

```
public void update() {  
    if (!isPlayerAlive()) return;  
    for (Direction direction : Direction.values())  
        if (Gdx.input.isKeyJustPressed(direction.getKey()))  
            move(direction);  
}
```

Programmieren mit Java

Hallo

```
private void move(Direction direction) {  
    playerPosition = playerPosition.add(direction.getPosition());  
    points += fields.get(playerPosition).getPoints();  
    fields.put(playerPosition, Field.NEUTRAL);  
}
```

```
for (int y = -10; y <= 10; y++)  
    for (int x = -10; x <= 10; x++)  
        System.out.println(x * x + y * y < 64 ? "x" : "");
```



Codeschnipsel

```
// Die Start-Methode
public static void main(String[] args) {
    // Hier beginnt das Programm. Gib dem Computer Befehle zum Ausführen!
}

// Etwas in der Konsole ausgeben
System.out.println();

// Rechnen mit Zahlen
3 + 4 * 8 - 10 / 2

// Strings erstellen
"Willkommen in meinem Spiel!"
```



Kommentare

```
// Ich bin ein Kommentar. Ich werde nicht ausgeführt.
```

```
/*  
Ich bin ein langer Kommentar, der  
sich über mehrere Zeilen erstreckt.  
Ich werde auch nicht ausgeführt.  
*/
```



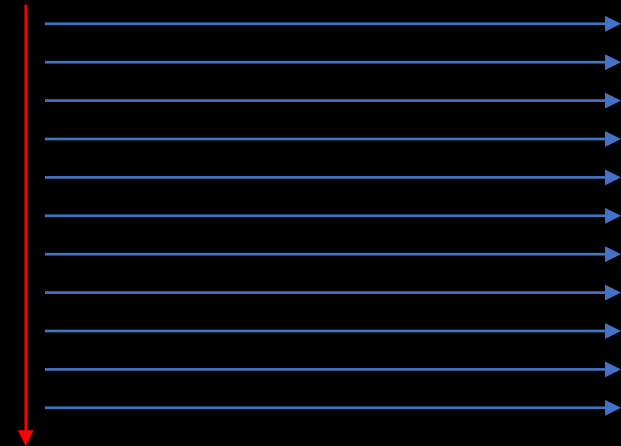
Start-Methode

```
/**
 * Ich bin JavaDoc. Ich erkläre den Zweck von Programmbestandteilen.
 * Die Start-Methode wird beim Start des Programms ausgeführt.
 *
 * @param args Kommandozeilen-Argumente für den Start des Programms.
 * Wir verwenden sie nicht.
 */
public static void main(String[] args) {
    // Hier stehen die Befehle, die das Programm ausführen soll.
}
```



Befehle

- Java ist imperativ.
 - Das bedeutet, dass man dem Computer Befehle zum Ausführen gibt.
- Befehle enden immer mit einem Semikolon (;).
- Befehle werden von oben nach unten und von links nach rechts ausgeführt.
 - So lesen wir auch Text.
 - Ausnahmen besprechen wir später





Primitive Datentypen

Name	Bedeutung	Werte
byte	Speichereinheit	[-128 bis 127] (2^8)
short	Ganze Zahl	[-32768 bis 32767] (2^{16})
<u>int</u>	Ganze Zahl	± 2 Milliarden (2^{32})
long	Ganze Zahl	± 9 Trillionen (2^{64})
<u>float</u>	Fließkommazahl	ca. 7 Ziffern
double	Fließkommazahl	ca. 16 Ziffern
<u>boolean</u>	Wahrheitswert	true (wahr) oder false (falsch)
char	Einzelnes Zeichen	z.B. 'a', '7', 'N', '-', '#', ' ' oder '('

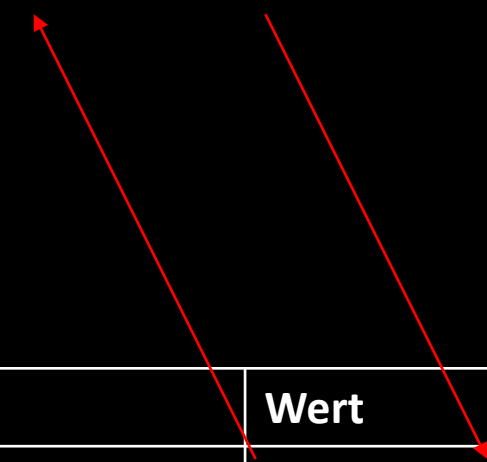


Variablen

```
// Variablen deklarieren: <Datentyp> <Name> ( = <Wert> );  
int sterneAmHimmel = 9095;  
boolean sterneLeuchten = true;  
char zweiterBuchstabe = 'B';  
float temperaturInGradCelsius = 20.21f;  
  
// Variablen verwenden: <Name>  
System.out.println(sterneAmHimmel); // Ergebnis: 9095  
  
// Variablen setzen: <Name> = <Neuer Wert>;  
sterneAmHimmel = sterneAmHimmel + 1;  
System.out.println(sterneAmHimmel); // Ergebnis: 9096
```

Berechne:

9095 + 1 = 9096



Variable	Wert
sterneAmHimmel	9095
sterneLeuchten	true
zweiterBuchstabe	'B'
temperaturInGradCelsius	20.21f



Begriffe - Variablen

Begriff	Bedeutung	Syntax
Variablen deklarieren	Variablen im Code erstellen.	<Datentyp> <Name>;
Variablen setzen / Werte zuweisen	Variablen verändern	<Name> = <Neuer Wert>;
Variablen initialisieren	Variablen erstmalig einen Wert zuweisen.	^^

Man kann Deklaration und Initialisierung kombinieren:

```
<Datentyp> <Name> = <Wert>;
```



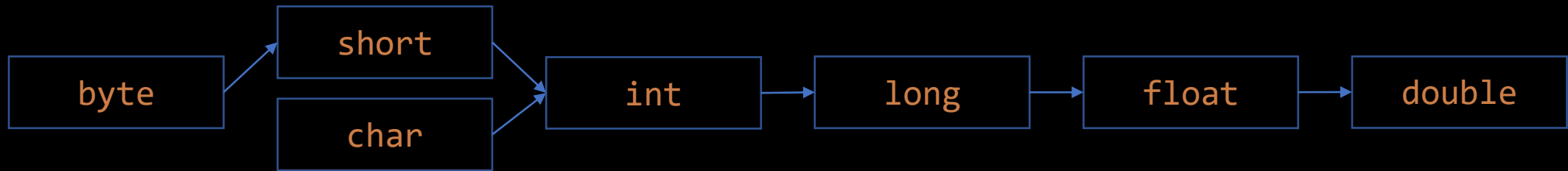

Operatoren und Ausdrücke

Name	Bedeutung	Syntax	Ergebnis
plus	Bei Strings: Verkettung	$a + b$	Zahl / String
minus		$a - b$	Zahl
mal		$a * b$	Zahl
geteilt		a / b	Zahl
kleiner als		$a < b$	boolean
größer als		$a > b$	boolean
und	prüfen, ob zwei Dinge gelten	$a \&\& b$	boolean
oder	prüfen, ob mindestens eins von beiden gilt	$a b$	boolean
gleich	prüfen, ob zwei Werte <u>dieselben</u> sind	$a == b$	boolean
ungleich	prüfen, ob zwei Werte <u>nicht dieselben</u> sind	$a != b$	boolean
nicht	einen Boolean verneinen	$!a$	boolean



primitive type casting

- Automatisch
 - Beispiel: Bei `int` → `float` wird



- Manuell
 - Beispiel: Bei `float` → `int` fallen alle Nachkommastellen weg.

```
System.out.println((int) 1.7f); // Ausgabe?  
System.out.println((int) -3.5f); // Ausgabe?  
System.out.println((int) 5.0f); // Ausgabe?  
System.out.println((int) -103.725f); // Ausgabe?
```



Objekte

Name	Bedeutung	Speichert
<u>String</u>	Zeichenkette	Beliebig viele Zeichen (Text)
<u>Scanner</u>	Eingaben-Leser	Infos zum Einlesen von Text
<u>Array</u>	Datenfeld / Reihe	Beliebig viele Variablen

... weitere (z.B. selbst definierte)

Objekte sind immer Instanzen von Klassen. Mehr dazu unter [Objektorientierung](#).



Scanner

```
// Erstelle einen Scanner. Er soll aus der Konsole (System.in) lesen.  
Scanner scanner = new Scanner(System.in);  
  
// Eingaben einlesen  
System.out.println(scanner.nextLine());
```



Objektvariablen

```
// Variablen deklarieren: <Datentyp> <Name> = <Wert>;  
String nameDesHellstenSterns = "Sirius";  
Scanner scanner = new Scanner(System.in);  
int[] Lieblingszahlen = {2, 13, 42, 1111};
```

Wenn man Objektvariablen keinen Wert zuweist, haben sie den Wert null.

Beim Zugriff auf eine Objektvariable mit dem Wert null, also Zugriff auf eine Instanzmethode oder ein Attribut des (nicht existenten) Objekts, stürzt das Programm mit einer `NullPointerException` ab.



Codebeispiel

```
public static void main(String[] args) {  
    String frage = "Wie heißt du?";  
    System.out.println(frage);  
    Scanner leser = new Scanner(System.in);  
    String eingabe = leser.nextLine();  
    String ausgabe = "Hallo, " + eingabe + "!";  
    System.out.println(ausgabe);  
}
```



Typ einer Variable

- Variablen primitiver Datentypen speichern ihren Wert direkt.
- Objektvariablen speichern Referenzen auf Objekte (z.B. String, Scanner, Random). Objekte können in mehreren Variablen referenziert werden.

Objektvariablen

Datentyp	Name	Wert
String	frage	
Scanner	leser	
String	eingabe	
String	ausgabe	

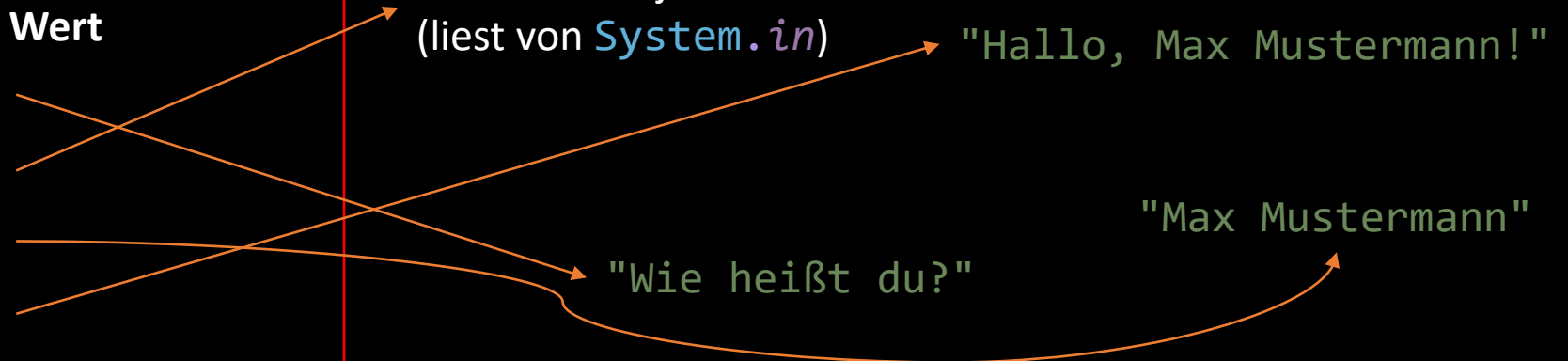
Scanner-Objekt
(liest von `System.in`)

"Hallo, Max Mustermann!"

"Wie heißt du?"

"Max Mustermann"

Irgendwo anders im Speicher





Was sind Methoden?

Methoden können zwei Dinge tun:

(1) Befehle ausführen

(2) Ergebnisse liefern (im Folgenden: "returnen")

Methoden verkapseln Programmcode.

```
// Verwendung einer Methode:  
// [Objekt.]<Name>(<Parameter>);
```




Beispiele für Methodenaufrufe

```
System.out.println("Hallo");
```

println	Gibt "Hallo" in der Konsole aus.	Befehl ausführen
---------	----------------------------------	------------------

```
int buchstabenInLangemWort = "Heizölrückstoßabdämpfung".length();
```

length	Returnt die Länge des Strings.	Ergebnis returnen
--------	--------------------------------	-------------------

```
System.exit(0);
```

exit	Beendet das Programm.	Befehl ausführen
------	-----------------------	------------------

```
boolean affeInGiraffe = "Giraffe".contains("affe");
```

contains	Enthält "Giraffe" irgendwo "affe"? Ja.	Ergebnis returnen
----------	--	-------------------

```
String eingabe = new Scanner(System.in).nextLine();
```

nextLine	Wartet auf eine Eingabe und returnt sie.	Beides (Warten: Befehl, Eingabe: Ergebnis)
----------	--	---

```
boolean feuerIstWasser = "Feuer".equals("Wasser");
```

equals	Handelt es sich um das gleiche Wort? Nein.	Ergebnis returnen
--------	--	-------------------



Warum eigene Methoden schreiben?

- Wiederverwendung
 - Vermeidung von ähnlichem Programmcode
- Methoden haben einen Namen.
 - Zweck leichter verstehen
- Unabhängigkeit vom Rest des Programms
 - Zerlegung des Programms in viele kleine Teilprobleme





Aufbau einer Methode

Um eine eigene Methode zu schreiben, müssen wir folgende Eigenschaften festlegen:

```
<Modifikatoren> <Rückgabetyp> <Name>(<Parameter>) {  
    <Befehle>  
    (return <Ergebnis>;)  
}
```



Ergebnis der Methode festlegen

- Mit dem **return**-Schlüsselwort bestimmen wir, welches Ergebnis unsere eigene Methode liefern soll.
- Das Ergebnis steht rechts von **return**.
- Methoden mit dem Rückgabetypen **void** liefern kein Ergebnis. Sie führen nur Befehle aus.
- Nach Ausführung von **return** ist die Methode vorbei. Es werden keine Befehle mehr ausgeführt.



Methoden in Aktion

```
private static void gibAus(String text) {  
    System.out.println(text);  
}  
  
private static String frag(String text) {  
    gibAus(text + "?");  
    Scanner scanner = new Scanner(System.in);  
    return scanner.nextLine();  
}  
  
public static void main(String[] args) {  
    String name = frag("Wie heißt du?");  
    gibAus("Hallo " + name + "!");  
    String wort = frag("Was ist dein Lieblingswort?");  
    gibAus("Ich mag das Wort " + wort + " auch, " + name + "!");  
}
```



Methoden vs. Funktionen

- In Java bedeuten die Begriffe das gleiche.
- In einigen anderen Sprachen:
 - Methoden führen Befehle aus, die den Zustand des Programms verändern.
 - Funktionen dienen ausschließlich dazu, Ergebnisse zu liefern.



Begriffe - Methoden

Begriff

Methode deklarieren

Methode aufrufen / ausführen

Methoden überschreiben

Bedeutung

Methoden selbst schreiben. Wir legen die Befehle fest, die die Methode ausführen soll.

Methoden verwenden. In den Klammern legen wir hier die Parameter für die Methode fest.

In abgeleiteten Klassen: Methoden neu deklarieren. (siehe Vererbung)



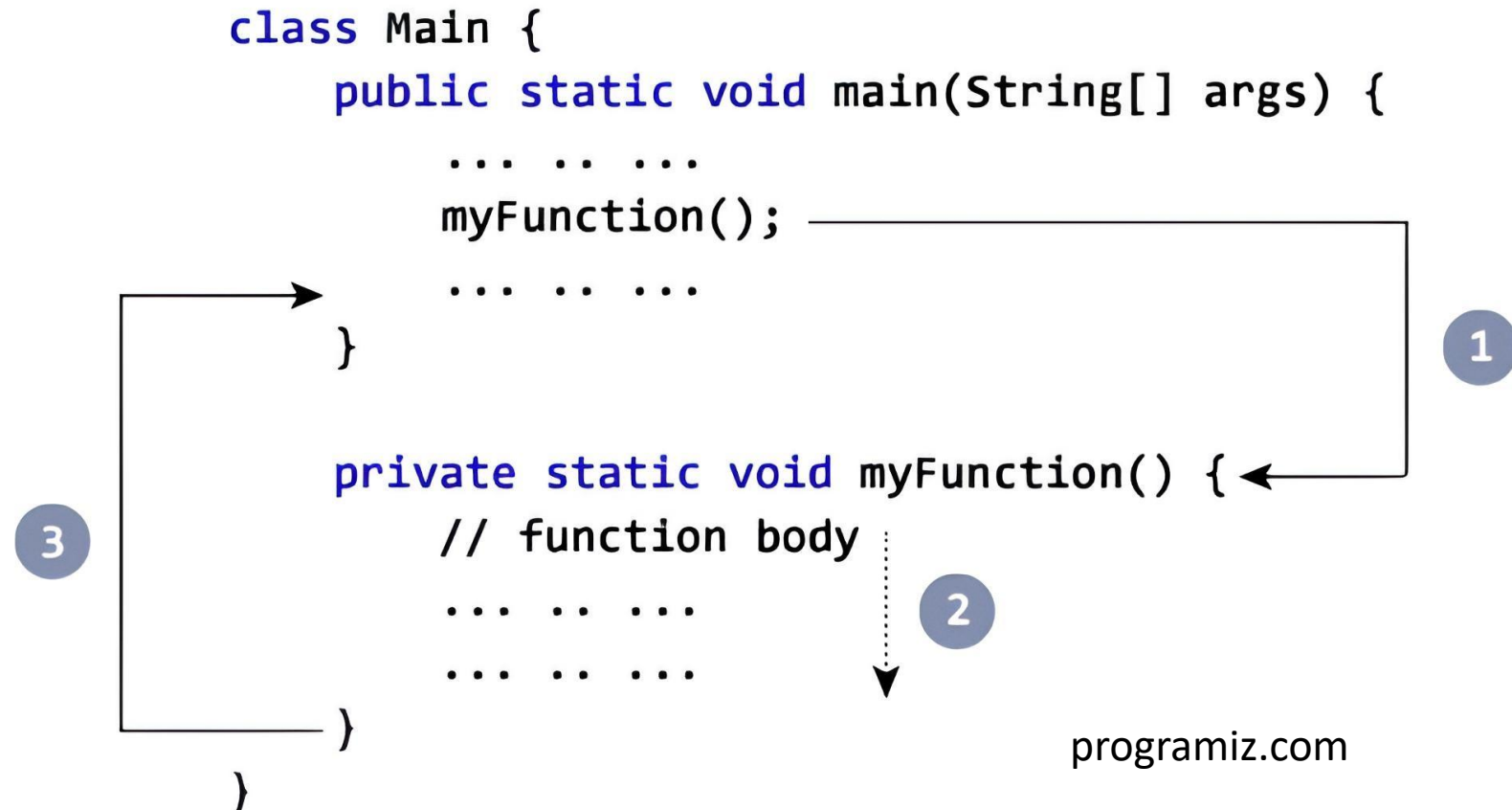
Arten von Methoden

Art	Bedeutung	Beispiel
Instanzmethode	Methode, die auf einem Objekt ausgeführt wird	Methode „nextLine“ der Klasse „Scanner“
Statische Methode	Methode, die immer, unabhängig von Objekten, verwendet werden kann	Methode „begrüßen“



Nicht verwechseln: Deklaration und Aufruf

- Bei der Deklaration wird die Methode definiert. Dabei entscheiden wir, was die Methode tut.
- Beim Aufruf einer Methode wird sie nur verwendet.
- Bild: Ablauf eines Methodenaufrufes





Was sind Klassen?

- Jedes Objekt hat einen Typen. Dieser wird durch Klassen definiert.
- Klassen können Variablen speichern (wir nennen sie „Attribute“).
 - „String“ speichert Zeichen.
 - Arrays speichern mehrere Werte oder Referenzen.
 - „Batch“ speichert Koordinaten, um sie bei „end“ an die GPU zu senden.
- Klassen können Methoden beinhalten.
 - „Scanner“ hat die Methode „nextLine()“, um die nächste Zeile auszulesen.
 - „Random“ hat die Methode „nextInt(int bound)“, um eine Zufallszahl zwischen 0 und bound zu erzeugen.
 - „Batch“ hat die Methode „draw(Texture texture, float x, float y)“, um eine Textur bei x und y darzustellen.
- Klassen sind autonom.
 - Sie entscheiden über ihre Attribute und Methoden und definieren deren Verhalten.



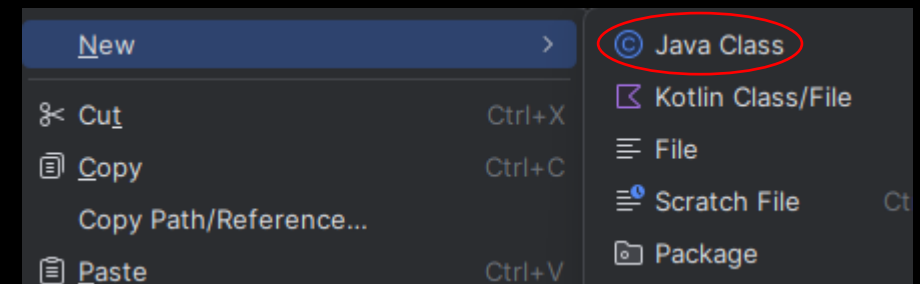
Objektorientierung

- Paradigma
- Modellierung logisch zusammengehöriger Eigenschaften durch Klassen
 - Beispiel: Attribute Vorname, Nachname und Geburtsdatum in Klasse "Person", Methode zur Berechnung des Alters
- Die Programmlogik ergibt sich aus dem Zusammenspiel von Objekten
 - Beispiel: Initialen ergeben sich aus Anfangsbuchstaben von Vorname und Nachname, die durch die Klasse "String" verwaltet werden.



Eigene Klassen erstellen

- Jede Klasse hat normalerweise eine eigene Datei.
- Rechtsklicke den Ordner (z.B. root.content)
- Wähle New › Java Class
- Gib einen Namen in UpperCamelCase ein.
- In die geschweiften Klammern kannst du sowohl Attribute als auch Methoden schreiben.





Konstruktor

- Ähnlich wie Methoden
- Initialisiert die Attribute eines Objekts
`<Modifikatoren> <Klassenname>(<Parameter>) {
}`
- `this` bezeichnet das Objekt, auf dem eine Instanzmethode ausgeführt wurde.
 - Mit `this(<Parameter>);` kann ein Konstruktor einen anderen Konstruktor aufrufen.
 - Mit `this.<Attribut>` kann auf Attribute zugegriffen werden, wenn die Parameter-Namen des Konstruktors gleich heißen.



Bezeichnung Methoden von Klassen

<Klasse>#<methode> bedeutet: <methode> kann auf einem Objekt von <Klasse> ausgeführt werden.

Beispiel: Scanner#nextLine kann so verwendet werden:

```
Scanner scanner = new Scanner(System.in);  
String input = scanner.nextLine();
```



Arten von Variablen

Art	Bedeutung	Beispiel
Lokale Variable	Variable innerhalb eines Blocks. (Methoden sind auch ein Block)	Um die kleinste Zahl aus einem Array zu finden, verwenden wir eine Schleife. Wir speichern dabei immer die aktuell kleinste Zahl.
Attribut / Instanzvariable	Variable innerhalb einer Klasse. Sie wird für jedes Objekt einmal gespeichert.	Die Klasse „Baum“ könnte das Attribut „Höhe“ haben. Jeder Baum hat eine andere Höhe.
Statische Variable / Klassenvariable	Variable innerhalb einer Klasse. Sie speichert, unabhängig vom Projekt, nur einen Wert.	Die Zahl π ist unabhängig vom Objekt immer gleich.
Parameter / Argument	Variable, die einer Methode übergeben wird.	Die Methode „frag“ bekommt den Parameter „text“, um den Fragetext zu kennen.



Sichtbarkeitsmodifikatoren

Können für Attribute und Methoden verwendet werden:

Schlüsselwort	Bedeutung
private	Kann nur innerhalb der Klasse verwendet werden. So können wir sie absichern.
<kein Modifikator>	Kann im gleichen Ordner (package) verwendet werden.
protected	Kann im gleichen Ordner (package) und abgeleiteten Klassen (siehe Vererbung) verwendet werden.
public	Kann überall verwendet werden



final-Modifikator

Sorgt dafür, dass ...

... Variablen kein neuer Wert zugewiesen werden kann.

- Achtung: Referenzierte Objekte können ihren Zustand ändern, es können nur keine neuen Objekte zugewiesen werden.

... Methoden nicht überschrieben werden können. (siehe Vererbung)

... Klassen nicht erweitert werden können. (siehe Vererbung)



String

- Ein String ist eine Zeichenkette.
- Er kann durch Anführungsstriche erstellt werden: "Hallo"
- Um ein beliebiges Zeichen auszulesen, gibt es die Methode `charAt(int index)`. Die Indizes starten bei 0!
 - Achtung: Wenn der Index negativ oder größer/gleich der Länge des Strings ist, stürzt das Programm mit einer `StringIndexOutOfBoundsException` ab.
- Um die Länge eines Strings zu erfahren, verwende `length()`.
- Um Strings untereinander oder mit anderen Werten zu kombinieren, verwende das Plus-Zeichen: `stringA + variableB`



System-Klasse

- Die System-Klasse bietet nützliche Attribute und Methoden:
 - Über `System.out` kann mit der Methode `println(String s)` etwas in der Konsole ausgegeben werden.
 - Über `System.in` können mit Hilfe des Scanners Eingaben aus der Konsole eingelesen werden.
 - Die Methode `System.currentTimeMillis()` liefert die Anzahl an Millisekunden (also 1000stel Sekunden) seit dem 1. Januar 1970.



equals-Methode

==

Operator

Sind zwei Werte identisch? ("die selben")

In die Sprache eingebaut

equals

Methode der Klasse Object

Sind zwei Objekte inhaltsgleich? ("die gleichen")

Verhalten wird von Klassen definiert

Beide geben einen boolean zurück.



Kontrollstrukturen

Syntax

if (<Wahrheitswert>) <Befehl>

... else <Befehl>

while (<Bedingung>) <Befehl>

for (<Datentyp> <Name> : <Array>)
<Befehl> (genannt: foreach)

do <Befehl> while (<Bedingung>)

x = <Bedingung> ? <Wert1> : <Wert2>

... 2 weitere (for, switch)

Bedeutung

Führe den Befehl nur dann aus, wenn der Wahrheitswert true ist.

Sonst und nur sonst führe den anderen Befehl aus.

Führe den Befehl aus, solange die Bedingung gilt.

Führt den Befehl für jedes Element des Arrays aus.

Führe den Befehl aus und wiederhole solange die Bedingung gilt.

Wenn die Bedingung gilt, soll x Wert1 sein, sonst soll x Wert2 sein



Ausnahmen: Reihenfolge der Ausführung

```
// Zuweisung von Variablen
boolean dreiIstVier = 3 == 4; // Ergebnis: false

// Punkt vor Strich
int wert = 1 + 2 * 2; // Ergebnis: 5, nicht 6

// Aufruf von Methoden
Scanner scanner = new Scanner(System.in);
// Wartet erst, macht dann weiter
String eingabe = scanner.nextLine();

// Verwendung von Kontrollstrukturen
while (!eingabe.equals("stopp"))
    // Springt zur Bedingung der Schleife zurück
    System.out.println("weiter");
```



Blöcke

```
// Um mehrere Befehle in Kontrollstrukturen auszuführen, werden Blöcke verwendet:  
if (scanner.nextLine().equals("Mach mir nach!")) {  
    // Die Befehle werden {in geschweifte Klammern} geschrieben.  
    System.out.println("Okay!");  
    System.out.println(scanner.nextLine());  
}
```

- Blöcke gruppieren mehrere Befehle zusammen.
- Alle Methoden sind Blöcke.
- Alle Variablen innerhalb eines Blocks sind nach dessen Ausführung weg.
 - Achtung: Das betrifft nur die Variablen, weil sie nicht mehr verwendet werden. Objekte innerhalb der Variablen, die noch von woanders im Programm (zum Beispiel in einer Attributvariable der Klasse) referenziert werden, werden nicht gelöscht.



Beispiele für Blöcke

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    String eingabe = scanner.nextLine();
    if (eingabe.equals("Mach mir nach, bis ich stopp sage.")) {
        System.out.println("Okay, mache ich.");
        while (true) {
            eingabe = scanner.nextLine();
            if (eingabe.equals("stopp")) {
                System.out.println("Okay, ich höre auf.");
                break;
            }
            System.out.println(eingabe);
        }
    }
}
```




Einlesen anderer Datentypen

Wir verwenden `scanner.nextLine()`, um einen String einzulesen. Es können aber auch andere Datentypen eingelesen werden:

```
int nextInt = scanner.nextInt();
```

Allerdings stürzt das Programm ab, wenn die Eingabe nicht diesem Datentyp entspricht. Daher sollten wir vor dem Einlesen mit Hilfe einer Schleife auf den richtigen Datentyp warten:

```
while (!scanner.hasNextInt()) { // Solange die Eingabe kein int ist:  
    scanner.nextLine(); // Überspringe die Eingabe.  
    System.out.println("Bitte gib einen int ein.");  
}  
int nextInt = scanner.nextInt();
```



Problem: Alle Geschichten ausgeben

```
// Deklariere ganze Zahl i. (Sie steht für den Index der aktuellen Geschichte.)
int i = 0;
// Solange die Anzahl der Geschichten nicht erreicht ist
while (i < geschichten.length) {
    // Gib die aktuelle Geschichte aus.
    System.out.println(geschichten[i]);
    // Erhöhe den Index um 1.
    i++;
}
```



Abkürzungen: Variablen verändern

Abkürzung	Langform	Bedeutung
<code>i += 2;</code>	<code>i = i + 2;</code>	Erhöhe i um 2.
<code>i *= 2;</code>	<code>i = i * 2;</code>	Verzweifache i.
Das funktioniert auch für andere Operatoren, z.B. Minus (-), Geteilt (/), Teiler Rest (%)		
<code>i++;</code>	<code>i = i + 1;</code>	Erhöhe i um 1.
<code>i--;</code>	<code>i = i - 1;</code>	Verringere i um 1.



for-Schleife

Diese while-Schleife gibt die Zahlen von 0 (inkl.) bis 20 (exkl.) aus.

```
int i = 0;
while (i < 20) {
    System.out.println(i);
    i++;
}
```

Diese for-Schleife tut das auch.

```
for (int j = 0; j < 20; j++)
    System.out.println(j);
```

for-Schleifen können verwendet werden, um zu zählen. Aufbau:

```
for (<Deklaration & Initialisierung>; <Bedingung>; <Schritt>) <Befehl>
```



Der Ternary-Operator

Wenn die Bedingung vor dem "?" gilt, wird der Wert links vom ":" zurückgegeben, sonst der Wert rechts vom ":".

```
Scanner scanner = new Scanner(System.in);
String eingabe = scanner.nextLine();

// lang (mit if-else)
if (eingabe.equals("Hallo"))
    System.out.println("Hallo zurück!");
else
    System.out.println("Wie bitte?");

// kurz (mit ternary)
System.out.println(eingabe.equals("Hallo") ? "Hallo zurück!" : "Wie bitte?");
```



Intervalle

Für reelle Zahlen a und b ist

- $[a, b]$ das Intervall aller reellen Zahlen von a inklusiv bis b inklusiv
- $[a, b)$ das Intervall aller reellen Zahlen von a inklusiv bis b exklusiv.

Beispiele:

- Integer aus $[4, 7] = \{4, 5, 6, 7\}$.
- Integer aus $[3, 6) = \{3, 4, 5\}$.
- Floats aus $[1, 2] = \{\text{sehr viele Zahlen von 1 bis 2 inklusiv, z.B. 1.25}\}$.



Random

```
Random random = new Random();

// Zufälliger Wert von 0 (inklusive) bis 4 (exklusiv),
// also aus {0, 1, 2, 3} (jeweils 25%)
int randomInt = random.nextInt(4);

// Zufälliger Wert von 0.0 (inklusive) bis 4.0 (exklusiv),
// also z.B. 0.75, 2.0, 3.125
float randomFloat = random.nextFloat(4);

// Entweder true (50%) oder false (50%)
boolean randomBoolean = random.nextBoolean();
System.out.println(randomBoolean ? "Glück" : "Pech");
```



Math-Klasse

- Stellt hilfreiche Mathe-Funktionen bereit, z.B.:

Name	Rückgabotyp	Parameter	Bedeutung
min	<Zahl>	<Zahl> a, <Zahl> b	Die Kleinere der beiden Zahlen
max	<Zahl>	<Zahl> a, <Zahl> b	Die Größere der beiden Zahlen
sin	double	double a	Sinus von a
cos	double	double a	Cosinus von a
tan	double	double a	Tangens von a
abs	<Zahl>	<Zahl> a	Betrag von a
round	int/long	float/double	Die nächste ganze Zahl (rundet bei .5 auf)
pow	double	double a, double b	a hoch b
PI	double (π ist eine Konstante)		Kreiszahl



Array

```
// Array erstellen
String[] geschichten = {
    "Es war einmal ein schrecklicher Drache. Er lebte in einer dunklen Höhle ...",
    "Tief verborgen im Zauberwald lebte die alte Hexe ...",
    "Vor langer, langer Zeit gab es einen Raben. Er flog nachts über die dunklen
    Bergkuppen ..."
};

System.out.println("Welche Geschichte möchtest du hören? (Drache, Hexe, Rabe)");
String wahl = scanner.nextLine();

// Zugriff auf Arrays mit <Name>[index], Indizes beginnen bei 0, nicht 1.
if (wahl.equals("Drache"))
    System.out.println(geschichten[0]);
else if (wahl.equals("Hexe"))
    System.out.println(geschichten[1]);
else if (wahl.equals("Rabe"))
    System.out.println(geschichten[2]);
else
    System.out.println("Diese Geschichte kenne ich nicht.");
```



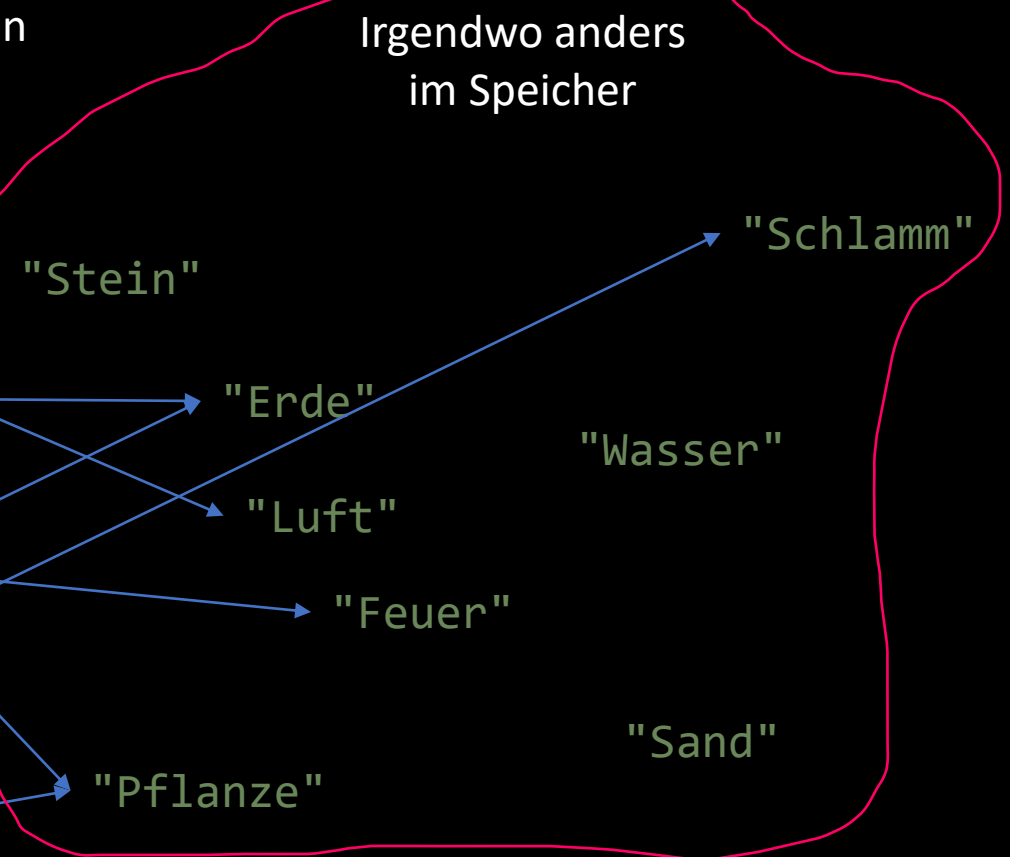
Array-Visualisierung

Array an primitiven Werten
Beispiel: `int[]`

Index	Wert
0	3
1	4234
2	-45
3	720
4	34
5	7
6	-23
7	0
8	-1

Array an Objektreferenzen
Beispiel: `String[]`

Index	Wert
0	
1	
2	
3	
4	
5	
6	
7	<code>null</code>
8	





Array-Inhalte verändern

```
int[] quadrate = new int[100]; // quadrate[0] .. quadrate[99]
for (int i = 0; i < quadrate.length; i++)
    quadrate[i] = i * i;
System.out.println(quadrate[3]); // Ausgabe: 9
quadrate[2] = quadrate[4];
System.out.println(quadrate[2]); // Ausgabe: 16
```



Algorithmus

Ablauf von Befehlen, der ein (mathematisches) Problem löst

Beispiele:

- Durchschnitt mehrerer Zahlen berechnen
- Besten Tick-Tack-Toe Zug finden
- Ausgang eines Labyrinths finden



Beispiel: Durchschnitt mehrerer Zahlen

```
private float durchschnitt(float[] zahlen) {  
    float summe = 0;  
    int i = 0;  
    while (i < zahlen.length) {  
        summe += zahlen[i];  
        i = i + 1;  
    }  
    return summe / zahlen.length;  
}
```



foreach-Schleife

```
// Ich will alle Geschichten hören.  
for (String geschichte : geschichten)  
    System.out.println(geschichte);  
  
// Ich will die längste Geschichte hören.  
String longest = "";  
// Für jede Geschichte  
for (String geschichte : geschichten) {  
    // Ist sie länger als die aktuell längste Geschichte?  
    if (geschichte.length() > longest.length())  
        longest = geschichte;  
}  
// Gib die längste aus  
System.out.println(longest);
```



Kontrolle von Schleifen

- `"break;"` bricht eine Schleife früher ab.
- `"continue;"` beendet die aktuelle Iteration einer Schleife.

```
String[] neuigkeiten = {"Die Wahrheit.", "Fake News!", "WOW!",  
    "Oh nein.", "Achtung, Wahrheit.", "Was?"};  
  
// Zensiere wahre Neuigkeiten.  
for (String neuigkeit : neuigkeiten) {  
    if (neuigkeit.contains("Wahrheit"))  
        continue;  
    System.out.println("Keine Zensur nötig: " + neuigkeit);  
}  
  
// Finde etwas Falsches.  
for (String neuigkeit : neuigkeiten) {  
    if (neuigkeit.contains("Fake")) {  
        System.out.println("Die totale Wahrheit: " + neuigkeit);  
        break;  
    }  
}
```



Verbessert: Durchschnitt mehrerer Zahlen

```
private float durchschnitt(float[] zahlen) {  
    float summe = 0;  
    for (float zahl : zahlen)  
        summe += zahl;  
    return summe / zahlen.length;  
}
```




Vererbung

- Klassen können andere Klassen erweitern. Beispiele:
 - „Main“ erweitert „ApplicationListener“, um Lifecycle-Methoden zu überschreiben.
 - „ExtendViewport“ erweitert „Viewport“, um den Rest des Bildschirms auszufüllen.
 - „DefaultAndroidInput“ erweitert „Input“, um Fingerbewegungen und Handydrehungen auszulesen und „DefaultLwjgl3Input“ erweitert „Input“, um Mausklicks und Tastatureingaben auszulesen.
- Bei Vererbung können Klassen Methoden überschreiben. So tun Objekte dieser Klasse nicht das, was in der Basisklasse, sondern was in der abgeleiteten Klasse festgelegt wurde.



Begriffe - Vererbung

Klasse B erweitert Klasse A

Name	Bedeutung	Beispiel
Superklasse / Basisklasse	Klasse A ist eine Superklasse bzw. Basisklasse von Klasse B.	Die Klassen „Pflanze“ und „Baum“ sind Superklassen der Klasse „Eiche“.
Subklasse / abgeleitete Klasse	Klasse B ist eine Subklasse bzw. abgeleitete Klasse von Klasse A.	Die Klassen „Baum“ und „Eiche“ sind Subklassen der Klasse „Pflanze“.
überschreiben	Eine Methode aus Klasse A wird in Klasse B mit der gleichen Vorschrift deklariert. Ihr Verhalten wird so für alle Objekte von B neu definiert.	Wir überschreiben die Lifecycle-Methoden und definieren damit deren Verhalten neu.



Object-Klasse

- Alle Klassen erweitern standardmäßig die Objekt-Klasse

Name	Rückgabetyt	Parameter	Bedeutung
toString	String		Wandelt das Objekt in Text um, damit wir es ausgeben können.
equals	boolean	Object other	Überprüft, ob das Objekt den gleichen Inhalt wie ein anderes Objekt hat.
hashCode	int		Wandelt das Objekt in eine ganze Zahl (int) um. Diese kann verwendet werden, um das Objekt schnell wiederzufinden, ähnlich wie Bücher, die in einer Bibliothek nach Textart und Genre sortiert sind.



Zugriffsfunktionen

Eine Methode heißt getter, wenn sie lediglich ein Attribut liefert:

```
public int getPoints() {  
    return points;  
}
```

Eine Methode heißt setter, wenn sie lediglich ein Attribut setzt:

```
public void setPoints(int value) {  
    this.points = value;  
}
```



Arten von Methoden 2

Art	Bedeutung
Finale Methode	Methode, die nicht überschrieben werden kann (siehe Vererbung)
Abstrakte Methode	Methode, die überschrieben werden muss, weil sie keine Definition hat (siehe Vererbung)



Überschriebene Methoden verwenden

- Wenn man eine Methode überschreibt, aber die überschriebene Methode noch verwenden möchte, kann man `super` verwenden:

```
public class Baum {  
  
    public ArrayList<String> eigenschaften() {  
        ArrayList<String> result = new ArrayList<>();  
        result.add("groß");  
        result.add("grün");  
        result.add("raschelt im wind");  
        return result;  
    }  
}  
  
public class Eiche extends Baum {  
  
    @Override  
    public ArrayList<String> eigenschaften() {  
        ArrayList<String> eigenschaften = super.eigenschaften();  
        eigenschaften.add("robust");  
        return eigenschaften;  
    }  
}
```



object/class type casting

- Automatisch
 - Objekte von Unterklassen sind gleichzeitig Objekte der Basisklasse.
 - Das Objekt verändert sich nicht.
 - Beispiel: Eine Eiche ist ein Baum, bleibt deshalb aber trotzdem eine Eiche.
- Manuell
 - Wenn das Objekt nicht den Typen hat, zu dem umgewandelt wird, stürzt das Programm mit einer ClassCastException ab.
 - Beispiel: Beim Casten einer Birke zu einer Eiche stürzt das Programm ab.



Klasse Position

```
public class Position {
    private final int x, y;

    public Position(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public Position add(Position other) {
        return new Position(x + other.x,
            y + other.y);
    }
}
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() !=
            o.getClass()) return false;

        Position position = (Position) o;
        return x == position.x && y == position.y;
    }

    @Override
    public int hashCode() {
        int result = x;
        result = 31 * result + y;
        return result;
    }
}
```




Record

Ein Record ist eine Abkürzung für eine Klasse, die als reiner Datenspeicher dient. Er hat folgende Eigenschaften:

- Alle Attribute sind final
- Objekte sind genau dann equals, wenn es ihre Attributwerte sind.

Beispiel:

```
public record Position(int x, int y) {  
    public Position add(Position other) {  
        return new Position(x + other.x, y + other.y);  
    }  
}
```



Enum

- Eine Klasse, die eine vordefinierte Anzahl an Objekten hat.

```
public enum Monat {  
    Januar, Februar, März, April, Mai, Juni, Juli, August, September, Oktober, November, Dezember  
}
```

```
public enum Schachfigur {  
    Bauer(1), Turm(5), Läufer(3), Springer(3), König(Integer.MAX_VALUE), Dame(9);  
  
    public final int wert;  
  
    Schachfigur(int wert) {  
        this.wert = wert;  
    }  
}
```



Enum-Methoden

Sei E ein Enum mit Enum-Konstanten C1, C2, C3.

Aufruf	Rückgabetyt	Bedeutung
C<x>.ordinal()	int	Index des Enum-Werts
E.values()	E[]	Array mit allen Enum-Werten
E.valueOf(String name)	E	Enum-Wert mit dem Namen name



Datenstrukturen

- Datenstrukturen speichern und organisieren mehrere Objekte.
 - Arraylisten nummerieren Objekte durch. Sie beginnen bei 0 und können beliebig viele Objekte beinhalten.
 - HashSets merken sich alle Objekte, die sie enthalten, auf eine strukturierte Weise. So kann man schnell herausfinden, ob ein Objekt vorhanden ist, ohne alle Objekte zu durchstöbern.

Beispiel: Wenn in einer Bibliothek „Der Herr der Ringe“ nicht unter „Fantasyromane mit D“ oder „Fantasyromane mit H“ zu finden ist, muss man im Rest der Bibliothek gar nicht erst danach suchen.

Außerdem können Sets (Mengen) im allgemeinen jedes Objekt nur einmal enthalten.
 - HashMaps ordnen Objekten andere Objekte zu. So kann man z.B. Positionen auf einem schachbrettartigen Spielfeld die Figur zuordnen, die sich darauf befindet.



Generics

- Generics machen es möglich, Klassen und Methoden für beliebige Typen zu verwenden.
- Generics sind durch eine gespitzte Klammer um den Namen des generischen Typen gekennzeichnet, z.B. `<T>`
- Wenn der Typ unbestimmt ist, wird ein `?` eingesetzt, z.B. `<?>`



Collection<E>

Name	Rückgabotyp	Parameter	Bedeutung
size	int		Anzahl der Elemente
isEmpty	boolean		Ob es keine Elemente gibt
contains	boolean	Object o	Ob das Objekt „o“ enthalten ist
containsAll	boolean	Collection<?> c	Ob alle Objekte aus c auch hier enthalten sind
add	boolean	E e	Fügt das Element „e“ hinzu.
remove	boolean	Object o	Entfernt das Objekt „o“, wenn enthalten.
addAll	boolean	Collection<? extends E> c	Fügt alle Elemente auf c hinzu.
removeAll	boolean	Collection<?> c	Entfernt alle Elemente aus c.
clear	void		Leert die Collection.
retainAll	boolean	Collection<?> c	Entfernt alle Elemente, bis auf die aus c.



ArrayList<E> (ist eine Collection)

Nummeriert Objekte durch. Beginnt bei 0 und kann beliebig viele Objekte beinhalten.
Anders als ein Array kann sich die Länge einer ArrayList ändern.

Name	Rückgabetyp	Parameter	Bedeutung
get	E	Int index	Liefert das Objekt mit der Nummer index.
indexOf	int	Object o	Liefert, falls o enthalten, den (ersten) Index von o, sonst -1.



HashSet<E> (ist eine Collection)

Merkt sich alle Objekte, die sie enthalten, auf eine strukturierte Weise. So kann man schnell herausfinden, ob ein Objekt vorhanden ist, ohne alle Objekte zu durchstöbern.

Beispiel:

Wenn in einer Bibliothek „Der Herr der Ringe“ nicht unter „Fantasyromane mit D“ oder „Fantasyromane mit H“ zu finden ist, muss man im Rest der Bibliothek gar nicht erst danach suchen.

Sets (Mengen) allgemein:

- Können jedes Element nur einmal enthalten. Entweder ist es enthalten, oder nicht.
- Haben, anders als Listen, keine Information über die Reihenfolge der Objekte. Sie ist entweder komplett willkürlich oder hängt von Eigenschaften der Objekte ab.
 - Es gibt keine Indizes.



HashMap<K, V>

Ordnet Objekten andere Objekte zu. So kann man z.B. Positionen auf einem schachbrettartigen Spielfeld die darauf befindliche Figur zuordnen.

Name	Rückgabebetyp	Parameter	Bedeutung
size	int		Anzahl der Elemente
isEmpty	boolean		Ob es keine Elemente gibt
clear	void		Leert die HashMap.
put	V	K key, V value	Ordnet key value zu.
get	V	Object key	Liefert das zu key zugeordnete Objekt, oder null wenn keines zugeordnet ist.
containsKey	boolean	Object key	Ob key ein Objekt zugeordnet ist
containsValue	boolean	Object value	Ob irgendeinem Objekt value zugeordnet ist



exception handling

...



Interfaces

Gibt vor, welche Methoden abgeleitete Klassen bereitstellen müssen.

Eignen sich als Methode, die man übergeben kann.

```
public interface CoordinateAction {  
    void execute(int x, int y);  
}
```