervariable* `i` wird eine Ganzzahl gespeichert, die beim Start der Schleife mit dem Wert 0 initialisiert wird. Mit jedem Schleifendurchlauf wird dieser Wert um 1 inkrementiert (`i++`). Sobald `i` den Wert 3 erreicht hat, also nicht mehr \* kleiner als 3\* ist, bricht die Schleife ab.

# UNVERÄNDERBARE VARIABLEN: KONSTANTEN

Variablen werden genutzt, um veränderbare Werte zu speichern und gegebenenfalls zu manipulieren. Fixe Werte werden (unveränderlich) in sogenannten Konstanten abgelegt und am Anfang einer Klasse (außerhalb von Methoden) definiert.

```
1 public static final TYPE NAME = VALUE;  
2  
3 // Ein Beispiel  
4 public static final double PI = 3.14159265;
```

Für Konstanten gelten besondere Namenskonvention: Konstanten werden immer in GROSSBUCHSTABEN\_MIT\_UNTERSTRICHEN notiert um sie von veränderbaren Variablen zu unterscheiden.

# KLASSEN UND OBJEKTE: ERSTE INFORMATIONEN

# WAS BEDEUTET *OBJEKTORIENTIERTE PROGRAMMIERUNG* EIGENTLICH?

- Java-Code besteht immer aus einer oder mehreren Klassen: Diese sind die zentralen Elemente eines Programms und stellen Muster / Schablonen für Objekte dar
- Zur Laufzeit werden aus Klassen, Objekte instanziert, mit denen der Computer interagiert und das von Ihnen geschriebene Programm umsetzt.

## KLASSEN ALS VORLAGE ÄHNLICHER DINGE

Klassen sind Sammlungen von Merkmalen und Verhalten, die eine Vorlage für eine gleiche oder ähnliche Gruppe von Dingen bilden. Klassen sind Konzepte, keine greifbaren Dinge.

---

**Ein Beispiel:** Die Klasse Dozierende beschreibt Personen, die an der Uni arbeiten und Kurse halten. Sie fasst die Eigenschaften und Tätigkeiten zusammen, die alle Dozierende haben.

## OBJEKTE ALS KONKRETE EXEMPLARE EINER KLASSE

Objekte sind Dinge, die über das Verhalten und die Merkmale von Klassen verfügen. Jedes Objekt ist eine Instanz bzw. eine konkret Repräsentation einer bestimmten Klasse.

---

**Ein Beispiel:** Niels Henze und Alexander Bazo sind Instanzen der Klasse Dozierende. Beide arbeiten an der Uni und halten dort Kurse. Die beiden „Objekte“ vom Typ Dozierende unterscheiden sich z. B. durch ihre Namen, Kurse oder ihr Alter.

# DIE KLASSE "HUND"

# INSTANZEN UND OBJEKTE



## ÄHNLICHE UND UNTERSCHIEDLICHE KLASSEN

Unterschiedliche Klassen können gleiche Merkmale oder Verhaltensmuster besitzen.  
Katzen und Hunde z.B. haben gewisse Gemeinsamkeiten, unterscheiden sich aber auch:

# EIN EINFACHES BEISPIEL FÜR EINE KLASSENHIERARCHIE: DAS HAUSTIER (1/2)

## EIN EINFACHES BEISPIEL FÜR EINE KLASSENHIERARCHIE: DAS HAUSTIER (2/2)

Die Klassen Dog und Cat sind Subklassen der Klasse Pet und teilen sich alle Eigenschaften (und Verhaltensmuster), die einem Haustier (Pet) eigen sind: Fell & Fressen

## OBJEKTORIENTIERUNG UND DIE REALITÄT

Klassen & Objekte orientieren sich häufig aber nicht immer an Dingen aus der realen Welt! In Ihren Programmen müssen Sie auch Klassen und Objekte nutzen bzw. anlegen, die keine oder keine direkte Entsprechung in der realen Welt haben: *Bouncer* & *BouncerApp*, Dateien auf dem Computer, Auswahllisten, Emails, ...

# EIN BEREITS BEKANNTES BEISPIEL FÜR DIE VERWENDUNG VON KLASSEN

```
1 public class FirstRoom extends BouncerApp {  
2  
3     public void bounce() {  
4         loadMap("Empty");  
5         bouncer.move();  
6     }  
7  
8 }
```

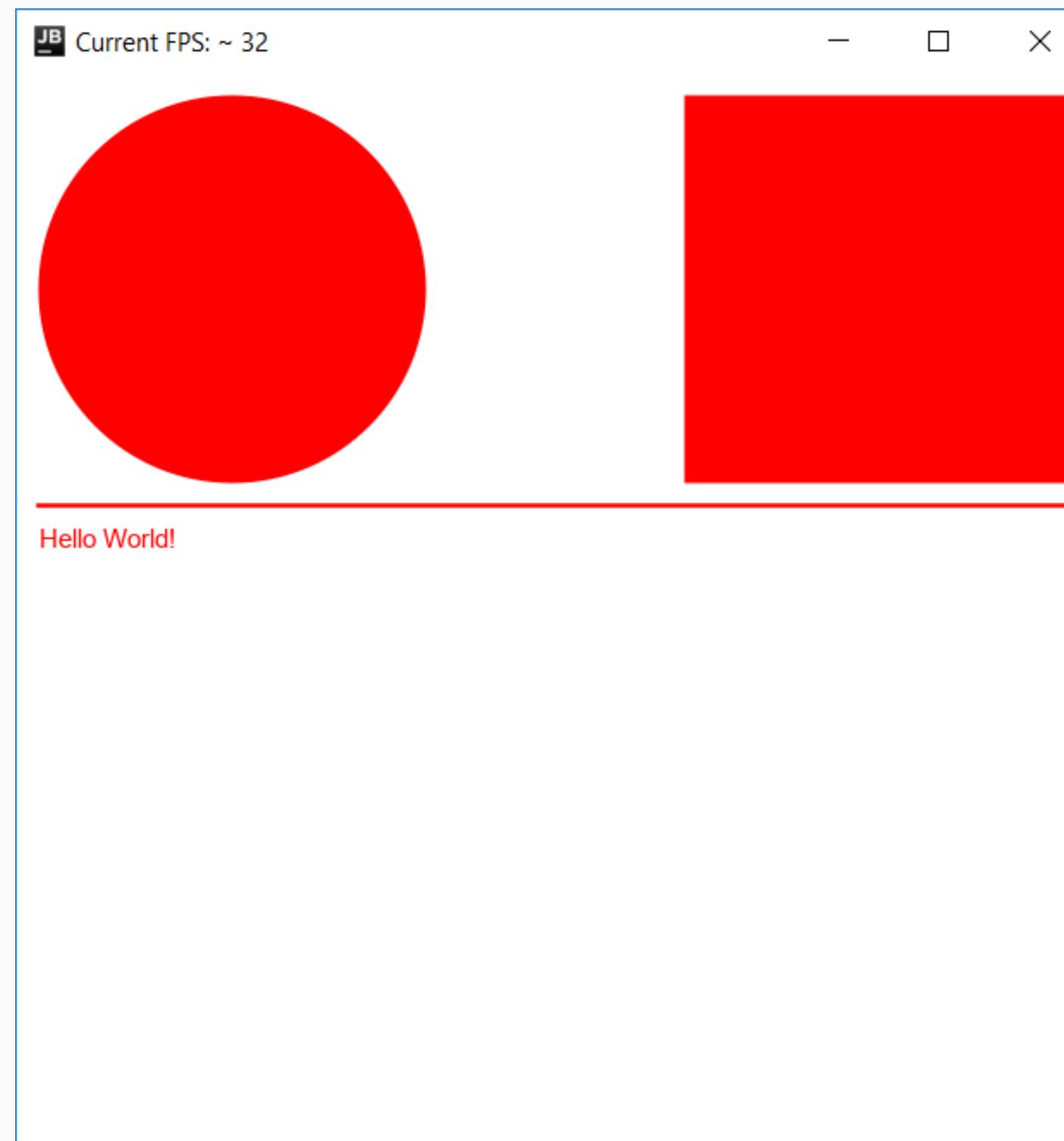
Wir haben bereits mehrere *Bouncer*-Anwendungen geschrieben. Diese bestanden **immer** aus einer Klasse mit ähnlichem Aufbau. Alle *Bouncer*-Programme haben sich z.B. die Methode `bounce` geteilt, in der aber von Programm zu Programm **unterschiedliche** Befehle standen.

# DIE GRAPHICS APP-UMGEBUNG

## UNTERSCHIEDLICHE ARTEN VON PROGRAMMEN FÜR UNTERSCHIEDLICHE ZWECKE

- Mit den *Bouncer*-Programmen haben Sie die ersten Schritte im Bereich der Programmierung gemacht und *Grundlagen gelernt*
- Mit den reinen Konsolenprogrammen (siehe *Hello World* aus der letzten Woche) können Sie Anwendung ohne oder mit geringer Nutzerinteraktion erstellen
- Für graphische Anwendungen wie z.B. Animationen oder kleine Spiele verwenden wir in diesem Kurs die *GraphicsApp*

# DIE GRAPHICSAPP



GraphicsApp-Anwendungen verfügen über eine Zeichenfläche. Auf dieser Fläche können Sie graphische Objekte einzeichnen und deren Aussehen und Position verändern. Einige graphische Objekte sind vorgegeben. Andere können Sie durch Kombination dieser Vorgaben oder Bilder bzw. Texte selbst zusammensetzen.

# EINE GRAPHICSAPP-ANWENDUNG ERSTELLEN

```
1 // In der zentralen Klasse Ihres Programms können Sie den Typen durch die Klassenhierarchie
2 // angeben. Jede GraphicsApp ist eine Subklasse von GraphicsApp.
3 public class HelloGraphicsApp extends GraphicsApp {
4
5     // Diese Methode stellt den Einstiegspunkt in die Anwendung da (Vgl. main oder bounce).
6     public void initialize() {
7         // Hier können Sie allgemeine Einstellungen an der Anwendung z.B. die Fenstergröße vornehmen.
8     }
9
10    // Hier können Sie Befehle zum Zeichnen graphischer Elemente notieren.
11    // Die Methode wird während der Laufzeit des Programms wiederholt, d.h. ca. 60x pro Sekunde,
12    // aufgerufen. Dadurch können wir später z.B. Animationen erzeugen, in dem wir in jedem "Frame"
13    // andere Elemente zeichnen!
14    public void draw() {
15        // Hier werden die Elemente erstellt und gezeichnet.
16    }
17
18 }
```

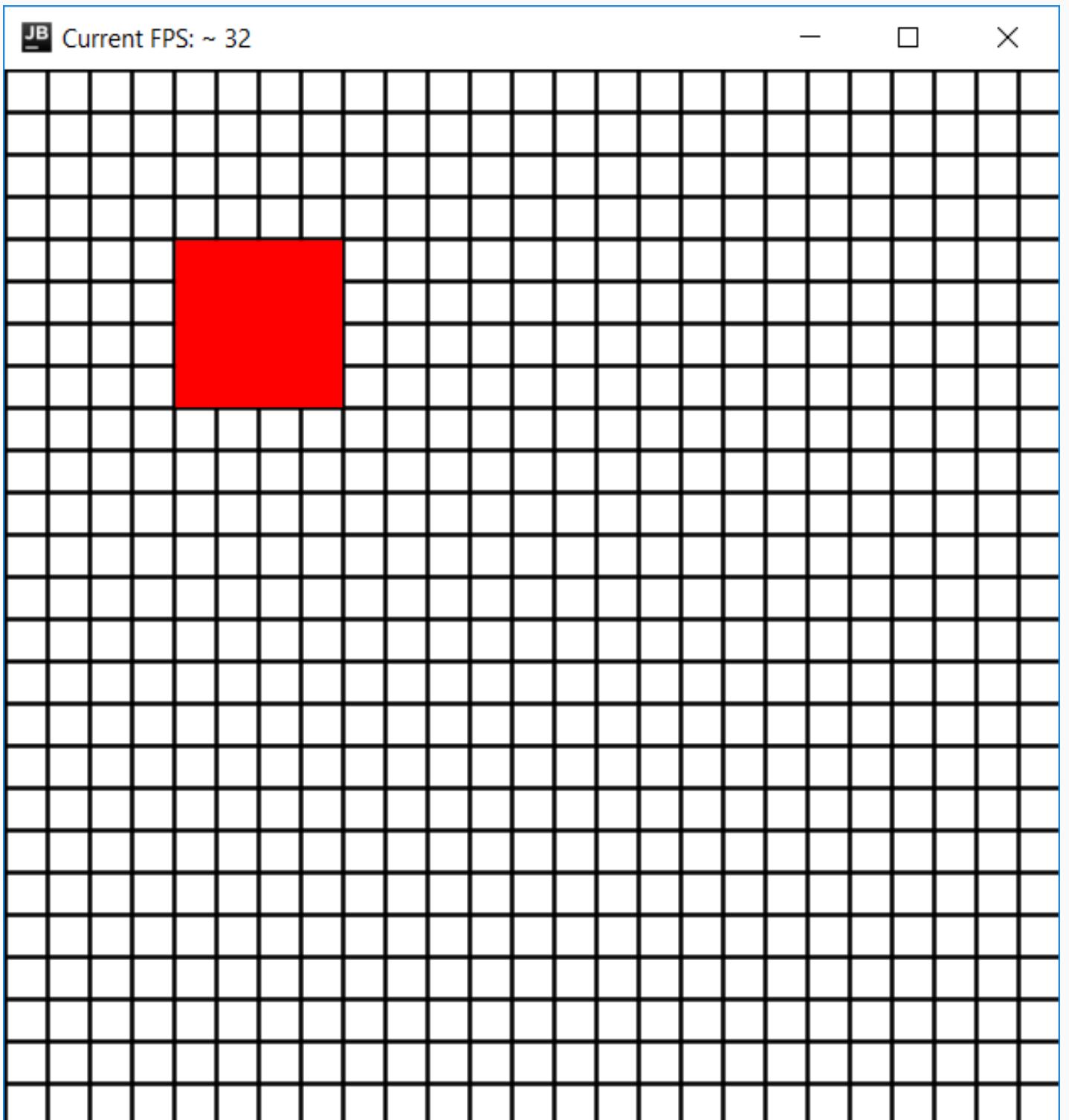
## EINE GRAPHICSAPP-ANWENDUNG STARTEN

Um eine *GraphicsApp* zu starten benötigen Sie eine zweite Datei, deren Klassen über eine `main`-Methode verfügt:

```
1 public class HelloGraphicsAppLauncher {  
2  
3     public static void main(String[] args) {  
4         GraphicsAppLauncher.launch("HelloGraphicsApp");  
5     }  
6  
7 }
```

Wie bei *Bouncer* wird hier der Name des zu startenden Programms angegeben (hier: `GraphicsAppLauncher.launch("HelloGraphicsApp");`). Die Anwendung wird über diese Datei gestartet (Rechtsklick und `run`). Keine Sorge, wir bereiten das in der Regel für Sie vor.

# DAS KOORDINATENSYSTEM DER GRAPHICSAPP



Das Raster für das Koordinatensystem bilden die Pixel auf Ihrem Bildschirm. Wenn Sie ein Objekt an die Position (2,2) verschieben, erscheint es 2 Pixel vom oberen und 2 Pixel vom linken Rand der Zeichenfläche versetzt.

## VERFÜGBARE GRAPHISCHE ELEMENTE

- Alle Elemente, die Sie auf dem Bildschirm zeichnen können, werden durch Klassen bzw. Objekte repräsentiert.
- Jede dieser Klassen ist eine Subklassen von GraphicsObject und hat z.B. eine Größen oder eine Position auf dem Bildschirm.
- Sie können Rechtecke (Klasse Rectangle), Ellipsen und Kreise (Ellipse bzw. Circle), Linien (Line), Texte (Label) oder Bilder (Image) zeichnen.
- Wenn Sie ein solches Element auf den Bildschirm zeichnen wollen, erzeugen Sie ein Objekt (Instanz) der jeweiligen Subklasse und spezifizieren das Aussehen.

## OBJEKTE IN JAVA (UND DER GRAPHICSAPP) ERZEUGEN

```
Ellipse circle = new Ellipse(200, 200, 100, 100,  
    Colors.RED);
```

Über den **Klassennamen** (`Ellipse`) spezifizieren Sie die gewünschte *Art* bzw. Klasse des Elements. Diese stellt den **Datentyp** der Variable (hier: `circle`) dar, in der Sie das neue Element speichern wollen. Über das Schlüsselwort `new` wird das Erstellen einer neuen **Instanz** eingeleitet. Dazu müssen eine Reihe von **Parametern** angegeben werden, die die initialen **Eigenschaftswerte** der Instanz beschreiben.

Die zu übergebenen Parameter und deren Bedeutung unterscheiden sich von Klasse zu Klasse. Eine genaue Dokumentation darüber, wie die einzelnen Elemente erzeugt werden können, finden Sie im GRIPS-Kurs

## FARBEN IN DER GRAPHICSAPP

Überall dort, wo Sie Farben angeben können oder müssen, benötigen Sie eine **Instanz** der Klasse `Color`. Sie können diese auf unterschiedliche Art und Weise erhalten:

- In der Klasse `Colors` finden Sie einige Eigenschaften, die vorgegeben Farben repräsentieren, z.B. `Colors.RED`.
- Über die **Methode** `Colors.getRandomColor()` können Sie eine zufällige Farbe erstellen und z.B. in einer Variable speichern: `Color randomColor = Colors.getRandomColor();`.
- Sie können selbstständig Farben aus dem *RGB-Raum mischen*:  
`Color myColor = new Color(255,0,0);` (Hier wird die Farbe *Rot* erzeugt).

## MIT OBJEKten KOMMUNIZIEREN (1/2)

- Neben Eigenschaften besitzen Klassen bzw. Objekte auch *Verhalten*. Dieses Verhalten ist über Methoden innerhalb der jeweiligen Klassen definiert. Was eine Klasse machen kann (welche Methoden es besitzt), können Sie in der Dokumentation nachschlagen.
- Sie können eine Nachricht an ein Objekt (Instanz einer Klasse) schicken, um dieses aufzufordern, ein bestimmtes Verhalten auszuführen: NAMEDESOBJEKTS.METHODE();.

## MIT OBJEKten KOMMUNIZIEREN (2/2)

- Das haben wir auch schon mit *Bouncer* so gemacht:  
`bouncer.move();` (d.h. auch, irgendwo muss eine Variable mit dem Namen bouncer erstellt worden sein ...)
- Sie müssen diesen Mechanismus nutzen, um die erstellen Objekte zu zeichnen, da diese sonst nicht auf dem Bildschirm erscheinen:

```
1 Ellipse circle = new Ellipse(200, 200, 100, 100, Color.CYAN);
2 circle.draw();
```

# DEMO: DIE ERSTEN SCHRITTE MIT DER GRAPHICSAPP

*Hint:* Mit dem Befehl `drawBackground()` können Sie den Hintergrund der Zeichenfläche einfärben, z.B. so:

```
drawBackground(Colors.BLACK);
```

## DEMO: EINEN KREIS ZENTRIEREN

*Hint:* Über die Methoden `getHeight()` und `getWidth()` erhalten Sie Zugriff auf die aktuelle Höhe und Breite der Zeichenfläche und können diese in mathematischen Operationen verwenden.

# ZUSAMMENFASSUNG (VARIABLEN)

- Variablen speichern Werte während der Programmausführung (Laufzeit). Sie sind Platzhalter für unterschiedliche Daten, funktionieren aber immer nur mit einem bestimmten Datentyp.
- Werte, die während der Implementierung festgelegt werden und zur Laufzeit unveränderlich sind, nennt man Konstanten.

# ZUSAMMENFASSUNG (KLASSEN UND OBJEKTE)

- Java ist eine objektorientierte Programmiersprache.
- Klassen definieren Eigenschaften und Verhalten für ein oder mehrere Objekte
- Objekte sind die Instanzen der Klassen
- Klassen sind hierarchisch angeordnet: Superklassen können Eigenschaften und Verhalten definieren, die für alle Subklassen gelten
- Objekte werden mit dem Aufruf new erzeugt.
- ProgrammiererInnen können Nachrichten an Objekte senden und dazu deren Methoden aufrufen.

# ZUSAMMENFASSUNG (GRAPHICSAPP)

- Mit Hilfe der GraphicsApp können wir einfache graphische Elemente auf dem Bildschirm zeichnen
- Alle Elemente werden durch Instanzen der entsprechenden Klassen repräsentiert
- Über die Eigenschaften der Objekte/Instanzen wird deren Erscheinungsbild definiert
- Die Elemente sind erst sichtbar, nachdem ihre draw-Methode aufgerufen wurde

## 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](mailto:alexander.bazo@ur.de) ). Bei organisatorischen Fragen zu den Studienleistungen und Übungsgruppen schreiben Sie bitte Florin Schwappach ([florin.schwappach@ur.de](mailto:florin.schwappach@ur.de) ). Bei inhaltlichen Fragen zu den Übungen, Beispielen und Studienleistungen schreiben Sie uns unter [mi.oop@mailman.uni-regensburg.de](mailto:mi.oop@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.