7.3.2 Klasse Stack (gb.data)

Die Klasse *Stack* implementiert einen Stapel von Elementen. Diese Klasse verwendet den Datentyp Variant für seine Elemente.

Ein Stack oder Stapel ist eine Datenstruktur, die nach dem LIFO-Prinzip funktioniert. Das Prinzip "Last in, first out" beschreibt abstrakt, was man sich bildhaft zum Beispiel als Papierstapel in einer Box vorstellen kann. Wenn man dem Stapel ein Blatt Papier hinzufügen möchte, so muss man es oben auf den Stapel legen. Der Aufwand, Teile des Stapels hoch zu heben, um das Papier an eine beliebige Stelle zu legen, ist einfach zu groß. Aus diesem Grund kann man auch nur das oberste Blatt Papier des Stapels wieder entfernen. Das letzte hinzugefügte Element ist also das zuerst wieder entfernte (LIFO). In diesem Sinne ist der Stack eine der Queue entgegengesetzte Datenstruktur. Das letzte Element eines Stacks wird auch als dessen oberstes Element bezeichnet. Für den Stack ist es charakteristisch, dass man nur am *oberen* Ende der Datenstruktur operieren kann – Elemente hinzufügen und wieder entfernen.

7.3.2.1 Eigenschaften

Die Eigenschaften eines Stapels werden in der folgenden Tabelle angegeben:

Eigenschaft	Datentyp	Beschreibung
.Size	Integer (ReadOnly)	Anzahl der Elemente des Stacks
.IsEmpty	Boolean (ReadOnly)	Wahr, wenn keine Elemente im Stack sind, sonst falsch

Tabelle 7.3.2.1.1: Eigenschaften der Klasse Stack

7.3.2.2 Methoden eines Stack-Objektes

Mit diesen 4 Methoden kommt ein Stack als Datenstruktur nach dem LIFO-Prinzip aus:

Methode	Beschreibung
Clear()	Entfernen aller Elemente vom Stapel (Stack)
Push(vElement)	Ein Element (oben) auf den Stack legen
Pop()	Entfernen und Zurückgeben des obersten Elements vom Stapel
Peek()	Rückgabe des obersten Elements vom Stapel - ohne es zu entfernen

Tabelle 7.3.2.2.1: Methoden der Klasse Stack

7.3.2.3 Projekt zum Einsatzes der Klasse Stack

Der Gambas-Interpreter verwaltet ebenfalls einen Stack, auf dem u.a. alle Funktionsaufrufe protokolliert werden. Diese Informationen können sich im Fehlerfall in der IDE, zum Beispiel bei einem Programmabsturz, als unschätzbar wichtig zur Identifikation und Beseitigung des Programmfehlers herausstellen. Sie können besser nachvollziehen, wie es zu dem Fehler kam.

Auf dem Stapel der Klasse Stack im Projekt werden alle Funktionen gespeichert, die im aktuellen Code-Pfad aufgerufen wurden. Ein *Code-Pfad* ist eine Verkettung von Funktionsaufrufen. Der zugehörige Stack wird auch als "Call trace" oder "Backtrace" bezeichnet. Ein Backtrace wird über den folgenden einfachen Algorithmus erstellt:

- Zuerst wird beim Eintritt in eine Funktion ein Eintrag für diese Funktion auf den Backtrace-Stack geschoben → Push(vElement),
- dann wird die Funktion ausgeführt und
- abschließend müssen die Informationen beim Austritt aus der Funktion wieder vom Stack entfernt werden → Pop().

Eine Funktion, die eine andere Funktion aufruft, bekommt den Stack in dem Zustand zurück, wie er vor dem Funktionsaufruf war. Sollte allerdings in einer verschachtelten Funktion ein Programmfehler

auftreten, so kann der Backtrace-Stack untersucht werden. Dann können alle Funktionen identifiziert werden, die im Fehler erzeugenden Code-Pfad aufgerufen wurden, da ihre Stack-Elemente noch nicht wieder entfernt werden konnten. Die zuletzt aufgerufene, den Fehler direkt verursachende Funktion, ist somit immer das oberste Element des Backtrace-Stacks.

Ein *Backtrace* ist ein klassisches Beispiel für einen Stack. Im Projekt für die Stack-Klasse wird ein solcher *Backtrace* erzeugt:

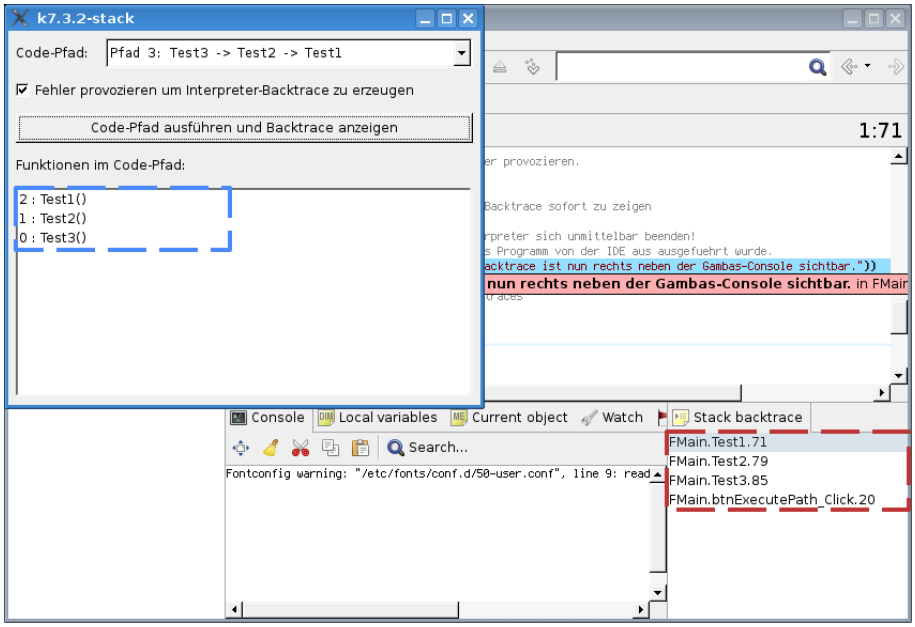

Abbildung 7.3.2.3.1: Backtrace des Anwendungsprogramms (blau) und des Interpreters (rot) bei einem Laufzeitfehler

Neben Backtraces finden Stacks auch Anwendung bei der Implementierung von Parsern:
→ http://de.wikipedia.org/wiki/Umgekehrte_Polnische_Notation