Prof. Dr. Franzen
FB MNI
11. Juli 2005

Klausur zu Programmieren 3 (Java), SS 2005

Füllen Sie bitte zunächst dieses Deckblatt vollständig aus und unterschreiben es.

Name, Vorname:	
Matrikelnummer:	
Studiengang (I, BI, MI), Semester:	
Prüfungsordnung:	
Bei PO 2004, Anzahl Fehlversuche:	

Ich bestätige die Richtigkeit der gemachten Angaben.

Unterschrift:

Bearbeiten Sie die Aufgaben auf den folgenden Blättern. Verwenden Sie bei Bedarf auch die Rückseiten. Jede Aufgabe sollte allerdings nur auf dem Blatt bearbeitet werden, auf dem sich die Aufgabenstellung befindet. Für Aufgabe 6 verwenden Sie zusätzlich das letzte, leere Blatt!

Bearbeitungsdauer: 110 Min.

Erlaubte Hilfsmittel: Sämtliche schriftlichen Unterlagen.

Viel Erfolg!

Aufgabe	Max. Punkte	Erreichte Punkte
1	16	
2	16	
3	18	
4	14	
5	10	
6	56	
Summe	**120**	
Bonus HÜ		

Aufgabe 1 (Programmstruktur)

(a) Erläutern Sie das Konzept des Java Bytecodes: Wie entsteht er, wie wird er ausgeführt und wieso werden Java-Programme durch ihn plattformunabhängig?

(3)

(b) In der Datei C.java haben Sie mittels

```
package p.q;
class C {...}
```

eine Klasse definiert. Jetzt wollen Sie der Klasse den neuen Namen `D` geben. Außerdem soll sie nach der Umbenennung zum Paket `r.s` gehören. Was müssen Sie tun? In welchem Verzeichnis muss die Datei stehen?

(3)

(c) Erläutern Sie genau, was der Ausdruck

```
new ActionListener(void actionPerformed(ActionEvent ae){...});
```

bewirkt!

(4)

(d) Dürfen zwei Attribute einer Klasse den gleichen Namen besitzen? Dürfen zwei Methoden einer Klasse den gleichen Namen besitzen? Begründen Sie Ihre Antwort!

(3)

(e) Das Datenfeld `x` sei innerhalb der Klasse `C` im Paket `p` ohne eines der Sichtbarkeitsattribute (**`public`**, **`private`** oder **`protected`**) definiert:

```
package p;
class C{int x;}
```

Erläutern Sie, an welchen Stellen eines Programmes der Name `x` sichtbar ist und wo nicht!

(3)

Aufgabe 2 (Initialisierung, Typen, Speicherverwaltung)

(a) Zählen Sie alle Arten der Initialisierung von Datenfeldern einer Klasse auf, in der Reihenfolge ihrer Ausführung!

(4)

(b)Ist es möglich, die genaue Ausführungsdauer einer Anweisung anzugeben? Begründen Sie!

(3)

(c)Ist für ein Array, wie z.B. mit `int[] ia = new int [8];` eingeführt, der Ausdruck `ia.toString();` zulässig? Begründen Sie!

(3)

(d)Unter welchen Umständen wird die Methode `finalize()` der Klasse `Object` ausgeführt? Ist garantiert, dass diese Methode für *jedes* Objekt irgendwann einmal aufgerufen wird?

(4)

(e)Wie kann man in einem Java-Programm den nicht mehr benötigten Speicherplatz eines Objektes explizit wieder freigeben?

(2)

Aufgabe 3 (Nebenläufigkeit)

(a) Was versteht man unter einem kritischen Abschnitt?

(3)

(b) Wie läßt sich in Java wechselseitiger Ausschluss garantiereren? Welche Objekte können als Sperre dienen?

(4)

(c) In der API-Dokumentation werden die Methoden `suspend()`, `resume()` und `stop()` der Klasse `Thread` als *deprecated* gekennzeichnet und werden möglicherweise in zukünftigen Versionen der Klassenbibliothek nicht mehr vorhanden sein. Davon abgesehen, warum wird dringend davon abgeraten, diese Methoden zu verwenden?

(3)

(d) Auf ein Objekt der Klasse `java.lang.String` sollen mehrere Threads zugreifen können. Was ist zu tun? Begründen Sie! Gilt Ihre Antwort auch für ein `StringBuffer`-Objekt?

(4)

(e) Innerhalb eines Methodenrumpfes wird die Methode `wait()` aufgerufen. Bei Ausführung des Programmes wird an dieser Stelle aber eine `IllegalMonitorStateException` ausgelöst. Woran mag das liegen?

(4)

Aufgabe 4 (Vererbung)

(a) Die Methode `A.meth(X,Y)` wird in der Klasse `B` überschrieben (s.u.). In welchem Zusammenhang müssen die Klassen `AException` und `BException` stehen? Begründen Sie!

```
class A {...
 void meth(X x, Y y)throws AException{..}
}
class B extends A{...
 void meth(X x, Y y)throws BException{...}
}
```

(3)

(b) Eine Methode `B.meth(...)` werde von einer Methode `A.meth(...)` überschrieben. Wie verhält es sich mit den Vor- und Nachbedingungen der beiden Methoden, dürfen sie abgeschwächt oder verschärft werden? Begründen Sie Ihre Antwort mit dem Ersetzungsprinzip!

(4)

(c) Es sei `B` eine Unterklasse von `A`. Wie wird gewährleistet, dass bei Erzeugung eines `B`-Objektes auch ein Konstruktor von `A` aufgerufen wird. Wie lautet die Regel, wenn `B` überhaupt keinen Konstruktor definiert?

(4)

(d) Warum werden Konstruktoren nicht an abgeleitete Klassen vererbt?

(3)

Aufgabe 5 (Schnittstellen)

(a) Im Paket `java.lang` ist die Schnittstelle `Runnable` deklariert. Kann man mit

```
Runnable r = new Runnable();
```

ein `Runnable` Objekt erzeugen? Begründen Sie!

(2)

(b) Können in einer Schnittstellendefinition Attribute eines Objektes definiert werden?

(1)

(c) Erläutern Sie den Unterschied zwischen einer Schnittstelle und einer abstrakten Klasse!

(2)

(d) Die Klasse `Thread` besitzt u.a. einen Konstruktor `Thread(Runnable)`, dessen Parameter den Schnittstellentyp `Runnable` besitzen muss. Was bewirkt dieser Konstruktor? An welchen Stellen kann bei Verwendung dieses Konstruktors die Methode `run()` implementiert werden? Vergleichen Sie mit der Schnittstelle `ActionListener` und ihrer Verwendung bei der Registrierung an der Ereignisquelle mittels `addActionListener(..)`.

(5)

Aufgabe 6 (Implementierung eines Beispiels)

Gegegeben sei die folgende Schnittstelle für einen Stapel (Stack) beschränkter Größe:

```
interface Stapel1000{
// Maximale Größe des Stapels
public static final int MAXSIZE = 1000;

//legt elem auf dem Stapel ab
public void push(Object elem)throws StapelVollException;

//liefert das oberste Element des Stapels und entfernt es
public Object pop() throws StapelLeerException;

//versetzt den Stapel in den Anfangszustand 'leer'
public void reset();

//liefert das oberste Element des Stapels, ohne es zu entfernen
public Object peek();

//liefert die momentane Größe des Stapels
public int getSize();
}
```

(a) Definieren Sie eine Klasse `Stapel`, die diese Schnittstelle implementiert und zwei Konstruktoren besitzt:
1) einen parameterlosen Konstruktor, der den Stapel in den Zustand „leer" versetzt, und
2) einen Konstruktor `Stapel(Object[] inhalt)`, der den Stapel mit den Elementen von `inhalt` füllt und möglicherweise eine Ausnahme wirft.
Die Elemente des Stapels sollen intern in einem Feld (Array) abgelegt werden. Auf dieses Feld soll außerhalb dieser Klasse nicht zugegriffen werden können. Die Klassen `StapelVollException` und `StapelLeerException` werden als bereits definiert angesehen.
Warnung: Eine Implementierung als verkettete Liste oder mit Hilfe von `java.util.Vector` wird nicht akzeptiert!

(30)

(b) Formulieren Sie die Klasseninvariante der Implementierung in (a) sowie die Vor- und Nachbedingungen aller Methoden.

(7)

(c) Schreiben Sie eine Methode `test()`, das die Zeichenketten `"unten"`, `"mitte"`, `"oben"` ablegt und danach wieder entfernt und ausgibt. Geben Sie die ausgegebenen Zeichenketten in der richtigen Reihenfolge an.
Weiterhin soll die Methode prüfen, ob die Ausnahmen `StapelVollException` und `StapelLeerException` tatsächlich auftreten!

(12)

(d) Wie können Sie Ihre Implementierung "thread-sicher" machen, d.h. den nebenläufigen Zugriff auf einen Stapel durch mehrere Threads so einschränken, dass keine Wettbewerbssituationen (race conditions) eintreten können?

(7)