

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);`

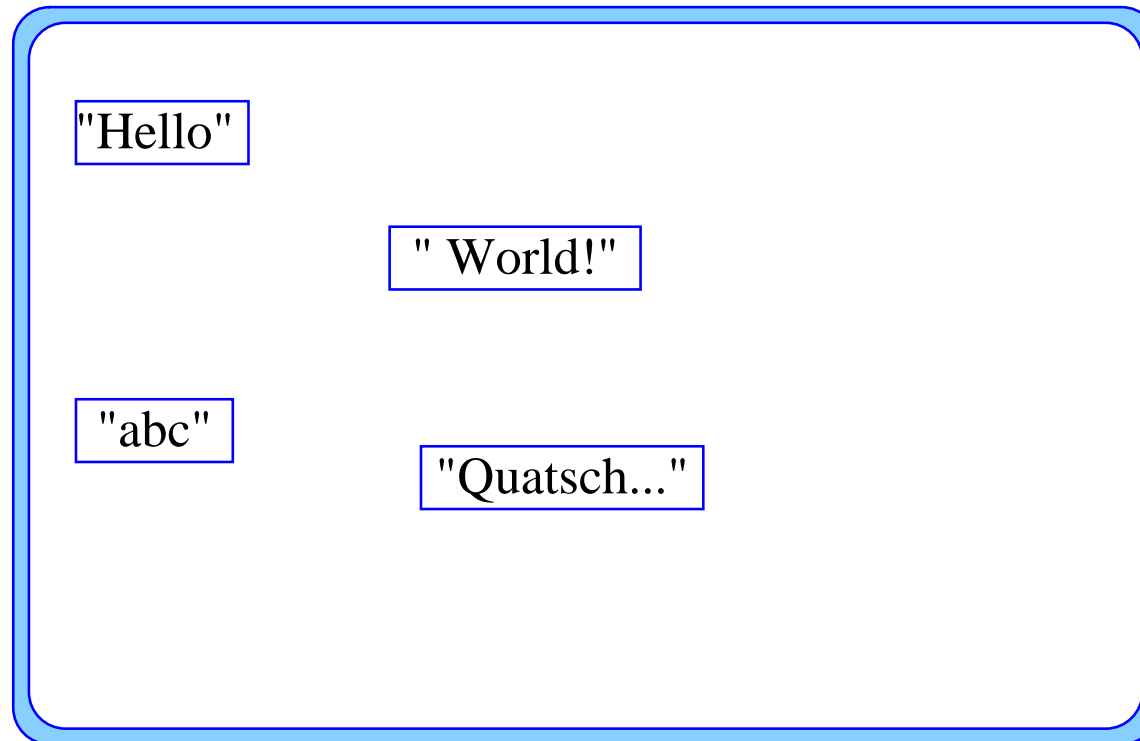
... weitere Objekt-Methoden:

- `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.

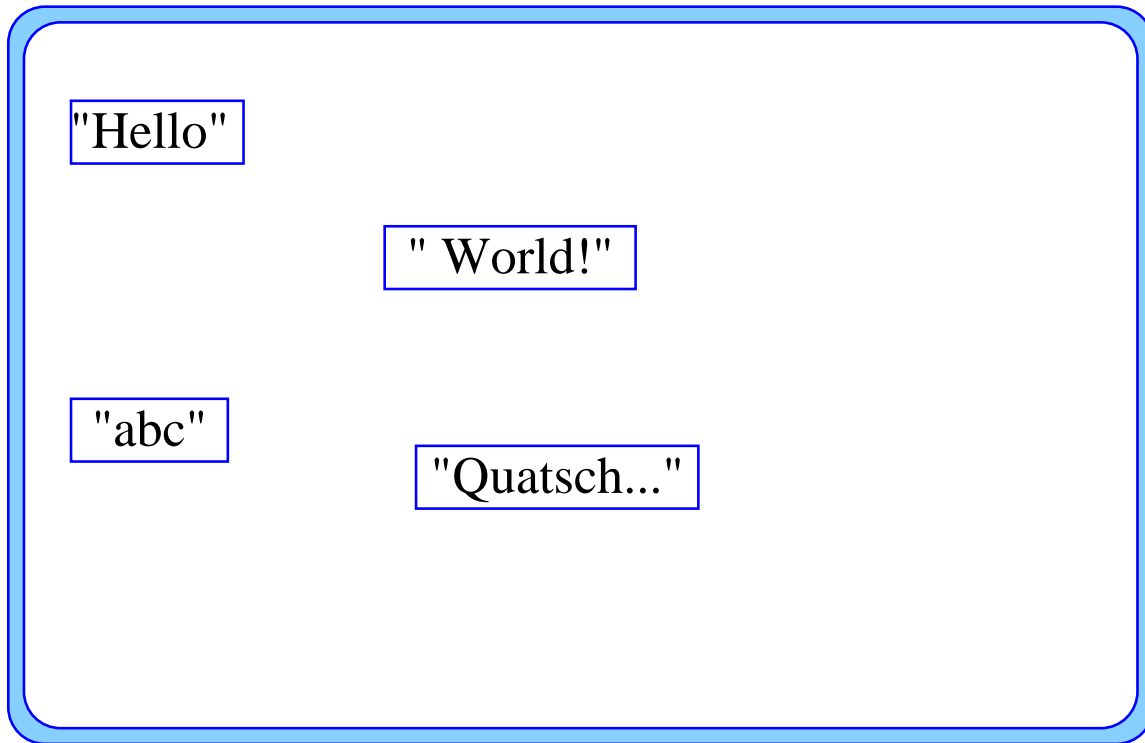
... sowie viele weitere :-)

Zur Objekt-Methode intern() :

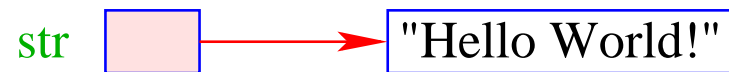
- Die **Java**-Klasse `String` verwaltet einen privaten String-Pool:

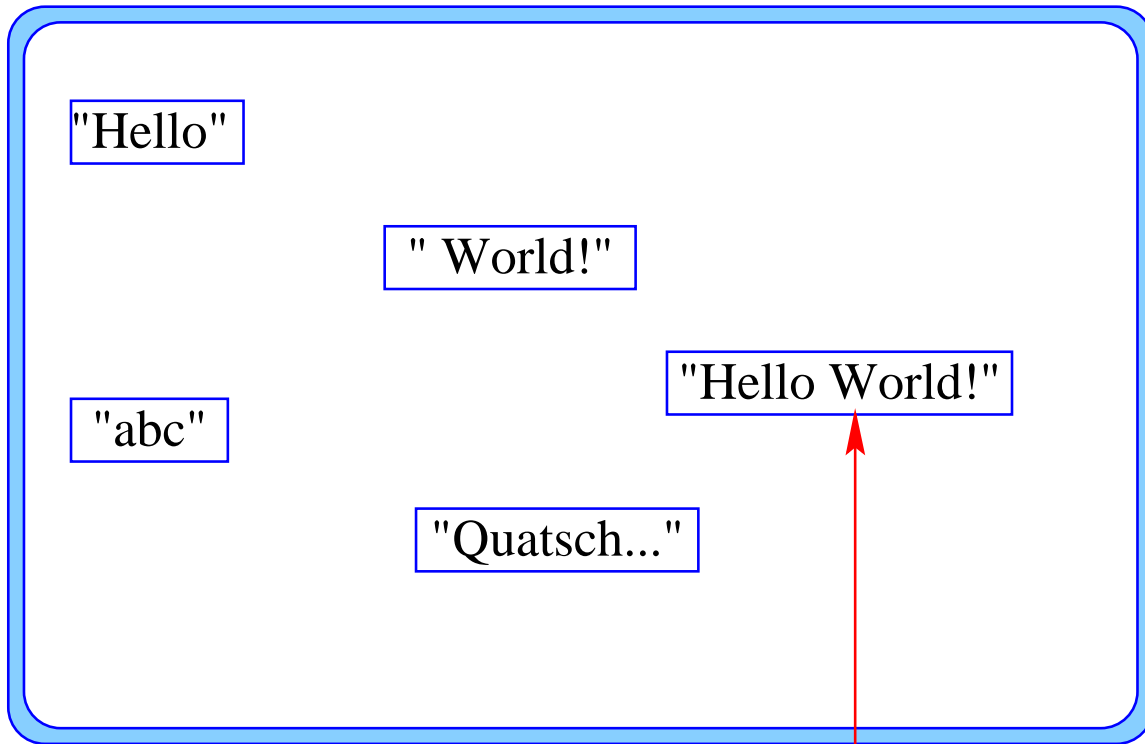


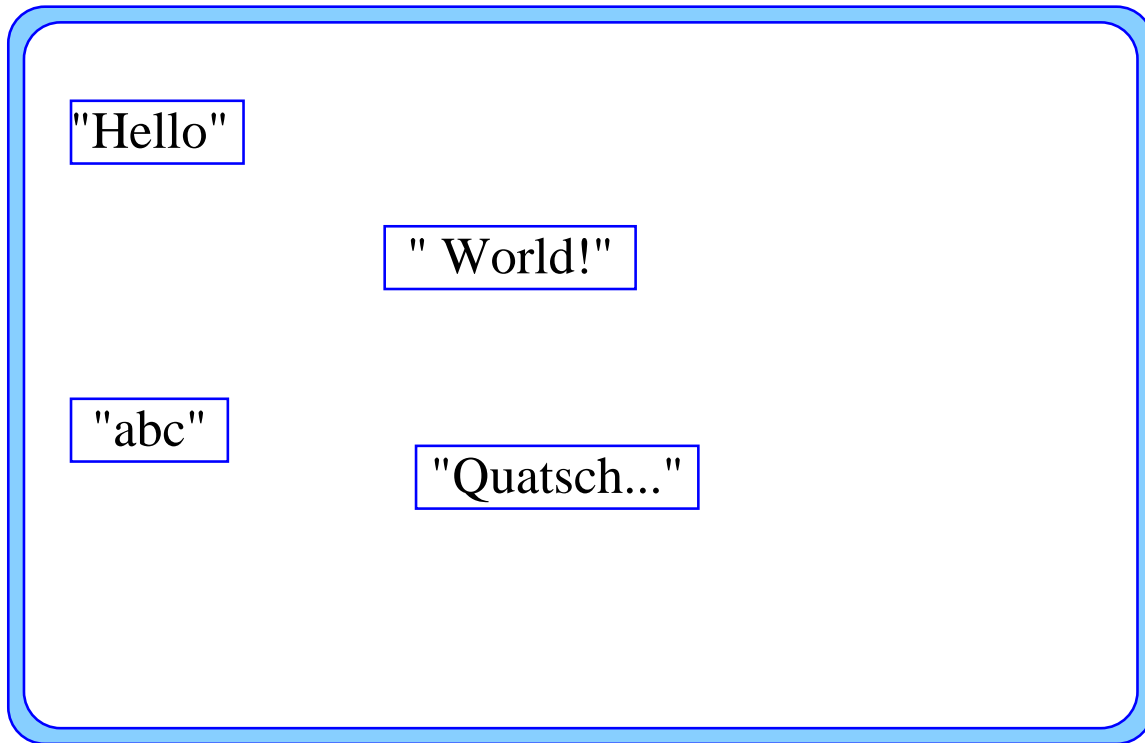
- Alle String-Konstanten des Programms werden automatisch in den Pool eingetragen.
- `s.intern()`; überprüft, ob die gleiche Zeichenfolge wie `s` bereits im Pool ist.
- Ist dies der Fall, wird ein Verweis auf das Pool-Objekt zurück gegeben.
- Andernfalls wird `s` in den Pool eingetragen und `s` zurück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



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

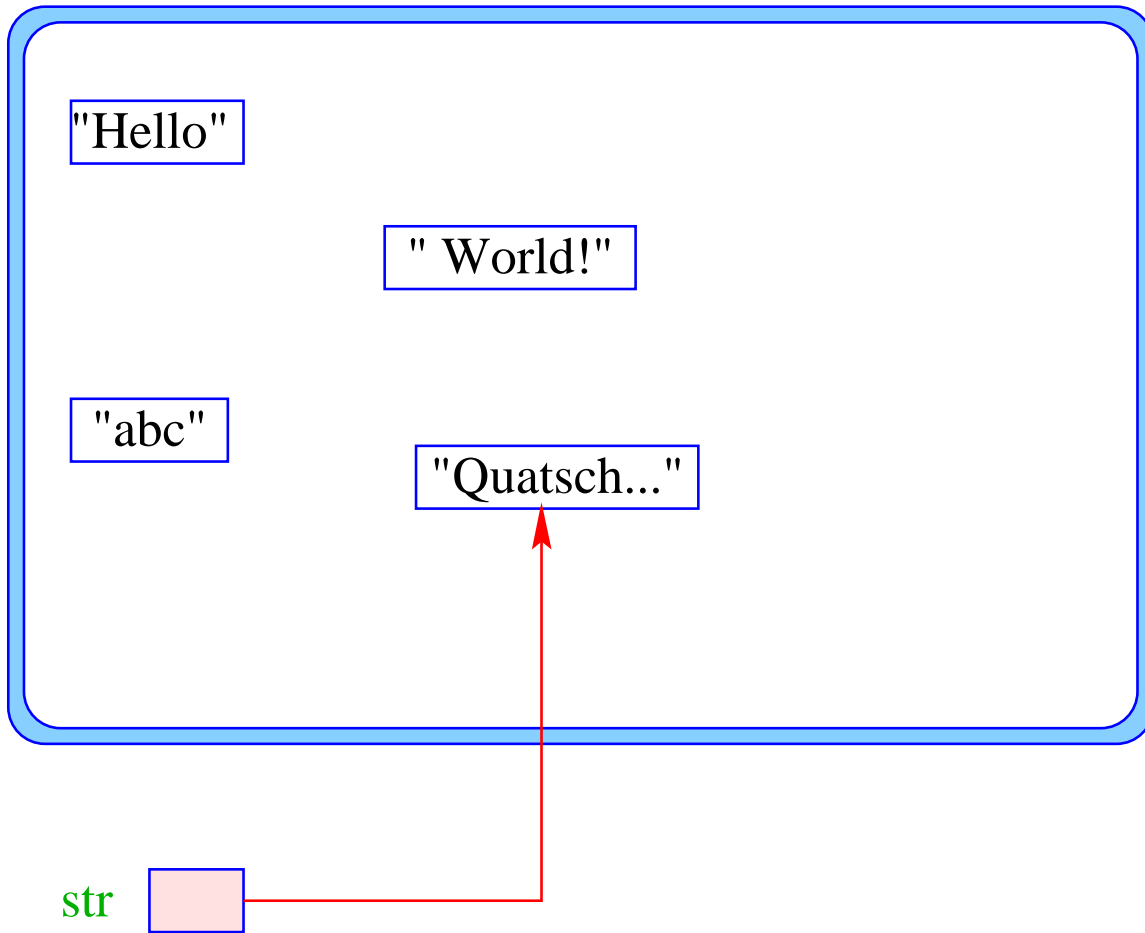






```
str = str.intern();
```





Vorteil:

- Internalisierte Wörter existieren nur einmal :-)
- Test auf Gleichheit reduziert sich zu Test auf Referenz-Gleichheit, d.h. “==”
⇒ erheblich effizienter als zeichenweiser Vergleich !!!

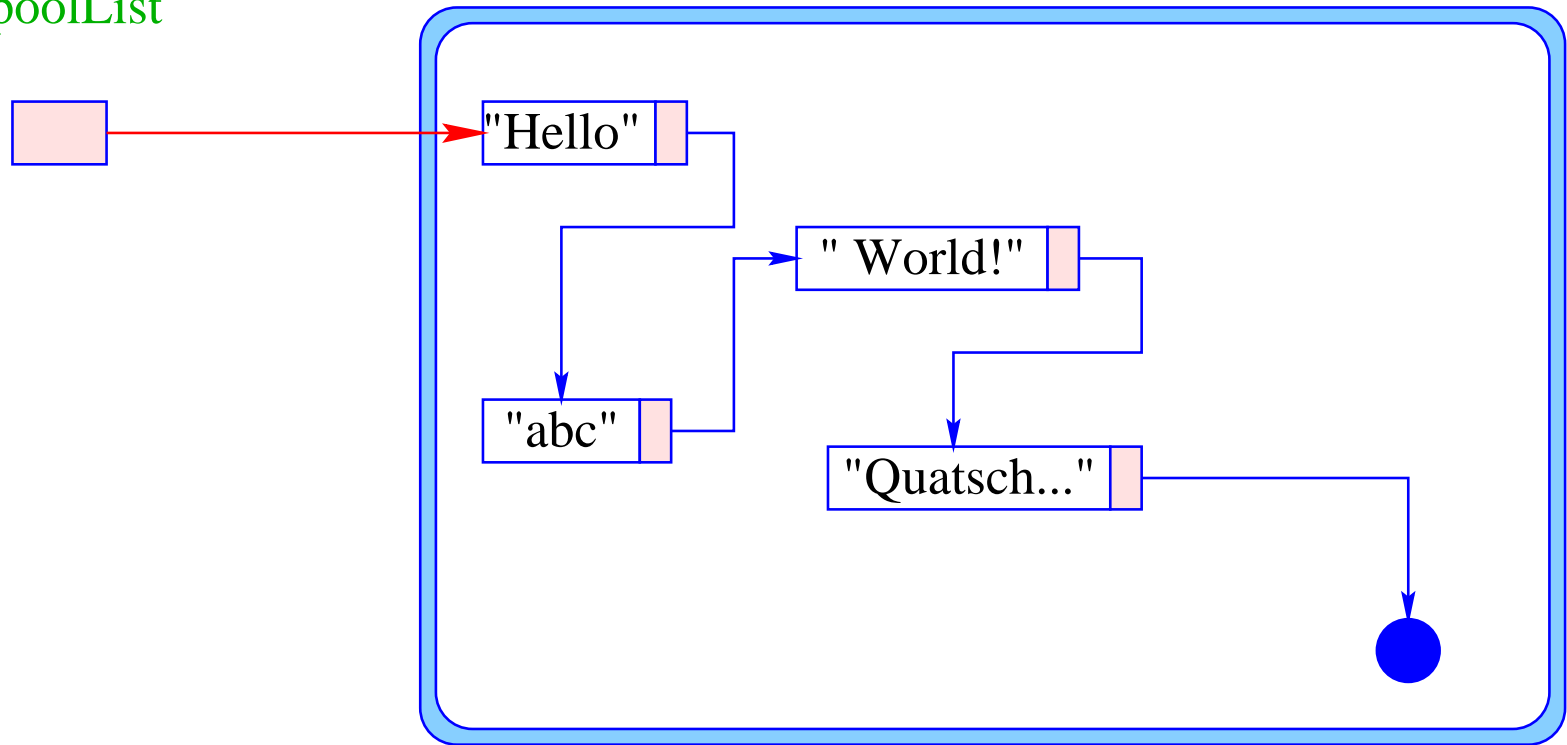
... bleibt nur ein Problem:

- Wie findet man heraus, ob ein gleiches Wort im Pool ist ??

1. Idee:

- Verwalte eine Liste der (Verweise auf die) Wörter im Pool;
- implementiere `intern()` als eine List-Methode, die die Liste nach dem gesuchten Wort durchsucht.
- Ist das Wort vorhanden, wird ein Verweis darauf zurückgegeben.
- Andernfalls wird das Wort (z.B. vorne) in die Liste eingefügt.

poolList

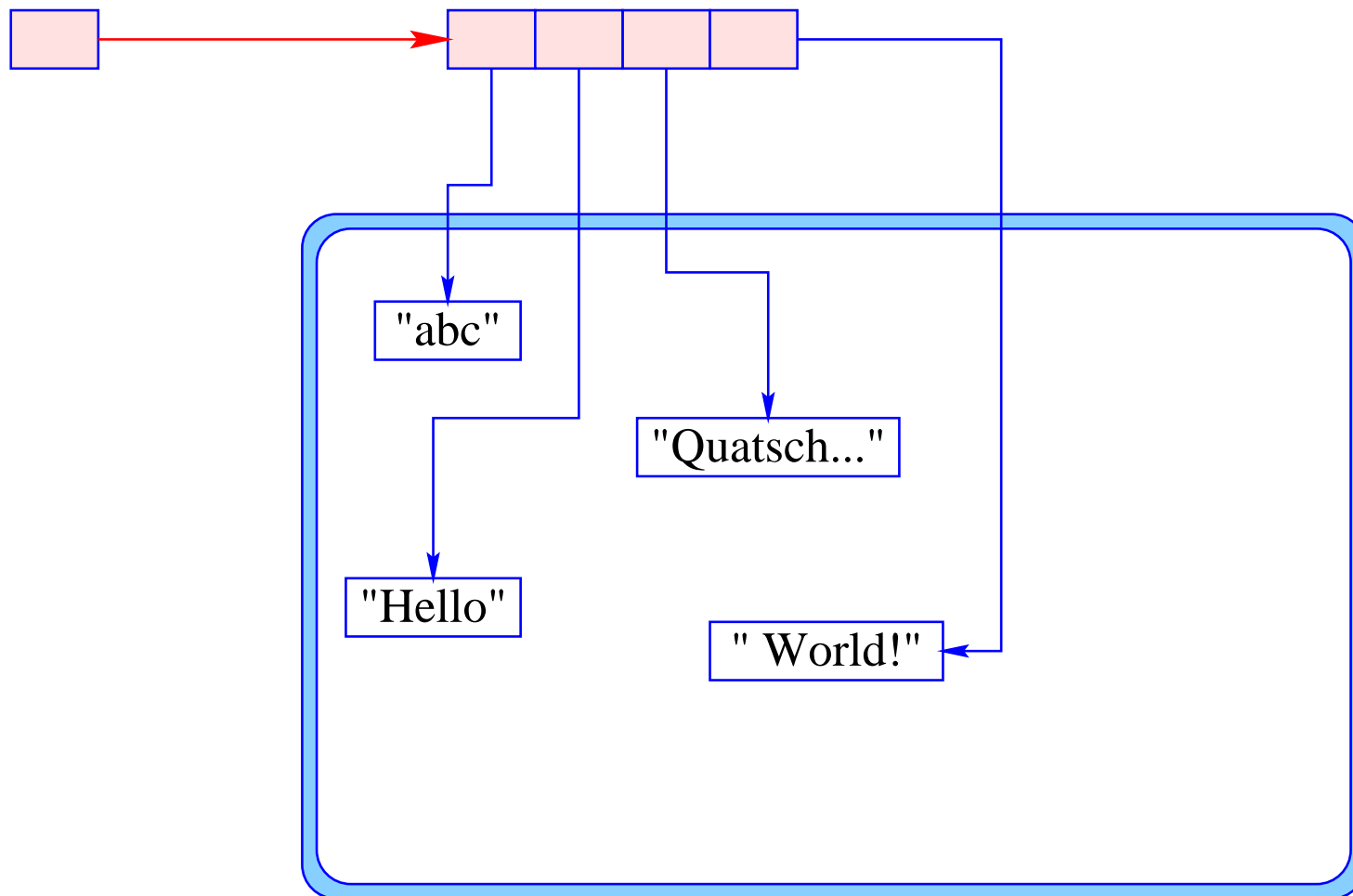


- + Die Implementierung ist einfach.
- die Operation `intern()` muss das einzufügende Wort mit **jedem** Wort im Pool vergleichen \implies immens teuer !!!

2. Idee:

- Verwalte ein sortiertes Feld von (Verweisen auf) String-Objekte.
- Herausfinden, ob ein Wort bereits im Pool ist, ist dann ganz einfach ...

poolArray

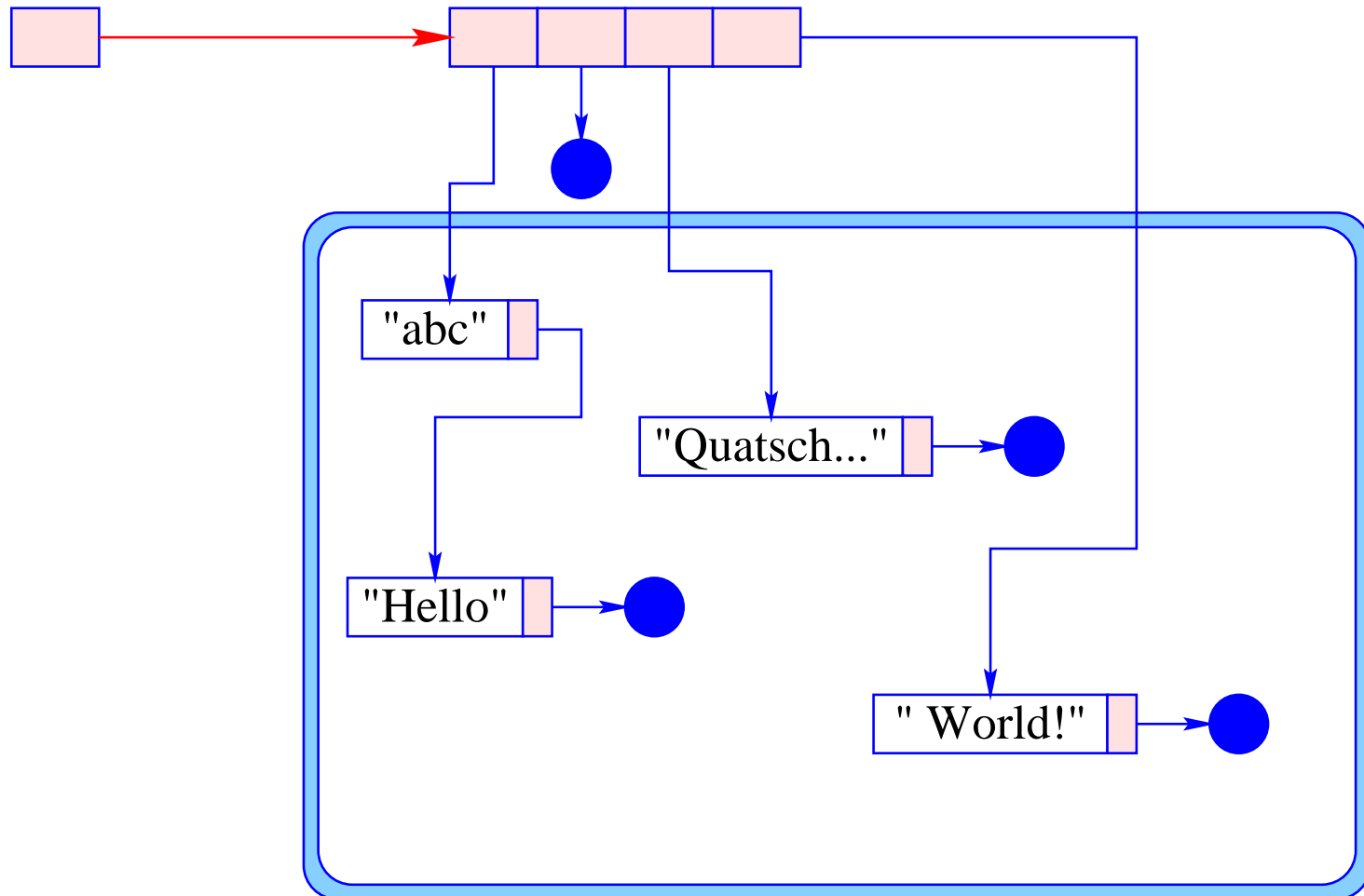


- + Auffinden eines Worts im Pool ist einfach.
- Einfügen eines neuen Worts erfordert aber evt. Kopieren aller bereits vorhandenen Verweise ...
 - ⇒ immer noch sehr teuer !!!

3. Idee: Hashing

- Verwalte nicht eine, sondern **viele** Listen!
- Verteile die Wörter (ungefähr) gleichmäßig über die Listen.
- Auffinden der richtigen Liste muss **schnell** möglich sein.
- In der richtigen Liste wird dann sequentiell gesucht.

hashCode



Auffinden der richtigen Liste:

- Benutze eine (leicht zu berechnende :-)) Funktion `hash: String -> int;`
- Eine solche Funktion heißt **Hash-Funktion**.
- Eine Hash-Funktion ist gut, wenn sie die Wörter (einigermaßen) gleichmäßig verteilt.
- Hat das Feld `hashSet` die Größe m , und gibt es n Wörter im Pool, dann müssen pro Aufruf von `intern()`; nur Listen einer Länge ca. n/m durchsucht werden !!!

Sei s das Wort $s_0s_1 \dots s_{k-1}$.

Beispiele für Hash-Funktionen:

- $h_0(s) = s_0 + s_{k-1}$;
- $h_1(s) = s_0 + s_1 + \dots + s_{k-1}$;
- $h_2(s) = (\dots ((s_0 * p) + s_1) * p + \dots) * p + s_{k-1}$ für eine krumme Zahl p .

(Die String-Objekt-Methode `hashCode()` entspricht der Funktion h_2 mit $p = 31$.)

String	h_0	h_1	$h_2 (p = 7)$
alloc	196	523	276109
add	197	297	5553
and	197	307	5623
const	215	551	282083
div	218	323	5753
eq	214	214	820
fjump	214	546	287868
false	203	523	284371
halt	220	425	41297
jump	218	444	42966
less	223	439	42913
leq	221	322	6112
...		...	

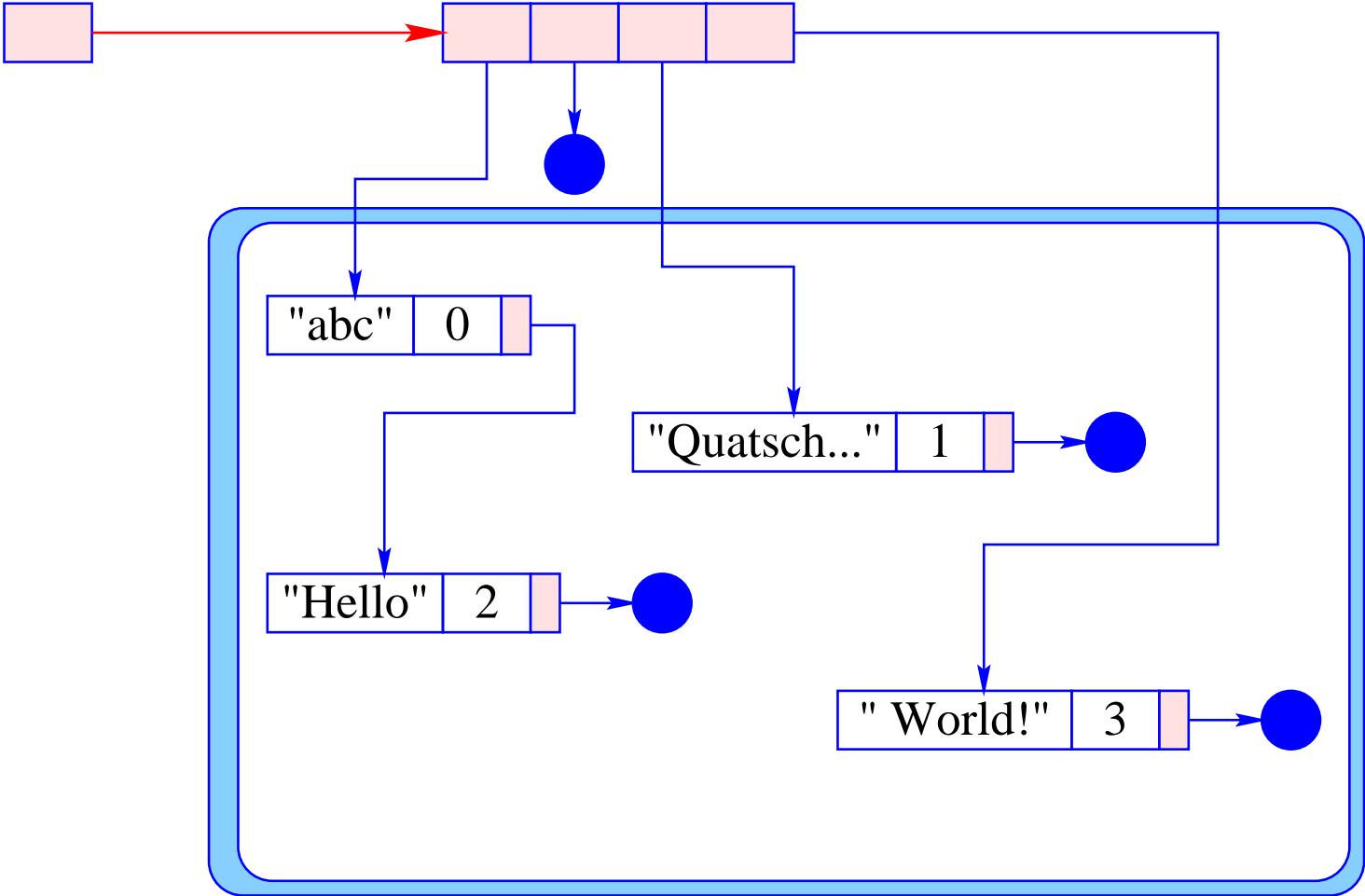
String	h_0	h_1	h_2
...		...	
load	208	416	43262
mod	209	320	6218
mul	217	334	6268
neq	223	324	6210
neg	213	314	6200
not	226	337	6283
or	225	225	891
read	214	412	44830
store	216	557	322241
sub	213	330	6552
true	217	448	46294
write	220	555	330879

Mögliche Implementierung von intern():

```
public class String {
    private static int n = 1024;
    private static StringList[] hashSet = new StringList[n];
    public String intern() {
        int i = (Math.abs(hashCode()))%n;
        for (StringList t=hashSet[i]; t!=null; t=t.next)
            if (equals(t.info)) return t.info;
        hashSet[i] = new StringList(this, hashSet[i]);
        return this;
    } // end of intern()
    ...
} // end of class String
```

- Die Methode `hashCode()` existiert für sämtliche Objekte.
- Folglich können wir (wenn wir Lust haben :-)) ähnliche Pools auch für andere Klassen implementieren.
- **Vorsicht!** In den Pool eingetragene Objekte können vom Garbage-Collector nicht eingesammelt werden ... :-|
- Statt nur nachzusehen, ob ein Wort `str` (bzw. ein Objekt `obj`) im Pool enthalten ist, könnten wir im Pool auch noch einen Wert hinterlegen
 \implies Implementierung von beliebigen Funktionen `String -> type` (bzw. `Object -> type`)

hashTable



Weitere Klassen zur Manipulation von Zeichen-Reihen:

- `StringBuffer` – erlaubt auch destruktive Operationen, z.B. Modifikation einzelner Zeichen, Einfügen, Löschen, Anhängen ...
- `java.util.StringTokenizer` – erlaubt die Aufteilung eines `String`-Objekts in **Tokens**, d.h. durch Separatoren (typischerweise White-Space) getrennte Zeichen-Teilfolgen.

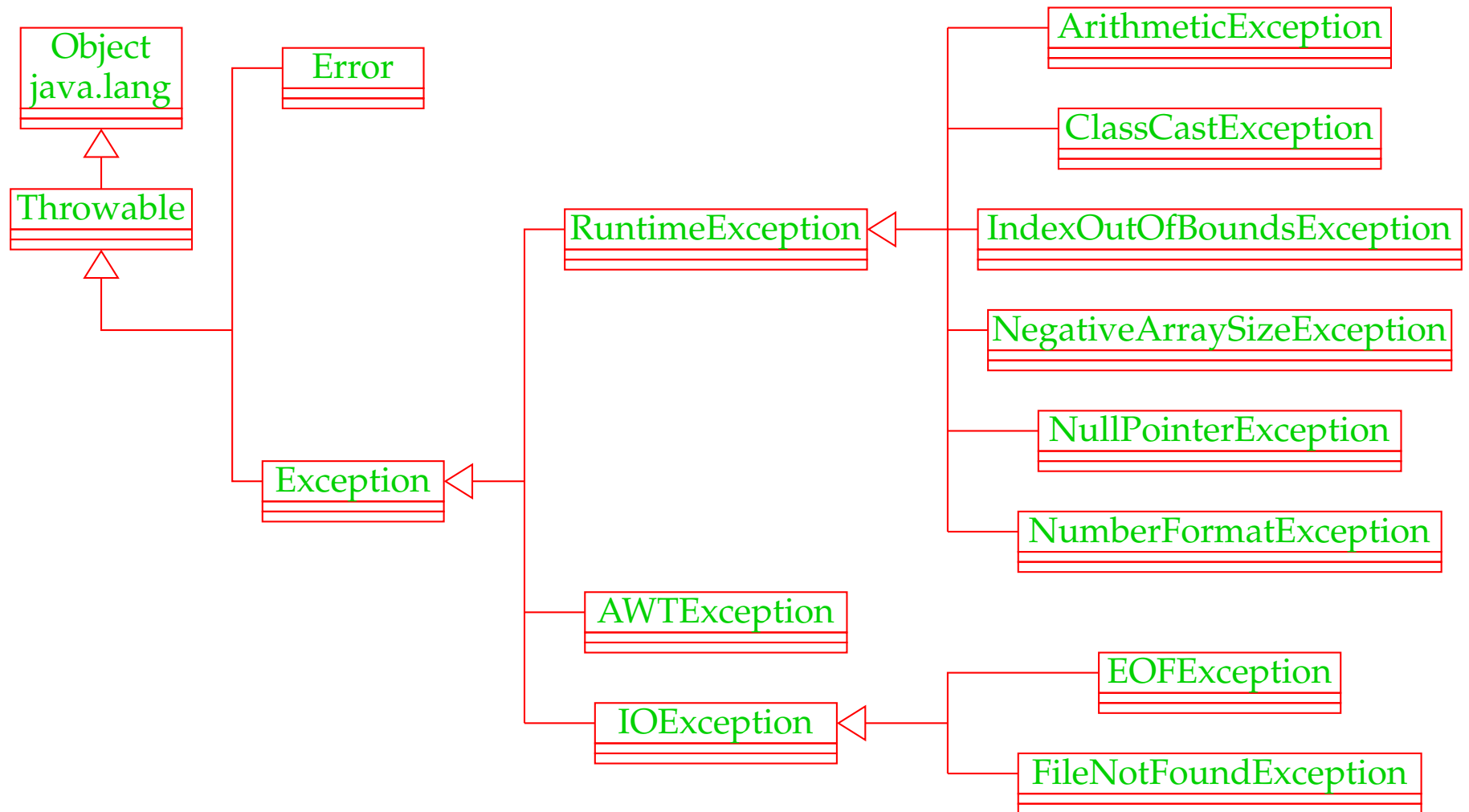
17 Fehler-Objekte: Werfen, Fangen, Behandeln

- Tritt während der Programm-Ausführung ein Fehler auf, wird die normale Programm-ausführung abgebrochen und ein Fehler-Objekt erzeugt (**geworfen**).
- Die Klasse `Throwable` fasst alle Arten von Fehlern zusammen.
- Ein Fehler-Objekt kann **gefangen** und geeignet **behandelt** werden.

Idee: Explizite Trennung von

- normalem Programm-Ablauf (der effizient und übersichtlich sein sollte); und
- Behandlung von Sonderfällen (wie illegalen Eingaben, falscher Benutzung, Sicherheitsattacken, ...)

Einige der vordefinierten Fehler-Klassen:



Die direkten Unterklassen von `Throwable` sind:

- `Error` – für fatale Fehler, die zur Beendigung des gesamten Programms führen, und
- `Exception` – für bewältigbare Fehler oder Ausnahmen.

Ausnahmen der Klasse `Exception`, die in einer Methode auftreten können und dort nicht selbst abgefangen werden, müssen **explizit** im Kopf der Methode aufgelistet werden !!!

Achtung:

- Die Unterklasse `RuntimeException` der Klasse `Exception` fasst die bei normaler Programm-Ausführung evt. auftretenden Ausnahmen zusammen.
- Eine `RuntimeException` kann jederzeit auftreten ...
- Sie braucht darum nicht im Kopf von Methoden deklariert zu werden.
- Sie kann, muss aber nicht abgefangen werden :-)

Achtung:

- Die Unterklasse `RuntimeException` der Klasse `Exception` fasst die bei normaler Programm-Ausführung evt. auftretenden Ausnahmen zusammen.
- Eine `RuntimeException` kann jederzeit auftreten ...
- Sie braucht darum nicht im Kopf von Methoden deklariert zu werden.
- Sie kann, muss aber nicht abgefangen werden :-)

Arten der Fehler-Behandlung:

- Ignorieren;
- Abfangen und Behandeln dort, wo sie entstehen;
- Abfangen und Behandeln an einer anderen Stelle.