

Test zu Grundlagen der Programmierung

Leitung: Susanne Guth/Michael Hahsler

2. Mai 2003

Name	
Matrikelnummer	
Unterschrift	

Bitte kreuzen Sie das Studium an, für das Sie diese Prüfung ablegen:

- Bakkalaureat Wirtschaftsinformatik
- SBWL Wirtschaftsinformatik
- SBWL Informationswirtschaft

Diese Angabe umfasst inklusive Deckblatt 11 Seiten.

- Bitte lesen Sie die Angaben aufmerksam und sorgfältig durch.
- Beachten Sie bei Ihrer Beantwortung, dass nur **einwandfrei lesbare und eindeutige Antworten** bewertet werden können.
- Verwenden Sie dazu einen nicht radierbaren Kugelschreiber.
- Für diese Prüfung sind **keinerlei Unterlagen** erlaubt!
- Bitte beantworten Sie die Fragen in den dafür vorgesehen Bereichen. Alle anderen Notizen und dergleichen können bei der Beurteilung nicht berücksichtigt werden.
- und nehmen Sie das Fragenheft nicht auseinander!

Für die Bearbeitung der Fragen haben Sie **60 Minuten** Zeit. Insgesamt können Sie bei dieser Prüfung **60 Punkte** erreichen, kalkulieren Sie also pro Punkt eine Minute Bearbeitungszeit.

Viel Erfolg!

Punkte	
Note	

1) Grundkonzepte von Objektorientierung

- a) **Erklären Sie schlagwortartig das Konzept der Datenkapselung ("Data Encapsulation", "Data Hiding"). Mit welchen Schlüsselworten und Methoden wird Datenkapselung in Java erreicht?**

Punkte
3

Verstecken der Implementierung einer Klasse hinter den Methoden (Interface), die auf ihre internen Daten ausgeführt werden.

1 Punkt

private für versteckte Daten und Methoden
public get und set-Methoden für den Zugriff (Interface),

2 Punkte

- b) **Beschreiben Sie schlagwortartig den Prozess der Überleitung von realen Objekten zu Software-Objekten. Wie könnte ein Software-Objekt für das reale Objekt "Prüfungsangabe" aussehen (kein Code, muss nicht vollständig sein)?**

Punkte
3

Bei der Überleitung werden die Zustände und das Verhalten von realen Objekten in Variablen und Methoden der Software-Objekte (Klassen) abgebildet.

1 Punkt

Beispiel:
UML oder nur Aufzählung der Variablen und Methoden

2 Punkte

2) Datentypen und Variablen

- a) **Welche Kategorien von primitiven Datentypen unterstützt Java?
Geben Sie jeweils das Schlüsselwort des Datentyps und 2 mögliche Werte an.**

Punkte
3

Ganzzahlen: byte, short, int, long ... 3, 6
Gleitkommazahlen: float, double ... 3.14, 20.0
Zeichen: char ... 'a', '\n'
Wahrheitswerte: boolean ... true, false

3 Pkt (je Gruppe 1 Pkt, es reichen also 3 der 4 Gruppen)

- b) **Welche der folgenden Aussagen sind richtig? Kreuzen Sie die richtige(n) Aussage(n) an.**

Punkte
3

- Java kann implizit den Datentyp eines `int`-Werts auf `double` erweitern.
- Java kann implizit den Datentyp eines `double`-Werts auf `int` erweitern.
- In Java kann explizit der Datentyp eines `double`-Werts in `int` umgewandelt werden.
- Der Wert `5.0` ist ganzzahlig und daher vom Datentyp `int`.
- Der Wert `3.14` ist eine Kommazahl und daher vom Datentyp `double` oder `float`.
- Variablen müssen bei der Deklaration auch Initialisiert werden damit sie verwendet werden können.
- Explizite Datenumwandlung wird in Java nicht unterstützt.
- Konstanten werden in Java mit dem Schlüsselwort `const` deklariert.
- Variablen können nur Werte vom deklarierten Datentyp speichern.

Pro falscher Antwort 0.5 Punkte Abzug!

3) Operatoren

a) Was bewirken die Operatoren `!`, `&&` und `||`? Mit welchen Datentypen werden Sie verwendet? Geben Sie ein Beispiel, das alle 3 Operatoren verwendet (Code).

Punkte
3

! -> log. nicht
 && -> log. Und
 || -> log. Oder
 Arbeiten mit Wahrheitswerten
(2 Punkte)

Bsp.:
 int a = 7;
 boolean c = false;
 boolean d = (a > 5) && (!c || a < 3);
 // d <- (true) && (true || false) = true

Für Beispiel 1 Pkt

b) Was ergeben folgende Ausdrücke (Wert und Datentyp)?

Punkte
3

```
int i=10;
int j=3;
double d=3.0;
boolean b = true;
```

Ausdruck	Wert	Datentyp
<code>i%j</code>	1	int
<code>i/5.0</code>	2.0	double
<code>d>3.0</code>	false	boolean
<code>d!=j</code>	false	boolean
<code>i=j</code>	3	int
<code>(int) (d*i)</code>	30	int

2 Richtige / Pkt

4) Ablaufsteuerung

a) *Was gibt folgender Code aus?*

Punkte
2

```
int a=2;
while (100>=a) {
    System.out.println(a);
    a *= 2;
}
```

Ausgabe:

2

4

8

16

32

64

Ende da $100 \geq 128$ falsch ergibt (2 Punkte)

b) *Ersetzen Sie die While-Schleife aus Beispiel aus Frage 4a durch eine For-Schleife und geben Sie den Code dazu an.*

Punkte
3

```
for (int a=2; 100>=a; a*=2) {
    System.out.println(a);
}
```

3 Punkte

c) *Schreiben Sie eine If-else-Anweisung die überprüft ob der Wert der int-Variable i größer als 99 ist. Ist der Wert größer soll "i ist größer als 99" und sonst "i ist kleiner oder gleich 99" ausgegeben werden.*

Punkte
1

```
if (i>99) {
    System.out.println("i ist größer als 99");
} else {
    System.out.println("i ist kleiner gleich 99");
}
```

1 Punkte

5) Methoden

- a) Schreiben Sie eine Methode mit dem Namen `nFactorielle`, die die mathematische Funktion $n! = \prod_{i=1}^n i = 1 \times 2 \times 3 \times \dots \times n$ ausrechnet (die Multiplikation aller ganzen Zahlen von 1 bis n). Die Methode soll folgendermaßen aufgerufen werden können:

Punkte
4

```
System.out.println(a+"! = "+ nFactorielle(11));
```

```
int nFactorielle(int n) {  
    int fact=1;  
    for (int i=1;i<=n; i++) {  
        fact *= i;    // oder fact = fact * i;  
    }  
    return fact;  
}
```

2 Pkt ... Signatur

2 Pkt ... Schleife

- b) In der API Dokumentation von Java finden Sie unter `java.lang` die Klasse `Math`, die unter anderem folgende Methode deklariert hat:

Punkte
2

```
static double exp(double a)
```

Returns Euler's number e raised to the power of a `double` value.

Schreiben Sie den Code, der diese Methode verwendet um

$e^{1.2}$ auszurechnen.

```
double result = Math.exp(1.2);
```

oder

```
double result = java.lang.Math.exp(1.2);
```

2 Punkte

6) Methoden II

- a) **Was ist der Unterschied zwischen "Überladen" (Method-Overloading) und "Überschreiben" von Methoden? Geben Sie je ein Beispiel in Java Code (der Code in den Methoden kann durch ... angedeutet werden).**

Punkte
4

Überladen: Methode mit **gleichem Namen aber anderen Parametern**. Es gibt danach 2 Methoden mit gleichem Namen und der Compiler wählt nach den Parametern beim Aufruf die passende M. aus.

Bsp:

```
class Beispiel {  
int max (int a, int b) {...}  
int max (int a, int b, int c) {...}  
}
```

Überschreiben: Bei der **Vererbung** kann die Subklasse eine Methode der Superklasse ersetzen. Die Methode muss die **gleiche Signatur** haben. Die Einstellung der Zugriffskontrolle darf zwar erweitert aber nicht eingeschränkt werden.

Bsp.:

```
class Father {  
protected void play(Toy aToy) {...}  
}  
  
class Child extends Father {  
public void play(Toy aToy) {...}  
}
```

je 2 Punkte

- b) **Welche der folgenden Aussagen sind richtig? Kreuzen Sie die richtige(n) Aussage(n) an.**

Punkte
2

- Arrays werden bei der Übergabe als Parameter als Referenz übergeben.
- Lokal in einer Methode deklarierte Variablen können nicht als Rückgabewert benutzt werden.
- Instanzen von selbst implementierten Klassen können als Parameter und auch als Rückgabewert von Methoden verwendet werden.
- Methoden können in Java maximal einen Rückgabewert aber eine beliebige Anzahl an Parametern haben.
- Methoden mit dem Schlüsselwort `static` haben keinen Rückgabewert.
- Primitive Datentypen werden bei der Übergabe als Parameter kopiert.

Pro falscher Antwort 0.5 Punkte Abzug!

7) Klassen

a) **Was ist eine Klassendefinition und was verstehen Sie unter einem Konstruktor? Antworten Sie jeweils mit einem Satz!**

Punkte
2

Die Klassendefinition ist der Bauplan für Instanzen und beschreibt Eigenschaften und Verhalten.

Der Konstruktor ist eine spezielle Methode, die bei der Instanzierung aufgerufen wird.

2 Pkt

b) **Was versteht man unter Zugriffskontrolle bei Methoden und Variablen in der Klassendefinition? Welche Einstellungen gibt es in Java und was bedeuten sie?**

Punkte
4

Legt den Gültigkeitsbereich der Variablen und auch der Methoden fest, d.h. festlegen, ob bzw. welche anderen Objekte auf eine Variable oder Methode der programmierten Klasse zugreifen können.

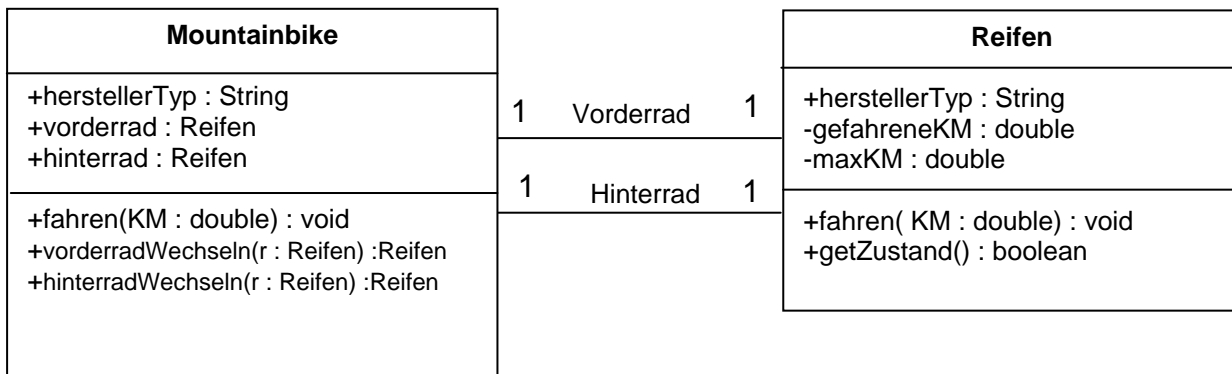
1 Pkt

Einstellungen:

- `private`: Methoden oder Variablen können nur innerhalb der Klasse, in der sie geschrieben sind, verwendet oder aufgerufen werden.
- `protected`: Wie privat. Hat nur im Zusammenhang mit Vererbung eine zusätzliche Bedeutung. *Private* Methoden oder Variablen können von abgeleiteten Klassen nicht verwendet werden, *Protected* schon.
- `public`: Methoden oder Variablen lassen sich generell von allen anderen Klassen und natürlich auch von Ableitungen aufrufen.
- `package` (Standardeinstellung) : Methoden oder Variablen können innerhalb der Klasse, in der sie geschrieben sind, sowie innerhalb des Packages verwendet oder aufgerufen werden.

3 Punkte (1 Punkt pro Einstellung mit richtiger Erklärung)

8) Klassendefinitionen



Punkte
5

- a) **Implementieren Sie die Klasse Reifen** mit Konstruktor der die 3 Variablen initialisiert. Die Instanzvariable *herstellerTyp* gibt den Reifentyp an z.B. "IRC Mythos SXC 1.3 Zoll". Jeder Reifen kann, abhängig vom Reifentyp, nur eine bestimmte Anzahl an Kilometer (*maxKM*) verwendet werden, danach muss er ersetzt werden. *gefahrenKM* speichert die Kilometer, die mit dem Reifen schon gefahren wurden. Wenn der Reifen verwendet wird, wird die Methode *fahren* aufgerufen. Die Methode *getZustand* gibt zurück ob der Reifen noch weiter verwendet werden kann oder ersetzt werden muss.

```
public class Reifen {
    public String herstellerTyp;
    private double gefahrenKM;
    private double maxKM;

    Reifen(String hT, double max) {
        HerstellerTyp=hT; maxKM=max; gefahrenKM=0.0;
    }

    public void fahren(double KM) {
        gefahrenKM += KM;
    }

    boolean getZustand() {
        return (maxKM>gefahrenKM);
    }
}
```

Definition und Variablen 2 Pkt

Konstruktor 1 Pkt

Methoden 2Pkt

b) Die Klasse Mountainbike ist folgendermaßen implementiert (Ausgelassener Code ist mit ... dargestellt). Ergänzen Sie die fehlende Methode fahren(double KM). Diese Methode wird aufgerufen, wenn mit dem Mountainbike gefahren wird (vergessen Sie dabei die Reifen nicht).

Punkte
2

```
public class Mountainbike {
    public String herstellerTyp;
    public Reifen vorderrad; Reifen hinterrad;

    Mountainbike(String hat, Reifen vR, Reifen hR) {
        herstellerTyp = hT; vorderrad = vR; hinterrad = hR;
    }
    public Reifen vorderradWechseln(Reifen r) {
        Reifen tmp = vorderrad;
        vorderrad = r;
        return tmp
    }
    public Reifen hinterradWechseln(Reifen r) {...}
}
```

```
public fahren(double KM) {
    vorderrad.fahren(KM); hinterrad.fahren(KM);
}
```

```
}
```

c) Schreiben Sie die Klasse Rennen, die in der main-Methode eine Instanz von Mountainbike (Typ: Trek 8000 mit 2 Reifen des Typs iRC Mythos 2.1 Zoll mit denen man 2000km fahren kann) erzeugt, damit 1000km fährt und dann den Hinterreifen durch einen neuen Hinterreifen (Typ: Richey Omega 1.8 Zoll mit dem man max. 1500km fahren kann) ersetzt.

Punkte
5

```
class Rennen {
    public static void main(String[] args) {
        Mountainbike myBike = new Mountainbike("Trek 8000",
            new Reifen("iRC Mythos 2.1 Zoll")),
            new Reifen("iRC Mythos 2.1 Zoll"));

        myBike.fahren(1000);

        Reifen alterReifen= myBike.hinterradWechseln(
            new Reifen("Richey Omega 1.8 Zoll"))
    }
}
```

Klassendef. + Main 1 Pkt
Instanz von Mountainbike 2 Pkt
Fahren 1 Pkt
Reifen wechseln: 1 Pkt

9) Wiederverwendung

a) **Welche der folgenden Aussagen sind richtig? Kreuzen Sie die richtige(n) Aussage(n) an.**

Punkte
3

- Klassen aus anderen Paketen müssen in der Regel vor der Verwendung importiert werden.
- Abstrakte Klassen dürfen nur abstrakte Methoden enthalten.
- Abstrakte Methoden dürfen nur in abstrakten Klassen enthalten sein.
- Das Konstrukt `Interfaces` sorgt in Java für einheitliche Schnittstellen.
- `Final` deklarierte Klassen können auch ohne Instanzierung verwendet werden.
- Um Klassen aus der Java API verwenden zu können, müssen deren Interfaces implementiert werden.
- Mit dem Schlüsselwort `super` lassen sich auch überschriebene Methoden der Elternklasse aufrufen.
- Bei Vererbung werden die Konstruktoren in der Klassenhierarchie von unten (speziellere Klasse) nach oben (allgemeinere Klasse) aufgerufen.
- Methoden, die in Interfaces deklariert werden stellen sicher, dass die implementierende Klasse zumindest diese Methoden implementiert.

Pro falscher Antwort 0.5 Punkte Abzug!

b) **Was bedeuten im Zusammenhang mit Vererbung die Begriffe "Upcasting" und "dynamic Binding"? Beschreiben Sie je ein Beispiel.**

Punkte
3

Upcasting: erlaubt die Verwendung eines Objekts einer Subklasse als Objekt seiner Superklasse. Instanzen vom Datentyp Subklasse können dabei einer Variablen vom Datentyp Superklasse zugewiesen werden.

Dynamic Binding: Welche Methode bei einem Objekt aufgerufen wird, kann erst zur Laufzeit (run-time) bestimmt werden, weil erst dann die aktuelle Objektklasse bekannt ist.

Je 1 Pkt

1 Pkt für die Beispiele