Aufgabenblatt 11

Bearbeitungsende: 08.01.2023

Drei der bewerteten Aufgaben gelten als Bonusaufgaben: die erreichbaren Punkte werden zwar angerechnet, aber nicht in der Menge der erreichbaren Punkte mitgezählt.

Hinweis: Dies ist das letzte Blatt, dessen Bewertung maßgeblich für die Zulassung ist. Sie konnten/-können insgesamt 22 (+ 4 Zusatzpunkte) erwerben, benötigen also **mindestens 11 Punkte** für die **Zulassung**.

Auf Blatt 12 können weitere Bonuspunkte erlangt werden, allerdings bei verkürzter Bearbeitungszeit.

Aufgabe 1 [Programmierung]

Schreiben Sie eine Klasse `Feld`, die folgende Klassenmethoden bereitstellt:

(a) `istHomogen(int[] f)` gibt als Wahrheitswert zurück, ob alle in dem Feld vorkommenden Werte gleich sind. Dies ist auch der Fall, wenn das Feld leer ist.

Z.B. ist das Feld `{4, 4, 4, 4}` homogen.

(b) `istSortiert(int[] f)` gibt als Wahrheitswert zurück, ob die in dem Feld vorkommenden Werte aufsteigend sortiert sind. Das ist genau dann der Fall, wenn der Nachfolger jedes Elementes, sofern vorhanden, gleich groß ist wie oder größer ist als das Element. Dies ist auch der Fall, wenn das Feld leer ist.

Z.B. ist das Feld `{-2, 3, 3, 5}` (aufsteigend) sortiert.

(c) `maxPos(int[] f)` gibt die Position zurück, an der sich das größte Element in dem übergebenen Feld `f` befindet. Haben mehrere Elemente den maximalen Wert, so soll die Position des ersten Vorkommens zurückgegeben werden.

Z.B. ergibt der Aufruf für das Feld `f = {3, 1, -8, 4, -1 ,4}` den Wert `3`.

(d) `maxElement(int[] f)` gibt das größte Element des übergebenen Feldes `f` zurück.

(e) `max(int[] f1, int[] f2)` gibt ein neues Feld zurück, das an jeder Position den größeren der beiden Einträge an der entsprechenden Position in `f1` und `f2` enthält. Alle beteiligten Felder müssen dieselbe Länge haben.

Z.B. ergibt der Aufruf für `f1 = {3, 1, 8}` und `f2 = {2, 4, -3}` ein Feld `{3, 4, 8}`.

Die Methoden sollen für alle nicht behandelbaren Argumente eine `IllegalArgumentException` werfen. Für `null` als Argument soll jedoch eine `NullPointerException` geworfen werden.

Hinweis: Sie können die Methode jeweils per Schleife oder Rekursion (in separater Hilfsmethode, mit Startposition als Zusatzargument) implementieren. Zur Übung wird empfohlen, beides auszuprobieren.

Aufgabe 2 [Programmierung]

Schreiben Sie eine Klasse `Feld`, die folgende Klassenmethoden bereitstellt:

(a) `umkehren(int[] f)` nimmt ein `int`-Feld an und gibt ein neues `int`-Feld zurück, das die Einträge des übergebenen Feldes in umgekehrter Reihenfolge enthält. Z.B. ergibt sich für das Feld `{3, 1, 8, 5}` das Ergebnis `{5, 8, 1, 3}`.

(b) `umkehrenInPlace(int[] f)` nimmt ein `int`-Feld an und kehrt die Reihenfolge seiner Einträge innerhalb des Feldes um, ohne dass ein neues Feld angelegt wird. Es wird nichts zurückgegeben.

Hinweis: Sie können die Methode jeweils per Schleife oder Rekursion (in separater Hilfsmethode, mit Startposition als Zusatzargument) implementieren. Zur Übung wird empfohlen, beides auszuprobieren.

Aufgabe 3 [Programmierung]

Schreiben Sie eine Klasse `DynFeldVonInt`, die ein *dynamisches „Feld“* von ganzen Zahlen modelliert, welches keine feste Länge hat, sondern wachsen und schrumpfen kann. Da Java-Arrays eine feste Länge haben, muss man diese Funktionalität selbst implementieren. Intern wird dazu ein „normales“ Array in einer Instanzvariable verwaltet und bei Bedarf mit einer anderen Länge neu angelegt. Die gespeicherten Werte müssen dann in das neue Array umkopiert werden.

Die Klasse bietet folgende Funktionalitäten:

- Der einzige Konstruktor erwartet eine positive ganze Zahl und legt ein Feld dieser Länge an.
- `length()` gibt die Länge des Feldes zurück.
- `get(int i)` liefert den Wert an der Position `i` des Feldes.
- `set(int i, int v)` setzt das Feldelement an der Position `i` auf den Wert `v`.
- `insert(int i, int k)` fügt `k` (muss positiv) neue, mit `0` initialisierte Elemente vor der Position `i` in dem Feld ein. Dazu wird intern ein neues, um `k` Elemente längeres Array angelegt, die Werte in dem bestehenden Array werden an die entprechenden Positionen des neuen Arrays kopiert, und das neue Array ersetzt das alte.
- `remove(int i, int k)` löscht `k` (muss positiv sein) Elemente ab der Position `i` des Feldes. (Dies setzt voraus, dass es ab dieser Position mindestens `k` Elemente in dem Feld gibt.) Dazu wird intern ein neues, um `k` Elemente kürzeres Array angelegt, die Werte in dem bestehenden Array – außer denen an den zu löschenden Positionen – werden an die entprechenden Positionen des neuen Arrays kopiert, und das neue Array ersetzt das alte.
- `equals` nimmt ein weiteres dynamisches Feld an und gibt zurück, ob beide Felder die gleichen Werte enthalten.
- `toString` gibt die Zeichenkette "`[`$wert_0$`,`$wert_1$`,...,`$wert_{n-1}$`]`" zurück.

Für ungültige Positionswerte soll eine `IndexOutOfBoundsException` geworfen werden, für andere ungültige Argumentwerte eine `IllegalArgumentException`.

Beispiel:

```
DynFeldVonInt feld = new DynFeldVonInt(3);
feld.set(0, 3);
feld.set(1, 7);
feld.set(2, -1);
System.out.println(feld);   // [3,7,-1]
feld.insert(1, 2);
System.out.println(feld);   // [3,0,0,7,-1]
feld.set(2, 5);
feld.set(1, -3);
System.out.println(feld);   // [3,-3,5,7,-1]
feld.remove(2, 3);
System.out.println(feld);   // [3,-3]
feld.remove(1, 2);          // IllegalArgumentException
```

Aufgabe 4 [Programmierung]

Eine *Matrix* ist eine Tabelle von Zahlen. Dafür lassen sich bestimmte Rechenoperationen definieren (siehe folgende Aufgabe). Matrizen (Plural von Matrix) sind ein wichtiges Mittel, um Zusammenhänge zwischen Daten (aus Naturwissenschaft, Technik, Wirtschaft, Gesellschaft, ...) in Software darzustellen und Fragen dazu rechnerisch zu beantworten. Näheres dazu wird Ihnen in Mathematikveranstaltungen erklärt.

Schreiben Sie eine Klasse `Matrix`, die als (private) Instanzvariable ein zweidimensionales Feld von Gleitkommazahlen enthält. Die Klasse stellt folgende Konstruktoren bereit:

- Bei Angabe zweier positiver ganzer Zahlen `z` und `s` wird eine Matrix von `z` Zeilen und `s` Spalten angelegt, deren Einträge den Wert 0.0 haben.
- Bei Angabe einer positiven ganzen Zahl `n` wird eine quadratische Matrix von `n` Zeilen und `n` Spalten angelegt, deren Einträge den Wert 0.0 haben.
- Bei Angabe einer `Matrix`-Referenz wird eine vollständige Kopie dieser Matrix erzeugt.

Nachdem eine Matrix erzeugt wurde, ändert sich ihre Größe nicht mehr.

Die Klasse stellt folgende Instanzmethoden bereit:

- `zeilen()` liefert die Anzahl der Zeilen.
- `spalten()` liefert die Anzahl der Spalten.
- `get(int z, int s)` liefert den Eintrag in Zeile `z` und Spalte `s`.
- `set(int z, int s, double e)` setzt den Eintrag in Zeile `z` und Spalte `s` auf den Wert `e` und liefert die Matrix als Resultat.
- `setAll(double e)` setzt alle Einträge der Matrix auf den Wert `e` und liefert die Matrix als Resultat.
- `setZeile(int z, double[] f)` setzt die Einträge in Zeile `z` auf die Werte in `f` und liefert die Matrix als Resultat.
- `setSpalte(int s, double[] f)` setzt die Einträge in Spalte `s` auf die Werte in `f` und liefert die Matrix als Resultat.

- `equals(Matrix m)` liefert als Wahrheitswert, ob die Matrix in Struktur und Werten gleich der als Argument gegebenen Matrix ist.

  Hinweis: Verwenden Sie die in der Vorlesung vorgestellte und in LEA bereitgestellte Methode `Mathe.equals` zum Vergleich zweier Gleitkommawerte unter Berücksichtigung von Rundungsabweichungen. Diese Klasse `Mathe` wird im Praktomat zu Tests bereitgestellt.

- `toString()` erzeugt eine Darstellung der Matrix als Zeichenkette. Die Darstellung erfolgt zeilenweise.

  Vor und nach jedem Element steht genau ein Leerzeichen. Jede Zeile ist in runde Klammern `( )` eingeschlossen, gefolgt von einem Zeilenumbruch.

  Zum Beispiel sähe diese Zeichenkette für die Matrix mit den Werten `{{1,2},{3,4},{5,6}` folgendermaßen aus:

  ```
  "( 1.0 2.0 )
  ( 3.0 4.0 )
  ( 5.0 6.0 )
  "
  ```

- `eingabe(java.util.Scanner sc)` liest von `sc` Werte für die Einträge der Matrix in der gleichen Darstellung ein, wie sie durch `toString` erzeugt wird. Die Matrix wird zurückgegeben.

  Wenn die Eingabe keine korrekte Form hat, soll eine `java.util.InputMismatchException` geworfen werden. Dabei sollen jedoch nicht Art und Vielfachheit der White Spaces geprüft werden, sondern nur die druckbaren Zeichen.

Werfen Sie für ungültige Argumente eine `IllegalArgumentException`. Für `null` als Argument soll jedoch eine `NullPointerException` geworfen werden.

Aufgabe 5 [Programmierung]

Schreiben Sie eine Klasse `SWIFT` mit Klassenmethoden zur Erzeugung von IBAN (internationalen Kontonummern). Zur Vereinfachung soll dabei allgemein das System angewendet werden, das zur Bestimmung von IBAN deutscher Konten verwendet wird (obwohl für Konten in anderen Ländern eigentlich andere Systeme gelten). Eine IBAN ist demnach aus folgenden Elementen zusammengesetzt:

(a) Kürzel des Landes (zwei Buchstaben, hier stets Großbuchstaben, z.B. `DE` für Deutschland)

(b) Prüfzahl (2 Ziffern)

(c) Bankleitzahl (8 Ziffern, keine 0 an erster Stelle)

(d) Kontonummer (10 Ziffern, 0 an jeder Stelle möglich, aber nicht an allen zugleich)

Die Klassenmethode `iban` erhält als Eingaben

- Landeskürzel als Zeichenkette der Länge 2 (z.B. `"DE"` oder `"de"` oder eine Mischform davon)
- Bankleitzahl als 8-stellige Ganzzahl (z.B. 12345678)
- Kontonummer als max. 10-stellige Ganzzahl (z.B. 123456)

und gibt die entsprechende IBAN als 22-stellige Zeichenkette zurück.

Die Umrechnung erfolgt gemäß folgender Vorgehensweise. Für jeden Schritt soll eine eigene Klassenmethode bereitgestellt werden.

(a) Das zweibuchstabige Landeskürzel wird in Großbuchstaben umgewandelt.

(Methode `landeskuerzel`, die eine Zeichenkette annimmt und eine zurückgibt)

(b) Die Bankleitzahl wird in eine 8-stellige Zeichenkette verwandelt.

(Methode `bankleitzahl`, die eine Ganzzahl annimmt und eine Zeichenkette zurückgibt)

(c) Die Kontonummer wird in eine 10-stellige Zeichenkette verwandelt, ggf. durch führende `'0'`en – z.B. 123456 in `"0000123456"`.

Hinweis: Überlegen Sie, welchen Zahltyp Sie verwenden müssen, um alle Kontonummern darstellen zu können.

(Methode `kontonummer`, die eine Ganzzahl annimmt und eine Zeichenkette zurückgibt)

(d) Aus der Bankleitzahl und der Kontonummer wird durch Aneinanderhängen der Bankleitzahl gefolgt von der 10-stelligen Form der Kontonummer die 18-stellige BBAN erzeugt – diese wäre z.B. `"123456780000123456"` für die Bankleitzahl 12345678 und die Kontonummer 123456.

(Methode `bban`, die Bankleitzahl und Kontonummer als Ganzzahlen annimmt und eine Zeichenkette zurückgibt)

(e) Jeder der beiden (Groß-)Buchstaben des Ländercodes lässt sich auf folgende Weise in eine 2-stellige Zahl umgewandeln:

(1) Von dem Buchstaben wird der Buchstabe `'A'` abgezogen.
Hinweis: Man kann mit Zeichen wie mit Zahlen rechnen – dies haben wir noch nicht behandelt, kann hier aber einfach verwendet werden: Wenn die Zeichenvariable `b` einen Großbuchstaben enthält, gibt der Ausdruck `b - 'A'` die Position des Buchstaben im Alphabet an, gezählt ab 0.

(2) Zu dem resultierenden Wert wird 10 addiert.

So ergibt sich jeweils eine Zahl von 10 (für den Buchstaben `'A'`) bis 35 (für `'Z'`) – für `'D'` erhält man $3 + 10 = 13$, für `'E'` $4 + 10 = 14$.

(Methode `zahlFuerUpper` nimmt ein Zeichen an, das ein Großbuchstabe sein muss, und gibt den entsprechenden Zahlwert zurück)

(f) Für die folgende Berechnung ist die BBAN um einen temporären Anhang zu ergänzen. Dieser setzt sich zusammen aus:

(1) der 2-stelligen Zahldarstellung des ersten Buchstabens des Landeskürzels

(2) der 2-stelligen Zahldarstellung des zweiten Buchstabens

(3) zwei Nullen

Es ergibt sich ein 6-stelliger Anhang als Zeichenkette – für die o.g. Beispiele `"131400"`.

(Methode `bbanErgaenzung` nimmt das Landeskürzel als Zeichenkette aus zwei Großbuchstaben an und gibt den Anhang als Zeichenkette zurück)

(g) Für die aus der BBAN und dem Anhang zusammengesetzte 24-stellige Zahl (im gegebenen Beispiel `"123456780000123456131400"`) wird der Modulo (Rest) bezüglich 97 berechnet.

Eine 24-stellige Dezimalzahl übersteigt den Wertebereich von `int` und `long`. Deshalb verwendet man dafür folgenden Algorithmus (hier ohne Beweis):

(1) Extrahiere die Zahl, die aus den ersten max. 9 (ggf. weniger) Ziffern gebildet wird.
Hinweis: Die Klassenmethode `parseLong` in der Klasse `Long` nimmt eine Zeichenkette an, die aus Ziffern besteht. Sie gibt die Ganzzahl zurück, die durch diese Ziffern dargestellt wird.

(2) Berechne den Modulo dieser Zahl bezüglich 97.

(3) Dieser (ein- oder zweistellige) Wert wird vor die übrige (zu Beginn $24 - 9 = 15$-stellige) Zeichenkette gehängt.

(4) Fahre wieder ab dem ersten Schritt fort, bis keine Ziffern mehr übrig sind.

(5) Wenn die gesamte Zeichenkette verarbeitet ist (nach max. 4 Wiederholungen), ist der zuletzt berechnete Modulo der der gesamten Zahl.

Für das o.g. Beispiel `123456780000123456131400`:

$$\begin{aligned} 123456780 &\equiv 30 \mod 97 \\ 300001234 &\equiv 22 \mod 97 \\ 225613140 &\equiv 64 \mod 97 \\ 640 &\equiv 58 \mod 97 \end{aligned}$$

Also ist $123456780000123456131400 \equiv 58 \mod 97$.

(Methode `zahlAlsStringMod97` nimmt eine Zeichenkette aus Ziffern an und gibt den Modulo als Ganzzahl zurück)

(h) Danach subtrahiert man diesen Modulo-Wert von dem Wert 98 und betrachtet das Ergebnis als Zeichenkette. Wenn das Ergebnis einstellig, also kleiner als 10 ist, fügt man eine führende Null hinzu, so dass sich stets eine zweistellige Zeichenkette ergibt. Diese beschreibt die Prüfzahl. Im o.g. Beispiel ist die Prüfzahl $98 - 58 = 40$.

(Methode `pruefzahl` nimmt das Landeskürzel in Großbuchstaben sowie die BBAN als Zeichenketten an und gibt die Prüfzahl als zweistellige Zeichenkette zurück.)

Hinweise:

- Das Verfahren ist in genau dieser Form nur für deutsche Konten definiert, für andere Länder in ähnlicher, aber anderer Weise. Ihre Implementierung soll auch mit anderen Länderkürzeln in gleicher Form umgehen (z.B. `"FR"` für Frankreich), obwohl das offizielle Verfahren für diese Länder ein anderes ist.
- Für deutsche Testdaten können Sie Ihre Ergebnisse mit denen eines der vielen online-IBAN-Rechner vergleichen.

Aufgabe 6 [Programmierung]

Ergänzen Sie Ihre Lösung zu Aufgabe 5, so dass von jeder Methode für jede ungültige Eingabe eine `IllegalArgumentException` geworfen wird.

Es gilt aber nicht als Fehler, ein anderes Länderkürzel als das deutsche anzugeben. Auch ungültige Länderkürzel wie `XY` müssen nicht abgefangen werden.

Hinweis: Fangen Sie jeden Fehler (nur) an der Stelle ab, wo er tatsächlich ein Problem bewirkt. So vermeiden Sie wiederholtes Überprüfen derselben Bedingung.

Lösungen zu mit [**Programmierung**] markierten Aufgaben sind im **Praktomat** einzureichen.

Lösungen zu mit [**Programmierung – nicht bewertet**] markierten Aufgaben können ebenfalls im **Praktomat** eingereicht werden, werden jedoch nicht bewertet.

Allgemeine **Fragen** zu den Aufgaben können Sie im **LEA-Forum „Übungsaufgaben“** stellen.

Hilfe bei der Lösung der Aufgaben erhalten Sie in den **Übungen** und in der **Studierwerkstatt** .