Einführung in die Informatik 1

Prof. Dr. Harald Räcke, R. Palenta, A. Reuss, S. Schulze Frielinghaus 18.04.2017

Wiederholungsklausur

Vorname	Nachname
Matrikelnummer	Unterschrift

- Füllen Sie die oben angegebenen Felder aus.
- Schreiben Sie nur mit einem dokumentenechten Stift in schwarzer oder blauer Farbe.
- Verwenden Sie kein "Tipp-Ex" oder ähnliches.
- Die Arbeitszeit beträgt **90** Minuten.
- Prüfen Sie, ob Sie **10** Seiten erhalten haben.

vorzeitige Abgabe um ..... Hörsaal verlassen von ..... bis .....

1	2	3	4	5	6	7	Σ

Aufgabe 1 Multiple Choice [12 Punkte]

Kreuzen Sie zutreffende Antworten an. Punkte werden nach folgendem Schema vergeben:

- Richtige Antwort: 1 Punkt
- Falsche Antwort: −1 Punkt
- Keine Antwort: 0 Punkte

Eine negative Gesamtpunktzahl wird zu 0 aufgerundet.

	Wahr	Falsch
Der Ausdruck `"5".equals("" + 5)` evaluiert zu `true`	☐	☐

Sei `list` eine nicht leere `LinkedList<Integer>` aus der Java-Standardbibliothek. Das Statement
`for (int i = 0; i < list.size(); ++i) list.add(i);`
ist äquivalent zu folgendem
`int n = list.size();`
`for (int i = 0; i < n; ++i) list.add(i);`
	☐	☐

Folgende Klasse kompiliert *nicht*, da die Variable `bar` nicht initialisiert wurde:
`class Foo {`
`  int baz() {`
`  int bar;`
`  return bar;`
`  }`
`}`
	☐	☐

Folgende Klasse kompiliert:
`class Foo<T> {`
`  void bar(T t) { }`
`  void bar(Object t) { }`
`}`
	☐	☐

Programme, die zur Synchronisierung nur `synchronized` verwenden, sind immer frei von Deadlocks.	☐	☐

Folgende Klassen kompilieren:
`class Bar { }`
`class Foo<T extends Bar> { }`
	☐	☐

	Wahr	Falsch
Angenommen `e1` und `e2` sind Ausdrücke, die sich zu einem Boolean evaluieren. Dann wird beim Evaluieren des Ausdrucks `(e1 && e2)` immer `e1` und `e2` evaluiert.	☐	☐
Statische Methoden können überladen werden.	☐	☐
Auch durch Operatoren wie z. B. `/` (Division) oder `%` (Modulo) können Exceptions ausgelöst werden.	☐	☐
Ein Konstruktor kann `final` sein.	☐	☐
Folgendes Programm kompiliert: `class Foo { public static void main(String[] args) { int x = 0, y = 42; try { y /= x; } System.out.println("/ by 0"); catch(Exception e) { System.out.println("error"); } } }`	☐	☐
Folgendes Programm gibt `5` auf der Konsole aus: `public class Foo { static int x = 10; public static void main(String[] args) { for (int x = 0; x < 5; ) { ++x; } System.out.print(x); } }`	☐	☐

Aufgabe 2 Rekursion [8 Punkte]

Gegeben ist die folgende Methode:

```
public static int f(int n) {
    int s = 0;
    while (n / 10 > 0) {
        s = s + (n % 10);
        n = n / 10;
    }
    return s + n;
}
```

Welchen Rückgabewert liefert die Methode bei den folgenden Aufrufen?

`f(5)`: _____ `f(1234)`: _____

Schreiben Sie eine Methode `public static int fRec(int n)`, die für alle Integer den gleichen Rückgabewert liefert wie die Methode `f`.

Sie dürfen weitere Methoden schreiben, aber nur selbstgeschriebene Methoden verwenden. Es dürfen keine Schleifen vorkommen.

Geben Sie für jede Ihrer Methoden an, ob diese endrekursiv ist, und begründen Sie Ihre Antwort *kurz*.

Aufgabe 3 Ausdrücke [12 Punkte]

Zu welchen Werten werden die folgenden Ausdrücke ausgewertet? Kennzeichnen Sie `String`s durch Anführungszeichen, z. B. `"Text123"`. Verwenden Sie für boolesche Werte `true` und `false`. Gehen Sie davon aus, dass vor jeder Teilaufgabe `a = 2` und `b = 5` gilt.

(i) `!(a != ++b) ? (a == b) : (a == a++)`

Antwort: _____

(ii) `a-- - --a + b++ - b--`

Antwort: _____

(iii) `4 + a * b + "" + 2 * 0`

Antwort: _____

Welche Werte haben die Variablen `a` und `b` nach Auswertung der jeweiligen Ausdrücke? Gehen Sie davon aus, dass vor jeder Teilaufgabe `a = -3` und `b = 4` gilt.

(a) `b = a-- - --b + ++a`

`a` = _____ `b` = _____

(b) `a = b-- + --a - (b+1)`

`a` = _____ `b` = _____

(c) `b = --b + ++a + b - b--`

`a` = _____ `b` = _____

Geben Sie durch eine vollständige Klammerung des Ausdrucks an, in welcher Reihenfolge die Teilausdrücke bei der Auswertung des gesamten Ausdrucks ausgewertet werden. Beachten Sie, dass die Operatoren die bekannte Funktionalität haben, jedoch hier eine **andere Präzedenzordnung** gelten soll. Die Präzedenzordnung ist in der Tabelle unten dargestellt. Ein Operator in einer Zeile hat eine höhere Priorität als die Operatoren in den Zeilen darunter. In einer Zeile haben die Operatoren die gleiche Priorität. Operatoren gleicher Priorität werden in der Reihenfolge von links nach rechts ausgewertet.

Beispiel: Die Auswertungsreihenfolge von `3*2-1` ist durch die vollständige Klammerung `((3*2)-1)` bestimmt.

Operatoren
`/`
`+ *`
`%`
`-`

(1) `3 + 4 / 5 * 6 % 7` Antwort: _____

(2) `1 - 4 + 7 / 8` Antwort: _____

(3) `2 * 4 + "" + 4` Antwort: _____

Aufgabe 4 Häufigkeit [10 Punkte]

Implementieren Sie die Methode `public static int most(int[] input)`. Die Methode gibt die Zahl zurück, die am häufigsten im Array `input` vorkommt. Gibt es mehrere Zahlen mit dem gleich häufigsten Vorkommen, darf irgendeine dieser Zahlen mit dem häufigsten Vorkommen zurückgegeben werden.

Die Größe eines Arrays `a` kann durch `a.length` bestimmt werden.

Sie dürfen keine weiteren Methoden schreiben und auch keine weiteren Methoden verwenden. Sie können davon ausgehen, dass `input` nicht `null` ist und mindestens eine Zahl enthält.

Aufgabe 5 Textsuche [13 Punkte]

Wir betrachten folgenden einfachen Algorithmus zur Suche des ersten Vorkommens einer Zeichenfolge `pattern` der Länge m in einer anderen Zeichenfolge `txt` der Länge n, wobei $m \leq n$ gilt. Der Index einer Zeichenfolge beginnt bei 0. Beispiel: Das erste Vorkommen der Zeichenfolge `tre` in der Zeichenfolge `trtrtretretre` ist an Position 4.

Der Algorithmus basiert im Wesentlichen auf zwei Indizes i und k. Der Index i gibt die Position im zu durchsuchenden Text `txt` an, die mit der Position k im gesuchten Muster `pattern` verglichen wird.

Beide Indizes werden zu Beginn mit 0 initialisiert. Wird das erste Zeichen des gesuchten Musters `pattern` im Text `txt` an Position p gefunden, müssen die $m-1$ darauffolgenden Zeichen im Text `txt` mit den folgenden Zeichen im Muster `pattern` verglichen werden. Stimmen die Zeichen überein, kann die Position des ersten Vorkommens des Musters `pattern` im Text `txt` ermittelt und zurückgegeben werden. Stimmen nicht alle $m-1$ darauffolgenden Zeichen überein, muss ab Position $p+1$ mit der Suche nach dem ersten Zeichen des Musters `pattern` im Text `txt` fortgefahren werden.

Die Methode `search` wirft eine `IllegalArgumentException`, falls die Parameter `txt` oder `pattern` `null` sind oder `pattern` die Länge 0 hat. Die Methode gibt -1 zurück, falls die Länge des zu durchsuchenden Textes `txt` kleiner als die Länge des Musters `pattern` ist oder das Muster `pattern` im Text `txt` nicht gefunden wurde. Andernfalls gibt die Methode die Position (des ersten Zeichens) des ersten Vorkommens des Musters `pattern` im Text `txt` zurück.

Vervollständigen Sie die folgende Implementierung des beschriebenen Algorithmus zur Textsuche. Schreiben Sie in jede Box einen Ausdruck.

Aufgabe 6 Polymorphie [12 Punkte]

Betrachten Sie die Aufrufe 1 bis 4 in dem folgenden Programm.

```
public class A {
   public int  min(C c, B b) { return 0; }    // Methode 1
   public void min(A a, B b) { }              // Methode 2
}
public class B extends A {
   public void min(A a1, A a2) { }            // Methode 3
}
public class C extends B {
   public B min(A a, C c) { return new B(); } // Methode 4
}
public class Poly {
   public static void main (String[] args) {
      A a = (B)(new C());
      B b = new B();
      C c = new C();
      c.min(a, c);          // Aufruf 1
      b.min(a, (B)c);       // Aufruf 2
      ((B)c).min(c, (B)c); // Aufruf 3
      ((A)b).min((B)a, b); // Aufruf 4
   }
}
```

Bestimmen Sie für jeden der vier Aufrufe $e_0.f(e_1, \ldots, e_k)$

a) die statischen Typen $T_0, \ldots, T_k$ der Ausdrücke $e_0, \ldots, e_k$.

b) den dynamischen Typ des Objekts, zu dem sich e_0 auswertet.

c) die aufgerufene Methode (1, 2, 3 oder 4).

Schreiben Sie die Lösung für a) bis c) in die folgende Tabelle. Geben Sie dabei $T_0, \ldots, T_k$ für a) als mit Komma getrennte *geordnete* Liste an. Bestimmen Sie die aufgerufene Methode für c) durch Angabe der entsprechenden Methodennummer.

Aufruf	a) (als Liste)	b)	c) Methode
1			
2			
3			
4			

Aufgabe 7 Threads [8 Punkte]

Betrachten Sie folgendes Programm:

```
public class NetworkDevice implements Runnable {
    public NetworkDevice partner;

    public synchronized void send(int x) {
        partner.receive(x);
    }

    public synchronized void receive(int x) {
        System.out.println("Received: " + x);
    }

    public void run() {
        send(42);
    }
}

public class Deadlock {
    public static void main(String[] args) {
        NetworkDevice d1 = new NetworkDevice();
        NetworkDevice d2 = new NetworkDevice();
        d1.partner = d2;
        d2.partner = d1;
        new Thread(d1).start(); // Thread 1
        new Thread(d2).start(); // Thread 2
    }
}
```

Zeigen Sie im Folgenden, dass das Programm nicht frei von Deadlocks ist. Füllen Sie dazu die Tabelle weiter unten aus. Jede Zeile steht für einen Zeitschritt und darf daher auch nur genau einen Eintrag enthalten. Als Einträge sind *ausschließlich* folgende erlaubt:

- `d1.run:Enter` / `d1.run:Return` und analog für `d2`
- `d1.send:Enter` / `d1.send:Return` und analog für `d2`
- `d1.receive:Enter` / `d1.receive:Return` und analog für `d2`
- `d1.lock:Enter` / `d1.lock:Return` und analog für `d2`
- `d1.unlock:Enter` / `d1.unlock:Return` und analog für `d2`

Dabei bedeuten die Einträge folgendes:

- `x.foo:Enter` gibt an, dass die Methode `foo` auf dem Objekt `x` aufgerufen wurde. Das Pendant `x.foo:Return` gibt an, dass der Aufruf zum Aufrufer zurückkehrt.
- `x.lock:Enter` versucht, das Lock auf dem Objekt `x` zu akquirieren. Das Pendant `x.lock:Return` gibt an, dass das Lock akquiriert wurde.

- `x.unlock:Enter` versucht, das Lock auf dem Objekt `x` freizugeben. Das Pendant `x.unlock:Return` gibt an, dass das Lock freigegeben wurde.

Wird eine `synchronized`-Methode mit dem Namen `foo` von dem Objekt `x` aufgerufen, so wird erst `x.foo:Enter` ausgeführt und dann das jeweilige Lock `x.lock:Enter` akquiriert.

Geben Sie nun einen Programm-Ablauf an, sodass ein Deadlock entsteht (das Programm terminiert also nicht). Unterstreichen Sie die Lock-Statements, die zu einem Deadlock führen.

Thread 1	Thread 2
`d1.run:Enter`	
	`d2.run:Enter`