Aufgabe: Generische Bäume

Warum ist es meist schwieriger, einen generischen Baum zu implementieren als eine generische Liste?

A: Weil man stets zwischen den beiden Teilbäumen unterscheiden muss.

B: Weil die Elemente miteinander vergleichbar sein müssen.

C: Weil man für Iteratoren keine Rekursion verwenden kann.

D: Falsch! Wenn man es richtig macht, ist es nicht schwieriger.

Strategien

Vorgefertigte Teile – Hindernisse

Hinderung an der Verwendung fertiger Teile durch

- Unkenntnis,
- unterschiedliche Modelle,
- mangelndes Vertrauen,
- „Ich kann es besser“,

Verwendung zahlt sich aus, ist aber mit Arbeit verbunden

Top-Down

```
public static void main(String[] args) {
    List<Integer> nums = readNums();
    Collections.sort(nums);
    print(nums);
}
private static List<Integer> readNums() {
    Scanner sc = new Scanner(System.in);
    List<Integer> nums = new LinkedList<Integer>();
    while (sc.hasNextInt())
        nums.add(sc.nextInt());
    return nums;
}
private static void print(List<Integer> nums) {
    for (int i: nums)
        System.out.println(i);
}
```

Bottom-Up

```
public class SortedNums {
    private GenTree<Integer> nums = new GenTree<Integer>();
    public void readFrom(Scanner in) {
        while (in.hasNextInt())
            nums.add(in.nextInt());
    }
    public void printTo(PrintStream out) {
        for (int i: nums)
            out.println(i);
    }
    public static void main(String[] args) {
        SortedNums nums = new SortedNums();
        nums.readFrom(new Scanner(System.in));
        nums.printTo(System.out);
    }
}
```

Aufgabe: Warum Strategien?

Warum soll man bewährte Strategien einsetzen?

A: Damit man nicht immer wieder das Rad neu erfinden muss.

B: Damit die Programme durchschnittlich effizienter laufen.

C: Damit man eher die Gesamtheit im Blick hat als einzelne Teile.

D: Damit man etwas davon hat, dass man sie gelernt hat.

Qualitätssicherung

Qualitätssicherung – Themenübersicht

Spezifikation ..... festlegen was zu erreichen ist

Programmverständnis ..... verstehen ohne probieren

Testen ..... praktisch ausprobieren

Ablauf nachvollziehen ..... interne Details nachprüfen

Ausnahmebehandlung ..... was tun im Fehlerfall?

Validierung ..... sind Daten und Programme brauchbar?

Spezifikationen

Was ist eine Spezifikation?

mehr oder weniger rigorose Beschreibung eines Systems

Anforderungsdokumentation

wird im Laufe der Entwicklung genauer (Zusicherungen)

spätestens Implementierung macht Spezifikation formal

Spezifikation ist Basis für

- statisches Verstehen,
- Verifikation,
- Testen

Design-by-Contract (DbC)

Spezifikation als Vertrag zwischen Server und Client

Objekt = Server: bietet Dienste (Methoden) an,
Objekt = Client: nutzt Dienste eines Servers

Bestandteile des Software-Vertrags sind

- Methodensignaturen in Interfaces und Klassen,
- durch Namen suggerierte Eigenschaften,
- Zusicherungen auf Schnittstellen (auch Kommentare),
- Usancen = übliche Gepflogenheiten (stillschweigend)

Typischer Software-Vertrag

Client kann Nachricht an Server senden, wenn

- entsprechende Methode in Server sichtbar,
- Argumenttypen Untertypen der formalen Parametertypen,
- alle Vorbedingungen der Methode erfüllt

Server garantiert nach Ausführung der Methode, dass

- Objekt des Ergebnistyps zurückgegeben,
- Nachbedingungen der Methode erfüllt,
- Invarianten und History-Constraints des Servers erfüllt
- (Achtung: Invariante $\neq$ Schleifeninvariante)

Zusicherungen auf Objektschnittstellen

Arten aus fortlaufendem Text herauslesen:

Vorbedingung: Einschränkung auf Parameter typisch
= alles, worum sich Aufrufer kümmern muss,
= Bedingung, die vor Methodenausführung erfüllt sein muss

Nachbedingung: was Methode tun und zurückgeben soll
= Großteil der Zusicherungen auf Methodenschnittstelle

Invariante: unveränderliche Eigenschaft des Objekts
= was sich auf ganzes Objekt bezieht (nicht einzelne Methode)

History-Constraint: Änderung des Objektzustands
= was sich auf ganzes Objekt oder Gruppe von Methoden bezieht

Zusicherungen und Untertypen

Vorbedingung:

im Untertyp schwächer als im Obertyp

im Untertyp Verknüpfung mit ODER

Bsp.: Obertyp: x > 0 bzw. y != null
Untertyp: x >= 0 bzw. jedes y erlaubt

Nachbedingung, Invariante, History-Constraint:

im Untertyp stärker als im Obertyp

im Untertyp Verknüpfung mit UND

Bsp.: Obertyp: x >= 0 bzw. jedes y erlaubt
Untertyp: x > 0 bzw. y != null

Usancen

nicht ausdrücklich angeschriebene übliche Zusicherungen
gelten dennoch, weil man sich darauf verlässt

wichtig: **Objektzustand darf sich nicht unerwartet ändern**

Änderung des Objektzustands verboten, wenn nicht durch Methodennamen
oder Zusicherungen ausdrücklich erlaubt

Änderungen, die kein Client sehen kann, sind aber erlaubt,
z.B. Teilergebnisse speichern statt immer neu berechnen

durch Namen suggerierte Eigenschaften müssen gelten