Hausaufgabe 7

Tim Wende

November 28, 2021

Veranstaltungsverwaltungssystem

Gegeben sei folgende Beschreibung für ein Veranstaltungsverwaltungssystem:
Personen haben Zeichenketten als `Name`. Studierende sind Personen, erben also die Eigenschaften von Person, haben aber zusätzlich eine ganzzahlige `Matrikelnummer` und nehmen an beliebig vielen Veranstaltungen teil. Eine Veranstaltung hat potentiell beliebig viele Teilnehmende, wird aber von einem, zwei oder drei MitarbeiterInnen betreut. Eine Veranstaltung hat eine `Veranstaltungsnummer` und einen `Titel`. Seminare und Vorlesungen sind spezielle Veranstaltungen. Ein Seminar hat eine begrenzte `Anzahl an Plätzen`, für eine Vorlesung wird eine Klausur angeboten oder nicht. MitarbeiterInnen sind Personen und betreuen eine bis fünf Veranstaltungen und haben eine `Personalnummer`. ProfessorInnen und AssistentInnen sind Mitarbeitende. AssistentInnen sind bei genau einem/r ProfessorIn beschäftigt und haben eine bestimmt `Finanzierung` (Zeichenkette). Ein/e ProfessorIn hat ein `Lehrgebiet` (Zeichenkette), beschäftigt beliebig viele AssistentInnen und ist InhaberIn von genau einem Lehrstuhl. Ein Lehrstuhl hat eine `Bezeichnung` und genau einen/e ProfessorIn als InhaberIn.
Erstellen Sie anhand der obigen Beschreibung ein **Klassendiagramm**. Ihr Diagramm sollte folgende Punkte beinhalten:

- **Generalisierungsbeziehungen**,
- **Assoziationen** mit Assoziationsnamen und Leserichtung,
- **Multiplizitäten** sowie
- **Attributnamen** und (sinnvolle) –**typen**.

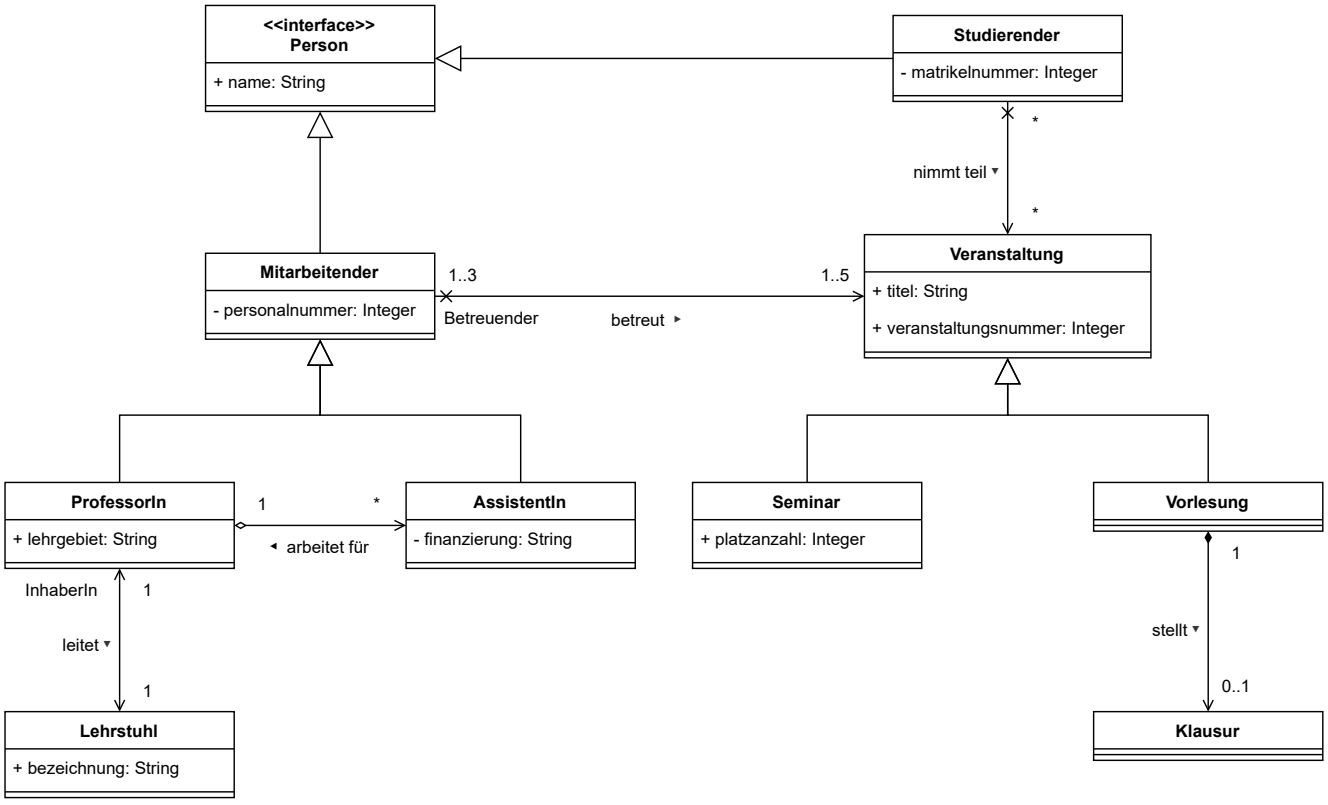

Figure 1: `class_diagram`

Finden Sie jeweils ein Beispiel, bei dem eine Aggregations- und eine Kompositionsbeziehung sinnvoll ist. Erläutern Sie kurz den Unterschied zwischen Aggretation und Komposition anhand des Bespiels.

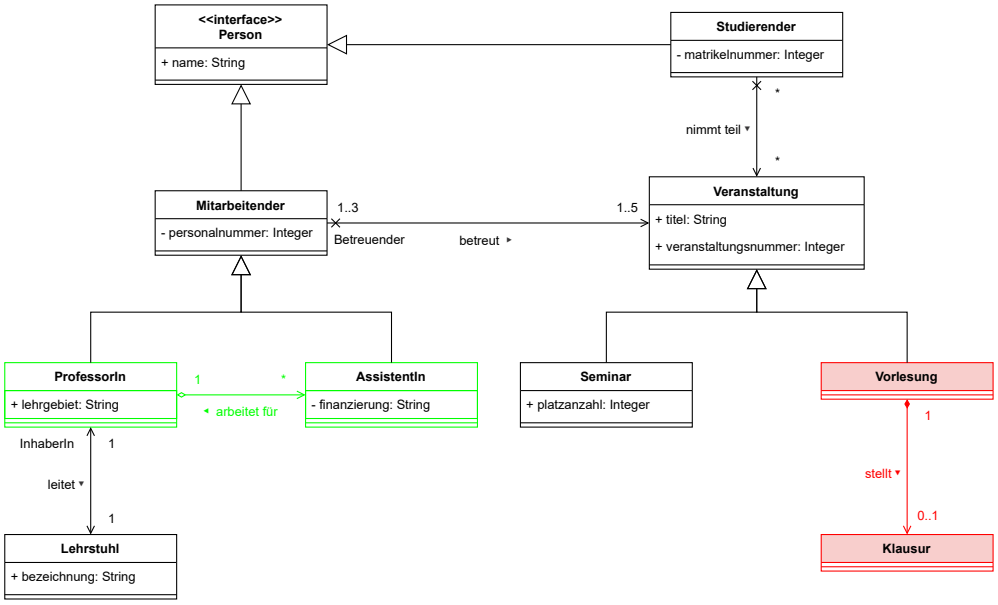

Figure 2: `class_diagram_diff`

Als Beispiel für die Aggregation habe ich die Beziehung zwischen ProfessorIn und AssistentIn gewählt. Im Gegensatz dazu steht die Komposition, welche durch die Beziehung zwischen Vorlesung und Klausur verkörpert wird.

Einleitend fass Wikipedia es sehr gut zusammen (stark gekürzt):
Aggregation:
„Eine exakte Definition wird in der UML2 nicht gegeben [...]. Ein konkreter Nutzen lässt sich z. B. ableiten, indem man einem Ende einer Assoziation eine besondere Betonung zukommen lässt“
Komposition:
Der Unterschied zur Aggregation ist, dass ein Objekt, das als Ganzes Teile enthält, für die Existenz der Teile verantwortlich ist.
Also anschaulich:

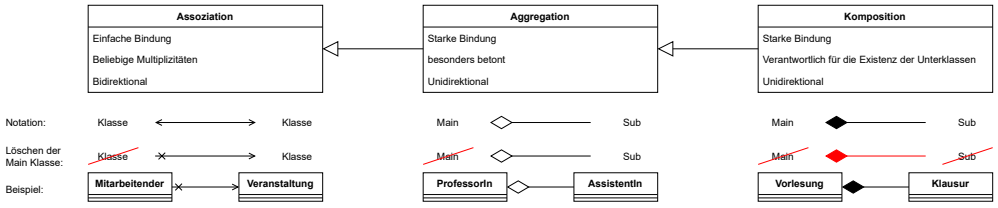

Figure 3: `class_diagram_chart`

Um diese Informationen auf unser Beispiel anzuwenden (bzg. der Klassen; nicht der Objekte):
Ein/e AssistentIn ist besonders *betont abhängig* von einem/einer ProfessorIn. Diese Bindung ist beispielsweise stärker, als die Bindung eines Studierenden zu einer Veranstaltung, jedoch ist der/die ProfessorIn nicht verantwortlich für die Existenz eines/einer AssistentIn.
Die Klausur existiert jedoch ausschließlich, wenn die Vorlesung existiert. Fällt diese Weg, wird das Objekt Klausur automatisch entfernt.

Anmerkungen zum Diagramm:

- Ich gehe davon aus, dass eine Klausur für nur eine Vorlesung genutzt werden kann. Da Dies nicht so genau aus der Aufgabenstellung erkennbar ist, sei dies hier erwähnt.
- Die Multiplizität am Pfeil `nimmt teil` bei Studierender könnte auch mit Hilfe des Attributs `platzanzahl` aus Seminar beschrieben werden. So könnte (um bei Java zu bleiben):
  `(Veranstaltung.isClass(Seminar)) ? ((Seminar) Veranstaltung).platzanzahl : *`
  die Genauigkeit der Obergrenze erhöhen.

Kohäsion und Kopplung

1. Beschreiben Sie in eigenen Worten das Prinzip der Kohäsion und Kopplung in der Implementierung von Software-Projekten.

   Die Bindung verschiedener Objekte (Klassen/ Methoden/ ganzen Modulen/ ...), entsteht durch gegenseitige Methodenaufrufe oder Referenzen. Die Dichte bzw. Anzahl dieser Bindungen wird durch das Maß der Kohäsion beschrieben. Dieses Maß beschreibt den logischen Zusammenhang der jeweiligen Objekte. Die Bindung verschiedener Pakete wird Kopplung genannt. Man möchte die paketintern Kohäsion der Bindung also so hoch wie möglich halten, und die Bindung der verschiedenen Pakete so gering wie möglich.

2. Warum sind hohe Kohäsion und lose Kopplung der niedrigen Kohäsion und enger Kopplung vorzuziehen?

   Wenn man seinen Code in Pakete aufteilt, möchte man paketintern einen großen logischen Zusammenhang. Zusätzlich möchte man so wenige Querverbindungen zwischen den Paketen wie möglich erstellen, da man sonst in Gefahr läuft, dass jede Klasse in irgend einer Art und Weise abhänhig von einer Zweiten ist. Wenn man zusätzlich diese bereits reduzierten Querverbindungen auf eine Schnittstelle begrenzt, kann man „hinter“ dieser „Proxy“-Klasse den Code beliebig ändern, ohne, dass man andere Klassen ebenfalls ändern muss. Die einzige Klasse, welche beeinflusst wird, ist die „Proxy“-Klasse. So hat man nur eine (ver-)Bindung und in den Paketen eine hohe Kohäsion.

3. Geben Sie ein **eigenes** Beispiel in Form eines Klassendiagrammes für hohe Kohäsion und lose Kopplung an.

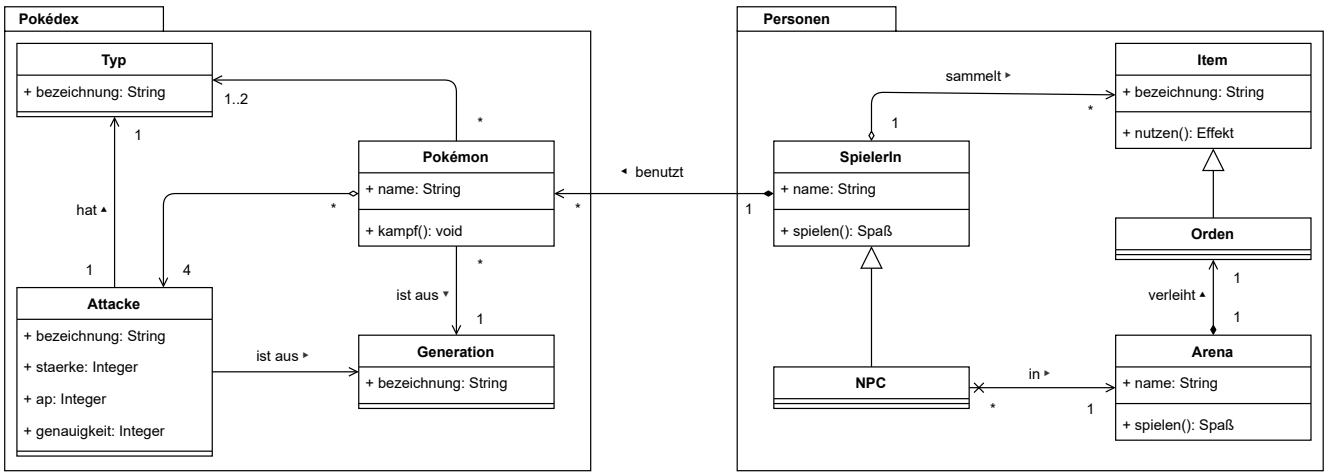

Figure 4: `class_diagram_example`

Klassendiagramm Implizierungen

Gegeben sei folgendes UML-Klassendiagramm mit ähnlichem Kontext wie aus der ersten Aufgabe.
Zusätzliche Annahmen:

- Jede Person hat nur einen Personalausweis. Wir vernachlässigen den Fakt, dass man diesen verlegen oder verlieren kann.
- Studierende können sich bei der Bibliothek auf Wartelisten für Bücher setzen lassen Die Bibliothek benachrichtigt die Studierenden nach dem FIFO-Prinzip, wenn das Buch verfügbar ist Das soll sich in der verwendeten Datenstruktur für die Assoziation zwischen Bibliothek und Studierender in der Bibliothek widerspiegeln.

Hinweis: Achten Sie darauf, dass die Multiplizitäten vom Code auch abgebildet werden! Zum Beispiel, dass eine Person keine Null-Referenz auf einen Personalausweis haben darf!

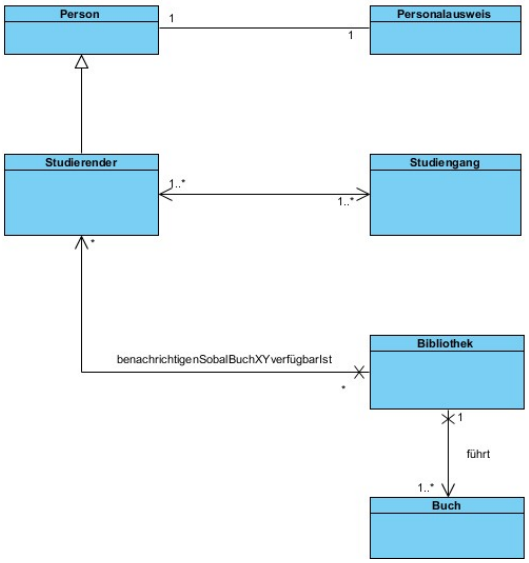

Figure 5: `class_diagram`

Tragen Sie in die vorgegebenen Java-Klassen, die Umsetzung der Assoziationen mit korrekter Multiplizität ein. An welchen Stellen ergibt es Sinn, die Assoziationen weiter über (unique, ordered) einzuschränken?
Hier mein Klassendiagramm, welches nach dem Javacode erstellt wurde:

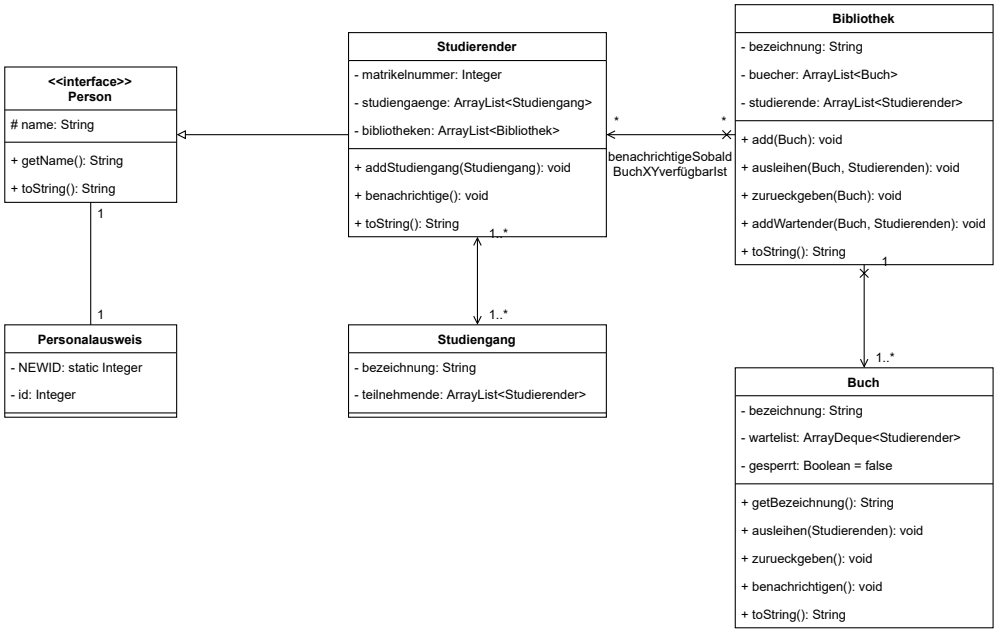

Figure 6: `class_diagram_after`

Zuerst gebe ich aus debugging Zwecken alle erstellten Studierenden aus. Dann folgt das Erstellen, sowie initiales nutzen der Bibliothek. Und zu guter letzt werden verschiedene Methoden ausprobiert.
Um den Code zu besser verstehen zu können, werde ich diesen nun Stück für Stück präsentieren. Hierzu verwende ich folgende Reihenfolge:

1. Output der Testklasse
2. Klasse Test inklusive `main Methode`
3. Klasse Studiengang
4. Klasse Person
5. Klasse Personalausweis
6. Klasse Studierender
7. Klasse Bibliothek
8. Klasse Buch

Der folgende Output wird von der `Test Klasse` erzeugt:

```
+--------------------------------------------------------------------------+
|                                Tim Wende                                 |
|                                                                          |
|  Name:              Tim Wende                                            |
|  Personalausweis:   1                                                    |
|  Matrikelnummer:    3281514                                              |
|                                                                          |
|  Studiengänge:                                                           |
|  +---------------------------------------------------------------------+ |
|  | Bezeichnung:         Angewandte Mathematik und Informatik          | |
|  | Teilnehmende:        Tim Wende, Paddel                             | |
|  | Anzahl Teilnehmende: 2                                             | |
|  |                                                                    | |
|  | Bezeichnung:         Softwaretechnik                               | |
|  | Teilnehmende:        Tim Wende                                     | |
|  | Anzahl Teilnehmende: 1                                             | |
|  +---------------------------------------------------------------------+ |
+--------------------------------------------------------------------------+

+--------------------------------------------------------------------------+
|                                 Paddel                                   |
|                                                                          |
|  Name:              Paddel                                               |
|  Personalausweis:   2                                                    |
|  Matrikelnummer:    1234567                                              |
|                                                                          |
|  Studiengänge:                                                           |
|  +---------------------------------------------------------------------+ |
|  | Bezeichnung:         Angewandte Mathematik und Informatik          | |
|  | Teilnehmende:        Tim Wende, Paddel                             | |
|  | Anzahl Teilnehmende: 2                                             | |
|  +---------------------------------------------------------------------+ |
+--------------------------------------------------------------------------+
```

```
Tim Wende hat sich swt_wende_tim_h05.pdf vorgemerkt
Paddel hat sich swt_wende_tim_h05.pdf vorgemerkt
Paddel hat sich swt_wende_tim_h06.pdf vorgemerkt

+-------------------------------------------------------------------------+
|                                RWTH Bib                                 |
|                                                                         |
|  Bezeichnung: RWTH Bib                                                  |
|                                                                         |
|  Bücher:                                                                |
|  +--------------------------------------------------------------------+ |
|  | Bezeichnung:     swt_wende_tim_h01.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h02.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h03.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h04.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h05.pdf                             | |
|  | Wartende:        Tim Wende, Paddel                                 | |
|  | Anzahl Wartende: 2                                                 | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h06.pdf                             | |
|  | Wartende:        Paddel                                            | |
|  | Anzahl Wartende: 1                                                 | |
|  +--------------------------------------------------------------------+ |
+-------------------------------------------------------------------------+

Tim Wende hat swt_wende_tim_h05.pdf ausgeliehen
swt_wende_tim_h05.pdf wurde zurückgegeben
@Paddel: swt_wende_tim_h05.pdf ist verfügbar
Paddel hat swt_wende_tim_h05.pdf ausgeliehen

+-------------------------------------------------------------------------+
|                                RWTH Bib                                 |
|                                                                         |
|  Bezeichnung: RWTH Bib                                                  |
|                                                                         |
|  Bücher:                                                                |
|  +--------------------------------------------------------------------+ |
|  | Bezeichnung:     swt_wende_tim_h01.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h02.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h03.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h04.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h05.pdf                             | |
|  |                                                                    | |
|  | Bezeichnung:     swt_wende_tim_h06.pdf                             | |
|  | Wartende:        Paddel                                            | |
|  | Anzahl Wartende: 1                                                 | |
|  +--------------------------------------------------------------------+ |
+-------------------------------------------------------------------------+
```

Schauen wir uns dazu erstmal die `Test Klasse` an:

```
/**
 * Test class
 *
 * @author Tim Wende
 *
 */
public class Test {
	public static int MAX_LENGTH = 75;

	public static void main(String[] args) {
		Studiengang matse = new Studiengang("Angewandte Mathematik und Informatik");
		Studiengang bwl = new Studiengang("Softwaretechnik");

		Studierender tim = new Studierender("Tim Wende", 3281514, new Personalausweis(), matse);
		tim.addStudiengang(bwl);

		Studierender paddel = new Studierender("Paddel", 1234567, new Personalausweis(), matse);

		System.out.println(tim);
		System.out.println();
		System.out.println(paddel);

		System.out.println();

		Bibliothek rwthbib = new Bibliothek("RWTH Bib");

		Buch h05 = new Buch("swt_wende_tim_h05.pdf");
		rwthbib.addWartender(h05, tim);
		rwthbib.addWartender(h05, paddel);

		Buch h06 = new Buch("swt_wende_tim_h06.pdf");
		rwthbib.addWartender(h06, paddel);

		rwthbib.add(new Buch("swt_wende_tim_h01.pdf"));
		rwthbib.add(new Buch("swt_wende_tim_h02.pdf"));
		rwthbib.add(new Buch("swt_wende_tim_h03.pdf")); // mein Lieblingsbuch
		rwthbib.add(new Buch("swt_wende_tim_h04.pdf"));
		rwthbib.add(h05);
		rwthbib.add(h06);

		System.out.println();
		System.out.println(rwthbib);

		// rwthbib.ausleihen(h05, paddel); // throws error

		System.out.println();
		rwthbib.ausleihen(h05, tim); // gesperrt
		rwthbib.zurueckgeben(h05); // entsperrt
		rwthbib.ausleihen(h05, paddel);

		System.out.println();
		System.out.println(rwthbib);
	}
}
```

In der ersten Zeile der `main Methode` finden wir:

```
Studiengang matse = new Studiengang("Angewandte Mathematik und Informatik");
```

Hier wird ein Objekt der Klasse Studiengang erstellt. Dieses bekommt eine Bezeichnung; hier beispielsweise „Angewandte Mathematik und Informatik“; übergeben. Die dazuehörige Klasse sieht wie folgt aus:

```
import java.util.ArrayList;

/**
 * Studiengang Klasse
 *
 * @author Tim Wende
 *
 */
public class Studiengang {
	// .. Hier die entsprechenden Attribute einpflegen
	private String bezeichnung; // unique
	private ArrayList<Studierender> teilnehmende = new ArrayList<>(); // ordered

	/**
	 * default ctor
	 *
	 * @param bezeichnung Bezeichnung des Studiengangs
	 */
	public Studiengang(String bezeichnung) {
		this.bezeichnung = bezeichnung;
	}

	/**
	 * Fügt einen Teilnehmenden zum Studiengang hinzu
	 *
	 * @param teilnehmender Teilnehmender
	 */
	public void addTeilnehmender(Studierender teilnehmender) {
		teilnehmende.add(teilnehmender);
	}

	/**
	 * default toString()
	 */
	public String toString() {
		String ret = "Bezeichnung: " + " ".repeat(8) + this.bezeichnung;
		ret += "\nTeilnehmende: " + " ".repeat(7);
		String div = "";
		for (int i = 0; i <= 3; i++) {
			if (i >= teilnehmende.size())
				break;
			ret += div + teilnehmende.get(i).getName();
			div = ", ";
		}
		if (teilnehmende.size() > 4)
			ret += ", ...";
		ret += "\nAnzahl Teilnehmende: " + teilnehmende.size();
		return ret;
	}
}
```

Schauen wir uns die `main Methode` oder `addTeilnehmender` aus `Studiengang.java` an, bemerken wir, dass wir einen Studierenden benötigen:

```
Studierender tim = new Studierender("Tim Wende", 3281514, new Personalausweis(), matse);
```

Der Studierende benötigt einen Namen; hier „Tim Wende“; eine Matrikelnummer, einen Personalausweis, sowie einen ersten Studiengang; hier „matse“; Zusätzlich benötigen wir erstmal die Abstrakte Klasse `Person`, von welcher `Studierender` erben wird.

```
/**
 * abstract Person Klasse
 *
 * @author Tim Wende
 *
 */
public abstract class Person {
        // .. Hier die entsprechenden Attribute einpflegen
        protected String name;
        protected Personalausweis personalausweis; // unique

        /**
         * default ctor
         *
         * @param name             Name der Person
         * @param personalausweis Personalausweis der Person
         */
        public Person(String name, Personalausweis personalausweis) {
                this.name = name;
                this.personalausweis = personalausweis;
                personalausweis.setPerson(this);
        }

        /**
         * Gibt den Namen der Person zurück
         *
         * @return Name
         */
        public String getName() {
                return name;
        }

        /**
         * default toString()
         */
        public String toString() {
                String ret = "Name: " + " ".repeat(11) + name;
                ret += "\nPersonalausweis: " + personalausweis.getID();
                return ret;
        }
}
```

Hier sehen wir eine kleine Hilfsklasse `Personalausweis`. Diese wird ausschließlich von der Klasse `Person` genutzt:

```
/**
 * Personalausweis Klasse
 *
 * @author Tim Wende
 *
 */
public class Personalausweis {
        // .. Hier die entsprechenden Attribute einpflegen
        private static int NEWID = 1;
        private int id; // unique
        private Person person;

        /**
         * default ctor
         */
        public Personalausweis() {
                // I solemnly swear that I'll add a Person
                this.id = getNewID();
        }

        /**
         * Erzeugt einzigartige ID
         *
         * @return Unique ID
         */
        private static int getNewID() {
                return NEWID++;
        }

        /**
         * Setzt die Person
         *
         * @param person Person
         */
        public void setPerson(Person person) {
                this.person = person;
        }

        /**
         * Gibt die ID des Personalausweises zurück
         *
         * @return ID
         */
        public int getID() {
                return id;
        }
}
```

Nun, wie versprochen `Studierender.java`:

```java
import java.util.ArrayList;

/**
 * Studierender Klasse
 *
 * @author Tim Wende
 *
 */
public class Studierender extends Person {
	// .. Hier die entsprechenden Attribute einpflegen
	private final int matrikelnummer; // unique
	private ArrayList<Studiengang> studiengaenge = new ArrayList<>(); // ordered
	private ArrayList<Bibliothek> bibliotheken = new ArrayList<>();

	/**
	 * default ctor
	 *
	 * @param name            Name des Studierenden
	 * @param matrikelnummer  Matrikelnummer des Studierenden
	 * @param personalausweis Personalausweis des Studierenden
	 * @param studiengang     Erster Studiengang des Studierenden
	 */
	public Studierender(String name, int matrikelnummer, Personalausweis personalausweis, Studiengang
		super(name, personalausweis);
		this.matrikelnummer = matrikelnummer;
		addStudiengang(studiengang);
	}

	/**
	 * Fügt einene Studiengang zum Studierenden hinzu
	 *
	 * @param studiengang Studiengang
	 */
	public void addStudiengang(Studiengang studiengang) {
		studiengaenge.add(studiengang);
		studiengang.addTeilnehmender(this);
	}

	/**
	 * Werde benachrichtigt
	 */
	public void benachrichtige() {
	}

	/**
	 * default toString() in schön
	 */
	@Override
	public String toString() {
		String ret = "|" + " ".repeat((Test.MAX_LENGTH - 2 - name.length()) / 2) + name;
		ret += "\n|\n|  " + super.toString().replace("\n", "\n|  ").replace(":", ":" + " ".repeat
		ret += "\n|  Matrikelnummer: " + " ".repeat(3) + matrikelnummer;
		ret += "\n|\n|  Studiengänge:";
		ret += "\n|  +" + "-".repeat(Test.MAX_LENGTH - 8) + "+";
		String div = "";
		for (Studiengang studiengang : studiengaenge) {
			ret += div + "\n|  | " + studiengang.toString().replace("\n", "\n|  | ");
			div = "\n|  |";
		}
		ret += "\n|  +" + "-".repeat(Test.MAX_LENGTH - 8) + "+";
```

```
                String out = "+" + "-".repeat(Test.MAX_LENGTH - 2) + "+";
                for (String line : ret.split("\n")) {
                        if (line.startsWith("|  |"))
                                out += "\n" + line + " ".repeat(Test.MAX_LENGTH - 4 - line.length()) + "
                        else
                                out += "\n" + line + " ".repeat(Test.MAX_LENGTH - 1 - line.length()) + "
                }

                return out + "\n+" + "-".repeat(Test.MAX_LENGTH - 2) + "+";
        }
}
```

Als nächstes wird in der `main Methode` eine Bibliothek erstellt. Diese trägt den kreativen Namen „RWTH Bib“:

```
Bibliothek rwthbib = new Bibliothek("RWTH Bib");
```

Da die Klasse `Bibliothek` als „Proxy“-Klasse gebraucht wird (siehe 2.2), leitet sie mehrere Funktionen „dumm“ weiter. Diese werden also meistens auf dem Objekt selber ausgeführt:

```
import java.util.ArrayList;

/**
 * Bibliothek Klasse
 *
 * @author Tim Wende
 *
 */
public class Bibliothek {
        // .. Hier die entsprechenden Attribute einpflegen
        private String bezeichnung; // unique
        private ArrayList<Buch> buecher = new ArrayList<>(); // ordered
        private ArrayList<Studierender> studierende = new ArrayList<>();

        /**
         * default ctor
         *
         * @param bezeichnung Bezeichnung der Bibliothek
         */
        public Bibliothek(String bezeichnung) {
                this.bezeichnung = bezeichnung;
        }

        /**
         * Fügt ein Buch in die Bibliothek hinzu
         *
         * @param buch Buch
         */
        public void add(Buch buch) {
                buecher.add(buch);
        }

        /**
         * Leiht ein Buch aus
         *
         * @param buch          Buch
         * @param ausleihender Ausleihender
         */
        public void ausleihen(Buch buch, Studierender ausleihender) {
                System.out.println(ausleihender.getName() + " hat " + buch.getBezeichnung() + " ausgeliel
                buch.ausleihen(ausleihender);
        }

        /**
         * Gibt ein Buch zurück
         *
         * @param buch Buch
         */
        public void zurueckgeben(Buch buch) {
                System.out.println(buch.getBezeichnung() + " wurde zurückgegeben");
                buch.zurueckgeben();
                benachrichtigeSobaldBuchXYverfuegbarIst(buch);
        }

        /**
         * Fügt einen Wartenden zur Warteliste des Buches hinzu
```

```
         *
         * @param buch          Buch
         * @param vormerkender Vormerkender
         */
        public void addWartender(Buch buch, Studierender vormerkender) {
                System.out.println(vormerkender.getName() + " hat sich " + buch.getBezeichnung() + " vorg
                buch.addWartender(vormerkender);
        }

        /**
         * Benachrichtige sobald BuchXY verfuegbar ist
         *
         * @param buchXY Buch
         */
        public void benachrichtigeSobaldBuchXYverfuegbarIst(Buch buchXY) {
                buchXY.benachrichtige();
        }

        /**
         * default toString() in schön
         */
        public String toString() {
                String ret = "|" + " ".repeat((Test.MAX_LENGTH - 2 - bezeichnung.length()) / 2) + bezeicl
                ret += "\n|\n|  Bezeichnung: " + bezeichnung;
                ret += "\n|\n|  Bücher:";
                ret += "\n|  +" + "-".repeat(Test.MAX_LENGTH - 8) + "+";
                String div = "";
                for (Buch buch : buecher) {
                        ret += div + "\n|  | " + buch.toString().replace("\n", "\n|  | ");
                        div = "\n|  |";
                }
                ret += "\n|  +" + "-".repeat(Test.MAX_LENGTH - 8) + "+";

                String out = "+" + "-".repeat(Test.MAX_LENGTH - 2) + "+";
                for (String line : ret.split("\n")) {
                        if (line.startsWith("|  |"))
                                out += "\n" + line + " ".repeat(Test.MAX_LENGTH - 4 - line.length()) + "
                        else
                                out += "\n" + line + " ".repeat(Test.MAX_LENGTH - 1 - line.length()) + "
                }

                return out + "\n+" + "-".repeat(Test.MAX_LENGTH - 2) + "+";
        }
}
```

Nun erstellen wir ein paar Bücher und fügen sie unserer Bibliothek hinzu. Als Buchnamen mussten alte Hausaufgaben von mir herhalten. Teilweise werden die Bücher direkt vorgemerkt.

```
        rwthbib.addBuch(new Buch("swt_wende_tim_h01.pdf"));
```

```
import java.util.ArrayDeque;

/**
 * Buch Klasse
 *
 * @author Tim Wende
 *
 */
public class Buch {
        // .. Hier die entsprechenden Attribute einpflegen
        private String bezeichnung;
        private ArrayDeque<Studierender> warteliste = new ArrayDeque<>(); // ordered
        private boolean gesperrt = false;

        /**
         * default ctor
         *
         * @param bezeichnung Bezeichnung des Buches
         */
        public Buch(String bezeichnung) {
                this.bezeichnung = bezeichnung;
        }

        /**
         * Gibt die Bezeichnung des Buches zurück
         *
         * @return Bezeichnung
         */
        public String getBezeichnung() {
                return bezeichnung;
        }

        /**
         * Leiht das Buch aus
         *
         * @param ausleihender Ausleihender
         */
        public void ausleihen(Studierender ausleihender) {
                if (gesperrt)
                        throw new ArithmeticException("Buch " + bezeichnung + " gesperrt");
                if (warteliste.size() == 0) {
                        gesperrt = true;
                        return;
                }
                if (warteliste.getFirst() == ausleihender)
                        warteliste.pop();
                else
                        throw new ArithmeticException(
                                        "Buch " + bezeichnung + " von " + warteliste.getFirst().getName()
                gesperrt = true;
        }

        /**
         * Gibt das Buch zurück
         */
        public void zurueckgeben() {
                gesperrt = false;
        }
```

```
        /**
         * Fügt einen Wartenden zur Warteliste hinzu
         *
         * @param wartender Wartender
         */
        public void addWartender(Studierender wartender) {
                warteliste.offer(wartender);
        }

        /**
         * Benachrichtigt ersten Wartenden
         */
        public void benachrichtige() {
                if (warteliste.size() == 0)
                        return;
                Studierender vormerkender = warteliste.getFirst();
                vormerkender.benachrichtige();
                System.out.println("@" + vormerkender.getName() + ": " + bezeichnung + " ist verfügbar")
        }

        /**
         * default toString()
         */
        public String toString() {
                String ret = "Bezeichnung: " + " ".repeat(4) + bezeichnung;
                if (warteliste.size() == 0)
                        return ret;
                ret += "\nWartende: " + " ".repeat(7);
                String div = "";
                for (int i = 0; i <= 3; i++) {
                        if (i >= warteliste.size())
                                break;
                        ret += div + ((Studierender) (warteliste.toArray()[i])).getName();
                        div = ", ";
                }
                if (warteliste.size() > 4)
                        ret += ", ...";
                ret += "\nAnzahl Wartende: " + warteliste.size();
                return ret;
        }
}
```

Diese werden nun teilweise vorgemerkt:

```
                rwthbib.addWartender(h05, tim);
```

ausgeliehen:

```
                rwthbib.ausleihen(h05, tim); // gesperrt
```

oder zurückgegeben:

```
                rwthbib.zurueckgeben(h05); // entsperrt
```

Die zugehörigen Ausgaben sind in `out.txt` zu finden.