Lineare Datenstrukturen
List

Lernziele

- Sie wissen, was unter «Lineare Datenstruktur» verstanden wird
- Sie kennen die Unterschiede zwischen einem Array und einer dynamischen Liste
- Sie wissen, wie Sie eine List implementieren können

Linear?

- Lineare Datenstruktur
  - ordnet die Elemente sequentiell an
  - es kann nur ein Element direkt angesprochen werden
  - Bsp: Array, LinkedList
- Nicht-Lineare Datenstruktur
  - keine sequentielle Struktur
  - jedes Element ist mit mehreren anderen Elementen verbunden
  - die Verbindung ist spezifisch zur abgebildeten Beziehung
  - Bsp: Tree, Graph

Array (1)

- Einfachster Typ einer Datenstruktur
- Eine Sammlung von aufeinanderfolgenden Elementen
- Speicherung in aufeinanderfolgenden Speicherbereichen
- Jedes Element hat den gleichen Datentyp
- Arraytypen
  - Eindimensionale Arrays
  - Multidimensionale Arrays
  - Jagged
- Anzahl Dimensionen und die Länge jeder Dimension werden bei der Erstellung festgelegt und können nicht geändert werden

Array (2)

- Eindimensionales Array

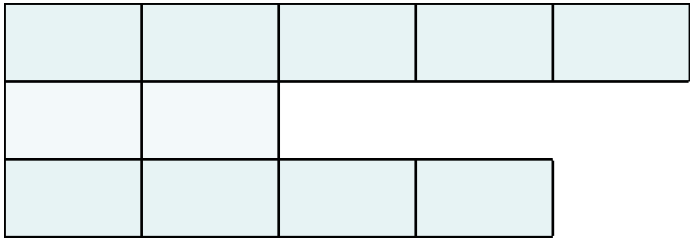
```
int[] array = new int[5];
```

- Multidimensionales Array

```
int[,] array = new int[2, 4];
int[,,] array1 = new int[2, 4, 3];
```

- Jagged Array

```
int[][] jaggedArray = new int[3][];
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[2];
jaggedArray[2] = new int[4];
```

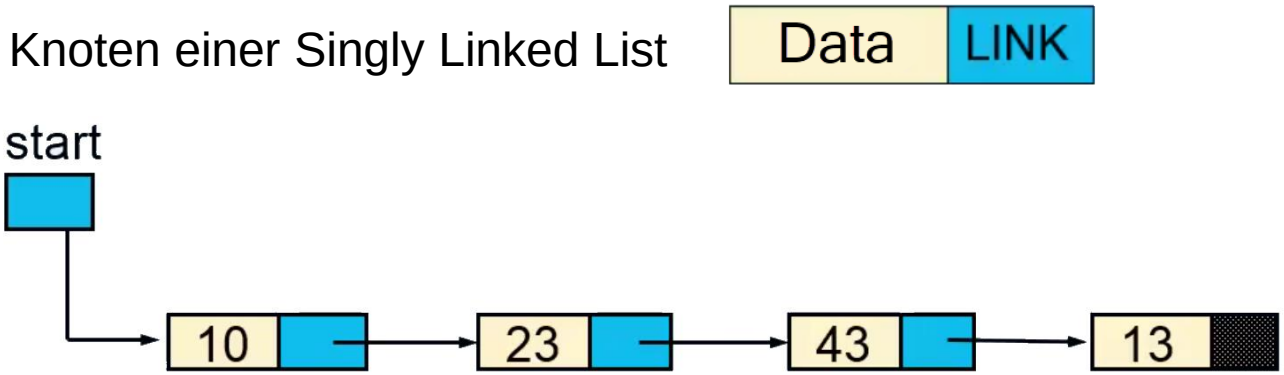


List

- beliebig viele Elemente, ohne vorher die Grösse festlegen zu müssen
- Daten einfügen und löschen ohne viel Aufwand
- Implementierung über Linked List oder Array

Singly Linked List (einfach verkettete Liste)

- Dynamische Datenstruktur bestehend aus Knoten
- Daten werden nicht in aufeinanderfolgenden Speicherbereichen gespeichert
- Einfügen und Löschen von Elementen ist einfacher als in Arrays
  - Keine Neuerstellung und umkopieren nötig
- Kann für die Implementierung von List, Stack, Queue verwendet werden



Implementierung

- Knoten

```
private sealed class Node {
    public object Data { get; set; }
    public Node Link { get; set; }
}
```

- Welche Methoden und Eigenschaften sollen für die Linked List implementiert werden?

```
public void Add(Object item)
public bool Contains(Object item)
public bool Remove(Object item)
public bool FindByIndex(int index)
public int Count
```

Add-Methode

```
public void Add(object data)
```

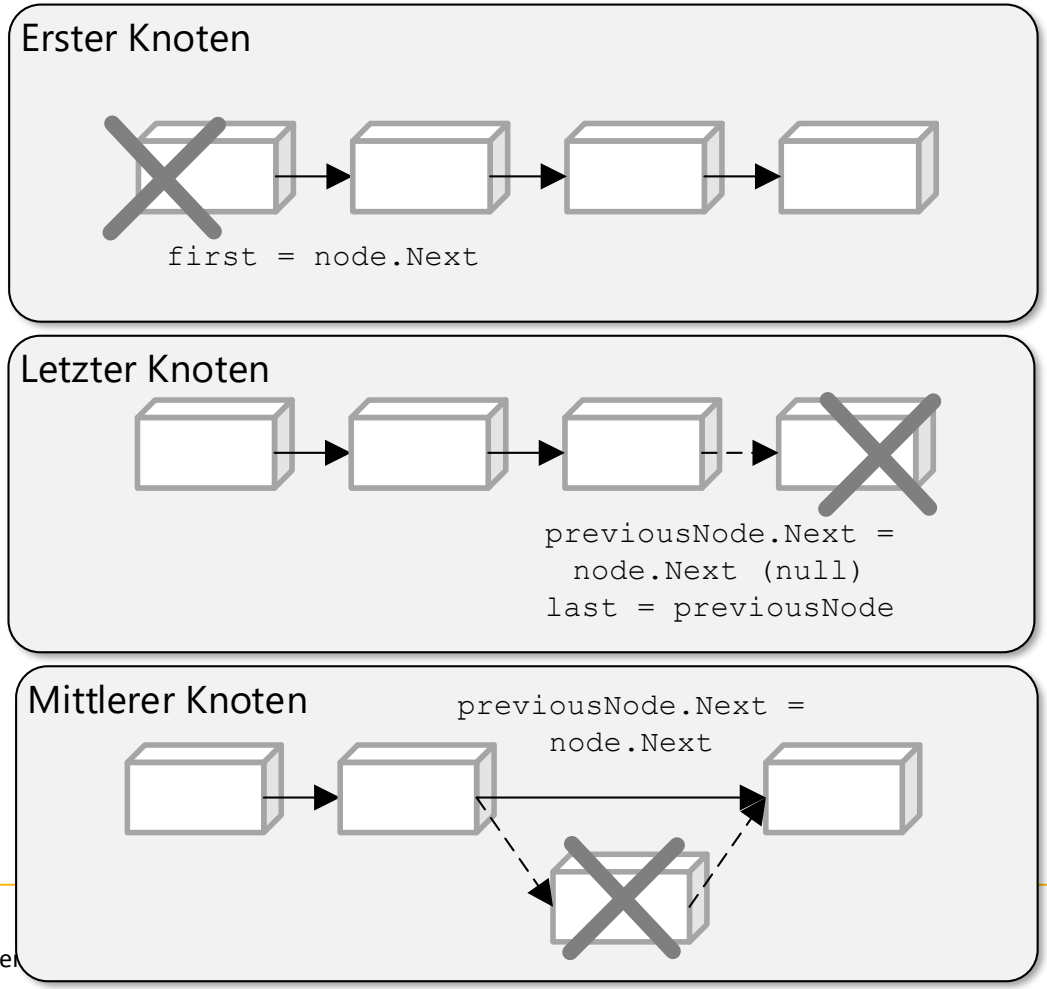
1. Neuen Knoten instanziieren
2. Überprüfen, ob es sich um den ersten Knoten handelt
3. Ermitteln des letzten Knotens in der Liste
4. Next-Eigenschaft des letzten Knotens auf neuen Knoten setzen

Contains-Methode

```
public bool Contains(object data)
{
    return Find(data) != null;
}
```

Remove-Methode

```
public bool Remove(object data)
```



Find-Methode

```
public bool FindByIndex(int index)
```

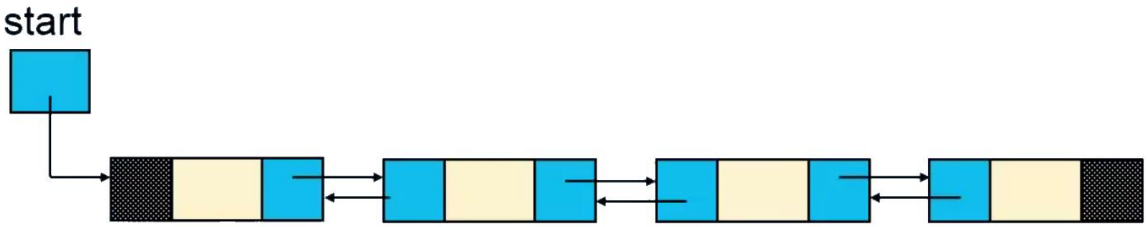
Exkurs «Indexer»

- Zugriff auf Elemente in einer Klasse über einen Index wie bei einem Array

- Deklaration

```
public object this[int index] {
    get {
        …
    }
    set {
        …
    }
}
```

Doubly Linked List (doppelt verkettete Liste)



- Vorteile
  - Können in beide Richtungen traversiert werden
  - Implementierung wird einfacher: Einfügen und Löschen
- Nachteile
  - Zusätzlicher Speicher
  - Es muss eine zusätzliche Referenz verwaltet werden

Implementierung

- Knoten

```
private sealed class Node {
    public object Data { get; set; }
    public Node Link { get; set; }
    public Node PrevLink { get; set; }
}
```

Liste über Array

- Vorteil
  - direkter Zugriff auf Element: Laufzeit O(1)
- Nachteil
  - statische Grösse
  - Lösung
    - Methode zum Vergrössern/Verkleinern des Arrays
    - kostet Zeit
    - daher nicht bei jedem Einfügen/Löschen

System.Collections.ArrayList (1)

```
public virtual int Add(Object value) {
    Contract.Ensures(Contract.Result<int>() >= 0);
    if (_size == _items.Length) EnsureCapacity(_size + 1);
    _items[_size] = value;
    _version++;
    return _size++;
}

private void EnsureCapacity(int min) {
    if (_items.Length < min) {
        int newCapacity = _items.Length == 0? _defaultCapacity: _items.Length * 2;
        // Allow the list to grow to maximum possible capacity (~2G elements) before encountering overflow.
        // Note that this check works even when _items.Length overflowed thanks to the (uint) cast
        if ((uint)newCapacity > Array.MaxArrayLength) newCapacity = Array.MaxArrayLength;
        if (newCapacity < min) newCapacity = min;
        Capacity = newCapacity;
    }
}
```

System.Collections.ArrayList (2)

```
public virtual int Capacity {
    get {
        Contract.Ensures(Contract.Result<int>() >= Count);
        return _items.Length;
    }
    set {
        if (value < _size) {
            throw new ArgumentOutOfRangeException("value", Environment.GetResourceString("ArgumentOutOfRange_SmallCapacity"));
        }
        Contract.Ensures(Capacity >= 0);
        Contract.EndContractBlock();
        // We don't want to update the version number when we change the capacity.
        // Some existing applications have dependency on this.
        if (value != _items.Length) {
            if (value > 0) {
                Object[] newItems = new Object[value];
                if (_size > 0) {
                    Array.Copy(_items, 0, newItems, 0, _size);
                }
                _items = newItems;
            }
            else {
                _items = new Object[_defaultCapacity];
            }
        }
    }
}
```

System.Collections.ArrayList (3)

```
// Removes the element at the given index. The size of the list is
// decreased by one.
//
public virtual void RemoveAt(int index) {
    if (index < 0 || index >= _size) throw new ArgumentOutOfRangeException("index", I
    Contract.Ensures(Count >= 0);
    //Contract.Ensures(Count == Contract.OldValue(Count) - 1);
    Contract.EndContractBlock();

    _size--;
    if (index < _size) {
        Array.Copy(_items, index + 1, _items, index, _size - index);
    }
    _items[_size] = null;
    _version++;
}
```

Vergleich ArrayList - LinkedList

Methode	Komplexitäts-klasse	Hinweise
ArrayList.Add	O(1)	bei Vergrößerung des Arrays: O(n)
LinkedList.Add	O(1)	

Selbststudium

- Lesen Sie Kapitel 2.2 in [Cordts2014
  - SinglyLinkedList: S.25 – 47
    - Bearbeiten Sie das Beispiel «Rechtschreibeprüfung»
    - Implementieren Sie die Klasse SinglyLinkedListEnumerator

      Studieren Sie dazu das Interface IEnumverator

      Studieren Sie das Schlüsselwort yield
    - Ändern Sie die Klasse Klasse SinglyLinkedList so, dass generische Typen verwendet werden können
  - DoublyLinkedList: S. 47 – 50
  - ArrayList: S. 50 - 59
- Lösen Sie die Übungsaufgaben zu Kapitel 2.2