

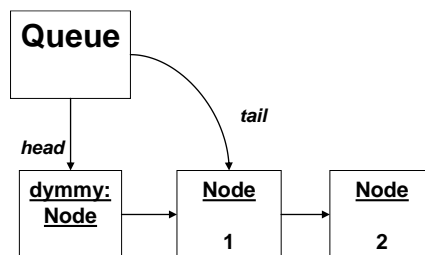
## Übung 6: Lock-Free Algorithms

### Aufgabe 1: LinkedList

In der Vorlesung haben wir die Implementierung des Lock-Free LinkedList Algorithmus von Michael & Scott diskutiert. Sie finden den Code, so wie er im Buch Java Concurrency in Practice abgedruckt ist, auch auf dem AD im Projekt 06\_AS\_Lock\_Free.zip.

In dieser Klasse ist nur die Methode put implementiert. Ergänzen Sie diese Klasse und implementieren Sie zusätzlich die Methode get. Diese ist etwas einfacher, da beim get nur das Head-Element zurückgegeben werden muss. Aber auch beim get müssen die Invarianten eingehalten werden.

Es könnte z.B. passieren, dass in eine Queue mit einem Element ein zweites Element "halb" eingefügt ist.



Überlegen Sie sich was passiert, wenn Sie in dieser Situation zwei get() Operationen ausführen. Wenn beide Operationen erfolgreich ausgeführt werden, so werden die Elemente 1 und 2 zurückgegeben und die Queue muss weiterhin in einem konsistenten Zustand sein.

Bemerkung:

- Als Dummy-Element wird jeweils das nächste Element verwendet, d.h. nach der ersten get-Operation ist der Knoten Node:1 das neue Dummy Element.
- Falls die Queue leer ist, dann soll eine NoSuchElementException geworfen werden.

### Aufgabe 2: CopyOnWriteList

In dieser Aufgabe programmieren Sie eine CopyOnWriteList. Diese Datenstruktur heisst so, weil jede modifizierende Operation auf einer Kopie der zugrundeliegenden Datenstruktur gemacht wird. Die geänderte Datenstruktur wird nach der Änderung „live“ geschaltet.

Vorteil: Wenn ein Thread die Liste gerade iteriert und ein anderer Thread ein add Aufruft, wird dadurch der Iterator des ersten Threads nicht invalidiert (keine ConcurrentModificationException).

Hier sehen Sie das Interface:

```

public interface CoWList<E> extends java.util.Iterable<E> {
    Iterator<E> iterator();
    void addFirst(E e);
    void removeFirst();
    int size();
}
  
```

Implementieren Sie das obige Interface mittels einer AtomicReference und verwenden sie als zugrundeliegende Datenstruktur eine java.util.LinkedList. Wrappen Sie den Iterator und unterbinden Sie die remove Methode (werfen Sie eine UnsupportedOperationException) da sonst die Liste über den Iterator geändert werden könnte. Als Alternative können Sie das remove auch implementieren so dass das zuletzt von next zurückgegebene Element aus der Liste gelöscht wird (auch bei dieser modifizierenden Operation wird die zugrundeliegende Datenstruktur kopiert).