- Das Programm entnimmt der Kommando-Zeile den Datei-Pfad sowie eine Zahl `n`.
- Es wird ein `DataOutputStream` für die Datei eröffnet.
- In die Datei werden die Zahlen 0,...,`n-1` binär geschrieben.
- Das sind also 4`n` Bytes.

Achtung:

- In der Klasse `System` sind die zwei vordefinierten Ausgabe-Ströme `out` und `err` enthalten.
- `out` repräsentiert die Ausgabe auf dem Terminal.
- `err` repräsentiert die Fehler-Ausgabe für ein Programm (i.a. ebenfalls auf dem Terminal).
- Diese sollen jedes Objekt als Text auszugeben können.
- Dazu dient die Klasse `PrintStream`.

- Die `public`-Objekt-Methoden `print()` und `println()` gibt es für jeden möglichen Argument-Typ.
- Für (Programmierer-definierte) Klassen wird dabei auf die `toString()`-Methode zurückgegriffen.
- `println()` unterscheidet sich von `print`, indem nach Ausgabe des Arguments eine neue Zeile begonnen wird.

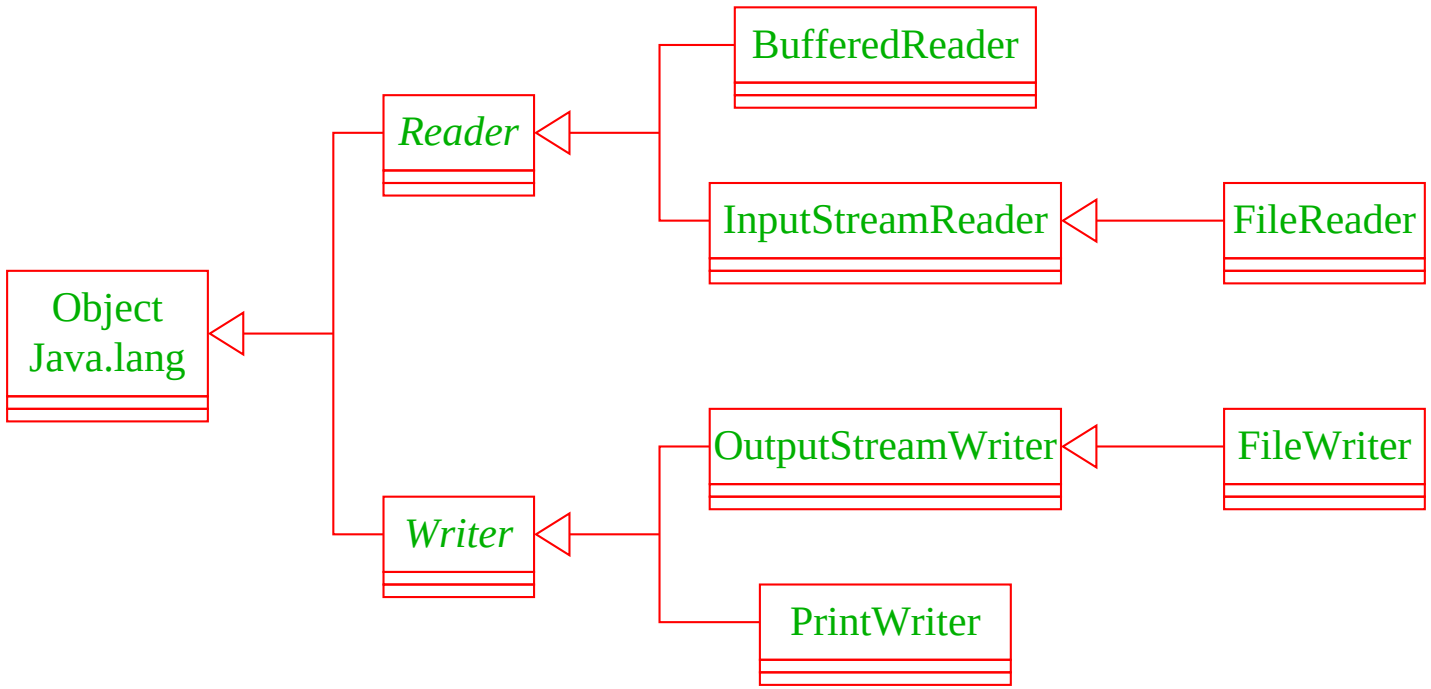
Achtung:

- `PrintStream` benutzt (neuerdings) die Standard-Codierung des Systems, d.h. bei uns Latin-1.
- Um aus einem (Unicode-)String eine Folge von sichtbaren (Latin-1-)Zeichen zu gewinnen, wird bei jedem Zeichen, das kein Latin-1-Zeichen darstellt, ein '?' gedruckt ...

$\Longrightarrow$ für echte (Unicode-) Text-Manipulation schlecht geeignet ...

15.2 Textuelle Ein- und Ausgabe

Wieder erstmal eine Übersicht über hier nützliche Klassen ...



- Die Klasse `Reader` ist abstrakt.

Wichtige Operationen:

- `public boolean ready() throws IOException` :
  liefert `true`, sofern ein Zeichen gelesen werden kann;
- `public int read() throws IOException` :
  liest ein (Unicode-)Zeichen vom Input als `int`-Wert; ist das Ende des Stroms erreicht, wird -1 geliefert;
- `void close() throws IOException` : schließt den Eingabe-Strom.

- Ein `InputStreamReader` ist ein spezieller Textleser für Eingabe-Ströme.
- Ein `FileReader` gestattet, aus einer Datei Text zu lesen.
- Ein `BufferedReader` ist ein `Reader`, der Text zeilenweise lesen kann, d.h. eine zusätzliche Methode `public String readLine() throws IOException;` zur Verfügung stellt.

Konstruktoren:

- `public InputStreamReader(InputStream in);`
- `public InputStreamReader(InputStream in, String encoding) throws IOException;`
- `public FileReader(String path) throws IOException;`
- `public BufferedReader(Reader in);`

Beispiel:

```
import java.io.*;
public class CountLines {
public static void main(String[] args) throws IOException {
   FileReader file = new FileReader(args[0]);
   BufferedReader buff = new BufferedReader(file);
   int n=0; while(null != buff.readLine()) n++;
   buff.close();
   System.out.print("Number of Lines:\t\t"+ n);
   } // end of main
} // end of CountLines
```

- Die Objekt-Methode `readLine()` liefert `null`, wenn beim Lesen das Ende der Datei erreicht wurde.
- Das Programm zählt die Anzahl der Zeilen einer Datei :-)

- Wieder stehen analoge Klassen für die Ausgabe zur Verfügung.
- Die grundlegende Klasse für textuelle Ausgabe heißt `Writer`.

Nützliche Operationen:

- `public void write(` type `x) throws IOException` :
  gibt es für die Typen `int` (dann wird ein einzelnes Zeichen geschrieben), `char[]` sowie `String`.
- `void flush() throws IOException` :
  falls die Ausgabe gepuffert wurde, soll sie nun tatsächlich ausgegeben werden;
- `void close() throws IOException` : schließt den Ausgabe-Strom.

- Weil `Writer` abstrakt ist, gibt keine Objekte der Klasse `Writer`, nur Objekte von Unterklassen.

Konstruktoren:

- `public OutputStreamWriter(OutputStream str);`
- `public OutputStreamWriter(OutputStream str, String encoding);`
- `public FileWriter(String path) throws IOException;`
- `public FileWriter(String path, boolean append) throws IOException;`
- `public PrintWriter(OutputStream out);`
- `public PrintWriter(Writer out);`

Beispiel:

```
import java.io.*;
public class Text2TextCopy {
public static void main(String[] args) throws IOException {
   FileReader fileIn = new FileReader(args[0]);
   FileWriter fileOut = new FileWriter(args[1]);
   int c = fileIn.read();
   for(; c!=-1; c=fileIn.read())
      fileOut.write(c);
   fileIn.close(); fileOut.close();
   System.out.print("\t\tDone!!!\n");
   } // end of main
} // end of Text2TextCopy
```

Wichtig:

- Ohne einen Aufruf von `flush()` oder `close()` kann es passieren, dass das Programm beendet wird, bevor die Ausgabe in die Datei geschrieben wurde :-(
- Zum Vergleich: in der Klasse `OutputStream` wird `flush()` automatisch nach jedem Zeichen aufgerufen, das ein Zeilenende markiert.

Bleibt, die zusätzlichen Objekt-Methoden für einen `PrintWriter` aufzulisten...

- Analog zum `PrintStream` sind dies
  `public void print( type x);` und
  `public void println( type x);`

- ... die es gestatten, Werte jeglichen Typs (aber nun evt. in geeigneter Codierung) auszudrucken.

Text-Ein- und Ausgabe arbeitet mit (Folgen von) Zeichen.

Nicht schlecht, wenn man dafür die Klasse `String` etwas näher kennen würde ...

16 Hashing und die Klasse `String`

- Die Klasse `String` stellt Wörter von (Unicode-) Zeichen dar.
- Objekte dieser Klasse sind stets konstant, d.h. können nicht verändert werden.
- Veränderbare Wörter stellt die Klasse ↑`StringBuffer` zur Verfügung.

Beispiel:

```
String str = "abc";
```

... ist äquivalent zu:

```
char[] data = new char[] {'a', 'b', 'c'};
String str = new String(data);
```

Weitere Beispiele:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc"+cde);
String c = "abc".substring(2,3);
String d = cde.substring(1,2);
```

- Die Klasse `String` stellt Methoden zur Verfügung, um
  - einzelne Zeichen oder Teilfolgen zu untersuchen,
  - Wörter zu vergleichen,
  - neue Kopien von Wörtern zu erzeugen, die etwa nur aus Klein- (oder Groß-) Buchstaben bestehen.
- Für jede Klasse gibt es eine Methode `String toString()`, die eine `String`-Darstellung liefert.
- Der Konkatenations-Operator "+" ist mithilfe der Methode `append()` der Klasse `StringBuffer` implementiert.

Einige Konstruktoren:

- `String();`
- `String(char[] value);`
- `String(String s);`
- `String(StringBuffer buffer);`

Einige Objekt-Methoden:

- `char charAt(int index);`
- `int compareTo(Object obj);`
- `int compareTo(String anotherString);`
- `boolean equals(Object obj);`
- `String intern();`
- `int indexOf(int chr);`
- `int indexOf(int chr, int fromIndex);`
- `int lastIndexOf(int chr);`
- `int lastIndexOf(int chr, int fromIndex);`

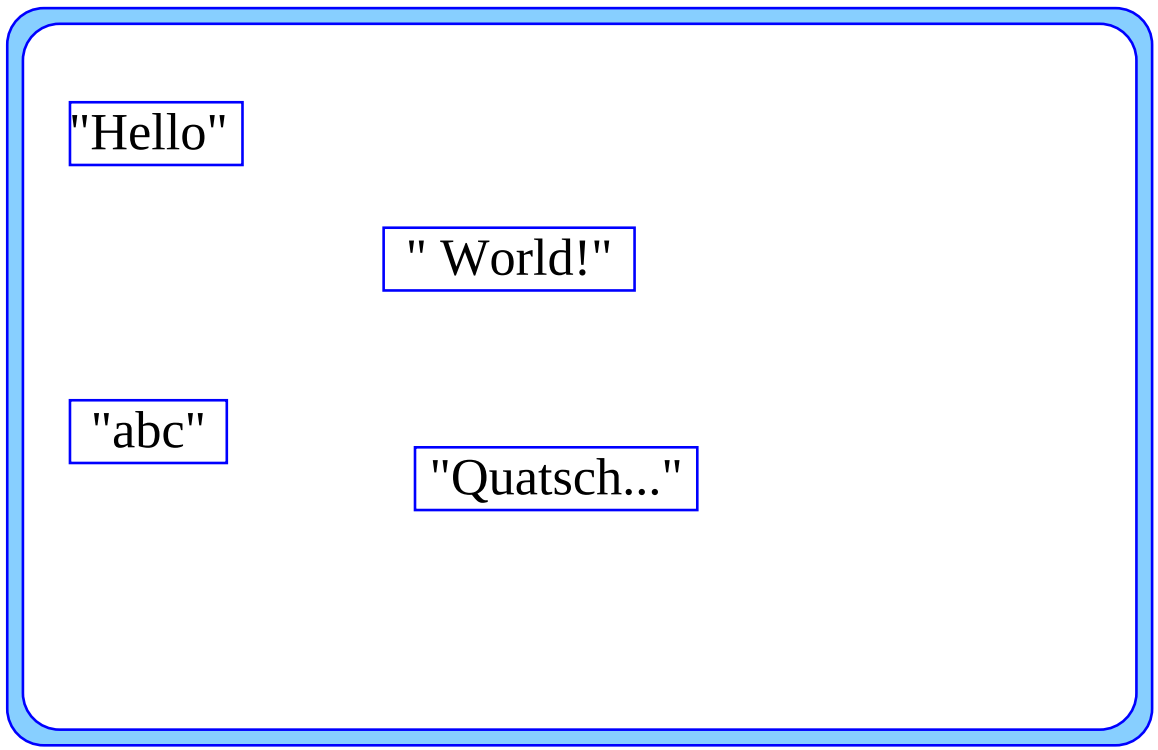
... weitere Objekt-Methoden:

- `int length();`
- `String replace(char oldChar, char newChar);`
- `String substring(int beginIndex);`
- `String substring(int beginIndex, int endIndex);`
- `char[] toCharArray();`
- `String toLowerCase();`
- `String toUpperCase();`
- `String trim();` : beseitigt White Space am Anfang und Ende des Worts.

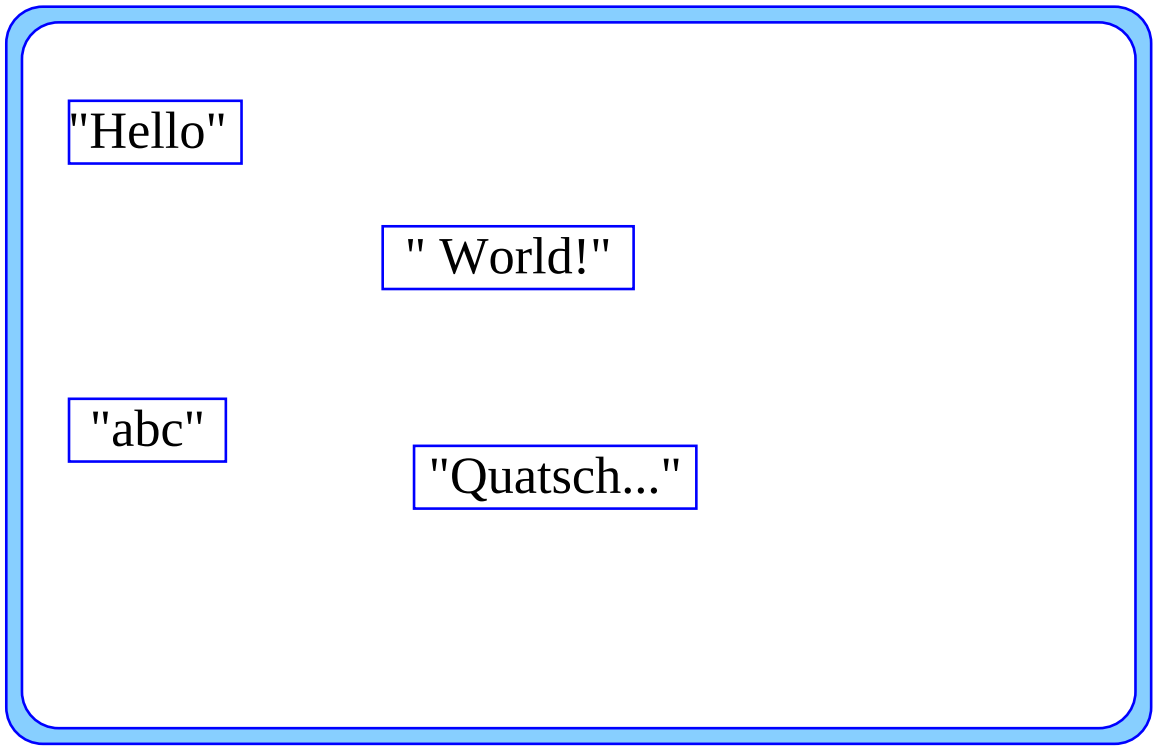
... sowie viele weitere :-)

Zur Objekt-Methode `intern()`:

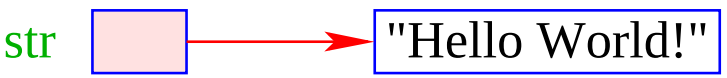
- Die Java-Klasse `String` verwaltet einen privaten String-Pool:

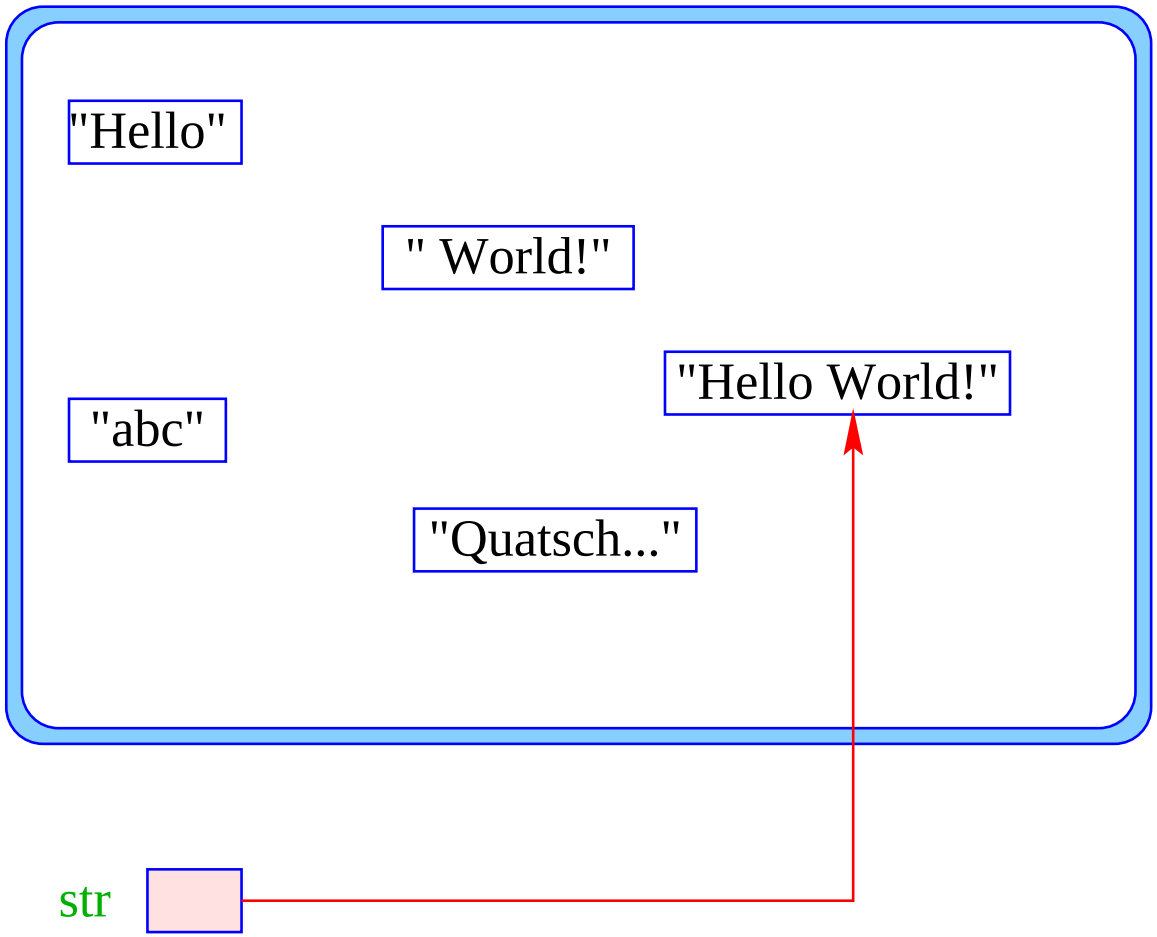


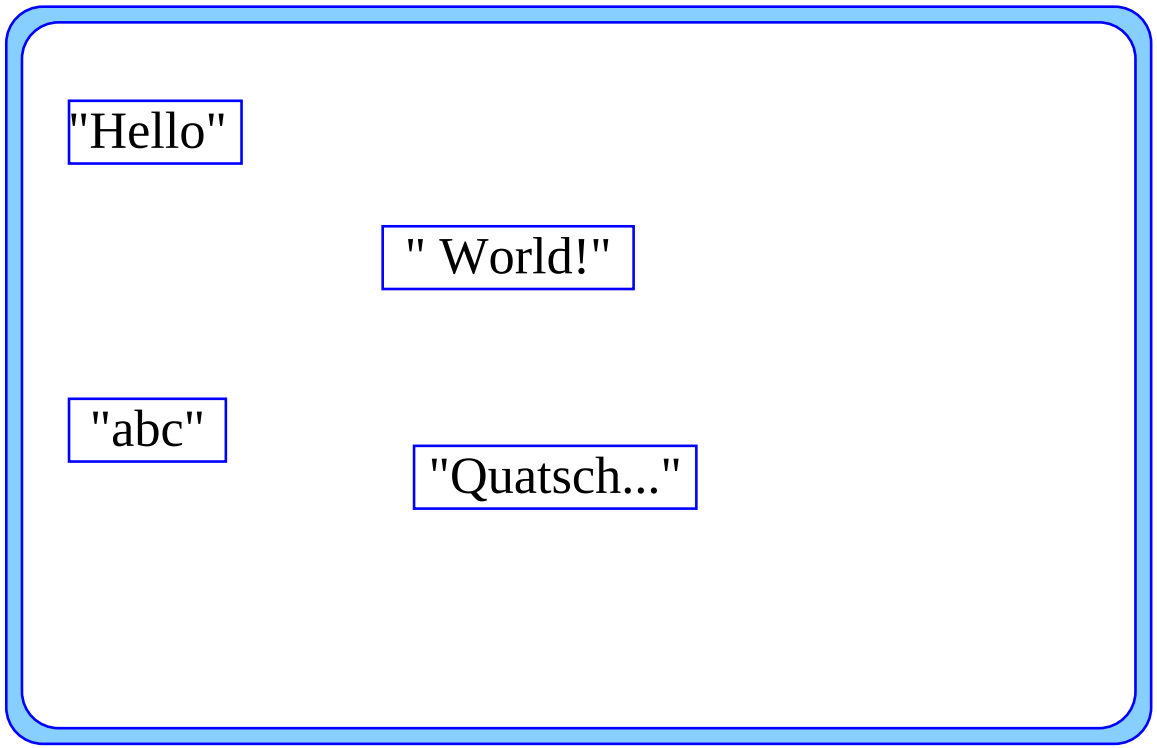
- Alle `String`-Konstanten des Programms werden automatisch in den Pool eingetragen.
- `s.intern();` überprüft, ob die gleiche Zeichenfolge wie `s` bereits im Pool ist.
- Ist dies der Fall, wird ein Verweis auf das Pool-Objekt zurück gegeben.
- Andernfalls wird `s` in den Pool eingetragen und `s` zurück geliefert.



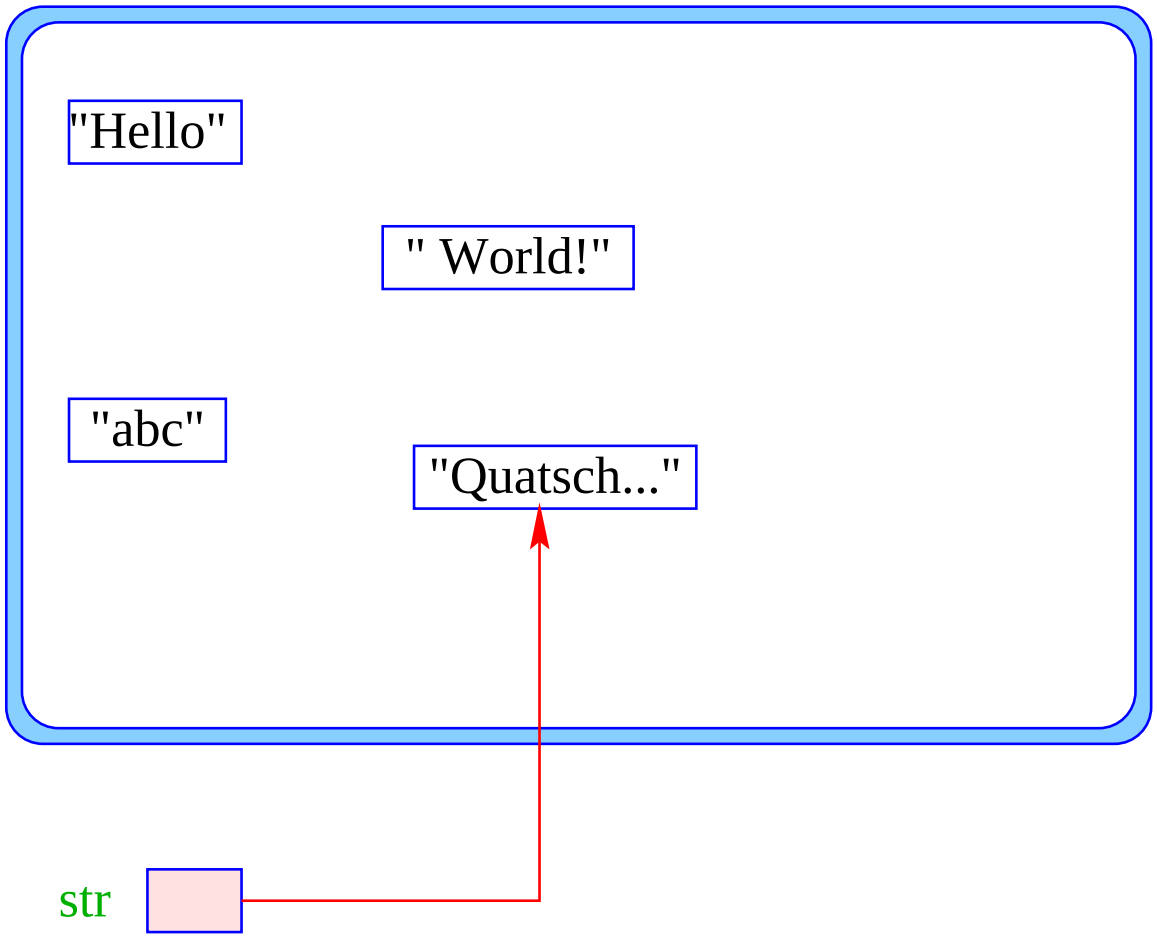

```
str = str.intern();
```




"Hello"
" World!"
"Hello World!"
"abc"
"Quatsch..."
str




```
str = str.intern();
```


"Hello"
" World!"
"abc"
"Quatsch..."
str