

# EINFÜHRUNG IN DIE PROGRAMMIERUNG MIT JAVA (OOP)

## Kursaufbau, Grundlagen & erste Schritte mit Bouncer

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

Wir befinden uns in der ersten Kurswoche. Der Kurs setzt keine besonderen Kenntnisse voraus. Solange Sie einigermaßen vertraut mit der grundlegenden Bedienung eines PCs sind, können Sie ohne Probleme starten.

Wir **starten wir bei Null**. Vorwissen auf dem Gebiet der Programmierung kann vom Vorteil sein, ist aber keine Voraussetzung für das erfolgreiche Bestehen des Kurses.

# DAS PROGRAMM FÜR HEUTE

- Allgemeine Informationen zum Kurs und zum Team
- Einführung in die Programmierung
- Erste Schritte mit Bouncer
- Hinweise zu den Übungsaufgaben

# ALLGEMEINE INFORMATIONEN

# DAS MODUL PI-BA-M01 (1/2)

*In diesem Modul werden Kernkonzepte der praktischen Informatik vermittelt. Dabei handelt es sich um eine Einführung in die Programmierung und Programmiersprachen, die neben einem Überblick zu unterschiedlichen Programmierkonzepten eine Einführung in die objektorientierte Programmierung als derzeit vorherrschendem Entwicklungsparadigma bietet.*

Quelle: [Modulbeschreibung BA Medieninformatik](#)

# DAS MODUL PI-BA-M01 (2/2)

*Nach Abschluss des Moduls kennen Studierende grundlegende Konzepte der objektorientierten Programmierung (beispielsweise Syntax, essentielle Programmkonstrukte, Vererbung, Komposition) und können vorgegebene Problemstellungen analysieren, algorithmische Lösungen dafür entwickeln und diese in einer Programmiersprache implementieren.*

Quelle: [Modulbeschreibung BA Medieninformatik](#)

# DIE KURSINHALTE

- Sie **lernen** die Grundlagen der objektorientierten Programmierung am Beispiel einfacher, interaktiver Anwendungen kennen
- Sie entwickeln die **Kompetenz**, zur selbständig Lösung einfache Programmierproblem
- Sie eignen sich grundlegende Regeln und Empfehlungen zur Gestaltung und Formatierung von Quellcode an

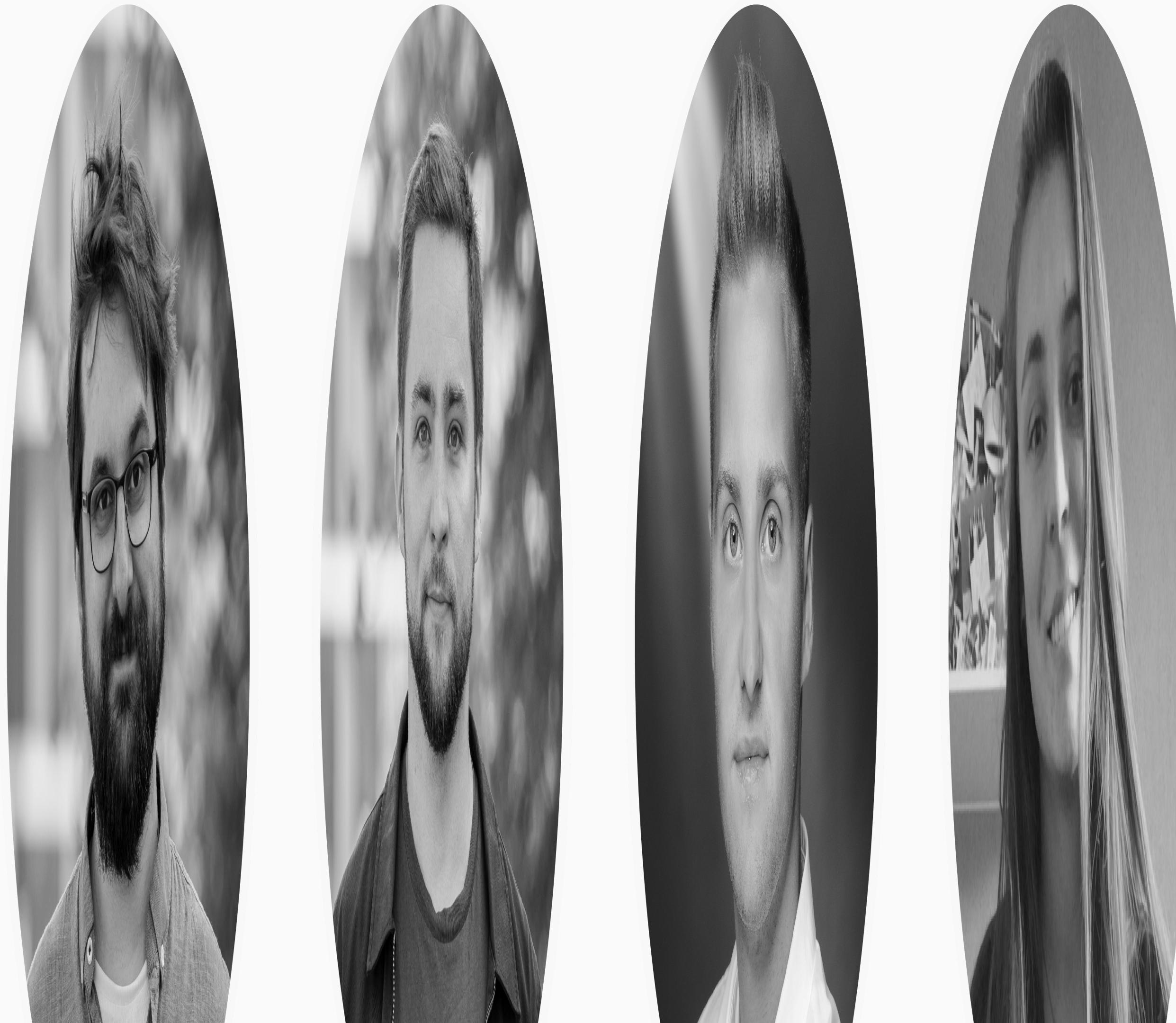
**Dazu verwenden wir die Programmiersprache Java**

# OOP UND DAS STUDIUM DER MEDIENINFORMATIK

Die Medieninformatik ist eine angewandte Informatik. Die Entwicklung von Software ist elementare Teil und wichtige Grundlage der Auseinandersetzung mit den Untersuchungsgegenständen des Fachs.

- In weiterführenden Kursen des Studiums werden Sie die hier gewonnenen Fähigkeiten ausbauen **und anwenden**
- Die Entwicklung von Softwareartefakten ist häufiges Ziel der praktisch ausgerichteten Seminare und Übungen
- Für (eigener) Forschungsvorhaben entwickeln und untersuchen Medieninformatiker und -Informatikerin Softwaresysteme

# DAS OOP-TEAM



# IHRE WOCHE MIT UNS

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
08-10					
10-12	Übung		Übung	Übung	
12-14	Übung (2x)	Vorlesung	Übung	Übung (2x)	
14-16		Zentralübung			
16-18			Übung		
18-20			Übung		

Die Inhalte von Vorlesung und Zentralübung werden in den Übungen ab Mittwoch aufgegriffen. Die Übungen am Montag behandeln Themen aus der vorherigen Woche.

# VORLESUNG: THEORIE

- Theoretische Einführung in die verschiedenen Kursthemen
- Vorstellung der zentralen Elemente prozeduraler und objektorientierte Programmierung
- Überblick über Syntax und Besonderheiten der eingesetzten Programmiersprache *Java*
- Methoden und *Best Practices* in der Softwareentwicklung

In der Vorlesung wird **Wissen vermittelt.**

# ZENTRALÜBUNG: DEMONSTRATION

- Praktische Demonstration der in Vorlesung vorgestellten Themen
- Kommentiertes Entwickeln passender Programmbeispiele
- Diskussion der angewandten Lösungsstrategien

In der Zentralübung wird die **Anwendung des Wissens demonstriert.**

# ÜBUNGEN: ANWENDUNG

- Selbstständige Arbeit an Programmieraufgaben
- Sammlung mehrerer Übungsaufgaben zum aktuellen Themenbereich
- Aufgabenbeschreibung und Grundgerüst (*Code*) für jede Aufgabe
- Individuelle Unterstützung durch die TutorInnen
- Kommentierte Lösungsvorschläge werden in der darauffolgenden Woche veröffentlicht

In der Übung setzten Sie das Gehörte **praktisch um.**

# STUDIENLEISTUNG (1/2)

- Drei komplexerer Programmieraufgaben im Laufe des Semesters: Zwei von drei Aufgaben müssen für den Abschuss des Moduls bestanden werden
- Selbstständige Erarbeitung einer funktional und qualitativ ausreichenden Lösung
- Individuelle Bewertung (*bestanden/nicht bestanden*) und Feedback
- Für die Studienleistungen ist eine zusätzliche Anmeldung im Prüfungssystem *Flexnow* notwendig

# STUDIENLEISTUNG (2/2)

- Bei den Studienleistungen ist **keine** Zusammenarbeit mit Kommilitonen erlaubt
- Verwendete Quellen müssen angegeben werden
- Alle Abgaben werden maschinell und manuell auf Übereinstimmungen geprüft
- Plagiate in den Studienleistungen führen für alle Beteiligten automatisch zum *nicht bestehen* des Kurses

Durch die Studienleistung weisen Sie nach, dass Sie Lernziele erreicht haben und die erworbenen Kompetenzen selbstständig anwenden können. Weitere Informationen finden Sie im GRIPS-Kurs.

# KLAUSUR

- *Klassische Klausur am 18. Februar*
- Für die Klausur ist eine zusätzliche Anmeldung im Prüfungssystem *Flexnow* notwendig
- Die Klausurnote bildet zu 100% die Abschlussnote für das Modul *PI-BA-M01*

# ARBEITSMATERIALIEN

Alle Materialien zum Kurs finden Sie im GRIPS-Kurs

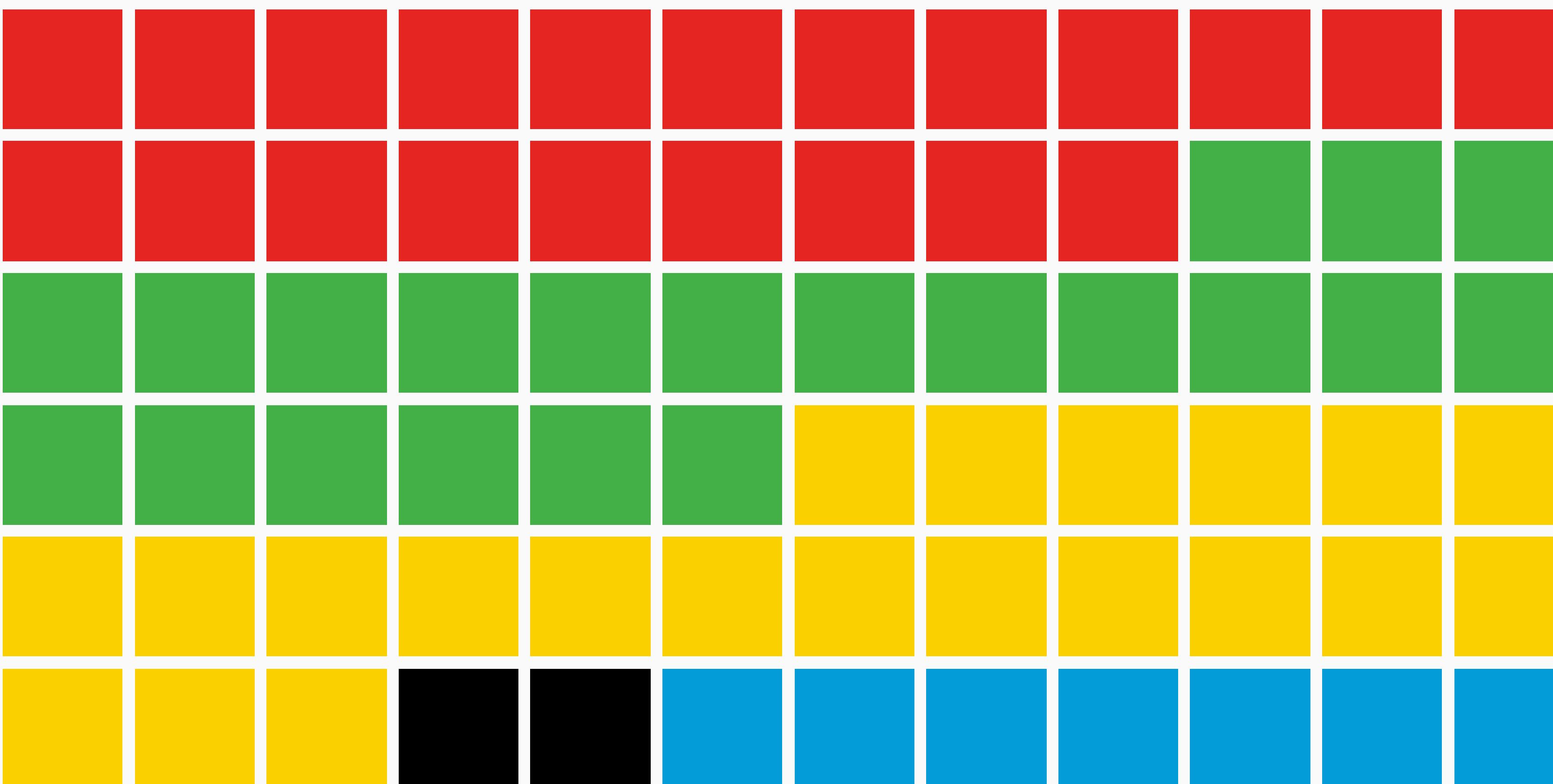
- Links zu Zusammenfassungen, Foliensätze und Übungsblätter
- Informationen, Dokumentationen und Lesetexte
- Forum für Ankündigungen und Fragen
- Material und *Upload*-Möglichkeit für Studienleistungen
- Lesetexte und Vorlesungsaufzeichnungen

# THE ART & SCIENCE OF JAVA (ROBERTS 2014)

- Inhaltliche Grundlage für einige der Vorlesungsthemen
- Passende Textauszüge sind im GRIPS-Kurs den einzelnen Sitzungen zugeordnet
- Das vollständige Buch kann aus der **Universitäts Bibliothek** entliehen werden
- Versuchen Sie, die vorgeschlagenen Kapitel nach der Vorlesung selbstständig durchzuarbeiten

# IHR ARBEITSPENSUM IN DIESEM SEMESTER

6LP: 180 Stunden Arbeitsaufwand



# FEEDBACK UND UNTERSTÜTZUNG

Bei inhaltlichen Problemen finden Sie Unterstützung in den Übungen oder dem Forum. Sie können sich auch direkt an uns wenden: [mi.oop@mailman.uni-regensburg.de](mailto:mi.oop@mailman.uni-regensburg.de)

# EINE KURZE LISTE MIT FORDERUNGEN

- Besuchen Sie Vorlesung und Übung (und wenn möglich auch die Zentralübung)!
- Arbeiten Sie selbstständig Beispiele, Materialien und Aufgaben durch!
- Lösen Sie die Studienleistungen selbst!
- Lenken Sie in Vorlesung und Übung weder sich selbst noch andere ab!
- Geben Sie bei allen eingereichten Arbeiten die verwendeten Quellen an!

# SEMESTERÜBERBLICK

Einführung

Ein- und Ausgabe

Klassen und Methoden

Kontrollstrukturen und Variablen

Arrays und komplexe Schleife

Grundlagen der Klassenmodellierung

Vererbung und Sichtbarkeit

Event-basierten Programmierung

String- und Textverarbeitung

Planhaftes Vorgehen bei der Softwareentwicklung

Listen, Maps und die Collections

Qualitätsaspekte von Quellcode

# EINFÜHRUNG IN DIE PROGRAMMIERUNG

# INFORMATIK UND PROGRAMMIEREN

*“What would we like our children - the general public of the future - to learn about computer science in schools? We need to do away with the myth that computer science is about computers. Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes or chemistry is about beakers and test tubes. Science is not about tools, it is about how we use them and what we find out when we do.”*

Quelle: Micheal R. Fellows, Ian Parberry: SIGACT trying to get children excited about CS (1993)

# EINE DEFINITION DES BEGRIFFS PROGRAMMIEREN

*"Computer programming [...] is a process that leads from an original formulation of a computing problem to executable computer programs. Programming involves activities such as analysis, developing understanding [...] and implementation [...] of algorithms in a target programming language."*

Quelle: *Computer programming* (Wikipedia)

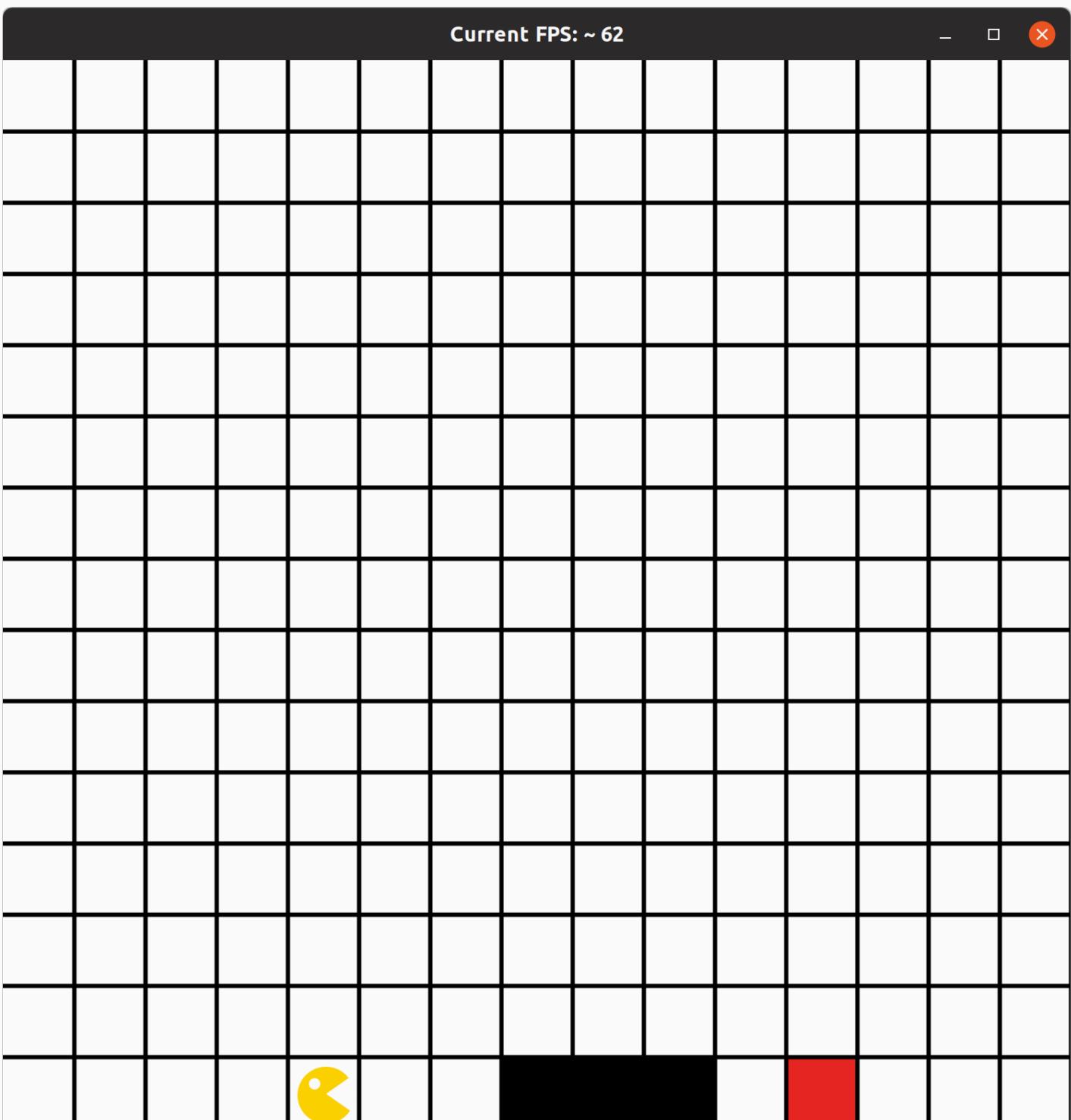
# PROGRAMMIEREN ALS MITTEL DER KOMMUNIKATION MIT DEM COMPUTER

# EINE KURZE GESCHICHTE DER COMPUTER-PROGRAMMIERUNG

# BOUNCER

Ein einfacher Einstieg in die Programmiersprache Java

# BOUNCERS WELT



Bouncers Welt besteht aus 225 Feldern in einem  $15 \times 15$  Einheiten großen Koordinatensystem. Blockierte Felder sind schwarz gekennzeichnet und können von Bouncer nicht betreten werden. Die übrigen Feldern können farblich markiert sein und Bouncer kann sich frei auf diesen bewegen. Bouncer kann sich um seine eigene Achse drehen und mit jedem Schritt ein Feld weit in die aktuelle Blickrichtung gehen.

# BOUNCER-PROGRAMME SCHREIBEN

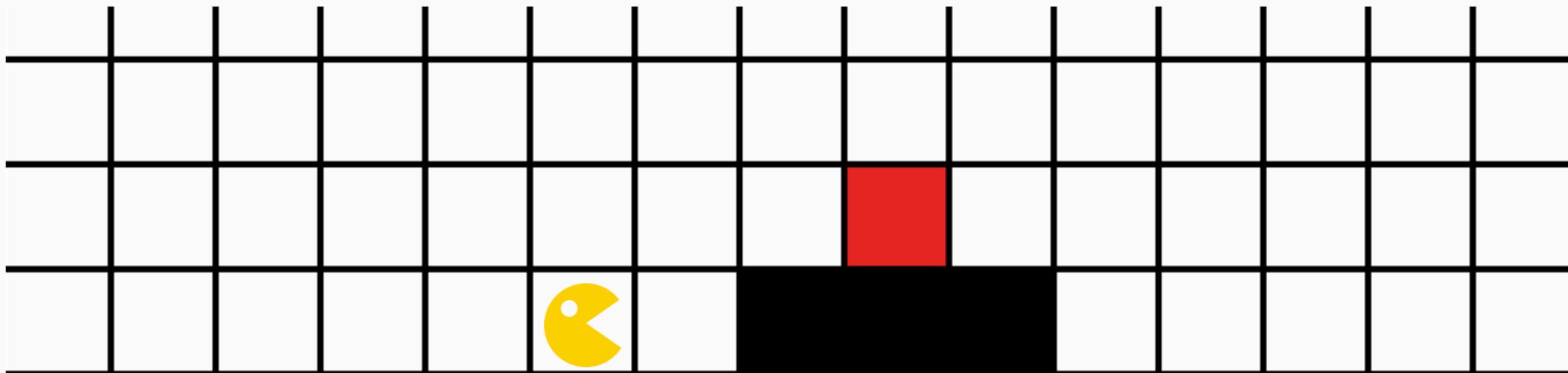
Ein Bouncer-Programm besteht aus einer beliebigen Anzahl an **Befehlen**. Nach dem Start des Programms führt Bouncer diese Befehle nacheinander in der Reihenfolge aus, in der sie notiert wurden. Die Abfolge wird an einer bestimmten Stelle einer **Datei** notiert. Der gesamte Inhalt dieser Datei bildet den **Quellcode** des Programms. Zum Editieren der Datei und zum Ausführen des Programms wird ein besonderes Computerprogramm, die **Entwicklungsumgebung** verwendet.

# EINIGE DER BEFEHLE, DIE BOUNCER VERSTEHT

Befehl	Beschreibung
bouncer.move()	Bouncer bewegt sich einen Schritt weiter in die aktuelle Blickrichtung
bouncer.turnLeft()	Bouncer dreht sich um 90° nach links
bouncer.paintField(FieldColor.RED oder FieldColor.GREEN oder FieldColor.BLUE)	Bouncer färbt das Feld auf dem er steht farblich ein

Die **Syntax** aller Befehle besteht aus dem Präfix bouncer, dem verbindenden Element `.` (*Punktoperator*) und dem eigentlichen Namen des Befehls, gefolgt von zwei Klammern `()`. Innerhalb des Programms werden einzelne Befehle durch das Zeichen `,` separiert.

# DAS ERSTE PROGRAMM



Bouncers Aufgabe: Auf das Hindernisse klettern und auf dem roten Feld stehen bleiben.

# DER CODE UNSERES ERSTEN PROGRAMMS

```
1 import de.ur.mi.bouncer.apps.BouncerApp;  
2  
3 public class BouncerDemo extends BouncerApp {  
4     @Override  
5     public void bounce() {  
6         loadMap("HelloBouncer");  
7         bouncer.move();  
8         bouncer.turnLeft();  
9         bouncer.move();  
10        bouncer.turnLeft();  
11        bouncer.turnLeft();  
12        bouncer.turnLeft();  
13        bouncer.move();  
14        bouncer.move();  
15    }  
16 }
```

Das Programm besteht aus mehreren verschachtelten Blöcken, die durch die Zeichen { und } gegliedert werden.

## DAS PROGRAMM IM ÜBERBLICK: IMPORT

Hier werden im Hintergrund die Inhalte importiert, die uns die Verwendung der *Bouncer*-Befehle erlauben.

```
1 import de.ur.mi.bouncer.apps.BouncerApp;
```

## DAS PROGRAMM IM ÜBERBLICK: KLASSENDEFINITION

**Klassen** bilden die zentrale Struktureinheit von Programmen in Java. Jedes Programm besteht aus mindestens einer Klasse.

```
1 public class BouncerDemo extends BouncerApp {
```

## DAS PROGRAMM IM ÜBERBLICK: METHODE

Innerhalb einer Klasse können mehrere **Methoden** existieren. Methoden bündeln mehrere **Befehle** und können über ihren **Namen** gestartet werden. Die Methode mit dem Namen `bounce` wird automatisch beim Ausführen des Programms gestartet.

```
1 public void bounce() {  
2     loadMap("HelloBouncer");  
3     bouncer.move();  
4     bouncer.turnLeft();  
5     bouncer.move();  
6     bouncer.turnLeft();  
7     bouncer.turnLeft();  
8     bouncer.turnLeft();  
9     bouncer.move();  
10    bouncer.move();  
11 }
```

## DIE METHODE IM ÜBERBLICK: METHODENKOPF

Eine Methode beginnt mit dem **Methodenkopf** in der ersten Zeile. Hier wird unter anderem der **Name** der Methode definiert.

```
1 public void bounce() {
```

## DIE METHODE IM ÜBERBLICK: METHODENRUMPF

Die Liste der Befehle bildet den **Rumpf** der Methode. Beim Aufruf der Methode werden alle Befehle in der angegebenen Reihenfolge ausgeführt. Mit dem ersten Befehl loadMap bestimmten Sie, welche Karte vor dem Ausführen der nächsten Befehle geladen werden soll. Der Name der Karte wird zwischen den Anführungszeichen angegeben. Unterschiedliche Karten beinhalten unterschiedlichen Hindernisse, eingefärbte Felder und Startpositionen für Bouncer.

```
1 loadMap("FirstRoom");
2 bouncer.move();
3 bouncer.turnLeft();
4 bouncer.move();
5 bouncer.turnLeft();
6 bouncer.turnLeft();
7 bouncer.turnLeft();
8 bouncer.move();
9 bouncer.move();
```

# METHODEN ALS STRUKTURELEMENT EINES PROGRAMMS

```
1 public void bounce() {  
2     loadMap("HelloBouncer");  
3     moveForward();  
4     climbObstacle();  
5     moveForward();  
6 }  
7 public void moveForward() {  
8     bouncer.move();  
9 }  
10 public void climbObstacle() {  
11     bouncer.turnLeft();  
12     bouncer.move();  
13     bouncer.turnLeft();  
14     bouncer.turnLeft();  
15     bouncer.turnLeft();  
16     bouncer.move();  
17 }
```

# EINZELNE BEFEHLE WIEDERHOLEN

Um Bouncer einen oder mehrere Befehl für eine bestimmte Anzahl an Durchläufen wiederholen zu lassen, verwenden wir die **for-Schleife**:

```
1 for(int i=0; i < N; i++) {  
2     // Alle Anweisungen (Befehle) in diesem Klammerpaar werden N-mal wiederholt  
3 }
```

Im **Schleifenkopf** (vgl. *Methodenkopf*) definieren Sie die Anzahl der Durchläufe. Der Wert *i* wird schrittweise vom Ausgangswert 0 bis zum von Ihnen angegebenen Wert *N* **inkrementiert**. In jedem Schritt werden der oder die Befehle innerhalb des Klammerpaares nacheinander ausgeführt.

# ENTSCHEIDUNGEN TREFFEN

Bouncer kann einzelne oder mehrere Befehle in Abhängigkeit bestimmter Zustände (**Bedingungen**) ausführen:

```
1 if(BEDINGUNG) {  
2     // Alle Anweisungen in diesem Klammerpaar werden ausgeführt, falls die Bedingung zutrifft  
3 }  
4  
5 if(BEDINGUNG) {  
6     // Alle Anweisungen in diesem Klammerpaar werden ausgeführt, falls die Bedingung zutrifft  
7 } else {  
8     // Alle Anweisungen in diesem Klammerpaar werden ausgeführt, falls die Bedingung NICHT zutrifft  
9 }
```

Mit Hilfe unterschiedlicher Bedingungsprüfung können Sie Informationen über die Umgebung erfragen, in der sich Bouncer befindet. So können Sie Bouncer auf unterschiedliche Situationen **reagieren** lassen.

# BEDINGUNGEN, DIE BOUNCER PRÜFEN KANN (1/2)

Befehl	Beschreibung
canMoveForward()	Prüft ob das Feld vor Bouncer frei ist
canNotMoveForward()	Prüft ob das Feld vor Bouncer blockiert ist
canMoveLeft()	Prüft ob das Feld links neben Bouncer frei ist
canNotMoveLeft()	Prüft ob das Feld links neben Bouncer blockiert ist
canMoveRight()	Prüft ob das Feld rechts neben Bouncer frei ist
canNotMoveRight()	Prüft ob das Feld rechts neben Bouncer blockiert ist

## BEDINGUNGEN, DIE BOUNCER PRÜFEN KANN (2/2)

Befehl	Beschreibung
isOnFieldWithColor(RED)	Prüft ob das Feld, auf dem Bouncer aktuell steht, rot ist
isOnFieldWithColor(GREEN)	Prüft ob das Feld, auf dem Bouncer aktuell steht, grün ist
isOnFieldWithColor(BLUE)	Prüft ob das Feld, auf dem Bouncer aktuell steht, blau ist

Die Bedingungen geben einen Wahrheitswert an Bouncer zurück - Wenn die abgefragte Bedingung zu trifft, ist sie WAHR (True), ansonsten FALSCH (False). Alle Bedingungen, die sich auf Bouncers Umgebung beziehen, werden immer von seiner aktuellen Blickrichtung ausgehend überprüft.

# BEFEHLE BEDINGT WIEDERHOLEN

Mit der **while-Schleife** lassen wir Bouncer einen oder mehrere Befehle wiederholen, solange eine Bedingung zutrifft:

```
1 while(BEDINGUNG) {  
2 // Alle Anweisungen in diesem Klammerpaar werden ausgeführt, solange die Bedingung zutrifft  
3 }
```

Solange die Bedingung zutrifft, werden die Anweisungen zwischen den Klammern durchgeführt. Dabei wird zuerst geprüft, ob die Bedingung zutrifft. Wenn dies der Fall ist, werden die Anweisungen in den Klammern **EINMAL** ausgeführt. Anschließend wird die Bedingung erneut geprüft. Trifft sie immer noch zu, werden die Anweisungen ein zweites Mal ausgeführt – dies geschieht solange, bis die Bedingung nicht mehr zutrifft.

# ALGORITHMEN

# EINE DEFINITION DES BEGRIFFS ALGORITHMUS

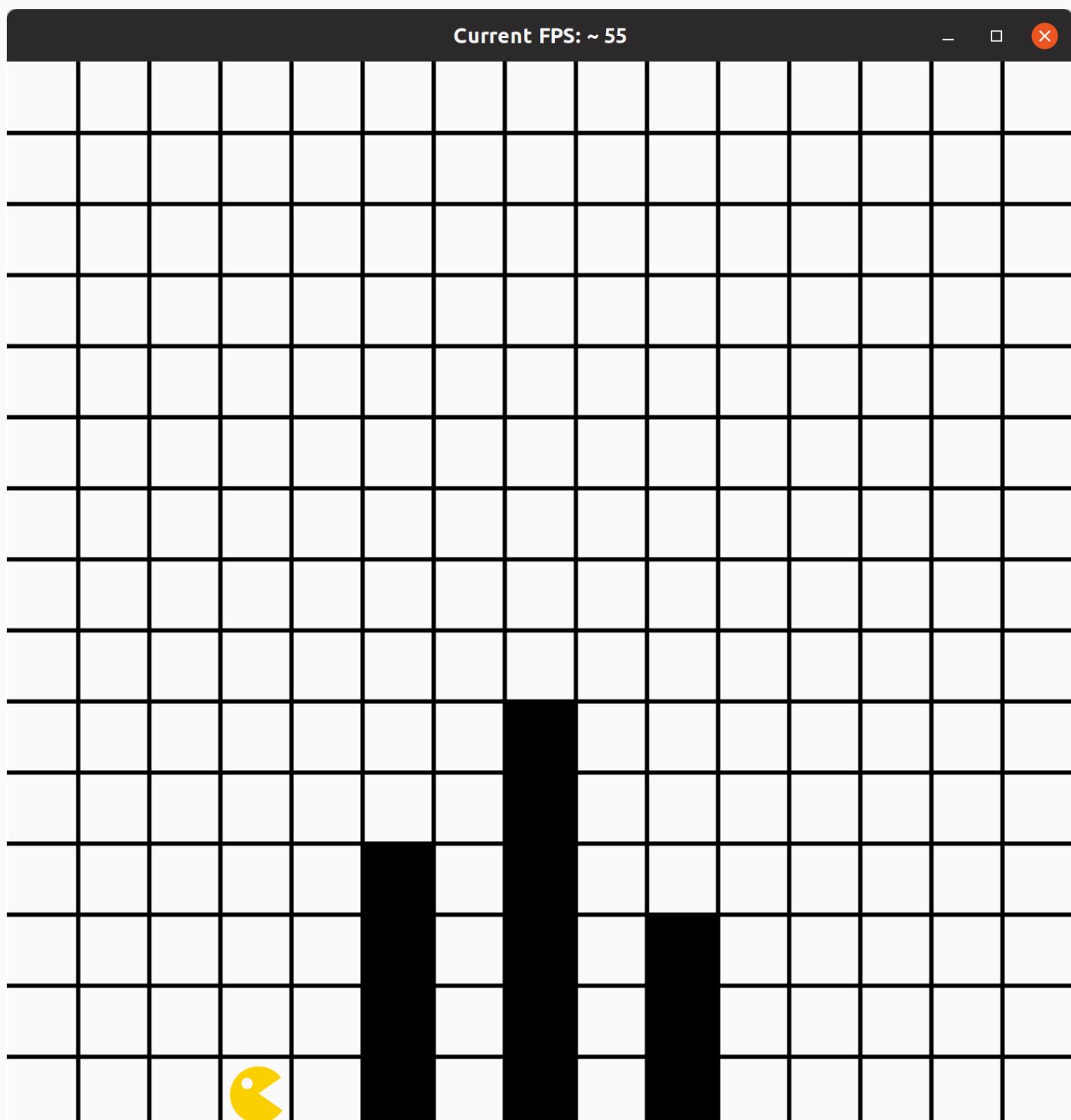
*„Starting from an initial state and initial input [...], the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state.“*

Quelle: *Algorithm* (Wikipedia)

# ALGORITHMEN MIT BOUNCER

Ein Algorithmus überführt einen gegebenen Anfangszustand in einen definierten Endzustand und definiert dazu eine Reihe von Befehlen die nacheinander abgearbeitet werden und jeweils eindeutige Zwischenzustände erzeugen.

## EIN BEISPIEL: BOUNCER BEIM HÜRDENLAUF



Bouncer startet in der linken unteren Ecke des Bildschirm ( $0,14$ ) und blickt nach rechts. Er soll sich über die drei, beliebig hohen, Hindernisse vor ihm bewegen. Wir wissen, dass sich zwischen den Hürden jeweils genau ein freie Feld befindet. Bouncer kann nur eine Hürde auf ein mal überqueren; er muss also über jedes Hindernis einzeln klettern. Am Ende soll Bouncer auf dem Feld rechts unten neben der letzten Hürde stehen

Wie schaut der Algorithmus aus? Welche Teilaufgabe muss wie oft erledigt werden? Aus welchen Abschnitten besteht diese Teilaufgabe?

# DOKUMENTIEREN UND KOMMENTIEREN VON PROGRAMMEN

Sie können an jeder Stelle Ihres Programmes Kommentare einfügen – diese werden vom Computer bei der Ausführung nicht beachtet und dienen Ihnen und anderen Entwicklern und Entwicklerinnen zum besseren Verständnis der Software. Hier werden die Ein- und Ausgangsbedingungen einer Methode beschrieben.

```
1 /**
2 * Moves bouncer to the next free field by jumping over a hurdle
3 * Pre-condition: Bouncer stands in front of a hurdle, facing east
4 * Post-condition: Bouncer stands behind hurdle, facing east
5 */
6 private void jumpOverHurdle() {
7     ascendHurdle();
8     bouncer.move();
9     descendHurdle();
10 }
```

# ZUSAMMENFASSUNG UND AUFGABEN

# ZUSAMMENFASSUNG (1/2)

- In diesem Kurs lernen Sie einen wichtigen Teil der Grundlagen, die Sie für ein erfolgreiches Studium der Medieninformatik benötigen.
- Es geht in diesem Kurs (und im späteren Studium) darum, Anwendungen für bestehende Probleme zu entwickeln – Das Werkzeug dazu sind Computer und Programmiersprachen.

# ZUSAMMENFASSUNG (2/2)

- Anweisungen werden in Java/Bouncer in Methoden zusammengefasst – innerhalb der Methoden können andere Methoden aufgerufen werden.
- Der Programmablauf eines Java/Bouncer-Programms kann über Kontrollstrukturen gesteuert werden: for-Schleife, while-Schleife und if-Abfrage.
- Ein Algorithmus löst Probleme, in dem er ein System durch eine Abfolge von Anweisungen in einen vorher definierten Endzustand überführt.

# TODOS (1/3)

- Falls noch nicht geschehen: Melden Sie sich **jetzt** für die Übungen an!
- Klären Sie offene Fragen zur Anmeldung oder Anrechnung möglichst bald
- Stellen Sie sicher, dass Sie im GRIPS-Kurs angemeldet sind

# TODOS (2/2)

- Arbeiten Sie die Inhalte von Vorlesung und Zentralübung durch
- Richten Sie Ihre persönliche Arbeitsumgebung ein und probieren Sie das bereitgestellten Testprojekten aus
- Besuchen Sie die ihnen zugewiesene Übungssitzung

# VIELEN DANK FÜR IHRE AUFMERKSAMKEIT. WIR SEHEN UNS IM ANSCHLUS 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

Micheal R. Fellows, Ian Parberry, *SIGACT trying to get children excited about CS, Computing Research News, Januar 1993 Seite 7ff*, Computing Research Association

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