Prof. Dr. Stefan Brass
Dipl.-Inform. Steffen Schiele
Institut für Informatik
MLU Halle-Wittenberg

25. März 2014

Vorlesung "Objektorientierte Programmierung"
— Klausur —

Name: _____

Matrikelnummer: _____

Studiengang: _____

Programmierkenntnisse vor der Vorlesung: ☐ keine ☐ etwas ☐ viel

Aufgabe	Punkte	von	Zeit
1 (Programmierung: Klasse)		12	15 min
2 (Programmierung: Array)		12	15 min
3 (Wertausdrücke, Bedingung, Referenztyp)		5	5 min
4 (Methoden, while, if)		3	10 min
5 (Klassen, Parameter, Verschattung)		3	10 min
6 (Exceptions)		3	5 min
7 (Vererbung)		3	10 min
8 (Fehlersuche)		3	10 min
9 (Bonus-Aufgabe)		0+3	10 min
Summe		44	90 min

- Ich fühle mich gesundheitlich in der Lage, diese Prüfung abzulegen. (Bitte sprechen Sie mit dem Aufsichtspersonal, falls Sie sich krank fühlen. Eine Krankmeldung muss vor Ende der Prüfung erfolgen.)
- Ich erkläre, dass ich diese Prüfung nicht bereits endgültig nicht bestanden habe.
- Falls ich nicht korrekt zu dieser Prüfung angemeldet sein sollte, erkläre ich mich damit einverstanden, dass ich rückwirkend angemeldet werde.

Unterschrift: _____

Hinweise:

- Bearbeitungsdauer: 90 Minuten
- Es dürfen drei Din A4-Blätter mit Notizen verwendet werden (Vorder- und Rückseite, also 6 Seiten, auch gedruckt). Weitergehene schriftliche Unterlagen (z.B. Bücher, Skripte), Notebooks, PDAs, etc. dürfen nicht verwendet werden. Mobiltelefone bitte ausschalten (oder mit Aufsicht besprechen).
- Die Klausur hat 14 Seiten. Bitte prüfen Sie die Vollständigkeit.
- Bitte benutzen Sie den vorgegebenen Platz. Wenn Sie auf die Rückseite oder ein Zusatzblatt ausweichen müssen, markieren Sie klar, dass es eine Fortsetzung gibt.
- Tauschen Sie keinesfalls irgendwelche Dinge mit den Nachbarn aus. Notfalls rufen Sie eine Aufsichtsperson zur Kontrolle.
- Versuchen Sie, alle Aufgaben zu bearbeiten (notfalls raten Sie). Wenn Sie nichts hinschreiben, haben Sie den Punkt für die jeweilige Teilaufgabe auf jeden Fall verloren.
- Fragen Sie, wenn Ihnen eine Aufgabe nicht klar ist!
- Schreiben Sie lesbar! Verwenden Sie keinen roten Stift. Bleistift ist nicht verboten, aber problematisch, wenn Sie sich nach der Klausureinsicht beschweren wollen.
- Mit 60% der Punkte haben Sie garantiert bestanden, die Grenze wird ggf. gesenkt.
- Die von den Programmen gedruckten Markierungen "`a)`" u.s.w. für die Teilaufgaben brauchen Sie nicht nochmals aufzuschreiben (sind bereits vorgedruckt).
- Sie dürfen vor 11^{30} nicht gehen, danach nicht mehr auf Toilette.

Zum Nachschlagen:

- Tabelle mit den Prioritätsstufen der Operatoren (von hoch nach niedrig):

1	`o.a`, `i++`, `a[]`, `f()`, ...	Postfix-Operatoren
2	`-x`, `!`, `~`, `++i`, ...	Präfix-Operatoren
3	`new C()`, `(type) x`	Objekt-Erzeugung, Cast
4	`*`, `/`, `%`	Multiplikation etc.
5	`+`, `-`	Addition, Subtraktion
6	`<<`, `>>`, `>>>`	Shift
7	`<`, `<=`, `>`, `>=`, `instanceof`	kleiner etc.
8	`==`,`!=`	gleich, verschieden
9	`&`	Bit-und, logisches und
10	`^`	Bit-xor, logisches xor
11	`|`	Bit-oder, logisches oder
12	`&&`	logisches und (bedingt)
13	`||`	logisches oder (bedingt)
14	`?:`	Bedingter Ausdruck
15	`=`, `+=`, `-=`, `*=`, `/=`, ...	Zuweisungen

  Bei gleicher Priorität sind alle Operatoren (außer Präfixoperatoren, Zuweisungen, bedingter Ausdruck) implizit von links geklammert (linksassoziativ).

- Bei der Integer-Division wird immer in Richtung 0 gerundet, für positive Eingabewerte also abgerundet, z.B. liefert `8/3` das Ergebnis `2`.

Aufgabe 1 (Programmierung: Klasse) 12 Punkte

Beim "Korrespondenzzirkel Mathematik" gab es eine Aufgabe der folgenden Art: "Anna hat einen Farbeimer, der 8 Liter fasst, und voll gefüllt ist. Außerdem hat sie noch zwei leere Eimer, der eine fasst 5 Liter, der andere 3 Liter. Kann Sie damit 1 Liter Farbe abmessen?" Nachdem Sie eine Weile herumprobiert haben, entschließen Sie sich, mit einem Programm alle möglichen Verteilungen der Farbe auf die drei Eimer auszuprobieren.

- In dieser Aufgabe (Aufgabe 1) soll nur eine Klasse `Zustand` programmiert werden, welche die Füllstände der drei Eimer beschreibt (zu jeweils einem Zeitpunkt).
- Aufgabe 2 soll eine Liste von Zuständen verwalten (für die erreichbaren Zustände), wobei Duplikate vermieden werden müssen.
- Erst in der freiwilligen Bonusaufgabe 9 geschieht das eigentliche Umfüllen.

Hier ist also eine Klasse "`Zustand`" zu definieren, die den aktuellen Inhalt der drei Eimer festhält. Z.B. sind im Startzustand 8 Liter Farbe im 8 Liter Eimer, und 0 Liter Farbe im 5 Liter und im 3 Liter Eimer. Die Klasse `Zustand` soll folgende Schnittstelle haben:

- Der Konstruktor soll drei Parameter für die Inhalte der drei Eimer (in der Reihenfolge 8 Liter, 5 Liter, 3 Liter) haben. Die Angaben sind ganze Zahlen. Sie brauchen die Werte nicht auf Korrektheit zu prüfen (negative Werte wären eigentlich nicht sinnvoll, aber Sie speichern einfach in dem Objekt, was übergeben wird). Der initiale Zustand könnte also so konstruiert werden:

  ```
  Zustand z0 = new Zustand(8, 0, 0);
  ```

- Die Methode `inhalt(i)`, die den aktuellen Inhalt von Eimer `i` liefert (also eine ganze Zahl). Dabei sei Eimer `0` der 8-Liter Eimer, Eimer `1` der 5-Liter Eimer, und Eimer `2` der 3-Liter Eimer. Falls der Parameter `i` keine gültige Eimer-Nummer ist, soll `-1` geliefert werden. Aufrufe könne so aussehen:

  ```
  System.out.println(z0.inhalt(0)); // Druckt 8
  System.out.println(z0.inhalt(1)); // Druckt 0
  System.out.println(z0.inhalt(5)); // Druckt -1
  ```

- Die Methode `gesucht()` (ohne Parameter), die `true` liefert, wenn einer der drei Eimer genau 1 Liter Farbe enthält (die Füllung der beiden anderen Eimer spielt keine Rolle). Sonst soll sie `false` liefern. Der Wahrheitswert soll als Funktionswert/Rückgabewert geliefert werden, und nicht etwa von der Methode selbst ausgedruckt werden. Ausdrucken könnte man ihn dann so:

  ```
  System.out.println(z0.gesucht()); // Druckt false
  ```

- Die Methode `print()` (ohne Parameter und ohne Rückgabewert), die den aktuellen Inhalt der drei Eimer ausgibt (in einer Zeile, durch Komma getrennt, mit einem Zeilenvorschub/umbruch am Ende). Ein Aufruf könnte so aussehen:

  ```
  z0.print(); // Muss Folgendes drucken: 8,0,0
  ```

Der Konstruktor und die drei Methoden sollen von außen aufrufbar sein, uns zwar auch von außerhalb des Paketes. Entsprechend muß die Klasse auch von außerhalb des Paketes aus benutzbar sein. Nichts sonst darf von außen zugreifbar sein (auch nicht von anderen Klassen des gleichen Paketes aus). Sie dürfen beliebige Attribute und Hilfsmethoden verwenden, die aber nur von innerhalb dieser Klasse aus zugreifbar sein dürfen.

Platz für die Lösung von Aufgabe 1 (Klasse `Zustand`):

Beachten Sie, dass auch für ein fehlendes Semikolon oder eine fehlende Klammer ein halber Punkt abgezogen werden kann. Versuchen Sie also, Syntaxfehler zu vermeiden. Bemühen Sie sich außerdem um Verständlichkeit und guten Programmierstil. Es können auch für schlechten Stil Punkte abgezogen werden! Ebenso für unnötig komplizierte Lösungen.

Aufgabe 2 (Programmierung: Array) 12 Punkte

Als nächstes benötigen Sie eine Klasse `ZustandsListe`, die eine Liste von Objekten der Klasse `Zustand` verwaltet. Dabei sollen Duplikate automatisch eliminiert werden. Wenn man z.B. erst den 5-Liter Eimer aus dem 8-Liter Eimer füllt, und dann die 5 Liter Farbe zurück in den 8-Liter Eimer gießt, erhält man den gleichen Zustand. Das ist uninteressant. Sie wollen alle erreichbaren Zustände konstruieren, und sehen, ob einer darunter ist, der die Bedingung erfüllt. Für die eigentliche Zustandskonstruktion reicht die Zeit leider nicht, aber bitte programmieren Sie eine Klasse `ZustandsListe` mit folgender Schnittstelle:

- Ein Konstruktor ohne Parameter, der die Liste auf die leere Menge initialisiert. Intern muss er ein Array anfordern, das Platz für 100 `Zustands`-Objekte bietet (Sie können voraussetzen, das niemals mehr als 100 Zustände konstruiert werden). Eine Beispiel-Anwendung des Konstruktors ist:

  ```
  ZustandsListe liste = new ZustandsListe();
  ```

- Eine Methode `add(z)`, der ein Objekt der Klasse `Zustand` übergeben wird. Die Methode hat keinen Rückgabewert. Es ist auch möglich, beim Aufruf eine `null`-Referenz zu übergeben, solche Aufrufe sollen einfach ignoriert werden, d.h. die Liste bleibt unverändert. Ansonsten muss die Methode prüfen, ob es ein Objekt mit den gleichen Füllungsmengen der drei Eimer in der Liste schon gibt. Auch in diesem Fall soll die Liste unverändert bleiben. Ansonsten (es gibt den Zustand noch nicht), soll er hinten an die Liste angehängt werden. Beachten Sie, dass es unterschiedliche `Zustand`-Objekte geben kann, bei denen die Methode `inhalt(i)` für alle drei Eimer (`i=0`, `i=1`, `i=2`) den gleichen Wert liefert. Diese Objekte repräsentieren dann die gleichen Füllungsstände, und es soll nur das erste solche Objekt eingetragen werden. Beispiel-Aufrufe sind:

  ```
  Zustand z0 = new Zustand(8, 0, 0);
  Zustand z1 = new Zustand(8, 0, 0); // gleicher Zustand!
  liste.add(null); // Aendert nichts
  liste.add(z0);   // z0 wird eingetragen
  liste.add(z1);   // Aendert nichts
  ```

- Eine Methode `size()` ohne Parameter, die die Anzahl Elemente in der Liste liefert. Z.B. enthält die Liste nach den obigen drei `add(...)`-Aufrufen nur das eine Element `z0`, so dass folgende Ausgabe-Anweisung `1` druckt:

  ```
  System.out.println(liste.size()); // 1
  ```

- Eine Methode `get(i)` mit einem ganzzahligen Parameter `i`, der das `i`-te Element der Liste liefert (also ein Objekt der Klasse `Zustand`). Wie bei Arrays soll der Index `0` das erste Element liefern, und `liste.size() - 1` das letzte. Falls der Index nicht in diesem Bereich ist, soll `null` geliefert werden.

  ```
  if(liste.get(0) == z0)
      System.out.println("ok"); // Wird ausgefuehrt
  ```

Die Anwendung wird in einer Bonus-Aufgabe am Ende der Klausur vervollständigt. Dort finden sich auch weitere Beispielaufrufe. Zugriffsrechte wie bei Aufgabe 1. Sie können nur die in Aufgabe 1 genannten Methoden der Klasse `Zustand` verwenden.

Platz für die Lösung (Klasse `ZustandsListe`):

Aufgabe 3 (Wertausdrücke, Bedingung, Referenztyp) 5 Punkte

Was gibt dieses Programm aus?

```
class Aufg3 {
    public static void main(String[] args) {
    // a)
        System.out.print("a) ");
        System.out.println(10 - 4 - 2);
    // b)
        System.out.print("b) ");
        System.out.println(13 % 5 + 20 / 3);
    // c)
        System.out.print("c) ");
        int n = 17;
        int m = 100;
        System.out.println(n < 5 && m >= n);
    // d)
        System.out.print("d) ");
        String s = "1";
        String t = "2";
        System.out.println(s + t + 5);
    // e)
        System.out.print("e) ");
        int[] a = new int[3];
        a[0] = 1;
        int[] b = a;
        b[0] = 2;
        System.out.println(a[0]);
    }
}
```

a) _____

b) _____

c) _____

d) _____

e) _____

Aufgabe 4 (Methoden, while, if) 3 Punkte

Was gibt dieses Programm aus?

```
public class Aufg4 {
    static int a(int i, int n) {
        while(i < n) {
            i = i + 1;
            if(i == 3)
                return 10;
        }
        return i;
    }

    static int b(int n, int m) {
        int i = 0;
        int j = 1;
        while(i > n)
            n = n * 2;
        if(i == n && i < m)
            j = j * 3;
        else if(i < n && i == m)
            j = j * 5;
        else
            j = j * 7;
        return j;
    }

    static int c(int[] arr) {
        int i = 0;
        while(arr[i] != 3)
            i = i + 1;
        return i;
    }

    public static void main(String[] args) {
        System.out.println(a(0, 3));    // a)
        System.out.println(b(3, 5));    // b)
        int[] x = { 1, 2, 3, 4, 5 };
        System.out.println(c(x));       // c)
    }
}
```

a) _____

b) _____

c) _____

Aufgabe 5 (Klassen, Parameter, Verschattung(!)) 3 Punkte

Was gibt dieses Programm aus?

```
public class Aufg5 {
    int a;
    static int b = 1;

    Aufg5() { a = 3; }

    Aufg5(int n) { b = n; } // Achtung! b, nicht a

    static void s(Aufg5 x, int b) {
        x.a += 4;
        b   += 4;
    }

    void m(int n) {
        a++;
        b++;
        n++;
    }

    public static void main(String[] args) {
        Aufg5 o = new Aufg5();
        Aufg5 x = new Aufg5(5);
        int n = 6;
        s(o, b);
    // a)
        System.out.println("a) " + o.a);
    // b)
        System.out.println("b) " + b);
        b = 7;
        o.m(n);
        x.m(n);
    // c)
        System.out.print("c) ");
        System.out.println(100*x.a + 10*b + n);
    }
}
```

a) _____

b) _____

c) _____

Aufgabe 6 (Exceptions) 3 Punkte

Was gibt dieses Programm aus?

```
class Ober6 { }
class Unter6 extends Ober6 { }

public class Aufg6 {
    static int[] a;
    public static void main(String[] args) {
        Ober6 o = new Ober6();
        Unter6 u = new Unter6();
        int[] b = new int[10];
    // a)
        System.out.print("a) ");
        try {
             a[0] = 1;
             System.out.println("ok");
        } catch(Exception e) {
             System.out.println(e);
        }
    // b)
        System.out.print("b) ");
        try {
             b[10] = 0;
             System.out.println("ok");
        } catch(Exception e) {
             System.out.println(e);
        }
    // c)
        System.out.print("c) ");
        try {
             Ober6 x = u;
             System.out.println("ok");
        } catch(Exception e) {
             System.out.println(e);
        }

    }
}
```

a) ☐ ok ☐ ArithmeticException ☐ NullPointerException
☐ ArrayIndexOutOfBoundsException ☐ ClassCastException

b) ☐ ok ☐ ArithmeticException ☐ NullPointerException
☐ ArrayIndexOutOfBoundsException ☐ ClassCastException

c) ☐ ok ☐ ArithmeticException ☐ NullPointerException
☐ ArrayIndexOutOfBoundsException ☐ ClassCastException

Aufgabe 7 (Vererbung) 3 Punkte

Was gibt dieses Programm aus?

```
class Ober7 {
    Ober7()  { System.out.print("A"); }
    void f() { System.out.print("B"); }
    void g() { f(); }
}

class Unter7 extends Ober7 {
    Unter7() { System.out.print("C"); }
    void f() { System.out.print("D"); }
}

public class Aufg7 {
    public static void main(String[] args) {

    // a)
        System.out.print("a) ");
        Ober7 o = new Ober7();
        Unter7 u = new Unter7();
        Ober7 x = new Unter7();
        System.out.println();

    // b)
        System.out.print("b) ");
        o.f();
        u.f();
        x.f();
        System.out.println();

    // c)
        System.out.print("c) ");
        o.g();
        u.g();
        x.g();
        System.out.println();
    }
}
```

a) _____

b) _____

c) _____

Aufgabe 8 (Fehlersuche) 3 Punkte

Das folgende Programm enthält (mindestens) 4 Fehler. Finden Sie drei davon und geben Sie jeweils die Zeile und die Fehlermeldung des Compilers an. Falls Sie mehr als drei Fehler angeben, werden nur die ersten drei gewertet (es gibt keine Extrapunkte).

```
 1  public class Aufg8 {
 2      private int n = 0;
 3      void f() {
 4          System.out.println("f");
 5      }
 6      public static void main(String[] args) {
 7          int n = 0;
 8          boolean b = (n > 0);
 9          Aufg8 x = new Aufg8();
10          f();
11          if(x.n < 0 || b)
12              n = "12";
13          if(n = 0) {
14              String s = "x.a";
15              System.out.println(s + n);
16          }
17          else if(s != null)
18              while(n++ < 100) System.out.print('.');
19      }
20  }
```

Wählen Sie Fehlermeldungen aus folgender Liste und kreuzen Sie unten das entsprechende Kästchen an. Es kann mehrere Fehler (in verschiedenen Zeilen) mit der gleichen Fehlermeldung geben. Pro Zeile gibt es nur einen Fehler. Es gibt keine Folgefehler.

A: `';' expected`

B: `'else' without 'if'`

C: `cannot find symbol, symbol: ..., location: ...`

D: `... is already defined in main(java.lang.String[])`

E: `incompatible types, found: ..., required: ...`

F: `variable ... might not have been initialized`

G: `... has private access in ...`

H: `non-static method ... cannot be referenced from a static context`

I: `missing return statement`

	A	B	C	D	E	F	G	H	I
a) Zeile: _____	☐	☐	☐	☐	☐	☐	☐	☐	☐
b) Zeile: _____	☐	☐	☐	☐	☐	☐	☐	☐	☐
c) Zeile: _____	☐	☐	☐	☐	☐	☐	☐	☐	☐

Aufgabe 9 (Bonus-Aufgabe) 3 Extra-Punkte

Beachten Sie, dass es für diese Aufgabe nur wenige Punkte gibt, und auch nur bei mehr oder weniger perfekter Lösung. Bei ernsteren Fehlern gibt es einfach 0 Punkte. Sehr umständliche oder schwer zu verstehende Lösungen werden nicht korrigiert (0 Punkte).

Deshalb sollten Sie diese Bonusaufgabe erst bearbeiten, wenn Sie mit den anderen Aufgaben fertig sind. Die Erzeugung aller möglichen Zustände soll mit folgendem Hauptprogramm ausgeführt werden, in dem noch eine Methode `umfuellen` fehlt:

```
public static void main(String[] args) {
    ZustandsListe liste = new ZustandsListe();
    liste.add(new Zustand(8, 0, 0));
    for(int n = 0; n < liste.size(); n++) {
        Zustand z = liste.get(n);
        z.print();
        // Alle Moeglichkeiten, von Eimer i in Eimer j umzufuellen,
        // werden durchprobiert, und die neuen Zustaende in die
        // Liste eingetragen (bis keine neuen Zustaende mehr):
        for(int i = 0; i < 3; i++)
            for(int j = 0; j < 3; j++)
                if(i != j)
                    liste.add(umfuellen(z, i, j));
    }
    System.out.println("Insgesamt " + liste.size() + " Zustaende.");
    for(int n = 0; n < liste.size(); n++) {
        if(liste.get(n).gesucht()) {
            System.out.println("Raetsel ist loesbar!");
            return;
        }
    }
    System.out.println("Raetsel nicht loesbar");
}
```

Schreiben Sie eine statische Methode `umfuellen(z, i, j)`, die einen neuen Zustand erzeugt, der sich aus Zustand `z` durch Umfüllen von Eimer `i` in Eimer `j` ergibt. Dazu müssen Sie feststellen, wieviel Platz noch in Eimer `j` ist, also wie viele Liter noch bis zu seiner Kapazitätsgrenze hineinpassen. Dann nehmen Sie das Minimum von dieser Zahl und dem Inhalt von Eimer `i`. Dieses Minimum ist die Anzahl der Liter, die von Eimer `i` in Eimer `j` umgefüllt werden können. Falls das `0` Liter sind, das Umfüllen also gar nicht möglich ist, liefern Sie die `null`-Referenz (das spart den Duplikat-Test in `liste.add(...)`) Ansonsten erzeugen Sie ein neues Objekt der Klasse `Zustand`, das sie mit den neuen Füllungsständen initialisieren. Ein möglicher Aufruf könnte so aussehen:

```
Zustand neu = umfuellen(z0, 0, 2);
```

Dies würde einen Zustand liefern, in dem Eimer 0 (der 8-Liter Eimer) noch 5 Liter enthält, Eimer 1 (der 5-Liter Eimer) unverändert leer ist, und Eimer 2 (der 3-Liter Eimer) nun 3 Liter enthält.

Das Hauptprogramm und die Methode `umfuellen(...)` stehen in einer dritten Klasse, Sie können also auf Klassen `Zustand` und `ZustandsListe` nur über die oben definierten Schnittstellen zugreifen.

Die Korrektheit der Argumentwerte von `umfuellen(...)` brauchen Sie nicht zu prüfen.

Platz für die Lösung (Statische Methode `umfuellen(z, i, j)`):