

21. Datenstrukturen

21.1 Einleitung

21.2 Listen

21.3 Assoziative Speicher

21.4 Bäume

21.5 Mengen

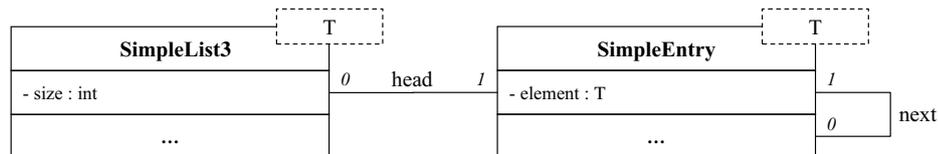
21.6 Das Collections-Framework in Java

- Eine *Menge* ist eine (ungeordnete) Zusammenfassung verschiedener Objekte. Insbesondere enthält eine Menge keine Duplikate.
- Eine *Multimenge* ist eine Menge, die Duplikate enthalten kann.
- Mengen und Multimengen sind wichtige Konzepte, u.a. um Assoziationen zwischen Objekten verschiedener Klassen umzusetzen.
- Im folgenden wiederholen wir grundlegende Funktionalitäten von Mengen. Eine Implementierung sollte entsprechende Methoden für diese Funktionalitäten anbieten. Diese gelten natürlich genauso für Multimengen.

- Leere Menge:
 $\emptyset : \rightarrow \text{Set}(T)$
- Elementbeziehung:
 $\in : T \times \text{Set}(T) \rightarrow \mathbb{B}$
- Teilmengenbeziehung:
 $\subseteq : \text{Set}(T) \times \text{Set}(T) \rightarrow \mathbb{B}$
- Vereinigung:
 $\cup : \text{Set}(T) \times \text{Set}(T) \rightarrow \text{Set}(T)$
- Durchschnitt:
 $\cap : \text{Set}(T) \times \text{Set}(T) \rightarrow \text{Set}(T)$
- Differenz:
 $\setminus : \text{Set}(T) \times \text{Set}(T) \rightarrow \text{Set}(T)$
- Kardinalität:
 $|\cdot| : \text{Set}(T) \rightarrow \mathbb{N}$

- Zur effizienten Realisierung von Mengen spielen folgende Gedanken eine Rolle:
 - Die grundlegenden Operationen sollten effizient sein.
 - Das Einfügen eines neuen Elementes sollte effizient sein. Insbesondere ist darauf zu achten, dass keine Duplikate eingefügt werden.
 - Das Löschen eines Elementes sollte effizient sein.
 - Die Suche nach einem Element sollte effizient sein.
- Für die Realisierung von Multimengen spielen natürlich die selben Gedanken eine Rolle. Allerdings muss nun auf Duplikate keine Rücksicht genommen werden.

- Als Beispiel schauen wir uns eine Implementierung einer Menge auf Basis einer typisierten (einfach) verketteten Liste an.
- Zur Erinnerung:



```

public class Menge<T>
{
    private SimpleList3<T> elemente;

    public int cardinalitaet()
    {
        return this.elemente.length();
    }

    public T[] getElemente()
    {
        return this.elemente.asArray();
    }

    public void einfuegen(T element)
    {
        if(!this.enthaelt(element))
        {
            this.elemente.add(element);
        }
    }
    ...
}
  
```

```
public boolean enthaelt(T t)
{
    return this.elemente.contains(t);
}

public void vereinigung(Menge<T> andereMenge)
{
    T[] neueElemente = andereMenge.getElemente();
    for(T t : neueElemente)
    {
        this.einfuegen(t);
    }
}
```

```
public boolean teilmengeVon(Menge<T> superMenge)
{
    boolean istTeilmenge = true;
    T[] elemente = this.getElemente();
    for(T t : elemente)
    {
        istTeilmenge = istTeilmenge && superMenge.enthaelt(t);
    }
    return istTeilmenge;
}
```

Vergleich der durchschnittlichen Laufzeiten

	einfaches Array	einfache Liste	doppelt verankerte Liste	doppelt verkettete Liste	perfekte Hashverfahren (assoz. Array)	binärer Suchbaum
dynamisch	nein	ja	ja	ja	nein	ja
Einfügen (Schlüssel)	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$ bzw. $O(n)$	$O(\log n)$
Löschen (Schlüssel)	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(\log n)$
Suchen (Schlüssel)	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(\log n)$

- Eine Standard-Implementierung einer Menge in Java ist z.B. die Klasse `java.util.HashSet`, die auf einem assoziativen Speicher beruht. Dabei wird intern die Methode `hashCode()` für die Objekte der Menge verwendet. Achtung: Die Laufzeit dieser Klasse degeneriert, falls die Hashfunktion (die auf der Methode `hashCode()` basiert), nicht optimal ist!

21. Datenstrukturen

21.1 Einleitung

21.2 Listen

21.3 Assoziative Speicher

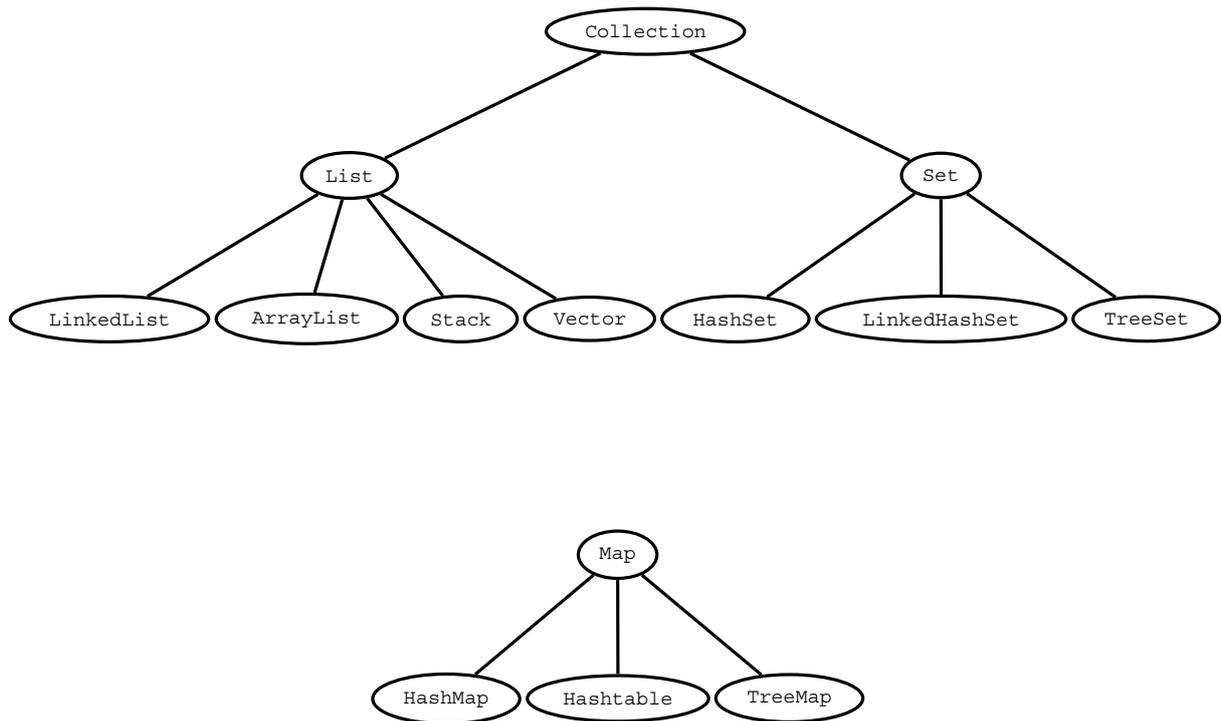
21.4 Bäume

21.5 Mengen

21.6 Das Collections-Framework in Java

- In Java gibt es eine große Menge an vordefinierten Klassen für mengenartige Datenstrukturen.
- Im folgenden geben wir einen kurzen Überblick über das Collections-Framework von Java.
- Zur Vertiefung empfehlen wir das intensive Studium der Dokumentationen der entsprechenden Klassen.

- Alle Klassen mengenartiger, assoziativer Datenstrukturen implementieren das Interface `java.util.Map`. Das Interface stellt Basis-Funktionalitäten wie Einfügen (`put`), Löschen (`remove`) und verschiedene Formen der Suche (`get`, `containsKey`, `containsValue`) zur Verfügung.
- Alle Klassen mengenartiger, nicht-assoziativer Datenstrukturen implementieren das Interface `java.util.Collection`. Das Interface stellt Basis-Funktionalitäten wie Einfügen (`add`), Löschen (`remove`) und Suchen (`contains`) zur Verfügung.
- Vom Interface `java.util.Collection` abgeleitete Interfaces sind u.a.:
 - `java.util.Set`: Klassen, die Mengen ohne Duplikate modellieren, implementieren dieses Interface.
 - `java.util.List`: Klassen, die Multimengen modellieren, implementieren dieses Interface.



```
public static void accessFor(Integer[] intArray)
{
    Integer currentInteger;
    for(int i = 0; i < intArray.length; i++)
    {
        currentInteger = intArray[i];
    }
}
```

Laufzeit?

Um iterative Aktionen auf allen Elementen einer Collection (oder einer bestimmten Teilmenge davon) durchzuführen, sind nicht alle Schleifenkonstrukte gleichwertig.

Beispiel:

```
public static void accessFor(List<Integer> list)
{
    Integer currentInteger;
    for(int i = 0; i < list.size(); i++)
    {
        currentInteger = list.get(i);
    }
}
```

Laufzeit?

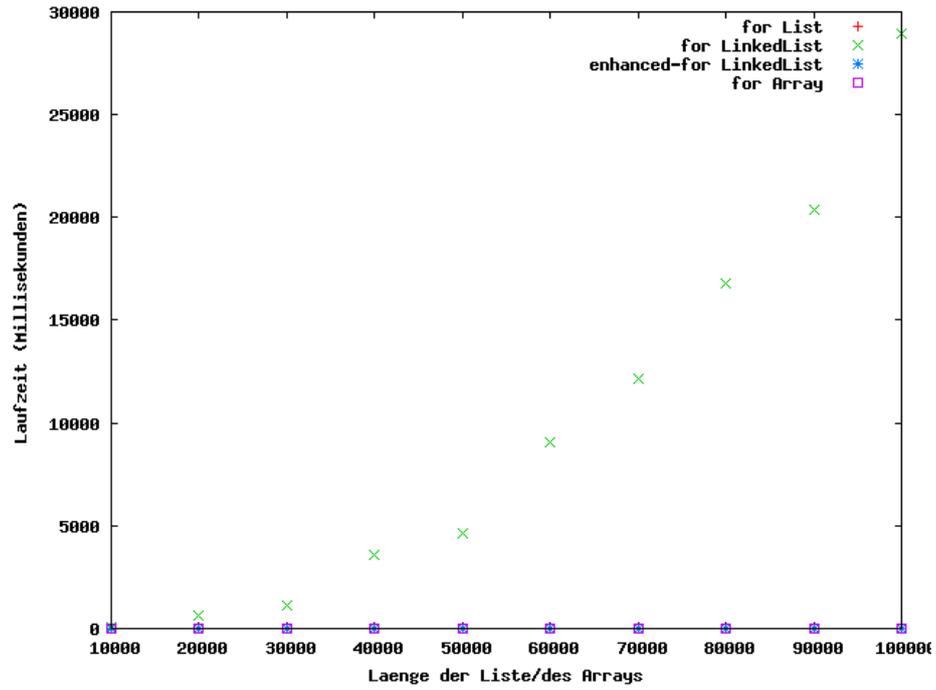
Die `Iterator-` und `Iterable-`Interfaces und *enhanced for*

Eine Collection implementiert das Interface `Iterable`, das nur vorschreibt, dass ein `Iterator` zurückgegeben werden kann, der es ermöglicht, die Collection zu durchwandern. Dieser `Iterator` wird auch implizit im sogenannten *enhanced for* verwendet:

```
public static void accessIterable(
    Iterable<Integer> intIterable)
{
    Integer currentInteger;
    for(Integer currentInt : intIterable)
    {
        currentInteger = currentInt;
    }
}
```

Laufzeit für eine verkettete Liste?

Test für die verschiedenen Iterationen auf verschiedenen Listen und einem Array



Test-Klasse:

http://www.dbs.ifi.lmu.de/Lehre/EIP/WS_2008/skript/programmbeispiele/datenstrukturen/IterationTest.java