Zustandsbezogener Test

Container-Objekte wie Listen oder Vektoren stellen üblicherweise eine Möglichkeit zur Verfügung, wie auf die in ihnen gespeicherten Elemente zugegriffen werden kann, ohne dass dabei die interne Struktur der Container-Objekte offen gelegt wird. Eine mögliche Lösung für diese Aufgabe stellt das Iterator Design Pattern dar [GOF]. Ein Iterator verfügt (neben einem Konstruktor und einem Destruktor, je nach Programmiersprache und Implementierung) über die öffentlichen Methoden `currentItem`, `next`, `first` und `isDone`.

Ein Iterator ist sozusagen ein Zeiger oder ein Lesezeichen in einem Container, der zu jeder Zeit entweder auf ein bestimmtes Objekt der Liste zeigt, das mittels der Methode `currentItem` abgefragt werden kann, oder auf das Ende der Liste zeigt („hinter" das letzte Objekt des Containers). In diesem Fall soll ein Aufruf der Methode `currentItem` zu einem Fehler führen.

Die Methode `isDone` gibt genau dann `true` zurück, wenn sich der Iterator hinter dem letzten Element des Containers befindet oder der Container leer ist, sonst `false`. Die beiden Methoden `first` und `next` dienen der Navigation des Iterators. Die Methode `first` bewegt dabei den Zeiger auf das erste Element des Containers (oder „hinter das letzte" Element, falls der Container leer ist). Die Methode `next` bewegt den Zeiger auf das nächste Element oder hinter das letzte Element, falls er sich vor diesem Aufruf von `next` auf dem letzten Element befunden hat. Wird die Methode `next` aufgerufen, wenn sich der Zeiger bereits hinter dem letzten Element des Containers befindet, so führt dies zu einem Fehler.

Nach Ausführung des Konstruktors, der den Iterator komplett initialisiert, befindet sich der Iterator im gleichen Zustand, wie nach einem Aufruf der Methode `first`.

Aufgabe 1) Entwerfen Sie ein Zustandsdiagramm für einen Iterator. Bei Ihren Überlegungen zu der Frage, wie viele und welche Zustände ein Iterator hat, sollten Sie stets überprüfen, ob Sie für jeden entworfenen Zustand feststellen können (durch Methodenaufrufe), ob sich der Iterator in diesem Zustand befindet oder nicht. Ein nicht überprüfbarer Zustand ist in diesem Kontext unnütz. Denken Sie daran, sich für die einzelnen Zustandsübergänge zu überlegen, ob und welche Bedingungen für diesen Übergang erfüllt sein müssen. Gehen Sie zu diesem Zweck davon aus, dass `itemCount` die Anzahl der Elemente im Container bezeichnet. Benötigen Sie weitere Variablen? Gehen Sie ferner davon aus, dass ein Iterator jederzeit zerstört werden kann.

Aufgabe 2) Leiten Sie aus Ihrem Zustandsdiagramm aus Aufgabe 1 den zugehörigen Übergangsbaum ab.

Aufgabe 3) Leiten Sie logische Tests zur vollständigen Transitionsüberdeckung von Containern mit genau vier Elementen ab.

[GOF] Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, ISBN 978-0201633610, Addison Wesley, 1994