hhu.

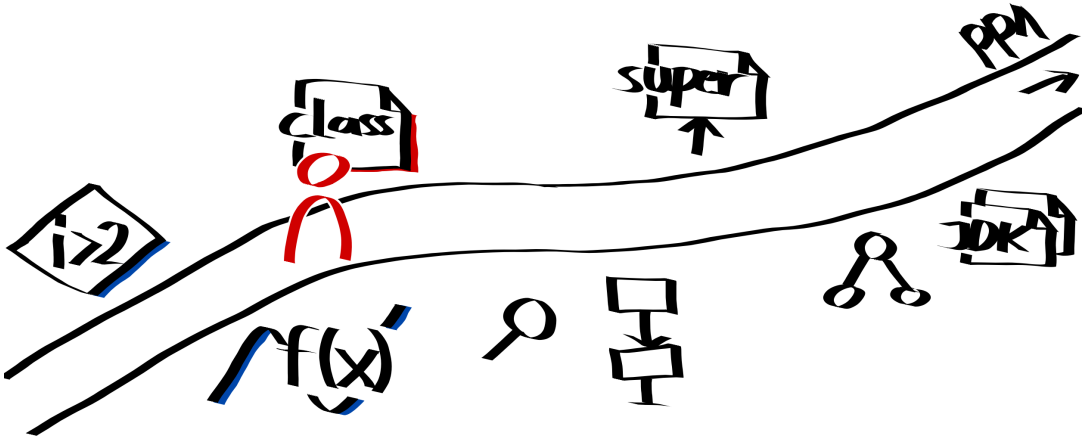
Kapitel 3: Objekte, Speicher & Klassen

VL 9: Objekte & Strings

Wintersemester 2023/24

Stand 8. November 2023, 07:39 Uhr

Wo stehen wir gerade?



Vorlesungsfeedback

- ø Zeit: 15 Stunden ($n = 1$)
- \+ Quiz
- – Mittippen wird schwieriger

Codierung von Zeichen

Heinrich Heine
Universität Düsseldorf

Frage

Wie speichert ein Computer Buchstaben, wenn er doch nur mit Zahlen umgehen kann?

Codierung von Zeichen

Heinrich Heine
Universität Düsseldorf

Frage

Wie speichert ein Computer Buchstaben, wenn er doch nur mit Zahlen umgehen kann?

⇒ Einigung auf eindeutige Zuordnung von Buchstaben zu Zahlen

- Datentyp bestimmt, ob Zahlenwert als Zeichen oder Zahl zu verstehen

ASCII

Heinrich Heine Universität Düsseldorf

- American Standard Code for Information Interchange (60er)

0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	‘	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Unicode

- beinhaltet mehr als 130.000 Zeichen
- erste 127 Zeichen wie in ASCII
- Beispiele:
  - 65: A *(wie in ASCII)*
  - 228: ä
  - 945: α
  - 8469: $\mathbb{N}$
  - 8704: $\forall$
- Speicherung in Java als 16-Bit-Zahl(en) (UTF-16)

Miscellaneous mathematical symbols

2200 ∀ FOR ALL
= universal quantifier

2201 ∁ COMPLEMENT
→ 0297 ɔ latin letter stretched c

2202 ∂ PARTIAL DIFFERENTIAL

2203 ∃ THERE EXISTS
= existential quantifier

2204 ∄ THERE DOES NOT EXIST
≡ 2203 ∃ 0338 ̸

2205 ∅ EMPTY SET
= null set
• used in linguistics to indicate a null morpheme or phonological "zero"
→ 00D8 Ø latin capital letter o with stroke
→ 2300 ⌀ diameter sign
~ 2205 FE00 ∅ zero with long diagonal stroke overlay form

2206 Δ INCREMENT
= Laplace operator

Quelle: http://www.unicode.org/charts/PDF/U2200.pdf

Character

- Bezeichnung in Java: `char`
- Werte: alle Unicode-Zeichen bis Nummer U+FFFF, z. B. `a`, `3`, `ø`
- Operationen: `+`, `-`, `*`, `/`, `%` (Rechnung mit Unicode-Nummer, ergibt Integer)
- Beispiele für Ausdrücke:

Ausdruck	Wert	Bemerkung
`'a'`	a	Zeichen a (≠ String a)
`'1'`	1	Zeichen 1 (≠ Integer 1)
`'\n'`	*Zeilenumbruch*	
`'\''`	'	Escaping
`(int)'A'`	65	65 ist Unicode-Nummer von A
`'A' + 3`	68	65 + 3
`(char)('A' + 3)`	D	68 ist Unicode-Nummer von D
`'D' > 'A'`	true	Unicode-Nummern von D größer als die von A
`'Z' - 'A'`	25	Abstand der Unicode-Nummern von Z und A

Strings[1]

- Bezeichnung in Java: `String` (Objekt-Typ)
- Werte: alle Zusammensetzung von Zeichen
- Operationen: `+` (String-Konkatenation/Verkettung)
- Beispiele für Ausdrücke:

Ausdruck	Wert	Bemerkung
`"Hallo"`	Hallo	
`"H"`	H	String mit nur einem Zeichen
`""`		leerer String
`"\""`	"	Escaping
`"Hal" + "lo"`	Hallo	String + String = String
`"Zahl" + 1`	Zahl1	String + Integer = String
`"Buchstabe" + 'A'`	BuchstabeA	String + Char = String

[1] Zeichenketten

Wichtige Methoden auf Strings I

- `int length()`: Anzahl der Character im String
  `"Hallo".length()` ⇒ 5
- `String toUpperCase()`: Umwandeln in Großbuchstaben (Original bleibt unverändert)
  `"Hallo".toUpperCase()` ⇒ HALLO
- `char charAt(int)`: Gibt Character an Position zurück
  `"Hallo".charAt(1)` ⇒ a
- `boolean equals(String)`: Prüft Strings auf inhaltliche Gleichheit
  `"Hallo".equals("Hello")` ⇒ false

Wichtige Methoden auf Strings II

- `int compareTo(String)`: Bestimmt alphabetische Reihenfolge der Strings
  `"Hallo".compareTo("Hello")` ⇒ Wert kleiner 0
  `"Hallo".compareTo("Hallo")` ⇒ 0
  `"Hello".compareTo("Hallo")` ⇒ Wert größer 0
- `String[] split(String)`: Auftrennen an bestimmten Zeichen:
  `"Hallo - du - da".split(" - ")` ⇒ { "Hallo", "du", "da"}
- `char[] toCharArray()`: Umwandeln in Character-Array:
  `"Hi".toCharArray()` ⇒ { 'H', 'i'}
- mehr unter `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html`

Beispiel: Wörter zählen I

Heinrich Heine Universität Düsseldorf

WordCounter.java — Java

```java
import java.util.Scanner;

public class WordCounter {
    public static void main(String[] args) {
        Scanner stdin = new Scanner(System.in);
        int wordsSoFar = 0;

        while(stdin.hasNext()) {
            String line = stdin.nextLine();
            String[] words = line.split(" ");
            int wordsInLine = words.length;
            wordsSoFar += wordsInLine;
        }

        System.out.println(wordsSoFar + " Wörter gezählt.");
    }
}
```

Beispiel: Wörter zählen II

Heinrich Heine
Universität Düsseldorf

```
% cat text.txt
Lorem ipsum dolor sit amet, consectetur
adipisici elit
% java WordCounter < text.txt
8 Wörter gezählt.
```

Erzeugung von Strings

- per String-Literal
- per Konstruktor-Aufruf mit String als Parameter
- per Konstruktor-Aufruf mit char-Array als Parameter

Strings.java — Java

```java
public class Strings {
    public static void main(String[] args) {
        String a = "Hello"; // neuer String mit String-Literal

        // unüblicher, aber manchmal sinnvoll:
        String b = new String("Hello"); // neuer String durch Konstruktor-Aufruf

        char[] chars = {'H', 'e', 'l', 'l', 'o'};
        String c = new String(chars); // neuer String aus Character-Array
```

Was wird hier ausgegeben?

Strings.java (Fortsetzung) Java

```
11          System.out.println(a.charAt(0)); // erster Buchstabe des Strings "Hello"
12          System.out.println((char)(a.charAt(0) + 1));
```

Was wird hier ausgegeben?

Heinrich Heine Universität Düsseldorf

Strings.java (Fortsetzung) Java

```
11          System.out.println(a.charAt(0)); // erster Buchstabe des Strings "Hello"
12          System.out.println((char)(a.charAt(0) + 1));
```

```
% java Strings
H
I
```

! klappt intuitiv nur gut bei ASCII-Zeichen, aber z. B. nicht mit Umlauten

Was wird hier ausgegeben?

Heinrich Heine Universität Düsseldorf

Strings.java (Fortsetzung) Java

```
14          System.out.println(a == b); // nicht dasselbe String-Objekt
15          System.out.println(a.equals(b)); // aber gleicher Inhalt
```

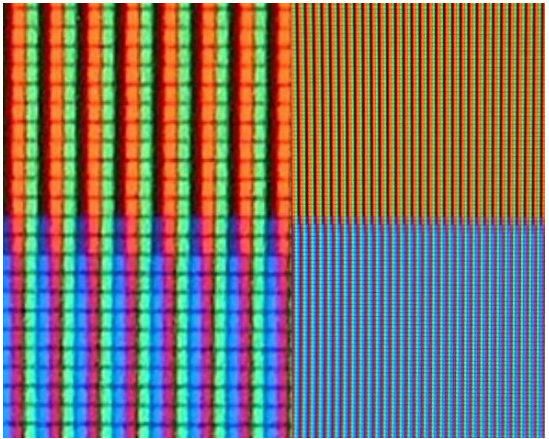
Was wird hier ausgegeben?

Strings.java (Fortsetzung) Java

```
14          System.out.println(a == b); // nicht dasselbe String-Objekt
15          System.out.println(a.equals(b)); // aber gleicher Inhalt
```

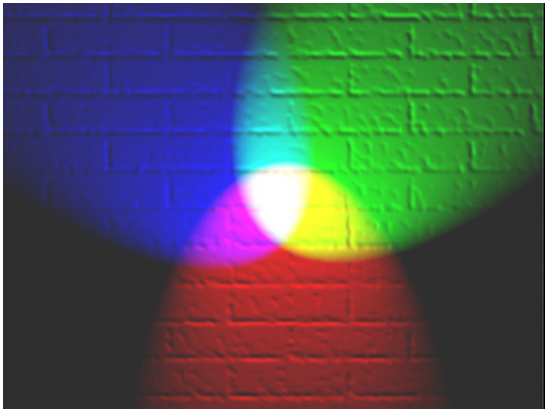
```
false
true
```

Farbdarstellung am Bildschirm

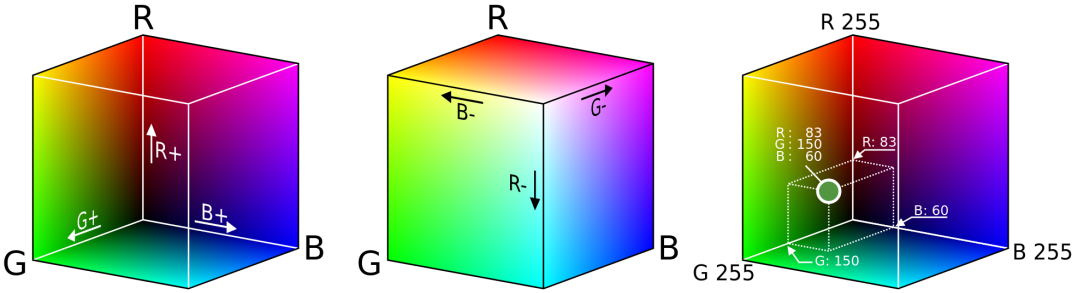


Stan Zurek (https://commons.wikimedia.org/wiki/File:RGB_pixels.jpg), „RGB pixels“, CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/)

Additive Farbmischung



Repräsentation von Farben



User:Maklaan, Horst Frank commonswiki (https://commons.wikimedia.org/wiki/File:RGB_color_cube.svg), „RGB color cube“, CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/)

Datentypen Color & Picture

- Datentyp `Color`
  - Teil der JVM
  - erfordert `import java.awt.Color;`
  - neue Farbe erstellen: `new Color(red, green, blue)`
  - Farbwerte einer Color-Variablen auslesen: `int colorVariable.getRed()`

- Datentyp `Picture`
  - von Robert Sedgewick und Kevin Wayne
  - erfordert `Picture.java` (siehe Ilias)
  - Bild aus Datei laden: `new Picture("dateipfad.jpg")`
  - Bild anzeigen: `void myPictureVariable.show()`
  - Breite eines Bildes in Pixel: `int myPictureVariable.width()`
  - Farbe eines Pixels auslesen: `Color myPictureVariable.get(x, y)`
  - Farbe eines Pixels setzen: `void myPictureVariable.set(x, y, colorVariable)`

Invertieren & Luminanz

- r, g, b: alte Werte für ein Pixel
- Farbe invertieren:
  $r' = 255 - r$
  $g' = 255 - g$
  $b' = 255 - b$
- Farbe zu Helligkeitswert (Luminanz):
  $0{,}299 \cdot r + 0{,}587 \cdot g + 0{,}114 \cdot b$

0, 106, 179
Original

255, 149, 76
Invertiert

82, 82, 82
Entfärbt

Primitive Datentypen

- in Java eingebaut
- vorab definierter Speicherbedarf
- nicht aus kleineren Datentypen zusammengesetzt ($\approx$ Atome)

Primitive Datentypen in Java (vollständige Auflistung):

- ganze Zahlen: `byte`, `short`, `int`, `long`
- gebrochene Zahlen: `float`, `double`
- Wahrheitswerte: `boolean`
- Zeichen: `char`

Objekte I

- aus primitiven Datentypen oder anderen Objekten zusammengesetzt (≈ Moleküle)
  - Integer-Arrays
  - Strings
  - Scanner-Objekte
- *auf* Objekten können Methoden aufgerufen werden (Objekte haben Verhalten)
  - `"Hi".toCharArray()`
  - `stdin.hasNext()`
- Objekte haben auch Datentyp
  - Beispiele: `Scanner`, `String`
  - Objekttypen per Konvention großgeschrieben

Objekte II

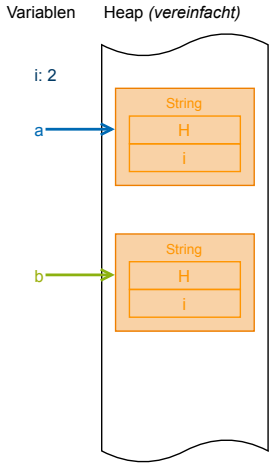
- Variablen mit Objekten enthalten „nur“ Verweis auf Speicherstelle im Heap[2]
  - Heap: Bereich im Arbeitsspeicher, wo Objekte gespeichert werden
  - Variable selbst enthält nur Referenz ($\approx$ Adresse) in den Heap
  - spannende Implikation für Parameter von Methoden ($\rightarrow$ nächste VL)
- Weitere Art der Abstraktion: Mir ist egal, wie Strings intern funktionieren, mich interessiert nur, was Strings können.
- Eigene Objekttypen können mit einer Klasse definiert werden ($\rightarrow$ nächste Woche)

[2] Halde

new

- `new` erstellt neues Objekt (oder *Instanz*)
- reserviert Speicherplatz im Heap für die Eigenschaften des Objekts
- nach `new` steht Aufruf des Konstruktors
  - Konstruktor: spezielle Methode, die Objekteigenschaften setzt und Referenz (≈ Speicheradresse/Pfeil) auf das Objekt liefert

```
1 public static void main(String[] args) {
2     int i = 2;
3     String a = new String("Hi");
4     String b = new String("Hi");
5 }
```



equals vs. ==

- Werte primitiver Datentypen direkt in Variable gespeichert
- Inhalt von Objekten in Heap gespeichert
  - Variable selbst beinhaltet nur Referenz

- `==` prüft nur Inhalt der Variablen, nicht Inhalt von Objekten
- Objekte definieren (meistens) `equals` für einen Inhalts-Vergleich

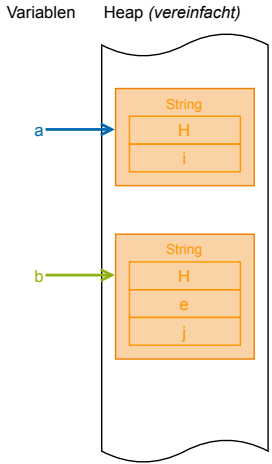
```java
public static void main(String[] args) {
    String a = new String("Hi");
    String b = new String("Hi");

    // Zeigen die "Pfeile" auf dieselbe Stelle?
    System.out.println(a == b); // false
    // Ist der Inhalt im Heap gleich?
    System.out.println(a.equals(b)); // true
}
```

Semantik von =

- kopiert, was in der Variablen steht:
  - primitive Datentypen: tatsächlicher Wert
  - Objekt-Typen: Referenz

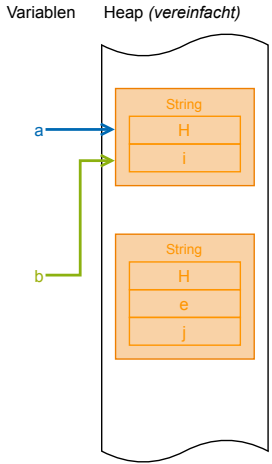
```
1 String a = new String("Hi");
2 String b = new String("Hej");
3 b = a;
4 System.out.println(a == b);
5 System.out.println(a.equals(b));
```



Semantik von =

- kopiert, was in der Variablen steht:
  - primitive Datentypen: tatsächlicher Wert
  - Objekt-Typen: Referenz

```
1 String a = new String("Hi");
2 String b = new String("Hej");
3 b = a;
4 System.out.println(a == b);
5 System.out.println(a.equals(b));
```



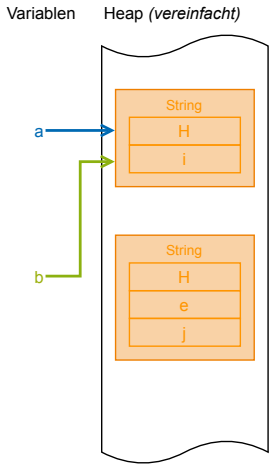
Semantik von =

- kopiert, was in der Variablen steht:
  - primitive Datentypen: tatsächlicher Wert
  - Objekt-Typen: Referenz

```java
String a = new String("Hi");
String b = new String("Hej");
b = a;
System.out.println(a == b);
System.out.println(a.equals(b));
```

```
% java Assignment
true
true
```

! beide Variablen zeigen auf denselben Heap-Inhalt



Besonderheit bei String-Konstanten

Heinrich Heine Universität Düsseldorf

- Compiler nutzt dasselbe String-Objekt, wenn zur Compilezeit klar, dass Strings gleich
- erlaubt, weil String-Objekte einmal angelegt unveränderlich[3] sind

```
1 String a = "Hi";
2 String b = "Hi";
3 System.out.println(a == b);
4 System.out.println(a.equals(b));
```

```
% java StringEquals
true
true
```

[3] immutable

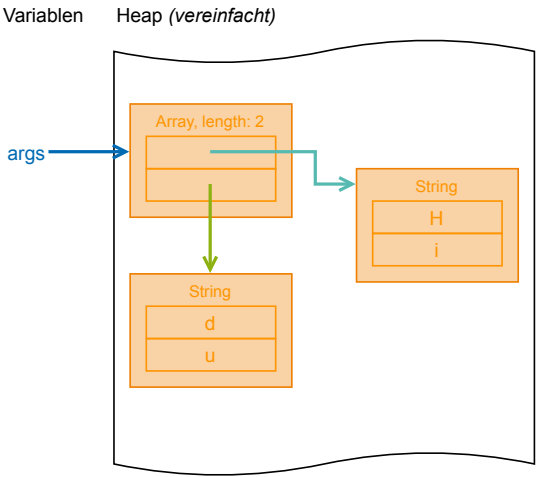
Können Objekte Objekte beinhalten?

Heinrich Heine
Universität Düsseldorf

Können Objekte Objekte beinhalten?

Beispiel: args-Array

```
% java Beispiel Hi du
```



Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche ...

- den Datentyp `char` **verwenden**.
- Methoden des Datentyps `String` **verwenden**.
- Strings in Character und umgekehrt **umwandeln**.
- **erklären**, was Objekte sind.
- **erklären**, wo Objekte gespeichert sind.
- `==` und `equals` richtig **verwenden**.

Wichtige Begriffe

ASCII	char	String
charAt	primitiver Datentyp	Objekt
Referenz	Instanz	Heap
equals		