Algorithmik 11. Klasse

Java ist eine sogenannte objektorientierte Programmiersprache. Das bedeutet, dass der Aufbau und das Zusammenspiel der Bestandteile (***Klassen***, ***Objekte***, ***Attribute***, ***Methoden***,…) der Programmiersprache dem dir bereits bekannten Muster folgt.

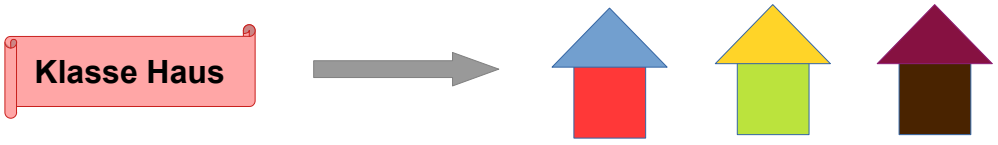
Für die Programmierung wird eine Entwicklungsumgebung benötigt, in der Programmiercode geschrieben, ausgeführt und getestet werden kann. Wir werden für den Einstieg ***Greenfoot*** und später ***BlueJ*** verwenden:

Download für alle Betriebssysteme: https://www.greenfoot.org/download

Download für alle Betriebssysteme: https://www.bluej.org/

Wiederholung der Grundlagen (AB1)

Eine ***Klasse*** ist ein Bauplan. Sie legt fest, welche ***Attribute*** (=***Eigenschaften***) und welche ***Methoden*** (=***Fähigkeiten***) ***Objekte*** dieser Klasse haben sollen. Mittels der Klasse kann man (mehrere) Objekte dieser Klasse ***erzeugen***.



Objekte der Klasse Haus

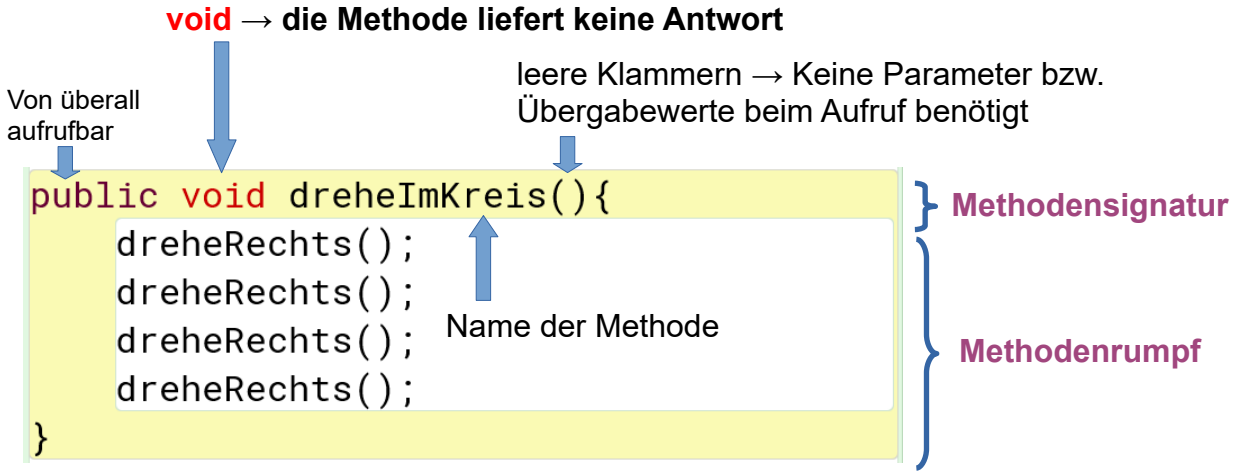
Datentypen in Java (AB1)

Datentypen legen fest, von welcher Art (logischen Struktur) ein Wert ist.

Datentyp	**Bedeutung**	**Beispielwerte**	**Anmerkung**
int	Ganze Zahl	***7 -8***	
double	Dezimalzahl	***13.24 -0.5***	Mit Dezimalpunkt
String	Zeichenkette	***“blau“ “Hallo“ “5+7“ ““***	In Anführungszeichen **Ausnahme**: Datentyp wird großgeschrieben!
boolean	Wahrheitswert	***true false***	Eigennamen für diese Werte
char	Einzelnes Zeichen	***‘c‘ ‘8‘ ‘+‘ ‘ ‘***	In Hochkommata

Einfache Methoden (AB2)

Methoden sind die Fähigkeiten von Objekten. Beispiel:



Bedingte Wiederholung mit while (AB3)

Manchmal kommt es vor, dass man bestimmte Befehle mehrfach ausführen möchte oder einen Befehl solange ausführen möchte bis eine bestimmte Bedingungen erfüllt bzw. nicht mehr erfüllt ist.

Beispiel:

```
while( ! istAufGegenstand("pilz") ) {
        einsVor();
}
aufnehmen();
```

Für die ***Bedingung*** in der while-Wiederholung werden häufig Methoden genutzt, die eine ***true/false-Antwort*** liefern: istVorne(…), istVorneFrei(), istAufGegenstand(…), usw.

Das ***!*** steht für „***Nicht***“: Solange Mario ***nicht*** auf dem Pilz ist, soll er einen Schritt nach vorne machen.

Der Aufbau für die bedingte Wiederholung wird mit dem Schlüsselwort ***while*** eingeleitet, gefolgt von einem ***runden Klammerpaar*** und einem ***geschweiften Klammerpaar***.

```
while( Bedingung ) {
        // Befehle für die Wiederholung
}
// Hier geht es weiter
```

Solange die ***Bedingung zu true ausgewertet*** wird, wird weiter der Code in der while-Schleife wiederholt.

Sobald die ***Bedingung zu false ausgewertet*** wird, wird der Code an dieser Stelle ausgeführt.

Bedingungen können auch ***Vergleiche*** sein: **==** **!=** **<** **>** **<=** **>=**
Ein Vergleich liefert immer einen (Antwort-)Wert vom Datentyp boolean (***true/false***).
Beispiele: getAnzahl() >= 1 **getEnergie() < 90** **getRotation() == 0**

Verzweigungen (if-else) (AB4)

Manchmal sollen bestimmte Befehle nur dann ausgeführt werden, wenn eine Bedingung erfüllt ist und ansonsten soll etwas anderes getan werden. ***Die Bedingung bei if funktioniert analog zu while.***

Bedingte Anweisung: WENN . . . DANN

Hin und wieder soll ein Befehl nur unter einer bestimmten Bedingung ausgeführt werden:

WENN ***if (istAufGegenstand("pilz")) {***
DANN ***aufnehmen();***
}

Fallunterscheidung: WENN . . . DANN . . . SONST

Man kann auch einee alternative Anweisung angeben.

WENN ***if (istVorneFrei()) {***
DANN ***einsVor();***
} else {
SONST ***dreheUm();***
}

Methoden mit Rückgabetyp (AB5)

Methoden können Antworten geben. Hierfür muss der ***Datentyp der Antwort*** angegeben werden und im Methodenrumpf mittels ***return*** ein Wert zurückgegeben werden. Der return-Befehl beendet zudem die Methode.

Die Methoden antworten auf eine Fragen und geben einen Wert des angegeben Datentyps zurück.

return → Befehl zur Rückgabe des Werts

```
public boolean istPortalErreicht(){
    if( istAufGegenstand("Portal") ){
        return true;
    } else {
        return false;
    }
}
```

Logische Verknüpfung von Bedingungen (AB5)

Manchmal müssen mehrere Bedingungen kombiniert werden oder die Antwort eines Vergleichs verneint werden. Hierfür stehen folgende Operatoren zur Verfügung:

UND ***&&*** ***istEnergieLeer() && getAnzahl("Akku") == 0*** //Problem

ODER ***||*** ***istWandVorne() || istEnergieLeer()*** //kein Schritt möglich

Auch verknüpfte Vergleiche liefern immer einen (Antwort-)Wert vom Datentyp boolean (***true/false***).

Merksätze:
Eine Verundung von Bedingungen ist nur dann wahr/true, wenn **alle** Teilbedingungen wahr/true sind.
Eine Veroderung von Bedingungen ist dann wahr/true, wenn **mindestens eine** Teilbedingungen wahr/true ist.

Variablen allgemein (AB6)

Variablen sind (in Java) ***Platzhalter*** für einen Wert. Dabei bekommt jeder Platzhalter bzw. Variable einen ***Datentyp*** und einen ***Namen*** zugewiesen, damit man weiß, wie man wieder auf diesen Wert zugreifen kann.

Du kannst dir Variablen wie eine Art ***Schublade oder Box*** vorstellen, die durch einen Namen gekennzeichnet ist. In diese Schublade kann immer ***nur genau ein Wert*** (auf einem Zettel) gleichzeitig abgelegt werden. Die ***Form und Größe*** der Schublade würde dem Datentyp entsprechen.

Attribute/Objektvariablen (AB6)

Bevor eine Variable genutzt werden kann, muss diese erstmals ***einmalig deklariert*** (=***Festlegung des Datentyps***) werden. Das bedeutet, bevor ein Wert in die Schublade gelegt werden kann, muss die Schublade mit der entsprechenden Form und Größe (→ Datentyp) gebaut werden.

Deklaration von Attributen

Attribute werden ***außerhalb von Methoden*** deklariert.

```
public class ROBOTER {
    int energie;
    public ROBOTER(){

    }
```

Initialisierung von Attributen (AB6)

Damit die Variable benutzt werden kann, muss diese initialisiert werden. Die ***Initialisierung*** einer Variablen ist das ***erstmalige Zuweisen eines Werts***. Bezogen auf das Modell wäre es der erste Zettel, der in der Schublade abgelegt wird, mit welchem man anschließend arbeiten kann. Vorher hat die Variable in Java den „Nullwert“ (alle Bits sind 0 → Interpretation je nach Datentyp).

Attribute werden in der ***Konstruktor-methode*** initialisiert. Das ist die Methode, die ausgeführt wird, wenn das Objekt erzeugt wird. Sie heißt genauso wie die Klasse und besitzt keinen Rückgabetyp (auch kein void!).

```
public class ROBOTER {
    int energie;
    public ROBOTER(){
        energie = 100;
    }
```

Benutzung von Variablen (AB6)

Sobald die Variablen deklariert und initialisiert sind, kann man mit ihnen bzw. mit ihren zugewiesenen Werten arbeiten. Die Art der Benutzung der Variablen ist auch abhängig vom Datentyp (→ mit Zahlen kann man rechnen und mit Zeichenketten nicht).

Variablen kann ein ***neuer*** Wert mittels **=** zugewiesen werden.

```
anzahl = 0;   name = "James Bond";   leuchtet = true;
```

Zählen mit Variablen

Mit Zahlenvariablen (Datentyp ***int*** und ***double***) kann gerechnet werden. Mit **++** kann ihr Wert um 1 erhöht und mit **--** um eins verringer werden.

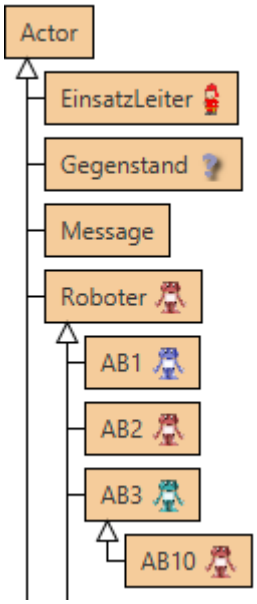
```
energie++;          energie--;
```

Get-Methoden (Methoden mit Rückgabe) (AB6)

Funktioniert analog zu Methoden mit Rückgabe aus AB5. Der Rückgabetyp muss dem Variablendatentyp entsprechen.

```
public int getEnergie(){
    return energie;
}
```

Vererbung (AB7) – Spezialisierung (Erweiterung)

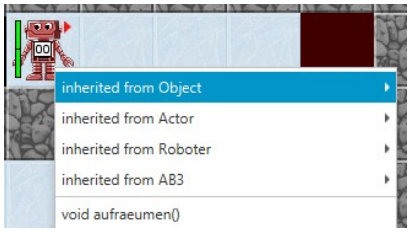


Eine Klasse kann von einer anderen Klasse mittels des Schlüsselworts ***extends*** erben.

In diesem Fall ist **Roboter** die ***Oberklasse*** bzw. ***Superklasse*** und **AB1** ist die ***Unterklasse*** bzw. ***Subklasse***.

```
public class AB1 extends Roboter
{
```

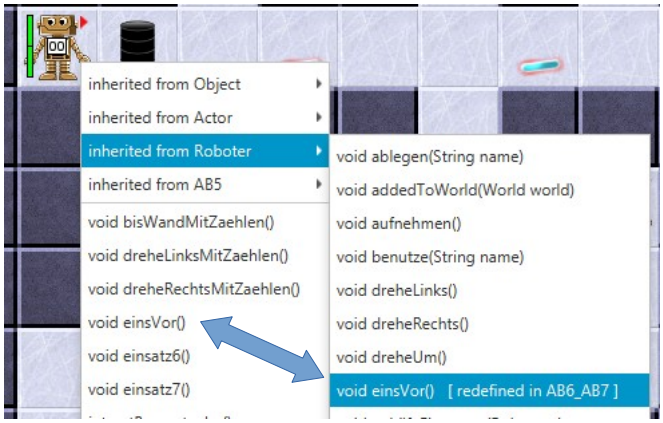
Die Klasse AB1 ***erbt alle Attribute und Methoden der Oberklasse Roboter***. Somit sind logischerweise Objekte der Klasse AB1 auch Roboter. Dies gilt auch ***transitiv***: Klasse AB10 erbt von AB3, AB3 erbt von Roboter, Roboter erbt von Actor → Somit ist AB10 ein Actor, Roboter und AB3.



In der Unterklasse AB1 können ***eigene Attribute und Methoden*** implementiert werden und so die Eigenschaften und Fähigkeiten von Roboter ***erweitert*** werden. Dadurch haben dann die Objekte von AB1 mehr Eigenschaften und Fähigkeiten als ein gewöhnlicher Roboter und sind dadurch Roboter mit ***zusätzlichen spezielleren*** Attributen und Fähigkeiten von Robotern.

Vererbung – Überschreiben von Methoden (AB7)

Bei der Spezialisierung kann es dann sein, dass Methoden der Oberklasse ***nicht mehr passend*** für die Objekte der Unterklasse sind und müssen ***daher angepasst werden***. Man implementiert in einer Unterklasse ***eine schon existierende Methode erneut*** mit der exakt gleichen Methodensignatur. Dadurch wird die bestehende Methode überschrieben und somit wird ab ***jetzt immer der Code der überschriebenen Methode ausgeführt***.



Falls man in der überschriebenen Methode den Code der alten Methode benutzen möchte, kann man dies über super.methodenname() realisieren.

Lokale Variablen (AB8)

Manchmal muss man sich bestimmte Werte nur vorübergehend merken, um eine Aufgabe zu erfüllen. Sobald die Aufgabe erfüllt und eine Antwort gegeben wurde, kann man den Wert wieder vergessen (im Gegensatz zu Attributwerten, die „dauerhaft“ gespeichert sind).

	Lokale Variable	Attribut

Deklaration	Innerhalb der Methode:
	
	public class AB8 {
	…
	//Methoden
	
	public int berechneXY() {
	int zaehler2;
	…
	}
	…
	}
		Außerhalb/vor den Methoden:
		
		public class AB8 {
		int zaehler1;
		…
		//Methoden
		
		…
		}

Initialisierung	Direkt nach der Deklaration:
	
	public class AB8 {
	…
	//Methoden
	
	public int berechneXY() {
	int zaehler2;
	zaehler2 = 0;
	…
	}
	…
	}
		Im Konstruktor der Klasse:
		
		public class AB8 {
		int zaehler1;
		…
		public AB8() {
		zaehler1 = 0;
		}
		
		//Methoden
		…
		}

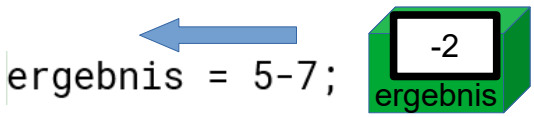
Verwendung	identisch ***zaehler2++;***	identisch ***zaehler1++;***

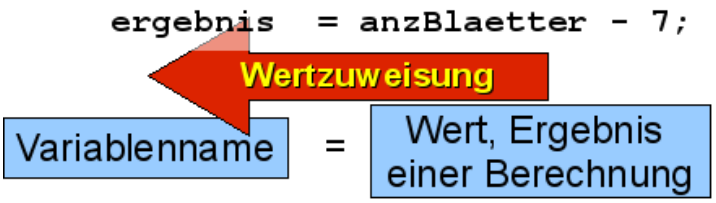
Rechnen mit Variablen (AB8)

Rechnen mit Attributen und lokalen Variablen funktioniert identisch!

Mit Zahlen (Datentyp ***int*** und ***double***) kann gerechnet werden. Das ***Ergebnis*** der Rechnung kann ***wieder einer Variablen zugewiesen*** werden. Die Operatoren für das Rechnen sind **+**, **-**, *****, **/**, **%**. ***Modulo***-Rechnung (Divisionsrest) ist bei einer fünfte Grundrechenart der Informatik: 17 % 5 = 2 33 % 3 = 0 9 % 2 = 1

Grundsätzlich gilt: Rechts vor Links! Zuerst wird die Rechnung/Methode auf der rechten Seite ausgeführt und dann der Variable links zugewiesen.





Beispiele:

Der Wert der Variablen zahl wird mal 2 gerechnet und dieser berechnete Wert wird dann wieder in der Variable zahl gespeichert. Kurz: zahl wird verdoppelt!

```
ergebnis = breite * hoehe;
energie = getEnergie() + 20;
zahl = zahl * 2;
```

Mit Variablen muss häufig und viel gerechnet werden. Daher gibt es in Java Kurzschreibweisen, um schneller programmieren zu können.

Im folgenden Beispiel wird die Variable **`int z`** um den Wert **`a`** verändert:

	Standard	Kurzversion	Veränderung um genau 1
Addition	z = z + a;	z += a;	z++;
Subtraktion	z = z – a;	z –= a;	z– –;
Multiplikation	z = z * a;	z *= a;	
Division	z = z / a;	z /= a;	
Modulo	z = z % a;	z %= a;	

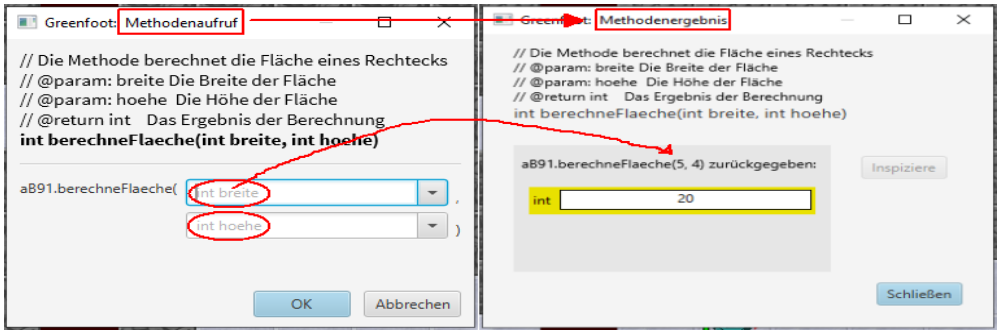
Methoden mit Parametern (AB9)

Bei einigen Methoden musste man immer Werte mit übergeben: istVorne(“Fass“), istAufGegenstand(“Portal“), ablegen(“Schraube“), usw. Man könnte auch istVorne(“Wand“) und ablegen(“Akku“) aufrufen. Die Methode erwartet also den Namen (String) des Gegenstands auf welcher verwendet werden soll.

Diese Methoden können je nach übergebenen Wert flexibel reagieren. Dies wird mittels ***Übergabeparameter*** realisiert.

Als Parameter sind alle Datentypen möglich! So erwartet beispielsweise die Methode mit der Methodensignatur macheWasXMal(int anzahl) eine Ganzzahl. → mögliche Methodenaufrufe: macheWasXMal(7); macheWasXMal(1234); auch macheWasXMal(-3)

Es kann auch ***mehrere Parameter*** geben. So hat die Methode int berechneFlaeche(int breite, int hoehe) zwei Parameter vom Typ int, die zur Flächenberechnung genutzt werden. Der Methodenaufruf berechneFlaeche(3, 5) würde somit den Wert 15 als Methodenergebnis liefern.



Letztendlich sind Übergabeparameter nicht anderes als lokale Variablen, die in der Methodensignatur deklariert und bei Methodenaufruf initialisiert werden.

Punktnotation (AB9)

Bisher haben wir immer nur uns selbst gesteuert. Also der Roboter hat sich über seinen eigenen Code in der eigenen Klasse selbst Befehle gegeben. Falls man einem anderen Objekt Befehle geben möchte, muss man dieses per ***Punktnotation*** ansprechen:

```
roboter1.einsVor();
roboter2.ablegen("Schraube");
```

Bei Methoden mit ***Übergabeparameter*** muss der Wert für diesen (dem Datentyp entsprechend) in den Klammern übergeben werden.

Merkhilfe*:* Wer macht was? Falls keiner angesprochen wird, bin ich gemeint.

Zählwiederholung mit for (AB10)

Für die Kontrollstruktur der Zählschleife wird durch das Schlüsselwort ist ***for*** eingeleitet, gefolgt von einem ***runden Klammerpaar*** und einem ***geschweiften Klammerpaar***.

```
for( int i = startwert; i < maximalwert; i = i + 1 ){
        // Befehle für die Wiederholung
}
```

In den runden Klammern stehen 3 Informationen (**mit Strichpunkt getrennt**):

Zählvariable und Startwert

int i = 0	Die lokale (Zähl-)Variable heißt i und startet mit dem Wert 0.
int i = 10	Die lokale (Zähl-)Variable heißt i und startet mit dem Wert 10.

Bedingung für das Ausführen der Befehle

Die Befehle in den geschweiften Klammern werden nur ausgeführt, wenn diese Bedingung ***true*** ist.

i < 5	Intervall von i von 0 bis 4	(nur sinnvoll beim Erhöhen der Zählvariablen)
i >= 0	Intervall von i von 10 bis 0	(nur sinnvoll beim Verringern der Zählvariablen)

Wert der Zählvariablen ändern

Wenn die Befehle in den geschweiften Klammern einmal ausgeführt wurden, dann muss der Wert der Zählvariablen verändert werden.

i = i + 1	Wert um 1 erhöhen	(jede Zahl von 0 bis 4 = 5 Wiederholungen)
i = i – 2	Wert um 2 verringern	(jede zweite Zahl von 10 bis 0 = 6 Wiederholungen)

Nutzung der Zählvariable

Auf den ersten Blick erschließt sich nicht, warum in Java die Zählwiederholung mit Startwert, Endwert und Schrittweite so kompliziert aufgebaut ist, denn man könnte sie ja auch einfach folgendermaßen definieren: Wiederhole X mal folgenden Befehl.

Der Sinn hierbei liegt darin, dass man die ***lokale (Zähl-)Variable i*** für viele Berechnungen oder Anwendungsfälle ***nutzen kann***.

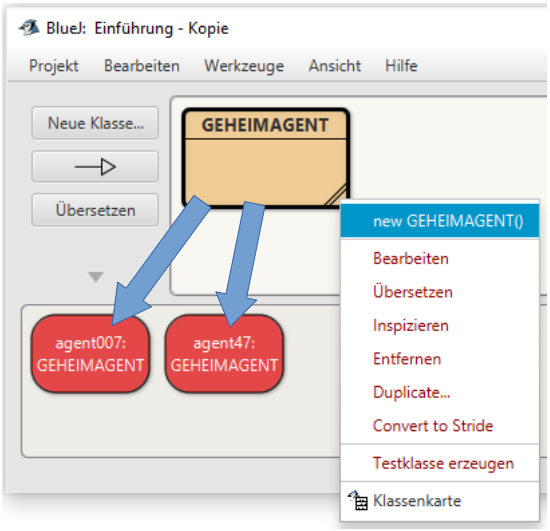
Beispiel: Zähle die ersten 100 Zahlen zusammen.

```
public int addiereDieErsten100Zahlen() {
        int ergebnis;
        ergebnis = 0;
        for( int i = 1; i <= 100; i++) {
                ergebnis += i;
        }
        return ergebnis;
}
```

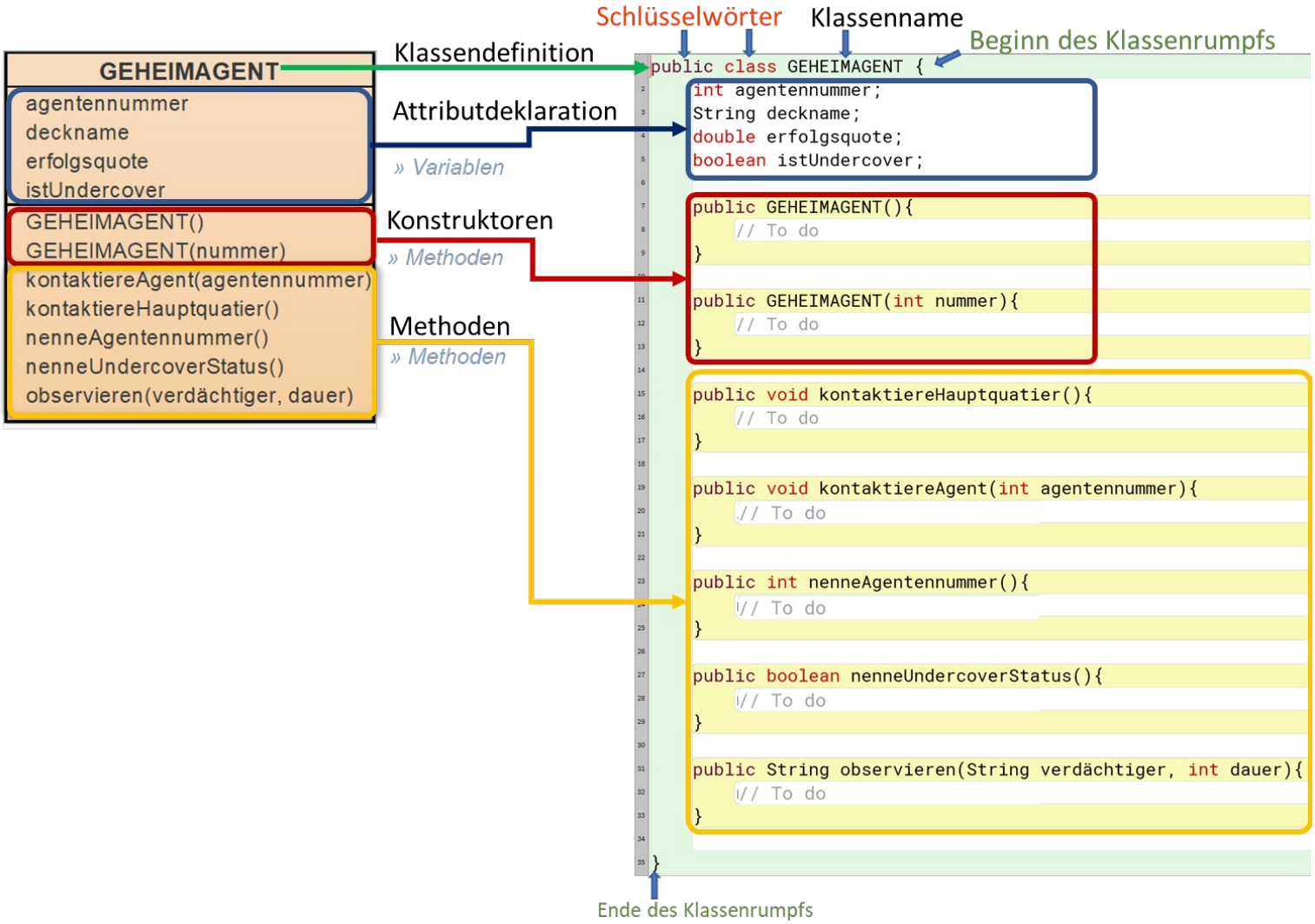
» Weitere Anwendungsbeispiele folgen in der 10. Jahrgangsstufe

Arbeiten mit BlueJ

Das Programmieren in BlueJ funktioniert genauso wie in Greenfoot, nur die Darstellungsweise ist etwas anders. Man schreibt/programmiert sich eine Klasse und kann dann aus dieser Klasse mittels Rechtsklick Objekte dieses Klassentyps erstellen.



Allgemeiner Klassenaufbau



Eine ***Klasse*** definiert als Schablone die ***Attribute*** und ***Methoden***, die alle Objekte dieser Klasse haben sollen.

Konstruktor(-methode)

```
public class GEHEIMAGENT{
```

Identische Schreibweise

```
public GEHEIMAGENT(){
    //
}
```

Jede Klasse besitzt in der Regel mindestens eine Konstruktormethode, damit Objekte dieser Klasse erzeugt werden können. Im Methodenrumpf des Konstruktors werden in der Regel die ***Attribute initialisiert*** und anschließend ***gibt*** der Konstruktor dieses erstellte ***Objekt zurück***.

Wichtig: unterschiedliche Methodensignatur als bei normalen Methoden
→ Methodenname und Rückgabetyp identisch! (kein void)

Objekterzeugung: Aufruf der Konstruktormethode

Bisher können wir nur Variablen mit (primitivem) Datentyp (int, double,…) erstellen. Variablen, die ein Objekt zugewiesen bekommen, nennt man ***Referenz(-variablen)*** bzw. ***Referenzattribut***, sofern es sich bei der Variablen um ein Attribut handelt. Die Handhabung mit Referenzen im Vergleich zu Variablen ist etwas ***unterschiedlich***:

Deklaration

Der Datentyp ist bei Referenzen die Klasse

```
int treffer;
double xKoordinate;
boolean sichtbar;
```

```
KREIS sonne;
FIGUR mario;
TEXT anzeige;
```

Initialisierung

```
treffer = 0;
xKoordinate = 4.7;
sichtbar = true;
```

Konstruktoraufruf mit **new**
ggf. Werte für Parameter

```
sonne = new KREIS();
mario = new FIGUR("mario.png");
anzeige = new TEXT(0,8,3,"Hallo");
```

Benutzung

Veränderung durch passende Operatoren

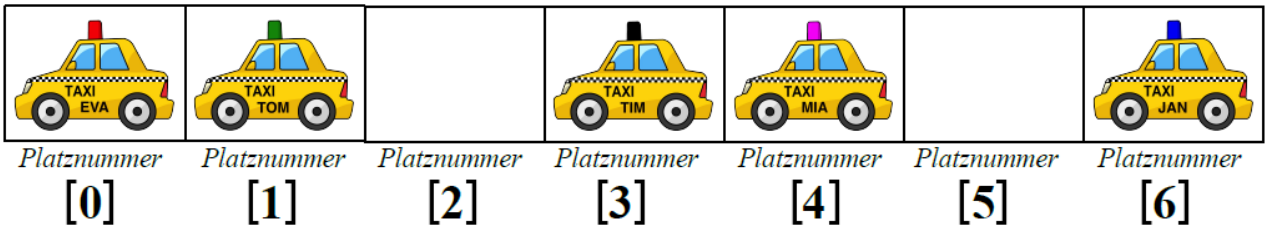
```
treffer++;
xKoordinate -= 4.7;
sichtbar = false;
```

Methodenaufruf mit Punktnotation

```
sonne.setzeMittelpunkt(10,7);
mario.verschiebenUm(2.7,0);
anzeige.setzeInhalt("Morgen");
```

Arrays / Felder

Die Datenstruktur ***Array*** speichert (beliebig) ***viele Objekte desselben Datentyps***. Man kann Arrays mit einem Taxistand vor dem Bahnhof vergleichen:



- Es ist immer eine ***festgelegte Größe*** an Plätzen vorhanden (hier: 7 Stück).
- Auf einem Taxistand dürfen nur Taxis stehen, nichts anderes. → Ein Array verwaltet auch nur Objekte ***eines gemeinsamen Datentyps***.
- Es müssen nicht alle Plätze benutzt werden, einige können auch leer sein.
- Es können nicht mehr Objekte als Felder vorhanden gespeichert werden.
- Die Plätze werden – ***beginnend mit [0]*** – durchnummeriert.

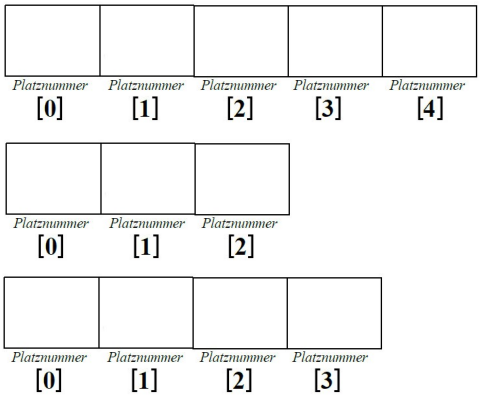
Programmierung in Java

Deklaration mit eckigen Klammern hinter dem Datentyp:

```
FIGUR[ ] nüsse;
int[ ] punkte;
String[ ] namen;
```

Das Array (der Parkplatz, der Container) selbst ist ein ***eigenständiges Objekt*** und muss mit einer ***festen Größe initialisiert*** werden:

```
nüsse = new FIGUR[5];

punkte = new int[3];

namen = new String[4];
```

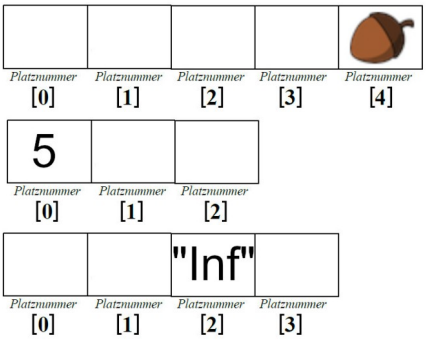


Die Arrays sind nach dem Erzeugen allerdings noch leer und speichern noch nichts!

Man kann jeden Platz im Array einzeln ansprechen und darin Werte/Objekte speichern.

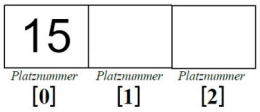
Beachte, dass bei Arrays von Objekten jedes einzelne Objekt erst noch erzeugt werden muss.

```
nüsse[4] =  new FIGUR("nuss.png");

punkte[0] = 5;

namen[2] = "Inf";
```



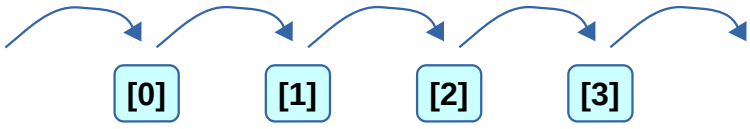
Man mit den gespeicherten Inhalten (primitive Datentypen als auch Referenzobjekten) arbeiten, indem man sie über ihren Platz im Array anspricht:

```
nüsse[4].setzeMittelpunkt(10,10);
punkte[0] = punkte[0] + 10;
namen[2] = "Informatik";
```



Iteration über ein Array mit der Zählschleife

Ein Array zu ***iterieren*** (= durchlaufen) bedeutet, für alle Mitglieder dieser Sammlung eine (ähnliche) Aktion auszuführen.




```
for( int i = 0; i < hühner.length; i++ ){
        hühner[i] = new FIGUR("moorhuhn.png");
        hühner[i].setzeMittelpunkt(0,0);
}
```

```
for( int i = 0; i < zahlen.length; i++ ){
        zahlen[i] = zahlen[i] * 5;
}
```

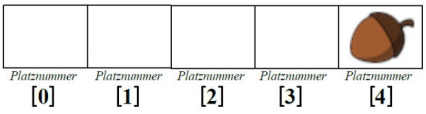
Das Array selbst ist ein Objekt und somit ist das Array durch die Punktnotation „ansprechbar". Mittels des Befehls ***array.length*** nennt das Array seine Länge, mit welcher es initialisiert wurde.

Hinweis: length ist keine Methode, sondern ein Attribut (eine Eigenschaft) des Arrays. Deshalb werden nach length auch keine Klammern () benötigt.

Mögliche Fehlerausgaben (Exceptions)

Neben ***Syntaxfehlern***, die durch den Übersetzer/Compiler ***beim Programmieren*** direkt rot angestrichen werden, gibt es noch ***Laufzeitfehler***. Diese treten erst ***beim Ausführen des Programms*** auf und führen zum Absturz.

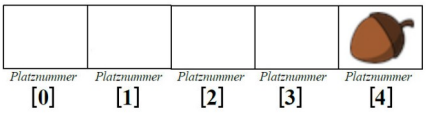
NullPointerException: Es wird eine Methode auf einer Referenz aufgerufen an deren Speicheradresse kein Objekt initialisiert wurde.



nüsse[2].setzeMittelpunkt(0,0);

```
java.lang.NullPointerException: Cannot invoke "FIGUR.setzeMittelpunkt(double, double)" because "this.nüsse[2]" is null
```

ArrayIndexOutOfBoundsException: Es wird auf eine Platznummer/Index im Array zugegriffen, welcher im Array nicht existiert.



nüsse[10].setzeMittelpunkt(0,0);
nüsse[-1].setzeMittelpunkt(0,0);

```
java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 5
```

Geschickte Nutzung der Zählvariable

Falls Objekte oder Zahlen ein bestimmtes Muster aufweisen sollen, kann man die Zählvariable geschickt verwenden, um dieses Muster ***in Abhängigkeit der Zählvariable*** darzustellen:

Beispiel: Das Array soll aufsteigend mit geraden Zahlen befüllt werden.

```
private int[ ] zahlen;
zahlen = new int[9];
for( int i = 0; i < zahlen.length; i++ ){
        zahlen[i] = i*2;
}
```

Hier ist das Array zahlen mit den zugehörigen Belegungen veranschaulicht: