Projektblatt P8 (18 + 3* P)

Abgabe: Donnerstag 7. November 2024, 12:00h

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: Die Methode `matches` 10 (=1+1+1+2+2+3) Punkte

In dieser Aufgabe sollen Sie mehrere Klassenmethoden implementieren, welche jeweils eine Zeichenkette auf eine bestimmte Eigenschaft hin untersuchen. In der Klasse `RegEx` sind hierzu sechs Methoden `check1` - `check6` definiert, die jeweils den Default-Wert `false` zurückgeben. Darüberhinaus definiert die `main`-Methode für jede dieser Methoden mehrere Teststrings, für die die Methode aufgerufen und die Ergebnisse dann ausgegeben werden.
Implementieren Sie nun die sechs Klassenmethoden `check1` - `check6`, die jeweils einen `String` Parameter besitzen und einen boole'schen Wert zurückgeben, der angibt, ob der String eine bestimmte Eigenschaft besitzt. Ist dies der Fall soll die Methode den Wert `true` zurückgeben werden, ansonsten den Wert `false`; Verwenden Sie dazu jeweils die Methode `matches` der Klasse `String` und einen passenden **Regulären Ausdruck**.

`check1` Diese Methode soll überprüfen, ob der als Parameter übergebene String eine positive, ganze Zahl ohne führende Nullen repräsentiert, welche eine ungerade Anzahl von Ziffern besitzt. Beispiele hierfür sind die Strings "1","123" und"999999999".

`check2` Diese Methode soll überprüfen, ob der als Parameter übergebene String eine Primzahl aus dem Intervall $[1, 20]$ repräsentiert. Beispiele hierfür sind die Strings "2" und "19".

`check3` Diese Methode soll überprüfen, ob der als Parameter übergebene String nur aus Ziffern besteht und dabei mindestens einmal die Ziffer 7 enthält. Beispiele hierfür sind die Strings "7" und "0123456789".

`check4` Diese Methode soll überprüfen, ob der als Parameter übergebene String höchstens zweimal den Buchstaben X (als Groß- oder Kleinbuchstaben) enthält. Das gilt z.B. für die Strings "aXb", "Xx", "XaABbX" und "abc".

`check5` Diese Methode soll überprüfen, ob der als Parameter übergebene String eine Zahl repräsentiert, deren erste und letzte Ziffer identisch sind und dabei ungerade Zahlen darstellen. Alle Zeichen zwischen dem ersten und dem letzten Zeichen in dem String sollen identische Ziffern sein (mindestens eine), die gerade Zahlen repräsentieren. Beispiele hierfür sind "10001", "7887" und "929".

`check6` Diese Methode soll überprüfen, ob der als Parameter übergebene String eine positive oder negative Gleitkommazahl in normaler Darstellung (also keine Scientific Notation) repräsentiert. Dabei soll die Zahl keine führenden bzw. zusätzlice Nachkommastellen mit Nullen enthalten. Wenn nur eine Vor- bzw- Nachkommastelle existiert, dann darf diese auch eine Null sein. Beispiele hierfür sind die Strings "-321.0", "100.09" und "-0.1".

Hinweis: Leerzeichen werden in Regulären Ausdrücken als solche interpretiert: Der Ausdruck "a | b" beschreibt die Menge der Wörter {"a "," b"} und nicht die Menge der Wörter {"a","b"}
Eine Programmlauf sollte dann folgende Ausgabe produzieren:

```
Test 1:
"1" -> true
"123" -> true
"135" -> true
"999999999" -> true
"0" -> false
"13" -> false

Test 2:
"2" -> true
"19" -> true
"9" -> false
"21" -> false

Test 3:
"0123456789" -> true
"7" -> true
"117" -> true
"012345689" -> false

Test 4:
"aXb" -> true
"Xx" -> true
"XaABbX" -> true
"abc" -> true
"xaxax" -> false
```

```
"xXx" -> false

Test 5:
"10001" -> true
"7887" -> true
"929" -> true
"1010" -> false
"33" -> false

Test 6:
"1.35" -> true
"0.0" -> true
"-12.345" -> true
"-0.1" -> true
"3.0001" -> true
"2.030" -> false
"-00.1" -> false
```

Aufgabe 2: Die Methode `replaceAll` 6 (=2+2+2) Punkte

Im Folgenden sollen Sie in der Klasse `RegReplace` drei Methoden schreiben, mit deren Hilfe Sie Ersetzungen in einem als `String` übergebenen Parameter vornehmen. Verwenden Sie hierzu jeweils die Instanzmethode `replaceAll` aus der Klasse `String`.

(a) Die Methode `swapChars` soll in einem als Parameter übergebenen `String` alle Vorkommen von zwei, ebenfalls als Parameter übergebenen Zeichen austauschen. Um diese Zeichen beim Aufruf der Methode `replaceAll` in einen `String` umzuwandeln, verwenden Sie hier bitte eine passende Methode aus der Klasse `Character`. Hinweis: Gehen Sie analog zum Austausch von zwei Variablenwerte vor: Definieren Sie sich hierzu eine Hilfsvariable mit der String-Repräsentation des Nullzeichens (`int`-Wert 0) und führen Sie einen Ringtausch durch. Die Testaufrufe in der `main`-Methode sollte folgende Ausgabe generieren:

```
aabbccc --> bbaaccc
128 895 556 788 999 --> 129 985 556 799 888
```

(b) Der String, der der Methode `changeNameOrder` als Parameter übergeben wird, beinhaltet jeweils einen vollständigen Namen, der aus ein oder mehreren Vornamen sowie einem Nachnamen besteht. Alle Namensteile sind jeweils durch ein oder mehrere Whitespace Zeichen (Leerzeichen etc.) voneinander getrennt. Jeder Namensteil kann wiederum ein oder mehrere Wortzeichen, die Umlaute ä, ö oder ü sowie einen Bindestrich beinhalten. Die Methode soll einen String zurückgeben, der aus dem Nachnamen, gefolgt von einem Komma und einem Whitespace Zeichen sowie allen Vornamen (inklusive der Whitespace Zeichen zwischen den Vornamen) besteht. Sie dürfen davon ausgehen, dass der Nachname keine Leerzeichen (oder andere Whitespace Zeichen) enthält.

Die Testaufrufe in der `main`-Methode sollte folgende Ausgabe generieren:

```
Tina Meyer -> Meyer, Tina
Thomas  Müller -> Müller, Thomas
Marie-Agnes Strack-Zimmermann -> Strack-Zimmermann, Marie-Agnes
Rüdiger Ägidius Öhmig  -> Öhmig, Rüdiger Ägidius
Franz-Ferdinand Maximilian Alexander vOstwestfalen-Lippe ->
  vOstwestfalen-Lippe, Franz-Ferdinand Maximilian Alexander
```

(c) Die Methode `replaceUmlaute` soll in einem als Parameter übergebenen `String` die Umlaute ä, ö und ü (bzw. Ä, Ö und Ü) durch die Darstellungen ae, oe und ue (bzw. Ae, Oe und Ue) ersetzen. Hierzu ist in der Methode bereits ein zweidimensionales Feld vorgegeben. Jede Zeile in diesem Feld beinhaltet ein Paar von zwei Strings, welche die Ersetzungen für die Umlaute (nur Kleinbuchstaben) beschreiben.

Durchlaufen Sie das vorgegebene Feld und führen Sie für jedes Paar zwei Ersetzungen in dem als Parameter übergebenen `String` durch: Ersetzen Sie zunächst die für jedes Paar den ersten String in dem Feld durch den zweiten. Führen Sie diese Ersetzung dann nochmals durch, wobei Sie jeweils den ersten Buchstaben in den beiden Strings in einen Großbuchstaben umwandeln, bevor Sie diesen als Parameter an die Methode `replaceAll` verwenden. Die Testaufrufe in der `main`-Methode sollte folgende Ausgabe generieren:

```
Thomas  Müller -> Thomas  Mueller
Rüdiger Ägidius Öhmig  -> Ruediger Aegidius Oehmig
```

Aufgabe 3: Die Klassen `Pattern` und `Matcher` 2 (=2+3*) Punkte

In dieser Aufgabe sollen Sie mit Hilfe der Klassen `Pattern` und `Matcher` alle Vorkommen eines Musters in einem Text herausfiltern und die Matches in der Konsole ausgeben. Ergänzen Sie hierzu die Klasse `RegPatternMatcher` wie im Folgenden beschrieben:

(a) Die Methode `findAdverbs` soll die Anzahl aller Adverben in einem englischen Text bestimmen und dazu die gefundenen Adverben ausgeben. Dabei legen wir hier fest, dass ein Adverb ein Wort ist, welches mit den beiden Buchstaben `ly` endet. Davor muss aber mindestens ein Buchstabe stehen. Ein Wort sei wiederum durch eine Zeichenfolge definiert, die nur aus Klein- und Großbuchstaben besteht.

Die Methode `findAdverbs` wird aus der `main`-Methode mit einzelnen Texten (in diesem Fall einzelnen Sätzen) aufgerufen. Am Ende jeden Satzes steht ein Punkt oder ein anderes Satzzeichen. Sammeln Sie die Adverben jeweils in einem `StringBuffer` Objekt und geben Sie am Ende den Text (Satz), die Adverben und die Anzahle aus.

Die durch die Aufrufe in der `main`-Methode generierte Ausgabe sollte dann wie folgt aussehen:

```
The text "He was happily, crazily, foolishly over the moon."
contains 3 adverb(s): happily crazily foolishly

The text "The horse acted aggressively and stubbornly."
contains 2 adverb(s): aggressively stubbornly

The text "She forgot where to buy the lysol."
contains 0 adverb(s):

The text "She writes poetry beautifully."
contains 1 adverb(s): beautifully

The text "Ilya ran to the store."
contains 0 adverb(s):
```

(b***) Der Methode `findRecipesWithoutChocolate` wird ein String übergeben, der aus mehreren Zeilen besteht, die jeweils eine Zutatenliste für einen Kuchen beinhalten. Jede Zutatenliste beginnt dabei mit dem Schüsselwort `Cake` und endet mit einem Zeilenumbruchs (Newline) Zeichen.

Sie sollen nun alle Zutatenlisten von Kuchen ausfiltern und ausgeben, die keine Schokolade verwenden, d.h. in denen der Substring `chocolate` nicht vorkommt.

Verwenden Sie hiezu in dem Regulären Ausdruck einen negativen Lookahead.

Die durch die Aufrufe in der `main`-Methode generierte Ausgabe sollte dann (bis auf die Zeilenumbrüche) wie folgt aussehen:

```
Cake 2: cream cheese, sugar, vanilla extract, crescent rolls,
  cinnamon, butter, honey
Cake 4: flour, baking powder, salt, cinnamon, butter, sugar, egg,
  vanilla extract, milk, chopped walnuts
Cake 5: gingersnap cookies, chopped pecans, butter, cream cheese, sugar,
  vanilla extract, eggs, canned pumpkin, cinnamon
Cake 7: wafers, cream cheese, sugar, eggs, vanilla extract,
  cherry pie filling
```

Hinweis:
Wenn Sie im Texteditor Probleme mit den Umlauten haben, verwenden Sie die zugehörigen Unicode-Darstellungen:

Ä	\u00c4	ä	\u00e4
Ö	\u00d6	ö	\u00f6
Ü	\u00dc	ü	\u00fc