

Name:

Vorname:

Matr.Nr.:

Technische Universität München  
Fakultät für Informatik  
Prof. Dr. Dr. h.c. M. Broy

Sommersemester 2006  
21. Juli 2006

## EINFÜHRUNG IN DIE SOFTWARETECHNIK

**Endklausur****Aufgabe 1: Multiple-Choice-Fragen zu den Entwicklungsphasen (18 Punkte)**

Geben Sie für jede der folgenden Aussagen durch Ankreuzen an, ob Sie der Aussage zustimmen oder nicht.

(Hinweis zur Punktebewertung einer Aussage: kein Kreuz = 0 Punkte, zwei Kreuze = -1 Punkt, ein richtiges Kreuz = +1 Punkt, ein falsches Kreuz = -1 Punkt)

Ja	Nein	Aussage
<input type="checkbox"/>	<input type="checkbox"/>	Transaktionsmonitore (z.B. UTM, CICS) werden überwiegend für Client-Server-Architekturen verwendet.
<input type="checkbox"/>	<input type="checkbox"/>	Eine Systemstudie wird in der Anforderungsanalyse zur Detaillierung eines bereits beschlossenen Systementwicklungsprojektes erstellt.
<input type="checkbox"/>	<input type="checkbox"/>	Das Lastenheft wird häufig zur öffentlichen Ausschreibung eines Systementwicklungsprojektes verwendet.
<input type="checkbox"/>	<input type="checkbox"/>	Unter Domänenanalyse versteht man die Analyse der Daten eines Systems.
<input type="checkbox"/>	<input type="checkbox"/>	Anforderungen an die Performanz (Skalierung) sind Teil der funktionalen Anforderungen eines Systems.
<input type="checkbox"/>	<input type="checkbox"/>	In der Anforderungsdefinition werden Use-Cases verwendet, um beispielhaft Anwendungsfälle zu beschreiben.
<input type="checkbox"/>	<input type="checkbox"/>	Die Schichten der 3-Schichten-Architektur (3-Tier-Architektur) stellen verschiedene Abstraktionsebenen für die Gesamtsicht auf das System dar.
<input type="checkbox"/>	<input type="checkbox"/>	Die Applikationsarchitektur ist eine Abstraktion der Deployment-Architektur.
<input type="checkbox"/>	<input type="checkbox"/>	Design-by-Contract ist eine Technik für vertragliche Festlegungen in Softwareentwicklungsprojekten.
<input type="checkbox"/>	<input type="checkbox"/>	Middleware (z.B. CORBA, RMI) bezeichnet Software, die unter anderem Dienste für Nachrichtenaustausch, Datenzugriff, Prozeduraufrufe, etc. auf Hostsystemen bereit stellt.
<input type="checkbox"/>	<input type="checkbox"/>	Datenflussdiagramme sind insbesondere gut geeignet zur Beschreibung der Architektur von parallelen Systemen.
<input type="checkbox"/>	<input type="checkbox"/>	Transaktionsmonitore dienen zur Ablaufsteuerung paralleler Systeme auf sequentieller Hardware.
<input type="checkbox"/>	<input type="checkbox"/>	Dynamische Architektur Aspekte lassen sich insbesondere gut durch Datenflussdiagramme beschreiben.
<input type="checkbox"/>	<input type="checkbox"/>	Design-Patterns sind vorgefertigte Teillösungen für Systementwürfe für bestimmte Problemklassen.
<input type="checkbox"/>	<input type="checkbox"/>	Frameworks sind vorgefertigte Codeteile, die zusätzlichen Aufwand erfordern, um ablauffähige Codeteile zu erhalten.
<input type="checkbox"/>	<input type="checkbox"/>	Die bei einer Systemintegration auftretenden Integrationsfehler können stets auch schon beim Test der Einzelmodule erkannt werden.
<input type="checkbox"/>	<input type="checkbox"/>	Logische Verifikation ist ein Mittel, um vor Ausführung eines Programms eine Aussage über seine Korrektheit zu erhalten.
<input type="checkbox"/>	<input type="checkbox"/>	Leistungsengpässe lassen sich erst nach Abschluss der Implementierungsphase erkennen.

## Aufgabe 2: Qualitätssicherung (42 Punkte)

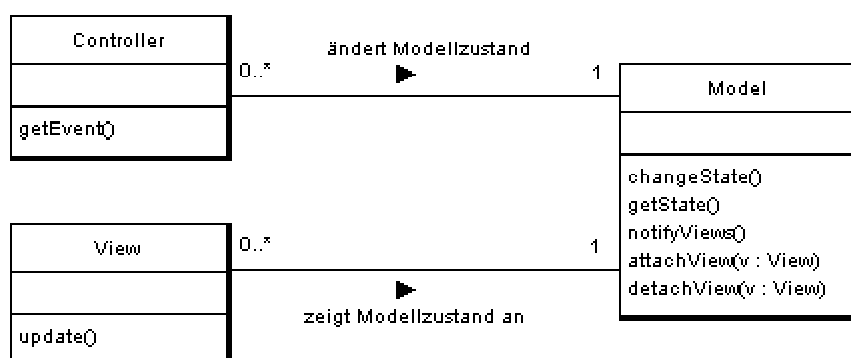
Die vierte Seite dieser Angabe enthält Software-Quellcode, der einen Algorithmus zur binären Suche implementiert. Dieser ist durch Inspektion zu überprüfen. Im Folgenden sind die Regeln der Inspektion angegeben.

RM1	(Dokumentation)	Jede Quellcode-Datei beginnt mit einem Kommentar, der den Klassennamen, Versionsinformationen, Datum und Urheberrechtsangaben enthält.
RM2	(Dokumentation)	Jede Methode wird kommentiert. Der Kommentar enthält eine vollständige Beschreibung der Signatur sowie eine Design-by-Contract-Spezifikation.
RM3	(Dokumentation)	Deklarationen von Variablen werden kommentiert.
RM4	(Dokumentation)	Jede Kontrollstruktur wird kommentiert.
RM5	(Formatierung)	Zwischen einem Schlüsselwort und einer Klammer steht ein Leerzeichen.
RM6	(Formatierung)	Zwischen binären Operatoren und den Operanden stehen Leerzeichen.
RM7	(Programmierung)	Variablen werden in der Anweisung initialisiert, in der sie auch deklariert werden.
RM8	(Bezeichner)	Klassennamen werden groß geschrieben, Variablennamen klein.

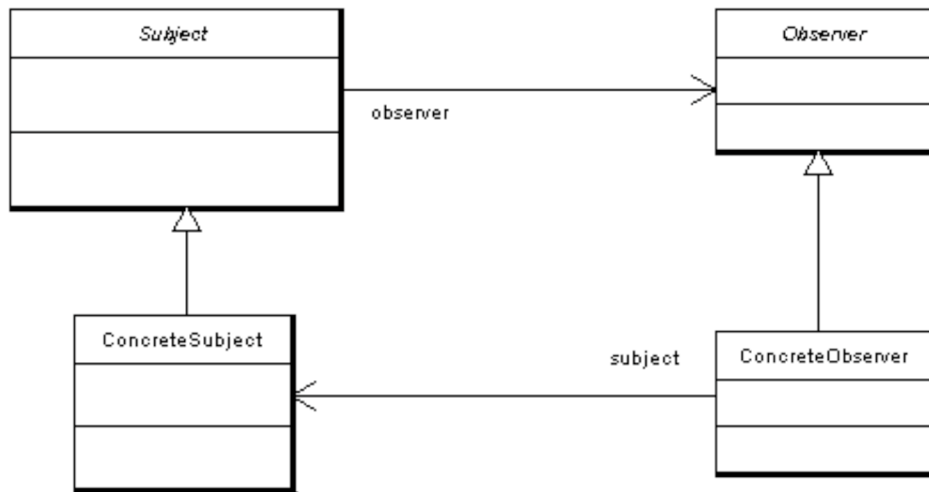
- Überprüfen Sie durch Inspektion, ob die obigen Regeln für den Quellcode eingehalten wurden. Erstellen Sie eine Liste mit allen Verletzungen der Regeln. Geben Sie für jede Verletzung einer Regel die Zeilennummer, Regelnummer und Kommentar an, z.B. (07, RM4, while nicht kommentiert). Schreiben Sie nicht in den Quellcode.
- Entspricht die Methode „binarySearch“ ihrer Spezifikation, die durch Vor- und Nachbedingungen angegeben ist? Geben Sie gegebenenfalls Korrekturen der Methode an.
- Beschreiben alle Kommentare ab Zeile 24 die Semantik des Codes korrekt? Geben Sie zu jedem falschen Kommentar einen korrigierten Kommentar mit Zeilennummer an.
- Geben Sie den Kontrollflussgraphen für die Methode „binarySearch“ an.
- Geben Sie maximal drei Testfälle für die Methode „binarySearch“ an, die insgesamt eine vollständige Anweisungsüberdeckung leisten.

## Aufgabe 3: Entwurfsmuster (33 Punkte)

Model-View-Controller ist eines der wichtigsten Architekturprinzipien zur Erstellung von Systemen mit interaktiven Benutzerschnittstellen. Seine Realisierung erfolgt meist durch Nutzung des Observer-Musters, das bereits in der Vorlesung vorgestellt wurde. Das Model-View-Controller-Prinzip wird durch folgendes Klassendiagramm skizziert:



- Identifizieren Sie das Model-View-Controller-Prinzip im Observer-Muster, indem Sie die Klassen des Observer-Musters den Komponenten Model, View oder Controller zuordnen. Das Observer-Muster wird durch folgendes Klassendiagramm skizziert:



- b) Nennen und erläutern Sie drei Vorteile des Model-View-Controller Prinzips.
- c) Wenden Sie das Observer-Muster auf die Terminplaner-Anwendung an. Dabei sollen zwei Views modelliert werden: ein View auf das Adressbuch und ein weiterer auf den Kalender. Die Klassen Adressbuch und Kalender modellieren die logischen Inhalte dieser Views. Geben Sie die Lösung als Klassendiagramm an.

#### Aufgabe 4: Zustandsübergangsdiagramm (45 Punkte)

Gegeben sei ein Geldautomat, wie er bei Banken üblicherweise zu finden ist. Dieser soll als formales Zustandsübergangsdiagramm mit Ein- und Ausgabe, Vor- und Nachbedingungen modelliert werden. Zu modellieren sind dabei:

- die Authentifizierung mittels PIN-Eingabe,
- die Auszahlung eines gewünschten Geldbetrags,
- ein maximaler Auszahlungsbetrag von 500,- Euro,
- der Einzug der Kundenkarte nach dreimalig falscher Eingabe der PIN.
- Abbruch in jedem Zustand möglich

Folgende Hilfsfunktionen können als gegeben betrachtet werden:

- `check_PIN(pin : Int) : Bool` prüft ob es sich um die korrekte PIN handelt.
- `gibGeldscheine(betrag: Int) : void` gibt die Geldscheine an den Benutzer aus.

Nicht berücksichtigt werden müssen weitere Menüoptionen wie Kontostandsabfrage und Überweisungen sowie die Stückelung des Geldes in Scheine und Münzen. Verwenden Sie die in der Vorlesung gebrauchte Syntax.

- a) Geben Sie die Ein- und Ausgaben des Geldautomaten an.
- b) Geben Sie alle Zustandsattribute an, die für die Modellierung des Geldautomaten notwendig sind und beschreiben Sie deren Verwendungszweck (z.B. „kontostand: Integer repräsentiert den Kontostand“).
- c) Identifizieren Sie anhand der Zustandsattribute die Zustände des Geldautomaten und geben Sie eine Charakterisierung der Zustände durch Angabe der möglichen Wertebereiche der Zustandsattribute an. Welcher der Zustände ist Anfangszustand?
- d) Zeichnen Sie das Zustandsübergangsdiagramm. Verwenden Sie hierzu die Syntax mit Ein- und Ausgabe, Vor- und Nachbedingungen. Es dürfen nur die in Ihrer Lösung zu den Teilaufgaben a), b) und c) bereits deklarierten Elemente des Geldautomaten verwendet werden.

## Dieses Blatt bitte nicht beschriften!

```
00 /*
01  * BinarySearch.java
02  *
03  * Eine Implementierung der "Binaere Suche"
04  * mit einem iterativen Algorithmus
05  */
06 class BinarySearch {
07
08  /**
09   * Binaere Suche
10   * a:      Eingabefeld
11   * item:    zu suchendes Element
12   * returnValue: der Index des zu suchenden Elements oder -1
13   *
14   * Vorbedingung:
15   * a.length > 0
16   * a ist ein linear geordnetes Feld:
17   *  $\forall k: (1 \leq k < a.length) \implies (a[k-1] \leq a[k])$ 
18   *
19   * Nachbedingung:
20   * Wenn item in a, dann gibt es ein k mit a[k] == item und returnValue == k
21   * Genau dann wenn returnValue == -1 gibt es kein k mit  $0 \leq k < a.length$ 
22   * und a[k] == item.
23   */
24  public static int binarySearch(float a[], float item) {
25
26      int End;          // exklusiver Index für das Ende des
27                        // zu durchsuchenden Teils des Arrays
28
29      int start = 1; // inklusiver Index fuer den Anfang der Suche
30      End = a.length;
31
32      // Die Schleife wird verlassen, wenn keine der beiden Haelften das
33      // Element enthaelt.
34      while(start < End) {
35
36          // Teilung des Arrays in zwei Haelften
37          // untere Haelfte: [0, mid[
38          // obere Haelfte:  ]mid, End[
39          int mid = (start + End) / 2;
40
41          if (item > a[mid]) {
42              // Ausschluss der oberen Haelfte
43              start = mid + 1;
44          } else if (item < a[mid]) {
45              // Ausschluss der unteren Haelfte
46              End = mid-1;
47          } else {
48              // Das gesuchte Element wird zurueckgegeben
49              return (mid);
50          }
51      } // end of while
52
53      // Bei Misserfolg der Suche wir -1 zurueckgegeben
54      return (-1);
55  }
56 }
```