Programmieren in Java

Vorlesung 05: Generics

Prof. Dr. Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

SS 2015

Inhalt

Generische Klassen und Interfaces

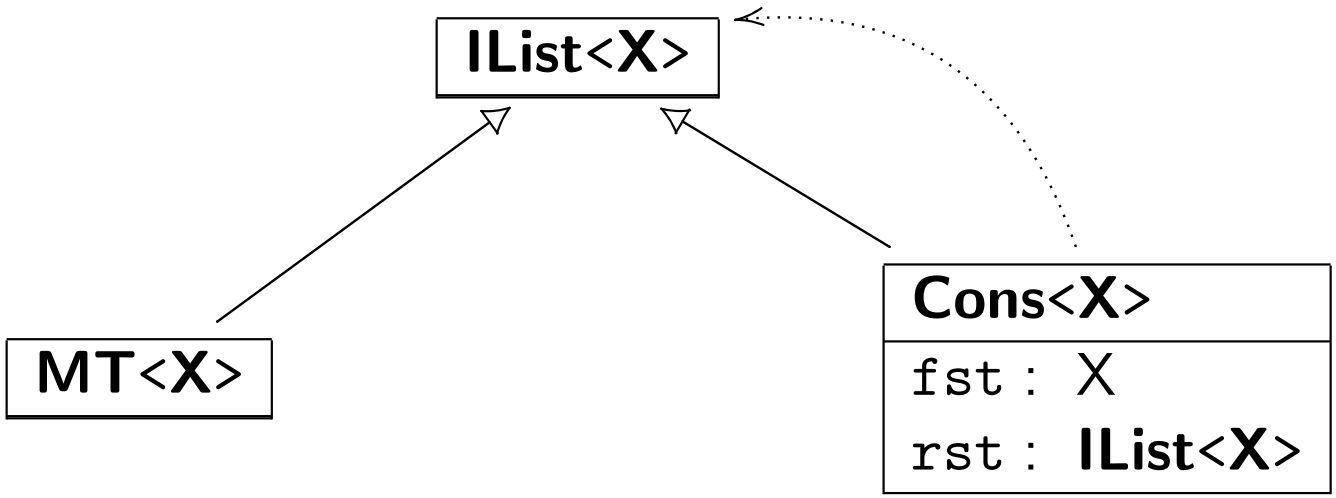
Generics

- *Generische Klassen, Interfaces und Methoden* erlauben die Abstraktion von den konkreten Typen der Objekte, die in Instanzvariablen und lokalen Variablen gespeichert werden oder als Parameter übergeben werden.
- Hauptverwendungsbereiche:
  - Containerklassen (Collections)
  - Abstraktion eines Deklarationsmusters

Generisches Paar

```
1  public class GenericPair<X,Y> {
2      private X fst;
3      private Y snd;
4      GenericPair(X fst, Y snd) {
5          this.fst = fst;
6          this.snd = snd;
7      }
8      public X getFst() {
9          return this.fst;
10     }
11     public Y getSnd() {
12         return this.snd;
13     }
14 }
```

Generische Listen



- **IList<X>** ist ein *generisches Interface*
- **MT<X>** und **Cons<X>** sind *generische Klassen*
- **X** ist dabei eine *Typvariable*
- **X** steht für einen beliebigen Referenztyp (Klassen- oder Interfacetyp), **nicht** für einen primitiven Typ

Implementierung: Generische Listen

```
1  // Listen mit beliebigen Elementen
2  interface IList<X> {
16 }
```

```
1  // Variante leere Liste
2  class MT<X> implements IList<X> {
3      public MT() {}
16 }
```

```
1  // Variante nicht-leere Liste
2  class Cons<X> implements IList<X> {
3      private X fst;
4      private IList<X> rst;
5
6      public Cons (X fst, IList<X> rst) {
7          this.fst = fst;
8          this.rst = rst;
9      }
27 }
```

Verwendung von generischen Listen

Liste von int bzw. **Integer**

- Achtung: **Typvariablen können nur für Referenztypen stehen!**
- Anstelle von primitiven Typen müssen die Wrapperklassen verwendet werden (Konversion von Werten automatisch dank *Autoboxing*)

```
1  // Aufbau der Liste
2  IList<Integer> i1 = new MT<Integer> ();
3  IList<Integer> i2 = new Cons<Integer> (32168, i1);
4  IList<Integer> i3 = new Cons<Integer> (new Integer ("32768"), i2);
5  IList<Integer> i4 = new Cons<Integer> (new Integer (-14), i3);
```

Exkurs: Wrapperklassen

- Für jeden primitiven Datentyp stellt Java eine Klasse bereit, deren Instanzen einen Wert des Typs in ein Objekt verpacken.
- Beispiele

primitiver Typ	Wrapperklasse
int	**java.lang.Integer**
double	**java.lang.Double**
boolean	**java.lang.Boolean**

- Klassen- und Interfacetypen heißen (im Unterschied zu primitiven Typen) auch *Referenztypen*.

Methoden von Wrapperklassen

- Wrapperklassen beinhalten (statische) Hilfsmethoden und Felder zum Umgang mit Werten des zugehörigen primitiven Datentyps.
- Vorsicht: Java konvertiert automatisch zwischen primitiven Werten und Objekten der Wrapperklassen. (*autoboxing*)

Beispiel: **Integer** (Auszug)

```
1  static int MAX_VALUE; // maximaler Wert von int
2  static int MIN_VALUE; // minimaler Wert von int
3
4  Integer (int value);
5  Integer (String s); // konvertiert String -> int
6
7  int compareTo(Integer anotherInteger);
8  int intValue();
9  static int parseInt(String s);
```

Iterator

Erinnerung: Das Iterator Interface (abgekürzt)

```
1  public interface Iterator<E> {
2      /**
3       * @return {@code true} if the iteration has more elements
4       */
5      boolean hasNext();
6
7      /**
8       * @return the next element in the iteration
9       */
10     E next();
11
12     /**
13      * Removes from the underlying collection the last element returned
14      * by this iterator (optional operation). This method can be called
15      * only once per call to {@link #next}.
16      */
17     void remove();
18 }
```

Iterable

- Ein Iterator kann aus jedem Referenztyp gewonnen werden, der das Interface **java.lang.Iterable<X>** implementiert.
- Jede Collection implementiert **Iterable**.
- Ein Array **[T]** implementiert **Iterable<T>**
- **Iterable** wird auch für `foreach`-Schleifen benötigt.

```
1 public interface Iterable<T> {
2
3     /**
4      * @return an Iterator over a set of elements of type T.
5      */
6     Iterator<T> iterator();
7 }
```

Aufgabe: Implementiere Iterator mit IList

- Definiere IList mit Methoden empty(), getElement() und getRest().
- Um einen Iterator zu stellen, muss IList auch die iterator() Methode implementieren.
- Daher ist IList<X> eine **Erweiterung** von Iterable<X>.

```
1 public interface IList<X> extends Iterable<X> {
2   boolean empty();
3   X getElement();
4   IList<X> getRest();
5 }
```

Implementierung des Iterators

Immutable Wrapper

```
1  public class IListIterator<E> implements Iterator<E> {
2    private IList<E> list; // current position in the list
3    public IListIterator(IList<E> list) { this.list = list; }
4
5    public boolean hasNext() {
6      return !list.empty();
7    }
8
9    public E next() {
10     if (list.empty()) {
11       throw new IllegalStateException("Iterator called next on empty collection");
12     }
13     E element = list.getElement();
14     list = list.getRest();
15     return element;
16   }
17 }
```

Implementierung der Listenklassen

```
1  public class Nil<X>
2    implements IList<X> {
3
4    public Iterator<X> iterator() {
5      return new IListIterator<X>(this);
6    }
7
8
9
10
11   public Nil() {
12   }
13
14
15 }
```

```
1  public class Cons<X>
2    implements IList<X> {
3
4    public Iterator<X> iterator() {
5      return new IListIterator<X>(this);
6    }
7
8    private final X element;
9    private final IList<X> rest;
10
11   public Cons(X elem, IList<X> rest) {
12     this.element = elem;
13     this.rest = rest;
14   }
15 }
```

- Gleiche Implementierung von iterator()!

Implementierung der Listenklassen

```
1  public class Nil<X>
2    implements IList<X> {
3      // see above
4
5    public boolean empty() {
6      return true;
7    }
8
9    public X getElement() {
10     return null;
11   }
12
13   public IList<X> getRest() {
14     return null;
15   }
16 }
```

```
1  public class Cons<X>
2    implements IList<X> {
3      // see above
4
5    public boolean empty() {
6      return false;
7    }
8
9    public X getElement() {
10     return this.element;
11   }
12
13   public IList<X> getRest() {
14     return this.rest;
15   }
16 }
```

iterator als default Methode

- Die Implementierung von iterator war dieselbe in den Klassen Nil<X> und Cons<X>.
- Die Implementierung einer **public** Methode kann auch ins Interface geschoben werden.
- Das Schluesselwort **default** zeigt an, dass es sich um eine vordefinierte Implementierung handelt, die gilt, wenn sonst keine Implementierung definiert ist.

```
1 public interface IList<X> extends Iterable<X> {
2   boolean empty();
3   X getElement();
4   IList<X> getRest();
5   default Iterator<X> iterator() {
6     return new IListIterator<X>(this);
7   }
8 }
```

Fragen?