

# **Java**

## **Entwicklung & Design: Gliederung**

Version: 06.04.2011

- **Entwicklungsgeschichte**
- **Java**
- **JavaScript**

# Java



- **Java ist eine**
  - leistungsfähige,
  - nicht-standardisierte und
  - plattform-unabhängige Erweiterung des WWW, die Interaktion ermöglicht.
- **Java wurde 1995 mit „Hot Java“ von SUN eingeführt.**
- **Mittlerweile von Oracle übernommen**

# Eigenschaften von Java - 1

**einfach**: Kommt ohne kritische  
Sprachelemente, z.B. Zeiger, aus

**leistungsfähig**: großer  
Sprachumfang

**objektorientiert**: Konzentration auf  
Daten/Methoden statt Prozeduren

# Eigenschaften von Java - 2

**sicher:** Erschwert Angriffe aufs System



**plattform-neutral:** Applikationen können auf versch. Rechnern eingesetzt werden.

**portabel:** Keine implementierungsabhängigen Sprachdefinitionen

**interpretiert:** Es wird ByteCode interpretiert

# Eigenschaften von Java - 3

**dynamisch:**

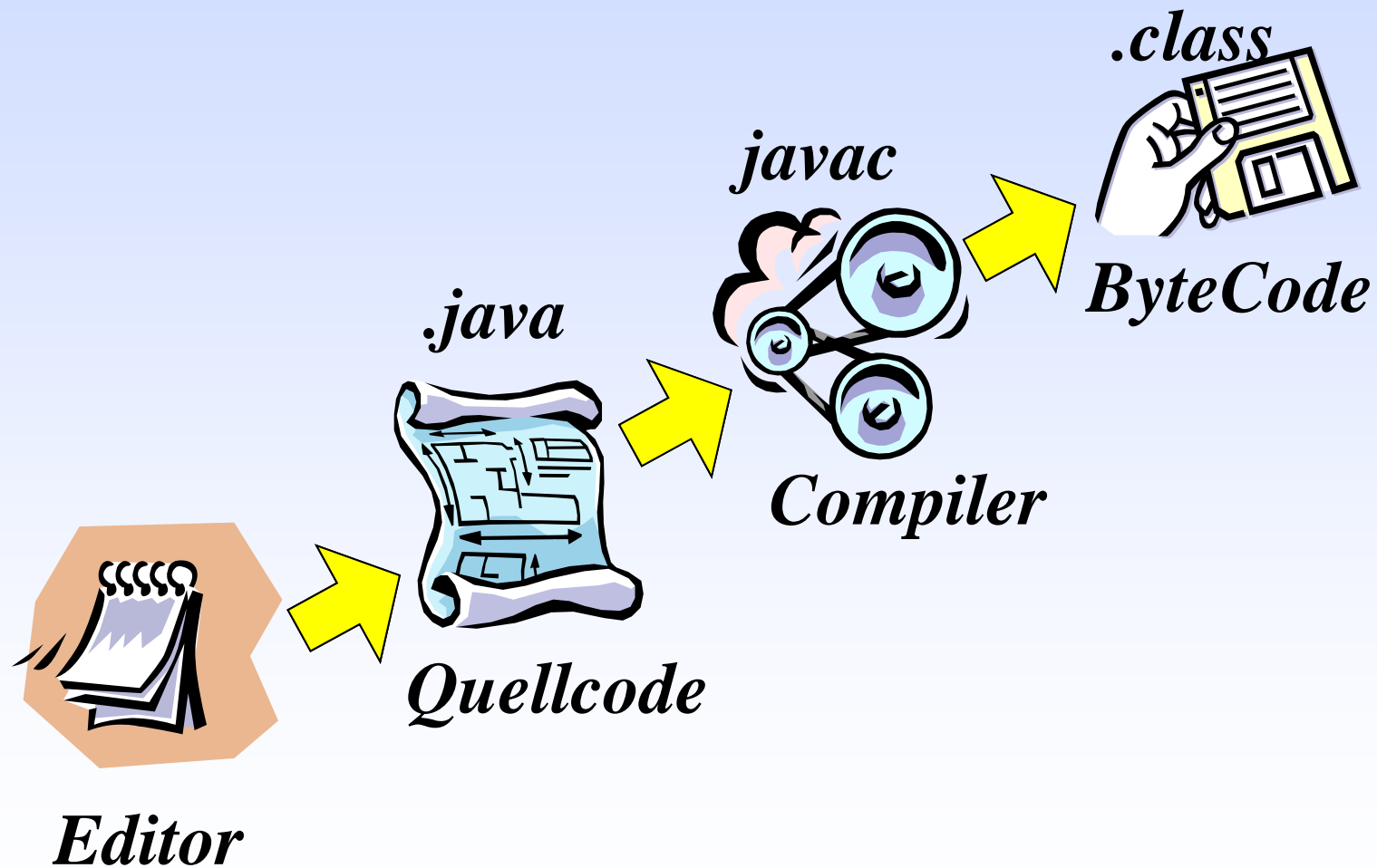
**Java lädt seine Klassen erst, wenn sie benötigt werden.**

**multi-threaded: Gleichzeitiges Ausführen mehrerer Threads (Kontrollflüsse)**

# **Unterschiede – JavaScript zu Java**

- **Eigenständiges Produkt**
- **Direkt in HTML-Dokumente eingebettet**
- **Läuft nur im Web-Browser**
- **Code wird nicht compiliert**
- **Einfach zu analysieren**
- **Einfacher konzipiert**
- **Nur objektbasierte Skriptsprache**

# Der Prozess der Programmerstellung



# Java-Quellcode erzeugen - erstes Beispiel

Als Werkzeug genügt ein ASCII-Editor

```
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello, world");
    }
}
```



# ByteCode des Beispiels

**Ëp<sup>03/4</sup> - ()V**  
**(Ljava/lang/String;)V ([Ljava/lang/String;)V**  
**<init> Code**  
**ConstantValue Exceptions Hello, world**  
**HelloWorld HelloWorld.java**  
**LineNumberTable Ljava/io/PrintStream;**  
**LocalVariables SourceFile**  
**java/io/PrintStream java/lang/Object**  
**java/lang/System main out println**

# Entwicklungsziele

- ☑ ***Java Application*: Programm ist eine eigenständige Anwendung.**
- ☑ ***Java-Applet*: Ausführung erfolgt über einen Browser.**
- ☑ ***JavaBeans*: Implementieren Methoden und Eigenschaften.**

# **Bestandteile des Java Development Kit (JDK)**

**Compiler (javac) - Interpreter (java)**

**Debugger (jdb) - Disassembler (javap)**

**Schnittstellengenerator (javah)**

**Programmdokumentation (javadoc)**

**Ausführung von Applets (appletviewer)**

# Der Java Compiler - javac

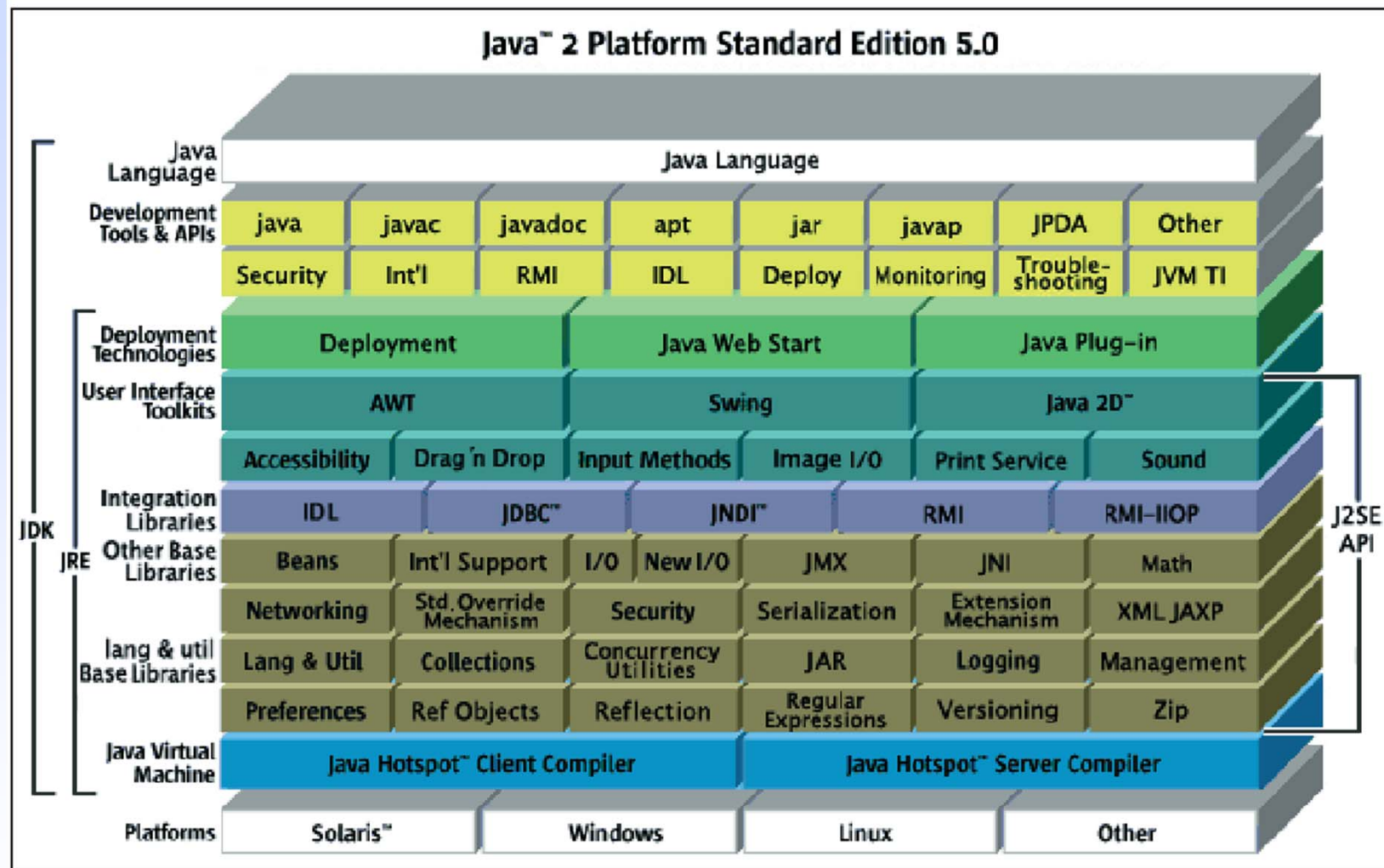
- **c:\>javac [<Optionen>] <Name>.java**
- **Acht Optionen, z.B.**
  - g** erzeugt Debug-Informationen**c:\>javac -g Beispiel.java**
- **Für jede deklarierte Klasse erzeugt der Compiler eine Datei mit dem Namen <Klassenname>.class**

# Der Java Interpreter - java

- **c:\>java [<Optionen>] <Klassenname>  
<Argumentliste>**
- **Klasse muß Methode *main()* haben:  
public static void main(String args[])**
- **c:\>java Beispiel 5 10000**
- **16 Optionen, z.B.  
-help      Anzeigen der Optionen**

# **Integrierte Entwicklungsumgebungen (IDE)**

- **NetBeans (Sun)**
- **JBuilder (Borland)**
- **Eclipse (Opensource, vormals IBM)**
- **Grundlage ist immer JDK (Java Development Kit)**



Das Java Development Kit (JDK) mit seinen Komponenten im Überblick

# Pakete in API 1

## Einbinden mit import ...

- package java.io: Ein- und Ausgabe
- package java.lang: Basisklassen
- package java.math: lange Zahlen
- package java.text: Textverarbeitung
- package java.util: Nützliches
- ...



## Pakete in API 2

- package java.awt: **graphisches User Interface**
- package java.applet: **Internet-Anwendungen**
- package java.sql: **Datenbank-Anbindung**
- package java.beans: **Erweiterungen**

# Paketname muß vorangestellt werden

```
...  
double sqrt2 = Math.sqrt(2.0);  
...  
  
System.out.println(sqrt2);  
...
```

---

```
> java sqrt2  
1.4142135623730951
```

# Das erste Standalone-Beispiel

```
class MeinErstes
// Zeile 1
{
public static void main(String args [])
{
System.out.print("\nMit print und println
kann");
System.out.println(" man \nschreiben und

rechnen:");
System.out.println("6 * 7 = \t" + 6 * 7);
//Zeile 7
} }
```

# Die Bildschirmausgabe

```
C:\>javac MeinErstes.java
```

```
C:\>java MeinErstes
```

Mit `print` und `println` kann man  
schreiben und rechnen:

```
6 * 7 =          42
```

# Escape-Sequenzen & arithm. Operatoren

*Arithmetische Operatoren* sind:

<b>+</b>	<b>Addition</b>	<b>-</b>	<b>Subtraktion</b>
<b>*</b>	<b>Multiplikation</b>	<b>/</b>	<b>Division</b>
<b>%</b>	<b>Modulo</b>		

*Esc-Sequenzen* dienen der Textsteuerung

<b>\b</b>	<b>backspace</b>	<b>\t</b>	<b>horizontal tab</b>
<b>\n</b>	<b>line feed</b>	<b>\f</b>	<b>form feed</b>
<b>\r</b>	<b>carriage return</b>	<b>\“</b>	<b>Anführungszeichen</b>

## **Variablendefinition und -bezeichnung**

**byte anzahlKinder;**

**int Wert;**

**float messWert;**

**float kontostand;**

## Gleitkommazahlen-Datentypen

<u>Datentyp</u>	<u>Bits</u>	<u>Wertebereich</u>
float	32	- $3,4..*10^{38}$ + $3,4..*10^{38}$
double	64	- $1,7..*10^{308}$ + $1,7..*10^{308}$

## Ganze Zahlen-Datentypen

<u>Datentyp</u>	<u>Byte</u>	<u>Wertebereich</u>
<b>byte</b>	<b>1</b>	<b>- 127 bis +128</b>
<b>short</b>	<b>2</b>	<b>-32767 bis +32768</b>
<b>int</b>	<b>4</b>	<b>-2.147.483.657 bis +2.157.583.648</b>
<b>long</b>	<b>8</b>	<b>-9,22*10<sup>18</sup> bis +9,223*10<sup>18</sup></b>



## Unicode

<u>Zeichen</u>	<u>Code</u>	<u>Charts</u>
A	00 41	Basic Latin
Ä	00 C4	Latin-1 Supplement
α	03 B1	Greek

## **Zeichen-Datentypen**

<b><u>Datentyp</u></b>	<b><u>Bits</u></b>	<b><u>Wertebereich</u></b>
<b>char</b>	<b>16</b>	<b>65536 Zeichen</b>
<b>boolean</b>	<b>1</b>	<b>true / false</b>

## Typenkonvertierung - Cast

```
byte anzahlDerKinder;
```

```
float faktor;
```

```
...
```

```
anzahlDerKinder = 3;
```

```
faktor = anzahlDerKinder;
```

```
...
```

```
faktor = 4;
```

```
anzahlDerKinder = (byte)faktor;
```

```
...
```

```
faktor = (float) 2.999; //double
```

```
ohne cast
```

```
anzahlDerKinder = (byte)faktor;
```

```
...
```

## Felder

z.B.:

```
int jahr [] = new int[9];  
{  
    jahr[0] = 1996;  
    jahr[2] = 1910;  
    jahr[4] = 2000;  
}
```

# **Konstanten**

**Mit Vorsatz final**

```
final int MAXWERT=100;
```

# Namenskonventionen

In Java gibt es folgende Namenskonventionen:

- **Symbolische Namen** werden vollständig groß geschrieben (z. B. `PI`)
- **Bezeichner für Klassen** beginnen mit einem großen Buchstaben und werden dann klein geschrieben. Setzt sich der Name aus mehreren Bestandteilen zusammen, beginnt jeder Namensbestandteil mit einem großen Buchstaben (z. B. `MeineKlasse`)
- **Bezeichner für Variablen, Methoden und Elemente** beginnen mit einem kleinen Buchstaben. Setzt sich der Name aus mehreren Bestandteilen zusammen, beginnt jeder nachfolgende Namensbestandteil mit einem großen Buchstaben (z. B. `ersteVariable`)

# **Aufgabe**

## **Datentypen, Felder und Cast-Anweisung**

# **Kontrollstrukturen**

- **Bedingte-Anweisung**
- **for-Schleife**
- **while-Schleife**



## **Kontrollfluss -Java-Schlüsselwörter**

**z.B.:**

**if**

**else**

**case**

**while**

**break**

**return**

**...**

## Bedingte Anweisung

- **if (<logischeBedingung>)**  
    **{ <Anweisung-1> }**  
    **[else**  
        **{ <Anweisung-2>} ]**
- **if (i > 5)**  
    **{ System.out.println("i > 5"); }**  
    **else**  
        **{ System.out.println("i <= 5"); }**

# Vergleichs- und logische Operatoren

## Vergleichsoperatoren

<	kleiner als
>	größer als
<=	kleiner gleich
>=	größer gleich
==	gleich
!=	ungleich

## Log. Operatoren

&&	UND
	ODER
!	NICHT

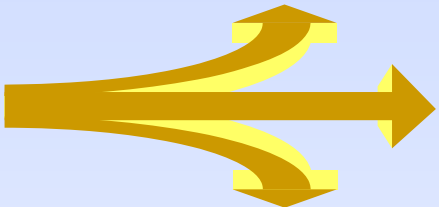
*Zum Beispiel:*

*if (x==2 && y<3)*

## Abfragenkette

```
int z=4;  
if (z == 1)  
{ System.out.println("z=1"); }  
else if (z == 2)  
{ System.out.println("z=2"); }  
else  
{ System.out.println("z ist <1  
oder >2"); }
```

# Switch-Anweisung

```
switch (<Ausdruck>)   
{  
  case <konstanter Ausdruck>:  
    <Anweisung>  
  case <konstanter Ausdruck>:  
    <Anweisung>  
  ...  
  default:  
    <Anweisung> }
```

# Switchtest-Quellcode

```
class Switchtest
{ public static void main(String args[])
{   int i;
    for (i=0;i<11;i=i+1)
    {
        switch(i)
        { case 0: case 2: case 4: case 2*3: case 8:
            System.out.println("Gerade Zahl");
            break;
          case 1: case 3: case 5: case 7: case 9:
            System.out.println("Ungerade Zahl");
            break;
          default: System.out.println(">9  !"); }
    }
}
```

# for-Schleife

- **Syntax:**

```
for (startwert;  
    [<logischerAusdruck>];  
    schleifenwert)  
    { <Anweisung> }
```

- **schleifenwert** verwendet oft

**Inkrementoperatoren:** `i++`, `i = i+1`

**Dekrementoperatoren:** `i--`, `i = i-1`

## for-Schleife-Ergänzungen

- `for (int i=10; i>=1; i--)`
- `++i` Operator angewandt, bevor der Wert des Ausdrucks zurückgegeben wird  
Beispiel: `int i=4711;`  
`System.out.print (i++ + "`  
`" + ++i);`
- Ergebnis: 4711 4713



# Quellcode

```
class Geschachtelt
{ public static void main(String args[])
{
    for (int i=1; i<=5; i++)
    {
        for (int j=1; j<=5; j++)
        { System.out.print(i*j + "\t"); }
        System.out.println();
    }
}
}
```

## Beispiel-Ausgabe

<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>2</b>	<b>4</b>	<b>6</b>	<b>8</b>	<b>10</b>
<b>3</b>	<b>6</b>	<b>9</b>	<b>12</b>	<b>15</b>
<b>4</b>	<b>8</b>	<b>12</b>	<b>16</b>	<b>20</b>
<b>5</b>	<b>10</b>	<b>15</b>	<b>20</b>	<b>25</b>

# ***while*-Schleife**

**Syntax:**

```
while ([<logischerAusdruck>])  
{  
    <Anweisung>  
}
```

## *do-while*-Schleife

**Syntax:**

```
do  
{  
    <Anweisung>  
} while ([<logischerAusdruck>])
```

# Breaktest - Quellcode

```
class Breaktest
{
    public static void main(String args[])
    {
        for (int i=1; i<100 000; i++)
        {
            System.out.println("Schleife: " +i);
            if (i>5)
                break;
        }
    }
}
```



# **Aufgabe**

## **Berechnung von Pi**

# Zeichenketten

- **Zeichenketten (Strings) sind Objekte**
- **Zwei Klassen von Strings:**
  - **String:**  
nur einfache Operationen, z. B. + zum Verbinden von Strings
  - **StringBuffer:**  
vielfältige Operationen

# Strings

- *Strings definieren:*

```
String nName = new String („Maier");  
String vName = „Fritz";
```

- *Strings verbinden:*

```
System.out.print(vName+", "+nName);
```

- Methode *„concat“* verbindet Strings:

```
nName=nName.concat(vName);
```

- Methode *„length“* liefert

Stringlänge:

```
System.out.print(nName.length());
```



# Umwandlung von Strings

- Methode „valueOf“ :  
z.B.  

```
String test = " ";  
boolean bool_var =  
true;  
test =  
String.valueOf(bool_var);
```
- test enthielte dann Zeichen „true“

# Umwandlung der Datentypen

```
test = "false";  
bool_var=Boolean.valueOf(test).booleanValue();
```

```
test="19";  
i=Int.valueOf(test).intValue();  
i=Integer.parseInt(test);
```

```
test ="8423.90321";  
double_var=Double.valueOf(test).doubleValue();
```

# Umwandlung der Datentypen

- **Erzeugen einer Zahl aus einem String**
  - `int i = Integer.parseInt("12345");`
  - `short h = Short.parseShort("12345");`
  - `long l = Long.parseLong("12345");`
  - `float f = Float.parseFloat("3.14159");`
  - `double d = Double.parseDouble("3.14159");`
- **Erzeugen eines Strings aus x-Wert**
  - `String s = String.valueOf(x);`
  - **Typ von x** kann sein
  - `char`, `int`, `long`, `float`, `double`, `boolean` oder `char[]`
- **Erzeugen eines char-Arrays mit dem Inhalt des Strings s**
  - `char[] a = s.toCharArray();`

# Stringbuffer - Quellcode

```
class StringBuffer
{ public static void main(String args[])
{
StringBuffer puffer = new StringBuffer();
puffer.append("Andreas ").append("Maier");
puffer.insert(8,"Fritz ");
System.out.println(puffer);
} }
```

# **Stringbuffer - Ergebnis**

```
c:\>javac stringbuffer.java
```

```
c:\>java stringbuffer
```

**Andreas Fritz Maier**