Hochschule Harz	**FB Automatisierung und Informatik**
Programmierung2	Dipl.-Inf., Dipl.-Ing. (FH) M. Wilhelm
Tutorial / Hausaufgabe 06:	„Programmierung 2“ für MI / WI Thema: **Einfachverkettete Liste**

Versuchsziele

Kenntnisse in der Anwendung von:

- einfach verketteten Listen
  - manueller Aufbau von Listen
  - manuelle Ausgabe der Liste in einem Editor
  - Aufbau einer Klasse „SimpleList“
    - Implementierung von Löschmethoden
    - Implementierung von Einfügen (add, addSort, addLast)
    - Implementierung von Abfragemethoden

Tutorial- Hausaufgabe06: einfache Listen

In dieser Aufgabe soll eine grafische Oberfläche für das Testen einer zu implementierenden Liste entwickelt werden. Benötigt werden zwei Aktionen für das Tutorial und drei für die Hausaufgabe. Neben dem Schließen sind es also sechs Aktionen die „verdrahtet“ werden müssen. Die Technik dafür ist komplett Ihnen überlassen.

Aufgaben

1. Teilaufaufgaben: Projekt erstellen und aufbauen: Zeit 3 Minuten

- Erstellen Sie ein neues Eclipse–Projekt:
  - Projektname: Aufgabe06
- Erstellen Sie eine neue Klasse:
  - Menü File, Eintrag New, Eintrag class
  - Name: Aufgabe06
- folgenden Klassen müssen im Laufe des Projektes erstellt werden:
  - Aufgabe06 (Tutorial)
  - Node (Tutorial)
  - SimpleList (Hausaufgabe)

2. Teilaufaufgaben: Event-Methoden: Zeit 10 Minuten

- Bauen Sie in Ihr Projekt sechs Auswahl-Elemente ein. Damit sollen sechs Event-Methoden angesteuert werden.
- Die Technik (Menüs, Schalter oder andere) ist beliebig.
- Namen der Events:
  - Tutorial_test1
  - Tutorial_test2
  - Aufgabe_test1
  - Aufgabe_test2
  - Aufgabe_test3
- Erstellen und verknüpfen Sie die sechs Event-Methoden
- **Sie können die auf meiner Homepage zur Verfügung gestellten Java-Beispiele benutzen.**

Klassenübersicht:

- **Aufgabe06**
  - Hauptframe
    - verwaltet die GUI-Elemente.
    - steuert die Event-Methoden.
    - hat, optional, auch eine Middleware.
    - beinhaltet die sechs Event-Methoden.

- **Node**
  - speichert ein Datensatz vom Typ „Object“.
  - verweist auf ein anderen Node (next).
  - Attribute:
    - Node next;
    - Object data; // sollte später geändert werden

    Bietet folgende Methoden an:
    - Konstruktor
    - toString

- **SimpleList**
  - Verwaltet eine lineare Liste.
  - Attribute:
    - Node root
    - Node last; // nur für die Hausaufgabe (addLast)
  - Bietet Methoden für die Änderungen an:
    - delete (mit Rückgabewert, ob die Löschung erfolgreich war)
    - toString()
    - setRoot
    - get(int index)
    - add
    - clear()
    - addLast
    - addSort

3. Teilaufaufgabe: Tutorial_test1

Diese Aufgabe baut manuell eine Liste auf, um das Schema einer linearen Liste zu erkennen. Nach dem Erstellen der Liste soll diese dann in dem Editor ausgegeben werden. Im nächsten Schritt soll mit der zu implementierenden Methode „searchNode“ ein Knoten gesucht werden.

1. Teil:

- Erstellen Sie die Klasse „**Node**“ mit den notwendigen Attributen und Methoden.
  - speichert ein Datensatz vom Typ „Object“. Der Datentyp wird später geändert.
  - verweist auf ein anderen Node (next).
  - Attribute:
    - Node next;
    - Object data;

    Bietet folgende Methoden an:
    - Konstruktor
    - toString
- In der Methode „Tutorial_test1“ werden genau fünf Objekte vom Typ „Node“ mit den Datenobjekten: „Node1“, „Node2“, „Node3“, „Node4“ und „Node5“ erstellt.
- Verknüpfen Sie diese Nodes, in Reihenfolge 1-5 miteinander. Dazu muss die Referenz „next“ lediglich auf den nächsten „Node“ zeigen.
- Geben Sie die Liste in einer Schleife in dem Editor aus.

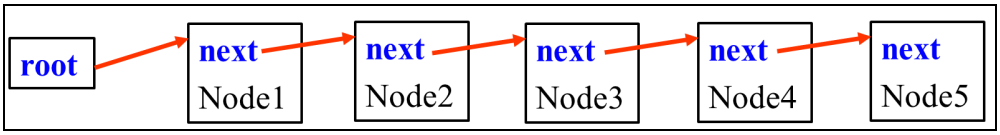

Abbildung 1 Prinzip-Skizze

Ausgabe:

```
Tutorial_test1:
Node1
Node2
Node3
Node4
Node5
```

2. Teil:

- Diese Liste von fünf Einträgen soll nun durchsucht werden. Dazu brauchen wir eine neue Methode namens searchNode.

- Erstellen Sie die Methode „searchNode“ in der Klasse „Aufgabe06“. Als Parameter wird ein Knoten „root“ und ein Objekt übergeben. Der Knoten symbolisiert die Liste, in dem nach dem Objekt gesucht wird. Diese Methode soll angefangen von der Wurzel der Liste die Listeneinträge nach dem gesuchten Datenobjekt durchsuchen. Wenn dieser Listeneintrag gefunden wurde, wird genau dieser Listeneintrag zurückgegeben, ansonsten null.

- In der Methode „Tutorial_test1“ wird diese Methode nun aufgerufen. Verwenden Sie dort die Methode „searchNode“, um nach dem "**Node3**" zu suchen. Geben Sie im Editor aus, ob dieser Node gefunden bzw. nicht gefunden wurde. Dazu reicht es den Rückgabewert von der Methode searchNode mit einer if-Abfrage auf ungleich null überprüfen. Die Ausgabe im Editor erfolgt durch editor.append(„...“).

- Wiederholen Sie den Suchvorgang noch einmal für eine Knoten „**Node13**“. Die zu erwartende Ausgabe ist, das Node3 gefunden wurde und Node13 nicht gefunden wurde.

Ausgabe des Suchens:
Suchen:
Node 3 gefunden
Node 13 nicht gefunden

Komplettausgabe
Tutorial_test1:
Node1
Node2
Node3
Node4
Node5

Suchen:
Node 3 gefunden
Node 13 nicht gefunden

4. Teilaufaufgabe: Tutorial_test2

Diese Aufgabe benutzt die Kenntnisse der ersten Aufgabe und implementiert eine Klasse „SimpleList“, in der die Methoden „clear“, „setRoot“, „toString“ und „delete“ implementiert werden sollen. Nach dem Aufbau der Klasse sollen die Methode getestet werden. Ein Testszenario baut manuell eine Liste auf und setzt das Root in der Liste. Getestet wird in einer Schleife die Methode „delete“ und „toString“., um das Schema einer linearen Liste zu erkennen. Nach dem Erstellen der Liste soll diese dann in dem Editor ausgegeben. Im nächsten Schritt soll mit der zu implementierenden Methode „searchNode“ ein Knoten gesucht werden. In einer Schleife werden Knoten mittels Indizes gesucht. Dabei soll auch der Fehlerfall berücksichtigt werden.

- Erstellen Sie die Klasse „SimpleList“
  - Attribute:
    - Node root
  - Methoden:
    - setRoot setzt den Root auf eine Liste
      - Erfordert einen Parameter
    - clear() löscht **alle** Knoten
    - toString()
    - boolean delete(int index) Löschen des i-te Knotens
      - Rückgabewert, ob die Löschung erfolgreich war

- Erstellen Sie in der Klasse „Aufgabe06“ die Methode „testCase1“. Diese soll die Knoten des Test1 in eine Liste eintragen. Die Liste wird als Parameter übergeben.

- In der Methode „Tutorial_test2“ erstellen Sie eine Instanz der Klasse „SimpleList“. Implementieren Sie eine Schleife, die von 0-7 läuft. In jedem Schleifendurchlauf soll die Liste mit der Methode „testcase1“ initialisiert werden. Danach soll in der Liste versucht werden, das Element an der i-ten Stelle zu entfernen. Sollte das Löschen nicht erfolgreich sein, soll eine entsprechende Meldung in dem Editor ausgegeben werden. Dazu kann man den Rückgabewert der Methode „delete“ vergleichen. Zur Kontrolle wird danach die eventuell verkürzte Liste in dem Editor ausgegeben werden, liste+"", (eine Zeile).

Ausgabe:

```
Tutorial_test2:
Original:
Node1
Node2
Node3
Node4
Node5

delete: 0
Node2
Node3
Node4
Node5

delete: 1
Node1
Node3
Node4
Node5

delete: 2
Node1
Node2
Node4
Node5

delete: 3
Node1
Node2
Node3
Node5

delete: 4
Node1
Node2
Node3
Node4

delete: 5
Das Loeschen ist
fehlgeschlagen
Node1
Node2
Node3
Node4
Node5

delete: 6
Das Loeschen ist
fehlgeschlagen
Node1
Node2
Node3
Node4
Node5

delete: 7
Das Loeschen ist
fehlgeschlagen
Node1
Node2
Node3
Node4
Node5
```

Hausaufgabe 06

5. Teilaufaufgabe: Aufgabe_test1

In dieser Aufgabe wird die Methode „add“ implementiert und getestet. Drei „Knoten“ werden unsortiert in die Liste eingefügt. Nach jedem Einfügen soll zur Kontrolle die gesamte Liste ausgegeben werden.

- Implementieren Sie in der Klasse „SimpleList“ die Methode „add“
    - Parameter: Object
    - Inhalt: der neue Knoten soll immer ans Ende angehängt werden.
- Sie können natürlich eine weitere Methode in der Klasse „SimpleList“ erstellen.
- JFrame „Aufgabe06“, Methode „Aufgabe_test1“
    - Fügen Sie drei Strings in die Liste ein.
    - Geben Sie das Ergebnis jeweils in dem Editor aus.
        - 123
        - 13
        - 33

Beim Einfügen existieren folgende Fälle:

- Die Liste ist leer.
- Der neue Knoten muss ans Ende eingefügt werden.

Ergebnis:

```
Original: add 123 / 13 / 33
123

123
13

123
13
33
```

6. Teilaufaufgabe: Aufgabe_test2

In dieser Aufgabe wird die Methode „addSort“ implementiert und getestet. Sechs Integer-„Knoten“ werden sortiert in die Liste eingefügt. Der kleinste Wert steht am Anfang, der größte Wert am Ende der Liste. Nach jedem Einfügen soll zur Kontrolle die gesamte Liste ausgegeben werden. Da die Klasse „Object“ nicht sortiert werden kann, muss der Datentyp des Attributs „data“ hier geändert werden. Überlegen Sie, welcher Datentyp resp. Schnittstelle hier erforderlich ist.

- Implementieren Sie in der Klasse „SimpleList“ die Methode „addSort“
    - Parameter: nicht Object
    - Inhalt: der neue Knoten soll nach dem Wert in die Liste einsortiert werden
    - Um die Werte zu sortieren, benötigen Sie die Methode „compareTo“.
    - Sie können natürlich eine weitere Methode in der Klasse „SimpleList“ erstellen.
- JFrame „Aufgabe06“, Methode „Aufgabe_test2“
    - Fügen Sie **sechs Integer-Zahlen** in die Liste ein
    - Geben Sie das Ergebnis jeweils in dem Editor aus.
        - 523
        - 13
        - 200
        - 66
        - 33
        - 44

Beim Einfügen existieren folgende Fälle:

- Die Liste ist leer
- Der neue Knoten ist „kleiner“ als der erste Knoten.
- Der neue Knoten liegt in der „Mitte“, also zwischen zwei Knoten
- Der neue Knoten ist größer als alle vorhandenen. Er muss ans Ende eingefügt werden.

Ergebnis:

```
Original: Add: 523 / 13 / 200 / 66/ 33 / 44
523

11
523

11
200
523

11
66
200
523

11
33
66
200
523

11
33
44
66
200
523
```

5. Teilaufaufgabe: Aufgabe_test3

In dieser Aufgabe werden die Methode „add“ und „addLast“ bezüglich der Performance mit 100000 Daten getestet (Konstante MAX). In jeweils einer Schleife werden MAX-Daten in die Liste eingefügt. Mit „System.nanoTime“ wird die benötigte Zeit gemessen.

- **Diese Aufgabe ist optional.**
- Hier werden die beiden Methoden „add“ und „addLast“ getestet. Als Referenz wird dazu noch die in Java implementierte Liste „LinkedList“ getestet.
- Addieren Sie MAX-Werte in eine Liste mittels der Methode „add“. Geben Sie die benötigte Zeit in dem Editor aus.
- Addieren Sie MAX-Werte in eine Liste mittels der Methode „addLast“. Geben Sie die benötigte Zeit in dem Editor aus.
- Addieren Sie MAX-Werte in eine Liste mittels der Methode „addLast“ der Klasse „LinkedList“. Geben Sie die benötigte Zeit in dem Editor aus.

Ergebnis mit MAX=5:

Test Zeiger auf den letzten Knoten
Zeit add: 179971

Zeit addLast: 5864
Zeit addLast: Faktor: 30

Zeit addLast LinkedList: 9897
Zeit addLast LinkedList: Faktor: 18

Ergebnis mit MAX=100000:

Test Zeiger auf den letzten Knoten
Zeit add: 10601008280

Zeit addLast: 1620109
Zeit addLast: Faktor: 6543

Zeit addLast LinkedList: 6381102
Zeit addLast LinkedList: Faktor: 1661

Ergebnis mit MAX=500000:

Test Zeiger auf den letzten Knoten
Zeit add: 420193343113

Zeit addLast: 98538530
Zeit addLast: Faktor: 4264

Zeit addLast LinkedList: 107538930
Zeit addLast LinkedList: Faktor: 3907

Fast vollständiger Rahmen:

```
private void Aufgabe_test3() {
    final int MAX = 100000;
    long t1, t2, told;
    SimpleList liste = new SimpleList();
    editor.setText("Test Zeiger auf den letzten Knoten\n");
    t1 = System.nanoTime();
    for (…) {
     liste.add(…);
    }
    t2 = System.nanoTime();
    told=t2-t1;
    editor.append("Zeit add: "+told+"\n" );

    t1 = System.nanoTime();
    for (…) {
     liste.addLast(…);
    }
    t2 = System.nanoTime();
    editor.append("\nZeit addLast: "+(t2-t1)+"\n" );
    editor.append("Zeit addLast: Faktor: "+told/(t2-t1)+"\n" );

    LinkedList<Integer> liste2 = new LinkedList();
    t1 = System.nanoTime();
    for (…) {
     liste2.addLast(…);
    }
    t2 = System.nanoTime();
     if (MAX<20) editor.append(liste + "\n");
    editor.append("\nZeit addLast LinkedList: "+(t2-t1)+"\n" );
    editor.append("Zeit addLast LinkedList: Faktor: "+told/(t2-t1)+"\n" );
  }
```