# W06H02 - PUM Server Upgrade

Hausaufgabe Medium

Points: 6 of 6 Assessment: automatic

Submission due: 8 months ago

Complaint due: 8 months ago

Resume practice in exercise

100% (8 months ago) GRADED

## Recent results:

0% (9 months ago) GRADED

100% (8 months ago) GRADED

100% (8 months ago) GRADED

Show all results

Tasks:

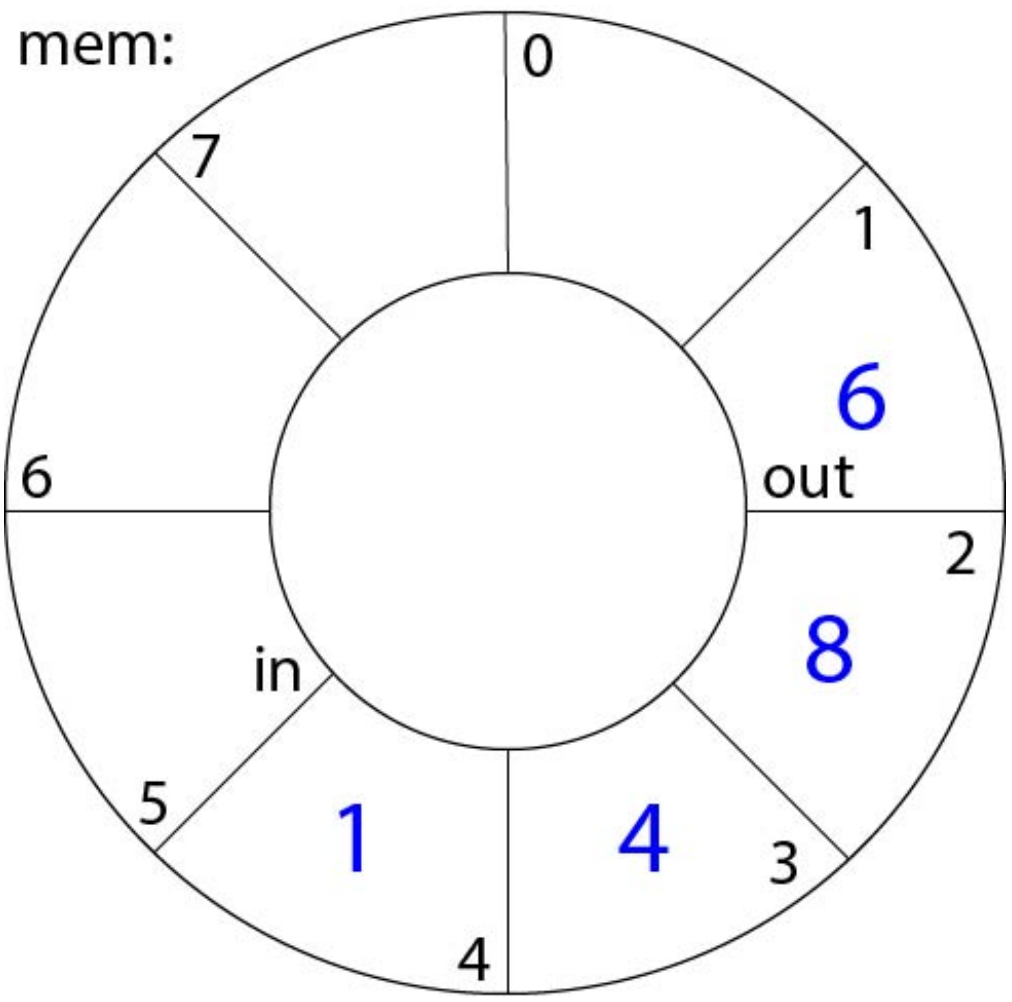
## PUM Server Upgrade

Die Forschuine der PUM sind begeistert von unserem LRZ. Um noch besser forschen zu können, wollen sie nun auch einen eigenen Hochleistungsrechner bauen. Die Softwareuine wurden beauftragt, einige notwendige Komponenten zu implementieren. Als exzellenter Freund unterstützt du sie dabei.

## RingBuffer

Um Daten asynchron zwischen den Server-Nodes hin und her zu senden, benötigen die Pinguine einen effizienten Buffer, genauer einen `RingBuffer`. Ein `RingBuffer` wird mit seiner Kapazität initialisiert (das ist die maximale Anzahl an Einträgen, die gespeichert werden kann). In den `RingBuffer` können dann Werte gelegt (`put(int)`) und aus ihm wieder entnommen (`get()`) werden.

Hier siehst du eine Visualisierung eines `RingBuffer`s:




Wie du siehst, speichern wir die Werte in einem Array (`mem`). Um festzuhalten, in welchem Bereich valide Daten gespeichert sind, benutzen wir die beider Felder `in` und `out`. `in` speichert dabei den Index, an dem der nächste Wert, der via `put(int)` eingefügt werden soll, in `mem` abgespeichert wird. `out` zeigt auf den Index, von dem bei der nächsten `get()`-Operation gelesen wird. Durch eine Sequenz von `put`- und `get`-Operationen kann dieser Bereich also durch das Array "wandern", auch über die Grenzen des Arrays hinaus. Dazu musst du dir vorstellen, dass das Array ein Ring ist. Wir interpretieren also `mem[0]` als Nachfolger von `mem[mem.length-1]` und `mem[mem.length-1]` als Vorgänger von `mem[0]`.

Die folgenden Teilaufgaben beschreiben die Implementierung step-by-step:

1. **Als aller Erstes!** No results
   Lies dir zunächst die folgenden Teilaufgaben durch und erstelle erst die erforderten Methoden, um sie anschließend zu implementieren. So hast du schonmal Code, den Artemis kompilieren und testen kann. Dies bezieht sich auch auf die Methoden, die in `MultiStack` implementiert werden

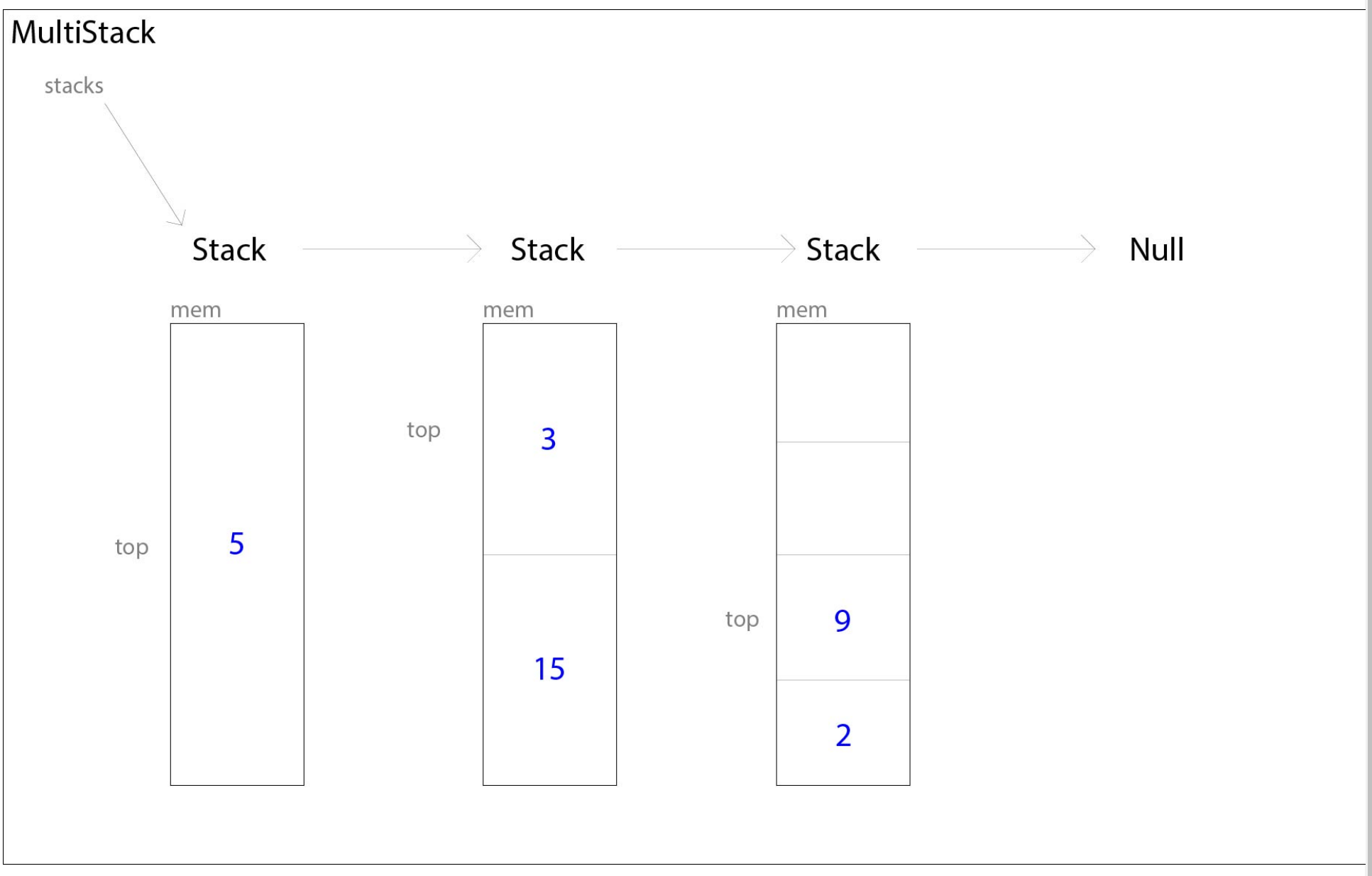
müssen.

2. ? **isEmpty** No results
   Erstelle die Methode `isEmpty`, die einen `boolean` zurückgeben soll: `true` - keine Einträge im `RingBuffer` gespeichert, `false` - einer oder mehr Einträge im `RingBuffer` gespeichert.
3. ? **isFull** No results
   Erstelle die Methode `isFull`, die einen `boolean` zurückgeben soll: `true` - der `RingBuffer` speichert die maximale Anzahl von Einträgen, `false` - der `RingBuffer` speichert weniger als die maximale Anzahl von Einträgen.
4. ? **put** No results
   Erstelle die Methode `put`, die einen `int` erwartet, um diesen im `RingBuffer` zu speichern. Sollte der Buffer voll sein, dürfen keine Werte in den Buffe eingefügt werden, stattdessen akzeptiert der `RingBuffer` den neuen Wert nicht. Ist das Speichern erfolgreich, soll der `boolean true` zurückgegeben werden, andernfalls `false`.
5. ? **get** No results
   Erstelle die Methode `get`, die einen `int` zurückgeben soll. Wenn möglich soll der Wert zurück gegeben werden, der am längsten im `RingBuffer` lieg - dieser wird dann auch aus dem aktuellen Speicherbereich des Buffers entfernt - andernfalls erwarten wir den Default-Rückgabewert `Integer.MIN_VALUE`.
6. ? **Konstruktor & Performance** No results
   Der erste Test dient nur als Sicherheit, damit du sofort erkennst ob du ausversehen den Konstruktor verändert hast. Die Performance wird dir erst nach der Deadline angezeigt. Mach dir aber nicht zu viele Gedanken. Entscheidend ist nur, dass du nicht zu viele irrelevante Operationen ausführst (z.B. Kopien des Arrays erstellst).

## MultiStack

Die Pinguine möchten auf ihrem HPC natürlich viele Java-Programme ausführen. Um komplexere Programme entwickeln zu können, benötigen sie einen Stack, der mehr Daten speichern kann. Ein erster Prototyp dafür soll in den folgenden Teilaufgaben entwickelt werden.

`MultiStack` nutzt die Klasse `Stack`, in der die Daten gespeichert werden. Jeder `Stack` ist dabei wie ein Listen-Element und enthält, abgesehen von dem Array `mem` (zum Speichern der Daten) und `top` (um auf den obersten Wert dieses `Stack`s zu zeigen), auch eine Referenz auf einen (möglichen) Nachfolger `Stack`. Der Beginn dieser Liste von `Stack`s wird als Referenz `stacks` in `MultiStack` gespeichert.




1. ? **Als aller Erstes!** No results
   Lies dir zunächst die folgenden Teilaufgaben durch und erstelle erst die erforderten Methoden, um sie anschließend zu implementieren. So hast du schonmal Code, den Artemis kompilieren und testen kann.
2. ? **push** No results
   Erstelle die Methode `push`, die einen `int` auf den `MultiStack` legen soll. Dabei soll wie folgt vorgegangen werden: Da der Wert in der Liste aus `Stack`s gespeichert wird, müssen wir zunächst den Ort finden, an dem der Wert abgelegt werden soll. Dazu starten wir bei dem ersten `Stack`. Ist dieser bereits gefüllt, führen wir unsere Suche im Nachfolger (`next`) fort bis wir uns im passenden `Stack` befinden. Sollten alle `Stack`s gefüllt sein, müssen wir einen neuen `Stack` mit der doppelten Kapazität des Vorgängers erstellen (Je weiter du durch die Liste iterierst, desto größer werden die Stacks - Faktor 2). Da wir nun den richtigen `Stack` gefunden oder initialisiert haben, können wir in dem entsprechenden `Stack` den Wert in `mem` speichern und das Feld `top` anpassen. *Tipp:* Dieses Verfahren kann am einfachsten rekursiv implementiert werden, d.h. der `MultiStack` ruft die gleichnamige rekursive Methode in `Stack` auf. (Rekursion ist hier aber nicht unbedingt verlangt.)
3. ? **top** No results

Erstelle die Methode `top`, die den Wert (`int`) zurückgeben soll, der ganz oben auf dem `MultiStack` liegt. (Das ist der Wert, der als letztes eingefügt wurde.) Der Wert soll dabei *NICHT* entfernt werden. Ist der Stack leer, erwarten wir den Default-Rückgabewert `Integer.MIN_VALUE`. *Tipp:* Auch diese Teilaufgabe lässt sich sehr einfach rekursiv lösen.

4. ? **pop** No results
   Erstelle die Methode `pop`, die den Wert (`int`) zurückgeben *UND* entfernen soll, der ganz oben auf dem `MultiStack` liegt. (Das ist der Wert der als letztes eingefügt wurde.) Ist der letzte Stapel der `Stack`-Liste leer, soll dieser aus der Liste gelöscht werden. Ausgenommen ist davon der erste Stac mit Kapazität `1`. Dieser soll immer erhalten bleiben, auch wenn er aktuell keine Werte speichert. (Mach dir gerne Gedanken darüber, warum.) Ist der Stack leer, erwarten wir den Default-Rückgabewert `Integer.MIN_VALUE`. *Tipp:* Wie du dir bestimmt denken kannst, kannst du auch diese Teilaufgabe durch eine rekursive Methode in `Stack` lösen.
5. ? **Konstruktor** No results
   Der Test dient nur als Sicherheit, damit du sofort erkennst ob du ausversehen den Konstruktor verändert hast.

? **Öffentliche Tests** No results
Testet deine Abgabe nach jedem Push. Hier siehst du, ob du bereits alle öffentlichen Tests bestehst bzw. welche du noch nicht bestehst.

? **Versteckte Tests** No results
Testet deine Abgabe nach der Deadline. Diese Tests sind auch in die obigen Tasks an den entsprechenden Stellen integriert.

? **Grading Tests** No results
Testet deine Abgabe nach der Deadline und verteilt Punkte.

Viel Erfolg!

Lösungsvorschlag (nach der Deadline)

Tests (nach der Deadline)

| Exercise details | |
|---|---|
| Release date: | Nov 24, 2022 18:30 |
| Submission due: | Dec 11, 2022 18:00 |
| Complaint due: | Dec 18, 2022 18:00 |

Every student is allowed to complain once per exercise. In total 1000 complaints are possible in this course. You still have **998** complaints left.