# Dokumentation zum Aufgabenblatt a02

Die geforderten Interfaces für einen Stack und fuer eine Queue sind anhand der Vorgaben aus dem Vorlesungsscript erstellt worden. An Punkten, wo Fehler auftreten koennen (z.B. Overflow oder Underflow) wurde eine Exceptionbehandlung eingefuegt. Zusaetzlich wurde in das Interface fuer die Queue die Methode size() eingefuegt, um einfach die Groeße der Queue abfragen zu koennen.

Anschließend wurden die beiden geforderten Varianten einer Queue implementiert. In Zeile 22 der Klasse RingQueue.java wird die „unchecked“ Warnung unterdrueckt. Im folgenden Konstruktor muss ein Cast auf den uebergebenen Typ erfolgen. Die Alternative waere den Cast in den Methoden durchzufuehren, die ein Element zurueckliefern (front()).
Die Nutzung der LinkedList mit antizipativer Indizierung (Aufgabenstellung a01) ist in diesem Fall nicht empfehlenswert, da es eine Erweiterung bzw. Modifikation des vorhandenen Codes bedeuten wuerde. Es muessten entweder weitere Datenstrukturen fuer die Speicherung von Referenzen geschaffen werden oder die Listenimplementierung dahingehend geaendert werden, dass man bei append() und delete() einfach auf head und tail zugreifen koennte. Aus genannten Gruenden wurde die LinkedList aus java.util als einfache und sichere Implementierung gewaehlt.
Die Funktion der beiden Implementierungen sind mittels JUnit Testfaellen sichergestellt worden (siehe Testklasse QueueJUnit.java).
Die Simulation der „Druckerwarteschlangen“ findet sich in der Klasse PrinterQueue. Die Anzahl der Durchlaeufe und die Anzahl der Durchlaeufe nach denen Elemente geloescht wird werden ueber Konstanten gesteuert. Der Betrieb in einer Endlosschleife ist ebenfalls denkbar. Beim Lauf der Druckerwarteschlange erfolgt eine Ausgabe auf der Konsole, welche Elemente in welche Queue eingefuegt und geloescht werden. Am Ende werden nochmals die einzelnen Queues samt Inhalt auf der Konsole ausgegeben.

Das Interface fuer den Stack wurde in der Klasse Stack.java implementiert. Der Stack verwendet intern, wie bereits bei der Listenvariante der Queue, eine java.util.LinkedList. Auch hier wuerde die Implementierung der Liste mit antizipativer Indizierung zu Mehraufwand fuehren.
Die Funktion des Stacks wurde mit Hilfe der JUnit-Testklasse StackJUnit.java sichergestellt.
Die geforderte Implementierung des „Zwei Stack Algorithmus“ befindet sich in der Klasse twoStackAlgo.java
Fuer die Realisierung wurden zwei Stacks genutzt, einer fuer die Operatoren (Stack von Character) und einer fuer die Operanden (Stack von Integer). Uebergeben werden die vollstaendig geklammerten Ausdruecke als String. Die einzelnen Symbole sind durch Leerzeichen zu trennen, da

der String anhand von Leerzeichen gesplittet wird. Anschließend wird geprueft, um welches Symbol es sich handelt und entsprechend reagiert.
Die einzelnen Zwischenschritte und das Endergebnis des Programms werden auf der Konsole ausgegeben.

Die Methode toString() wurde in den drei Implementierungen nur zu Testzwecken ueberschrieben.

Die UML Diagramme und die uebersichtliche Darstellung des zeitlichen Verlaufs der Belegung der beiden Stacks befindet sich im Anschluss an diese Dokumentation.

**Martin Slowikowski**
Matrikelnummer: 199 91 66

**Tell Mueller-Pettenpohl**
Matrikelnummer: 198 99 82

**Teamname**: Tugend und Laster

**Quellenangaben**

TIB3-AD-skript.pdf
Letzte Änderung 22.03.2011

# Darstellung des zeitlichen Verlaufs der Belegung der beiden Stacks aus dem Programmierteil 2

In der folgenden tabellarischen Uebersicht sind anhand eines Beispiels die Belegung und der Ablauf beim Auswerten eines vollstaendig geklammerten Ausdrucks in einzelnen Zeitschritten dargestellt.

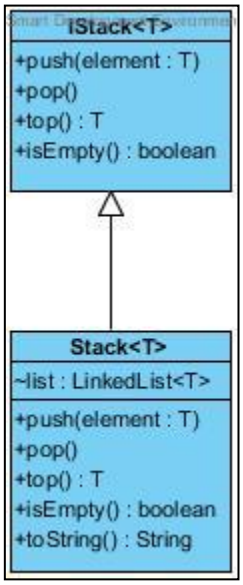
Ausdruck: ( ( ( ( 3 * 5 ) - 9 ) / 2 ) + ( ( 4 / 2 ) + 5 ) )

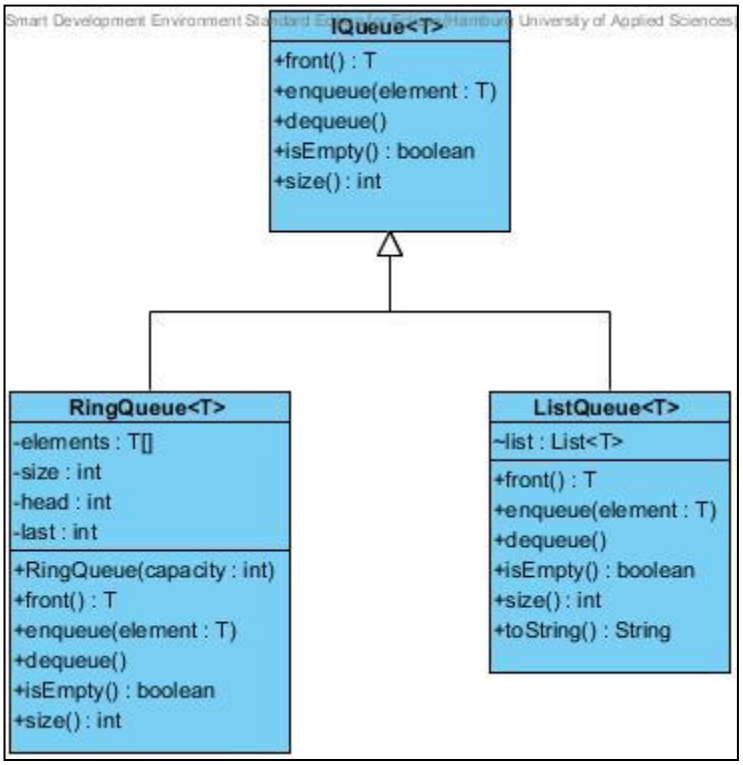
| Zeit | Vorgang | Symbol | Operation | Stack1 (Operanden) | Stack2 (Operatoren) |
|---|---|---|---|---|---|
| 1 | einlesen | ( | ignorieren | | |
| 2 | einlesen | ( | ignorieren | | |
| 3 | einlesen | ( | ignorieren | | |
| 4 | einlesen | ( | ignorieren | | |
| 5 | einlesen | 3 | push Stack1 | 3 | |
| 6 | einlesen | * | push Stack2 | | * |
| 7 | einlesen | 5 | push Stack1 | 5, 3 | * |
| 8 | einlesen | ) | ignorieren | 5, 3 | * |
| 9 | Operator holen | | top Stack2 | 5, 3 | * |
| 10 | Operator löschen | | pop Stack2 | 5, 3 | |
| 11 | Operanden holen | | top Stack1 | 5, 3 | |
| 12 | Operanden löschen | | pop Stack1 | 3 | |
| 13 | Operanden holen | | top Stack1 | 3 | |
| 14 | Operanden löschen | | pop Stack1 | | |
| 15 | rechnen | | calc(3*5) | | |
| 16 | Ergebnis speichern | 15 | push Stack1 | 15 | |
| 17 | einlesen | - | push Stack2 | 15 | - |
| 18 | einlesen | 9 | push Stack1 | 9, 15 | - |
| 19 | einlesen | ) | ignorieren | 9, 15 | - |
| 20 | Operator holen | | top Stack2 | 9, 15 | - |
| 21 | Operator löschen | | pop Stack2 | 9, 15 | |
| 22 | Operanden holen | | top Stack1 | 9, 15 | |
| 23 | Operanden löschen | | pop Stack1 | 15 | |
| 24 | Operanden holen | | top Stack1 | 15 | |
| 25 | Operanden löschen | | pop Stack1 | | |
| 26 | rechnen | | calc(15-9) | | |
| 27 | Ergebnis speichern | 6 | push Stack1 | 6 | |
| 28 | einlesen | / | push Stack2 | 6 | / |
| 29 | einlesen | 2 | push Stack1 | 2, 6 | / |
| 30 | einlesen | ) | ignorieren | 2, 6 | / |
| 31 | Operator holen | | top Stack2 | 2, 6 | / |
| 32 | Operator löschen | | pop Stack2 | 2, 6 | |

| Zeit | Vorgang | Symbol | Operation | Stack1 (Operanden) | Stack2 (Operatoren) |
|---|---|---|---|---|---|
| 33 | Operanden holen | | top Stack1 | 2, 6 | |
| 34 | Operanden löschen | | pop Stack1 | 6 | |
| 35 | Operanden holen | | top Stack1 | 6 | |
| 36 | Operanden löschen | | pop Stack1 | | |
| 37 | rechnen | | calc(6/2) | | |
| 38 | Ergebnis speichern | 3 | push Stack1 | 3 | |
| 39 | einlesen | + | push Stack2 | 3 | + |
| 40 | einlesen | ( | ignorieren | 3 | + |
| 41 | einlesen | ( | ignorieren | 3 | + |
| 42 | einlesen | 4 | push Stack1 | 4, 3 | + |
| 43 | einlesen | / | push Stack2 | 4, 3 | /, + |
| 44 | einlesen | 2 | push Stack1 | 2, 4, 3 | /, + |
| 45 | einlesen | ) | ignorieren | 2, 4, 3 | /, + |
| 46 | Operator holen | | top Stack2 | 2, 4, 3 | /, + |
| 47 | Operator löschen | | pop Stack2 | 2, 4, 3 | + |
| 48 | Operanden holen | | top Stack1 | 2, 4, 3 | + |
| 49 | Operanden löschen | | pop Stack1 | 4, 3 | + |
| 50 | Operanden holen | | top Stack1 | 4, 3 | + |
| 51 | Operanden löschen | | pop Stack1 | 3 | + |
| 52 | rechnen | | calc(4/2) | | |
| 53 | Ergebnis speichern | 2 | push Stack1 | 2, 3 | + |
| 54 | einlesen | + | push Stack2 | 2, 3 | +, + |
| 55 | einlesen | 5 | push Stack1 | 5, 2, 3 | +, + |
| 56 | einlesen | ) | ignorieren | 5, 2, 3 | +, + |
| 57 | Operator holen | | top Stack2 | 5, 2, 3 | +, + |
| 58 | Operator löschen | | pop Stack2 | 5, 2, 3 | + |
| 59 | Operanden holen | | top Stack1 | 5, 2, 3 | + |
| 60 | Operanden löschen | | pop Stack1 | 2, 3 | + |
| 61 | Operanden holen | | top Stack1 | 2, 3 | + |
| 62 | Operanden löschen | | pop Stack1 | 3 | + |
| 63 | rechnen | | calc(2+5) | | |
| 64 | Ergebnis speichern | 7 | push Stack1 | 7, 3 | + |
| 65 | einlesen | ) | ignorieren | | |
| 66 | Operator holen | | top Stack2 | 7, 3 | + |
| 67 | Operator löschen | | pop Stack2 | 7, 3 | |
| 68 | Operanden holen | | top Stack1 | 7, 3 | |
| 69 | Operanden löschen | | pop Stack1 | 3 | |
| 70 | Operanden holen | | top Stack1 | 3 | |
| 71 | Operanden löschen | | pop Stack1 | | |
| 72 | rechnen | | calc(3+7) | | |
| 73 | Ergebnis speichern | 10 | push Stack1 | 10 | |

# Klassen- und Objektdiagramme (UML) fuer die Klassen ListQueue, RingQueue und Stack

Es folgen Klassendiagramme für die beiden zu implementierenden Interfaces für einen Stack und eine Queue, weiterhin die implementierenden Klassen (Stack, Queue als Ring, Queue als Liste).



Die folgenden Objektdiagramme bilden die genutzten Datensstrukturen aus der Aufgabenstellung a02 ab.

Die Objektdiagramme zeigen eine statische Entwurfssicht, ähnlich einem „Schnappschuss“ des Systems zu einem bestimmten Zeitpunkt.