Übungen zu Objektorientierte Programmierung 2

frieder.loch@ost.ch

Woche 9: Queue und Heap

Aufgabe 1: Priority Queue

Bei einem Flughafen soll die Reihenfolge bestimmt werden, in der die Flugzeuge landen sollen. Dabei soll immer das Flugzeug als nächstes die Landeerlaubnis erhalten, welches am wenigsten Treibstoff im Tank hat. Um die Aufgabe zu vereinfachen wird angenommen, dass die Flugzeuge während dem Warten keinen Treibstoff verbrauchen. Implementieren sie ein System `LandingOrder.java` mit Hilfe einer Priority Queue `PriorityQueue.java`, welche Sie selber implementieren.

Die Priority Queue sollen Sie auf einer LinkedList von Java aufbauen. Dazu implementieren Sie die deklarierten Methoden in der entsprechenden Klasse. Ihr System (`LandingOrder.java`) soll die folgende Methode zur Verfügung stellen um neue Flugzeuge in die Warteschlange einzufügen:

```
public void addAirplane(Airplane airplane);
```

Ausserdem soll die folgende Methode implementiert werden, welche das nächste zu landende Flugzeug aus der Warteschlange herausnimmt und es zurückgibt:

```
public Airplane nextAirplane();
```

Aufgabe 2: Adaptable Priority Queue mit einer LinkedList

Implementieren Sie eine Adaptable Priority Queue mit Hilfe einer LinkedList. Dabei soll die `Entry<K, V>` Klasse als Node fungieren. Implementieren Sie alle Methoden des `IAdaptablePriorityQueue` Interface.

```
public class Entry<K extends Comparable<? super K>, V> {

    private K key;
    private V value;
    private Entry<K, V> next;
    ...
}
```

Eine Adaptable Priority Queue hat zusätzlich zu den Funktionen einer Priority Queue noch folgende Methoden:

```
  Entry<K, V> remove(Entry<K, V> entry);

  boolean replaceValue(Entry<K, V> entry, V value);

  boolean replaceKey(Entry<K, V> entry, K key);
```

Aufgabe 3: Ring-Buffer

Von einem Sensor werden periodisch Daten gesammelt und gespeichert. Die Sensordaten sollen dabei in einem Ring-Buffer gespeichert werden. Implementieren Sie dazu das IRingBuffer Interface.

Aufgabe 4: Heapsort

Implementieren Sie die beiden Methoden `heapify` und `sort` in der Klasse `Heapsort`. Beide Methoden sollen das gegebene Array **in-place** verarbeiten, d. h. es darf kein zusätzlicher Speicher für Kopien verwendet werden.

Eine Besonderheit dieser Implementierung ist, dass es sich nicht um einen klassischen Min- oder Max-Heap handelt. Stattdessen soll die Ordnung durch einen übergebenen **Comparator** bestimmt werden. Dadurch kann sowohl aufsteigend als auch absteigend sortiert werden oder sogar benutzerdefinierte Ordnungen angewendet werden.

heapify

Diese Methode soll ein gegebenes Array in einen Heap umwandeln – gemäss der Ordnung, die durch den Comparator vorgegeben ist.

Hinweis: Der Index des letzten inneren Knotens (d.h. des letzten Knotens mit mindestens einem Kind) beträgt:

$$\left\lfloor \frac{n}{2} \right\rfloor - 1.$$

sort

Die Sortiermethode soll das Array mithilfe des Heaps sortieren – in Anlehnung an das Vorgehen aus der Vorlesung. Dabei wird der Heap genutzt, um schrittweise das jeweils grösste bzw. kleinste Element (abhängig vom Comparator) an das Ende des Arrays zu verschieben. Anschliessend muss die Heap-Eigenschaft durch Percolate-Down an der neuen Wurzel wiederhergestellt werden.