Projektblatt P7 (27+ 2* P)

Abgabe: Montag 7. November 2021, 10:00h
Entpacken Sie zunächst die Archiv-Datei `vorgaben-p7.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.P7.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.P7.zip *.java
```

Aufgabe 1: Strings — 10 Punkte

In dieser Aufgabe sollen Sie einen von der Kommandozeile übergebenen String, welcher eine DNA-Sequenz darstellt, zunächst in eine RNA-Sequenz und dann in eine Folge von Aminosäuren übersetzen. Ergänzen Sie die Klasse `DNA2AA` hierzu wie im Folgenden beschrieben:

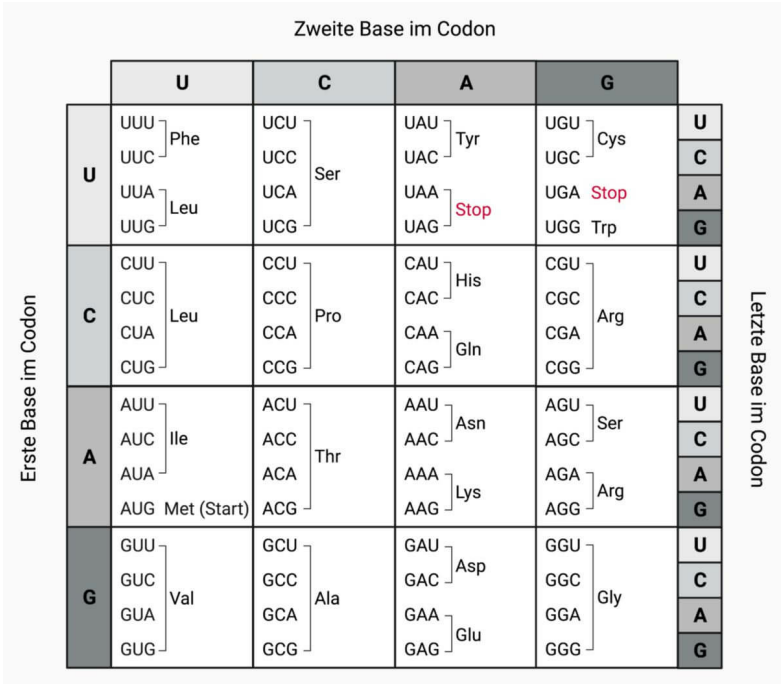
(a) Implementieren Sie zunächst die Methode `isDNA`, die für einen als Parameter übergebenen `String` überprüfen soll, ob dieser eine gültige DNA-Sequenz darstellt. Hierzu muss der `String` ausschließlich aus Zeichen aus der Menge {A, T, C, G} bestehen. Ist dies der Fall, soll die Methode den Wert `true` zurückgeben, ansonsten den Wert `false`. Sie können hier entweder alle Zeichen aus dem `String` einzeln prüfen oder einen passenden *Regulären Ausdruck* und die Methode `matches` verwenden.

(b) Implementieren Sie nun die Methode `dna2rna`, die eine als Parameter übergebene DNA-Sequenz in eine RNA-Sequenz umschreiben (transkribieren) soll. Hierzu müssen die Buchstaben A, T, G und C wie folgt ersetzt werden:

$$\begin{array}{lcl} A & \longrightarrow & U \\ T & \longrightarrow & A \\ C & \longrightarrow & G \\ G & \longrightarrow & C \end{array}$$

Bauen Sie das Ergebnis zeichenweise in einer `StringBuffer` Instanz zusammen und geben Sie diese am Ende zurück. Sie dürfen davon ausgehen, dass der als Parameter übergebene String eine gültige DNA-Sequenz darstellt.

(c) Implementieren Sie die Methode `mapTriplet`, die zu einem als Parameter übergebenen, aus exakt drei Buchstaben bestehenden String die zugehörige Aminosäure bestimmt und zurückgibt. Für die korrekte Zuordnung von Buchstaveb-Tripletts zu Aminosäuren (bzw. zum Stop-Signal) verwenden Sie die folgende Tabelle:



Für den String "GCC" muss beispielsweise der String "Ala" zurückgegeben werden und für jeden der Strings "UAA", "UAG" und "UGA" lautet das Ergebnis "Stop".

Sie dürfen davon ausgehen, dass der Methode nur Strings übergeben werden, die die Länge 3 haben und die aus den Buchstaben der Menge {U, A, G, C} bestehen (sogenannte *Codone*).

(d) Implementieren Sie die Methode `rna2aa`, die eine als Parameter übergebene RNA-Sequenz (bestehend aus den Buchstaben der Menge {U, A, G, C}) in eine Sequenz von Aminosäuren übersetzt. Die Übersetzung soll allerdings erst nach dem Startsignal (Codon "AUG") beginnen und entweder bei einem Stop-Signal enden oder beim letzten vollständigen Triplett in der Sequenz). Gehen Sie hierzu wie folgt vor:

- Prüfen Sie zunächst, an welcher Position das Startsignal "AUG" in der RNA-Sequenz (das erste Mal) vorkommt. Falls die Startsequenz gar nicht in der als Parameter übergebenen RNA-Sequenz enthalten ist, geben Sie eine entsprechende Fehlermeldung aus und geben Sie die Nullreferenz zurück.

- Legen Sie für das Ergebnis eine neue `StringBuffer`-Instanz an, die Sie am Ende der Methode zurückgeben.
- Beginnend mit dem ersten Zeichen nach dem Startcodon, filtern Sie in einer Schleife jeweils drei aufeinanderfolgende Zeichen aus der RNA-Sequenz und übersetzen Sie diese mit Hilfe der Methode `mapTriplet` in eine Aminosäure. Achten Sie darauf, dass Sie das Ende der Sequenz nicht überschreiten und ignorieren Sie ggf. ein oder zwei überzählige Zeichen. Wenn die Methode `mapTriplet` eine Aminosäure (also nicht den String "Stop") zurückgibt, hängen Sie diese sowie den (aus drei Zeichen bestehenden) Trennstring `" - "` an das Ergebnis (die Zeichenkette in dem neuen `StringBuffer`) an. Wenn die Methode `mapTriplet` den String "Stop" zurückgibt, löschen Sie die letzten drei Zeichen in dem `StringBuffer` (um den letzten Trennstring zu entfernen) und brechen Sie die Schleife ab. Anmerkung: Wenn die RNA-Sequenz kein Stop-Codon enthält, sieht man das im Ergebnis daran, dass noch ein Trennstring am Ende steht.

Sie dürfen davon ausgehen, dass der als Parameter übergebene StringBuffer eine gültige RNA-Sequenz enthält.

(e) Ergänzen Sie nun noch die `main`-Methode der Klasse wie folgt:

- Überprüfen Sie, ob das erste Argument eine gültige DNA-Sequenz darstellt.
- Ist dies der Fall, wandeln Sie diese mit Hilfe der Methoden `dna2rna` und `rna2aa` um und geben Sie die Eingabesequenz (DNA), das Zwischenergebnis (RNA) und das Endergebnis (Aminosäuren) in der Konsole aus.

Für die DNA-Sequenz `TACAAGCAGTTAGTCGTGGAAACACCAAGTATC` sollte die Ausgabe dann wie folgt aussehen:

```
DNA: TACAAGCAGTTAGTCGTGGAAACACCAAGTATC
RNA: AUGUUCGUCAAUCAGCACCUUUGUGGUUCAUAG
Aminosaeuren:
 Phe - Val - Asn - Gln - His - Leu - Cys - Gly - Ser
```

Aufgabe 2: Kalenderdaten 5 Punkte

In dieser Aufgabe sollen Sie eine Terminserie generieren und ausgeben, die über 14 Wochen jeweils zwei wöchentliche Termine (Montag, 10:15h und Donnerstag 15:30) beinhaltet und am ersten Montag nach einem vorgegebenen Tag beginnt. Ergänzen Sie hierzu die Klasse Termine wie folgt:

(a) Implementieren Sie die Methode `termineMoDo`, so dass ein Feld mit 28 Objekten des Typs `LocalDateTime` generiert und am Ende als Ergebnis zurückgegeben wird, welches folgende Bedingungen erfüllt:

- Der erste Termin soll am ersten Montag nach dem als Parameter übergebenen Tag `d` liegen. Fällt `d` auf einen Montag, sollte der erste Termin eine Woche danach sein. Der zweite Termin soll am Donnerstag in der gleichen Woche sein.
- Die weiteren Termine sollen in wöchentlichem Abstand montags und donnerstags folgen.
- Insgesamt sollen sich die Termine dabei über 14 Wochen erstrecken.
- Die Uhrzeit für die Termine am Montag ist 10:15h, die Uhrzeit für die Termine am Donnerstag ist 15:30h.

Bestimmen Sie zunächst (in einer Schleife) den ersten Tag, der nach dem als Parameter übergebenen Tag liegt und auf einen Montag fällt. Generieren Sie dann das Feld und legen Sie die ersten beiden Termine (an diesem Montag und dem darauffolgenden Donnerstag) in dem Feld an. Legen Sie dann alle weiteren Termine in dem Feld in einer Schleife an, indem Sie diese systematisch aus den bereits vorhandenen Terminen berechnen.

(b) Implementieren Sie die Methode `printDates`, so dass die als Parameter übergebenen Termine in der Konsole ausgegeben werden. Die Ausgabe sollte dann wie folgt aussehen (hier nur die ersten und letzten vier Termine).

```
Mo. 05. September 2022 (10:15h)
Do. 08. September 2022 (15:30h)
Mo. 12. September 2022 (10:15h)
```

```
Do. 15. September 2022 (15:30h)
...
Mo. 28. November 2022 (10:15h)
Do. 01. Dezember 2022 (15:30h)
Mo. 05. Dezember 2022 (10:15h)
Do. 08. Dezember 2022 (15:30h)
```

Hinweise: Schauen Sie sich die Klassen `LocalDateTime`, `LocalTime` und `LocalDate` in der API-Dokumentation an und überlegen Sie sich, wie Sie mit den in diesen Klassen vorhandenen Konstruktoren und Methoden die geforderten Termine erzeugen können. Verwenden Sie auch den Aufzählungstyp `DayOfWeek`. Um für die Ausgabe deutsche Wochentage und Monatsnamen zu erhalten, verwenden Sie das `Locale` Objekt `Locale.GERMANY`.

Aufgabe 3: Reguläre Ausdrücke 9+2*Punkte

In dieser Aufgabe sollen Sie mehrere Klassenmethoden implementieren, welche jeweils eine Zeichenkette auf eine bestimmte Eigenschaft hin untersuchen. In der Datei `Regex.java` ist hierzu eine Klasse `Regex` definiert, welche 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, ansonsten den Wert `false`; Verwenden Sie dazu jeweils die Methode `matches` der Klasse `String` und einen passenden **Regulären Ausdruck**. In einigen Fällen benötigen Sie auch zwei Aufrufe der `matches` Methode mit verschiedenen Regulären Ausdrücken, die Sie durch logische Operatoren geeignet kombinieren müssen.

`check1` Diese Methode soll überprüfen, ob der als Parameter übergebene String eine positive ganze Zahl repräsentiert, die ohne Rest durch 5 teilbar ist. Führende Nullen sind hierbei erlaubt.

`check2` Diese Methode soll überprüfen, ob der als Parameter übergebene String eine Hexadezimalzahl repräsentiert. Hierzu muss der String mit

"0x" oder "0X" beginnen und danach können beliebig viele Ziffern und / oder die Buchstaben 'A' - 'F' bzw. 'a' - 'f' folgen. Groß- und Kleinbuchstaben dürfen dabei gemischt werden. Hierbei muss aber mindestens ein Zeichen (Ziffer oder Buschstabe) nach dem "0x" bzw "0X" vorhanden sein.

`check3` Diese Methode soll überprüfen, ob der als Parameter übergebene String aus genau vier Ziffern besteht, aber nicht der Ziffernfolge "1234" entspricht.

`check4` Diese Methode soll überprüfen, ob der als Parameter übergebene String eine beliebig lange Zeichenkette darstellt, die keine Whitespace-Zeichen, aber mindestens einen Großbuchstaben und mindestens eine Ziffer enthält (auch nicht am Anfang oder Ende)

`check5` Diese Methode soll überprüfen, ob der als Parameter übergebene String aus genau drei Ziffern besteht, die aber nicht alle identisch sein dürfen. Der String muss also mindestens zwei verschiedene Ziffern enthalten.

`check6` Diese Methode soll überprüfen, ob der als Parameter übergebene String aus genau drei Ziffern besteht, die alle paarweise verschieden sind.

Hinweise:

- Manchmal ist es leichter eine Eigenschaft nicht direkt zu prüfen, sondern das Gegenteil, also z.B. anstelle "alle Ziffern unterschiedlich" prüft man die Eigenschaft "nicht alle Ziffern gleich".
- 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:
Die Zeichenkette repraesentiert eine positive Zahl, die ohne Rest
durch 5 teilbar ist.
"20" -> true
"25" -> true
```

```
"000" -> true
"001" -> false
"5123456789" -> false

Test 2:
Die Zeichenkette stellt eine Hexadezimalzahl dar
"0xa" -> true
"0XfF" -> true
"0x1" -> true
"0x" -> false
"0X A" -> false

Test 3:
Die Zeichenkette repraesentiert eine vierstellige positive
ganze Zahl, aber nicht 1234
"1233" -> true
"1111" -> true
"00000" -> false
"123" -> false
"1234" -> false

Test 4:
Die Zeichenkette stellt eine beliebig lange Zeichenkette ohne
Whitespace Zeichen, aber mit mindestens einem Großbuchstaben
und einer Ziffer dar
"Abcde1" -> true
"1a2B3c" -> true
"000000" -> true
" 000000" -> false
"1a2b3c" -> false
"abcdeF" -> false

Test 5:
Die Zeichenkette besteht aus genau drei Ziffern, die aber nicht
alle gleich sein duerfen
"123" -> true
"747" -> true
"255" -> true
```

```
"1234" -> false
"999" -> false

Test 6:
Die Zeichenkette besteht aus genau drei unterschiedlichen Ziffern
"123" -> true
"951" -> true
"121" -> false
"355" -> false
"110" -> false
```

Aufgabe 4: Reguläre Ausdrücke 3 Punkte

In dieser Aufgabe sollen Sie mit Hilfe eines Regulären Ausdrucks sowie der Klassen `Pattern` und `Matcher` die Anzahl der Ziffern in einem Text bestimmen und ausgeben. In der Klasse `Text` ist hierzu eine Klassenvariable `beispiel` mit einem Beispieltext vorgegeben, der in der `main`-Methode der Klasse `Ziffern` verwendet wird.
Ergänzen Sie die Methode `zaehlen` in der Klasse `Ziffern` so, dass mit Hilfe eine Regulären Ausdrucks alle Ziffernfolgen aus dem als Parameter übergebenen Text ausgefiltert werden. Halten Sie sich dabei an das Beispiel von Folie 53 (Foliensatz 08) und berechnen Sie dabei die Summe der Längen über alle Matches. Geben Sie das Ergebnis am Ende aus.
Das Ergebnis sollte der Antwort auf die "endgültige Frage nach dem Leben, dem Universum und dem ganzen Rest" entsprechen.