EI Info J2 2011-12	*INFORMATIK* 1. Klausur - Probeklausur	

Vorgeschlagene Bearbeitungszeit: 60 Minuten. In der Klausur wird es möglich sein, eine (zwei für „Neue“) Teilaufgaben zu streichen! Hier sind mehr Übungsaufgaben als Aufgaben in der Arbeit drankommen!

1. Aufgabe

a) Erläutere, was objektorientierte Programmierung ist.

In der objektorientierten Programmierung versucht man, alles in Klassen zu unterteilen und diese voneinander zu unterscheiden (abzukapseln). Wendet man diese Idee auf die Welt an und ist man bspw. Gott, dann könnte das so aussehen: Wir überlegen uns, dass es schön wäre, wenn es Pferde auf unserer World „Erde“ gibt. Also erschaffen wir eine public class Pferd. Nun haben wir viele Einfälle für Wesen, die auch Pferd sind; wir stellen uns ein Pony vor, ein Zebra und einen weißen Schimmel. Alle diese Wesen wären extends Pferd, also von der Klasse Pferd und hätten deren grundlegende Eigenschaften wie „laufen()“, „schlafen()“, „fressen()“, „wiehern()“ und was ein Pferd halt noch so ausmacht („Fellfarbe“, „Geschlecht“). Einmal die Klasse Pferd erschaffen, ist es leicht, neue Pferdearten zu programmieren!

b) Erläutere, was ein „Konstruktor“ ist und welche Aufgabe er übernimmt, wenn er aufgerufen wird. Gib ein Konstruktor-Beispiel aus einem der beiden angehängten Quellcodes.

Beziehen wir uns auf Teil a), dann haben wir uns die Klasse Pferd überlegt und sagen wir mal, uns zwei Sorten von Pferden, Zebra, Pony, definiert. Nun gibt es aber auf der Welt noch lange keine Pferde, nur die Idee ist da. Der Konstruktor erschafft uns Pferde! Im Konstruktur vom Pony stehen die Startbedingungen für alle Parameter des Ponys, wobei viele vom Pferd geerbt wurden. Dabei ist die Größe bspw. ein anderer Wert als beim Konstruktor vom Zebra, denn das Pony ist kleiner.

2. Aufgabe

a) Erläutere, was die Vorteile einer if-else-Bedingung in der Programmierung ist.

Um Fallunterscheidungen zu machen. Beispielsweise möchte ich in meiner World „Welt“ unterscheiden, was passiert, wenn die class „Mensch“ von einem Pony oder von einem Zebra in die Hand gebissen wird. Dort würde man if (es ist ein Pony) und if (es ist ein Zebra) verwenden…

b) Erläutere, was die Vorteile einer for-Schleife in der Programmierung ist.

Mit der for-Schleife lassen sich auf einfache Weise sich wiederholende Befehle geben. Beispielsweise möchte ich eine Liste mit Zufallszahlen füllen. Dann verwende

ich eine for-Schleife in der Art „for (int i; i<LängeDerListe; i++) {Liste[i]=get.RandomNumber();}".

3. Aufgabe

a) Was macht folgende for-Schleife in Java:

for (int i=1; i<10; i=i+2) {System.out.println(„Das ist eine ungerade Zahl: "+i);}

Hier werden die ersten ungeraden Zahlen bis 9 ausgegeben; es beginnt bei i=1 mit der 1, dann wird i um 2 erhöht und man gibt die 3 aus, dann 5, 7 und 9. Hier endet die Schleife, denn mit i=9 ist i noch kleiner 10 wie in der Schleife gefordert. Danach ist mit i=11 aber i<10 verletzt!

b) Schreibe eine for-Schleife, die in einem unsortierten Array mit „Int" als Einträgen das Maximum herausfindet. Pseudocode ist erlaubt!

Wir gehen davon aus, dass die Listeneinträge mit Liste[i] bezeichnet werden können und die Liste die Länge LaengeListe besitzt. Außerdem haben wir eine Hilfsvariable „max" (ein int) weiter oben definiert und max=0 gesetzt. Der Pseudocode:

```
for (int i=1; i<LaengeListe; i++)
{
        if (max<Liste[i];) {max=Liste[i];}
}
```

Hier kann es passieren, dass die Liste nur Einträge kleiner 0 aufweist oder gar keine Zahlen, aber das fangen wir jetzt nicht mit Test-ifs ab.

In der Schleife wird das max wohl schon mit Liste[1] überschrieben (außer, es ist Null, aber davon gehen wir wie gesagt nicht aus). Dann wird im nächsten Schritt Liste[2] mit dem neuen max verglichen. Entweder gewinnte Liste[2] und ersetzt max oder halt nicht. Am Ende der Liste hat man so ein max gefunden. Aber nur „ein" max, nicht „das" max, denn es könnte mehrere gleichgroße max-Werte geben. Dann hätten wir das letzte gespeichert. Aber ein max haben wir gefunden!

4. Aufgabe

Entscheide, ob die folgenden Ausdrücke in Java den booleschen Wert „true" oder „false" liefern. Stelle dabei deinen Lösungsweg nachvollziehbar dar!

a) !(23>4) b) (!(false)) || (false) c) ((13>31) || (32>31)) && (!(false))

Zu a): 23 ist größer 4. Wegen ! wird aus true aber FALSE.

Zu b): ! und false ist true. wahr ODER falsch ist TRUE.

Zu c): 13 ist nicht größer 31, aber wegen dem ODER ist das noch wahr, weil 32 wirklich größer als 31 ist. Jetzt kommt ein UND mit wahr, weil nicht falsch halt wahr ist und insgesamt TRUE.

5. Aufgabe
Wofür ist die Direkteingabe bei BlueJ? Erläutere kurz!

Mit der Direkteingabe kann man einzelne Befehlszeilen prüfen. Man könnte bspw. die Booleschen Ausdrücke aus A4) eingeben und sich das Ergebnis anzeigen lassen.

6. Aufgabe
Es gibt primitive Typen in der Informatik. Gib ein Beispiel und erläutere, was man darunter versteht.

Hier mal eine Liste der primitiven Typen!

Datentyp	Größe[a]	Wrapper-Klasse	Wertebereich	Beschreibung
boolean	1 Bit	java.lang.Boolean	true / false	Boolescher Wahrheitswert
char	16 Bit	java.lang.Character	U+0000 … U+FFFF	Unicode-Zeichen (z. B. `'A'` oder `'\uC3A4'`)
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	−9.223.372.036.854.775.808 … +9.223.372.036.854.775.807	Zweierkomplement-Wert
float	32 Bit	java.lang.Float	±1,4E−45 … ±3,4E+38	Gleitkommazahl (IEEE 754)
double	64 Bit	java.lang.Double	±4,9E−324 … ±1,7E+308	Gleitkommazahl doppelter Genauigkeit (IEEE 754)

[a] „Größe" gibt den minimalen Speicherverbrauch an.

In Aufgabe 1 ging es um objektorientierte Programmierung. Genauso, wie es Pferde gibt, gibt es ja auch INTEGER. Oder BOOLEAN. Aber das sind keine echten Objekte! Sie können sozusagen nix… bzw. nicht viel…

7. Aufgabe
Im Anhang findest Du die Quelltexte der zwei Klassen **Uhrenanzeige** und **Nummernanzeige**, die beide zum bekannten Projekt Zeitanzeige gehören. Die folgenden Teilaufgaben beziehen sich darauf:

a) Die Klasse **Uhrenanzeige** besitzt zwei Konstruktoren. Erläutere deren grundsätzlichen Unterschied.

Beim einen Konstruktor wird eine Uhr erschaffen, die sofort bei 00:00 Uhr startet. Beim zweiten Konstruktor wird eine Uhr erschaffen, die zu einer vom Erschaffer festzulegenden Zeit startet.

b) Gibt es eine Möglichkeit, sich schnell eine Stoppuhr mit Sekunden und Hunderstel zu bauen? Wie müsste man den Konstruktor verändern?

Klar, man übergibt 60 und 100 an den Konstruktor von Nummernanzeige. Man muss sogar nicht einmal aufpassen, wegen der drei Stellen der 100, da man ja nach 99 auf 00 umspricht ;-)

c) Erläutere an Beispielen aus dem Quelltext die Unterschiede zwischen Methoden mit und ohne (void) Rückgabewert.

Methoden mit void geben nix zurück, bspw. beim Uhrzeitsetzen möchte man nichts zurück, die Zeit soll einfach neu gestellt sein. Man könnte natürlich noch einen Rückgabewert einbauen, wenn man eine Bestätigung möchte!

d) In der Methode **taktsignalGeben** taucht die Anweisung **„minuten.Erhoehen()"** auf. Wieso wird diese Schreibweise verwendet und was geschieht hier?

Hier wird auf das von uns erstellte Objekt vom Typ Nummernanzeige zugegriffen, welches wir minuten genannt haben und ruft die dort bekannte Methode Erhoehen auf, womit wohl die Anzeige um 1 hochgesetzt wird.

e) In der Methode **setzeWert** der **Nummernanzeige** taucht die Bedingung **„if((ersatzwert >= 0) && (ersatzwert < limit))"** auf. Erläutere kurz den Zweck dieser Bedingung.

Diese Bedingung schaut nach, ob die Variable ersatzwert größergleich 0 ist und kleiner als das für die Anzeige mögliche Limit (weil es bspw. nur 2 Stellen gibt). Damit kein Quatsch eingestellt werden kann.

Quelltext der Klasse Uhrenanzeige

```
/**
 * Die Klassen Uhrenanzeige implementiert die Anzeige einer Digitaluhr.
 *
 * @author Michael Kölling und David J. Barnes
 * @version 2008.03.30
 */
public class Uhrenanzeige
{
    private Nummernanzeige stunden;
    private Nummernanzeige minuten;
    private String zeitanzeige;    // simuliert die tatsächliche Anzeige

    /**
     * Konstruktor für ein Exemplar von Uhrenanzeige.
     */
    public Uhrenanzeige()
    {
        stunden = new Nummernanzeige(24);
        minuten = new Nummernanzeige(60);
        anzeigeAktualisieren();
    }

    public Uhrenanzeige(int stunde, int minute)
    {
        stunden = new Nummernanzeige(24);
        minuten = new Nummernanzeige(60);
        setzeUhrzeit(stunde, minute);
    }

    public void taktsignalGeben()
    {
        minuten.erhoehen();
        if(minuten.gibWert() == 0) {
            stunden.erhoehen();
```

```
        }
        anzeigeAktualisieren();
    }

    public void setzeUhrzeit(int stunde, int minute)
    {
        stunden.setzeWert(stunde);
        minuten.setzeWert(minute);
        anzeigeAktualisieren();
    }
    public String gibUhrzeit()
    {
        return zeitanzeige;
    }


    private void anzeigeAktualisieren()
    {
        zeitanzeige = stunden.gibAnzeigewert() + ":"
                    + minuten.gibAnzeigewert();
    }
          }
```

Quelltext der Klasse Nummernanzeige

```
/**
 * Die Klasse Nummernanzeige repräsentiert Darstellungen von digitalen Werten.
 *
 * @author Michael Kölling und David J. Barnes
 * @version 2008.03.30
 */
public class Nummernanzeige
{
    private int limit;
    private int wert;

    /**
    public Nummernanzeige(int anzeigeLimit)
    {
        limit = anzeigeLimit;
        wert = 0;
    }

    public int gibWert()
    {
        return wert;
    }

    public String gibAnzeigewert()
    {
        if(wert < 10) {
            return "0" + wert;
        }
        else {
            return "" + wert;
        }
    }

    public void setzeWert(int ersatzwert)
    {
        if((ersatzwert >= 0) && (ersatzwert < limit)) {
            wert = ersatzwert;
```

```
    }
}

public void erhoehen()
{
    wert = (wert + 1) % limit;
}}
```