TU Kaiserslautern – FB Informatik
Dr. Patrick Michel

17.12.2019

# Algorithmen und Datenstrukturen (WS 2019)

# Aufgabenblatt 7

**zu bearbeiten bis: 06.01.20 / 08.01.20**

---

Auf diesem Blatt fangen wir an unsere Bibliothek an Standardtypen aufzubauen. Gegeben sind dazu die Schnittstellen der **abstrakten Datentypen** (ADTs) `List`, `Stack` und `Queue`, die wir aus den Übungen oder der Vorlesung kennen. Für alle ADTs gelten die folgenden Regeln:

- Sind sind parametrisch im Typ `T` ihrer enthaltenen Elemente (Java Generics).
- ADT Definitionen (Java Interfaces) liegen im Paket `exercise.adt`.
- Konkrete Implementierungen (Java Classes) liegen im Paket `exercise.adt.impl`.

Für Abgaben werden Sie auf diesem Blatt (und in Zukunft) Ihre Implementierungen hochladen (und in Zukunft die von Ihnen zur Lösung verwendeten Implementierungen), die Schnittstellen sind bereits im System und brauchen nicht hochgeladen zu werden.

**Aufgabe 7.1 (ADT - Listen - Praxis)**
Den Anfang machen wir mit dem Interface `List`, welches Listen von Elementen repräsentiert. Schauen Sie sich die gegebene Schnittstelle an und vergleichen Sie mit Ihrer `ArrayHelper` Implementierung von Blatt 03. **Kopieren** Sie dann Ihre `ArrayHelper` Implementierung in den Ordner `app/exercise/adt/impl`, benennen Sie Datei und Klasse in `ArrayList` um und implementieren Sie die `List` Schnittstelle. Der Kopf Ihrer Datei wird dann so aussehen:

```
package exercise.adt.impl;

import exercise.Exercise;
import exercise.adt.List;

@Exercise
public class ArrayList<T> implements List<T> {
    private T[] data;
    private int next;
    ...
}
```

Hinweise zur Implementierung:

- Ersetzen Sie den konkreten Typ `Person` in der gesamten Datei duch die Typvariable `T` (wie oben angefangen).
- Um in Java ein Array vom Typ `T[]` der Länge `len` anzulegen können Sie folgenden Ausdruck verwenden: `(T[]) new Object[len]`.

- Entfernen Sie alle Methoden rund um Sortierung und den Typ `Comparator`.
- Ersetzen Sie die Implementierung der Methode `checkCapacity` durch die auf Blatt 03 beschriebene, welche beim Erreichen der Kapazität das Array vergrößert, anstatt eine Exception zu werfen.
- Enffernen Sie die Deklaration der beiden Ausnahmen (`CapacityReachedException` und `NoSuchElementException`), verwenden Sie die in `List` beschriebene Exception.
- Implementieren Sie fehlende Methoden bzw. benennen Sie alte Methoden um, damit sie auf die Schnittstelle von `List` passen.
- Schließlich sollten Sie alle Methoden entfernen, die ein externes Array als `data` übernehmen oder eine Referenz auf das interne `data` Array nach außen geben.

Ob damit die `List` Schnittstelle korrekt implementiert ist können Sie wie immer durch Ausführen der Tests validieren. Stellen Sie sicher, dass Ihre Implementierung mindestens einen Konstruktor ohne Parameter besitzt, so dass die Tests eine Instanz Ihrer Implementierung erzeugen können.

**Aufgabe 7.2 (ADT - Stacks und Queues - Praxis)**
Wir implementieren nun die beiden ADTs `Stack` und `Queue`, beide mit (theoretisch) unbegrenzter Größe. Beide Datentypen können sehr einfach durch `ArrayList` implementiert werden.

- Implementieren Sie in `ArrayList` nun **zusätzlich** zu `List` auch das Interface `Stack`. Fügen Sie dazu die neuen Operationen zu der Klasse hinzu und rufen Sie die entsprechend passende Methode von `List` auf.
- Implementieren Sie in `ArrayList` nun **zusätzlich** das Interface `Queue`.
  Fügen Sie dazu wieder die neuen Operationen zu der Klasse hinzu und rufen Sie jeweils die passende Methode von `List` auf.
- Bewerten Sie für beide Implementierungen wie gut diese sind. Geben Sie dazu jeweils die Komplexitäten der Methoden der Interfaces in Ihrer Implementierung an (best/avg/worst).

**Aufgabe 7.3 (ADT - Effiziente Queues - Praxis)**
Implementieren Sie nun das Interface `Queue` erneut, diesmal in der Klasse `DoubleStackQueue`. Die Idee der Klasse soll dabei sein:

- Die Implementierung benutzt zwei private Stacks, einen `enqueue` Stack und einen `dequeue` Stack.
- Die `enqueue` Operation wird durch ein `push` auf dem entsprechenden Stack umgesetzt.
- Die `dequeue` Operation wird durch ein `pop` auf dem entsprechenden Stack umgesetzt. Sollte dieser Stack allerdings leer sein, übertragen Sie zuerst alle Elemente des `enqueue` Stacks auf den `dequeue` Stack. (Die Elemente werden dabei in der Reihenfolge invertiert.)
- Die Queue ist genau dann leer, wenn beide Stacks leer sind und die Größe der Queue ist die Summe der Größen der beiden Stacks.

Bewerten Sie die Güte dieser Implementierung, indem Sie wieder die Komplexitäten der Methoden angeben!

---