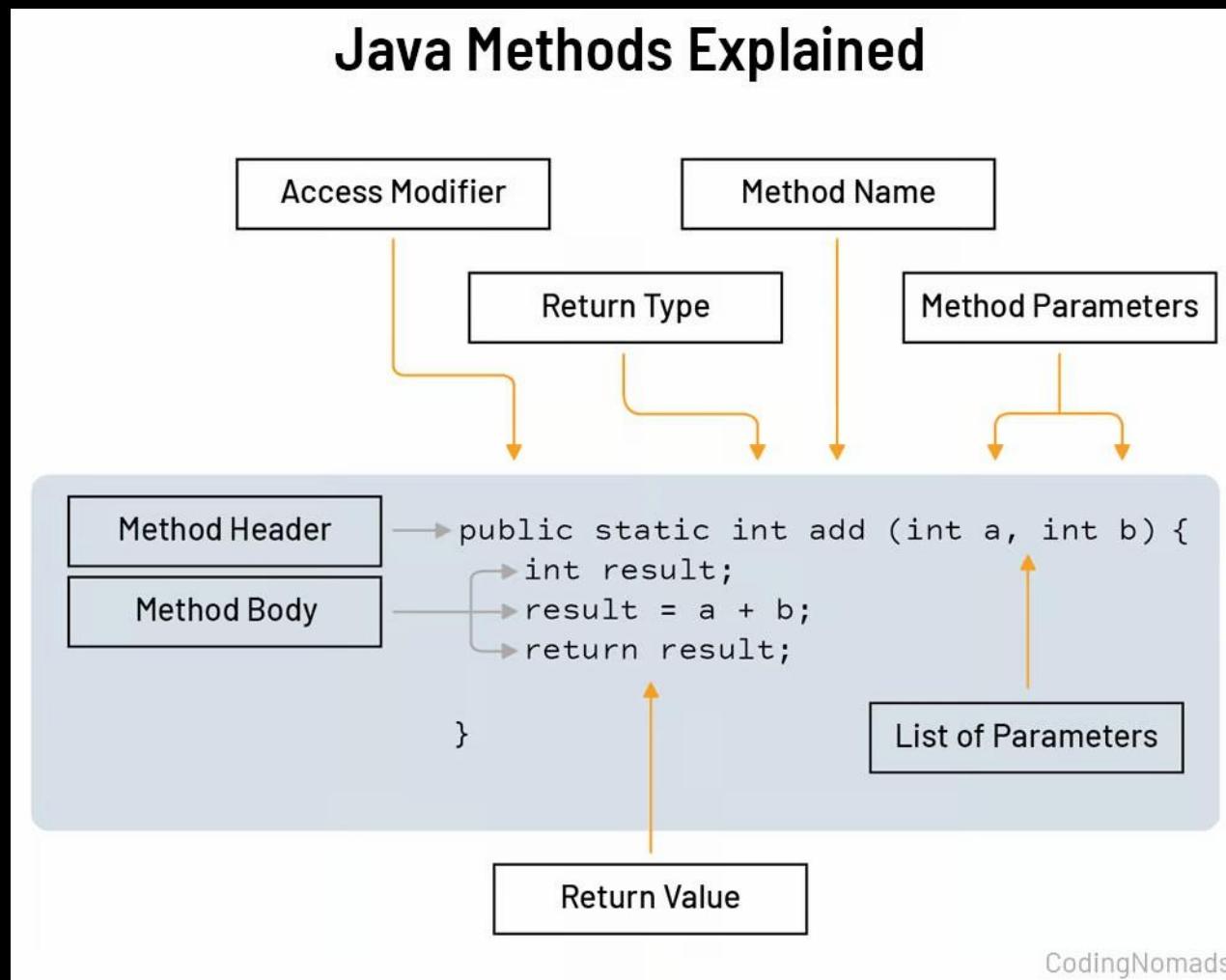


Warm-Up

- Hier Quiz einfügen

Noch ein bisschen Theorie ...



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

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?");
```

Random

```
Random random = new Random();
```

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

```
// Zufälliger Wert von 0.0 (inklusiv) 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();
```

Math-Klasse

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

Name	Rückgabetyp	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

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;  
    }  
}
```

equals-Methode

- Die equals-Methode überprüft, ob zwei Objekte gleich sind, also den gleichen Inhalt haben.

Achtung: Der Vergleich mit `==` gilt nur für die selben Objekte, der Vergleich mit `.equals()` für alle gleichen Objekte, also auch solche, die zwar verschieden sind, aber den gleichen Inhalt haben.

Beispiel: Zwei Gläser Erdbeermarmelade vom gleichen Hersteller sind gleich (oder zumindest ähnlich), aber nicht die selben.

Daher verwendet bei Objekten immer die `.equals()`-Methode.

Blöcke

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

- Blöcke gruppieren mehrere Befehle zusammen.
- Alle Methoden sind Blöcke.
- Wenn wir innerhalb von Blöcken Variablen erstellen, werden die Variablen beim Ende des Blocks wieder gelöscht.
 - 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

```
private static void warte(long anzahlMillisekunden) {  
    if (anzahlMillisekunden > 0) try {  
        Thread.sleep(anzahlMillisekunden);  
    } catch (InterruptedException e) {  
        throw new RuntimeException(e);  
    }  
}
```

```
private static void zahlenProgramm() {  
    Scanner scanner = new Scanner(System.in);  
    String eingabe = scanner.nextLine();  
    if (eingabe.equals("Zähle von 1 bis 10.")) {  
        System.out.println("Okay, mache ich.");  
        for (int i = 0; i < 10; i++) {  
            warte(500);  
            System.out.println(i);  
        }  
    }  
}
```

Arrays

```
// 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-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 = summe + zahlen[i];  
        i = i + 1;  
    }  
    return summe / zahlen.length;  
}
```

Problem: Alle Geschichten ausgeben

```
// Alle Geschichten ausgeben
int i = 0; // Erstelle eine ganze Zahl i, die den Index der aktuellen Geschichte repräsentiert.
while (i < geschichten.length) { // Solange wir nicht die Anzahl der Geschichten erreicht haben
    System.out.println(geschichten[i]); // Gib die aktuelle Geschichte aus.
    i = i + 1; // Erhöhe den Index der Geschichte um 1.
}
```

foreach-Schleife

```
// Ich will alle Geschichten hören.  
for (String geschichte : geschichten) { // Für jede Geschichte  
    System.out.println(geschichte); // Gib sie aus.  
}
```

```
// Ich will die längste Geschichte hören.  
String längsteGeschichte = "";  
for (String geschichte : geschichten) { // Für jede Geschichte  
    // Wenn sie länger ist als die bisher längste, soll die bisher längste die neue sein.  
    if (geschichte.length() > längsteGeschichte.length()) längsteGeschichte = geschichte;  
}  
System.out.println(längsteGeschichte); // Gib die längste aus
```

Verbessert: Durchschnitt mehrerer Zahlen

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

Zufallszahlen generieren

```
Random random = new Random();
```

```
// Zufallszahl von 0 (inklusiv) bis 10 (exklusiv)  
System.out.println(random.nextInt(10));
```

```
// Zufallszahl von 0.0 (inklusiv) bis 2.0 (exklusiv)  
System.out.println(random.nextFloat(2.0f));
```

```
// Zufalls-Wahrheitswert  
System.out.println(random.nextBoolean() ? "Glück" : "Pech");
```

Übungen zu libGDX



Im Kurs ...

Objektorientierung

- 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.

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 upper camel case ein, z.B. MeineTolleKlasse
- In die geschweifte Klammer kannst du sowohl Attribute als auch Methoden schreiben.

Konstruktor

- Ähnlich wie Methoden
- Werden aufgerufen, wenn eine Klasse instanziert wird (ein Objekt einer Klasse erstellt wird):
`<Modifikatoren> <Klassenname>(<Parameter>) {
}`
- Mit `this(<Parameter>);` kann ein Konstruktor einen anderen Konstruktor aufrufen.
- Schaue dir mal die Konstruktoren der Klasse

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 Instanzen/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.

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-Klasse

- Alle Klassen erweitern standardmäßig die Objekt-Klasse

Name	Rückgabetyp	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.

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	Rückgabetyp	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.

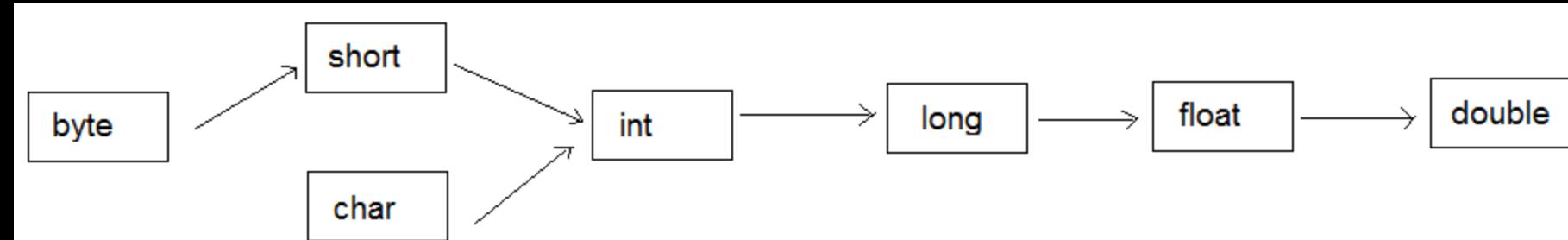
HashMap<K, V>

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

Name	Rückgabetyp	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

Automatische Typenkonvertierung (auto casting)

- Einige primitive Datentypen können in Java automatisch in andere konvertiert werden.
 - Integer können verwendet werden, als wären sie doubles. Das ergibt auch Sinn, da jede ganze Zahl eine Dezimalzahl ist.
- Objekte von Unterklassen können wie Objekte der Basisklassen verwendet werden.
 - Achtung: Anders als bei primitiven Datentypen wird das Objekt hierbei nicht verändert, nur anders aufgefasst. Wenn wir eine Eiche mit „der Baum“ bezeichnen, ändert es nichts daran, dass es sich um eine Eiche handelt.



Manuelle Typenkonvertierung (manual casting)

- Primitive Datentypen können manuell zu anderen umgewandelt werden.
 - Dabei können Informationen verloren gehen.
 - Wenn eine Dezimalzahl in eine ganze Zahl umgewandelt wird, fallen alle Nachkommastellen weg.

```
private static void gibAlsGanzeZahlAus(float dezimalzahl) {  
    int ganzeZahl = (int) dezimalzahl;  
    System.out.println(ganzeZahl);  
}
```

```
public static void main(String[] args) {  
    gibAlsGanzeZahlAus(1.7f); // 1  
    gibAlsGanzeZahlAus(-3.2f); // -3  
    gibAlsGanzeZahlAus(5.0f); // 5  
    gibAlsGanzeZahlAus(-103.725f); // -103  
}
```

Manuelle Typenkonvertierung (manual casting)

- Objekte können auch manuell umgewandelt werden.
- Wenn das Objekt nicht den Typen hat, zu dem umgewandelt wird, stürzt das Programm mit einer ClassCastException ab.
- Wenn die Konvertierung geklappt hat, können ab nun auch Methoden der abgeleiteten Klasse verwendet werden. Ein Baum könnte zum Beispiel eine Methode „fällen“ haben, die die Höhe des Baums auf 0 setzt.

Beispiel

```
private static class Pflanze {}  
private static class Blume extends Pflanze {}  
private static class Baum extends Pflanze {  
    private int Höhe = 10;  
  
    public void fällen() {  
        Höhe = 0;  
    }  
}  
private static class Eiche extends Baum {}  
  
public static void main(String[] args) {  
    Pflanze pflanze = new Eiche(); // Speichere eine Eiche in "pflanze".  
    pflanze.fällen(); // Die Klasse "Pflanze" hat die Methode "fällen" nicht.  
    Eiche eiche = (Eiche) pflanze; // Wir wissen: "pflanze" ist eine Eiche.  
    eiche.fällen(); // Funktioniert  
  
    Pflanze anderePflanze = new Blume(); // Speichere eine Blume in "anderePflanze".  
    Baum baum = (Baum) anderePflanze; // Die Blume ist kein Baum. Das Programm stürzt ab.  
}
```

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

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

Interfaces

- Interfaces werden immer von Klassen implementiert. Implementation ist ähnlich wie Erweiterung und verwendet das implements-Keyword
- Interfaces haben nur abstrakte Methoden, geben also nur die Methoden einer Klasse vor, nicht aber deren Verhalten
- Eignen sich als Methode, die man übergeben kann.

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

Umfrage

Bitte alles ausfüllen und auf "Absenden" klicken! ☺

