Karlsruhe Institute of Technology

Institut für Angewandte Informatik und Formale Beschreibungsverfahren

Prof. Dr. D. Ratz

Philipp Stegmaier (philipp.stegmaier@kit.edu)
Helen Schneider (helen.schneider@kit.edu)
Ferdinand Mütsch (muetsch@kit.edu)

Programmierung kommerzieller Systeme - Anwendungen in Netzen mit Java

Sommersemester 2023

Abgabeblatt

(Bearbeitung im Zeitraum vom 05.06.2023 bis zum 25.06.2023)

1 Aufgabenstellung

Sie essen oft und gerne in der Mensa des KIT. Eines Eintopf-Mittwochnachmittags haben Sie beschlossen, gerne einen besseren Überblick über Ihre Mahlzeiten gewinnen zu wollen. Verfassen Sie hierzu ein **lauffähiges** Java Programm - der **"Mensa Food Tracker"** - das Sie dabei unterstützt. Es soll Ihnen erlauben, aus dem wochenaktuellen Speiseplan die von Ihnen gewählten Gerichte in Form einer Essenshistorie zu erfassen und verschiedene Kennzahlen, insbesondere Kosten und Nährwerte, aggregiert darzustellen. Das Programm soll über eine **grafische Oberfläche (GUI)** verfügen, damit auch wenig technikaffine Personen das Programm bedienen können.

Das Programm ist von jedem Studierenden **individuell** zu erstellen und hat folgende Anforderungen zu erfüllen:

1.1 Funktionale Anforderungen

- Es soll eine **Liste aller angebotenen Mensa-Gerichte** für einen vom Benutzer / der Benutzerin **auszuwählenden Tag** angezeigt werden, sofern Daten für diesen Tag vorliegen. Der **Defaultwert** beim Starten des Programmes sollte immer auf den **aktuellen Tag** gesetzt werden. In der Liste sollen mindestens **Name** und **Preis** des jeweiligen Gerichtes angezeigt werden, optional und nach Belieben auch weitere Attribute. Sie bekommen eine **externe Bibliothek** bereitgestellt, mithilfe derer Sie die benötigten Daten anfragen können. Mehr dazu in Abs. 1.5.
- Nutzer:innen sollen aus der Liste ein Gericht auswählen und in ihre **Essenshistorie** übernehmen können. Gleichermaßen sollen Gerichte wieder aus der Historie **entfernt** werden können.
- Die **Essenhistorie** ist eine Liste aller bislang erfassten Gerichte, (absteigend) **nach Datum sortiert**. Es sollen für jedes Gericht **Name**, **Datum** und **Preis**, sowie mindestens drei der vorliegenden **Nährwertangaben** wiedergegeben werden. Pro Tag können mehrere Gerichte in der Historie vorhanden sein (auch das gleiche Gericht mehrmals). Wurde für einen Tag kein Gericht erfasst, soll dieser Tag nicht in der Historie angezeigt werden.
- Weiterhin soll es eine Übersicht geben, die verschiedene **Kennzahlen** über alle in der Historie befindlichen Gerichte aufkumuliert anzeigt. Dazu gehören: gesamte **Energiezufuhr in Kilokalorien (kcal)**, die insgesamt aufgenommenen Mengen an **Protein**, **Kohlenhydraten** und **Fett**, jeweils in Gramm (g), die **Gesamtkosten in Euro (€)**, sowie der **Veggie Anteil in Prozent (%)**, d.h. das Verhältnis von vegetarischen bzw. veganen Gerichten zu allen Gerichten in der Historie. Wird ein Gericht der Historie hinzugefügt oder ein anderes entfernt, sollen die Kennzahlen **automatisch aktualisiert**, d.h. neu berechnet, werden.
- Damit die Historie auch über mehrere Programmstarts hinweg erhalten bleibt, soll der Zustand der Anwendung automatisch beim **Schließen** des Programms in eine **Datei gespeichert** und beim nächsten **Start** des Programms

wieder aus dieser **Datei eingelesen** werden. Ist bei der initialen Verwendung noch keine Datei vorhanden, soll das Programm mit einer leeren Historie starten. Details zum zu verwendenten Dateiformat finden Sie in Abs. 1.4.

- Zusammengefasst soll die Anwendung in **drei wesentliche Bereiche** unterteilt sein:
  - Der erste Bereich beinhaltet Komponenten, um einen **Tag (Datum) auswählen**, sich den zugehörigen **Speiseplan anzeigen** und einzelne Gerichte der Historie **hinzufügen** zu können.
  - Der zweite Bereich beinhaltet die **Historie**, d.h. im Wesentlichen eine nach Datum sortierte Liste aller bereits hinzugefügten Gerichte, und die Option, diese wieder **entfernen** zu können.
  - Der dritte Bereich zeigt die besagten aggregierten **Kennzahlen** grafisch ansprechend an.

1.2 Nicht-funktionale Anforderungen

- Verwenden Sie **Java 11** oder neuer.
- Bitte folgen Sie bei der Implementierung Ihrer Anwendung dem in der Vorlesung vorgestellten **MVC-Ansatz**.
- Für bessere Verständlichkeit sollen alle öffentlich einsehbaren Klassen und Methoden Ihres Programms mit nützlichen **JavaDoc Kommentaren** versehen werden.
- Die **erwarteten Fehlerfälle**, dass eine Datei nicht gelesen oder geschrieben werden kann oder dass für einen angefragten Tag keine Daten abgerufen werden können, sollen **behandelt** und der / die Nutzer:in auf das Problem (z.B. mithilfe von `JOptionPane`) **hingewiesen** werden.
- Die Benutzeroberfläche soll auch mit variierender Fenstergröße noch angemessen **bedienbar** bleiben. Insbesondere sollen (z.B. durch Scrollen) stets alle Listenelemente angezeigt werden können.

1.3 Hinweise

- Die Aufgabenstellung ist absichtlich offen formuliert. Für die Lösung der Aufgabe - insbesondere bei der Gestaltung der Oberfläche - dürfen Sie Ihrer Kreativität freien Lauf lassen. Alles, was nicht explizit spezifiziert ist, obliegt Ihrer Interpretationsfreiheit. Wichtig ist, dass die genannten Anforderungen erfüllt sind. Gerne dürfen sie aber auch darüber hinausgehen!
- Bitte verwenden Sie bei der Implementierung keine externen Bibliotheken, außer der Ihnen explizit zur Verfügung gestellten.
- Achten Sie auf einen **gut lesbaren Quellcode**, d.h. insbesondere strukturieren Sie Ihren Code übersichtlich und verwenden Sie aussagekräftige Bezeichner für die Variablen.
- Für die grafische Oberfläche dürfen Sie sämtliche Bestandteile aus Swing und AWT verwenden. GUI-Editoren wie *WindowBuilder* sind nicht grundsätzlich ausgeschlossen. Bei Problemen bei der Verwendung sind Sie jedoch auf sich alleine gestellt.
- Um vor dem Beenden der Anwendung den aktuellen Zustand als **CSV** zu speichern (sh. Abs. 1.4), können z.B. `WindowListener` verwendet werden.
- Für die Datumsauswahl eignet sich beispielsweise `JSpinner.DateEditor`.
- **Achtung:** Ein Programm, das nicht kompiliert, hat nahezu keine Chance die Abgabe zu bestehen. Achten Sie darauf, ein funktionsfähiges Programm abzugeben. Auch auskommentierter Code wird im Regelfall nicht bewertet.
- **Achtung:** Wir werden eine Software verwenden um mögliche Plagiate zu erkennen. Diese Software analysiert den Kontrollfluss Ihres kompilierten Quellcodes und lässt sich nicht durch z.B. Umbenennung oder Verschiebung von Klassen, Variablen, Methoden oder Ähnlichem austricksen. Wenn ein Plagiat vorliegt, werden wir nicht ermitteln wer von wem abgeschrieben hat. Beide Abgaben werden dann als mindestens *nicht bestanden* gewertet. ⇒ **Schreiben Sie nicht von anderen Studierenden ab und verteilen Sie auch nicht Ihre eigenen Lösungen**.

1.4 Dateiformat

Um Ihre Essenshistorie auch nach dem Schließen des Programms weiterhin zu erhalten, muss sie in einer Datei persistent gespeichert werden. Verwenden Sie **Comma-Separated Values (CSV)** als Dateiformat. Dabei handelt es sich um ein sehr einfaches, tabellarisches Format, bei dem einzelne Einträge (hier: Mensa-Gerichte) als **Zeilen** und deren Attribute (z.B. `price`) als **Spalten** abgebildet sind. Spalten sind durch ein beliebiges, aber zuvor fest definiertes Symbol - üblicherweise ein Komma (`,`) - voneinander abgtrennt. Verwenden Sie hier ein **Semikolon (`;`) als Trennzeichen**. Zudem beinhalten CSV-Datein üblicherweise einen **Header** in der ersten Zeile, der zur besseren Lesbarkeit die Namen der Spalten angibt. Ein vereinfachtes Beispiel könnte so aussehen:

```
name;price;kcal;type
Pasta in Tomatensoße mit Speck, Peperoncini und Reibekäse;3.2;943.0;PORK
Pasta in Tomaten - Broccolisoße mit Reibekäse;3.2;856.0;VEGETARIAN
```

Implementieren Sie Methoden, die es Ihnen erlauben, Objekte der Klasse `MensaMeal` in eine CSV Datei zu schreiben ("serialisieren") und wieder aus selbiger zu lesen ("deserialisieren").

Tipp: Sie können die Instanzmethode **`split()`** der Klasse `String` benutzen, um einen `String` anhand eines Trennungszeichens in mehrere `String`-Komponenten zu spalten.

1.5 Externe Bibliothek: `mensascraper-lib.jar`

Zum Abfragen des Mensa-Speiseplans stellen wir Ihnen eine externe Bibliothek als **JAR-Datei** zur Verfügung. Sie kann als .jar-Datei heruntergeladen und in Eclipse (oder anderen IDEs) eingebunden werden. Das Vorgehen hierzu haben Sie im Rechnerpraktikum P00 kennengelernt.

Die Bibliothek stellt lediglich eine öffentliche Methode zur Verfügung, die mit einer abzufragenden Mensa (hier: Mensa am Adenauerring), sowie einem Datum aufgerufen werden kann und eine Liste aller an dem jeweiligen Tag angebotenen Speisen (`MensaMeal`) zurückgibt. Machen Sie sich mithilfe der JavaDoc Dokumentation und dem gegebenen Beispiel mit der Verwendung der Bibliothek und den zur Verfügung stehenden Klassen vertraut.

- Download: `https://github.com/muety/kit-mensa-scraper/releases/latest`
- Dokumentation: `https://docs.muetsch.io/kit-mensa-scraper`
- Beispiel: `https://github.com/muety/kit-mensa-scraper#usage-example`

1.5.1 Hinweise

- Das Studierendenwerk stellt keine historischen Daten zur Verfügung, d.h. es kann immer nur der Speiseplan des **aktuellen Tages oder höher** angefragt werden. Insbesondere steht z.B. *nicht* der Speiseplan von gestern zur Verfügung. Stehen für einen Tag keine Daten zur Verfügung, wird eine leere Liste zurückgegeben. Wird ein Tag in der Vergangenheit angefragt oder schlägt das Abrufen der Daten aus anderen Gründen fehl, wird eine **ungeprüfte** `MensaScraperException` geworfen.
- Die Verwendung der Library erfordert eine aktive **Internetverbindung**. Sollte das für Sie ein ernsthaftes Hindernis darstellen, kontaktieren Sie bitte die Übungsleitung.
- Die Library verwendet die seit Java 8 verfügbare `java.time` API, insbesondere `LocalDate`. In Swing / AWT hingegen wird oftmals die ältere `java.util.Date` Klasse verwendet. Eine kurze Internetrecherche sollte die Konvertierung zwischen den Klassen erklären.

2 Abgabe der Leistung

- Die Abgabe des fertigen Programms erfolgt als **Archive-File, also als zip-Datei**, über ILIAS. Eine Anleitung zum Export von Projekten in Eclipse finden Sie auf dem Rechnerpraktikumsblatt P00.
- Die Abgabe erfolgt unter dem Übungsobjekt *Pflichtabgabe* über den Button `Datei abgeben`. Benennen Sie die Datei mit **"Nachname_Vorname_u-Kürzel_Abgabe.zip"** (z.B. "Mustermann_Max_uabcd_Abgabe.zip"). Abgaben die keiner Person zugeordnet werden können, werden mit **nicht bestanden** bewertet.
- Die Abgabefrist endet am **25.06.2023 um 23:55 Uhr**. Nicht fristgerecht erfolgte Abgaben werden mit **nicht bestanden** bewertet. Sollte es zu Problemen beim Upload kommen, so melden Sie sich bitte direkt bei Ihren Tutoren oder der Übungsleitung. Auch hierbei gilt die genannte Abgabefrist.

3 Offene Fragen?

- Für inhaltliche und organisatorische Fragen bezüglich der Abgabe dürfen Sie gerne das *Forum für Fragen zur Abgabe* nutzen.
- Für grundlegende Fragen zur Java-Thematik dürfen Sie gerne das *Forum für inhaltliche Fragen* nutzen und sich an Ihre Tutoren wenden.

Die Aufgabe ist von jedem Studierenden in Einzelarbeit zu lösen. Betrugsversuche führen für alle beteiligten Studierenden zur Bewertung `nicht bestanden` **und werden bei der Prüfungskommission gemeldet.**