Einführung in die Programmierung

by André Karge

Übung - Enum, Foreach, Recursion, Listen

Heute

- Enum
- Foreach
- Recursion
- Listen

Enum & Foreach

Enum Typen - Insel 9.4

Beschreibung

- Enum typen sind Gruppierungstypen zusammengehöriger Konstanten
- Beispiel: Wochentage, Farben, Monate, Jahreszeiten, usw.
- Dafür brauche man nicht extra Member-Konstanten oder Klassen erzeugen
- Stichwort: *enum*

Enum Typen - Insel 9.4

Beschreibung

- Enum typen sind Gruppierungstypen zusammengehöriger Konstanten
- Beispiel: Wochentage, Farben, Monate, Jahreszeiten, usw.
- Dafür brauche man nicht extra Member-Konstanten oder Klassen erzeugen
- Stichwort: *enum*

```
public enum Wochentag {
  Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag;
  // 0  , 1       , 2       , 3         , 4      , 5      , 6
}

public class MeinProgramm {
  public static void main(String[] args) {
    Wochentag heute = Wochentag.Donnerstag; // Initialisierung über Konstante
    Wochentag morgen = Wochentag.values()[4]; // Initialisierung über Index
    System.out.println("heute ist " + heute);
  }
}
```

ForEach - Insel 3.8.10

- For-Schleifen über Elemente eines Arrays / einer Liste

```
String[] wochentage = {"Montag", "Dienstag", "Mittwoch", "Donnerstag",
                       "Freitag", "Samstag", "Sonntag"};
for(int i=0; i<wochentage.length; i++) {
  System.out.println(wochentage[i]);
}
```

ForEach - Insel 3.8.10

- For-Schleifen über Elemente eines Arrays / einer Liste

```
String[] wochentage = {"Montag", "Dienstag", "Mittwoch", "Donnerstag",
                       "Freitag", "Samstag", "Sonntag"};
for(int i=0; i<wochentage.length; i++) {
  System.out.println(wochentage[i]);
}
```

- Einfacher mit ForEach-Schleifen
- Iterativer Zugriff auf jedes Element in der Liste über Schleifenvariant *tag*

```
for(String tag : wochentage) {
  System.out.println(tag);
}
```

Recursion

Recursion - Insel 2.7.14

(Source: https://xkcd.com/1557/)

Recursion - Insel 2.7.14

Beispiel

Auf dem Weg durch den Wald treffen wir eine Fee, die uns 3 Wünsche gibt. Man könnte jetzt sich 3 Explizite Wünsche ausdenken. Damit wären die 3 Wünsche aber verbraucht.
Was wäre, wenn wir uns mit dem letzten Wunsch nochmal 3 Wünsche wünschen würden? Das könnten wir so lange machen, bis alle Wünsche der Welt erfüllt sind.

```
static void feenWunsch() {
  wunsch(); // Frieden auf der Welt
  wunsch(); // Beim Programmieren nicht mehr Semikolon vergessen
  feenWunsch(); // 3 Wünsche
}
```

Recursion - Insel 2.7.14

- Rekursion ist der Aufruf einer Methode in sich selbst
- kann die Laufzeit des Programmes verändern
- Bei jedem Methodenaufruf wird die aufgerufene Methode auf dem Stack abgelegt (Bei Rekursion also immer wieder die selbe Methode - ohne Sie wieder vom Stack zu entfernen)
- Kann bei zu tiefer Rekursion einen Stack Overflow hervorrufen

Recursion - Insel 2.7.14

Übungsaufgabe (15min)

Schreiben Sie eine Klasse Guess mit Methoden, die eine von Ihnen gedachte Zahl zwischen 1 und 1000 erraten.
Dazu sollen die Methoden eine Zahl vorschlagen und Sie antworten mit $<$, $>$ oder $=$

1. Implementieren Sie eine Methode mit einer while-Schleife
2. Implementieren Sie eine Methode mit Recursion

Das Programm benötigt hier keinen Zufallswert. Überlegen Sie sich eine Taktik, mit der Sie möglichst schnell auf die Lösung kommen.

Fibonacci Formel

$$f_n = f_{n-1} + f_{n-2} \text{ für } n > 2$$
$$f_1 = f_2 = 1$$

Beispiel

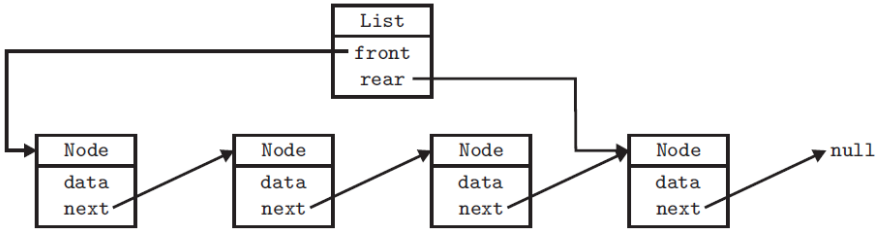
$$f_1 = 1 \tag{1}$$
$$f_2 = 1 \tag{2}$$
$$f_3 = f_2 + f_1 = 1 + 1 = 2 \tag{3}$$
$$f_4 = f_3 + f_2 = (f_2 + f_1) + f_2 = (1 + 1) + 1 = 3 \tag{4}$$
$$f_5 = f_4 + f_3 = 3 + 2 = 5 \tag{5}$$

Listen

Listen

Beschreibung

- Arbeit mit Daten unbestimmter Länge / unterschiedlicher Länge
- Bessere Operationen im Vergleich zu Arrays (insert, delete, push, pop, ...)
- Ist ein Datenobjekt, welches Referenzen speichert



Listen

Beispiel

```
public class Node { // Container für Integer Listenelement
  private int data; // Inhalt eines Listenelements
  private Node next; // Referenz auf das nächste Element in der Liste
  //...
}
public class List {
  private Node front; // Referenz auf das erste Element in der Liste
  private Node rear; // Referenz auf das letzte Element in der Liste
  private int nodeCount; // Übersicht, wie viele Nodes wir haben
  public List() {
    this.front = null;
    this.rear = null;
    this.nodeCount = 0;
  }
}
```

Listen

Übungsaufgabe (15 min)

1. Schreiben Sie ihre eigene Klasse für eine einfach verkettete Liste für Integer (eine *List*-Klasse und eine *Node*-Klasse)
2. Achten sie auf korrekte Sichtbarkeit (public vs. private) der Attribute und Methoden
3. Schreiben Sie jeweils eine Methode um Nodes am Ende, zu einem bestimmten Index und am Anfang der Liste einzufügen
4. Schreiben Sie eine Methode, die ein Element anhand eines übergebenen Index' zurück gibt
5. Testen Sie die Liste in einer geeigneten main-Methode

Fragen?