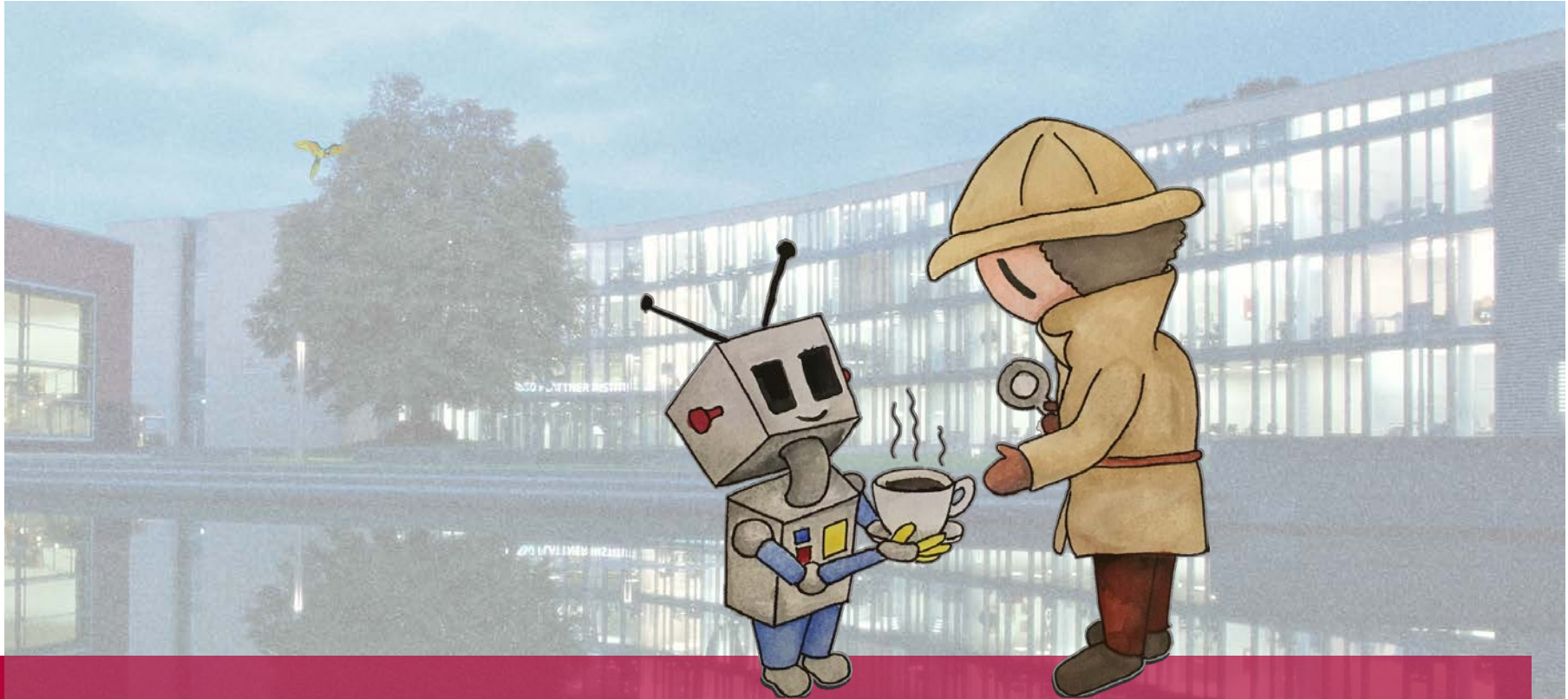




Objektorientierte Programmierung mit Java – Woche 1

openHPI-Java-Team
Hasso-Plattner-Institut



Ein erstes Programmierbeispiel

openHPI-Java-Team

Hasso-Plattner-Institut



```
1  class HelloPaco{  
2  
3  
4  
5  }
```

Klasse

- Grundeinheit der objektorientierten Programmierung
- Schlüsselwort `class`



Syntax: Geschweifte Klammern

```
1  class HelloPaco{  
2  
3  
4  
5  }
```

Geschweifte Klammern

- Strukturieren unser Programm in Code-Blöcke
- Inhalt zwischen { und } wird für den aktuellen Bezeichner ausgewertet (in diesem Fall `class`)
- Code zwischen { und } (Code-Block) wird nach Konvention eingerückt



```
1  class HelloPaco{  
2      public static void main(String[] args){  
3          //hier startet unser Programm  
4      }  
5  }
```

main()-Methode

- Gibt es genau einmal pro Programm



```
1  class HelloPaco{
2      public static void main(String[] args){
3          // ein einzeliger Kommentar
4          /* ein mehrzeiliger
5             Kommentar */
6      }
7  }
```

Kommentare

- Werden bei der Programmausführung ignoriert
- Dienen der Erklärung des Programmes für Entwickler



```
1  class HelloPaco{
2      public static void main(String[] args){
3          System.out.println("Hallo Paco");
4      }
5  }
```

System.out.println()

- gibt eine Zeile aus
- Englisch: print line
- Ist eine Methode



```
1 class HelloPaco{
2     public static void main(String[] args){
3         System.out.println("Hallo Paco");
4     }
5 }
```

Ausgabertext

Ausgabertext

- Text innerhalb der Klammern wird ausgegeben
- Anführungszeichen sorgen dafür, dass es als Text erkannt wird
- In Java wird Text auch String genannt
- "Hallo Paco" wird Argument der Methode genannt



Syntax: Semikola

```
1  class HelloPaco{  
2      public static void main(String[] args){  
3          System.out.println("Hallo Paco");  
4      }  
5  }
```

Semikolon

- Damit endet jede Anweisung



```
1  class HelloPaco{
2      public static void main(String[] args){
3          System.out.println("Hallo Paco");
4      }
5  }
```

Ausgabe:

Hallo Paco



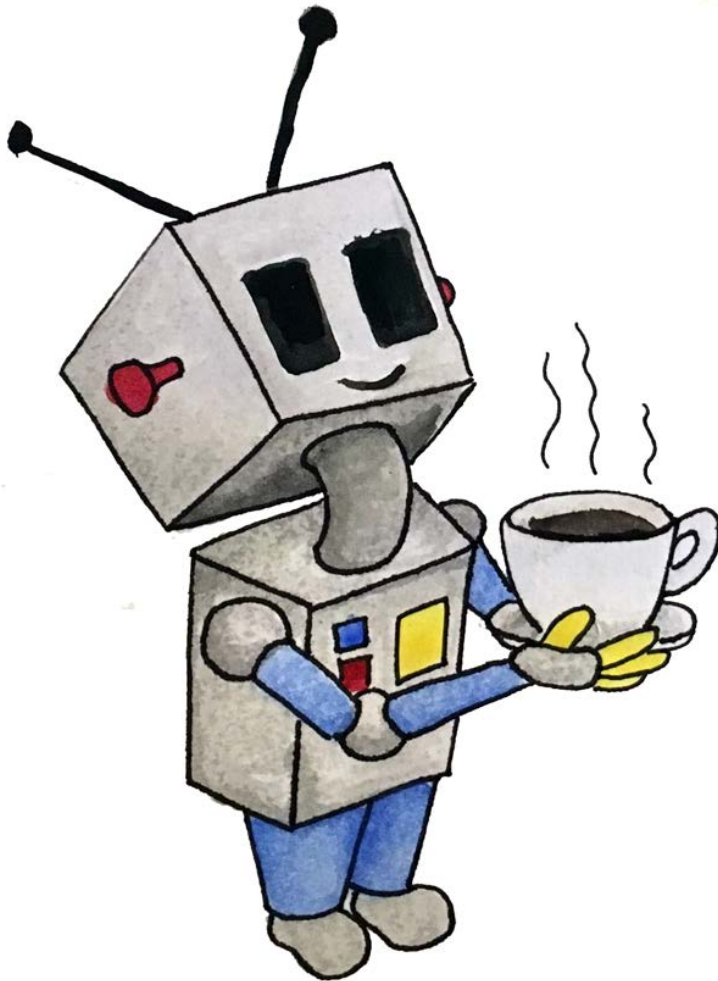
Objektorientierung: Klassen und Objekte

openHPI-Java-Team
Hasso-Plattner-Institut



Objektorientierte Programmierung (OOP)

- Strukturierung nach menschlicher Denkweise
- Abbilden von Gegenständen durch Klassen und Objekte
- Zusammenspiel kooperierender Objekte

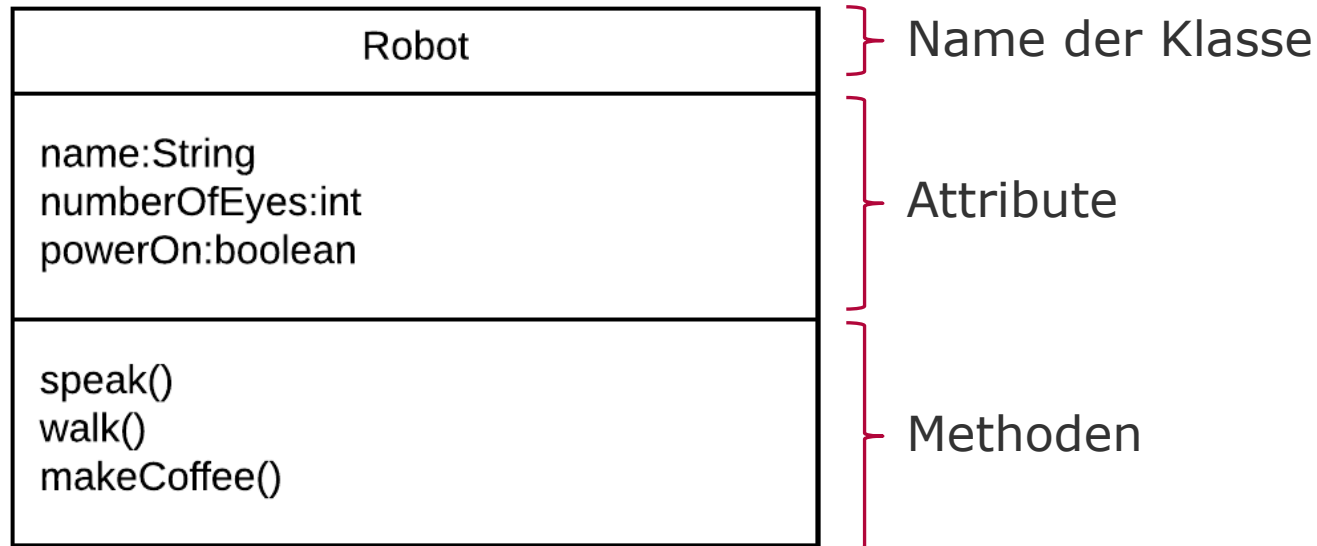


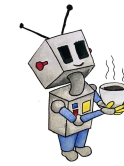
Zustand:

- Name: Robin
- Anzahl Augen: 2
- Eingeschaltet: ja
- ...

Verhalten:

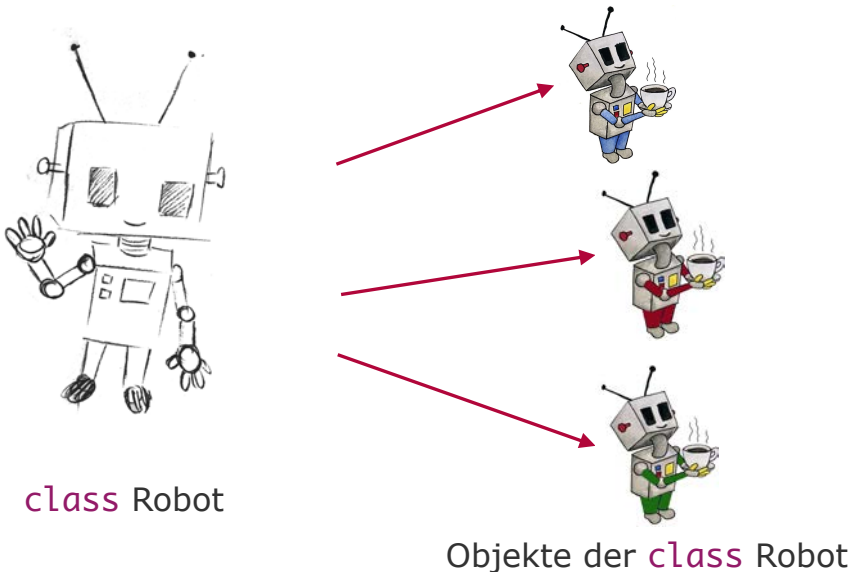
- Kann sprechen
- Kann laufen
- Kann Kaffee kochen
- ...





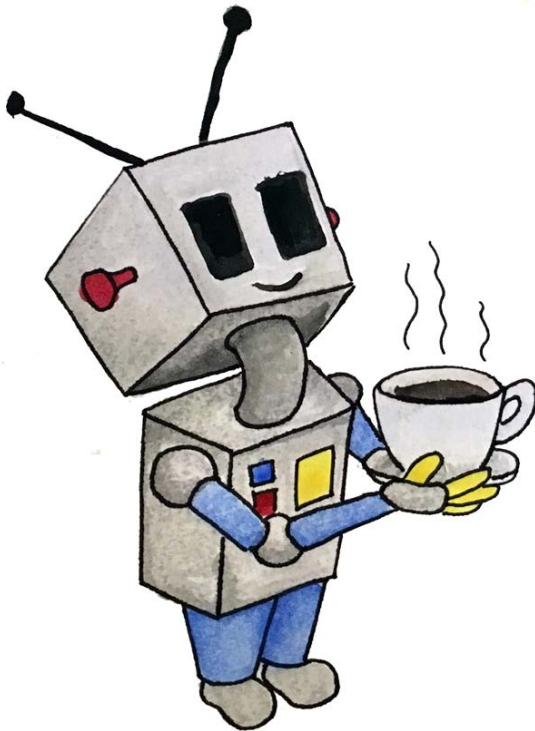
Klassen (**class**)

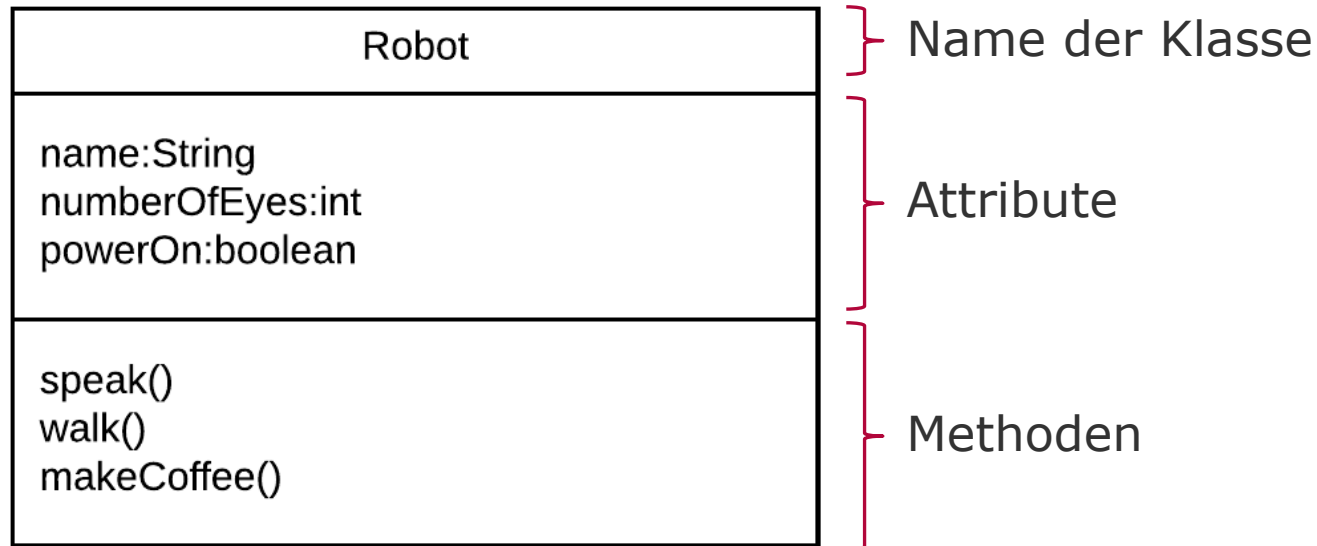
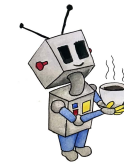
- Sind Baupläne für Konstrukte aus der realen Welt
- Haben Zustand (Attribute) und Verhalten (Methoden)
- Eine Klasse ist eine Vorlage für viele Objekte
- **Achtung!** Die Klasse Robot ist kein Roboter sondern nur eine Vorlage für viele konkrete Roboter!





- Instanz (konkrete Umsetzung) einer Klasse
- Attribute haben eigene Werte





Implementierung der Klasse Robot



```
1  class Robot{
2      String name;
3      int numberOfEyes;
4
5      void speak(){
6          //...
7      }
8      void walk(){
9          //...
10     }
11 }
```

} Name der Klasse

} Attribute

} Methoden



State

- Zustand eines Objektes
- Beschrieben durch Attribute

Behaviour

- Verhalten eines Objektes
- Beschrieben durch Methoden

Der Zustand (**state**) eines Objektes beeinflusst sein Verhalten (**behaviour**). Das Verhalten eines Objektes beeinflusst seinen Zustand.

Beispiele:

Das Gewicht eines Roboters beeinflusst seine Geschwindigkeit.

Die Methode `walk()` beeinflusst die Position des Roboters.



Variablen (1/2)

openHPI-Java-Team
Hasso-Plattner-Institut



- Container für Daten
- Benötigen einen eindeutigen Variablennamen
- Format: <Datentyp> <Bezeichner>;

String output;

↑ ↑

Datentyp Bezeichner



Wiederverwendung von Variablen

```
1 class HelloPaco{
2     public static void main(String[] args){
3         String output;
4         output = "Hallo Paco";
5         System.out.println(output);
6         System.out.println(output);
7     }
8 }
```

Ausgabe:

Hallo Paco

Hallo Paco



- Container für Daten
- Benötigen einen eindeutigen Variablennamen
- Format: <Datentyp> <Bezeichner> = <Wert>;

```
String output = "Hallo Paco";
```

↑ ↑ ↑

Datentyp Bezeichner Wert



Wiederverwendung von Variablen

```
1 class HelloPaco{
2     public static void main(String[] args){
3         String output = "Hallo Paco";
4         System.out.println(output);
5         System.out.println(output);
6     }
7 }
```

Ausgabe:

Hallo Paco

Hallo Paco



Neuzuweisung von Variablen

```
1  class HelloPaco{
2      public static void main(String[] args){
3          → String output = "Hallo Paco";
4              System.out.println(output);
5          → output = "Hallo Duke";
6              System.out.println(output);
7      }
8  }
```

Datentyp

- wird nur initial angegeben
- Wird nach der Initialisierung nicht mehr hingeschrieben
 - Initialisieren ist das Anlegen einer Variable



Neuzuweisung von Variablen

```
1 class HelloPaco{
2     public static void main(String[] args){
3         String output = "Hallo Paco";
4         System.out.println(output);
5         output = "Hallo Duke";
6         System.out.println(output);
7     }
8 }
```

Ausgabe:

Hallo Paco

Hallo Duke



Name	Art	Beispiele
String	Text, Zeichenkette	String name = "Robin";
char	Buchstabe, Zeichen	char country = 'd'; char cedille = 'ç';
int	Ganzzahl	int age = 2; int truth = -42;
double	Kommazahl	double speed = 98.7;



Verkettung von Datentypen (1/4)

```
1 void print(){  
2     String output = "Hallo Paco";  
3     System.out.println(output + "!");  
4 }
```

Strings

- Werden mit + zusammengefügt



```
1 void print(){  
2     String output = "Hallo Paco";  
3     System.out.println(output + "!");  
4 }
```

Ausgabe:

Hallo Paco!



Verkettung von Datentypen (3/4)

```
1 void print(){  
2     String welcome = "Hallo";  
3     String name = "Paco";  
4     System.out.println(welcome + " " + name);  
5 }
```

Formatierung

- Auf Leerzeichen zwischen Wörtern achten



```
1 void print(){
2     String welcome = "Hallo";
3     String name = "Paco";
4     System.out.println(welcome + " " + name);
5 }
```

Ausgabe:

Hallo Paco



Variablen (2/2)

openHPI-Java-Team
Hasso-Plattner-Institut



Verkettung von Datentypen (1/2)

```
1 void print(){  
2     String first = "5";  
3     String second = "5";  
4     System.out.println(first + second);  
5 }
```

Ausgabe:

55



Verkettung von Datentypen (2/2)

```
1 void print(){  
2     int number1 = 5;  
3     int number2 = 5;  
4     System.out.println(number1 + number2);  
5 }
```

Ausgabe:

10

Vorsicht!

- Addition bei Integers
- Konkatenation bei Strings

Eine Auswahl an Datentypen



Name	Art	Beispiele
String	Text, Zeichenkette	String name = "Robin";
char	Buchstabe, Zeichen	char country = 'd'; char cedille = 'ç';
int	Ganzzahl	int age = 2; int truth = -42;
double	Kommazahl	double speed = 98.7;



```
1 void printCases(){
2     int solvedCases = 0;
3     int unsolvedCases = 1;
4     int numberOfCases = solvedCases + unsolvedCases;
5     System.out.println(numberOfCases);
6 }
```

Ausgabe:

1



```
1 void printCases(){  
2     int solvedCases = 0;  
3     int unsolvedCases = 1;  
4     System.out.println(solvedCases + unsolvedCases);  
5 }
```

Formatierung

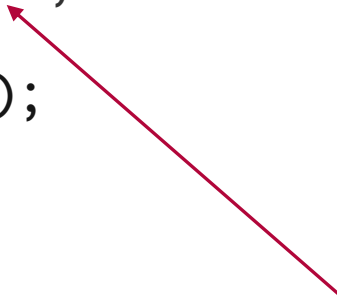
- Kein Semikolon innerhalb der Klammern

Ausgabe:

1



```
1 void print(){
2     String text = "Ich mag die Zahl ";
3     int number = 7;
4     System.out.println(text + number);
5 }
```



Java Ausgabe

- Erlaubt auch verschiedene Datentypen
- Tipp: Leerzeichen mit in den String (`text`) schreiben

Ausgabe:

Ich mag die Zahl 7



```
1 void calculate(){
2     int number1 = 42;
3     int number2 = 21;
4     System.out.println(number1 - number2);
5     System.out.println(number1 / number2);
6     System.out.println(number2 * 2);
7 }
```

Ausgabe:

21

2

42


Operatoren der Grundrechenarten bei `int`



Operator	Operation
+	Addition
-	Subtraktion
*	Multiplikation
/	Division (ganzzahliger Anteil bei <code>int</code>)
%	Modulo (Rest der Division bei <code>int</code>)



```
1 void calculate(){  
2     int number1 = 0;  
3     number1 = number1 + 2;  
4  
5 }
```



Auswertungsreihenfolge

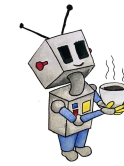
- Es wird immer erst die rechte Seite ausgewertet
- Und dann der linken Seite zugewiesen



```
1 void calculate(){  
2     int number1 = 0;  
3     number1 = number1 + 2;  
4     System.out.println(number1);  
5 }
```

Ausgabe:

2



```
1 void calculate(){  
2     int number1 = 22;  
3     number1 = number1 + 2;  
4     System.out.println(number1);  
5 }
```

Ausgabe:

24



```
1 void calculate(){  
2     int number = 0;  
3     number = number + 2;  
4     number = number + 1;  
5 }
```

```
1 void calculate(){  
2     int number = 0;  
3     number += 2;  
4     number++;  
5 }
```

Operation	Kurzschreibweise
Addition	+=
Subtraktion	-=
Multiplikation	*=
Division	/=
Inkrementieren um 1	++
Dekrementieren um 1	--



```
1 void calculate(){
2     int number1 = 5;
3     number1 /= 2;
4     System.out.println(number1);
5 }
```

Ausgabe:

2

Achtung! Die Division bei `int` errechnet nur den ganzzahligen Anteil



```
1 void calculate(){
2     int number1 = 5;
3     number1 = number1 % 2;
4     System.out.println(number1);
5 }
```

Ausgabe:

1

Achtung! Der Modulo-Operator liefert den Rest der ganzzahligen Division

$$5 = 2 * 2 + 1$$

Eine Auswahl an Datentypen



Name	Art	Beispiele
String	Text, Zeichenkette	String name = "Robin";
char	Buchstabe, Zeichen	char country = 'd'; char cedille = 'ç';
int	Ganzzahl	int age = 2; int truth = -42;
double	Kommazahl	double speed = 98.7;

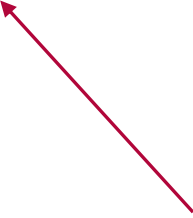


Operator	Operation
+	Addition
-	Subtraktion
*	Multiplikation
/	Division



Rechnen mit Fließkommazahlen

```
1 void calculate(){  
2     double weight = 42.2;  
3     weight--;  
4     weight *= 2;  
5 }
```



Fließkommazahlen

- Werden in Java zum Beispiel durch `double` dargestellt
- **Achtung!** Java nutzt einen Punkt statt einem Kommazeichen um Vor- und Nachkommastellen zu trennen
 - Also 3.1415 statt 3,1415

Division bei `double`



```
1 void calculate(){
2     double number1 = 5;
3     number1 /= 2;
4     System.out.println(number1);
5 }
```

Ausgabe:

2.5



```
1 void calculate(){  
2     double number1 = 0.3;  
3     double number2 = 0.1;  
4     System.out.println(number1 - number2);  
5 }
```

Ausgabe:

0.19999999999999998

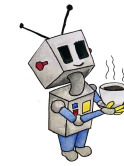
Vorsicht! bei Verwendung von `double` und anderen Fließkommazahlen



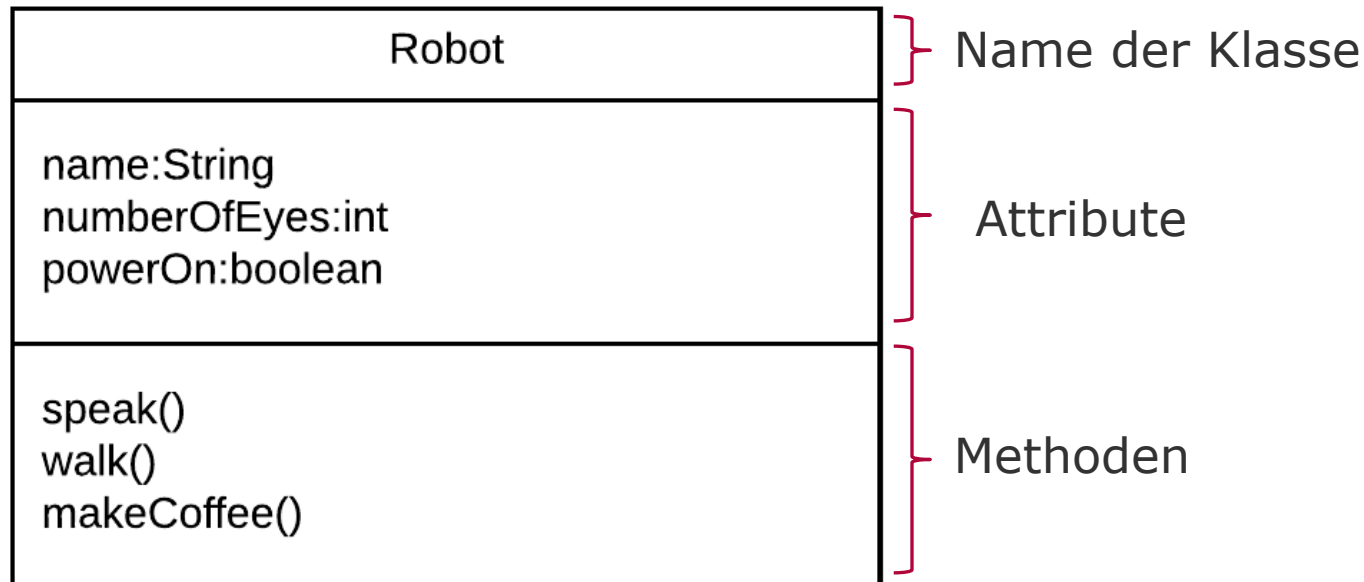
Attribute

openHPI-Java-Team

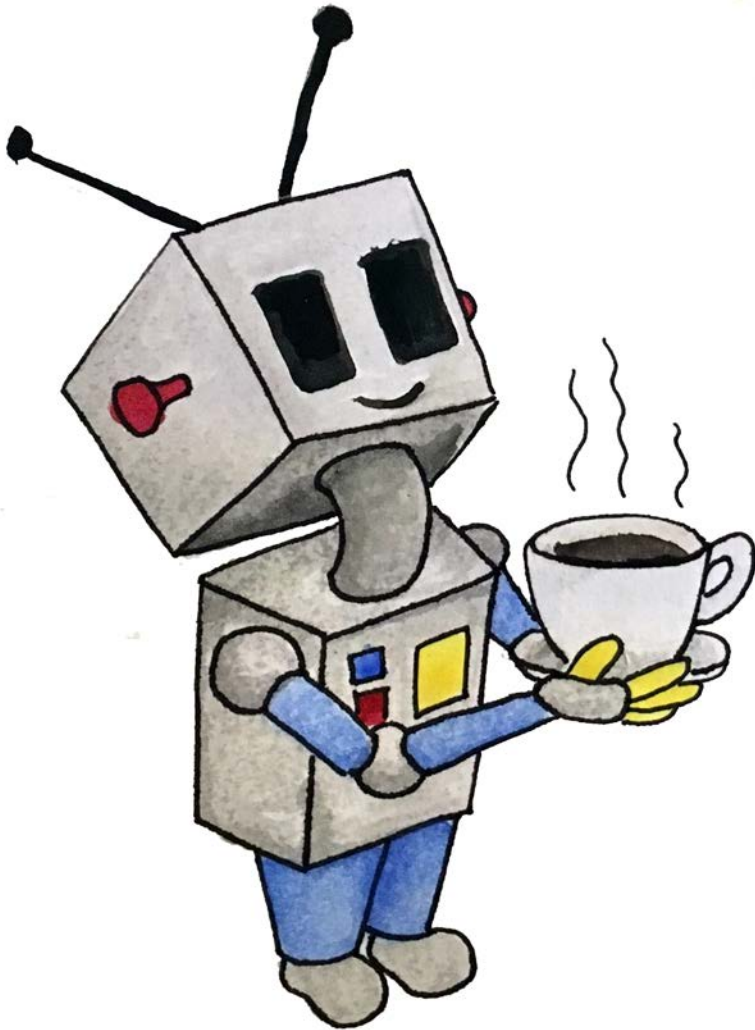
Hasso-Plattner-Institut



- Beschreiben Eigenschaften bzw. Merkmale der Klasse
- Format in Java: `<Datentyp> <Bezeichner>;`



Der Name unseres Roboters



Datentyp: String

Bezeichner des Attributs: `name`

Wert: `"Robin"`



```
1 class Robot {  
2     String name;  
3     //more attributes  
4  
5     //methods here  
6 }
```

Attribute

- Werden innerhalb der Klasse definiert
- Alle Objekte dieser Klasse haben jeweils diese Attribute
- Syntax: <Datentyp> <Bezeichner>;



```
1 class Robot {  
2     String name = "Robin";  
3 }
```

Syntax für Attribute

- Mit Defaultwert: <Datentyp> <Bezeichner> = <DefaultWert>;
- Damit wird das Attribut initial für alle Objekte auf den Defaultwert gesetzt
 - Der Wert des Attributes kann für jedes Objekt wieder geändert werden



```
1 Robot ourRobot = new Robot();
```

Instanziierung

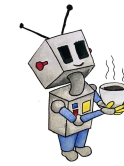
- Hier wird ein Objekt der Klasse Robot erzeugt
- Der Bezeichner `ourRobot` dient zur Referenzierung des Objektes
- Objekte einer Klasse werden mit
<Klassenname> <Bezeichner> = `new` <Klassenname>(); instanziiert

Instanziierung von Objekten mit `new`



```
1 class Story{  
2     public static void main(String[] args) {  
3         Robot ourRobot = new Robot();  
4     }  
5 }
```

- Code erzeugt ein Objekt der Klasse Robot
- Durch den Bezeichner `ourRobot` ist dieses Objekt wiederauffindbar



```
1 class Story{
2     public static void main(String[] args){
3         Robot ourRobot = new Robot();
4         ourRobot.name = "Robin";
5         System.out.println(ourRobot.name);
6     }
7 }
```

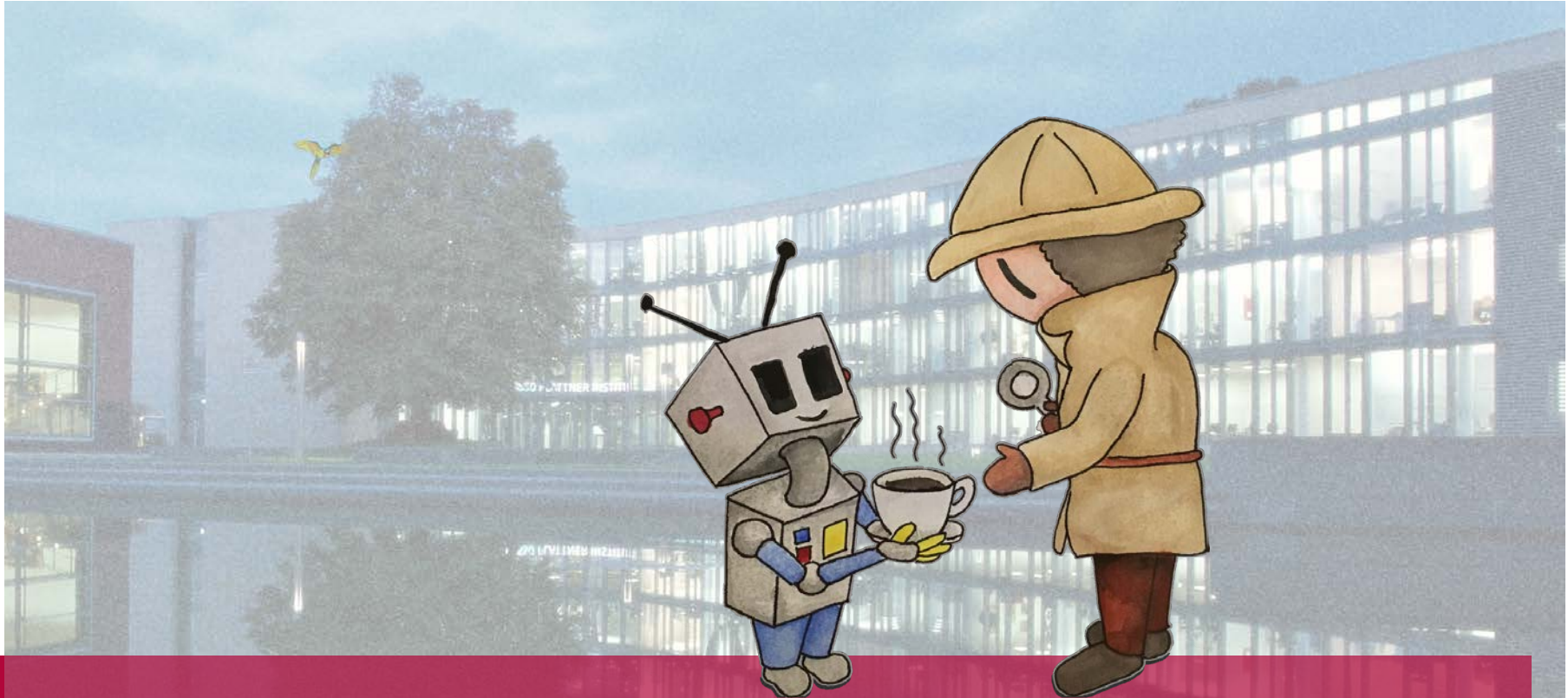
Ausgabe:

Robin

Punkt-Operator (Dot-Operator)

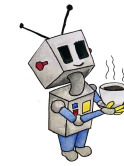
- Mit dem Punkt-Operator können wir auf Attribute und Methoden zugreifen

Achtung! Zuweisen eines Wertes ist hier notwendig

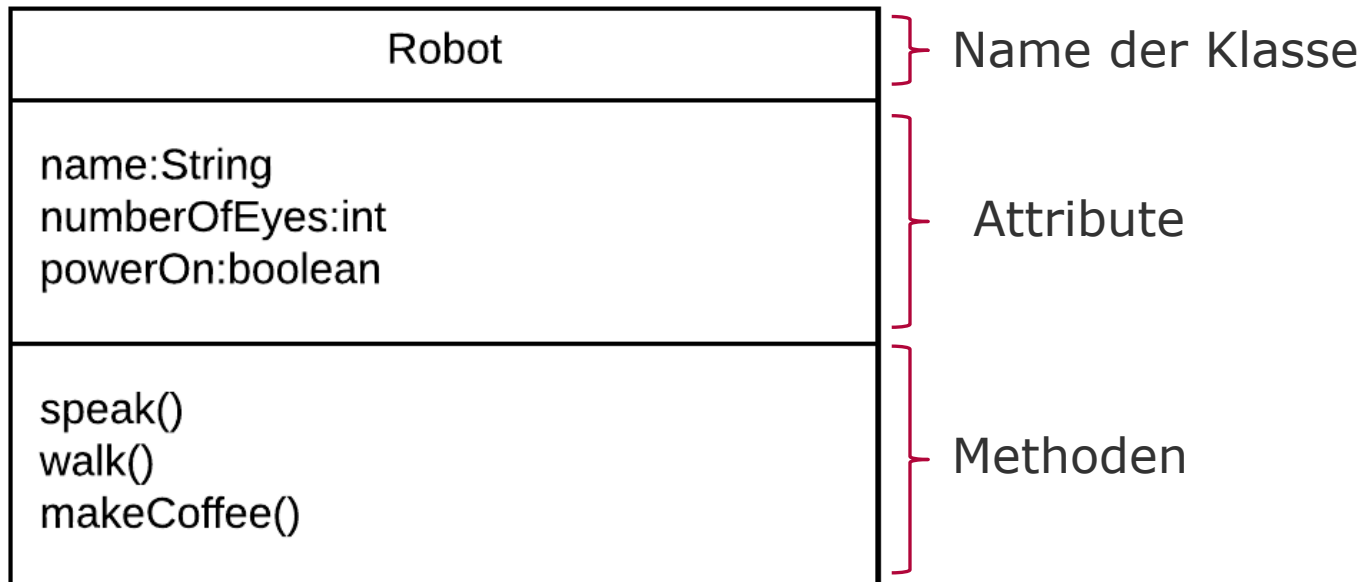


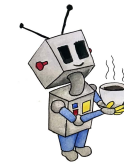
Methoden

openHPI-Java-Team
Hasso-Plattner-Institut



- Spezifizieren Operationen für alle Objekte einer Klasse
- Methodendefinitionen erfolgen **immer** innerhalb von Klassen
- Methoden definieren Verhalten





```
1 class Robot{  
2     void speak(){  
3         System.out.println("Hallo");  
4     }  
5 }
```

Methoden

- Syntax: <Rückgabetyp> <methodName>(){}
■ In den Codeblock zwischen { und } schreiben wir Anweisungen
 - Diese werden ausgeführt, wenn die Methode aufgerufen wird



Beispiel: leere Rückgabe

```
1 class Robot{  
2     void speak(){  
3         System.out.println("Hallo");  
4     }  
5 }
```

void

- Bedeutet, dass die Methode nichts zurück gibt
- "leere Rückgabe"



```
1 class Robot{  
2     void speak() { ←  
3         System.out.println("Hallo");  
4     }  
5 }  
6 class Story{  
7     static void main(String[] args){ ←  
8         Robot robin = new Robot();  
9         robin.speak(); ←  
10    }  
11 }
```

1. Programm startet in main()
2. Instanziierung des Objekts `robin`
3. Aufruf von `speak()` auf dem Objekt `robin` (der Klasse `Robot`)



```
1 class Robot{  
2     void speak() { ←  
3         System.out.println("Hallo");  
4     }  
5 }  
6 class Story{  
7     static void main(String[] args){ ←  
8         Robot robin = new Robot();  
9         robin.speak(); ←  
10    }  
11 }
```

Allgemeine Form: `objectName.methodName();`

Ausgabe:

Hallo



```
1 class Robot{  
2     void speak() { ←  
3         System.out.println("Hallo");  
4     }  
5 }  
6 class Story{  
7     static void main(String[] args){ ←  
8         Robot robin = new Robot();  
9         robin.speak(); ←  
10    }  
11 }
```

- Der Methodenaufruf von `speak()` erfolgt aus der Klasse `Story` auf dem Objekt der Klasse `Robot`
 - also außerhalb der Klasse in der `speak()` definiert wurde



```
1 class Robot {  
2     void sayHelloWorld() { ←  
3         sayHello(); ←  
4         System.out.println("Welt");  
5     }  
6     void sayHello() { ←  
7         System.out.println("Hallo");  
8     }  
9 }
```

1. Aufruf von `sayHelloWorld()` von außen
2. `sayHelloWorld()` ruft `sayHello()` auf
3. Nach Ausführen von `sayHello()` wird der Rest von `sayHelloWorld()` ausgeführt



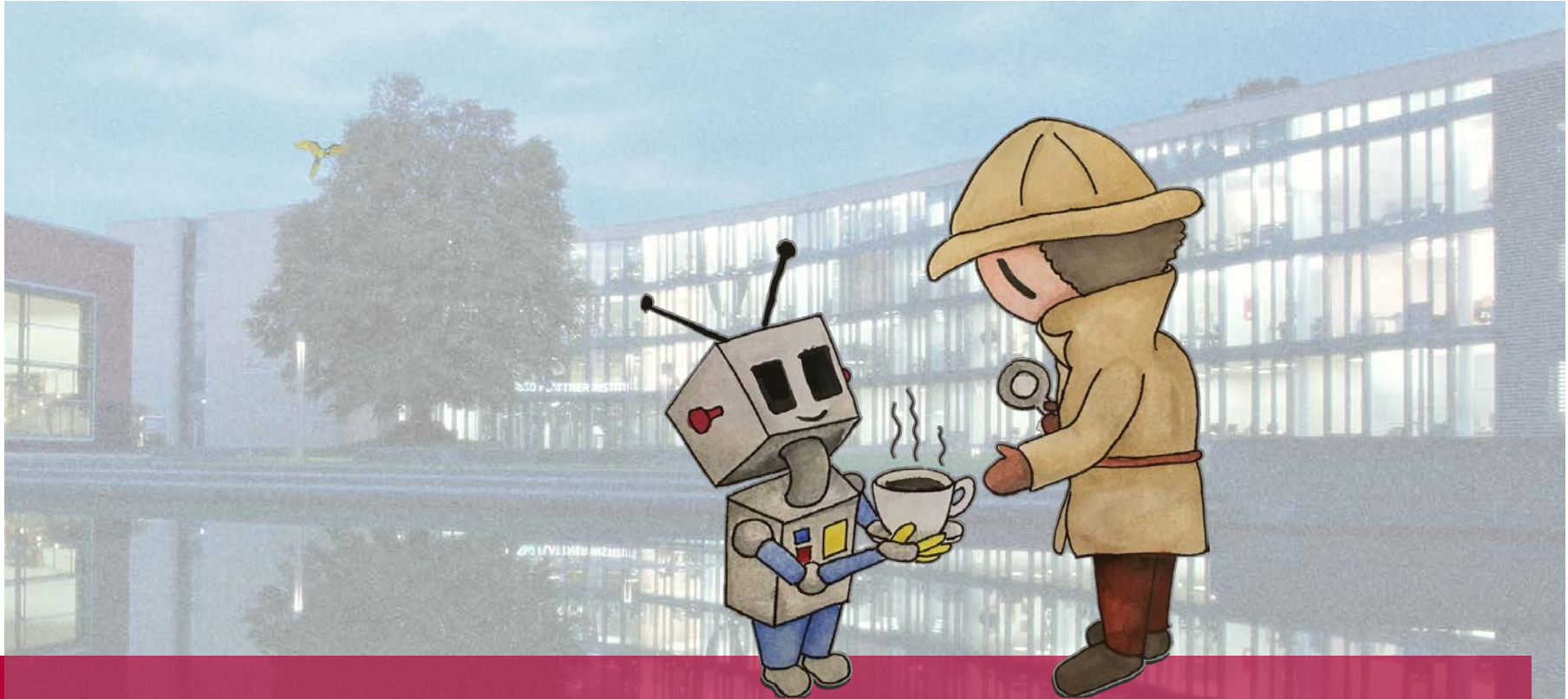
```
1 class Robot {  
2     void sayHelloWorld() {  
3         sayHello();  
4         System.out.println("Welt");  
5     }  
6     void sayHello() {  
7         System.out.println("Hallo");  
8     }  
9 }
```

Allgemein: `methodName()`;
ruft Methoden innerhalb derselben Klasse auf

Ausgabe:

Hallo

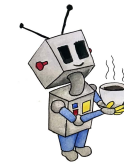
Welt



Methoden mit Rückgabewerten

openHPI-Java-Team

Hasso-Plattner-Institut



```
1 class Robot{  
2     void speak(){  
3         System.out.println("Hallo");  
4     }  
5 }
```

Rückgabetypen

- Datentyp oder **void**
- Ein Rückgabetyp muss immer bei der Methodendeklaration angegeben werden

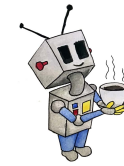


Beispiel: keine Rückgabe

```
1 class Robot{  
2     void speak(){  
3         System.out.println("Hallo");  
4     }  
5 }
```

Wiederholung: void

- Bedeutet, dass die Methode nichts zurück gibt



Beispiel: Rückgabe von Strings

```
1 class Robot{  
2     String getName(){  
3         return "Robin";  
4     }  
5 }
```

Rückgabebetyp String

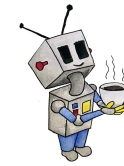
- Bedeutet, dass die Methode einen String zurückgibt
- `return` gibt diesen String zurück (in diesem Fall "Robin")

return



```
<Rückgabetyp> <Methodenname>(){  
    //mehr Quellcode  
    return <Rückgabewert>;  
}
```

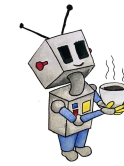
- Rückgabewert muss dem Rückgabetyp entsprechen
- **return** gibt den Rückgabewert an die aufrufende Methode zurück



Verwendung von Rückgabewerten

```
1 class Robot{
2     String getName(){
3         return "Robin";
4     }
5 }
6 class Story{
7     static void main(String[] args){
8         Robot robin = new Robot();
9         String robotName = robin.getName();
10        //mehr Quellcode
11    }
12 }
```

- Nutzen Zuweisung um den Rückgabewert zu nutzen
- Können anschließend `robotName` wie gewohnt weiter verwenden



```
1 class Robot{
2     String getName(){
3         return "Robin";
4     }
5 }
6 class Story{
7     static void main(String[] args){
8         Robot robin = new Robot();
9         String robotName = robin.getName();
10        System.out.println(robotName);
11    }
12 }
```

- Benutzen Zuweisung um die Rückgabe der Methode zu speichern
- Können anschließend die Variable `robotName` wie gewohnt weiter verwenden

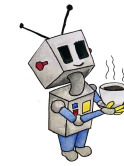


```
1 class Robot{
2     String getName(){
3         return "Robin";
4     }
5 }
6 class Story{
7     static void main(String[] args){
8         Robot robin = new Robot();
9         String robotName = robin.getName();
10        System.out.println(robotName);
11    }
12 }
```

Ausgabe:

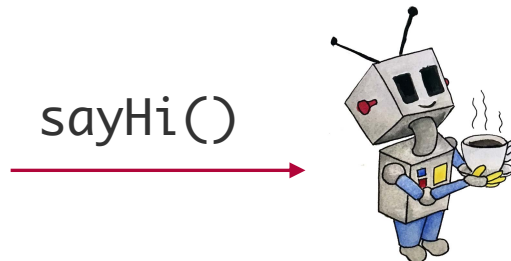
Robin

Ein etwas komplexeres Beispiel (1/3)

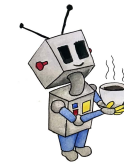


```
1 class Story{  
2     public static void main(String[] args){  
3         Robot robo1 = new Robot();  
4         robo1.sayHi();  
5     }  
6 }
```

- Aufruf der Methode sayHi() auf robo1



Ein etwas komplexeres Beispiel (2/3)



```
1 class Robot{
2     String name = "Robin"; ←
3     String sayHi(){ ←
4         String me = getName(); ←
5         System.out.println("Hi, ich bin " + me);
6     }
7     String getName(){ ←
8         return name; ←
9     }
10 }
```

1. Aufruf von sayHi() von außen

2. Aufruf von getName()

a) Zugriff auf das Attribut name der Klasse Robot

b) Der Rückgabewert ("Robin") vom Typ String wird mit return zurückgegeben

3. Dieser wird der Variable me zugewiesen

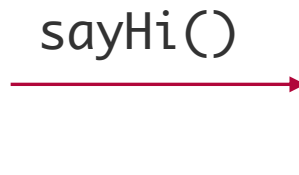
Ein etwas komplexeres Beispiel (3/3)



```
1 class Story{  
2     public static void main(String[] args){  
3         Robot robo1 = new Robot();  
4         robo1.sayHi();  
5     }  
6 }
```

Ausgabe:

Hi, ich bin Robin



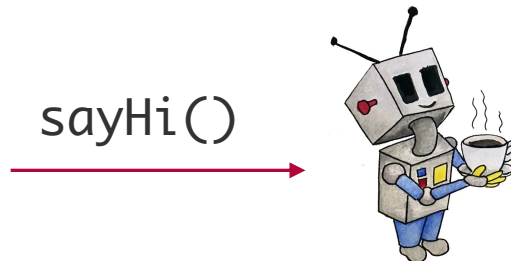
Hi, ich bin Robin

Ein noch komplexeres Beispiel (1/3)



```
1 class Story{  
2     public static void main(String[] args){  
3         Robot robo1 = new Robot();  
4         System.out.println(robo1.sayHi());  
5     }  
6 }
```

- Aufruf von sayHi() auf robo1



Ein noch komplexeres Beispiel (2/3)



```
1 class Robot{
2     String name = "Robin";
3     String sayHi(){ ←
4         return "Hi, ich bin " + getName();
5     }
6     String getName(){ ←
7         return name;
8     }
9 }
```

1. Aufruf von getName()
2. getName() gibt den Wert des Attributs `name` zurück
 - a) Der Rückgabewert ("`Robin`") vom Typ `String` wird mit `return` zurückgegeben
3. sayHi() gibt den konkatenierten Satz zurück

Ein noch komplexeres Beispiel (3/3)

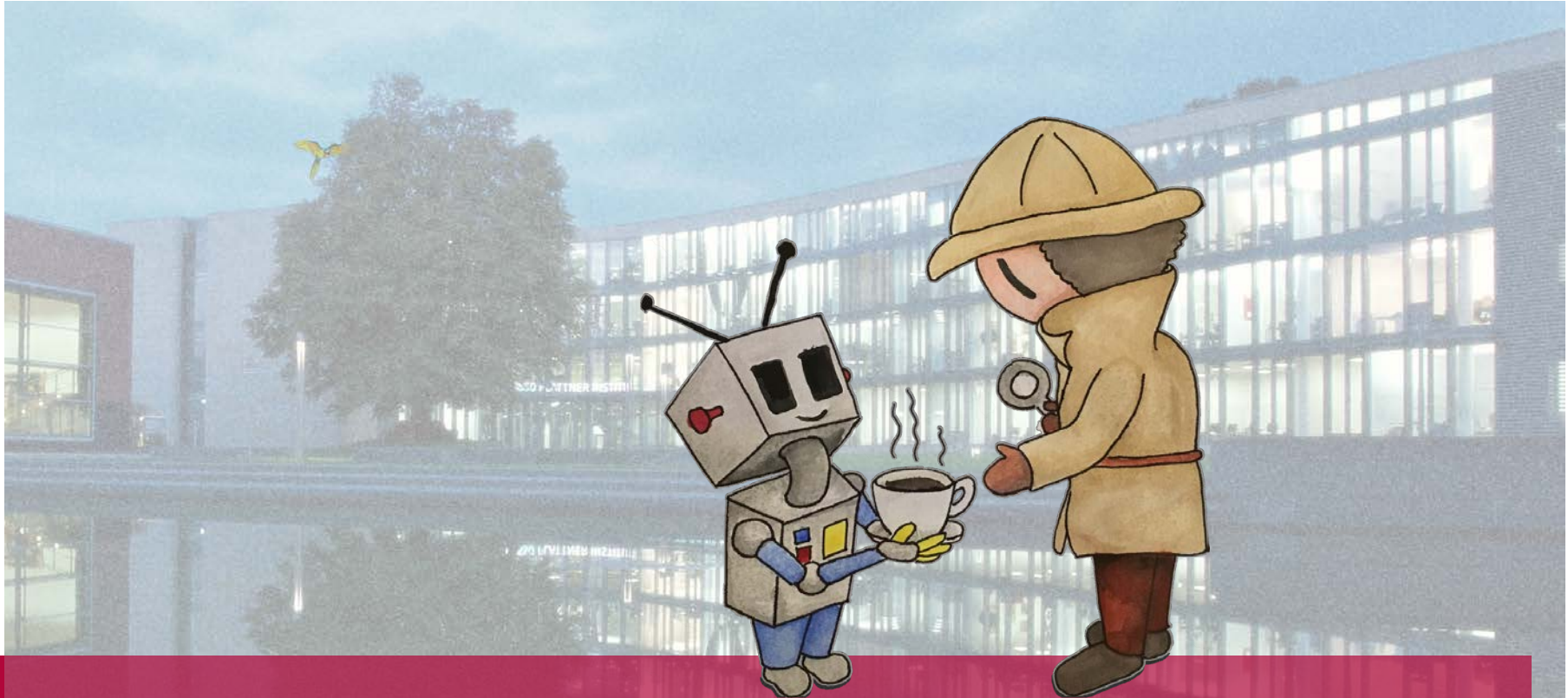


```
1 class Story{
2     public static void main(String[] args){
3         Robot robo1 = new Robot();
4         System.out.println(robo1.sayHi());
5     }
6 }
```

Ausgabe:

Hi, ich bin Robin





Deep Dive Java

openHPI-Java-Team
Hasso-Plattner-Institut



```
1  class Story{  
2      String title = "Finde Paco";  
3      public static void main(String[] args){  
4          Robot ourLittleRobot = new Robot();  
5      }  
6  }
```

Konventionen

- Klassennamen werden groß geschrieben
- Objekte, Attribute und Methodennamen werden klein geschrieben (Bezeichner)
- CamelCase
- Sinnvolle Bezeichner wählen



print() vs println()

```
1 class HelloPaco{
2     public static void main(String[] args){
3         System.out.print("Hallo ");
4         System.out.print("Paco");
5         System.out.println();
6     }
7 }
```

print()

- Kein impliziter Zeilenumbruch

println()

- Impliziter Zeilenumbruch

print() vs println()



```
1  class HelloPaco{
2      public static void main(String[] args){
3          System.out.print("Hallo ");
4          System.out.print("Paco");
5          System.out.println();
6      }
7  }
```

Ausgabe:

Hallo Paco



Verschiedene Anführungszeichen

```
1 class FooBar{  
2     String text = "Hallo Paco";  
3     char cedille = 'ç';  
4     //methods here  
5 }
```

Anführungszeichen

- chars werden in einfache Anführungszeichen (' ') geschrieben
- Strings werden in doppelte Anführungszeichen (" ") geschrieben
- **Achtung!** bei Copy und Paste von Anführungszeichen



Name	Art	Beispiele	Komplex/ primitiv
String	Text, Zeichenkette	String text; String name = "Robin";	komplex
char	Buchstabe, Zeichen	char country = 'd'; char cedille = 'ç';	primitiv
double	Kommazahl	double speed = 98.7;	primitiv
int	Ganzzahl	int age = 2; int truth = -42;	primitiv
Robot	Roboter	Robot robin = new Robot(); Robot robby = new Robot();	komplex



```
1 void stringManipulation(){
2     String output = "Hallo Paco";
3     System.out.println(output.toUpperCase());
4     System.out.println(output.charAt(7));
5     System.out.println(output.reverse());
6 }
```

- toUpperCase() übersetzt einen String komplett in Großbuchstaben
- charAt(x) gibt einem einen char an der Position x zurück
 - Das erste Zeichen der Zeichenkette wird mit 0 indexiert

Ausgabe:

HALLO PACO

a

???



```
1 void calculate(String[] args) {  
2     double x = 42.1;  
3     double y = Math.floor(x);  
4     System.out.println(y);  
5 }
```

- Math stellt Mathematik-Funktionen bereit
 - z.B. floor(), ceil(), pow(2)
- Andere Bibliotheken muss man importieren



- public
- static
- void
- class
- new
- return
- int
- char
- float
- u.v.m.



Ausgabe von Text
auf dem Bildschirm

OOP

Klassen

Objekte

Methoden

Attribute

Variablen

