ETH zürich

Informatik II - Übung 07

Katja Wolff
katja.wolff@inf.ethz.ch

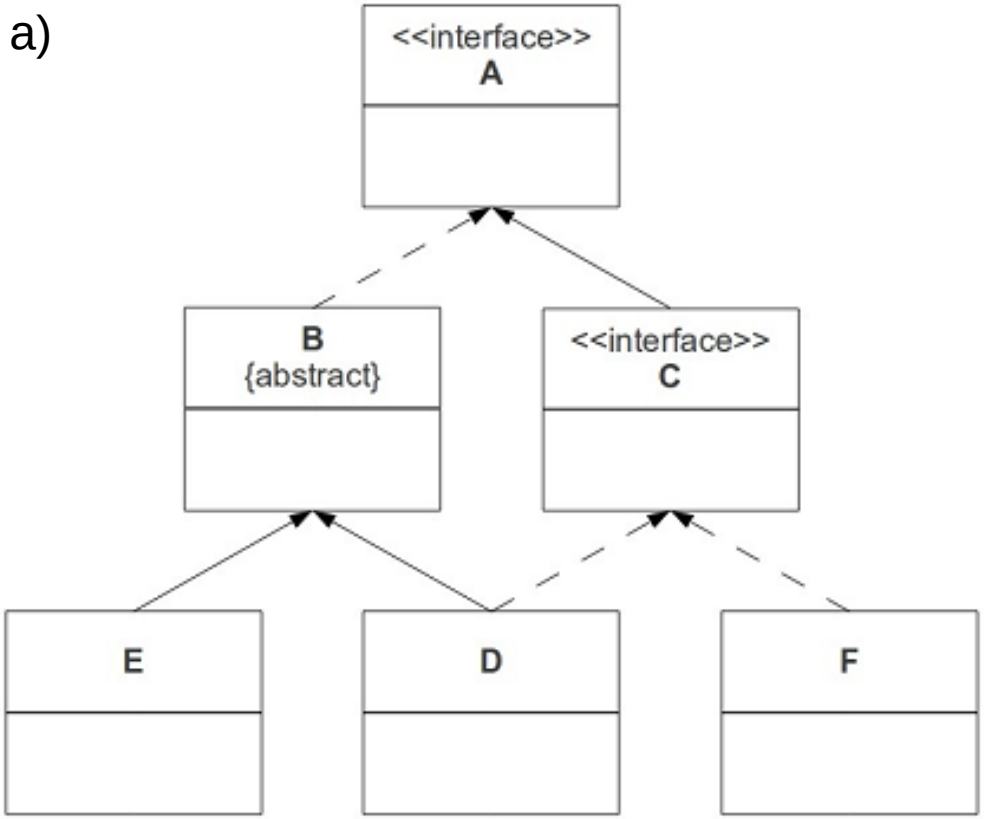
Besprechung Übungsblatt 6

Übungsblatt 6

- 1.) Klassen, Schnittstellen, Typumwandlung
- 2.) Schnittstellen und Implementierungen
- 3.) Polymorphie
- 4.) Stacks und Optimierung

U6.A1: Klassen, Interfaces, Casts

a)

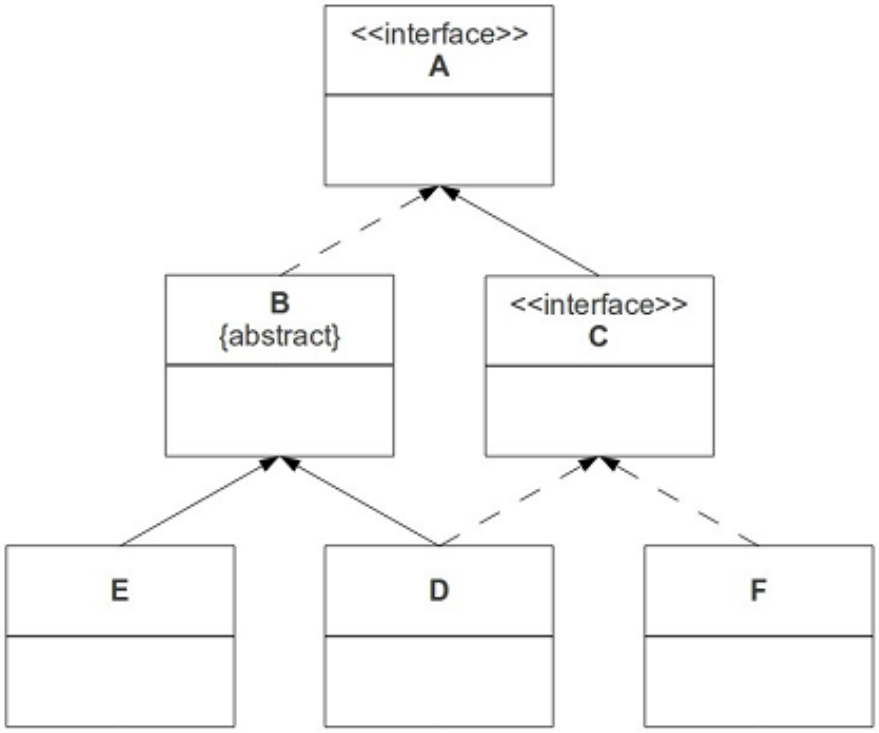


b) Instanziiert werden können: (D, E, F)

U6.A1: Klassen, Interfaces, Casts

- Statischer Cast (*implicit cast*)
  - Nur von Kindklasse zu Elternklasse

```
public static void c1()
{
    public static void c2()
    {
        public static void c3()
        {
            public static void c4()
            {
                public static void c5()
                {
                    public static void c6()
                    {
                        C c = new F();
                        A a = c;
                        B b = c;
                        D d = c;
                        E e = c;
                        F f = c;
                    }
                }
            }
        }
    }
}
```

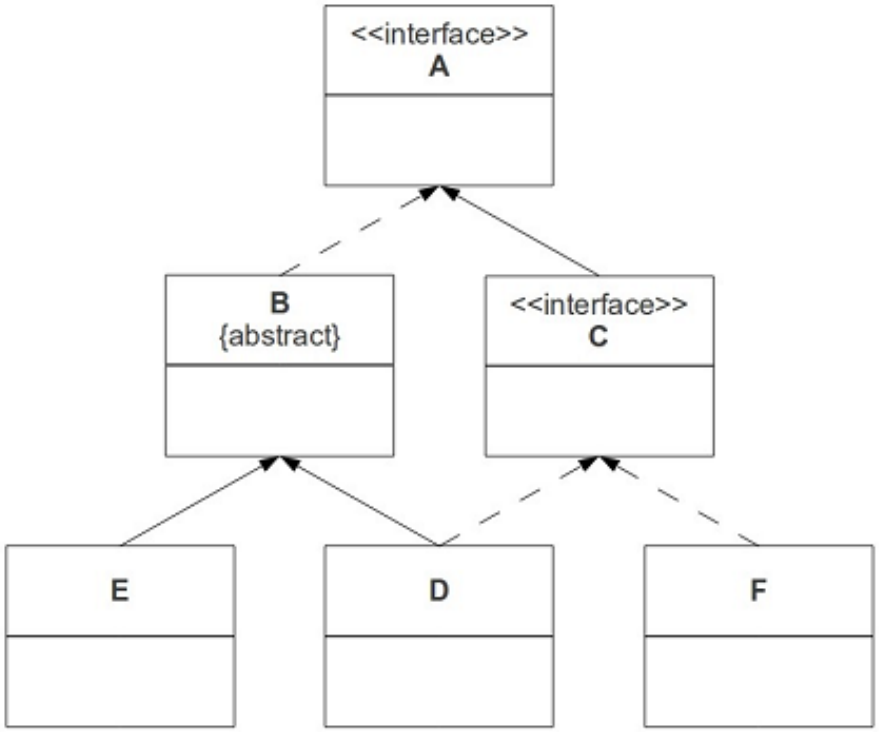


U6.A1: Klassen, Interfaces, Casts

- Statischer Cast (*implicit cast*)
  - Nur von Kindklasse zu Elternklasse

- Dynamisch (*explicit cast*)
  - `T t = (T)obj;`
  - Geht, falls `obj` vom Typ `T` ist (inkl. alle Kindertypen von `T`)

```
public static void d1()
{
}
public static void d2()
{
}
public static void d3()
{
    B b = new D();
    A a = (A) b;
    C c = (C) b;
    D d = (D) b;
    E e = (E) b;
}
```



U6.A2: Implementierung von IStack

a) Fabrikmethode:

```
public class StackFactory {
  public static IStack create() {
    //return new u6a4.ChunkedStack();
    return new ListStack();
  }
}
```

b) empty():

```
public boolean empty();
```

c) Test:

```
@Test public void empty() {
  IStack stack = StackFactory.create();
  Assert.assertTrue(stack.empty());
  stack.push(42);
  Assert.assertFalse(stack.empty());
}
```

U6.A3: ListUtils

a)

- Fabrikmethode: praktisch wie in A2.
- Interface IListUtils muss in ListUtils implementiert werden:

```
public interface IListUtils {
  public String toString(GenericList list)
  public GenericList add(GenericList list, Object value);
  public int size(GenericList list);
  public GenericList insertSorted(GenericList list, Object value);
  public GenericList sort(GenericList list);
}
```

b) Einfach, wenn man a) verstanden hat.

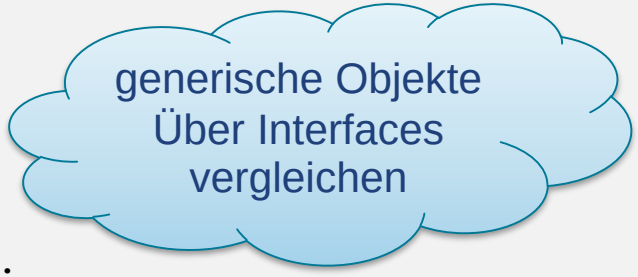
c) Nächste Folie:

U6.A3: ListUtils

```
/**
 * Inserts a value into a sorted list so that
 * the resulting list is still sorted.
 * The sort order is ascending.
 */
private GenericList insertSorted(GenericList list, Object value)
{
   if (list == null)
      return new GenericList(value, null);

   Comparable lhs = (Comparable) value;
   Comparable rhs = (Comparable) list.value;
   if (lhs.smallerThan(rhs))
      return new GenericList(value, list);

   list.next = insertSorted(list.next, value);
   return list;
}
```



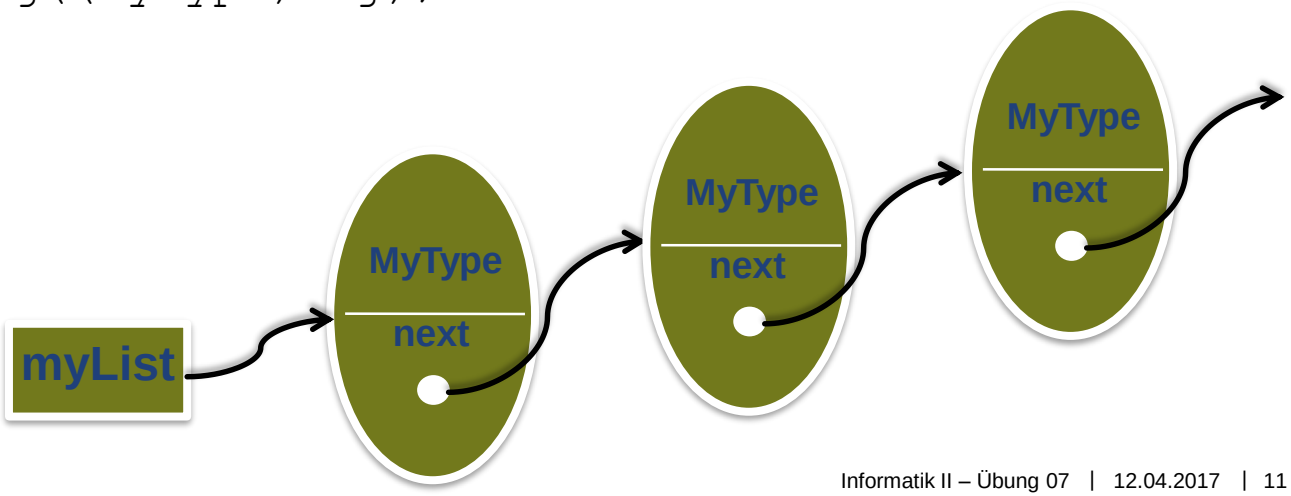
Vorschau
Blatt 7

U7.A1: Generics

- Was sind Generics?
- Vorteile?

- Beispiel: Liste mit Elementen vom Typ `MyType`

```
Object obj = list.getNext();
If(obj instanceof MyType)
    doSomething((MyType)obj);
```



U7.A1: Generics

- Container `ArrayList`
- Zweifach geschachtelt
  - Es gibt viele Gruppen

    ```
    ArrayList<ArrayList<Student>> groups;
    ```

  - Eine Gruppe besteht aus Studenten

    ```
    ArrayList<Student> group
    ```

- Ziel: Filter, der abfragt, wer das Testat erhält

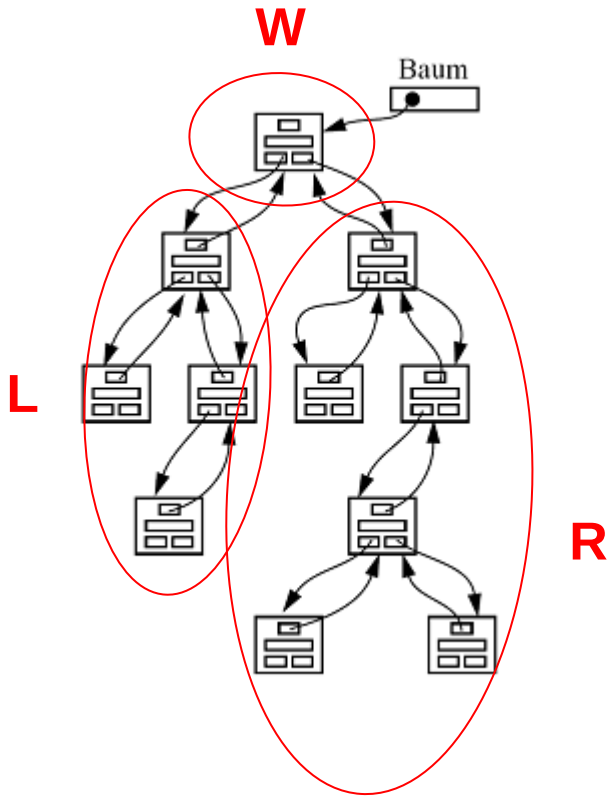
→ Haltet euch an Aufgabentext – Filter-Funktion besteht aus verschachelter Schleife, in der geprüft wird, ob der Student genügend Punkte hat. Ergebnis: Liste aller Studenten, die das Testat bestehen. *Hinweis*: Bei (b) müsst ihr einiges casten, bei (c) nicht mehr!

U7.A2: Binärbäume

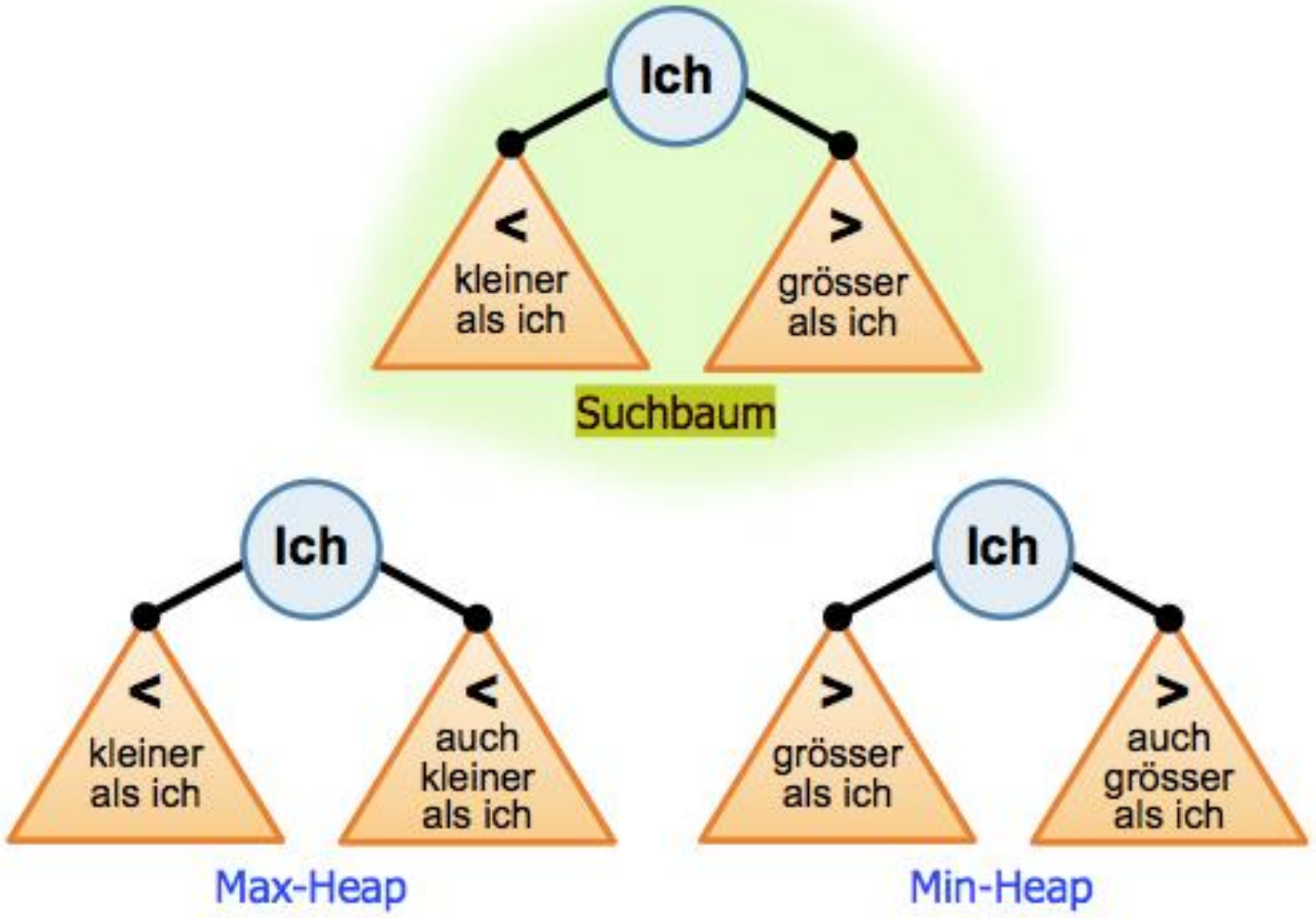
- Was ist ein Binärbaum?

- Jeder Knoten enthält Zeiger auf:
  - Linker Nachfolger
  - Rechter Nachfolger
  - (Vater)

- Verschiedene Traversierungen:
  - Pre-order: W-L-R
  - In-order: L-W-R
  - Post-order: L-R-W



Heaps vs. Suchbäume



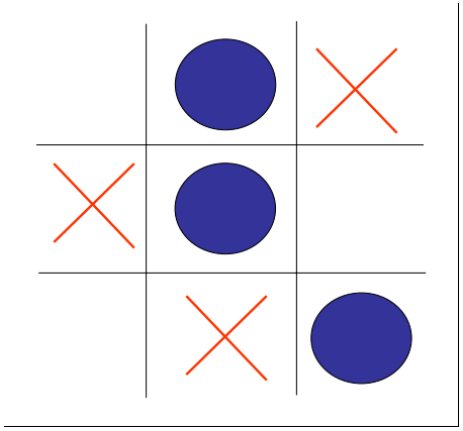
U7.A2: Binäre Suchbäume

- Struktur – Jeder Knoten hat ...
  - Ein Schlüsselattribut (key)
  - Eine Referenz auf ein Datenelement (record)
  - Verweise auf die Kinder (left, right)
  - Evtl. Verweis auf das Elternelement (parent)

- Die Menge der Schlüsselattribute ist total geordnet ($a \leq b$). Für jeden Knoten mit Schüsselattribut s gilt:
  - Alle Schlüssel im linken Unterbaum sind kleiner als s
  - Alle Schlüssel im rechten Unterbaum sind grösser als s

- Elementare Methoden (sh. Vorlesung): Suchen, Höhe berechnen, Einfügen, Löschen

U7.A2: Tic-Tac-Toe-Spielbaum



Und nun zur Aufgabe! Tic-Tac-Toe

a) Baum für restlichen Verlauf des Spiels
b) Markieren des Baumes bzgl. Aussicht auf Gewinn
c) Könnt ihr gewinnen?

U7.A3: Binäre Suchbäume

a) Löschen von Elementen aus einem gegebenen Baum: Ersetzen durch kleinstes Element des rechten Teilbaums

b) Implementierung!

- Implementieren von `IBinarySearchTreeUtils<T>`

- `UtilsFactory.create()` soll ein Utils-Objekt für den Typ `String` erzeugen → `new MyTreeUtils<String>()`

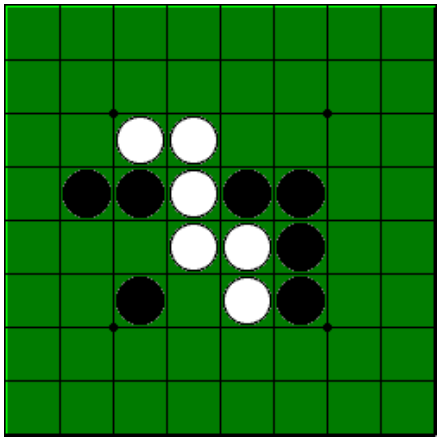
Folgende Methoden müssen implementiert werden (sh. Vorlesungsslides!): `height`, `isLeaf`, `hasOneChild`, `preOrder`, `inOrder`, `postOrder`, `insert`, `find`, `remove`

U7.A3: Binäre Suchbäume

- Aus der Vorlesung: Höhe eines Baumes rekursiv berechnen?

```
int height() {
  if (left!=null && right!=null)
    return 1 + Math.max(left.height(),
                        right.height());
  else if (left!=null)
    return 1+left.height();
  else if (right!=null)
    return 1+right.height();
  else
    return 0;
}
```

U7.A3: Reversi!!!



- Serie von Aufgaben
- Spielregeln und weitere Infos:
  - http://www.vs.inf.ethz.ch/edu/FS2017/I2/reversi
  - Login für reversi-papers:
    - username: i2bib
    - password: reversi
- Turnier am Ende des Semesters
  - Super Preise!

U7.A3: Reversi

- Zunächst: Grundprinzipien
- Später: „Strategien“ für den Computerspieler
  - Spieltheorie
- Prüfungsrelevanter Stoff (MinMax, Alpha-Beta-Suche, ...), der in der Reversi-Übung vertieft wird

U7.A3: Reversi

a) Aufsetzen des Frameworks
- Bindet die JavaDoc ein (muss in Eclipse explizit gemacht werden)
- Schaut euch die Möglichkeiten des Gameboards an
  - `Gameboard gb;`
  - Screenshot an mich

b) Implementierung „Random Player“
- `nextMove()`-Methode
- Gültiger Move? `gb.checkMove()`

…viel Spass!