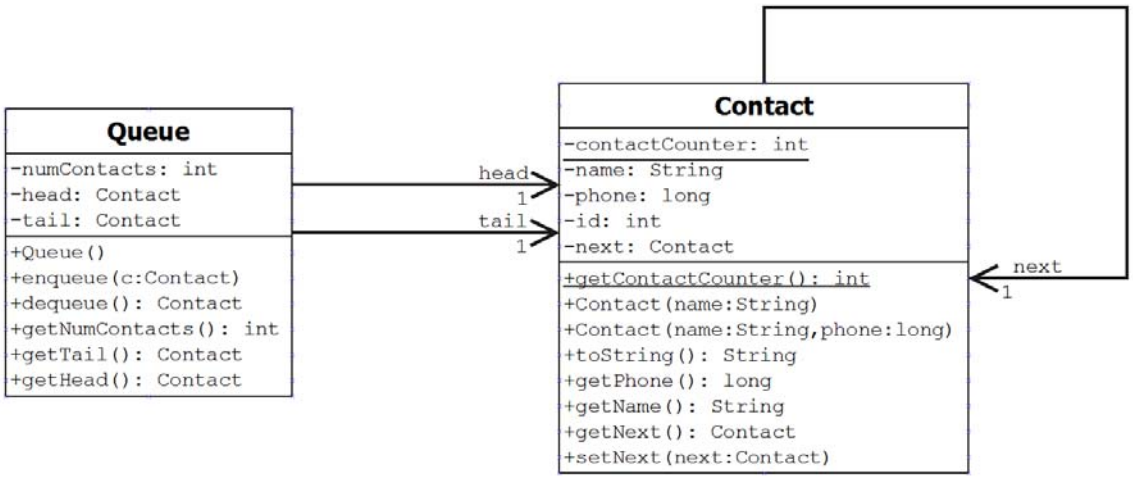
Übung 06: Warteschlangen, JUnit

Warteschlangen (engl. Queue) arbeiten nach dem ***First In – First Out (FIFO)*** Prinzip. In der Praxis kommen Warteschlangen z.B. bei Druckaufträgen zum Einsatz. Der älteste Druckauftrag wird immer zuerst bearbeitet, erst dann der zweitälteste, etc. Es können zu jeder Zeit neue Druckaufträge hinzukommen.

Auf diesem Übungsblatt soll die Grundlage für ein Ticketsystem implementiert werden. Personen, die zuerst ein Ticket ziehen, werden auch zuerst bedient. Dabei wird eine Person durch die vorgegebene und bereits bekannte Klasse `Contact` repräsentiert. Sie sollen eine Klasse `Queue` implementieren, die die folgenden Methoden bereitstellt.

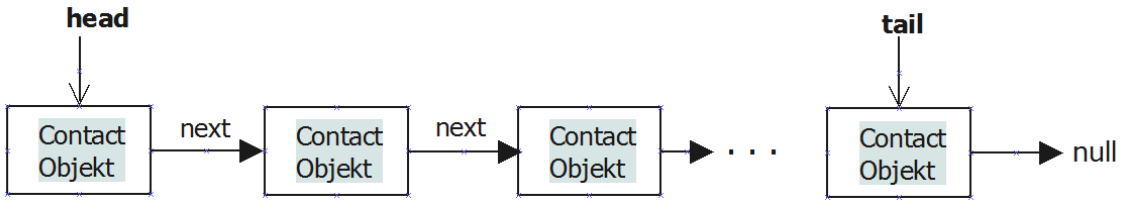
- `public Queue():` Default-Konstruktor, legt eine neue, leere Warteschlange an.
- `public void enqueue(Contact c):` Fügt der Warteschlange einen neuen Kontakt hinzu.
- `public Contact dequeue():` Entfernt den ältesten Kontakt aus der Warteschlange.

Neben diese 3 Methoden sollen ***Getter-Methoden*** gemäß dem dargestellten UML-Klassendiagramm implementiert werden.



Die Warteschlange wird über eine einfach verkettete Liste wie auf dem letzten Übungsblatt umgesetzt. Allerdings kann man von einem Objekt der Klasse `Queue` nun zu ***zwei*** Objekten der Klasse `Contact` „navigieren“:

- **`head:`** Zeigt auf den ältesten Kontakt, der als nächstes entfernt wird.
- **`tail:`** Zeigt auf den jüngsten Kontakt, der als letztes hinzugefügt worden ist.



Von jedem Kontakt gelangt man wie auf dem letzten Übungsblatt durch das Attribut `next` zum nächsten Kontakt.

(umblättern)

Aufgabe 1: Implementierung der Datenstruktur

Implementieren Sie die Klasse `Queue` gemäß den Vorgaben. Die Klasse `Contact` ist bereits vorgegeben.
Tipp: Setzen Sie in der Methode `enqueue(.)` vorsichtshalber das Attribut `next` des einzufügenden Kontaktes `c` auf `null`.

Aufgabe 2: Javadoc

Versehen Sie die Klasse `Queue` mit Javadoc-Kommentaren und erzeugen Sie die Javadoc-Dokumentation für die Klasse `Queue`.

Aufgabe 3: JUnit4

Testen Sie Ihre Implementierung der Klasse `Queue`, in dem Sie JUnit-Tests in der Testklasse `QueueTest` schreiben. Implementieren Sie die in der folgenden Tabelle aufgelisteten Testmethoden.

Hinweise:

- IntelliJ kann die Testklasse automatisch generieren, siehe Folie 13 der Vorlesung.
- Für jede Testmethode sollen *neue, eigene* Objekte (auch Kontakte) angelegt werden.
- Verwenden Sie `assertTrue, assertEquals, assertSame`, … Anweisungen!

Methode	Testbeschreibung
`setUp`	***Vor jeder*** Testmethode soll 1 Objekt der Klasse `Queue` und 3 Objekte der Klasse `Contact` angelegt werden. Referenzen auf diese 4 Objekte werden in Attributen der Testklasse gespeichert und sind somit in jeder Testmethode verfügbar.
`testCreateQueue`	Nach dem Erzeugen eines Objekts `Queue` enthält die `Queue` 0 Kontakte (`numContacts`) und sowohl `head` als auch `tail` sind `null`.
`testEnqueue`	Nach dem Einfügen von 3 Kontakten enthält eine `Queue` 3 Kontakte (`numContacts`). Der erste Kontakt (`head`) der Liste ist der zuerst eingefügte Kontakt, der letzte Kontakt (`tail`) das zuletzt in die Liste eingefügte Element.
`testDequeue`	Aus einer `Queue` mit 3 Kontakten, wird ein Kontakt entnommen. Der entnommene Kontakt ist der zuerst eingefügte Kontakt. Die Liste hat nun 1 Kontakt weniger. `head` und `tail` zeigen auf den richtigen Kontakt.
`testDequeueLastElement`	Auf einer `Queue` mit 1 Kontakt, wird `dequeue` aufgerufen. Die `Queue` ist danach leer, `numContacts` ist 0. `head` und `tail` sind jeweils `null`.
`testPerformance`	Das Einfügen von 1000 Kontakten darf nicht länger als ***50 ms*** dauern.
`testContactCounter`	Das Erzeugen eines Kontaktes soll die Klassenvariable `ContactCounter` der Klasse `Contact` um genau 1 erhöhen.