Aufgabe 7.1
Die Stapel-Operationen lassen sich auch mathematisch formulieren. Dabei wird - anders als in der Java-Syntax - der Stapel s als Parameter an die Funktion übergeben.
Die Semantik der Operatioenn lässt sich dann durch Bedingungen an diese Funktionen beschreiben, wie zum Beispiel:

- $isempty(push(s, e)) = false$
- $top(push(s, e)) = e$
- $top(pop(push(s, e))) = top(s)$

a) Schreiben Sie JUnit-Tests, mit denen diese Bedingungen getestet werden können.

b) Implementieren Sie den ADT Stapel (nach Ihrer Wahl mittels einer EVL oder eines dynamischen Arrays).
Testen Sie Ihre Implementierung mit den JUnit-Tests aus Aufgabe a).

c) Tauschen Sie mit Kommilitonen Implemtierungen aus (wenn Sie den Stapel mit EVL implementiert haben, dann lassen Sie sich eine Implementierung als DynArray geben und umgekehrt. Testen Sie auch die „fremde“ Implementierung.

Aufgabe 7.2
Entwickeln Sie einen Algorithmus, der zwei int-Stacks benutzt, um eine Folge von Integer-Werten zu sortieren. Die Grundidee ist eine Variante von „Sortieren durch Einfügen“:
Zu Beginn sind beide Stapel leer. Im Verlauf des Verfahrens werden die beiden Stapel so gefüllt, dass immer

- alle Werte im ersten Stapel kleiner sind als alle Werte im zweiten Stapel
- die Werte im ersten Stapel *aufsteigend* sortiert sind, sodass der größte Werte „oben“ steht
- die Werte im zweiten Stapel *absteigend* sortiert sind, sodass der kleinste Wert „oben“ steht

Zum Einfügen eines neuen Wertes e wird der Wert mit den top-Elementen der beiden Stapel verglichen, um so zu entscheiden, in welchen der beiden Stapel e eingefügt werden muss. Dann müssen ggf. Elemente von einem Stapel auf den anderen Stapel „umgeschichtet“ werden, um die richtige „Einfügeposition“ zu finden.

Implementieren (und testen) Sie eine statische Methode **void** stackSort (**int** [ ] arr), die ein unsortiertes int-Array als Eingabe erhält. Die Methode soll zwei Stapel-Objekte als lokale Variable benutzen, um die int-Werte nach dem oben dargestellten Verfahren zu sortieren. Nach dem Sortiervorgang schreibt die Methode die Inhalte der beiden Stacks in (sortierter) Reihenfolge in das Array zurück.

Implementieren Sie „gegen das Interface“!
Das Verfahren sollte nicht davon abhängen, ob Sie für die beiden Stapel eine Array-Implementierung oder eine Listen-Implementierung (oder auch für beide Stapel unterschiedliche Implementierungen) gewählt haben.

Aufgabe 7.3 (Theorie)
Eine klassische Anwendung von Stacks ist die Auswertung von (arithmetischen) Ausdrücken.

Dazu gehen wir davon aus, dass der Ausdruck, den wir auswerten wollen, *vollständig geklammert* ist (also keine Vorrangregeln wie „Punkt vor Strich“ ausnutzt, um Klammern zu sparen) und dass alle Operator-Symbole *zweistellig* sind, also zwei Operanden benötigen.

Beispiel-Ausdruck: $((6 * (4 * 28)) + (9 - ((12/4) + 2)))$

Für die *Auswertung* eines solchen Ausdruck, also die Berechnung des Wertes, benutzen wir zwei Stacks (Stapel), nämlich einen *Operanden*-Stack (für die Zahlenwerte) und einen *Operator*-Stack (für die Rechenzeichen). Wir nehmen an, dass der Ausdruck als Folge von Zahlkonstanten, Klammern und Rechenzeichen gegeben ist.

Die Auswertung geschieht dann nach folgendem Algorithmus:

- Solange noch Zeichen in der Folge sind, lies das nächste Zeichen aus der Folge.
- Falls dieses Zeichen
  - eine öffnende Klammer ist: ignoriere es
  - eine Zahlkonstante ist: lege diesen Wert auf den Operandenstack
  - ein Operatorzeichen ist: lege es auf den Operatorstack
  - eine schließende Klammer ist:
    - nimm den obersten Operator vom Operatorstack
    - nimm die obersten beiden Werte vom Operandenstack
    - werte diesen Teilausdruck aus
      (Achtung bei nicht-kommutativen Operatoren wie zB minus: Reihenfolge der Operanden beachten!)
    - lege das Ergebnis auf den Operandenstack

Am Ende sollte der Operatorstack leer sein und der Operandenstack noch genau einen Zahlenwert enthalten. Dies ist dann der Wert des Ausdrucks.
Falls eine dieser beiden Bedingungen nicht erfüllt ist, war der Ausdruck syntaktisch nicht korrekt.

Beispiel für die Auswertung des Ausdrucks $((2 + 3) * 4)$
(Die Stacks lesen wir von links nach rechts: das top-Element steht rechts. Wir müssen in jeder Zeile alle drei Spalten füllen, weil wir sonst „keine Änderung“ nicht von „leerer Stack“ unterscheiden können.)

gelesenes Zeichen	Operanden-stack	Operator-stack
(	*empty*	*empty*
(	*empty*	*empty*
2	2	*empty*
+	2	+
3	2 3	+
)	5	*empty*
*	5	*
4	5 4	*
)	20	*empty*

Jetzt die Aufgabenstellung:

Erstellen Sie für den Ausdruck $((6 * (4 * 28)) + (9 - ((12/4) + 2)))$ eine solche „Tracetabelle“ für die Inhalte der beiden Stacks.

Aufgabe 7.4
Implementieren Sie den ADT Schlange

a) in einer Klasse SchlangeAlsArray<T> mittels eines dynamischen Arrays

b) in einer Klasse SchlangeAlsEVL<T> mittels einer verketteten Liste

c) in einer Klasse SchlangeAlsRingpuffer<T> mittels eines Ringpuffers „ohne Überschreiben“.
Die Kapazität des Puffers soll dem Konstruktor der Klasse SchlangeAlsRingpuffer<T> als Parameter übergeben werden. Falls der Puffer voll ist, löst eine weitere enqueue()-Operation eine „PufferVoll“-Exception aus.

Aufgabe 7.5
Wir wollen im Folgenden einen Temperatursensor modellieren. Dieser Sensor soll Temperaturwerte (vom Typ Float für die letzten 24 Stunden in einem Ringpuffer speichern. Er kann die aktuelle (zuletzt gemessene) Temperatur liefern und die Durschnittstemperatur der letzten Messungen (also maximal der letzten 24 Stunden). Werden diese Werte vom Sensor abgefragt, wenn der Sensor leer ist, soll NaN geliefert werden (siehe bspw. Java-API zur Klasse Float). Wenn der Speicher des Sensors voll ist, soll der älteste Wert überschrieben werden. Außerdem soll es die Möglichkeit geben den Sensor zu flushen, also den Speicher zu leeren.
Schreiben Sie hierzu eine Klasse Temperatursensor mit folgenden Methoden:

- neueMessung(Float wert): diese fügt einen neuen Wert im Ringpuffer hinzu.
- aktuelleTemperatur (): gibt den aktuellsten Wert zurück. Sollte der Temperatursensor keine Messung gemacht haben, soll Float.NaN zurückgegeben werden.
- durchschnittsTemperatur (): gibt den Durchschnitt aller Messungen zurück. Sollte der Temperatursensor keine Messungen gemacht haben, soll Float.NaN zurückgegeben werden.
- reset (): soll alle Werte aus dem Ringpuffer entfernen - der Zustand ist nach Aufruf also nach außen so, als wäre es ein leerer Puffer.

Aufgabe 7.6 (Theorie)
Wie entwickelt sich im Speicher das interne Array eines Integer-Ringpuffers mit capacity = 6 unter folgenden Operationen?

insert (1); insert (2); insert (3); remove(); remove(); insert (4); insert (5);
remove(); insert (6); insert (7); insert (8); insert (9); remove();

Verwenden Sie einen „Pointer“ p, der den Index des jeweils ersten (also „ältesten“) gültigen Eintrag angibt (vgl Folie 7-16). Geben Sie in einer Tabelle die Werte von p, $size$ und alle Array-Inhalte an. Lassen Sie nicht mehr gebrauchte Feldkomponenten unverändert, solange sie nicht überschrieben werden.