Institut für Informatik
Praktische Informatik und Bioinformatik
Prof. Dr. Ralf Zimmer

Wintersemester 2019/20

Übungen zum Propädeutikum Programmierung in der Bioinformatik

Blatt 6

Termin: Dienstag, 26. November 2019

Übung 1 *Erweiterung von `DnaSequence`*

Falls du bisher noch nicht Übung 1 vom Übungsblatt der letzten Woche gemacht hast, sollte diese erst fertiggestellt werden. Alternativ steht eine fertige `DnaSequence`-Klasse auf der Propädeutikumswebsite zum Download bereit. Versichere dich aber, dass du verstehst was die Klasse macht bevor du losprogrammierst.

1. Setze den Sichtbarkeitsmodifikator von allen Attributen in `DnaSequence` auf `private`. Schreibe dann zu allen Attributen der Klasse jeweils einen Getter und einen Setter.
2. Schreibe danach eine statische Variante von `isValidSequence()` mit der Signatur `isValidSequence(String seq)`. Überlege dir warum es sinnvoll sein kann, gerade diese Methode statisch zu implementieren.
3. Schreibe einen *zusätzlichen* Konstruktor `DnaSequence(String sequence)`. Dieser soll, nachdem keine `id` übergeben wird, der Sequenz einfach eine fortlaufende Zahl als `id` zuweisen.

Lösung 1

```java
public class DnaSequence {
    // 1.1: Auf private setzen
    private String id;
    private String sequence;
    private static int counter = 0; // 1.3: Statische Variable für das zuweisen als id

    // 1.1: Getter und Setter
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getSequence() {
        return sequence;
    }
    public void setSequence(String sequence) {
        this.sequence = sequence;
    }

    public DnaSequence(String id, String sequence) {
        this.id = id;
        if (isValidSequence(sequence)) {
          System.out.println("Achtung: Keine gültige DNA Sequenz!");
        }
        this.sequence = sequence;
    }

    // 1.3: Konstruktor der fortlaufende Zahl als id zuweist
    public DnaSequence(String sequence) {
        this.id = String.valueOf(counter);
        counter++;
```

```java
        this.sequence = sequence;
    }

    // 1.2: Statische Variante von isValidSequence
    public static boolean isValidSequence(String sequence) {
        for (char c : sequence.toCharArray()) {
            if (c != 'A' && c != 'G' && c != 'C' && c != 'T') return false;
        }
        return true;
    }

}
```

Übung 2 *Rosalind-Recycling: Transcribing DNA into RNA*

Auf Blatt 3 war verlangt die Rosalind-Aufgabe Transcribing DNA into RNA zu bearbeiten. Erweitere `DnaSequence` um eine Methode `getRnaSequence()` welche dir die Sequenz eines `DnaSequence`-Objekt in RNA übersetzt und als String zurückgibt.

Lösung 2

```java
// 2: Transcribing DNA into RNA
public String getRNASequence() {
   // Umwandeln des Sequenzstrings in ein char-Array, um es effizienter zu bearbeiten
   char[] rnaSequence = this.sequence.toCharArray();
   // Für jedes Nukleotid überprüfen ob es ein T ist,
   // wenn ja an dieser Position mit U austauschen
   for (int i = 0; i < rnaSequence.length; i++) {
       if (rnaSequence[i] == 'T') rnaSequence[i] = 'U';
    }
    // Nachdem auf einem char-Array gearbeitet wurde, jetzt aus diesem
    // einen String machen und returnen
    return new String(rnaSequence);
}
```

Übung 3 *Rosalind: `Protein`-Klasse*

In dieser Aufgabe sollst du eine weitere Klasse `Protein` schreiben, die eine Aminosäuresequenz modelliert.

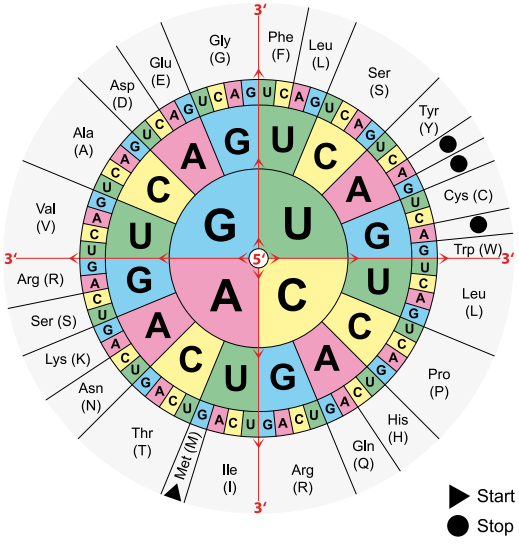
Nach dem Zentralen Dogma der Molekularbiologie wird DNA in RNA übersetzt, und die RNA in ein Protein. Auf vereinfachte Weise wurde bereits in der vorherigen Aufgabe die Transkription (DNA→RNA) durch die Methode `getRnaSequence()` modelliert. In der `Protein`-Klasse soll nun die Translation (RNA→Protein) modelliert werden.

1. Die Klasse soll (wie `DnaSequence`) die Attribute `String id`, `String sequence`, sowie Getter und Setter besitzen.
2. Die Methode `rnaToProtein(String sequence)` soll implementiert werden. Diese nimmt als Input einen RNA-String an und übersetzt diesen anhand des genetischen Codes[1] in eine Aminosäuresequenz. Diese Sequenz soll dann zurückgegeben werden.

   Eine Beispielsequenz würde (vgl. Abbildung) so übersetzt:

   $$\underbrace{\texttt{CCUAGAACCGAGAUAAAU}}_{\texttt{P R T E I N}}$$

3. Nun sollen *zwei* Konstruktoren implementiert werden:



Codon-Sonne an der von innen nach außen für jedes Basentriplett (*Codon*) die zugehörige Aminosäure abgelesen werden kann

[1] Im Rosalind-Testdatensatz wird der Einfachheit halber davon ausgegangen, dass jeder RNA-String mit einem Start-Codon beginnt und einem Stop-Codon endet. Es muss also nicht nach dem Beginn gesucht werden.

(a) `Protein(String rnaseq)`

(b) `Protein(DnaSequence dnaseq)`

Ob dein Algorithmus korrekt funktioniert kannst du wieder auf Rosalind: Translating RNA into Protein überprüfen.

Lösung 3

```java
import java.util.Map;
import static java.util.Map.entry;

public class Protein {
    private String id;
    private String sequence;
    // Variante 1: Mit einer Map des Codon Tables
    private static final Map<String, String> CODON_TABLE = Map.ofEntries(
       entry("UUU", "F"),  entry("CUU", "L"),  entry("AUU", "I"),  entry("GUU", "V"),
       entry("UUC", "F"),  entry("CUC", "L"),  entry("AUC", "I"),  entry("GUC", "V"),
       entry("UUA", "L"),  entry("CUA", "L"),  entry("AUA", "I"),  entry("GUA", "V"),
       entry("UUG", "L"),  entry("CUG", "L"),  entry("AUG", "M"),  entry("GUG", "V"),
       entry("UCU", "S"),  entry("CCU", "P"),  entry("ACU", "T"),  entry("GCU", "A"),
       entry("UCC", "S"),  entry("CCC", "P"),  entry("ACC", "T"),  entry("GCC", "A"),
       entry("UCA", "S"),  entry("CCA", "P"),  entry("ACA", "T"),  entry("GCA", "A"),
       entry("UCG", "S"),  entry("CCG", "P"),  entry("ACG", "T"),  entry("GCG", "A"),
       entry("UAU", "Y"),  entry("CAU", "H"),  entry("AAU", "N"),  entry("GAU", "D"),
       entry("UAC", "Y"),  entry("CAC", "H"),  entry("AAC", "N"),  entry("GAC", "D"),
       entry("UAA", ""),   entry("CAA", "Q"),  entry("AAA", "K"),  entry("GAA", "E"),
       entry("UAG", ""),   entry("CAG", "Q"),  entry("AAG", "K"),  entry("GAG", "E"),
       entry("UGU", "C"),  entry("CGU", "R"),  entry("AGU", "S"),  entry("GGU", "G"),
       entry("UGC", "C"),  entry("CGC", "R"),  entry("AGC", "S"),  entry("GGC", "G"),
       entry("UGA", ""),   entry("CGA", "R"),  entry("AGA", "R"),  entry("GGA", "G"),
       entry("UGG", "W"),  entry("CGG", "R"),  entry("AGG", "R"),  entry("GGG", "G")
    );
    public Protein(String id, String sequence) {
        this.id = id;
        this.sequence = sequence;
    }


    // Methoden die zu Variante 1 gehören:
    public static String codonToAminoAcidCodonTable(String codon) {
        return CODON_TABLE.get(codon);
    }

    public static String rnaToProteinCodonTable(String rna) {
        StringBuilder protein = new StringBuilder();
        for (int i = 0; i <= rna.length()-3; i += 3) {
            String codon = rna.substring(i, i+3);
            protein.append(codonToAminoAcidCodonTable(codon));
        }
        return protein.toString();
    }

    // Methoden die zu Variante 2 gehören:
    public static String codonToAminoAcidSwitch(String codon) {
        switch (codon.charAt(0)) {
            case 'G':
                switch (codon.charAt(1)) {
                    case 'G':
                        return "G";
```

```
                    case 'A':
                        switch (codon.charAt(2)) {
                            case 'G':
                            case 'A':
                                return "E";
                            case 'C':
                            case 'U':
                                return "D";
                        }
                    case 'C':
                        return "A";
                    case 'U':
                        return "V";
                }
            case 'A':
                switch (codon.charAt(1)) {
                    case 'G':
                        switch (codon.charAt(2)) {
                            case 'G':
                            case 'A':
                                return "R";
                            case 'C':
                            case 'U':
                                return "S";
                        }
                    case 'A':
                        switch (codon.charAt(2)) {
                            case 'G':
                            case 'A':
                                return "K";
                            case 'C':
                            case 'U':
                                return "N";
                        }
                    case 'C':
                        return "T";
                    case 'U':
                        switch (codon.charAt(2)) {
                            case 'G':
                                return "M";
                            case 'A':
                            case 'C':
                            case 'U':
                                return "I";
                        }
                }
            case 'C':
                switch (codon.charAt(1)) {
                    case 'G':
                        return "R";
                    case 'A':
                        switch (codon.charAt(2)) {
                            case 'G':
                            case 'A':
                                return "Q";
                            case 'C':
                            case 'U':
                                return "H";
```

```
                        }
                    case 'C':
                        return "P";
                    case 'U':
                        return "L";
                }
            case 'U':
                switch (codon.charAt(1)) {
                    case 'G':
                        switch (codon.charAt(2)) {
                            case 'G':
                                return "W";
                            case 'A':
                                return "";
                            case 'C':
                            case 'U':
                                return "C";
                        }
                    case 'A':
                        switch (codon.charAt(2)) {
                            case 'G':
                            case 'A':
                                return "";
                            case 'C':
                            case 'U':
                                return "Y";
                        }
                    case 'C':
                        return "S";
                    case 'U':
                        switch (codon.charAt(2)) {
                            case 'G':
                            case 'A':
                                return "L";
                            case 'C':
                            case 'U':
                                return "F";
                        }
                }
            default:
                // Keine sehr elegante Lösung, aber in diesem Fall ausreichend:
                return "?";
        }
    }

    public static String rnaToProteinSwitch(String rna) {
        StringBuilder protein = new StringBuilder();
        for (int i = 0; i <= rna.length()-3; i += 3) {
            String codon = rna.substring(i, i+3);
            protein.append(codonToAminoAcidSwitch(codon));
        }
        return protein.toString();
    }
}
```