Praktikum Grundlagen der Programmierung

Prof. Dr. Harald Räcke, R. Palenta, A. Reuss, S. Schulze Frielinghaus WS 2016/17

Übungsblatt 5 Abgabefrist: 28.11.2016 (vor 05:00 Uhr)

Σ Punkte: 20 | Σ Bonuspunkte: 5

Aufgabe 5.1 (P) Grundlegende Array-Methoden

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Bausteine (siehe Anhang)!

Implementieren Sie in einer Klasse `Arrays` die folgenden Methoden:

- `public static void print(int[] feld)` - gibt das Array `feld` auf der Konsole (`System.out.println(String text)`) aus. Das Array soll mit einer öffnenden geschweiften Klammer beginnen und mit einer schließenden geschweiften Klammer enden; die einzelnen Elemente des Arrays sollen durch ein Komma und Leerzeichen getrennt sein.

  Beispiel: `print(new int[] {1, 2, 3, 4, 5})` liefert auf der Konsole die Ausgabe `{1, 2, 3, 4, 5}`.

- `public static void minUndMax(int[] feld)` - gibt Minimum und Maximum des Arrays `feld` auf der Konsole (`System.out.println(String text)`) aus. Das Array wird dabei nur einmal durchlaufen.

  Beispiel: `minUndMax(new int[] {1, 10, 25, -13, 1000})` liefert auf der Konsole die Ausgabe
  `Minimum = -13, Maximum = 1000`.

- `public static int[] invertieren(int[] feld)` - gibt ein neues Array zurück, das die Elemente von `feld` in umgekehrter Reihenfolge enthält.

  Beispiel: `invertieren(new int[] {0, 1, 2, 3})` liefert ein Array `{3, 2, 1, 0}` zurück.

- `public static int[] schneiden(int[] feld, int laenge)` - gibt ein neues Array zurück, dass `laenge` Zeichen lang ist und die Elemente von `feld` in der gleichen Reihenfolge und so viele wie möglich enthält. Sollte das zurückgegeben Feld größer sein als das übergebene, sollen die zusätzlichen Positionen den Wert 0 haben.

  Beispiel: `schneiden(new int[] {1, 2, 3}, 2)` liefert ein Array `{1, 2}` und `schneiden(new int[] {1, 2, 3}, 5)` liefert ein Array `{1, 2, 3, 0, 0}`.

- `public static int[] linearisieren(int[][] feld)` - gibt ein neues eindimensionales Array zurück, das die Werte des übergebenen zweidimensionalen Arrays `feld` enthält, zurück. Die Zeilen des Arrays `feld` sollen dabei nacheinander in der ihrem Zeilenindex entsprechenden Reihenfolge in dem eindimensionalem Array abgelegt werden. Beachten Sie, dass Zeilen nicht gleich lang sein müssen.

  Beispiel:

  ```
  linearisieren(new int[][] {{1, 3}, {25}, {7, 4, 6, 9}})
  ```

  liefert ein Array `{1, 3, 25, 7, 4, 6, 9}`.

Aufgabe 5.2 (P) Palindrom

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Bausteine (siehe Anhang)!

Schreiben Sie ein Programm `Palindrom.java` mit dessen Hilfe der Benutzer für eine eingegebene Zeichenkette ermitteln kann, ob es sich bei der Zeichenkette um ein Palindrom handelt. Ein Palindrom ist eine Zeichenkette, die von vorne und hinten gelesen gleich lautet, z.B. „anna“ oder „rotor“. Wir nehmen für diese Aufgabe an, dass ein Buchstabe in Groß- und Kleinschreibung gleich ist, d.h. auch „Anna“ ist ein Palindrom. Gehen Sie wie folgt vor:

- Schreiben Sie eine Methode `public static String toLowerCase(String input)`, die als Rückgabewert einen String liefert, der die gleiche Zeichenfolge wie `input` ist, wobei alle Großbuchstaben `A` bis `Z` in den entsprechenden Kleinbuchstaben umgewandelt sind.
- Schreiben Sie eine Methode `public static char[] toCharArray(String input)`, die den String `input` in ein `char`-Array umwandelt und dieses als Rückgabewert zurückgibt. Dabei gilt, dass der erste Buchstabe an der ersten Position im Array steht, der zweite Buchstabe an der zweiten Position usw.
- Schreiben Sie eine Methode `public static boolean isPalindrome(char[] input)`, die das Array `input` mit Hilfe einer Schleife durchläuft und bestimmt, ob es sich bei der *gesamten* Zeichenkette um ein Palindrom handelt. Der Rückgabewert is `true`, wenn die Zeichenkette ein Palindrom ist, ansonsten `false`. Überlegen Sie sich, ob Sie das Array in seiner vollen Länge durchlaufen müssen, oder ob es reicht, über die Hälfte zu iterieren.
- Lesen Sie mittels `readString()` einen nicht-leeren String vom Benutzer ein, für den Sie mithilfe der zuvor implementierten Methoden überprüfen, ob es sich bei der Zeichenkette um ein Palindrom handelt. Teilen Sie das Ergebnis dieser Überprüfung dem Benutzer mittels `write(String txt)` mit.

Aufgabe 5.3 (P) Matrix- und Vektormultiplikation

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Bausteine (siehe Anhang)!

Vektoren und Matrizen lassen sich durch Arrays darstellen. Eine Matrix ist hier durch einen Vektor von Zeilen gegeben. D.h. der Eintrag `m[2][0]` steht in der dritten Zeile an erster Stelle. Vektoren werden durch eindimensionale Arrays dargestellt, d.h. es wird nicht zwischen Zeilen- und Spaltenvektoren unterschieden. Im folgenden sollen Methoden zur Multiplikation von Matrizen und Vektoren erstellt werden. Sie können in allen Methoden davon ausgehen, dass die Dimensionen der übergebenen Arrays entsprechend der Operation zueinander passen.

1. Die Multiplikation zweier Vektoren ist folgendermaßen definiert:

$$\begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \sum_{j=1}^{n} a_j \cdot b_j$$

   Schreiben Sie eine Methode `public static int vecvecmul(int[] a, int[] b)`, welche die Multiplikation zweier Vektoren implementiert.

2. Die Multiplikation einer Matrix mit einem Vektor ist folgendermaßen definiert:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{pmatrix}$$

wobei die Einträge c_i gegeben sind durch

$$c_i = \sum_{j=1}^{n} a_{ij} \cdot b_j.$$

Schreiben Sie eine Methode
`public static int[] matvecmul(int[][] a, int[] b)`, welche die Multiplikation einer Matrix und eines Vektors implementiert.

3. Die Transposition einer Matrix ist folgendermaßen definiert:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}^{\top} = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{pmatrix}$$

Schreiben Sie eine Methode `public static int[][] transpose(int[][] a)`, welche die Transponierte zur Matrix a zurückgibt.

4. Die Multiplikation zweier Matrizen ist folgendermaßen definiert:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{n2} & \dots & c_{mk} \end{pmatrix}$$

wobei die Einträge c_{ij} gegeben sind durch

$$c_{ij} = \sum_{l=1}^{n} a_{il} \cdot b_{lj}.$$

Schreiben Sie eine Methode
`public static int[][] matmatmult(int[][] a, int[][] b)`, welche das Produkt der Matrizen a und b zurückgibt.

5. Schreiben Sie eine Methode `public static void main(String[] args)`, um Ihre Methoden zu testen.

Aufgabe 5.4 (H) Linja [20 Punkte + 2 Bonuspunkte]

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Bausteine (siehe Anhang)!

Schreiben Sie ein MiniJava-Programm `Linja.java`, dass das Spiel *Linja* für zwei Spieler implementiert. Die Regeln für das Spiel finden Sie unter `http://www.steffen-spiele.de/fileadmin/Spiele/PDFs/Linya_Anl._2015_dt.pdf`. Die optionale Bonusregel muss

nicht implementiert werden. Beachten Sie, dass die Startlinie des einen Spielers die Ziellinie des anderen Spielers ist. Steine, die die Ziellinie erreicht oder überschritten haben, werden auf dem Spielfeld nicht mehr dargestellt.

Die Spielerinteraktion soll über die von *MiniJava* zur Verfügung stehenden Methoden erfolgen. Die Ausgabe des Spielfelds soll über die Konsole erfolgen. Überprüfen Sie alle Eingaben auf Korrektheit/Gültigkeit. Sollte eine Eingabe nicht zulässig sein, soll der Spieler darauf hingewiesen werden und erneut nach der Eingabe gefragt werden.

Verwenden Sie das beigefügte Programmgerüst und implementieren Sie die fehlenden Methoden (markiert mit `// TODO`) entsprechend ihrer beschriebenen Funktionalität. Der vorgegebene Code im Programmgerüst darf nicht verändert werden. Das mehrdimensionale Array `spielfeld` ist als globale Variable definiert. D.h. dass alle Methoden auf diese Variable zugreifen können und alle Änderungen in allen Methoden sichtbar sind.

Hinweise: Beginnen Sie mit den Methoden, die Ihnen am einfachsten erscheinen, z.B. `findeStein(int stein)` oder `gueltigeEingabe(int stein, int spieler)`. Fügen Sie eventuell bei noch nicht implementierten Methoden eine sinnvolle standardmäßige Rückgabe ein, um die Methoden weitestgehend unabhängig voneinander implementieren zu können. Testen Sie jede Ihrer Methoden einzeln. Auch Methoden wie `spielende()` oder `zaehlePunkte()` können zu Beginn, auch wenn die Spiellogik noch nicht implementiert ist, erstellt werden und mit entsprechenden Testaufrufen auf Korrektheit überprüft werden.

Bonusaufgabe: Erweitern Sie Ihr Programm um die optionale Bonusregel. Der Spieler soll zu Beginn des Spieles gefragt werden, ob er mit oder ohne optionale Bonusregel spielen möchte. Erweitern Sie dazu die Methode `void spielerZieht(int spieler)` zu `void spielerZieht(int spieler, boolean bonus)`. Wenn `bonus == true`, dann wird die Bonusregel beachtet, ansonsten nicht.

Lösungsvorschlag 5.4

Korrekturbemerkungen:

- `main(String args[])`: 1 Punkt
- `spielerZieht(int spieler, boolean bonus)`:
  - Anfangszug: 1 Punkt
  - Folgezug: 2 Punkte
  - Bonuszug: 3 Punkte
- `zaehlePunkte()`: 3 Punkte
- `spielende()`: 2 Punkte
- `setzeZug(int stein, int weite, boolean vorwaerts)`: 5 Punkte
- `steineInReihe(int reihe)`: 1 Punkt
- `findeStein(int stein)`: 1 Punkt
- `gueltigeEingabe(int stein, int spieler)`: 1 Punkt

Nicht genau definiert nach den offiziellen Regeln sind unter anderem folgende Aspekte.

- Dürfen eigene Steine im Bonuszug von der Ziellinie zurückbewegt werden oder sind diese außerhalb des Spielfelds und werden somit nicht mehr bewegt? Beide Varianten sind in somit in Ordnung.
- Dürfen auf der Startlinie mehr als sechs Steine dieses Spielers stehen? Die Startlinie des Spielers A ist gleichzeitig die Ziellinie des Spielers B. Für Spieler B gilt klar, dass beliebig viele Steine in der Ziellinie stehen dürfen. Darf Spieler A mehr als sechs seiner eigenen Steine in seine Ziellinie stellen ist jedoch nicht genau definiert. Hier

darf man davon ausgehen, dass nur sechs eigene Steine auf der Ziellinie eines Spielers stehen dürfen. Die Variante, dass beliebig viele eigene Steine auf der Ziellinie eines Spielers stehen, ist ebenfalls erlaubt.
- Mögliche Deadlocks, die im Spiel entstehen können (kein Spieler kann mehr einen gültigen Zug machen, aber das Kriterium fürs Spielende ist nicht erreicht), sind nicht erläutert. Solche Situationen müssen nicht extra geprüft oder abgefangen werden. D.h. bei jeglicher Eingabe eines zu ziehnden Steins kommt eine erneute Abfrage, welcher Stein gezogen werden soll.
- Dürfen Steine, die bereits auf der Ziellinie stehen weiter vorwärts bewegt werden. Ob dies als ungültiger Zug zählt oder als Zug für den alle Schritte entfallen, kann frei gewählt werden.

Es gibt sicherlich weitere Aspekte, die nach den offiziellen Spielregeln nicht eindeutig definiert sind. In diesen Fällen kann selbständig eine vernünftige Auslegung der Spielregeln implementiert werden.

Im Lösungsvorschlag sind maximal sechs eigene Steine auf der eigenen Startlinie erlaubt. Diese werden im Spielfeld dargestellt. Steine des gegenerischen Spielers, die die Ziellinie erreicht haben, werden nicht dargestellt, können aber im Bonuszug rückwärts bewegt werden. Soll ein Stein, der die Zielline erreicht hat, vorwärts bewegt werden, wird dies als ungültiger Zug behandelt.

Aufgabe 5.5 (H) Der *funktion*ierende Cäsar [3 Bonuspunkte]

Verwenden Sie für diese Aufgabe nur die erlaubten Java-Bausteine (siehe Anhang)!

In der Aufgabe B4A6 haben wir die Cäsar-Chiffre kennen gelernt. Wir stellten fest, dass die Verschlüsselung eines Buchstabens ein zyklisches Verschieben im Alphabet ist. Wir folgern daraus, dass die Verschlüsselung gleich der Entschlüsselung mit invertiertem Schlüssel darstellt und vice versa. Angenommen wir hätten eine Methode

```
public static String encrypt(String s, int k)
```

welche einen `String s` anhand des Schlüssels `k` verschlüsselt, dann können wir eine Methode `decrypt` wie folgt definieren:

```
public static String decrypt(String s, int k) {
    return encrypt(s, -(k % 26));
}
```

Im Folgenden wollen wir sukzessiv die Methode `encrypt` programmieren:

1. Schreiben Sie eine Methode `public static char shift(char c, int k)` welche einen `Character c` und einen beliebigen `Integer k` entgegen nimmt und den `Character c` zyklisch im Alphabet um `k` Stellen verschiebt. Ein verschieben findet nur für Buchstaben `a` bis `z` und `A` bis `Z` statt. Alle anderen Zeichen werden unverändert zurück gegeben. Beachten Sie, dass `k` negativ sein kann! Beispiele:

   `shift('c', 5)` liefert `'h'`

   `shift('c', -5)` liefert `'x'`

2. Schreiben Sie nun eine Methode `public static String encrypt(String s, int k)` welche einen `String s` anhand des Schlüssels `k` verschlüsselt. Verwenden Sie dafür die eben definierte Methode `shift`. Die Methode `encrypt` muss die folgenden Eigenschaften haben. Für alle `Strings s` und alle `Integer k` muss die Komposition von `encrypt` und `decrypt` die Identitätsfunktion sein:

   `decrypt(encrypt(s, k), k)` sowie `encrypt(decrypt(s, k), k)` liefern `s`

3. Gegeben ist folgender Geheimtext: "Mrn pnvnrwbcnw Jdopjknw bcnuuc Ajyqjnuj"

   Wie lautet der Klartext?

Schreiben Sie Ihre Lösung in die Datei "`FunctionalCaesar.java`".

Ja, bei dieser Aufgabe handelt es sich um eine *reine* Bonushausaufgabe.

Lösungsvorschlag 5.5

Korrekturbemerkung:

- Für die Methode `shift` gibt es 2 Punkte
- Für die Methode `encrypt` gibt es 1 Punkt
- Für den Klartext gibt es 0 Punkte, denn die Aussage von dem Klartext ist natürlich falsch und somit 0 Punkte wert!

Erlaubte Java-Methoden

MiniJava:

```
public static int readInt()
public static int readInt(String s)
public static int read()
public static int read(String s)
public static String readString()
public static String readString(String s)
public static void write(String output)
public static void write(int output)
public static int drawCard()
public static int dice()
```

returns an integer from the interval $[2, 11]$

returns an integer from the interval $[1, 6]$

String:

`public char charAt(int index)`
Returns the char value at the specified index. An index ranges from `0` to `length() - 1`. The first char value of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Example: `String s = "Hello Students"; char c = s.charAt(7);` saves the character `'t'` in the variable `c`.

`public boolean isEmpty()`
Returns `true` if, and only if, `length()` is `0`.

`public int length()`
Returns the length of this string. The length is equal to the number of Unicode code units in the string.

Example: `String s = "Hello Students"; int l = s.length();` saves the value `14` in the variable `l`.

`public boolean equals(Object obj)`
Compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.
Example: `String s = "Hello Students"; boolean v = s.equals("Hello Students");` here `v` is evaluated to `true`.

System:

`System.out.print(x)` prints the object `x` to the console
`System.out.println(x)` prints the object `x` to the console and terminates the line

Selbstgeschriebene Methoden, die selber nur erlaubte Methoden verwenden, sind erlaubt, sofern dies nicht explizit in der Aufgabenstellung verboten wurde. Methoden, die im gegebenen Programmgerüst definiert wurden, dürfen verwendet werden.