Übung 3: Software-Entwurf

Aufgabe A: Modularitätsmechanismen von Programmiersprachen

Welche Mechanismen und Sprachkonstrukte bietet Java (oder eine andere Ihnen vertraute Sprache) für Modularität. Diskutieren Sie insbesondere ob und wie die 5 Kriterien und die 5 Prinzipien unterstützt werden.

Aufgabe B: Bewertung eines Entwurfs

i) Gegeben ist folgendes Programm (auf der nächsten Seite), welches mathematische Ausdrücke evaluieren und ausgeben kann. Diskutieren Sie kritisch die Modularität dieser Implementierung, gehen Sie dabei auf die Kriterien und Prinzipien aus der Vorlesung ein.

ii) Welche Änderungen können modular durchgeführt werden, welche nicht?
Betrachten Sie unter anderem folgende Beispiele:

- Ein Fehler in der Textausgabe von negativen Zahlen soll korrigiert werden (zusätzliche Klammern)
- Neben Drucken und Evaluieren soll auch Vereinfachen von Ausdrücken, z.B. x+0=x und --x=x unterstützt werden.
- Neben Addition soll auch Multiplikation unterstützt werden

Aufgabe C: Veränderung des Entwurfs

i) Verschlechtern Sie das Programm aus Aufgabe B. Wie würde eine Implementierung aussehen, welche einzelnen oder allen Kriterien, Prinzipien und Regeln stärker wiederspricht.

ii) Verbessern Sie das Programm aus Aufgabe B entsprechend der Kriterien, Prinzipien und Regeln aus der Vorlesung. Erläutern Sie inwiefern ihre Implementierung besser ist. Welche Änderungen können jetzt modular durchgeführt werden, welche nicht?

Aufgabe D: Grenzen von Information Hiding

Gibt es Situationen in den Information Hiding (Datenkapselung) zwischen zwei Modulen nicht sinnvoll ist? Geben Sie ein Beispiel und erläutern Sie es.

Ihre Lösung dieses Übungszettels geben Sie bitte wie gewohnt bis zum 4.11.2012 23:59 Uhr per Email an selecture@mathematik.uni-marburg.de ab.

```
public class Main {
	public static void main(String[] args) {
		Expr e = new Add(new Add(new Number(1), new Neg(new Number(2))),
				new Number(4));

		System.out.print("Der Ausdruck ");
		new Printer().print(e);
		System.out.println(" evaluiert zu " + new Evaluator().eval(e));
	}
}

public interface Expr {}

public class Number implements Expr {
	public int value;
	public Number(int v) { this.value=v; }
}

public class Add implements Expr {
	public Expr left, right;
	public Add(Expr left, Expr right) { this.left=left; this.right=right; }
}

public class Neg implements Expr {
	public Expr expr;
	public Neg(Expr expr) { this.expr=expr; }
}

public class Printer {
	public void print(Expr e) {
		if (e instanceof Number)
			System.out.print(((Number) e).value);
		if (e instanceof Add) {
			System.out.print("(");
			print(((Add) e).left);
			System.out.print("+");
			print(((Add) e).right);
			System.out.print(")");
		}
		if (e instanceof Neg) {
			System.out.print("-");
			print(((Neg) e).expr);
		}
	}
}

public class Evaluator {
	public int eval(Expr e) {
		if (e instanceof Number)
			return ((Number) e).value;
		if (e instanceof Add)
			return eval(((Add) e).left) + eval(((Add) e).right);
		if (e instanceof Neg)
			return -eval(((Neg) e).expr);
		throw new RuntimeException("not a valid expression");
	}
}
```