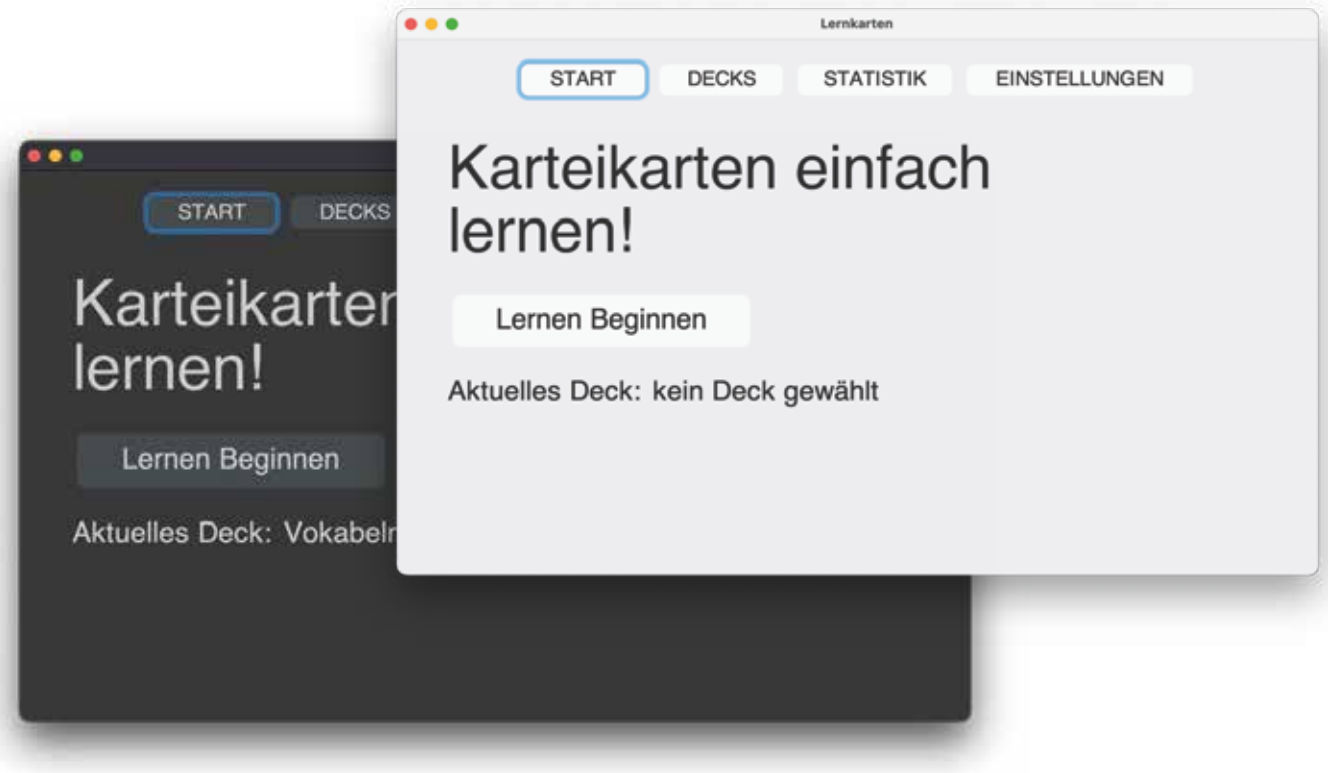
Projektdokumentation
Karteikartenlernapp



Gruppe: Do-08

Julian Dillmann & Mark Sterkel

Programmiertechnisches Praktikum 2022

Inhaltsverzeichnis

MOTIVATION UND GROBBESCHREIBUNG

Während des Brainstormings bezüglich Projektideen werden wir uns schnell einig, dass wir gerne ein Programm mit einem tatsächlichen Nutzen entwickeln würden. Wichtig ist uns, auch ein Projekt zu wählen, das für unerfahrene Entwickler nicht zu komplex ist. Es sollte trotzdem noch viel Erweiterungspotential bieten. So können wir die grundlegenden Funktionen implementieren und sofern noch Zeit übrig ist, weitere Features entwickeln. Da wir in unserem Studentenalltag selbst gerne die Lernsoftware „ANKI“ nutzen, kommt uns die Idee, eine solches Karteikartenlernprogramm zu schreiben.

Eine Karteikartenlernsoftware ermöglicht es Nutzern, Karteikarten mit einer Vorder- und Rückseite zu erstellen, sowie zu lernen. Auf der Vorderseite steht meist eine Frage oder bei Fremdsprachen eine Vokabel und auf der Rückseite die jeweilige Antwort bzw. Übersetzung. Zunächst wird nur die Vorderseite angezeigt, der Nutzer sagt dann die entsprechende Lösung auf. Im Anschluss kann er sich per Knopfdruck die Rückseite mit der Lösung zur Kontrolle anschauen. Die Karteikarten werden pro Themengebiet in sogenannten „Decks“ gruppiert. Meist lernt man alle Karten eines Decks direkt hintereinander.

KONZEPTION UND PROJEKTPLANUNG

Nachdem die Projektidee steht, entwickeln wir mögliche Anforderungen und diskutieren deren Umsetzbarkeit im Rahmen der Aufgabe 1.3 (Projektbeschreibung). Von Beginn an nutzen wir für die Planung das Tool Notion. Hier können mehrere Personen gemeinsam an einem Dokument arbeiten. Auch später im Projektmanagement wird das Tool von großem Vorteil sein.

USER STORIES

Als Student möchte ich Fragen und entsprechende Antworten in einem Modul lernen, damit ich diese schnell beherrsche und in der Klausur wiedergeben kann.

Als Student möchte ich Screenshots aus dem Skript auf der Vorder- und Rückseite der Karteikarten einfügen können, damit ich den Text nicht abtippen muss.

Als Nutzer möchte ich, dass meine Karteikarten auch nach einem Neustart der Anwendung noch verfügbar sind, damit ich diese nicht immer wieder neu vor jedem Lernen erstellen muss.

Als Nutzer möchte ich Fragen und Antworten ändern können, wenn ich z. B. einen Rechtschreibfehler entdecke.

Als Nutzer möchte ich, um Zeit zu sparen, eine übersichtliche Oberfläche haben, die ich schnell überblicken kann.

Als Nutzer möchte ich meinen Lernfortschritt sehen können.

Als Nutzer will ich Karteikarten kategorisieren können, je nachdem wie gut ich die Fragen beantworten kann. Karteikarten sollen gegebenenfalls häufiger angezeigt werden.

ANFORDERUNGEN

Auf Basis unseres Brainstormings und der User Stories legen wir folgende Anforderungen fest:

- Karteikarten können erstellt und gelöscht werden.
- Karteikarten können in Decks organisiert/gruppiert werden.
- Decks und Karteikarten werden auf der externen Festplatte gespeichert (verfügbar nach Programmneustart).
- Karten werden je nach Lernstand öfter oder weniger oft angezeigt (Anzeige-algorithmus).
- Lernfortschritt wird durch eine Statistik angezeigt.
- Auf Karteikarten können Bilder eingefügt werden.
- Eine ansprechende grafische Benutzeroberfläche ist vorhanden.

ZEITPLAN

In unserem Zeitplan finden sich alle Anforderungen mit einer von uns festgelegten Priorität, sowie einer geschätzten, benötigten Stundenzahl wieder. Jede Funktion teilen wir in eine technische und eine GUI-seitige Implementation. Des Weiteren ergeben sich noch Begleit-aufgaben wie z. B. das Erstellen eines UML Diagramms. In der folgenden Notiontabelle halten wir unsere erste Zeitplanung fest.

Fortschritt:

Tasks	Fortschritt	Berarbeiter	Zeit (Soll)	Priorität	Quick Notes	Zeit (Ist)
Planung Klassenstruktur + UML erstellen	-	Julian, Mark	8	mustHave		
Karteikarten erstellen und löschen	-	Julian	10	mustHave	Speicherung?	
GUI: Karteikarten erstellen und löschen	-	Mark	8	mustHave		
Karteikarten können in Decks organisiert/gruppiert werden	-	Julian	10	mustHave		
GUI: Karteikarten können in Decks organisiert/gruppiert werden	-	Mark	8	mustHave		
Decks und Karteikarten werden auf dem Computer gespeichert (sind nach Programmneustart noch verfügbar)	-	Julian	10	mustHave	Recherche über Möglichkeiten: CSV, TXT, XML, Datenbank?	
Optische ansprechende grafische Benutzeroberfläche	-	Mark	10	mustHave	Prototyp in Figma?	
Karten werden je nach Lernstand öfter oder weniger oft angezeigt (Anzeigealgorithmus)	-			niceToHave		
Auf Karteikarten können Bilder eingefügt werden	-			niceToHave		
Anzeigen einer Statistik über den Lernfortschritt	-			notNecessary		
+ New						
			SUM 64			SUM 0

Abbildung 1 (erster Zeitplan in Notion)

IMPLEMENTATIONSKONZEPT

In dem ersten Teil des Praktikums haben wir das Architekturmuster „Model-View-Controller“ kennengelernt und uns in die Struktur des Entwurfsmusters eingearbeitet. Daher ist uns früh klar geworden, dass wir unsere Implementation nach dem MVC-Ansatz aufbauen, um unser gelerntes Wissen zu festigen.

TECHNISCHE UMSETZUNG

Für unser Karteikarten-App ist eine Persistenz elementar, da sonst die erstellten Karteikarten beim Schließen des Programms gelöscht werden. Daher müssen wir uns um ein

Persistenzkonzept kümmern. Nach einer Recherche stehen uns mehrere Formate zur Verfügung:

1. Abspeicherung als XML Dateien
2. Abspeicherung als TXT Dateien
3. Abspeicherung als CSV Dateien

Wir entscheiden uns für die Abspeicherung als CSV Datei (mehr in Kapitel: Implementation des Programms). Da wir jedoch unsere Software modular aufbauen und die Persistenz von der Steuerung getrennt ist, kann man die Abspeicherung in andere Dateiformate unkompliziert erweitern. Nachdem das Konzept der Persistenz entwickelt ist, implementieren wir einen funktionalen Prototyp, bei dem wir die technische Umsetzbarkeit testen (z. B. Erstellen einer CSV Datei u. Speichern von Daten in CSV Dateien). Danach folgt die Implementation.

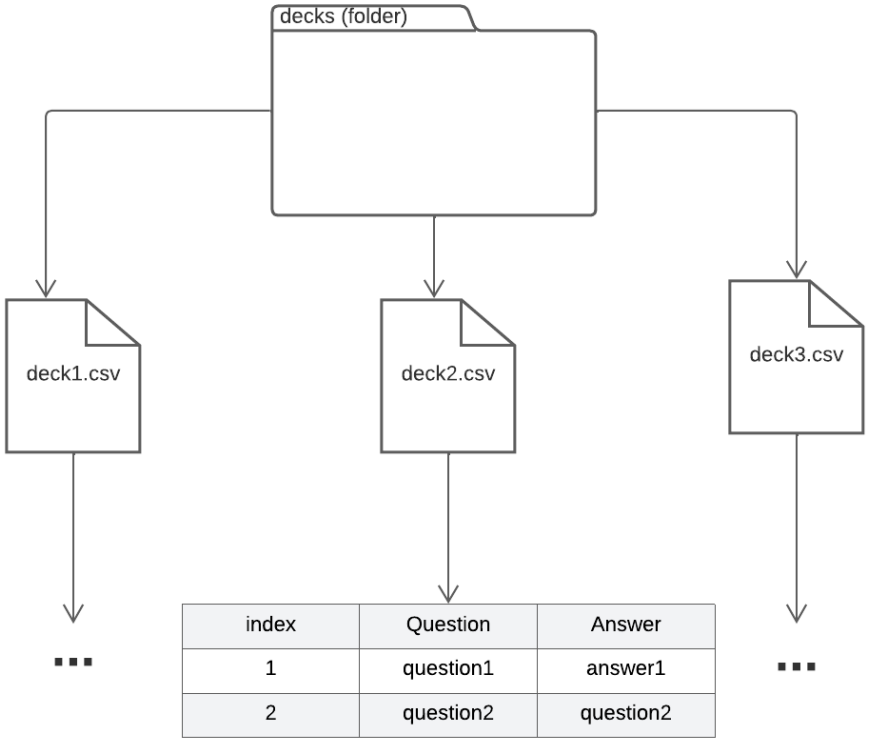


Abbildung 2 Konzept der persistenten Speicherung der Decks mit Karteikarten

GRAPHISCHE OBERFLÄCHE

Um die spätere GUI-Implementation zu vereinfachen, erstellen wir einen Oberflächenprototyp in der Webanwendung Figma. Dadurch haben wir einen konkreten Entwurf für die Aufteilung der GUI-Elemente und können unsere Vorstellungen für das Programmlayout abgleichen. Wir entscheiden uns für eine Navigationsleiste, über welche die verschiedenen Programmseiten erreicht werden können. Der Prototyp beinhaltet den Start-Bildschirm, die Lernansicht, die Deckübersicht und den Deckbearbeiten-Bildschirm.

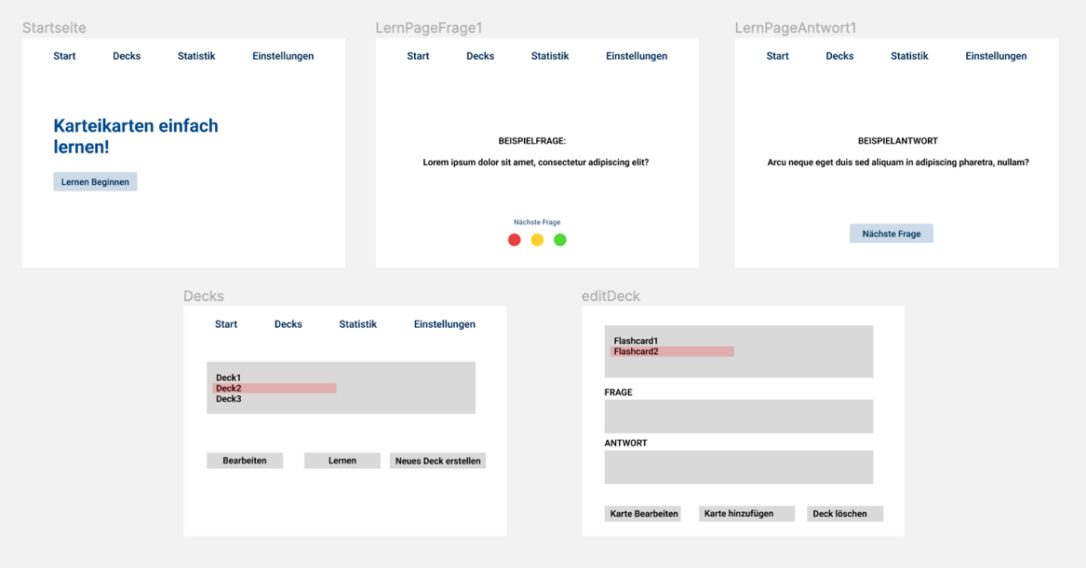


Abbildung 3 (Prototyp in Figma)

TESTKONZEPT/TESTPLAN

Für die Tests möchten wir nach einem ähnlichen Konzept wie in Aufgabe 1 (Draw Projekt) vorgehen. Es soll eine API-Klasse als Schnittstelle geben, in der die jeweiligen Funktionen des Programms angeboten werden. Diese Klasse kann dann mithilfe von JUnit Tests getestet werden. Dabei möchten wir positive wie auch negative Tests einsetzen. Von Anfang an ist uns klar, dass unser Programm zusätzlich auch ausreichend interaktiv getestet werden muss. Nur so können eventuelle Fehler in der GUI gefunden werden. In den Tests müssen die Daten intern und extern verglichen werden. Da die Daten auch extern als CSV Dateien im externen Speicher des Computers gespeichert werden, muss die Konsistenz der Daten intern und extern geprüft werden. Wir versuchen, den TestFirst-Ansatz zu berücksichtigen.

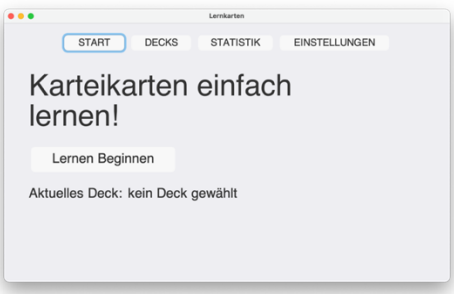
FUNKTIONSBESCHREIBUNG

SYSTEMVORAUSSETZUNG

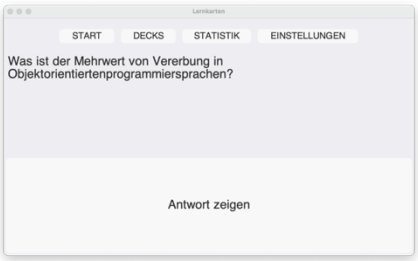
Das Programm läuft als .jar Datei sowohl auf Windows- als auch Mac-Systemen. Voraussetzung ist, dass Java SE 17 installiert ist. Durch den Programmstart wird im Home-Verzeichnis des aktuellen Users ein Ordner mit dem Namen „Decks“ erstellt. In diesem werden während der Programmnutzung die Decks mit ihren Karteikarten als CSV Dateien gespeichert. Ist dieser Ordner bereits vorhanden, werden die vorhandenen Decks eingelesen.
Nach dem Öffnen des Programmcodes in Eclipse müssen gegebenenfalls die Build Paths angepasst werden.

BEDIENUNGSANLEITUNG

Nach dem Starten des Programms öffnet sich zunächst die Startseite. Hier wird das aktuell ausgewählte Deck angezeigt und es kann direkt mit dem Lernen begonnen werden. Das Wechseln der verschiedenen Ansichten erfolgt über die Navigationsleiste mit den Knöpfen *Start*, *Decks*, *Statistik* und *Einstellungen*.

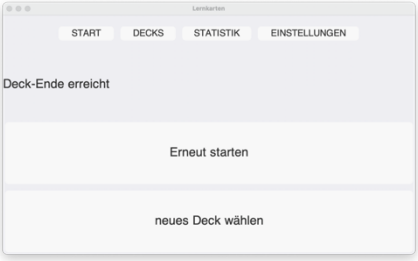


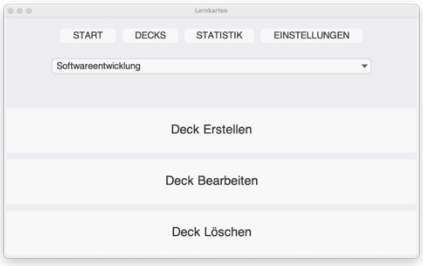
Drückt man auf *Lernen Beginnen* gelangt man direkt zur ersten Karteikarte des gewählten Decks. Erst wird nur die Frage angezeigt. Per Klick auf *Antwort zeigen* (links) kann man sich die passende Lösung anzeigen lassen (rechts). Über *Nächste Frage* gelangt man zur nächsten Karteikarte im Deck.



Ist das Ende des Kartendecks erreicht, gelangt man auf den folgenden Bildschirm. Hier kann zwischen zwei Optionen gewählt werden:

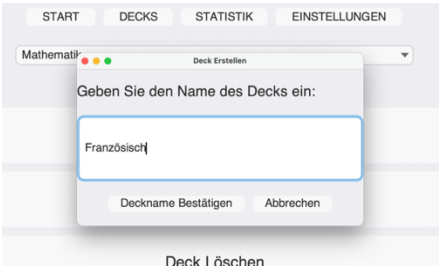
1. aktuelles Deck erneut starten und lernen
2. ein neues Deck auswählen

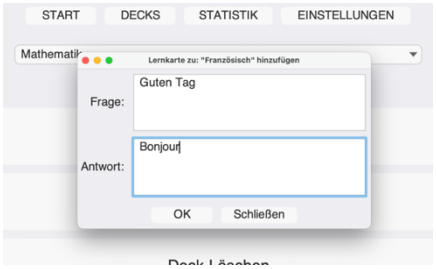




Über *neues Deck wählen* oder jederzeit über die Navigationsleiste kann der Decks-Bildschirm erreicht werden. Hier können Decks ausgewählt, erstellt, gelöscht und bearbeitet werden.

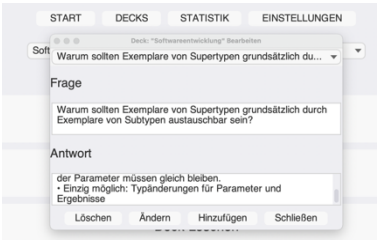
Durch *Deck Erstellen* öffnet sich ein neues Fenster, in dem zunächst der Deckname des neuen Decks angegeben werden muss. Decknamen dürfen nur aus Buchstaben und Zahlen bestehen. Leerzeichen oder bereits vorhandene Decknamen sind nicht zulässig. Der Deckname kann per *Deckname Bestätigen* Knopf oder mit der *Entertaste* bestätigt werden.

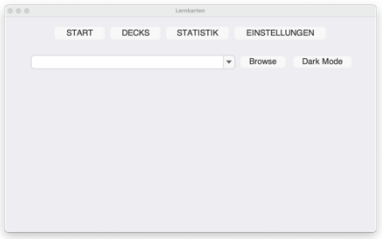




Im Anschluss können die einzelnen Karteikarten mit Frage und Antwort erstellt werden. Nicht zulässig ist hier lediglich ein leeres Textfeld oder ein Text, der nur aus Leerzeichen besteht. Mit *OK* oder alternativ der *Optionstaste* wird eine Karteikarte hinzugefügt, über *schließen* kann der Vorgang abgeschlossen werden. Auf Falscheingaben wird per Fehlermeldung hingewiesen. Das erstellte Deck taucht jetzt im Auswahlmenü auf.

Durch *Deck bearbeiten* öffnet sich ein neues Fenster, in dem die einzelnen Karteikarten aus einem Deck bearbeitet, gelöscht oder hinzugefügt werden können.





Unter Einstellungen kann per Knopfdruck zwischen dem Light- und Darkmode der Anwendung gewechselt werden. Das Wählen eines Pfades für den Decksordner ist bislang noch nicht möglich. Unter Statistik sind ebenfalls noch keine Funktionen verfügbar.

IMPLEMENTATION

Änderungen am Model werden der View über das Observer Pattern mitgeteilt. Die View kann dann entsprechend auf Änderungen reagieren. Wir wollen einen defensiven Programmierstil umsetzen, sodass wir allgemein versuchen, über Exceptions und Exceptions Handling fehlerhafte Eingaben abzufangen und entsprechend den User auf den Fehler hinzuweisen.

MODEL

Die Aufgabe der Persistenz übernimmt das „model“ in unserer Implementation. Dafür verwaltet die Klasse „DeckOrganizer“ die einzelnen Decks, in denen die zugehörigen Karteikarten in einer Liste gespeichert werden. Für die Speicherung der Daten auf dem jeweiligen System des Nutzers nutzen wir das Dateiformat CSV (Comma-separated values). CSV bietet uns folgende Vorteile:

- Das Format ist vordefiniert mit *Index, Frage und Antwort.*
- Das Lesen und Schreiben der Daten ist durch eine einheitliche Struktur einfach zu realisieren.
- Es ist möglich Decks bzw. Karteikarten über ein Tabellenkalkulationsprogramm z. B. Microsoft Excel zu erstellen und dann in unser Karteikartenprogramm zu laden.

Zeilenumbrüche in den jeweiligen Fragen bzw. Antworten haben wir in den CSV-Dateien mit einem Backslash realisiert. Somit werden Absätze z. B. für Auflistungen berücksichtigt.

Damit die internen Daten und die Daten, welche als CSV-Dateien gespeichert sind, Konsistenz aufweisen, wird bei jeder Änderung eines Decks (z. B. Änderung einer Karteikarte oder Hinzufügen einer neuen Karteikarte) die CSV Datei gelöscht und wieder neu geladen. Der Deckmanager ist die Schnittstelle des Modells und bietet den Klienten (View u. Controller) Dienstleistungen an. Somit wird nur von einer Seite auf das Modell zugegriffen und wir verhindern dadurch Seiteneffekte. Die Klassen Deckorganizer, Deck und Flashcard sind von außerhalb nicht zugreifbar. Bei dem Löschen von Dateien erstellt das Betriebssystem macOS sogennante DS.Store Dateien. Damit diese das Laden der CSV Dateien in das Programm nicht beeinflussen, werden nur Dateien mit der Endung „.csv“ geladen. Dies übernimmt die Klasse Deck.

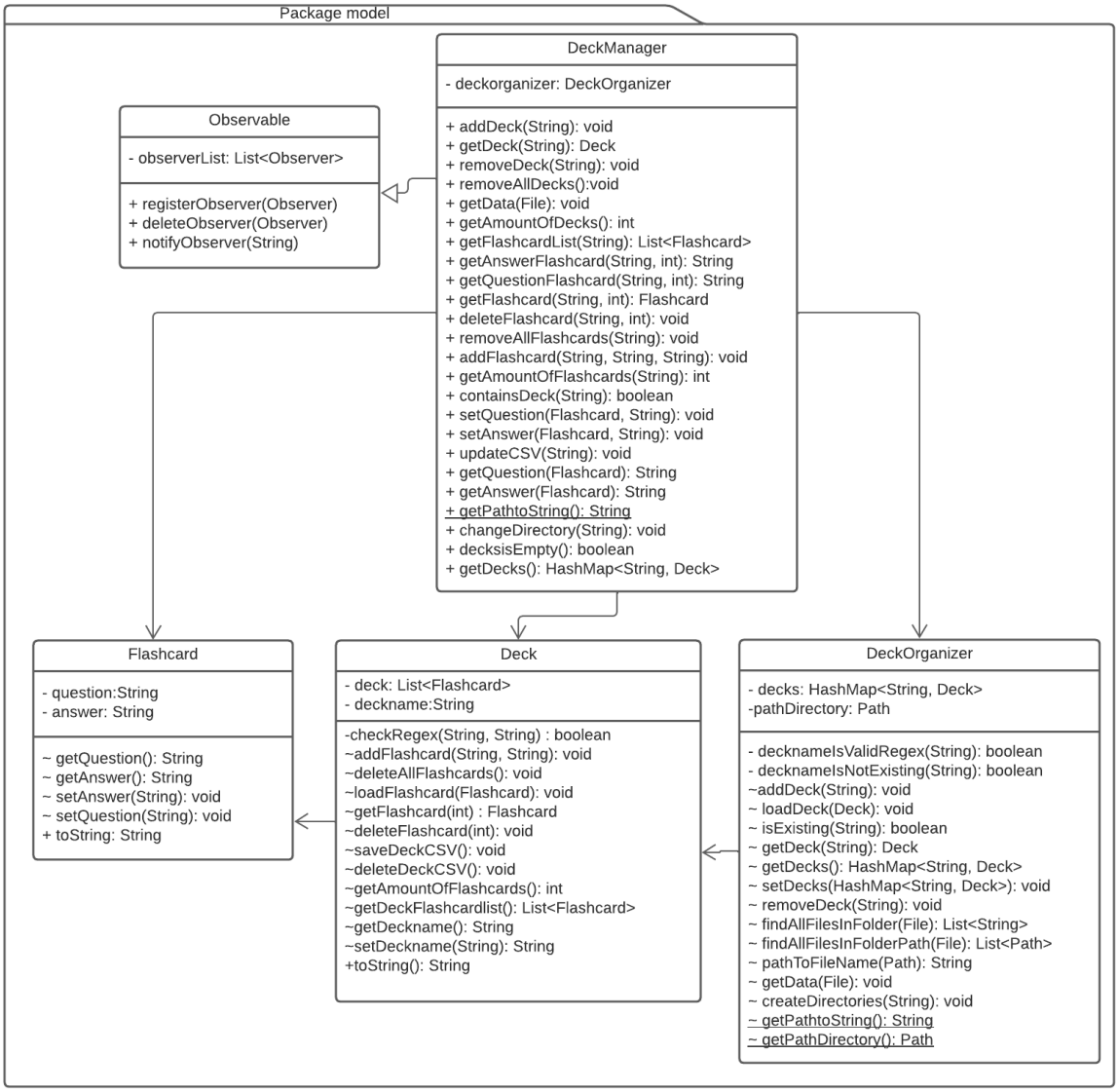


Abbildung 4 *UML model*

VIEW (GUI)

Die GUI ist auf insgesamt sieben Klassen aufgeteilt. Davon besitzt jede einen eigenen Controller (ActionListener), der auf die Interaktionen des Nutzers reagiert. Die „MainGui“ ist die zentrale GUI Klasse. Ihr „JFrame“ öffnet sich beim Starten des Programms. Sie beinhaltet ebenfalls das „topButtonPanel“, welches die Navigationsleiste über vier „JButtons“ realisiert. Dort kann der Nutzer über die verschiedenen Seiten navigieren. In der Implementation ist dies über ein CardLayout realisiert. Wenn der Nutzer auf einen Button klickt, wird durch den MainGuiListener die jeweilige Karte aufgerufen. Beim Erstellen und Bearbeiten eines Decks öffnen sich jeweils weitere Fenster, welche die MainGui überdecken. Eine Besonderheit ist das Spring Layout der „CreateDeckGUI“. Hier nutzen wir die von Oracle implementierte Klasse SpringUtilities. Des Weiteren setzen wir in der Main-Methode ein eigenes Look and Feel. Dafür greifen wir auf die entsprechende Library „flatlaf-themes“ zurück. Die Anwendung des Look and Feel hat drei elementare Vorteile:

1. Design der einzelnen Elemente ist einheitlich
2. einfaches Ändern des Designs (wechseln zwischen Dark u. Light Theme)
3. betriebssystemunabhängiges Design (gleiches Design bei macOS und Windows)

Ebenfalls wird in der main-Methode über „System.getProperty()“ das „theme“ des „frames“ an den aktuell im Betriebssystem gewählten Modus angepasst. Dies ist jedoch nur für macOS verfügbar und notwendig. Wir wollen ein möglichst robustes Programm erstellen, das eine intuitive Bedienung ermöglicht. Deshalb haben wir zum Beispiel eine automatische De-aktivierung des „Lernen Beginnen“ Knopfs implementiert. Der Button wird automatisch ausgegraut sofern kein Deck oder ein Deck, das keine Karteikarten enthält, ausgewählt ist.

CONTROLLER

Zu jeder GUI Klasse gibt es die entsprechende Controller-Klasse, welche auf die Interaktion des Nutzers mit der View (GUI) reagiert. Damit man bei dem Erstellen einer Karteikarte mit Enter bestätigen kann, ohne klicken zu müssen, ist in dem Controller Package ein Key Listener implementiert, welcher dann beim Klicken weiterschaltet.

TESTS

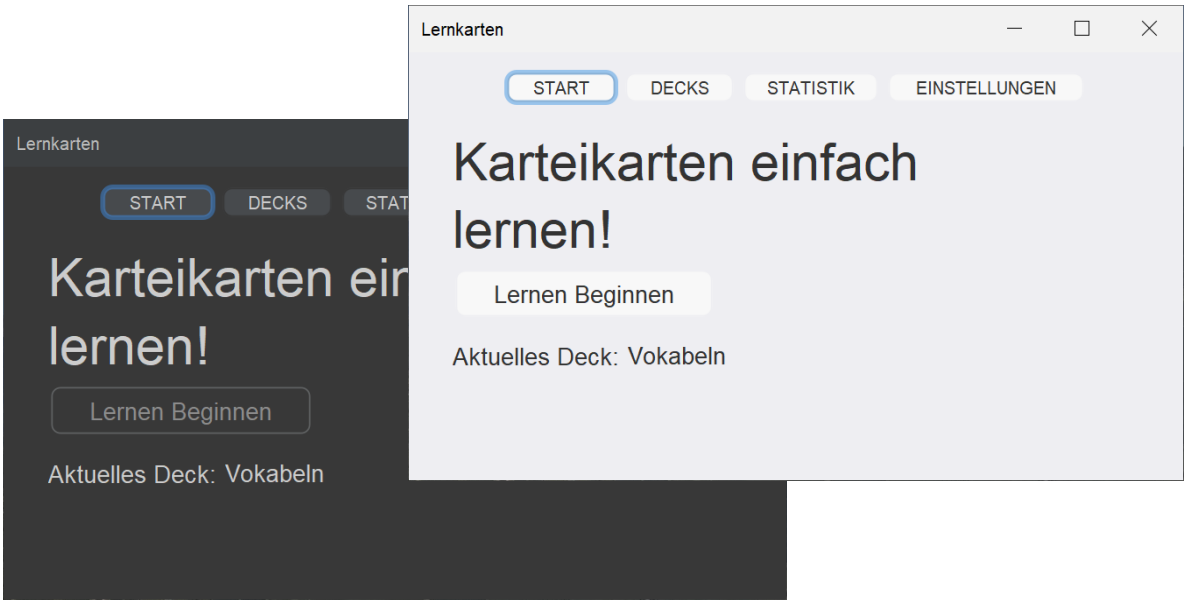
In der Implementation der Tests versuchen wir Positiv- und Negativtests einzubinden. Außerdem testen wir die Konsistenz des Systems, indem wir z. B. durch den Ordner „decks“ iterieren und überprüfen, ob die CSV Datei vorhanden ist oder nicht. Für die JUnit Tests erstellen wir einen eigenen Test-Ordner, in den dann die in dem Test erstellten CSV-Dateien abgespeichert werden. Somit ist gewährleistet, dass beim Ausführen der Tests intern im Programm keine Testdecks zu sehen sind.

FAZIT

STAND DES PROJEKTS

Von unseren definierten Anforderungen schafften wir es nur die „Must-Haves“ umsetzen:

- Karteikarten können erstellt und gelöscht werden.
- Karteikarten können in Decks organisiert/gruppiert werden.
- Decks und Karteikarten werden auf dem externen Speicher gespeichert und sind nach Programmneustart noch verfügbar.
- Die GUI ist optisch ansprechend, durch zum Beispiel Dark- und Light-Mode.



Startseite Design Windows (Dark- u. Lightmode)

Im Laufe des Projekts mussten wir jedoch einige Ideen verwerfen. Dies hatte meistens zeitliche Gründe, jedoch war manchmal die Umsetzung schwer oder gar nicht möglich. CSV-Dateien sind sehr eingeschränkt. Die Idee, dass wir in unserem Programm Screenshots einbinden können, mussten wir erstmal verwerfen. Man kann nur Text in CSV-Dateien abspeichern und somit ist es nur möglich, eine Referenz auf ein Bild zu speichern. Dies ist uns noch nicht gelungen. Des Weiteren ist es uns noch nicht gelungen eine Statistik oder einen Lernalgorithmus einzubinden. Den Reiter haben wir dennoch schon in der Navigationsleiste verankert, da eine Implementation zeitnah erfolgen soll. Auch das Wählen eines Speicherpfades für den Decks-Ordner soll bald in den Einstellungen möglich sein.

ARBEITSTEILUNG

Für ein solches Projekt ist die Kommunikation entscheidend. Dies konnten wir schon während unserer Zusammenarbeit in den vorherigen Aufgabenteilen (Draw Projekt) feststellen. Besonders bewährten sich die häufigen Treffen zu kleinen Zwischenmeetings. So waren wir beide immer auf demselben Stand und konnten den jeweils anderen auch um Hilfe bei aufkommenden Problemen bitten.
Sehr gut unterstützt wurde unser Austausch durch das Notion Dashboard. Wir konnten hier alle anstehenden Aufgaben in Form einer Tabelle wie in Abbildung 2 festhalten. Im Projektalltag konnten sich so jeder nach Verfügbarkeit selbst Aufgaben zuweisen. Nach Fertigstellung wurden den Aufgaben der Status „fertig“ zugeordnet.
Generell konzentrierte sich Mark viel auf die View betreffenden Aufgaben und Julian auf das Modell. Die Spezialisierung half uns, den Einarbeitungsaufwand in neue Programmier-techniken/Konzepte zu begrenzen. Wichtig war uns jedoch auch, dass wir zu jedem Zeitpunkt einen guten Überblick über das gesamte Programm sowie dessen Struktur hatten.

WORKLOAD

Den tatsächlichen Workload haben wir definitiv unterschätzt. Unser Zeitplan war sehr ungenau. Jede Funktion konnte nicht einfach an einem Stück implementiert werden. Eine einzelne Anforderung unterteilte sich oft in viele kleine Aufgaben. Den angestrebten Workload von 48h haben wir deutlich überschritten. Wir kommen mit der Vorbereitung des Vortrags und der Dokumentation auf über 100 Stunden, wobei der Workload gleich verteilt ist.

PROBLEME

Es war definitiv ein Fehler, dass wir keine Zeit für das Refactoring eingeplant haben. Oft gelang es zwar, schnell ein „Feature Prototyp“ in das Programm zu integrieren, doch eine saubere Lösung benötigt viel mehr Zeit. So mussten wir oft Datenstrukturen nochmal nachträglich ändern oder Anpassungen vornehmen, um weniger Abhängigkeiten und mehr Kommunikation über Schnittstellen zu erreichen.

Da wir vor dem Praktikum noch nie mit GIT gearbeitet haben, mussten wir uns mit der Versionsverwaltung erst einmal auseinandersetzen. Es dauerte teilweise lange, bis wir das Problem mit der Versionskontrolle gelöst hatten. Dadurch hatten wir dann weniger Zeit für unser Projekt. Besonders schwer war in diesem Zusammenhang das Problem mit der eigebundenen Library von FlatLaf.

Allgemein hatten wir wenig Wissen (nur SE1 u. SE2), wie man den Entwurf und die Architektur einer Software gestaltet. Vor allem bei der GUI hatten wir keine klare Herangehensweise. Wir haben zuerst den GUI Builder von Eclipse angewendet, um uns Code generieren zu lassen. Da wir daraufhin jedoch viel manuell anpassen mussten und die Klassen viel überflüssigen Code hatten, haben wir die Methode der Code Generierung verworfen.

Die GUI des Programms weist einen Fehler auf. Bei dem Erstellen eines neuen Decks wird ein neues Fenster geöffnet. Die MainGUI ist jedoch daraufhin nicht deaktiviert, so können unendlich viele neue Fenster geöffnet werden. Dies ist nach unseren Anforderungen nicht gewollt.

In der Endphase des Projekts mussten wir uns oft zwischen der Implementation neuer Funktionen oder Clean Code (Refactoring) entscheiden. Dabei sind wir nach dem defensiven Programmierstil vorgegangen und haben dadurch weniger Features umsetzen können.

WAS LIEF GUT?

Durch das Projektmanagementtool Notion haben wir unser Projekt strukturiert organisieren können. Wir haben dadurch den Überblick, welche Anforderungen an das System schon umgesetzt sind und welche Anforderungen noch fehlen. In den „QuickNotes“ kann jeder dazu noch Probleme oder erste Lösungsansätze dokumentieren. Notion eignet sich auch, um Dokumente (z. B. UML Diagramme) zu teilen.

In unserem Programm konzentrierten wir uns vor allem auf die Robustheit konzentriert und haben das Programm sehr häufig interaktiv getestet, um mögliche Bugs zu finden. Dabei entwickelten wir einen Plan auf, wie wir systematisch das Karteikartenlernprogramm analysieren. Wenn wir Bugs entdeckt haben, notierten wir dies sofort in unserer Notiondatei und teilten die Tasks untereinander auf. Robustheit war uns dementsprechend wichtiger, als schnell neue Features zu implementieren. Dadurch wurden die Codequalität und Korrektheit des Programms signifikant gesteigert.

Unsere Learnings:

- Projektmanagement mit Notion
- Teamarbeit
- Versionsverwaltung mit GIT
- Implementation von Entwurfsmustern (MVC, Observern Pattern)
- Umgang mit Frameworks und Libraries
- strukturiertes Refactoring
- Datenstrukturen
- defensiver Programmierstil

Allgemein ist das Praktikum bzw. das Projekt eine Chance gewesen, das theoretische Wissen in der Praxis anzuwenden. Durch unsere Unerfahrenheit in der Softwareentwicklung haben wir manchmal viel Zeit gebraucht uns fehlendes Wissen anzueignen (z. B. zum Thema GIT). Für unser erstes eigenes Programmierprojekt sind wir jedoch mit dem Ergebnis zufrieden. In folgenden Projekten werden wir durch das gelernte Wissen aus dem Praktikum schneller unsere Ziele erreichen und größere Projekte umsetzen können.