Übersicht

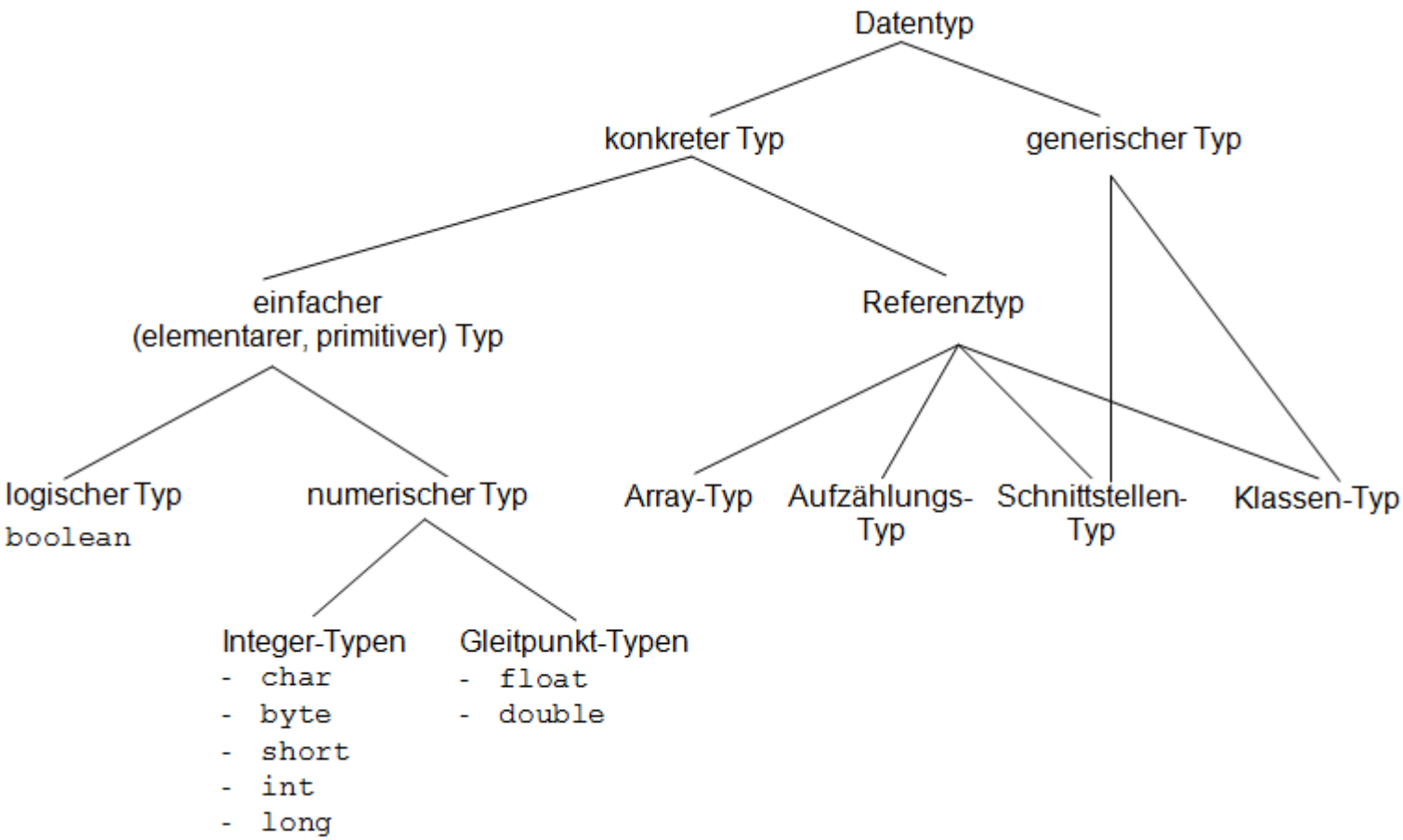
1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Klassifikation und Datentypen
- Variable
- Array-Typ und Aufzählungstyp
- Zeichenketten
- StringBuffer
- Wandlung von Datentypen

Klassifikation und Datentypen

- Unterscheidung generell zwischen konkreten und generischen Datentypen



Klassifikation und Datentypen

- Java ist streng typisiert, d. h. der Datentyp einer Variablen muss zur Übersetzungszeit bekannt sein.
- Mit der Deklaration kann gleich ein Wert zugewiesen werden.
- Typname Variablenname [Zuweisung] ;

```
1  public class Kohl
2  {
3      public static void main( String[] args)
4      {
5          String name = "Helmut Kohl";
6          int hausnummer, alter = 68;
7          double einkommen = 120000;
8          char geschlecht = 'M';
9          boolean lKanzler = true;
10     }
11 }
```

Referenztyp

Klassifikation und Datentypen: Der Klassen-Typ

- Klassendefinition gibt den Namen eines neuen Datentyps bekannt und definiert zugleich dessen Methoden und Datenfelder.
- Eine Deklaration gibt dem Compiler einen neuen Namen bekannt. Die Definition einer Klasse erfolgt innerhalb eines Klassenrumpfes.
- Eine Methode ist eine Anweisungsfolge, die unter einem Namen abgelegt ist und über ihren Namen aufrufbar ist.
- Die Methoden eines Objektes haben direkten Zugriff auf die Datenfelder und Methoden desselben Objektes.
- Klassen-Typen sind Referenztypen, Referenztypen haben als Variablen Zeiger auf Objekte
- Von einem fremden Objekt aus wird eine Methode ohne bzw. mit Parameter eines anderen Objektes aufgerufen, indem das fremde Objekt den Punktoperator anwendet und den Methodennamen gefolgt von einem leeren (runden) Klammerpaar oder benötigten Parametern angibt. p.print() oder p.println(„Hallo Welt“)

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Klassifikation und Datentypen
- ✓ Variable

Array-Typ und Aufzählungstyp

Zeichenketten

StringBuffer

Wandlung von Datentypen

Variablen I

- Unterscheidung in statische und dynamische Variablen
- Eine Variable, die in einer Methode definiert wird, ist sowohl eine lokale wie auch eine statische Variable

```
1  // TestPunkt7.java
2
3  public class TestPunkt7
4  {
5     public static void main (String[] args)
6     {
7        int x = 3;                // x ist eine statische Variable
8                                  // eines einfachen Datentyps
9        Punkt7 p;                 // Die Referenzvariable p ist
10                                 // eine statische Variable
11
12       p = new Punkt7();         // Erzeugen einer dynamischen
13                                 // Variablen mit dem new-Operator
14       p.setX (x);               // Aufruf der Methode setX()
15       System.out.println ("Die Koordinate des Punktes p ist: ");
16       System.out.println (p.getX());
17    }
18 }
```

Variablen II

- Referenzvariable ermöglichen den Zugriff auf Objekte im Heap
- Ein Objekt wird vom Laufzeitsystem als dynamische Variable auf dem Heap, der ein Speicherreservoir für dynamische Variablen darstellt, angelegt.
- Referenzvariable zeigen in Java entweder auf Objekte oder nichts (*null* Referenz)
- Referenzen gibt es in Java nur auf Objekte, nicht auf Variablen einfacher Datentypen.
- Objekte können in Java nicht direkt manipuliert werden. Man kann auf Objekte nur indirekt mit Hilfe von Referenzen zugreifen (Beispiel TV-Fernbedienung).
- Ein Objekt wird in Java erzeugt durch die Anweisung

  new Klassenname();
- Definition einer Referenzvariablen und die Erzeugung eines Objektes

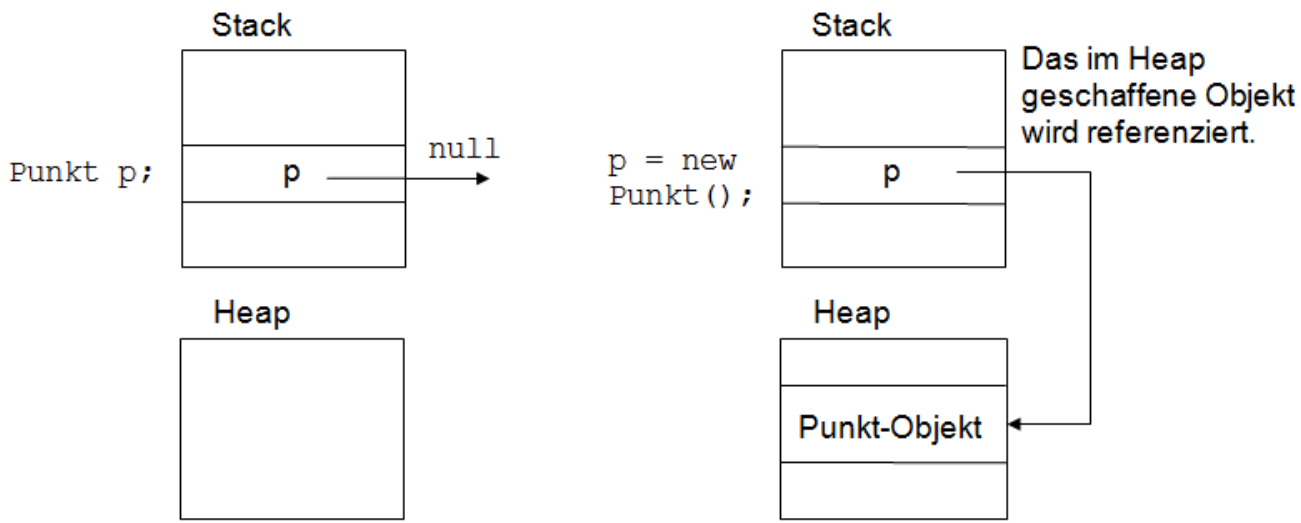
  Klassenname var = new Klassenname();

- Hinweis: Referenz auf ein Objekt vs. Objekt. Wird oft synonym benutzt

Variablen III

- Eine Referenzvariable als lokale Variable in einer Methode wird vom Compiler nicht automatisch initialisiert.

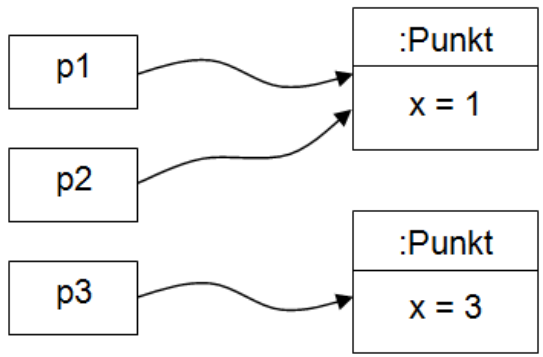
```
44  public class VariablenTypen
45  {
46      public static void main (String[] args)
47      {
48          Punkt p;
49          p = new Punkt();          // Punkt erzeugen
50          // weitere Anweisungen
51      }
52  }
```

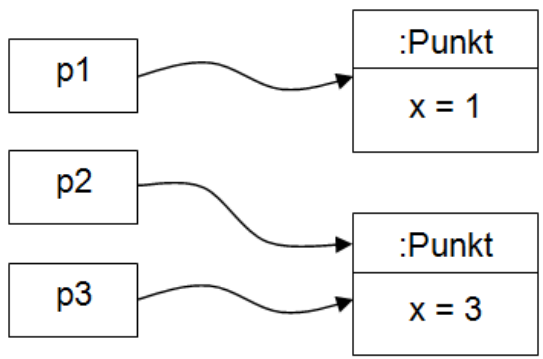


Variablen IV

- Beispiel mit zwei Punkt-Objekten und drei Referenzvariablen p1, p2 und p3
- p1 zeigt auf ein Punkt-Objekt; p3 zeigt auf ein anderes Punkt-Objekt

```
1  // Datei: TestPunkt8.java
2
3  public class TestPunkt8
4  {
5     public static void main (String[] args)
6     {
7        Punkt8 p1 = new Punkt8(); // Anlegen eines Punkt-Objektes
8        p1.setX (1);              // Dieses enthält den Wert x = 1
9        Punkt8 p2;                // Anlegen einer Referenzvariablen
10                                 // vom Typ Punkt
11       Punkt8 p3 = new Punkt8(); // Anlegen eines Punkt-Objektes
12       p3.setX (3);              // x wird 3
13
14       p2 = p1;                  // Nun zeigt p2 auf dasselbe Objekt
15                                 // wie p1
16       System.out.println ("p1.x hat den Wert " + p1.getX());
17       System.out.println ("p2.x hat den Wert " + p2.getX());
18       System.out.println ("p3.x hat den Wert " + p3.getX());
19
20       p2 = p3;                  // Nun zeigt p2 auf dasselbe
21                                 // Objekt wie p3
22       System.out.println ("p2.x hat den Wert " + p2.getX());
23
24       p2.setX (20);
25       System.out.println ("p2.x hat den Wert " + p2.getX());
26       System.out.println ("p3.x hat den Wert " + p3.getX());
27    }
28 }
```





Variablen V

- Beim Zugriff auf eine mit ***null*** initialisierte Referenzvariable erzeugt die Laufzeitumgebung eine Exception (Software-Ausnahme) vom Typ ***NullPointerException***. Diese Exception führt zu einem Programmabsturz..

```
1  // Datei: CompilerTest2.java
2
3  class Punkt
4  {
5     private int x;
6
7     public void print()
8     {
9        System.out.println ("x: " + x);
10    }
11 }
12
13 public class CompilerTest2
14 {
15    public static void main (String[] args)
16    {
17       // Anlegen einer mit null initialisierten lokalen Variablen
18       Punkt p = null;
19
20       // Zugriff auf die mit null initialisierte lokale Variable p
21       p.print();
22    }
23 }
```

Variablen VI

- In Java gibt es drei Arten von Variablen:
- Klassenvariable: Werden für jede Klasse einmal angelegt.
- Instanzvariable Werden für jede Instanz einer Klasse (also für jedes Objekt) angelegt.
- Lokale Variable Gibt es in Methoden. Die Gültigkeit der lokalen Variablen kann sich auf den Methodenrumpf oder auf einen inneren Block, z.B. in einer *for*-Schleife, erstrecken.

```
1  // Datei: VariablenTypen.java
2
3  public class VariablenTypen
4  {
5     private int x;           // dies ist eine Instanzvariable
6     private static int y;    // dies ist eine Klassenvariable
7
8     public void print()
9     {
10       int z = 0;            // dies ist eine lokale Variable
11    }
12 }
```

Variablen VII: Speicherbereiche

- ***Stack***: Dient bei Programmen als Speicherbereich, um Daten zu organisieren. Bei einem
- Methodenaufruf werden auf dem Stack die lokalen Variablen einer Methode und die Rücksprungadresse einer Methode hinterlegt, die durch den Aufruf einer anderen Methode in ihren eigenen Anweisungen unterbrochen wurde.
- ***Heap***: ist ein Speicherbereich, in dem von der virtuellen Maschine die dynamischen Objekte angelegt werden. Wird ein Objekt auf dem Heap von keiner Referenzvariablen mehr referenziert, so wird der von dem Objekt belegte Speicherplatz durch den Garbage Collector freigegeben.
- Den Speicherbereich, in dem die virtuelle Maschine den Programmcode einer Klasse und die Klassenvariablen ablegt, bezeichnet man als ***Method Area***.

Aufgabe 03.a

- Erstellen Sie in Eclipse / Netbeans die Klasse Auto mit folgenden Variablen:

  - ***int ps*** und String ***hersteller*** als Instanzvariablen
  - ***int anzahl*** als Klassenvariable

- Erstellen Sie einen parameterlosen Konstruktor, der die Instanzvariablen initialisiert und die Variable anzahl inkrementiert.
- Erstellen Sie zusätzlich einen Konstruktor mit zwei Parametern, der die beiden Instanzvariablen initialisiert und die Variable anzahl inkrementiert.

- Testen Sie die Klasse mit der Klasse AutoMain, welche die main()-methode enthält und zwei unterschiedliche Auto-Objekte erzeugt.

Variablen VIII: Konstante

- In Java kann jede Art von Variable konstant gemacht werden. Das heißt ihr Wert ist konstant und kann nicht mehr verändert werden.
- Dieses trifft für Referenzvariablen ebenso zu.
- Aber: Es gibt in Java keine Möglichkeit, ein *Objekt* konstant zu machen
- Mit Hilfe des Modifikators *final* wird eine Variable zur Konstanten:

```
final int kVar = 1;
final Punkt p = new Punkt p();
```

Modifikatoren I

- Die folgende Tabelle zeigt, welche Zugriffsrechte bei den Modifikatoren bestehen:

	Die Klasse selbst	Paket-Klassen/ innere Klassen	Unterklassen	Sonstige Klassen
private	Ja	Nein[1]	Nein	Nein
(ohne)	Ja	Ja	Nein	Nein
protected	Ja	Ja	Ja	Nein
public	Ja	Ja	Ja	Ja

[1] Um inneren Klassen den Zugriff auf private Methode und Eigenschaften dennoch zu ermöglichen, werden vom Compiler statische, paket-private Methoden erstellt, die den Aufruf, das Setzen bzw. das Auslesen emulieren. Diese Methoden tragen den Namen `access$xxx`, wobei `xxx` für eine fortlaufende Nummer steht.

Modifikatoren II

- Die folgende Tabelle zeigt, welcher Modifikator mit einem Datenfeld, einer Methode, einem Konstruktor, einer Klasse oder einer Schnittstelle eingesetzt werden darf:

	Datenfeld	Methode	Konstruktor	Klasse	Schnittstelle
abstract		ja		ja	
final	ja	ja [1]		ja	
native		ja			
private	ja	ja	ja	ja	ja
protected	ja	ja	ja	ja	ja
public	ja	ja	ja	ja	ja
static	ja	ja		ja	ja
synchronized		ja			
transient [2]	ja				
volatile [3]	ja				

1) Kann nicht überschrieben werden.
2) Datenfelder die bei der Serialisierung (sichern und wiederherstellen des Objektzustandes) nicht berücksichtigt werden sollen.
3) Datenfelder die gleichzeitig von mehreren Threads verändert werden können.

Übersicht

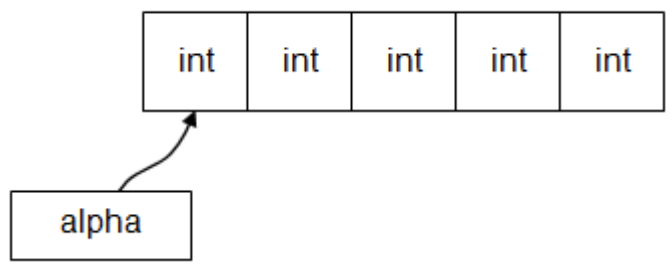
1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Klassifikation und Datentypen
- ✓ Variable
- ✓ Array-Typ und Aufzählungstyp
- Zeichenketten
- StringBuffer
- Wandlung von Datentypen

Array-Typ I

- Ein Array ist ein Objekt, das aus Komponenten (Elementen) zusammengesetzt ist, wobei jedes Element eines Array vom selben Datentyp sein muss.
- Man kann in Java ein Array aus Elementen eines einfachen Datentyps oder aus Elementen eines Referenztyps anlegen. Ein Element eines Arrays kann auch selbst wieder ein Array sein. Dann entsteht ein mehrdimensionales Array.
- Der Zugriff auf ein Element eines Arrays erfolgt über den Array-Index. Hat man ein Array mit n Elementen definiert, so läuft der *Index von 0 bis n-1*. Vorteil: Leichtere Bearbeitung innerhalb von Schleifen.
- Definition einer Array-Variablen bedeutet nicht das Anlegen eines Array, sondern die Definition einer Referenzvariablen, die auf ein Array-Objekt zeigen kann. (Dieses Array-Objekt muss im Heap angelegt werden.) int[] alpha;



Array-Typ II

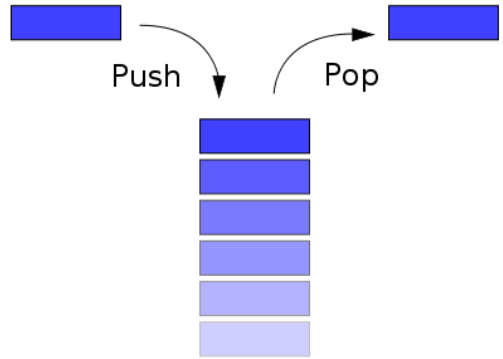
- Arrays werden in drei Schritten angelegt:

  1) Definition einer Referenzvariablen
  2) Erzeugen des Arrays, d.h. eines Array-Objektes, welches aus Komponenten (Elementen) besteht
  3) Belegen der Array-Elemente mit Werten, d.h. Initialisierung des Arrays.

- Array-Variable ist eine Referenzvariable mit deren Definition das Array-Objekt noch nicht angelegt ist.

- Es ist nicht möglich, über die Grenzen eines Arrays hinaus andere Speicherbereiche zu überschreiben oder auszulesen. Bei einem solchen Versuch wird sofort eine *ArrayIndexOutOfBoundsException* geworfen.

Aufgabe 03.02



Ein Stack ist eine stapelförmige Datenstruktur.

Elemente werden übereinander gestapelt nach dem

Prinzip Last-In-First-Out (LIFO).

- Legen Sie ein int-Array als Stack der Größe 5 an. Implementieren Sie die folgenden Methoden:
  - public void push(int i) — einen int-Wert auf den Stapel legen
  - public int pop() — den obersten int-Wert vom Stapel nehmen
  - public void stackPrint() — den Stack-Inhalt auf der Systemausgabe ausgeben
- Befüllen Sie den Stack zunächst per push mit drei Werten und geben Sie den Inhalt aus. Danach nehmen Sie das oberste Element per pop vom Stapel und geben es aus. Nun geben Sie drei weitere Werte hinzu und geben erneut den Inhalt aus.
- Abschließend fügen Sie einen weiteren (den 6. Wert!!!) hinzu, beobachten Sie, was passiert.
- Legen Sie das int-Array als Instanzvariable an. Zusätzlich werden Sie eine int-Variable für den jeweils aktuellen Index benötigen. Abfragen bzgl. der Einhaltung der Array-Grenzen brauchen nicht implementiert werden.

Array-Typ III

- Für das Erzeugen des Array-Objektes bestehen zwei Möglichkeiten:
- Array wird mit *new* erzeugt:

```
byte[] bArray;
bArray = new byte [4] ;
```

oder in einem Schritt:

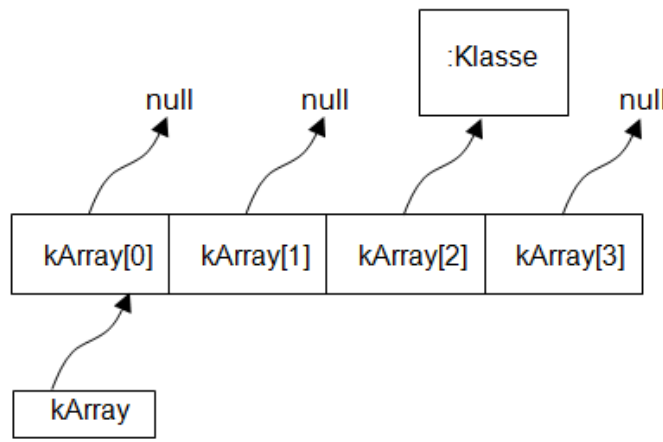
```
byte[] bArray = new byte [4] ;
```

- Implizites Erzeugen mit einer Initialisierungsliste:

```
byte[] bArray = {1,2,3,4};
```

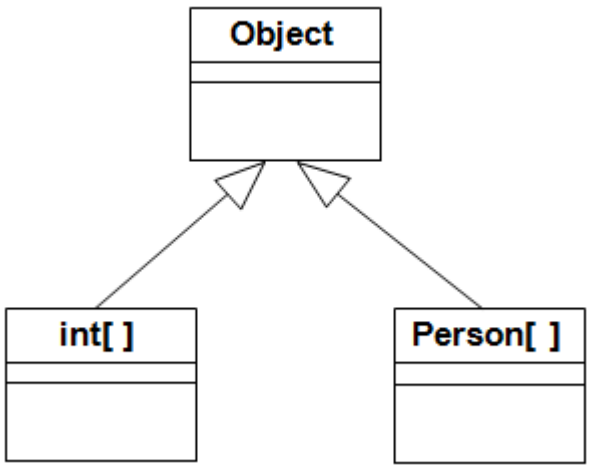
- Arrays aus Referenztypen möglich:

```
Punkt[] aPunkte;
aPunkte = new Punkt[4];
Punkt a = new Punkt();
aPunkte[2] = a;
```



Array-Typ IV

- Objektcharakter von Arrays. Alle Objekte erben von der Oberklasse Object



  - Jedes Objekt erbt die Methode equals() zur Überprüfung auf Gleichheit der Referenzen

    public boolean equals(Object ref)

  - Die Methode a.equals(b) gibt ein true zurück, wenn a und b Referenzen auf das selbe Objekt sind.

- Mehrdimensionale Arrays sind Arrays aus Arrays.

  int [] [] [] dreiDimArray = new int [10] [20] [30];

  dreiDimArray [3] [1] [2] = 12;

- Java erlaubt mehrere Syntax-Varianten

  int zahl []; oder char[] buffer;

Mehrdimensionale Arrays

- Mehrdimensionale Arrays sind Arrays aus Arrays
- zur Darstellung einer Matrix
  - *im einfachsten Fall eine Tabelle (also 2-dimensional)*

```
int[][] zweiDimArray = new int[6][4];
zweiDimArray[2][1] = 8;
```

[0]	[0] 0	[1] 0	[2] 0	[3] 0
[1]	[0] 0	[1] 0	[2] 0	[3] 0
[2]	[0] 0	[1] 8	[2] 0	[3] 0
[3]	[0] 0	[1] 0	[2] 0	[3] 0
[4]	[0] 0	[1] 0	[2] 0	[3] 0
[5]	[0] 0	[1] 0	[2] 0	[3] 0

Aufzählungstyp

- Aufzählungstypen sind mit dem JDK 5.0 hinzugekommen
- Aufzählungstypen werden auch enums genannt.
- enums sind Datentypen, deren Wertebereich eine endlich geordnete Menge von Konstanten zulässt.
- Ein Aufzählungstyp trägt einen Namen

```
enum AmpelFabe {ROT, GELB, GRUEN};
```

Aufgabe 03.05

- Wochentage

  Definieren Sie einen Aufzählungstyp Wochentag, der die Tage der Woche repräsentiert, und eine Klasse Aufg_03_05. In der main()-Methode der Klasse Aufg_03_05 soll die Methode values() des Aufzählungstyps verwendet werden, um alle Wochentage auszugeben. Zu jedem Wochentag soll die jeweilige Ordinal-Zahl mit Hilfe der Methode ordinal() ausgegeben werden. Die Ausgabe soll folgendermaßen aussehen:

```
MONTAG ist der 1. Tag der Woche.
DIENSTAG ist der 2. Tag der Woche.
MITTWOCH ist der 3. Tag der Woche.
DONNERSTAG ist der 4. Tag der Woche.
FREITAG ist der 5. Tag der Woche.
SAMSTAG ist der 6. Tag der Woche.
SONNTAG ist der 7. Tag der Woche.
```

Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Klassifikation und Datentypen
- ✓ Variable
- ✓ Array-Typ und Aufzählungstyp
- ✓ Zeichenketten

StringBuffer

Wandlung von Datentypen

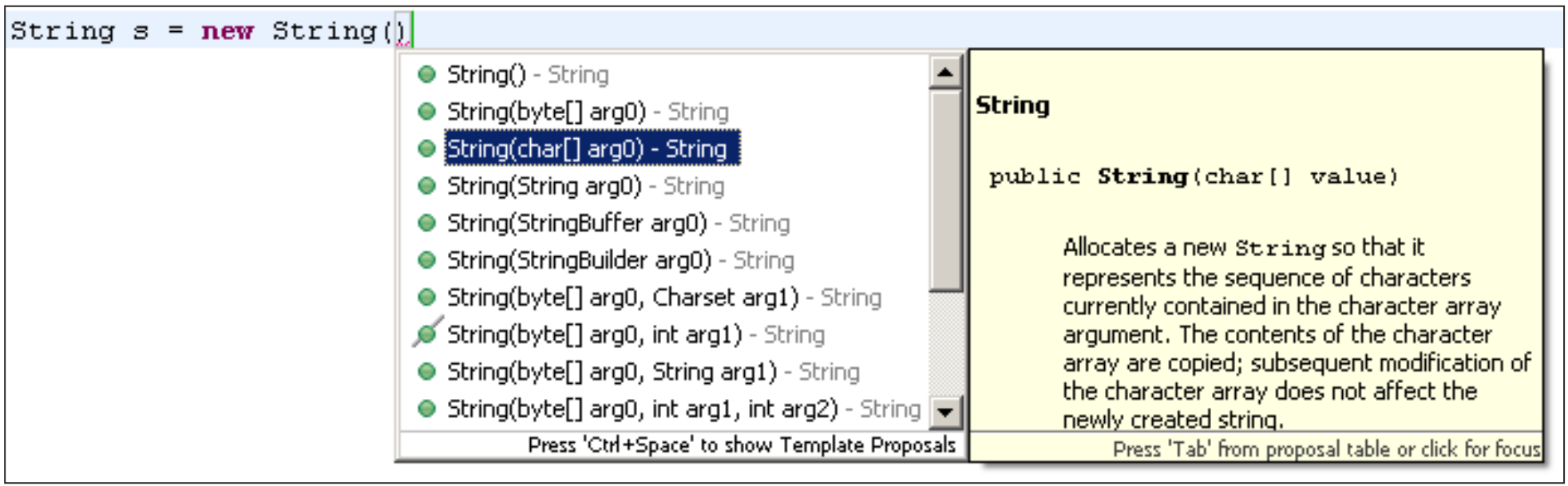
Zeichenketten I

- String sind Zeichenketten
- In Java sind Strings Objekte
- In Java gibt es drei verschiedene String-Klassen:
- String für konstante Zeichenketten
- StringBuffer und Stringbuilder für variable Zeichenketten (Im Rahmen der Vorlesung nur StringBuffer)
- Die Länge des Datenfelds vom Typ String lässt sich mit der Methode length() der Klasse String abfragen
- Erzeugung von Strings:
  - Mit dem *new*-Operator und Initialisierung mit einem Konstruktor
  - Implizites Erzeugen eines Objektes vom Typ String

Zeichenketten II

- Konstante Zeichenketten - String
- Erzeugung von String mit dem new-Operator und Initialisierung mit einem Konstruktor

```
String name1 = new String("Pia Alpha");
```



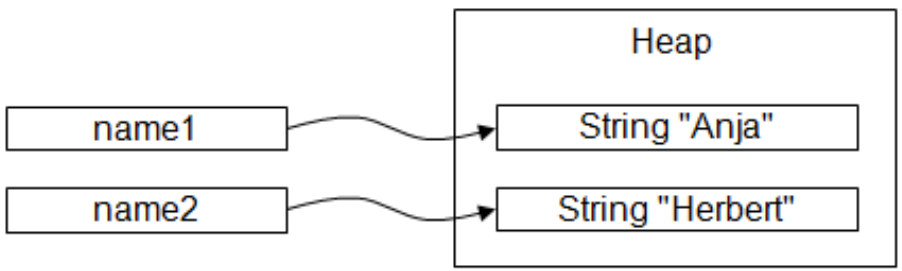
- Eclipse-Screenshot: Cursor zwischen die Klammern hinter new String positionieren und **Strg** + **Leertaste** drücken

Zeichenketten III

- Objekte vom Typ String können nicht abgeändert werden

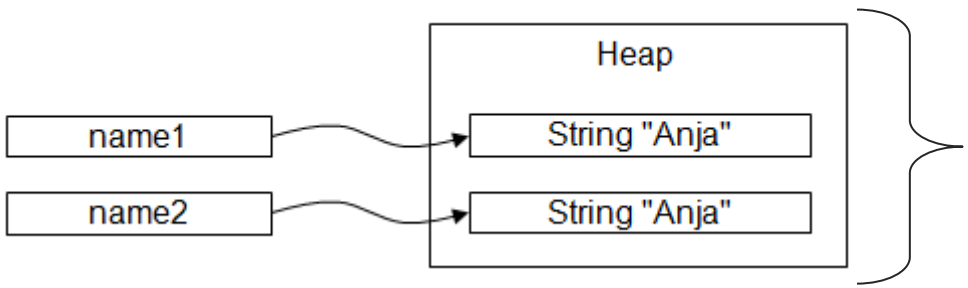
```
// ein anderer String-Konstruktor mit einem Array
von Zeichen
char[] data = {'A','d','a','m','a'};

String name = new String("Adama");
String gleicherName = new String(data);
```



Zeichenketten IV

- Vergleichen von Strings
- Würden zwei String Referenzvariablen mit Hilfe des == Operators verglichen werden, so würden die Werte, d.h. die Referenzen verglichen (Überprüfung, ob sie auf dasselbe Objekt zeigen)
  - Ob der Inhalt (also die Zeichenkette selbst) übereinstimmt, wird nicht geprüft
  - Würden zwei Zeichenketten, jeweils mit dem *new Operator* erzeugt, beim Vergleich mit == immer ungleich sein



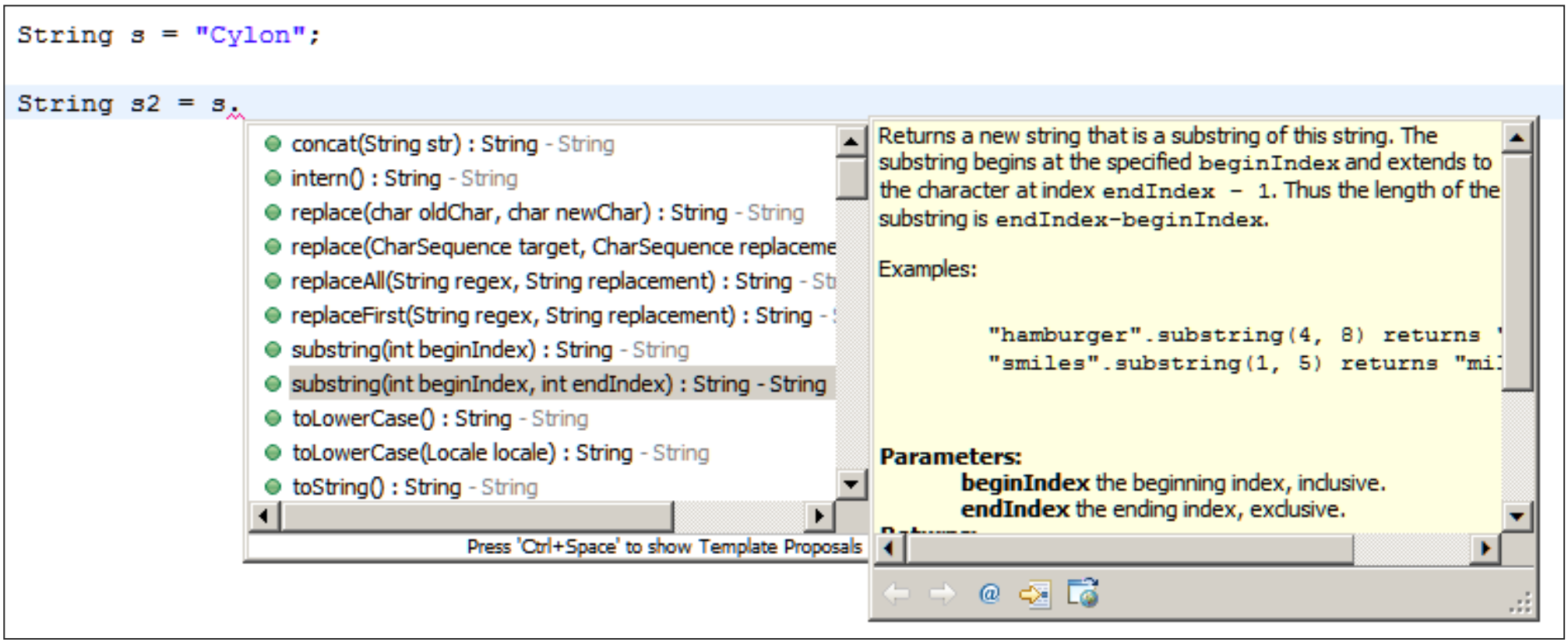
Die Zeichenketten sind gleich, aber es sind zwei unterschiedliche Objekte mit unterschiedlichen Referenzvariablen!

- Um den Inhalt zweier String-Objekte zu vergleichen, kommt die Methode equals() der Klasse String zum Einsatz

```
if(name1.equals(name2))…
```

Zeichenketten V

- Methoden der Klasse String
- eine Übersicht über alle String-Funktionen erhalten Sie in der Java-API oder in Eclipse
  - Variablenname + Punkt eingeben → dann erscheint die Autovervollständigung mit Vorschlägen



Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Klassifikation und Datentypen
- ✓ Variable
- ✓ Array-Typ und Aufzählungstyp
- ✓ Zeichenketten
- ✓ StringBuffer

Wandlung von Datentypen

StringBuffer I

- die Länge der Zeichenkette in einem StringBuffer-Objekt ist nicht festgelegt
- automatische Vergrößerung der Länge, wenn Zeichen angefügt werden
- der Inhalt eines Objektes lässt sich verändern

- Erzeugung eines StringBuffer-Objektes
  - ✓ Kann nicht implizit erzeugt werden!
  - ✓ Erzeugung ist nur mit dem new-Operator, also dem expliziten Aufruf eines seiner Konstruktoren möglich!
  - ✓ Es werden drei Konstruktoren besprochen:

StringBuffer()

StringBuffer(int length)

StringBuffer(String str)

StringBuffer II

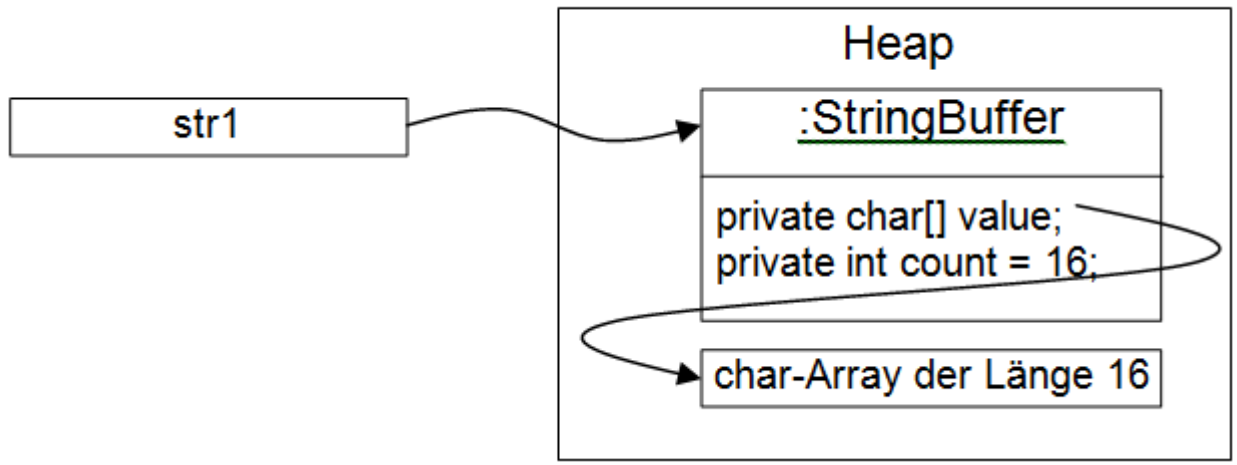
- Beim Aufruf des StringBuffer-Konstruktors wird „unsichtbar“ im Hintergrund ein Array vom Typ char angelegt, das die Zeichenkette repräsentiert

  - ✓ beim parameterlosen Konstruktor sieht das dann so aus

```
StringBuffer str1 = new StringBuffer();
```

  - ✓ Standardmäßig wird hierbei im Hintergrund ein char-Array der Größe 16 angelegt

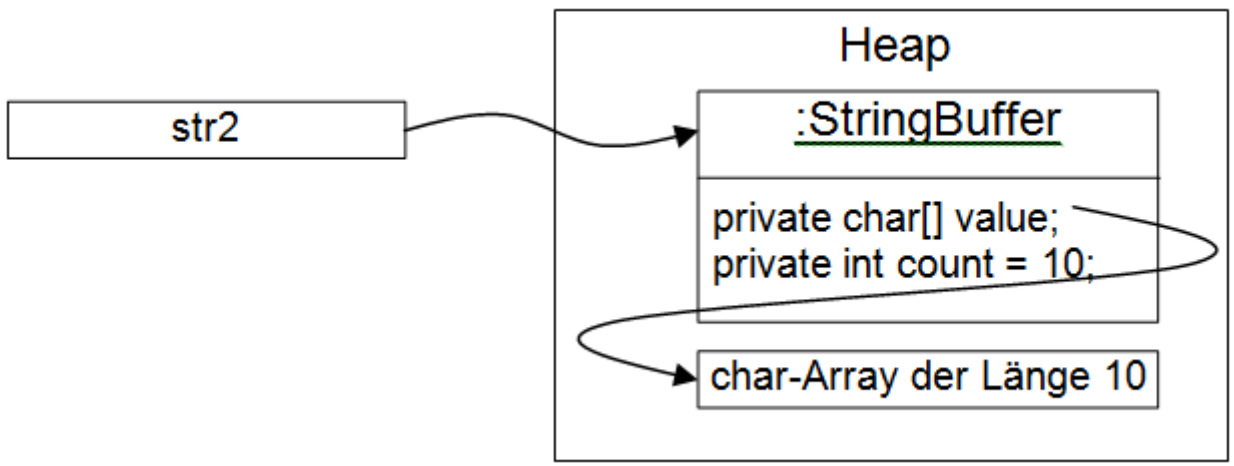
```
char[] value = new char[16];
```



StringBuffer III

- wird als Parameter des Konstruktors ein int-Wert übergeben, so wird ein StringBuffer-Objekt der Länge des übergebenen int-Wertes angelegt.

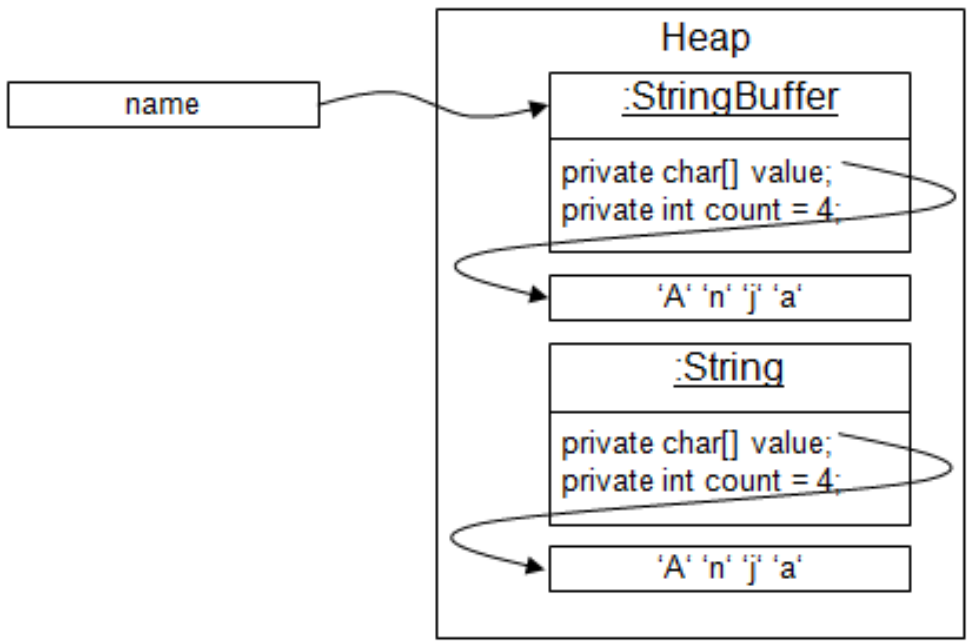
```
StringBuffer str2 = new StringBuffer(10);
```



StringBuffer IV

- wird als Parameter des Konstruktors eine konstante Zeichenkette übergeben, so wird das StringBuffer-Objekt damit initialisiert.

```
StringBuffer name = new StringBuffer("Anja");
```



StringBuffer Methoden

- Beispiele

  - public StringBuffer append(String s)
    - fügt die Zeichenkette s an den bisherigen Inhalt an

  - public StringBuffer insert(int offset, char[] str)
    - fügt die Zeichen aus dem char-Array in die bestehende Sequenz ab Position offset ein

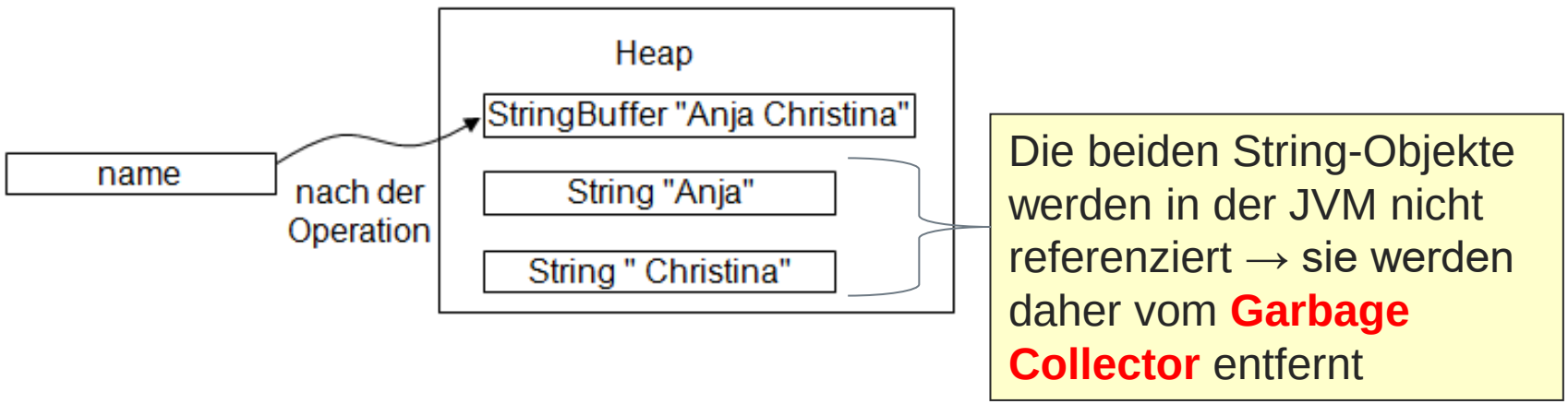
  - public StringBuffer delete(int start, int end)
    - entfernt die Teilzeichenkette zwischen Index start und end-1

  - public int indexOf(String str)
    - Liefert den Index des ersten Vorkommens von str innerhalb der Zeichenkette

StringBuffer Verkettung

- Verkettung von Strings und StringBuffern erfolgt mit der Methode append(String str)

```
StringBuffer name = new StringBuffer("Anja");
name.append(" Christina");
```

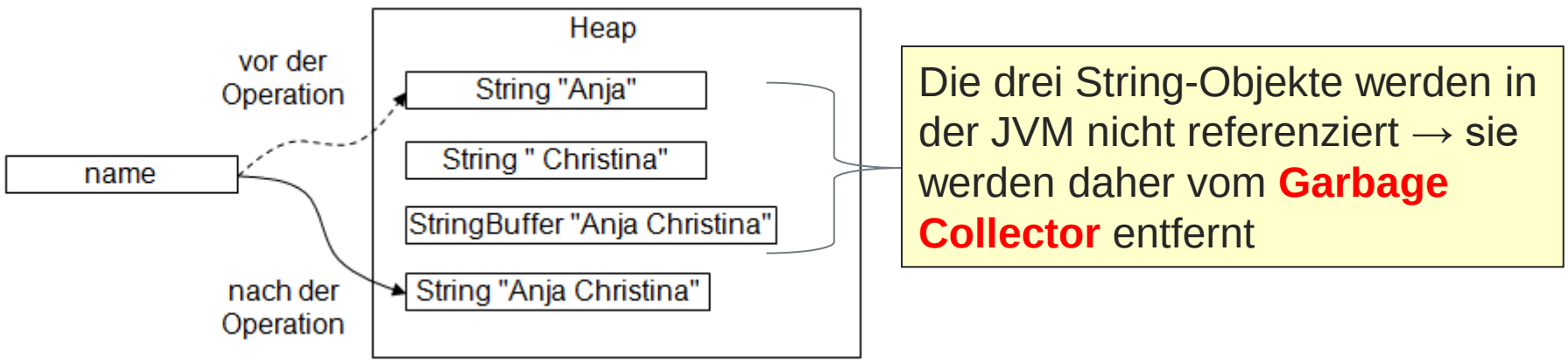


Verkettung von String

- Verkettung von Strings dies erfolgt mit dem Operator +

```
String name = "Anja";
name = name + " Christina";
```

- Was ist da denn los? String-Objekte können doch nicht verändert werden!
  - Ist auch nicht so. Die Verkettung erfolgt intern über einen StringBuffer.
  - Es wird ein neues String-Objekt erzeugt und der Referenzvariablen name zugewiesen



Übersicht

1. Grundbegriffe der Programmierung
2. Einfache Beispielprogramme
3. Datentypen und Variablen
4. Ausdrücke und Operatoren
5. Kontrollstrukturen
6. Blöcke und Methoden
7. Klassen und Objekte
8. Vererbung und Polymorphie
9. Pakete
10. Ausnahmebehandlung
11. Schnittstellen (Interfaces)
12. Geschachtelte Klassen
13. Ein-/Ausgabe und Streams
14. Applets / Oberflächenprogrammierung

Inhalte

- ✓ Klassifikation und Datentypen
- ✓ Variable
- ✓ Array-Typ und Aufzählungstyp
- ✓ Zeichenketten
- ✓ StringBuffer
- ✓ Wandlung von Datentypen

Wandlung von Datentypen I

- es geht um die Wandlung von Variablen eines einfachen Datentyps in Variablen eines Klassen-Typs und umgekehrt
- Wrapper-Klassen dienen dazu, ein nicht-objektorientiertes Konstrukt (z.B. einfacher Datentyp) in die Form einer Klasse einzubetten
- für alle einfachen Datentypen gibt es im Paket java.lang die folgenden Wrapper-Klassen:

Einfache Datentypen	Wrapper-Klassen
`Char`	`Character`
`Boolean`	`Boolean`
`Byte`	`Byte`
`Short`	`Short`
`Int`	`Integer`
`Long`	`Long`
`Double`	`Double`
`Float`	`Float`

Wandlung von Datentypen II

- diese Wrapper-Klassen stellen Methoden bereit, um die einfachen Datentypen zu bearbeiten
  - ✓ z.B. Methoden zur Umwandlung von String in einen Zahlenwert

```
String s1 = "100";
int summe = 200 + Integer.parseInt(s1);
```

Aufgabe 03.01

- Erzeugen Sie innerhalb der main-Methode zwei StringBuffer-Objekte mit den zugehörigen Referenzvariablen. Das erste Objekt (sb1) soll mit der Zeichenkette „Super Java!“ und das zweite (sb2) mit der Zeichenkette „30“ initialisiert werden.
- Legen Sie eine dritte Variable (sb3) vom Typ StringBuffer an und lassen Sie sie auf das erste StringBuffer-Objekt sb1 zeigen. Geben Sie den Inhalt von sb3 über die Systemausgabe aus.
- Definieren Sie eine int-Variable namens summe und weisen Sie ihr die Summe aus 20 und dem Integer-Wert aus dem zweiten StringBuffer-Objekt („30“) zu. Geben Sie das Ergebnis über die Systemausgabe aus.
- Erweitern Sie die Zeichenkette aus dem zweiten StringBuffer-Objekt sb2 um die Zeichenkette „ Grad warmes Wasser“. Lassen Sie nun die Referenzvariable sb3 auf das zweite StringBuffer-Objekt sb2 zeigen und geben Sie den Inhalt von sb3 auf der Systemausgabe aus.

Aufgabe 03.03

- Implementieren Sie ein zweidimensionales String-Array. Die erste Dimension enthält 3 und die zweite Dimension jeweils 2 Elemente. Befüllen Sie die Elemente mit Vornamen Ihrer Wahl.
- Geben Sie über die Systemausgabe von den ersten beiden Elementen den Anfangsbuchstaben gefolgt von einem Punkt aus.
- Geben Sie über die Systemausgabe von den zweiten beiden Elementen jeweils die Anzahl der Zeichen aus.
- Geben Sie über die Systemausgabe die dritten beiden Elemente komplett in Großbuchstaben aus.

- Verwenden Sie die Methoden der Klasse String (Schauen Sie bei Bedarf in die API oder verwenden Sie die Autovervollständigung von Eclipse).

Aufgabe 03.04

- Benutzen Sie die Methoden der Klasse String, um die Klasse Aufg_03_04 als Parser zu implementieren. Diese Klasse hat die Aufgabe, aus einem vollständigen Pfad in Form eines Strings das Verzeichnis, den Dateinamen und die Extension der Datei zu ermitteln. Lautet z.B. der gesamte Pfad:

```
C:\Eigene Dateien\Javatest\Beispiel.java
```

dann soll das Programm folgendes extrahieren:

```
Extension:          java
Dateiname:          Beispiel
Verzeichnis:        C:\Eigene Dateien\Javatest
```