Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren/
programmieren-vorlesung@cs.kit.edu

Programmieren – Wintersemester 2024/25

Übungsblatt 2 Version 1.0	20 Punkte	Ausgabe:	13.11.2024, ca. 12:00 Uhr
		Abgabestart:	20.11.2024, 12:00 Uhr
		Abgabefrist:	28.11.2024, 06:00 Uhr

Plagiarismus

Es werden nur selbstständig angefertigte Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt jederzeit (auch nachträglich) zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden. Alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Beachten Sie darüber hinaus die Richtlinien der Fakultät zum Verwenden von Generativer KI [1].

Studierende, die den ordnungsgemäßen Ablauf einer Erfolgskontrolle stören, können von der Erbringung der Erfolgskontrolle ausgeschlossen werden. Ebenso stellt unter anderem die Weitergabe von Teilen von Testfällen oder Lösungen bereits eine Störung des ordnungsgemäßen Ablaufs dar. Auch diese Art von Störungen können ausdrücklich jederzeit zum Ausschluss der Erfolgskontrolle führen. Dies bedeutet ausdrücklich, dass auch nachträglich die Punktzahl reduziert werden kann.

Kommunikation und aktuelle Informationen

In unseren *FAQs*[2] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen. Beachten Sie zudem die Hinweise im Wiki[3].

In den *ILIAS-Foren* oder auf *Artemis* veröffentlichen wir gelegentlich wichtige Neuigkeiten. Eventuelle Korrekturen von Aufgabenstellungen werden ebenso auf diesem Weg bekannt gemacht. Das aktive Beobachten der Foren wird daher vorausgesetzt.

[1] https://www.informatik.kit.edu/faq-wiki/doku.php?id=generative_ki
[2] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ
[3] https://sdq.kastel.kit.edu/programmieren/

Überprüfen Sie das Postfach Ihrer *KIT-Mailadresse* regelmäßig auf neue E-Mails. Sie erhalten unter anderem eine Zusammenfassung der Korrektur per E-Mail an diese Adresse. Alle Anmerkungen können Sie anschließend im Online-Einreichungssystem[4] einsehen.

Bearbeitungshinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der verpflichtenden Tests für eine erfolgreiche Abgabe von Übungsblatt 2 notwendig ist. Ihre Abgabe wird automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Sie müssen zuerst die verpflichtenden Tests bestehen, bevor die anderen Tests ausgewertet werden können. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie ausschließlich *Java SE 17*.
- Sofern in einer Aufgabe nicht ausdrücklich anders angegeben, verwenden Sie keine Elemente der Java-Bibliotheken. Ausgenommen ist die Klasse `java.util.Scanner` und alle Elemente aus den folgenden Paketen: `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 140 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- `System.exit()`, `Runtime.exit()` oder ähnliches dürfen nicht verwendet werden.

Diese folgenden Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe. Dennoch wird Ihre Abgabe durch das Abgabesystem *nicht* automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Orientieren Sie sich zudem an den Bewertungskriterien im ILIAS-Wiki.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.
- 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.
- Geben Sie im Javadoc-Autoren-Tag nur Ihr u-Kürzel an.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

[4] `https://artemis.praktomat.cs.kit.edu/`

Checkstyle

Das Online-Einreichungssystem überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen das Online-Einreichungssystem die Abgabe mit null Punkten bewertet, da diese Regeln verpflichtend einzuhalten sind. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki im ILIAS beschreibt, wie Checkstyle verwendet werden kann.

Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 20.11.2024, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 28.11.2024, 06:00 Uhr, hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären. Falls Sie mit Git abgeben, *muss immer* auf den `main`-Branch gepusht werden.

- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe A in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.
- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe B in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.
- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe C in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.

Wiederverwendung von Lösungen

Falls Sie für die Bearbeitung der Abschlussaufgaben oder Übungsblätter Beispiellösungen aus diesem Semester wiederverwenden, *müssen* Sie in die entsprechenden Klassen "Programmieren-Team" ins Autor-Tag eintragen. Dies ist nötig, um die Checkstyle-Kriterien zu erfüllen.

Aufgabe A: String-Utility (6 Punkte)

Implementieren Sie die folgenden sechs öffentlichen Methoden in einer Utility-Klasse `StringUtility`. Diese Klasse darf keine `main`-Methode enthalten. Stellen Sie zudem sicher, dass alle Methoden statisch sind und die Klasse keine Instanzattribute beinhaltet.

Sie können davon ausgehen, dass die als Methodenparameter übergebenen Zeichenketten immer mindestens ein gültiges Zeichen enthalten. Beachten Sie bei diesen Zeichenketten sowohl die Groß- und Kleinschreibung als auch Sonder- und Zahlzeichen. Verwenden Sie selber für diese Aufgabe *keine* weiteren Klassen direkt aus der Java-API, mit Ausnahme von `String` und `Character`.

1. Schreiben Sie eine Methode `String removeVowels(String word)`, die alle Vokale (Groß- und Kleinschreibung) aus einer gegebenen Zeichenkette entfernt. Die Methode gibt die Zeichenkette ohne Vokale zurück.

   Beispiel: `removeVowels("Hello World")` gibt `"Hll Wrld"` zurück.

2. Schreiben Sie eine Methode `String replaceSubstring(String word, String target, String replacement)`, die alle Vorkommen einer Teilzeichenkette *target* in der gegebenen Zeichenkette *word* durch eine neue Zeichenkette *replacement* ersetzt. Die Methode gibt die geänderte Zeichenkette zurück.

   Beispiel: `replaceSubstring("Hello World", "World", "Java")` gibt `"Hello Java"` zurück.

3. Schreiben Sie eine Methode `int wordCount(String sentence)`, die die Anzahl der Wörter in einem gegebenen Satz zählt. Die Methode geht davon aus, das Wörter immer durch Leerzeichen getrennt sind, und gibt die Anzahl der Wörter als Ganzzahl zurück.

   Beispiel: `wordCount("Hello there, world!")` gibt `3` zurück.

4. Schreiben Sie eine Methode `String rotateString(String word, int positions)`, die die Zeichen in einer gegebenen Zeichenkette um eine bestimmte Anzahl von Positionen nach rechts verschiebt. Zeichen, die das Ende überschreiten, werden an den Anfang verschoben. Die Methode gibt die rotierte Zeichenkette zurück.

   Beispiel: `rotateString("abcdef", 2)` gibt `"efabcd"` zurück.

5. Schreiben Sie eine Methode `String removeNonLetters(String sentence)`, die alle Zeichen, die keine Buchstaben sind (siehe `Character::isLetter`), entfernt. Die Methode gibt die bereinigte Zeichenkette zurück.

   Beispiel: `removeNonLetters("H3llo W4rl-d")` gibt `"HlloWrld"` zurück.

6. Schreiben Sie eine Methode `boolean isRotation(String word, String rotation)`, die überprüft, ob die Zeichenkette *rotation* eine Drehung der Zeichenkette *word* ist. Die Methode gibt *true* zurück, wenn dies der Fall ist, andernfalls *false*.

   Beispiel: `isRotation("waterbottle", "erbottlewat")` gibt `true` zurück.

Aufgabe B: Minkowski-Distanz (8 Punkte)

Die Minkowski-Distanz[5] ist eine allgemeine Metrik zur Berechnung der Distanz zwischen zwei Punkten in einem n-dimensionalen Raum. Sie ist benannt nach dem Mathematiker Hermann Minkowski[6]. Die Minkowski-Distanz kann als Verallgemeinerung der euklidischen Distanz und der Manhattan-Distanz betrachtet werden, da diese beiden Metriken spezielle Fälle der Minkowski-Distanz darstellen.

Die Formel für die Minkowski-Distanz lautet:

$$D_{\text{Minkowski}}(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Hierbei ist $\mathbf{x}$ der erste Vektor, $\mathbf{y}$ der zweite Vektor und p ein positiver Parameter, der den Grad der Distanz angibt. Wenn $p = 1$, entspricht die Minkowski-Distanz der Manhattan-Distanz und wenn $p = 2$, entspricht sie der euklidischen Distanz.

Implementieren Sie eine Klasse für das Minkowski-Distanzmaß, welche erlaubt, die Minkowski-Distanz für zwei Vektoren vom Typ `int[]` und einem bestimmten Distanz-Grad vom Typ `int` zu berechnen. Falls der Distanz-Grad echt kleiner 1 ist, soll der Wert 1 verwendet werden.

Fügen Sie zudem der Klasse zwei Methoden hinzu, welche die euklidische Distanz und die Manhattan-Distanz berechnen. Für diese Methoden werden jeweils nur zwei Vektoren übergeben vom Typ `int[]` als Parameter übergeben.

Implementieren Sie weiterhin eine Klasse, um den Nearest-Neighbor-Pfad zu berechnen. Dieser Pfad berechnet gegeben einer Menge von Vektoren (ausgehend vom ersten Vektor) den Pfad über die jeweils nächstgelegenen Nachbarn. Jeder Vektor ist nur einmal im Pfad vorhanden. Dazu beginnt man mit dem ersten Vektor und wählt dann in jedem Schritt denjenigen Vektor aus, der am nächsten zum aktuellen Vektor liegt und noch nicht im Pfad enthalten ist. Gibt es mehrere Vektoren mit gleicher Distanz zum aktuellen Vektor, wird der erste Vektor nach der ursprünglichen Sortierung der Vektoren ausgewählt. Dieser wird dann zum Pfad hinzugefügt und als neuer, aktueller Vektor verwendet. Dieser Schritt wird so lange wiederholt, bis alle Vektoren im Pfad enthalten sind.

Als Distanzmaß soll hier die Minkowski-Distanz genutzt werden. Deshalb lässt sich auch die Pfadberechnung durch das Grad p parametrisieren.

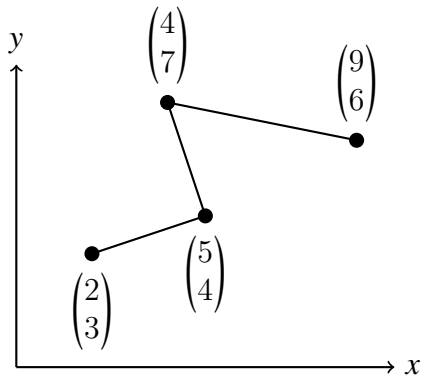
B.1 Beispiel

Gegeben die Vektoren (v_1 dient als Startvektor):

$$v_1 = \begin{pmatrix} 2 \\ 3 \end{pmatrix} v_2 = \begin{pmatrix} 5 \\ 4 \end{pmatrix} v_3 = \begin{pmatrix} 9 \\ 6 \end{pmatrix} v_4 = \begin{pmatrix} 4 \\ 7 \end{pmatrix}$$

Dann ist der Nearest-Neighbor-Pfad für die euklidische Distanz:

[5] `https://en.wikipedia.org/wiki/Minkowski_distance`
[6] `https://de.wikipedia.org/wiki/Hermann_Minkowski`



B.2 Schnittstelle

Erstellen Sie zudem eine Hauptklasse, in welcher sich eine Main-Methode befindet. In dieser werden die Kommandozeilenargumente (`String[] args`) and das Programm übergeben. Das erste Argument ist der Grad für die Minkowski-Distanz und damit immer eine positive Ganzzahl in Textdarstellung. Alle weiteren Argumente sind Vektoren in Textdarstellung. Hierbei werden nur Ganzzahlen verwendet und die Zahlen jeweils mit Kommas getrennt (z.B. `"2,3"`). Sie können davon ausgehen, dass alle Vektoren zweidimensional sind. Sie können davon ausgehen, dass `args` nicht null und immer mindestens drei Argumente enthält (der Grad und min. zwei Vektoren) und kein Vektor doppelt auftritt. Sie können zudem davon ausgehen, dass die textuelle Darstellung der Argumente korrekt ist und müssen dies nicht überprüfen. Ihr Programm soll dann für die Vektoren den Nearest-Neighbor-Pfad mit der Minkowski Distanz mit dem übergebenen Grad berechnen. Die Ausgabe des Pfades soll als zweidimensionales Feld erfolgen, wobei leere Felder mit `-`, der erste Vektor mit `S` und der letzte Vektor mit `E` ausgegeben werden. Die anderen Vektoren werden als Zahlen von `1` bis `n-1` dargestellt. Das Feld soll die minimal nötige Größe haben, um alle Vektoren darzustellen.

B.3 Beispielinteraktion

Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Der Programmaufruf wird mit `%>` gefolgt von einem Leerzeichen eingeleitet, diese Zeichen sind ebenfalls kein Bestandteil des eingegebenen Befehls, sondern dienen der Unterscheidung zwischen Ein- und Ausgabezeilen. Ebenso ist der Programmname `PathFinder` nur beispielhaft und nicht vorgeschrieben.

Beispielinteraktion

```
1 %> java PathFinder 2 2,3 5,4 9,6 4,7
2 ---2-----
3 --------E
4 ---------
5 ----1----
6 -S-------
7 ---------
8 ---------
```

Aufgabe C: Modellierung einer Lichtsignalanlage (6 Punkte)

In dieser Aufgabe sollen Sie eine sehr stark vereinfachte Lichtsignalanlage modellieren.

- Eine *Ampel* hat immer genau zwei *Licht*er, ein rotes und ein grünes.
- Ein Licht ist entweder an oder aus.
- Der initiale Zustand einer Ampel ist: Rotes Licht an, grünes Licht aus.
- Jede *Straßeneinmündung* hat eine Lichtsignalanlage, bestehend aus $n \geqslant 2$ Ampeln, die über Zahlen 1 bis n identifiziert werden.

Fügen Sie der Klasse für das Licht einen Konstruktor mit einem Parameter hinzu, der den initialen Zustand (an oder aus) auf den übergebenen Wert setzt. Fügen Sie weiter eine Methode hinzu, die „den Lichtschalter drückt“, d. h. das Licht ausschaltet, falls es an ist, bzw. es anschaltet, falls es aus ist. Stellen Sie ebenso eine Methode zur Verfügung, die zurückgibt, ob das Licht momentan an ist.

Fügen Sie der Klasse für die Ampel einen Konstruktor hinzu, der den initialen Zustand der Ampel wie vorgegeben festlegt. Implementieren Sie eine Methode, um die Ampel von rot auf grün bzw. umgekehrt umzuschalten. Des Weiteren fügen Sie der Klasse eine Methode hinzu, die überprüft, ob die Ampel im Moment grün zeigt.

Die Klasse für die Straßeneinmündung erhält einen Konstruktor, der die Lichtsignalanlage erstellt und initialisiert. Implementieren Sie diesen Konstruktor so, dass er den oben genannten Anforderungen genügt. Zusätzlich sollen in diesem Konstruktor für den initialen Zustand der Straßeneinmündung die erste Ampel von Rot auf Grün umgeschalteten werden.

Fügen Sie der Klasse eine Methode hinzu, welche die Ampeln durchschaltet. D. h. die aktuell grüne Ampel wird rot und die nächste in der Reihenfolge wird grün. Wenn die letzte Ampel grün ist und erneut geschaltet wird, wird diese rot und die erste wieder grün. Achten Sie darauf, dass bei mehrmaligem Aufruf dieser Methode auch alle Ampeln einmal grün werden. Als letztes entwerfen Sie eine Methode, welche prüft, ob es sicher (grüne Ampel) ist, die Ampel (über ihre Nummer identifiziert) zu überqueren. Hinweis: Diese Methode soll einen Wahrheitswert zurückgeben (`boolean`).

Greifen Sie innerhalb einer Klasse ausschließlich auf deren **eigene** Attribute zu, jedoch niemals auf Attribute von Objekten anderer Klassen. Nutzen Sie stattdessen die Methoden, um den Zustand „fremder“ Objekte zu ändern.

Zum Testen ihrer Funktionalität liefern wir ihnen eine Main-Methode, die sie nicht verändern dürfen.

Main.java

```java
public class Main {
    public static void main(String[] args)
    {
        Crossing crossing = new Crossing(3);
        System.out.println(crossing.isAllowedToCross(1));
        System.out.println(crossing.isAllowedToCross(2));
        System.out.println(crossing.isAllowedToCross(3));
        crossing.toggleTrafficLights();
        System.out.println(crossing.isAllowedToCross(1));
        System.out.println(crossing.isAllowedToCross(2));
        System.out.println(crossing.isAllowedToCross(3));
        crossing.toggleTrafficLights();
        System.out.println(crossing.isAllowedToCross(1));
        System.out.println(crossing.isAllowedToCross(2));
        System.out.println(crossing.isAllowedToCross(3));
        crossing.toggleTrafficLights();
        System.out.println(crossing.isAllowedToCross(1));
        System.out.println(crossing.isAllowedToCross(2));
        System.out.println(crossing.isAllowedToCross(3));
        crossing.toggleTrafficLights();
        System.out.println(crossing.isAllowedToCross(1));
        System.out.println(crossing.isAllowedToCross(2));
        System.out.println(crossing.isAllowedToCross(3));
    }
}
```

Hinweis: Um Code-Duplikation zu Vermeiden, könnte man den Test-Code auch mit Schleifen implementieren.