

Übungsblatt 4 - Abgabe vor der Vorlesung 15.11.05

Aufgabe 0 (Wiederholung) [Optional]

- Ausnahmen (throw, catch, finally, throws, Behandlung, Laufzeitausnahmen, Ausnahmekonzept durch andere Mittel realisieren (Sprungmarken))
- Polymorphie
- Vererbung (((in)direkte)Unter-/Oberklasse, Spezialisierung/Generalisierung, Vererbungshierarchie)

Aufgabe 1 (Vererbung)

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

- 1) Eine Unterklasse ist eine Spezialisierung einer Oberklasse.
- 2) Methoden und Attribute der Oberklasse stehen in der Unterklasse zur Verfügung, wenn sie nicht als `private` deklariert wurden.
- 3) Eine Klasse kann in Java von mehreren Oberklassen erben.
- 4) Vererbung wird in Java durch das Schlüsselwort `inherit` durchgeführt.
- 5) Eine Schnittstelle kann in Java von mehreren Oberschnittstellen erben.
- 6) Bei der Vererbung werden auch Schnittstellen mitvererbt (mit allen Rechten und Pflichten).
- 7) Konstruktoren und statische Methoden werden in Java vererbt.
- 8) Ist die Klasse X Oberklasse von Y, so darf an jeder Stelle, an der ein Ausdruck vom Typ X erwartet wird, auch einer vom Typ Y stehen.
- 9) Klasse A ist mit Klasse B verträglich, wenn A Unterklasse von B ist.

Was ist der Unterschied zwischen Vererbung bei Klassen und der Implementierung von Schnittstellen durch Klassen? Wieso sind beide Mechanismen getrennt wichtig? Führen Sie die Antwort dieser Fragen auf einer halben Seite aus.

Vervollständigen Sie folgende Tabelle "Wenn eine Methode oder ein Attribut durch den Modifizierer ... gekennzeichnet ist, dann ist sie/es sichtbar in":

	Klasse selbst	Paket der Klasse	Unterklasse	anderswo
<code>private</code>				
kein Modifizierer				
<code>protected</code>				
<code>public</code>	J	J	J	J

Aufgabe 2 (Typverträglichkeit, Statischer/Dynamischer Typ)

Zur Erweiterung der Bank darf eine ordentliche Ausstattung an Geldautomaten natürlich nicht fehlen. Erstellen Sie für folgende Klassen ein UML-Diagramm in Anlehnung an die Folien (z.B. 2.1 S. 13). Achten Sie auf die Feinheiten! Finden Sie anschließend alle statischen Typfehler in folgender main-Methode.

```
public interface PinKeyboard { /*...*/ }

public interface QwertyKeyboard { /*...*/ }

public class Automat { /*...*/ }

public class GeldAutomat extends Automat
    implements PinKeyboard { /*...*/ }

public class KontoauszugsDrucker extends Automat
    implements PinKeyboard { /*...*/ }

public class UeberweisungsAutomat extends KontoauszugsDrucker
    implements QwertyKeyboard { /*...*/ }

public static void main(String[] args){

    Automat a = new Automat();
    GeldAutomat g = new GeldAutomat();
    KontoauszugsDrucker k = new KontoauszugsDrucker();
    UeberweisungsAutomat u = new UeberweisungsAutomat();
    PinKeyboard p = new PinKeyboard(){
        public void enterPin(int pin) {
            System.out.println("Anonymous PinKeyboard: " + pin);
        }
    };
    QwertyKeyboard q = new QwertyKeyboard(){
        public void pressKey(char c) {
            System.out.println("Anonymous QwertyKeyboard" + c);
        }
    };

    a = u = k;
    k = u;
    g = u;
    g = (GeldAutomat)u;
    u = (UeberweisungsAutomat)g;
    p = k; q = k;
    q = (UeberweisungsAutomat)k;
    p = q; q = p;
    p = (PinKeyboard)q;
    q = (QwertyKeyboard)p;
}
```

Aufgabe 3 (Polymorphie)

Vollziehen Sie nach, was in folgendem Java-Code passiert und geben Sie die Ausgabe des Programms ab. Erklären Sie, was passiert.

```
class X {
    int a = 4;
    int get() { return a; }
}

class Y extends X {
    static int a = 7;
    int get() { return a; }
    static void set(int x) { a = x; }
    static void set(char c) { a = 2*c; }
}

class Z extends Y {
    static int b = 3;
    int get() { return b+a; }
    static int get(X x) { return x.a; }
    static void set(int i) { a = 3*i; }
    static void set(X x, int i) { a = i; }

    static void test() {
        Z z = new Z();
        print(z.a);
        print( get(z) );
        print( ((X)z).get() );
        z.set('c'-'a'-1);
        print( get(z) );
        print( z.get() );
        Y y = z;
        y.set(2);
        print( z.get() );
        z.set(y, 0);
        print( y.get() );
    }
}
```

Aufgabe 4 (Ausnahmebehandlung)

Welche der folgenden Behauptungen sind korrekt? Erklären Sie jeweils in einem Satz (Aufgaben von MaTA RWTH)!

- 1) Alle Ausnahmen, die eine Methode meldet (direkt oder indirekt), aber nicht behandelt, müssen im Methodenkopf deklariert werden.
- 2) Fehler in Konstruktoren sollten nicht mit Hilfe von Ausnahmen behandelt werden.
- 3) Der `finally`-Block in einer Methode wird auch dann ausgeführt, wenn diese wegen einer nicht gefangenen Ausnahme abgebrochen wird.
- 4) Eine in `main` aufgetretene, aber nicht behandelte Ausnahme führt zum Programmabsturz.

- 5) Durch Deklaration der Ausnahme aus 4) im Kopf von `main` lässt sich ein solcher Absturz verhindern.
- 6) Selbstdefinierte Ausnahmen werden bei der Übersetzung und zur Laufzeit anders behandelt als vordefinierte Ausnahmen aus `java.lang`.

Was gibt folgendes Programm aus und wieso brauchen `a1` und `a2` keine Ausnahmedeklaration in der Signatur, `a3` und `main` aber schon?

```
public static void a1(int start, int end) {
    int failures = 0;
    for (int i = start; i < end; i++){
        try {
            if (i % 4 == 0 || 4 / i == -1)
                throw new RuntimeException("A1: NO FOO!");
            if (i % 2 == 0)
                throw new RuntimeException("A1: NO BAR!");
        } catch (RuntimeException e) {
            System.out.println("A1: Try failed: "
                + e.getMessage());
            failures++;
            if (failures >= 3)
                throw new RuntimeException("A1: 3 times");
        } finally {
            System.out.println("A1: Done with " + i);
        }
    }
    throw new RuntimeException("A1: Catch me");
}

public static void a2() {
    a1(3, 15);
}

public static void a3() throws Exception {
    try {
        a1(0,2);
    } catch (Exception e) {
        System.out.println("A3: A1: " + e.getMessage());
        throw e;
    }
}

public static void main(String[] args) throws Exception {
    try {
        try {
            a2();
        } catch (Exception e) {
            System.out.println("Main: A2 threw exception: " + e.getMessage());
        }
        a3();
    } catch (Exception e) {
        System.out.println("Main: An exception occurred:");
        e.printStackTrace(System.out);
        System.out.println("Main: I am going to rethrow it now!");
        throw e;
    } finally {
        System.out.println("Bye");
    }
}
```

Beantworten Sie die Frage auf den Vorlesungsfolien ALP2-1.7 Seite 20: Könnte man anstelle eines Marken-Arguments auch ein Prozedur-Argument einsetzen? Begründen Sie kurz!

Stellen Sie nun rückblickend sicher, dass Sie die Aufgabe und Lösung zu Ausnahmen auf dem zweiten Übungsblatt verstanden haben.

Java 1.5 Exkurs-Folge 2 - Aufzählungstyp enum

Der Aufzählungstyp kann verwendet werden, wenn eine Reihe von Konstanten zu einer Einheit zusammengefasst werden sollen. Beispiel:

```
public final int SPRING = 0;
public final int SUMMER = 1;
public final int FALL    = 2;
public final int WINTER = 3;
```

Dieser Ansatz ist nicht elegant (z.B. fehlender Namensraum) und führt leicht zu Fehlern:

```
int season = SPRING;
int season = 42; // Korrekt Anweisung (keine statische Typsicherheit)
```

Besser mit enum:

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

Bessere Ausgabe:

```
int season = SUMMER;
System.out.println(season); // 1

Season season = Season.SUMMER;
System.out.println(season); // "SUMMER"
```

Einfaches Einlesen:

```
Season season = Season.valueOf("SPRING");
```

Die switch-Anweisung und if funktioniert mit Enums:

```
if (season == Season.SUMMER) advertiseIceCream();
switch (thisSeason) {
    case SUMMER: swimming(); break; // Achtung: nicht Season.SUMMER!
    case WINTER: skiing(); break;
    default: ...
}
```

Wer mehr wissen will, kann nachlesen unter:

<http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html>

Es gibt viele Feinheiten, die wir hier nicht erläutern können (z.B. Enums sind Klassen (final, public, static), implementieren Comparable und haben keine öffentlichen Konstruktoren). Hier noch ein Beispiel, was alles möglich ist:

```
public enum Coin {
    // Enums können Attribute haben
    private final int value;
    // Konstruktoren sind erlaubt, dürfen aber nicht öffentlich sein
    Coin(int value) { this.value = value; }
    // Jeder Aufzählungswert ruft wirklich den Konstruktor auf
    PENNY(1), NICKEL(5), DIME(10),
    // und jeder Wert darf auch noch eigene spezielle Methoden haben.
    QUARTER(25) { String toString(){ return "1/4" } };
    // Und natürlich kann man auch ganz normal Methoden definieren.
    public int value() { return value; }
}
```

Java Wiederholung - Folge 3

Wie bereits auf der Mailingliste angekündigt, gibt es unter <http://projects.mi.fu-berlin.de/w/bin/view/SS/JavaWiederholung> neue Java-Aufgaben.