Typische Lösungsstrategien

Vorgefertigte Teile

- Collection, Iterator: Paradebeispiele für vorgefertigte Teile
- schreibt man kaum selbst, sondern verwendet sie einfach
- erprobt und gut unterstützt

```
static void printAll (Collection<String> ws) {
    for (String s: ws)
        System.out.println(s);
}
```

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);
    }
}
```

Schrittweise Verfeinerung

- große Aufgaben $\Rightarrow$ Erfahrungen mit Programm erst spät
- zuerst kleinen Teil lösen, schrittweise vervollständigen
- kann flexibel auf Änderungen reagieren
- schwierigste und wichtigste Teile zuerst
- Kosten und Fortschritt schwer planbar
- führt zu inkrementellen und agilen Prozessen

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 des Programms
  - Verifikation des Programms
  - Testen des Programms

Design by Contract

- Spezifikation als Vertrag zwischen Server und Client
- Objekt = Server: bietet Dienste an
- Objekt = Client: nutzt Dienste eines Servers
- Vertragsbestandteile sind
  - Methodensignaturen in Interfaces und Klassen
  - durch Namen suggerierte Eigenschaften
  - Zusicherungen (meist als Kommentare)

Typischer 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:

- Objekt des Ergebnistyps zurückgegeben
- Nachbedingungen der Methode erfüllt
- Invarianten des Servers erfüllt

Zusicherungen und Untertypen

Vorbedingung:

- im Untertyp schwächer als im Obertyp
- Verknüpfung mit ODER

Nachbedingung, Invariante:

- im Untertyp stärker als im Obertyp
- Verknüpfung mit UND

Unterscheidung von Zusicherungen

Arten aus fortlaufendem Text herauslesen:

Vorbedingung: Einschränkungen auf Parametern

alles, worum sich Aufrufer kümmern muss

Nachbedingung: was Methode tut

Großteil der Zusicherungen

Invariante: unveränderliche Eigenschaften des Objekts

Einfache Zusicherung

```
// gib mittlere Zahl in nums zurück
// nums sortiert
public static int median (int[] nums) {
    return nums[nums.length / 2];
}
```

Abstraktion statt Zusicherung

```
import java.util.Arrays
public class SortedIntArray {
    private int[] elems;   // elems bleibt stets sortiert
    public SortedArray (int[] e) {
        elems = Arrays.copyOf (e, e.length);
        Arrays.sort(elems);
    }
    public int median() {  // gib mittlere Zahl zurück
        return elems[elems.length / 2];
    }
    public boolean member (int x) {  // x enthalten?
        ... /* binäre Suche */
    }
}
```

Arten der Typäquivalenz

Namensgleichheit in statischen Sprachen wie Java

- explizite Spezifikation der Untertypbeziehungen
- können implizit Zusicherungen ausdrücken
- umständlicher, aber sicherer

Strukturgleichheit in dynamischen Sprachen

- Untertypbeziehungen ergeben sich automatisch
- keine Zusicherungen ausdrückbar
- erfordert viel Programmierdisziplin