

Vorlesung Einführung in die Programmierung Fragen I

- Die folgenden Fragen sollen helfen, Wissenslücken aufzudecken und zu schließen.
- Es können jeweils 0-4 Antworten richtig sein.
- Musterlösungen werden nicht veröffentlicht (Jeder soll selbst darüber nachdenken).
- Sie sollten die Fragen nicht nur einfach mit ja oder nein beantworten können, sondern auch das Hintergrundwissen dazu haben.

ASCII (American Standard Code for Information Interchange)

- A) kann deutsche Umlaute darstellen; Falsch
- B) benötigt 7 Bits zur Darstellung von Zeichen;
- C) benötigt 8 Bits zur Darstellung von Zeichen;
- D) ist der Standard für die Zeichendarstellung in Java. Falsch, Unicode ist Standard bei Java

Unicode

- A) dient zur Darstellung von Integerzahlen; Richtig
- B) dient zur Darstellung von Buchstaben; Richtig
- C) dient zur Darstellung von Zeichenketten; Richtig
- D) dient zur verschlüsselten Übertragung von Daten. Falsch

Eine Informationseinheit in Unicode benötigt

A) 1 Bit Falsch

B) 8 Bit Richtig

C) 16 Bit Richtig

D) mehr als 16 Bit. Richtig

Standardisierte Kodierung aller Zeichen in einem System mit ≥ 16 Bits.
Kompatibel mit ASCII.

UTF-8

- A) ist der Fußballverein des Java-Entwicklerteams; Falsch
- B) hat mit Unicode zu tun (was?); Richtig, unterstützt UTF-8 Zeichenketten bis zu einer Länge von vier Byte, auf die sich – wie bei allen UTF-Formaten – alle Unicode-Zeichen abbilden lassen.
- C) hat mit ASCII zu tun (was?); Richtig, die ersten 128 Zeichen (Indizes 0–127) von UTF-8 sind mit ASCII deckungsgleich
- D) ist eine kompakte Darstellung von Unicode. Richtig, UTF-8 ist eine redundanzarme Version von Unicode

Zu den primitiven Datentypen in Java gehören

- A) byte Richtig
- B) int Richtig
- C) Unicode Falsch
- D) Arrays. Falsch

Eine Klassendefinition wie z.B.

```
public class Test { ... }
```

A) kann in einem File stehen, der in einem beliebigen Namen hat; Falsch

B) muss der File Test.java heißen; Richtig

C) muss der File Test.class heißen; Falsch, übersetztes Programm

D) muss auch eine main-Methode haben. Richtig, bzw. könnte sie auch wo anders implementiert werden

Der Java-Compiler

- A) compiliert eine Klasse in den Maschinencode des Rechners, wo er gestartet wird; Falsch
- B) compiliert eine Klasse in ein Java-spezifisches Maschinenformat; Richtig
- C) führt ein Java Programm direkt aus; Falsch
- D) verbessert Programmierfehler selbständig. Falsch

A) Jede Klasse muss eine main-Methode haben.

Falsch, es muss aber mindestens eine Klasse mit der "main"-Methode geben, damit die JVM weiß wo sie suchen muss z. B. Aufruf: `java >>Klassenname<<`

B) Die Java Virtual Machine führt die main-Methode aus. Richtig

C) Die main-Methode kann beliebige Argumente haben.

Falsch, das Array `String[] args` enthält die Parameter, die beim Aufruf des Programms mit java eingegeben wurden

D) Die main-Methode kann static oder nicht static sein.

Falsch, sie muss static sein

In Deklarationen wie

`int i = 3;`

- A) muss die Variable immer i heißen, wenn sie ein int-Wert haben soll; Falsch
- B) muss die Variable immer mit i anfangen, wenn sie einen int-Wert haben soll; Falsch
- C) spielt der Name der Variablen keine Rolle, sollte aber so gewählt werden, dass man sieht, wozu sie gut ist; Richtig
- D) ist der Name der Variablen prinzipiell egal. Richtig

Eine Java-Anweisung

`i = i+1;`

- A) ist ein mathematischer Widerspruch, und daher unzulässig in Java; Falsch
- B) erhöht den Wert von i um 1; Richtig
- C) wird vom Compiler als Syntaxfehler gemeldet; Falsch
- D) führt u.U. zu einem Laufzeitfehler. Falsch

Ein Java-Programmstück

```
{int i = 3;  
  {int i = 4;}}
```

- A) ist in Java unzulässig; Richtig, Überschattung - „Überdefinition“ (in andere Programmiersprachen erlaubt)
- B) führt zu einer Überschattung des äußeren i durch das innere i; Richtig
- C) wird von Java durch automatische Umbenennung des inneren i; Falsch verarbeitet;
- D) führt dazu, dass das innere i den Wert 34 bekommt. Falsch

Ein Java-Programmstück

```
{int i = 3;}
```

```
{int i = 4;}
```

- A) ist in Java unzulässig; Falsch
- B) ist kein Problem; die beiden i's stören sich nicht; Richtig
- C) wird von Java durch automatische Umbenennung des unteren i verarbeitet; Falsch
- D) Das letzte i ist auch nach verlassen des Blocks sichtbar. Falsch

Ein Stack

- A) speichert Zahlen immer in sortierter Reihenfolge; Falsch
- B) kann eine neue Zahl an beliebiger Stelle aufnehmen; Falsch
- C) kann eine neue Zahl nur am oberen Ende (top of stack) aufnehmen; Richtig
- D) kann Zahlen nur vom oberen Ende her löschen. Richtig, LIFO-(Last-in-First-out-)Prinzip

Eine Folge von Deklarationen

```
int i = 3;
```

```
short j = 10;
```

- A) führt dazu, dass auf dem Stack 6 Bytes (4 für int und 2 für short) reserviert werden; Richtig
- B) führt dazu, dass auf dem Stack 8 Bytes (jeweils 4 Bytes) reserviert werden; Falsch
- C) führt dazu, dass auf dem Stack 13 Bytes (3 + 10) reserviert werden; Falsch
- D) führt dazu, dass auch auf der Halde Speicherplatz reserviert wird. Falsch

Folgende Typumwandlungen sind erlaubt

A) `short i = 3; int j = i;`

Richtig

B) `int i = 3; short j = i;`

Falsch

C) `double a = 3.5; int i = (int) a;`

Richtig, wobei die Nachkommastelle abgeschnitten wird

D) `long b = 12345L; double c = b;`

Richtig.

Zusatz long:

Um einen Wert vom Typ long zu erzeugen, muss an die Zahl der Buchstabe L angehängt werden. Dieser kann dabei klein oder groß geschrieben werden. Allerdings sollte die Verwendung des Kleinbuchstaben vermieden werden, da er in manchen Schriftarten leicht mit der Ziffer 1 verwechselt werden kann.

Nach einer Deklaration

`int i = 0;`

führen folgende Ausdrücke zu einem Laufzeitfehler:

A) `(i != 0) && (100/i > 1)` Falsch

B) `(i != 0) & (100/i > 1)` Richtig

C) `true || (100/i > 1)` Falsch

D) `true | (100/i > 1)` Richtig

Nach einer Deklaration

`final int TAGE_PRO_JAHR = 365;`

sind folgende Java-Anweisungen erlaubt;

A) `TAGE_PRO_JAHR++;` Falsch, es handelt sich um eine Konstante

B) `System.out.println(TAGE_PRO_JAHR);` Richtig

C) `final int TAGE_PRO_SCHALTJAHR = TAGE_PRO_JAHR + 1;` Richtig

D) `final short TAGE_PRO_SCHALTJAHR = TAGE_PRO_JAHR + 1;`

Richtig, `short` entspricht einem Wertebereich von -32768 ... 32767

Folgendes gilt für while-Schleifen

`while(<Bedingung>) {Anweisungen;}`

- A) Die Anweisungen werden auf jeden Fall einmal ausgeführt. Falsch
- B) Eine Schleife `while(true) {Anweisungen;}` terminiert nie. Falsch
- C) In dem Anweisungsblock können keine neuen Variablen deklariert werden. Falsch
- D) In dem Anweisungsblock darf keine weitere while-Schleife vorkommen. Falsch

Folgendes gilt für while-Schleifen

`while(<Bedingung>) {Anweisungen;}`

- A) Die Anweisungen werden so lange durchgeführt, bis die Bedingung falsch wird. Richtig
- B) Die Anzahl der Durchläufe durch die Schleife liegt bei Eintritt in die Schleife schon fest. Falsch
- C) Die Anweisungen können einen direkten Schleifenabbruch erzwingen. Richtig, break
- D) Die continue-Anweisung im Schleifenbody bewirkt eine Endlosschleife. Falsch, Vorzeitiger Sprung zum nächsten Schleifendurchgang

In der Anweisung

if(<Boolescher Ausdruck>) Programmstück1;
else Programmstück2;

- A) werden beide Programmstücke nacheinander ausgeführt; Falsch
- B) werden beide Programmstücke parallel ausgeführt; Falsch
- C) kann auch der else-Teil fehlen; Richtig
- D) hängt die Auswahl der auszuführenden Programmstücke vom Booleschen Ausdruck ab. Richtig

In einer Anweisung

```
for (int i = start; i < end; i++) {Programmstück;}
```

- A) wird das Programmstück mindestens einmal ausgeführt; Falsch
- B) kann die Variable `i` auch im Programmstück verändert werden; Richtig
- C) kann die Variable `,end'` auch im Programmstück verändert werden; Richtig
- D) beeinflusst die Änderung der Variable `,start'` im Programmstück das Verhalten der Schleife. Falsch

Die break;-Anweisung

- A) kann in while- und for-schleifen vorkommen; Richtig
- B) bricht das Programm sofort ab; Falsch
- C) bricht die Methode, in der ,break;' vorkommt, sofort ab; Falsch, z. B. aber mit return 0
- D) bricht die Schleife, in der ,break;' vorkommt, sofort ab. Richtig

Folgendes gilt für Klassen in Java:

- A) Klassen und Instanzen sind dasselbe. Falsch
- B) Jede Klasse hat genau eine Instanz. Falsch
- C) Von jeder Klasse können beliebig viele Instanzen gebildet werden. Richtig
- ✗ D) Instanzen Klassen werden ebenfalls auf dem Stack abgelegt. Falsch

Folgendes gilt für Instanzvariablen:

- A) Instanzvariablen können nur primitive Datentypen haben. Falsch
- B) Instanzvariablen werden mit static gekennzeichnet. Falsch, Klassenvariablen
- C) Auf Instanzvariablen kann auch von statischen Methoden aus zugegriffen werden. Falsch
- ✗ D) Jede Instanz einer Klasse hat gleich viele Instanzvariablen. Richtig

Folgendes gilt für Klassenvariablen:

- A) Klassenvariablen werden mit static gekennzeichnet. Richtig
- B) Auf Klassenvariablen kann man auch von Instanzmethoden aus zugreifen. Richtig
- C) Klassenvariablen eignen sich, um die Anzahl der Instanzen einer Klasse zu zählen. Richtig
- D) Klassenvariablen eignen sich zur Kommunikation zwischen Instanzen einer Klasse. Richtig

Folgendes gilt für Konstruktormethoden

- A) Es gibt immer mindestens eine Konstruktormethode. Richtig
- B) Man kann beliebig viele Konstruktormethoden definieren. Richtig
- C) Man kann den Namen einer Konstruktormethode beliebig wählen. Falsch
- D) Konstruktormethoden werden mit ‚new‘ aufgerufen. Richtig

Folgendes gilt für Instanzmethoden

- A) Die Instanzmethoden in verschiedenen Klassen müssen auch verschiedene Namen haben. Falsch
- B) In einer Klasse kann man nur jeweils eine Instanzmethode mit einem Namen definieren. Falsch, man kann zwar nur eine einzige Methode mit einem bestimmten Namen definieren, wenn sie die exakt selben Parameter akzeptieren, aber Methoden mit unterschiedlichen Parametern können den selben Namen haben
- C) Instanzmethoden können auch folgendermaßen aufgerufen werden: <Klassenname>.<Methodenname>(<Argumente>). Falsch
- D) Bei einem Aufruf <Objekt>.<Methodenname>(<Argumente>) kann die Methode auch auf die Instanzvariablen anderer Instanzen der Klasse zugreifen. Falsch

Folgendes gilt für Klassenmethoden

- ✗ A) Klassenmethoden werden mit static gekennzeichnet. Richtig
- B) Klassenmethoden können auch über Instanzen aufgerufen werden. Richtig, sollte aber auf den static-Weg erfolgen
- C) Klassenmethoden können auch auf Instanzvariablen zugreifen. Falsch
- D) Klassenmethoden sind prinzipiell immer public. Falsch, können auch private oder auch public sein

- A) Instanzen einer Klasse werden auf der Halde abgelegt. Richtig
- B) In einem Block
 {Punkt2D p = new Punkt2D(); ...}
ist diese neue Instanz der Klasse Punkt2D auf jeden Fall nach verlassen des Blocks gelöscht. Falsch
- C) Die Halde ist ein Speicherbereich, der auch Lücken enthalten kann. Richtig
- D) Es kann immer nur einen Zeiger auf ein Objekt der Halde geben. Falsch

- A) getter- und setter-Methoden werden nur für Instanzvariablen definiert. Falsch
- B) getter- und setter-Methoden müssen immer definiert werden. Falsch
- C) getter- und setter-Methoden helfen, die innere Struktur einer Klasse abzuschotten. Richtig
- D) getter- und setter-Methoden machen Programme änderungsfreundlicher. Richtig

getter- und setter-M. erlauben den kontrollierten Zugriff auf die Variablen.

- A) Jede Klasse kann beliebig viele Oberklassen haben. Falsch
- B) Die Unterklasse einer Klasse kann keine Methode der Oberklasse neu definieren. Falsch
- C) Nur Instanzvariablen werden an Unterklassen vererbt. Falsch
- ✗ D) Methoden von Unterklassen haben auch Zugriff auf die entsprechenden Methoden der Oberklasse. Richtig

Betrachte folgende Klassenhierarchie:

```
public class A {  
    public int f() {return 1;}}  
  
public class B extends A {  
    public int f() {return 2;}  
    public static void main(String[] args) {  
        A a = new A(); B b = new B(); A c = b;  
        System.out.println(a.f() + b.f() + c.f());}}}
```

Was wird ausgedruckt?

- A) 121
- ☒ B) 122
- C) 112
- D) 111

*richtet sich nach dem
}-Objekt.
Wenn c.super.f() dann
würde es 121 sein*

In einer Klassenhierarchie gilt:

A) Der Typ einer Variablen (z.B. Punkt2D p;) bestimmt, auf welche Methoden mit p.<Methodenname>(…) zugegriffen werden darf.

Falsch

B) Dies wird vom Compiler überprüft.

~~ergibt sich erst zur Laufzeit~~ →

Richtig

X C) Die Objektklasse bestimmt in der Vererbungshierarchie, welche der vererbten Methoden aufgerufen wird.

Richtig

X D) Dies wird zur Laufzeit festgelegt.

Richtig

- A) Mit **private** gekennzeichnete Instanzvariablen sind auch in Unterklassen zugreifbar. Falsch
- B) Mit **public** gekennzeichnete Klassenvariablen sind auch in Instanzmethoden zugreifbar. Richtig
- C) Mit **protected** gekennzeichnete Instanzvariablen können nicht geändert werden. (mit final) Falsch
- D) Mit **public** gekennzeichnete Methoden einer Klasse können nur von anderen Klassen aus aufgerufen werden. Falsch

- A) Generische Klassen können beliebig viele Typvariablen haben. Richtig
- B) Bei Aufrufen der Konstruktoren von generischen Klassen können die Typvariablen auch durch primitive Datentypen ersetzt (instantiiert) werden. Falsch
- C) Die Vererbungshierarchie von Klassen setzt sich auch auf die Vererbungshierarchie von generischen Klassen fort (z.B. Kloster<Frau> Unterklasse von Kloster<Person>). Falsch

✗ D) Die Typvariablen generischer Klassen können eingeschränkt werden (wie).

*<T extends kleiner>
(mit super auf Oberklassen (sehr unnützlich))*