Einführung in die Informatik

Objekte

Referenzen, Methoden, Klassen, Variablen, Objekte

Wolfram Burgard
Cyrill Stachniss

Verwendung von `PrintStream`-Objekten

Wenn wir die Nachricht

```
println("something to display")
```

an das durch `System.out` referenzierte `PrintStream`-Objekt senden, führt das zur Ausgabe des entsprechenden Textes auf dem Bildschirm.

Jedes weitere Zeichen danach wird am Beginn der nächsten Zeile ausgegeben.

Die Nachrichten `print` und `println()`

Alternative:

```
print("something to display")
```

Bei Verwendung von `print` erscheint das nächste gedruckte Zeichen in der gleichen Zeile:

```
System.out.print("JA");

System.out.print("VA");
```

ergibt die Ausgabe

```
JAVA
```

Eine Variante der `println`-Nachricht ist `println()`. Die Nachricht

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

bewirkt, dass nachfolgende Ausgaben in der nächsten Zeile beginnen.

Referenzen

- Eine Referenz in Java ist jede Phrase, die sich auf ein Objekt bezieht.
- Referenzen werden verwendet, um dem entsprechenden Objekt eine Nachricht zu schicken.
- Streng genommen ist `System.out` kein Objekt sondern nur eine Referenz.

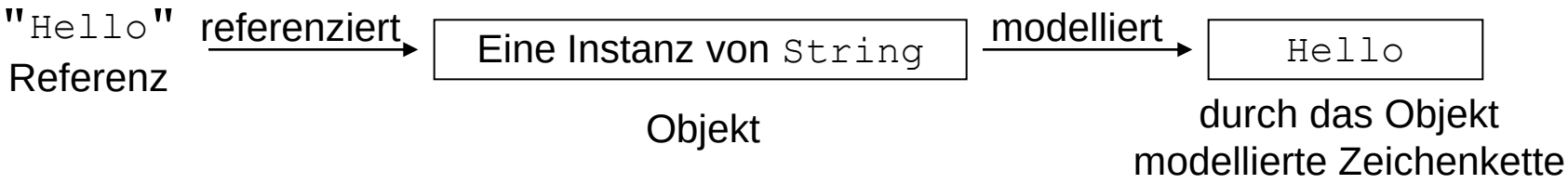
Bezeichnung: Das `System.out`-Objekt

Ausführen von Nachrichten

- Java-Statements werden in der Reihenfolge ausgeführt, in der sie im Programm stehen.
- Wenn eine **Message** an ein Objekt (den Empfänger) geschickt wird, wird der Code des Senders unterbrochen, bis der Empfänger die Nachricht erhalten hat.
- Der Empfänger führt die durch die Nachricht spezifizierte Methode aus. Dies nennen wir „*Aufrufen einer Methode*“.
- Wenn der Empfänger die Ausführung seines Codes beendet hat, *kehrt die Ausführung zum Code des Senders zurück*.

Die `String`-Klasse

- `String` ist eine vordefinierte Klasse.
- Sie modelliert Folgen von Zeichen (Characters).
- Zu den zulässigen Zeichen gehören Buchstaben, Ziffern, Interpunktionssymbole, Leerzeichen und andere, spezielle Symbole.
- In Java werden alle Folgen von Zeichen, die in Hochkommata eingeschlossen sind, als Referenzen auf Zeichenketten interpretiert.



Die `String`-Methode `toUpperCase`

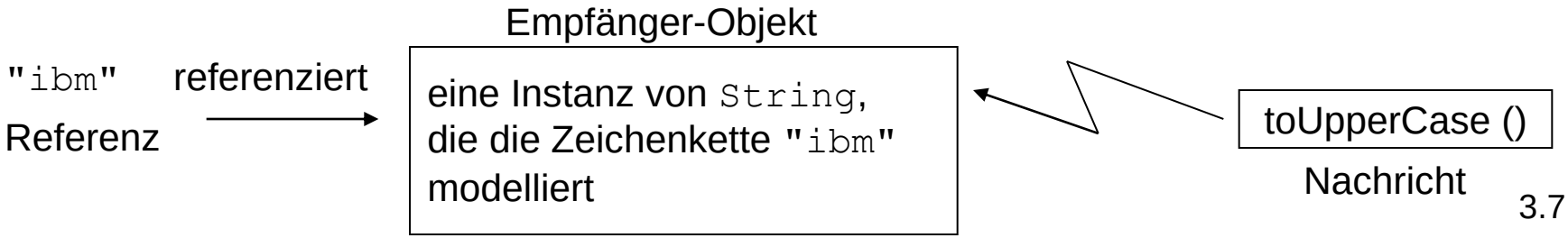
Eine Methode, die von der `String`-Klasse zur Verfügung gestellt wird, ist `toUpperCase`, welche keine Argumente hat.

Um eine Nachricht an die `String`-Klasse zu senden, verwenden wir die übliche Notation:

Referenz .Methodenname (Argumente)

Anwendungsbeispiel: `"ibm".toUpperCase()`

Der Empfänger der `toUpperCase`-Nachricht ist das `String`-Objekt, welches durch `"ibm"` referenziert wird.



Effekte von `String`-Methoden

- Eine Methode der Klasse String ändert nie den Wert des Empfängers.
- Stattdessen liefert sie als Ergebnis eine Referenz auf ein neues Objekt, an welchem die entsprechenden Änderungen vorgenommen wurden.

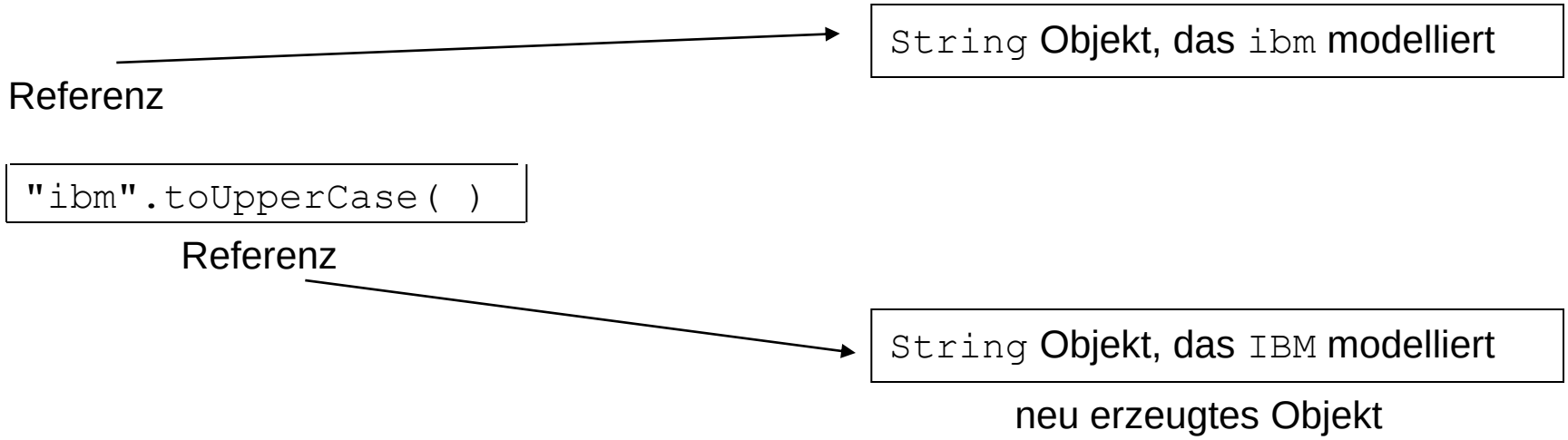
Beispiel:

- `"ibm".toUpperCase()` sendet nicht nur die `toUpperCase`-Nachricht an das `"ibm"`–Objekt.
- Der Ausdruck liefert auch eine Referenz auf ein neues `"IBM"`-Objekt.
- Wir sagen: „Der Return-Wert von `"ibm".toUpperCase()` ist eine Referenz."

Beispiel

Da die `println`-Methode der Klasse `PrintStream` eine Referenz auf ein `String`-Objekt verlangt, können wir schreiben:

```
System.out.println("ibm".toUpperCase());
```



Methoden, Argumente und Return-Werte

Klasse	Methode	Return-Wert	Argumente
PrintStream	println	kein	kein
PrintStream	println	kein	Referenz auf ein `String`-Objekt
PrintStream	print	kein	Referenz auf ein `String`-Objekt
String	toUpperCase	`String`-Objekt	kein

Signatur einer Methode: Bezeichnung der Methode plus Beschreibung seiner Argumente

Prototyp einer Methode: Signatur + Beschreibung des Return-Wertes

Methoden ohne Return-Wert

- Viele Methoden liefern eine Referenz auf ein Objekt zurück.
- Das gilt insbesondere für Methoden, die ein Ergebnis liefern, wie z.B. die `String`-Methode `toUpperCase()`, die eine Referenz auf ein `String`-Objekt liefert.
- Aber welchen Typ sollen Methoden wie `println()` haben, die kein Ergebnis liefern?
- Wir sagen: „Methoden ohne Ergebnis haben den Typ `void`."

Referenz-Variablen

- Eine **Variable** ist ein Bezeichner, dem ein Wert zugewiesen werden kann, wie z.B. sei `x=5`.
- Sie wird Variable genannt, weil sie zu verschiedenen Zeitpunkten verschiedene Werte annehmen kann.
- Eine **Referenz-Variable** ist eine Variable, deren Wert eine **Referenz** ist.
- Angenommen **`line`** ist eine **`String`-Referenz-Variable** auf ein `String`–Objekt welches folgende Zeichenkette repräsentiert:
  `xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
- Folgende Anweisungen geben zwei Zeilen von x-en aus:
  ```
  System.out.println(line);
  System.out.println(line);
  ```
- Möglich ist auch:
  ```
  System.out.println(line.toUpperCase());
  ```

Deklaration von Referenz-Variablen und Wertzuweisungen

- Um in Java eine **Referenz-Variable** zu **deklarieren**, geben wir die Klasse und den Bezeichner an und schließen mit einem Semikolon ab:

```
String      greeting; // Referenz auf einen Begrüßungs-String
PrintStreamoutput;    // Referenz auf dasselbe PrintStream-
                      // Objekt wie System.out
```

- Um einer Variable einen Wert zu geben, verwenden wir eine so genannte **Wertzuweisung**:

Variable = Wert;

Beispiele:

```
output = System.out;
greeting = "Hello";
```

Wir können jetzt schreiben:

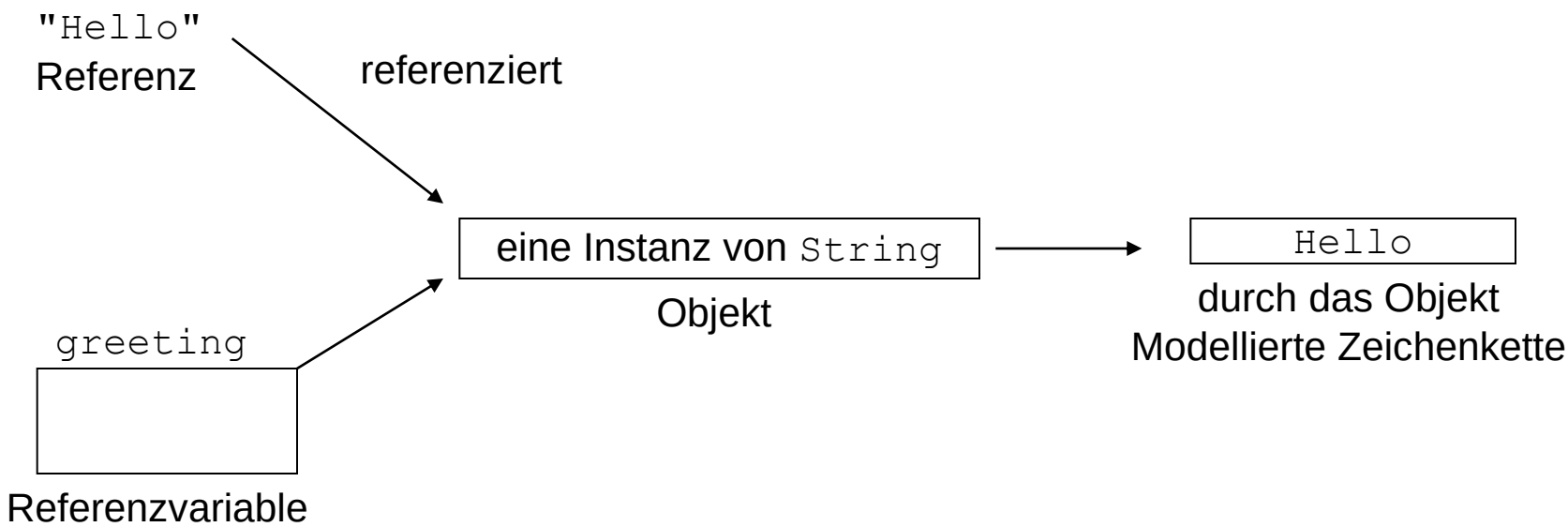
```
output.println(greeting.toUpperCase());
```

Effekt einer Wertzuweisung an eine Referenz-Variable

Nach

```
greeting = "Hello";
```

referenzieren `"Hello"` und `greeting` **dasselbe** Objekt.



Wertzuweisung versus Gleichheit

- Betrachte

```
t = "Springtime";
t = "Wintertime";
```

- **Eine Wertzuweisung ordnet einer Variablen den Wert auf der rechten Seite des Statements zu**.
- **Der bisherige Wert der Variablen geht verloren**.
- Nach der ersten Zuweisung ist der Wert von `t` die Referenz auf das durch `"Springtime"` referenzierte `String`-Objekt.
- Nach der zweiten Zuweisung ist der Wert von `t` die Referenz auf das `"Wintertime"`–Objekt.
- Wir sagen: „**Eine Wertzuweisung ist *imperativ***."
- **Variablen enthalten immer nur den letzten, ihnen zugewiesenen Wert**.

Rollen von Variablen

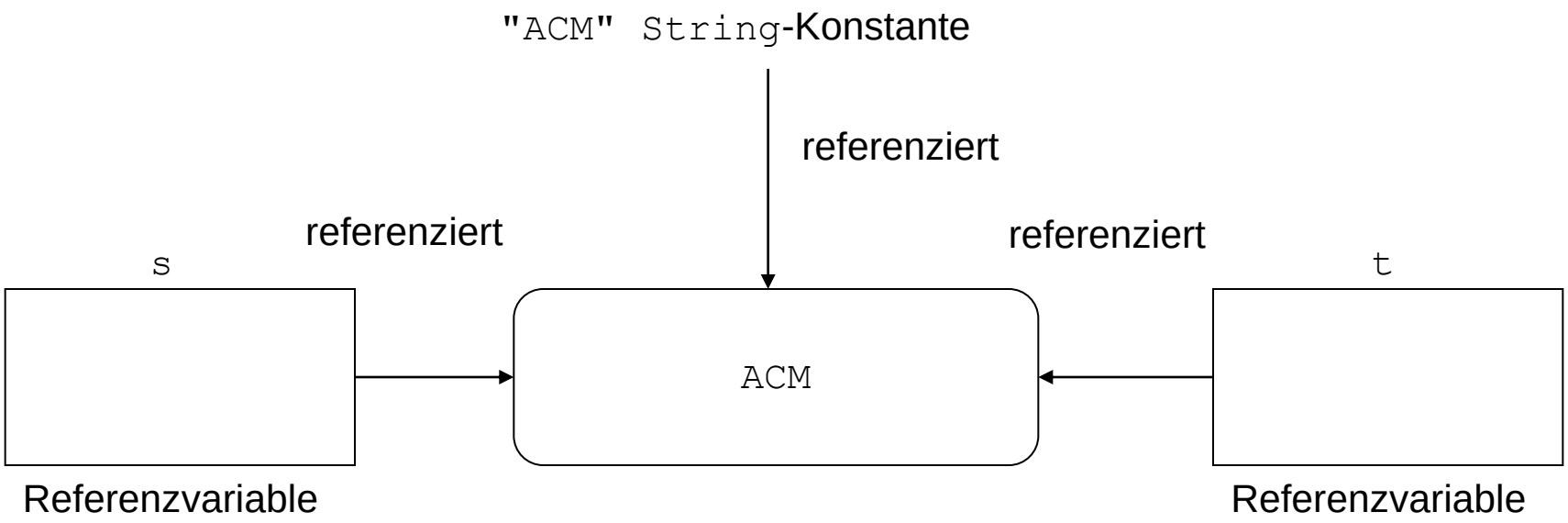
- Je nachdem, wo Variablen auftreten, können sie
  1. Informationen speichern oder
  2. den in ihnen gespeicherten Wert repräsentieren.
- Beispiele:

```
String s, t;
s = "Springtime";
t = s;
```

- Die Erste Zuweisung speichert in `s` die Referenz auf das `"Springtime"`–Objekt **(Fall 1)**.
- Die zweite Zuweisung bewirkt, dass `t` und `s` dasselbe Objekt referenzieren **(Fall 1 für `t` und Fall 2 für `s`)**.

Mehrere Referenzen auf dasselbe Objekt

```
String s, t;
s = "ACM";
t = s;
```



Unabhängigkeit von Variablen

```
String s;
String t;
s = "Inventory";
t = s;
```

- Nach der Anweisung `t = s` referenziert `t` dasselbe Objekt wie `s`.
- Wenn wir anschließend `s` einen neuen Wert zuweisen, z.B. mit `s = "payroll"`, ändert das den Wert für `t` nicht.
- Durch `t = s` wird lediglich der Wert von `s` in `t` kopiert.
- Eine Wertzuweisung realisiert keine permanente Gleichheit.
- Variablen sind unabhängig voneinander, d.h. ihre Werte können unabhängig voneinander geändert werden.

Reihenfolge von Statements (erneut) (1)

```
String greeting;
String bigGreeting;
greeting = "Yo, World";
bigGreeting = greeting.toUpperCase();
System.out.println(greeting);
System.out.println(bigGreeting);
```

Ausgabe

```
Yo, World
YO, WORLD
```

Reihenfolge von Statements (erneut) (2)

- Alternativ dazu hätten wir auch die folgende Reihenfolge verwenden können:

```
String greeting;
greeting = "Yo, World";
System.out.println(greeting);
String bigGreeting;
bigGreeting = greeting.toUpperCase();
System.out.println(bigGreeting);
```

- **Deklarationen** können an jeder Stelle auftauchen, vorausgesetzt sie **gehen jedem anderen Vorkommen der deklarierten Variablen voraus**.
- Deklarationen können auch so genannte **Initialisierungen** enthalten:

```
String greeting = "Yo, World";
```

Methoden, Argumente und Return-Werte

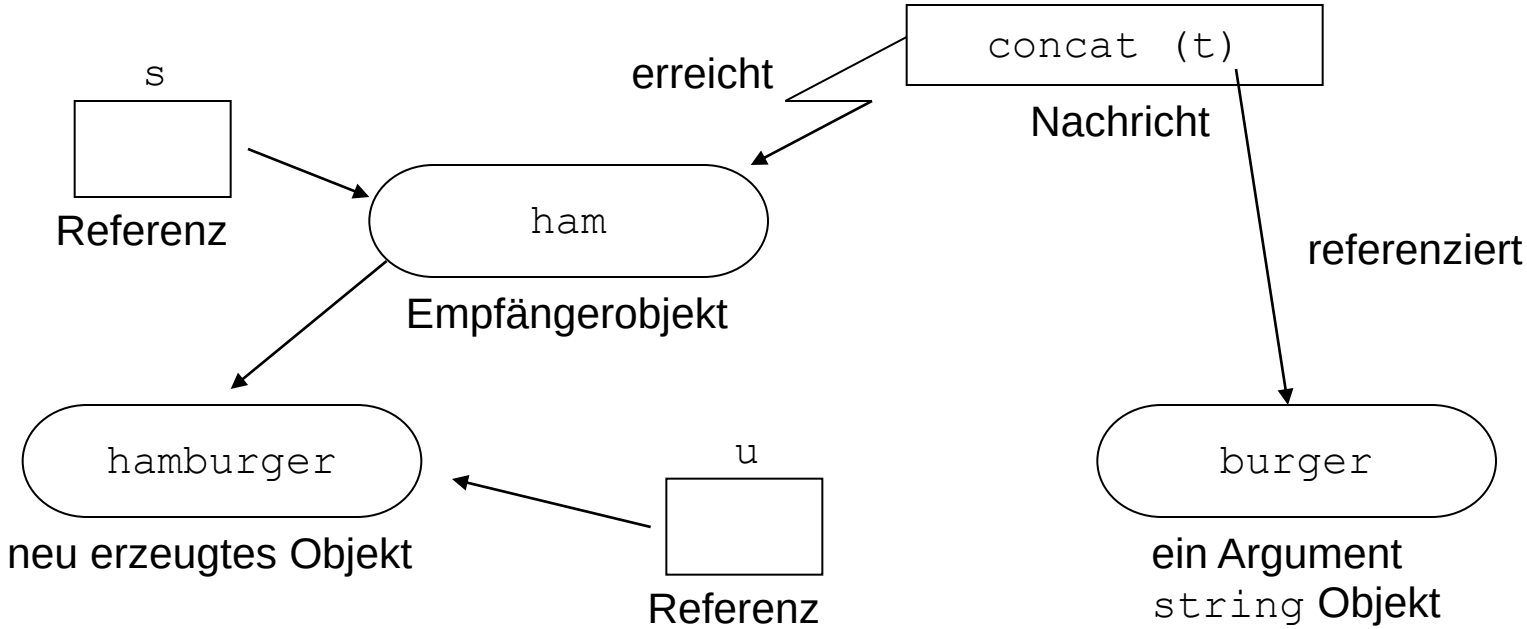
Methode	Return-Wert	Argumente
toUpperCase	`String`-Objekt-Referenz	keine
toLowerCase	`String`-Objekt-Referenz	keine
length	eine Zahl	keine
trim	`String`-Objekt-Referenz	keine
concat	`String`-Objekt-Referenz	`String`-Objekt-Referenz
substring	`String`-Objekt-Referenz	eine Zahl
substring	`String`-Objekt-Referenz	`zwei Zahlen`

Eigenschaften dieser Methoden

- `length` gibt die Anzahl der Zeichen Empfängerobjekt zurück.
  `"Hello".length()` ist der Wert `5`.
- `trim` liefert eine Referenz auf ein String-Objekt, welches sich durch das Argument dadurch unterscheidet, dass führende oder nachfolgende Leer- oder Tabulatorzeichen fehlen.
  `" Hello ".trim()` ist eine Referenz auf `"Hello"`.
- `concat` liefert eine Referenz auf ein String-Objekt, welches sich durch das Anhängen des Argumentes an das Empfängerobjekt ergibt.
  `"ham".concat("burger")` ist eine Referenz auf `"hamburger"`.

Wirkung der Methode `concat`

```
String s, t, u;
s = "ham";
t = "burger";
u = s.concat(t);
```

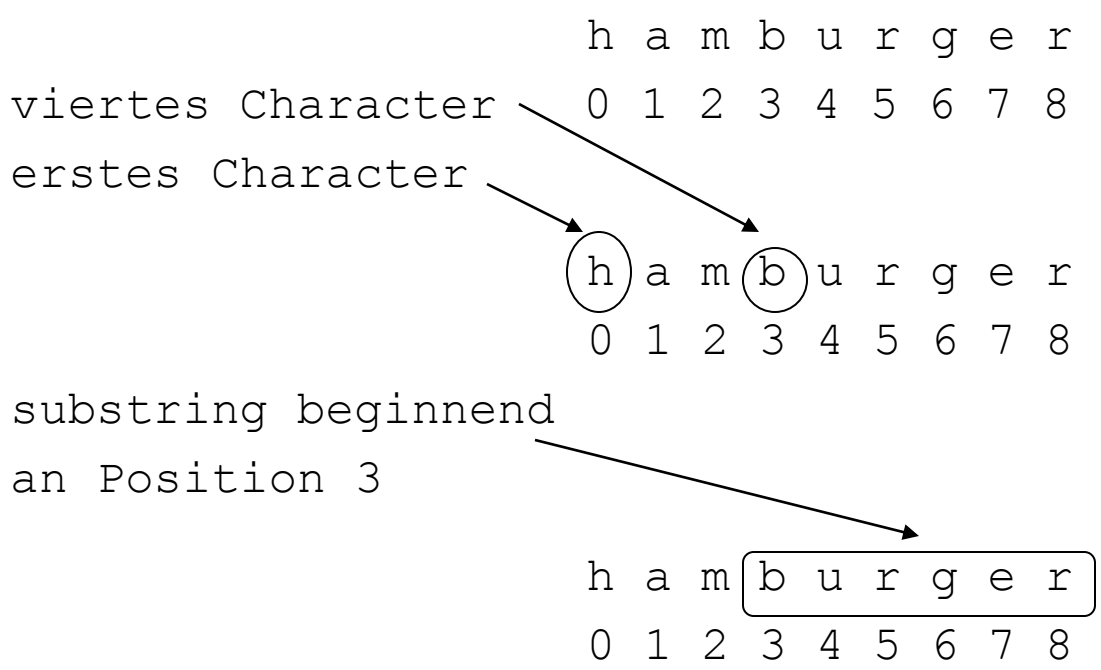


Die Varianten der Methode `substring`

- `substring` **erzeugt eine Referenz auf eine Teilsequenz der Zeichenkette des Empfängerobjekts**.
- **In Java startet die Nummerierung bei 0**.
- Die Variante **mit einem Argument** gibt eine **Referenz** auf den **Teilstring** zurück, der an der durch den Wert des **Argumentes spezifizierten Position beginnt**.
- Die Version mit **zwei Argumenten** gibt eine **Referenz** auf den **Teilstring**, der an der durch den Wert des **ersten Argumentes gegebenen Position beginnt** und **unmittelbar vor** der durch das **zweite Argument gegebenen Position endet**.

Funktion der Methode `substring` mit einem Argument

```
String s, t;
s = "hamburger";
t = s.substring(3);
```

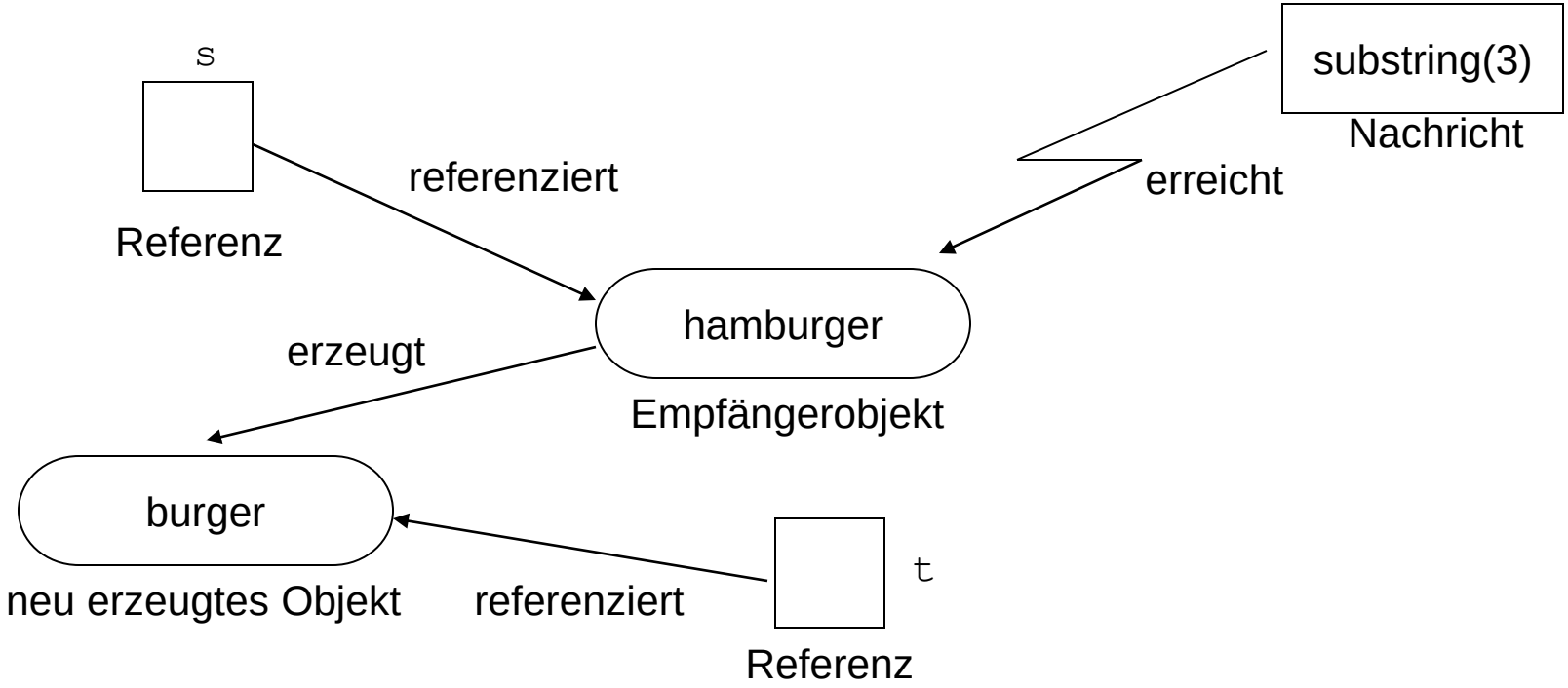


Funktion der Methode `substring` mit einem Argument

```
String s, t;

s = "hamburger";

t = s.substring(3);
```



Funktion der Methode `substring` mit zwei Argumenten

```
String s, t;

s = "hamburger";

t = s.substring(3,7);
```

`"hamburger".substring(3,7)`

hamburger, from 3 to 7

substring endet hier

substring beginnt hier

h a m b u r g e r

0 1 2 3 4 5 6 7 8

Beispiel: Finden des mittleren Zeichens einer Zeichenkette

```
String word = "antidisestablishmentarianism";
```

Zur Berechnung der mittleren Position des Wortes verwenden wir

```
word.length() / 2
```

Das mittlere Zeichen berechnen wir dann mit:

```
word.substring(word.length()/2, word.length()/2 + 1);
```

Das komplette Programm

```
class Program2 {
      public static void main(String[] arg) {
            String word = "antidisestablishmentarianism";
            String middle;
            middle = word.substring(word.length()/2,
                                    word.length()/2 + 1);
            System.out.println(middle);
      }
}
```

Ausgabe des Programms:

```
s
```

Überladung/Overloading

- Die `String`-Klasse hat **zwei Methoden** `substring`.
- Beide Methoden haben verschiedene **Signaturen**, denn sie unterscheiden sich in den Argumenten, die sie benötigen.
- Und sie haben unterschiedliche Funktionalität.
- **Methoden mit gleichem Namen und unterschiedlichen Signaturen** heißen **überladen** bzw. **overloaded**.
- Der Technik, Klassen mit mehreren Methoden gleichen Namens zu entwerfen, heißt **Überladung** bzw. **Overloading**.

Kaskadieren von Methodenaufrufen

```
String s1 = "ham", s2 = "bur", s3 = "ger", s4;
```

Um eine Referenz auf die Verkettung der drei durch `s1`, `s2`, und `s3` referenzierten `String`-Objekte zu erzeugen, müssten wir schreiben:

```
s4 = s1.concat(s2);
s4 = s4.concat(s3);
```

Dies geht jedoch einfacher mit `s4 = s1.concat(s2).concat(s3);`

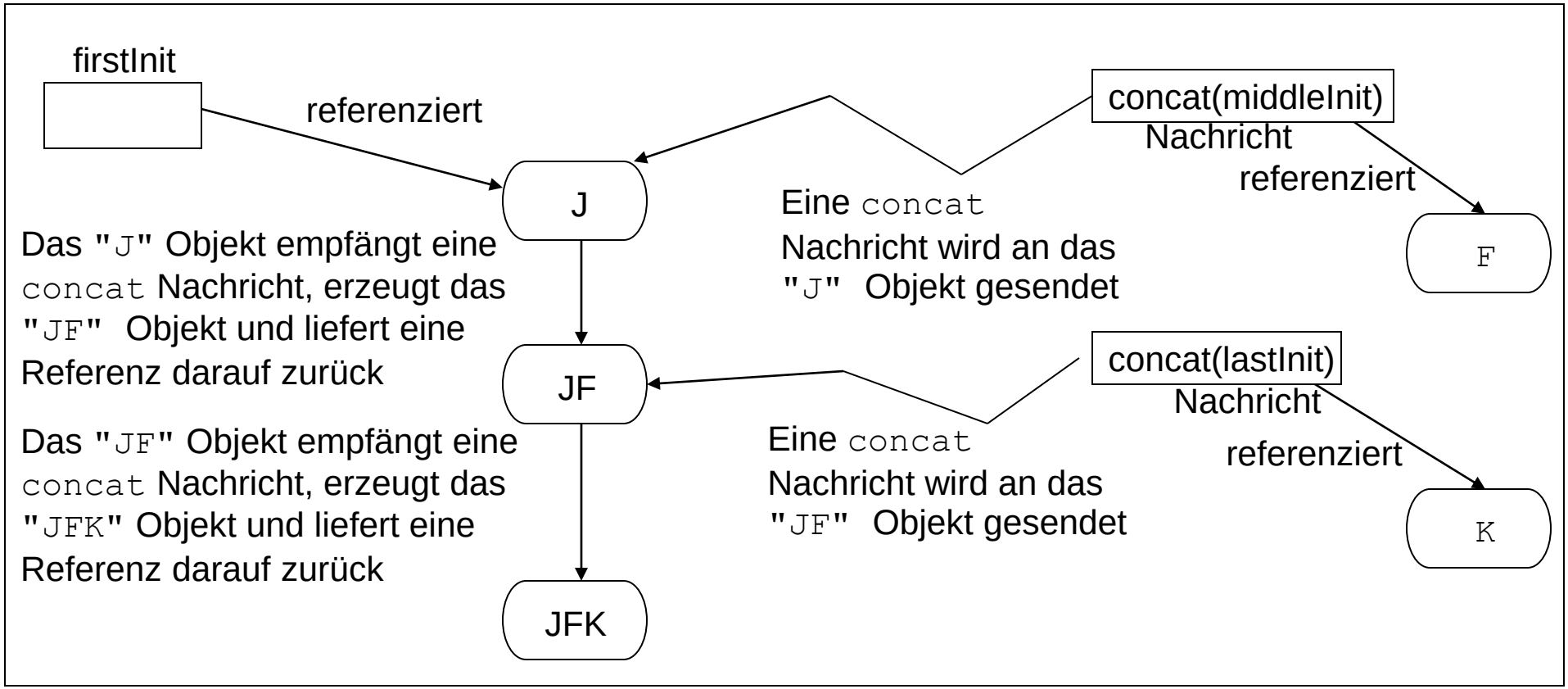
Funktionsweise:

1. Die Message `concat(s2)` wird an `s1` gesendet.
2. Die Message `concat(s3)` wird an das Ergebnisobjekt gesendet.
3. Die Referenz auf das dadurch erzeugte `String`-Objekt wird `s4` zugewiesen.

Funktion kaskadierter Nachrichten

```
String firstInit = "J", middleInit = "F", lastInit = "K";

System.out.println(firstInit.concat(middleInit).concat(lastInit));
```



Schachteln von Methodenaufrufen / Komposition

Alternativ zu Kaskaden wie in

```
s4 = s1.concat(s2).concat(s3);
```

können wir Methodenaufrufe auch schachteln, was einer Komposition entspricht:

```
s4 = s1.concat(s2.concat(s3));
```

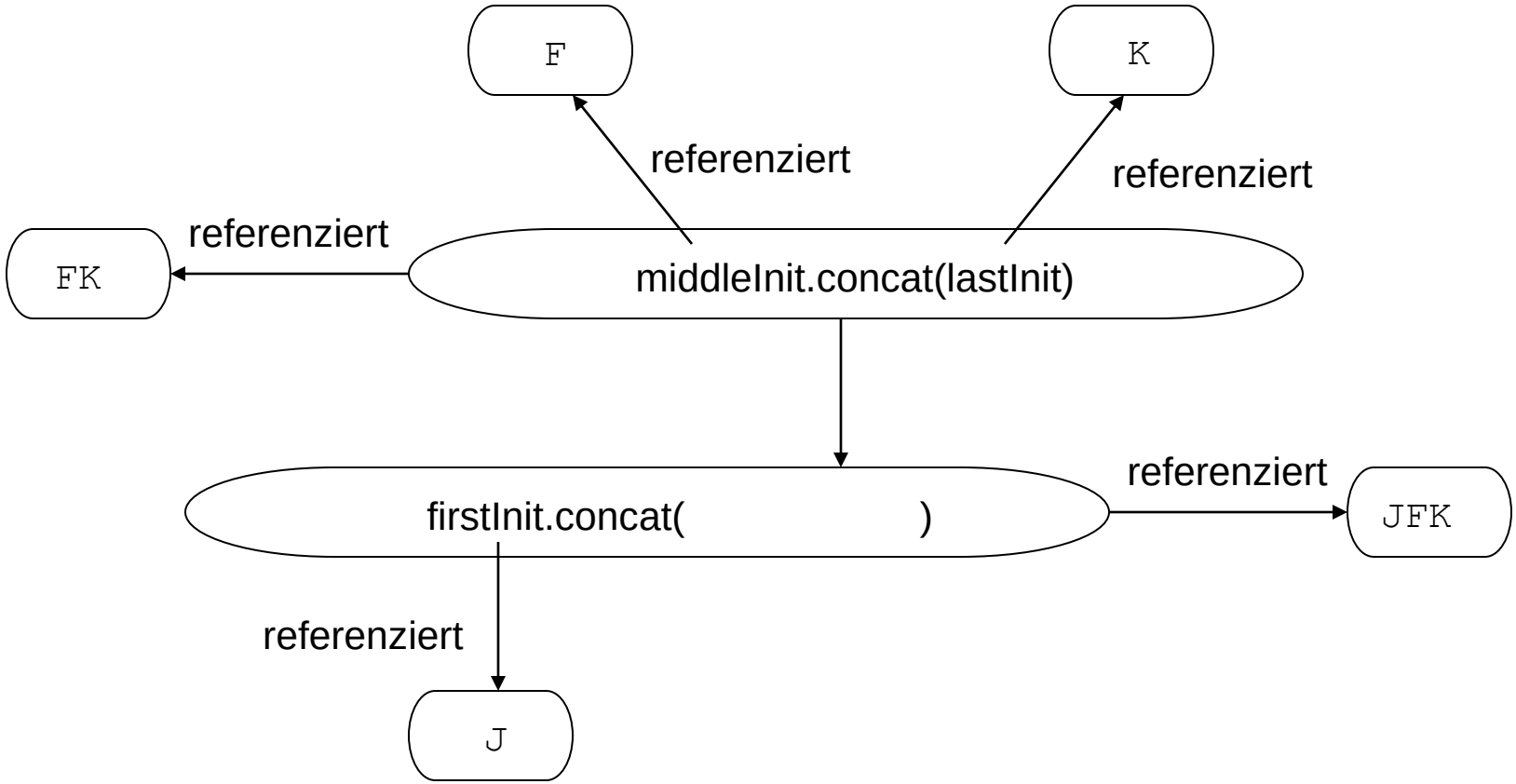
Auswertung des Ausdrucks auf der rechten Seite:

1. Die Message `concat(s3)` wird an `s2` gesendet.
2. An das `s1`-Objekt wird eine `concat`-Nachricht geschickt, die als Argument eine Referenz auf das in Schritt 1 erzeugte Ergebnisobjekt hat.
3. Die Referenz auf das dadurch erzeugte `String`-Objekt wird schließlich `s4` zugewiesen.

Wirkung der Komposition

```
String firstInit = "J", middleInit = "F", lastInit = "k";

System.out.println(firstInit.concat(middleInit.concat(lastInit)));
```



Weitere Eigenschaften von String-Objekten

- `String`-Objekte können nicht verändert werden, da alle Funktionen als Return-Wert neue Objekte liefern.
- Der leere String `""` hat die Länge 0, d.h. `"".length()` liefert 0.

Erzeugen von Objekten

- Jede Klasse hat wenigstens eine Methode zum Erzeugen von Objekten.
- Solche Methoden heißen Konstruktoren.
- Der Name eines Konstruktors stimmt stets mit dem der Klasse überein.
- Wie andere Methoden auch, können Konstruktoren Argumente haben.
- Da wir mit Konstruktoren neue Objekte erzeugen wollen, können wir sie an kein Objekt senden.
- In Java verwenden wir das Schlüsselwort `new`, um einen Konstruktor aufzurufen:

```
new String("hello world")
```

- Dies erzeugt ein `String`-Objekt und sendet ihm die Nachricht

```
String("hello world").
```

Die Operation `new`

- Das Schlüsselwort `new` bezeichnet eine Operation, die einen Wert zurückgibt.
- Der Return-Wert einer `new`-Operation ist die Referenz auf das neu erzeugte Objekt.
- Wir nennen `new` einen Operator.

Sichern neu erzeugter Objekte

- Der `new`-Operator liefert uns eine Referenz auf ein neues Objekt.
- Um dieses Objekt im Programm verwenden zu können, müssen wir die Referenz in einer Referenzvariablen sichern.

Beispiel:

```
String s, t, upper, lower;
s = new String("Hello");
t = new String(); // identisch mit ""

upper = s.toUpperCase();
lower = s.toLowerCase();

System.out.println(s);
```

Zusammenfassung (1)

- Das **Verhalten von Objekten** wird durch **Methoden** spezifiziert.
- Die **Signatur** einer Methode besteht aus dem **Namen** der Methode sowie der **Anzahl und den Typen der Argument**e.
- Der **Prototyp** einer Methode ist die **Signatur zusammen mit dem Return-Wert**.
- Wird eine **Nachricht** an ein Objekt **gesendet**, wird der **Code des Aufrufers unterbrochen** bis die Methode ausgeführt worden ist.
- Einige Methode haben **Return-Werte**. Wenn eine Methode **keinen Return-Wert** hat, ist der Return-Typ **void**.
- **Variablen** können **Werte zugeordnet** werden.

Zusammenfassung (2)

- **Verschiedene Variablen** sind **unabhängig** voneinander.
- Jede Variable hat zu einem Zeitpunkt **nur einen Wert**.
- **Wertzuweisungen sind destruktiv**,d.h. sie löschen den vorhergehenden Wert der Variablen.
- **Referenzwert**e, die von Methoden zurückgegeben werden, **können Variablen zugewiesen werden**.
- **Return-Werte** können aber auch **Empfänger** neuer Nachrichten sein. Dies heißt **Kaskadierung**.
- **Return-Werte** können auch als **Argumente** verwendet werden. Dieser Prozess heißt **Komposition**.

Zusammenfassung (3)

- Neue Objekte einer Klasse können mit dem `new`-**Operator** erzeugt werden.
- Zusammen mit dem `new`-Operator verwenden wir den **Konstruktor**, der den gleichen Bezeichner hat wie die Klasse selbst.