Dokumentationskopf

Team: 5, Constantin Wahl & Marc Kaepke

Aufgabenaufteilung:

Gemeinsame Erarbeitung der Skizze

Quellenangaben:

Vorlesungsscript

Bearbeitungszeitraum:

Marc Kaepke:

14.10.15 => 1.5 Stunden (ADT Liste)

15.10.15 => 1.5 Stunden (ADT Stack, Queue, Array)

Constantin Wahl:

15.10.15 => 2 Stunden (ADT Liste, Stack, Queue, Array)

Aktueller Stand:

Update: ADT Array #setA() überarbeitet -> Einfügen an Position -1 wirft einen Fehlercode

Die Skizze ist vollständig erstellt. Mit der Implementierung wurde noch nicht begonnen.

ADT Liste – Skizze

Vorgabe:
Funktional (nach außen)
- Die Liste beginnt bei Position 1
- Die Liste arbeitet destruktiv, d.h. wird ein Element an einer vorhandenen Position eingefügt, wird das dort stehende Element um eine Position verschoben
- Die Elemente sind vom Typ „ganze Zahl“

Technisch (nach innen)
- Die Liste ist intern mittels eines Arrays zu realisieren

Objektmengen:
pos – gibt die Position in der Liste an und ist eine ganze Zahl größer gleich 1
elem – ist ein Element innerhalb der Liste und eine ganze Zahl (.. -2, -1, 0, 1, 2, ..), in Java Integer-Typ
list – ist eine Liste

Operationen:

Methodensignatur	Beschreibung	Return Value
#create()	• Erzeugt eine neue Liste	Kein Rückgabewert
#isEmpty(list)	• Prüft, ob eine gegebene Liste leer ist -> keine Elemente enthält	Bool’scher Wert: true -> kein Element vorhanden false -> Element(e) vorhanden
#laenge(list)	• Gibt die Länge der gegebene Liste zurück	Ganze Zahl größer gleich 0 (-> siehe Fehlerbehandlung)
#insert(list, pos, elem)	• Fügt an die gegebene Liste an einer bestimmten Position ein weiteres Element ein • Ist die Position bereits belegt wird dieses Element nach hinten geschoben • 0 ≤ pos ≥ (#laenge(list) + 1)	Destruktive Liste mit einem neuen Element an entsprechender Position
#delete(list, pos)	• Löscht das Element an gegebener Position aus gegebener Liste • Nach dem Entfernen rücken die anderen Elemente auf -> keine Lücke • 1 ≤ pos ≥ #laenge(list)	Destruktive, zusammenhängende Liste mit einem Element weniger
#find(list, elem)	• Gibt die erste Position des gegebenen Elements zurück • Das Element wird aus der gegebenen Liste entfernt und die Elemente rücken auf	Die Position des ersten Elements in der gegebenen Liste
#retrieve(list, pos)	• Gibt das Element aus der Liste an gegebener Position zurück • Das Element wird nicht aus der Liste genommen • 1 ≤ pos ≥ #laenge(list)	Das Element an gegebener Position
#concat(list, list)	• Konkateniert beide gegebenen Listen • Der ersten Liste wird die zweite Liste angehängt	Destruktive Liste

Fehlerbehandlung:

Methodensignatur	Fehlerbehandlung
#create()	
#isEmpty(list)	• Boolean Wert wie bei Java Klasse ArrayList
#laenge(list)	• Wenn Liste leer ist, wird 0 zurückgegeben
#insert(list, pos, elem)	• Wenn pos ungültig ist, wird die Input-Liste unverändert zurückgegeben (keine Mutmaßung welche Position gemeint sein könnte!)
#delete(list, pos)	• Wenn pos ungültig ist, wird die Input-Liste unverändert zurückgegeben
#find(list, elem)	• Wenn die gegebene Liste das Element nicht enthält, wird die Position -1 zurückgegeben und signalisiert einen Fehler
#retrieve(list, pos)	• Wenn pos ungültig ist wird Integer.MINVALUE zurückgegeben
#concat(list, list)	• Wenn eine Liste leer ist, ist der Rückgabewert die nicht-leere-Liste • Wenn beide Listen leer sind, ist der Rückgabewert eine leere Liste

JUnit-Tests:

Dateiname: adt_liste_junit.jar

Bei den JUnits müssen die Fehlerbehandlungen abgeprüft werden -> ungültige Einfüge/Löschposition (beginnt die Liste wirklich bei Position 1), Entnahme eines Elements aus einem leeren ADT Typ

ADT Stack – Skizze

Vorgabe:
Technisch (nach innen)

- Die ADT Stack ist mittelt ADT Liste zu realisieren

Objektmengen:
elem – ist ein Element innerhalb des Stacks und eine ganze Zahl (.. -2, -1, 0, 1, 2, ..), in Java Integer-Typ
stack – ein Stack

Operationen:

Methodensignatur	Beschreibung	Return Value
#create()	• Erzeugt einen neuen Stack	
#push(stack, elem)	• Legt das gegebene Element oben auf den Stack	Destruktiver Stack mit einem neuen Element
#pop(stack)	• Entfernt das oberste Element vom Stack	Destruktiver Stack mit einem Element weniger
#top(stack)	• Auch „peek“ genannt • Liest das oberste Element vom Stack, löscht es aber nicht	Ein Element
#isEmptyS(stack)	• Prüft, ob der gegebene Stack Elemente enthält oder nicht	Bool’scher Ausdruck: true -> kein Element enthalten false -> Element(e) enthalten

Fehlerbehandlung:

Methodensignatur	Fehlerbehandlung
#create()	
#push(stack, elem)	
#pop(stack)	• Wenn der Stack leer ist, wird ein leerer Stack zurückgegeben
#top(stack)	• Wenn der Stack leer ist, wird Integer.MINVALUE als Fehlercode zurückgegeben
isEmptyS(stack)	

JUnit-Tests:
Dateiname: adt_stack_junit.jar
Bei den JUnits müssen die Fehlerbehandlungen abgeprüft werden -> ungültige Einfüge/Löschposition, Entnahme eines Elements aus einem leeren ADT Typ

ADT Queue – Skizze

Vorgabe:
Technisch (nach innen)

- Die ADT Queue ist mittels ADT Stack, wie in der Vorlesung, zu realisieren. Es sind z.B. zwei explizite zu verwenden und das „umspabeln“ ist nur bei Zugriff auf einen leeren „out-Stack“ durchzuführen

Objektmengen:
elem – ist ein Element innerhalb der Queue und eine ganze Zahl (.. -2, -1, 0, 1, 2, ..), in Java Integer-Typ
queue – eine Queue (Schlange)

Operationen:

Methodensignatur	Beschreibung	Return Value
#create()	• Erzeugt eine neue Queue	
#enqueue(queue, elem)	• Fügt in die gegebene Queue hinten das neue Element ein	Eine Queue mit einem neuen Element
#dequeue(queue)	• Löscht das Element head aus der gegebenen Queue • Mutator	Eine Queue ohne dem head-Element
#front(queue)	• Liefert das Element head zurück • head wird nicht aus der Queue gelöscht • Nicht destruktiv	Das head-Element
#isEmptyQ(queue)	• Prüft, ob die gegebene Queue leer ist	Bool’scher Ausdruck: true -> kein Element enthalten false -> Element(e) vorhanden

Fehlerbehandlung:

Methodensignatur	Fehlerbehandlung
#create()	
#enqueue(queue, elem)	
#dequeue(queue)	• Wenn die Queue leer ist, wird eine leere Queue zurückgegeben
#front(queue)	• Wenn die Queue leer ist, wird Integer.MINVALUE als Fehlercode zurückgegeben
#isEmptyQ(queue)	

JUnit-Tests:
Dateiname: adt_queue_junit.jar
Bei den JUnits müssen die Fehlerbehandlungen abgeprüft werden -> ungültige Einfüge/Löschposition, Entnahme eines Elements aus einem leeren ADT Typ

ADT Array – Skizze

Vorgabe:

Funktional (nach außen)

- Das Array beginnt bei Position 0
- Das Array arbeitet destruktiv, d.h. wird ein Element an einer vorhandenen Position eingefügt, wird das dort stehende Element überschrieben
- Die Länge des Arrys wird bestimmt durch die bis zur Abfrage größten vorhandenen explizit beschriebenen Position im Array
- Das Array ist mit 0 initialisiert, d.h. greift man auf eine bisher noch nicht beschriebenen Position im Array zu erhält man 0 als Wert
- Das Array hat keine Größenbeschränkung, d.h. bei der Initialisierung wird keine Größe vorgegeben

Technisch (nach innen)

- Die ADT Array ist mittels ADT Liste zu realisieren

Objektmengen:

pos – gibt die Position in dem Array an und ist eine ganze Zahl größer gleich 1
elem – ist ein Element innerhalb des Arrays und eine ganze Zahl (.. -2, -1, 0, 1, 2, ..), in Java Integer-Typ
array – ist ein Array

Operationen:

Methodensignatur	Beschreibung	Return Value
#initA()	• Erzeugt ein neues Array	
#setA(array, pos, elem)	• Fügt in das gegebene Array an der Position das gegebene Element ein • 0 ≤ pos ≥ (#lengthA(arry) +1)	Array mit einem zusätzlichen Element
#getA(array, pos)	• Gibt das Element an gegebenen Position zurück • Das Element wird anschließend aus dem Array gelöscht und die Lücke geschlossen • 0 ≤ pos ≥ #lengthA(array)	Das Element an gegebener Position
#lengthA(array)	• Gibt die Länge des Arrays zurück • Die größte Position, im Array, zu diesem Zeitpunkt	Ganze Zahl

WICHTIG: ~~Die Methodensignatur von #lengthA unterscheidet sich von der Aufgabenstellung (#lengthjA)~~

Fehlerbehandlung:

Methodensignatur	Fehlerbehandlung
#initA()	
#setA(array, pos, elem)	• Wenn die Position ungültig ist, wird das Input-Array unverändert zurückgegeben • Wenn die Einfügeposition bereits belegt ist, wird das Element überschrieben
#getA(array, pos)	• Wenn die Position ungültig ist, wird 0 als Wert zurückgegeben
#lengthA(array)	• Wenn das Array leer ist, wird 0 als Wert zurückgegeben

JUnit-Tests:

Dateiname: adt_array_junit.jar

Bei den JUnits müssen die Fehlerbehandlungen abgeprüft werden -> ungültige Einfüge/Löschposition, Entnahme eines Elements aus einem leeren ADT Typ,
beginnt der Array bei Position 0