

# SE2, Aufgabenblatt 1

Modul: Softwareentwicklung II – Sommersemester 2016

## Eclipse, Projekt Mediathek, Interfaces

CommSy-Projektraum ..... SE2 CommSy SoSe 2016

Ausgabedatum ..... 07. April 2016

### 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.

### 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. Auf den Rechnern der SE2-Labore ist Eclipse bereits installiert. (Für die Installation auf eigenen Rechnern könnt ihr Eclipse von der Website des Eclipse-Projekts herunterladen, <http://www.eclipse.org/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 Luna“ auswählt. Beim ersten Starten erhaltet ihr einen Dialog, in welchem ihr den so genannten Workspace angebt. Dies ist ein Ordner, in dem Eclipse alle Projekte ablegt und verwaltet. Wählt hier einen Unterordner in eurem Home-Verzeichnis (unter Windows ist dieses als Netzlaufwerk „Z:“ eingebunden, wählt also z.B. „Z:\SE2\workspace“). Falls dieser Dialog nicht erscheint, muss 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-Commsy 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 der *Outline-View* ist, und erklärt es eurem Betreuer/eurer Betreuerin.
- 1.1.4 Eclipse kann euch beim Einhalten der Quelltextkonventionen helfen. Ladet aus dem CommSy 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. Der Formatierer läuft allerdings nicht selbständig im Hintergrund. Der Shortcut zum Formatieren ist Strg-Shift-F.

## 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 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?
-  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. **Erarbeitet euch schriftlich** die Antworten zu den folgenden Fragen, so dass ihr sie den BetreuerInnen erklären könnt:
  - ☐ 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`. 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 Diagramm werdet ihr auf den nächsten Aufgabenblättern brauchen.

Ihr könnt das Diagramm auf einem Blatt Papier zeichnen oder beispielsweise den unter Windows installierten freien UML-Editor Violet UML verwenden, zu finden über *Start → Tools → Violet 2.0.1 UML*. Ein System-unabhängiges Jar-File gibt es unter <http://violet.sourceforge.net/>.

## 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? **Schreibt als Schnittstellenkommentar zu jeder Methode, was sie 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. **Schreibt auf**, warum ihr so einfach ein Objekt einer anderen Klasse an diese Stelle verwenden könnt.

- 1.3.3 Startet das Programm und testet die Oberfläche. Welcher Fehler tritt jetzt noch beim Verleih eines Mediums auf? Mit diesem Problem beschäftigen wir uns in der kommenden Woche.
- 1.3.4 In den kommenden Wochen werden wir an demselben Projekt weiterarbeiten. Sorgt ab jetzt nach *jedem* Übungstermin dafür, dass beide Programmierpartner die Mediathek auf ihren Netzlaufwerken zur Verfügung haben. Ihr könnt das Projekt aus Eclipse exportieren mit dem Menüpunkt *File->Export*. Dort wählt ihr *General->Archive File*. Führt euren BetreuerInnen vor, dass ihr das Projekt mit beiden Usern öffnen könnt.

## 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, indem ihr ein Eclipse-Tutorial im Internet durchforstet oder in ein Eclipse-Buch hineinschaut.

- 1.4.1 Legt ein neues Java-Projekt namens `Hello_World` an.
- 1.4.2 Fügt dem Projekt eine neue Java-Klasse `Hello.Java` hinzu.
- 1.4.3 Schreibt in dieser Klasse eine main-Methode, die einen Begrüßungstext auf die Konsole schreibt.
- 1.4.4 Startet das Projekt.