Aufgabenblatt 5
Grundlegende Datenstrukturen

- Abgabetermin: **Samstag, 02.07.2016 23:55 Uhr**
- Zur Prüfungszulassung müssen in einem Aufgabenblatt mind. 25% der Punkte erreicht werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte bestanden werden.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Jede Hausaufgabe enthält eine optionale Zusatzaufgabe, die bei erfolgreicher Bearbeitungen 1 Klausurpunkt zählt.
- Abgabe:
  - Die Abgabe erfolgt über CodeOcean[1]. CodeOcean ermöglicht die Bearbeitung und Ausführung der Quelldateien im Browser. Bei Bedarf können die Sources (Quelltexte) und Unit Tests von der Übungs-Webseite[2] heruntergeladen werden, um lokal zu entwickeln.
  - CodeOcean benutzt Java Version 8.
  - Alle Aufgaben sind über das CodeOcean einzureichen. CodeOcean ermöglicht das Anfügen von Kommentaren bei der Abgabe, darüber hinaus können auch Kommentare in den Source-Code geschrieben werden.
  - Geben Sie bei **jeder** Aufgabe beide Namen Ihrer Gruppe an.
  - Achten Sie darauf, dass Ihr Quelltext hinreichend kommentiert ist.

Aufgabe 1: Sequenzen

12 P

Implementieren sie die sechs Klassen im Paket `assignment5.sequences`: `ArraySequence`, `SinglyLinkedSequence`, `DoublyLinkedList` und die jeweiligen `Fixed*` Versionen. Jede Klasse muss ein oder mehrere Interfaces implementieren. Schauen Sie sich die vorhandene Dokumentation in den Klassen und Interfaces an und basieren Sie Ihre Implementierung darauf. Verwenden Sie in Ihrer Implementierung nicht die Klassen der Java Collections API.

Versuchen Sie in Ihrer Implementierung so wenig Funktionalität wie möglich doppelt zu implementieren. Dokumentieren Sie außerdem für jede Methode in einem Kommentar deren Laufzeit / Komplexität.

[1] `https://open.hpi.de/courses/pt2-2016`
[2] `https://hpi.de/naumann/teaching/current-courses/ss-16/uebung-programmierechnik-ii.html`

Aufgabe 2: Queues vs. Priority Queues

6 P

Sie arbeiten im Verkauf und müssen täglich viele Telefonate mit Kunden führen. Zu Beginn ihres achtstündigen Arbeitstages haben sie bereits ganze zehn Anrufe in Abwesenheit zu beantworten. Für jeden möglichen Rückruf wissen Sie: Er dauert zwischen 1 und 15 Minuten und bringt zwischen 1 und 100 Euro Gewinn ein. Außerdem sind Ihre Kunden ungeduldig – wer länger als eine Stunde nicht von Ihnen zurückgehört hat, geht lieber zur Konkurrenz. Als wäre das nicht genug kommt jedes Mal, wenn Sie ein Telefonat beendet haben und ein neues beginnen, ein weiterer Anruf den es zu beantworten gilt.

In der Klasse `Sales` finden Sie diesen Sachverhalt abgebildet und mittels einer Queue gelöst. Implementieren Sie die Methoden `remainingTime` und `maxProfit`, in denen Sie klüger an das Problem herangehen. Optimieren Sie, in dem Sie die Telefonate mittels *Priority Queues*[3] nach verbleibender Zeit zum Anruf bzw. höchsten Gewinnchancen priorisieren.

Beobachten Sie die Resultate und erläutern Sie in Quelltextkommentaren, wie diese zustandekommen. Welche Gewinnerwartung haben die einzelnen Herangehensweisen jeweils im Durchschnitt? Wonach lässt sich noch priorisieren?

Aufgabe 3: Bäume

10 P

a) Fügen Sie nacheinander die Zeichenfolge `P R O G R A M M I E R E N` in einen anfangs leeren binären Suchbaum ohne Duplikate ein (`A` < `B` < ... < `Z`). Beantworten Sie die folgenden Aufgaben als Quelltextkommentar in der Datei `assignment5.BST`.

- Zeichnen Sie den entstehenden Suchbaum.
- Ab dem Einfügen welches Zeichens wird die AVL-Bedingung verletzt?
- Welche Zeichen könnte man zur Wurzel machen, um den Baum zu balancieren?
- Angenommen, die Zeichenkette wäre zufällig sortiert und würde in einen binären Suchbaum eingefügt. Welche der folgenden Sequenzen können *nicht* die untersuchte Schlüsselfolge bei der Suche nach dem Zeichen `M` sein?

  1) `P O G M`
  2) `N A E G I M`
  3) `N R O P M`
  4) `A R G O E M`
  5) `E I G N M`

b) Gegeben sei die Implementierung eines Binären Suchbaums in `assignment5.BST`. Für die Put-Methode haben Sie bereits eine rekursive Variante, für die Get-Methode eine iterative Variante gegeben. Implementieren Sie nun auch `putIterative(Key key, Value val)` und `getRecursive(Key key)`. Schreiben Sie weiterhin eine Methode `heightRecursive()`, die die Höhe des Baums zurückgibt.

[3] `https://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html`

Zusatzaufgabe: Archäologie

Ein Archäologe hat mehrere beschriftete Tonscherben gefunden. Aus den größeren Stücken kann der Archäologe vermuten, dass die Zeilen immer symmetrisch waren, also die Reihenfolge der Zeichen einer Zeile von links nach rechts die gleiche wie von rechts nach links war. Auf jeder Scherbe steht ein Teil einer Zeile und aus der Größe der Scherben kann er schließen, dass immer höchstens zwei Scherben eine Zeile bilden. Die Scherben hat der Archäologe durchnummeriert.
Da der Archäologe sehr abergläubisch ist und die Scherben im Grab eines mächtigen Schamanen gefunden wurden hat er Angst, den Fluch des Schamanen auszulösen. Deswegen können nur wenige heilige Operationen auf den Scherben durchgeführt werden. Sie können diese Operationen in der Klasse `Shard` nachsehen. Der Archäologe besteht darauf, dass kein `String` aus einer `Shard` rekonstruiert wird, aus Furcht vor einem schrecklichen Fluch.
Finden Sie alle Scherbenpaare, die eine Zeile bilden könnten! Implementieren Sie dazu die statische Funktion `findCombinations(Shard[] words)` in der Klasse `Archeology`. Alle Scherben sind ausschließlich mit Großbuchstaben beschriftet. Geben Sie als Ergebnis die Indexpaare aller möglichen Kombinationen von Scherben aus. Halten Sie sich dabei an die Wünsche des Archäologen. Achten sie auf eine möglichst effiziente Implementierung. Die Laufzeit sollte in $\mathcal{O}(n \cdot k^2)$ liegen, wobei n = Anzahl der Scherben; k = Länge der längsten Beschriftung.
Hinweis: Viele der Java-internen Datenstrukturen können wegen der Einschränkungen nicht genutzt werden. Implementieren Sie eine eigene Datenstruktur die Ihnen bei der Lösung hilft.