Prüfung Computation, Programming

1. Computation: Reguläre Ausdrücke [5 Punkte]

Zusammenfassung reguläre Ausdrücke

a	Das Zeichen a
.	Ein beliebiges Zeichen
[abc]	Ein beliebiges Zeichen aus der Menge {a, b, c}
[a-f]	Ein beliebiges Zeichen aus der Menge {a, b, c, d, e, f}
\d	eine Ziffer [0-9]

X|Y	X oder Y
X*	Eine beliebige Wiederholung von X
X+	Mindestens einmal X
X?	Höchstens einmal X
{n}	der vorangehende Ausdruck muss exakt n mal vorkommen
{m,n}	der vorangehende Ausdruck muss mindestens m mal vorkommen und darf höchstens n mal vorkommen
(xyz)	Gruppierung: xyz müssen miteinander vorkommen
[^xyz]	Nicht x, nicht y, nicht z
\X	X ein Sonderzeichen \()[]*+?{}.|^$-

Die Fahrpläne der SBB werden auf sbb.ch wie folgt dargestellt:

Details	Bahnhof/Haltestelle	Datum	Zeit	Dauer	Umst.	Reise mit	Belegung
1	Zürich HB Aarau	Do, 04.11.10	ab 20:38 an 21:05	0:27	0	RE	1. 2.
2	Zürich HB Aarau	Do, 04.11.10	ab 21:04 an 21:28	0:24	0	ICN	1. 2.
3	Zürich HB Aarau	Do, 04.11.10	ab 21:08 an 21:35	0:27	0	IR	1. 2.
4	Zürich HB Aarau	Do, 04.11.10	ab 21:38 an 22:05	0:27	0	RE	1. 2.

1. Markieren Sie in der obigen Abbildung deutlich, welche Textstellen durch den regulären Ausdruck \d{2}:\d{2} und welche durch den regulären Ausdruck \d{2}\.\d{2}\.\d{2} gefunden werden.

2. Beschreiben Sie in je einem Satz, was diese beiden Ausdrücke erkennen:

Mögliche Lösung [2 Punkte total für 1. und 2.]

\d{2}:\d{2}	Uhrzeiten im 24h-Format - 20:38, 21:05 usw.
\d{2}\.\d{2}\.\d{2}	Datumsangaben im Format TT.MM.JJ - 04.11.10

3. Bei der Eingabe von Mastercard Kreditkarten-Nummern in einem Webformular können Programme folgenden regulären Audruck verwenden, um die Gültigkeit der Nummern zu überprüfen, nachdem alle Leerzeichen entfernt wurden: 5[1-5][0-9]{14}

Beschreiben Sie in Worten präzise, welches Format Mastercard Kreditkartennummern haben:

Mögliche Lösung [1 Punkt]

Mastercard Kreditkartennummern beginnen mit einer Zahl zwischen 51 und 55, dann folgen 14 Ziffern.

4. Das Format von Schweizer Postkontonummern lautet: Zwei Ziffern, Bindestrich, ein bis sechs Ziffern, Bindestrich, eine Ziffer. Die Glückskette zum Beispiel hat die Kontonummer 10-15000-6. Auch 12-345678-9 ist eine vom Format her gültige Postleitzahl.

Geben Sie einen regulären Ausdruck an, der Schweizer Postkontonummern erkennt:

Mögliche Lösung [2 Punkt]

\d{2}\-\d{1,6}\-\d

2. Computation: Aussagenlogik [5 Punkte]

Im Unterricht haben wir in LogicTraffic die Sicherheit von Kreuzungen mit Wahrheitstabellen und aussagenlogischen Formeln beschrieben.

Für diese Aufgabe betrachten wir den sogenannten Produktausschluss: Gewisse Produkte, die zum Beispiel ein schweizerisches Telekom-Unternehmen verkauft, können nicht miteinander kombiniert werden (die im folgenden verwendeten Produkte sind fiktiv):

Produktausschlussmatrix

	Flatrate	**Halbpreis**	**Plaudern**
Flatrate	---	nicht ok	nicht ok
Halbpreis	nicht ok	---	ok
Plaudern	nicht ok	ok	---

Das heisst zum Beispiel, Halbpreis kann nicht mit Flatrate kombiniert werden, aber mit Plaudern.

Hinweis: Die Tabelle von links nach rechts oder von oben nach unten gelesen werden - die Einträge sind symmetrisch.

Wahrheitstabelle

Flatrate	**Halbpreis**	**Plaudern**	**ok?**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

1. Füllen Sie oben rechts die Wahrheitstabelle aus (rechts Spalte "ok"): Welche Produktkombinationen sind zulässig? [Lösung oben eingetragen, 1 Punkt]

Bei den drei Produktspalten bedeutet 0, das Produkt ist nicht gewählt; 1 bedeutet, das Produkt ist gewählt. Geben Sie in der Spalte "ok" mit 0 unzulässige Kombinationen an und mit 1 zulässige Kombinationen.

2. Leiten Sie aus der Produktausschlussmatrix die Formel in Implikationsschreibweise her, die angibt, welche Produktkombinationen zulässig sind.

Hinweis: Wenn Sie wollen, können Sie dabei folgende Abkürzungen verwenden: Flatrate = A, Halbpreis = B, Plaudern = C.

Hinweis: Sie können die Schreibweise für die Formeln wählen:
- die Wörter ODER, UND, NICHT
- die Symbole wie bei LogicTraffic (v für ODER, ^ für UND, ¬ für NICHT)

Und nicht vergessen - im Zweifelsfall Klammern () verwenden, um die Gruppierung richtig hinzubekommen.

Mögliche Lösungen [1 Punkt]

(Flatrate => NICHT Halbpreis) UND (Flatrate => NICHT Plaudern)
(A => ¬B) ^ (A => ¬C)

Flatrate => (NICHT Halbpreis UND NICHT Plaudern)
A => ¬B ^ ¬C

Flatrat => NICHT (Halbpreis ODER Plaudern)
A => ¬(B v C)

3.a. Geben Sie eine andere Formel als bei 2. an, die Sie direkt aus aus der Produktausschlussmatrix herleiten können:

Mögliche Lösungen [1 Punkt]

NICHT (Flatrate UND Halbpreis) UND NICHT (Flatrate und Plaudern)
¬(A ^ B) ^ ¬(A ^ C)

(NICHT Flatrate ODER NICHT Halbpreis) UND (NICHT Flatrate ODER NICHT Halbpreis)
(¬A v ¬B) ^ (¬A v ¬C)

3.b. Beschreiben Sie in ein bis drei Sätzen, wie Sie die Formel in 3.a. hergeleitet haben:

Mögliche Lösungen [1 Punkt]

Eine “nicht ok” Produktkombination von A und B wird wie folgt beschrieben:
NICHT(A UND B) = NICHT A ODER NICHT B

Anschliessend werden alle “nicht ok” Produktkombinationen mit UND verknüpft, weil keine davon erfüllt sein darf.

4. Verwenden Sie Ihre Wahrheitstabelle, um die Formel in KKNF oder KDNF anzugeben - wählen Sie dabei diejenige Formel, die kürzer ist:

Mögliche Lösungen [1 Punkt]

Da nur drei Einträge 0 ergeben, ist KKNF kürzer:

(NICHT Flatrate ODER Halbpreis ODER NICHT Plaudern) UND
(NICHT Flatrate ODER NICHT Halbpreis ODER Plaudern) UND
(NICHT Flatrate ODER NICHT Halbpreis ODER NICHT Plaudern)

(¬A v B v ¬C) ^ (¬A v ¬B v C) ^ (¬A v ¬B v ¬C)

3. Programming: Programm lesen [5 Punkte]

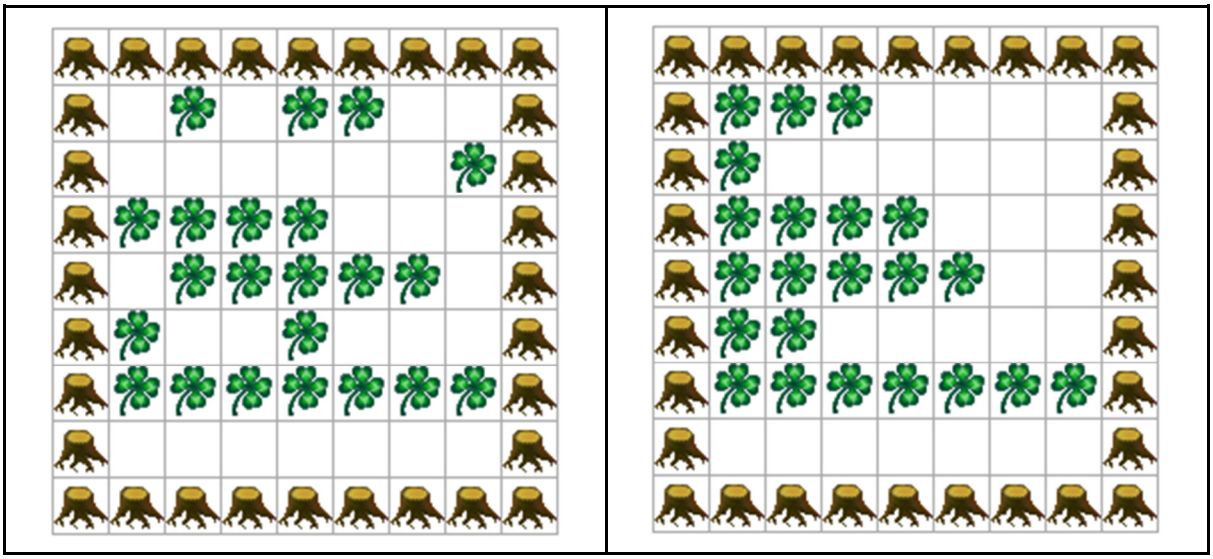
Sie erhalten ein JavaKara-Programme, das ein anderer Programmierer geschrieben hat. Leider hat er sich keine Mühe gegeben, das Programm lesbar zu gestalten:

```
public void myMainProgram() {
	for (int i = 1; i < world.getSizeY() - 1; i++) {
		int j = methode1(i);
		methode2(i, j);
	}
}

int methode1(int y) {
	int t = 0;
	for (int x = 1; x < world.getSizeX() - 1; x++) {
		if (world.isLeaf(x, y)) {
			world.setLeaf(x, y, false);
			t++;
		}
	}
	return t;
}

void methode2(int y, int n) {
	for (int x = 1; x <= n; x++) {
		world.setLeaf(x, y, true);
	}
}
```

Immerhin hat er Ihnen eine Welt hinterlassen, an der Sie das Programm testen können (links in folgender Abbildung).

Zeichnen Sie in die Welt rechts in der folgenden Abbildung, wie die Welt nach Ausführung des Programmes aussieht:

Lösung [2 Punkte]



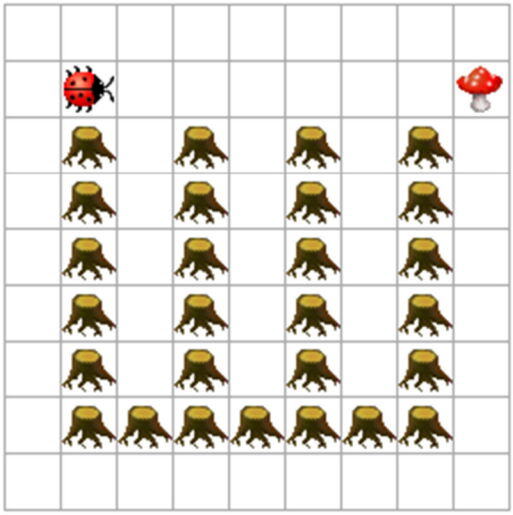
Beschreiben Sie in je ein, zwei Sätzen, was welche Methode macht:

Mögliche Lösungen [3 Punkte, 1 Punkt pro Methode]

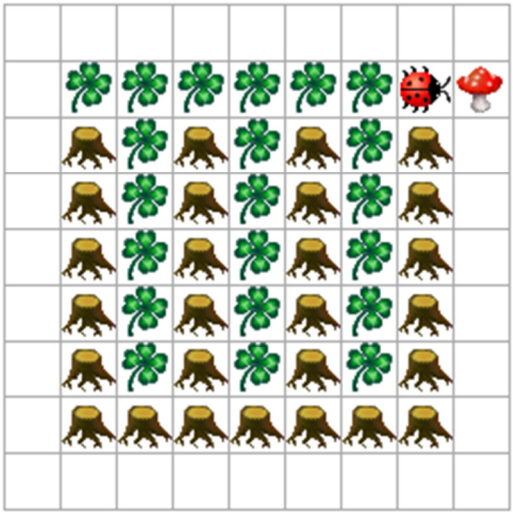
methode1	Die Methode gibt zurück, wieviele Kleeblätter es in der Zeile y hat, und entfernt alle gezählten Kleeblätter.
methode2	Die Methode legt n Kleeblätter in Zeile y, startend bei x=1.
myMainProgram	Das Hauptprogramm entfernt in jeder Zeile alle Kleeblätter und legt dann die entfernte Anzahl Kleeblätter wieder hin, von links her aneinander gereiht.

4. Programming: Programm schreiben [5 Punkte]

Schreiben Sie ein Programm, das Kara in einer Welt wie in der Abbildung links gezeigt Kleeblätter legen lässt. Kara soll aufhören, wenn vor ihm ein Pilz steht:



Welt vorher



Welt nachher

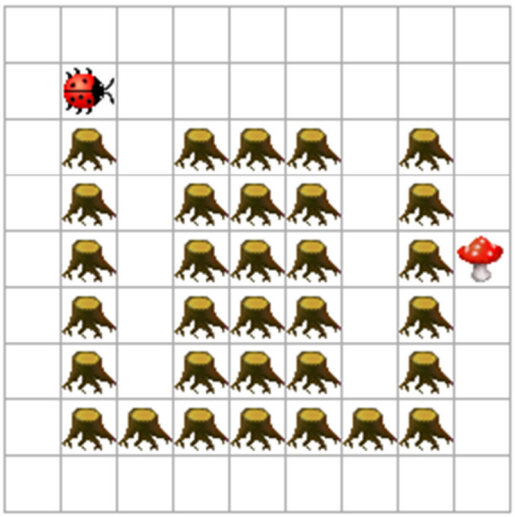
Der Anfang ist bereits gemacht. Ergänzen Sie den fehlenden Programmteil. Überlegen Sie sich, welche Situationen Kara antreffen kann und wie er darauf reagieren muss. Es kann Ihnen auch helfen, zunächst nur ein Programm zu schreiben, das den Käfer zum Pilz laufen lässt, und erst anschliessend das Pilzlegen zu ergänzen. Schreiben Sie unten auf jeden Fall Ihre vollständige Lösung auf:

Mögliche Lösung [1 Punkte für Weg richtig laufen, 2 Punkte für Kleeblätter richtig legen]

```
public void myMainProgram() {
        while (!kara.mushroomFront()) {
                if (!kara.treeRight()) {
                        kara.turnRight();
                }
                if (!kara.treeFront()) {
                        if (!kara.onLeaf()) {
                                kara.putLeaf();
                        }
                        kara.move();
                } else {
                        kara.turnLeft();
                }
        }
}
```

Hinweis: Die genaue Java-Schreibweise ist nicht entscheidend. Ihr Programm muss aber unmissverständlich zu lesen und die Bedeutung klar sein.

Analysieren Sie Ihre Lösung: Funktioniert Sie auch in einer Welt wie in der folgenden Abbildung gezeigt, wo der Pilz woanders steht und wo eine der “Säulen” dicker ist?



Ja, meine Lösung funktioniert auch in dieser Welt, oder nein, meine Lösung funktioniert in dieser Welt nicht:

Ja.

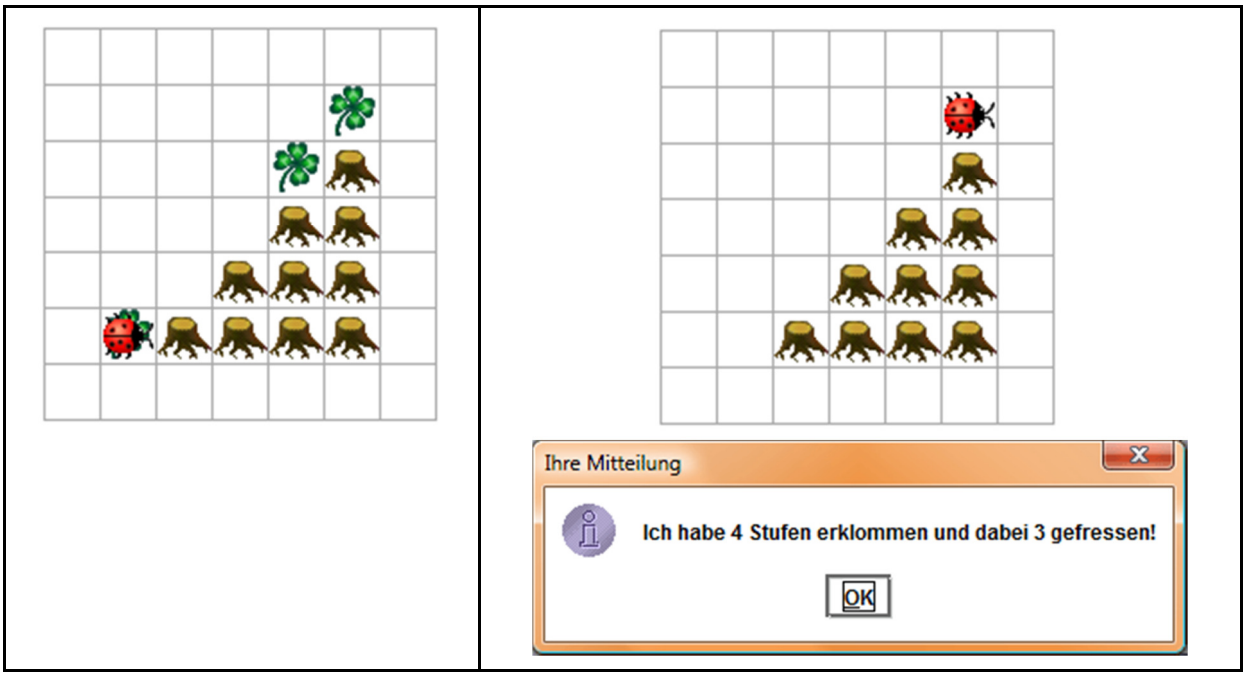
Begründen Sie Ihre Antwort in zwei, drei Sätzen. Begründen Sie einerseits die in Bezug auf die “dicke” Säule, andererseits in Bezug auf die veränderte Position des Pilzes:

Mögliche Lösung [2 Punkte - 1 Punkt für Begründung zu Pilz, 1 Punkt für Begründung zu Bäumen]

Ja. Das Programm hört auf, sobald es vor einem Pilz steht, egal, wo der Pilz steht. Das Programm lässt Kara Bäumen entlang laufen, solange vor ihm kein Baum ist. Wie “dick” die “Säulen” sind, spielt daher keine Rolle.

5. Programming: Programm schreiben [5 Punkte]

Kara soll eine Treppe aus Bäumen besteigen und dabei alle Kleeblätter fressen. Wenn er das Ende der Treppe erreicht hat, soll er die Anzahl Treppenstufen und die Anzahl Kleeblätter ausgeben:



1. Schreiben Sie zunächst ein Programm, das Kara die Treppe hochlaufen lässt. Er soll dabei noch nicht zählen, weder Stufen noch Kleeblätter; er soll nur sicher das obere Ende der Treppe erreichen:

Mögliche Lösung [2 Punkte - 1 Punkt für Abbruchbedingung, 1 Punkt für Befehle]

```
public void myProgram() {
        while (kara.treeFront()) {
                kara.turnLeft();
                kara.move();
                kara.turnRight();
                kara.move();
        }
}
```

Hinweis: Die genaue Java-Schreibweise ist nicht entscheidend. Ihr Programm muss aber unmissverständlich zu lesen und die Bedeutung klar sein.

2. Für das Zählen der Kleeblätter und Stufen hat jemand schon für Sie den Anfang gemacht. Ergänzen Sie das Programm unten so, dass Kara die Treppe hochläuft und dabei die **Stufen zählt** und die **Kleeblätter frisst und zählt**.

Mögliche Lösung [3 Punkte - 1 für Kleeblätter minus 1, 1 für alle Kleeblätter, 1 für Stufen]

```
public void myProgram() {
  int kleeblaetter = 0;
  int stufen = 0;

  while (kara.treeFront()) {
    if (kara.onLeaf()) {
      kara.removeLeaf();
      kleeblaetter++;
    }
    kara.turnLeft();
    kara.move();
    kara.turnRight();
    kara.move();
    stufen++;
  }

  if (kara.onLeaf()) {
    kara.removeLeaf();
    kleeblaetter++;
  }

  tools.showMessage("Ich habe " + stufen + " Stufen erklommen und dabei "
                    + kleeblaetter + " gefressen!");
}
```