

Übungen 2 - Abgabe vor der Vorlesung 01.11.05

Aufgabe 0 (Wiederholung) [optional]

Stellt sicher, dass ihr alle Begriffe aus der letzten Woche gut verstanden habt:

- Geheimnisprinzip, Prozedurale Abstraktion / Datenabstraktion
- Sichtbarkeit (von Klassen, Methoden, Attributen)
- Innere Klassen (Globale Klassen, Attributklassen, Lokale Klassen)
- Module und Pakete.
- Als konkrete Beispiele: Schlange implementiert durch Ringpuffer und Geflecht in Java und Listen in Haskell.

Wenn diese Begriffe euch eher noch Angst machen, was wollt ihr tun um die Lage zu ändern?

Aufgabe 1 (Geschachtelte Klassen)

a.) Betrachten Sie das Beispiel der Klasse `Bank` und der geschachtelten Attributklasse `Account` aus der Vorlesung:

```
public class Bank {
    private int total;
    public class Account {
        private int balance; .....
        public void deposit(int amount){
            balance += amount;
            total += amount;
        }
    }
}
```

Wie aus der Vorlesung bekannt, darf jedes Exemplar der Klasse `Account` auch auf die Attribute und Methoden desjenigen Exemplars der Klasse `Bank` zugreifen über welches der `new`-Operator ausgeführt wurde:

```
Bank geldBank = new Bank();
Bank.Account meinKonto = geldBank.new Account();
// meinKonto hat Zugriff auf geldBank.total
```

- Besteht auch die umgekehrte Möglichkeit, d.h. kennt auch das Exemplar der Klasse `Bank`, alle Konten, die über es erstellt wurden? Erläutern Sie!
- Betrachten Sie folgende Methode der Klasse `Bank`:

```
public Account newAccount(int initialBalance){
    Account result = new Account();
    result.deposit(initialBalance);
    return result;
}
```

Können Sie erklären, wieso dem `new`-Operator in diesem Fall kein Exemplar der Klasse `Bank` vorangestellt werden muss?

b.) Wandeln Sie die Klasse `Bank` (mit Erweiterungen aus a. = Methode `newAccount`) so ab, dass Sie ohne geschachtelte Klassen auskommen. Welche Vor- und Nachteile ergeben sich? Können Sie die Klassen so in einem Paket zusammenstellen, dass sich dieselbe Sichtbarkeit für die Klassen, Methoden und Attribute ergeben, wenn Sie mit Ihren Tests von außen das Paket einbinden und die Klassen benutzen?

Aufgabe 2 (Ausnahmebehandlung in Java)

Bei Ihren Tests mit der Klasse `Queue` (Übungsblatt 1 Aufgabe 5) wird Ihnen aufgefallen sein, dass die deklarierte Ausnahme der Methode `remove` sich problematisch auf die Anwendung auswirkt. So mussten Sie an aufrufender Stelle entweder die Ausnahme behandeln (mittels eines `try-catch`-Blocks) oder selbst deklarieren (mittels `throws`). Stellen Sie auf einer halben Seite eine Regel auf, wann man eine Ausnahme deklarieren sollte (mittels `throws` in der Sprache Java) und wann man eine Laufzeitausnahme (`RuntimeException`) verwenden sollte, die nicht angezeigt werden muss! Illustrieren Sie diese Regel an zwei passenden Beispielen.

Aufgabe 3 (Schnittstellen in der Praxis):

Laden Sie sich von der Homepage der Veranstaltung den Quelltext, des Malprogramms `JHotDraw` herunter und stöbern Sie nach folgenden Sachen durch den Code: <http://www.inf.fu-berlin.de/lehre/WS05/ALP3/material/JHotDrawSrc.zip>

- a.) Finden Sie eine Klasse, die 3 oder mehr Schnittstellen implementiert. Welche Aufgaben übernimmt die Klasse durch die Implementierung dieser Schnittstellen?
- b.) Finden Sie eine Schnittstelle, die von 3 oder mehr Klassen implementiert wird. Wieso gibt es so viele unterschiedliche Implementierungen? Sollte eine Implementierung nicht ausreichen?
- c.) Verschaffen Sie sich einen Überblick, ob generell mehr Schnittstellen oder mehr Klassen als Parameter und Variablentypen verwendet werden. Wie argumentieren Sie für oder wider den jeweiligen Einsatz?

Tipps:

- Erstellen Sie ein neues Eclipse-Project und ziehen Sie den Quellcodeordner "org" (dieser befindet sich in der Zip-Datei) in das Projekt.
- Verwenden Sie STRG-Links-Klick auf einen Bezeichner, um zu dessen Deklaration zu springen.
- Das Paket `org.jhotdraw.framework` ist ein sinnvoller Startpunkt.

Aufgabe 4 (Typverträglichkeit und Innere Klassen)

Finden Sie im folgenden Java-Code sämtliche Fehler auf Programmiersprachenebene (und nicht im Bezug auf Netzprogrammierung) und erklären Sie, was hier falsch gemacht wurde.

```
7 interface Socket {
8     int send(Buffer out, int howMuch);
9
10    int receive(Buffer in, int maxReceive);
11 }
12
13 class NetworkAdapter {
14     private int sendBytes;
15
16     private TcpSocket s = new TcpSocket();
17
18     class TcpSocket implements Socket {
19         int send(Buffer buf, int howMuch) {
20             // .. Interessiert uns hier nicht
21             return howMuch;
22         }
23
24         int receive(Buffer in, int max) {
25             // .. Hier uninteressant
26             return max;
27         }
28     }
29
30     static class UdpSocket {
31         private boolean recordStatistics;
32
33         int send(Buffer out, int howMuch) {
34             // .. Interessiert uns hier nicht
35             if (recordStatistics)
36                 sendBytes += howMuch;
37             return howMuch;
38         }
39
40         int receive(Buffer in, int maxReceive) {
41             // .. Hier uninteressant
42             return maxReceive;
43         }
44     }
45 }
46
47 class Main {
48     public static void main() {
49         NetworkAdapter a = new NetworkAdapter();
50         NetworkAdapter.TcpSocket b = new NetworkAdapter.TcpSocket();
51         NetworkAdapter.UdpSocket c = new NetworkAdapter.UdpSocket();
52         Socket i = new Socket();
53         i = c;
54         i = a.s;
55         i = (new NetworkAdapter()).new TcpSocket();
56     }
57 }
```

Aufgabe X (Java Wiederholung) [optional]

Alle, die noch ein bisschen mit Java kämpfen, finden unter <http://projects.mi.fu-berlin.de/w/bin/view/SS/JavaWiederholung> ein paar kleinere Aufgaben zur Wiederholung. Für diese werden nach dem Abgabetermin Testfälle veröffentlicht, so dass ihr überprüfen könnt, ob Ihr eine korrekte Lösung implementiert habt.

Java 1.5 Exkurs-Folge 1 - Var-Args

Mit Java 1.5 wurde es möglich eine variable Anzahl von Parametern an eine Funktion zu übergeben. Hierzu dient folgende Syntax:

```
public void function(TypName...VariablenNamen)
```

Der Übersetzer interpretiert dies nahezu identisch zu:

```
public void function(Type[] variable)
```

Hier ein konkretes Beispiel, dass die Unterschiede klar macht:

```
19 public static void smallCapsPrinterNoVarArgs(String[] input) {
20     for (String s : input)
21         System.out.println(s.toLowerCase());
22 }
23
24 public static void smallCapsPrinter(String... input) {
25     for (String s : input)
26         System.out.println(s.toLowerCase());
27 }
28
29 public static void main(String[] args) {
30
31     String[] input = { "Hallo", "HIER KÖNNEN",
32                       "beliebig viele Strings stehen" };
33     smallCapsPrinterNoVarArgs(input);
34
35     smallCapsPrinter("Hallo", "HIER KÖNNEN",
36                     "beliebig viele Strings stehen");
37     // -> Ausgabe: "hallo\nhier können\nbeliebig viele strings stehen\n"
38
39     smallCapsPrinter(input);
40     smallCapsPrinter();
41
42     smallCapsPrinterNoVarArgs("Hallo", "HIER KÖNNEN",
43                             "beliebig viele Strings stehen");
```

Wie man sieht sind VarArgs flexibel und funktionieren auch mit Feldern oder leerer Eingabe, während bei der Benutzung des Feldes vorher umständlich ein Feldinitialisierer verwendet werden muss. Eine wichtige Einschränkung existiert aber: Es darf nur ein VarArgs-Parameter geben und dieser muss der letzte in der Parameterliste sein.

Wer Lust hat kann damit mal folgende Aufgabe lösen:

Schreiben Sie eine Funktion `ggT`, welche den größten gemeinsamen Teiler der übergebenen Zahlen ermittelt. Die Funktion soll so gestaltet sein, dass der Übersetzer statisch feststellen kann, ob eine gültige Anzahl von Parametern übergeben wurden ($n \geq 2$).