Lehrstuhl Computergrafik
Prof. Dr.-Ing. Hendrik P.A. Lensch
Patrick Wieschollek, Sebastian Herholz, Fabian Groh

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Informatik II

Übungsblatt 5

Ausgabe: Mi, 25. Mai 2016 - **Abgabe: Di, 31. Mai 2016 - 12:00 Uhr**

Aufgaben

5.1 Naive Sorting (1 + 2 + 1 + 3 + 1 = 8 Punkte)

In dieser Aufgabe soll der Umgang mit Arrays anhand naiver Sortierung gelernt werden. Schreiben Sie dazu ein einfaches Java-Programm das Sie in die Vorlage `NaiveSorting.java` einfügen.

a) Die Klasse `NaiveSorting` soll ein Integer-Array mit dem Namen `elements` verwalten. Das Array soll eine maximal Kapazität von 10 Elementen haben. **Hinweis:** Vermutlich brauchen Sie zum richtigen lösen dieser Aufgabe weitere Attribute!

b) Beginnend soll Ihr Programm so lange wie möglich `int`-Zahlen in das Array speichern. Implementieren Sie hierfür eine Methode `addNextElement`, welche als Argument einen Integer bekommt und diesen (wenn möglich, d.h. wenn das Array noch nicht voll ist) in das Array speichert.

Tipp: Denken Sie daran in der Main Methode ein NaiveSorting - Objekt zu initialisieren, auf dessen Array Sie dann arbeiten können.

c) Implementieren Sie die Methode `getMinimum`. Durchsuchen Sie hier das Array `elements` nach dem kleinsten Element und geben Sie dieses als Rückgabewert der Methode zurück.

d) Anschließend sollen die Werte in dem Array aufsteigend sortiert werden. Gehen Sie dabei so vor, dass jeweils das minimale Element gesucht wird und dieses Element dann an die entsprechende Position getauscht wird. Schreiben Sie dafür die öffentliche Methode `sortArray`, welche weder Argumente noch einen Rückgabewert hat.

Tipp: Überlegen Sie sich wie Sie verhindern, dass nachdem das kleinste Element gefunden und an die neue Stelle getauscht wurde, dieses im nächsten Durchlauf nochmal gefunden wird.

e) Finden Sie nun das größte Element und geben Sie dieses aus. Dafür schreiben Sie die Methode `getSortedMax`, welche als Rückgabewert das größte Element eines vorher sortierten Arrays zurück gibt. **Hinweis:** Sie können annehmen, dass das Array beim Aufruf der Methode bereits sortiert worden ist.

5.2 Spülmaschine (3 + 2 + 2 + 2 + 2 + 3 + 3 + 2 + 3 = 22 Punkte)

In dieser Aufgabe sollen Sie erste Erfahrungen mit der objektorientierten Programmierung sammeln. Dazu werden Sie zwei Klassen entwickeln: Eine Klasse `Dish.java` und eine Hauptklasse `Dishwasher.java` in der dann Objekte vom Typ `Dish` erzeugt werden.
In eine Spülmaschine kann man verschiedenes Geschirr mit verschiedenem Namen (z.B. "plate") einsortieren. Außerdem kann das Geschirr jeweils unterschiedlich schmutzig sein. Wir verwenden dazu einen `dirtyLevel`(float) der sich auf einer Skala von 0.0 für sauber bis 1.0 für komplett schmutzig bewegt. Erstellen Sie dazu die beiden Klassen:

a) Dish

i) Implementieren Sie die Klasse Dish, welche als Felder den Namen eines Geschirrstücks (`name:String`) ("plate", "cutlery", "cup", "pot"), seinen Schmutzigkeitgrad (`dirtyLevel:float`) und seine Größe (`size:int`) haben soll. Diese sollen alle als private deklariert werden (*Datenkapselung*).

ii) Schreiben Sie einen Konstruktor der den Namen des Geschirrstücks übergeben bekommt. Der Konstruktor soll eine privaten (statische) Funktion `getSizeFromName` (weiterhin in der Klasse `Dish`) aufrufen um die Größe des Objektes aufgrund seines Namens zu bestimmen.

- Name: "cutlery" → size: 1
- Name: "plate" → size: 2
- Name: "cup" → size: 2
- Name: "pot" → size: 5

Ein neues Geschirr ist bei seiner Erstellung sauber.

iii) Erzeugen Sie nun die "Getter"-Methoden `getSize`, `getName` und `getDirtLevel`, die Alle als public deklariert sein sollen und jeweils die `size`, den `name` und `dirtyLevel` des jeweiligen Objekts zurückgeben sollen.

iv) Zusätzlich benötigt die Klasse `Dish` die "Setter"-Methode `setDirtyLevel`. Achten Sie darauf, dass die Methode Werte außerhalb des erlaubten Bereiches in den erlaubten "cropped".

v) Implementieren Sie in der Klasse `Dish` die Funktion `public void reduceDirt()`. Diese Funktion soll den Schmutzigkeitsgrad des Objekts um 0.3 verringern. Achten Sie weiterhin darauf, dass sie innerhalb des erlaubten Wertebereichs bleiben.

b) `Dishwasher`

i) Implementieren Sie in `Dishwasher.java` eine weitere Klasse `Dishwasher`, mit den Feldern `capacity:integer`, einem Dish-Array `dishesInside` und einem Integer `occupied`. Verfolgen Sie auch hier das Prinzip der Datenkapselung. Schreiben Sie einen Konstruktor der die Kapazität der Maschine als Eingabe erhält sowie die "Getter"-Methoden für die Attribute.

ii) Weiter benötigt die Klasse die öffentliche Methode `addDish` die als Argument ein Dish-Element erhält und als boolean zurück gibt, ob das Element erfolgreich zum Array `dishesInside` hinzugefügt werden konnte. Füllen Sie darin ihre Spülmaschine nur, solange sie noch nicht ihre Kapazität erreicht hat.

iii) Erstellen Sie die Methode `run`. Diese erhält keine Eingabeparameter und soll auch nichts zurückgeben. Lassen Sie damit ihre Spülmaschine laufen (Reduzieren des Schmutzigkeitsgrades der Elemente in `dishesInside` um 0.3).

iv) Schreiben Sie nun noch die Methode `public void removeCleanDishes()`, welche alle sauberen Teile aus der Spülmaschine entfernt (als sauber gelten Teile deren Schmutzigkeitsgrad kleiner oder gleich 0.1 ist). Zum Entfernen genügt es, das Array an der entsprechenden Stelle auf "null" zu setzen.

Tipp: Es ist hilfreich sich vor dem Implementieren ein Klassendiagramm mit den Feldern und Methoden inklusive Signaturen zu erstellen.
Tipp: Um Ihr Programm zu testen können Sie in `Dishwasher` eine main-Funktion erstellen, die verschiedenen Methoden sinnvoll aufrufen und sich die Ergebnisse ausgeben lassen.
Hinweis: Überlegen Sie sich welche Größe das Array dishesInside mindestens braucht.
Bonus: Warum ist ein Array hier ungeeignet? Kennen Sie bereits andere Datenstrukturen die hier besser passen würden. (unbewertet)
Bonus: Wo würde es hier Sinn machen den modifier `final` zu nutzen. (unbewertet)