Aufgabe 5.1 Obstbaum

Schreiben Sie ein Programm, das einen Obstbaum simuliert.

- Ein Obstbaum besteht aus Wurzeln, Ästen, Blättern und Früchten.
- Wurzeln haben eine Tiefe und können Wasser an den Baum liefern. Jede Wurzel liefert eine Einheit Wasser.
- Äste haben einen Länge, tragen Blätter sowie Früchte und können wachsen.
- Blätter haben eine Farbe, die sich ändern kann.
- Früchte haben ein Gewicht und können reifen.
- Der Obstbaum selber hat eine Sorte, eine Höhe, ein Alter und eine Menge an Wasser. Diese ist entscheidend, ob er etwas tun kann oder nicht. Der Baum kann Wasser aufnehmen, Ästen wachsen lassen, Farbe der Blätter verändern und Früchte reifen lassen. Jede Aktion verbraucht 5 Einheiten Wasser, bis auf Wasser aufnehmen.
- Alle Teile eines Baums können ausgegeben werden
- Sie können den Baum mit 5 Ästen, 5 Früchten/5 Blättern pro Ast und 5 Wurzeln initialisieren.
- Der Einfachheit halber wird eine Aktion immer für alle Objekte ausgeführt. (Alle Früchte reifen auf einmal)

Aufgabe 5.2 Studentenverwaltung

Schreiben Sie ein Java-Programm, welches eine Studentenverwaltung simuliert.

Dieses Programm soll folgende Funktionalitäten besitzen:

- Anlegen eines neuen Studenten
- Bearbeiten eines Studenten
- Kursnoten eines Studenten eintragen
- Einzelinformationen eines Studenten ausgeben
- Übersicht über alle Informationen ausgeben

Folgendes ist dabei zu beachten.

- Alle Studenten gehören einer Universität/Hochschule an
- An dieser Hochschule werden verschiedene Studiengänge angeboten
- Jeder Studiengang hat eine bestimmte Anzahl an Kursen
- Ein Kurs hat die Attribute Credits, Multiplikator, Note und Name
- Ein Student hat die Attribute Name, Matrikelnummer (nummerisch und 8-stellig) und E-Mail
- Jedem Studenten müssen der Studiengang und die zugehörigen Kurse zugeordnet sein

Aufgabe 5.2.1 (Schematisches Klassendiagramm)

Zeichnen Sie zunächst ein Klassendiagramm mit allen Klassen, Methoden und Attributen. Eine Assoziation beschreibt die Beziehung zwischen den einzelnen Klassen, tragen Sie diese auch in Ihr Diagramm ein.

Aufgabe 5.2.2 (Grunddaten anlegen)

In unserem Beispiel, werden die Daten der Hochschule, der Studiengänge und der Kurse beim Starten des Programmes geladen und können nicht verändert werden.

Initialisieren Sie Ihr Programm mit folgenden Daten:

- Name der Hochschule: OTH Regensburg
- Studiengänge mit Kursauswahl. Die Anzahl der Credits und des Multiplikators ist frei wählbar. (Kann auch vom Modulhandbuch entnommen werden)

    1. Wirtschaftsinformatik (Bachelor)
        1. Programmieren 1
        2. Programmieren 2
        3. BWL 1
        4. BWL 2
    2. Wirtschaftsinformatik (Master)
        1. Moderne Datenbankkonzepte
        2. Business Consulting
    3. Allgemeine Informatik (Bachelor)
        1. Programmieren 1
        2. Programmieren 2

Beispiel zum Befüllen einer Liste:

```
List<String> list = new ArrayList<String>();
list.add("item");
```

Aufgabe 5.2.3 (Anlegen und Bearbeiten der Studenten)

Es soll die Möglichkeit geschaffen werden, über die Konsole neuen Studenten anzulegen, diese zu bearbeiten und Kursnoten einzugeben.

Hier ein Beispiel zur Neuanlage:

```
Name:
Mustermann
Vorname:
Max
```

```
Matrikelnummer:
1234567
E-Mail:
max.mustermann@st.oth-regensburg.de
1: Allgemeine Informatik (Bachelor)
2: Wirtschaftsinformatik (Bachelor)
3: Wirtschaftsinformatik (Master)
Wählen Sie einen Studiengang aus:
2
```

Hier ein Beispiel zum Noteneingabe:

```
1: 1234567[Matrikelnummer]
2: 7654321[Matrikelnummer]
Wählen Sie einen Studenten:
1
1: Programmieren 1
2: Programmieren 2
3: BWL 1
4: BWL 2
Wählen Sie einen Kurs:
3
Bitte Note eingeben:
2
```

Aufgabe 5.2.4 (Ausgabe der Daten)

Um einen Übersicht über die Daten zu erhalten, soll noch eine Konsolenübersicht über alle Daten der Studierenden entwickelt werden.

Dabei soll sowohl die Möglichkeit alle Informationen inkl. Notendurchschnitt (Beachtung der Credit-Anzahl und des Multiplikators) eines ausgewählten Studierenden zu erhalten, aber auch eine komplette Übersicht über alle Studenten der Hochschule. (Zur Vereinfachung können Kurse, welche noch nicht absolviert wurden, einfach mit der Note 0 bewertet werden.)

Hier ein Beispiel:

```
Name des Studierenden: Max Mustermann
(angestrebter)  Abschluss:  Bachelor
Fach: Wirtschaftsinformatik
Matrikelnummer: 1234567
E-Mail: max.musterman@st.oth-regensburg.de
 -- Pruefungen --
  Name: Programmieren 1
  Credit: 8
  Multiplikator: 1
  Note:       2,00
 -- --
  Name: Programmieren 2
  Credit: 8
```

```
  Multiplikator: 2
  Note:       0,00
 -- --
  Name: BWL 1
  Credit: 8
  Multiplikator: 2
  Note:       0,00
 -- --
  Name: BWL 2
  Credit: 8
  Multiplikator: 2
  Note:       0,00
 -- --
Notendurchschnitt: 2.00
---------
Name des Studierenden: Isabell Musterfrau
(angestrebter)  Abschluss:  Master
Fach: Wirtschaftsinformatik
Matrikelnummer: 7654321
E-Mail: isabell.musterfrau@st.oth-regensburg.de
 -- Pruefungen --
  Name: Moderne Datenbankkonzepte
  Credit: 8
  Multiplikator: 2
  Note:       3,00
 -- --
  Name: Business Consulting
  Credit: 5
  Multiplikator: 2
  Note:       1,00
 -- --

Notendurchschnitt: 2.23
---------
```

Aufgabe 5.3 - Artikelverwaltung

Aufgabe 5.3.1

Schreiben Sie ein Programm zur Verwaltung von Artikeln und arbeiten Sie dabei mit Klassen und Objekten.

Ein Artikel hat eine Artikelnummer, eine Bezeichnung, eine Farbe und einen Preis.

Eine Farbe ist festgelegt durch die drei Grundfarben rot, grün und blau (RGB-Farbraum), deren jeweilige Anteile am Farbton in Form von Integer-Zahlen dargestellt werden können.

Der Preis des Artikels soll auch die Bezeichnung der Währung umfassen.

Fügen Sie im Code beliebig viele Artikel in die Artikelliste hinzu. Geben Sie am Ende eine Liste aller erfassten Artikel und deren Attribute aus. Verwenden Sie für die Artikelliste den Typ „Arraylist“ wie folgt:

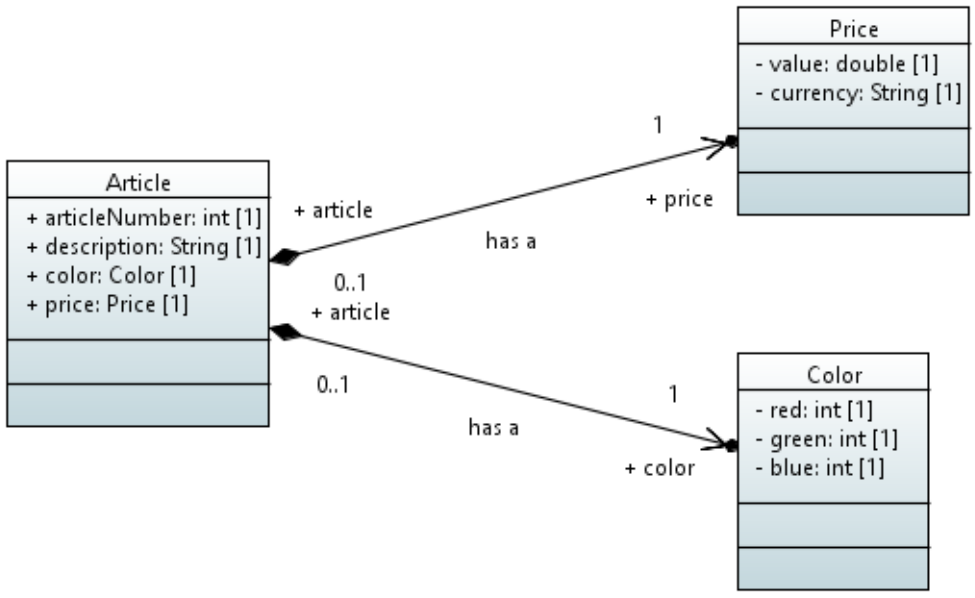
```
import java.util.ArrayList;
import java.util.List;


List<Article> lstArticles = new ArrayList<Article>();
Article article1 = new Article();

//Add item to lstArticles
lstArticles.add(article1);
```

(Zusatz) Programmieren Sie eine Erfassungsmaske (in der Konsole/Command-Line), welche es dem Benutzer erlaubt, beliebig viele Artikel in die Artikelliste hinzuzufügen. Möchte der Benutzer keine weiteren Artikel mehr hinzufügen, gibt das Programm eine Liste aller erfassten Artikel (mit Bezeichnung, Artikelnummer, Preis, Währung und Farbe) aus.

Das schematische Klassendiagramm sieht wie folgt aus:



Aufgabe 5.3.2

Erweitern Sie das bestehende Programm. Suchen Sie einen bestimmten Artikel anhand seiner Artikelnummer und geben Sie ihn aus. Sollte der Artikel nicht in Ihrer Liste vorhanden sein, muss eine Meldung ausgegeben werden, beispielsweise „Article not found“.

Aufgabe 5.3.3

Erweitern Sie nun ihr Programm so, dass alle Artikel mit einem bestimmten Preis ausgegeben werden, z.B. der Preis soll größer 100€ sein. Hier ist wiederum auf die Benutzerführung zu achten. Wird der Artikel nicht gefunden, soll eine Meldung erscheinen, z.B. „No articles found“.

Aufgabe 5.3.4 [Zusatzaufgabe]

Erweitern Sie ihr Programm um die Klasse Warenkorb. In diesem Warenkorb können beliebig viele Artikel gespeichert werden. Zusätzlich soll es noch folgende Funktionen haben:

- int getSize(): Gibt die Anzahl der beinhalteten Artikel zurück
- void addArticle(Article a): Fügt einen Artikel der Liste hinzu
- List<Article> getArticles(): Gibt alle Artikel als Liste zurück
- Boolean remove(int articleNumber): Löscht einen Artikel aus der Liste
- void print(): Gibt alle Artikel, deren Preise und Gesamtpreis aller Artikel auf der Console aus
- int getNewArtNr(): Generiert eine neue Artikelnummer und gibt diese zurück

Aufgabe 5.4 - Schneckenrennen

In dieser Übung geht es um die Simulation eines Schneckenrennens. Es gibt Rennschnecken, das Rennen an sich und ein Wettbüro.

Die Rennschnecke

Eine Rennschnecke kann mit einer bestimmten Geschwindigkeit kriechen und gehört einer Rasse an. Die Geschwindigkeit ist nur durch die Strecke definiert, welche eine Schnecke in einer Zeiteinheit zurücklegen kann. Weiterhin besitzt sie eine Startnummer und weiß zu jedem Zeitpunkt über ihre zurückgelegte Strecke Bescheid. Jeder Schnecke soll sich ausgeben können, z.B. mit Hilfe der „toString()“-Methode

Das Rennen

Das Rennen beinhaltet Informationen über die Anzahl der teilnehmenden Schnecken, die zurückgelegte Strecke und einen Namen, der das Rennen identifiziert. Weiterhin können die Schnecken zum Kriechen veranlasst und der Gewinner ermittelt werden. Wird eine Schnecke zum Kriechen gebracht, so legt sie eine zufällige Strecke zurück. Die zurückgelegte Strecke kann zwischen 0 und dem Wert der Geschwindigkeit hinterlegt wurde variieren. Ein Rennen läuft so lange bis alle Rennschnecken im Ziel angekommen sind. Erreicht eine Schnecke das Ziel, so wird der erreichte Platz der Rennschnecke zugewiesen. Die Informationen über ein Rennen sollen zusätzlich abgefragt und evtl. auf der Konsole ausgegeben werden können. Des Weiteren können Schnecken sowohl einem Rennen hinzugefügt als auch entfernt werden.

Das Wettbüro

Das Wettbüro nimmt Rennwetten entgegen und organisiert das Rennen. Das Wettbüro weiß für welches Rennen es Wetten angenommen hat. Eine Wette gehört immer zu einem bestimmten Rennen und einer bestimmten Rennschnecke, somit können keine Wetten abgegeben werden, die für mehrere Rennen oder Rennschnecken gelten. Die Ergebnisse der Wetten sollen auch ausgegeben werden können.

Sollten weitere Attribute und/oder Methoden benötigt werden, so ergänzen Sie diese.

Aufgabe 5.4.1

Zeichnen Sie ein vereinfachtes Klassendiagram mit den jeweiligen Attributen und Methoden. Getter/Setter können in der Zeichnung vernachlässigt werden.

Aufgabe 5.4.2

Implementieren Sie die Klassen und erstellen Sie ein Testprogramm.

Aufgabe 5.5 - Vier gewinnt

Programmieren Sie ein Vier-Gewinnt spiel, das von zwei Spielern mit Hilfe einer Konsolen-Eingabe bedient werden kann. Arbeiten Sie dabei mit Klassen und Methoden („objektorientiert“).

Das Spielfeld von Vier-Gewinnt entspricht einer Matrix mit 7 x 6 Feldern. Jeder der beiden Spieler hat Spielsteine in einer bestimmten Farbe (Spieler 1: rot, Spieler 2: gelb). Die

Spielsteine werden abwechselnd von oben in das Spielfeld eingeworfen. Es gewinnt der Spieler, dem es gelingt, als Erster vier Spielsteine in horizontaler, vertikaler oder diagonaler unmittelbarer Nachbarschaft anzuordnen. Weitere Details zum Spiel finden sie hier: http://de.wikipedia.org/wiki/Vier_gewinnt

Programmieren Sie das Spiel mit Hilfe einer einfachen ANSI-Graphik in der Konsole, wie beispielsweise:

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
-----------------------------
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
| O |   | X |   |   |   |   |
| O |   | X |   |   |   |   |
| x | X | O |   |   |   |   |
| x | O | O |   |   |   |   |
-----------------------------
Spieler 1 (O) wirft in Spalte:
```

Die Spieler werden abwechselnd gefragt, in welche Spalte der nächste Spielstein geworfen werden soll. Die Spalten sind von links nach rechts durchnummeriert. Wählt der Spieler 1, dessen Spielsteine mit O dargestellt werden, im obigen Beispiel die Spalte 5, so wird im nächsten Spielschritt das folgende Spielfeld angezeigt:

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
-----------------------------
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
| O |   | X |   |   |   |   |
| O |   | X |   |   |   |   |
| x | X | O |   |   |   |   |
| x | O | O |   | O |   |   |
-----------------------------

Spieler 2 (X) wirft in Spalte:
```

Nun wird Spieler 2, dessen Spielsteine mit X dargestellt werden, aufgefordert, eine Spalte zu wählen.

Geben Sie eine Meldung aus, sobald einer der beiden Spieler gewonnen hat. Zeigen Sie an, welche Spielsteine für den Gewinn verantwortlich sind (Bemerkung: Dies können mehr als 4 Steine sein). Prüfen Sie, ob eine Spalte noch genügend Platz hat. Geben Sie auch eine Meldung aus, wenn das Spiel ohne Gewinner endet.

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
-----------------------------
```

```
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
| O |   | X |   |   |   |   |
| O |   | X |   |   |   |   |
| x | X | O |   |   |   |   |
| x | O | O | O | O |   |   |
-----------------------------
Spieler 1 (O) wirft in Spalte: 4
Spieler 1 (O) hat gewonnen (Steine: 1/2, 1/3, 1/4, 1/5).
Wollen Sie weiterspielen (J/N)?
```