

Fragen, Antworten, Kommentare zur aktuellen Vorlesung

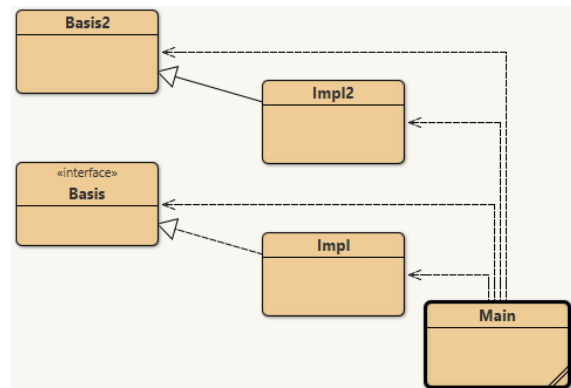
Die Online-Befragung zur genutzten alternativen Veranstaltungsform und zur Lehrevaluation ist online. Bitte ausfüllen: <https://forms.gle/MLCEGPeBxFZMGbcj9>. Sie werden eventuell aufgefordert sich bei Google anzumelden, das ist nur notwendig, wenn Sie in der Bearbeitung eine Pause machen wollen und das Teilergebnis zwischenspeichern wollen. Die Befragung endet am 22.12., die Ergebnisse stehen in einem nachfolgenden Fragen&Antworten-Dokument auf der Webseite der Veranstaltung.

Frage: In Aufgabe 36 fällt der Begriff Interface, hatten wir das schon?

Antwort: Sorry, stimmt, das kommt erst am Mo. Stellen Sie sich ein Interface als eine Klasse vor, die Methoden vorgibt, die alle eine leere Implementierung haben. Wenn Sie von dieser Klasse „erben“ wird statt dem Schlüsselwort extends das Schlüsselwort implements benutzt. Es ist dann sicherzustellen, dass alle Methoden aus dem Interface realisiert werden (oder sie müssen abstract sein). Das folgende Beispiel zeigt beide Konzepte.

```
public interface Basis {
    public int gib42(); // neu: keine Implementierung angegeben
}
```

```
public class Impl implements Basis{
    @Override
    public int gib42(){
        return 42;
    }
}
public class Basis2 {
    public int gib41(){
        return 0;
    }
}
```



```
public class Impl2 extends Basis2{
    @Override
    public int gib41(){
        return 41;
    }
}
```

Die Klassen können in einem Beispielprogramm genutzt werden.

```
public class Main {
    public static void ausfuehren(){
        Basis b1 = new Impl();
        Basis2 b2 = new Impl2();
        System.out.println(b1.gib42() + " " + b2.gib41());
    }
}
```

Die Ausgabe ist:

42 41

Hinweis: Unter <https://youtu.be/by9k47IUIAU> finden Sie die Erklärung einer Beispiellösung zu den Teilaufgaben `alleWerteGleichOft(.)` und `gleicheWerte(.)` der Messreihe mit Erklärungen zu möglichen Ansätzen.

Es handelt sich nebenbei um keine Musterlösungen, da bereits jetzt eine Komplexität erreicht wird, dass es verschiedene, teilweise gleich gute Lösungen gibt. Ein konkretes Beispiel ist die individuelle Überprüfung von null-werten. (Das hat für Lehrende nebenbei den positiven Seiteneffekt, dass Lösungskopien schnell erkannt werden). Musterlösungen haben zwei negative Effekte. Zunächst wie angedeutet, dass kreative Lösungen eventuell nicht als gut erkannt werden. Ein zweiter Effekt ist ein Sammlungseffekt bei Studierenden und sie nur gestapelt werden, da gehofft wird, die Aufgaben am Ende schon zu verstehen. Bedenken Sie immer, dass der Schritt von der Fähigkeit ein Programm zu lesen, die fast alle erreichen zum selbständigen Schreiben von Programmen ein riesengroßer ist, der nur durch intensive individuelle Übung machbar wird.

In den Lösungen zeige ich einen systematischen Ansatz und versuche möglichst unsere Coding Guidelines zu erfüllen, die auf der Veranstaltungsseite stehen. Mit etwas mehr Programmierer- und Software-Engineering-Erfahrungen würden die Lösungen teilweise anders aussehen, aber das ist Thema Ihrer nächsten Semester.

Denken Sie daran, dass es immer Möglichkeiten gibt, über Ihre individuellen Lösungen zu diskutieren. Dies ist u. a. Teil des zweiten Praktikumstermins. Sie können aber auch zur Vorlesungszeit vorbeischauchen oder mir eine E-Mail schreiben. Meine Betreuungszeit ist während der Vorlesungszeit aktuell noch nicht ausgelastet.

Hinweis: Ein gerne gemachter objektorientierter Fehler wird hier am Beispiel von `gleicheWerte(.)` gezeigt, dabei wird eine bereits vorher geschriebene Hilfsmethode `vorkommenVon()` genutzt, die die Anzahl der Vorkommen eines Wertes zählt.

```
public int vorkommenVon(Messreihe mr, int wert) { // sehr schlecht!!!
    if (mr == null || mr.getMesswerte() == null) {
        return 0;
    }
    int ergebnis = 0;
    for (int i: mr.getMesswerte()) {
        if (i == wert) {
            ergebnis = ergebnis + 1;
        }
    }
    return ergebnis;
}

public boolean gleicheWerte(Messreihe m) {
    if (this.messwerte == null || this.messwerte.isEmpty()) { // oder .size() == 0
        return m == null || m.getMesswerte() == null
            || m.getMesswerte().isEmpty();
    }
    if (m == null || m.getMesswerte() == null) {
        return false;
    }
    for (int i: this.messwerte) {
        if (m.vorkommenVon(m, i) == 0) {
            return false;
        }
    }
}
for (int i: m.getMesswerte()) {
```

```

        if (this.vorkommenVon(this, i) == 0) {
            return false;
        }
    }
    return true;
}

```

Generell läuft die Lösung, aber der Ansatz der Methode `vorkommenVon()` ist falsch, da hier als erster Parameter eine Messreihe (genauso schlecht eine `ArrayList`) übergeben wird. Wir haben damit in der Klasse `Messreihe` eine Methode geschrieben, die eine andere Messreihe verarbeiten soll. Betrachtet man die Idee von `vorkommenVon()` aber genauer, bezieht sich der Aufruf immer genau auf die Messreihe, deren Methode aufgerufen wird. Daraus folgt das der Parameter hier überflüssig ist. Aus Sicht der Objektorientierung ist das ein massiver Fehler, den man genau einmal machen darf. Die Korrekturen sehen wie folgt aus.

```

public int vorkommenVon(int wert) {
    if (this.messwerte == null) {
        return 0;
    }
    int ergebnis = 0;
    for (int i: this.messwerte) {
        if (i == wert) {
            ergebnis = ergebnis + 1;
        }
    }
    return ergebnis;
}

```

```

public boolean gleicheWerte(Messreihe m) {
    if (this.messwerte == null || this.messwerte.isEmpty()) {
        return m == null || m.getMesswerte() == null
            || m.getMesswerte().isEmpty();
    }
    if (m == null || m.getMesswerte() == null) {
        return false;
    }
    for (int i: this.messwerte) {
        if (m.vorkommenVon(i) == 0) {
            return false;
        }
    }
    for (int i: m.getMesswerte()) {
        if (this.vorkommenVon(i) == 0) {
            return false;
        }
    }
    return true;
}

```

Frage: Können oder sollen wir nicht Eclipse, Git und CodeTogether nutzen?

Dies sind sehr wichtige Werkzeuge im Laufe des Studiums, aber aus meiner Sicht frühestens ab dem zweiten Semester. Generell haben meine Erfahrungen in ersten Semestern gezeigt, dass jedwedes weitere nützliche Werkzeug dazu führt, dass unsichere Studierende noch unsicherer werden, da sie zusätzlich das neue Tool verstehen müssen und so oft nicht zum Mehrwert kommen. Jede kleine zusätzliche Stufe führt zum Verlust von Personen am Anfang des Studiums.

Eclipse schauen wir kurz am Ende der Vorlesung an. BlueJ ist das einzige Werkzeug mit dem ein einfacher sauberer Einstieg in die Objektorientierung funktioniert, was bei Studierenden zu besseren Lernerfolgen führt (Erfahrungen der Entwickelnden von BlueJ und auch meine Erfahrung). IntelliJ sieht besser aus, kann im Wesentlichen nicht mehr als Eclipse und erfüllt die Installationsanforderungen für die identische Form für die Hochschule und KleukersSEU nicht.

Git ermöglicht es systematisch verschiedene Versionen der gleichen Software zu verwalten und zugänglich zu machen. Das unterstützt einzelne Personen und Gruppenarbeiten mit gemeinsamem Git-Zugang. Git steht auf meiner Empfehlungsliste zum Selbststudium zwischen erstem und zweitem Semester.

CodeTogether ermöglicht, dass mehrere Personen gemeinsam in einem Editor Programmcode lesen und parallel bearbeiten können (live sharing). Das ist sehr cool und kann bei einer organisierten Zusammenarbeit sehr interessant sein. CodeTogether arbeitet dabei z. B. über Werkzeuggrenzen hinaus und eine Teilnahme im Browser kann sogar möglich sein. Ein Einführungstext steht in <https://dzone.com/articles/remote-pair-programming-with-intellij-eclipse-and>. Wieder ein tolles Programm, wenn alle Personen ein gewisses Programmier-Niveau erreicht haben. In den ersten beiden Semestern steht bewusst aber der Kampf „ein Mensch und eine Programmieraufgabe“ im Mittelpunkt und soll durch Gruppenarbeiten nur sinnvoll flankiert werden. Es ist auch möglich KleukerSEU um weitere Plugins zu erweitern.

Frage: Kann es nicht mehr Unterstützung für MacOS geben?

Kurze Antwort: nein. Das Praktikum findet in der Hochschule unter Windows statt, als Bonus-Service wird die Umgebung auch für zu Hause nutzungsbereit angeboten. Später in Unternehmen geben die typischerweise das zu nutzende System vor. Jedes Betriebssystem mehr erhöht weiterhin den Arbeitsaufwand bei der Erstellung der SEU, da eventuelle Abhängigkeiten und Einschränkungen beachtet werden müssen.

MacOS ist eventuell für die reine Nutzung einfacher zu bedienen, für die Entwicklung in einer beliebigen Software mit Sicherheit nicht. Apple versucht diese Sprachenvielfalt zu verhindern, da Apple-Tools mit Apple-Sprachen genutzt werden sollen. Um andere Sprachen, wie Java zu nutzen, ist ein recht genaues Wissen über Installationspfade und Systemeinstellungen notwendig, die typischerweise über die Konsole erfolgen und zumindest nicht einfacher als in Windows sind. Starten programmierunerfahrene Studierende mit MacOS ist das wie beim Start eines Marathonlaufes mit zusammengeknoteten Schnürsenkeln loszulaufen. Das Ziel bleibt erreichbar, es muss aber einiges aufgeholt werden.