Aufgabebeschreibung:

die Aufgabe besteht darin, umfangreich verfügbare Roher mit Längen 2-5 Meter in Schnitten mit unterschiedlicher Länge zu teilen, sodass die die Vierschnittsumme eine optimale möglichst klein bleibt. Die Informationen zu den Schnitten werden in einem Text Datei gelesen, wo bei die Anzahl jedes Schnitts gegeben wurde.

Die Programm und Datenstruktur, die ich für die Lösung dieses Problems (Minimierung der Verschnitt) verwenden würde, sieht folgende Maßnahme aus:

Es gibt eine Klasse Client, die die benötige Objekte erzeugt und Hauptmethode der beteiligte Klasse in der richtigen Reihenfolge aufruft.Weiterhin gibt es funktionale Interfaces für Ein-und Ausgabe File(hier *DataSource bzw. Consumable*), welche Methoden zur Ein- bzw. Ausgabe von implementierenden der Klasse fordern.

Für das Einlesen aus einer Textdatei und das Ausgabe in einer Textdatei gib es jeweils eine Klasse, welche des oben entsprechenden Interfaces implementiert. Die Klasse CuttingInstructions enthält die Methode der Hauptalgorithmen, also Logik um die Rohre zur mehren unterschiedlichen Schnitten zu teilen, ohne große Vierschnittmenge abzufallen.

Die Attribute dieser Klasse kann eine Liste von Paaren (List< Float>, wobei Float als die Deklaration der Länge bestimmter Schnitt verweisen (Siehe der Hauptalgorithmus).

Einlesen und Initialisieren der Daten:

das Einlesen der Daten findet in der Methode *read()* der Klasse der *FileDataSource()* statt!!.

Der Dateiname oder Path wird als String übergeben. Nachdem die Datei geöffnet wurde, wird iterative Zeile von Zeile überprüft, ob eine Zeile mit zwei Wörter, die durch ein Komma voneinander getrennt sind, vorhanden ist. Sollte es nicht der Fall ist wird Nutzer mit einem passenden Exception darüber informiert. Weiteres wird iterative nach eine Zeilen mit Ziffern gesucht und wenn alle Bedingung, die in der Anforderungen bzw. in der Beispiele verankert zutreffen, werden zusammen Objekt Cuts als instantiiert, was wiederum als Rückgabe-Type der Methode *read()* ist. Andernfalls wird durch eine anderen Exception eine Fehlermeldung losgehen. Der Objekt Cuts hat eine Attribut *cuts* List< *Map.Entry<Integer, Float>>*, wobei Integer die Anzahl der Schnitte und Float zugehörige Schnitte verweisen. Alternative ist die Wahl eines Maps wie *HashMap<Float, Integer>* möglich, wobei der Schlüssel und dessen Wert, die Länge bzw. die Anzahl der Schnitte vertreten. Als zweites Attribut kommt *name* vom *Type* String zum Einsatz. Dieses enthält tatsächlich der Name und Nachname der Instruktor, getrennt durch eine Komma. Ab hängig der Sichtbarkeit der Attributen kann man für jedes davon einen *getter ()* zu definieren.

Zur Spannung des Problems stellen wir vor, dass mehrere Dateien zur Verarbeitung zu Verfügung stehen. Dazu wird die *run()* Methode aus dem Interface *Threadeable zur Parallelisierung implementiert.*

Entwurf des Algorithmus:

Der Hauptalgorithmus passiert in der Methode wateMinimizer*()* der Klasse *CuttingInstructions* Dabei wird versucht Variable ***d*** in der Folgende Term möglichst zu minimieren:

$$d = \sum_{k=2}^{k=5} k \,.\, m_k - \sum_{i=1}^{i=c} n_i \,.\, l_i \; ; \; m_k, n_i \in \mathbb{N} ; \; d, l_i \in \mathbb{R}^+$$

Zum Finden der minimale Verschnitt Summe nämlich ***d*** wird *Gready Algorithmus zum Einsatz* kommen. Dabei wird einen *Heap<Tree>*als passende Datenstruktur gewählt. Das *Heap<Tree>* enthält ein Feld von Bäumen, welche durch Kno*ten mit dem Gewicht und deren Anzahl belegt sind.*

Eine Methode *remove()* vom Objekt *Heap* sorgt für das kleinste Gewicht in einem kombinierten Baum. Das Gewicht jedes Knotens ist in der Tat hier.

Das Objekt Tree wird als Objekt eine statistisch *optimalCuttings()* eingeführt bzw. implementiert Diese Objekt wird wiederum durch unterschiedliche Konstrukteur vom Objekt *Node* besetzt. Die Klasse *Node* hat einen Integer und Float Attribute jeweils für die Speicherung die Anzahl und das zugehörige Gewicht vorgesehen.

Ausgabe gemäß der Aufgabestellung:

Die Ausgabe der Daten findet in der Methode wri*t()* der Klasse *FileConsumable* statt. Außerdem wird ein Objekt der Klasse der *FileConsumable* übergeben, welches alle Daten für die Ausgabe kapselt. Der genante Klasse *Data* verbindet Prozessdaten mit dem verarbeiteten Daten zusammen. Also ist sie in der Tat eine Datei Container, welche gelesen Input Daten durch der Methode *read()* und die Ausgabe der Liste mit der günstigsten Vierschnittsumme enthält.