Grundlagen der Informatik 1

Wintersemester 2009/2010

Prof. Dr. Max Mühlhäuser, Dr. Rößling
http://proffs.tk.informatik.tu-darmstadt.de/teaching

Üung 10 Version: 1.0 11. 01. 2010

---

1 Mini Quiz

Bitte kreuzen Sie die korrekten Aussagen an.

Vererbung

- ☐ Eine Klasse kann gleichzeitig von mehreren Klassen erben.
- ☐ Eine Unterklasse kann immer *alle* geerbten Methoden überschreiben.
- ☐ Vererbung wird in Java durch das Schlüsselwort *inherit* markiert.
- ☐ Aufrufe über **super** werden statisch gebunden.
- ☐ Aufrufe über **this** werden dynamisch gebunden.

Abstrakte Klassen und Interfaces

- ☐ Eine Klasse kann gleichzeitig mehrere Interfaces implementieren.
- ☐ Ein Interface kann von einem anderen Interface erben.
- ☐ In jeder nicht abstrakten Klasse muss immer mindestens ein Konstruktor implementiert werden.
- ☐ *Interfaces* enthalten nur Methoden und Konstanten, aber keine Attribute.
- ☐ *Abstrakte Klassen* deklarieren nur Attribute und Methoden, aber keine Konstanten.

2 Fragen

1. Ein wichtiges Konzept der Objektorientierung ist die Vererbung. Welche Vorteile ergeben sich dadurch?
2. Erklären Sie mit eigenen Worten den Begriff *Inkrementelles Programmieren*.
3. Worin liegen die Unterschiede zwischen *Interfaces* und *abstrakten Klassen*? Wann macht es Sinn, diese zu nutzen? Welches der beiden Konzepte unterstützt ganz besonders Reusability (Wiederverwendung)?
4. Was versteht man unter dem Begriff des Information Hiding?
5. Vererbungs-Wissenstest: Streichen Sie falsche Antworten, so dass die Aussagen wahr werden.
   - Es gibt eine / keine Basisklasse, von der alle Java-Klassen direkt oder indirekt abgeleitet sind.
   - Subklasse ist ein anderer Ausdruck für Unterklasse / Oberklasse.

- Superklasse ist ein anderer Ausdruck für Unterklasse / Oberklasse.
- Eine Klasse kann / kann nicht Superklasse für mehrere Subklassen sein. Eine Klasse kann / kann nicht Subklasse mehrerer Superklassen sein.

3 The Final Countdown

Für die folgende Aufgabe werden Kentnisse über das Schlüsselwort **final** vorausgesetzt. Machen Sie sich bitte damit vertraut und erklären Sie kurz die Auswirkungen von **final** bei der Anwendung auf:

- Attribute und Methodenparameter
- Methoden
- Klassen

Infos zu diesem Schlüsselwort finden Sie hier:

- `http://de.wikipedia.org/wiki/Java-Syntax` (deutsch)
- `http://renaud.waldura.com/doc/java/final-keyword.shtml` (englisch)

Schauen Sie sich nun bitte folgenden Java Code an. Beheben Sie sämtliche eingebauten Fehler, um den Code lauffahig zu machen. Wenden Sie Ihr neu erworbenes Wissen an, um zu erklären, weshalb die Vererbung nicht so funktioniert, wie sie hier ursprünglich gedacht war.
Hinweis: Für die Bearbeitung dieser Aufgabe sollten Sie *keine* Computer-Unterstützung benutzen. Denken Sie dran: In der Klausur werden Sie ebenfalls keine Hilfsmittel wie Eclipse einsetzen können...

```
1  public final class A {
2    private int value1 = 0, value2 = 0;
3
4    private final int value3 = 0;
5
6    private int getValue1() {
7      return value1;
8    }
9
10   private int getValue2() {
11     return value2;
12   }
13
14   private void setValue1(int newValue1) {
15     value1 = newValue1;
16   }
17
18   private void setValue2(int newValue2) {
19     value2 = newValue2;
20   }
21
22   public final void changeValue3(final int newValue3) {
23     value3 = 0;
24   }
25 }
26
27 class B extends A {
28   public void changeValue3(final int newValue3) {
29     value3 = newValue3;
30   }
31
32   public static void main(String args[]) {
33     B obj = new B();
```

```
34
35      obj.setValue1(29);
36      obj.setValue2(3);
37      obj.value3 = 2010;
38
39      String result = "The final exam for GdI / ICS 1 will be on "
40    + getValue1() + "." + getValue2() + "." + obj.value3
41    + ". Please keep this in mind!";
42
43      System.out.println(result);
44    }
45  }
```

Was müssen Sie im Code ändern, um die folgende Ausgabe zu erhalten?
"The final exam for GdI / ICS 1 will be on 29.3.2010. Please keep this in mind!"

4 Wer vererbt wem was...?

Vollziehen Sie bitte nach, wie das folgende Programm arbeitet. Geben Sie die jeweilige Ausgabe des Aufrufs *Z.test()* hinter den Kommentaren der Methode test() in der Klasse Z an.
Es ist *nicht* Ziel dieser Aufgabe, dass Sie Eclipse starten, den Code einfügen und das Ergebnis einfach ablesen!

```
1   class X {
2     int a = 4;
3
4     int get() {
5       return a;
6     }
7   }
8
9   class Y extends X {
10    static int a = 7;
11
12    int get() {
13      return a;
14    }
15
16    static void set(int x) {
17      a = x;
18    }
19
20    static void set(char c) {
21      a = 2 * c;
22    }
23  }
24
25  class Z extends Y {
26    static int b = 3;
27
28    int get() {
29      return b + a;
30    }
31
32    static int get(X x) {
33      return x.a;
34    }
35
36    static void set(int i) {
37      a = 3 * i;
38     }
39
40    static void set(X x, int i) {
```

```
41      a = i;
42    }
43
44    static void test() {
45      Z z = new Z();
46
47      System.out.println(Y.a); // --------
48      System.out.println(get(z)); // --------
49
50      Z.set('c' - 'a' - 1); // ASCII Werte: 'c' = 99, 'a' = 97
51      System.out.println(get(z)); // --------
52      System.out.println(z.get()); // --------
53
54      Y y = z;
55      Y.set(2);
56      System.out.println(z.get() + 1); // --------
57      Z.set(y, 0);
58      System.out.println(y.get() + 4); // --------
59    }
60
61    public static void main(String args[]) {
62      Z.test();
63    }
64  }
```

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren. Falls Sie die Hausübung in einer Lerngruppe bearbeitet haben, geben Sie dies bitte deutlich bei der Abgabe an. Alle anderen Mitglieder der Lerngruppe müssen als Abgabe einen Verweis auf die gemeinsame Bearbeitung einreichen, damit die Abgabe im Portal auch für sie bewertet werden kann.

Abgabedatum: Freitag, 22. 01. 2010, 16:00 Uhr

Denken Sie bitte daran, Ihren Code hinreichend gemäß der Vorgaben zu kommentieren (Scheme: Vertrag, Beschreibung und Beispiel sowie zwei Testfälle pro Funktion; Java: JavaDoc). Zerlegen Sie Ihren Code sinnvoll und versuchen Sie, wo es möglich ist, bestehende Funktionen wiederzuverwenden. Wählen Sie sinnvolle Namen für Hilfsfunktionen und Parameter.

5 Sudoku (11 Punkte)

In dieser Übung wollen wir Sudoku mit Hilfe von Backtracking lösen. Ein Sudoku-Spielfeld besteht aus 3x3 Blöcken mit jeweils Platz für 3x3 Zahlen mit Werten zwischen 1 und 9[1].
Das Ziel des Spiels ist es, ein Spielfeld mit einer gegebenen teilweisen Belegung von Zahlen so zu vervollständigen, dass in jedem der 3x3-Blöocke sowie in jeder der 9 Zeilen und in jeder der 9 Spalten jede Ziffer nur ein einziges Mal auftaucht.
Zu diesem Zweck steht im Portal ein Template bereit, welches Sie nutzen und ausprogrammieren sollen.
Falls Sie Sudoku bereits kennen, können Sie den folgenden Teilabschnitt überspringen und direkt mit der Beschreibung der Aufgabe fortfahren.

[1] Siehe `http://de.wikipedia.org/wiki/Sudoku` und `http://sudoku.de`

5.1 Sudoku: Spielregeln

Wie oben beschrieben, muss bei einer korrekten Sudoku-Lösung in jedem 3x3-Block, jeder Zeile und jeder Spalte jede der Zahlen 1-9 **genau einmal** auftreten.

	0	1	2	3	4	5	6	7	8
0		9		3	6		1		4
1		7	5			2	3		
2									7
3			3	4	8				
4		4						7	
5					2	5	4		
6	8								
7			7	8			6	3	
8	6		4		3	1		2	

	0	1	2	3	4	5	6	7	8
0	2	9	8	3	6	7	1	5	4
1	4	7	5	1	9	2	3	8	6
2	3	1	6	5	4	8	2	9	7
3	7	6	3	4	8	9	5	1	2
4	5	4	2	6	1	3	8	7	9
5	1	8	9	7	2	5	4	6	3
6	8	3	1	2	7	6	9	4	5
7	9	2	7	8	5	4	6	3	1
8	6	5	4	9	3	1	7	2	8

Abbildung 1: Beispiel für eine Aufgabenstellung und ihre Lösung

Betrachten wir dazu das Beispiel für ein vorgegebenes Sudoku sowie dessen Lösung in der Abbildung. Nummerieren wir zunächst in Gedanken die Zeilen und Spalten jeweils mit 0-8 (in der Abbildung kleiner geschrieben). Der erste 3x3-Block umfasst dann die Felder mit einer Spaltenposition 0, 1 oder 2 in den ersten drei Zeilen. In diesem Block sind bereits drei Werte vorgegeben:

- die 9 an Position (0, 1),
- die 7 an Position (1, 1)
- sowie die 5 an Position (1, 2).

Für die in diesem Block noch vorhandenen freien Stellen bedeutet dies, dass nur noch die fehlenden Werte (also 1, 2, 3, 4, 6 oder 8) auf diese insgesamt 6 freien Position verteilt werden können.
Entsprechend befinden sich in Zeile 0 bereits die Werte 1, 3, 4, 6 und 9. Die vier noch leeren Positionen dürfen also nur mit den Werten 2, 5, 7 und 8 belegt werden. In Spalte 4 befinden sich schon die Werte 2, 3, 6 und 8, so dass hier nur noch die Zahlen 1, 4, 5, 7 und 9 verwendet werden können.
Betrachten wir nun Zeile 8 (ganz unten). Hier befinden sich die Zahlen 1, 2, 3, 4 und 6; es fehlen also noch 5, 7, 8 und 9. Wir wollen nun die Lücke an Position (8, 1) schließen. Dazu prüfen wir der Reihe nach alle möglichen Zahlen (zum besseren Verständnis in diesem konkreten Beispiel von der 9 abwärts):

- Die 9 kann nicht an Position (8, 1) eingefügt werden. In der gleichen Spalte - Spalte 1 - steht an Position (0, 1) bereits eine 9. Wenn wir in (8, 1) eine 9 einfügen, käme diese entsprechend in dieser Spalte doppelt vor. Die 9 scheidet also aus.
- Die 8 kann ebenfalls nicht an Position (8, 1) eingefügt werden. Im gleichen Block wie die Lücke - das heißt in den Spalten 0-2 der Zeilen 6-8 - befindet sich bereits eine 8 (an Position (6, 0)). Da jede Zahl nur einmal in einem 3x3-Block auftreten darf, scheidet die 8 aus.
- Die 7 scheidet aus dem gleichen Grund wie die 9 und die 8 aus: sie befindet sich sowohl in der gleichen Spalte—an Position (1, 1)—als auch im gleichen Block—an Position (7, 2).
- Die 5 tritt weder im 3x3-Block noch in Zeile 8 oder in Spalte 1 auf. Da die 5 die einzige gültige Möglichkeit ist für diese Position, können wir hier also eine 5 eintragen.

Das Spiel endet, wenn alle Lücken korrekt geschlossen wurden, d.h.

- Es gibt keine Lücken mehr;
- In jeder der Zeilen 0-8 tritt jede Zahl von 1-9 genau einmal auf;
- In jeder der Spalten 0-8 tritt jede Zahl von 1-9 genau einmal auf;
- In jedem der mit Doppellinien markieren 3x3-Blöcke (also jeweils Spalte 0-2 / 3-5 / 6-8 sowie Zeile 0-2 / 3-5 / 6-8) tritt jede Zahl von 1-9 genau einmal auf.

5.2 Ihre Aufgabe

Erstellen Sie ein zunächst ein neues Java-Projekt und importieren Sie das Template aus dem Portal. Binden Sie die `acm.jar` sowie die `JUnit` Bibliotheken in ihr Projekt ein, wie im Portal beschrieben. Wenn Sie alles richtig gemacht haben, sollte nach dem Ausführen bereits ein Sudoku-Spielfeld mit einigen voreingesetzten Zahlwerten erscheinen.
Hinweis: Wir verwenden zur Codierung der Feldwerte den Datentyp **int** und nutzen `0` für unbelegte Felder.
Benennen Sie nun die Klasse `SudokuTemplate` in `Sudoku` um.
Im Template wird die grafische Oberfläche aufgebaut und die Funktionalität für die verschiedenen grafischen Elemente bestimmt. In der `main`-Methode werden einige Konfigurationen des Spielfeldes vordefiniert; das Sudoku-Programm kann mit einem dieser Felder initialisiert werden. Ihre Aufgabe ist es letztlich, die Methode `int[][] solve(int[][])` zu implementieren, welche eine gültige Lösung für die übergebene Konfiguration des Spiefeldes zurückliefern soll.
Zur Lösung des Problems sollten Sie Backtracking verwenden. In jedem Schritt soll für ein aktuell betrachtetes Spielfeld eine *mögliche* (zulässige) Belegung bestimmt werden. Da sich beim weiteren Ausfüllen des Spielfeldes ergeben kann, dass eine getroffene Entscheidung falsch war, müssen wir diese Entscheidung später nochmals überprüfen.
Eine konkrete Belegung des Sudokufeldes bezeichnen wir als *Konfiguration*.
Eine Möglichkeit ist, in jedem Rekursionsschritt für das erste freie Feld alle Werte von 1-9 durchzuprobieren oder zur übergeordneten Konfiguration zurückzukehren, falls die aktuelle Konfiguration nicht gültig ist.

1. Schreiben Sie eine Methode `public boolean reject(int[][])`, welche eine gegebene Konfiguration entgegennimmt und entscheidet, ob es sich nicht mehr um einen Lösungskandidaten handelt. Eine Konfiguration ist dabei genau dann ein Lösungskandidat, wenn für die belegten Felder keine der oben genannten Sudoku-Regeln verletzt wird—also keine der Zahlen 1-9 mehr als einmal in einer beliebigen Zeile oder Spalte oder einem der 9 3x3-Blöcke auftritt. (2 Punkte).

   Hinweis: Verwenden Sie für die Sichtbarkeit der `reject` Methode bitte eine öffentliche Sichtbarkeit, um das Testen zu erleichtern.

2. Schreiben Sie eine Methode `int getNextFreeField(int[][])`, die eine Konfiguration entgegennimmt und die Position des ersten freien Feldes zurückliefert. Falls es keine freie Position mehr gibt, können Sie einfach `-1` zurückgeben (2 Punkte).

   Hinweis: Ein Feld ist frei oder leer, wenn es den Zahlwert 0 enthält.

   Hinweis: Um die Position eines Eintrags in einem zwei-dimensionalen Array auf einen einzelnen Zahlwert abzubilden, müssen Sie sich für eine geeignete Ordnung über den Einträgen entscheiden. Auch wenn hier vieles denkbar ist, hat sich bewährt, die Elemente einfach zeilenweise durchzuählen.

3. Schreiben Sie nun eine Methode `int[][] getNextExtension(int[][], int)`. Diese erwartet eine Konfiguration und eine Position und liefert einen neuen Kandidaten zurück, bei dem der Wert des Feldes um 1 erhöht wurde. Falls der neue Wert größer als 9 wäre, geben Sie einfach den leeren Wert `null` zurürck (2 Punkte).

4. Implementieren Sie nun `int[][] solve(int[][], int)`, welche einen Lösungskandidaten und die Position eines zu variierenden Feldes entgegennimmt, den übergebenen Kandidaten mit `getNextExtension` erweitert und für jeden dieser neuen Lösungskandidaten sich rekursiv mit der Position des nächsten freien Feldes aufruft. Diese Methode soll die folgende Funktionalität implementieren (4 Punkte):

   a) kein gültiger Lösungskandidat mehr? → return

   b) alle Felder belegt? → Lösung gefunden und zurückgeben

   c) Zu variierende Position ungültig? → return

   d) Für alle Erweiterungen: rufe solve mit der Erweiterung auf.

   Ändern Sie dann noch die vorgegebene Implementierung von solve(**int** [][]) so ab, dass es solve{**int** [][], **int**) "passend" aufruft. Sie können Ihre Lösung anschließend interaktiv in der grafischen Umgebung testen, wenn Sie das Spiel starten!

   Hinweis: Vergessen Sie nicht, Ihre Methoden passend zu kommentieren (1 Punkt)!