

Grundlagen der OO- Programmierung in Java

Einführung

Überblick

- Java Umgebung, Architektur, ...
- Eclipse: Editor und Debugging
 - Einstellungen, Funktionalität
 - Erstellen eines Projekts
- Datentypen
 - Speicherverwaltung
- Statische Methoden

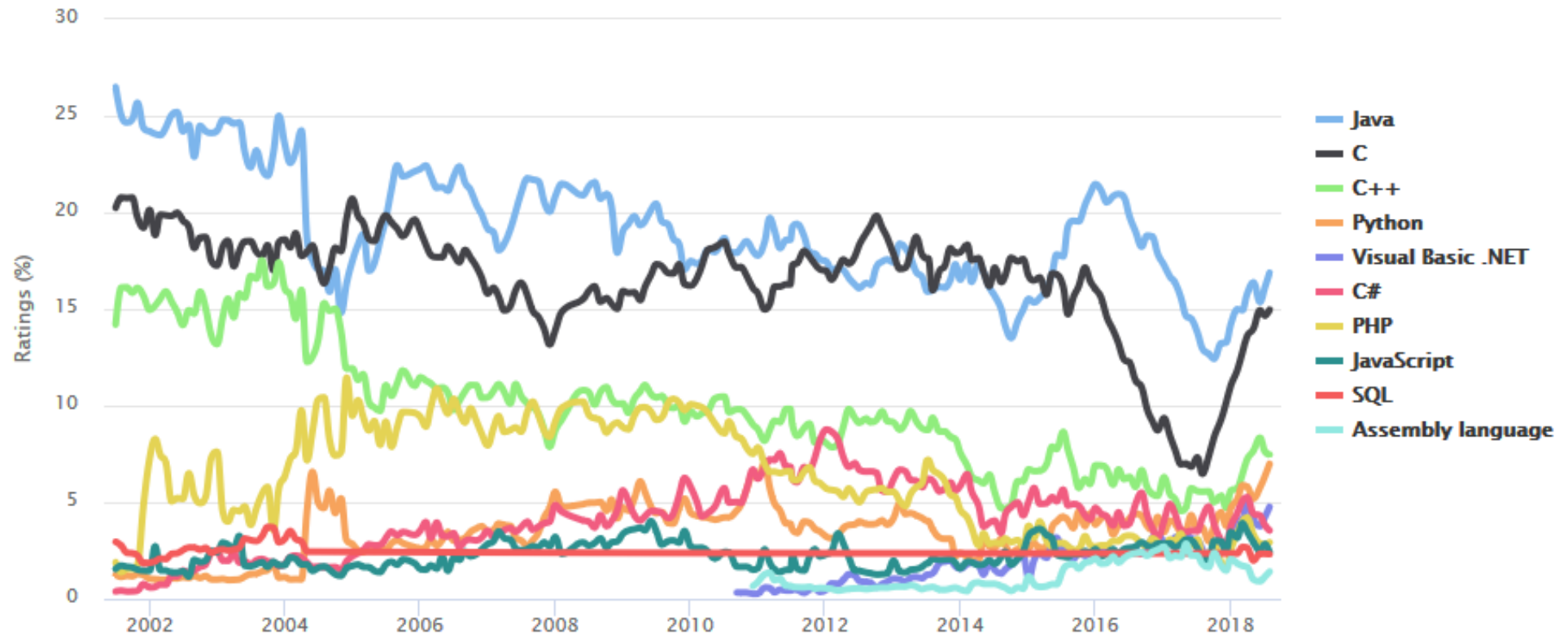
Warum Java?

Top 10 Programming Languages

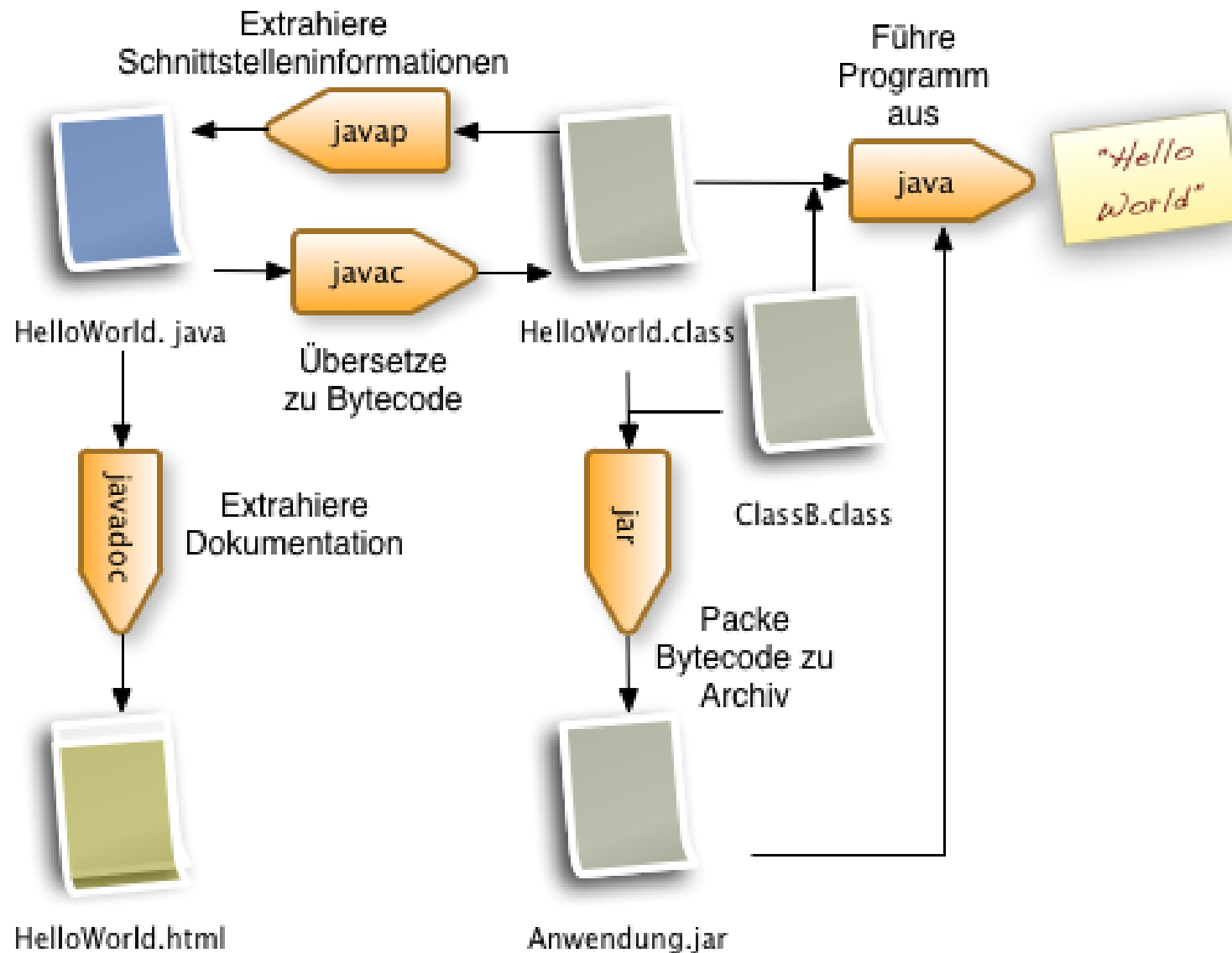
Beliebtheit, Anzahl neue Applikationen, Anzahl Zeilen Code, ...

TIOBE Programming Community Index

Source: www.tiobe.com



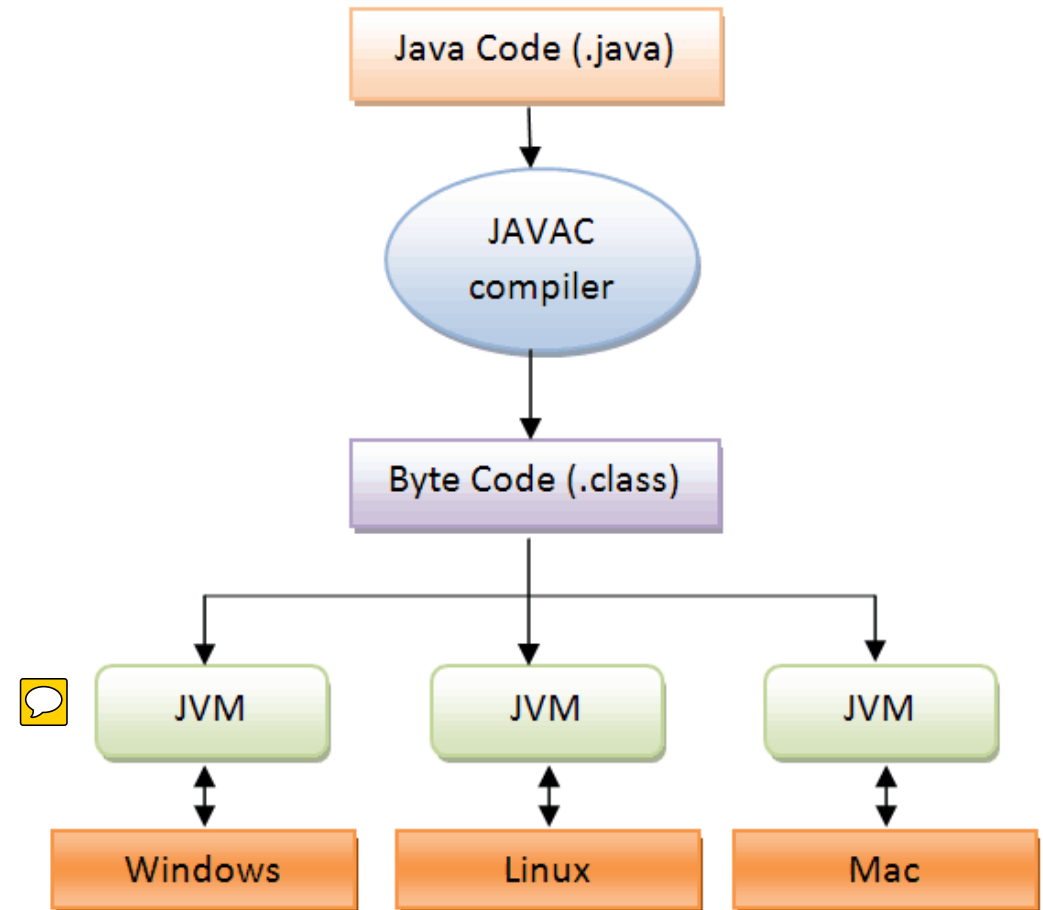
Die wichtigsten Funktionen der JDK



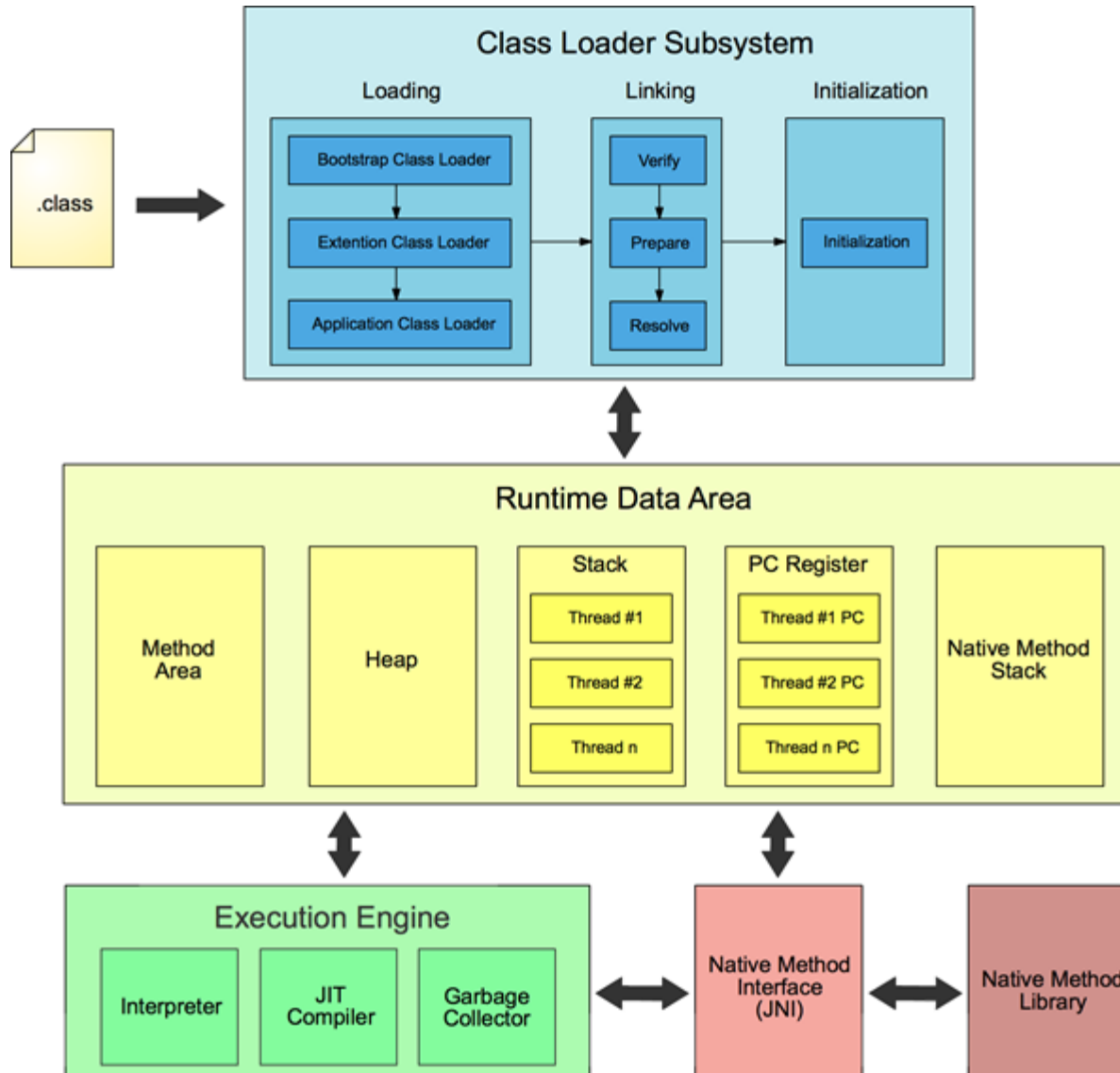
Die Java Virtual Machine (JVM)

Write once, run everywhere

- Für jede Plattform existiert eine separate JVM, welche den Bytecode im Maschinencode übersetzt und ausführt.



Die Aufgaben der Java Virtual Machine (JVM)



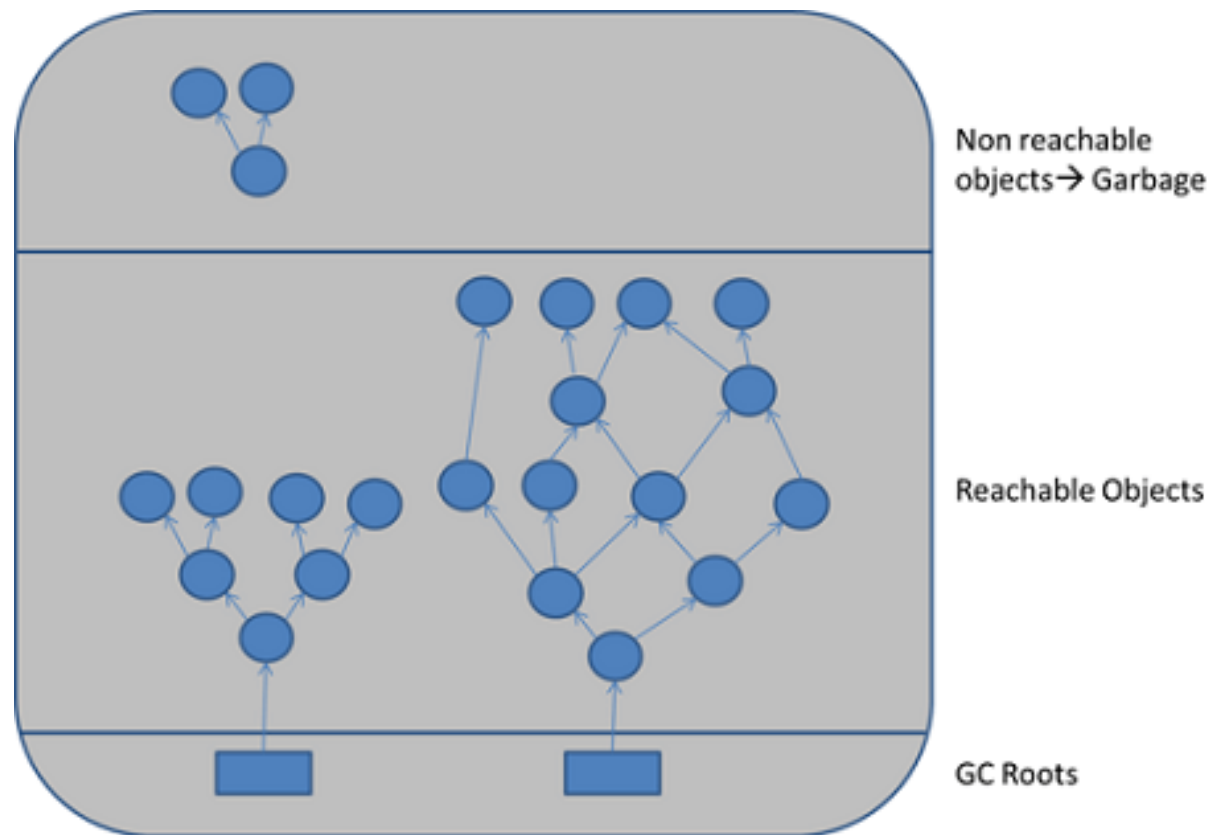
Klassen ins Memory laden, auf Vollständigkeit prüfen, ev. weitere Klassen nachladen und initialisieren (z.B. statische Variablen oder Blöcke).

Memory Bereiche für Methoden, Objekte, Variablen und für (Laufzeit)-Register bereit stellen und verwalten.

Code interpretieren, übersetzen und ausführen. Heap (sobald nötig) aufräumen.

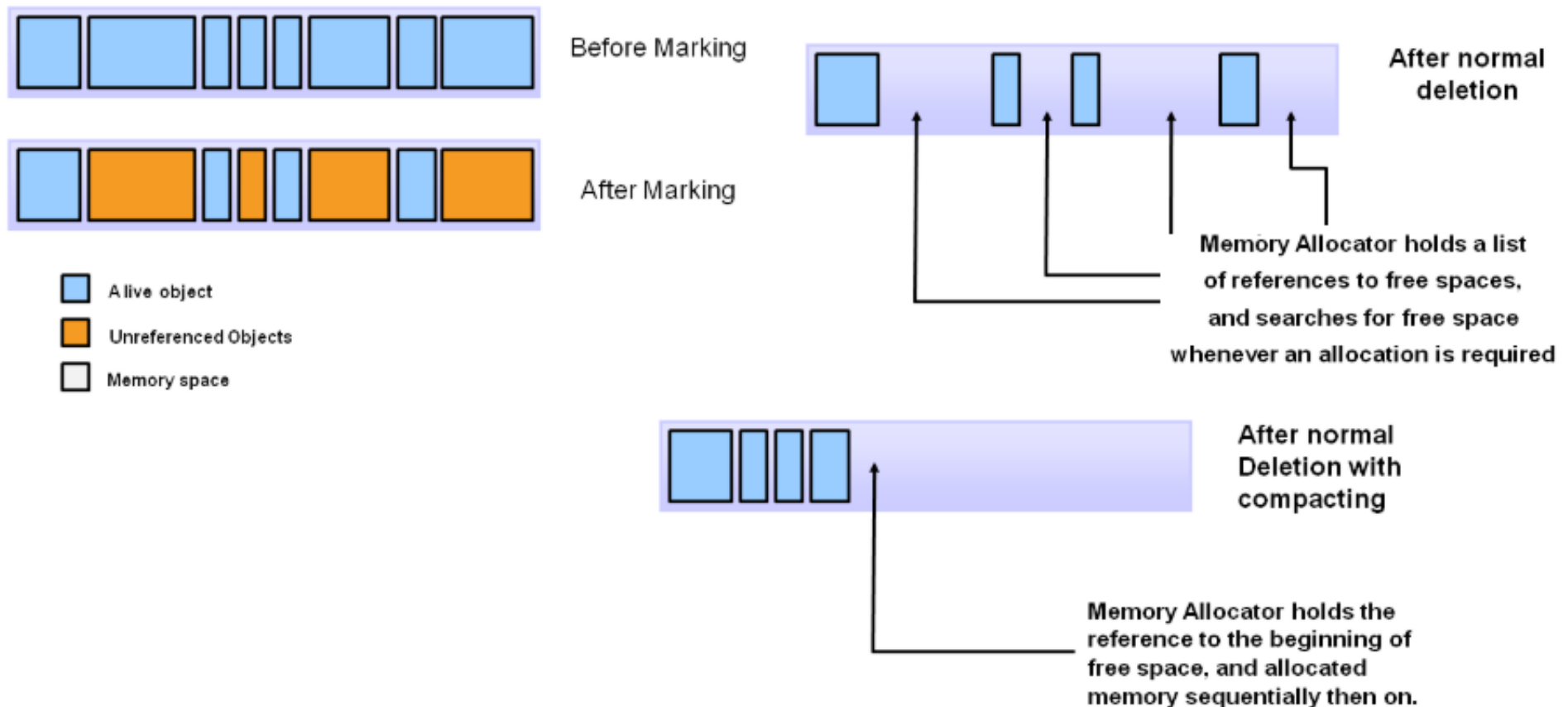
Der Garbage Collector

Der Gargabe Collector überprüft, welche Speicherbereiche im Heap (noch) verwendet werden und markieren diese. Die nicht mehr referenzierten Objekte werden automatisch weggeräumt (Freigabe des Speicherbereichs).



Der Garbage Collector

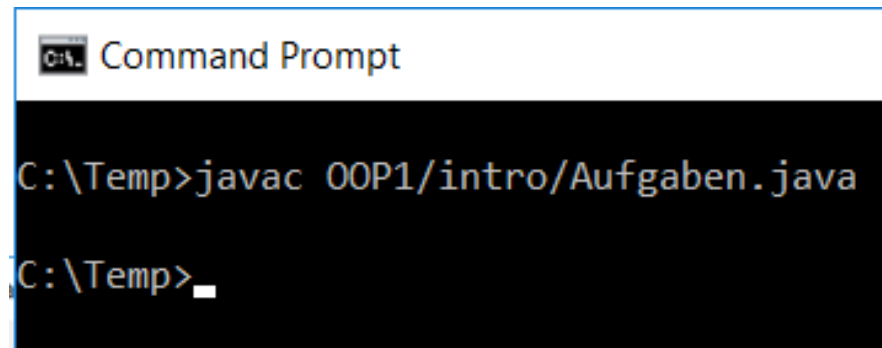
Die Speicherbereiche werden markiert, nicht referenzierte Objekte gelöscht und zuletzt die übrig gebliebenen Bereiche komprimiert.



Java Programm auf der Konsole

- Java Programm können auch auf der Konsole kompiliert und laufen gelassen werden

- Kompilieren

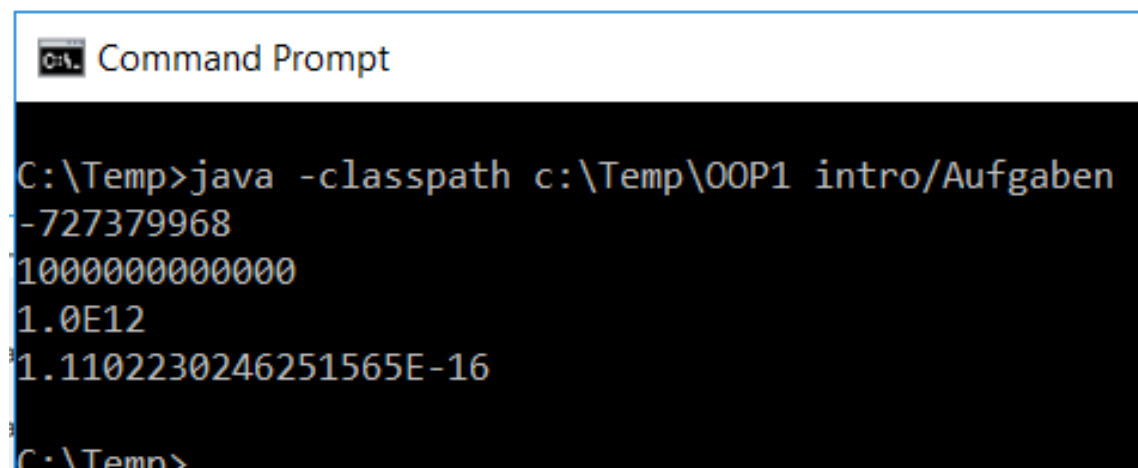


```
Command Prompt

C:\Temp>javac OOP1/intro/Aufgaben.java

C:\Temp>
```

- Ausführen

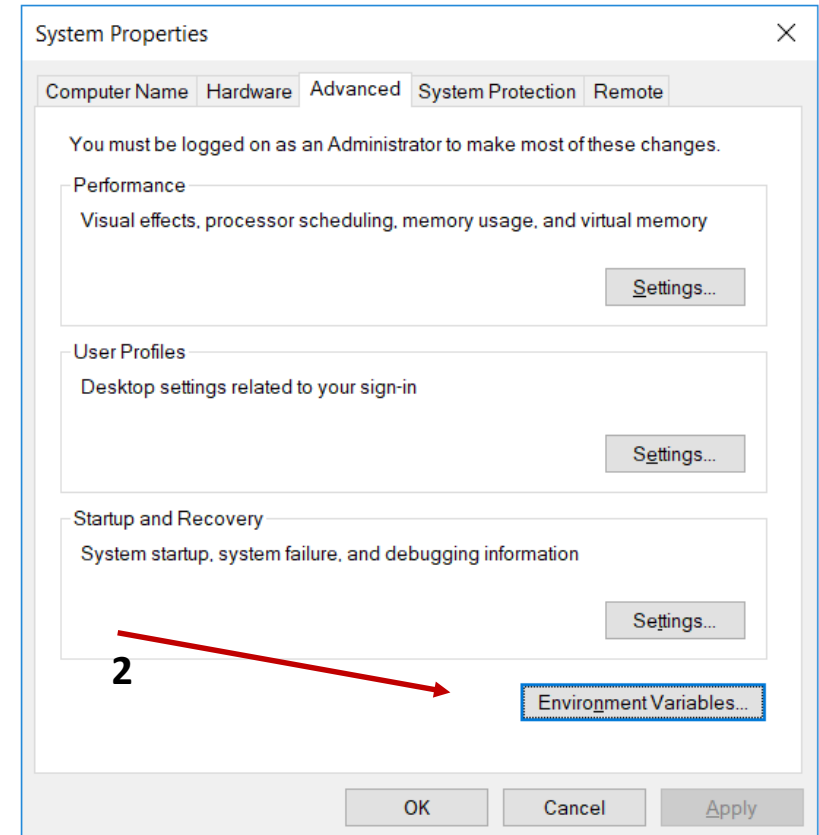


```
Command Prompt


C:\Temp>java -classpath c:\Temp\OOP1 intro/Aufgaben
-727379968
1000000000000
1.0E12
1.1102230246251565E-16

C:\Temp>
```

- Pfad setzen, falls nötig:



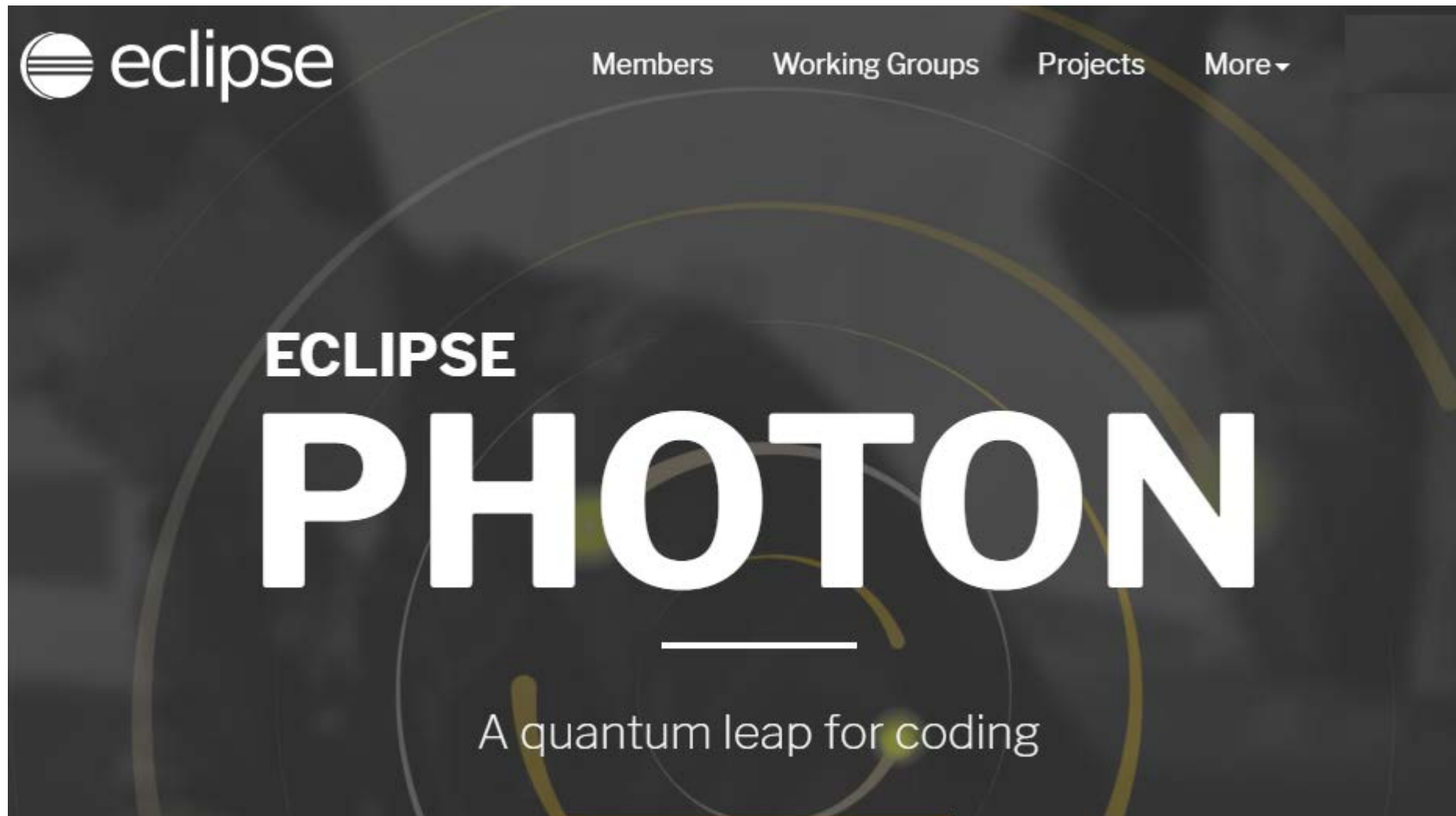
3



The screenshot shows the 'System variables' tab in the Windows Environment Variables control panel. A red arrow points to the 'Path' variable, which is highlighted in blue. The 'Path' variable's value is 'C:\Program Files\Java\jdk-10\bin;C:\Progra'. Other visible variables include OS (Windows_NT), PATHEXT (.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF), and PROCESSOR_ARCHITECTURE (AMD64).

Variable	Value
OS	Windows_NT
Path	C:\Program Files\Java\jdk-10\bin;C:\Progra
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF
PROCESSOR_ARCHITECTURE	AMD64

Eclipse IDE (Integrierte Entwicklungsumgebung)



Übersicht der Funktionalität

Eclipse

- ist eine integrierte Entwicklungsumgebung (IDE)
- stellt für die verschiedenen Arten und Dokumente eines Projekts verschiedene Editoren und Assistenten zur Verfügung
 - Projekt Explorer
 - File Ex
- stellt eine einfache Umgebung zum Testen des Codes zur Verfügung (JUnit Tests).

Überblick über die verschiedenen Fenster

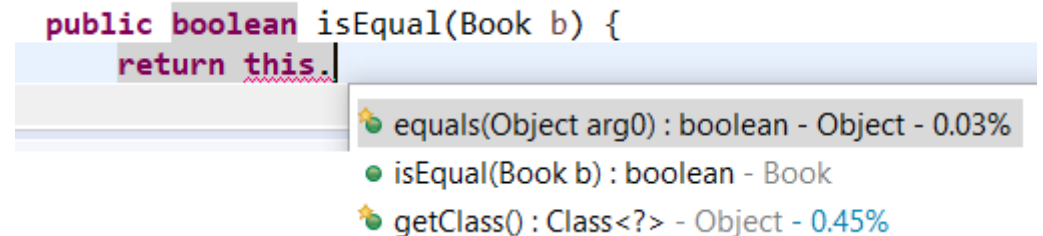
The screenshot displays an IDE interface with several windows and annotations:

- Packages Übersicht:** A tree view on the left showing the project structure, including `ArrayExamples`, `KlassenExamples`, `src`, `library` (containing `Book.java`, `Music.java`, `Person.java`, `Video.java`), `libraryTest`, `JRE System Library [JavaSE-10]`, `JUnit 5`, `PolymorphieExample`, and `StringExamples`.
- Java Code Editor:** The central window showing the code for `Book.java`. The code includes a `Book` class with private attributes (`id`, `title`, `writer`, `pages`), a constructor, and a `getId()` method. A red label "Java Code Editor" is placed over the code.
- Outline:** A window on the right showing the structure of the active class `Book`, listing attributes (`id : int`, `title : String`, `writer : Person`, `pages : int`) and methods (`Book(String, Person, int)`, `getId() : int`, `getTitle() : String`, `getWriter() : Person`). A red label "Attribute und Methoden der aktiven Klasse" is placed over the outline.
- Problems:** A window at the bottom left showing a list of errors and warnings. It contains two entries: `StringBuilderTest: 16573` and `StringTest: 119889`. A red label "Fehler und Warnungen" with two yellow arrows pointing to the entries is placed below the window.
- Console:** A window at the bottom right showing the output of the program. It contains the text: `<terminated> kap2Tests [JUnit] C:\Program Files\Java\jre-10\bin\javaw.exe (19.08.2018, 14:22:42)`. A red label "Programm Ausgabe" with two yellow arrows pointing to the text is placed below the window.
- Search:** A window at the bottom center showing search results. A red label "Such-Ergebnisse" with a yellow arrow pointing to the window is placed below it.

Der Editor

- Der Editor unterstützt das Arbeiten während der Text-Eingabe mit Hilfe von Code Completion
- Die Code Completion arbeitet nach dem Prinzip der Volltextsuche und zeigt dabei jeweils folgende Details an:
 - Signatur der Methode oder des Typs

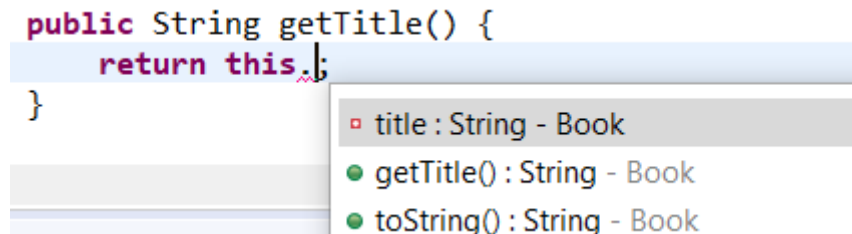
```
public boolean isEqual(Book b) {  
    return this.  
}
```



- equals(Object arg0) : boolean - Object - 0.03%
- isEqual(Book b) : boolean - Book
- ★ getClass() : Class<?> - Object - 0.45%

- Parameterliste
- Relevanz

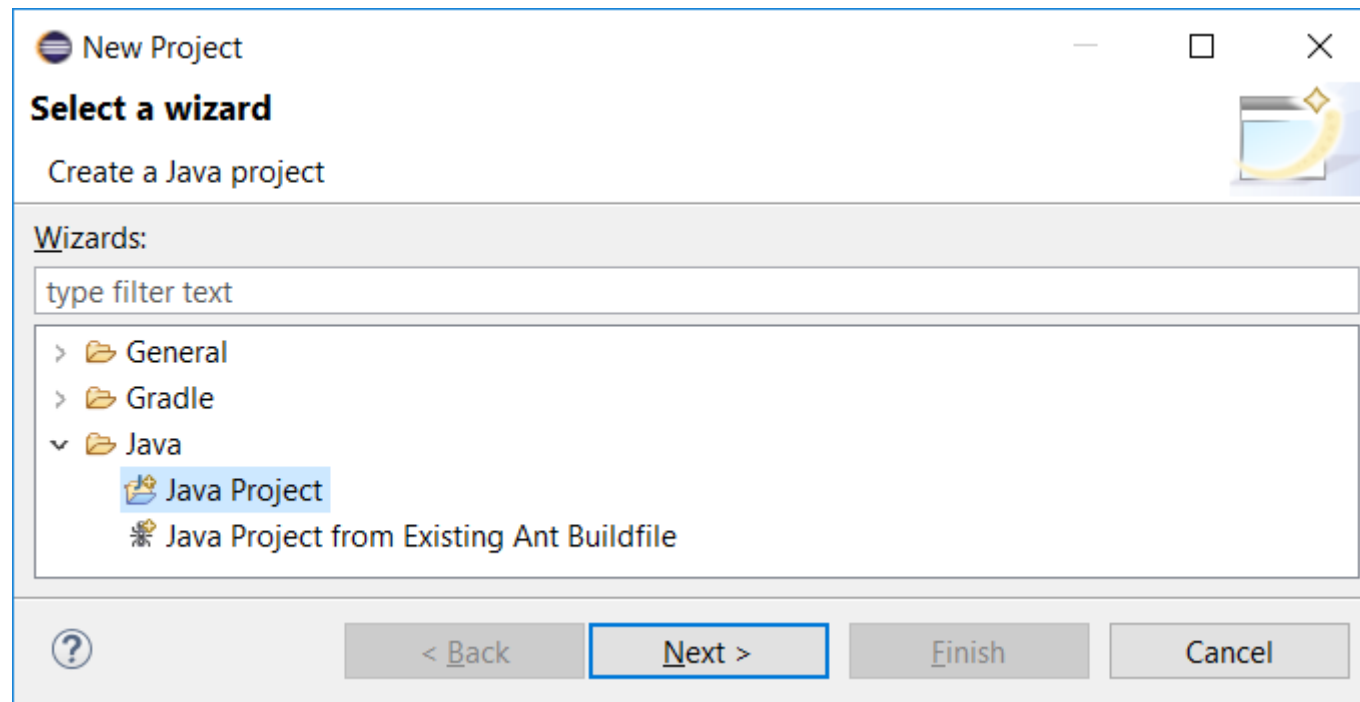
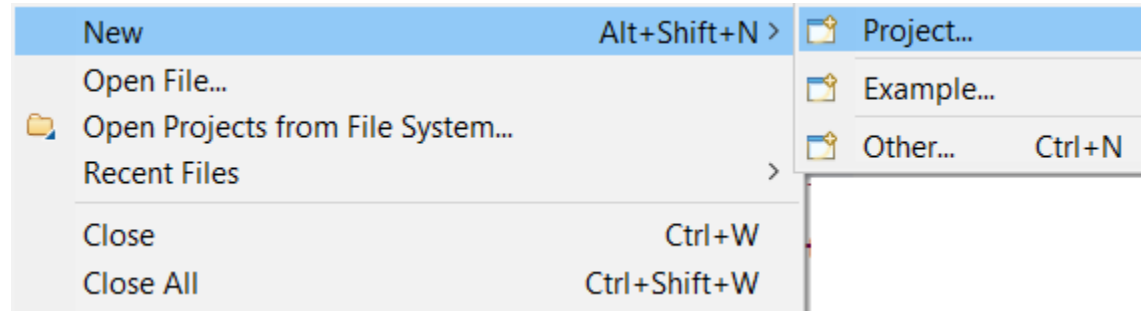
```
public String getTitle() {  
    return this.  
}
```



- title : String - Book
- getTitle() : String - Book
- toString() : String - Book

Java Projekt erzeugen

- Neues Projekt erstellen



Details definieren

Projekt Name und Ort (Location)

JRE wählen

Sourcen und
Binaries getrennt
halten

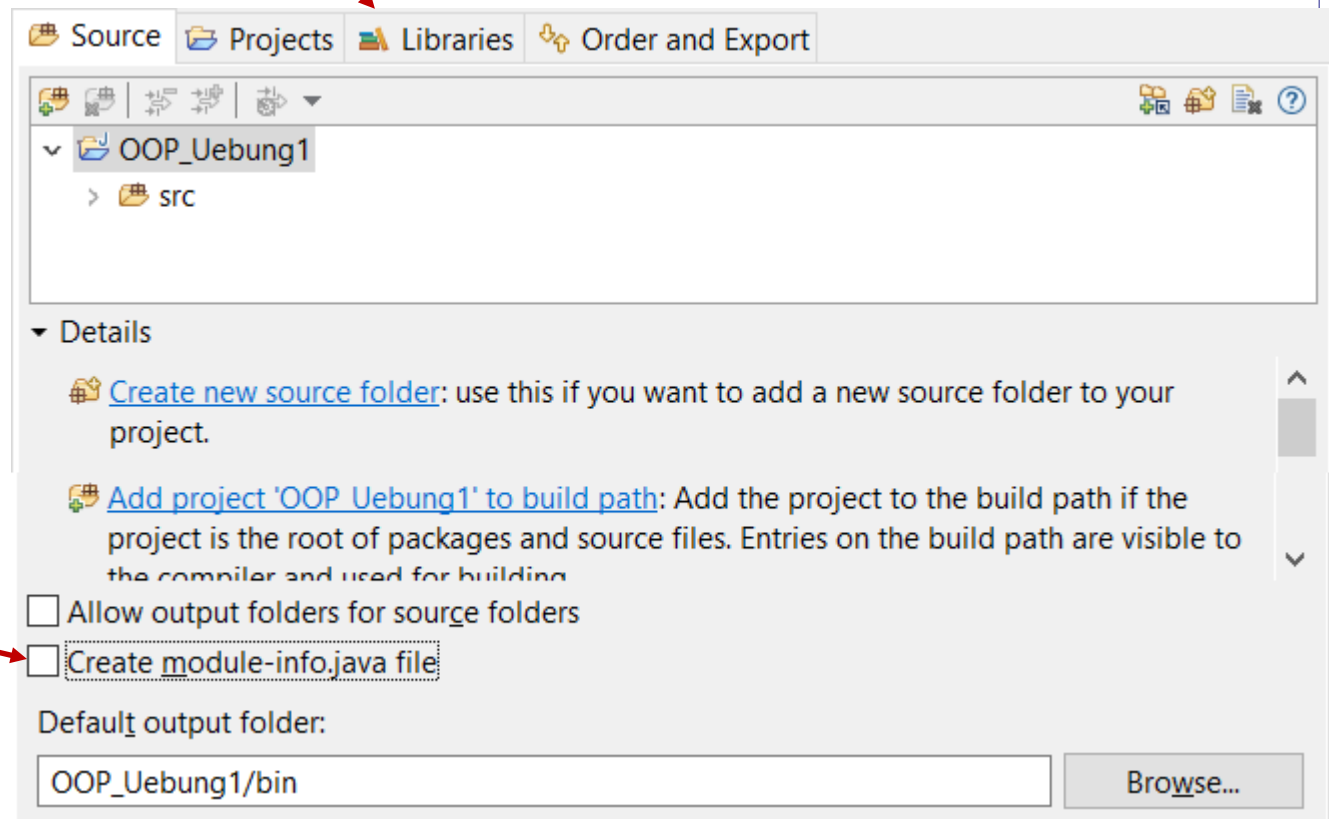
The screenshot shows a 'Details' dialog box with the following sections:

- Project name:** A text field containing 'OOP_Uebung1'.
- Use default location:** An unchecked checkbox.
- Location:** A text field containing 'C:\Temp\Ueb1' and a 'Browse...' button.
- JRE:** A section with three radio buttons:
 - ☒ Use an execution environment JRE: A dropdown menu showing 'JavaSE-10'.
 - ☐ Use a project specific JRE: A dropdown menu showing 'jre-10'.
 - ☐ Use default JRE (currently 'jre-10') with a link to [Configure JREs...](#)
- Project layout:** A section with two radio buttons:
 - ☐ Use project folder as root for sources and class files.
 - ☒ Create separate folders for sources and class files with a link to [Configure default...](#)

Details definieren

Ev. Libraries einbinden (z.B: JUnit)

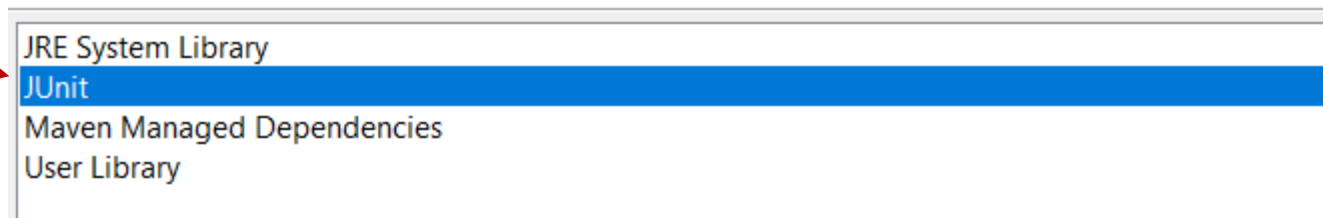
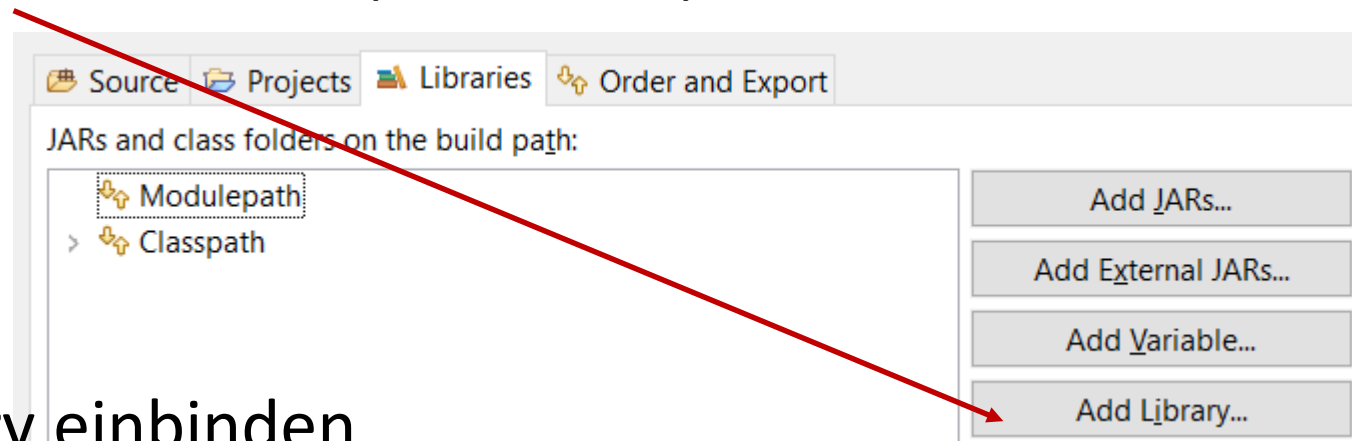
Kein module-info.java
File erzeugen lassen



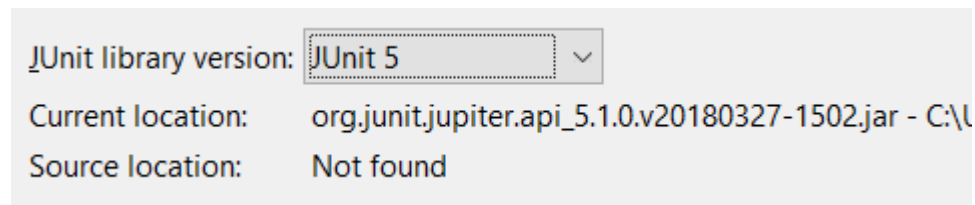
Details definieren

Ev. Libraries einbinden (z.B: JUnit)

JUnit Library einbinden



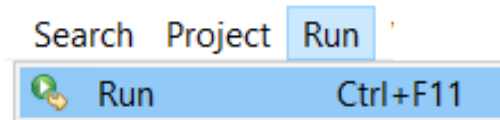
Installierte JUnit Version wählen.



Java Anwendung starten

Java Anwendungen können direkt in Eclipse gestartet werden

- Menu-Punkt -> Run

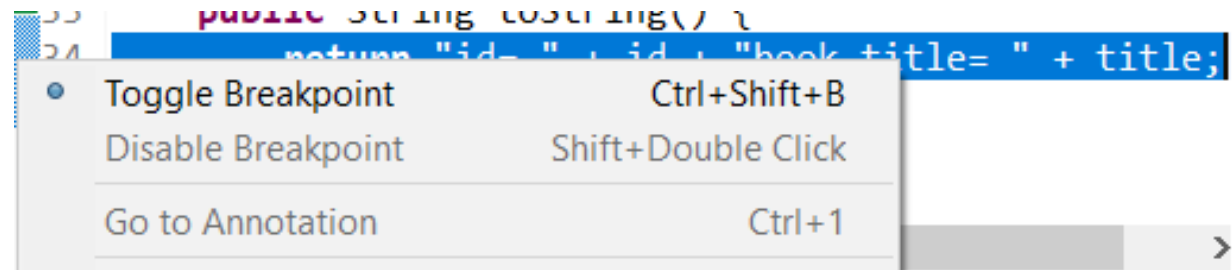
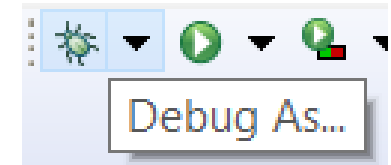


- Oder Run-Icon in der Toolbar



Anwendungen debuggen

- Der Debugger erlaubt das Unterbrechen der Programmausführung.
- Der Haltepunkt wird definiert durch Setzen eines Breakpoints (rechte Maustaste)



- Ist der Programmfluss angehalten, wird der aktuelle Ausführungspunkt im Quellcode angezeigt

Name	Value
no method return	
▼ this	Motor (id=413)
> ▢ model	"RX-400" (id=414)
⦿ power	35.5
▲ speed	0
Ⓛ s	200



Datentypen

Warum Typen?

- Gegeben sei der folgende Speicherinhalt:

0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	1	1	1	1	0	0	0	1	0	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- Was ist dessen Bedeutung?
 - Als int Typ → 881389854
 - Als float Typ → 2.55074E-07
 - Als char Typ → "S & A 2" (irgend ein String, abhängig von Codierung)

Variablen

- Eine Variable steht für eine Adresse im Speicher (einen Speicherplatz).
- Die Grösse des Speicherplatzes für diese Variable hängt vom Typ der Variable ab.
- Über den Bezeichner/Namen der Variable kann das Programm (später) auf den darin gespeicherten Wert zugreifen.
- Lokale Variablen werden innerhalb einer Methode definiert, der zugehörige Speicherplatz wird nach dem Ablauf der Methode wieder freigegeben (der Wert gelöscht).
- Klassen-Variablen existieren genau so lange wie die Klasse zu welcher sie gehören.

Variablen-Deklaration und Initialisierung

Die Deklaration erfolgt über die Syntax

Typ Name;

Beispiele

`int a;` 

`string b;` 

`float c;`

Deklaration mehrerer Variablen

`int a1, a2, a3;`

`string b1, b2;`

Initialisierung (Wert-Belegung)

Bei der Deklaration

`int a = 17;` *// Deklaration mit Initialisierung*

`string b = "Das ist ein Text";`

`float c = 3.14159F;`

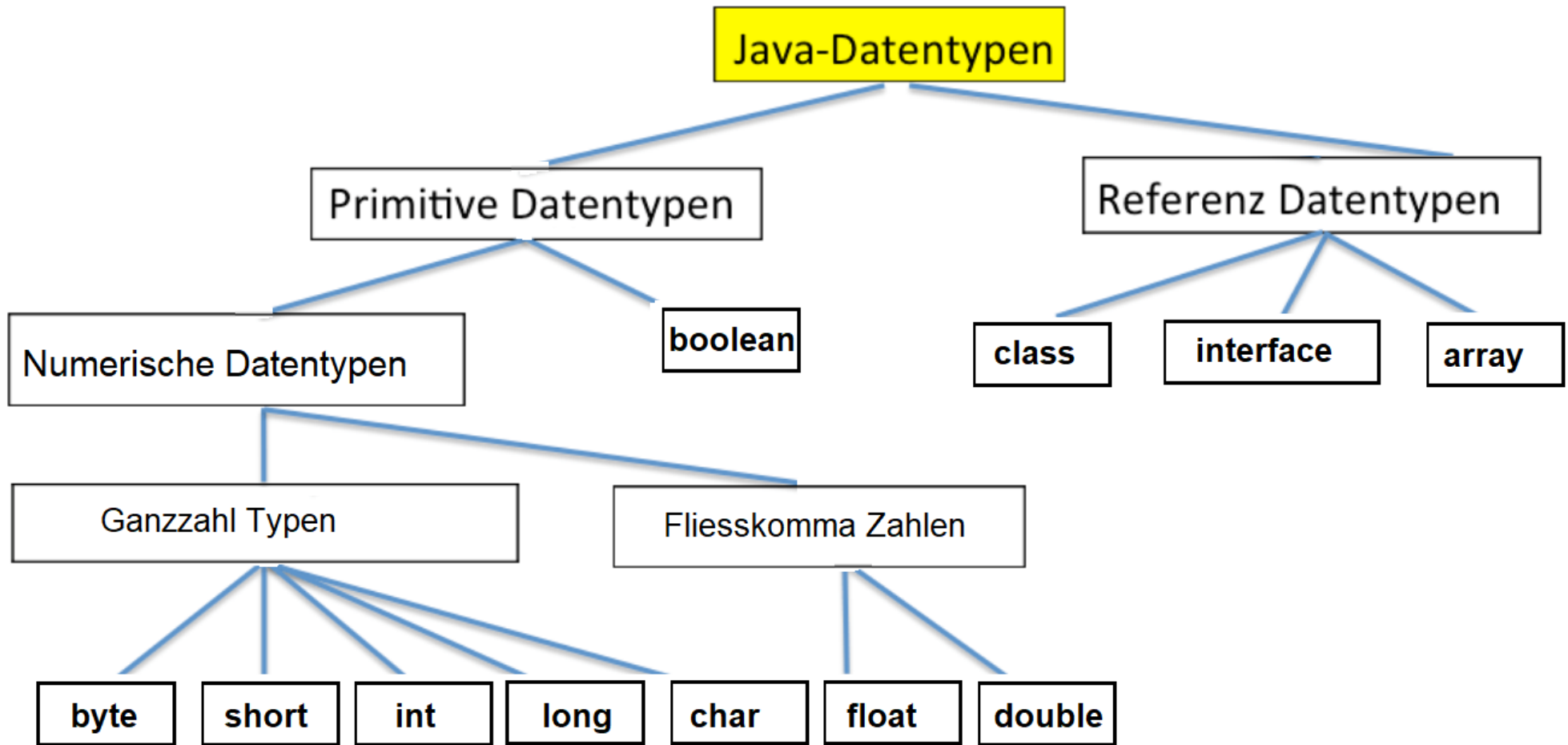
Oder zu einem späteren Zeitpunkt

`int a1, a2;` *// Deklaration*

`a1 = 17;` *// Initialisierung danach*

`a2 = 243;`

Datentypen Grundstruktur



Ganzzahl-Typen

Typ	Minimum	Maximum	Grösse	Vorzeichen
byte	0	255	8 Bit	Nein
sbyte	-128	127	8 Bit	Ja
short	-32.768	32.767	16 Bit	Ja
int	-2.147.483.648	2.147.483.647	32 Bit	Ja
long	-9.223.372.036.854.775.808	9.223.372.036.854.775.807	64 Bit	Ja

Darstellung eines sbyte Werts:

17 → 0 001 0001

$$v_z \quad 2^6 2^5 2^4 \quad 2^3 2^2 2^1 2^0$$

Darstellung eines short Werts:

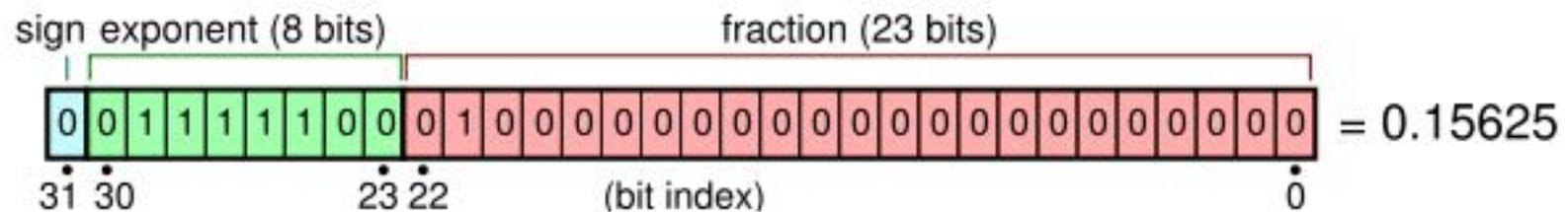
385 → 0 000 0001 1000 0001

$$\mathbf{v_Z} \quad 2^{14}2^{13}2^{12} \quad 2^{11}2^{10}2^92^8 \quad 2^72^62^52^4 \quad 2^32^22^12^0$$

Gleitkomma-Typen

Typ	Bereich	Nachkommastellen (Dezimal-Stellen)	Grösse	Verwendung
float	$\pm 1.5\text{e-}38$ bis $\pm 3.4\text{e}38$	7	32 Bit	float x = 1.234F;
double	$\pm 5.0\text{e-}308$ bis $\pm 1.7\text{e}308$	15-16	64 Bit	double y = 1.234; double z = 3D;
decimal	$\pm 1.0 \times 10^{-28}$ bis $\pm 7.9 \times 10^{28}$	bis 29	128 Bit	decimal d = 9.1m

IEEE Darstellung für float (32 Bit)

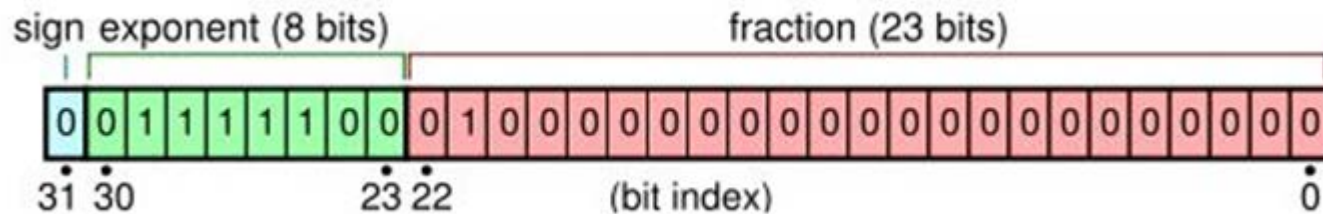


Typ-Umwandlung

- Gegeben sei die folgende Zahl als int Typ → 13771716

0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	1	1	1	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

→ Umwandlung/Umrechnung in float ?



Eventuell ist die Zahl nicht exakt darstellbar als Zweierpotenz multipliziert mit der Mantisse → Approximation.

Typ-Umwandlung

- Beim Umwandeln einer int Zahl in eine float Zahl gehen Stellen (Präzision) verloren:

```
public static void main(String[] args) {  
    int a = 137717111;  
    float b = a;  
    System.out.printf("%n a= %d", a);  
    System.out.printf("%n b= %.0f", b);  
    float c = 567.12345f;  
    System.out.printf("%n c= %.5f", c);  
}
```

```
a= 137717111  
b= 137717104  
c= 567.12347
```

Bereiche verschiedener Datentypen

```
int a = 62352;
int b2 = 1142342;
long b3 = 1142342;
float b4 = 1142342;
double b5 = 1142342;
float b6 = 1142341.9F;
double b7 = 1142341.91234123;
BigDecimal b8 = new BigDecimal(1142341.91234123);

p.printf("int:      %d * %d = %d %n", a, b2, a * b2);
p.printf("long:     %d * %d = %d %n", a, b3, a * b3);
p.printf("float:    %d * %.1f = %f %n", a, b4, a * b4);
p.printf("double:   %d * %.1f = %f %n", a, b5, a * b5);
p.printf("float b6: = %.3f %n", b6);
p.printf("double:  %d * %.8f = %.10f %n", a, b7, a * b7);
p.printf("BigDecimal multiply %.10f %n", b8.multiply(new BigDecimal(a)));
```

```
int:      62352 * 1142342 = -1787135648
long:     62352 * 1142342 = 71227308384
float:    62352 * 1142342.0 = 71227310080.000000
double:   62352 * 1142342.0 = 71227308384.000000
float b6: = 1142341.875
double:   62352 * 1142341.91234123 = 71227302918.3003800000
BigDecimal multiply 71227302918.3003781512
```

Implizite/Explizite Typ-Konversion

<code>int a1 = 5;</code>	<code>// keine Konversion, 5 ist ein int.</code>
<code>int a2 = 3.723;</code>	<code>// Fehler</code>
<code>int a3 = (int)3.723;</code>	<code>// ok, wird «abgeschnitten»</code>
<code>float b1 = 123;</code>	<code>// implizite Konversion, 123 ist ein int</code>
<code>float b2 = 1.23;</code>	<code>// Fehler, 1.23 ist ein double, implizite Konversion nicht möglich</code>
<code>float b3 = 1.23F;</code>	<code>// keine Konversion, 1.23F ist ein float</code>
<code>double c1 = 1.23;</code>	<code>// keine Konversion, 1.23 ist ein double</code>
<code>double c2 = 1.23F;</code>	<code>// implizite Konversion, 1.23F ist ein float</code>
<code>boolean d1 = true;</code>	<code>// ok</code>
<code>boolean d2 = (boolean) 0;</code>	<code>// 0 ist ein int und kann nicht zu boolean konvertiert werden.</code>
<code>char d3 = 'c';</code>	<code>// keine Konversion, c ist ein char.</code>
<code>char d4 = (char)99;</code>	<code>// explizite Konversion. Der Wert von d2 ist gleich 'c'.</code>
<code>long e1 = 678;</code>	<code>// implizite Konversion, 123 ist ein int.</code>
<code>long e2 = 678L;</code>	<code>// ok</code>
<code>int e3 = 678L;</code>	<code>// Fehler</code>
<code>int e4 = (int)678L;</code>	<code>// explizite Konversion möglich, falls der long genügend kurz ist.</code>

Weitere vordefinierte Typen

Typ	Grösse	Werttyp	Beschreibung
Object	-	nein	Referenz auf ein Objekt
String	-	nein	Referenz auf einen String (Text-Zeichen)
bool	8 Bit	ja	Logischer Wert true oder false
char	16 Bit	ja	Unicodezeichen 0 ... 65.535

Die wichtigsten Arithmetischen Operatoren

Operator	Bedeutung	
+	1: 2:	Addition von zwei numerischen Werttypen. Positives Vorzeichen vor einem numerischen Werttyp
-	1: 2:	Subtraktion von zwei numerischen Werttypen. Negatives Vorzeichen vor einem numerischen Werttyp
*		Multiplikation von zwei numerischen Werttypen.
/		Division von zwei numerischen Werttypen.
%		Modulo (Restwert einer Division zweier Operanden).
++	1: 2:	Inkrement eines Werttyps als Präfix. Er erhöht den Inhalt des Operanden um 1. Der als Präfix geführte Operator liefert den um 1 erhöhten Wert bevor der Ausdruck ausgewertet ist. Inkrement eines Werttyps als Suffix. Er erhöht den Inhalt des Operanden um 1. Der als Suffix geführte Operator liefert den um 1 erhöhten Wert erst nachdem der Ausdruck ausgewertet ist.
--	1: 2:	Dekrement eines Werttyps als Präfix. Er erniedrigt den Inhalt des Operanden um 1. Der als Präfix geführte Operator liefert den um 1 erniedrigten Wert bevor der Ausdruck ausgewertet ist. Dekrement eines Werttyps als Suffix. Er erniedrigt den Inhalt des Operanden um 1. Der als Postfix geführte Operator liefert den um 1 erniedrigt Wert erst nachdem der Ausdruck ausgewertet ist.

Logische Operatoren

Operator	Bedeutung
&&	Logischer UND Operator. Der Operator wertet zuerst den links vom Operatorzeichen stehenden Ausdruck aus. Wenn dieser Ausdruck true ist, und nur dann, wird der rechts vom Operatorzeichen stehende Ausdruck ausgewertet. Sind beide Ausdrücke true, gibt der Operator true zurück, ansonsten false.
//	Logischer ODER Operator. Der Operator wertet zuerst den links vom Operatorzeichen stehenden Ausdruck aus. Wenn dieser Ausdruck true ist, gibt der Operator sofort true zurück, ansonsten wertet der Operator den rechts vom Operatorzeichen stehenden Ausdruck aus und dessen Wahrheitswert zurück.
true	Operator für true. Dieser Operator kann für Schleifensteuerungen und für Verzweigungen verwendet werden.
false	Operator für false. Dieser Operator kann für Schleifensteuerungen und für Verzweigungen verwendet werden.

Vergleichsoperatoren

Operator	Bedeutung
==	Vergleichsoperator. Prüft die Ausdrücke der beiden Operanden auf Gleichheit. Ist diese gegeben, gibt der Vergleichsoperator true zurück.
!=	Ungleichheitsoperator. Prüft die Ausdrücke der beiden Operanden auf Ungleichheit. Ist diese gegeben, gibt der Vergleichsoperator true zurück.
<	Kleiner Operator. Prüft die Ausdrücke der beiden Operanden auf kleiner. Ist der Wert des linken Ausdrucks kleiner als derjenige des rechten Ausdrucks, gibt der Operator true zurück.
>	Grösser Operator. Prüft die Ausdrücke der beiden Operanden auf grösser. Ist der Wert des linken Ausdrucks grösser als derjenige des rechten Ausdrucks, gibt der Operator true zurück.
<=	Kleiner oder gleich Operator. Prüft die Ausdrücke der beiden Operanden auf kleiner oder gleich. Ist der Wert des linken Ausdrucks kleiner oder gleich dem Wert des rechten Ausdrucks gibt der Operator true zurück.
>=	Grösser oder gleich Operator. Prüft die Ausdrücke der beiden Operanden auf grösser oder gleich. Ist der Wert des linken Ausdrucks grösser oder gleich dem Wert des rechten Ausdrucks gibt der Operator true zurück.

Zuweisungsoperatoren

Operator	Bedeutung
=	Weist x den Wert von y zu.
+=	Weist x den Wert von $x + y$ zu.
-=	Weist x den Wert von $x - y$ zu.
*=	Weist x den Wert von $x * y$ zu.
/=	Weist x den Wert von x / y zu.

Beispiel	Bedeutung
<code>a = 3 ;</code>	Weist a den Wert 3 zu.
<code>a += 3 ;</code>	Weist a den Wert von $a + 3$ zu.
<code>a -= 3 ;</code>	Weist a den Wert von $a - 3$ zu.
<code>a *= 3 ;</code>	Weist a den Wert von $a * 3$ zu.
<code>a /= 3 ;</code>	Weist a den Wert von $a / 3$ zu.

Präzedenz von Ausdrücken (Bindungsstärke)

Bindungsstärke nimmt ab

Gruppe	Operatoren
Primär	x.y, f(x), a[x], x++, x-- ¹
Unär	+, -, !, ~, ++x, --x
Arithmetisch multiplikativ	*, /, %
Arithmetisch additiv	+, -
Relational	<, >, <=, >=
Gleichheit / Ungleichheit	==, !=
Zuweisung	=, +=, -=, *=, /=, ...

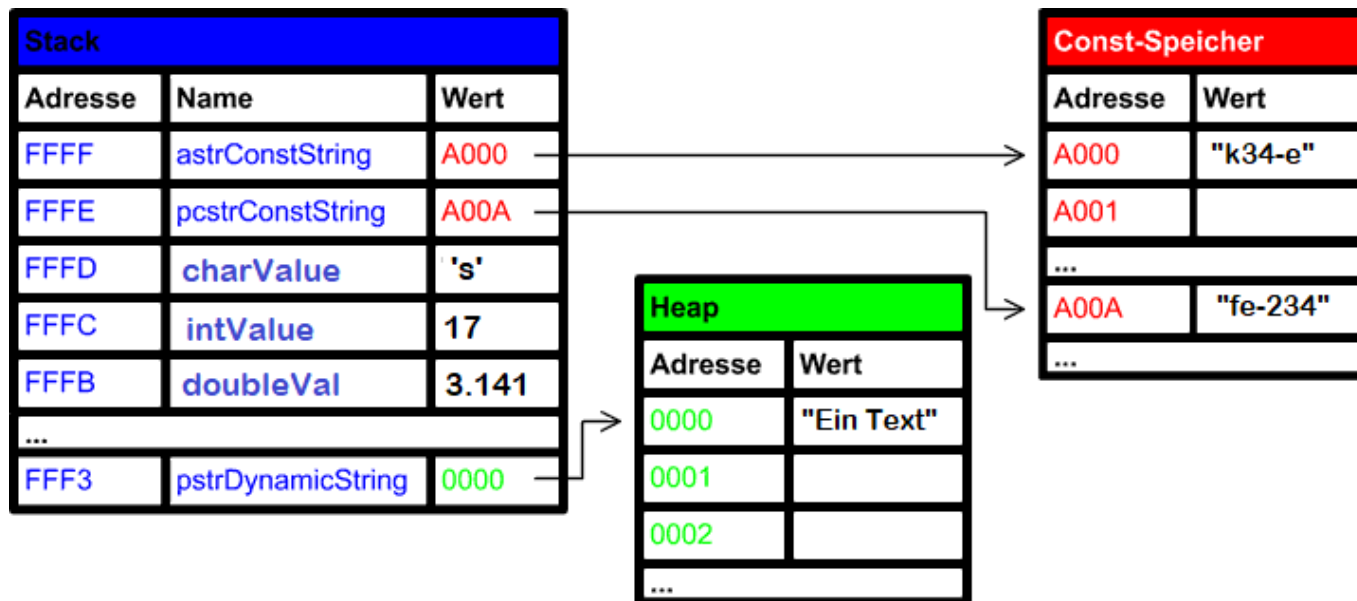
¹ x=1; y=1+x++ gibt die Werte x=2 und y=2 zurück

Bindungsstärke

Was ist die Ausgabe des folgenden Programms?

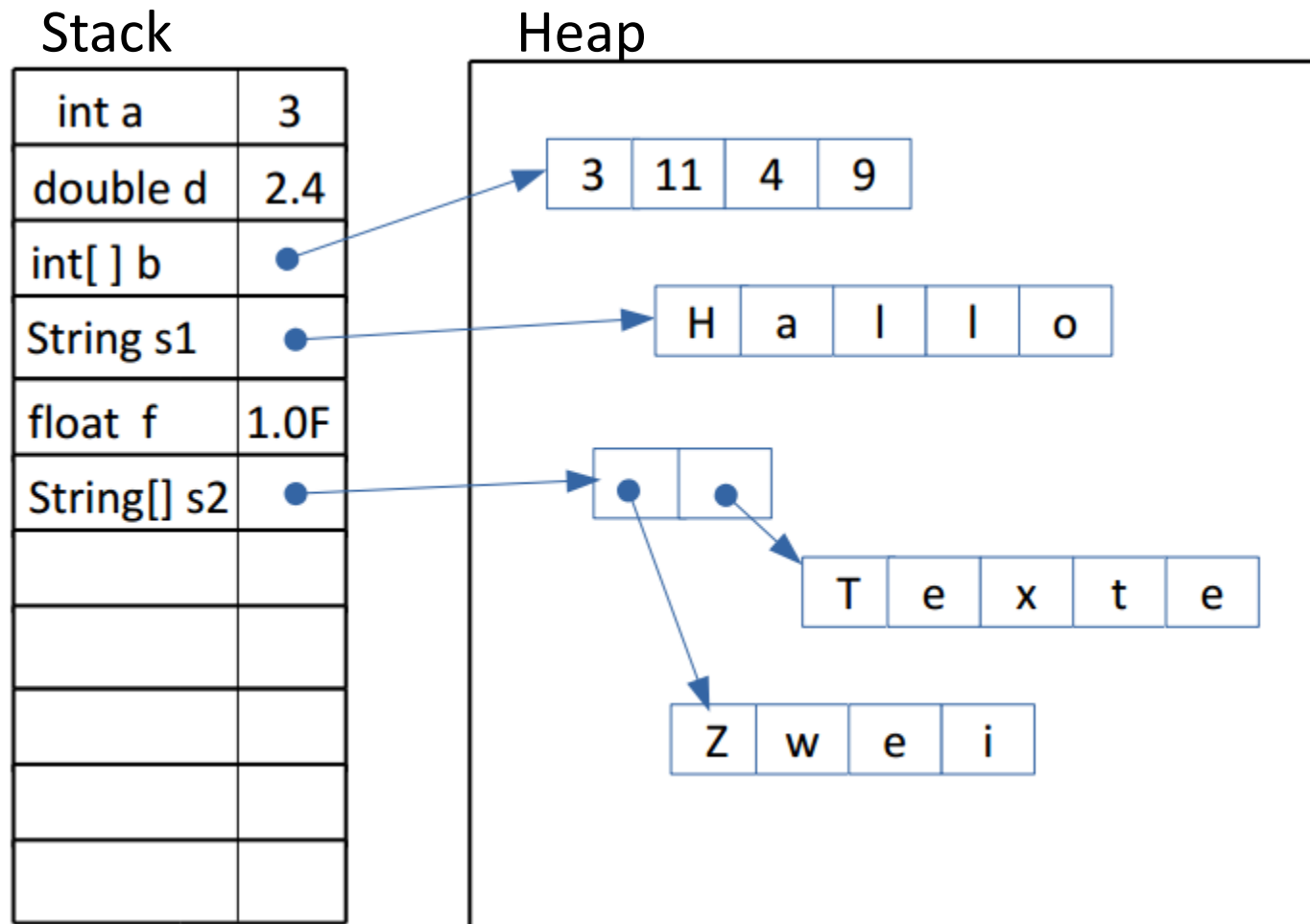
```
int a = 2; int b = 3; int c;  
System.out.printf("a=%d, b=%d, c = a + 2 * b = %d %n",  
    a, b, c = a + 2 * b);  
  
System.out.printf("a=%d, b=%d, c = ++a * b-- = %d %n",  
    a, b, c = ++a * b--);  
  
System.out.printf("a=%d, b=%d, c += -a = %d %n",  
    a, b, c += -a);  
  
System.out.printf("a=%d, b=%d, c *= --b = %d %n",  
    a, b, c *= --b);  
  
System.out.printf("a=%d, b=%d, c = c modulo ++a = %d %n",  
    a, b, c = c % ++a);
```

Die Speicherverwaltung



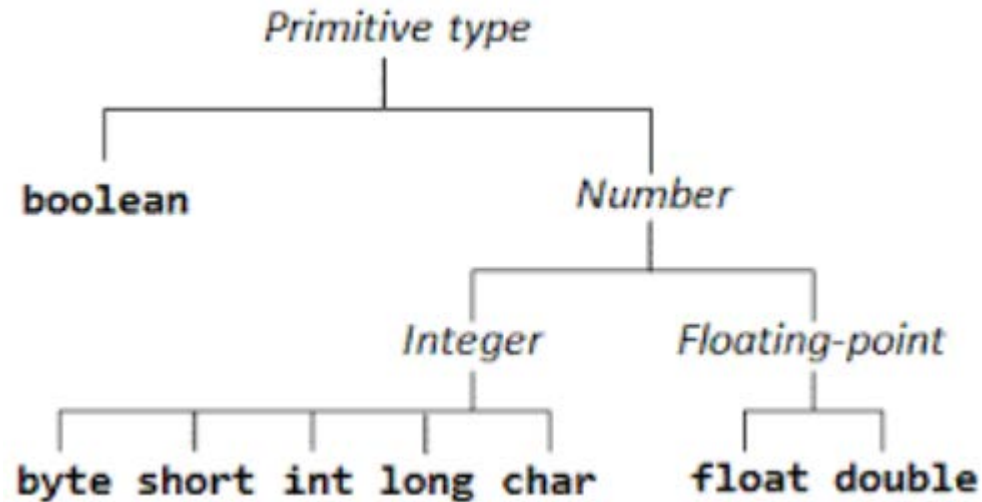
Die Speicherverwaltung: Stack und Heap

Alle Daten werden im Speicher gehalten. Dieser ist aufgeteilt in den Stack (Stapel) und den Heap (Halde).



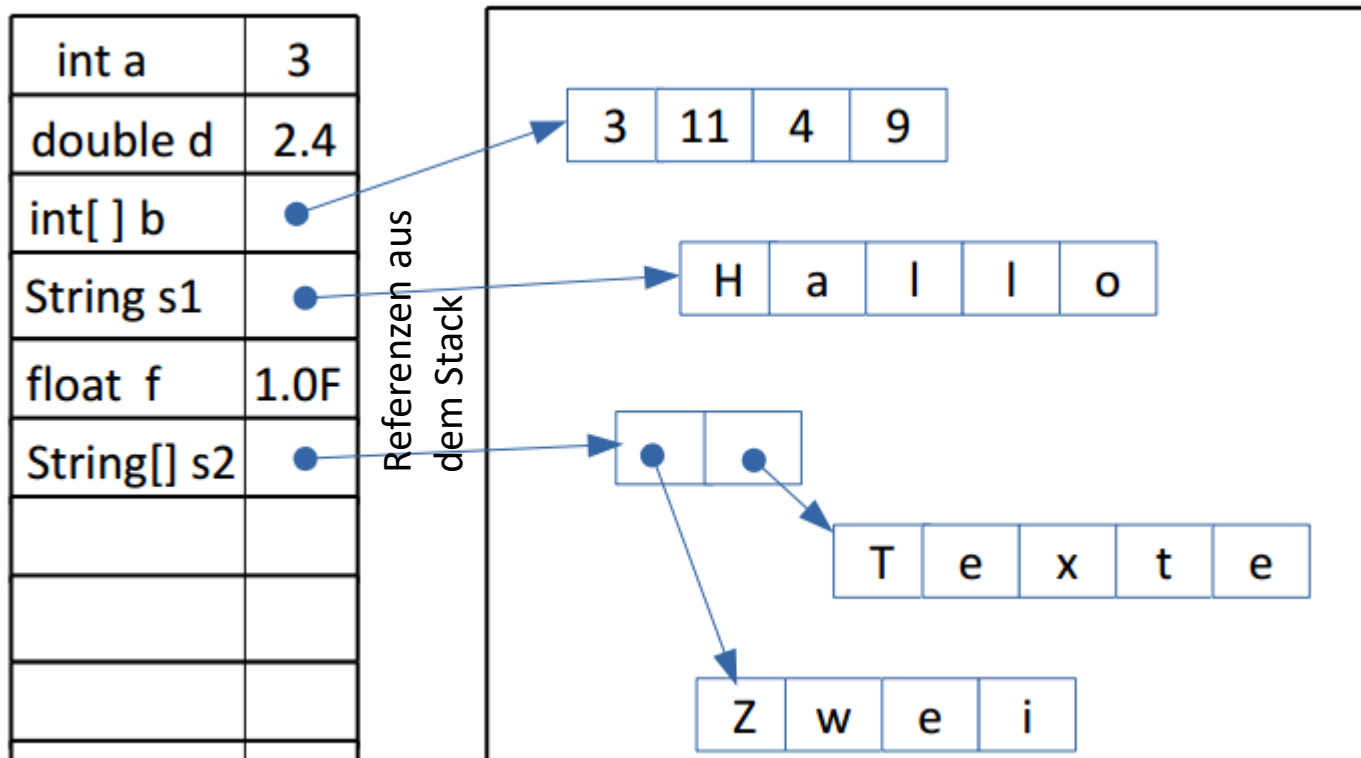
Die Speicherverwaltung: Stack

- Der Stack wird durch **Code Manager** verwaltet.
- Auf dem Stack werden
 - die Adressen der Funktionseintritte (Methodenaufrufe)
 - und alle Daten verwaltet, deren Typen eine feste Grösse aufweisen (Primitive Typen wie boolean, int, float, double, ...).



Die Speicherverwaltung: Heap

- Der Heap beinhaltet alle restlichen Daten (deren Typen **keine feste Grösse** aufweisen, wie z.B. Zeichenketten (Strings) oder Objekte (Instanzen von Klassen). Die Daten auf dem Heap werden aus dem Stack referenziert.



Werttypen \leftrightarrow Referenztypen

■ Werttypen

Werttypen (engl. value type) sind Typen, die im Typsystem eine feste Länge (Anzahl Bytes) aufweisen.

- Der Speicherort von Werttypen ist der Stack.

■ Referenztypen

Referenztypen (Verweistyp, Objekttyp, engl. reference type) sind Typen, die im Typsystem keine feste Größe aufweisen.

- Der Speicherort von Referenztypen ist der Heap.
- Für das Anlegen von Objekten muss die Laufzeitumgebung eine Speicheranforderung an die Heap-Verwaltung absetzen.

Zuweisung bei Wert- oder bei Referenztypen

```
public static void main(String[] args) {  
    System.out.println("Wert-Typ --> Integer");  
    int a = 5;  
    int b = a;  
    System.out.printf("a=%d, b=%d %n", a, b); // a und b sind gleich  
    a = 7; // Wert von a ändern  
    System.out.printf("a=%d, b=%d %n", a, b); // Wert von a und b sind verschieden  
    System.out.println("Referenz-Typ --> Array");  
    double[] s = { 12, 2.2 };  
    double[] t = s;  
    System.out.printf("s=%.1f, t=%.1f %n", s[0], t[0]);  
    s[0] = 17; // Wert des Arrays ändern  
    System.out.printf("s=%.1f, t=%.1f", s[0], t[0]); // s und t sind gleich  
}
```

Wert-Typ --> Integer

a=5, b=5

a=7, b=5

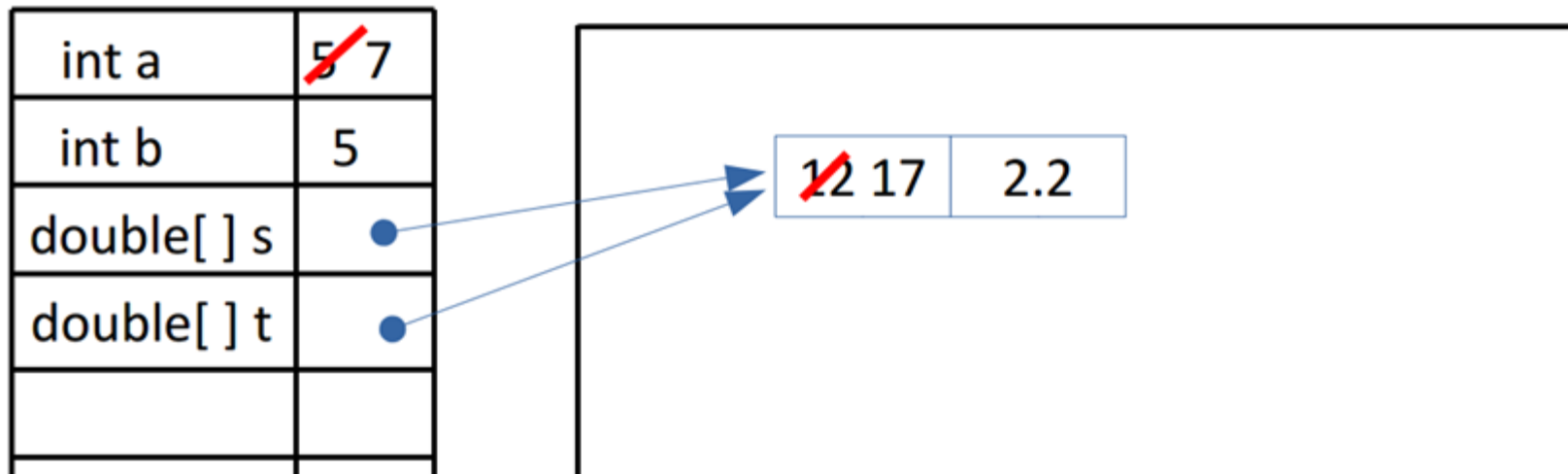
Referenz-Typ --> Array

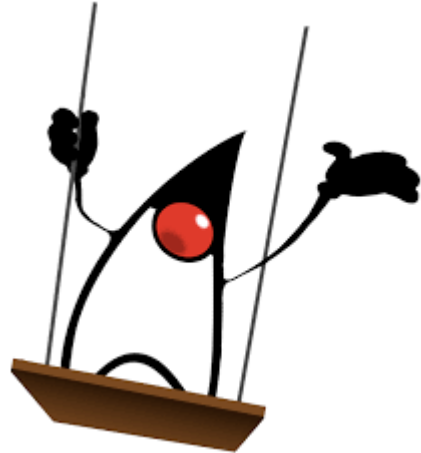
s=12.0, t=12.0

s=17.0, t=17.0

Das Speicherbild

- Die int Variable b ist eine echte Kopie. Änderungen auf dem Original berühren die Kopie b nicht.
- Die Kopie t zeigt auf das gleiche Feld wie s (ist keine «echte» Kopie).
- Beim Ändern von Werten, welche auf dem Heap liegen, werden auch die Inhalte der Kopien ebenfalls verändert.
- Im Code ist der Unterschied nicht einfach zu erkennen.





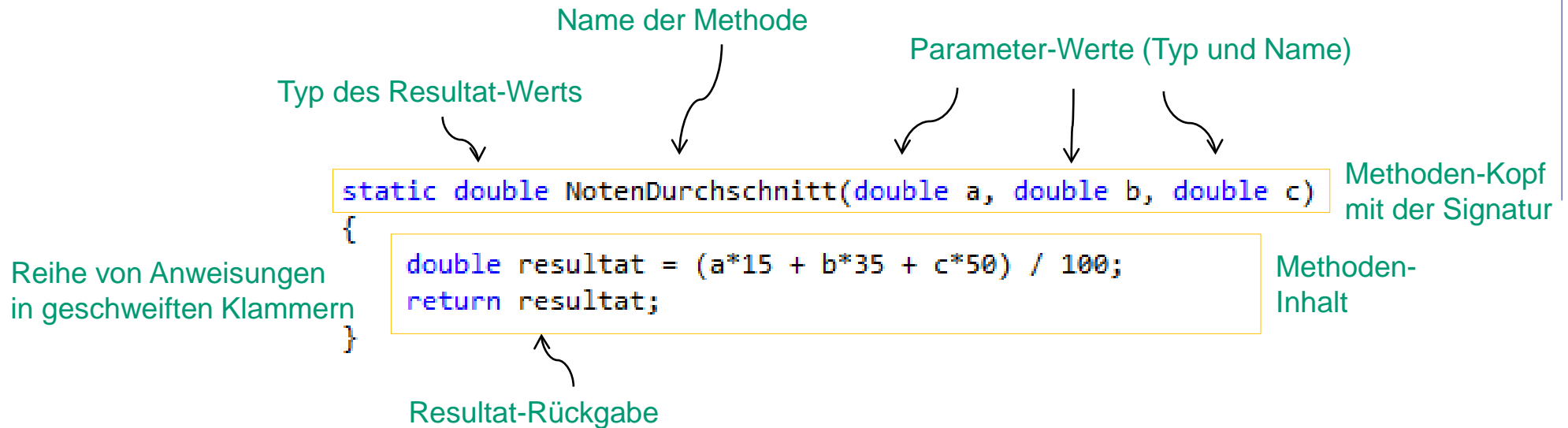
Statische Methoden

Eigene statische Methoden schreiben und aufrufen

Speicherbild

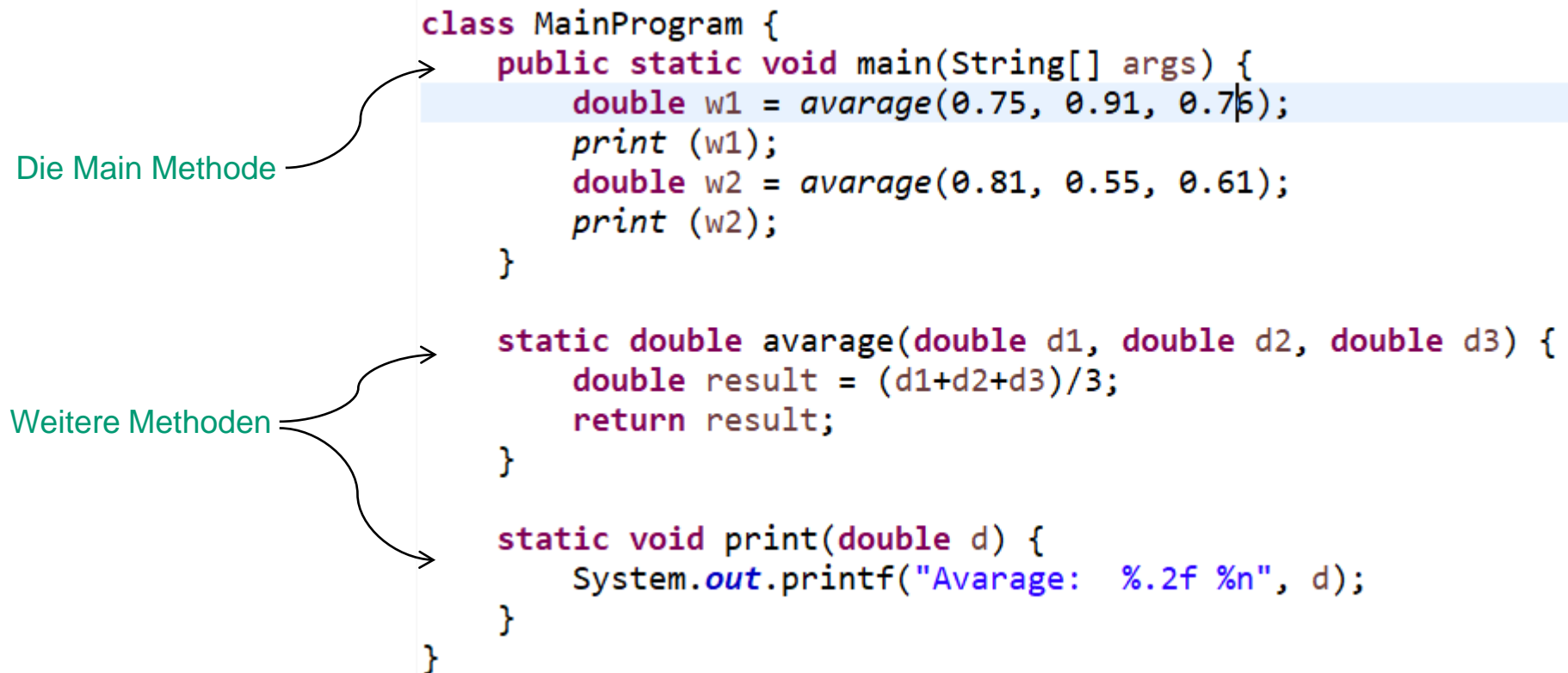
Definition

Eine Methode ist ein Codeblock, der eine Reihe von Anweisungen enthält.
Die Anweisungen einer Methode stehen in zwei geschweiften Klammern { und }:



Die main Methode

Die statische main-Methode ist der Einstiegspunkt jedes Java-Programms
Sie wird beim Starten des Programms von der Java Virtual Machine (JVM) automatisch aufgerufen.



```
class MainProgram {  
    public static void main(String[] args) {  
        double w1 = avarage(0.75, 0.91, 0.76);  
        print (w1);  
        double w2 = avarage(0.81, 0.55, 0.61);  
        print (w2);  
    }  
  
    static double avarage(double d1, double d2, double d3) {  
        double result = (d1+d2+d3)/3;  
        return result;  
    }  
  
    static void print(double d) {  
        System.out.printf("Avarage:  %.2f %n", d);  
    }  
}
```

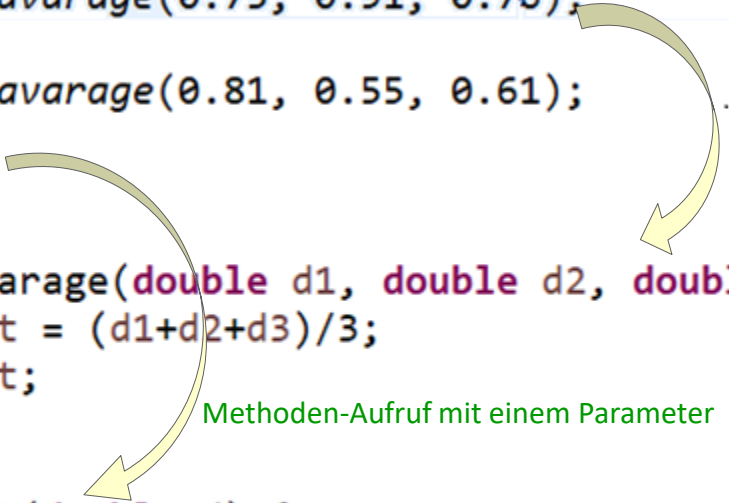
Die Main Methode

Weitere Methoden

Methoden-Aufruf

Die Anweisungen einer Methode werden ausgeführt, sobald das Programm die Methode aufruft und die erforderlichen Methoden-Argumente (als Parameter) übergibt

```
class MainProgram {  
    public static void main(String[] args) {  
        double w1 = avarage(0.75, 0.91, 0.76);  
        print (w1);  
        double w2 = avarage(0.81, 0.55, 0.61);  
        print (w2);  
    }  
  
    static double avarage(double d1, double d2, double d3) {  
        double result = (d1+d2+d3)/3;  
        return result;  
    }  
  
    static void print(double d) {  
        System.out.printf("Avarage:  %.2f %n", d);  
    }  
}
```



Methoden-Aufruf mit drei Parametern

Methoden-Aufruf mit einem Parameter

```
<terminated> mainPr  
Avarage:  0.81  
Avarage:  0.66
```

Argumente \leftrightarrow Parameter

Beim Aufrufen einer Methode müssen die richtige Anzahl Argumente mit dem richtigen Typ (*Erfüllen der Signatur*) angegeben werden, sonst gibt es einen Compiler-Fehler.

```
class MainProgram {  
    public static void main(String[] args) {  
        double w1 = avarage(0.75, 0.91, 0.76);  
        print (w1);  
        double w2 = avarage(0.81, 0.55, 0.61);  
        print (w2);  
    }  
  
    static double avarage(double d1, double d2, double d3) {  
        double result = (d1+d2+d3)/3;  
        return result;  
    }  
  
    static void print(double d) {  
        System.out.printf("Avarage: %.2f %n", d);  
    }  
}
```

Die Argumente werden beim Methoden-Aufruf in der angegebenen Reihenfolge als Parameter in die Methode eingesetzt.

Die Namen der Parameter dienen für die Definition der Methode.
Sie werden benutzt um zu erklären, was mit den übergebenen Werten/Argumenten berechnet werden soll.

Methoden mit verschiedenen Parameter-Typen

Beispiel: Signatur ist erfüllt → korrekter Methoden-Aufruf

```
class MainProgram1 {  
    public static void main(String[] args) {  
        setPersonData("Julia Roberts", 28, "October", 1967);  
    }  
  
    static void setPersonData(String name, int day, String month, int year) {  
        String birthDate = day + ". " + month + " " + year;  
        System.out.println(name + ": " + birthDate);  
    }  
}
```

Die Typen der Argumente müssen den Parameter-Typen (wie in der Methoden-Definition angegeben) entsprechen.

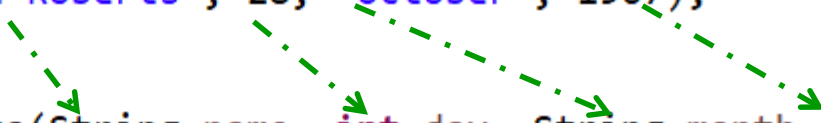
```
> terminated> mainProgram1.java Application  
Julia Roberts: 28. October 1967
```

Methoden mit verschiedenen Parameter-Typen

Beim Aufrufen einer Methode müssen die richtige Anzahl Argumente mit dem richtigen Typ angegeben werden, sonst gibt es einen Compiler-Fehler.

Die Reihenfolge der Aufruf-Argumente muss stimmen und den Parameter-Typen in der Definition der Methode entsprechen.

```
class MainProgram1 {  
    public static void main(String[] args) {  
        setPersonData("Julia Roberts", 28, "October", 1967);  
    }  
  
    static void setPersonData(String name, int day, String month, int year) {  
        String birthDate = day + ". " + month + " " + year;  
        System.out.println(name + ": " + birthDate);  
    }  
}
```

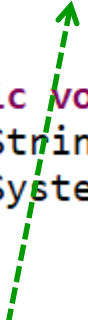


The diagram consists of four green dashed arrows pointing from the arguments in the `main` method call to the corresponding parameters in the `setPersonData` method definition. The arrows connect: 1. `"Julia Roberts"` to `String name`, 2. `28` to `int day`, 3. `"October"` to `String month`, and 4. `1967` to `int year`.

Methoden mit verschiedenen Parameter-Typen

Beispiel: Signatur ist nicht erfüllt

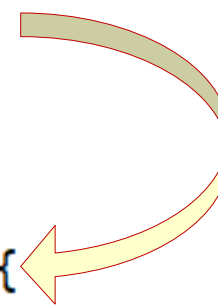
```
class MainProgram1 {  
    public static void main(String[] args) {  
        setPersonData("Julia Roberts", "October", 28, 1967);  
        setPersonData("Julia Roberts", 28, 10, 1967);  
        setPersonData("Julia", "Roberts", 28, "October", 1967);  
    }  
  
    static void setPersonData(String name, int day, String month, int year) {  
        String birthDate = day + "." + month + " " + year;  
        System.out.println(name + ": " + birthDate);  
    }  
}
```



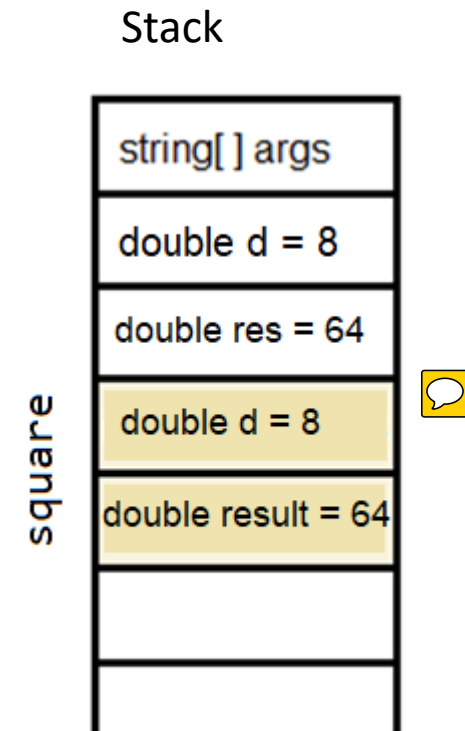
Falls die Parameterliste nicht mit der Definition übereinstimmt, zeigt eclipse einen Fehler an.

Speicherbild: Ein Wert-Parameter

```
class MainProgram1 {  
    public static void main(String[] args) {  
        double d=8;  
        double res = square(d);  
        System.out.println(res);  
    }  
  
    static double square(double d) {  
        double result = d*d;  
        return result;  
    }  
}
```



Methoden-Aufruf



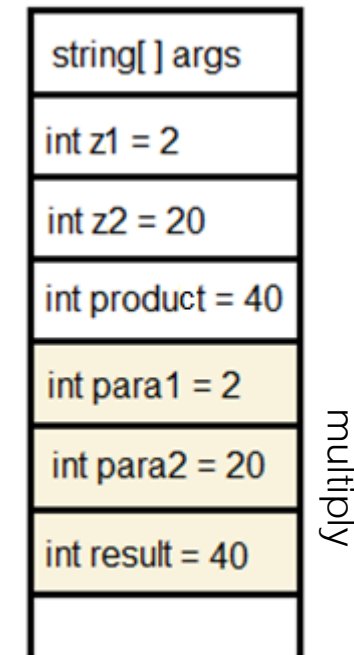
Beim Aufruf der Methode (square) wird der Wert in der Klammer (8) an die aufgerufene Methode als Parameter übergeben (kopiert).

Speicherbild: Zwei Wert-Parameter

```
class MainProgram1 {  
    public static void main(String[] args) {  
        double z1 = 2;  
        double z2 = 20;  
        double product = multiply(z1, z2);  
        System.out.println(product);  
    }  
  
    static double multiply(double para1, double para2) {  
        double result = para1 * para2;  
        return result;  
    }  
}
```

Methoden-Aufruf

Stack




Beim Aufruf der Methode (multiply) werden die Werte in der Klammer (2, 20) als Parameter übergeben (kopiert). Der Resultat-Wert der Methode wird in die Variable product abgespeichert. Danach wird der gelbe Bereich auf dem Stack gelöscht.

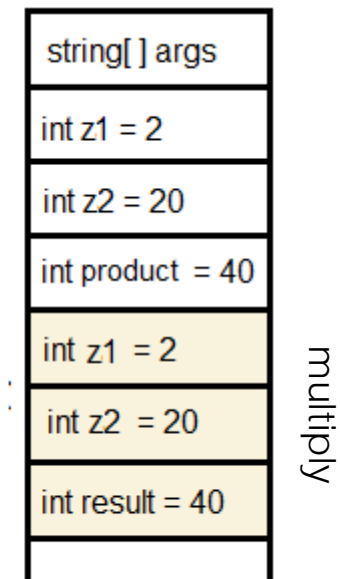
Speicherbild: Kontext

```
class MainProgram1 {  
    public static void main(String[] args) {  
        double z1 = 2;  
        double z2 = 20;  
        double product = multiply(z1, z2);  
        System.out.println(product);  
    }  
  
    static double multiply(double z1, double z2) {  
        double result = z1 * z2;  
        return result;  
    }  
}
```

Methoden-Aufruf



Stack



Die Werte werden auch kopiert wenn die Namen gleich sind. Die Parameter z1 und z2 sind für den Kontext von multiply neu.

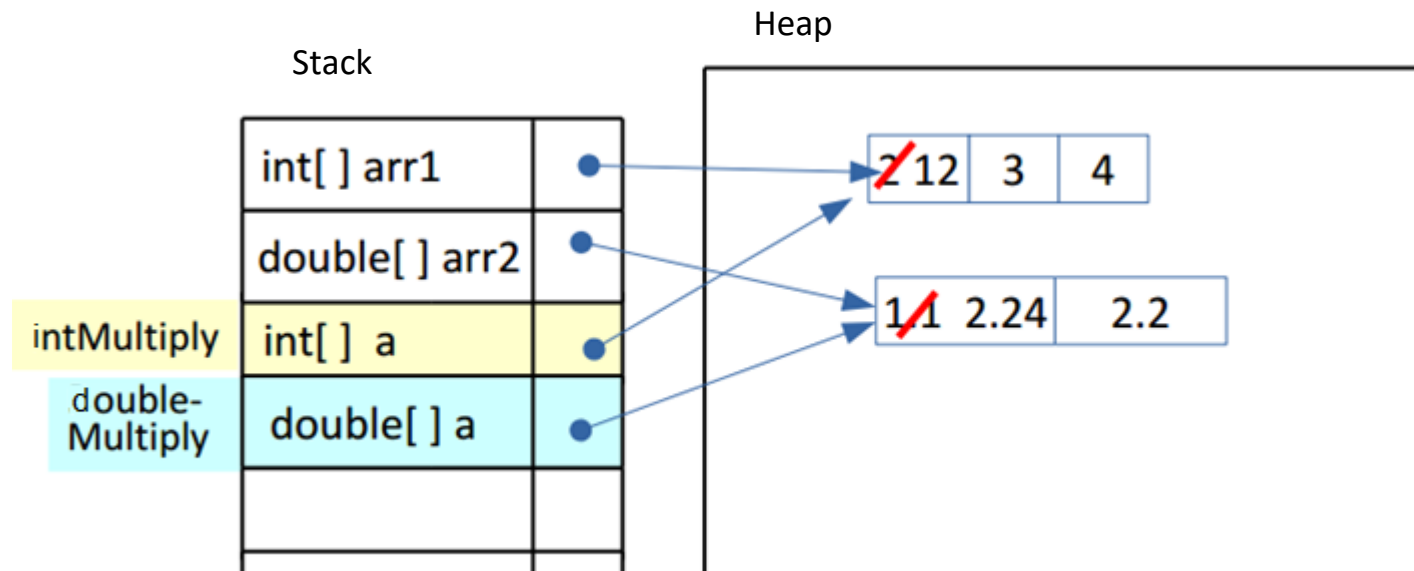
Methoden mit Referenz-Typen als Parameter

- Parameter, welche Referenz-Typen sind, werden nicht kopiert, sie erhalten bloss einen Verweis (Referenz) auf den Heap.

```
public static void main(String[] args) {  
    int[] arr1 = {2,3,4};  
    double[] arr2 = {1.1, 2.2};  
    intMultiply(arr1);  
    System.out.printf("%d , %.2f", arr1[0] , doubleMultiply(arr2));  
}  
  
static void intMultiply(int[] a) {  
    a[0] = a[1] * a[2];  
}  
  
static double doubleMultiply(double[] a) {  
    return a[0] * a[1];  
}
```

Referenz-Typen-Parameter

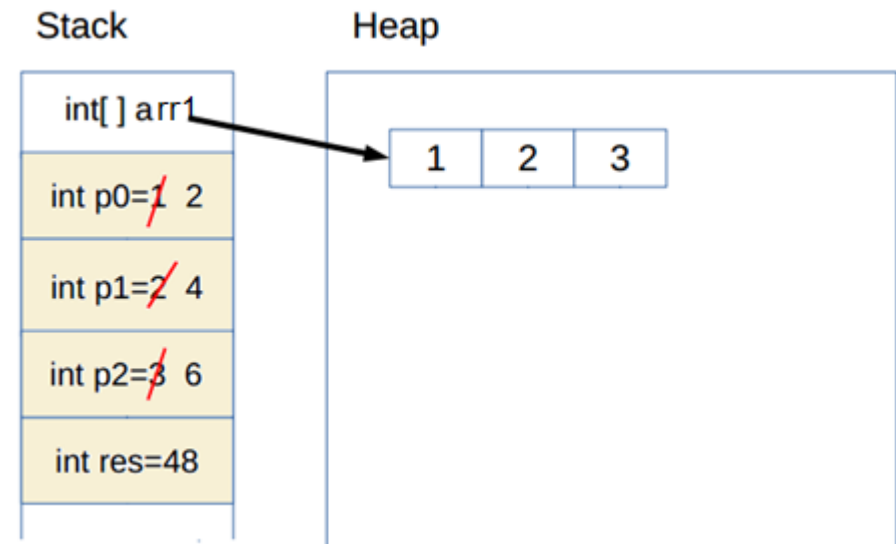
```
public static void main(String[] args) {  
    int[] arr1 = {2,3,4};  
    double[] arr2 = {1.1, 2.2};  
    intMultiply(arr1);  
    System.out.printf("%d , %.2f", arr1[0] , doubleMultiply(arr2));  
}  
  
static void intMultiply(int[] a) {  
    a[0] = a[1] * a[2];  
}  
  
static double doubleMultiply(double[] a) {  
    return a[0] * a[1];  
}
```



Speicherbild: Array «auspacken»

- Statt einer Referenz können auch die einzelnen Array-Werte übergeben werden.

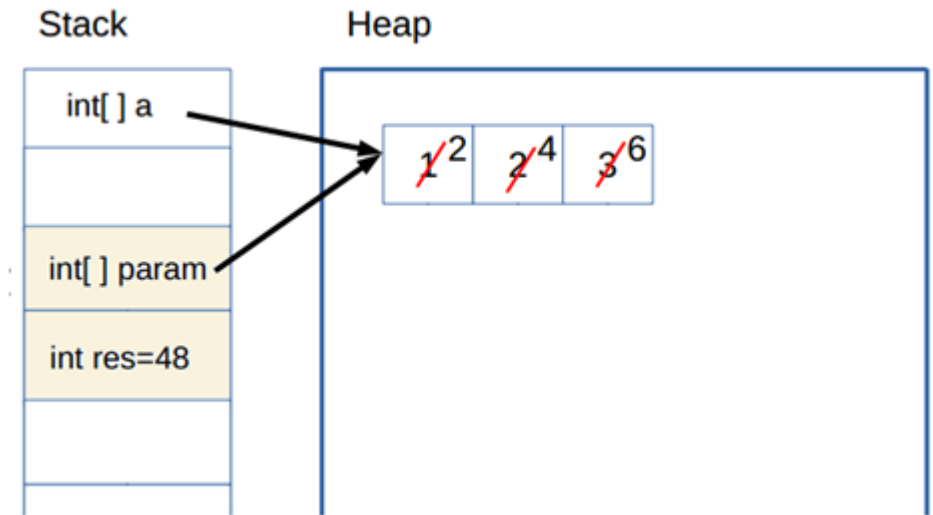
```
public static void main(String[] args) {  
    int[] arr1 = { 2, 3, 4 };  
    multiply(arr1[0], arr1[1], arr1[2]);  
}  
  
static int multiply(int p0, int p1, int p2) {  
    p0 = 2 * p0;  
    p1 = 2 * p1;  
    p2 = 2 * p2;  
    int res = p0 * p1 * p2;  
    return res;  
}
```



Beim Aufruf der Methode (`multiply`) werden die einzelnen Array Werte in der Klammer (1, 2 und 3) in den Bereich der aufgerufenen Methode kopiert.

Speicherbild: Array als Parameter übergeben

```
public static void main(String[] args) {  
    int[] arr1 = { 2, 3, 4 };  
    multiply(arr1);  
}  
  
static int multiply(int[] param) {  
    param[0] = 2 * param[0];  
    param[1] = 2 * param[1];  
    param[2] = 2 * param[2];  
    int res = param[0] * param[1] * param[2];  
    return res;  
}
```



Beim Aufruf der Methode (`multiply`) wird der Wert in der Klammer (die Adresse, bzw. die Referenz von `arr1`) an die aufgerufene Methode als Parameter übergeben.