Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Architecture-driven Requirements Engineering (ARE)
https://sdqweb.ipd.kit.edu/wiki/Programmieren
programmieren-vorlesung@ipd.kit.edu

Programmieren – Sommersemester 2020

Übungsblatt 2
Version 1.1

20 Punkte

Ausgabe:	11.05.2020, ca. 13:00 Uhr
Praktomat:	18.05.2020, 13:00 Uhr
Abgabefrist:	26.05.2020, 06:00 Uhr

Abgabehinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe dieses Blattes notwendig ist. Der Praktomat wird Ihre Abgabe zurückweisen, falls eine der nachfolgenden Regeln verletzt ist. Eine zurückgewiesene Abgabe wird automatisch mit null Punkten bewertet. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie keine Elemente der Java-Bibliotheken, ausgenommen Elemente der Pakete `java.lang` und `java.io`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.

Bearbeitungshinweise

Diese Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe, jedoch wird der Praktomat Ihre Abgabe **nicht** zurückweisen, falls eine der nachfolgenden Regeln verletzt ist.

- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im ILIAS-Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

- Beachten Sie, dass Überläufe und andere eingabebasierte Fehler nicht behandelt werden müssen.

Plagiat

Wichtig: Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen wie beispielsweise der Lehrveranstaltung oder dem Übungsbetrieb selbst, ist ein Täuschungsversuch und führt zur Bewertung nicht bestanden. Dies beinhaltet unter anderem auch die Lösungsvorschläge des Übungsbetriebes. Es werden nur rein selbstständig erarbeitete Lösungen akzeptiert. Für weitere Ausführungen sei auf die Einverständniserklärung (Disclaimer) verwiesen.

Checkstyle

Der Praktomat überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen der Praktomat die Abgabe zurückweist, da diese Regel verpflichtend einzuhalten ist. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki (ILIAS) beschreibt, wie das funktioniert.

Terminal-Klasse

Laden Sie für diese Aufgabe die Terminal-Klasse herunter und platzieren Sie diese unbedingt im Paket `edu.kit.informatik`. Die Methode `Terminal.readLine()` liest eine Benutzereingabe von der Konsole und ersetzt `System.in`. Die Methode `Terminal.printLine()` schreibt eine Ausgabe auf die Konsole und ersetzt `System.out`. Verwenden Sie für jegliche Konsoleneingabe oder Konsolenausgabe die Terminal-Klasse. Verwenden Sie in keinem Fall `System.in` oder `System.out`. Fehlermeldungen werden ausschließlich über die Terminal-Klasse ausgegeben und müssen aus technischen Gründen unbedingt mit `Error,␣` beginnen. Laden Sie die Terminal-Klasse **niemals** zusammen mit Ihrer Abgabe hoch.

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Montag, den 18. Mai 2020 um 13:00 Uhr**, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Praktomaten bei der richtigen Aufgabe vor Ablauf der Abgabefrist hochzuladen.

Geben Sie Ihre Lösungen zu den jeweiligen Aufgaben als `*.java`-Dateien ab.

Aufgabe A: Matrix-Operationen (4 Punkte)

In dieser Aufgabe soll eine quadratische mathematische $n \times n$ Matrix mit $n > 0$ für 32-Bit-Ganzzahlen modelliert werden. Diese soll folgende Operationen unterstützen:

- Addition (`add`)
- Matrix-Vektor-Multiplikation (`multiply`)
- Darstellung der Matrix (`show`)

Die Matrix-Vektor-Multiplikation multipliziert dabei eine $n \times n$ Matrix mit einem Spaltenvektor. Ein Spaltenvektor entspricht einer $n \times 1$ Matrix.

Erstellen Sie dafür eine Klasse `MathMatrix`, die die oben genannten Operationen implementiert. In der Matrix-Klasse soll keine Ein- oder Ausgabe stattfinden. Implementieren Sie stattdessen für die Darstellung eine `toString()`-Methode, die sie auch für der Operation `show` verwenden können. Achten Sie darauf, dass die Operation `show` eine textuelle Repräsentation zurückgibt. Die Matrix wird als durch Zeilenumbrüche separierte Zeilen dargestellt. Jede Zeile besteht aus durch jeweils ein Leerzeichen separierte 32-Bit-Ganzzahl. Die Matrix wird mithilfe eines öffentlichen Konstruktors `MathMatrix(int[][] matrix)` erstellt.

Zusätzlich erstellen Sie eine Klasse `Main`, welche eine `main()`-Methode enthält, in der die Ein- und Ausgabe stattfindet. Die Eingabe findet via Kommandozeilenargumente (Command Line Arguments) statt. Die Kommandozeilenargumente können Sie aus dem *args*-Parameter der `main()`-Methode auslesen. Als erstes Argument wird die Operation angegeben (s.o.), danach folgen eine Matrix bei `show` bzw. zwei Matrizen bei `add` und `multiply`. Eine Matrix wird dabei als durch Semikolon separierte Liste von Zeilen dargestellt. Eine Zeile besteht aus 32-Bit-Ganzzahlen, die jeweils durch Kommata separiert werden. Sie können davon ausgehen, dass alle Eingaben syntaktisch und semantisch korrekt sind. Das Programm soll daraufhin das Ergebnis der Operation ausgeben.

Beispielablauf

```
> java Main add 1,2;0,3 3,4;3,4
4 6
3 7
> java Main multiply 1,2;0,3 1;4
9
12
```

Aufgabe B: Primzahlen (2 Punkte)

In dieser Aufgabe soll die n-te Primzahl auf der Kommandozeile ausgegeben werden, wobei n eine natürliche Zahl ist. Eine Primzahl ist per Definition eine natürliche Zahl größer 1, die nur durch 1 und sich selbst teilbar ist (d.h. 2, 3, 5, 7, 11, 13, 17, 19,... sind Primzahlen). Die Zahl n wird als Kommandozeilenparameter übergeben. Sie können davon ausgehen, dass nur natürliche Zahlen eingegeben werden und die Eingabe korrekt ist.

Beispielablauf

```
> java Main prime 2
3
> java Main prime 1
2
> java Main prime 4
7
> java Main prime 6
13
```

Aufgabe C: Holoalphabetischer Satz (4 Punkte)

In dieser Aufgabe soll überprüft werden, ob ein Satz ein holoalphabetischer Satz ist, d.h. ein Satz bei dem alle Buchstaben des Alphabets [A-Za-z] mindestens einmal vorkommen. Sonderzeichen können vorkommen, spielen aber bei der Überprüfen keine Rolle.

Nach dem Start kann Ihr Programm über die Konsole mittels `Terminal.readLine()` Befehle entgegennehmen, die im Folgenden beschrieben werden. Nach Abarbeitung eines Befehls wartet das Programm auf einen weiteren Befehl bis das Programm durch `quit` beendet wird. Sie können davon ausgehen, dass alle Eingaben syntaktisch und semantisch korrekt sind.

Der `holoalphabetic?`-Befehl

Dieser Befehl überprüft einen Satz `<sentence>` ohne Zeilenumbrüche auf holoalphabetische Eigenschaften.

Eingabeformat `holoalphabetic? <sentence>`

Ausgabeformat Es ergeben sich bei der Ausgabe drei Fallunterscheidungen:

(i) Enthält der Eingabesatz `<sentence>` alle Buchstaben des Alphabets mindestens einmal, wird `pangram` ausgegeben.

(ii) Enthält der Eingabesatz `<sentence>` alle Buchstaben des Alphabets genau einmal, wird `isogram` ausgegeben.

(iii) Enthält der Eingabesatz `<sentence>` nicht alle Buchstaben des Alphabets, wird `false` ausgegeben.

Der `quit`-Befehl

Dieser Befehl ermöglicht es, das laufende Programm vollständig zu beenden.

Beachten Sie, dass hierfür keine Methoden wie `System.exit()` oder `Runtime.exit()` verwendet werden dürfen.

Eingabeformat `quit`

Ausgabeformat Hierbei findet keine Konsolenausgabe statt.

Beispielablauf

```
> holoalphabetic? abcd efgh ijklmn op qrst uvwxyz
isogram
> holoalphabetic? aBcD efgh ijklmn opqrst uvWxyz
isogram
> holoalphabetic? Aabc dDefg h ijklmMn op qrRsTtt uvwxyz
pangram
> holoalphabetic? abc
false
> quit
```

Aufgabe D: Objektorientierte Modellierung (10 Punkte)

In dieser Aufgabe soll der Luftverkehr einer Fluggesellschaft vereinfacht modelliert werden. Die folgenden Elemente sollen Sie bei Ihrer Modellierung berücksichtigen:

Ein Flug hat einen Start- und einen Zielflughafen sowie eine eindeutige Identifikationsnummer. Die Identifikationsnummer wird beim Anlegen des Fluges vergeben und entspricht der Anzahl der bereits angelegten Flüge. Somit erhält der erste Flug die Nummer "0", der zweite die Nummer "1" usw. Zudem hat jeder Flug ein Abflugdatum, das durch Tag, Monat und Jahr spezifiziert ist, und eine Flugdauer, die in Minuten angegeben ist. Außerdem hat jeder Flug genau 100 Passagiere und wird mit einem Flugzeug durchgeführt.

Ein Passagier trägt einen Namen und besitzt ein Geburtsdatum, welches durch Tag, Monat und Geburtsjahr beschrieben wird. Name und Geburtsdatum eines Passagiers dürfen aus Sicherheitsgründen nicht verändert werden.

Außerdem hat jeder Passagier eine Anschrift. Diese setzt sich zusammen aus Straßenname, Hausnummer, Postleitzahl, Stadt und Land.

Jeder Passagier hat eine Beförderungsklasse gebucht: `ECONOMY`, `ECONOMYPLUS`, `BUSINESS` oder `FIRST`.

Ein Flugzeug hat einen Markennamen, eine Seriennummer und einen Heimatflughafen.

Ein Flughafen hat einen Namen und eine Anschrift, die sich aus Straßenname, Hausnummer, Postleitzahl, Stadt und Land zusammensetzt.

Ergänzen Sie alle modellierten Klassen um Konstruktoren mit entsprechenden Parametern. Erweitern Sie alle Klassen um sinnvolle `getter()`- und `setter()`-Methoden. Berücksichtigen Sie dabei Aspekte der Datenkapselung und begründen Sie Ihre Entscheidung mit einem kurzen Kommentar.

Implementieren Sie für jede Klasse die `toString()`-Methode. Wählen Sie hierfür eine geeignete textuelle Darstellung und begründen Sie Ihre Entscheidung ebenfalls mit einem kurzen Kommentar.

Des Weiteren soll es möglich sein, Flugzeuge zu vergleichen. Flugzeuge sind genau dann gleich, wenn ihr Markenname und ihre Seriennummer übereinstimmen (der Heimatflughafen wird beim Vergleich nicht berücksichtigt). Implementieren Sie hierzu die `equals()`-Methode in der entsprechenden Klasse. Halten Sie dabei die Vorgaben der Java API[1] ein.

[1] https://docs.oracle.com/javase/11/docs/api/java/lang/Object.html#equals (java.lang.Object)