SE2, Aufgabenblatt 1

Modul: Softwareentwicklung II – Sommersemester 2012

Eclipse, Projekt Mediathek, Interfaces, Testen

CommSy-Projektraum ..... SE2 CommSy SoSe 2012
Ausgabedatum ..... 05. April 2012

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* (bitte lest das nach!). 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/. Eclipse ist in verschiedenen Varianten verfügbar, für SE2 ist die „Eclipse IDE for Java Developers" sinnvoll.)

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

1.1.1 Öffnet das Programm Eclipse. Beim ersten Starten erhaltet ihr einen Dialog, in dem 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\eclipse_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 unterstützen. Mit *Strg+Shift+F* startet ihr das automatische *Code-Formatting*. Unter *Preferences>Java>Code Style>Formatter* müsst ihr es noch an unsere Konventionen anpassen. Für SE2 gibt es neue, erweiterte Konventionen. Formatiert eure Quelltexte entsprechend. Exportiert euren Formatter als Datei, damit ihr die Konventionen auch auf anderen Rechnern importieren könnt.

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 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 alle benötigte Funktionalität. In den nächsten Wochen werdet ihr die Software nach und nach ausbauen. Die grafische Benutzungsoberfläche wird von einem separaten Team programmiert. Wir sind für die Umsetzung der benötigten Funktionalität verantwortlich.

1.2.1 In BlueJ lassen sich Exemplare interaktiv erzeugen und beliebige Operationen dieser Exemplare 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 `StartUp` 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 guten 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 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.

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 Hausaufgabe: sechs Tutorialvideos zur Eclipse IDE durcharbeiten

Die Zeit bis zum zweiten Übungstermin in der kommenden Woche solltet Ihr zum Kennenlernen der Eclipse IDE nutzen. Sie besitzt eine bestimmte Logik bei der Präsentation von Daten in Views und Editoren, deren Verhalten es kennenzulernen gilt. Außerdem gibt es viele zeitsparende Shortcuts. Eine der einfachsten Arten hierfür ist es, sich zeigen zu lassen, wie alles funktioniert. Hierfür gibt es unter http://eclipsetutorial.sourceforge.net Videos von Mark Dexter. Die für uns relevanten Lektionen findet ihr als Material verlinkt im CommSy.

1.4.1 Ladet die Archiv-Datei *workbench_tutorial.zip* über den Link im CommSy herunter und entpackt die Inhalte in einen neuen Ordner. Das Verzeichnis enthält alle Dateien, die ihr benötigt. Wechselt in den Ordner *lesson01* und öffnet die Datei *lesson01.html* in einem Browser. *Hinweis: Der Adobe-Flash-Player muss installiert sein*! Schaut euch das Video an und arbeitet die dort gezeigten Aufgaben selbstständig durch.

1.4.2 Schaut euch nacheinander die Tutorialvideos in *lesson02* bis *lesson06* an und arbeitet die jeweiligen Aufgaben selbstständig durch.

Aufgabe 1.5 Hausaufgabe: gute Tests schreiben lernen

Gute Tests zu schreiben ist schwierig. Es braucht viel Erfahrung hierfür, weswegen wir das Testen auch immer wieder zum Thema unserer Übungsaufgaben machen werden. Wer sich mit den folgenden Texten auseinandergesetzt hat, ist hierbei klar im Vorteil.

1.5.1 Lest euch die Kapitel *4. Writing Tests* und *7. Best Practices* der Internetseite http://junit.sourceforge.net/doc/faq/faq.htm sorgfältig durch.