Script **generated by TTT**

Title: Westermann: Einführung in die Informatik 1 (16.01.2012)

Date: Mon Jan 16 17:28:21 CET 2012

Duration: 58:19 min

Pages: 31

1 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 (immutable).
- Die Zeichenkette ist als privates char-Array gesichert.
- Die Länge der Zeichenkette ist auch ein privates Attribut und nur über eine Methode zugänglich.
- Veränderbare Wörter stellt die Klasse ↑`StringBuffer` zur Verfügung.

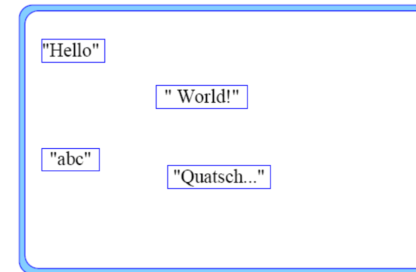
Informationen für die Verbindung mit eduroam erforderlich.
Klicken Sie hier, um weitere Informationen anzuzeigen.

Beispiele:

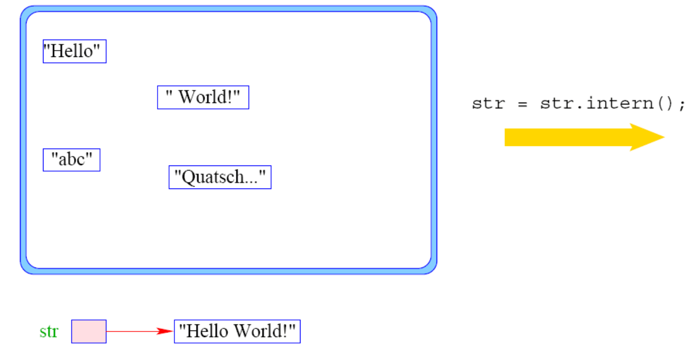
```
String s = "Es   wird  alles gut";
System.out.println(s.replaceAll(" +", " "));
// liefert: Es wird alles gut
System.out.println(s.replaceFirst(" +", " " ));
// liefert: Es wird  alles gut

// Ersetzen durch reguläre Ausdrücke
// X* : X kommt keinmal oder beliebig oft vor
// . :    steht für beliebige Zeichen
// X+ : X kommt einmal oder beliebig oft vor.
// X? : X kommt einmal oder keinmal vor.
```

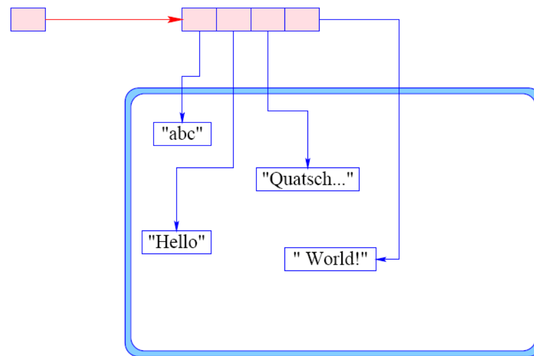

Zur Objekt-Methode `intern()`:

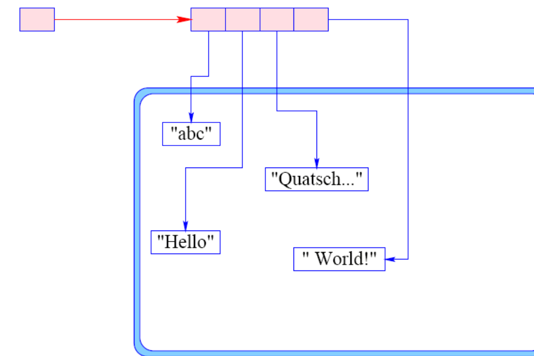
- Die Java-Klasse `String` verwaltet einen privaten String-Pool:



- 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.
- Für Strings `s` und `t` liefert `s.intern() == t.intern()` genau dann `true` wenn `s.equals(t)` `true` ist.







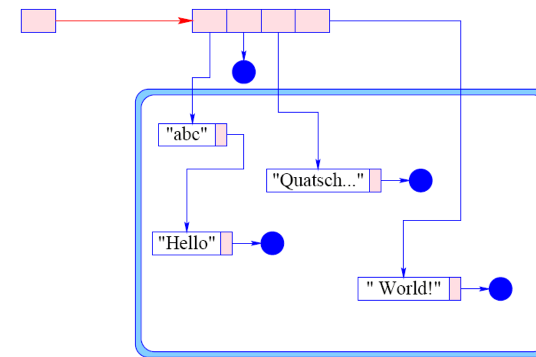
+ Auffinden eines Worts im Pool ist einfach.

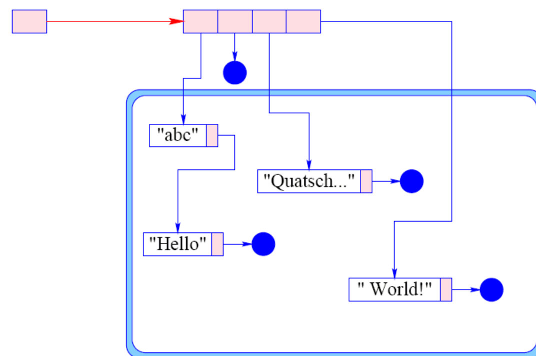
- Einfügen eines neuen Worts erfordert aber evt. Kopieren aller bereits vorhandenen Verweise ...

$\Longrightarrow$ 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.





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 !!!

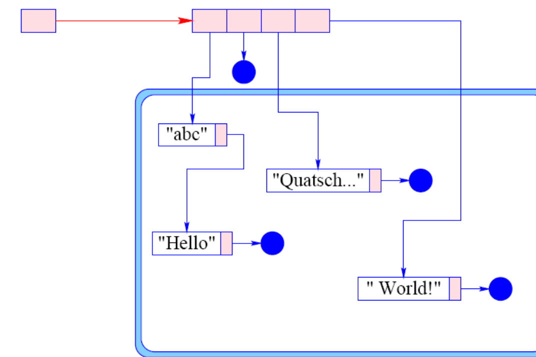
+ Auffinden eines Worts im Pool ist einfach.

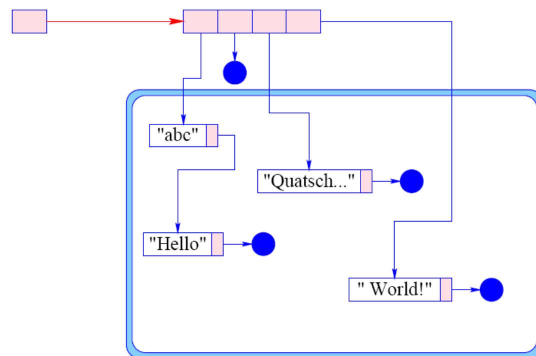
- Einfügen eines neuen Worts erfordert aber evt. Kopieren aller bereits vorhandenen Verweise ...

   ⟹ 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.





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 !!!

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

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

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 !!!

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

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 $s_0 s_1 \ldots s_{k-1}$.

Beispiele für Hash-Funktionen:

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

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

Mögliche Implementierung von `intern()`:

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


Sei s das Wort $s_0 s_1 \ldots s_{k-1}$.

Beispiele für Hash-Funktionen:

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

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

Mögliche Implementierung von `intern()` :

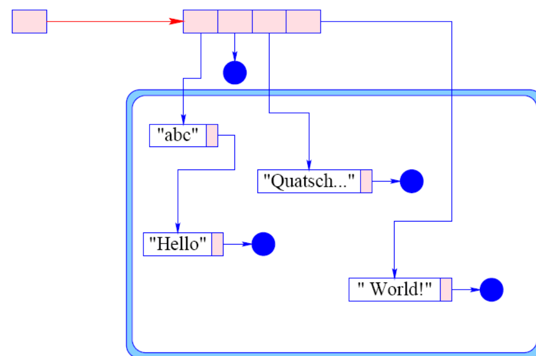
```
public class String {
   private static int n = 1024;
   private static List<String>[] hashSet = new List<String>[n];
   public String intern() {
      int i = Math.abs(hashCode()%n);
      for (List<String> t=hashSet[i]; t!=null; t=t.next)
         if (equals(t.info)) return t.info;
      hashSet[i]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


Sei s das Wort $s_0 s_1 \ldots s_{k-1}$.

Beispiele für Hash-Funktionen:

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

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

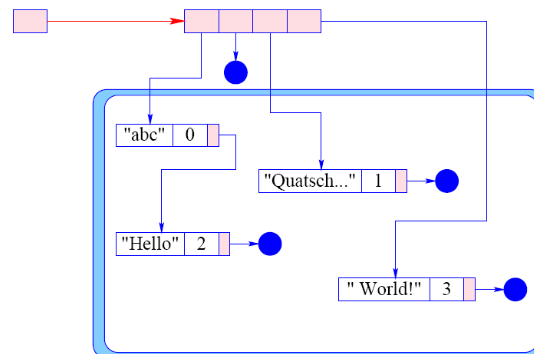


Mögliche Implementierung von `intern()` :

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

- Die Methode `hashCode()` existiert für sämtliche Objekte.
- Folglich können wir 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
  ⟹ Implementierung von beliebigen Funktionen `String -> type` (bzw. `Object -> type`)

- Die Methode `hashCode()` existiert für sämtliche Objekte.
- Folglich können wir 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
  ⟹ Implementierung von beliebigen Funktionen `String -> type` (bzw. `Object -> type`)



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.