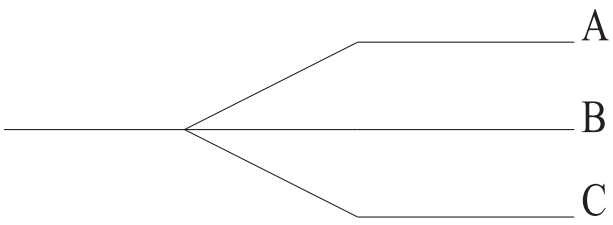
Einführung

Bei der Besprechung der Vorgänge bei der Rekursion und im Zusammenhang mit maschinennaher Programmierung begegnete uns die dynamische Datenstruktur *Stack*. Eine solche Struktur wird häufig auch in Anwendungen benötigt.

Beispiel

Vorgegeben ist ein Güterbahnhof mit folgendem Gleisbild. Auf Gleis A stehen nummerierte Waggons, die so rangiert werden sollen, dass sie anschließend in einer gewünschten Reihenfolge auf Gleis C stehen sollen. Folgende Vorgaben müssen beachtet werden:

- Die Lok kann immer nur einen Waggon ziehen.
- Man hat zwei Helfer: einen an der Spitze der Waggons in A und einen in C. Diese Helfer können immer nur die Nummer des am Ende des Gleises (zur Lokomotive hin) stehenden Waggons ablesen.
- Gleis B (oder auch später A) können als Abstellgleis benutzt werden.



Es sei nun angenommen, dass vom Prellbock zur Lok hin gelesen die Waggons mit den Nummern 16, 11, 15 und 14 stehen. Die Waggons sollen nun am Ende z.B. sortiert auf Gleis C stehen (kleinste Nummer am Prellbock).

Lösung von Hand

1. Hole Waggon 14 aus A und bringe ihn nach C.
2. Hole Waggon 15 aus A und bringe ihn nach C.
3. Hole Waggon 15 aus C und bringe ihn nach B.
4. Hole Waggon 14 aus C und bringe ihn nach B.
5. Hole Waggon 11 aus A und bringe ihn nach C.
6. Hole Waggon 16 aus A und bringe ihn nach C.

7. Helfer von A nach B.
8. Hole Waggon 16 aus C und bringe ihn nach A.
9. Hole Waggon 14 aus B und bringe ihn nach C.
10. Hole Waggon 15 aus B und bringe ihn nach C.
11. Helfer von B nach A.
12. Hole Waggon 16 aus A und bringe ihn nach C.

Allgemeine Strategie

1. Waggons werden so lange von A nach C gebracht, wie dies die gewünschte Ordnung zulässt. Wenn C noch leer ist, passt der erste Waggon aus A auf alle Fälle.
2. Wenn der nächste abzuholende Waggon in A eine kleinere Nummer hat als der vorderste in C, müssen die „falschen“ Waggons aus C in B abgestellt werden.
3. Wenn Gleis A leer ist, werden die Funktionen der Gleise A und B getauscht.

Benötigte Klassen und Objekte

1. Jedes Gleis kann als ein Stack aufgefasst werden. Die Waggons können durch `int` dargestellt werden.
2. Das Verschieben von Waggons geschieht durch `push` und `pop` auf den jeweiligen Stacks.
3. Ein Stack wird als lineare Liste realisiert, bei der nur am Anfang ein Element eingefügt, gelesen oder gelöscht werden kann.
4. Eine Methode `isEmpty` soll feststellen, ob ein Stack leer ist.

Aufgaben:

1. Formuliere eine Klasse `Stack` mit den Methoden `push`, `pop`, `readTop` und `isEmpty`.
2. Teste die Klasse mit Hilfe eines Hauptprogramms, das Eingaben von `int`-Zahlen zum Aufbau eines Stacks benutzt und den Stack zur Kontrolle mit einer Schleife wieder abbauen kann.
3. Versuche, die Strategie zur Lösung des Rangierproblems in einen geeigneten Algorithmus umzusetzen.
4. **Erweiterung 1:** Es soll noch ein dritter Helfer für Gleis B vorhanden sein. Damit können auch Waggons aus B geholt werden, wenn sie in die Sortierung passen.

5. **Erweiterung 2:** In der Praxis ist folgendes Vorgehen wohl häufiger anzutreffen: Der Lokführer bekommt eine Liste (Stack) der Waggons, die angibt, wie sie auf dem Zielgleis C stehen sollen.

Stacks in der Java-Praxis:

Ein Stack wird — wie schon erwähnt — sehr häufig benötigt. Aus diesem Grund gibt es in der Java-Bibliothek eine vordefinierte Klasse `Stack`. Hier eine vereinfachte Darstellung der Klasse:

```
public class Stack
Stack() // Konstruktor; erzeugt einen leeren Stack
boolean empty()          // Stellt fest, ob der Stack leer ist.
Object peek()            // Liefert das Element an der Spitze
                         // des Stacks,
                         // ohne es vom Stack zu entfernen.
Object pop()             // Liefert das Element an der Spitze
                         // des Stacks
                         // und entfernt es.
Object push(Object item) // Bringt ein Element auf die Spitze
                         // des Stacks.
                         // Rueckgabewert ist unwichtig.
```

Diese Klasse wurde so angelegt, dass sie prinzipiell mit beliebigen Datentypen zurecht kommt. Der Datentyp muss `Object` sein, was eigentlich nichts anderes bedeutet, als dass es ein Zeiger auf einen eigenen beliebigen Datentyp in der Anwendung sein muss.
Wie arbeitet man damit in der Praxis?

1. Die Stack-Klasse wird durch `import java.util.Stack;` in einem Programm verfügbar gemacht.

2. Damit man mit den einfachen DatenTypen wie `int`, `float`, `double` usw. arbeiten kann, gibt es in `java.lang` z.B. eine immer verfügbare Klasse `Integer`. Es ist eine sogenannte *wrapper-class*, d.h. diese Klasse „verhüllt“ (wraps) einen einfachen Datentyp (wrap: einhüllen), damit man ihn als `Object` in anderen Klassen (wie z.B. in `Stack`) benutzen kann. Letztlich wird dadurch nur ein Zeiger aufgebaut, der auf einen Wert des einfachen Datentyps zeigt.

3. Wie arbeitet man mit diesen Klassen?

   a) Anlegen eines Stacks:
   `Stack meinstack = new Stack();`

   b) *Push*en einer Integer-Zahl auf den Stack:
   `meinstack.push(new Integer(13));`
   Der `int`-Wert 13 wird mit `Integer(13)` zu einem Zeiger (bzw. zu einem `object`) gemacht, was man übergeben kann.

c) *Pop*en eines `Integer`-Typs:

```
Integer intobj;
intobj = (Integer)meinstack.pop()
```

Der Rückgabewert der Methode ist vom Typ `Object`. Damit er als Typ `Integer` interpretiert wird, muss `(Integer)` voran gestellt werden. Das funktioniert, weil alle `Object`-Typen sowieso Zeiger sind.

d) Verarbeiten des `int`-Wertes eines `Integer`-Typs:
if (intobj.intValue()=3) ...