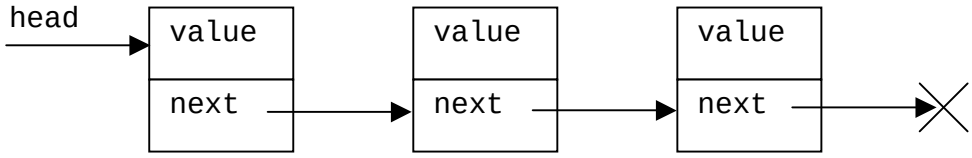
Dynamische Datenstrukturen

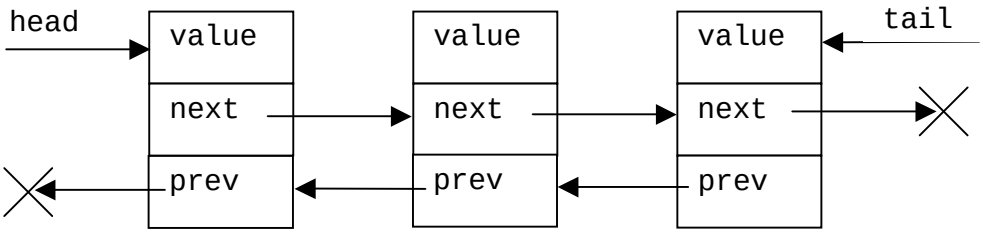
Einfach verkettete Liste:




```
class Node {
	int value;
	Node next;

	public Node(int value, Node next) {
		this.value = value;
		this.next = next;
	}
}
```

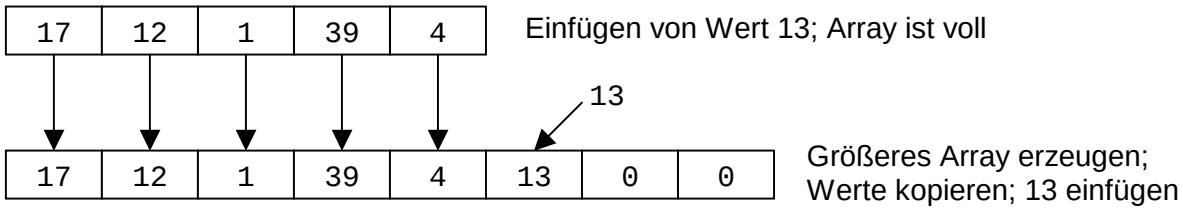
Doppelt verkettete Liste:




```
class Node {
	int value;
	Node next;
	Node prev;

	public Node(int value, Node next, Node prev) {
		this.value = value;
		this.next = next;
		this.prev = prev;
	}
}
```

Anwachsendes Array / Vektor



Bsp 1: Einfach verkettete Liste

Schreiben Sie eine Implementierung einer einfach verketteten Liste von Elementen des Typs `java.lang.Object` zu folgendem Interface:

```
package at.fhvie.progprakt.collection;

public interface Collection {
	/**
	 * Fügt ein Element an erster Stelle in die Collection ein.
	 * @param o Das einzufügende Element.
	 */
	void addFirst(Object o);

	/**
	 * Hängt ein Element am Ende der Collection an.
	 * @param o Das anzuhängende Element.
	 */
	void addLast(Object o);

	/**
	 * Fügt ein Element an einer bestimmten Position in die Collection ein.
	 * @param index Die Position, an der das Element eingefügt werden soll. Gültig
	 * von 0 bis size().
	 * @param o Das einzufügende Element.
	 * @throws InvalidIndexException Wird geworfen, wenn ein ungültiger Index
	 * übergeben wurde.
	 */
	void insertAt(int index, Object o) throws InvalidIndexException;

	/**
	 * Löscht ein Element an einer bestimmten Position.
	 * @param index Die Position, an der das Element gelöscht werden soll. Beginnt mit 0.
	 * @throws InvalidIndexException Wird geworfen, wenn ein ungültiger Index
	 * übergeben wurde.
	 */
	void deleteAt(int index) throws InvalidIndexException;

	/**
	 * Löscht alle Elemente in der Collection.
	 */
	void clean();

	/**
	 * Liefert das Element an einer bestimmten Position.
	 * @param index Die Position, von der das Element geliefert werden soll.
	 * @return Das Element an Position index.
	 * @throws InvalidIndexException Wird geworfen, wenn ein ungültiger Index
	 * übergeben wurde.
	 */
	Object getAt(int index) throws InvalidIndexException;

	/**
	 * Liefert die Elementanzahl der Collection.
	 * @return
	 */
	int size();
}
```

Das Interface steht Ihnen auch in einer Datei namens Blatt_12_Angabe.zip zur Verfügung. Erstellen Sie die Liste im Package `at.fhvie.progprakt.collection.liste`. Schützen Sie Ihre Node-Klasse, sodass diese nur innerhalb des Packages sichtbar ist. Die InvalidIndexEception müssen Sie selbst implementieren.
Testen Sie Ihre Liste aus dem Package `at.fhvie.progprakt.listetest` heraus.

Bsp 2: Doppelt verkettete Liste

Schreiben Sie eine Implementierung einer doppelt verketteten Liste zum Interface `Collection` aus Beispiel 1.

Bsp 3: Vektor

Schreiben Sie eine Implementierung eines Vektors zum Interface `Collection` aus Beispiel 1. Definieren Sie einen Konstruktor, um den Vektor mit spezifizierbarer initialer Kapazität zu erzeugen. Die Wachstumsfaktor des Vektors soll 2 sein, d.h. eine Verdoppelung.

Bsp 4: Performancetest

Öffnen Sie Beispiel 4 aus der Datei „Blatt_12_Angabe.zip“. Analysieren und Starten Sie das Programm; Interpretieren Sie die Messergebnisse.

Bsp 5: Zufallswerte

Erzeugen Sie eine java.util.ArrayList und befüllen diese mit 10 Zufallswerten (siehe `Math.random()`). Kopieren Sie die Werte in eine java.util.LinkedList und geben Sie die Werte der LinkedList auf die Console aus.

Bsp 6: Gehaltstabelle 2

Lösen Sie das Beispiel 2 „Gehaltstabelle“ von Blatt 8
-) mit einer java.util.LinkedList
-) mit einer Ihrer Klassen aus Bsp 1, 2 oder 3 von diesem Blatt

Setzen Sie die Containerklasse ein, um die 1-zu-n Relation zwischen Unternehmen und Mitarbeiter (Arbeiter/Manager) darzustellen.