Fachinformatik
Anwendungsentwicklung Klasse FI 24

Projektarbeit Java III

ByteCodeTool

Autor Malte Macé

Betreuer Sven Lilienthal

29. 06. 2025

Inhaltsverzeichnis

1 Kurzbeschreibung des Auftrags 3

2 Hinweise zur verwendeten Hard- / Software 4
2.1 Hardware 4
2.2 Software 4
2.2.1 Auswahl der Software und Entwicklungssprachen 4

3 Ressourcenplanung 6

4 Kostenkalkulation 7

5 Prozesschritte und -ergebnisse 8
5.1 Analyse 8
5.2 Entwurf 8
5.3 Realisierung 9
5.4 Fazit 10

Anhang 14

1 Kurzbeschreibung des Auftrags

Im Rahmen des Moduls Java III der Umschulung zum Fachinformatiker Anwendungsentwicklung wurde von den Teilnehmenden verlangt, eine eigenständige Projektarbeit im Umfang von 80 Stunden durchzuführen. Hierbei standen Projektvorschläge des Dozenten zur Auswahl, alternativ konnten eigene Projektvorschläge eingereicht werden. Bei dem umgesetzten Projekt, dem ByteCodeTool, handelt es sich um einen Projektvorschlag des Dozenten.
Das ByteCodeTool ist ein Werkzeug zur grafischen Aufbereitung und Analyse binärer Java class-Dateien. Mit Hilfe der grafischen Ausgabe kann der User Ansatzpunkte für das Reverse Engineering der eingelesenen binären Klassendatei finden. In seinem Aussehen und seiner Funktionalität orientiert sich das Programm dabei an der Freeware DirtyJOE v1.7, allerdings ohne den vollen Umfang der Vorlage umzusetzen. Dabei wurde sich für einen Entwurf der Programmarchitektur nach dem Model-View-Controller Design Pattern entschieden. Eine Liste der Anforderungen befindet sich im Anhang unter Anhang A.

2 Hinweise zur verwendeten Hard- / Software

2.1 Hardware

Für die Erstellung der Projektarbeit stand dem Teilnehmer zum einen sein im Rahmen der Umschulung bei cbm zur Verfügung gestellter Desktop-Computer zur Verfügung. Unglücklicherweise hat er versäumt, die technischen Daten der Kiste auszulesen und jetzt ist Wochenende und der Laden hat zu. Das sollte aber auch nicht weiter gravierend sein, da es sich nicht um eine besonders ressourcenintensive Anwendung handelt. Das Betriebssystem ist Windows 10 oder Windows 11.
Die zweite Maschine, auf der gearbeitet wurde, war das private Lenovo ThinkPad X240 von anno Tobak des Teilnehmers, mit vier Intel® Core™ i5-4210U Prozessoren und 8GB RAM. Das Betriebssystem hierauf ist Ubuntu 24.04.2 LTS.
Es bestanden keine realistischen Optionen zur Nutzung anderer Hardware, die Ausstattung erwies sich allerdings als funktional, wenn auch in Teilen unbequem durch geringe Bildschirmgröße und zwei unterschiedliche Betriebssysteme.

2.2 Software

Die IDE zur Entwicklung des ByteCodeTool war Eclipse, um die Entwicklungssprache Java möglichst bequem umsetzen zu können. Versionskontrollsystem war git, beziehungsweise github.

2.2.1 Auswahl der Software und Entwicklungssprachen

Die Auswahl der Entwicklungssprache, des Editors und des Versionskontrollsystems fand nach Nutzwertanalysen statt. Bei der Gewichtung wurde dabei besonderen Wert auf die Einarbeitung in, sowie die zur Verfügung stehenden Hilfe für, die jeweiligen Sprachen und Applikationen gelegt.

	Gewichtung	python	Punkte	Java	Punkte	C++	Punkte
Einarbeitung	50%	7	3,5	9	4,5	2	1,0
Lernmaterialien	20%	9	1,8	9	1,8	8	1,6
Testframeworks	10%	7	0,7	8	0,8	8	0,8
Langlebigkeit	10%	9	0,9	8	0,8	8	0,8
Schnittstellen	10%	6	0,6	9	0,9	9	0,9
Gesamtsumme	100%		7,5		8,8		5,1

Tabelle 2.1 Nutzwertanalyse Entwicklungssprache

Eine Besonderheit ergab sich bei der Wahl des Editors aufgrund der persönlichen Präferenzen des Autors:

	Gewichtung	Emacs	Punkte	Eclipse	Punkte
Nutzerkomfort	50%	2	1,0	8	4,0
Einarbeitung	30%	2	0,6	8	2,4
Anforderungen	10%	3	0,3	8	0,8
Masochismusfaktor	10%	10	1,0	1	0,1
Gesamtsumme	100%		2,9		7,3

Tabelle 2.2 Nutzwertanalyse IDE

3 Ressourcenplanung

Zur Umsetzung des Projektes war die wichtigste Ressource die vorhandene Zeit. Alle anderen Ressourcen, wie zum Beispiel Geldmittel, waren entweder fiktiv, oder bereits vorhanden, wie etwa die Hardware. Daher wurde in der Planungsphase ein Gantt-Diagramm der Arbeitspakete mit geschätzten Werten für den jeweiligen Zeitaufwand angefertigt.

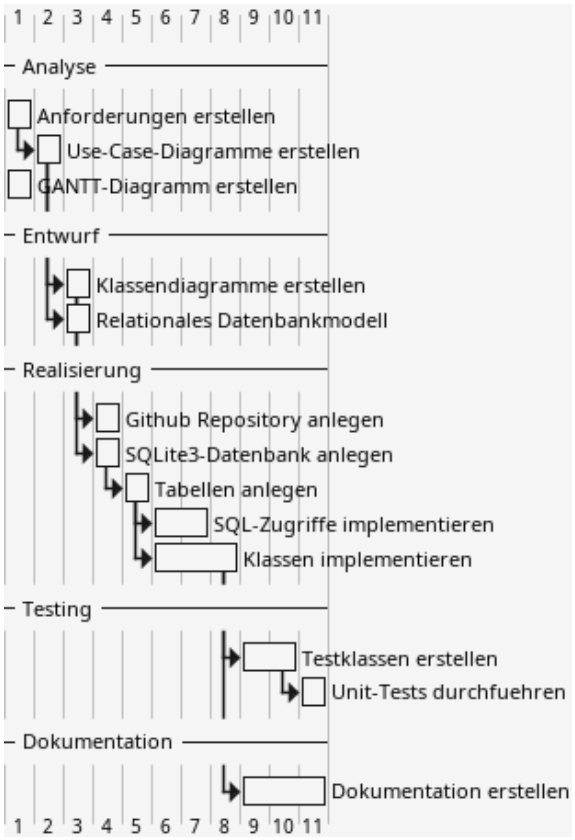

Abbildung 3.1 Geplanter Zeitaufwand

Die zeitliche Planung und die Reihenfolge der Arbeitsschritte wurden jedoch sehr früh aufgegeben. Der Hauptgrund hierfür bestand darin, dass der Teilnehmer zwar bereits einigermaßen erfahren war, was den Umgang mit den Java Swing Bibliothekenanging, allerdings bisher noch keinerlei Erfahrung mit der Analyse von binären gemacht hatte oder sich mit der Java Virtual Machine (JVM) in der Tiefe auseinandergesetzt. Ein sinnvoller Entwurf von Klassendiagrammen stellte sich daher für die Teile des Programms, welche nicht das GUI beinhalteten, als schwierig heraus. Weitere Ausführungen folgen in Kapitel 5.

4 Kostenkalkulation

Die Kostenkalkulation für die Entwicklung des Gartenplaners muss zwangsläufig aus einer Reihe an Schätzungen, guesstimates und Fantasiewerten bestehen, da die meisten Löhne und Preise für die Beteiligten Personen nicht wirklich zu ermitteln sind, bzw. teil von Pauschalen sind. Nichtsdestotrotz soll eine exemplarische Rechnung die ungefähren Kosten verdeutlichen.
Die größten Kostenpunkte sind hierbei der Lohn des Entwicklers und des Betreuers der Arbeit. Beide Größen sind nicht präzise bekannt. Daher werde ich für den Entwickler vereinfachend annehmen, dass sein Lohn sich aus dem monatlichen Bürgergeldsatz, umgelegt auf die anteilige Projektarbeitszeit an der monatlichen Schulzeit ergibt. Für den Betreuer der Arbeit, welcher auch während des Projektes beriet und Hilfestellung gab, wurde nach oberflächlicher Recherche der Stundenlohn auf etwa 30€ geschätzt.

Posten	Faktor	Satz	Summe
Entwicklungsstunden	80	6,65€	532,00€
Betreuungsstunden	5	30,00€	150,00€
Kosten Hardware	1	16,66€	16,66
Kosten Software	1	0 €	0€
Gesamtsumme			698,66

Tabelle 4.1 Kostenkalkulation

Da die Software zur Erstellung der Arbeit ausschließlich Open Source bzw. Freeware war, wurden die Softwarekosten mit 0€ kalkuliert. Die Hardware befand sich zum einen am Arbeitsplatz und war zum anderen der private Laptop des Teilnehmers. Beide zusammen wurden mit 1000€ bepreist, bei einer Abschreibung auf null innerhalb von 60 Monaten (5 Jahren).

5 Prozesschritte und -ergebnisse

5.1 Analyse

Während der Analysephase des Projektes wurden die funktionalen und nichtfunktionalen Anforderungen erstellt, eine Liste befindet sich im Anhang A.
Die zeitliche Planung des Projektes wie bereits unter Abbildung 3.1 mit Hilfe eines Gantt-Diagrammes statt.
Seitens des Betreuers der Arbeit wurde zwar ein Mock-Up der grafischen Oberfläche empfohlen, da es sich allerdings um einen Nachbau des dirtyJOE v1.7 handelt, wurde auf ein Mock-Up verzichtet. Nur geringfügige Änderungen wie etwa das parallele Laden einer zweiten Klassendatei oder der Verzicht auf einige der Funktionalitäten des dirtyJOE erübrigten einen Vorabentwurf der grafischen Oberfläche.
Es ließ sich ebenfalls bereits absehen, dass nicht sämtliche Funktionen eines professionellen Bytecodeanalysewerkzeuges im Rahmen des zweiwöchigen Projektes zu implementieren sein würden, daher wurde beschlossen, nur so weit wie möglich die Konstanten, Felder und Methoden der eingespeisten Klassen auszulesen und darzustellen. Ein Ändern der Einträge oder die Darstellung der Methoden wie etwa mit Hilfe von javap -v SOMETHING.class wurde von Beginn an ausgeschlossen.

5.2 Entwurf

In der Entwurfsphase war eigentlich geplant, aus dem letzten Projekt, dem Gartenplaner 3000 aus dem Modul Java II, zu lernen und noch vor dem ersten Öffnen der IDE das Programm komplett in Klassendiagrammen zu entwerfen. Während des Java II-PProjektes hatte es während der Realisierungen Schwierigkeiten gegeben, die Übersicht und saubere Trennung, insbesondere der einzelnen MVC-Komponenten, einzuhalten. Diese Schwierigkeiten wurden auf einen mangelnden Entwurf der Architektur zurückgeführt. Ziel war eigentlich, diese Probleme im Projekt Java III gezielt zu vermeiden, jedoch stellten sich Schwierigkeiten ein:
Bestanden im Projekt Java II die Komplikationen noch in mangelnder Erfahrung im Umgang mit Java Swing und der Umsetzung des MVC-Patterns, so bestand die Schwierigkeit im Projekt Java III darin, dass der Teilnehmer keinerlei Erfahrung im Umgang mit Bytecode oder der Funktionsweise der JVM hatte.
Daraus ergab sich, dass der Entwurf der grafischen View-Klassen möglich war, nicht jedoch der Controller- und Modellklassen. Dies fiel dem Teilnehmer auf, als er begann, die Views für seine Anwendung zu entwerfen. An dieser Stelle begann die intensive Auseinandersetzung mit der Spezifaktion der JVM [1]. Da diese jedoch sehr

umfänglich ist, ging der Teilnehmer dazu über, kleinere Teile der ClassFile-Struktur zu erlernen und anschließend direkt umzusetzen.
Somit entstand dasselbe Muster wie bereits im Projekt Java II: ein Übergang von Entwurf zu Realisierung, ohne den Entwurf vollständig abgeschlossen zu haben. Im Nachhinein entsteht der Eindruck, dass ein voller Überblick über die Funktionsweise der JVM sicher dabei geholfen hätte, einen optimaleren Programmentwurf zu schaffen, im Rahmen eines 80-stündigen Projektes aber nicht oder nur sehr schwierig zu erreichen gewesen wäre.
Positiv hervorheben lässt sich jedoch, dass die Umsetzung des MVC-Design-Patterns sowie der Umgang mit Java Swing deutlich routinierter vonstatten gingen und die Erfahrungen aus dem letzten Projekt einen zufriedenstellenden Grundentwurf der Architektur erlaubten, wenn man von den einzelnen Methoden und Feldern der Modell- und Controllerklassen absieht.

5.3 Realisierung

Einen bewussten Start der Realisierungsphase, nach einer abschließend erfolgten Entwurfsphase, gab es daher also nicht. Dennoch wurde in der Realisierungsphase planvoll vorgegangen:
Da es Ziel war, die grafische Oberfläche des dirtyJOE zu imitieren, wurden zunächst sämtliche Views implementiert. Das reine Erstellen der Views verlangte kein besonderes Wissen über die JVM, es ähnelte eher eine Document-Object-Model Puzzlespiel.
Nach den Views wurde zunächst anhand der JVM-Spezifikation ein Datencontainer für das Modell der Java-Klasse erstellt. Dabei wurden die Felder sowie jeweilige getter und setter modelliert. Auch hier war noch kein besonders tiefes Wissen über die Funktionsweise der JVM vonnöten, aber eine kontinuierliche Auseinandersetzung mit der Dokumentation von Vorteil.
Die größte Schwierigkeit bestand anschließend darin, Mit Hilfe des Controllers die Daten korrekt auszulesen. Einige der Felder waren dabei eher einfach zu befüllen, nämliche jene mit den fixen Längen. Schwieriger war es, mit den flexiblen Längen einzelner Konstanten umzugehen. Hierbei wurde eine abstrakte Klasse erstellt, von der anschließend sämtliche unterschiedlichen Konstantenklassen abgeleitet und mit ihren individuellen Feldern ausgestattet. Auch für die Feld-, Methoden- und Attributinformationen wurden eigene Modelle erstellt. Bei den Attributen wäre eine ähnliche Umsetzung mit Hilfe von Abstraktion und Spezialisierung ebenfalls hilfreich gewesen, gegen Ende des Projektes blieb jedoch keine Zeit mehr, die über 30 unterschiedlichen Kindklassen, welche vonnöten gewesen wären, zu erstellen.

Die Realisierung selbst wurde zum größten Teil selbständig und ohne die Hilfe von KI erledigt. Bei schwierigen Fragen oder Umsetzungen, welche sich auch nach eigener Recherche nicht klären ließen, wurde der Betreuer hinzugezogen und unterstützte mit großem Einsatz die Arbeit.
(Im Nachhinein) Erstellte Klassendiagramme befinden sich im Anhang B.

5.4 Fazit

Mein Fazit für das Projekt ist zweierlei: Wenn ich die strukturierte Umsetzung eines Softwareprojektes als Maßstab nehme, dann sehe ich weiterhin Mängel im Ablaufplan. Ich glaube allerdings, dass diese vor allem darauf zurückzuführen sind, dass es wieder viel Neues zu lernen gab und die Struktur von Analyse über Entwurf zu Realisierung für mich schwer umzusetzen ist, wenn ich kein exaktes Bild der Lage, bzw. Materie habe. Ein Projekt auf einem bereits bearbeiteten Feld würde ich mutmaßlich deutlich leichter innerhalb dieser Projektstruktur umsetzen können, da weniger tiefe Recherche in der Analyse- und Entwurfsphase nötig wäre.
Wenn ich hingegen das Projekt als eine Heranführung an ein für mich neues Feld auf dem Gebiet der Anwendungsentwicklung betrachte, bin ich sehr zufrieden. Die Auseinandersetzung mit Bytecode war sehr hilfreich für mein Verständnis von Maschinencode und ich habe das Gefühl, einen ersten Einblick in das Feld des Reverse Engineering bekommen zu haben.
Mit der Qualität meines Codes bin ich zu etwa 90% zufrieden, aber das bleibt natürlich dem Prüfer zu beurteilen. Ein längere Dauer des Projektes hätte mit Sicherheit noch deutlich mehr Umsetzungen erlaubt.

Abbildungs- und Tabellenverzeichnis

Abbildungsverzeichnis

Abbildung 3.1: Geplanter Zeitaufwand 6

Tabellenverzeichnis

Tabelle 2.1: Nutzwertanalyse Entwicklungssprache 5
Tabelle 2.2: Nutzwertanalyse IDE 5

Tabelle 4.1: Kostenkalkulation 7

Literatur

[1] https://docs.oracle.com. *The Java® Virtual Machine Specification Java SE 24 Edition*. URL: https://docs.oracle.com/javase/specs/jvms/se24/jvms24.pdf (besucht am 29. 06. 2025).

Anhang

Anhangsverzeichnis

A Funktionale / Nicht-Funktionale Anforderungen 16

B Klassendiagramme 17

A Funktionale / Nicht-Funktionale Anforderungen

ByteCodeTool
Funktionale Anforderungen
Anforderung: <Einlesen von Java class Dateien> Beschreibung: <Der User soll in der Lage sein, über ein Menü Java class-Dateien aus seinen Verzeichnissen einzulesen.>
Anforderung: <binäre Dateien auslesen> Beschreibung: <Die Software muss die binären Dateien auslesen und in den ModellContainern ablegen.>
Anforderung: <GUI erstellen> Beschreibung: <Mit Hilfe von awt und swing muss ein GUI zur Anzeige und Steuerung erstellt werden.>
Anforderung: <Darstellung der ModellContainer> Beschreibung: <Das GUI muss die unterschiedlichen ModellDaten in ihren jeweiligen Tabs und Feldern korrekt darstellen.>
Nicht-funktionale Anforderungen
Anforderung: <MVC-Design> Beschreibung: <Die Verbindung von Modell und Darstellung soll mit Hilfe von MVC und Oberser Pattern realisiert werden.>

B Klassendiagramme