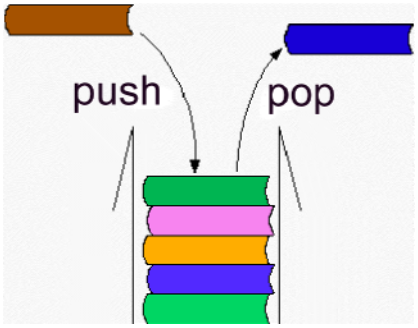
Keller (Stack) in der Informatik

Ein Keller, auch Kellerspeicher, Stapel oder Stack genannt, ist ein Objekt, um Daten zu speichern. Die Besonderheit des Kellers besteht darin, dass stets das zuletzt eingefügte Element eines Kellers als erstes wieder entfernt werden muss. Die Kellerverwaltung arbeitet nach dem LIFO-Prinzip: Last In First Out. Was zuletzt in den Keller kam, kommt zuerst aus dem Keller auch wieder raus.
Man kann sich einen Keller als eine Bücherkiste vorstellen, in die man einzeln Bücher legen und wieder herausnehmen kann. Üblicherweise stehen folgende fünf Kelleroperationen zur Verfügung:



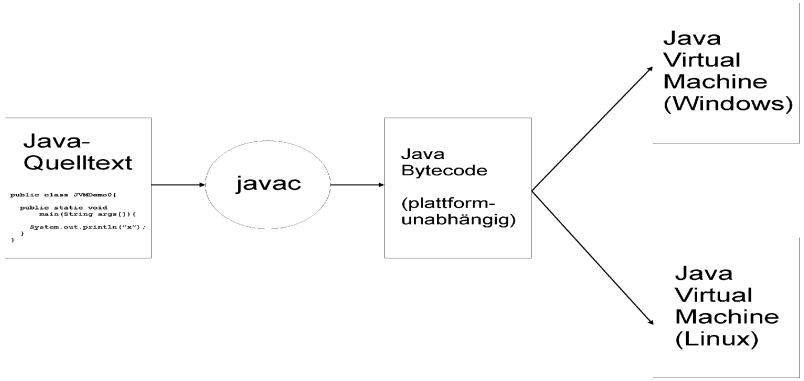
- **Push** legt ein Element auf dem Keller ab.
- **Pop** holt ein Element vom Keller.
- **Top** schaut nach, welches Element oben auf dem Keller liegt. (Fehler, wenn Keller leer!)
- **Init** legt einen neuen Keller an und initialisiert ihn.
- **Empty** prüft nach, ob der Keller leer ist.

Man kann mit Hilfe von Kellern sogar rechnen, wie wir weiter unten erkennen können.

Andere Beispiele für Keller sind z.B. Kartenstapel, Aktenstapel, ineinander geschachtelte Puppen, gestapelte Einkaufskörbe usw.

Keller spielen in der Informatik in vielen Bereichen eine große Rolle, insbesondere auch für bei den heute üblichen Virtual Machines. Letztere gewinnen immer mehr an Bedeutung, da die heute die Rechner schnell genug sind, um Programme so zu verarbeiten, ohne das zu große Performance-Verluste auftreten. Wichtige Virtual Machines, die nach dem Konzept eines Kellers arbeiten, sind die Java Virtual Machine und die Common Language Runtime, die Microsoft im Rahmen der .NET-Platform vertreibt.

Die Java Virtual Machine (JVM)



Wie wir bereits wissen, muss man, um aus einem Java-Quelltext ein lauffähiges Programm zu erzeugen, den Quelltext zunächst mit Hilfe des Java-Compilers javac in eine binäre Form kompilieren. Auf diese Binärdatei mit der Dateiendung .class werden wir nun etwas näher

eingehen. Javac erzeugt keinen richtigen Maschinencode, sondern sogenannten Byte-Code, der unabhängig vom Prozessor und vom Betriebssystem ist. Deshalb brauchen wir zum Ausführen von Java-Programmen immer das Programm java. Damit wird die Java Virtual Machine gestartet und der Bytecode darin ausgeführt. Der Name Byte-Code kommt übrigens daher, dass für die Darstellung von Befehlen jeweils ein Byte verwendet wird.

Das Besondere an Java ist, dass man Java-Programme einmal kompilieren muss und dass man diese auf allen Computern, auf denen eine Java Virtual Machine existiert, ohne (größere) Änderungen mit dem gleichen Byte-Code ausführen kann.
Die JVM besteht aus drei Teilen: dem Classloader, der Execution Engine und dem Native Method Interface. Der *Classloader* lädt Java class-Dateien in den Speicher und bereitet sie für die Ausführung vor. Nachdem die class-Datei im Speicher der JVM ist, tritt die *Execution Engine* auf den Plan. Sie führt schrittweise den Bytecode aus. Falls diese auf Interaktion mit dem Betriebssystem angewiesen ist, z.B. bei Ein- und Ausgaben, wird diese über *das Native Method Interface* durchgeführt.
Die Ausführung des Byte-Codes kann entweder durch schrittweises *Interpretieren* erfolgen oder auch direkt vor dem Aufruf in den Maschinencode der Hardware-Plattform compiliert werden (*Just-in-time - Compilierung*).

Beispiel: Addition von zwei Zahlen in Java
Den Bytecode in class-Dateien kann man mit dem im JDK mitgelieferten Programm *javap* untersuchen. **Dies ist aber nur für eigene Programme legal!**

```
public class JVMDemo2{

  public static void
   main(String args[]){
    int a,b,c;

    a=2;
    b=3;
    c=a+b;
  }
}
```

gekürzte Ausgabe von javap:

```
public static void
main(java.lang.String[]);
  Code:
    0:    iconst_2
    1:    istore_1
    2:    iconst_3
    3:    istore_2
    4:    iload_1
    5:    iload_2
    6:    iadd
    7:    istore_3
    8:    return
```

Der Java-Quelltext auf der linken Seite sollte klar sein: Es werden drei int-Variablen deklariert und in a der Wert 2 gespeichert und in b der Wert 3. Schließlich werden a und b und der Wert in c gespeichert.
Rechts ist der passende Byte-Code in gekürzter Form dargestellt: Wichtig für das Verständnis ist, dass **alle Rechnungen in der JVM über den Keller ablaufen**. Konstanten und Variablen werden jedoch an bestimmten Speicherstellen gespeichert, genauso wie bisher!
Die 2. Konstante wird mittels iconst_2 auf den Keller gelegt. Der oberste Wert vom Keller wird dann durch istore_1 in der ersten Variablen abgelegt. In den Zeilen 4 und 5 werden dann die Werte der 1. und 2. Variablen auf den Keller gelegt und anschließend in Zeile 6 durch iadd addiert. Das Ergebnis der Rechnung wird wieder auf den Keller gelegt und dann schließlich in der 3. Variablen gespeichert (Zeile 7). Mit return wird in diesem Fall das Programm beendet.

Unterschied zwischen JVM und von-Neumann-Architektur:

Bei der von-Neumann-Architektur wird jeder Befehl zum Verschieben von Werten im Speicher oder zum Rechnen über Register ausgeführt. Bei Virtual Machines hingegen geschieht dies alles über den Keller.