ETH zürich

Serie 2

Felix Friedrich, Sandro Huber, Fabian Stutz, Lars Widmer
Übungslektion, Informatik II D-BAUG

11. März 2014

ETH zürich

Übersicht

ETH zürich

Ist das nicht inkonsequent?

Wer kann das erklären?

1. `if (str.charAt(i) == 'e') ...`
   funktioniert **bestens**!
2. `if (str=="*") ...`
   geht **gar nicht**!
   Stattdessen müssen wir folgendes verwenden:
   `if (str.equals("*")) ...`

ETH zürich

Wertetypen

Frage

Nenne Beispiele von **Wertetypen** rsp. **Basistypen**?

ETH zürich

Wertetypen: Beispiele

- `char`: Buchstaben
- `int`: Ganzzahlen
- `long`: Grosse Ganzzahlen
- `short`: Kurze Ganzzahlen
- `float`: Fliesskommazahlen
- `double`: Fliesskommazahlen mit doppelter Genauigkeit

ETH zürich

Erkennen von Wertetypen

Fragen

Wie erkennt man **Wertetypen**?

ETH zürich

Erkennen von Wertetypen

- Eclipse druckt Wertetypen rot und in Fettschrift.
- Es wird kein `new` statement benötigt, um z. B. eine `int` Variable zu erzeugen.
- Variablen von Wertetypen haben keine Methoden.
- Wertetypen sind alle vordefiniert. Wir können keine eigenen Wertetypen definieren.
- Wertetypen werden komplett klein geschrieben.
- Wertetypen brauchen nur begrenzt Speichplatz.
- ✓ Daher bezeichnen wir Wertetypen nicht als Klassen.
- ✓ Und Variablen von Wertetypen nennen wir nicht Objekte.

ETH zürich

Referenztypen

Frage

Nenne Beispiele von **Referenztypen**?

Referenztypen: Beispiele

- `String`: Zeichenkette
- `Array`: Liste von Elementen
- Jegliche Klassen
- ...

Erkennen von Referenztypen

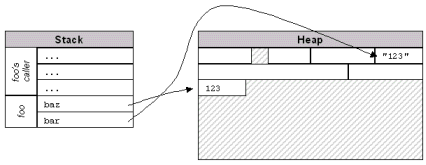
Frage

Woran erkennt man **Referenztypen**?

Erkennen von Referenztypen

- Eclipse markiert Referenztypen nicht speziell.
- Referenztypen fangen mit einem Grossbuchstaben an.
- Es wird stets das `new` Statement benötigt, um ein Objekt zu erstellen. *Ausnahme: Bei Strings wie `''Hallöchen''` erstellt java im Hintergrund ein Stringobjekt. The Regel stimmt trotzdem, aber das `new` Statement ist im Code nicht explizit sichtbar.*
- Die meisten Objekte bringen Methoden mit.
- Wir können unsere eigenen Klassen definieren.

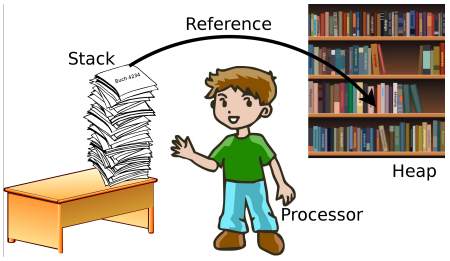
Referenzen sind Zeiger (Wegweiser)



Source: kdgregory.com

- Die Pfeile stellen Referenzen, also Zeiger (Pointer) dar.

Falls du ein Prozessor bist ...



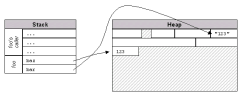
Du wachst auf und bist im Computer

Dein Büro ist ein Computer, du bist der Prozessor und die Hausaufgaben sind das Programm.

- Der **Stack** ist ein Stapel Blätter auf deinem Pult.
- Als **Objekte** stellen wir uns Bücher vor.
- Wenn du 100 Bücher auf dein Pult knallst, kannst du unmöglich weiterarbeiten. Deswegen stapelst du ausschliesslich Blätter auf deinem **Stack**.
- Bücher gehören ins Büchergestell. Daher ist das Büchergestell dein **Heap** (Haufenspeicher).
- Du hast zu den benötigten Büchern einen Notizzettel auf deinem Stapel. So eine Notiz ist eine **Referenz**. **Referenzen** liegen auf dem **Stack**, zeigen aber in den **Heap**.

Referenzen sind Pointer

- Jedes Programm hat seinen eigenen Stack.
- Jeder Methodenaufruf hat seinen eigenen Bereich im Stack.
- Jede lokale Variable einer Methode liegt im Stackframe dieser Methode.
- Weil grössere Elemente (wie Objekte) nicht im Stack abgelegt werden können, speichern wir sie im Heap.



Referenzen als Werte

- Tatsächlich haben Objektvariablen auch einen Wert!
- Die **Referenz** ist der Wert eines **Objekts**.
- Die Referenz zeigt an die Adresse, wo das Objekt im Heap liegt. Der Speicher ist mit Nummern adressiert.
- Wenn wir nun `TestClass a = new TestClass();` ausführen, wird ein Objekt von `TestClass` erstellt und im Speicher abgelegt. In der Variablen `a` ist aber nur die **Adresse** (**Referenz**) des Objekts gespeichert.
- Bei Wertetypen hingegen wird keine Referenzierung verwendet. Variablen von Wertetypen liegen direkt im Stack. Sie haben nur ihren Wert, wie es der Name schon sagt.

Referenz- & Wertetypen

1. Deswegen vergleicht `(str==''*'')` nur die beiden Referenzen.
2. Die beiden zu vergleichenden Objekte werden aber unabhängig von eineander erstellt. Sie liegen daher **nie** an der gleichen Stelle.
3. Daher ist `(str==''*'')` **immer** `false`.
4. Das ist eine **sehr häufige Falle**, merkt sie euch gut! Es gibt deswegen Leute, die finden, dass java diesen Fall anders handhaben sollte, aber das wäre nicht konsequent.

Übersicht

Vektor-Matrix Multiplikation

Das Ziel ist zu zeigen, wie ein Vektor zu einem fixen Wert konvergiert, wenn er wiederholt mit der selben Matrix multipliziert wird.

$V_1 = V_0 \times M$
$V_2 = V_1 \times M$
$V_3 = V_2 \times M$
$\ldots$
$V_{n+1} = V_n \times M$

Für ein grosses n soll das folgende gelten:
$V_n \to V_\infty$

Einzelner Schritt

$$V_0 = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

$$M = \begin{pmatrix} 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$V_1 = V_0 \times M$$

$$V_1 = \begin{pmatrix} 1 \times 3 + 2 \times 5 & 1 \times 4 + 2 \times 6 \end{pmatrix}$$

$$V_1 = \begin{pmatrix} 13 & 16 \end{pmatrix}$$

Konvergenz

$$M = \begin{pmatrix} 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$V_0 = \begin{pmatrix} 1 & 2 \end{pmatrix}, V_1 = \begin{pmatrix} 13 & 16 \end{pmatrix} \ldots$$

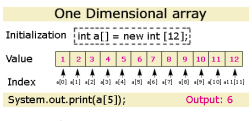
Offene Frage

Wird dieses Beispiel konvergieren?
. . . Probier es aus in Übung 2.

Übersicht

Arrays

Wie in Pascal werden Arrays auch in java verwendet, um Listen von Elementen zu speichern. Alle Elemente haben immer den gleiche Typ.



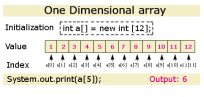
Source: http://www.roseindia.net

Arrays

Man kann sich ein Array wie eine vordefinierte Klasse vorstellen.

Aber . . .

1. Was heisst das?
2. Wie gebrauchen wir sie?



Source: http://www.roseindia.net

Arrays

1. Ist ein Referenz Typ (nicht ein Werte-/Basistyp, wie z. B. `int`).
2. Das heisst ein Array Objekt liegt im Speicher (Heap).
3. Deshalb brauchen wir das Keyword `new` um ein Array-Objekt zu erzeugen.
4. Beispiel: `int[] list = new int[20];` (Grösse 20: Indices 0..19)
5. Die Linie oberhalb macht 2 Dinge auf einmal:
   - `int[] list;` (Deklaration)
   - `list = new int[20];` (Erzeugung + Initialisierung)
6. Da `int` ein Wertetyp ist, brauchen wir kein `new` Statement für jeden einzelnen `int` im Array.
7. Das ist anders, wenn wir z. B. ein Array von `String`s wollen.

Übersicht

Umkehren: Der gegebene Code

```
1  import java.util.Scanner;
2  public class Reverse {
3      public static void main(String[] args) {
4          String str = "";
5          String rev = "";
6          Scanner sc = new Scanner(System.in);
7          System.out.print("Please␣enter␣a␣word:␣");
8          str = sc.next();
9          System.out.println(str);
10         for (int i=0; i<str.length(); ++i)
11             rev = str.charAt(i) + rev;
12         System.out.println(rev);
13     }
14 }
```


Umkehren: Berechnungen

```
1 for (int i=0; i<str.length(); ++i)
2     rev = rev + str.charAt(i);
```

Rätsel:

1. Wer findet den Unterschied?
2. Was ist das Resultat?

```
1 for (int i=0; i<str.length(); ++i)
2     rev = str.charAt(i) + rev;
```


Umkehren: Falsche Berechnung

```
1 for (int i=0; i<str.length(); ++i)
2     rev = rev + str.charAt(i);
```

Dieser Code durchläuft das Wort von Anfang bis Ende. Ein Buchstabe nach dem Anderen wird der Variable `rev` hinzugefügt. In der Tat wird das Resultat gleich wie der Input sein.

Umkehren: Korrekte Berechnung

```
1 for (int i=0; i<str.length(); ++i)
2     rev = str.charAt(i) + rev;
```

Dieser Code verhält sich anders: Er durchläuft immernoch das Wort von Anfang bis Ende. Ein Buchstabe nach dem Anderen wird der Variable `rev` hinzugefügt. **Aber** sie werden am **Anfang** des Strings angehängt. Deshalb ist das Resultat der umgekehrte String.

Übersicht

Buchstaben zählen: Ganze Methode

```
1  import java.util.Scanner;
2  public class CountLetter {
3      public static void main(String[] args) {
4          int count = 0;
5          Scanner sc = new Scanner(System.in);
6          System.out.print("Please␣enter␣a␣word:␣");
7          String str = sc.next();
8          for (int i=0; i<str.length(); ++i)
9              if (str.charAt(i) == 'e')
10                 ++count;
11         System.out.println(count);
12     }
13 }
```

Buchstaben zählen: Zweierlei For-Schleifen

```
for (int i=0; i<str.length(); ++i)
    rev = str.charAt(i) + rev;
```

Frage

Vergleiche diese zwei Code-Fragmente. Was ist der Unterschied?

```
for (int i=0; i<str.length(); ++i)
    if (str.charAt(i) == 'e')
        ++count;
```


Buchstaben zählen: Das eigentliche Zählen

```
for (int i=0; i<str.length(); ++i)
    if (str.charAt(i) == 'e')
        ++count;
```

. . . Das Wort wird immernoch von Anfang bis Ende durchlaufen. Aber jedesmal wenn ein “e” vorkommt, wird der Wert von `count` inkrementiert.

Übersicht

Jumbled Strings

Arniodccg to a recheaesrr at Cmbragide Ueytrvsiin, it dosen’t mettar in what oedrr the leertts in a wrod aer, the only itpramnot thing is taht the frist and lsat ltteer be at the rhigt plcae. The rest can be a toatl mses and you can sltil read it whtuoit pberolm. This is bscueae the hamun mnid deos not read erevy letter by istlef but the word as a wolhe.

Originaltext

According to a researcher at Cambridge University, it doesn’t matter in what order the letters in a word are, the only important thing is that the first and last letter be at the right place. The rest can be a total mess and you can still read it without problem. This is because the human mind does not read every letter by itself but the word as a whole.

Buchstabensalat: Äussere Programmstruktur

```
import java.util.LinkedList;
import java.util.Random;
import java.util.Scanner;
import java.lang.Character;

public class MixMiddle {
    // ...
}
```

Eine `LinkedList` ist eine Datenstruktur, die einem `Array` sehr ähnlich ist. Sie erlaubt unter Anderem das Löschen von jeglichen Elementen, also auch jene in der Mitte der Liste. Dieses Feature vereinfacht die Implementierung von MixMiddle.

Buchstabensalat: Ganze Methode

```
private static String jumble(String str) {
    int len = str.length();
    if (len < 4) return str;
    Random rand = new Random();
    LinkedList<Character> letters
            = new LinkedList<Character>();
    for (int i=1; i<len-1; ++i)
        letters.add(str.charAt(i));
    String mix = "" + str.charAt(0);
    for (int i=0; i<len-2; ++i)
        mix += letters.remove(
                rand.nextInt(letters.size()));
    mix += str.charAt(len-1);
    return mix;
}
```


Buchstabensalat: Return Statements

Frage

Was ist das Ziel der zwei folgenden `return`-Statements?

```
private static String jumble(String str) {
    int len = str.length();
    if (len < 4) return str;
    // ...
    return mix;
}
```

Buchstabensalat: Return Statements

```
1 private static String jumble(String str) {
2     int len = str.length();
3     if (len < 4) return str;
4     // ...
5     return mix;
6 }
```

Das `return`-Statement beendet eine Methode sofort. Das muss nicht am Ende der Methode sein. Return an belieben Stellen ist sicher kurz & bündig, aber nicht sehr schön, da es irreführend sein kann. Genau deshalb sollt ihr es kennen.

Buchstabensalat: LinkedList

Frage

Was macht das folgende Code-Fragment?

```
1     LinkedList<Character> letters
2             = new LinkedList<Character>();
3     for (int i=1; i<len-1; ++i)
4         letters.add(str.charAt(i));
```


Buchstabensalat: String kopieren

```
1     LinkedList<Character> letters
2             = new LinkedList<Character>();
3     for (int i=1; i<len-1; ++i)
4         letters.add(str.charAt(i));
```

Hier konvertieren wir das Wort in eine `LinkedList` von Characters (Buchstaben).
Den **ersten** und den **letzten** Buchstaben nehmen wir nicht.

Buchstabensalat: For-Schleife

Frage

Und was geht hier vor sich?

```
1     String mix = "" + str.charAt(0);
2     for (int i=0; i<len-2; ++i)
3         mix += letters.remove(
4                 rand.nextInt(letters.size()));
5     mix += str.charAt(len-1);
```


Buchstabensalat: Zufällige Buchstaben ziehen

```
1     String mix = "" + str.charAt(0);
2     for (int i=0; i<len-2; ++i)
3         mix += letters.remove(
4                 rand.nextInt(letters.size()));
5     mix += str.charAt(len-1);
```

Die Resultat-Variable `res` wird eingeführt. Für den ersten und letzten Buchstaben nehmen wir das Original, da wir diese nicht verschieben wollen.
Dazwischen setzen wir zufällige Buchstaben aus der `LinkedList` und fügen sie der Variable hinzu. Die `LinkedList` stellt sicher, dass wir **nicht** zweimal den gleichen Buchstaben benutzen.

Buchstabensalat: Übersicht

Die Methode durchläuft folgende Phasen:

1. Wortlänge prüfen, bei Wörtern kürzer als vier Buchstaben abbrechen
2. Wort in eine `LinkedList` kopieren
3. Den ersten Buchstaben unverändert ins Resultat übernehmen
4. Jeden Buchstaben zwischen dem Ersten und dem Letzten, zufällig ziehen und zum Resultat hinzufügen
5. Den letzten Buchstaben unverändert ins Resultat übernehmen
6. Resultat zurückgeben

Buchstabensalat: Hauptmethode

```
1 public static void main(String[] args) {
2     System.out.println("Enter * to exit");
3     Scanner sc= new Scanner(System.in);
4     String str = "-";
5     while (!str.contains("*") && sc.hasNext()) {
6         str = sc.next();
7         System.out.print(jumble(str) + " ");
8     }
9     System.out.println("\n... program done");
10 }
```


Buchstabensalat: Fragen

Knobelfragen

1. Warum müssen wir überprüfen, ob sich ein ‘‘*’’ im String befindet?
2. Können wir anstatt `str.contains(‘‘*’’)` auch `str.equals(‘‘*’’)` gebrauchen?
3. Können wir `(str==‘‘*’’)` an Stelle von `str.equals(‘‘*’’)` verwenden?

```
1 while (!str.contains("*") && sc.hasNext()) {
```

Buchstabensalat: Beenden

```
1  while (!str.contains("*") && sc.hasNext()) {
2      str = sc.next();
3      System.out.print(jumble(str) + "␣");
4  }
```

1. Das brauchen wir als einfachen Weg das Programm zu beenden, wenn die Wörter direkt eingegeben werden. (**Nicht** wenn man ein File via Pipe für den Input benutzt: `cat text.txt | java MixMiddle`).

Buchstabensalat: Stringvergleich

```
1  while (!str.contains("*") && sc.hasNext()) {
2      str = sc.next();
3      System.out.print(jumble(str) + "␣");
4  }
```

2. Zur Wiederholung: `String` ist ein `reference type` gleich wie `Arrays` in Java, aber unterschidlich zu `int` (value type). Das Statement `(str==''*'')` vergleicht deshalb nur zwei Referenzen. Die Referenz sagt, wo das `String`-Objekt im Speicher lokalisiert ist. Da die zwei Objekte seperat erstellt werden, haben sie nie die gleiche Speicheraddresse. Deshalb wird `(str==''*'')` immer false liefern.
   Merkt euch das! Es wird oft falsch gemacht.

Übersicht

Binärzahlen

Berechne den “normalen” Wert (zur Basis 10) der folgenden Binärzahlen:

1. 001
2. 010
3. 100
4. 011
5. 111
6. 0100 0000 0000
7. 1000 0000 0000

Hexadezimalzahlen

Berechne den “normalen” Wert (zur Basis 10) der folgenden Hexadezimalzahlen:

1. *A*
2. *B*
3. *F*
4. *FF*
5. 14
6. 21
7. *A0*

Fragen?

Wir sind froh um

- Fragen?
- Feedback?
- Wünsche?
- Anmerkungens?
- ...

Viel Glück!