# Fragen zur Klausurvorbereitung

Sonntag, 21. Februar 2021 10:24

**1. Zahlensysteme**
a) Wie sind positive ganze Zahlen in Java realisiert?

Im Zweierkomplement?

Das *Zweierkomplement* definiert für positive und negative Ganzzahlen folgende Kodierung:

- Das Vorzeichen einer Zahl bestimmt ein Bit, das 1 bei negativen und 0 bei positiven Zahlen ist.
- Um eine 0 darzustellen, ist kein Bit gesetzt.

Java kodiert die Ganzzahldatentypen byte, short, int und long immer im Zweierkomplement (der Datentyp char definiert keine negativen Zahlen). Mit dieser Kodierung gibt es eine negative Zahl mehr als positive, da es im Zweierkomplement keine positive und negative 0 gibt, sondern nur eine »positive« mit der Bit-Maske 0000…0000.

Aus <http://openbook.rheinwerk-verlag.de/javainsel/22_001.html>

Durch den Datentyp **int** werden ganze Zahlen repräsentiert. Für eine int-Zahl werden 4 Byte zur Abspeicherung verwendet; durch sie werden alle ganzen Zahlen von $-2^{31}$ bis $2^{31}-1$ dargestellt. Kommt man mit diesem Zahlenbereich nicht aus, kann man den Grunddatentyp **long** verwenden; kleinere Zahlenbereiche werden durch die Grunddatentypen short, byte erfasst.

b) Wie sind negative ganze Zahlen in Java realisiert?

c) Wie sind Gleitkommazahlen in Java realisiert?

float, double

d) Welchen Zahlenbereich decken die Zahltypen von Java ab?

-1,7*10^308 bis 1,7*10^308

Der größere Fließkomma-Datentyp wird als **double** bezeichnet. Double besitzt eine Größe von 64 Bit und der Wertebereich erstreckt sich von -1,7 * 10^308 bis 1,7 * 10^308
https://www.java-tutorial.org/datentypenundvariablen.html

| Typ | Wertebereich | Länge |
|---|---|---|
| byte | -128…127 | 8 Bit |
| short | -32768…32767 | 16 Bit |
| int | -2147483648…2147483647 | 32 Bit |
| long | -9223372036854775808…9223372036854775807 | 64 Bit |

**2. Zeichencodes**
a) Geben Sie die binäre Darstellung eines U als ASCII, Unicode, UTF-8 und UTF-16 an.

ASCII: 0101 0101 (binär)
Unicode: U+0055
UTF-8: 0101 0101
UTF-16: 0000 0000 0101 0101

b) Geben Sie die binäre Darstellung eines Ü als Unicode, UTF-8 und UTF-16 an.

ASCII: 1101 1100 (binär)
Unicode: U+00DC
UTF-8: 1100

**3. Datentypen**
a) Nennen Sie alle primitiven Datentypen von Java.

| Typname | Größe[1] | Wrapper-Klasse | Wertebereich | Beschreibung |
|---|---|---|---|---|
| boolean | undefiniert[2] | java.lang.Boolean | true / false | Boolescher Wahrheitswert, Boolescher Typ[3] |
| char | 16 bit | java.lang.Character | 0 … 65.535 (z. B. 'A') | Unicode-Zeichen (UTF-16) |
| byte | 8 bit | java.lang.Byte | -128 … 127 | Zweierkomplement-Wert |
| short | 16 bit | java.lang.Short | -32.768 … 32.767 | Zweierkomplement-Wert |
| int | 32 bit | java.lang.Integer | -2.147.483.648 … 2.147.483.647 | Zweierkomplement-Wert |
| long | 64 bit | java.lang.Long | $-2^{63}$ bis $2^{63}-1$, ab Java 8 auch 0 bis $2^{64}-1$[4] | Zweierkomplement-Wert |
| float | 32 bit | java.lang.Float | +/-1,4E-45 … +/-3,4E+38 | 32-bit IEEE 754, es wird empfohlen, diesen Wert nicht für Programme zu verwenden, die sehr genau rechnen müssen. |
| double | 64 bit | java.lang.Double | +/-4,9E-324 … +/-1,7E+308 | 64-bit IEEE 754, doppelte Genauigkeit |

b) Welche Referenztypen gibt es in Java?

Objekte, Strings und Arrays. Außerdem die vordefinierte Konstante null, die eine leere Referenz bezeichnet.

In Folie 2-11:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 00 | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL |
| 01 | BS | HT | NL | VT | NP | CR | SO | SI |
| 02 | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB |
| 03 | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 04 | SP | ! | " | # | $ | % | & | ' |
| 05 | ( | ) | * | + | , | - | . | / |
| 06 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 07 | 8 | 9 | : | ; | < | = | > | ? |
| 10 | @ | A | B | C | D | E | F | G |
| 11 | H | I | J | K | L | M | N | O |
| 12 | P | Q | R | S | T | U | V | W |
| 13 | X | Y | Z | [ | \ | ] | ^ | _ |
| 14 | ` | a | b | c | d | e | f | g |
| 15 | h | i | j | k | l | m | n | o |
| 16 | p | q | r | s | t | u | v | w |
| 17 | x | y | z | { | \| | } | ~ | DEL |

- 1-Byte-Zeichen: 0xxxxxxx — 7 Bit
- 2-Byte-Sequenz: 110xxxxx 10xxxxxx — 11 Bit
- 3-Byte-Sequenz: 1110xxxx 10xxxxxx 10xxxxxx — 16 Bit
- 4-Byte-Sequenz: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx — 21 Bit

Beispiele:

| Zeichen | UTF-16 | UTF-16 binär | UTF-8 binär |
|---|---|---|---|
| A | U+0041 | 00000000 01000001 | 01000001 |
| Ä | U+00C4 | 00000000 11000100 | 11000011 10000100 |
| € | U+20AC | 00100000 10101100 | 11100010 10000010 10101100 |

$$11,11_2 = (-1)^0 \cdot 1.1111 \cdot 2^{128-127}$$

128

| 0 | 10000000 | 11100000000000000000000 |
|---|---|---|

"this" verweist auf das aktuelle initialisierte Objekt indem sich die Methode befindern in der this aufgerufen wird.
"this" kann nur in Instanzmethoden benutzt werden, nicht in statischen Methoden.
"this" ist immer der erste Übergabeparameter bei einer Instanzmethode (unsichtbar).

- *Kapselung*: Daten (Klassen, Methoden, Variablen) können in Java mit Hilfe von Modifikatoren (modifier; private/public/protected) gekapselt werden, damit der Zugriff darauf genau geregelt ist (Geheimnisprinzip).
  z.B.
  private String titel;
  vs.
  public String titel;

- *Vererbung*: erlaubt die Definition ähnlicher Klassen, indem man Gemeinsamkeiten in Oberklassen zusammenfasst und Eigenheiten in davon abgeleiteten Unterklassen ergänzt
  z.B.
  public final class Unterklasse extends Oberklasse {…}

- *Polymorphie*: ein und dieselbe Variable kann zur Laufzeit Objekte unterschiedlicher Klassen referenzieren. Stichwort Subtyping
  z.B.
  Object x;
  x = new Beispiel();
  x = 1;

- *Dynamische Bindung*: erst zur Laufzeit wird entschieden, welche Methodenimplementierung aufgerufen wird
  z.B.
  new Integer(1234).intValue();
  (Methode intValue ist auch in der Oberklasse Number definiert und es wird erst zur Laufzeit erkannt, welches intValue() aufgerufen wird

b) Was ist bei Klassen für Wertobjekte (value objects) zu beachten?

- pro Objekt ein unveränderlicher Wert
- Wert -> Instanzvariablen, final
- gleicher Wert -> Wertobjekte sind gleich

c) Erklären Sie, was eine Schnittstelle ist und wie sie verwendet wird.

- Zur Definition einheitlich benutzbarer Klassen
- abstrakte Oberklasse mit ausschließlich öffentlichen Methoden, ohne Instanzvariablen, ohne Konstruktor

**Referenztypen**

*die ausgegrauten Referenztypen werden in Teil 4 und 5 behandelt*

- Felder: *Typ* `[]`
- Klassen
  > Fundamentalklassen:
  *Zeichenketten* `String`, ...
  *Wrapperklassen* `Boolean`, ...
  *Wurzelklassen* `Object`, `Throwable`
  *Typinformation* `Class<T>`, ...
  ...
  > Anwendungsklassen:
  `enum` *Name*
  `class` *Name*
  `interface` *Name*

c) Erklären Sie den Unterschied zwischen Referenztypen und Werttypen.

Werttyp-Variablen speichern den unmittelbaren Wert des angegebenen Typs. Der Wert wird bei einer Zuweisung kopiert.

Variablen mit Referenztyp speichern nur die Information, *wo* der Wert des angegebenen Typs steht. Der eigentliche Wert wird bei einer Zuweisung nicht kopiert, nur die Referenz (d.h. die Adresse der Werte).

**4. Variablen**

a) Erklären Sie an einer Beispielklasse, was lokale Variablen, Parameter, Klassenvariablen und Instanzvariablen sind. Wo und wie werden sie jeweils definiert und wo und wie können sie benutzt werden?

Siehe Bences Word-Datei

b) Nennen Sie alle Möglichkeiten, die unterschiedlichen Variablen aus a) zu initialisieren.

**5. Ausdrücke**

Wie bestimmt man die Ausführungsreihenfolge eines Ausdrucks?

| *Operator* | *Name* | *Stelligkeit* | *Assoziativität* | *Vorrang* [1] |
|---|---|---|---|---|
| ++ | Postfix-Inkrement | unär | – [2] | 1 |
| -- | Postfix-Dekrement | unär | – [2] | 1 |
| . *Komponente* | Auswahl | unär | – [2] | 1 |
| [ *Index* ] | Indizierung | unär [3] | – [2] | 1 |
| ( *Parameterliste* ) | Methodenaufruf | unär [3] | – [2] | 1 |
| ++ | Präfix-Inkrement | unär | – [2] | 2 |
| -- | Präfix-Dekrement | unär | – [2] | 2 |
| + | Unäres Plus | unär | – [2] | 2 |
| - | Unäres Minus | unär | – [2] | 2 |
| ! | Logische Negation | unär | – [2] | 2 |
| ~ | Bitweise Invertierung | unär | – [2] | 2 |
| (*Typ* ) | Typanpassung | unär | – [2] | 2 |

Siehe Folie 3-4ff.

**6. Anweisungen**

a) Für welche Datentypen kann man eine Fallunterscheidung mit switch formulieren?

byte, short, char, int, String, enum

b) Welche Schleifen gibt es in Java? Formulieren Sie jeweils ein Anwendungsbeispiel.

for, for-each, while, do while

c) Was ist bei der Reihenfolge der catch-Blöcke einer Ausnahmebehandlung zu beachten?

Absteigende Reihenfolge der Spezialisierung von Ausnahmen (erste passende exception wird ausgeführt)

**7. Methoden**

a) Erklären Sie an einer Beispielklasse, was Klassenmethoden, Instanzmethoden und abstrakte Instanzmethoden sind. Wo und wie werden sie jeweils definiert und wo und wie können sie aufgerufen werden?

b) Erklären Sie das Überladen (Overloading) von Methoden.

Beim Overloading können Methoden einer Klasse denselben Namen haben,
wenn sie sich in der Parameterliste unterscheiden. Der Compiler wählt die passende aus.

c) Erklären Sie das Überschreiben (Overriding) von Methoden.

Unterklassen können (vor allem abstrakte) Instanzmethoden ihrer Oberklasse(n) überschreiben, d.h. anders implementieren.

**8. Zeichenketten**

a) Gegeben seien zwei String-Variablen s und t. Welche Bedeutung hat der Ausdruck s + t und wie kann das gleiche mit der Klasse java.lang.StringBuilder realisiert werden?

Mit StringBuilder und .append()

b) Wie vergleicht man Strings?

.equals()

**9. Klassen**

a) Geben Sie jeweils ein Beispiel für

- eine Klasse, die nur / auch / nicht als Oberklasse verwendbar ist
- eine Unterklasse
- eine instanziierbar Klasse / eine Utility-Klasse

nur Oberklasse: abstract

nicht als Oberklasse: final

Unterklasse: public class Unterklasse extends Oberklasse

eine instanziierbare Klasse:

Utility-Klasse: enthalten nur Klassenmethoden und (konstante) Klassenvariablen, Konstruktor meist private

```
public class UtilityKlasse {
    public final static double PI = 3.141592653589793;

    static double berechneKreisFlaeche(double radius) {
    double flaeche = PI*(radius*radius);
    return flaeche;
    }
}
```

b) Was ist bei Klassen für Wertobjekte (value objects) zu beachten?

- pro Objekt ein unveränderlicher Wert
- Wert -> Instanzvariablen, final
- gleicher Wert -> Wertobjekte sind gleich

c) Erklären Sie, was eine Schnittstelle ist und wie sie verwendet wird.

- Zur Definition einheitlich benutzbarer Klassen
- abstrakte Oberklasse mit ausschließlich öffentlichen Methoden, ohne Instanzvariablen, ohne Konstruktor

d) Erklären Sie die Begriffe Kapselung, Vererbung, Polymorphie und dynamische Bindung

- *Kapselung*: Daten (Klassen, Methoden, Variablen) können in Java mit Hilfe von Modifikatoren (modifier; private/public/protected) gekapselt werden, damit der Zugriff darauf genau geregelt ist (Geheimnisprinzip).
  z.B.
  private String titel;
  vs.
  public String titel;

- *Vererbung:* erlaubt die Definition ähnlicher Klassen, indem man Gemeinsamkeiten in Oberklassen zusammenfasst und Eigenheiten in davon abgeleiteten Unterklassen ergänzt
  z.B.
  public final class Unterklasse extends Oberklasse {…}

- *Polymorphie*: ein und dieselbe Variable kann zur Laufzeit Objekte unterschiedlicher Klassen referenzieren. Stichwort Subtyping
  z.B.
  Object x;
  x = new Beispiel();
  x = 1;

- *Dynamische Bindung*: erst zur Laufzeit wird entschieden, welche Methodenimplementierung aufgerufen wird
  z.B.
  new Integer(1234).intValue();
  (Methode intValue ist auch in der Oberklasse Number definiert und es wird erst zur Laufzeit erkannt, welches intValue() aufgerufen wird