Programmierzertifikat Objekt-Orientierte Programmierung mit Java

Vorlesung 06: Generics

Peter Thiemann

Universität Freiburg, Germany

SS 2008

Inhalt

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*.
- Achtung: auf Referenztypen sollte die Gleichheit mit `equals` getestet werden (vgl. String)

Methoden von Wrappenklassen

- Wrapperklassen beinhalten (statische) Hilfsmethoden und Felder zum Umgang mit Werten des zugehörigen primitiven Datentyps.
- Vorsicht: Ab Version 5 konvertiert Java automatisch zwischen primitiven Werten und Objekten der Wrapperklassen. (*autoboxing*)

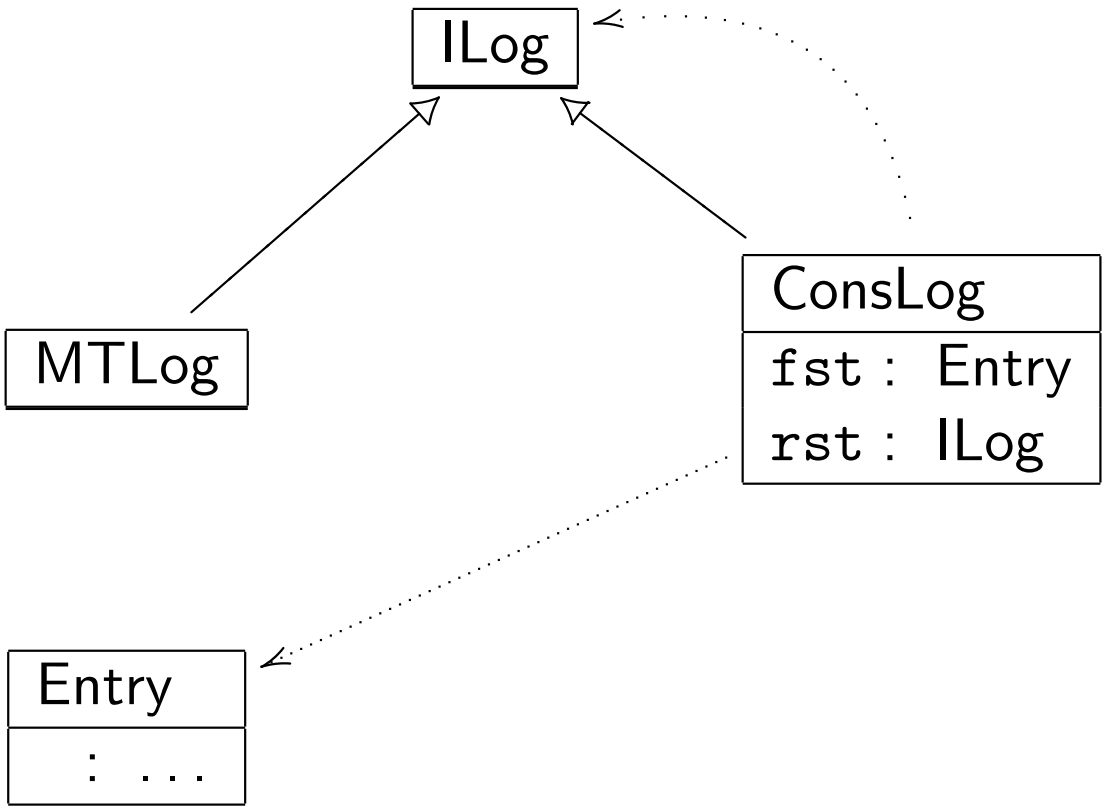
Beispiel: Integer (Auszug)

```
static int MAX_VALUE; // maximaler Wert von int
static int MIN_VALUE; // minimaler Wert von int

Integer (int value);
Integer (String s); // konvertiert String -> int

int compareTo(Integer anotherInteger);
int intValue();
static int parseInt(String s);
```

Generische Klassen und Interfaces

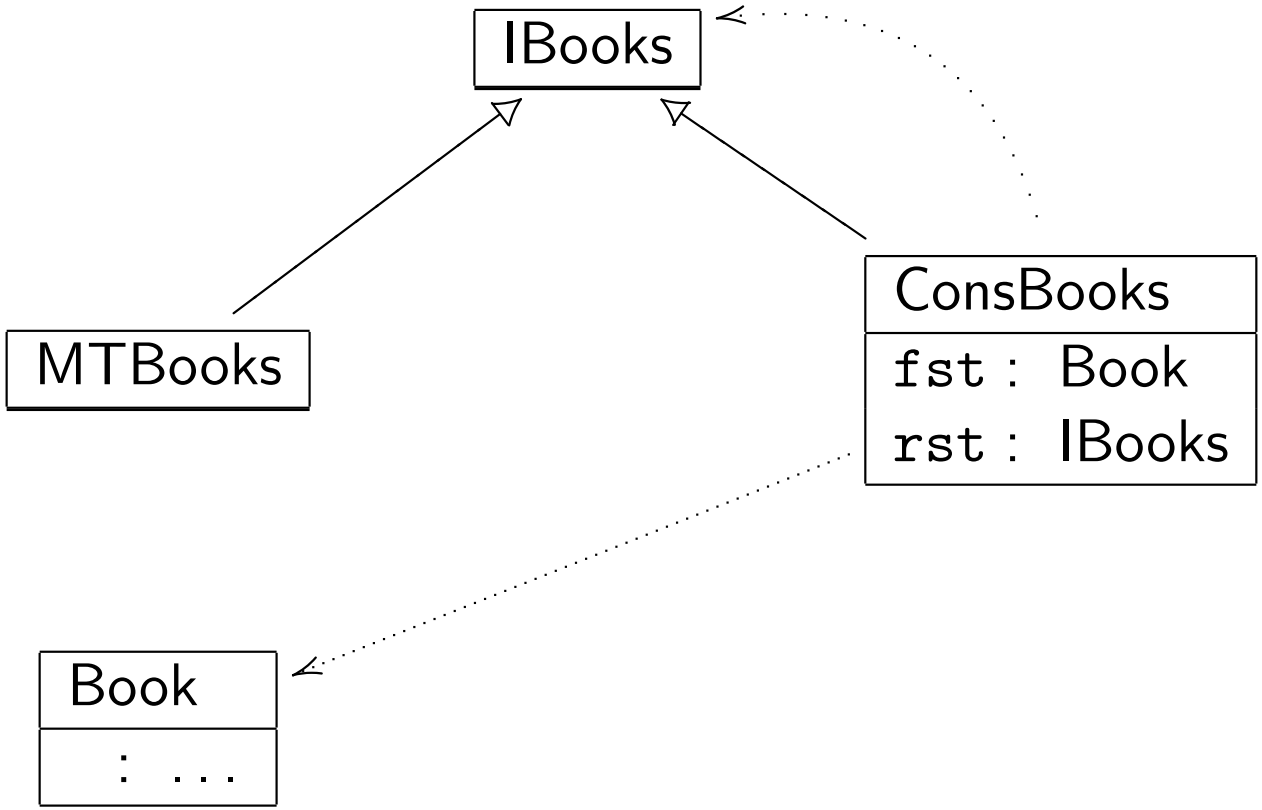
Listen sind überall

Listen von Tagebucheinträgen



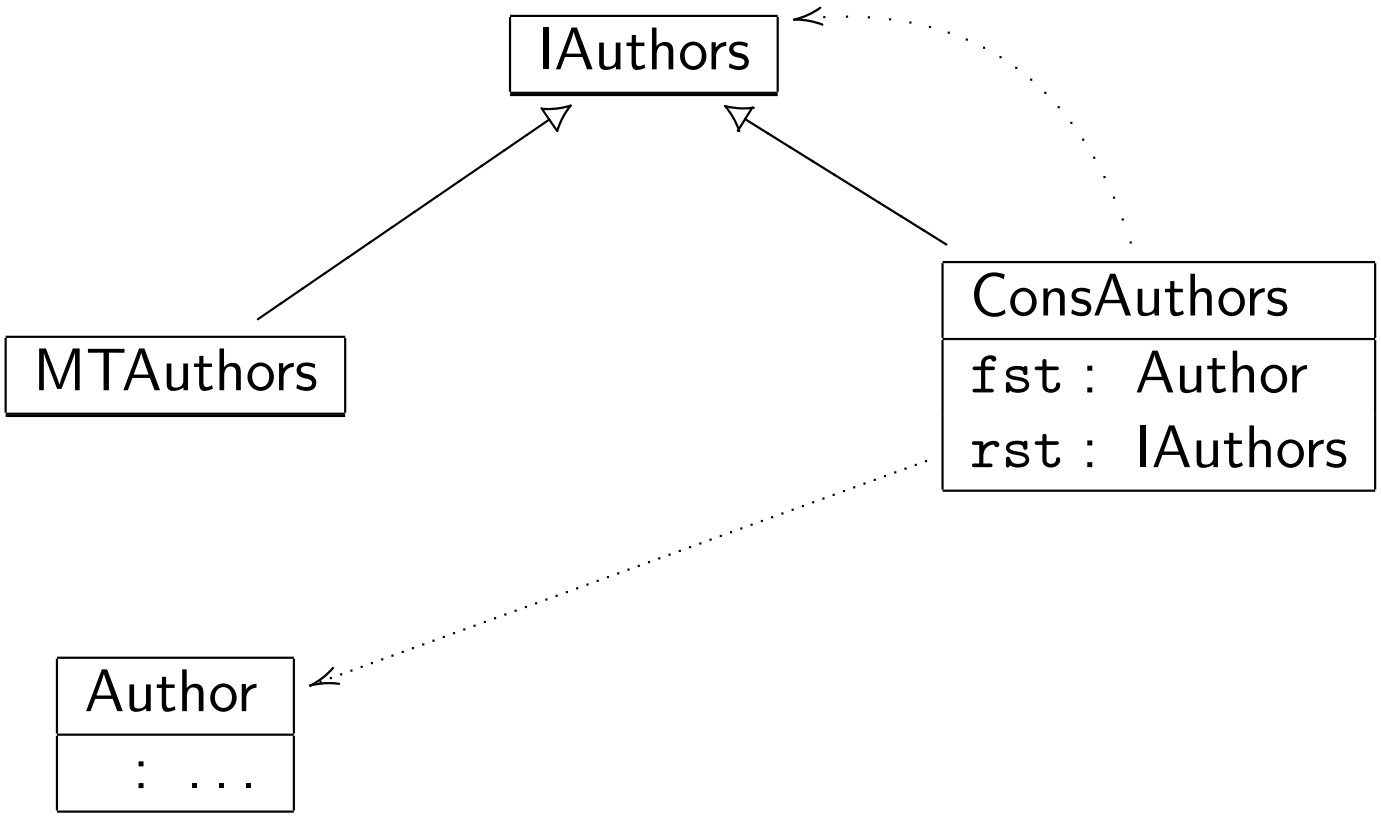
Listen sind überall

Listen von Büchern



Listen sind überall

Listen von Autoren



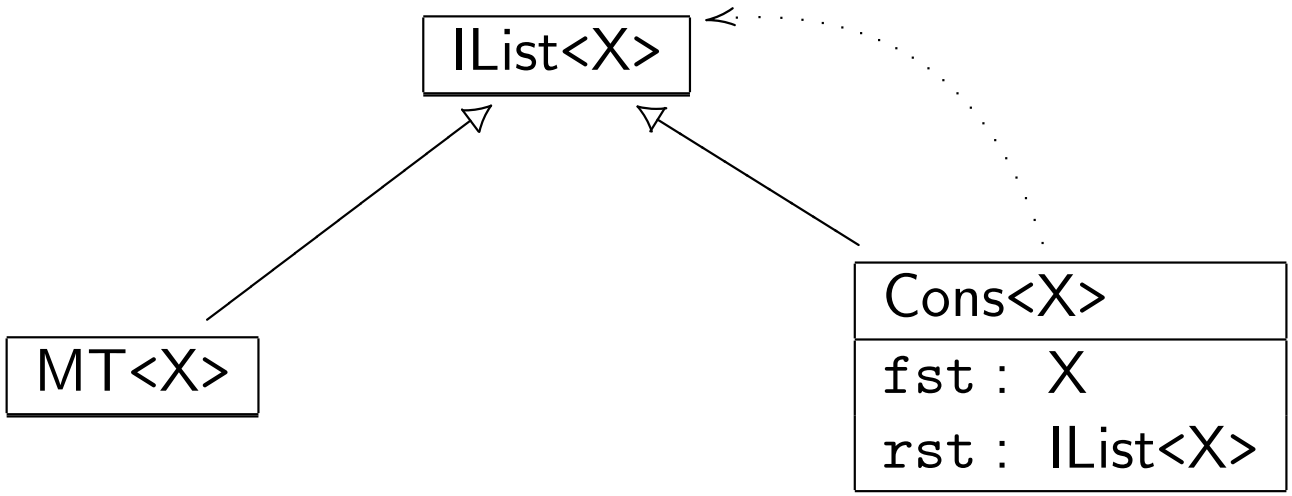
Abstraktion

- Die Klassendiagramme sind gleich (bis auf den Elementtyp).
- Die Implementierungen sind gleich (bis auf den Elementtyp).
- Naheliegender Wunsch: Vermeide die Wiederholung durch **Abstraktion des Deklarationsmusters vom Elementtyp**.

Abstraktion

- Die Klassendiagramme sind gleich (bis auf den Elementtyp).
- Die Implementierungen sind gleich (bis auf den Elementtyp).
- Naheliegender Wunsch: Vermeide die Wiederholung durch **Abstraktion des Deklarationsmusters vom Elementtyp**.
- Mittel dazu *Java Generics*
- Zunächst: generische Klassen und Interfaces

Generische Listen



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

Implementierung: Generische Listen

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

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

```
// Variante nicht-leere Liste
class Cons<X> implements IList<X> {
    private X fst;
    private IList<X> rst;

    public Cons (X fst, IList<X> rst) {
        this.fst = fst;
        this.rst = rst;
    }
}
```

Verwendung von generischen Listen

Liste von Tagebucheinträgen

```
// die Einträge der Liste
Entry e1 = new Entry (new Date (5,6,2003), 8.5, 27, "gut");
Entry e2 = new Entry (new Date (6,6,2003), 4.5, 24, "müde");
Entry e3 = new Entry (new Date (23,6,2003), 42.2, 150, "erschöpft");
// Aufbau der Liste
IList<Entry> i1 = new MT<Entry> ();
IList<Entry> i2 = new Cons<Entry> (e1, i1);
IList<Entry> i3 = new Cons<Entry> (e2, i2);
IList<Entry> i4 = new Cons<Entry> (e3, i3);
```

Verwendung von generischen Listen

Liste von Daten

```
// die Einträge
Date d1 = new Date (28,4,1789);
Date d2 = new Date (28,4,1945);
Date d3 = new Date (28,4,1906);
// Aufbau der Liste
IList<Date> i1 = new MT<Date> ();
IList<Date> i2 = new Cons<Date> (d1, i1);
IList<Date> i3 = new Cons<Date> (d2, i2);
IList<Date> i4 = new Cons<Date> (d3, i3);
```

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)

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

Generische Suche

Generische Suche

Filtere aus einer IList<Entry> *diejenigen aus, die ein bestimmtes Suchkriterium erfüllen.*

Beispiele

- Finde alle Läufe von mehr als 10km Länge.
- Finde alle Läufe im Juni 2003.
- ...

Generische Suche

Funktional

Alter Ansatz

Entwickle Methoden

- `IList<Entry> distanceLongerThan (double length);`
- `IList<Entry> inMonth (int month, int year);`
- ...

denen allen das Durchlaufen der Liste und das Zusammenstellen der Ergebnisliste gemeinsam ist.

Generische Suche

Funktional

Alter Ansatz

Entwickle Methoden

- `IList<Entry> distanceLongerThan (double length);`
- `IList<Entry> inMonth (int month, int year);`
- ...

denen allen das Durchlaufen der Liste und das Zusammenstellen der Ergebnisliste gemeinsam ist.

Generischer Ansatz

Entwickle *eine* Methode mit dieser Funktionalität und parametrisiere sie so, dass alle anderen Methoden Spezialfälle davon werden.

Generischer Ansatz

Generische Auswahl

- Definiere das Auswahlkriterium durch ein separates Interface ISelect, welches von Elementtypen erfüllt sein soll.
- Dieses Interface muss entsprechend über den Elementtypen parametrisiert sein:

```
// generische Auswahl
interface ISelect<X> {
    // ist obj das Gesuchte?
    public boolean selected (X obj);
}
```

- Entwurfsmuster *Strategy*
  - Suche mit abstrakter Selektion
  - Instantiiert durch konkrete Selektionen

Instanzen der generischen Auswahl

```
// teste ein Entry ob er eine längere Entfernung enthält
class DistanceLongerThan implements ISelect<Entry> {
    private double limit;
    public DistanceLongerThan (double limit) {
        this.limit = limit;
    }

    public boolean selected (Entry e) {
        return e.distance > this.limit;
    }
}
```

Instanzen der generischen Auswahl

```
// teste ob ein Entry in einem bestimmten Monat liegt
class EntryInMonth implements ISelect<Entry> {
    private ISelect<Date> selectdate;
    public EntryInMonth (int month, int year) {
        this.selectdate = new DateInMonth(month, year);
    }
    public boolean selected (Entry e) {
        return this.selectdate.selected (e.d);
    }
}
```

```
// teste ob ein Date in einem bestimmten Monat liegt
class DateInMonth implements ISelect<Date> {
    private int month; private int yearM;
    public DateInMonth (int month, int year) {
        this.month = month; this.year = year;
    }
    public boolean selected (Date d) {
        return d.month == this.month && d.year == this.year;
    }
}
```

Implementierung Generische Auswahl

- in IList<X>

```
public IList<X> filter (ISelect<X> pred);
```

- in MT<X>

```
public IList<X> filter (ISelect<X> pred) {
    return new MT<X>();
}
```

- in Cons<X>

```
public IList<X> filter (ISelect<X> pred) {
    IList<X> filteredrest = this.filter (pred);
    if (pred.selected (this.fst)) {
        return new Cons<X>(this.fst, filteredrest);
    } else {
        return filteredrest;
    }
}
```

Verwendung Generische Auswahl

Läufe von mehr als 10km Länge

```
IList<Entry> myRuns = ...;
ISelect<Entry> moreThan10 = new DistanceLongerThan (10);
IList<Entry> myLongRuns = myRuns.filter (moreThan10);
```

Verwendung Generische Auswahl

Läufe von mehr als 10km Länge

```
IList<Entry> myRuns = ...;
ISelect<Entry> moreThan10 = new DistanceLongerThan (10);
IList<Entry> myLongRuns = myRuns.filter (moreThan10);
```

Läufe im Juni 2003

```
IList<Entry> myRuns = ...;
ISelect<Entry> inJune2003 = new EntryInMonth (6, 2003);
IList<Entry> myJuneRuns = myRuns.filter (inJune2003);
// Alternative
IList<Entry> myJulyRuns = myRuns.filter (new EntryInMonth (7, 2003));
```

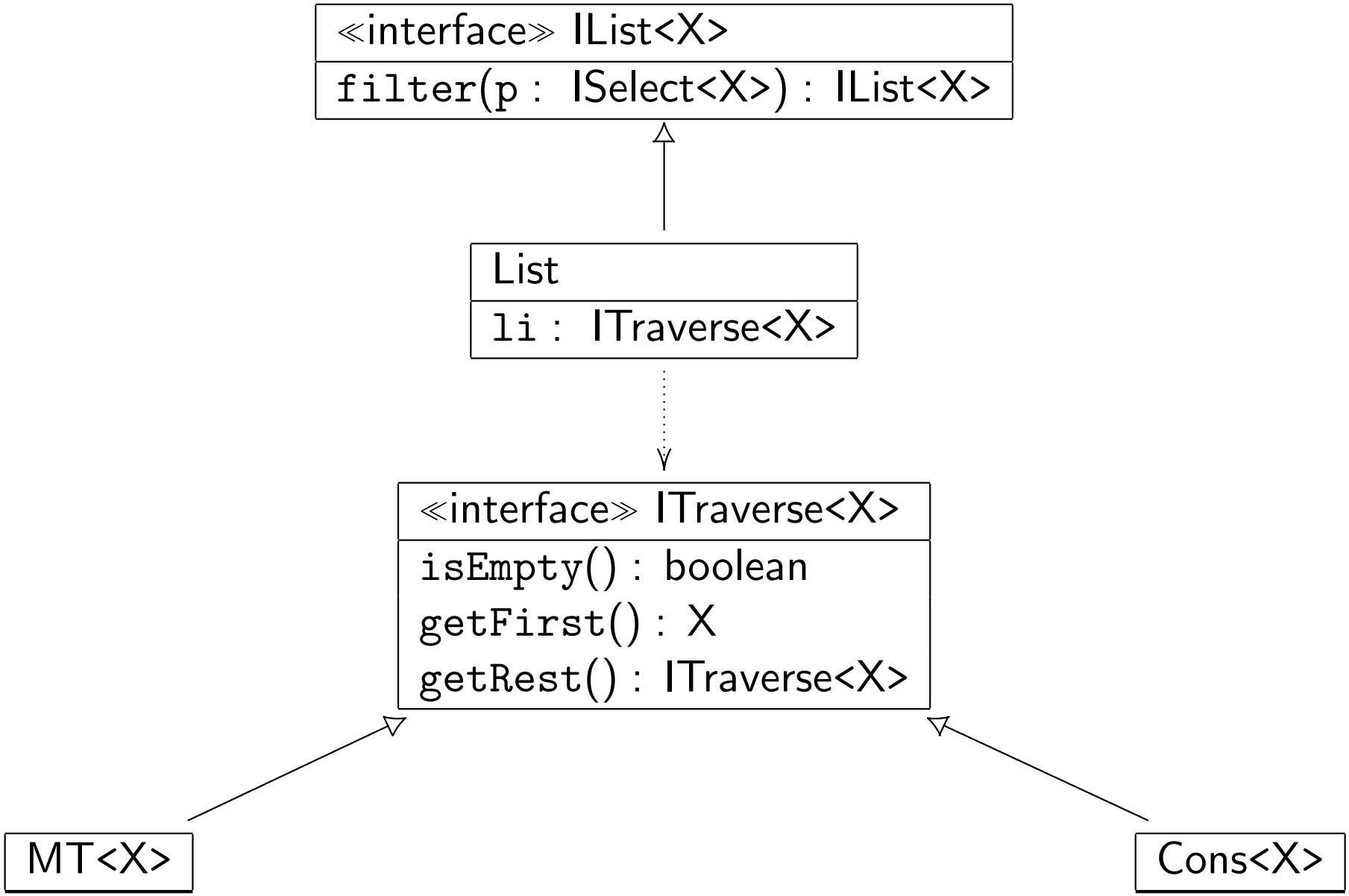
Generischer Durchlauf

Generischer Durchlauf

Idee des Durchlaufinterfaces

- Abkopplung der Funktionalität vom Durchlaufen der Datenstruktur
- Änderung der Implementierung der Datenstruktur ohne Änderung der Funktionalität
- Veränderliche Datenstrukturen

Organisation



Implementierung Generischer Durchlauf

- in ITraverse<X>

```
public boolean isEmpty ();
public X getFirst();
public IList<X> getRest();
```

- in MT<X> `implements` ITraverse<X>

```
public boolean isEmpty () { return true; }
public X getFirst() { return null; }
public IList<X> getRest() { return null; }
```

- in Cons<X> `implements` ITraverse<X>

```
public boolean isEmpty () { return false; }
public X getFirst() { return this.fst; }
public IList<X> getRest() { return this.rst; }
```

Implementierung Generische Suche

mit Durchlaufinterface

```
class List<X> implements IList<X> {
    private ITraverse<X> li;
    public List (ITraverse<X> li) { this.li = li; }
    public IList<X> filter (ISelect<X> pred) {
        ITraverse<X> newli = this.filterAux (this.li, pred);
        return new List (newli);
    }
    private ITraverse<X> filterAux (ITraverse<X> li, ISelect<X> pred) {
        if (!li.isEmpty()) {
            X elem = li.getFirst ();
            ITraverse<X> filteredrest = filterAux (li.getRest(), pred);
            if (pred.select (elem)) {
                return new Cons<X>(elem, filteredrest);
            } else {
                return filteredrest;
            }
        } else {
            return new MT<X>();
        }
    }
```

Implementierung Generische Suche

mit Durchlaufinterface und `while`

```
class List<X> implements IList<X> {
    private ITraverse<X> li;
    public List (ITraverse<X> li) { this.li = li; }
    public IList<X> filter (ISelect<X> pred) {
        ITraverse<X> newli = this.filterAux (pred);
        return new List (newli);
    }
    private ITraverse<X> filterAux (ISelect<X> pred) {
        ITraverse<X> li = this.li;
        ITraverse<X> acc = new MT<X>();
        while (!li.isEmpty()) {
            X elem = li.getFirst();
            if (pred.select(elem)) {
                acc = new Cons<X>(elem, acc);
            }
            li = li.getRest();
        }
        return acc;
    }
}
```

Listen transformieren

Listen transformieren

Aufgabe: Ändere alle Einträge im Lauftagebuch von km auf Meilen.

- Das Abändern von Einträgen macht auch für andere Listentypen Sinn.

$\Rightarrow$ entwerfe generische Methode

$\Rightarrow$ entwerfe zunächst Änderungsinterface

Listen transformieren

Aufgabe: Ändere alle Einträge im Lauftagebuch von km auf Meilen.

- Das Abändern von Einträgen macht auch für andere Listentypen Sinn.

$\Rightarrow$ entwerfe generische Methode

$\Rightarrow$ entwerfe zunächst Änderungsinterface

Änderungsinterface

```
// change something
interface ITransform<X> {
    public X transform (X x);
}
```

Listen transformieren

Funktional

- in IList<X>

```
public IList<X> transformAll (ITransform<X> f);
```

- in List<X>

```
public IList<X> transformAll (ITransform<X> f) {
    ITraverse<X> newli = this.transformAux (this.li, f);
    return new List (newli);
}
private ITraverse<X> transformAux (ITraverse<X> li, ITransform<X> f) {
    if (!li.isEmpty()) {
        X elem = li.getFirst ();
        ITraverse<X> transformedrest = transformAux (li.getRest(), f);
        return new Cons<X>(f.transform (elem), transformedrest);
    } else {
        return new MT<X>();
    }
}
```

Km in Meilen umwandeln

```
class ChangeKmToMiles implements ITransform<Entry> {
    public ChangeKmToMiles () {}
    // Umrechnungsformel
    private static double kmToMiles (double km) {
        return km * 0.6214;
    }
    // Transformation
    public Entry transform (Entry e) {
        return new Entry (e.d,
                          kmToMiles(e.distance),
                          e.duration,
                          e.comment);
    }
}
```

Km in Meilen umwandeln

```
class ChangeKmToMiles implements ITransform<Entry> {
    public ChangeKmToMiles () {}
    // Umrechnungsformel
    private static double kmToMiles (double km) {
        return km * 0.6214;
    }
    // Transformation
    public Entry transform (Entry e) {
        return new Entry (e.d,
                          kmToMiles(e.distance),
                          e.duration,
                          e.comment);
    }
}
```

Verwendung

```
IList<Entry> logInKm = ...;
ITransform<Entry> kmToMiles = new ChangeKmToMiles ();
IList<Entry> logInMiles = logInKm.transformAll (kmToMiles);
```

Intermezzo: Vergleichen mit Generics

Vergleichen mit Generics

```
package java.lang;
interface Comparable<T> {
  int compareTo (T that);
}
```

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Verwendung

```
Integer i1 = new Integer (42);
Integer i2 = new Integer (4711);
int result = i1.compareTo (i2);
// result < 0
```

Vergleichbar machen

```
class Entry implements Comparable<Entry> {
  ...
  // Vergleich für Comparable<Entry>
  public int compareTo (Entry that) {
    int result = this.d.compareTo (that.d);
    if (result == 0) {
      double r1 = this.distance − that.distance;
      if (r1 < 0.1) {
        result = this.duration − that.duration;
        if (result == 0) {
          return this.comment.compareTo (that.comment);
        } else {
          return result;
        }
      } else {
        return (int)(10∗r1);
      }
    } else {
      return result;
    }
  }
}
```

Finite Map

Zurück zum Weingroßhändler

Generische "Finite Map"

Ein Weingroßhändler will seine Preisliste verwalten. Er wünscht folgende Operationen

- *zu einem Wein den Preis ablegen,*
- *einen Preiseintrag ändern,*
- *den Preis eines Weins abfragen.*

- Abstrakt gesehen ist die Preisliste eine **endliche Abbildung** von Wein (repräsentiert durch einen String) auf Preise (repräsentiert durch ein Integer). (*finite map*)
- Da in der Preisliste einige tausend Einträge zu erwarten sind, sollte sie als Suchbaum organisiert sein.

Finite Map

- Das Suchproblem erfordert ein Interface FiniteMap<> gesucht, das den Definitionsbereich (Schlüssel, *key*) und den Wertebereich (*value*) der Abbildung festlegt.

$\Rightarrow$ Das Interface benötigt **zwei Parameter**, Key und Value.

Finite Map

- Das Suchproblem erfordert ein Interface FiniteMap<> gesucht, das den Definitionsbereich (Schlüssel, *key*) und den Wertebereich (*value*) der Abbildung festlegt.

$\Rightarrow$ Das Interface benötigt **zwei Parameter**, Key und Value.

- Für die Suchbaumeigenschaft muss Key vergleichbar sein, d.h. es muss gelten

```
Key implements Comparable<Key>
```

Finite Map

- Das Suchproblem erfordert ein Interface FiniteMap<> gesucht, das den Definitionsbereich (Schlüssel, *key*) und den Wertebereich (*value*) der Abbildung festlegt.

$\Rightarrow$ Das Interface benötigt **zwei Parameter**, Key und Value.

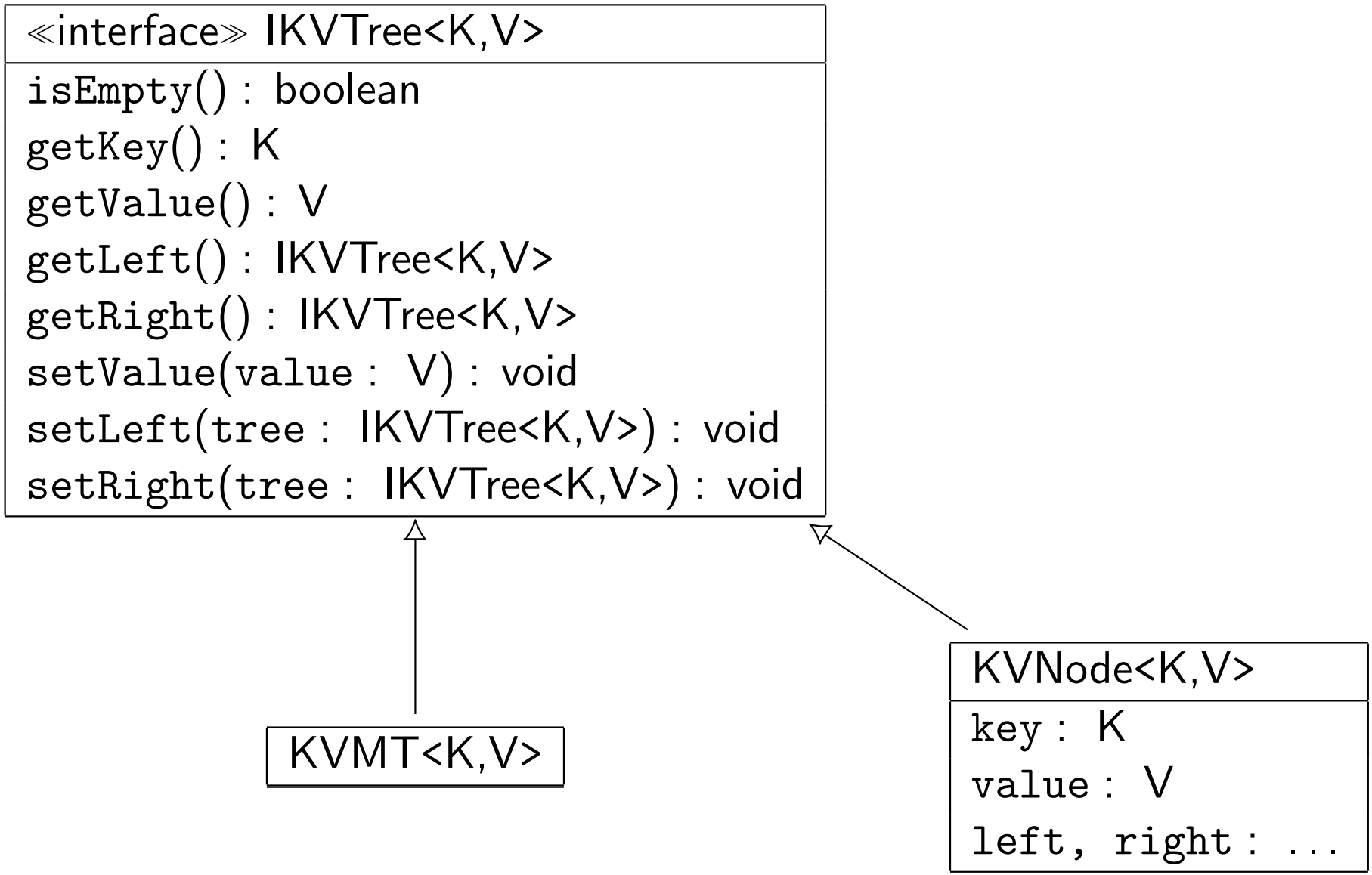
- Für die Suchbaumeigenschaft muss Key vergleichbar sein, d.h. es muss gelten

```
Key implements Comparable<Key>
```

- Als Vorbedingung (*constraint*) im Interface:

```
interface FiniteMap<Key implements Comparable<Key>,Value> {
    // liefert den mit key assoziierten Wert oder null
    Value find (Key key);
    // legt eine neue Assoziation ab
    void put (Key key, Value value);
}
```

Durchlaufinterface: Generischer, imperativer Binärbaum



Implementierung der FiniteMap

Suchen

```
class BTreeMap<K implements Comparable<K>, V>
    implements FiniteMap<K, V> {
    private IKVTree<K, V> bt;
    public BTreeMap () { this.bt = new KVMT<K,V>(); }
    public Value find (K key) {
        IKVTree<K, V> scan = bt;
        while (!scan.isEmpty()) {
            int cmp = key.compareTo(scan.getKey());
            if (cmp == 0) {
                return scan.getValue();
            } else if (cmp < 0) {
                scan = scan.getLeft();
            } else {
                scan = scan.getRight();
            }
        }
        // nicht gefunden
        return null;
    }
}
```

Implementierung der FiniteMap

Eintragen

```
public void put (K key, V value) {
    IKVTree<K,V> scan = bt;
    IKVTree<K,V> next;
    while (!scan.isEmpty()) {
        int cmp = key.compareTo(scan.getKey());
        if (cmp == 0) {
            scan.setValue (value); return;
        } else if (cmp < 0) {
            next = scan.getLeft();
            if (next.isEmpty ()) {
                scan.setLeft (mkNode (key, value)); return;
            }
        } else {
            next = scan.getRight();
            if (next.isEmpty ()) {
                scan.setRight (mkNode (key, value)); return;
            }
        }
        scan = next;
    }
```

Verwendung

```
FiniteMap<String,Integer> winelist = new BTreeMap<String,Integer> ();
//
winelist.put ("Chateau Latour 1953 1ere Grand Cru Classe Pauillac", 76007);
winelist.put ("Pommery Grand Cru Vintage Champagne 1989 Methuselah", 68417);
winelist.put ("Dom Perignon Vintage Champagne 1999", 13934);
//
winlist.find ("Asti Spumante"); // == null
```

Nächstes Thema: Java Collections