**Team**: 17, (1.) Assiel Taher / (2.) Michael Müller

**Aufgabenaufteilung**:

1. Skizze zum Stack / Array,

Beides Komplett

1. Skizze zur Liste / Queue,

Beides Komplett

**Quellenangaben**: Ihre Aufgabe / Vorlesung.

**Bearbeitungszeitraum**:   
08.10.15 14:30 – 15:00 Michael Müller

09.10.15 13:00 – 13:30 Assiel Taher

10.10.15 17:30 – 18:00 Assiel Taher

10.10.15 13:45 – 14:15 Michael Müller

12.10.15 14:30 – 16:00 Michael Müller

12.10.15 16.00 – 17.00 Assiel Taher

12.10.15 22:00 – 22:30 Michael Müller

12.10.15 21:30 – 22:30 Assiel Taher

13.10.15 10:30 – 11:15 Michael Müller

**Aktueller Stand**: Skizze zur Liste / Stack / Queue / Array erstellt.

**Änderungen in der Skizze**:

Komplett Überarbeitung (ADT Liste, ADT Queue)

ADT Array wurde um Tests erweitert.

ADT Stack wurde um Tests erweitert.

**Skizze**:

Alle Operationen sind in OO-Notation. Sollten die Operationsnamen am Anfang fälschlicherweise großgeschrieben sein so ist es auf die Word-Autokorrektur zurückzuführen.

**ADT List (adtList.jar):**

**Funktional:**

1. Eine leere Liste hat keine erste Position. Sollte dennoch eine gefordert sein gib -1 zurück.
2. In einem Array werden Objekte beim Einfügen überschrieben oder das Array erweitert. Ersteres (überschriebene Objekte) darf hier nicht passieren.
3. Jedes Element in der Liste muss eine ganze Zahl sein.

**Technisch:**

1. Als Basis der zu implementierenden ADT Liste steht ein Java-Array

**Objektmengen:** pos (Position eines Objektes innerhalb der Liste),

elem (Objekt, außer- /innerhalb der Liste),

list (Eine ADT Liste)

**Operationen („-1“ entspricht: Nutzer soll auf Fehler aufmerksam gemacht werden**

**// Tests sind Pseudocode bzw. sollen abstrakt beschreiben was getestet werden soll):**

create(output: AdtList):

Neue leere ADT Liste wird erschaffen.

Test: Die neue Liste sollte gleich einer Leeren Liste sein.

isEmpty(output: boolean):

Eine Liste ist Leer wenn sie keine Elemente erhält, gibt true/false zurück.

Test: [] -> true, sonst immer false.

laenge(output: int):

Zählt Anzahl der Elemente und gib diese Zahl zurück.

Test: [] -> 0, [1] -> 1, [1,5] -> 2

insert(input: int pos, int elem | output: AdtList):

Funktional 2. Beachten. Hat 2 Argumente, zuerst die Position des neuen Objektes danach das Element selbst. Ausgegeben wird die neue Liste. Wird versucht ein Element in die Liste einzufügen das keine ganz Zahl ist, dann wird eine unveränderte Liste zurückgegeben. Wird eine Postition im Aufruf übergeben die gar nicht vorhanden ist wird das Element vorne angefügt.

Test: [1].insert(2,“test“) -> [1], [1].insert(1,6) -> [6,1], [1].insert(3,6) -> [6,1]

delete(input: int pos | output: AdtList):

Hat 1 Argument, die Position des zu löschenden Objektes innerhalb der Liste. Zurückgegeben wird die neue Liste. Wichtig zu beachten ist, dass alle Objekte nach der Position wieder um 1 nach Links verschoben werden müssen. Wird eine Position übergeben die es nicht gibt, so wird eine unveränderte Liste zurückgegeben.

Test: [1].delete(1) -> [], [1,2,3].delete(2) -> [1,3], [].delete(2] -> []

find(input: int elem | output: int):

Hat 1 Argumente, das Objekt nach dem gesucht werden soll. Gib die Position des ersten passenden Objektes zurück. Wurde kein passendes Objekt gefunden mache den Nutzer darauf aufmerksam.

Test: [].find(2) -> -1, [1,3].find(3) -> 2, [1,3,3,3,3].find(3) -> 2

retrieve(input: int pos | output: int): Hat 1 Argument, die Position des gewünschten Objektes. Gibt das Element an der Position zurück ohne es zu löschen. Sollte eine Position eingegeben worden sein die es in der Liste nicht gibt mache den Nutzer darauf aufmerksam.

Test: [].retrieve(1) -> -1, [1,2,5].retrieve(3) -> 5

concat(input: AdtList list | output: AdtList): Hat 1 Argumente, die anzuhängende Liste. Es sollte die zweite Liste an der ersten angehängt werden ohne die Reihenfolge zu verändern. Ausgegeben wird die neue Liste.

Test: [1,2,3].concat([]) -> [1,2,3], [].concat([]) -> [], [].concat([1,2,3]) -> [1,2,3]

**ADT – Stack**

Für einen Stack kann man eine Liste verwenden, da diese eine geordnete Sammlung (mit einer Reihenfolge) von Daten ist.

Der Stack ist mittels ADT Liste zu realisieren.

Hinweis: Die folgenden Tests sind keine syntaktisch korrekten java Tests. Sie sind lediglich da, um Beispiele für Testergebnisse zu zeigen bzw. was für ein Ergebnis in bestimmten Fällen auftritt.

**Objektmengen:**

Elem: Die Menge der Elemente eines Arrays.

stack: Die Menge der stack.

**Operationen:**

Create: Es wird ein leerer Stack konstruiert.

Test:

create(output: void)

Push: Diese Methode fügt dem Stack ein neues Element hinzu und das Top Element müsste jetzt das neu eingefügte Element sein.

Test:

Push(input: int elem | output: void)

Pop: Das Top-Element wird gelesen und entfernt vom Stack. Ist der Stack leer so wird ein Hinweis gegeben, dass er leer ist.

Test:

top(output: int)

Top: Liefert das Element, welches als letztes eingefügt wurde. Ist der Stack leer so wird ein Hinweis gegeben, dass er leer ist.

Test:

top(output: int)

isEmpty: Prüft ob ein Element im Stack enthalten ist. Rückgabewert ist hier boolean

Test:

isEmpty(output: boolean)

Der ADT Stack wird abgespeichert als “adtStack.jar”.

**ADT Queue (adtQueue.jar):**

**Technisch:**

1. Es gibt 2 ADT Stacks die zu einer Queue werden. Dadurch das der erste Stack alle hinzukommenden Elemente aufnimmt und der zweite Stack alle ausgehenden Elemente enthält. Dadurch das bei der Queue FiFo (First in First out) aber bei einem Stack LiFo (Last in First out) gilt, muss einmal umgedreht werden. Dies geschieht wenn wir den 1. Stack leeren und in umgedrehter Reinfolge in den 2. Stack füllen. Es wird immer erst in den 2. Stack übergeben wenn der 2. Stack leer ist. Sonst würde die Reihenfolge durcheinander kommen und Elemente die als erstes Eingefügt wurden als letztes herauskommen (obwohl sie ja auch als erstes wieder herauskommen sollten). Zu beachten ist, dass die Queue auf einem Stack basiert der wiederum auf einer Liste fußt die nur ganze Zahlen annimmt!

**Operationen („-1“ entspricht: Nutzer soll auf Fehler aufmerksam gemacht werden**

**// Tests sind Pseudocode bzw. sollen abstrakt beschreiben was getestet werden soll**

**// []-[] entspricht [1.Stack]-[2.Stack]):**

createQ(output: AdtQueue):

Neue Queue wird erschaffen, Konstruktor erzeugt 2 Stacks die die Queue bilden (vgl. Teschnisch-1.).

enqueue(input: int elem | output: AdtQueue):

Hat 1 Argument, das neue Element für den 1. Stack. Ausgegeben wird die neue Queue. Es wird ganze Links (ganz oben) eingefügt. Wird versucht ein Element hinzuzufügen welches keine ganze Zahl ist, so wird der Nutzer darauf aufmerksam gemacht.

Test: [1,2,3]-[1].enqueue(2) -> [2,1,2,3]-[1], [1]-[].enqueue(„test“) -> -1

dequeue(output: AdtQueue):

Entnimmt ein Element aus dem 2. Stack und liefert dem Nutzer die neue Queue. Ist im 2. Stack kein Element so wird der erste Stack (vgl. Queue-Technisch-1.) in umgedrehter Reihenfolge in den 2. Stack überführt und danach das erste Element des 2. Stacks ausgegeben. Sind beide Stacks leer so wird eine leere Queue übergeben.

Test: [1]-[].dequeue() -> []-[], [1,2,3]-[5].dequeue() -> [1,2,3]-[], []-[].dequeue() -> []-[]

front(output: int): Gibt die Position zurück welches Element ganz rechts (erste Stelle) im rechten Stack steht ohne es zu entnehmen. Steht dort kein Element wird der erste Stack in den zweiten in umgedrehter Reihenfolge überfürht. Ist der 2. Stack dann immer noch leer soll der Nutzer darauf aufmerksam gemacht werden.

Test: [1,2,3]-[].front -> -1, [3]-[2,1].front -> 1

isEmptyQ(output: boolean): Wenn beide Stacks leert sind, ist es auch die Queue.

Test: [1]-[].isEmptyQ -> false, []-[1].isEmptyQ: -> false, []-[].isEmptyQ -> true

**ADT Array**

**Funktional:**

1. Anders als der ADT Liste beginnt der ADT Array bei Position 0.
2. Anders als in der ADT Liste muss hier nichts verschoben werden, sondern überschrieben.
3. Die Länge wird bestimmt durch den an der höchsten Position beschriebenen Wert. Die Position ist der Rückgabewert.
4. Das Array ist mit 0 initialisiert, d.h. greift man auf eine bisher noch nicht beschriebene Position im Array zu erhält man 0 als Wert.
5. Bei der Initialisierung des Arrays wird keine Größenbeschränkung angegeben.

**Technisch:**

Der ADT Array ist mittels ADT Liste zu realisieren.

**Objetktmengen:**

Pos: Die Menge der Positionen im Array.

Elem: Die Menge der Elemente eines Arrays.

Array: Die Menge der Arrays.

**Operationen:**

initA: Ein Array wird konstruiert.

Test:

initA(output: void)

setA: Schreibt ein Element in eine Position

Test:

setA(input: int pos, int elem | output: void)

getA: Gibt das Element an der Stelle „pos“ zurück.

Test:

getA(input: int pos | output: int)

lenghthA: Gibt die länge des Arrays wieder.

Test:

lengthA(output: int)

Der ADT Array wird abgespeichert als “adtArray.jar”.