**Skizze Aufgabe 1:**

Invariante für **alle** ADTs:

* Alle Operationen der ADTs sind objekt-orientiert:  
  a) Konstruktor (K)  
  b) Mutatoren (M)

c) Selektoren (S)

**Aufgabe: 1.1**

Ziel: ADT-Liste implementieren

Folgendes soll immer gelten (Invarianten):

1. funktional
   * Die Liste beginnt bei Position 1
   * Die Liste arbeitet nicht destruktiv
   * Der Element-Typ sind „ganze Zahlen“
2. technisch
   * Interne Realisierung soll mittels Java-Array umgesetzt werden (…new int[?]…)

Die ADT-Liste enthält folgende Objektmengen:

* pos: Position eines Elementes innerhalb der Liste
* elem: Ein Element der ADT-Liste
* list: Die ADT-Liste, mit ggf. enthaltenen Elementen

Folgende Operationen sollen bereitgestellt werden (semantische Signatur):

* *create*: eine leere ADT-Liste erstellen   
  („nichts“ -> list)

Fehlerbehandlung: ignorieren (es wird kein Fehler geworfen)

* *isEmpty*: Abfrage, ob ADT-Liste kein Element enthält   
  (list -> Wahrheitswert)

Fehlerbehandlung: ignorieren

* *laenge*: Abfrage, wie viele Elemente die ADT-Liste enthält   
  (list -> ganze Zahl)

Fehlerbehandlung: ignorieren

* *insert*: Ein Element an selbstgewählter Position in die ADT-Liste einfügen  
  (list x Position x Element -> list)  
  Fehlerbehandlung: ignorieren
* *delete*: Ein Element an selbstgewählter Position aus der ADT-Liste entfernen

(list x pos -> list)

Fehlerbehandlung: Wenn an dieser Position kein Element vorhanden ist, soll  
folgende Exception geworfen werden: „NoElementAtThisPosition“

* *find*: Abfrage, an welcher Position sich ein Element befindet (Abfrage links nach rechts)

(list x elem -> pos)

Fehlerbehandlung: 0 zurückgeben

* *retrieve*: Element aus ADT-Liste an selbstgewählter Position abfragen (Element wird kopiert

und zurückgegeben)

(list x pos -> elem)

Fehlerbehandlung: 0 zurückgeben

* *concat*: zwei ADT-Listen zusammenfügen (linke Liste wird um Elemente der rechten erweitert)

(list1 x list2 -> list1)

Fehlerbehandlung: ignorieren

**Aufgabe: 1.2**

Ziel: ADT-Stack implementieren

Folgendes soll immer gelten (Invarianten):

1. funktional
   * LIFO (jüngstes hinzugefügtes Element wird als erstes zurückgegeben)
2. technisch
   * Der ADT-Stack soll mittels ADT-Liste implementiert werden

Der ADT-Stack enthält folgende Objektmengen:

* elem: Ein Element des ADT-Stacks
* stack: Ein ADT-Stack, mit Elementen

Folgende Operationen sollen bereitgestellt werden (semantische Signatur):

* *createS*: ein leeren ADT-Stack erstellen   
  („nichts“ -> stack)

Fehlerbehandlung: ignorieren (es wird kein Fehler geworfen)

* *push*: ein Element auf den Stack ablegen, dieses Element liegt ganz oben

(stack x elem -> stack)

Fehlerbehandlung: ignorieren

* *pop*: oberstes Element aus dem Stack löschen

(stack -> stack)

Fehlerbehandlung: ignorieren

* *top*: oberstes Element als Kopie zurückgeben

(stack -> elem)

Fehlerbehandlung: 0 zurückgeben

* *isEmptyS*: Abfrage, ob ADT-Stack kein Element enthält   
  (stack -> Wahrheitswert)

Fehlerbehandlung: ignorieren

**Aufgabe: 1.3**

Ziel: ADT-Queue implementieren

Folgendes soll immer gelten (Invarianten):

1. funktional
   * FIFO (ältestes hinzugefügtes Element wird als erstes zurückgegeben)
2. technisch
   * Die ADT-Queue soll mittels ADT-Stack implementiert werden

Die ADT-Queue enthält folgende Objektmengen:

* elem: Ein Element der ADT-Queue
* queue: Ein ADT-Queue, mit Elementen

Folgende Operationen sollen bereitgestellt werden (semantische Signatur):

* *createQ*: eine leere ADT-Queue erstellen   
  („nichts“ -> queue)

Fehlerbehandlung: ignorieren (es wird kein Fehler geworfen)

* *front*: gibt eine Kopie des ältesten Elementes zurück

(queue -> elem)

Fehlerbehandlung: 0 zurückgeben

* *enqueue*: fügt ein Element an letzter Position der ADT-Queue ein

(queue x elem -> queue)

Fehlerbehandlung: ignorieren

* *dequeue*: löscht das älteste Element aus der ADT-Queue

(queue -> queue)

Fehlerbehandlung: ignorieren

* *isEmptyQ*: Abfrage, ob ADT-Queue kein Element enthält   
  (queue -> Wahrheitswert)

Fehlerbehandlung: ignorieren

**Aufgabe: 1.4**

Ziel: ADT-Array implementieren

Folgendes soll immer gelten (Invarianten):

1. funktional
   * das ADT-Array beginnt bei Position 0
   * das ADT-Array arbeitet destruktiv (wird ein Element an einer vorhandenen Position eingefügt, wird das dort stehende Element überschrieben)
   * Die Länge des ADT-Arrays wird durch die Position bestimmt, an der das letzte Element indiziert ist
   * Das ADT-Array ist mit 0 initialisiert (Fehlerbehandlung siehe unten)
   * Das ADT-Array hat keine Größenbeschränkung
2. technisch
   * Das ADT-Array soll mittels ADT-Liste implementiert werden

Die ADT-Queue enthält folgende Objektmengen:

* elem: Ein Element der ADT-Queue
* array: Ein ADT-Array, mit Elementen
* pos: Die Position eines Elementes im ADT-Array

Folgende Operationen sollen bereitgestellt werden (semantische Signatur):

* *initA*: einen leeren ADT-Array erstellen   
  („nichts“ -> array)

Fehlerbehandlung: ignorieren (es wird kein Fehler geworfen)

* *setA*: Einfügen eines Elementes an gewählter Position

(array x pos x elem -> array)

Fehlerbehandlung: ignorieren

* *getA*: Eine Kopie eines Elementes der gewählten Position zurückgeben

(array x pos -> elem)

Fehlerbehandlung: 0 zurückgeben

* *lengthA*: gibt als Länge die Position des letzten Elements im ADT-Array zurück (leeres ADT- Array hat die Länge -1)

(array -> pos)

Fehlerbehandlung: 0 zurückgeben