Das 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:

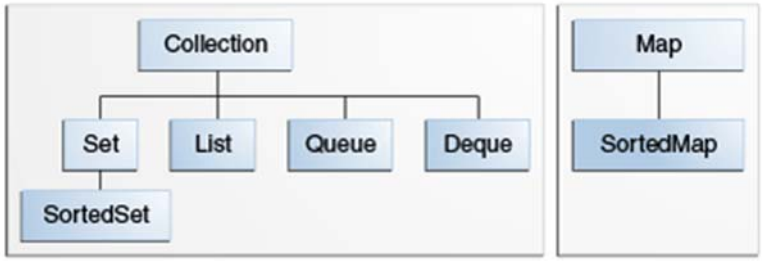
- Schnittstellen
- Implementierungen
- Algorithmen

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.

Schnittstellen

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



Implementierungen

		Implementierungen			
		Hash-Tabelle	Veränder-liches Feld	Balancierter Baum	Verkettete Liste
Schnitt-stellen	`Set`	`HashSet`		`TreeSet`	
	`List`		`ArrayList`		`LinkedList`
	`Queue`				`LinkedList`
	`Deque`		`ArrayDeque`		`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<String> list = new ArrayList<String>();

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

    for (String s : list)
        System.out.println(s);

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

Beispiele

Definition der Klasse `ArrayList`:

```
interface Collection<E> {...}

interface List<E> extends Collection<E> {...}

abstract class AbstractCollection<E>
implements Collection<E> {...}

abstract class AbstractList<E>
extends AbstractCollection<E>
implements List<E> {...}

class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable,
Serializable {...}
```


Beispiele

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

public class Sort1 implements Comparator<String>
{
  public int compare(String s1, String s2)
  {
    return s1.toLowerCase().compareTo(s2.toLowerCase());
  }

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

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

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

    // sort, ignoring case
    Collections.sort(list, new Sort1());
    for (String s : list)
      System.out.println(s);
  }
}
```

Beispiele

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

`ArrayList`: `List` implementiert mittels Array

`LinkedList`: `List` implementiert als verkettete Struktur

`HashSet`: `Set` implementiert mittels Hashtabelle

`TreeSet`: `SortedSet` implementiert als Baum

`HashMap`: `Map` implementiert mittels Hashtabelle

`TreeMap`: `SortedMap` implementiert als Baum

Beispiele

Verschiedene Arten von Schnittstellen:

`Collection`: eine Menge von Elementen

`Set`: eine Menge von Elementen ohne Duplikate

`SortedSet`: wie `Set`, aber die Elemente werden in sortierter Reihenfolge verwaltet

`List`: eine geordnete Collection oder Sequenz

`Map`: eine Abbildung von Schlüsseln auf Werte, die Schlüsselmenge enthält keine Duplikate

`SortedMap`: wie `Map`, aber die Elemente werden in sortierter Reihenfolge verwaltet

Beispiele

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

`add`: Hinzufügen des als Parameter übergebenen Objekts am Ende der Collection

`remove`: Entfernen des als Parameter übergebenen Objekts aus der Collection

`size`: Anzahl der aktuell enthaltenen Elemente

`isEmpty`: prüft, ob die Collection leer ist

`iterator`: liefert ein `Iterator`-Objekt zurück

`contains`: prüft, ob das als Parameter übergebene Objekt in der Collection enthalten ist

`toArray`: liefert ein Feld vom Typ `Object[]`, das alle Elemente der Collection enthält

Auf Listen sind zusätzlich die folgenden Operationen definiert:

`get`: Der Parameter bezeichnet einen Index. Das index-te Element wird zurückgegeben.

`indexOf`: liefert den Index des als Parameter übergebenen Objekts

Beispiele

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

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

    Iterator<Map.Entry<String, String>> iter =
      hm.entrySet().iterator();
    while (iter.hasNext())
    {
      Map.Entry<String, String> e = 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<String> hs = new HashSet<String>();
    hs.add("1");
    hs.add("2");
    hs.add("2");
    hs.add("3");

    Iterator<String> 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<Integer> list = new ArrayList<Integer>();
    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);
    }

    for (Integer i : list)
      System.out.println(i);
  }
}
```

Beispiele

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

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

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

    Collections.shuffle(list);

    Iterator<String> 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<String> list = new ArrayList<String>();
    for (int i = 1; i <= 10; i++)
      list.add(i + " * " + i + " = " + i * i);

    Collections.reverse(list);

    for (String s : list)
      System.out.println(s);
  }
}
```

Beispiele

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

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

    Iterator<String> 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 int getX() { return x;}
}

public class Compare1
{
  public static void main(String args[])
  {
    List<Test1> list = new ArrayList<Test1>();
    list.add(new Test1(5));
    list.add(new Test1(10));
    Collections.sort(list);
    for(Test1 t : list)
      System.out.println(t.getX());
  }
}
```

Beispiele

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

class Test2 implements Comparable<Test2> {
  private int x;
  public Test2(int i) {x = i;}
  public int getX() { return x;}
  public int compareTo(Test2 t) {
    int val = t.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<Test2> list = new ArrayList<Test2>();
    list.add(new Test2(5));
    list.add(new Test2(10));
    Collections.sort(list);
    for (Test2 t : list)
      System.out.println(t.getX());
  }
}
```


Beispiele

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

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

    ListIterator<String> 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<String> list = new ArrayList<String>();
    list.add("abc");
    list.add("def");
    list.add("ghi");

    ListIterator<String> iter =
      list.listIterator(list.size());

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