The Java Collections Framework

- Einführung
- Schnittstellen
- Implementierungen
- Algorithmen
- Beispiele

Einführung

Eine **Collection** ist ein Objekt, das mehrere Objekte zu einer Einheit zusammenfasst.

Collections werden auch **Container** oder **Behälter** genannt.

Ein **Collection Framework** stellt eine einheitliche Architektur zur Repräsentation und Manipulation von Collections zur Verfügung.

Ein Collection Framework enthält:

- Schnittstellen
- Implementierungen
- Algorithmen

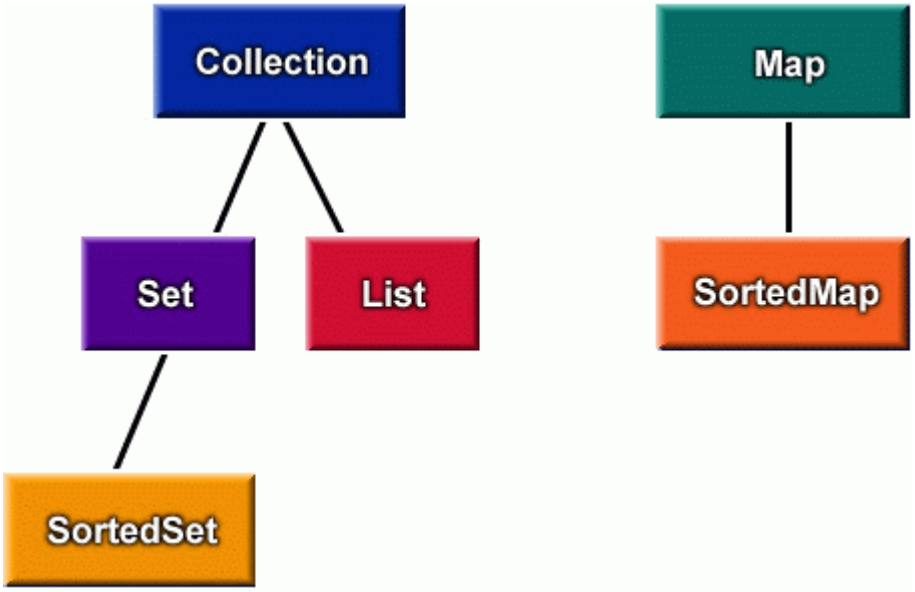
Ein Collection Framework bietet die folgenden Vorteile:

- Es reduziert den Programmieraufwand.
- Es unterstützt die Wiederverwendung von Software.
- Es erhöht die Software-Qualität.
- Das „Tuning“ von Software wird erleichtert.

Einführung

Schnittstellen

Das Java Collections Framework stellt die folgenden grundlegenden Schnittstellen zur Verfügung:



Schnittstellen

Implementierungen

		Implementierungen			
		Hash-Tabelle	Veränder-liches Feld	Balancierter Baum	Verkettete Liste
Schnitt-stellen	`Set`	`HashSet`		`TreeSet`	
	`List`		`ArrayList`		`LinkedList`
	`Map`	`HashMap`		`TreeMap`	

Algorithmen

Die *Klasse* `Collection`**`s`** stellt eine Reihe polymorpher Algorithmen zur Verfügung.

Die Algorithmen sind implementiert als *statische* Methoden, deren erstes Argument die `Collection` bezeichnet, auf welche die Operation angewandt werden soll.

Die meisten dieser Algorithmen arbeiten auf `List`-Objekten. Die Operationen `min` und `max` akzeptieren beliebige `Collection`-Objekte.

Beispiele

```
package collections;

import java.util.*;

public class Intro1
{
  public static void main(String args[])
  {
    List list = new ArrayList();

    for (int i = 1; i <= 10; i++)
      list.add(i + " * " + i + " = " + i * i);

    Iterator iter = list.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());
  }
}
```

Beispiele

Definition der Klasse `ArrayList`:

```
interface Collection {...}

interface List extends Collection {...}

abstract class AbstractCollection
implements Collection {...}

abstract class AbstractList
extends AbstractCollection
implements List {...}

class ArrayList extends AbstractList {...}
```

Beispiele

```
package collections;
import java.util.*;

public class Sort1 implements Comparator
{
  public int compare(Object o1, Object o2)
  {
    String s1 = (String)o1;
    String s2 = (String)o2;

    return s1.toLowerCase().compareTo(s2.toLowerCase());
  }

  public static void main(String args[])
  {
    List list = new ArrayList();

    list.add("abc");
    list.add("DEF");
    list.add("ghi");

    // standard sort
    Collections.sort(list);
    Iterator iter = list.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());

    // sort, ignoring case
    Collections.sort(list, new Sort1());
    iter = list.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());
  }
}
```

Beispiele

Die obige Hierarchie von Schnittstellen und Klassen gehorcht der folgenden Namenskonvention:

`ArrayList`: `List` implemented via an array

`LinkedList`: `List` implemented as a linked structure

`HashSet`: `Set` implemented via a hash table

`TreeSet`: `SortedSet` implemented as a tree

`HashMap`: `Map` implemented via a hash table

`TreeMap`: `SortedMap` implemented as a tree

Beispiele

Verschiedene Arten von Schnittstellen:

`Collection`:	a group of elements
`Set`:	a group of elements with no duplicates
`SortedSet`:	like `Set`, except the elements are kept in sorted order
`List`:	ordered collection or sequence
`Map`:	an object that maps keys to values, with no duplicate keys
`SortedMap`:	like `Map`, except the keys are kept in sorted order

Beispiele

Die bereitgestellten Methoden sind abhängig vom jeweiligen Interface. Allen Collection-Schnittstellen gemeinsam sind die folgenden Methoden:

`add`: add a new element

`remove`: remove an element

`size`: number of elements currently represented

`isEmpty`: check whether the collection is currently empty

`iterator`: obtain an `Iterator` object

`contains`: check whether a collection contains an element

`toArray`: return an `Object[]` with all the collection's elements

Auf Listen sind zusätzlich die folgenden Operationen definiert:

`get`: get an element based on an index

`indexOf`: find the index of a specified element

Beispiele

```
package collections;

import java.util.*;

public class Map1
{
  public static void main(String args[])
  {
    Map hm = new HashMap();
    hm.put("Mary", "123-4567");
    hm.put("Larry", "234-5678");
    hm.put("Mary", "456-7890");
    hm.put("Felicia", "345-6789");

    Iterator iter = hm.entrySet().iterator();
    while (iter.hasNext())
    {
      Map.Entry e = (Map.Entry)iter.next();
      System.out.println(e.getKey() + " "  +
                         e.getValue());
    }
  }
}
```

Beispiele

```
package collections;

import java.util.*;

public class Set1
{
  public static void main(String args[])
  {
    Set hs = new HashSet();
    hs.add("1");
    hs.add("2");
    hs.add("2");
    hs.add("3");

    Iterator iter = hs.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());
  }
}
```

Beispiele

```
package collections;

import java.util.*;

public class Search1
{
  public static void main(String args[])
  {
    List list = new ArrayList();
    Random rn = new Random();

    for (int i = 1; i <= 25; i++)
    {
      int n = (int)(rn.nextFloat() * 10 + 1);
      Integer ival = new Integer(n);
      int pos = Collections.binarySearch(list, ival);
      if (pos < 0)
        list.add(-pos-1, ival);
    }

    Iterator iter = list.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());
  }
}
```

Beispiele

```
package collections;

import java.util.*;

public class Shuffle1
{
  public static void main(String args[])
  {
    List list = new ArrayList();

    list.add("abc");
    list.add("def");
    list.add("ghi");
    list.add("jkl");

    Collections.shuffle(list);

    Iterator iter = list.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());
  }
}
```

Beispiele

```
package collections;

import java.util.*;

public class Reverse1
{
  public static void main(String args[])
  {
    List list = new ArrayList();
    for (int i = 1; i <= 10; i++)
      list.add(i + " * " + i + " = " + i * i);

    Collections.reverse(list);

    Iterator iter = list.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());
  }
}
```

Beispiele

```
package collections;

import java.util.*;

public class Max1
{
  public static void main(String args[])
  {
    Set hs = new HashSet();
    hs.add("1");
    hs.add("2");
    hs.add("3");
    hs.add("2");

    Iterator iter = hs.iterator();
    while (iter.hasNext())
      System.out.println(iter.next());

    System.out.println(
      "max = " + Collections.max(hs));
  }
}
```

Beispiele

```
package collections;

import java.util.*;

class Test1
{
  private int x;
  public Test1(int i) {x = i;}
}

public class Compare1
{
  public static void main(String args[])
  {
    List list = new ArrayList();
    list.add(new Test1(5));
    list.add(new Test1(10));
    Collections.sort(list);
  }
}
```

Beispiele

```
package collections;

import java.util.*;

class Test2 implements Comparable
{
  private int x;
  public Test2(int i) {x = i;}
  public int compareTo(Object o)
  {
    int val = ((Test2)o).x;
    if (x < val)
      return -1;
    else if (x > val)
      return 1;
    else
      return 0;
  }
}

public class Compare2
{
  public static void main(String args[])
  {
    List list = new ArrayList();
    list.add(new Test2(5));
    list.add(new Test2(10));
    Collections.sort(list);
  }
}
```

Beispiele

```
public class Iterator1
{
  public static void main(String args[])
  {
    List list = new ArrayList();
    list.add("abc");
    list.add("def");
    list.add("ghi");

    ListIterator iter1 = list.listIterator();
    boolean first = true;
    while (iter1.hasNext())
    {
      System.out.println(iter1.next());
      //list.add("xyz");
      if (first)
      {
        iter1.add("xyz");
        first = false;
      }
    }
  }
}
```

Beispiele

```
package collections;

import java.util.*;

public class Iterator2
{
  public static void main(String args[])
  {
    List list = new ArrayList();
    list.add("abc");
    list.add("def");
    list.add("ghi");

    ListIterator iter =
      list.listIterator(list.size());

    while (iter.hasPrevious())
      System.out.println(iter.previous());
  }
}
```