Teil V

Generics und Kollektionen in Java

Überblick

Motivation für Generics

- Implementierung einer Datenstruktur erfordert Angabe des Typs der Daten:

```
class Node {                    class Node {
   private int obj;                private String obj;
   Node next;                      Node next;
   ...                             ...
```

  - ähnlicher Programmtext muss mehrfach geschrieben werden
  - hoher Aufwand, schlechte Wartbarkeit

- Teilweise Lösung des Problems: Als Typ `Object` wählen.
  - Nachteil: keine Typprüfung durch Compiler möglich

Generics in Java

- seit Version J2SE 5.0: direkte Unterstützung von parametrisierten Datentypen $\rightsquigarrow$ Generics
- Notation: `Klasse<`*`Elementtyp`*`>`
- Generics werden insbes. für Kollektionen und Iteratoren benutzt:

```
List<RatNumber> list =
    new ArrayList<RatNumber>();
list.add(new RatNumber(1, 3));
list.add(new RatNumber(1, 4));

Iterator<RatNumber> iter = list.iterator();
while (iter.hasNext())
    RatNumber rn = iter.next();
```

Implementierung von Generics

- Angabe eines generischen Typen (”Platzhalter”) bei Implementierung

```
private class Node<T> {

    T obj;
    Node<T> next;

    ...
    public void setNext(Node<T> n) {
        next = n;
    }
}
```

Implementierung von Generics /2

```
private class LinkedList<T> {

   private Node<T> head = null;

   public LinkedList() {
      head = new Node<T>();
   }
   ...
```

Implementierung von Generics /3

- Typsicherheit bei Verwendung

```
...
LinkedList<Student> l =
      new LinkedList<Student>();
l.add(new Student("Meier"));
l.add(new Student("Schulze"));
l.add(new Student("Schmidt"));
...
Student s = l.getFirst();
...
```

Implementierung generischer Typen

- In C++ wird generisches Programmieren über Templates realisiert.
  - der Compiler erzeugt für jeden verwendeten Typ eigenen ausführbaren Code, also Code für `Stack<String>`, Code für `Stack<int>`, etc.

- In Java nutzt der Compiler die Generics zur Typprüfung und ersetzt dann alle Typen durch `Object` (Rohtyp).
  - Die Nutzung des Java Collection Frameworks ohne Typisierung ist möglich (aber unschön und führt zu Compiler-Warnung).

Java Collection Framework

- Teil des Pakets `java.util`
- Bereitstellung wichtiger Kollektionen (Listen, Mengen, Verzeichnisse) mit unterschiedlicher Implementierung
- Navigation mittels Iteratoren
- Implementierung häufig benutzter Algorithmen (Suchen, Sortieren, kleines/größtes Element, ...)
- Basis `java.util.Collection`

Schnittstelle für alle Kollektionsklassen

- Schnittstelle `java.util.Collection<E>`
- Größe der Kollektion

```
int size()
boolean isEmpty()
```

- Suchen von Elementen

```
boolean contains(E o)
boolean containsAll(Collection<E> c)
```

- Navigation

```
Iterator iterator()
```

Schnittstelle /2

- Hinzufügen/Entfernen

```
boolean add(E o)
boolean remove(E o)
boolean addAll(Collection<E> c)
boolean removeAll(Collection<E> c)
```

Ausgewählte Kollektionen

- `java.util.List<E>`
  Repräsentation von Listen (geordnet nach Einfügereihenfolge, Duplikate zulässig)
- `java.util.Set<E>`
  Repräsentation von Mengen (ungeordnet/geordnet, ohne Duplikate)
- `java.util.Map<K,V>`
  Verzeichnis (Dictionary) von Objekten (Schlüssel-Wert-Paare) für schnellen Zugriff über Schlüssel

java.util.List: Schnittstelle für Listen

- Zugriff über Position

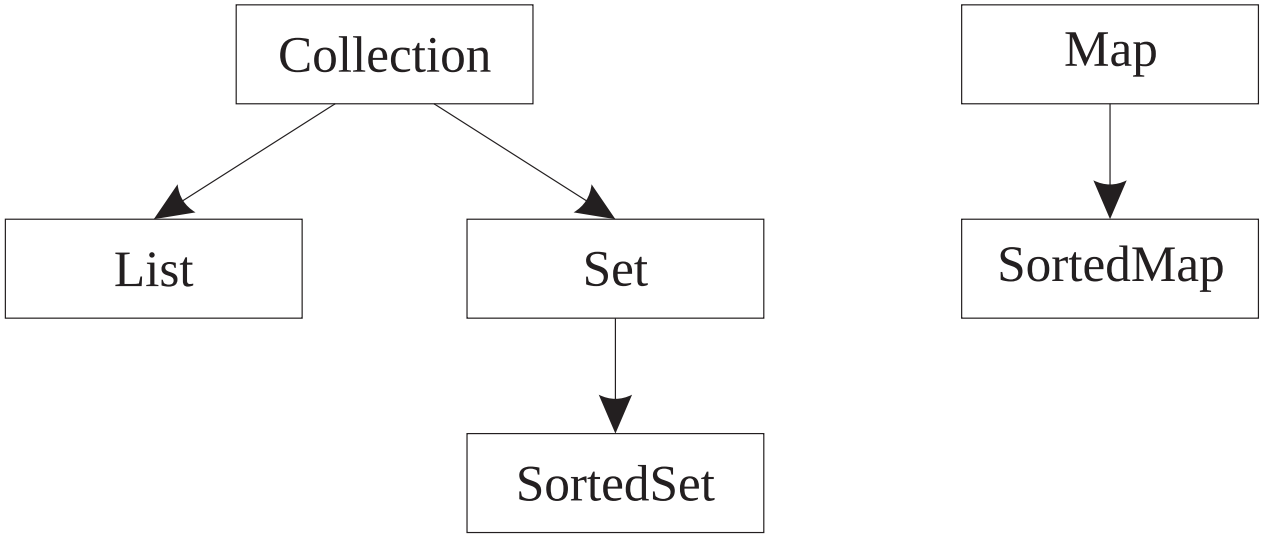
```
E get(int idx)
E set(int idx, E o)
void add(int idx, E o)
E remove(int idx)
```

- Bestimmen der Position eines Objektes

```
int indexOf(E o)
```

- konkrete Implementierungen
  - verkettete Liste: `LinkedList`
  - Feld: `ArrayList`

Struktur des Java Collection Framework



Java Collection Framework: Implementierungen

Implementierung	Schnittstelle		
	`Set`	`List`	`Map`
Hashtabelle	`HashSet`		`HashMap`
Feld		`ArrayList`	
Baum	`TreeSet`		`TreeMap`
Liste		`LinkedList`	

Algorithmen für Kollektionen

- Klasse `java.util.Collections`
- Sortieren von Listen[1]

```
static void sort(List list)
```

- Suchen in Listen[2]

```
public static int binarySearch(List list, T key)
```

[1] Genauer: `public static <T extends Comparable<? super T>> void sort(List<T> list)`

[2] Genauer: `public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)`

Algorithmen für Kollektionen /2

- Umkehren

```
static void reverse(List<?> list)
```

- kleinstes/größtes Element[3]

```
static <T> T min(Collection<T> col)
static <T> T max(Collection<T> col)
```

[3] Genauer:

```
public static <T extends Object & Comparable<? super T>>
                    T min(Collection<? extends T> coll)
```

Anwendungsbeispiel

```
// Liste erzeugen und Objekte einfügen
LinkedList<RatNumber> list
                  = new LinkedList<RatNumber>();
list.add(new RatNumber(1, 3));
list.add(new RatNumber(1, 4));
list.add(new RatNumber(1, 5));
// Ausgeben
Iterator<RatNumber> i = list.iterator();
while (i.hasNext())
   System.out.println(i.next());
// Sortieren
Collections.sort(list);
// Ausgeben
for (RatNumber rn : list) {
   System.out.println(rn);
   }
```

Zusammenfassung

- Parametrisierbare Datentypen in Java
- vordefinierte Klassen in der Java-Bibliothek
- Literatur:
  - Saake/Sattler: *Algorithmen und Datenstrukturen*, Kapitel 13
  - Mössenböck: *Verstehen Sie Java?*