# Klausur Softwaretechnik

**6.9.2007**

Prof. Dr. Walter F. Tichy
Dipl.-Inform. T. Gelhausen
Dipl.-Inform. G. Malpohl

Hier das Namensschild aufkleben.

Zur Klausur sind keine Hilfsmittel und kein eigenes Papier zugelassen.
Die Bearbeitungszeit beträgt 60 Minuten.
Die Klausur ist vollständig und geheftet abzugeben.
Mit Bleistift oder roter Farbe geschriebene Angaben werden nicht bewertet.

| **Aufgabe** | **1** | **2** | **3** | **4** | **5** | **∑** |
|---|---|---|---|---|---|---|
| Maximal | 17 | 3 | 17 | 16 | 7 | **60** |
| K1 | | | | | | |
| K2 | | | | | | |
| K3 | | | | | | |

## Aufgabe 1: Aufwärmen (17P)

a.) Nennen Sie die 8 in der Vorlesung vorgestellten Erhebungstechniken, die bei der Anforderungsanalyse eingesetzt werden. (2P)

Introspektion, Dokumenten-/Datenanalyse, Interviews, Umfragen bzw. Fragebögen, Gruppentechniken, Prototypen, Anwendungsfälle, Ethnografie
(0,25P pro Begriff, aufrunden auf 0,5P)

b.) Wie definieren wir „Zustand eines Objektes“ in der objektorientierten Analyse? (1P)

Zustand: Solange sich ein Objekt in einem Zustand befindet, reagiert es im gleichen Kontext immer gleich auf seine Umwelt. Ändert sich der Zustand, reagiert das Objekt in mindestens einem Kontext anders als zuvor. (Außensicht)

c.) Definieren Sie das „Substitutionsprinzip“ der objektorientierten Programmierung. (1P)

**Ist-ein-Semantik oder Substitutionsprinzip**: Jedes Exemplar einer Unterklasse hat die gleichen Eigenschaften* (0,5), die ein Exemplar der Oberklasse hätte; es lässt sich genau so verwenden (0,5).
*Eigenschaften: Attribute, Assoziationen, Zusicherungen und Zustände.

d.) Nennen Sie die in der Vorlesung behandelten Arten der Parameter-Varianz in objektorientierten Programmiersprachen und definieren Sie sie. (2P)

**Kovarianz** (Verwendung einer Spezialisierung des Parametertyps in der überschreibenden Methode)
**Kontravarianz** (Verwendung einer Verallgemeinerung des Parametertyps in der überschreibenden Methode)
**(Invarianz** keine Modifikation des Typs)
(nur Begriffe 1P, ab 2 von 3 (inkl. Erklärung!) volle Punktzahl)

e.) Beschreiben Sie, wie man mittels der „schrittweisen Verfeinerung“ von einer komplexen Problemstellung zu einer abstrakten Maschine gelangt. (2P)

Bei der schrittweisen Verfeinerung gliedert man die Problem- bzw. Aufgabenstellung in Teilprobleme (1P) auf (top-down, teile-und-herrsche)
Man verfeinert die Teilproblemstellung solange, bis man auf dem Niveau einer abstrakten Maschine angekommen ist (1P), die zur Lösung der Blätter der Problemstellungshierarchie eingesetzt werden kann.

f.) Viele Anfänger haben Probleme, die beiden Entwurfsmuster „Stellvertreter“ und „Kompositum“ zu unterscheiden. Beschreiben Sie Unterschiede und Gemeinsamkeiten zwischen den beiden Entwurfsmustern. (2P)

Gemeinsamkeit (1P): Die nach den Entwurfsmustern benannten Objekte nehmen den Platz der „eigentlichen“ Objekte ein. Struktur sehr ähnlich.
Unterschied (1P): Im Kompositum geht es darum Bestandshierarchien zu bilden. Die Funktionalität des Kompositums ändert die Funktionalitäten der enthaltenen Objekte nicht. Der Stellvertreter enthält nur ein einziges Objekt und verändert dessen nach außen sichtbare Funktionalität, fügt neue Funktionalität hinzu, oder schränkt die Funktionalität ein.

g.) Nennen Sie zwei Gründe für die Verwendung von Programmier-Richtlinien. (1P)

- Konsistenter Stil erleichtert die Lesbarkeit
- Beschleunigt die Einarbeitung bei Personalwechsel und Wiedereinarbeitung
- Zeitersparnis bei Fehlerfindung, Erweiterung und Pflege des Programms
- „Ein sauberer Stil ist Ehrensache“

(je 0,5P)

h.) **Nur für Informatiker:** Beschreiben Sie einen Implementierungszyklus der „testgetriebenen Entwicklung“. (2P)
**Nur für Informationswirte:** Welche Tätigkeiten empfehlen wir in der Abnahmephase durchzuführen? (2P)

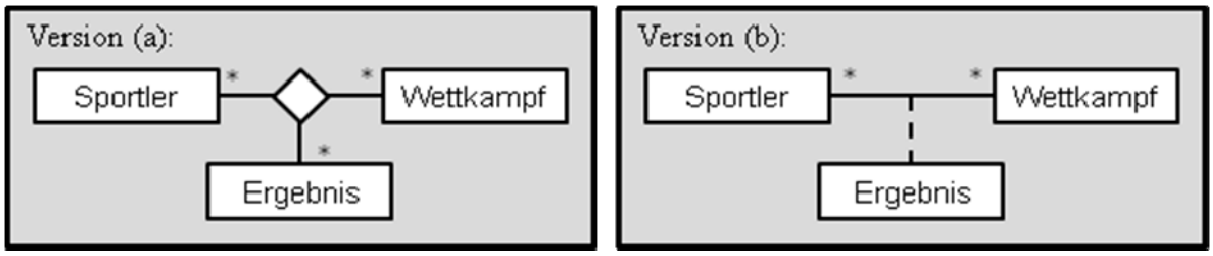
Informatiker (0,5+0,5+1P):
1. Test schreiben, der fehlschlägt. (+ gerade soviel Quelltext, dass übersetzt werden kann)
2. Gerade soviel Quelltext schreiben, dass alle Tests erfolgreich laufen.
3. Eliminiere „üble Gerüche“.

Informationswirte (4x0,5P):
- Übergabe des Gesamtprodukts einschl. der gesamten Dokumentation an den Auftraggeber
- Abnahmetest
- Belastungs- und Stresstests
- Abnahmeprotokoll

i.) Welche Auswirkungen hätte die Berücksichtigung von Ausnahmen (Exceptions) auf den Kontrollflussgraphen? Welchen Einfluss hätte sie folglich auf die Testkriterien Zweig- und Anweisungsüberdeckung? Zur Vereinfachung unterstellen wir, dass die Ausnahmen im zu testenden Code **nicht behandelt** werden. (4P)

Bei jeder Stelle, die zu einer Ausnahme führen könnte, müssen die Grundblöcke nochmals geteilt werden. (1P)
Ausgehend von diesen Stellen müssen zusätzliche Kanten eingefügt werden, die die Funktion verlassen. (1P)
Da keine „neuen“ Grundblöcke hinzugefügt werden (es werden ja nur existierende geteilt), ändert sich für die Erfüllung des Kriteriums der Anweisungsüberdeckung nichts. Es müssen also keine neuen Testfälle erzeugt werden. (1P)
Für die Zweigüberdeckung sind aufgrund der zusätzlichen Kanten zusätzliche Testfälle nötig. (1P)

## Aufgabe 2: Syntax und Semantik von UML (3P)




a.) Welcher gemeinsame Sachverhalt wird in den beiden Diagrammen modelliert? Beschreiben Sie den Sachverhalt in natürlicher Sprache. (1P)

Ein Sportler soll bei einem Wettkampf mitmachen können und dabei ein Ergebnis erhalten. Dieses Ergebnis ist abhängig davon, welcher Sportler an welchem Wettkampf teilgenommen hat.

b.) Diskutieren Sie Vor- und Nachteile der beiden Versionen: Welche ist die bessere? (2P)

Version (a) ermöglicht die Daten zu speichern, kann aber nicht sicherstellen, dass wenn ein Sportler aus dem Datenbestand gelöscht wird, dass dann auch seine Ergebnisse gelöscht werden. Die Modellierung verhindert nicht, dass Ergebnisobjekte übrigbleiben, für die kein Sportler mehr existiert; ein zusätzlicher OCL-Ausdruck wäre nötig. Version (b) kann das, wegen der Semantik der Assoziationsklassen, ohne OCL-Ausdruck sicherstellen.

# Aufgabe 3: Objektorientierte Analyse und Modellieren (17P)

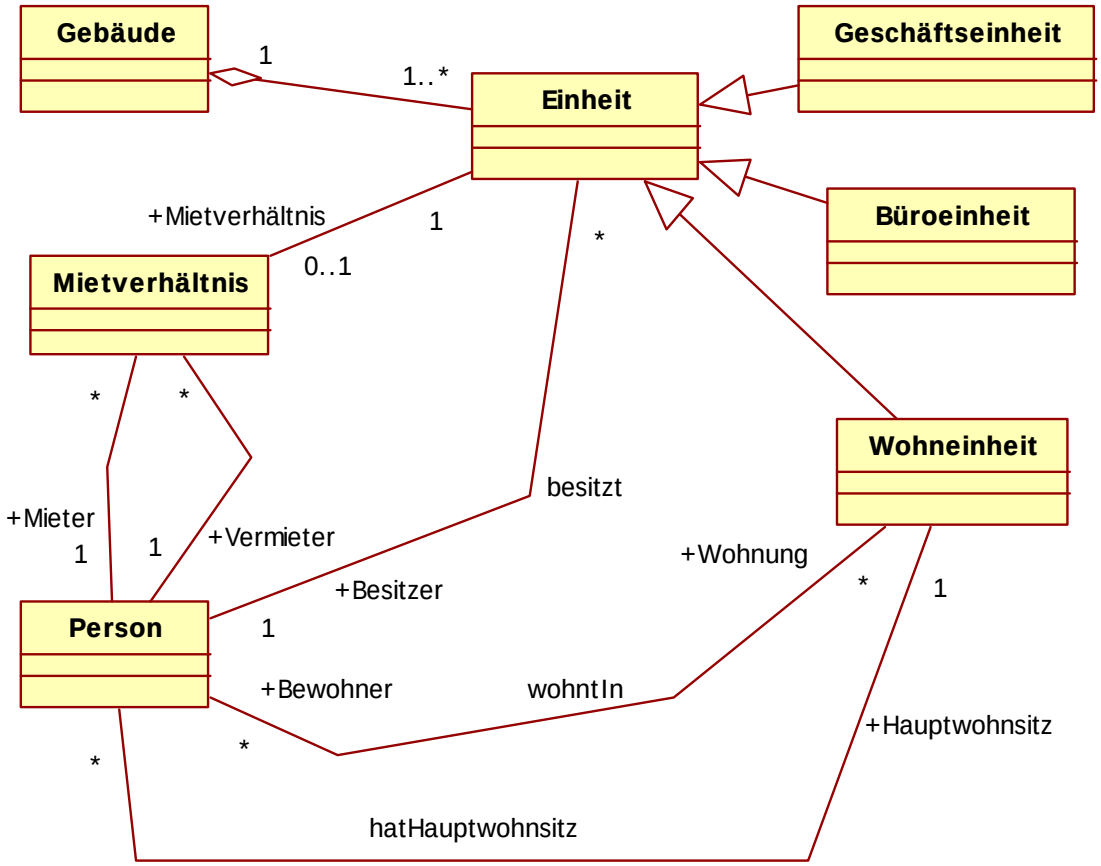
Modellieren Sie folgendes Szenario möglichst genau als UML-Klassendiagramm. Geben Sie, wenn nötig, zusätzlich Zusicherungen in OCL an, wobei Sie aber die „Bordmittel" der UML-Klassendiagramme bevorzugen: Drücken Sie also keine Zusicherung mit OCL aus, die man auch im Klassendiagramm hätte ausdrücken können. Modellieren Sie keine Attribute und Methoden. Geben Sie aber Multiplizitäten, Assoziationsnamen **und Rollen** an!

Tipp: Lesen Sie den Text sehr sorgfältig – die Details sind zum Punkte sammeln! Haken Sie am besten Satzglied für Satzglied ab. Anhaltspunkt: Eine *mögliche* Lösung enthält ungefähr 6-7 Klassen, 6-7 Assoziationen, ein paar Vererbungskanten und 2 OCL-Ausdrücke. Erstellen Sie ggf. geeignete Hilfs- und/oder Oberklassen.

***Szenario:*** *Ein Gebäude besteht aus einer oder mehreren Wohn-, Geschäfts- und/oder Büroeinheiten. Diese Einheiten haben jeweils einen Besitzer, beliebig viele Bewohner (je nachdem!) und ggf. einen Mieter und einen Vermieter. Wenn es einen Mieter gibt, muss es natürlich auch einen Vermieter geben und umgekehrt. Jede Person soll jederzeit ein Miet-, Besitz- oder Bewohn-Verhältnis eingehen oder beenden können. Ein Wohnungseigentümer muss seine eigenen Wohnungen bewohnen können oder fremde Wohnungen mieten können. Jede Person wohnt in mindestens einer Wohneinheit, genau eine dieser Wohneinheiten ist jedoch ihr Hauptwohnsitz. Von den Bewohnern einer Wohnung muss mindestens einer diese Wohnung besitzen oder mieten.*

**Jegliche weiteren Beziehungen und Zusicherungen**, sowie sämtliche Attribute und Methoden sind **nicht zu beachten**, insbesondere (aber nicht ausschließlich) Zusicherungen bezüglich

- Weitervermietung
- selbst-bewohnter Eigentumswohnungen, die nicht vermietet werden können
- Personen, die ihr eigenes Büro mieten




OCL-Ausdrücke:

- context Person inv: self.Wohnung->includes(self.Hauptwohnsitz)
  oder (falls „wohnt**Zusätzlich**In" statt „wohntIn" Beziehung modelliert wurde)
  context Person inv:
  not self.Wohnung->includes(self.Hauptwohnsitz)
- context Wohneinheit inv:
  self.Bewohner->includes(self.Mietverhältnis.Mieter)
  or
  self.Bewohner->includes(self.Besitzer)

## Aufgabe 4: Kontrollflussorientierter Strukturtest (16P)

Gegeben sei die folgende Java-Funktion:

```
01   public double f(double[] array) {
02    double result = Double.NaN;
03    if ((array != null) && (array.length != 0)) {
04     result = array[0];
05     for (int i = 1; i < array.length; i++)
06      if (array[i] > result)
07       result = array[i];
08    }
09    return result;
10   }
```

a.) Erstellen Sie auf der gegenüber liegenden Seite den Kontrollflussgraphen der Funktion **`f()`**. Wenden Sie dabei das aus der Vorlesung bekannte Verfahren an. (4P)

b.) Beschreiben Sie die Funktion von **`f()`**. Betrachten Sie auch die Sonderfälle! (2P)

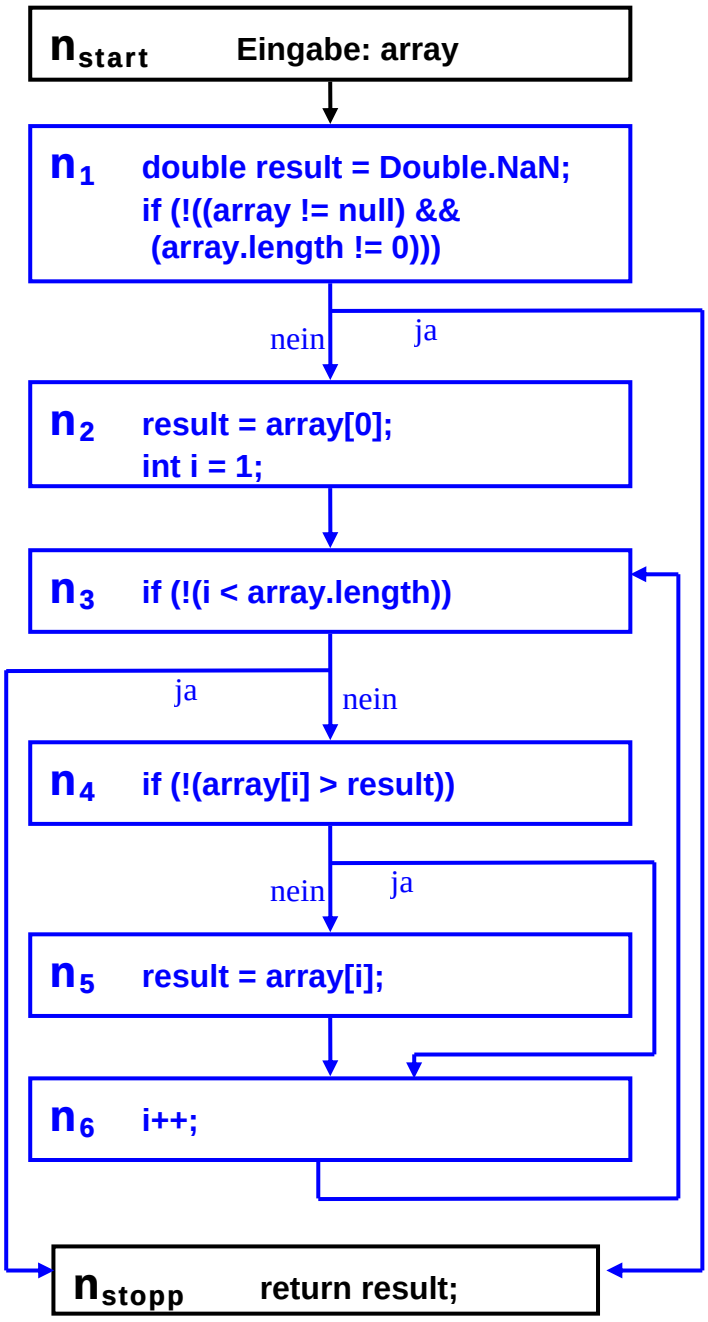
Die Funktion f(double[] array) gibt als Ergebnis den maximalen in der Folge „array" enthaltenen Wert zurück. (1P)
Werden eine null-Referenz oder eine leere Folge übergeben, liefert die Funktion den Wert „Double.NaN" zurück. (1P)

c.) Nennen Sie eine minimale Testfall-Menge für die Anweisungsüberdeckung und geben Sie zu jedem Testfall den durchlaufenen Pfad an. (1P)

{1.0, 2.0} start, 1, 2, 3, 4, 5, 6, 3, stopp
0,5 P 0,5 P

d.) Nennen Sie eine minimale Testfall-Menge für die Zweigüberdeckung des obigen Programms. Geben Sie zu jedem Testfall den durchlaufenen Pfad an. (2P)

null /{} start, 1, stopp 2x0,5 P
{1.0, 0.0, 2.0} start, 1, 2, 3, 4, 6, 3, 4, 5, 6, 3, stopp 2x0,5 P
bei nicht minimaler Testfallmenge: max. 1 P




kleine Fehler (zB. fehlende Bedingung): -0,5 P

e.) Nennen Sie die Teilpfade **der Schleifenquerer** (Boundary-Interior-Test). (1P)

3, 4, 5, 6 und 3, 4, 6 2x0,5 P
Bei vollständigen Pfaden: 0 P

f.) Geben Sie eine Testfallmenge für den **Grenztest** (Boundary-Interior-Test) an. (1P)

z.B. { {1.0, 2.0}, {1.0, 1.0} }. Ein Testfall durchläuft die Schleife genau 1x und deckt genau einen Schleifenquerer ab. Alle Schleifenquerer müssen abgelaufen werden. 2x0,5 P

g.) Geben Sie eine Testfallmenge für den **Interieurtest** (Boundary-Interior-Test) an. (1P)

z.B. { {1.0, 1.0, 2.0}, {1.0, 1.0, 1.0} }. Jeder Testfall durchläuft die Schleife min 2x. Im 2. Schleifendurchlauf müssen sie unterschiedliche Schleifenquerer benutzen. (1P, ganz oder gar nicht)

h.) Betrachten Sie die Zeile 3 des Programms. Geben Sie eine minimale Menge an Testfällen an, welche die minimale-mehrfache Bedingungsüberdeckung erfüllen. (2P)

```
03    if ((array != null) && (array.length != 0)) {
```

Fall1: { }
Fall2: { 1.0 }
Fall3: null
(nicht minimale Menge: max. 1P)

i.) Wie ändert sich die Funktionsweise von **`f()`**, wenn Zeile 3 wie folgt geändert wird? (2P)

```
03    if ((array != null) & (array.length != 0)) {
```

Durch den Wegfall der Kurzauswertung wirft die Funktion, wenn eine null-Referenz als Parameter übergeben wird, eine NullPointerException.
(Unterschied „array==null" erkannt 1P, „Kurzauswertung" 0,5P, „Exception" 0,5P)

## Aufgabe 5: Entwurfsmuster (7P)

In den Java-Bibliotheken lassen sich einige Entwurfsmuster finden. Identifizieren Sie anhand der Zitate aus der Dokumentation, welche das sind. Begründen Sie Ihre Aussage gut, sonst bekommen Sie keine Punkte. Geben Sie insbesondere an, **welche Klasse, Methode, Instanz, usw. welche Rolle** in dem von Ihnen angegebenen Muster einnehmen soll!

a.) Klasse javax.swing.JList (JDK Version 1.6). (2P)

**`public void addListSelectionListener(ListSelectionListener listener)`**

Adds a listener to the list, to be notified each time a change to the selection occurs; the preferred way of listening for selection state changes. JList takes care of listening for selection state changes in the selection model, and notifies the given listener of each change. ListSelectionEvents sent to the listener have a source property set to this list.

Parameters: listener - the ListSelectionListener to add

See Also: getSelectionModel(), getListSelectionListeners()

Entwurfsmuster: Beobachter (ohne Begründung/Rollen 0,5P, sonst 1P)

Rollen: JList-Instanz ist das Subjekt (0,5P)

Listener ist der Beobachter (0,5P)

b.) Klasse java.util.Arrays (JDK Version 1.3). (3P)

**`public static java.util.List asList(Object[] a)`**

Returns a fixed-size list backed by the specified array. (Changes to the returned list "write through" to the array.) This method acts as bridge between array-based and collection-based APIs, in combination with Collection.toArray. The returned list is serializable.

Parameters: a - the array by which the list will be backed.

Returns: a list view of the specified array.

See Also: Collection.toArray()

Entwurfsmuster: Adapter (1P)

Rollen: „a" ist das adaptierte Objekt (1P)

Rückgabewert ist der Adapter (List-Schnittstelle das Ziel) (1P)

c.) Klasse java.io.BufferedInputStream, erweitert FilterInputStream und damit InputStream, beide ebenfalls aus dem Paket java.io. (JDK Version 1.6). (2P)

A BufferedInputStream adds functionality to another input stream-namely, the ability to buffer the input and to support the mark and reset methods. When the BufferedInputStream is created, an internal buffer array is created. As bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time. The mark operation remembers a point in the input stream and the reset operation causes all the bytes read since the most recent mark operation to be reread before new bytes are taken from the contained input stream. […]

**`BufferedInputStream(InputStream in) // Constructor`**

Creates a BufferedInputStream and saves its argument, the input stream in, for later use.

Parameters: in - the underlying input stream.

Entwurfsmuster: Stellvertreter/Dekorierer
(ohne Begründung/Rollen 0,5P, sonst 1P)

Rollen: Die BufferedInputStream-Instanz ist der Stellvertreter (0,5P)

„in" ist das Subjekt (0,5P)