SWE 4x Übung zu Softwareentwicklung mit modernen Plattformen 4 — SS 2022, Übung 3

Abgabetermin: SA 8:00 in KW 16

□ **Gr. 1**, **M. Winkler, BSc MSc** **Name** _____ **Aufwand in h** _____

□ **Gr. 2**, **Dr. E. Pitzer**

Punkte _____ **Kurzzeichen Tutor / Übungsleiter** _____ / _____

Beispiel	L Lösungsidee	I Implementierung	T Tests	S = L+I+T	M Multiplikator	S*M
a	□□	□□□	□□□□□		4	
b	□□□	□□□□	□□□		4	
c	□□	□□	□□□□□□		2	
	Bitte Ankreuzen!			**Summe (Erfüllungsgrad)** **Bitte auch online ausfüllen!**		

Bitte erstellen Sie für diese Übung ein ZIP-Archiv in dem im Unterverzeichnis „`doc`“, die gesamte Dokumentation (inkl. Projektbeschreibung, Lösungsidee und formatiertem Quelltext) als PDF vorliegt, sowie der Quelltexte in den Angegebenen Unterverzeichnissen des Verzeichnis „`src`“.

Denken Sie bitte auch an die Angabe des Aufwandes auf dem Deckblatt und online!

Erstellen Sie außerdem ein Ant-Build-Script (`build.xml`) mit dem Ihre Abgabe auf der Kommandozeile übersetzt und getestet werden kann.

Prioritätswarteschlangen

In der ersten Übung zum Themengebiet Java haben wir uns mit Prioritätswarteschlangen und deren Implementierung auf Basis der Heap-Datenstruktur beschäftigt. In dieser Übungsaufgabe sollen Sie die Heap-basierte Lösung mit einem alternativen Ansatz vergleichen.

a) Erstellen Sie eine ausführliche Test-Suite in JUnit für die Heap-Implementierung. Überlegen Sie verschiedene Test-Szenarien und testen Sie zusätzlich mit zufällig generierten Werten bzw. Reihenfolgen. Achten Sie auf Randfälle wie z.B. leere und fast leere Warteschlangen, oder das Verhalten bei Elementen mit gleicher Priorität.

Um später eine alternative Implementierung leichter testen zu können, testen Sie gegen die Schnittstelle `PQueue` die Sie dann zur Klasse Heap hinzufügen.

```
public interface PQueue<T extends Comparable<T>> {

  boolean isEmpty();
  T peek();
  void enqueue(T value);
  T dequeue();

}
public class Heap<T extends Comparable<T>> implements PQueue<T> {
}
```

b) Der bisherigen Lösung liegen *binäre Heaps* zugrunde, da jeder innere Knoten (abgesehen vom letzten) genau zwei Nachfolger hat. Eine Verallgemeinerung der binären Heaps stellen *d*-äre Heaps (werden auch als *d*-Heaps bezeichnet) dar, bei denen innere Knoten *d* Nachfolger besitzen

können. Implementieren Sie eine Klasse `DHeapQueue` mit der gleichen Schnittstelle wie der binäre Heap.

c) Führen Sie für die Operationen `enqueue()` und `dequeue()` eine ausführliche Komplexitätsanalyse durch und stellen Sie die Ergebnisse übersichtlich (am besten graphisch) gegenüber:

- Bestimmen Sie zunächst die asymptotische Laufzeitkomplexität aller Operationen in Abhängigkeit von den Parametern *size* und *d*.
- Bestimmen Sie durch Zeitmessungen ein genaueres Maß für die Laufzeitkomplexität der verschiedenen Operationen. Ermitteln Sie jeweils eine Funktion, welche die Größe der Warteschlage auf die damit verbundene Laufzeit abbildet.

  Hinweis: Sie können folgendermaßen vorgehen: Generieren Sie mit Ihrem Java-Programm Messreihen, die Sie auf die Konsole schreiben und in einer Datei speichern. Wählen Sie ein Ausgabeformat, das Sie einfach in z.B. Excel importieren können und führen Sie die Komplexitätsanalysen dort durch. In Excel können die gemessenen Ergebnisse übersichtlich grafisch dargestellt werden. Auch die Berechnung von Regressionskurven ist in Excel relativ einfach möglich.
- Untersuchen Sie für *d*-äre Heaps, welchen Einfluss der Parameter *d* auf das Laufzeitverhalten der verschiedenen Operationen hat und berücksichtigen Sie dabei auch die Größe des Heaps (Parameter *size*).

Hinweis: Zeitmessungen kann man in Java folgendermaßen durchführen:

```
long start = System.nanoTime();
// do some computations here
long time = System.nanoTime() - start;
System.out.printf("time = %f sec%n", time/1000000000.0);
```

Sie dürfen bei dieser Übung (noch) nicht das JDK-Behälter-Framework vollständig verwenden mit Ausnahme der Klasse `ArrayList` und dem Interface `List`, die bereits in der Übung verwendet wurden, sowie der Klasse `java.util.PriorityQueue` zum Vergleich bei den zufällig generierten Testfällen.