TU Berlin, Fakultät IV
Brock/Páll/Bhatt/Röhr/Stahl
MAR 4-3 und MAR 5-1

Aufgabenblatt 1

Java Einführung I - Vererbung

Wichtige Ankündigungen

- **Erinnerung Prüfungsanmeldung**: (für die meisten: in Moses) Deadline ist am **15.05.2022**. 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://git.tu-berlin.de/algodat-sose22/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.

Abgabe (bis 15.05.2022 23:59 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 (Klausurvorbereitung)

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.

Die Lösung wird in dem Videotutorium 1 als Video bereitgestellt. Versuchen Sie erst selbst auf die Lösung zu kommen, aber mindestens sie danach selbst zu rekonstruieren.

Aufgabe 2: OOP (Klausurvorbereitung)

Diese Aufgabe soll Ihnen die Grundprinzipien der Objektorientierten Programmierung näher bringen, welche Sie auch in der Hausaufgabe brauchen. Schauen Sie sich das Videotutorium an und lösen Sie dabei die Aufgabenteile selbstständig, indem Sie die Videos pausieren. Sollte Ihnen das noch schwer fallen, schauen Sie sich die Videos komplett an und bearbeiten Sie die Aufgabenteile danach alleine, indem Sie umsetzen, was Sie gerade gelernt haben.

2.1 Definieren Sie Klassen und Objekte.

2.2 Implementieren Sie eine Klasse `Kegelrobbe` und testen Sie Ihre Implementation in einer `main`-Methode.

2.3 Erstellen Sie eine Hierachie von *Wirbeltieren* am Beispiel von zwei Klassen von Wirbeltieren mit jeweils zwei Vertretern dieser Klassen.

2.4 Wenn Sie nun diese Hierarchie implementieren mit `Wirbeltiere` 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. Welche dieser Attribute werden unter welchen Bedingungen vererbt?

2.5 Wie testen Sie die Funktionalität Ihrer Klassen? Wie erstellen Sie einen Array von `Vertebrata`? Können Sie unterschiedliche Tiere in diesen Array schreiben? Wenn ja, wie? Wie funktioniert Casting an diesem Beispiel?

2.6 Was ist Polymorphie? Implementieren Sie eine abstrakte Methode `essen()`.

Aufgabe 3: Vererbung (Hausaufgabe)

Sie werden ein Netzwerk von Neuronen implementieren, welches die Verarbeitung von Farben im Gehirn sehr vereinfacht modelliert.
Die Verarbeitung der Lichtwellen erfolgt in den Fotorezeptoren, die das Licht in ein neuronales Signal umwandeln und an die Interneuronen weiterleiten, welche wiederum, das Signal an die kortikalen Neuronen weitergeben. Dort kann das neuronale Signal dann als Farbe interpretiert werden. Die Weiterleitung erfolgt immer über Synapsen. Sie können sich die Synapsen als Verbindungsstücke zwischen den Neuronen vorstellen.

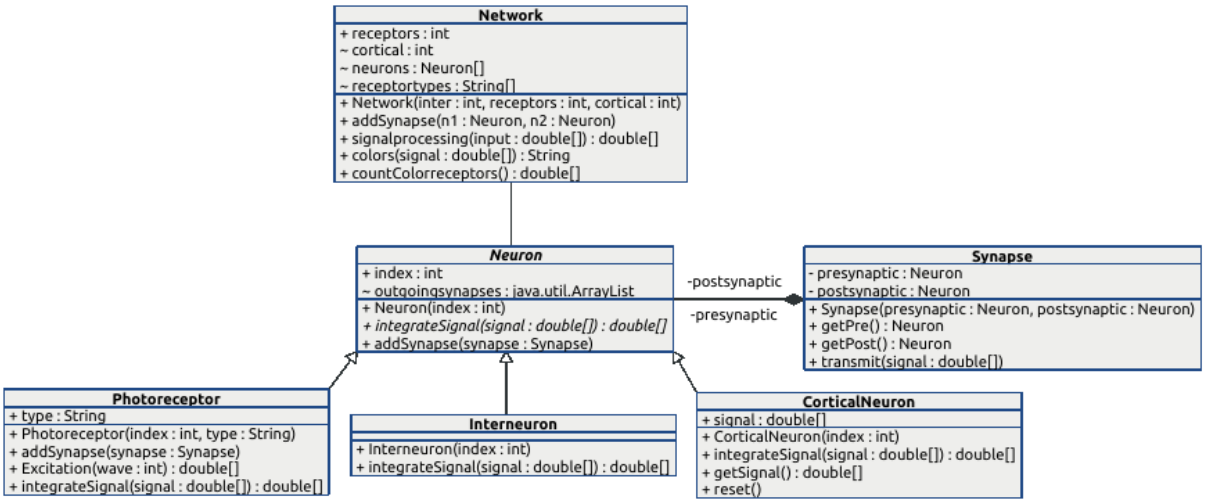
Hinweise:

- Nutzen Sie die Unterstützung die Ihnen in den Tutoriumssprechstunden angeboten wird.
- Gehen Sie alle Dateien, die Sie von uns bekommen, sorgfältig durch.
- Schauen Sie sich das Videotutorium an, darin bekommen Sie einige Hintergrundinformationen und Hinweise.
- In Ihren Konstruktoren sollten alle Attribute der jeweiligen Klasse initialisiert werden.
- Testen Sie Ihre Implementation z.B. durch main-Methoden der unterschiedlichen Klassen. Ein Beispiel ist in der Klasse `Network` auskommentiert.

- Lassen Sie sich Ihre Zwischenergebnisse zu Testzwecken ausgeben. Sie können den `index` der Neuronen nutzen, um diese eindeutig zuordbar auszugeben.
- **Zum Zeitaufwand dieser Aufgabe**: Sollten Sie für die Aufgabenteile 3.1.-3.3 länger als 50 min brauchen, um den Code zu schreiben und selbst zu testen, nutzen Sie die Unterstützungsmöglichkeiten, die Ihnen zur Verfügung stehen. Bearbeiten Sie außerdem die Aufgabe 2 dieses Blattes nochmal und arbeiten dann mit deren Lösung.
- **Code, der nicht kompiliert, wird mit** 0 **Punkten bewertet.**

Für die Aufgabe werden Ihnen folgende Klassen zur Verfügung gestellt:

Abbildung 1: UML für das Neuronen Netzwerk



3.1 Konstruktoren der Neuronen (24 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 (24 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. `Network.addSynapse` erstellt eine Synapse von dem `Neuron n1` zu `n2` und fügt sie dem präsynaptischen Neuron `n1` hinzu. `Photoreceptor.addSynapse()` fügt eine ausgehende Synapse einem Fotorezeptor hinzu. Bitte beachten Sie, dass ein Fotorezeptor nicht mehr als eine (ausgehende) Synapse haben kann. Werfen Sie eine entsprechende `RuntimeException`, sollten mehr Synapsen hinzugefügt werden. Sie können sich an der Klasse `Neuron` orientieren, von der die Klasse `Photoreceptor` erbt. Die Methode soll in der Unterklasse nur eine weitere Funktion erfüllen, nämlich das Werfen einer `RuntimeException`.

3.3 Netzwerk aufbauen (31 Punkte)

Implementieren Sie den Konstruktor der Klasse `Network`, der die Neuronen des Netzwerkes initialisiert:

```
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. Alle Neuronen sollten in dem Array `neurons` gespeichert werden.

Hinweis:

- Fügen Sie **keine** Synapsen in dem Konstruktor hinzu. Dies hätte zur Folge, dass nur eine feste Netzwerkstruktur möglich ist.
- 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. Lassen Sie keine Lücken (null) in dem Array.

3.4 Signalverarbeitung im Interneuron (21 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 (siehe `Photoreceptor.integrateSignal()` und `Synapse.transmit()`).

Damit sich das Signal im Netzwerk nicht mit der Anzahl der Synapsen potenziert, wird das Signal zu gleichen Teilen auf die ausgehenden Synapsen aufgeteilt.
Hinweis: Die Methode wird über den Rückgabewert getestet, auch wenn man diesen eigentlich nicht braucht. Folglich muss der Rückgabewert gesetzt werden. Der Rückgabewert ist das Signal, welches über die Synapsen weitergegeben wird. Das ist für alle ausgehenden Synapsen gleich.

3.5 Signalverarbeitung im Netzwerk (0 Punkte)

Für diese optionale Aufgabe gibt es keine Punkte; sie ist für Ihr eigenes Lernen gedacht. Sie erhalten aber trotzdem ein Feedback von den Unit Tests.

Implementieren Sie die Methode

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

in `Network`. Hierfür benötigen Sie die `integrateSignal`-Methoden der Neuronen. Machen Sie sich klar, wie das Signal in Ihrem Netzwerk verarbeitet wird und was davon bereits automatisch passiert, durch den Code, den Sie in den vorherigen Aufgabenteilen geschrieben haben. Den Prozess stoßen Sie an, indem Sie den Input an die Fotorezeptoren übergeben und am Ende rufen Sie das Ergebnis ab (`CorticalNeuron.getsignal()`) und addieren es für alle kortikalen Neuronen auf. 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:

- Der Input ist ein Array beliebiger Länge, welches Wellenlängen in nm enthält (Licht in verschiedenen Farben). Es reicht ein einelementiges Array zu nehmen, bei mehreren Elementen werden Farben gemischt. Alle Fotorezeptoren erhalten den gleichen Input und verarbeiten dann ihren Farbanteil des Lichts (in der vorgegebenen `integrateSignal`).
- Das neuronale Signal ist dreielementig, die Farben sind dabei in der Reihenfolge `blue`, `green`, `red`. `countColorreceptors()` gibt die Anzahl auch in dieser Reihenfolge zurück.
- Für das Testen des Netzwerkes in der main-Methode müssen Sie Synapsen hinzufügen. Dabei muss folgende Struktur beachtet werden: Auf Fotorezeptoren folgen Interneuronen in der Signalverarbeitung, auf Interneuronen können Interneuronen und kortikale Neuronen folgen. Rückverbindungen, die im Netzwerk einen gerichteten Kreis entstehen lassen, sind nicht zulässig.
- Um den Programmablauf zu verfolgen, können Sie z.B. in den `integrateSignal`-Methoden der Neuronen mit `Arrays.toString()` die Variable `signal` ausgeben oder alternativ den Debugger nutzen.
- 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)
- Wenn sie einem Netzwerk (z.B. in der `main`) mehrere Farben hintereinander geben, müssen Sie vor jeder neuen Farbe die Methode `reset()` in `CorticalNeuron` aufrufen für alle kortikalen Neuronen, weil sonst das Ergebnis verfälscht wird.

Was Sie nach diesem Blatt wissen sollten:

- was Objekte und Klassen sind.
- wie man Konstruktoren schreibt, Objekte erzeugt und Methoden auf diesen Objekten aufruft.
- wie Sie for-Schleifen und if-Bedingungen verwenden.
- was getter- und setter-Methoden sind und wie man sie schreibt.
- wie Sie Arrays von Objekten erstellen und über diese iterieren.
- was eine abstrakte Klasse ist und wie Klassen erben.
- wie Sie eine main-Methode schreiben und ausführen.
- wie Sie Ihre geschriebenen Methoden selbst ausprobieren und Kontrollvariablen ausgeben.
- wie Sie Exceptions werfen.

Es ist nicht schlimm, wenn Sie noch nicht hinter all diese Punkte einen Haken setzen können, Sie werden in den nächsten 2 Wochen noch Zeit und weitere Aufgaben haben, um diese Fähigkeiten zu vertiefen. Allerdings sollten Sie anstreben, diese Fähigkeiten in den nächsten Wochen zu erlernen, da die Aufgaben danach Algorithmen behandeln werden. Sie müssen auch nicht in allen Übungsblättern 100 Punkte erreichen, solltest aber versuchen, mehr als 50 Punkte in jedem Übungsblatt zu erreichen und die beiden Möglichkeiten der angebotenen 100 Bonuspunkte in Betracht ziehen: 10 pro Übungsblatt oder 100 für ein ausgewähltes Übungsblatt.