Schrittweise Verfeinerung

Entwurfsmethode für Algorithmen

Wie kommt man von der Aufgabenstellung zum Programm?

Beispiel

"Zähle die Häufigkeit von Wörtern in einem Text"

Welche Befehle würde man sich wünschen, um diese Aufgabe zu lösen?

- lies Wort
- speichere Wort in Tabelle
- erhöhe Wortzähler
- drucke Zähler für alle Worte

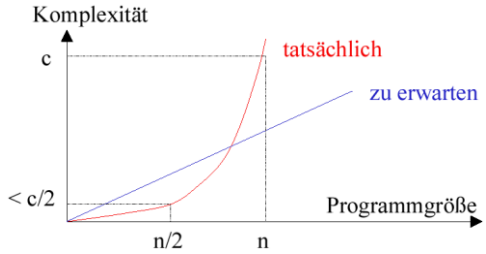
Leider gibt es keine Sprache mit diesen Befehlen!
Befehle als Methoden implementieren (d. h. gewünschte "Sprache" selbst bauen).

Vorgehensweise

Schrittweise Verfeinerung

1. Zerlege Aufgabe in Teilaufgaben und spezifiziere ihre Schnittstelle
2. Nimm an, daß die Teilaufgaben schon gelöst sind
3. Implementiere Gesamtaufgabe mit Hilfe der Teillösungen
4. Sind die Teilaufgaben einfach genug?
   ja => implementiere sie direkt in einer Programmiersprache
   nein => Zerlege sie weiter (Schritt 1)

Was nützt dies?



Komplexität steigt überproportional mit der Programmgröße

Halbierung der Programmgröße reduziert die Komplexität um mehr als die Hälfte!

Beispiel

Aufgabe : "Zähle die Häufigkeit von Wörtern in einem Text"

1. Zerlege Aufgabe in Teilaufgaben und spezifiziere ihre Schnittstelle

`word = readWord();` liefert nächstes Wort oder null, wenn kein Wort mehr von der aktuellen Eingabedatei gelesen werden kann."

Klasse *WordTable* mit folgenden Methoden:

`table. count( word);` trägt word in tab ein (falls noch nicht vorhanden) und erhöht Worthäufigkeit um 1

`table. print();` gibt Wörter und ihre Häufigkeiten aus

2. Nimm an, daß die Teilaufgaben schon gelöst sind

Beispiel (Forts.)

3. Implementiere Gesamtaufgabe mit Hilfe der Teillösungen

```
class WordCount {
    public static void main (String[] arg) {
        WordTable table = new WordTable();
        In. open(" input. txt");
        String word = readWord();
        while (word != null) {
            table. count( word);
            word = readWord();
        }
        In. close();
        table. print();
    }
}
```

Beispiel (Forts.)

4. Zerlege Teilaufgaben weiter (*readWord*)

ch = In. read();	lies ein Zeichen; **In. done == false**, wenn Dateiende
Character. isLetter(ch)	prüfe, ob **ch** ein Buchstabe ist
word. append(ch)	füge **ch** an das Wort **word** an

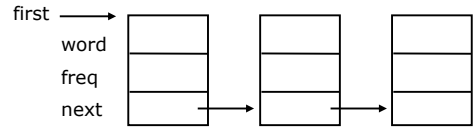
Alle Teilaufgaben sind bereits in Java implementiert.
Implementiere *readWord()* mit ihnen

```
static String readWord () {
    StringBuffer word = new StringBuffer();
    char ch;
    //----- skip nonletters
    do ch = In. read();    while ( In. done() && ! Character. isLetter( ch) );
    //----- build the word
    while ( In. done() && Character. isLetter( ch) ) {
        word. append( ch);
        ch = In. read();
    }                          // ! In. done() || ch is not a letter
    if (word. length() > 0) return word. toString();       else    return null;
}
```

Beispiel (Forts.)

Grobstruktur der Worttabelle:

Datenstruktur:
verkettete Liste von Wörtern
und Häufigkeiten




```
class Element {
    String word;
    int freq;
    Element next;
    Element (String w) {
        word = w; freq = 1;
    }
}
```

```
class WordTable {
    Element first;
    void count (String word) {...}
    void print () {...}
}
```

Beispiel (Forts.)

4. Zerlege Teilaufgaben weiter (*count*)

elem = table. find(word);	suche *word* in *table* und liefere entspr. Element oder Null,
table. enter(word);	trage *word* in *table* ein (es kommt noch nicht vor),
freq++;	erhöhe Worthäufigkeit.

```
void count (String word) {
    Element e = table. find( word);
    if (e == null) table. enter( word); else e. freq++;
}
```

find und *enter* sind so einfach, daß man sie sofort implementieren kann.

```
Element find (String word) {
    Element e = first;
    while (e != null && !word. equals( e. word))
        e = e. next;
        // e == null || word. equals( e. word)
    return e;
}
```

```
void enter (String word) {
    Element e = new Element( word);
    e. next = first;
     first = e;
}
```

Beispiel (Forts.)

4. Zerlege Teilaufgaben weiter (print).
   Ist so einfach, daß man es sofort implementieren kann!

```
void print () {
   for (Element e = first; e != null; e = e. next)
       System.out. println( e. word + ": " + e. freq);
}
```

Zusammensetzen der einzelnen Teile

```
class Element {
    String word;
    int freq;
    Element next;
    Element (String w) {
       word = w; freq = 1;
    }
}
```

```
class WordTable {
    Element first = null;

    Element find(String word) {
        Element e = first;
        while (e != null && !word. equals( e.word)) e = e. next;
        return e;
     }

    void enter(String word) {
       Element e = new Element( word);
       e. next = first; first = e;
    }

    void count(String word) {
       Element e = find( word);
       if (e == null) enter( word); else e. freq++;
     }

    void print() {
        for (Element e = first; e != null; e = e. next)
        System.out. println( e. word + ": " + e. freq);
   }
}
```

Zusammensetzen der einzelnen Teile

```
class WordCount {

    public static void main (String[] arg) {
        WordTable table = new WordTable();
        In. open(" input. txt");
        String w = readWord();
        while (w != null) {    table. count( w);   w = readWord();   }
        In. close();
        table. print();
    }

    static String readWord () {
         StringBuffer word = new StringBuffer();
          char ch;
          do ch = In. read(); while (In. done() && ! Character. isLetter( ch));
          while (In. done() && Character. isLetter( ch)) {
              word. append( ch);
              ch = In. read();
          }
          if (word. length > 0) return word. toString(); else return null;
     }
}
```