

SE2, Aufgabenblatt 1

Modul: Softwareentwicklung II – Sommersemester 2020

Eclipse, Projekt Mediathek, Interfaces

Moodle-Kurs Softwareentwicklung 2 SoSe2020

Ausgabedatum 20. April 2020

Kernbegriffe

Eine *integrierte Entwicklungsumgebung* (engl.: integrated development environment, kurz IDE) ist beim Programmieren neben der Programmiersprache ein wichtiges Werkzeug. Sie stellt u.a. Editoren für verschiedene Quelltexte zur Verfügung, lässt Quelltexte durch den Compiler übersetzen und bietet Unterstützung beim Debuggen von Programmen. Sie integriert viele der zentralen Tätigkeiten bei der Softwareentwicklung unter einer gemeinsamen Oberfläche.

In den Quelltextkonventionen unterscheiden wir zwei wichtige Arten von Kommentaren: *Implementationskommentare* und *Schnittstellenkommentare*. Während Implementationskommentare sich an Wartungsprogrammierer richten, die den Code einer Klasse verstehen und ändern müssen, dienen Schnittstellenkommentare dazu, die Schnittstelle einer Klasse verstehen und nutzen zu können. Die Sprache Java unterstützt die Unterscheidung, indem sie unterschiedliche Kommentar-Arten anbietet. Implementationskommentare werden mit den Zeichen `,/'` eingeleitet, sie sind nur im Quelltext für den Programmierer sichtbar. Für Schnittstellenkommentare werden so genannte *Javadoc-Kommentare* verwendet, aus denen eine API-Dokumentation generiert werden kann. Hierfür steht das konfigurierbare Werkzeug *javadoc* zur Verfügung, das in viele IDEs integriert ist. BlueJ z.B. benutzt es, um die Schnittstellen-Sicht einer Klasse im Editor zu erzeugen. Javadoc wertet auch die sogenannten *Javadoc-Tags* innerhalb dieser Kommentare aus, das sind vordefinierte Schlüsselwörter für bestimmte Aspekte der Dokumentation. Beispiele sind die Tags `@param`, `@return` und `@author`. Die komplette Dokumentation der Java-API ist aus javadoc-Kommentaren generiert.

Checkliste Abgabe

Um die erste Woche abzuschließen, müsst ihr:

- ☐ **Beide in der Gruppe:** Aufgabe Rahmenbedingungen abgeben
- ☐ **Beide in der Gruppe:** Aufgabe Kernbegriffe abgeben
- ☐ **Beide in der Gruppe:** Aufgabe SE1-Wissen abgeben
- ☐ **Beide in der Gruppe:** Scheinkriterien lesen
- ☐ **Einmal:** UML Klassendiagramm und Erklärung im Moodle hochladen
- ☐ **Beide in der Gruppe:** Aufgaben 1.1, 1.2, 1.3 und 1.4 im Moodle abgeben

Aufgabe 1.0 Scheinkriterien lesen

Um sicher zu gehen, dass jeder weiß, wie die Scheinkriterien aussehen, sollt ihr sie euch einmal durchlesen. Ihr findet die Scheinkriterien oben im Moodle. Ihr müsst **selber** darauf achten, dass ihr diese erfüllt um den Übungsschein zu erhalten. Ihr dürft zwar unabhängig davon die Klausur mitschreiben, allerdings benötigt ihr den Übungsschein, um das Modul zu bestehen. Falls ihr den Übungsschein nicht erhaltet, müsst ihr nächstes Jahr noch einmal an der Übung teilnehmen. Hakt bitte rechts neben dem Dokument im Moodle an, dass ihr es gelesen habt.

Aufgabe 1.1 Wechsel der Entwicklungsumgebung

BlueJ ist gut für den Einstieg in die objektorientierte Programmierung geeignet, für die Arbeit an größeren Projekten jedoch nicht. Deshalb steigen wir an dieser Stelle auf die Entwicklungsumgebung *Eclipse* um. Eclipse ist frei verfügbar und wird auch im professionellen Kontext häufig eingesetzt. Ihr könnt Eclipse auf der offiziellen Webseite <http://www.eclipse.org/download> herunterladen. Um Java 8 herunterzuladen geht ihr auf <https://www.java.com/download/>.


In dieser Aufgabe geht es darum, Eclipse kennen zu lernen. Es macht nichts, wenn ihr den Quelltext nicht komplett versteht. Für den ersten Umgang mit Eclipse werden wir das Projekt *Mediathek* nutzen.

- 1.1.1 Startet Eclipse, indem ihr im Windows-Startmenü „eclipse“ eingibt und dann „Eclipse“ auswählt. Beim ersten Starten erhaltet ihr einen Dialog, in welchem ihr den so genannten Workspace angibt. Dies ist ein Ordner, in dem Eclipse alle Projekte ablegt und verwaltet. Sucht euch selber einen geeigneten Ordner aus. Falls dieser Dialog nicht erscheint, kann der Workspace über *File>Switch Workspace>Other...* geändert werden.
- 1.1.2 Für einen reibungslosen Austausch von Daten über Plattformgrenzen hinweg sollte die Codierung für Textdateien auf UTF-8 gestellt werden. Geht dazu auf *Window>Preferences>General>Workspace* und stellt dort das *Text file encoding* auf *Other: UTF-8* ein.
Ladet anschließend die Datei *Mediathek_Vorlage_Blatt01-03.zip* aus dem SE2-Moodle auf euren Rechner. In diesem Archiv befindet sich ein Eclipse-Projekt, das wir nun importieren. Wählt dazu in Eclipse *File>Import...* und dann *Existing Projects into Workspace* (unter *General*) und gebt im folgenden Schritt für **Select archive file** die Datei *Mediathek_Vorlage_Blatt01-03.zip* an.
Wechselt mit *Window>Open Perspective>Java* in die so genannte *Java-Perspektive*. Wenn auf der linken Seite kein *Package Explorer* geöffnet sein sollte, öffnet ihn mit *Window>Show View>Package Explorer*. Klappt das Mediathek-Projekt auf und schaut euch an, wie Eclipse Klassen und deren Quelltexte darstellt (per Doppelklick auf eine Klasse wird ein Editor geöffnet).
- 1.1.3 Macht euch klar, was die verschiedenen Fenster der Eclipse Java-Perspektive anzeigen (wenn nicht gleich alles verständlich ist – nicht so schlimm). Findet heraus, was die *Outline-View* ist, und beantwortet dann die Fragen in der korrespondierenden Aufgabe **im Moodle**.
- 1.1.4 Eclipse kann euch beim Einhalten der Quelltextkonventionen helfen. Ladet aus dem Moodle die Datei *SE2Format.xml* herunter. Wählt dann *Preferences>Java>Code Style>Formatter*, drückt auf *Import...* und wählt die heruntergeladene XML-Datei. Bestätigt zuletzt mit OK. Unter *Preferences>Java>Editor>Save Actions* und dort den Haken bei *Perform the selected actions on save* und dann bei *Format source code* setzen.

Aufgabe 1.2 Mediathek kennen lernen

Mit dieser Aufgabe lernt ihr die Mediathek genauer kennen. Es handelt sich dabei um ein Entwicklungsprojekt, das uns während der gesamten Laborphase 1 begleiten wird. Die Mediathek ist eine Software für Mediathekare. Ein Mediathekar bedient über einen Touchscreen die Programmoberfläche, leiht Medien an Kunden aus und nimmt diese wieder zurück. Die aktuelle Version der Mediathek enthält noch nicht die gesamte benötigte Funktionalität. In den nächsten Wochen werdet ihr die Software nach und nach ausbauen. Die grafische Benutzungsoberfläche wird von uns gestellt. Wir sind für die Umsetzung der benötigten Funktionalität verantwortlich.

1.2.1 In BlueJ lassen sich Exemplare interaktiv erzeugen und beliebige Operationen an diesen Exemplaren aufrufen. Das ist eine besondere Eigenschaft von BlueJ, auf die wir nun verzichten müssen. Üblicherweise werden Java-Programme über die Methode `public static void main(String[] args)` gestartet. Wählt im Package Explorer die Klasse `StartUpMediathek_Blatt_01_03` aus und startet die Methode `main`, indem ihr *Run As>Java Application* aus dem Kontextmenü der Klasse auswählt. Spielt mit der Oberfläche herum. Welche Funktionalität ist bereits implementiert? Was wird noch nicht unterstützt, sollte aber in keiner Mediathek fehlen? Beantwortet diese Fragen online in der Aufgabe 1.2 im Moodle. Beachtet, dass auch 1.2.2. in der selben Aufgabe bearbeitet wird.

 1.2.2 Nun schauen wir uns den Quelltext näher an. Im src-Ordner findet ihr mehrere Klassen und Interfaces. Zu Beginn sind nur einige Klassen und Interfaces für uns interessant. Diese Aufgabe dient dazu, sich erst einmal zurechtzufinden. Beantwortet die folgenden Fragen im Moodle:

- ☐ Wie viele Test-Klassen gibt es?
- ☐ Die Klassen, die die grafische Benutzungsoberfläche gestalten, sind für uns nicht wichtig. Woran kann man sie erkennen?
- ☐ Die Benutzungsoberfläche arbeitet mit einem `VerleihService`. Schaut euch das Interface an. Welche Dienstleistungen bietet es durch seine Operationen an?

1.2.3 **Zeichnet ein UML-Klassendiagramm**, ausgehend vom Interface `VerleihService`. Im Moodle befindet sich eine Zusammenfassung der UML Pfeile. Das Klassendiagramm soll den `VerleihService` zeigen sowie alle Klassen und Interfaces, die der `VerleihService` direkt oder indirekt benutzt. Operationen und Variablen müssen nicht eingezeichnet werden. Das Klassendiagramm soll folgende Klassen und Interfaces enthalten: `VerleihService`, `Verleihkarte`, `Medium`, `CD`, `Geldbetrag`, `Kunde`, `Datum`, `PLZ`, `Kundennummer`.

Für das Diagramm könnt ihr draw.io oder [Lucidchart](https://www.lucidchart.com) verwenden, oder einfach eine schriftliche Zeichnung einscannen und hochladen. Im Moodle findet ihr unter Tools eine Anleitung, wie ihr zu den Tools kommt. Exportiert das erstellte Klassendiagramm als PDF oder Bilddatei.

Schreibt eine schriftliche Erklärung, wo ihr die Information gefunden habt, welchen Pfeil ihr zwischen welche Klassen setzen musstet. So können wir euren Gedankengang besser nachvollziehen und hilfreiches Feedback geben. Ihr könnt diese Erklärung als separates PDF, Word-Dokument oder ähnliches hochladen, oder gleich die beiden Dokumente zusammenführen. Achtet darauf, dass das UML gut lesbar bleibt!

Das UML und die Erklärung müsst ihr nur einmal pro Gruppe unter Abgabe UML hochladen. Wie ihr eine Abgabe hochladen könnt, seht ihr im Moodle unter Abgabeanleitung. Bewahrt die Datei, bzw. die Zeichnung zum Weiterarbeiten für **spätere Zettel** auf!

Aufgabe 1.3 Aufgabenteilung über Interfaces kennen lernen

Das GUI-Team arbeitet parallel zum SE2-Team. Damit die beiden Teams möglichst unabhängig voneinander arbeiten können, haben sie sich auf das Interface `VerleihService` geeinigt, über das die notwendigen Informationen ausgetauscht werden.

- 1.3.1 Das GUI-Team wollte nicht auf die Implementation des SE2-Teams warten und hat deshalb schon einmal einen so genannten *Dummy* geschrieben, der das Interface `VerleihService` implementiert. Untersucht die Klasse `DummyVerleihService`. Welchem Zweck dient sie? Implementiert sie das Interface so, wie die Autoren von `VerleihService` es sich gedacht haben? Welche Grenzen hat die Implementation? **Beantwortet im Moodle**, was jede Methode im Gegensatz zur Beschreibung aus dem Interface macht.
- 1.3.2 Zum Glück gibt es mittlerweile eine Implementation des SE2-Teams, die besser funktioniert: `VerleihServiceImpl`. Wir wollen nun den `DummyVerleihService` ersetzen. Hierzu müssen wir erstmal herausfinden, an welcher Stelle ein Objekt dieser Klasse erzeugt wird. Klickt im Editor der Klasse `DummyVerleihService` mit rechts auf den Konstruktornamen und wählt *References->Project*. Es öffnet sich eine View, in der angezeigt wird, wo der Konstruktor überall aufgerufen wird. Dies ist in unserem Programm nur eine Stelle. Mit einem Doppelklick könnt ihr nun dorthin springen und stattdessen den Konstruktor des `VerleihServiceImpl` einsetzen. **Beantwortet im Moodle**, warum ihr so einfach ein Objekt einer anderen Klasse an diese Stelle verwenden könnt.
- 1.3.3 Startet das Programm und testet die Oberfläche. Beantwortet folgende Frage **im Moodle**: Welcher Fehler tritt jetzt noch beim Verleih eines Mediums auf? Mit diesem Problem beschäftigen wir uns in der kommenden Woche.
- 1.3.4 Falls ihr eurem Partner das Java Projekt zur Verfügung stellen wollt, müsst ihr das Projekt exportieren. Das geht in Eclipse mit dem Menüpunkt *File->Export*. Dort wählt ihr *General->Archive File*. Auf späteren Zetteln werdet ihr so auch einige Aufgaben abgeben.

Aufgabe 1.4 Hello World

In der ersten Aufgabe habt ihr ein bestehendes Projekt importiert. Es schadet aber auch nicht zu wissen, wie man ein neues Projekt „from scratch“ anlegt. Findet selbständig heraus, wie das geht.

- 1.4.1 Klickt im Moodle auf Aufgabe 1.4
- 1.4.2 Legt eine Klasse namens "Hello" an, deren main-Methode "Hello World" auf die Konsole ausgibt.
- 1.4.3 Wenn ihr sicher seid, dass die Klasse richtig definiert ist, klickt auf Prüfen. Ihr bekommt sofort Feedback, ob eure Antwort korrekt ist.
- 1.4.4 Mit Versuch beenden und dem Klick auf Abgabe schließt ihr die Aufgabe ab.