Prof. Dr. J. Giesl
M. Brockschmidt, F. Emmes, Dr. C. Fuhs, C. Otto, T. Ströder

Präsenzübung Programmierung WS 2011/2012

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte **genau** einen markieren):

- ○ Informatik Bachelor
- ○ Informatik Lehramt (Bachelor)
- ○ Sonstiges: _____
- ○ Mathematik Bachelor
- ○ Informatik Lehramt (Staatsexamen)

	Anzahl Punkte	**Erreichte Punkte**
Aufgabe 1	**9**	
Aufgabe 2	**10**	
Aufgabe 3	**11**	
Aufgabe 4	**18**	
Summe	**48**	

Allgemeine Hinweise:

- **Auf alle Blätter** (inklusive zusätzliche Blätter) müssen Sie **Ihren Vornamen, Ihren Nachnamen und Ihre Matrikelnummer** schreiben.
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Schreiben Sie mit **dokumentenechten** Stiften, nicht mit roten oder grünen Stiften und nicht mit Bleistiften.
- Bitte beantworten Sie die Aufgaben auf den Aufgabenblättern (benutzen Sie auch die Rückseiten).
- Geben Sie für jede Aufgabe **maximal eine** Lösung an. Streichen Sie alles andere durch. Andernfalls werden alle Lösungen der Aufgabe mit **0 Punkten** bewertet.
- Werden **Täuschungsversuche** beobachtet, so wird die Übung mit **0 Punkten** bewertet.
- Geben Sie am Ende der Übung **alle Blätter zusammen mit den Aufgabenblättern ab**.

Name: **Matrikelnummer:**

Aufgabe 1 (Programmanalyse): (9 Punkte)

Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Tragen Sie hierzu jeweils die ausgegebenen Zeichen in die markierten Stellen hinter „`OUT:`“ ein.

```
public class A {
  public int x;
  public static int y = 0;

  public A() {
    this.x = 5;
    y++;
  }

  public A(int i) {
    this();
    y++;
  }

  public int f(double d) {
    return 1;
  }

  public int f(int i) {
    return 2;
  }
}
```

```
public class B extends A {
  public int z = 8;

  public B(int x) {
    z = 3;
  }

  public int f(double d) {
    return 3;
  }

  public int f(float f) {
    return 4;
  }
}
```

```
public class M {
  public static void main(String[] args) {
    A a = new A(6);
    System.out.println(a.x);              // OUT: [  ]

    int retA = a.f(1.2f);
    System.out.println(retA);             // OUT: [  ]

    B b = new B(7);
    System.out.println(b.y + " " + b.z); // OUT: [  ] [  ]

    int retB = b.f(12);
    System.out.println(retB);             // OUT: [  ]

    A ab = b;
    int retAB = ab.f(1.2f);
    System.out.println(retAB);            // OUT: [  ]
  }
}
```

Aufgabe 2 (Hoare-Kalkül): (8 + 2 = 10 Punkte)

Die Folge F_n der Fibonacci-Zahlen ist folgendermaßen definiert:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \text{ für } 2 \leq n \end{aligned}$$

Somit ist der Wert der Fibonacci-Zahlen mit den Indizes 0 und 1 genau ihr Index und jede weitere Fibonacci-Zahl entspricht der Summe ihrer beiden Vorgänger. Die ersten Elemente sind also $0, 1, 1, 2, 3, 5, 8, \ldots$.
Gegeben sei folgendes *Java*-Programm P, das zu einer Eingabe $n \geq 2$ die n-te Zahl in der Folge der Fibonacci-Zahlen berechnet. Diese Zahl nennen wir F_n und benutzen diese Schreibweise auch in den Zusicherungen (wie in der Nachbedingung zu sehen).

$\langle\, 2 \leq \mathtt{n} \,\rangle$ (Vorbedingung)

```
f0 = 0;
f1 = 1;
res = f0 + f1;
i = 2;
while (i < n) {
    f0 = f1;
    f1 = res;
    res = f0 + f1;
    i = i + 1;
}
```

$\langle\, \mathtt{res} = F_{\mathtt{n}} \wedge \mathtt{f1} = F_{\mathtt{n}-1} \,\rangle$ (Nachbedingung)

a) Vervollständigen Sie die Verifikation des Algorithmus P auf der folgenden Seite im Hoare-Kalkül, indem Sie die unterstrichenen Teile ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

Hinweise:

- Aus der Nachbedingung lässt sich die benötigte Schleifeninvariante ableiten. In der benötigten Schleifeninvariante treten die Variablen `res`, `f1`, `i` und `n` auf.
- Die Schleifeninvariante muss in den mit $(*)$ und $(**)$ markierten Zusicherungen stehen.
- Um die Zusicherungen im Schleifenrumpf zu bestimmen, gehen Sie rückwärts von der Schleifeninvariante aus vor und wenden Sie einfach die Zuweisungsregel des Hoare-Kalküls an. Die Zusicherung oberhalb der Anweisung `i = i + 1;` ergibt sich also, indem Sie in der Schleifeninvariante alle Vorkommen von `i` durch `i + 1` ersetzen etc.
- Falls Sie die Aufgabe nicht vollständig lösen können, versuchen Sie immerhin Teile der Aufgabe zu bearbeiten.
- Sie dürfen beliebig viele Zusicherungs-Zeilen ergänzen oder streichen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Bedenken Sie, dass die Regeln des Kalküls syntaktisch sind, weshalb Sie semantische Änderungen (beispielsweise von $x+1 = y+1$ zu $x = y$) nur unter Zuhilfenahme der Konsequenzregeln vornehmen dürfen.

Name: **Matrikelnummer:**

$\langle 2 \leq \mathtt{n} \rangle$
$\langle 2 \leq \mathtt{n} \wedge 0 = 0 \wedge 1 = 1 \wedge 0 + 1 = 0 + 1 \wedge 2 = 2 \rangle$

```
f0 = 0;
```

$\langle \_\_\_\_\_ \rangle$

```
f1 = 1;
```

$\langle \_\_\_\_\_ \rangle$

```
res = f0 + f1;
```

$\langle \_\_\_\_\_ \rangle$

```
i = 2;
```

$\langle 2 \leq \mathtt{n} \wedge \mathtt{f0} = 0 \wedge \mathtt{f1} = 1 \wedge \mathtt{res} = \mathtt{f0} + \mathtt{f1} \wedge \mathtt{i} = 2 \rangle$

(*)$\langle \_\_\_\_\_ \rangle$

```
while (i < n) {
```

$\langle \_\_\_\_\_ \rangle$

$\langle \_\_\_\_\_ \rangle$

```
    f0 = f1;
```

$\langle \_\_\_\_\_ \rangle$

```
    f1 = res;
```

$\langle \_\_\_\_\_ \rangle$

```
    res = f0 + f1;
```

$\langle \_\_\_\_\_ \rangle$

```
    i = i + 1;
```

(**)$\langle \_\_\_\_\_ \rangle$

```
}
```

$\langle \_\_\_\_\_ \rangle$
$\langle \mathtt{res} = F_{\mathtt{n}} \wedge \mathtt{f1} = F_{\mathtt{n}-1} \rangle$

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und unter Verwendung des Hoare-Kalküls die Terminierung unter der Voraussetzung $2 \leq \mathtt{n}$ bewiesen werden. Begründen Sie, warum es sich bei der von Ihnen angegebenen Variante tatsächlich um eine gültige Variante handelt.

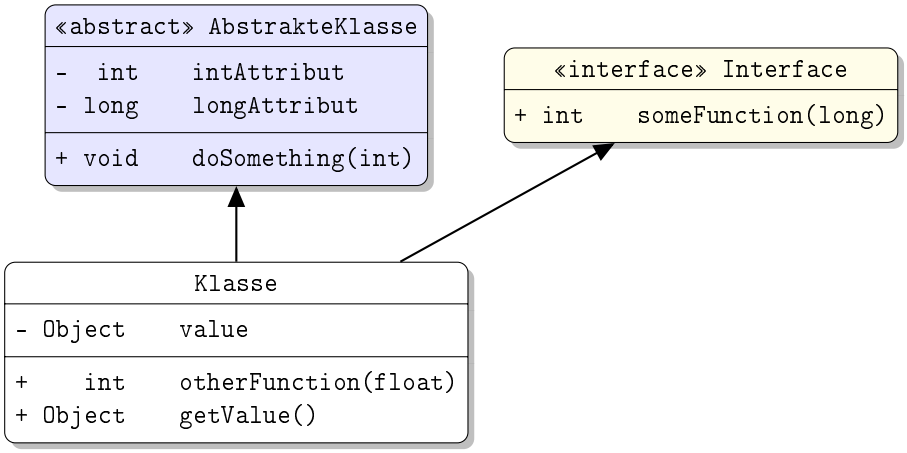
Aufgabe 3 (Klassen-Hierarchie): (7 + 4 = 11 Punkte)

Ziel dieser Aufgabe ist die Erstellung einer Hierarchie zur Verwaltung von Feuerwerkskörpern.

- Ein Feuerwerkskörper hat einen Preis, der als ganze Zahl in Euro angegeben wird. Außerdem bietet jeder Feuerwerkskörper eine Methode `abfeuern()` an, welche nichts zurück liefert.
- Eine Rakete ist ein Feuerwerkskörper und zeichnet sich durch die Farbe ihrer Leuchtspur aus, welche als `String` angegeben wird.
- Ein Heuler ist eine besondere Rakete.
- Ein Knallfrosch ist ein Feuerwerkskörper, welcher sich durch die Knall-Lautstärke auszeichnet. Diese wird ganzzahlig in Dezibel angegeben.
- Ein Geysir ist ein Feuerwerkskörper, welcher durch die Höhe seines Ausbruchs charakterisiert wird. Sie wird als `double`-Wert in Metern angegeben.
- Langanhaltende Feuerwerkskörper bieten eine Methode `brenndauer()` zur Berechnung ihrer Brenndauer an. Diese Methode liefert die Brenndauer in ganzen Sekunden zurück. Heuler und Geysire sind langanhaltende Feuerwerkskörper, während andere Raketen oder Knallfrösche das nicht sind.
- Es gibt keine anderen Feuerwerkskörper außer den hier genannten.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Feuerwerkskörpern. Notieren Sie keine Konstruktoren oder Selektoren. Sie müssen nicht markieren, ob Attribute `final` sein sollen. Achten Sie darauf, dass gemeinsame Merkmale in Oberklassen bzw. Interfaces zusammengefasst werden und markieren Sie alle Klassen als abstrakt, bei denen dies sinnvoll ist.

Verwenden Sie hierbei die folgende Notation:



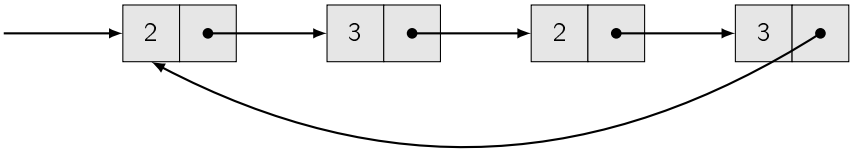
Eine Klasse wird hier durch einen Kasten beschrieben, in dem der Name der Klasse sowie Attribute und Methoden in einzelnen Abschnitten beschrieben werden. Weiterhin bedeutet der Pfeil $B \rightarrow A$, dass A die Oberklasse von B ist (also `class B extends A` bzw. `class B implements A`, falls A ein Interface ist). Benutzen sie `-` um `private` abzukürzen und `+` für alle anderen Sichtbarkeiten (wie z.B. `public`).

Tragen Sie keine vordefinierten Klassen (`String`, etc.) oder Pfeile dorthin in ihr Diagramm ein.

b) Schreiben Sie eine Methode mit der folgenden Signatur:
`public static void langeAbfeuern(Feuerwerkskoerper[] feuerwerk)`
Diese Methode soll alle langanhaltenden Feuerwerkskörper abfeuern, welche im übergebenen Array enthalten sind und deren Brenndauer größer als 5 Sekunden ist. Nehmen Sie dazu an, dass das übergebene Array `feuerwerk` nicht `null` ist.

Aufgabe 4 (Zyklische Listen): (5 + 5 + (2 + 6) = 18 Punkte)

Die Klasse `List` dient zur Repräsentation von Listen von Zahlen. Jedes Element wird als ein Objekt der Klasse `List` dargestellt und enthält einen Verweis auf das Nachfolger-Element im Attribut `next` und die eigentliche Zahl in `val`. Das Besondere an dieser Liste ist, dass das letzte Element wieder auf das erste Element zeigt. Die leere Liste würde als `null` dargestellt werden, wird aber in dieser Aufgabe nie betrachtet. Betrachten Sie als Beispiel folgende Darstellung der zyklischen Liste [2, 3, 2, 3], wobei der von links kommende Pfeil eine Referenz auf das erste Listenelement darstellen soll:



Wir verwenden zur Repräsentation der Datenstruktur die folgende Klasse `List`:

```
public class List {
  List next;
  int val;
}
```

a) Implementieren Sie die Methode `length()` in der Klasse `List`, die zurückgibt, wieviele Objekte eine `List` enthält. Dabei soll also für die Beispielliste 4 zurückgegeben werden, obwohl sowohl 2 als auch 3 sich wiederholen. Das Ergebnis soll nur von der Anzahl der verwendeten Objekte abhängen. Verwenden Sie dazu **ausschließlich Schleifen** und keine Rekursion.

Hinweise:

- Sie dürfen annehmen, dass das `next`-Attribut nie `null` ist.
- Stellen Sie sicher, dass Sie kein Objekt doppelt zählen. Sie können dafür den Operator == verwenden, der überprüft, ob zwei Werte auf dasselbe Objekt zeigen. Überprüfen Sie in jeder Iteration, ob Sie das erste Listenelement wieder erreicht haben.

b) Implementieren Sie die Methode `get(int i)` in der Klasse `List`, die den `int`-Wert des `i`-ten Elements einer Liste zurückgibt. Dabei soll der Aufruf `get(0)` das erste Element zurückgeben (wir zählen also 0-basiert), im Beispiel also 2. Ihre Implementierung soll auch Werte für `i` verarbeiten, die größer als die Listenlänge sind. Dies soll so realisiert werden, dass bei einer Listenlänge ℓ der Aufruf `get(i)` immer dasselbe Ergebnis wie der Aufruf `get(i % ℓ)` zurückgibt[1]. In unserem Beispiel sollte `get(13)` also `get(1)` entsprechen und daher den Wert 3 zurückgegeben. Verwenden Sie keine Schleifen, sondern **ausschließlich Rekursion**. Benutzen Sie **nicht** eine Methode wie `length()` aus a).

Hinweise:

- Sie dürfen annehmen, dass das `next`-Attribut nie `null` ist.
- Sie dürfen annehmen, dass der Parameter `i` immer nicht-negativ ist.
- Sie benötigen hier keine explizite %-Operation, da Sie beim Ablaufen der Liste mit Hilfe des `next`-Attributs nach ℓ Schritten wieder das erste Element besuchen.

[1] % ist der Modulo-Operator, der den Rest bei einer Division mit Rest zurückgibt.

c) i) Ändern Sie die Definition der Klasse `List` so ab, dass die Klasse einen Typparameter `T` hat und die Liste nicht mehr `int`-Werte speichert, sondern nur Werte vom Typ `T`.

ii) Implementieren Sie eine Methode `toSet()` in der Klasse `List<T>`, die ein `Set<T>` zurückgibt. `Set` ist hierbei das Interface, das alle vom `Collections`-Framework bereitgestellten Typen zur Darstellung von Mengen implementieren. Für die Beispielliste soll also ein Objekt zurückgegeben werden, das der Menge `{2,3}` entspricht.

Hinweise:

- Beachten Sie auch die Hinweise aus Aufgabenteil a).
- Sie dürfen davon ausgehen, dass das `Collections`-Framework mittels `import java.util.*` bereits importiert wurde.
- Alle Klassen, die `Set<T>` implementieren, stellen eine nicht-statische Methode `add(T element)` zur Verfügung, die `element` in die Menge einfügt, wenn `element` noch nicht in ihr enthalten ist.
- `HashSet<T>` ist eine geeignete konkrete Implementierung von `Set<T>`, die Sie hier verwenden können.