Universität
Osnabrück

Informatik B - Objektorientierte Programmierung in Java

Vorlesung 08:
Collections 2

Inhalt

- Collections
  - Implementierung linearer Listen
  - Verkettete Liste
  - Innere Klassen
  - „Feld“-Liste
  - Anhang: ArrayList oder LinkedList?
  - Beispiele

Implementierung linearer Listen

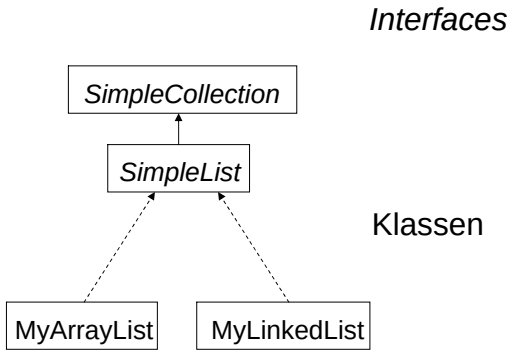
- Im folgenden werden verschiedene Möglichkeiten zur Implementierung linearer Listen in Java vorgestellt.
- Dies entspricht einem „Blick hinter die Kulissen“ der Java-Klassenbibliothek.
- Aus Gründen der Übersichtlichkeit wird die Implementierung linearer Listen anhand eines vereinfachten Ausschnitts der Java-Klassenbibliothek demonstriert.

Implementierung linearer Listen

- Der vereinfachte Ausschnitt der Java-Klassenbibliothek besteht aus:
  - Interface `SimpleCollection`:
    - Reduzierte Version des Interface `Collection`
  - Interface `SimpleList`:
    - Reduzierte Version des Interface `List`
  - Klasse `MyArrayList`:
    - Implementierung einer linearen Liste mit Hilfe eines Feldes
  - Klasse `MyLinkedList`:
    - Implementierung einer einfach verketteten Liste

Implementierung linearer Listen

- Klassenhierarchie



Interface SimpleCollection

```
public interface SimpleCollection
{
  public boolean add(Object o);

  public boolean remove(Object o);

  public boolean isEmpty();

  public int size();

  public boolean contains(Object o);

  public SimpleIterator iterator();
}
```


Interface SimpleList

```
public interface SimpleList
extends SimpleCollection
{
  public void add(int index, Object o);

  public Object remove(int index);

  public Object get(int index);

  public int indexOf(Object o);
}
```


Interface SimpleIterator

```
public interface SimpleIterator
{
  public boolean hasNext();

  public Object next();
}
```

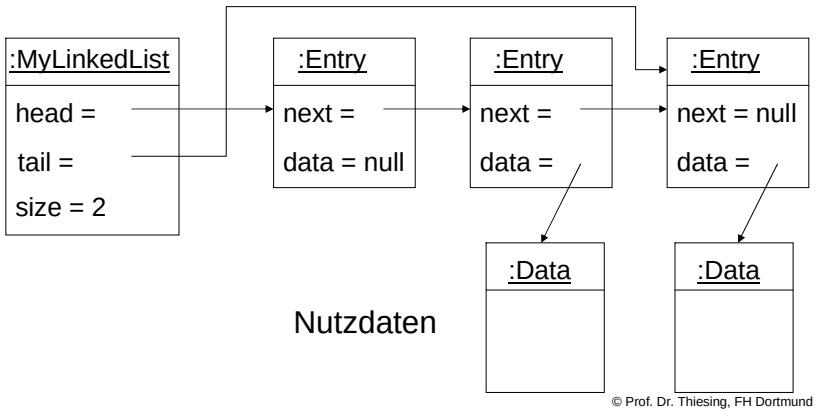
Verkettete Liste

- An der Realisierung einer linearen Liste als einfach verkettete Liste sind Objekte verschiedener Klassen beteiligt:
  - ein Objekt der eigentlichen Listenklasse (`MyLinkedList`)
  - ein oder mehrere Objekte der Klasse, die die einfach verkettete Liste durch Verkettung über Referenzen realisiert (`Entry`)
  - die Objekte, die als Nutzdaten in der Liste gespeichert sind, und einer beliebigen von `Object` abgeleiteten Klasse angehören (hier als `Data` bezeichnet)

Verkettete Liste

- Aufbau

Einfach verkettete Liste



Verkettete Liste

- Klasse `MyLinkedList`
  - verwaltet je einen Verweis auf den Anfang und das Ende der einfach verketteten Liste

```
public class MyLinkedList
implements SimpleList
{
  // Attribute
  private Entry head; // Anfang der Liste
  private Entry tail; // Ende der Liste
  private int size;   // Anzahl der Elemente

 ...
}
```


Verkettete Liste

- Die leere Liste besteht aus einem Objekt vom Typ `Entry`, dem kein Objekt als Nutzdaten zugeordnet ist.
- Dieses Objekt dient zur Vermeidung von Sonderfällen für die leere Liste beim Einfügen und Entfernen von Elementen der Liste.
- Konstruktor:

```
public MyLinkedList()
{
     this.head = new Entry(null, null);
     this.tail = this.head;
     this.size = 0;
}
```

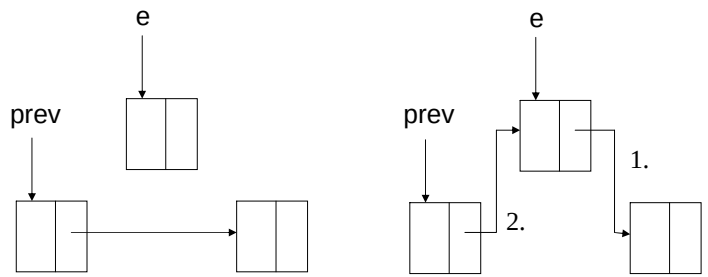
Verkettete Liste

- Klasse `Entry`
  - Innere Klasse der Klasse `MyLinkedList`, da die Elemente der einfach verketteten Liste nur in diesem Kontext einen Sinn ergeben:

```
public class MyLinkedList
implements SimpleList
{
  class Entry
  {    // Attribute
       Object data;  // Verweis auf Nutzdaten
       Entry next;   // Verweis auf Nachfolger
       // Konstruktor
       Entry(Object o, Entry next)
       {
       this.data = o;
       this.next = next;
      }
  }  ...
}
```

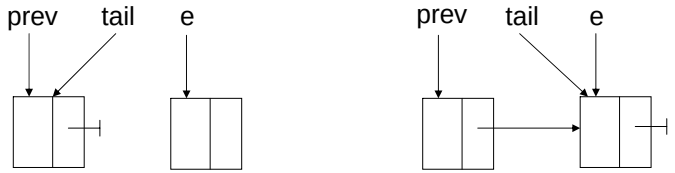

Verkettete Liste

- Grundoperationen
  - Einfügen eines Elements
    - Einfügen in der Mitte der Liste



Verkettete Liste

- Grundoperationen
  - Einfügen eines Elements
    - Einfügen am Ende der Liste



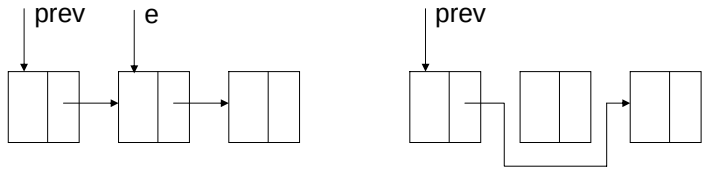
Verkettete Liste

- Grundoperation
  - Einfügen eines Elements

```
private void addAfter(Object o, Entry prev)
{
  Entry e = new Entry(o, prev.next);
  prev.next = e;
  if (prev==this.tail)
       this.tail = e;  // Listenende korrigieren
  this.size++;
}
```

Verkettete Liste

- Grundoperationen
  - Entfernen eines Elements
    - Entfernen in der Mitte der Liste



    - Entfernen am Ende der Liste

Verkettete Liste

- Grundoperationen
  - Entfernen eines Elements

```
private Object removeAfter(Entry prev)
{
   Object result;

   Entry e = prev.next;
   result = e.data;

   prev.next = e.next;
   if (e==this.tail)
       this.tail = prev; // Listenende korrigieren
   this.size--;

   return result;
}
```


Verkettete Liste

- Grundoperationen
  - Positionierung auf ein Element mit vorgegebenem Index

```
private Entry entry(int index)
{
   Entry e = this.head;

   for (int i=0; i<=index; i++)
      e = e.next;

   return e;
}
```


Verkettete Liste

- Operationen
  - Mit Hilfe der Grundoperationen können weitere Operationen, wie z.B. das Einfügen an einem bestimmten Index realisiert werden:

```
public void add(int index, Object o)
{
   if (o!=null)
   {
       // Überprüfung, ob index gültig ist
       rangeCheck(0, index, this.size);

       Entry prev = entry(index-1);
       addAfter(o, prev);
   }
}
```

Verkettete Liste

- Iterator
  - Innere Klasse der Klasse `MyLinkedList`, da der Iterator nur im Kontext der Klasse `MyLinkedList` Sinn macht:

```
public class MyLinkedList
{ ...
  class Iter
  implements SimpleIterator
  {
       // Attribut
       Entry e; // aktuelle Position des Iterators

       // Konstruktor
       Iter()
       { e = head;} ...
  }
}
```


Verkettete Liste

- Iterator
  - Operationen

```
public boolean hasNext()
{
     return e.next!=null;
}

public Object next()
{
     Object result = null;
     if (e.next!=null) {
        e = e.next;
        result = e.data;
     }
     return result;
}
```


Innere Klassen `Entry` und `Iter`

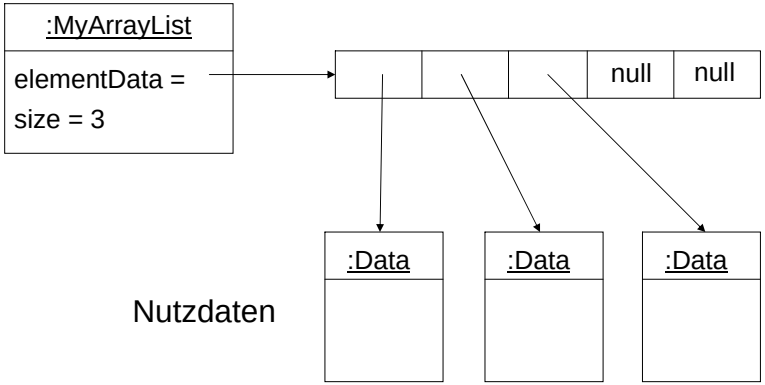
- Innere Klassen werden innerhalb einer anderen Klasse deklariert.
- Innere Klassen wie `Entry` und `Iter` werden auch als Mitgliedsklassen (engl. member classes) bezeichnet.
- Eine solche Mitgliedsklasse kann auf alle Attribute der äußeren Klasse (hier: `MyLinkedList`) zugreifen.
- Dies gilt auch für private Attribute der äußeren Klasse!

„Feld“-Liste

- An der Realisierung einer linearen Liste mit Hilfe eines Feldes sind Objekte verschiedener Klassen beteiligt:
  - ein Objekt der eigentlichen Listenklasse (`MyArrayList`)
  - ein Feld, das die Verweise auf die Nutzdaten enthält
  - die Objekte, die als Nutzdaten in der Liste gespeichert sind, und einer beliebigen von `Object` abgeleiteten Klasse angehören (hier als `Data` bezeichnet).

„Feld“-Liste

- Aufbau



„Feld“-Liste

- Klasse `MyArrayList`
  - verwaltet einen Verweis auf das Feld, welches wiederum die Verweise auf die Nutzdaten enthält:

```
public class MyArrayList
implements SimpleList
{
  // Attribute
  private Object[] elementData; //Feld für Verweise
  private int size;             //Anzahl der Elemente

  // Konstruktor
  public MyArrayList(){
    this.elementData = new Object[10]; //Anfangsgröße
    this.size = 0;
  } ...
}
```


„Feld“-Liste

- Grundoperationen
  - Erweiterung des Feldes

```
private void ensureCapacity(int minCapacity)
{
     int oldCapacity = this.elementData.length;

     if (minCapacity>oldCapacity) {
        Object oldData[] = this.elementData;
        ... // evtl. zusätzliche Strategie
        int newCapacity = minCapacity;
        this.elementData = new Object[newCapacity];

        // Kopieren der Daten in das neue Feld
        for (int i=0; i<oldCapacity; i++)
            this.elementData[i] = oldData[i];
     }
}
```


„Feld“-Liste

- Grundoperationen
  - Elemente um 1 nach links schieben

```
private void shiftLeft(int index)
{
   for (int i=index; i<this.size-1; i++)
      this.elementData[i] = this.elementData[i+1];
}
```

  - Elemente um 1 nach rechts schieben

```
private void shiftRight(int index)
{
   for (int i=this.size; i>index; i--)
      this.elementData[i] = this.elementData[i-1];
}
```

„Feld“-Liste

- Operationen
  - Einfügen eines Elements

```
public void add(int index, Object o)
{
  if (o!=null) { ...
     // Ausreichende Größe des Feldes sicherstellen
     ensureCapacity(this.size+1);

     // ab Position index um 1 nach rechts schieben
     shiftRight(index);

     // Objekt an der Position index speichern
     this.elementData[index] = o;
     this.size++;
  }
}
```


„Feld“-Liste

- Iterator
  - Innere Klasse der Klasse `MyArrayList`, da der Iterator nur im Kontext der Klasse `MyArrayList` Sinn macht:

```
public class MyArrayList
{ ...
  class Iter
  implements SimpleIterator
  {
    // Attribut
    int pos;  // aktuelle Position des Iterators

    //Konstruktor
    Iter()
    { pos = -1;}
    ...
}
```


„Feld“-Liste

- Iterator
  - Operationen

```
public boolean hasNext()
{
     return pos<size-1;
}

public Object next()
{
     Object result = null;

     pos++;
     if (pos<size)
        result = elementData[pos];
     return result;
}
```


Anhang: ArrayList oder LinkedList?

- Zurück zur doppeltverketteten Liste **`LinkedList`** und der **`ArrayList`** aus dem Framework. Es gibt zwei Entscheidungskriterien bei der Frage, welche Implementierung einer Liste gewählt werden sollte: Laufzeit und Speicherplatz.
- Die Entscheidung ist nicht einfach und nicht immer eindeutig. Darum sollten Sie Ihr Programm so gestalten, dass Sie die Implementierung der Liste leicht ändern können: siehe unten Programmiertipp.

Anhang: ArrayList oder LinkedList ?

- Laufzeit – das Verhalten bei drei verschiedenen Operationen

Operationen auf 25000 Elementen	ArrayList	LinkedList
Einfügen am Ende	0.3	0.3
Einfügen am Ende (1. Durchlauf) Lesen mit get (2. Durchlauf)	0.4	21.3
Einfügen am Anfang (1. Durchlauf) Löschen am Anfang (2. Durchlauf)	6.9	0.3

- Speicherplatz
  ArrayList nutzt den Speicherplatz effektiver, weil LinkList eine zusätzliche Datenstruktur benötigt (Entry).

Anhang: ArrayList oder LinkedList?

- Programmiertipp
  - Sorgen Sie dafür, dass Sie Ihr Programm leicht von einer zur anderen Implementierungsart ändern können. Verwenden Sie hierzu in Ihrem Programm die Interfaces statt die konkreten Klassen. Schreiben Sie z.B.

    List lst = new ArrayList ();

    und benutzen Sie "List" auch an allen anderen Stellen Ihres Programms (beispielsweise bei der Parametervereinbarung Ihrer Methoden).

Beispiele in

- SimpleCollection.java
- SimpleList.java
- SimpleIterator.java
- MyLinkedList.java
- MyArrayList.java
- ListTest.java