Software Qualität

Milos Babic & Tim Schmiedl

Pflichtaufgabe Nr. 4

2. Juni 2013

1. Fehler in Anforderungen & Programm

1.1 Spezifikation

- Die Remove-Methode ist in der Aufgabenstellung widersprüchlich definiert. So wird in der Auflistung der Rückgabewert „bool“ angegeben, während später für die Methode folgendes definiert ist: „Die Funktion *remove*() entfernt eine Person aus dem Behälter und gibt sie als Rückgabewert zurück.“
- Das genaue Verhalten des Stacks bzw. der Queue ist nicht genau definiert. Die hat evtl. zu den Unstimmigkeiten in der Implementierung geführt.
- Das Sequenzielle Durchgehen, im besonderen die Methode *getFirst*(), ist nicht eindeutig spezifiziert. Beispielsweise wird nicht definiert wo der Zeiger vor dem ersten Aufruf von *getFirst*() steht. Auch ist nicht klar, was passieren soll, wenn während des Iterieren Elemente eingefügt/entfernt werden.

1.2 Implementiertes Programm

- Der Stack falsch implementiert (entspricht nicht dem bekannten Stack-Verhalten).
- *getFirst*() steht initial vor dem letzten Element?
- Diverse Programmierfehler:
  - An mehreren Stellen im Code wird eine Membervariable im Index eines Arrays gesetzt und gibt so ein ungewünschten Wert zurück (Beispiel: in Lines:75, 77, 120)
  - Die Implementierung der Methode *getNext*() von Stack fürt unter Umständen zu einem Segmention Fault (Line:122)
  - In der Klasse Person ist der Typ von name ein String und nicht wie in der Beschreibung vorgesehen char

2. Welche Testmethode?

Aufgrund der Einteilbarkeit in Zustände bietet sich ein Zustandsbasierter Test an.

3. Zustandsbasierter Test

3.1. Diagramme

Wir haben für die Aufgabenstellung zwei wichtige Use-Cases gefunden. Diese beiden sind in den folgenden Diagrammen jeweils mit eigenen Zuständen abgebildet.

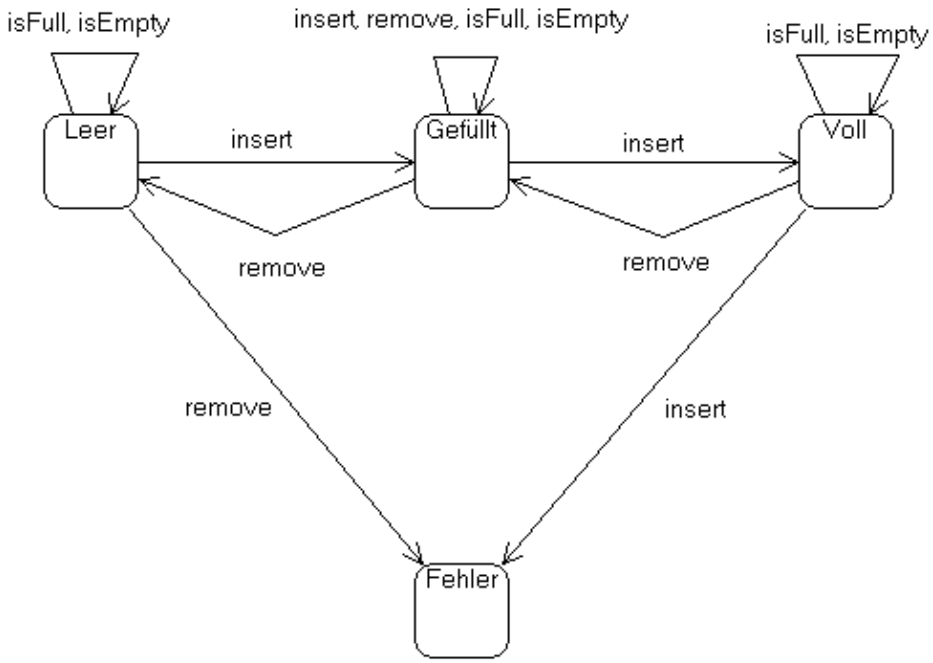

Abb. 1: Diagramm: Füllen und Leeren

In Abb.1 ist das Füllen und Leeren der Behälterklassen „Stack“ und „Queue“ abgebildet.

Da die Methoden „*hasNext*“, „*getNext*“ und „*getFirst*“ keine direkte Bindung mit dem hier dargestellten Zuständen haben sind sie hier nicht dargestellt, sondern treten nur im zweiten Diagramm auf.

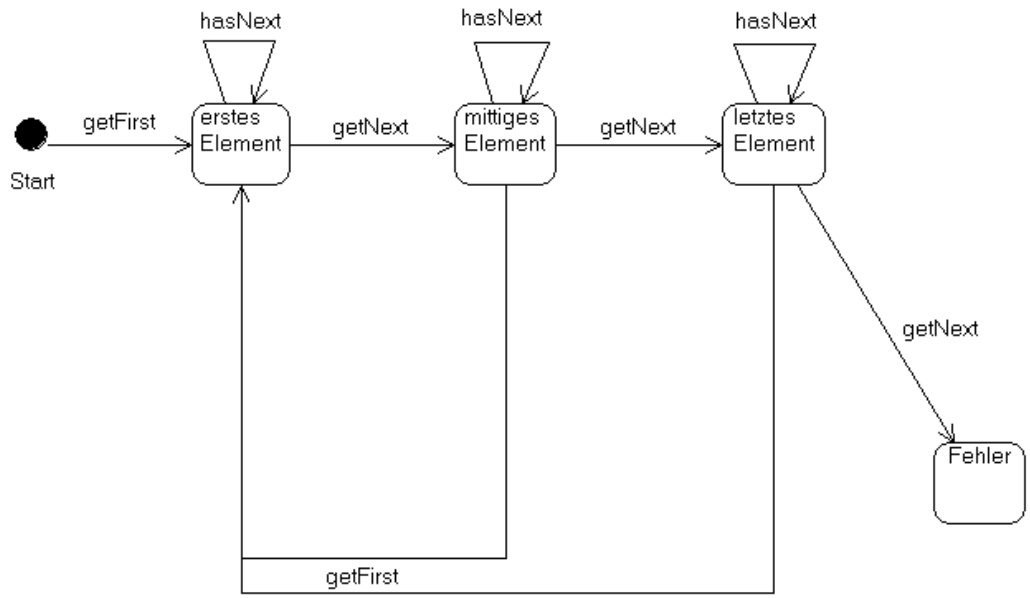

Abb. 2: Diagramm: Sequenzielle Iteration

Abb.2 zeigt die Iteration der Behälterklassen. Hierbei sind die Methoden „*hasNext*“, „*getNext*“ und „*getFirst*“ wichtig und sind hier abgebildet.

3.2. Tabellen

	leer	**gefüllt**	**voll**
isFull()	leer	gefüllt	voll
	-	-	-
isEmpty()	leer	gefüllt	voll
	-	-	-
insert()	gefüllt(voll)	gefüllt(voll)	FZ
	-	-	-
remove()	FZ	gefüllt(leer)	gefüllt(leer)
	-	-	-

	erstes Element	**mittiges Element**	**letztes Element**
getFirst()	erstes Element	erstes Element	erstes Element
	-	-	-
hasNext()	erstes Element	mittiges Element	letztes Element
	-	-	-
getNext()	mittiges Element (letztes Element)	mittiges Element (letztes Element)	FZ
	-	-	-

4. Testfälle

Weil wir der Übersicht halber das Zustandsdiagramm des Behälters in zwei Diagrammen aufgeteilt haben, werden die Testfälle ebenfalls in die zwei Kategorien geteilt.

4.1. Testfälle für die iterative Ausgabe des Behälters

Ausgabe von 3 Personen, Behälter ist voll, Position des Iterators ist beim ersten Element.

erstes Element, *getFirst*() → erstes Element, *hasNext*() →
erstes Element, *getNext*() → mittiges Element, *hasNext*() →
mittiges Element, *getNext*() → letztes Element, *hasNext*().

Ausgabe von 3 Personen, dann die erste Person wieder ausgeben.

erstes Element,*getFirst*() → erstes Element, *hasNext*() →
erstes Element, *getNext*() → mittiges Element, *hasNext*() →
mittiges Element, *getNext*() → letztes Element, *hasNext*()
letztes Element, *getFirst*() → erstes Element.

Bei letzter Person im Behälter *getNext*() ausführen.

letzes Element, *getNext*() → FZ.

4.2. Testfälle für das Befüllen/Entleeren des Behälters

Mehr *insert*() als Kapazität des Behälters zulässt.

leer,*insert*() → leer, *isFull*() → [gefüllt, *insert*() → gefüllt,*isFull*()] →
gefüllt, *insert*() → gefüllt,*isFull*() → FZ.

Mehr *remove*() als Anzahl der Personen, die sich im Behälter befinden.

[Gefüllt, *remove*() → gefüllt, *isEmpty*()] → leer, *remove*() → FZ