

Aufgaben für das Fach Programmieren 2 (PR2-MB) im WS07/08**Inhaltsverzeichnis**

Aufgabe 1: Ein kleiner Quiz über den Stoff des ersten Semesters (PR1).....	3
Aufgabe 2: Rekursion und Iteration.....	4
Aufgabe 3: Formattierungen mit printf.....	6
Aufgabe 4: XML-Dokumente erzeugen, validieren, bearbeiten etc.....	8
Aufgabe 5: Eine Collection-Klasse namens MeineSammlung entwickeln und testen.....	10
Aufgabe 6: Die Standard-Klasse HashSet testen.....	11
Aufgabe 7: Eine Set-Klasse namens MeinHashSet entwickeln und testen.....	12
Aufgabe 8: Ein kleiner Dialog mit einem Fibonacci-Server.....	13
Aufgabe 9: Ein reflektives Programm.....	14
Aufgabe 10: Zeitmessungen an Standardsammlungen.....	15

Regeln für die Bearbeitung und Bewertung der Aufgaben

1. Für die folgenden Aufgaben sollen Sie in Arbeitsgruppen zu je 2 (evtl. 3) Personen Lösungen entwickeln (während der Übungstermine und teilweise auch zu Hause). Jede Arbeitsgruppe muss für sich einen Gruppennamen wählen.

2. Jede Arbeitsgruppe muss (im Laufe des Semesters) für jede Aufgabe eine *akzeptable Lösung* abgeben, sonst dürfen die Mitglieder der Gruppe nicht an der Hauptklausur am Ende des Semesters (und auch nicht an der Nachklausur am Beginn des nächsten Semesters) teilnehmen. Eine Aufgabe ist akzeptabel, wenn sie deutlich zeigt, dass Sie sich ernsthaft mit ihr befasst haben. Auch eine noch nicht ganz korrekte Lösung kann akzeptabel sein (sprechen Sie mit der BetreuerIn Ihrer Übungstermine).

3. Wenn Ihre Gruppe alle Lösungen pünktlich abgibt, bekommen Sie dafür 10 Aufgabenpunkte (die zu den in der Hauptklausur erreichten Klausurpunkten addiert werden). Für jede unpünktlich abgegebene Lösungen werden Ihnen 2 Aufgabenpunkte abgezogen (aber nur bis 0, negative Punktzahlen gibt es nicht). Bei der Nachklausur werden die Aufgabenpunkte *nicht* mehr berücksichtigt.

4. Beim Abgeben einer Lösung müssen alle Mitglieder der Gruppe persönlich anwesend sein (damit die BetreuerIn Ihrer Übungstermine Ihnen ein paar Fragen zu der Lösung stellen kann).

5. Ihre Lösungen müssen alle mit einem 2-zeiligen Kopf beginnen, etwa so:

```
// Gruppe: Los Olvidados, Mitglieder: Dorothea Kupferberg, Ali Jasim.  
// Lösung zur Aufgabe 3, PR1-MB, SS07.
```

Solche einheitlichen Köpfe sollen Ihrer BetreuerIn die (nicht besonders angenehme) Arbeit des Überprüfens und Verwaltens Ihrer Lösungen etwas erleichtern.

6. Programmtexte müssen regelmäßig eingerückt sein (wie die Beispielprogramme im Buch „Java ist eine Sprache“) und jede Zeile muss (in der Seitenansicht und beim Ausdrucken) eine eigene *Zeilen-Nummer* haben.

Hinweis: Im Editor TextPad können Sie Zeilen-Nrn durch folgende Einstellung *ausdrucken* lassen: Menü Konfiguration, Einstellungen, auf das Pluszeichen vor Dokumentenklasse klicken, auf das Pluszeichen vor Java klicken, Drucken, Zeilennummern. Die *Zeilen-Nrn auf dem Bildschirm* kann man unabhängig davon ein- bzw. ausschalten (Menü Konfiguration, Einstellungen, Ansicht, Zeilennummern).

Bevor Sie (im TextPad) eine Seite ausdrucken lassen, sollten Sie sie unbedingt auf dem Bildschirm anzeigen lassen (indem Sie auf den Knopf Seitenansicht klicken) und sorgfältig überprüfen. Wenn eine Zeile keine eigene Zeilen-Nummer hat, so ist sie ein Teil einer *zu langen Zeile* (die vom Druckertreiber automatisch umgebrochen wurde, häufig an sehr unglücklich gewählten Stellen, weil der Druckertreiber keine Ahnung von Java-Programmen hat). In so einem Fall müssen Sie die zu lange Zeile von Hand umbrechen (natürlich an gut gewählten Stellen), und damit zeigen, dass Sie mehr Ahnung von Java-Programmen haben als der Druckertreiber.

7. Ihre Programmtexte müssen mindestens folgende *Kommentare* enthalten:

- Am Anfang jeder *Klasse* ein Kommentar, der den Sinn der Klasse (zumindest kurz) beschreibt.
- Am Anfang jeder *Methode* ein Kommentar, der (möglichst kurz) beschreibt, was die Methode macht.

Aufgabe 1: Ein kleiner Quiz über den Stoff des ersten Semesters (PR1)

1. Das Entwickeln und Ausführen von Programmen kann man als ein *Rollenspiel* auffassen. Nennen Sie möglichst viele der *Rollen* in diesem Spiel (mindestens die drei wichtigsten).
2. Welche charakteristischen *Tätigkeiten* sind den einzelnen Rollen zugeordnet?
3. Nennen Sie fünf wichtige ("die fünf wichtigsten") *Grundkonzepte* von Programmiersprachen.
4. Was ist eine *Variable*? Erläutern Sie die wichtigsten Eigenschaften einer Variable. Was kann der Programmierer bzw. der Ausführer mit einer Variablen alles machen?
5. Was ist ein *Typ*?
6. Was ist ein *Modul*? Erläutern Sie kurz die wichtigsten Eigenschaften eines Moduls.
7. Was ist eine *Klasse*?
8. Was ist eine *abstrakte Klasse*? Erläutern Sie, was man in einer abstrakten Klasse *vereinbaren* kann.
9. Was ist eine *Schnittstelle*? Erläutern Sie, was eine Schnittstelle *enthalten* kann.
10. Es gibt unüberschaubar viele Programmiersprache mit zahllosen Befehlen darin. Im Grunde genommen gibt es aber nur *3 Arten* von Befehlen (die der Programmier dem Ausführer geben kann). Wie heißen diese 3 Arten von Befehlen?
11. Vereinbaren Sie (in Java) eine Variable namens `otto` vom Typ `int` mit dem Anfangswert 17. Wie übersetzt man diese Vereinbarung (diesen Befehl des Programmierers an den Ausführer) ins Deutsche?
12. Befehlen Sie dem Java-Ausführer einen Wert zu berechnen, der um 3 grösser ist als das Quadrat von `otto`. Wie übersetzt man diesen Befehl ins Deutsche?
13. Befehlen Sie dem Java-Ausführer, er solle den Wert, der um 3 grösser ist als das Quadrat von `otto`, der Variablen `otto` zuweisen. Wie übersetzt man diesen Befehl ins Deutsche?
14. In Java kann man (wie in den meisten Programmiersprachen) eine Folge von Befehlen zu einem *Unterprogramm* (oder: zu einer *Methode*) zusammenfassen und mit einem Namen versehen. Welche *Vorteile* hat das typischerweise?
15. Es gibt *zwei Arten von Unterprogrammen* (oder: von *Methoden*). Wie heißen Unterprogramme der einen (bzw. der anderen) Art?
16. Was für eine *Art von Befehl* (siehe 10.) ist es, wenn man ein Unterprogramm der einen Art (bzw. der anderen Art) *aufruft*?
17. Eine Klasse enthält *Elemente* (engl. members). Diese Elemente kann man auf drei verschiedene Weisen in "Gruppen" einteilen. Wie nennt man die Elemente der verschiedenen "Gruppen"?
18. In Java unterscheidet man *zwei Arten von Typen*. Wie heißen Typen der einen bzw. der anderen Art? Nennen Sie ein paar Typen der einen bzw. der anderen Art als Beispiele.
19. In Java unterscheidet man *drei Arten von Referenztypen*. Wie heißen Typen der drei Arten allgemein? Nennen Sie von jeder Art ein paar Referenztypen als Beispiele.
20. In Java unterscheidet man (wie in den meisten Programmiersprachen) *zwei Arten von Anweisungen*. Wie heißen Anweisungen der einen bzw. der anderen Art? Nennen Sie ein paar Anweisungen der einen bzw. der anderen Art als Beispiele.
21. Woraus besteht ein Java-Programm (oder: Was sind die größten sinnvollen Teile eines Java-Programms)?
22. Welche Klassen gehören zu einem Java-Programm namens `Otto`?
23. Wann werden die einzelnen Klassen eines Java-Programms geladen?
24. Was ist ein Sammlungsobjekt? Nennen Sie Beispiele.
25. Was ist ein Grabo-Objekt?
26. Was ist ein Behälterobjekt? Nennen Sie Beispiele.
27. Die wichtigste positive Eigenschaft der Programmiersprache Java?

Aufgabe 2: Rekursion und Iteration

Betrachten Sie die Datei `ReIter01.java` (siehe unten). Fügen Sie am Anfang der Methoden `iter01`, `iter02`, `reku03`, `reku04` und `reku05` je einen Anfangskommentar (AKO) ein, und zwar jeweils am Anfang des *Rumpfes*, d. h. nach der öffnenden geschweiften Klammer `{`. Dieser AKO soll die Frage „*Was macht dieses Unterprogramm?*“ möglichst einfach und verständlich beantworten (als simples Beispiel siehe das Upro `schritt01`). Hier ein paar Regeln dazu, wie man solche AKOs gestalten sollte:

Regel 1: Der AKO beginnt normalerweise mit einem Verb ("*Gibt ... aus*", "*Formatiert ...*" etc.).

Regel 2: Der AKO einer Funktion beginnt normalerweise mit "*Liefert ...*" (oder "*Gibt ... zurück*").

Regel 3: Im AKO kommt jeder *Parametername* mindestens einmal vor.

Regel 4: Der AKO beschreibt, *was* das Upro leistet, nicht *wie* es diese Leistung erbringt (ob durch Aufrufe weiterer Methoden oder durch andere Befehle, ob iterativ oder rekursiv etc.).

Regel 5: Im AKO erwähnt man nichts, was schon davor ("in der ersten Zeile der Methode") steht.

Z. B. schreibt man statt "Gibt den übergebenen String-Parameter namens `karl_heinz` aus und schiebt unmittelbar anschließend den Bildschirmzeiger (cursor) an den Anfang der nächsten Zeile vor" kürzer "Gibt `karl_heinz` und einen Zeilenwechsel aus".

Unterprogramme mit gleicher Nr (z. B. `iter01` und `reku01`, oder `reku03` und `iter03` etc.) sollen *exakt das Gleiche* leisten (aber auf verschiedene Weise: iterativ bzw. rekursiv). Ersetzen Sie die „unsinnigen“ Rumpfe der Upros `reku01`, `reku02`, `iter03`, `iter04` und `iter05` entsprechend.

```

1 // Datei ReIter01.java
2 ...
3 class ReIter01 {
4     // -----
5     static private void schritt01(char char01) {
6         // Gibt char01 zur Standardausgabe aus.
7         AM.p(char01);
8     } // schritt01
9     // ----- Aufgabe 01
10    static public void iter01(final int N) {
11        for (int i=1; i<=N; i++) {
12            schritt01('X');
13        }
14        AM.pln();
15    } // iter01
16    // ----- Loesung 01
17    static public void reku01(final int N) {
18        // Ersetzen Sie diesen "unsinnigen" Rumpf
19        AM.pln("reku01, unsinniges Ergebnis!");
20    } // reku01
21    // ----- Aufgabe 02
22    static public void iter02(final int N) {
23        int n = N;
24        while (n > 0) {
25            char ziff = (char) (n%2 + '0');
26            schritt01(ziff);
27            n = n/2;
28        }
29        AM.pln();
30    } // iter02;
31    // ----- Loesung 02
32    static public void reku02(final int N) {
33        // Ersetzen Sie diesen "unsinnigen" Rumpf
34        AM.pln("reku02, unsinniges Ergebnis!");
35    } // reku02

```

```

36 // ----- Aufgabe 03
37 static public void reku03(final int N) {
38     char ziff = (char) (N%2 + '0'); // "Rechteste" Binaerziffer
39     if (N < 0) {
40         schritt01('-');
41         reku03(-N);
42     } else if (N == 0 || N == 1) { // Hoechstwertige Ziffer
43         schritt01(ziff);
44         return;
45     } else {
46         reku03(N/2);
47         schritt01(ziff);
48     }
49 } // reku03
50 // ----- Loesung 03
51 static public void iter03(final int N) {
52     // Ersetzen Sie diesen "unsinnigen" Rumpf
53     AM.p("iter03, unsinniges Ergebnis!");
54 } // iter03
55 // ----- Aufgabe 04
56 static public String reku04(String s) {
57     if (s.length() <= 1) {
58         return s;
59     } else {
60         char  letzt      = s.charAt (    s.length()-1);
61         String ohneLetzt = s.substring(0, s.length()-1);
62         return letzt + reku04(ohneLetzt);
63     }
64 } // reku04
65 // ----- Loesung 04
66 static public String iter04(String s) {
67     // Ersetzen Sie diesen "unsinnigen" Rumpf
68     return "iter04, unsinniges Ergebnis!";
69 } // iter04
70 // ----- Aufgabe 05
71 static public int reku05(int[] r1, int[] r2) {
72     return reku05a(r1, r2, 0);
73 }
74
75 static private int reku05a(int[] r1, int[] r2, int index) {
76     if (index >= r1.length || index >= r2.length) {
77         return r1.length - r2.length;
78     } else if (r1[index] != r2[index]) {
79         return r1[index] - r2[index];
80     } else {
81         return reku05a(r1, r2, index+1);
82     }
83 } // reku05a
84 // ----- Loesung 05
85 static public int iter05(int[] r1, int[] r2) {
86     // Ersetzen Sie diesen "unsinnigen" Rumpf
87     return -999999999; // unsinniges Ergebnis
88 } // iter05
89 // -----
90 static public void main(String[] _) {
91     ...
92 } // main
93 // -----
94 } // class ReIter01

```

Die "unvollständige" Datei `ReIter01.java` finden Sie am üblichen Ort. *Testen* Sie Ihre Lösung, indem Sie geeignete Befehle in die `main`-Methode schreiben und ausführen lassen.

Aufgabe 3: Formattierungen mit printf

Zur Lösung dieser Aufgabe sollten Sie sich mit den Methoden `printf` (alias `format`) in den Klassen `PrintStream` und `PrintWriter` vertraut machen (z.B. mit dem `printfApplet` auf der Netzseite www.tfh-berlin.de/~grude).

Ausgangspunkt dieser Aufgabe ist ein "unvollständiges" Programm `PrintfAufgabe`, welches eine *unschöne* Ausgabe produziert. Sie sollen daraus ein "vollständiges" Programm machen, indem Sie im Quelltext nur 2 Zeilen (die Vereinbarungen der Formatstrings `iFormat` und `dFormat`) ein bisschen verändern. Ihr "vollständiges" Programm soll eine "schöne" (genau vorgegebene) Ausgabe produzieren. Es folgt hier der Quelltext des unvollständigen Programms (beachten Sie auch den langen Kommentar am Ende):

```

1 // Datei PrintfAufgabeUnv.java
2 /* -----
3 Dieses Programm gibt ein paar Zahlen (der Typen Integer und Double)
4 formatiert zur Standardausgabe aus. "Unv" steht fuer "unvollstaendig".
5 ----- */
6 import java.util.Locale;
7
8 class PrintfAufgabeUnv {
9     // -----
10    static public void main(String[] _) {
11        printf("PrintfAufgabeUnv: Jetzt geht es los!\n");
12
13        String      iFormat = "%d Euro\n"; // Soll geaendert werden <---
14        String      dFormat = "%f Euro\n"; // Soll geaendert werden <---
15
16        Integer[]   iReih   = {          +12,   -45678, +90123,         -3};
17        Double []   dReih   = {+1234567.89, -2222.22, +3.333, -4444.5555};
18
19        Locale      deLoc   = Locale.getDefault();
20        Locale      ukLoc   = Locale.UK;
21        Locale      frLoc   = Locale.FRANCE;
22
23        Number [][] nurr    = {iReih,   dReih};
24        String []   foser   = {iFormat, dFormat};
25        Locale []   locr    = {deLoc,   ukLoc, frLoc};
26
27        for (int ind=0; ind<nurr.length; ind++) {
28            Number[] nr = nurr[ind]; // Eine Reihung von Number-Objekten
29            String   f = foser[ind]; // Ein Format-String fuer printf
30            for (Locale loc: locr) { // Eine Lokalitaet
31                printf("-----\n");
32                for (Number n : nr) { // Ein Number-Objekt
33                    printf(loc, f, n);
34                } // for n
35            } // for loc
36        } // for ind
37
38        printf("-----\n");
39        printf("PrintfAufgabeUnv: Das war's erstmal!\n");
40    } // main
41    // -----
42    // Zwei Methoden mit kurzen Namen:
43    static void printf(String f, Object... v)
44        {System.out.printf(f, v);}
45    static void printf(Locale loc, String f, Object... v)
46        {System.out.printf(loc, f, v);}
47    // -----
48 } // class PrintfAufgabeUnv
49 /* -----+-----+

```

50	Ist-Ausgabe des Programms	Soll-Ausgabe des Programms
51	PrintfAufgabe:	PrintAufgabe:
52	-----	-----
53	PrintfAufgabe: Jetzt geht es los!	PrintfAufgabe: Jetzt geht es los!
54	-----	-----
55	12 Euro	+12 Euro
56	-45678 Euro	-45.678 Euro
57	90123 Euro	+90.123 Euro
58	-3 Euro	-3 Euro
59	-----	-----
60	12 Euro	+12 Euro
61	-45678 Euro	-45,678 Euro
62	90123 Euro	+90,123 Euro
63	-3 Euro	-3 Euro
64	-----	-----
65	12 Euro	+12 Euro
66	-45678 Euro	-45 678 Euro
67	90123 Euro	+90 123 Euro
68	-3 Euro	-3 Euro
69	-----	-----
70	1234567,890000 Euro	1.234.567,89 Euro
71	-2222,220000 Euro	-2.222,22 Euro
72	3,333000 Euro	3,33 Euro
73	-4444,555500 Euro	-4.444,56 Euro
74	-----	-----
75	1234567.890000 Euro	1,234,567.89 Euro
76	-2222.220000 Euro	-2,222.22 Euro
77	3.333000 Euro	3.33 Euro
78	-4444.555500 Euro	-4,444.56 Euro
79	-----	-----
80	1234567,890000 Euro	1 234 567,89 Euro
81	-2222,220000 Euro	-2 222,22 Euro
82	3,333000 Euro	3,33 Euro
83	-4444,555500 Euro	-4 444,56 Euro
84	-----	-----
85	PrintfAufgabe: Das war's erstmal!	PrintfAufgabe: Das war's erstmal!
86	-----	----- */

Aufgabe 4: XML-Dokumente erzeugen, validieren, bearbeiten etc.

Betrachten Sie folgende Dokumenten-Typ-Definition:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- -----
3   Datei stundenplan.dtd:
4   Document Type Definition fuer die Dateien sp01.xml, sp02.xml etc.
5 ----- -->
6 <!ELEMENT stundenplan (kopf, lv+)>
7
8 <!ELEMENT kopf (semester, studiengang, name?)>
9 <!ELEMENT semester (#PCDATA)>
10 <!ELEMENT studiengang (MB | TB | MM | EM)>
11 <!ELEMENT name (#PCDATA)>
12
13 <!-- Die folgenden Elemente muessen immer leer sein! -->
14 <!ELEMENT MB EMPTY>
15 <!ELEMENT TB EMPTY>
16 <!ELEMENT MM EMPTY>
17 <!ELEMENT EM EMPTY>
18
19 <!-- "lv" wie "Lehrveranstaltung" -->
20 <!ELEMENT lv (#PCDATA)>
21
22 <!ATTLIST lv
23   kuerzel ID #REQUIRED
24   w_tag (mo | di | mi | do | fr | sa | so) #REQUIRED
25   block (b1 | b2 | b3 | b4 | b5 | b6 | b7 | bx) #REQUIRED
26   dauer ( 1 | 2 | 3 | dx) "1"
27 >

```

Erstellen Sie (mit einem Editor) ein XML-Dokument, welches entsprechend der obige DTD *gültig* (engl. valid) ist. Dieses Dokument soll eine Dokumenten-Typ-Deklaration enthalten, etwa so:

```
<!DOCTYPE stundenplan SYSTEM "stundenplan.dtd">
```

und mindestens 5 Lehrveranstaltungen (lv-Elemente) beschreiben. Speichern Sie das Dokument in einer Datei namens sp01.xml.

Programmieren Sie eine Klasse namens `LoesungXml01`, die eine `main`-Methode und eine Klassenmethode mit der folgenden Spezifikation enthält:

```

28   static org.jdom.Document createDoc() {
29       // Erzeugt ein JDOM-Dokument-Objekt, welches moeglichst genau mit der
30       // Datei sp01.xml uebereinstimmt.

```

Die `main`-Methode der Klasse `LoesungXml01` soll folgendes bewirken:

1. Die Datei `sp01.xml` wird als ein DOM-Dokument (des Typs `org.w3c.dom.Document`) eingelesen und gegen die Dokumenten-Typ-Definiton (DTD) in der Datei `stundenplan.dtd` validiert. Das DOM-Dokument und das Ergebnis der Validierung (`true` bzw. `false`) werden ausgegeben.
2. Ein JDOM-Dokument (des Typs `org.jdom.Document`) wird (mit `createDoc`) erzeugt, zum Bildschirm ausgegeben und in die XML-Datei `sp02.xml` geschrieben.
3. Die Datei `sp02.xml` wird als ein DOM-Dokument (des Typs `org.w3c.dom.Document`) eingelesen und gegen die Dokumenten-Typ-Definiton (DTD) in der Datei `stundenplan.dtd` validiert. Das DOM-Dokument und das Ergebnis der Validierung (`true` bzw. `false`) werden ausgegeben.

In Ihrem Programm `LoesungXml01` dürfen Sie Methoden aus den Moduln `XmlJStd` und `XmlJdom` aufrufen. Die Quelldateien dieser Module enthalten ziemlich umfangreiche Kommentare und Erläuterungen.

Freiwilliger Zusatz:

4. Ergänzen Sie in der Klasse `LoesungXml01` eine Methode mit der folgenden Spezifikation:

```
31 static void gibLvsAus(org.jdom.Document jdomDoc) {
32     // Das jdomDoc sollte gueltig bezueglich der DTD stundenplan.dtd
33     // sein. Gibt eine Liste aller Lehrveranstaltungen (lv-Elemente)
34     // aus (zur Standardausgabe), die im Stundenplan jdomDoc beschrieben
35     // werden.
```

Rufen Sie diese Methode (mit einem geeigneten Parameter) in der `main`-Methode auf, so dass eine Liste von Lehrveranstaltungen zum Bildschirm ausgegeben wird.

Aufgabe 5: Eine Collection-Klasse namens MeineSammlung entwickeln und testen

Als "Rohmaterial" für diese Aufgabe sind zwei Dateien vorgegeben:

In der Datei `MeineSammlung.java` wird eine Collection-Klasse namens `MeineSammlung` vereinbart, in der aber viele Methoden (`contains`, `containsAll`, `clear`, `size`, `isEmpty`, `equals`, `toArray`, `toString`, und in einer inneren Klasse namens `Iterator` die Methoden `hasNext` und `next`) nur einen unsinnigen `return`-Befehl als Rumpf haben. Sie sollen diese *unsinnigen Methodenrümpfe* durch *richtige Rümpfe* ersetzen. Jeder dieser unsinnigen Rümpfe ist mit dem Kommentar `// MUSS ERGAENZT ODER ERSETZT WERDEN` gekennzeichnet.

Die Datei `MeineSammlungJut.java` enthält ein JUnit-Programm zum Testen der Klasse `MeineSammlung`. Dieses Testprogramm enthält 16 Test-Methoden, von denen aber anfangs 10 auskommentiert ("deaktiviert") sind.

Wenn man die beiden Quelldateien (`MeineSammlung.java` und `MeineSammlungJut.java`) compiliert und das Testprogramm `MeineSammlungJut` ausführen läßt, sollte es einen grünen Balken zeigen.

Sie sollen die Klasse `MeineSammlung` schrittweise fertig entwickeln. Jeder Schritt besteht aus zwei Teilschritten:

Teilschritt-1: Aktivieren Sie in der Quelldatei `MeineSammlungJut.java` die nächste deaktivierte ("auskommentierte") Testmethode (z.B. die Testmethode `testContains01` oder die Testmethode `testContainsAll01` etc.) und compilieren Sie die Quelldatei.

Teilschritt-2: Ersetzen Sie in der Quelldatei `MeineSammlung.java` den unsinnigen Rumpf der entsprechenden Methode (z.B. den Rumpf der Methode `contains` oder den Rumpf der Methode `containsAll` etc.) durch einen richtigen Rumpf. Lassen Sie dann das Testprogramm `MeineSammlungJut` ausführen. Wenn es einen grünen Balken zeigt, ist dieser Schritt fertig. Ansonsten müssen Sie den Rumpf der aktuellen Methode (und manchmal auch die Rümpfe anderer Methoden, die von der aktuellen Methode aufgerufen werden) so lange verbessern, bis das Testprogramm einen grünen Balken zeigt.

Sie sind fertig, wenn Sie im Testprogramm alle Testmethoden aktiviert ("entkommentarisiert") und in der Datei `MeineSammlung.java` alle unsinnigen Methodenrümpfe durch richtige Rümpfe ersetzt haben und das Testprogramm dann einen grünen Balken zeigt.

Aufgabe 6: Die Standard-Klasse HashSet testen

Schreiben Sie eine JUnit-Klasse namens `HashSetJut01` zum Testen der Java-Standardklasse `HashSet`. Ein „Skelett“ dieser Klasse finden Sie in der Datei `HashSetJut01.java`. Dieses „Skelett“ enthält 17 Testmethoden, von denen aber 16 noch leere Rumpfe haben. Sie sollen diese leeren Rumpfe durch "richtige Rumpfe" ersetzen. Was die einzelnen Testmethoden testen sollen wird in ihren Anfangskommentaren kurz erläutert.

Aufgabe 7: Eine Set-Klasse namens MeinHashSet entwickeln und testen

Entwickeln Sie eine Set-Klasse namens `MeinHashSet`. Auch die als optional gekennzeichneten Methoden (siehe "Java ist eine Sprache", Abschnitt 18.2, S. 446) sollen bei `MeinHashSet`-Objekten "richtig funktionieren" (und nicht nur eine Ausnahme werfen).

Zur Erinnerung: Eine *Hash-Tabelle* ist eine *Reihung* (mit starrer Länge) von *Listen* (mit flexibler Länge). In welche der Listen man eine bestimmte Komponente k einfügt ermittelt man mit Hilfe des Hash-Codes von k (`k.hashCode()`). In Java enthält jedes (!) Objekt eine Methode mit dem Profil `int hashCode()`.

Anstelle von Reihungen sollen Sie `ArrayList`-Objekte verwenden. Als Listen sollen Sie `LinkedList`-Objekte verwenden. Jedes Objekt der Klasse `MeinHashSet` soll also als ein `ArrayList`-Objekt von `LinkedList`-Objekten enthalten, etwa so:

```
ArrayList<LinkedList<K>> all;
```

Die Anzahl der Listen in `all` (d.h. `all.size()`) soll bei der Erzeugung eines `MeinHashSet`-Objekts festgelegt werden und danach unveränderbar sein.

Gehen Sie bei der Entwicklung der Klasse von der "Skelettdatei" `MeinHashSet.java` aus (die Sie am üblichen Ort finden).

Außerdem sollen Sie ein JUnit-Testprogramm namens `MeinHashSetJut01` für Ihre Klasse `MeinHashSet` erstellen. Dazu dürfen (und sollten) Sie die Datei `HashSetJut01.java` (das ist Ihre Lösung der vorigen Aufgabe) in eine Datei `MeinHashSetJut01.java` kopieren und darin alle Vorkommnisse von `HashSet` durch `MeinHashSet` ersetzen.

Benützen Sie diese JUnit-Test-Klasse `MeinHashSetJut01` beim Entwickeln der Klasse `MeinHashSet` so früh wie möglich und so oft wie nötig (das ist wahrscheinlich ziemlich oft).

Zum Abschluß sollten Sie Ihre Klasse `MeinHashSet` auch mit der vorgegebenen Testklasse `MeinHashSetJut02` einem kleinen Massentest unterziehen.

Aufgabe 8: Ein kleiner Dialog mit einem Fibonacci-Server

Schreiben Sie ein Programm namens `FibonacciClient`, welches

1. eine bestimmte FRAGE zu einem bestimmten SERVER schickt (und zum Bildschirm ausgibt) und
2. die Antwort des Servers empfängt (und zum Bildschirm ausgibt).

Der SERVER hat die Adresse "<http://www.elharo.com/fibonacci/XML-RPC/>". **Achtung:** Das letzte Zeichen der Adresse muss ein Schrägstrich / sein!

Die FRAGE muss ein XML-Dokument sein und sollte etwa so aussehen:

```
<methodCall>
  <methodName>calculateFibonacci</methodName>
  <params>
    <param>
      <value><int>17</int></value>
    </param>
  </params>
</methodCall>
```

Wenn alles richtig läuft wird der Server auf diese FRAGE mit einem XML-Dokument antworten, in dem die Zahl 2584 vorkommt (das ist die 17. Fibonacci-Zahl). Falls Sie noch nicht wissen, was Fibonacci-Zahlen sind, können Sie sich z.B. bei Wikipedia darüber informieren (<http://de.wikipedia.org/wiki/Fibonacci-Folge>).

Zur Lösung dieser Aufgabe sollten Sie sich mit den folgenden Klassen, Konstruktoren und Methoden vertraut machen:

URL (Konstruktor, Methode `openConnection`)

URLConnection (kann manchmal nach `URLConnection` gecastet werden)

URLConnection (Methoden `setDoOutput`, `setDoInput`, `setRequestMethod`, `getOutputStream`, `getInputStream`)

OutputStream (Als Parameter für einen `OutputStreamWriter`-Konstruktor)

InputStream (Als Parameter für einen `InputStreamReader`-Konstruktor)

OutputStreamWriter (Konstruktor, Methode `write`, `flush`, `close`)

InputStreamReader (Konstruktor, Methode `read`, `close`)

Eine Verbindung zu einem Server kann man etwa so herstellen:

```
1  URL          serverUrl = new URL(server);
2  URLConnection uVerbind = serverUrl.openConnection();
3  HttpURLConnection hVerbind = (HttpURLConnection) uVerbind;
```

Als request method für die Verbindung `hVerbind` müssen Sie "POST" festlegen, etwa so:

```
4  hVerbind.setRequestMethod("POST");
```

Es folgt eine Skizze der wichtigsten Schritte:

1. Von `hVerbind` ein `OutputStream`-Objekt holen und mit einem `OutputStreamWriter`-Objekt `osw` verbinden.
2. Die Frage zum `osw` und zum Bildschirm ausgeben.
3. Von `hVerbind` ein `InputStream`-Objekt holen und mit einem `InputStreamReader`-Objekt `isr` verbinden.
4. Die Antwort vom `isr` lesen und zum Bildschirm ausgeben.

Aufgabe 9: Ein reflektives Programm

Schreiben Sie eine Klasse `ObjektFabrik02` und darin eine Methode wie folgt:

```

1  public Object erzeugeObjekt(Class c) throws Exception {
2      // Liefert ein Objekt der Klasse c (welches mit Hilfe eines Zufalls-
3      // generators erzeugt wird) oder null (wenn das Erzeugen misslingt).
4
5      ...
6  } // erzeugeObjekt
7

```

Zum Testen Ihrer Klasse `ObjektFabrik02` steht Ihnen das Testprogramm `ObjektFabrik02-Tst.java` zur Verfügung (Sie finden es am üblichen Ort).

Zur Vereinfachung dieser Aufgabe: Falls die Methode `erzeugeObjekt` mit einem Parameter `c` aufgerufen wird, der einen *Reihungstyp* repräsentiert (`c.isArray()` liefert `true`), dann darf sie `null` als Ergebnis liefern. D.h. Sie dürfen *Reihungstypen* in Ihrer Lösung weitgehend ausser Acht lassen.

Falls der Parameter `c` einen *primitiven Typ* repräsentiert (`Byte.TYPE`, `Character.TYPE`, `Short.TYPE`, `Integer.TYPE`, `Long.TYPE`, `Float.TYPE`, `Double.TYPE` oder `Boolean.TYPE`), soll die Methode `erzeugeObjekt` ein Objekt der entsprechenden *Hüllklasse* (`Byte`, `Character`, `Short`, ...) liefern. Der "eingehüllte Wert" des Objekts soll mit Hilfe eines *Zufallsgenerator*s festgelegt werden.

Falls der Parameter `c` eine *Hüllklasse* repräsentiert, soll die Methode `erzeugeObjekt` ebenfalls ein Objekt dieser *Hüllklasse* liefern. Auch in diesem Fall soll der "eingehüllte Wert" des Objekts mit Hilfe eines *Zufallsgenerator*s festgelegt werden.

Falls der Parameter `c` den Typ `String` repräsentiert, soll die Methode `erzeugeObjekt` ein `String`-Objekt liefern, dessen Länge und Zeichen mit Hilfe eines *Zufallsgenerator*s festgelegt wurden. Die Länge sollte allerdings auf "vernünftige Werte" beschränkt sein (z.B. auf Werte zwischen 2 und 10). Wer will, kann auch die zufällig gewählten *Zeichen*, die der erzeugte `String` enthält, auf "leicht lesbare Zeichen" wie z.B. Buchstaben und Ziffern einschränken, und "schwer lesbare Zeichen" wie z.B. Sonderzeichen und chinesische Schriftzeichen ausschließen.

Falls `c` einen anderen Typ (d.h. keinen Reihungstyp, keinen primitiven Typ, keine Hüllklasse und nicht den Typ `String`) repräsentiert, sollte Ihr Programm sich die Konstruktoren der Klasse `c` besorgen und sie der Reihe nach "durchprobieren", bis es gelingt, mit einem der Konstruktoren ein Objekt `ob` zu erzeugen (dann sollte die Methode `erzeugeObjekt` dieses Objekt `ob` als Ergebnis liefern) oder bis Sie alle Konstruktoren erfolglos durchprobiert haben (in diesem Fall sollte die Methode `erzeugeObjekt` den Wert `null` als Ergebnis liefern).

Bevor Sie versuchen können, mit einem bestimmten Konstruktor `k` ein Objekt zu erzeugen, müssen Sie die *Anzahl* und die *Typen* der Parameter von `k` ermitteln und für jeden Parameter ein geeignetes Objekt erzeugen (durch einen rekursiven Aufruf Ihrer Methode `erzeugeObjekt`). Dabei müssen Sie *Endlosrekursionen* vermeiden. Wenn Sie z.B. dabei sind, ein `Vector`-Objekt zu erzeugen, und dabei zu einem Konstruktor `k` kommen, der ein `Vector`-Objekt als Parameter erwartet, dann sollten Sie das (mit Hilfe einer Liste von "verbotenen Typen") erkennen und die aktuelle Ausführung von `erzeugeObjekt` beenden, indem Sie `null` als Ergebnis liefern).

Eine unvollständige Lösung zu dieser Aufgabe (geeignet als Ausgangspunkt für Ihre Lösung) finden Sie in der Datei `ObjektFabrik02.java` (am üblichen Ort). Die unvollständige Lösung können Sie sofort mit dem Testprogramm `ObjektFabrik02Tst.java` testen (allerding wird es dann "nur `null`-Werte statt richtige Objekte" ausgeben). Dann können Sie Ihre Klasse `ObjektFabrik02` schrittweise weiterentwickeln und nach jedem Schritt erneut testen.

Aufgabe 10: Zeitmessungen an Standardsammlungen

Schreiben Sie ein Programm namens `SammlungsVergleich`, welches je ein Objekte der 6 Standard-Sammlungsklassen `ArrayList`, `Vector`, `LinkedList`, `HashSet`, `TreeSet` und `LinkedHashSet` (insgesamt also 6 Sammlungsobjekte) erzeugt und misst, wieviel Zeit die 4 Methoden `add`, `contains` (im positiven Fall), `contains` (im negativen Fall) und `remove` dieser Sammlungs-Objekte benötigen. Insgesamt sollen also (6 mal 4 gleich) **24 Zeitmessungen** durchgeführt und als Ergebnis jeder Messung 3 Zahlen ausgegeben werden.

1. Wie oft wurde die betreffende Methode ausgeführt? (Anzahl)
2. Wieviele Millisekunden haben diese Methodenausführungen zusammen gedauert? (Dauer)
3. Wie oft konnte die betreffende Methode pro Sekunde ausgeführt werden? (ProSek)

Für *ein* Sammlungsobjekt soll die Ausgabe etwa so aussehen:

```

1 -----
2 java.util.ArrayList:
3           Anzahl  Dauer ProSek
4 add       : xxxxxxx xxxxxxx xxxxxxx
5 contains+ : xxxxxxx xxxxxxx xxxxxxx
6 contains- : xxxxxxx xxxxxxx xxxxxxx
7 remove    : xxxxxxx xxxxxxx xxxxxxx
8

```

Tip 1: Verwenden Sie *Strings* als die Objekte, die in die Sammlungen eingefügt, dort gesucht und wieder entfernt werden, z.B. Strings der Form "komponNNNNNNN", wobei NNNNNNNN eine 7-stellige Ganzzahl ist. Solche Objekte lassen sich relativ leicht "automatisch in einer Schleife erzeugen".

Tip 2: Erzeugen Sie zuerst eine *Reihung* namens `zuSamOb` ("zu sammelnde Objekte") mit `MAX_ANZ` vielen solcher `String`-Objekte und verwenden Sie diese Objekte bei allen Zeitmessungen. Dadurch wird erreicht dass die *Zeit zur Erzeugung* dieser Objekte nicht auf die Ausführungszeiten der einzelnen Methoden "angerechnet" wird. Anfangs ist 1000 ein guter Wert für die Konstante `MAX_ANZ`, für die endgültige Zeitmessung ist ein Wert wie $100 * 1000$ gut.

Tip 3: Begrenzen Sie jede einzelne Zeitmessung wie folgt durch *2 Größen*. Lassen Sie jede Methode höchstens `MAX_ANZ` mal ausführen. Brechen Sie aber auch ab, wenn die Gesamtzeit dieser Messung mehr als `MAX_MILLIS` beträgt. Dies ist der wichtigste Tip!

Tip 4: Wenn die Strings in eine Sammlung eingefügt werden (mit `add`) sollte das *nicht* in *sortierter* Reihenfolge passieren sondern in einer zufälligen Reihenfolge.

Tip 5: Wenn die Strings aus einer Sammlung entfernt werden (mit `remove`) sollte das *nicht* in der *gleichen* Reihenfolge passieren wie beim Einfügen, sondern in einer *anderen*, zufällig gewählten Reihenfolge.

Hier ein paar Java-Befehle die zur Lösung dieser Aufgabe nützlich sind:

```

1 // Zu Beginn einer Zeitmessung den Anfangs-Zeitpunkt festhalten und
2 // den "spätesten" End-Zeitpunkt berechnen:
3 long anfangZeit = System.currentTimeMillis();
4 long endeZeit = anfangZeit + MAX_MILLIS;
5 ...
6 // Prüfen, ob die maximale Zeit schon abgelaufen ist:
7 if (System.currentTimeMillis() >= endeZeit) ...
8 ...
9 // Die Komponenten einer Reihung r "mischen", d.h. "in eine
10 // zufällig gewählte Reihenfolge bringen":
11 Collections.shuffle(Arrays.asList(r));

```