Fragen, Antworten und Kommentare zur aktuellen Vorlesung

Bitte nehmen Sie an der anonymen Lehrevaluation unter https://forms.gle/erJ3qKAscLnbcKkU6 teil. Die von mir kommentierten Ergebnisse stehen im letzten Fragen&Antworten-Dokument des Semesters.

Zum Blatt 8 befinden sich Hinweise in ILIAS (Vorlesungsbereich).

Frage: Wir haben eine Frage bezüglich Polymorphie in einer Liste. Wir gehen von folgenden Klassen aus.

```
public class A {
    public A() {
    }
}
```

```
public class B extends A {
    private int x;

    public B() {
        super();
    }

    public int getX() {
        return this.x;
    }
}
```

```
public class C extends A {
    private int y;

    public C() {}

    public int getY() {
        return this.y;
    }
}
```

Gegeben ist eine Liste, welche die Oberklasse A verwaltet: `List<A> liste;`

Einfügen einer Spezialisierung von A: `liste.add(new B());`

Wie kann man dann polymorph erreichen, dass man in der liste an die Variable (private int x & private int y) der Unterklasse kommt

```
for (A a : liste) {
     //a.getX(); geht nicht
     //a.getY(); ""
}
```

Man kann natürlich in der Klasse A Methoden hinzufügen, welche dann in den Unterklassen übeschrieben werden. Wir gehen hier aber davon aus, dass A alleinstehend als Klasse funktioniert und diese (sonst überflüssigen) Methoden nicht implementieren muss. Oder wäre das hier für diese Funktionalität dann einfach notwendig? Gibt es da eine schönere Möglichkeit (evtl. mit Pattern) in der liste<A> an die Methoden getY() & getX() zu kommen?

Antwort: Evtl. nicht notwendig, aber definitiv die beste von Ihnen skizzierte Lösung ist, die Methoden in A abstract zu deklarieren. Denken Sie daran, wenn Sie eine Variable vom Typ List<A> deklarieren, behaupten Sie, dass Sie nur die A-Eigenschaften der Elemente nutzen wollen.

In seltenen Fällen, wie neu entwickelnden Frameworks, können Sie natürlich casten, was generell zu vermeiden ist. In Java gibt es die genaue Typprüfung mit

```
if (a.getClass() == B.class) // casten von a auf B
```

Wenn man Klassen ändern kann, könnte hier ein Visitor (ist später in VL) helfen, jede relevante Klasse muss dazu einen Visitor akzeptieren, hier die Kurzversion.

```
public interface Visitee {
  public int accept(Visitor v);
}

public abstract class A implements Visitee{ ...
```

```
public class B extends A {
  private int x;

  public B() {
      super();
  }

  public int getX() {
      return this.x;
  }

  @Override
  public int accept(Visitor v) {
      return v.visit(this);
  }
}
```

```
public class C extends A {

  private int y;

  public C() {
  }

  public int getY() {
      return this.y;
  }

  @Override
  public int accept(Visitor v) {
    return v.visit(this);
  }
}
```

eine Klasse für Berechnungen:

```
public class Visitor {

  public int visit(B b) {
    return b.getX();
  }

  public int visit(C c) {
    return c.getY();
  }
}
```

Nutzung:

```
List<A> listea = ...
Visitor visit = new Visitor();
for(A el:listea) {
  int wert = el.accept(visit);
  ...
}
```

Als weiterer Ansatz noch etwas Spielerei, generell sind Adapter auch einsetzbar (+ etwas Strategy)

```
public interface Adapter {
  public int getWert();
}

public class AdapterB
             implements Adapter {
  private B b;

  public AdapterB(B b) {
    this.b = b;
  }

  @Override
  public int getWert() {
    return this.b.getX();
  }
}

public class AdapterC
             implements Adapter {
  private C c;

  public AdapterC(C c) {
    this.c = c;
  }

  @Override
  public int getWert() {
    return this.c.getY();
  }
}
```

```
/ irgendjemand muss den Adapter anwenden, evtl. Klassen selbst
// in A
  public abstract Adapter adapt();

// in B
  @Override
  public Adapter adapt() {
    return new AdapterB(this);
  }

// in C
  @Override
  public Adapter adapt() {
    return new AdapterC(this);
  }
```

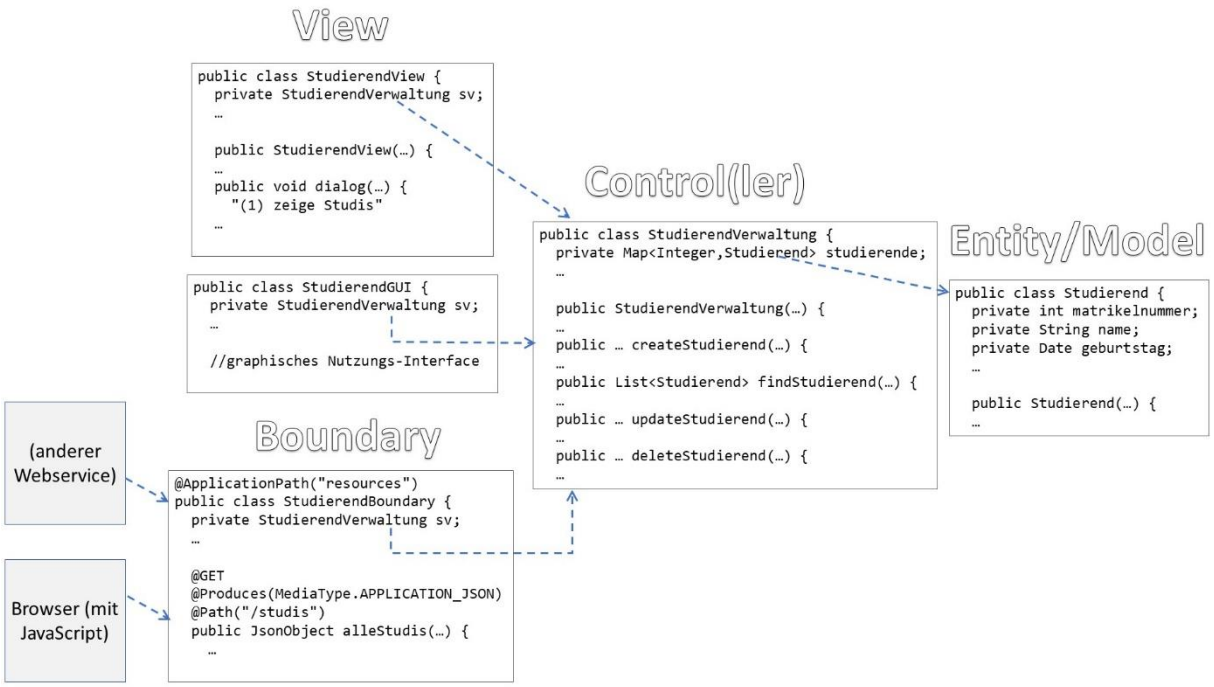
Nutzung:

```
    List<A> listeb;
    for(A el:listeb) {
      int wert = el.adapt().getWert();
      ...
    }
```

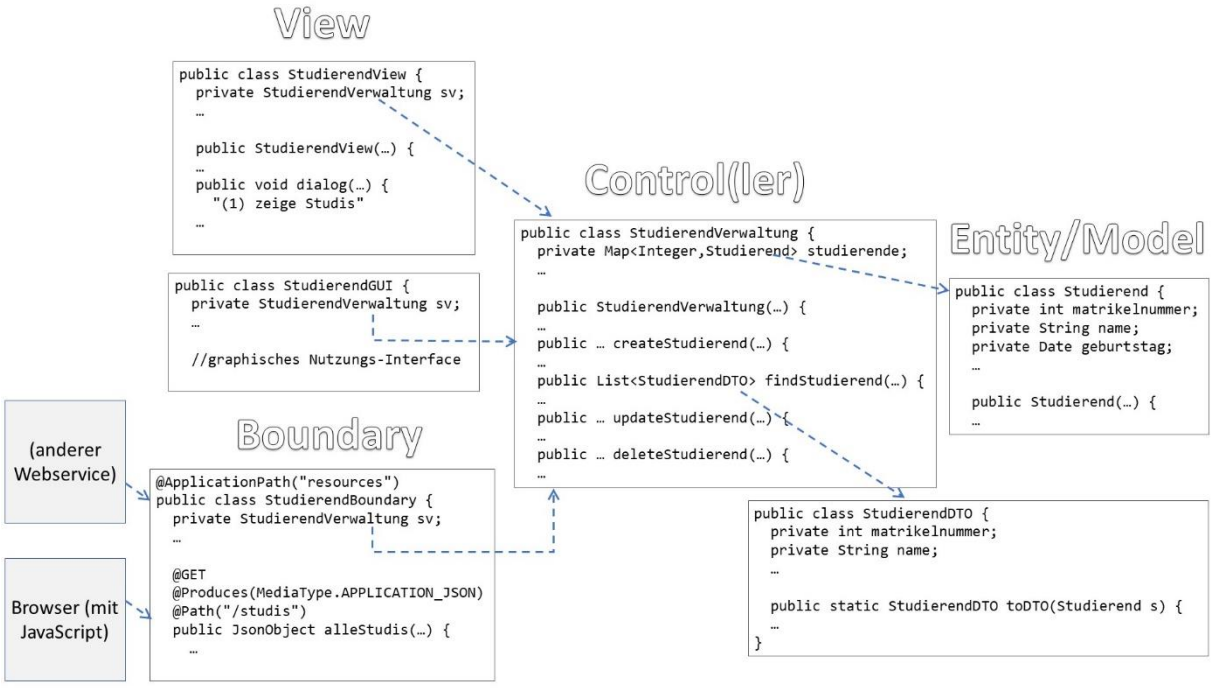
da zumindest der letzte Ansatz wenig intuitiv ist und eher die Frage nach dem mangelhaften ursprünglichen Klassendesign aufwirft, ist die Nutzung diskutabel.

Frage: Sollen wir Pakete nutzen, wenn ja wie.

Ich gehe davon aus, dass Sie in jeder Aufgabe Pakete nutzen. Die Aufteilung ist ein spannendes Thema mit vielen Antworten. Ein Weg ist hier, dass für MVC bzw. BCE es jeweils ein Paket pro „Buchstaben“ (z. B. entity, control, boundary) gibt. Darin darf es dann Unterpakete geben, die z. B. sich auf unterschiedliche Aufgaben, z. B. orientiert an den Use Cases geben. Die Begriffe habe ich auch in der folgenden Abbildung, ergänzend zu einem älteren FA-Dokument, visualisiert.



Natürlich kann es von jeder Klassenart mehrere Varianten geben. Auch ist es sinnvoll weitere Konzepte einzubeziehen. Im Beispiel sollen wahrscheinlich keine Referenzen auf Studierend-Objekte zurückgegeben werden, um zu vermeiden, dass andere Objekte als der StudierendController diese Objekte verändern kann. Hierzu sind z. B. Data-Transfer-Objekte geeignet, die nur Informationen enthalten, das eigentliche Objekt dadurch aber nicht verändert werden kann. Eine solche Klasse ist in der folgenden Abbildung ergänzt. Sie könnte zumindest zu zwei Paketen zugeordnet werden, ein eigenes DTO-Paket wäre aber auch denkbar. Meist wird der Controller nicht direkt die Entitäten verwalten, sondern einen Zugriff auf ein Persistence-Managementsystem haben, das die Entitäts-Objekte verwaltet.



Hier gibt es aber durchaus wesentlich andere Ansätze, die Sie im 5. Semester in der Software-Architektur andiskutieren. Als Randnotiz: Verantwortliche Personen für die Software-Architektur haben jahrelange Erfahrungen vorher in der Entwicklung gesammelt.

getrennte Info: In klassischen Vorlesungen verweise ich ab und zu auf die studentische Selbstverwaltung. Konkret sind Sie Mitglied der Fachschaft der Fakultät, die die Meinung aller Studierenden vertritt und dazu viele Studierende in unterschiedliche Entscheidungsgremien der Hochschule entsendet. Deren Meinung trägt wesentlich u. a. zur Weiterentwicklung von Studiengängen bei. Vereinfacht gilt, je mehr Informatik-Studierende dort aktiv sind, desto passender werden ihre Interessen vertreten. Nichts ist schlimmer als ein Maschinenbauer der annimmt wissen zu können, wie Informatiker denken. Gehen Sie zu treffen der Fachschaft, da auch der Austausch mit Studierenden anderer Studien- und Jahrgänge vieles Interessantes bietet. Die Teilnahme an der studentischen Selbstverwaltung ist ein wichtiger Bestandteil des studentischen Lebens an einer Fachhochschule.