**Objektorientiertes Konzept in Python:**

Zur Verbesserung und Einarbeitung des Feedbacks wurden ein paar Änderungen im Konzept und im UML-Diagramm vorgenommen. User können Daten von Prüfungen vorerst nur über die Kommandozeile hinzufügen da ein solches Dashboard im fertig Entwickelten Zustand über eine API-Schnittstelle die Daten direkt importieren sollte. Jedoch wurde für das hinzufügen von Lernzeiten die UI dementsprechend erweitert, um dies den User zu erleichtern. Ebenfalls wurde für die zusätzliche Übersicht der Lernzeiten ein Line-Chart eingefügt.

**Prototyp:**

Konstruktor:

__init__ ist in Python der Konstruktor, welche automatisch bei Aufruf der Klasse ausgeführt wird. In diesem Fall rufe ich als erstes den Konstruktor der Elternklasse (tkinter) da sonst das Fenster der GUI nicht initialisiert werden würde. Anschließend setze ich noch ein paar Standardwerte um danach die Funktion „_build“ die den Hauptaufbau der Benutzeroberfläche übernimmt auszuführen.

```
def __init__(self):  Fabian Linder
    super().__init__()
    self.title("Studien-Dashboard")
    self.geometry("1350x850")
    self.configure(bg=COLOR_BG)
    self._build()
```

JSON:
Das Speichern von Prüfungen sowohl auch Lernzeiten wird mit einem JSON-File umgesetzt. Es gibt dementsprechend 2 erweiterbare Teile im JSON „exams“ welche über die Kommando Zeile hinzugefügt werden können und „study_time“ welches über die UI erweitert werden kann.

```
{
  "semester": 2,
  "prüfungsname": "Interkulturelle und ethische Handlungskompetenzen",
  "note": 1.7,
  "ects": 5,
  "versuch": 1,
  "datum": "2025-10-01"
},
```

```
{
  "week_start": "2025-02-24",
  "hours": 32.5
},
```

Grundsätzlich ist das hinzufügen in das JSON-File bei beiden wegen code technisch sehr ähnlich. Nach Eingabe der Werte müssen diese Validiert werden (ob es der richtige Typ ist, ob werte im passenden Rahmen sind, ob es ein Update oder ein neuer Wert ist, etc.) wonach diese in das JSON gespeichert werden.

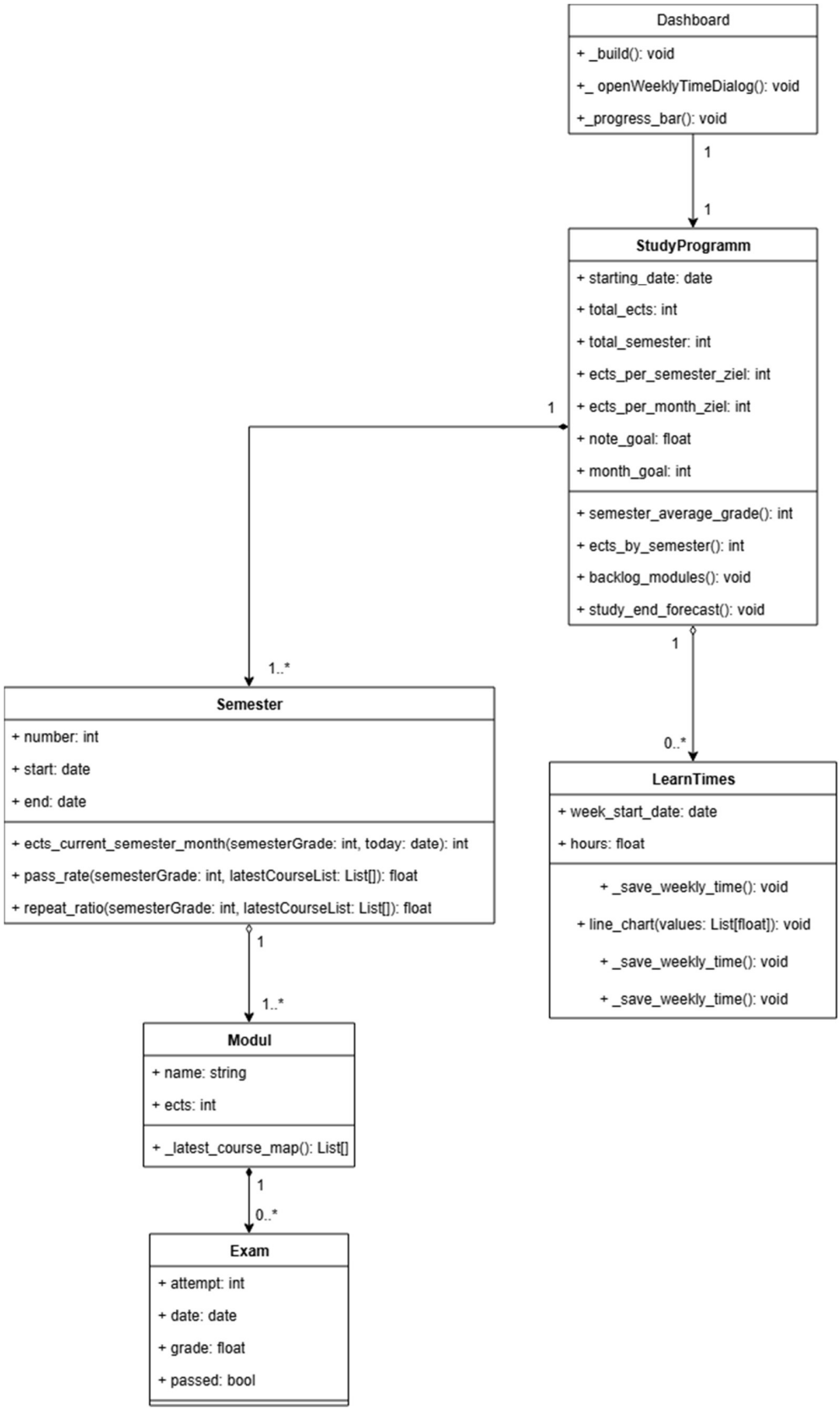
```
def save_json(data: Dict[str, Any]):  3 usages  Fabian Linder
    with open(DATA_FILE, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=2)
```

Private and Public:

Da es in Python keine Zugriffsmodifizierer gibt sind alle Funktionen grundsätzlich „public“. Jedoch werden Funktionen welche als „private“ angedacht sind mit einem Unterstrich markiert, um es für alle Developer verständlich zu machen das diese Funktion nur in der Ersteller Klasse aufgerufen werden soll („private“).

```
def _prev_month(self):  1 usage  Fabian Linder
    if self.month_var.get() > 1:
        self.month_var.set(self.month_var.get() - 1)
    else:
        self.month_var.set(12)
        self.year_var.set(self.year_var.get() - 1)
    self._update_week_selection()
```

**UML:**


Dashboard
+ _build(): void
+_ openWeeklyTimeDialog(): void
+_progress_bar(): void
1
1
StudyProgramm
+ starting_date: date
+ total_ects: int
+ total_semester: int
+ ects_per_semester_ziel: int
+ ects_per_month_ziel: int
+ note_goal: float
+ month_goal: int
+ semester_average_grade(): int
+ ects_by_semester(): int
+ backlog_modules(): void
+ study_end_forecast(): void
1
1..*
1
0..*
Semester
+ number: int
+ start: date
+ end: date
+ ects_current_semester_month(semesterGrade: int, today: date): int
+ pass_rate(semesterGrade: int, latestCourseList: List[]): float
+ repeat_ratio(semesterGrade: int, latestCourseList: List[]): float
LearnTimes
+ week_start_date: date
+ hours: float
+ _save_weekly_time(): void
+ line_chart(values: List[float]): void
+ _save_weekly_time(): void
+ _save_weekly_time(): void
1
1..*
Modul
+ name: string
+ ects: int
+ _latest_course_map(): List[]
1
0..*
Exam
+ attempt: int
+ date: date
+ grade: float
+ passed: bool

**Beschreibung:**

Dashboard

- Übernimmt die grafische Darstellung der Werte und Informationen

StudyProgramm:

- Ist zuständig für das gesamte Studium
- Hauptzielwerte welche am Anfang des Studiums festgelegt werden

Semester:

- Das Semester ist eine Komposition vom StudyProgramm
- Erstellt viele Berechnungen mit den Daten von „Modul“ und „Exam“.
- Start- Enddatum des Semesters für Berechnungen sowie die Nummer zur Bestimmung des aktuellen Semesters

Modul:

- Module sind im Aggregation zustand mit Semester da nicht festgelegt ist in welchem Semester diese absolviert werden.
- Hier sind die wichtigen Werte der Name und die ECTS des Moduls

Exam:

- Exams sind eine Komposition von Modul
- Da ein Modul auch ohne Prüfungsleistung existieren kann gibt es eine Kardinalität zwischen Modul und Exam
- Hier werden alle wichtigen Daten bezüglich der Prüfung aufgeführt

LearnTimes:

- Zwischen LearnTimes und StudyProgramm gibt es eine Kardinalität und ebenfalls herrscht ein Aggregation Zustand.
- LearnTimes ist für alles bezüglich der Lernzeiten zuständig