Listen

In Java implementieren gehören Listen zum Bereich der **Collections**. Es haben alle Untertypen von Collections einge gemeinsame Methoden, wie add oder get. Listen sind dynamische Datenstrukturen, deren Größe nicht initial angegeben werden muss. Das bedeutet, dass Listen mitwachsen und sich gegebenfalls auch verkleinern, um Speicher wieder freizugeben.

Listen können in den Variablen `List`, `ArrayList` / `LinkedList` werden. Es empfiehlt sich jedoch die kleinste gemeinsame `List` zu verwenden, damit der Listenart zu einem späteren Zeitpunkt leicht modifiziert werden kann.

```
1 | ArrayList <String> myList1 = new ArrayList<String>();
2 | LinkedList <String> myList2 = new LinkedList<String>();
3 |
4 | // Bessere Variante = Allgemeiner:
5 | List<String> myBestList = new ArrayList<String>();
```

Im folgenden wird primär auf die ArrayList eingegangen, da diese mit dem Verstänis von Arrays intuitiver erscheinen.

ArrayList

Eine ArrayList beruht auf der Datenstruktur Array. Hierbei wird im Hintergrund standarmäßig ein Array mit 10 Plätzen erstellt. Wenn die Menge der Einträge der Anzahl an Plätzen im Array übersteigt, so wird die ArrayList immer um den Faktor 1.5 vergrößert. Dies erfolgt im Hintergrund.

Erstellen

Wie beim Erstellen eines Arrays, muss auch bei Listen der Datentyp angegeben werden (`<String>`, `<MyClass>`, ...). Primitive Datentypen (`int`, `short`, `boolean`,...) müssen hierbei in sogenannte Wrapper-Klassen gepackt werden.

```java
import java.util.ArrayList; // import am Beginn der Klasse

List<String> strLst = new ArrayList<String>();
List<Integer> intLst = new ArrayList<Integer>();
List<Boolean> boolLst = new ArrayList<Boolean>();
List<MyClass> myLst = new ArrayList<MyClass>();
```

Wert hinzufügen/setzen

```java
strLst.add("Hello World");
intLst.add(1234);
boolLst.add(true);
myLst.add(new MyClass(...));

// Setzen von Wert auf bestimmten Index
strLst.set(0, "Hallo Welt");
```

Wert an Stelle 0 auslesen

```java
String str = strLst.get(0);
int i = intLst.get(0);
boolean b = boolLst.get(0);
MyClass my = myLst.get(0);
```

Liste durchlaufen

For-Schleife

```java
for(int i=0; i<myList.size(); i++){
  int aktuellesElement = myList.get(i);
  System.out.println(aktuellesElement);
}
```

ForEach

Die ForEach ist eine verkürzte Schreibweise, die immer ein nächstes Element liefert. (kein Index). Diese Variante wird verwendet, wenn nur die Elemente und nicht deren Indizes benötigt werden.

```java
for(int aktuellesElement : myList){
  System.out.println(aktuellesElement);
}
```

Werte löschen

```java
// Wert an Stelle 0 löschen
intLst.remove(0);

// erstes passende Objekt aus Liste löschen
strLst.remove("Hello World");

// Liste leeren => Elemente Löschen, nicht die Liste
intLst.clear();

intLst = null; // Liste wird vom GarbageCollector gelöscht
```

Abfragen

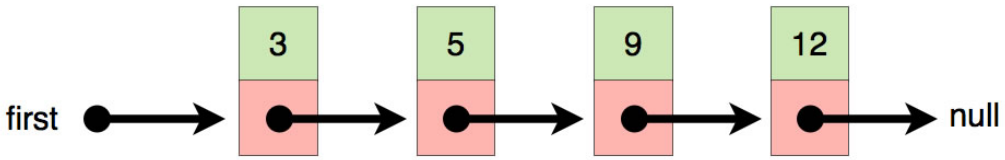
- `isEmpty()` prüft, ob die Liste leer ist (`size==0`)
- `strLst.contains("Hello World")` gibt true zurück, wenn die Liste den Wert beinhaltet

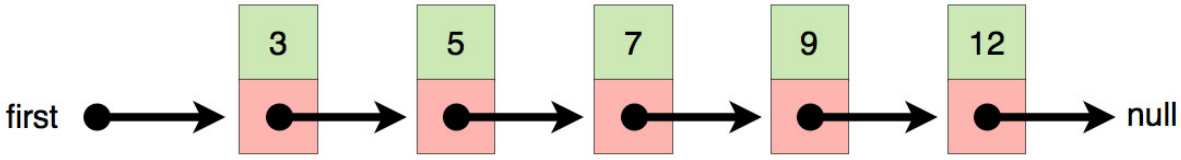
LinkedList

Die verkettete Liste ist um einiges dynamischer, da hier Klassen miteinander über einen "Link" verbunden werden. Verkettete Listen sind effizient, wenn in der Mitte Elemente hinzugefügt oder gelöscht werden.

Funktionalität

Jedes Element der Liste hat eine Referenz auf seinen Nachfolger. Da das letzte Element auf `null` zeigt, wird so das Ende der Liste illustriert. Die einfachse Version einer LinkedList ist die einfach verkettete Liste, die nur auf den Nachfolger referenziert. Eine doppelt verkettete Liste hat ebenso noch einen Verweis auf den jeweiligen Vorgänger. Diese Listen können auch effizient rückwärts durchlaufen werden, benötigen jedoch mehr Speicher.





Erstellen

```
1  import LinkedList; // import am Beginn der Klasse
2
3  List<String> strLst = new LinkedList<String>();
4  List<Integer> intLst = new LinkedList <Integer>();
5  List<Boolean> boolLst = new LinkedList <Boolean>();
6  List<MyClass> myLst = new LinkedList <MyClass>();
```

Den Großen Vorteil beim Speichern einer `LinkedList` im Datentyp `List` sind die vereinheitlichen Zugriffsmethoden. Erst wenn die benötigen Methoden von doppelt verketteten Listen benötigt werden, müssen die Listen spezifischer (`LinkedList<String> lst = ...`) angelegt werden.