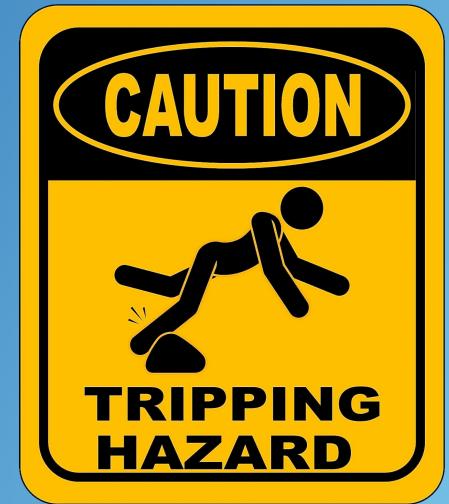




hochschule mannheim

Fakultät für Informatik

Die vierundvierzig fiesesten Fallen für Java-Anfänger



Teil 7: Die Sache mit den Objekten

Prof. Dr. Oliver Hummel



Voraussetzungen: Klassen als Datentypen

```
public class Student {  
    private String vorname, name;  
    private int matrikelnummer;  
  
    public Student(String vorname, String name, int matrikelnummer) {  
        this.vorname = vorname;  
        this.name = name;  
        this.matrikelnummer = matrikelnummer;  
    }  
  
    public String getVorname() {  
        return vorname;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public int getMatrikelnummer() {  
        return matrikelnummer;  
    }  
}
```

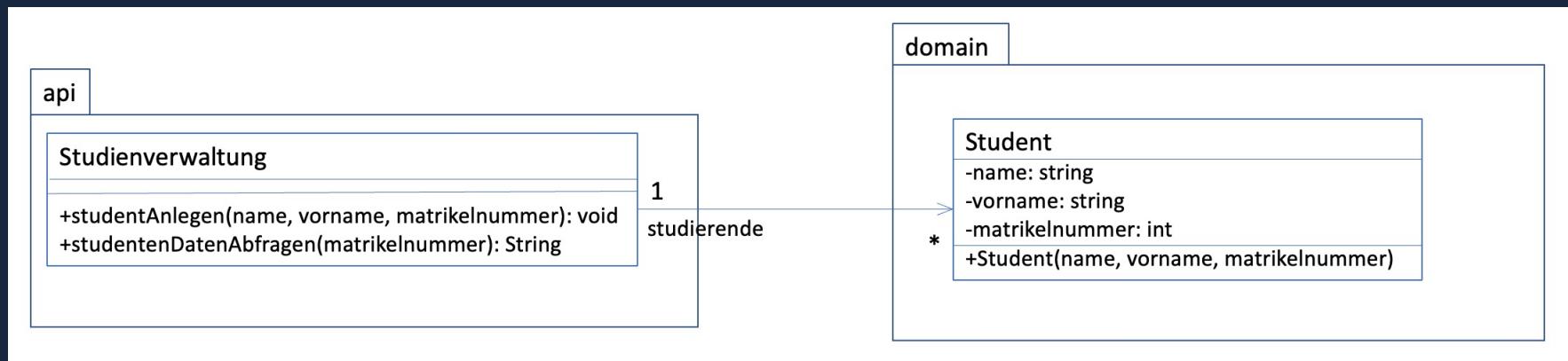
Vorname String	Name String	Matrikelnummer int
Klaus	Mayer	123456
Hans	Müller	234567
Katja	Fischer	224561
Christina	Vogel	324561
Claudia	Schneider	134569

```
Student s = new Student("Klaus", "Mayer", 123456);
```

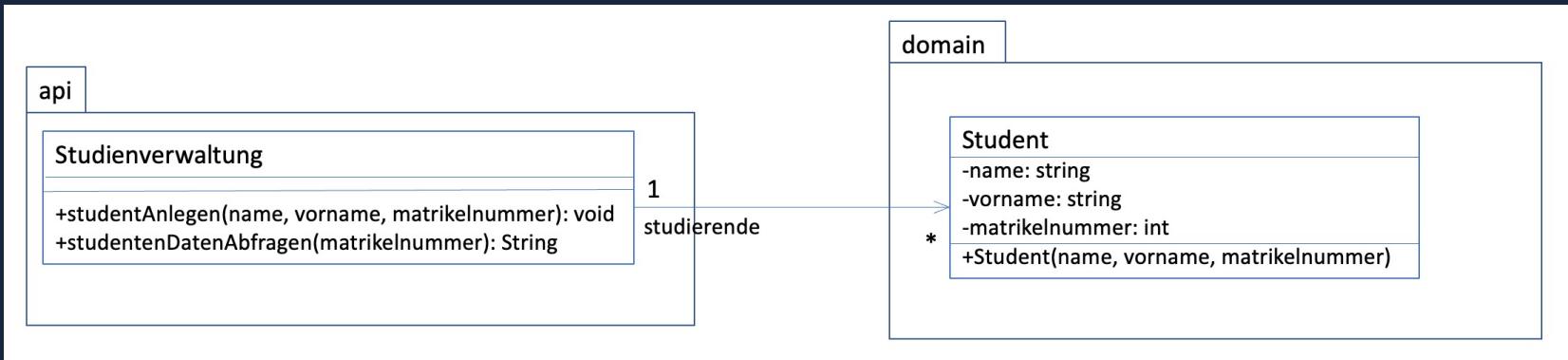


Eine Fassadenklasse Studienverwaltung verwaltet Studenten-Instanzen

- *vorbereitet für Model-View-Controller (MVC)*
- für die Einfachheit des Beispiels erfolgen Aufrufe über JUnit-Tests
 - und nicht über eine UI



- zur Wahrung des Geheimnisprinzips gibt die API keine Studenten-Instanzen nach außen, sondern nur Strings
 - Alternative wären sog. Data Transfer Objects (DTOs)



```

public class Studienverwaltung {
    private List<Student> studierende = new ArrayList<>();

    public void studentAnlegen(String vorname,
                               String name, int matrikelnummer) {
        studierende.add(new Student(vorname, name, matrikelnummer));
    }

    public String studentenDatenAbfragen(int matrikelnummer) {
        for (Student s : studierende)
            if (s.getMatrikelnummer() == matrikelnummer)
                return s.toString();

        return null;
    }
}

```



```
public class StudienverwaltungTest {  
  
    @Test  
    void testStudentAnlegen() {  
        Studienverwaltung stvw = new Studienverwaltung();  
        stvw.studentAnlegen("Klaus", "Mayer", 123456);  
  
        String daten = stvw.studentenDatenAbfragen(123456);  
        assertTrue(daten.contains("Klaus"));  
        assertTrue(daten.contains("Mayer"));  
        assertTrue(daten.contains("123456"));  
  
        daten = stvw.studentenDatenAbfragen(111111);  
        assertNull(daten);  
    }  
}
```

Objekt-Verlinkungen auf dem Heap

```
public class StudienverwaltungTest {
    @Test
    void testStudentAnlegen() {
        Studienverwaltung stvw = new Studienverwaltung();
        stvw.studentAnlegen("Klaus", "Mayer", 123456);

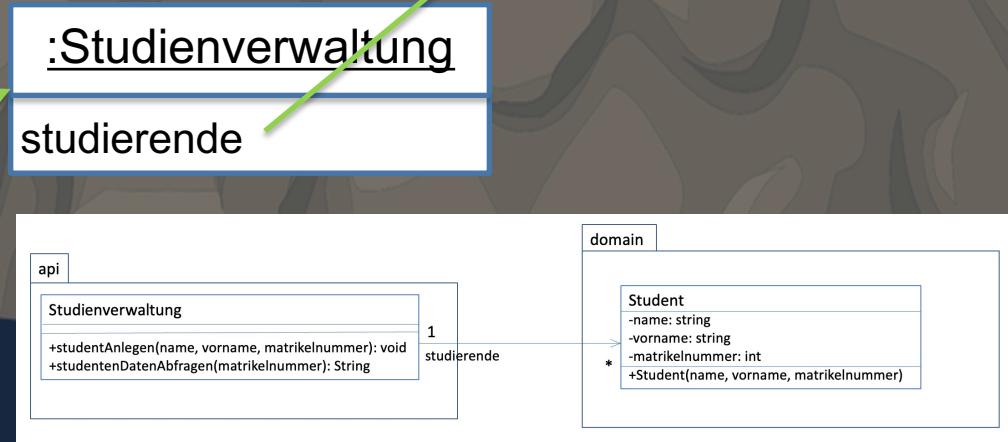
        String daten = stvw.studentenDatenAbfragen(123456);
        assertTrue(daten.contains("Klaus"));
        assertTrue(daten.contains("Mayer"));
        assertTrue(daten.contains("123456"));

        daten = stvw.studentenDatenAbfragen(111111);
        assertNull(daten);
    }
}
```

```
public class Studienverwaltung {
    private List<Student> studierende = new ArrayList<>();

    public void studentAnlegen(String vorname,
                               String name, int matrikelnummer) {
        studierende.add(new Student(vorname, name, matrikelnummer));
    }

    public String studentenDatenAbfragen(int matrikelnummer) {
    }
}
```





Analog zum Studenten können wir eine simple Klasse zum **Speichern von Prüfungen**, die im Semester angeboten werden, definieren

- wäre in der Realität sicher deutlich aufwändiger, um bspw. Prüfungen aus fortlaufenden Semestern oder Studiengängen speichern zu können

Prüfung
-name: string
-semester: int
-ects: int
+Prüfung(name, semester, ects)

Name String	Semester int	ECTS int
Programmierung 1	1	10
Programmierung 2	2	10
Datenmanagement	3	5

→ auch die Fassade muss um entsprechende Methoden zum Anlegen und Abfragen von Prüfungen erweitert werden



Da wir gleich eine (interne) Methode zum Finden von Studenten- und Prüfungsobjekten benötigen, bietet sich ein entsprechendes Refactoring an

- ... so dass studentenDatenAbfragen diese neue Methode benutzt

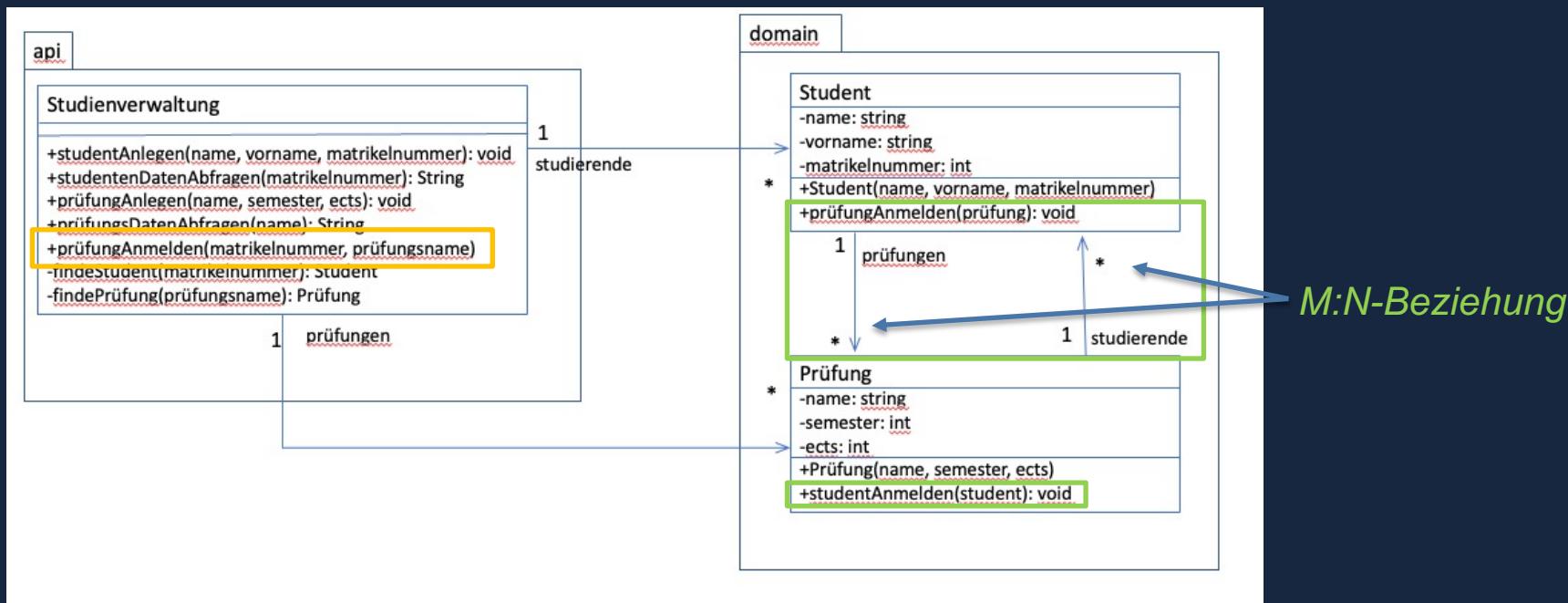
```
private Student findeStudent(int matrikelnummer) {  
    for (Student s : studierende)  
        if (s.getMatrikelnummer() == matrikelnummer)  
            return s;  
  
    return null;  
}
```

```
public String studentenDatenAbfragen(int matrikelnummer) {  
    Student s = findeStudent(matrikelnummer);  
  
    if (s == null)  
        return null;  
  
    return s.toString();  
}
```



Was benötigen wir, um einen **Studenten** zu einer Prüfung anzumelden?

- natürlich eine **Methode in der Fassade**
- eine **Möglichkeit („Link“)**, um **Studenten mit Prüfungen zu verknüpfen**
 - wir wollen herausfinden können, an welchen Prüfungen ein Student teilgenommen hat
 - und **umgekehrt** wollen wir wissen, welche Studenten an einer Prüfung teilgenommen haben





```
@Test  
void testPrüfungsAnmeldung() {  
    Studienverwaltung stvw = new Studienverwaltung();  
  
    stvw.studentAnlegen("Klaus", "Mayer", 123456);  
    stvw.studentAnlegen("Katja", "Fischer", 224561);  
  
    stvw.prüfungAnlegen("Programmierung 1", 1, 10);  
    stvw.prüfungAnlegen("Programmierung 2", 2, 10);  
  
    stvw.prüfungAnmelden(123456, "Programmierung 1");  
    stvw.prüfungAnmelden(224561, "Programmierung 1");  
  
    assertEquals(2, stvw.getAnzahlAnmeldungen("Programmierung 1"));  
    assertEquals(0, stvw.getAnzahlAnmeldungen("Programmierung 2"));  
}
```

Nochmal anschaulich auf dem Heap



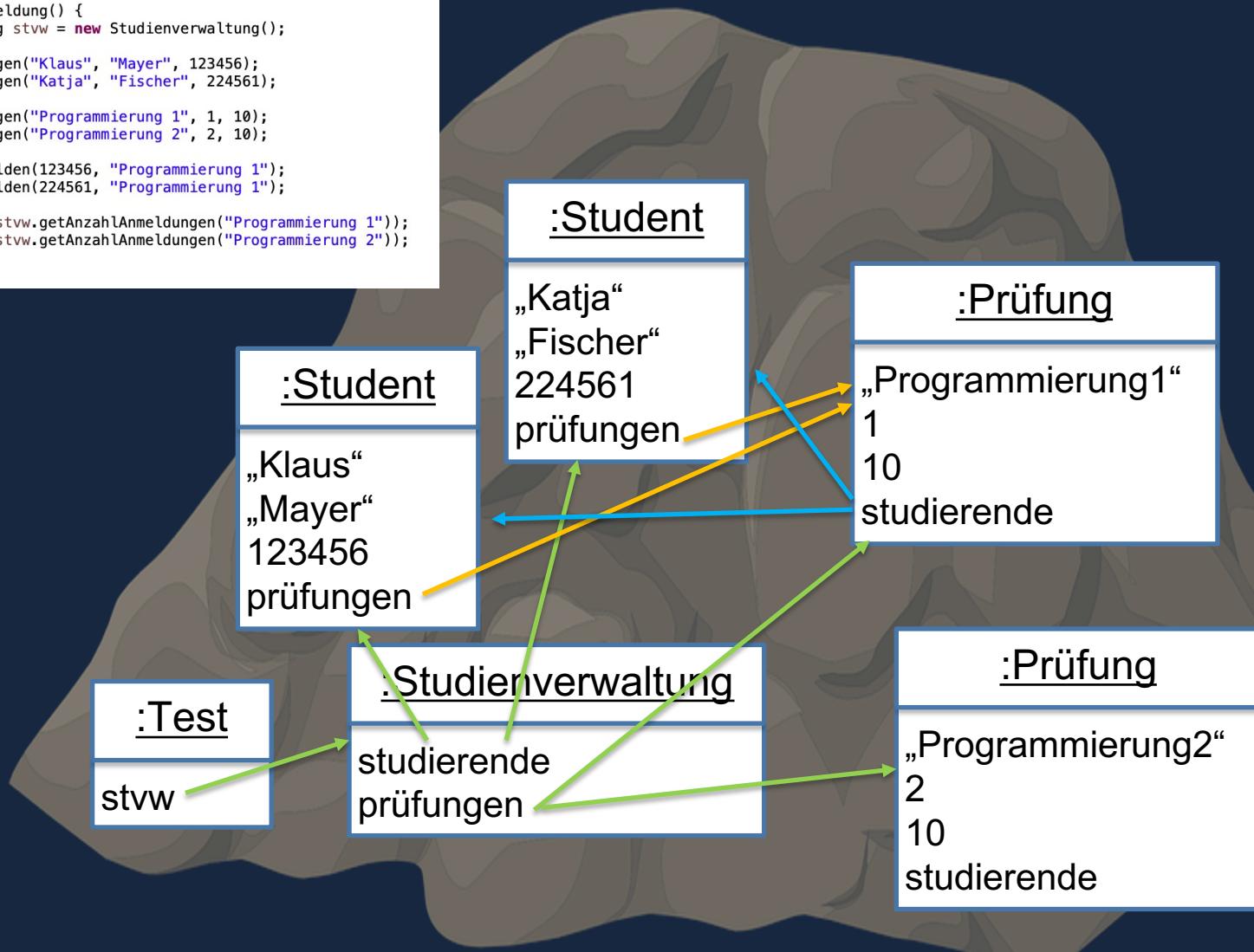
```
@Test
void testPruefungsAnmeldung() {
    Studienverwaltung stvw = new Studienverwaltung();

    stvw.studentAnlegen("Klaus", "Mayer", 123456);
    stvw.studentAnlegen("Katja", "Fischer", 224561);

    stvw.pruefungAnlegen("Programmierung 1", 1, 10);
    stvw.pruefungAnlegen("Programmierung 2", 2, 10);

    stvw.pruefungAnmelden(123456, "Programmierung 1");
    stvw.pruefungAnmelden(224561, "Programmierung 1");

    assertEquals(2, stvw.getAnzahlAnmeldungen("Programmierung 1"));
    assertEquals(0, stvw.getAnzahlAnmeldungen("Programmierung 2"));
}
```



Anm.: die verwendeten Listen sind eigentlich auch noch einmal eigene Objekte auf dem Heap

Eine Prüfung verweist also auf eine Liste von vielen Studierenden

```
public class Prüfung {  
  
    private ArrayList<Student> studierende = new ArrayList<Student>();
```

- und umgekehrt kann ein Studenten-Objekt viele Prüfungen verwalten

```
public class Student {  
  
    private ArrayList<Prüfung> prüfungen = new ArrayList<>();
```

→ *Es gibt nur leider noch keine Möglichkeit, eine Note zu speichern*

- da wir nicht zwei Mal eine Liste mit Noten synchronisiert zu den Anmeldungen halten möchten



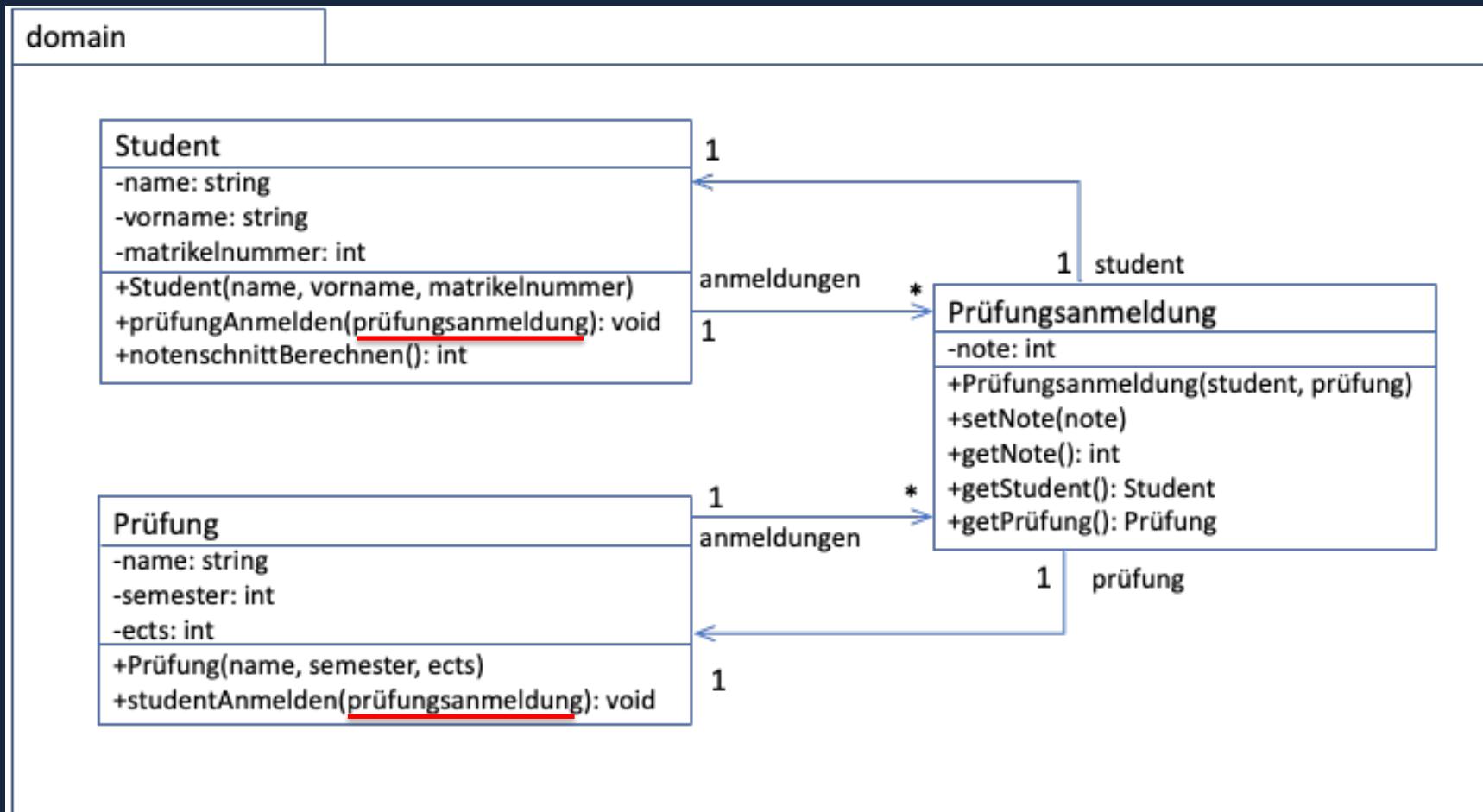
Um eigene Daten für M:N-Beziehungen speichern zu können, ist eine eigene Klasse notwendig

- es muss also für jede Prüfungsanmeldung eine Instanz erzeugt werden können, die die erhaltene Note speichern kann, bspw.:
 - Klaus Mayer erhält eine 2,0 in Programmierung 1
 - Katja Fischer erhält eine 1,7 in Programmierung 1

Die Klasse könnte folgendermaßen aussehen, sie speichert

- die Note
 - als int mit 100 multipliziert um ganzzahlig zu arbeiten
- eine Referenz zum Studenten-Objekt
- und eine Referenz zum Prüfungs-Objekt

Prüfungsanmeldung
-note: int
+Prüfungsanmeldung(student, prüfung)
+setNote(note)
+getNote(): int
+getStudent(): Student
+getPrüfung(): Prüfung



Schematische Darstellung auf dem Heap



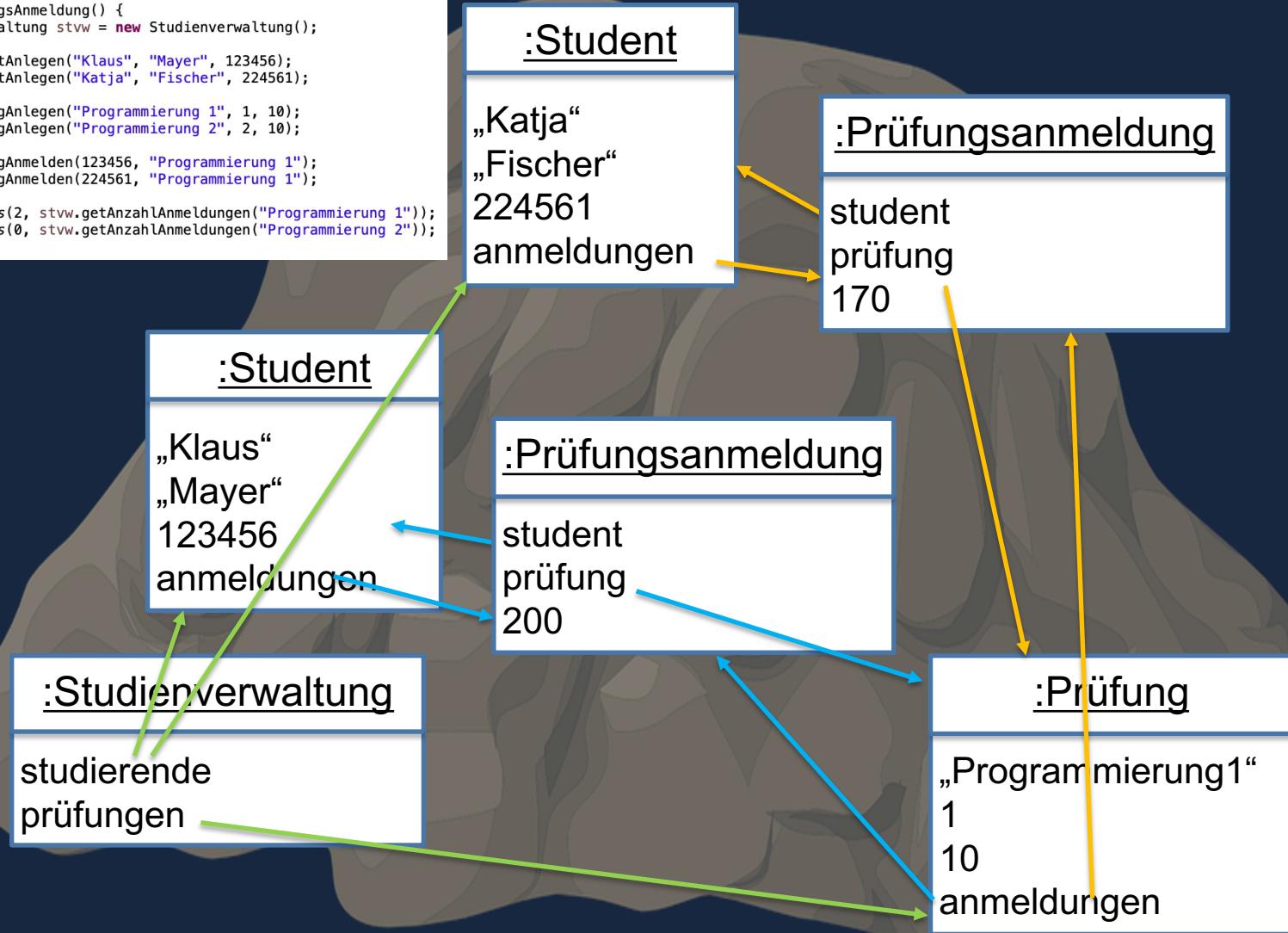
```
@Test
void testPrüfungsAnmeldung() {
    Studienverwaltung stvw = new Studienverwaltung();

    stvw.studentAnlegen("Klaus", "Mayer", 123456);
    stvw.studentAnlegen("Katja", "Fischer", 224561);

    stvw.prüfungAnlegen("Programmierung 1", 1, 10);
    stvw.prüfungAnlegen("Programmierung 2", 2, 10);

    stvw.prüfungAnmelden(123456, "Programmierung 1");
    stvw.prüfungAnmelden(224561, "Programmierung 1");

    assertEquals(2, stvw.getAnzahlAnmeldungen("Programmierung 1"));
    assertEquals(0, stvw.getAnzahlAnmeldungen("Programmierung 2"));
}
```



- 1 Studentin
- 2 Prüfungen
- und entsprechend
2 Prüfungsanmeldungen

→ Let's code it!

