

Übungsblatt 5 - Abgabe vor der Vorlesung 22.11.05

Aufgabe 0 (Wiederholung Vererbung) [Optional]

- Begriffe: Generalisierung/Spezialisierung, Oberklasse/Unterklasse, direkt/indirekt, Mehrfachvererbung
- primitiv UML
- Typverträglichkeit
- Namenskollision: Verdecken, Ersetzen (overloading/overriding), Zugriff (this/super, dynamischer Typ, statischer Typ, final)
- Polymorphie
- Kovarianz/Kontravarianz
- Konstruktoren
- Abstrakte Klassen
- Object (clone, deep/flat copy, toString, finalize)

Nehmen Sie sich die Wiederholung dieser Themen besonders zu Herzen, da Vererbung und Polymorphie die Kernstücke der Objektorientierung sind.

Aufgabe 1 (Namenskollision und Polymorphie)

Welche der folgenden Behauptungen sind korrekt? Erklären Sie jeweils in einem Satz!

- 1) Attribute und statische Methoden lassen sich in Java nur verdecken und nicht ersetzen.
- 2) Will man beim Ersetzen von nicht-statischen Methoden den Rückgabotyp ändern, so geht dies nur, wenn der neue Rückgabewert Obertyp des Typs des bisherigen Rückgabewertes ist.
- 3) Beim Ersetzen müssen mindestens immer alle Ausnahmen der überschriebenen Methode (bzw. Untertypen von diesen) deklariert werden.
- 4) Da Java keine Mehrfachvererbung bei Klassen erlaubt, kann es keine Kollisionen zwischen den Implementierungen von Methoden, sondern immer nur zwischen Signaturen geben.
- 5) Durch Implementierung von Interfaces kann es zu horizontalen Namenskollisionen im Bezug auf Attribute kommen, welche man aber durch explizite Bezugnahme auflösen kann.
- 6) Eine innere Klasse innerhalb einer umschließenden Klasse entspricht einer Vererbungsbeziehung zwischen den beiden.

- 7) `this` und `super` dürfen nur in nicht-statischen Methoden verwendet werden, da an allen anderen Orten kein Bezug auf ein Exemplar besteht.
- 8) Ein mit `final` markiertes Attribut darf bei der Vererbung nicht verdeckt werden.
- 9) Hat eine Oberklasse keinen Konstruktor, dann kann man in Java keine Unterklasse von ihr ableiten.
- 10) Es kann keine Variablen, Parameter oder Attribute vom Typ einer abstrakten Klasse geben.

Aufgabe 2 (Polymorphie in der Praxis)

Durchsuchen Sie JHotDraw mit Ihrem neuen Verständnis zum Thema Vererbung nach folgenden Punkten:

- Finden Sie drei Methoden, die überladen (overloaded) sind. Worin unterscheiden sich die verschiedenen Versionen?
- Finden Sie drei Methoden, die ersetzt (overridden) wurden. Wieso war es nötig die alte Version zu ersetzen? Kontrastieren Sie mit diesem Wissen kurz die Begriffe überladen und ersetzen.
- Finden Sie eine abstrakte Klasse, die sowohl abstrakte als auch bereits implementierte Methoden besitzt.

Tipps: In Eclipse gibt es die Ansicht "Type Hierachy" (Window - Show View - Hierarchy) , die Ihnen hierbei viel helfen kann. Ziehen Sie hierzu eine Klasse mit der Maus in die Ansicht. Im oberen Bereich der Ansicht wird dann die Klassenhierarchie (d.h. Ober und Unterklassen angezeigt) und im unteren Bereich erscheinen Attribute und Methoden. Sie werden sicherlich herausfinden, was die vorangestellten Icons jeweils bedeuten (z.B. wird Ersetzung prominent angezeigt).

Aufgabe 3 (Vererbung)

Bei Aufräumarbeiten auf Ihrer Festplatte haben Sie gerade eine antike Jar-Datei gefunden, welche zwar in Java mit Hilfe von Klassen geschrieben wurde, aber auf Vererbung und Schnittstellen vollständig verzichtet. Mit Ihrem neuen Wissen über Vererbung stürzen Sie sich also direkt in die Arbeit das Programm wesentlich zu vereinfachen, eleganter und erweiterbarer zu machen. Um sicher zu gehen, dass erstens die alte Lösung überhaupt richtig ist und sich bei der Umgestaltung keine Fehler einschleichen, sollte man vor der Änderung einen Satz Testfälle erstellen und fortlaufend prüfen, ob bei diesem sogenannten "Refactoring" die Funktionalität und die Schnittstellen erhalten bleiben.

Jar: <http://www.inf.fu-berlin.de/lehre/WS05/ALP3/material/U5.A3.jar>

Aufgabe 4 (Kollisionen)

Wie Sie sicher gesehen haben, beschäftigt sich eine der Aufgaben beim [Wettbewerb der Gesellschaft für Informatik](#) mit der Berechnung des größten

gemeinsamen Teilers (ggT) verteilt auf mehrere Computer. Als Einstieg hat ein eifriger Programmierer sich auch direkt an eine Implementierung für mehrere Zahlen gemacht ($ggT(a,b,c) = ggT(ggT(a,b),c)$). Problem bei der Sache: Seine Implementierung des ggT ist leider fehlerhaft und er hat diese inklusive Testfall bereits abgeschickt. Sie haben jetzt die Chance die Situation zu retten, indem Sie geschickt eine Unterklasse von A erstellen (Änderungen an A sind nicht erlaubt) und an dem Testfall nur ein einziges Zeichen ändern, damit dieser korrekt wird.

```
public class A {
    int ggt = -1;

    public int ggt(int a) {
        if (ggt == -1)
            ggt = a;
        else
            ggt = internalGgt(ggt, a);
        return ggt;
    }

    protected int internalGgt(int a, int b) {
        return a ^ b;
    }
}

public class TestA extends TestCase {

    public void testGGT(){
        A a = new A();
        assertEquals(150, a.ggt(150));
        assertEquals(150, a.ggt(300));
        assertEquals(50, a.ggt(350));
        assertEquals(1, a.ggt(7));
    }
}
```

Wer nicht mehr weiß, wie man den ggT implementiert, wird im Netz und in den ALP1/2 Unterlagen sicherlich fündig werden.

Java Wiederholung - Folge 5

Wieder gibt es unter <http://projects.mi.fu-berlin.de/w/bin/view/SS/JavaWiederholung> neue Java-Aufgaben, welche Mittwochs in Raum 055 um 16ct besprochen werden.

Testen mit JUnit Kurzübersicht

JUnit ist ein Testrahmenwerk für Java. In Eclipse ist Unterstützung für JUnit direkt eingebaut. Um eine Klasse zu testen geht man folgendermaßen vor

- Erstellen eines Testfalls (eine Klasse die von junit.framework.TestCase erbt).
- Implementierung von Methoden mit Name public void testXXXX.
- In diesen Methoden kann man auf Methoden mit Namen assertZZZZ zugreifen z.B. assertTrue, assertEquals.
- Anschließend lässt sich der Testfall ausführen (in Eclipse mittel Run -> As JUnit Test") und berichtet explizit über alle gescheiterten Zusicherungen.

Im Gegensatz zu klassischem Debugging, wo eine Ausgabe immer nur manuell vom ausführenden getestet wird, bleibt der Testfall also auch für später erhalten.