Laborprüfung

Für die Bearbeitung der Aufgaben in dieser Laborprüfung bekommen Sie ein vorbereitetes Eclipse-Projekt. Aktualisieren Sie das Projekt in Eclipse zu Beginn einmal durch Drücken der Taste F5. Für jede Aufgabe gibt es in dem Projekt ein entsprechendes Package. Nehmen Sie Änderungen zur Lösung einer Aufgabe nur innerhalb des zugehörigen Packages vor. Teilweise ist es notwendig, vorgegebenen Quellcode zu verändern. Teilweise müssen Sie neue Klassen erstellen.

1 Dreidimensionaler Punkt (14 Punkte)

package point

In diesem Package finden Sie eine Klasse `Point`. Diese repräsentiert einen dreidimensionalen Punkt, bestehend aus den drei Koordinaten `x`, `y`, und `z`.

1.1 Vergleich von Referenztypen

Zum Vergleichen von Referenztypen wird in Java meist die Methode `equals` verwendet. Überschreiben Sie die Methode `equals` der Klasse `Object` in der Klasse `Point`. Zwei Punkte vom Typ `Point` sind genau dann identisch, wenn ihre drei Koordinaten paarweise identisch sind.

1.2 Punkt mit Namen

Eine spezielle Version von `Point` besitzt zusätzlich einen Namen. Implementieren Sie eine neue Klasse `PointWithName` und leiten Sie diese von `Point` ab. Der Name wird zusammen mit den Koordinaten an den Konstruktor übergeben und dort gesetzt. Der Name soll über eine Methode `getName` abzufragen sein. Es soll nicht erlaubt sein, dass der Name leer gelassen wird oder den Wert `null` hat. In diesen Fällen, soll `PointWithName` eine `IllegalArgumentException` auslösen. Überschreiben Sie in `PointWithName` außerdem die Methode `toString`, sodass nicht nur die Koordinaten sondern auch der Name zurückgegeben werden. Der Beispielaufruf:

```
PointWithName point = new PointWithName(1,2,3,"Center");
System.out.println(point);
```

soll folgende Ausgabe auf der Konsole erzeugen:

```
Center: (1.0, 2.0, 3.0).
```

1.3 Liste von Punkten

Erweitern Sie die Klasse `Point` um eine `main`-Methode. Legen Sie dort eine `ArrayList` vom Typ `Point` an. Befüllen Sie die Liste mit mehreren Objekten vom Typ `Point` und `PointWithName`. Iterieren Sie dann über die Liste und geben Sie jedes Element auf der Konsole aus. Auf welche Art Sie die Liste durchlaufen, ist Ihnen freigestellt.

2 Baumstruktur (9 Punkte)

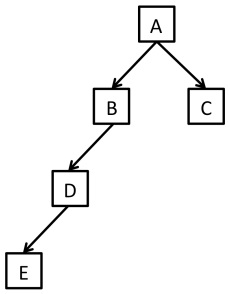
package tree

In diesem Package finden Sie das Interface `Node` und die Klasse `GenericNode`, die das Interface implementiert. Die Klasse `GenericNode` verwendet ein Array für die Ablage der Kinder. Das Interface Node beschreibt einen Knoten in einer Baumstruktur. Ein Knoten kann Kinder (`children`) haben, die ebenfalls vom Typ `Node` sind. Um einfacher mit der Datenstruktur experimentieren zu können, hat `GenericNode` eine Objektvariable `character`. In diese Variable vom Typ `char` können Sie beliebige Zeichen eintragen und diese z.B. zum Debuggen verwenden.

2.1 Elternknoten

Mit der bestehenden Implementierung kann ein Baum, der aus `Node`-Knoten besteht, nur von oben (Wurzelknoten) nach unten durchlaufen werden. Erweitern Sie das Interface `Node` um eine Methode `getParent`, die den Elternknoten eines Knotens zurückliefert, und eine entsprechende Setter-Methode `setParent`. Der Elternknoten des Wurzelknotens ist `null`. Entsprechend müssen Sie auch die notwendige Funktionalität in der Klasse `GenericNode` einfügen. Verwenden Sie den Bezeichner `parent`, um sich den Elternknoten als Objektvariable zu merken. Passen Sie die übrigen Methoden so an, dass die Elternknoten korrekt gesetzt werden.
Testen Sie die Funktionalität in der `main`-Methode der Klasse `GenericNode` durch das Aufrufen der relevanten Methoden und durch die Konsolenausgabe. Es ist nicht notwendig, hier das JUnit-Framework zu verwenden. Testen Sie insbesondere, ob

- ... die Anzahl der Kinder eines Knotens nach dem Hinzufügen eines Kindknoten korrekt ist,
- ... ein Kindknoten sich den richtigen Knoten als Elternknoten (`parent`) gemerkt hat,
- ... ein Elternknoten sich das richtige Objekt als Kindknoten gemerkt hat.



Die Abbildung zeigt einen Beispielbaum. Der Knoten A hat zwei Kinder, die Knoten B und C. Der Elternknoten `parent` des Knotens B ist demnach A, das gleiche gilt für den Knoten C.
Diesen Beispielbaum finden Sie auch in der `main`-Methode der Klasse `GenericNode`.

2.2 Anzahl der Knoten im Baum

Erweitern Sie das Interface `Node` um eine Methode `getNumberOfNodes`, die die Anzahl der Knoten unterhalb eines Knotens zurückliefert. Der Knoten selber soll ebenfalls mitgezählt werden. Ruft man die Methode für den Wurzelknoten auf, so erhält man also die Anzahl aller Knoten im Baum. Implementieren Sie die Funktionalität in der Klasse `GenericNode`. Testen Sie die Funktionalität in der `main`-Methode.

3 DVD-Sammlung (8 Punkte)

package dvd

In diesem Package finden Sie die aus dem Praktikum bekannten Klassen `Dvd` und `DvdCollection` zum Verwalten von DVDs. Außerdem ist eine Testklasse gegeben. In die Implementierung haben sich eine Reihe von Fehlern eingeschlichen.

3.1 Fehlerbehebung

Unternehmen Sie für jeden Fehler, den Sie finden, die folgenden Schritte:

- Beheben Sie den Fehler.
- Erläutern Sie den Fehler und Ihre Verbesserung mit einem Kommentar direkt beim Fehler.

Es befinden sich insgesamt drei Fehler im gegebenen Quellcode, die Sie durch die gegebenen Testmethoden erkennen können.
Hinweis: Beheben Sie die Fehler in der Reihenfolge, wie sie im Ergebnis des JUnit-Tests erscheinen.

3.2 Zusätzliche Testmethode

Die Klasse `Dvd` wurde so erweitert, dass sie das Interface `Comparable` implementiert. Dazu wurde die Methode `compareTo` geschrieben. Die Funktionsweise der Methode `compareTo` ist im zugehörigen Kommentar beschrieben. Es existiert noch keine Testfunktionalität dazu. Implementieren Sie in der Testklasse eine JUnit-Testmethode, die die Vergleichsfunktionalität in der Methode `compareTo` überprüft. Durch die Tests sollten Sie einen weiteren Fehler erkennen. Korrigieren Sie dann die Methode `compareTo`.

4 Verwaltung von TV Kanälen (15 Punkte)

package tv

In dieser Aufgabe sollen Sie eine einfache Kanalverwaltung für ein TV-System entwickeln. Die Verwaltung hat eine feste Anzahl von Kanälen. In jedem Kanal kann ein Sender gespeichert werden. Ein Sender wird über seinen Namen (`String`) dargestellt. Die Kanalverwaltung hat einen aktuellen Kanal. Der Sender auf dem aktuellen Kanal kann ausgegeben und neu belegt werden. Der aktuell gewählte Kanal kann auf den folgenden weitergeschaltet werden (Zapping). Außerdem kann nach einem Sender (und dem damit zugehörigen Kanal) gesucht werden.

4.1 Klasse TVChannelManagement

Schreiben Sie eine Klasse `TVChannelManagement`. Die Klasse merkt sich eine Liste (Array) von Sendern (`Strings`) und den Index des aktuellen Kanals. Die Klasse bietet die folgenden Methoden:

Konstruktor `TVChannelManagement(numberOfChannels:int)` - Der Konstruktor hat einen Parameter: die Anzahl der Kanäle.

Öffentliche Methode `switchNext()` - Sprung zum nächsten Kanal. Falls der Kanalindex den maximalen gültigen Kanalindex übersteigt, springt der Index zurück zur Position `0`.

Öffentliche Methode `saveChannel(senderName:String)` - Speichert einen neuen Sender an der aktuellen Kanalposition. Falls dort schon ein Sender abgespeichert ist, wird dieser überschrieben.

Öffentliche Methode `getCurrentChannelName():String` - Liefert den Sendernamen im aktuellen Kanal zurück. Die Methode soll `null` zurückgeben, falls im aktuellen Kanal noch kein Sender abgespeichert ist.

Öffentliche Methode `moveToChannel(senderName:String)` - Springt auf den ersten Kanal, in dem der Sender mit dem Namen `senderName` liegt. Kein Sprung, wenn der Sender nicht zu finden ist.

Im vorbereiteten Package finden Sie eine Klasse `TVChannelManagementApplication`. Die Ausgabe nach dem Durchlaufen der `main`-Methode in dieser Klasse sollte folgendermaßen aussehen, wenn Sie alle Funktionalität korrekt implementiert haben:

```
SAT1
ARD
NDR3
ZDF
SAT1
NDR3
NDR3
```

4.2 Benutzersteuerung

Entwickeln Sie eine Möglichkeit, das `TVChannelManagement` von einem Anwender steuern zu lassen. Schreiben Sie dazu eine Methode `use`. In der Methode wird der Anwender so lange nach einem Kommando (`String`) gefragt, bis er das Kommando `"exit"` eingibt. Jedes Kommando löst ein anderes Verhalten aus. Mögliche Kommandos sollen sein:

- `"exit"`: Beenden der Benutzersteuerung und damit der Methode.
- `"next"`: Weiterschalten zum nächsten Kanal.
- `"show"`: Ausgabe des aktuellen Kanals auf der Konsole.

Alle anderen Kommandos werden ignoriert.

Laborprüfung

Platz für Notizen

(Alles was Sie hier schreiben, fließt NICHT mit in die Bewertung ein.)