Aufgabenstellung für das *erste* Übungsbeispiel der LVA "Objektorientiertes Programmieren"

Abgabeschluss für den Source Code: Freitag 22.05.2020

WICHTIG:
Wird das Beispiel nicht fristgerecht abgegeben, haben Sie in Folge nicht mehr die Möglichkeit, die LVA im laufenden Semester erfolgreich abzuschließen!

Abgabegespräch:

Zwischen ***26.05.2020*** *und* ***28.05.2020*** (Jeweils zwischen 13:00 und 18:00 Uhr: Den Tag und die genaue Uhrzeit wählen Sie selbst in TUWEL.)

Allgemeines

Jeder Teilnehmer muss die Beispiele **eigenständig** ausarbeiten. Bei der Abgabe jedes Beispiels gibt es ein Abgabegespräch mit einem Tutor oder Assistenten. Diese Abgabegespräche bestehen aus den folgenden Punkten:

- Im Zuge der Abgaben müssen Sie eine Codeänderung vornehmen können.
- Beantworten von Fragen zu Ihrem Programm.
- Ihr Programm wird mit automatisierten Tests geprüft.
- Beantwortung theoretischer Fragen zu den jeweils in der Übung behandelten Konzepten.

Umsetzung und Abgabe

- Halten Sie sich exakt an die Beschreibungen der Klassen in der **JavaDoc**.
- Geben Sie Ihrem Eclipse-Projekt einen eindeutigen Namen. Halten Sie sich dabei an folgende Benennungsvorschrift:

  Eclipse-Projekt: *matrNr_Beispiel2_Nachname*

  Beispiel: *00123456_Beispiel2_Mustermann*

  ACHTUNG: Wenn Sie sich nicht an die Benennungsvorschrift halten, kann Ihre Abgabe nicht gewertet werden!
- Name und Matrikelnummer muss zu Beginn jeder Klasse in einem Kommentarblock angeführt werden!
- Exportieren Sie das Projekt (inklusive aller für Eclipse notwendigen Dateien, z.B.: .classpath, .project) als ZIP-Datei. Eine Anleitung dazu finden Sie in TUWEL.

Kontakt:

Inhaltliche Fragen: Diskussionsforum in TUWEL

Organisatorische Fragen: oop@ict.tuwien.ac.at

Aufgabe:
Datenverwaltung mittels generischen abstrakten Datentypen

Ziel:

Erwerb zusätzlicher konkreter Erfahrung und Kompetenz in der Umsetzung folgender Konzepte: **Klassendiagramme, Abstrakte Datentypen, Interfaces, Generics, Casting, Exceptions und Exception Handling**. Insbesondere das Verstehen des Unterschiedes zwischen Konkreten und Abstrakten Datentypen sowie die Fähigkeit, mit Hilfe beider wiederum Abstrakte Datentypen zu definieren.

Einleitung:

Ihre Aufgabe besteht darin, ***generische abstrakte Datentypen*** zu programmieren. Diese abstrakten Datentypen erlauben es, ohne konkretes Wissen über den inneren Aufbau der Datenstrukturen und deren Implementierung, Daten zu verwalten. Dafür sollen zwei generische Datentypen implementiert werden. Der erste Datentyp verwaltet Daten in einer **einfach verketteten Liste**, während mit dem zweiten die Daten mittels einer **Baumstruktur** strukturiert werden können. Eine strukturelle Übersicht der zu implementierenden Programmteile, inklusive der Packages, Interfaces sowie Klassen und deren Attribute, finden Sie in Abbildung 1 und Abbildung 6. Eine genaue Beschreibung der einzelnen Methoden dieser Klassen finden Sie in der beigelegten **JavaDoc**. Halten Sie sich **exakt** an die **Spezifikationen** der **Klassendiagramme** und der **JavaDoc**, da Ihr Programm im Rahmen der Abgabe automatisiert getestet wird. Verwenden Sie in Ihrer Implementierung dieser Datenstrukturen (soweit erforderlich) Exception Handling.

Die Aufgabenstellungen setzt sich aus folgenden Teilaufgaben zusammen:

1. **Implementierung einer einfach verketteten Liste (Container)**
   Die Implementierung dieser Teilaufgabe erfolgt im Package `container`. Das Klassendiagramm dieses Packages ist in Abbildung 1 dargestellt. Es ist anhand des *Collection*-Interfaces eine eigene Implementierung für eine generische Liste zu erstellen.
2. **Implementierung einer Baumstruktur zur geordneten Darstellung von Daten**
   Die Implementierung dieser Teilaufgabe erfolgt im Package `tree` und `tree.node`. Das Klassendiagramm dieses Packages ist in Abbildung 6 dargestellt.
3. **Implementierung einer Start-Klasse**
   Erstellen Sie eine Klasse *Application*, mit der Ihr Programm gestartet werden kann. Legen Sie in dieser Klasse einen Baum mit mindestens vier Knoten und zugehörigen Kindknoten an und geben Sie diesen auf der Konsole aus.

Automatisierte Testung:

Für diese Aufgabe gibt es einen Server der es Ihnen ermöglicht Ihre Abgabe vorab zu testen. Der Server erlaubt den Upload der von Ihnen erstellten ZIP-Datei. Dabei muss sich die Bennenung des Projekts und der ZIP-Datei an die Vorgaben halten. Nach dem Upload werden Sie auf eine Webseite mit den Testergebnissen bzw. der Log-Datei des Compilers weitergeleitet.

Die Erreichbarkeit des Servers wird bis 08.05.2020 in TUWEL bekannt gegeben.

Anforderungen damit Ihre Lösung akzeptiert wird:

Es werden nur ZIP-Datei die den folgenden Anforderungen entsprechen als gültige Abgabe akzeptiert.

- Die Namen des Eclipse-Projektes und des ZIP-Files müssen auf *Matrikelnummer_Beispiel2_Nachname* lauten
- Ihr Projekt (in der ZIP-Datei) muss bei uns kompilieren und sich genau an die **JavaDoc** halten. Die Kompilierbarkeit können Sie unter anderem mittels des Servers, für die automatisierte Testung, testen.
- Die Lösung muss mindestens 50 Prozent, der im Testset enthaltenen Testfälle, positiv absolvieren.

Wird eine ZIP-Datei abgegeben, die diesen Anforderungen nicht entspricht, wird dies so gewertet, als ob keine Abgabe erfolgt wäre und hat demensprechend einen Übungsabbruch zur Folge.

Wichtig: Der Upload auf den Server ersetzt **nicht** den Upload in TUWEL.

Teilaufgabe 1: Package container – Einfach verkettete Liste

Die erste Teilaufgabe besteht darin eine einfach verkettete Liste zu programmieren. Ein detailliertes Klassendiagramm des Package `container` ist in Abbildung 1 zu sehen. Überlegen Sie sich, welche Vorteile es bringt, wenn Sie den abstrakten Datentyp `Collection` aus der Java API als Ausgangsbasis für Ihren `Container` verwenden.

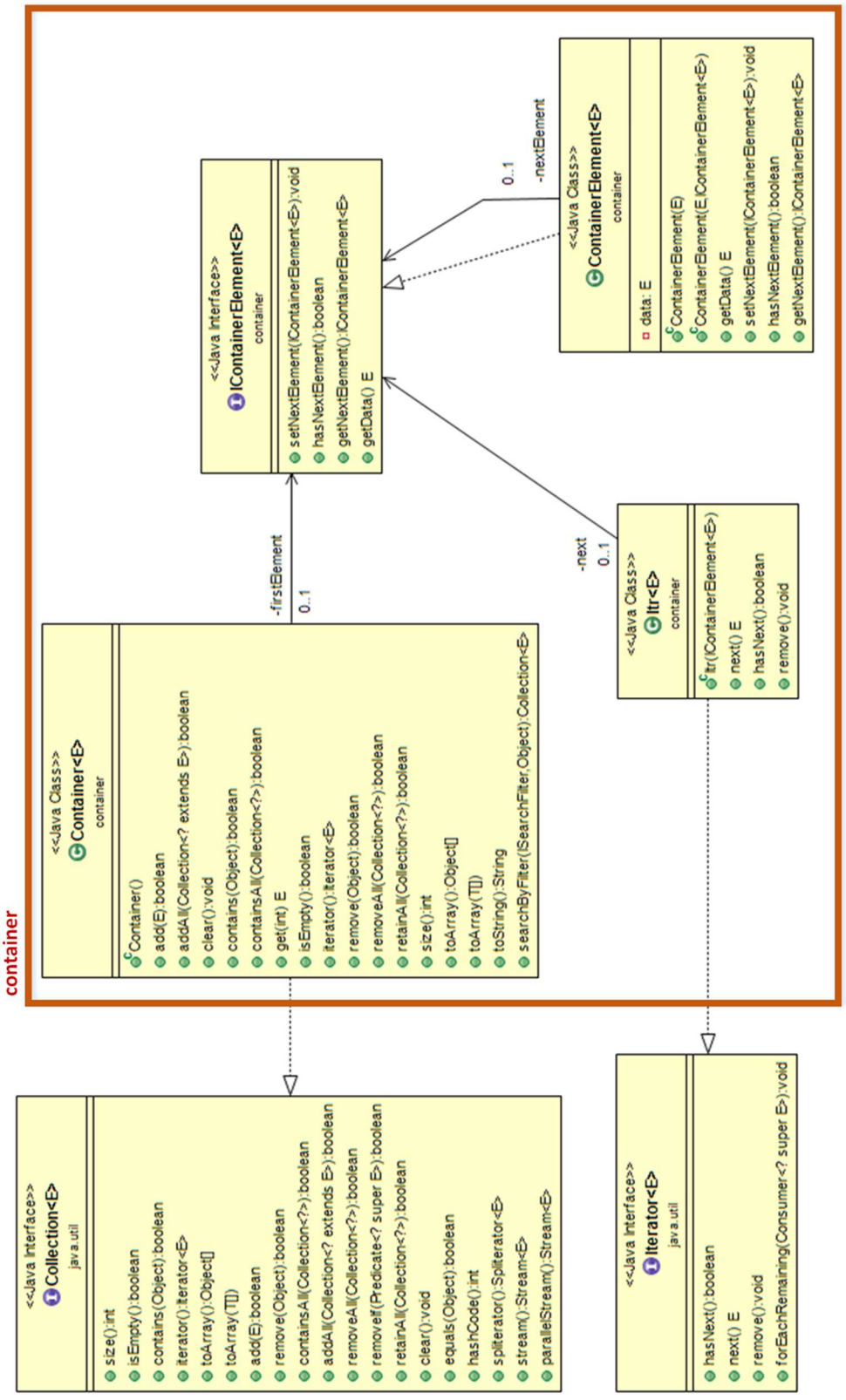

Abbildung 1 – Klassendiagramm des Packages: container

Eine einfach verkettete Liste (https://en.wikipedia.org/wiki/Linked_list) enthält mehrere `ContainerElemente` und jedes dieser `ContainerElemente` hat einen Zeiger auf seinen Nachfolger. Dadurch ergibt sich eine zusammenhängende Liste an `ContainerElementen`. Die Liste wird durch einen `Container` verwaltet, welcher unter anderem eine Referenz auf das erste `ContainerElement` der Liste hält und Operationen auf der Liste bereitstellt. Abbildung 2 zeigt den Aufbau einer einfach verketteten Liste.

Abbildung 2 – Eine einfach verkettete Liste

Java bietet bereits Schnittstellenbeschreibungen (`Interfaces`) an, die das Verhalten einer Liste spezifizieren. Ihre Aufgabe ist es eine konkrete Implementierung des `Collection`-Interfaces in dem Package `container` zu erstellen.

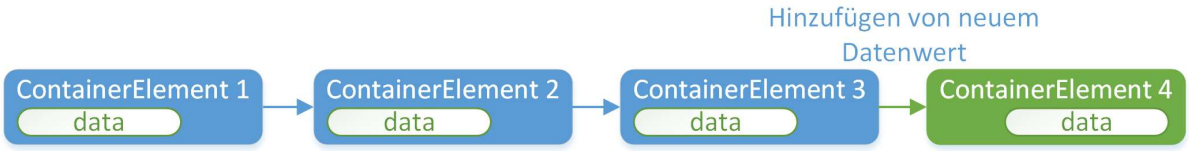

Abbildung 3 – Neuen Datenwert hinzufügen

Die Operationen, die auf der verketteten Liste möglich sind, sind im `Collection` Interface definiert. So sind beispielsweise Methoden definiert, die es ermöglichen Daten zu einer Liste hinzuzufügen (siehe Abbildung 3) oder zu entfernen (siehe Abbildung 4).

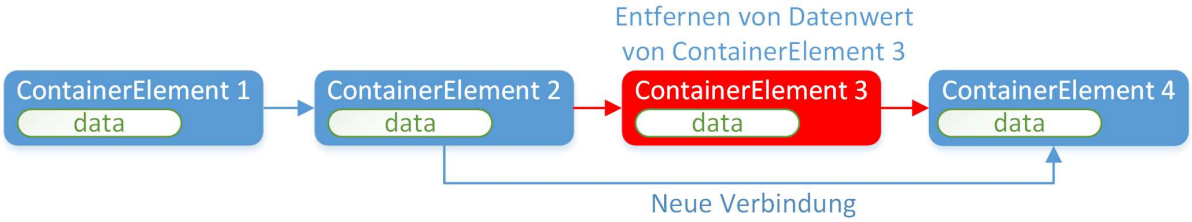

Abbildung 4 – Datenwert aus der Liste entfernen

Zusätzlich sind noch weitere Methoden, die nicht Teil des `Collection`-Interface sind, zu implementieren.

Die einzelnen Einträge in der verketteten Liste sind anhand des Interfaces `IContainerElement` in der Klasse `ContainerElement` umzusetzen. Der eigentliche Datenwert eines Eintrags ist als generisches Attribut in der `ContainerElement` Klasse gespeichert.

Die Klasse `Itr` ist eine Implementierung des Interfaces `Iterator` und ermöglicht in der einfach verketteten Liste zu navigieren. Durch die Implementierung dieses Interfaces unterstützt der Container unter anderem die Verwendung der verkürzten for-Schleife.

Eine ausführliche Beschreibung der zu implementierenden Methoden entnehmen Sie der JavaDoc.

Collection-Interface: http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html
Iterator-Interface: http://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

Teilaufgabe 2: Package tree – Baumstruktur

Die zweite Teilaufgabe besteht darin eine Baumstruktur zu programmieren. Ein detailliertes Klassendiagramm der Packages `tree` und `tree.node` ist in Abbildung 6 zu sehen.

Die beiden Interfaces `ITree` und `ITreeNode` definieren zwei abstrakte Datentypen, mit denen die Struktur des Baums festgelegt ist. Das Interface `ITree` legt die Methoden fest, um den Wurzelknoten zu setzen und abzufragen. Als weitere Funktion ist das Suchen nach einem Knoten definiert. Knoten im Baum sind durch das Interface `ITreeNode` festgelegt. Die Implementierung der abstrakten Datentypen erfolgt mit konkreten Datentypen (`GenericTree, GenericTreeNode,…`).

Beispielsweise könnte ein konkreter Baum vom abstrakten Datentyp `ITree` ein Baum mit beliebig vielen Kindknoten oder ein Binärer-Baum mit einem linken und rechten Kindknoten sein. Beide Varianten sind mit den abstrakten Datentypen abbildbar.

Sofern Sie die erste Teilaufgabe `Container` gelöst haben, müssen Sie in dieser Teilaufgabe diese Implementierung verwenden. Sollten Sie Probleme bei der Lösung des `Containers` haben, so verwenden Sie eine alternative Implementierung aus der Standard Java API (z.B. `ArrayList, …`).

Jeder `ITree` hat genau einen Wurzel-Knoten (den `rootNode`), welcher die oberste Ebene des Baumes darstellt. Ein Baum kann, ausgehend von einem Wurzel-Knoten, weitere Knoten hierarchisch in mehreren Ebenen anordnen (siehe Abbildung 5). Es ist dabei möglich, dass ein Knoten Referenzen auf weitere Kindknoten halten kann.

Baum-Datenstruktur: https://de.wikipedia.org/wiki/Baum_(Graphentheorie)

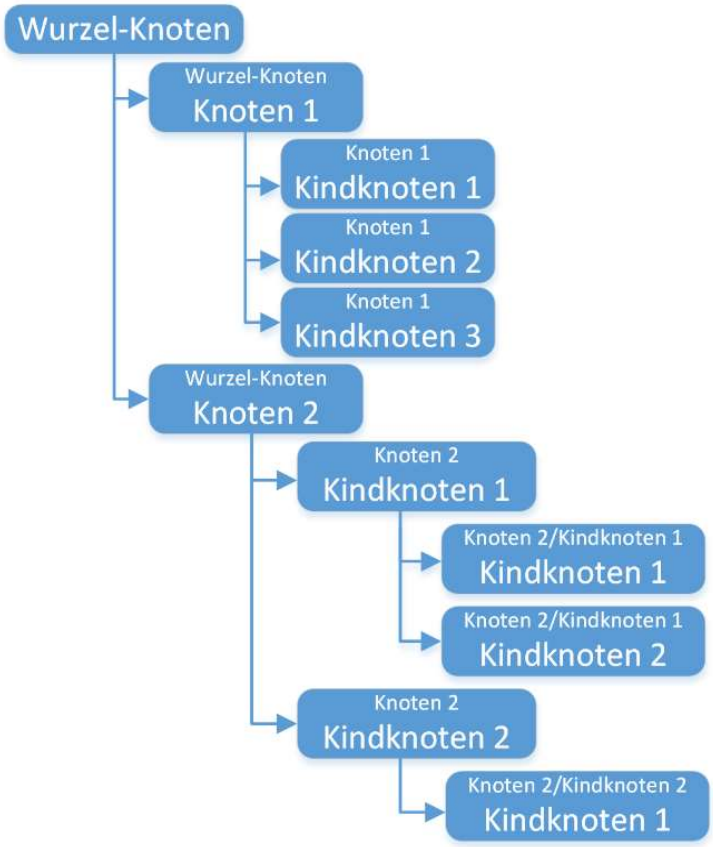

Abbildung 5 – Beispiel für eine verschachtelte Baumstruktur

Hinweis:

Verwenden Sie für Bäume die Produkte als Daten verwalten Ihre Implementierung des **`rbvs.product`** Packages aus **Beispiel 1**. Das Interface **IDeepCopy** müssen Sie dazu im Package **`rbvs.copy`** ablegen. Aktualisieren Sie die **import**-Statements in `rbvs.product` entsprechend.

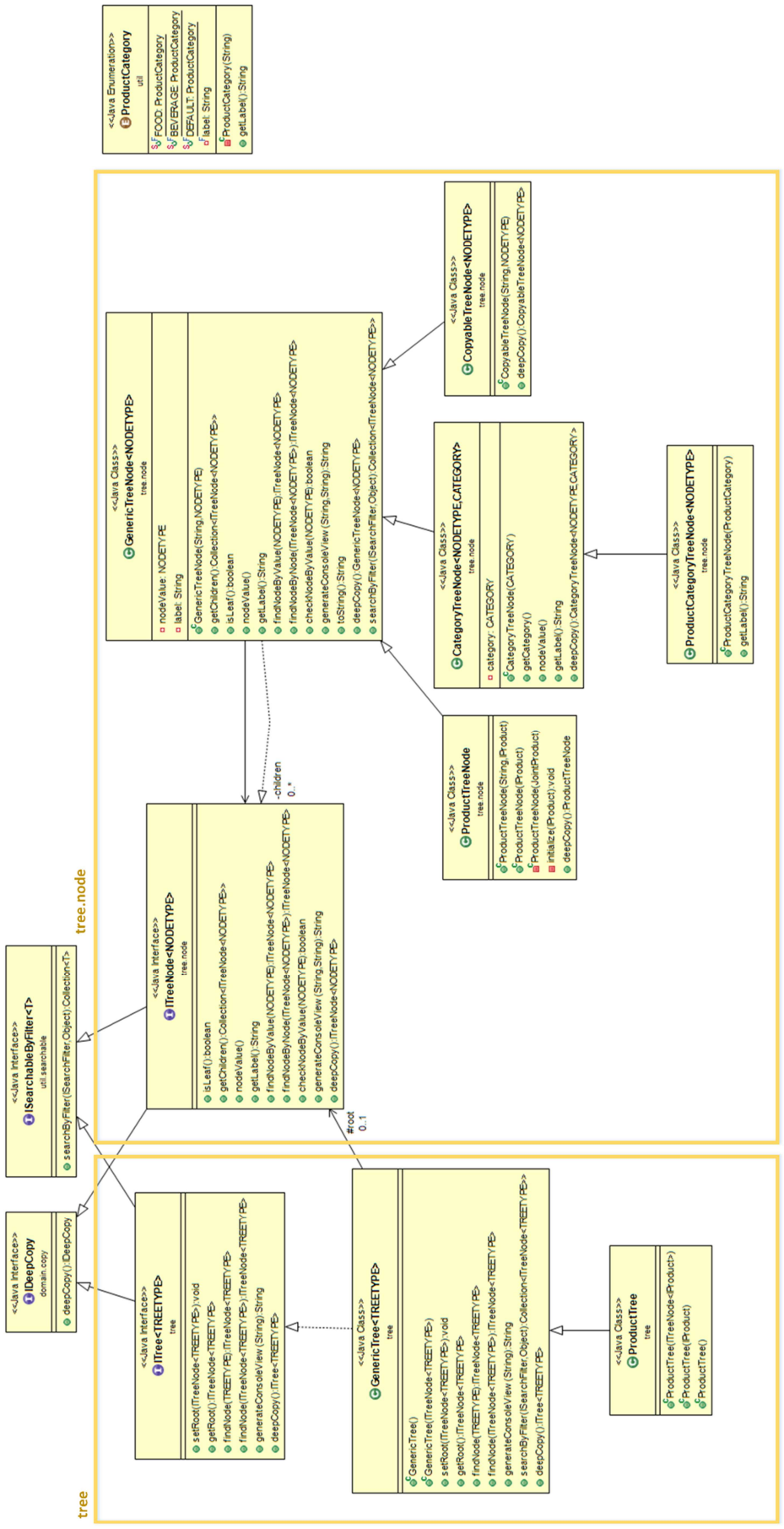

Abbildung 6 – Klassendiagramm der Packages: tree und tree.node

Teilaufgabe 3: Implementierung einer Start-Klasse

Erstellen Sie eine Klasse *Application*, mit der Ihr Programm gestartet werden kann. Legen Sie in dieser Klasse einen Baum mit mindestens vier Knoten und zugehörigen Kindknoten an und geben Sie diesen mittels der `generateConsoleView`-Methode auf der Konsole aus.

Name der Start-Klasse: Application.java

BONUSAUFGABE:
Containersynchronisierung mittels Netzwerkkommunikation

Implementieren Sie eine Spezialisierung des `Containers`, die eine Einweg-Synchronisierung über das Netzwerk ermöglicht. Dabei ist ein `Container` immer ein potentieller *Server* für genau einen anderen `Container` (*Client*). Zusätzlich kann jeder `Container` aber auch selbst, als Client, mit einem anderen Server verbunden sein. Dadurch ist eine Verkettung von `Containern` möglich. Somit kann jeder Client-Container kann auch gleichzeitig Server-Container für einen anderen Client-Container sein. Nach jeder ausgeführten Operation auf dem Server muss der Client benachrichtigt und synchronisiert werden.

Die Synchronisation wird bei jedem neu hinzukommenden oder gelöschten `IContainerElement` im *Server* ausgelöst. Der Server benachrichtigt den eventuell verbundenen Client über die Änderung. Der Client führt die auf dem Server ausgeführte Operation bei sich selbst mit den übermittelten Daten aus.

Die Spezialisierung Ihres `Containers` kann direkt auf den Typ `String` eingeschränkt werden. Sie müssen daher nur Strings in Ihrem `Container` synchronisieren können.

Um Ihren Netzwerkcontainer zu testen, erstellen Sie ein einfaches Menü mit zwei Operationen für *add* und *remove*.

Die Kommunikation über das Netzwerk erfolgt mittels `Sockets`. Eine Einführung zu `Sockets` finden sie unter dem Link „Beispiel Netzwerk Kommunikation“. Die detaillierten Beschreibungen der benötigten Java-Klassen finden Sie in der Java-API (Java-API Socket, Java-API ServerSocket).

Beispiel Netzwerk-Kommunikation:	http://docs.oracle.com/javase/tutorial/networking/sockets/
Java-API Socket:	https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html
Java-API ServerSocket:	https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html

Erstellen Sie zwei Applikationen (main-Methoden). In einer Applikation ist der Container als Server konfiguriert und in der zweiten als Client. Ihr netzwerkfähiger Container hat einen Thread der für das Versenden der Information über die durchzuführende Operation (add bzw. remove) und der zugehörigen Daten zuständig ist. Ein weiterer Thread ist für das Empfangen von Daten als Client zuständig. Der Thread der für die Client-Funktionalität zuständig ist führt die entsprechenden Operationen auf dem Container aus.

Durch die Entkopplung der Server- und Client-Funktionalität mittels Threads kann ihr Programm über die Konsole getestet werden, da es nicht zu einem blockierenden Warten im Zuge der Netzwerkkommunikation kommt.

In einer Java-Applikation können mehrere Threads parallel (unabhängig voneinander) ausgeführt werden. Wenn ein Thread blockiert (wenn zum Beispiel auf ein Objekt vom Server gewartet wird), können alle anderen Threads trotzdem weiterarbeiten. Threads arbeiten zwar eigenständig erlauben es aber über Referenzen auf andere Objekte zuzugreifen.

Eine Anleitung dazu finden Sie hier *https://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html* bzw. direkt zur Netzwerkkommunikation hier *http://www.oracle.com/technetwork/java/socket-140484.html* .