

## Übungsblatt 9 - Abgabe 03.01.05

### Aufgabe 1 (Verständnisfragen)

Achtung: Die Verständnisfragen können unmöglich den ganzen Bereich, den es im Bezug auf Spezifikation und Verifikation zu verstehen gilt abdecken. Eine systematische Arbeit von euch ist wichtig.

Entscheiden Sie, ob folgenden Aussagen jeweils RICHTIG oder FALSCH sind. Erläutern sie kurz!

- 1.) Eine Voraussetzung  $P(x) \equiv x \in M$  ist stärker, als eine Voraussetzung  $P'(x) \equiv x \in M \cup \{z\}$  mit  $z \notin M$ .
- 2.) Bei algebraischer Spezifikation interessiert uns nur das extern sichtbare Verhalten eines Objektes, welches wir über Traces, d.h. legitime Aufruffolgen samt Ergebnis spezifizieren (im funktionalen Fall spricht man natürlich besser von geschachtelten Ausdrücken).
- 3.) Bei der Vererbung von Spezifikationen wird das bisherige Modell durch das Modell der neuen Spezifikation ersetzt. Die bisherigen Voraussetzungen und Effekte beziehen sich nun auf das neue Modell.
- 4.) Ein S2-Objekt kann überall für ein S1-Objekt ersetzt werden, wenn S2 eine Erweiterung von S1 ist und wenn unter Berücksichtigung der tatsächlichen Invarianten, die Spezifikationen aller Operationen - auch der geerbten - konsistent ist.
- 5.) Typvererbung, Klassenvererbung und Verhaltensvererbung sind unabhängige Konzepte, d.h. es gibt für jedes Paar aus den drei Vererbungsarten Fälle, in denen das eine, aber nicht das andere vorliegt.
- 6.) Zum Übergang von einem abstrakten Modell zu einer konkreten Repräsentation benötigt man eine konkrete Wertemenge  $K$  (im Gegensatz zur abstrakten Wertemenge  $A$ ), eine Konsistenzbedingung  $I_K$  (im Gegensatz zur abstrakten Bedingung  $I_A$ ) und eine Abstraktionsfunktion  $\alpha$  von  $K$  nach  $A$ .
- 7.) Eine konkrete Invariante ist *nicht* korrekt für eine konkrete Repräsentation, wenn gilt:  $\exists k \in K I_A(\alpha(k)) \wedge \neg I_K(k)$ .
- 8.) Jeder Wert, der in der abstrakten Wertemenge  $A$  enthalten ist, muss sich durch genau einen Wert aus der konkreten Wertemenge  $K$  mittels der Abstraktionsfunktion  $\alpha$  abbilden lassen.
- 9.) Eine Spezifikation für eine Operation  $op$  heißt vollständig, wenn sie jedem  $a$  aus  $A$  ein  $op(a)$  aus  $A$  zuordnet.

### Aufgabe 2 (Folgen und Spezifikation)

Wir spezifizieren den ADT Keller wie folgt:

```

interface Stack<T> { // model: s :: [T]
    // inv: --
    void push(T x); // pre: --
    // post: s' == x:s
    T pop() throws StackUnderflow;
    // pre: not(null s) -- else StackUnderflow
    // post: s == result:s'
    int length(); // pre: --
    // post: result == length s && s'==s
}

```

Zur Repräsentation von Kellern, die Boole'sche Werte enthalten, ist eine Klasse `BooleanStack` gesucht:

```

class BooleanStack implements Stack<Boolean> {
    private int value = 1; // inv: value >= 1
    public void push ...
    public Boolean pop ...
    public int length ...
}

```

Der Witz bei dieser Repräsentation ist, dass der Wert eines Kellers als Integer dargestellt wird, ein Dummy-Bit dient als Endmarkierung. Nach dem Einkellern von `true`, `false`, `true` beispielsweise hat `value` den Wert 1101 binär, und ein anschließendes `pop` liefert ein `Boolean`-Objekt mit dem Wert `true` und hinterlässt 110. Ohne Endmarkierung wäre nicht erkennbar, ob der Keller leer oder mit `false`-Werten gefüllt ist.

- Wie lautet die Abstraktionsfunktion? Beweisen Sie die Korrektheit der Repräsentation!
- Spezifizieren Sie die konkreten Methoden (d.h. unter Bezugnahme auf die Repräsentation).
- Beweisen oder widerlegen Sie die Korrektheit dieser Spezifikationen!
- Lässt sich die Idee dieser Repräsentation auf einen Keller von `ints` übertragen? Begründen und erläutern Sie Ihre Antwort!

Hinweis: Verwenden Sie für a) - c) eine geeignete formale oder halbformale Schreibweise.

Wer diese Aufgabe im Vorjahr schon einmal gemacht hat, darf sich zusätzlich noch eine Repräsentation ausdenken, welche die Längenbeschränkung nicht besitzt, trotzdem speichereffizient ist und mit gleichem Laufzeitverhalten arbeitet.

### Aufgabe 3 (Verhaltensvererbung)

Die folgenden Klassen sind so einfach, dass es fast trivial ist, formale Spezifikationen dafür anzugeben; Sie sollen es trotzdem tun.

```

class Rectangle {
    protected float a,b;
    public Rectangle(float a, float b) {
        if(!(a>0 && b>0)) throw new IllegalArgumentException();
        this.a = a; this.b = b;
    }
    public void stretch(float x) {

```

```

        if(x<=0) throw new IllegalArgumentException();
        a=a*x; b=b*x;
    }
    public float length() { return 2*a + 2*b; }
}
class Square extends Rectangle {
    public Square(float a) { super(a,a); }
    public void stretch(float x) { a=a*x; }
    public float length() { return 4*a; }
}

```

- a) Geben Sie eine Spezifikation interface IRectangle für die Klasse Rectangle an (natürlich derart, dass Rectangle korrekt bezüglich dieser Spezifikation ist)!
- b) Spezifizieren Sie Square durch Angabe einer Unterspezifikation interface ISquare extends IRectangle! Liegt Verhaltensererbung vor? (Beweis!)

#### Aufgabe 4 (Praktische Javakenntnisse 1 - Ant)

Zum Kompilieren von größeren Java-Projekten wird in der Praxis gerne das "Build-Werkzeug" Ant der Apache Software Foundation eingesetzt. Ant hat sich in diesem Bereich als Standardlösung für Java etabliert, ähnlich wie make und Konsorten für das Übersetzen und Erstellen von C/C++ Projekten zuständig sind. Das Analogon zum makefile lautet bei Ant build.xml und ist eine XML-Datei, die typischerweise so aussieht:

```

<project name="simpleCompile" default="compile" basedir=".">
  <target name="init">
    <property name="sourceDir" value="src" / >
    <property name="outputDir" value="build" />
  </target>
  <target name="clean" depends="init">
    <deltree dir="${outputDir}" />
  </target>
  <target name="prepare" depends="clean">
    <mkdir dir="${outputDir}" />
  </target>
  <target name="compile" depends="prepare">
    <javac srcdir="${sourceDir}" destdir="${outputDir}" />
  </target>
</project>

```

Wie man sehen kann, besteht eine solche Build-Datei aus mehreren Zielen (Targets), Eigenschaften (Properties) und Befehlen (javac, deltree, mkdir). Ruft man jetzt Ant über die Kommandozeile mittels des Befehls ant auf, so wird das Standardziel (default) des Projektes unter Einbeziehung der Voraussetzungen (depends) erstellt. In unserem Fall würde also durch "ant" zuerst init, dann clean\*, dann prepare, dann compile durchgeführt (\*was vielleicht nicht so sinnvoll ist). Ant lässt sich beziehen von <http://ant.apache.org>. Die Beispiel Build.xml stammt von <http://www.javaworld.com/javaworld/jw-10-2000/jw-1020-ant.html>.

Als praktische Aufgabe findet ihr unter <http://www.inf.fu-berlin.de/lehre/WS05/ALP3/material/Jake2.zip> auf der Homepage die Java

Version von Quake 2, die mittels Ant von euch kompiliert werden soll. In dem Archiv findet ihr eine `build.xml`, die ihr über "`build.bat ...`" aufrufen könnt (d.h. es passiert das gleiche wie "`ant ...`"), wenn ihr kein Ant installiert habt. Leider wurde bei einigen Zielen (`default`, `compile` und `all`) die Abhängigkeiten nicht richtig gesetzt und eure Expertise ist gefragt. Repariert also die `Build.xml` und erstellt dann den Installer (`build.bat install`). Mit diesem könnt ihr dann Jake2 installieren und nach Installation der Demoversion (<ftp://ftp.fu-berlin.de/pc/msdos/games/idgames/idstuff/quake2/q2-314-demo-x86.exe>) auch Quake 2 in Java spielen. Einzureichen sind die reparierten Ziele (d.h. 3 korrigierte `depends`-Einträge; sollte auf 3 Zeilen passen). [Optional: ein Bildschirmfoto]

## Schöne Ferien!

