Aufgabe 1 für Studierende der Informatik (AI)

In diesem Semester werden Sie eine Anwendung implementieren, welche den Aufbau einer Fabrik und deren Inbetriebnahme simuliert.

Pflichtaufgabe 1 umfasst die grundlegenden Datenstrukturen und Auswertungs-Algorithmen
Pflichtaufgabe 2 wird die Umsetzung einer graphischen Oberfläche sein
Pflichtaufgabe 3 wird die Verwaltung der Daten flexibilisieren (Verkettete Liste, Suchen, Sortieren, ...)

Beratungstermine für diese Aufgabe sind am 1. und 8. Juni 2017

Schreiben Sie für alle Methoden eine kurze Inline-Dokumentation im JavaDocs Format!
Kapseln Sie Eigenschaften so gut wie möglich mit private oder protected!

Pflichtaufgabe 1:

a) Erstellen Sie eine Datenstruktur `Produkt`, die ein Produkt beschreibt.
   - Diese benötigt folgende `private` Eigenschaften:
     - `double kosten`
     - `String name`
     - `double verkaufswert`
   - Weiterhin benötigt diese Datenstruktur folgende Methoden:
     - Einen Konstruktor:
       `Produkt(String name, double kosten, double verkaufswert)`
     - Getter - Methoden für die `private` Variablen

b) Erstellen Sie die Datenstruktur `Warenspeicher`, in der die erzeugten Produkte einer Fabrik gespeichert und verwaltet werden.
   - Dafür benötigen Sie mindestens diese `private` Eigenschaft:
     - `ArrayList<Produkt> produkte;`
   - Erstellen Sie einen Konstruktor ohne Übergabeparameter, in dem die ArrayList initialisiert wird
   - Erstellen Sie folgende Methoden, um die Produkte zu verwalten:
     - `fuegeProduktHinzu(Produkt produkt)`
       →Diese Methode fügt ein neues Produkt der ArrayList `produkte` hinzu.
     - `anzahlImSpeicher(String gesuchtesProdukt)`
       → Dieser Methode wird der Name eines Produkts übergeben. Als return - Wert wird die Anzahl übereinstimmender Produkte im Warenspeicher ausgegeben.
     - `entferneProdukt(String unerwuenschtesProdukt)`
       → Diese Methode löscht ein einzelnes Produkt mit dem übergebenen Namen aus dem Warenspeicher.
     - `entferneProdukt(String unerwuenschtesProdukt, int anzahl)`
       → Diese Methode ermöglicht es, mehrere Produkte mit dem gleichen Namen zu löschen. Achten Sie darauf, dass die zu löschende Anzahl an Produkten nicht größer ist als die Anzahl im Warenspeicher. (Hier können bereits erstellte Methoden wieder verwendet werden)
     - `warenVerkaufen()`
       →Diese Methode simuliert den Verkauf aller Produkte im Warenspeicher. Hierzu wird der gesamte Verkaufswert aller Produkte berechnet und zurückgegeben. Achten Sie darauf, dass nach der Berechnung die ArrayList geleert werden muss.

c) Erstellen Sie die Datenstrukturen `Fabrik` und `Maschine`. Eine Fabrik besitzt ein Guthaben, welches für den Kauf und die Inbetriebnahme verschiedener Maschinen zur Verfügung steht. Da die Maschinen auf den Warenspeicher der Fabrik zugreifen müssen, um ihre Erzeugnisse zu vermerken, benötigen diese eine Referenz auf die zugehörige Fabrik. Da beide Klassen die Funktionen der jeweiligen anderen Klasse verwenden, sollten Sie zunächst deren "Skelette" erstellen, bevor Sie auf die Implementierung der Logik eingehen.

- Die `Fabrik` besitzt die `private` Eigenschaften:
  - `Warenspeicher warenspeicher`
  - `double guthaben`
  - `double testguthaben` (wird später für die Simulation benötigt)
  - `String name`
  - `ArrayList<Maschine> maschinen`
- Definieren Sie einen Konstruktor und Getter-Methoden für `testguthaben`, `warenspeicher` und `name`.
- Sie benötigen außerdem folgende Methoden:
  - `fuegeMaschineHinzu(Maschine maschine)`
    →Diese Methode fügt eine neue Maschine der `ArrayList maschinen` hinzu. Vorher muss sie jedoch die `setFabrik` –Methode der `maschine` aufrufen und sich selber übergeben.
  - `entferneMaschine(int index)`
    →Entfernt die Maschine an der Stelle `index` aus der `ArrayList maschinen`.
  - `testguthabenReduzieren(double kosten)`
    →Diese Methode reduziert das Testguthaben um die Kosten.
  - `firmaTesten(int rundenanzahl)`
    →Diese Methode simuliert den Aufbau und die Inbetriebnahme einer Fabrik. Hierzu muss das Testguthaben dem Guthaben gleichgesetzt und anschließend um die Kosten aller Maschinen reduziert werden. Nun folgt der Produktionsprozess indem jede Maschine der Fabrik ausgeführt wird. Anschließend wird der Verkauf durchgeführt und der Erlös wird dem Testguthaben wieder gutgeschrieben. Die Wiederholung des Produktionsprozesses soll der `rundenanzahl` entsprechen. Abschießend soll das verbliebene Testguthaben zurückgegeben werden.

    Tipp: Sie sollten auch einmal auf Papier testen, ob ihr Fabrikaufbau funktioniert.

- Die Klasse `Maschine` besitzt die `private` Eigenschaften:
  - `String name`
  - `double kosten`
  - `Fabrik fabrik`
- Definieren Sie einen Konstruktor, welcher `name` und `kosten` entgegennimmt und speichert. Die Fabrik wird hier zunächst mit dem null - Wert initiiert. Getter-Methoden werden für `name` und `kosten` und eine setter –Methode für `fabrik` benötigt.
- Sie benötigen außerdem eine Methode, um die Maschine zu starten:
  - `public void maschineStarten()`.
    → Diese Methode soll in der Konsole Auskunft darüber geben, dass die Maschine gestartet wurde.
    Bsp. Ausgabe auf der Konsole:
    `Apfelpresse hat die Arbeit aufgenommen.`

d) Erstellen Sie die Unterklasse `Erzeuger` von der Oberklasse `Maschine`.
→`Erzeuger` ist eine Maschine die zur Herstellung von Produkten dient.
Sorgen Sie dafür, dass die Unterklasse auf die Eigenschaften der Oberklasse zugreifen kann.

- `class Erzeuger` mit der erweiterten Eigenschaft:
  - `Produkt erzeugnis` (das Produkt das von dieser Maschine hergestellt wird)
- Und der Methode zur Erstellung von Produkten:
  - `void produktErzeugen()`
    → Diese Methode überprüft zunächst, ob das Testguthaben der Fabrik für die Herstellung des Produkts ausreichend ist. Ist das der Fall, so wird das `erzeugnis` dem Warenspeicher der Fabrik hinzugefügt. Andernfalls wird eine Fehlermeldung auf der Konsole ausgegeben.

    Bsp. Ausgabe auf der Konsole:
    ```
    Apfel konnte aufgrund fehlenden Guthabens nicht erzeugt werden.
    ```

  - Überschreiben Sie zusätzlich die Methode `maschineStarten()`der Oberklasse so, dass zusätzlich ein Produkt erzeugt wird.

    Bsp. Ausgabe auf der Konsole:

    ```
    Apfelpflücker hat die Arbeit aufgenommen.
    Apfel wurde produziert.
    ```

e) Erstellen Sie die Unterklasse `Verwerter` von der Oberklasse `Erzeuger`.

- `Verwerter` ist eine Maschine, die zur Verarbeitung von Produkten dient, um so neue Produkte zu erstellen. (z.B.: aus Äpfeln wird Apfelsaft)
- `class Verwerte` mit folgenden Erweiterungen:
  - `Produkt abhaengigkeit` (Produkt, das zur Verarbeitung benötigt wird)
  - `int anzahl` (des benötigten Produkts zur Verarbeitung z.B.: **3** Äpfel))
- Und der Methode:
  - `boolean checkAbhaengigkeit()`
    → Methode zur Überprüfung, ob genügend Produkte für die Verarbeitung vorhanden sind.
- Überschreiben Sie zudem die Methode `produktErzeugen()` der Oberklasse `Erzeuger`:
  - `produktErzeugen()`
    → Diese Methode zur Erzeugung von Produkten soll so erweitert werden, dass die zu verarbeitenden Produkte aus dem Warenspeicher entnommen werden. **Achtung**: Es müssen genügend erzeugte Produkte vorhanden sein, bevor sie verarbeitet werden können. Außerdem benötigen Sie genügend Guthaben, um die Verarbeitung auszuführen.

  Bsp. Ausgabe auf der Konsole:

  ```
  Apfelpresse hat die Arbeit aufgenommen.
  Apfelsaft wurde produziert.
  ```

f) Erstellen Sie die Klasse App mit einer `Main`-Methode.

- Erstellen Sie innerhalb dieser Methode eine beliebige Fabrik und drei beliebige Produkte, die in der Fabrik produziert werden. Außerdem erstellen Sie noch Maschinen (`Erzeuger` und `Verwerter`), die Sie zur Herstellung (`Erzeuger`) und Verarbeitung (`Verwerter`) Ihrer Produkte benötigen und fügen Sie diese in Ihre Fabrik ein. Starten Sie anschließend die Simulation.

Mehr Infos im Netz
ArrayLists: http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
Java-Dokumentation: http://docs.oracle.com/javase/8/docs/api/overview-summary.html
Java-Docs: https://de.wikipedia.org/wiki/Javadoc

VIEL ERFOLG!