Testat

Objektorientierte Programmierung Praktikum

Gruppe 2 (11:00)

Studiengang

Wirtschaftsinformatik und E-Business (Bachelor) WI PLUS

Prof. Dr. Thomas Bayer

Sommersemester 2022

(Bitte in Druckbuchstaben ausfüllen!)

Matrikel-Nr:	Datum: 06.07.2022
Sitzplatz:	Semester: **SS 2022**
Hilfsmittel: **keine**	Bearbeitungszeit: **90 Minuten**

(Bitte nicht ausfüllen!)

	Unterschrift Prüfer:					Note:				
Aufgabe	CL	Size	Get	Cont	Add	getS	Rem	Size		Gesamt
max. Punkte	**20**	**5**	**5**	**10**	**25**	**20**	**10**	**15**		**110**
Punkte										

Ausgangssituation: Benzinpreis-App

Es soll eine Datenstruktur für eine Benzinpreis-App erstellt werden. Die Struktur nimmt Tankstellen (Gasstation) mit ihren aktuellen Durchschnittspreisen auf (ein Preis pro Tankstelle). Die maximale Anzahl an Tankstellen kann angegeben werden (Konstruktor beachten).

Absrakte Klasse ALowPriceFinder

```
public abstract class ALowPriceFinder {

// Anzahl an gespeicherten Tankstellen
public abstract int size();

// Liefert die GasStation auf Rang rank (Rang 1 = bester Platz)
public abstract GasStation get(int rank);

// Liefert den Rang von GasStation.
// Ist GasStation nicht enthalten, wird 0 zurückgegeben
// Hinweis: Platz 1 bedeutet Rang 1
public abstract int contains(GasStation station);

// Fügt die GasStation station mit Durchschnittspreis averagePrice hinzu, falls
// GasStation != null und averagePrice > 0.
// Ist sie schon enthalten, wird der Preis auf averagePrice gesetzt
// Sind maxSize GasStations enthalten, wird die neue GasStation nur aufgenommen,
// wenn sie nicht die letzte in der Liste ist (wenn sie nicht die teuerste ist),
// In diesem Fall wird die letzte GasStation gelöscht (sodass nicht mehr als
// maxGasStation in der Liste sind) und die neue GasStation aufgenommen
// Rückgabe: Rang der aufgenommenen GasStation in der averagePrice-Liste;
// 0, falls die GasStation nicht hinzugefügt wurde
// Hinweis: Zur Vereinfachung sind alle averagePrice-Werte ungleich
// Beachten Sie: Eine Tankstelle kann nur einmal enthalten sein
public abstract int add(GasStation GasStation, int averagePice);

// Liefert eine Collection von Tankstellen (GasStation), deren averagePrice <=
// threshold ist
// Rückgabe: Collection, ggf. leer (nicht null)!
public abstract Collection<GasStation> getStations(int threshold);

// Löscht die GasStation GasStation aus der Liste.
// Rückgabe = Rang der GasStation, (0, falls GasStation nicht enthalten)
public abstract int remove(GasStation station);

// Anzahl an gespeicherter Tankstellen, die predicate.test() erfüllen
public abstract int size(Predicate<GasStation> predicate);
```

Vorbereitung

1. IntelliJ
    a. Projekt Testat-Matrikelnummer erstellen
    b. Entpacken Sie 2022-SS-Testat-G2-PriceFinder_IntelliJ.zip
    c. Kopieren des Inhaltes des src-Verzeichnis in das src-Verzeichnis des Projektes
    d. JUnit Version 5.x.y zum Projekt hinzufügen
2. Alternativ: Starten Sie Eclipse mit einem passenden Workspace: Java Version 11 (default)
    a. Importieren Sie das Projekt 2022-SS-Testat-G2-PriceFinder_eclipse.zip (Import existing project)!
3. Erstellen Sie die Testklasse **`LowPriceFinderTest#####`** die von **`ALowPriceFinderTest`** erbt.

Aufgaben

- Benennen Sie das Paket solution in G2_solution##### um
- **Verwenden Sie genau eine Collection oder ein Array**.
- Implementieren Sie die abstrakten Methoden der Klasse **`ALowPriceFinder`**
- ##### = Ihre Matrikelnummer

Vorgehen

- Implementieren Sie die Klasse **`LowPriceFinder#####`**, die von **`ALowPriceFinder`** erbt.
- Passen Sie Ihre Testklasse so an, dass eine Instanz Ihrer Klasse übergeben wird.
    a. Methode **`getInstance()`** – Geben Sie eine Instanz der Klasse **`LowPriceFinder#####`** zurück, die maximal 5 Tankstellen speichern kann (Konstruktor beachten!).

Abgabe

- Exportieren Sie Ihr Projekt (Testat-G2-#####) als Archiv in das Verzeichnis C:\InsightFiles
- Vor Abgabe melden Sie sich!

Hinweise

- Sie benötigen eine Collection/Array
- Um 2 Integers zu vergleichen, können Sie **Integer.compare(a,b)** nutzen
- Ein Comparator erleichtert die Implementierung, ist aber nicht vorgeschrieben
- List erbt von Collection

Hinweise zur Benotung (110 Punkte)

- **Es reichen 50 Punkte zum Bestehen!**
- **(Versuchte) Manipulation der gegebenen Klassen/Interfaces: max. 25 Punkte**
- **Kopieren der gegebenen Klassen/Interfaces in ein eigenes Paket: 0 Punkte**
- **Programm nicht lauffähig (Syntaxfehler): max. 25 Punkte**
- **Artefakte aus der anderen Gruppe tauchen auf: 0 Punkte!**
- Sowohl Unit-Tests als auch der Quelltext werden für die Benotung herangezogen
- Unit-Tests dienen nur zur Orientierung!

Collections

Interface/Klasse	Implementierende Klasse	Wichtige Methoden
Map	HashMap	put(k,v), get(k), size(), remove(k), keySet(), values(), clear()
Set	HashSet	add(e), contains(e), remove(e), clear()
List	ArrayList	add(e), get(i), clear() add(i,e): Fügt e an Position i ein, vorher wird die Liste ab i nach rechts verschoben, set(i, e):Überschreibt den Eintrag an Position i mit e contains(e), remove(e): true, falls e (einmal) entfernt wurde, sonst false remove(i): Löscht den Eintrag an Position i
Date	-	new Date()// aktuelles Datum
Consumer<T>	-	void accept(T t)
String		String[] split(separator), boolean contains(string), String replace(x,y) // ersetzt x durch y int length(), String toLowerCase(), toUpperCase()
Comparator<T>		int compare(t1,t2)
Predicate<T>		boolean test(T t)

Anhang

Eine innere Klasse wird wie folgt erstellt:

```
public class OuterClass {
…
      public class InnerClass {
      …
       }
…
}
```