Projektblatt P8 (30 Punkte)

Abgabe: Montag 14. November 2021, 10:00h
Entpacken Sie zunächst die Archiv-Datei `vorgaben-p8.zip`, in der sich die Rahmendateien für die zu lösenden Aufgaben befinden. Ergänzen Sie die Dateien durch Ihre Lösungen gemäß der Aufgabenstellung unten. Der hinzuzufügende Java-Sourcecode sollte **syntaktisch richtig** und **vollständig formatiert** sein. Alle Dateien sollten am Ende fehlerfrei übersetzt werden können.
Verpacken Sie die `.java` Dateien für Ihre Abgabe in einem ZIP-Archiv mit dem Namen `IhrNachname.IhrVorname.P8.zip`, welches Sie auf Ilias hochladen.
Führen Sie dazu in dem Verzeichnis, in dem Sie die Dateien bearbeitet haben, folgenden Befehl auf der Kommandozeile aus:

```
zip  IhrNachname.IhrVorname.P8.zip *.java
```

Aufgabe 1: Abstrakte Klassen, Interfaces 13 (3+5+5) P

In dieser Aufgabe sollen Sie zunächst eine Klasse entwickeln, die von einer abstrakten Klasse abgeleitet ist und die eine Zahlenfolge sortiert. Die Zahlenfolge ist hierbei durch ein Objekt einer Klasse gegeben, welche ein Interface implementiert, das die Zugriffsmöglichkeiten auf eine Zahlenfolge vorgibt.
Anschließend sollen Sie zwei Klassen entwickeln, die eine Zahlenfolge jeweils unterschiedlich speichern und die beide dieses Interface implementieren.
Hierzu sind das folgende Interface und die folgenden Klassen (teilweise) vorgegeben:

Sequenz Dieses Interface deklariert vier Methoden, die wie folgt für eine endliche Zahlenfolge (Sequenz) $(a_i)_{i=1,\ldots,n}$ von positiven Zahlen (also $\forall i$: $a_i > 0$) genutzt werden sollen:

- Die Methode `size` soll die Anzahl der Elemente der Sequenz zurückgeben.
- Die Methode `get` soll den i-ten Wert in einer Sequenz zurückgeben. Wenn dieser nicht existiert, soll der Wertt -1 zurückgegeben werden.
- Die Methode `vergleiche` soll zwei Werte a_i und a_j der Sequenz vergleichen, deren Index der Methode als Parameter übergeben werden. Ist $a_i > a_j$, soll die Methode einen positiver Wert zurückgeben. Ist $a_i < a_j$, soll ein negativer Wert zurückgegeben werden und ansonsten der Wert 0. Wenn einer der Werte a_i und a_j nicht definiert ist, soll der Wert 0 zurückgegeben werden.
- Die Methode `tausche` soll zwei Werte a_i und a_j in der Sequenz vertauschen. Wenn einer der Werte a_i und a_j nicht definiert ist, soll nichts gemacht werden.

AbstrakterSortierer Diese abstrakte Klasse definiert eine Methode `ausgeben`, mit der der Inhalt einer Zahlenfolge ausgegeben werden kann. Die Ausgabe beruht dabei nur auf Methoden der Klasse `Object` und den Methoden, die im Interface `Sequenz` definiert sind. Die Klasse besitzt darüberhinaus eine abstrakte Methode mit dem Namen `sortieren`.

IndizierteSequenz Diese Klasse repräsentiert eine Zahlenfolge, die ihre Elemente in einem Feld speichert. Mit dem vorgegebenen Konstruktor

kann eine Sequenz der Länge n angelegt werden, die aus Zufallszahlen aus dem Intervall $[1, 100]$ besteht.

VerketteteSequenz Diese Klasse repräsentiert eine Zahlenfolge, deren Elemente in einer verketteten Liste gespeichert werden. Hierzu ist in der Klasse Folgendes vorgegeben:

- Die innere Klasse `Element` repräsentiert ein Listenelement, welches einen Wert vom Typ `int` besitzt sowie eine Referenz auf das nächste Listenelement.
- Der Default-Konstruktor legt eine leere Liste an, deren Kopf durch das Attribut `head` und deren Ende durch das Attribut `z` repräsentiert werden.
- Der zweite Konstruktor legt eine leere Liste an und fügt darin n Zufallszahlen aus dem Intervall $[1, 100]$ ein. Der Wert von n bildet dabei den einzigen Parameter des Konstruktors.
- Die Methode `size` bestimmt die Anzahl der (echten) Elemente in der Liste und gibt diese als Ergebnis zurück.
- Die Methode `getElementAt` gibt eine Referenz auf das i-te Listenelement zurück, sofern die Liste mindestens i Elemente hat. anderenfalls gibt die Methode eine Nullreferenz zurück.

TestSequenzen Die Klasse dient dazu, zwei Zufallssequenzen der Länge N zu generieren, welche verschiedenen Klassen angehören und welche dann ausgegeben, sortiert und nochmals ausgegeben werden sollen. Diese Klasse kann erst nach erfolgreicher Implementierung der Aufgabenteile(a) - (c) erfolgreich compiliert werden.

(a) Legen Sie eine neue Datei mit dem Namen `Sortierer.java` an und leiten Sie darin eine neue, konkrete Klasse `Sortierer` von der abstrakten Klasse `AbstrakterSortierer` ab. Implementieren Sie in dieser Klasse die Methode `sortieren`, welche die als Parameter übergebene Sequenz sortieren soll. Verwenden Sie hierzu folgenden Sortieralgorithmus:

Durchlaufen Sie Zahlensequenz $n-1$ Mal (wobei n die Länge der Sequenz ist).
Betrachten Sie in jedem Durchlauf i nur die ersten $n-i$ Elemente. Vergleichen Sie dabei jeweils je zwei aufeinanderfolgende Elemente. Falls

das erste größer ist als das zweite, tauschen sie die beiden Werte innerhalb der Sequenz aus.

Diese Klasse sollte nun erfolgreich compiliert werden können.

(b) Ergänzen Sie die Klasse `IndizierteSequenz` so, dass diese das Interface `Sequenz` implementiert. Achten Sie bei Ihrer Implementierung darauf, dass sie die oben beschriebene Funktionalität umsetzen.

(c) Ergänzen Sie die Klasse `VerketteteSequenz` so, dass diese das Interface `Sequenz` implementiert. Achten Sie bei Ihrer Implementierung darauf, dass sie die oben beschriebene Funktionalität umsetzen.
Hinweis: Nutzen Sie hierzu die in der Klasse `VerketteteSequenz` vorgegebenen Methoden.

Wenn Sie Ihre Implementierung nun mit Hilfe der Klasse `TestSequenzen` testen, sollte die Ausgabe wie folgt aussehen:

```
Original: 3 55 56 92 46 99 13 70 69 1 72 1 12 42 71 2 25 56 46 41
Sortiert: 1 1 2 3 12 13 25 41 42 46 46 55 56 56 69 70 71 72 92 99
Original: 72 44 47 43 7 55 68 40 28 64 22 8 84 19 37 13 36 46 86 28
Sortiert: 7 8 13 19 22 28 28 36 37 40 43 44 46 47 55 64 68 72 84 86
```

Aufgabe 2: Interfaces, Strings 17 (6.5 + 8.5+1+1) P

In dieser Aufgabe sollen Sie zwei einfache Algorithmen implementieren, mit denen sich Textnachrichten verschlüsseln bzw. entschlüsseln lassen. Hierzu ist Folgendes vorgegeben:

- Das Interface `Codec` spezifiziert zwei Methoden, mit denen ein der Methode als `String` übergebener Parameter ver- bzw. entschlüsselt werden kann.
- Die abstrakte Klasse `Cipher` repräsentiert eine Basisklasse für verschiedene (einfache) Kryptoalgorithmen, die entweder ohne einen expliziten Schlüssel oder mit einem einfachen ganzzahligen Schlüssel arbeiten. Dieser Schlüssel kann ggf. über einen der beiden vorgegebenen Konstruktoren gesetzt werden.

- Die Klasse `TestCodec` dient zum Testen Ihrer Implementierung. In der `main`-Methode wird ein Feld mit $N = 10$ Objekten gefüllt, die jeweils einer Klasse angehören, die das `Codec` Interface implementieren und einer Unterklasse der abstrakten Klasse `Chiffre` angehören, die Sie in den Aufgabenteilen (a) und (b) entwickeln sollen. Anschließend wird der vorgegebene `String text1` mit allen Chiffre-Objekten in dem Feld verschlüsselt und ausgegeben.

  Diese Klasse compiliert erst, wenn Sie die Teilaufgaben (a) und (b) korrekt gelöst haben.

(a) Das *Atbasch*-Verfahren (siehe auch `https://de.wikipedia.org/wiki/Atbasch`) verschlüsselt einen Text, indem jeder Buchstabe in dem Text durch den "gespiegelten" Buchstaben ersetzt wird. Das bedeutet, dass der erste durch den letzten Buchstaben des Alphabets ersetzt wird (hier 'A' durch 'Z' bzw. 'a' durch 'z'), der zweite Buchstabe durch den zweitletzten (hier 'B' durch 'Y' bzw. 'b' durch 'y') usw.. Verschlüsselung und Entschlüsselung sind dabei identisch.

Entwickeln Sie eine neue Klasse `AtbaschCipher`, die von der Klasse `Cipher` abgeleitet ist und das Interface `Codec` implementiert. Gehen Sie dabei wie folgt vor:

- Schreiben Sie eine private Klassenmethode, die einen einzelnen Buchstaben wie oben beschrieben auf den jeweiligen "gespiegelten Buchstaben" abbildet (und diesen als Ergebnis zurückgibt).
  Hinweis: Sie können den Wert berechnen, indem Sie den Abstand eines Zeichens zu den Buchstaben 'a' und 'z' bzw. 'A' und 'Z' nutzen. Alle Zeichen, die keinen Buchstaben bilden, sollen von der Methode unverändert übernommen werden. Um zu überprüfen, ob ein Zeichen ein Buchstabe ist, verwenden Sie eine passende Klassenmethode aus der Klasse `Character`.
- Schreiben Sie eine private Klassenmethode, die eine als Parameter übergebenen `String` mit Hilfe der zuvor implementierten Methode zeichenweise codiert. Die codierten Zeichen sollen in einer `StringBuffer` gesammelt werden, dessen Inhalt am Ende als `String` zurückgegeben wird.
- Implementieren Sie die Methoden des Interfaces `Codec` unter Zuhilfenahme der zuvor entwickelten Methode.

(b) Die sogenannte Skytale Chiffre (siehe auch `https://de.wikipedia.org/wiki/Skytale`) geht auf eine alte griechische Verschlüsselungsmethode zurück und kann implementiert werden, indem man einen Text Zeichen für Zeichen zeilenweise in eine Matrix schreibt und diese dann spaltenweise ausliest. Die Anzahl der Zeilen der Matrix stellt dabei den Schlüssel bei der Verschlüsselung dar.

Entwickeln Sie eine neue Klasse `SkytaleCipher`, die von der Klasse `Cipher` abgeleitet ist und das Interface `Codec` implementiert. Gehen Sie dabei wie folgt vor:

- Definieren Sie einen Konstruktor mit einem `int`-Parameter und sorgen Sie dafür, dass damit die Instanzvariable `key` aus der Oberklasse initialisiert wird. Anmerkung: Dieser Schlüssel bildet später die Anzahl der Zeilen (bzw. Spalten) der Matrix bei der Verschlüsselung (bzw. Entschlüsselung).
- Bis auf die Anzahl der Zeilen und Spalten der für die Verschlüsselung benötigten Matrix sind Ver- und Entschlüsselung identisch.

  Schreiben Sie daher nun eine Methode, die einen als Parameter übergebenen Text ver- oder entschlüsselt und das Ergebnis als `String` zurückgibt. Definieren Sie einen zweiten Parameter vom Typ `boolean` für die Methode, der festlegt, ob der Text ver- oder entschlüsselt werden soll. Implementieren Sie die Methode dann wie im Folgenden beschrieben:

  (1) Legen Sie zunächst zwei Hilfsvariablen `M` und `N` für die Anzahl der Zeilen und der Spalten in der Matrix an und definieren Sie dann eine Matrix der Größe $M \times N$ mit dem Komponententyp `char`. Im Falle einer Verschlüsselung (bzw. Entschlüsselung) entspricht die Anzahl `M` der Zeilen (bzw. die Anzahl `N` der Spalten) dem Schlüsselwert `key`. Die Anzahl der Spalten `N` (bzw. der Zeilen `M`) ergibt sich dann, indem Sie die Länge des Texts ganzahlig durch `M` (bzw. `N`) teilen und den resultierenden Wert ggf. noch um 1 erhöhen, falls die die Division einen Rest ungleich 0 hat.
  Beispiel: Bei einem Schlüsselwert $key = 5$ und einer Textlänge von 48 bzw. 50 bzw. 55 Zeichen müßte die Matrix beim Verschlüsseln 5 Zeilen und 10 bzw. 10 bzw 11 Spalten haben.

Die Matrix beim Entschlüsseln müßte dann 5 Spalten und 10 bzw. 10. bzw. 11 Zeilen haben.

(2) Schreiben Sie die Zeichen aus dem als Parameter übergebenen Text zeilenweise (und in jeder Zeile spaltenweise) in die Matrix. Holen Sie dabei die Zeichen einzeln von links nach rechts aus dem Text. Achten Sie dabei darauf, die Länge des Texts nicht zu überschreiten. Füllen Sie die Matrix ggf. am Ende mit Leerzeichen auf.

(3) Lesen Sie die Matrix spaltenweise (und in jeder Spalte zeilenweise) aus der Matrix aus und erstellen Sie den Ergebnistext, indem Sie die Zeichen in einer `StringBuffer` Instanz sammeln.

(4) Geben Sie den in der `StringBuffer` Instanz gespeicherten String zurück, nachdem Sie mit Hilfe der Instanzmethode `trim` aus der Klasse `String` die Leerzeichen am Ende entfernt haben.

- Implementieren Sie die Methoden des Interfaces `Codec` unter Zuhilfenahme der zuvor entwickelten Methode (indem Sie diese mit den passenden Parametern aufrufen).

Beispiel: Für $key = 5$ und den Text "Programmieren lernt man nur durch Programmieren!" sollte die Matrix (der Größe 5×10 zum Verschlüsseln wie folgt aussehen:

P	r	o	g	r	a	m	m	i	e
r	e	n		l	e	r	n	t	
m	a	n		n	u	r		d	u
r	c	h		P	r	o	g	r	a
m	m	i	e	r	e	n	!		

Dementsprechend ist der verschlüsselte Text dann

"Prmrmreacmonnhig erlnPraeuremrronmn g!itdr e ua"

Wenn Sie jetzt die Klasse `TestCodec` übersetzen und ausführen, sollte Folgendes ausgegeben werden:

```
1. --> Prhrn otPg rrmoaagmnrm ainmeumrrie endr uelrnec!
2. --> Pnnor ugolrrge arrdmanummtrim ceimhrea ernPne r!
3. --> Prmrmreacmonnhig   erlnPraeuremrronmn g!itdr e ua
4. --> Pirnhmrenu mortrPige  rernmdora augemlnrrnme ca!
5. --> Pmladririenuoeoer rgrgrnncreretuhanan r m!m m Pm
6. --> Pmntnroirm  ucgeoilmrhrrgeea  aerrrndPmnaen urm!
7. --> Pmntnroi rm  ucge oilmrhrr geea  ae rrrndPmn aen urm!
8. --> Paremurrmermerarcomnomnnn hgi!gi t d re rel nuPar
9. --> Paremurrme rmerarcomn omnnn hgi! gi t d re  rel nuPar
10. --> Kiltiznnrvivm ovimg nzm mfi wfixs Kiltiznnrvivm!
```

(c) In der Klasse `TestCodec` ist der `String text2` mit einem verschlüsselten Text vorgegeben. Dieser wurde mit einemr Skytale Chiffre generiert, wobei der Schlüssel in dem Intervall $[5, 25]$ liegt. Finden Sie durch Ausprobieren aller möglichen Schlüssel heraus, welcher der Richtige ist (beim Entschlüsseln müsste ein "sinnvoller" Text herauskommen).

Ergänzen Sie die `main`-Methode der Klasse dann so, dass Sie ein entsprechendes Chiffre-Objekt erzeugen, den Text damit entschlüsseln und das Ergebnis in der Konsole ausgeben. Kommentieren Sie den Code, den Sie zum Ausprobieren der verschiedenen Schlüssel verwendet haben, aus.

(d) Ergänzen Sie die Datei `TestCodec` durch einen Blockkommentar, in dem Sie die folgenden Frage beantworten und Ihre Antwort kurz begründen.

Was müßte man tun, damit man den Komponententyp des Felds `codecs` in der `main`-Methode der Klasse `TestCodec` auch auf `Cipher` ändern könnte, so dass die Klasse weiter erfolgreich compiliert werden kann.