Übung 5 – Abstrakte Datentypen

Arbeiten Sie im Skript das *Kapitel Abstrakte Datentypen* durch. Suchen Sie sich weitere Informationen, wo Sie die Ausführungen nicht verstehen. Sprechen Sie mit Ihren Kommilitoninnen und Kommilitonen. Fragen Sie im Forum.
Lesen Sie die Aufgaben vollständig und markieren Sie sich zentrale Aspekte. Verwenden Sie **keine Klassen der Java API** außer als Test für Ihre Implementationen und beachten Sie konsistent die Zugriffrechte.

Hinweis: Zeichnen Sie vor der Entwicklung des Codes UML-Diagramme und machen Sie sich besonders für `insert`-Methoden vorab ein Speicherbild um die Funktionsweise zu verbildlichen.

Aufgabe 1 (alte Klausuraufgabe)

Erörtern Sie die Vor- und Nachteile der Implementierung des abstrakten Datentyps **Schlange** als:

1. Array (Feld) fester Größe
2. dynamisches Array, dessen Größe wachsen und schrumpfen kann
3. einfach verkettete Liste
4. doppelt verkettete Liste

Aufgabe 2

Schreiben Sie für den ADT Folge ein generisches Interface `Folge<T>`, das auch das generische Interface `Puffer` aus den vorigen Übungen erweitert:
`Folge` garantiert wahlfreien Zugriff und bietet zusätzlich zu den von `Puffer` geerbten Methoden, die Methoden `get(int pos)` und `set(int pos, T e)` und außerdem die Methoden `remove(int pos)` und `insert(int pos, T e)` an:

- Beim Entfernen von Elementen mit `remove(int pos)` soll das Element am Index `pos` entfernt werden und alle darauf folgenden Elemente nachrutschen.
- `insert(T e)` fügt ein Element am Ende der Folge an.
- `remove()` soll das erste Element in der Folge entfernen (*Welches Prinzip haben wir hier als Nebenprodukt auch umgesetzt?*).
- Bei der Methode `insert(int pos, T e)` wird `e` an der spezifizierten Position `pos` eingefügt. Das Element, das momentan an der Position steht und die folgenden (wenn vorhanden) werden nach rechts verschoben (die Indizes sind dann jeweils um eins erhöht).

- `set(int pos, T e)` überschreibt das Element an Index `pos` und gibt den **alten** Wert zurück.
- Der Wert von `pos` muss für `remove(int pos)`, `insert(int pos, T e)`, `set(int pos, T e)` und `get(int pos)` größer gleich 0 und kleiner als `size()` sein, sonst wird eine `IndexOutOfBoundsException` geworfen. Definieren Sie den Sonderfall für das Einfügen des ersten Elementes.

Skizzieren Sie ein UML-Diagramm mit allen Interfaces, die wir jetzt haben. Die Interfaces entsprechen den jeweiligen ADT.

Aufgabe 3

Nehmen Sie Ihre generischen Klassen für die Datenstrukturen `Ringpuffer` und `DynArray` aus dem vorherigen Aufgabenblatt und implementieren Sie den abstrakten Datentypen `Folge`. Fügen Sie dazu die noch fehlenden Methoden (`remove(int pos)` und `insert(int pos, T e)`) in Ihren Datenstrukturen hinzu. Stellen Sie auch hier für Ihre Implementierung mit `DynArray` einen parameterlosen Konstruktor zur Verfügung und für Ihre Implementierung mit `Ringpuffer` einen Konstruktor, der einen ganzzahligen Wert `capacity` annimmt und eine `Ringpuffer`-Instanz mit einer entsprechend großen Kapazität anlegt.

1. Implementieren Sie in Klasse `FolgeMitRing<T>` die Schnittstelle `Folge<T>` mittels eines Ringpuffers.
2. Implementieren Sie in Klasse `FolgeMitDynArray<T>` die Schnittstelle `Folge<T>` mittels eines dynamischen Arrays.
3. Testen Sie Ihre Implementierung mit JUnittests.

Aufgabe 4

Nehmen Sie Ihre generische Klasse für die Datenstruktur `Ringpuffer` und implementieren Sie den abstrakten Datentypen `Schlange`. Stellen Sie für Implementierung mit `Ringpuffer` einen Konstruktor, der einen ganzzahligen Wert `capacity` annimmt und eine `Ringpuffer`-Instanz mit einer entsprechend großen Kapazität anlegt.

1. Implementieren Sie in einer generischen Klasse `SchlangeMitRing<T>` die Schnittstelle `Schlange<T>` mittels eines Ringpuffers.
2. Testen Sie Ihre Implementierung mit JUnittests.
3. Führen Sie den gleichen Versuchsaufbau `TimeTestSchlange` wie in Blatt 3 für Ihre hier implementierten Klassen `FolgeMitDynArray`, `SchlangeMitRing` und `SchlangeMitEVL` aus einer vorigen Übung durch. *Was stellen Sie zeitlich fest? Was sind Vor- und Nachteile? Welchen Speicherverbrauch haben die Datenstrukturen theoretisch im Vergleich (Erinnern Sie sich wieviel Byte hat ein Integer)?*

Zusatzaufgabe (Programmierübung Projekt Euler)

Jeder neue Term in der Fibonacci-Reihe wird gebildet, indem die beiden vorherigen Zahlen addiert werden. Wenn man mit 1 und 2 beginnt, sind die ersten 10 Terme wie folgt:

$$1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...$$

Finden Sie die Summe aller geraden Terme der Fibonacci-Reihe, die 4 Millionen nicht überschreiten.

`https://projekteuler.de/problems/2`
Lösen Sie das Problem in einer Klasse `Euler2`. Geben Sie sowohl die Summe, als aus die Anzahl der Terme (obiges Beispiel 10) aus.