Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

Collections

Ziel: Anwendung von Collectionklassen mit Generics und ihren Standardmethoden

1.1. Kopieren Sie sich von Moodle die Zip-Datei „DatenArtikelContainer2“ und fügen Sie die gegebenen Klassen einem neuen Package hinzu. Starten Sie das Programm mit der Startklasse und korrigieren Sie eventuelle Fehler.

1.2. Ersetzen Sie in der Klasse `ArtikelContainer` beim Attribut `alleArtikel` die Array-Datenstruktur durch eine `ArrayList`-Datenstruktur mit Typparameter `<Artikel>`. Das int Attribut `iAnzAkt` kann entfallen.
Begründung? Wie kann die Anzahl der Einträge in einer List-Datenstruktur bestimmt werden?

1.3. Passen Sie den parametrierten Konstruktor so an, dass das Array durch die konkrete List-Datenstruktur `ArrayList` ersetzt wird. Der Parameter des Konstruktors wird verwendet, um die Anfangskapazität der `ArrayList` festzulegen. Warum ist die Kapazitätsangabe beim Erzeugen des Objekts optional?

1.4. Kommentieren Sie zunächst die Methode `sucheArtikelNachBezeichnung` in der Klasse `ArtikelContainer` und den Aufruf der Methode in `AuftragsVerwContUI` aus.

Korrigieren Sie nun alle Fehler, die durch den Austausch der Datenstruktur entstanden sind. Eventuell benötige Methoden finden Sie in der Java-Doku.

1.5. Nachdem das Programm wieder lauffähig ist, soll auch noch die Methode `sucheArtikelNachBezeichnung` wieder zum Laufen gebracht werden.

Es bietet sich an, in der Methode für das Aufsammeln der gefundenen Artikel statt dem `Array` ebenfalls eine `ArrayList` -Datenstruktur zu verwenden.

Die Methode wird jetzt um einiges einfacher, da sich die `ArrayList` im Gegensatz zum Array automatisch verlängert.

Da der Rückgabetyp der Methode nicht verändert werden soll, muss beim Return allerdings die `ArrayList` in ein `Array` konvertiert werden. Da dies etwas kryptisch ist, hier die entsprechende Anweisung:

```
return gefundeneArtikel.toArray(new Artikel[0]);
```

2.1. Gedächtnistrainingsprogramm

In das Prgramm gibt der Benutzer Städtenamen ein, die jeweils an eine `ArrayList` hinten angefügt werden. Nach jeder Eingabe eines neuen Namens wird der Benutzer aufgefordert nacheinander die bereits vorhandenen Namen einzugeben. Falls alles in Ordnung ist, kann ein weiterer Namen angefügt werden, sonst endet das Programm mit einer Fehlermeldung: siehe nachfolgenden Programmdialog:

```
Welche neue Stadt kommt hinzu?
Stuttgart
Wie sieht die gesamte Route aus? (Tipp: 1 Stadt)
Nächste Stadt auf der Route: Stuttgart
Prima, alle Städte in der richtigen Reihenfolge!
Welche neue Stadt kommt hinzu?
Ludwigsburg
Wie sieht die gesamte Route aus? (Tipp: 2 Städte)
Nächste Stadt auf der Route: Stuttgart
Nächste Stadt auf der Route: Ludwigsburg
Prima, alle Städte in der richtigen Reihenfolge!
Welche neue Stadt kommt hinzu?
Heilbronn
Wie sieht die gesamte Route aus? (Tipp: 3 Städte)
Nächste Stadt auf der Route: Stuttgart
Nächste Stadt auf der Route: Ludwigsburg
Nächste Stadt auf der Route: Heilbronn
Prima, alle Städte in der richtigen Reihenfolge!
Welche neue Stadt kommt hinzu?
Würzburg
Wie sieht die gesamte Route aus? (Tipp: 4 Städte)
Nächste Stadt auf der Route: Stuttgart
Nächste Stadt auf der Route: Ludwigsburg
Nächste Stadt auf der Route: Heilbronn
Nächste Stadt auf der Route: Würzburg
Prima, alle Städte in der richtigen Reihenfolge!
Welche neue Stadt kommt hinzu?
```

Für die Prüfschleife, zum Testen, ob der Benutzer noch alles weiß, soll ein Iterator verwendet werden.

2.2. Um die Sache etwas schwieriger zu machen, kann man die Liste auch schon beim Start mit einer bestimmten Anzahl Städten füllen, die sich der Benutzer auf jeden Fall schon mal merken muss; Bsp.:

```
[Stuttgart, Ludwigsburg, Heilbronn, Würzburg, Schweinfurt, Bad Hersfeld, Kassel]
Welche neue Stadt kommt hinzu?
Göttingen
Wie sieht die gesamte Route aus? (Tipp: 8 Städte)
Nächste Stadt auf der Route: Stuttart
Stuttart ist nicht richtig, Stuttgart wäre korrekt. Schade!
```

3.1. Ein Array (und entsprechend auch eine ArrayList) bietet einen sehr effizienten wahlfreien Zugriff auf eine Tabelle über einem ganzzahligem Index.
Hat man aber als „Index“ keine Zahl, sondern „Namen“ aus einer sehr großen Wertemenge, so bietet sich für einen wahlfreien Zugriff eine „Hashtabelle“ an. Java bietet hierfür die Collection-Klasse `HashMap`.

Als Ausgangspunkt können Sie die Datei `VerzeichnisUI.java` (aus Moodle) verwenden. Die Daten aus dem darin enthaltenen Array sollen in einer HashMap

gespeichert werden. Der Name dient als „Key“ und die Telefonnummer ist der zugehörige „Wert“ („Value“). Verwenden Sie zur Übertragung der Daten z.B. eine foreach Schleife.

Danach soll folgender Benutzerdialog ermöglicht werden:

```
Wessen Nummer suchen Sie? Fridolin
Die Telefonnummer lautet: 07071/182726
Wessen Nummer suchen Sie? Angela
Die Telefonnummer lautet: 07031/87345
Wessen Nummer suchen Sie? Peter
Kein Eintrag im Verzeichnis gefunden
Wessen Nummer suchen Sie? Michaela
Die Telefonnummer lautet: 0711/87654
Wessen Nummer suchen Sie?
Programm Ende
```

Das Programm endet, wenn der Benutzer bei der Eingabeaufforderung einfach nur `<Enter>` drückt (`Eingabe.getString()` liefert in diesem Fall `null`)

3.2. Ergänzen Sie das Programm von Aufgabe 3.1 so, dass man auch neue Einträge zur HashMap hinzufügen kann.