TU Berlin, Fakultät IV
Blankertz/Röhr/Stahl
MAR 4-3 und MAR 5-1

Aufgabenblatt 1

Java Einführung I - Vererbung

Wichtige Ankündigungen

- **Erinnerung Prüfungsanmeldung**: (für die meisten: in QISPOS) Deadline ist am **25.5.2019**. Ohne Prüfungsanmeldung können Sie nicht an der Klausur teilnehmen und bekommen keine Prüfungsleistungen angerechnet.
- Registrieren Sie sich unter `http://hyades.robotics.tu-berlin.de/index.php`, wenn Sie das noch nicht getan haben.
- Das Vorlesungsmaterial, die Übungsblätter und die Vorlagen für die Hausaufgaben finden Sie unter `https://gitlab.tubit.tu-berlin.de/algodat-sose19/Material.git`.
- Alle Übungen sind in Einzelarbeit zu erledigen. Kopieren Sie niemals Code und geben Sie Code in keiner Form weiter. Die Hausaufgaben sind Teil Ihrer Prüfungsleistung. Finden wir ein Plagiat (wir verwenden Plagiatserkennungssoftware), führt das zum Nichtbestehen des Kurses für alle Beteiligten.

Abgabe (bis 30.04.2019 19:59 Uhr)

Die folgenden Dateien müssen für eine erfolgreiche Abgabe im git Ordner eingecheckt sein:

Geforderte Dateien:	
`Blatt01/src/CorticalNeuron.java`	Aufgabe 3.1
`Blatt01/src/Interneuron.java`	Aufgabe 3
`Blatt01/src/Photoreceptor.java`	Aufgabe 3
`Blatt01/src/Synapse.java`	Aufgabe 3.2
`Blatt01/src/Network.java`	Aufgabe 3

Als Abgabe wird jeweils nur die letzte Version im git gewertet.

Aufgabe 1: Laufzeitoptimierung (Tut)

Das Two-Sum-Problem ist ein typisches Problem, was Ihnen z.B. in einem Vorstellungsgespräch gestellt werden könnte:
Es sind ein Array und eine Zahl k gegeben, beide vom Typ `int`. Schreiben Sie eine Methode, welche prüft, ob es in dem Array ein Paar gibt, welches in der Summe k ergibt.

1.1 Geben Sie je ein Beispielarray an, für den diese Methode für $k = 3$ `true` oder `false` ausgibt.

1.2 Geben Sie die Brute-force Lösung des Problems und deren Laufzeit an.

1.3 Überlegen Sie sich eine schnellere Lösung, indem Sie den Array sortieren und dann von vorne und hinten gleichzeitig anfangen.

Aufgabe 2: OOP (Tut)

Diese Aufgabe soll Ihnen die Grundprinzipien der Objektorientierten Programmierung näher bringen.

2.1 Definieren Sie Klassen und Objekte.

2.2 Erstellen Sie eine Hierachie von *Vertebrata* (Wirbeltieren) am Beispiel von zwei Klassen von Wirbeltieren mit jeweils zwei Vertretern dieser Klassen.

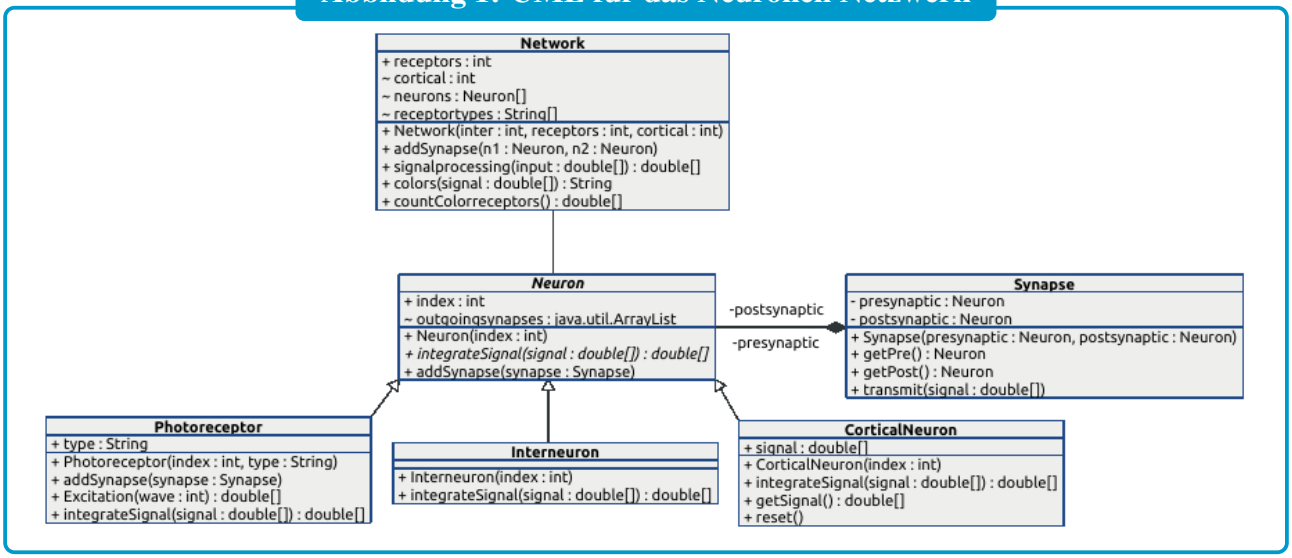
2.3 Wenn Sie nun diese Hierarchie implementieren mit `Vertrebrata` als abstrakte Klasse, wie sähe der Code dafür aus, wenn Sie in jeder Klasse nur den Konstruktor implementieren? Überlegen Sie sich Attribute, die Sie den Klassen geben könnten.

2.4 Wie erstellen Sie einen Array von `Vertebrata`? Können Sie unterschiedliche Tiere in diesen Array schreiben? Wenn ja, wie? Wie funktioniert Casting an diesem Beispiel?

Aufgabe 3: Vererbung (Hausaufgabe)

Sie werden ein Netzwerk von Neuronen implementieren, welches die Verarbeitung von Farben im Gehirn modelliert. Dafür werden Ihnen folgende Klassen zur Verfügung gestellt:

Abbildung 1: UML für das Neuronen Netzwerk



Hinweise:

- Gehen Sie alle Dateien, die Sie von uns bekommen, sorgfältig durch.
- Lesen Sie sich **kurz** in das Thema ein, d.h. in die Farbverarbeitung im Auge und die Weiterleitung bis zum Kortex. Wir haben den Prozess stark vereinfacht, trotzdem wird es Ihnen helfen die Programme besser zu verstehen.
- Testen Sie Ihre Implementation z.B. durch eine main-Methode.

- Lassen Sie sich Ihre Zwischenergebnisse zu Testzwecken ausgeben.
- **Code, der nicht kompiliert, wird mit** 0 **Punkten bewertet.**

3.1 Konstruktoren der Neuronen (15 Punkte)

Programmieren Sie gemäß den APIs die Konstruktoren der folgenden Klassen:

public class CorticalNeuron extends Neuron

`CorticalNeuron(int index)`	Erzeugt `CorticalNeuron` Objekt, ein Neuron mit gegebenem Index.

public class Interneuron extends Neuron

`public Interneuron(int index)`	Erzeugt `Interneuron` Objekt, ein Neuron mit gegebenem Index.

public class Photoreceptor extends Neuron

`public Photoreceptor(int index, String type)`	Erzeugt `Photoreceptor` Objekt, ein Neuron mit gegebenem Index und Typ.Wirft `RuntimeException`, wenn der Rezeptortyp nicht `green`, `blue` oder `red` ist.

3.2 Synapsen (10 Punkte)

Implementieren Sie die Methoden

```
public Neuron getPre()
public Neuron getPost()
```

in der Klasse `Synapse`, sowie die Methoden

```
public void addSynapse(Synapse synapse)
```

in `Photoreceptor` und

```
public void addSynapse(Neuron n1, Neuron n2)
```

in `Network`.
Hierbei gibt `getPre()` das präsynaptische Neuron zurück, `getPost()` das postsynaptische. `addSynapse` fügt eine Synapse zu dem präsynaptischen Neuron hinzu. Bitte beachten Sie, dass ein Fotoreceptor nicht mehr als eine (ausgehende) Synapse haben kann. Werfen Sie eine entsprechende `RuntimeException`, sollten mehr Synapsen hinzugefügt werden.

3.3 Netzwerk aufbauen (25 Punkte)

Implementieren Sie den Konstruktor der Klasse `Network`.

```
public Network(int inter, int receptors, int cortical)
```

Der Konstruktor bekommt jeweils die Anzahl der kortikalen Neuronen, der Interneuronen und der Fotorezeptoren übergeben. Man braucht alle drei Fotorezeptoren, um die Farben korrekt zu sehen. Werfen Sie also eine Exception, wenn weniger als drei Fotorezeptoren initiiert werden

sollen. Initiieren Sie dann die Fotorezeptoren so, dass möglichst gleich viele von jedem Typ vorhanden sind. (Die Modulo-Funktion wäre eine Möglichkeit.) Werfen Sie eine weitere Exception, falls weniger Interneuronen als Fotorezeptoren in dem Netzwerk wären.

Hinweis: Sortieren Sie die unterschiedlichen Neuronen in dem Array `neurons` so, dass sie stets wissen, welche Art von Neuron sich an welcher Stelle im Array befindet.

3.4 Signalverarbeitung im Interneuron (15 Punkte)

Implementieren Sie die Methode

```
public double[] integrateSignal(double [] signal)
```

in `Interneuron`. Die Methode bekommt ein synaptisches Signal und verteilt es auf alle nachgeschalteten Neuronen. Damit sich das Signal im Netzwerk nicht mit der Anzahl der Synapsen potenziert, wird das Signal gleichmäßig auf die ausgehenden Synapsen aufgeteilt.

3.5 Signalverarbeitung im Netzwerk (35 Punkte)

Implementieren Sie die Methode

```
public double[] signalprocessing(double [] input)
```

in `Network`. Hierfür benötigen Sie die `integrateSignal`-Methoden der Neuronen. Machen Sie sich klar, in welcher Reihenfolge das Signal verarbeitet wird und gehen Sie die Methoden der Reihenfolge nach durch. In dieser Methode müssen sie den Prozess anstoßen und das Ergebnis am Ende abrufen (`getsignal()`). Da nicht immer gleich viele Fotorezeptoren zur Verfügung stehen, müssen die abgerufenen Farben noch durch die Anzahl der Fotorezeptoren pro Farbe geteilt werden. Hierbei hilft die Funktion `countColorreceptors()`.

Hinweise:

- Das synaptische Signal ist dreidimensional, die Farben sind dabei in der Reihenfolge `blue`, `green`, `red`. `countColorreceptors()` gibt sie auch in dieser Reihenfolge zurück.
- Für das Testen des Netzwerkes können Sie dann das Ergebnis noch an die Methode `colors` übergeben. Diese gibt Ihnen dann die zu der Wellenlänge passende Farbe zurück. (Hier finden Sie eine Tabelle mit den zu erwartenden Werten: https://de.wikipedia.org/wiki/Licht)