Übungen zu Vorlesung
Praktische Informatik I
Wintersemester 2002/03

Methodenaufrufe und Laufzeitstack

(β-Version - Korrekturen, Ergänzungen, Kommentare und gute Wünsche bitte an `ulrichh@informatik.uni-bremen.de`)

```
class  ZweiFakultaet {
     static int  fak (int n) {
             int temp;     // lokale Variable
             if (n == 1)  return 1;
                else { temp = n * fak(n-1);
                       return temp; }
             }
     public static void main (String[] args){
                        int x;
                        x = fak(2);
                        }
    }
```

Zur Laufzeit dieses Programms wird dann zunächst für die `main` Methode ein Stackframe aufgebaut, dabei ist als Aufrufender ("Caller") die Java Virtual Machine eingetragen. Weil noch keine Methode aufgerufen wurde, enthält das Feld "Called" nur einen Nullverweis. Der Programcounter hat einen Wert *wert*. Für die lokale Variable x wird genau ein Wort reserviert (da vom Typ `int`) und mit 0 initialisiert. Auf dem Operandenstack liegen die Parameter der Methode, hier also eine *Referenz* auf *args* (Siehe Fig. 1).

Als nächste Anweisung soll also `x = fak (2)` abgearbeitet werden. Zur Vorbereitung des Methodenaufrufs wird der Parameter auf dem Operandenstack abgelegt (mittels einer `push` Operation - Fig. 2).

Danach erfolgt der eigentliche Methodenaufruf, die wir etwa mit `jump fak` bezeichnen. Für die Methode `fak` wird ein neuer Stackframe aufgebaut, mit den entsprechenden Einträgen:
Der Caller: - Eintrag des neuen Stackframes zeigt auf den zur aufrufenden Methode (hier also: main) gehörenden Stackframe.
Im aufrufenden Stackframe wird ein Verweis auf den neuen Stack eingetragen (Called:).
Um beim Rücksprung aus der aufgerufenen Methode sinnvoll weiterarbeiten zu können, wird der Wert des Program Counters gespe-

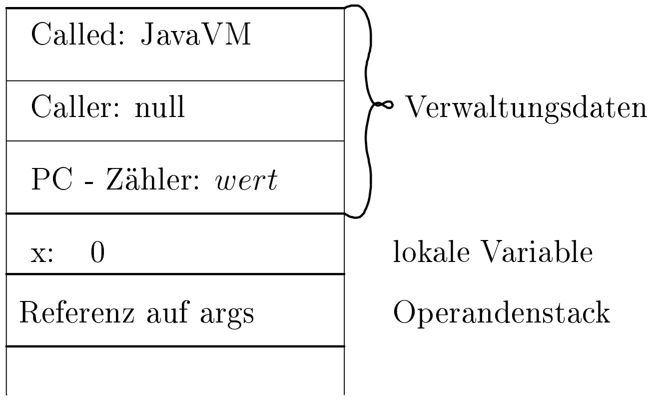

Figure 1: Stack nach `int x;`

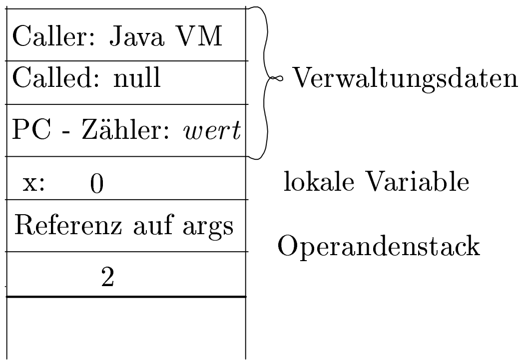

Figure 2: Stack vor Aufruf von `fak (2)`

ichert, der für die main-Methode bei der `jump`-Anweisung steht. Damit erhalten wir ein Bild des Laufzeitstacks, wie in Fig. 3 angegeben.

Als letzten Schritt der internen Abläufe beim Methodenaufruf erfolgt die Übergabe der aktuellen Parameter. Diese waren auf dem Stackframe der aufrufenden Methode abgelegt worden, dort werden sie jetzt wieder entfernt und auf den Operandenstack der aufgerufenen Methode abgelegt (Fig. 4). Nun kann der zur aufgerufenen Methode `fak` gehörende Code ausgeführt werden.

Die Methode `fak` testet zunächst, ob $n == 1$. Da dies nicht zutrifft, wird der `else`-Zweig ausgeführt. Um `temp` zu berechnen, muß n auf dem Operandenstack liegen, $n - 1$ wird als Parameter für den erneuten Aufruf von `fak` benötigt. Somit entspricht Fig. 5 dem Stackbild beim Aufruf von `fak(1 )`, analog zu Fig. 2 für den

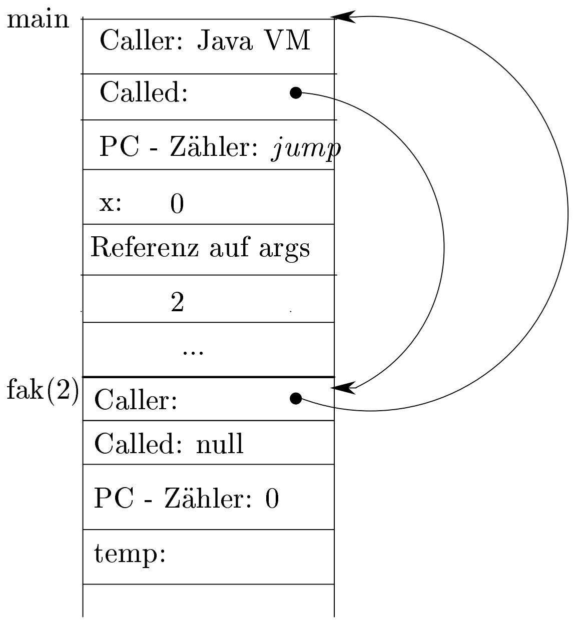

Figure 3: Neuer Stackframe für `fak (2)`

Aufruf von `fak(2)`.

Die Erzeugung des neuen Stackframes für `fak (1)` erfolgt völlig analog zum ersten Methodenaufruf, das Resultat ist auf der linken Seite von Fig. 6 zu sehen. Diese Methode testet erneut, ob der Wert des obersten Elements auf dem Stack == 1 ist, entfernt diesen (pop) und scheibt den Rückgabeparameter auf den Stack (push) - also wieder eine 1.

Das nächste Bild (Fig. 7) illustriert den Rücksprung von `fak(1)` zur aufrufenden Funktion. Durch den Verweis auf die aufrufende Methode (Caller) ist bekannt, zu welchem Stackframe zurückgesprungen werden muß. Der Rückgabeparameter wird vom aktuellen Stack entfernt und auf den Stack der aufrufenden Methode ge"push"t. Um die Abarbeitung von `fak(2)` fortsetzen zu können, wird der program counter auf den um 1 erhöhten gespeicherten Wert (also $jump' + 1$) gesetzt. Damit ist nun alle relevante Information übertragen bzw. wiederhergestellt worden, und der zu `fak(1)` gehörige Stackframe kann entfernt werden.

Die durch den Methodenaufruf unterbrochene Berechnung von `temp` in `fak(2)` wird jetzt fortgesetzt: n lag schon auf dem Stack,

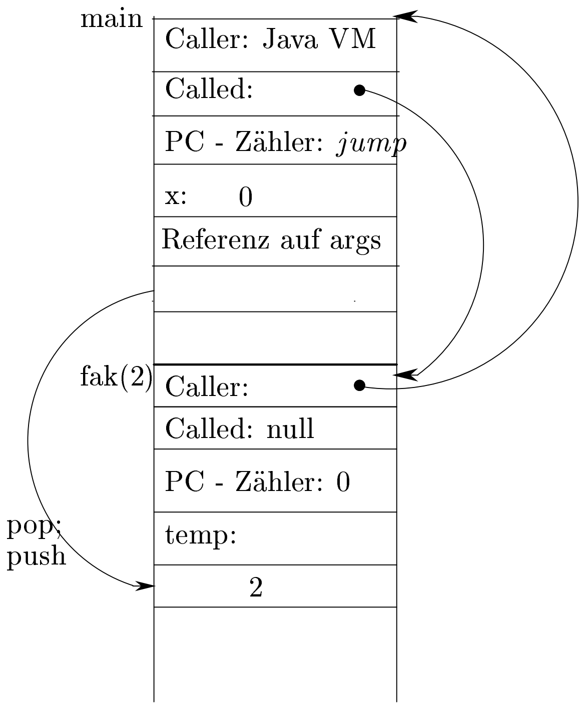

Figure 4: Stack ist bereit zur Ausführung von `fak (2)`

jetzt steht auch $fak(n-1)$ zur Verfügung. Die Java Virtual Maschine (JVM) arbeitet als Stack-Maschine, die Operationen verwenden die Daten auf dem Operandenstack. Die Multiplikationsoperation $temp = n * fak(n-1)$ kann also jetzt erfolgen, indem die Operanden vom Stack genommen werden, die Multiplikation ausgeführt wird und das Resultat wieder auf den Stack geschrieben wird. Anschließend wird dieses vom Stack genommen und als lokale Variable *temp* gespeichert. Der durch $*$) markierte Ablauf im linken Bild von Fig. 8 kann im Sinn der JVM Operationen als $push(pop * pop); store_{temp}$ dargestellt werden.

Die Methode erreicht nun ihre `return`-Anweisung, der Rückgabeparameter (*temp*) wird auf den Operandenstack abgelegt.

Der Rücksprung zur main-Methode erfolgt analog dem von `fak(1)` und ist in Fig. 9 dargestellt.

Abschließend wird dieser Rückgabewert noch der lokalen Variablen x zugewiesen.

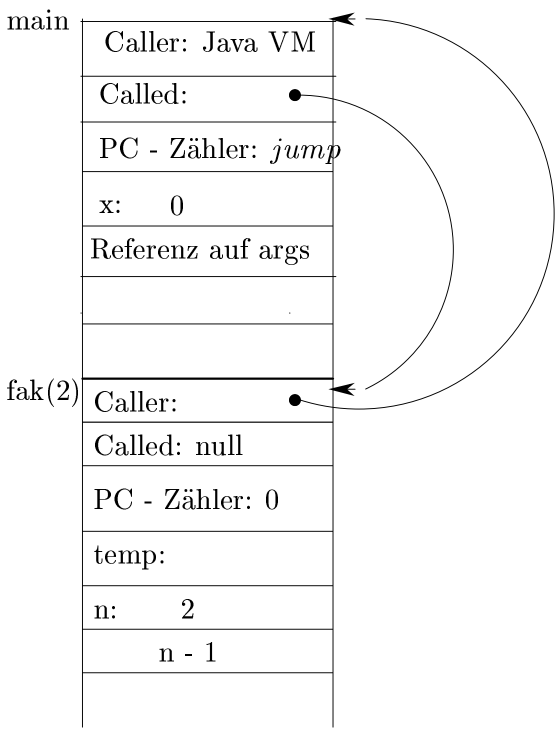

Figure 5: Stack vor Aufruf von `fak (1)`

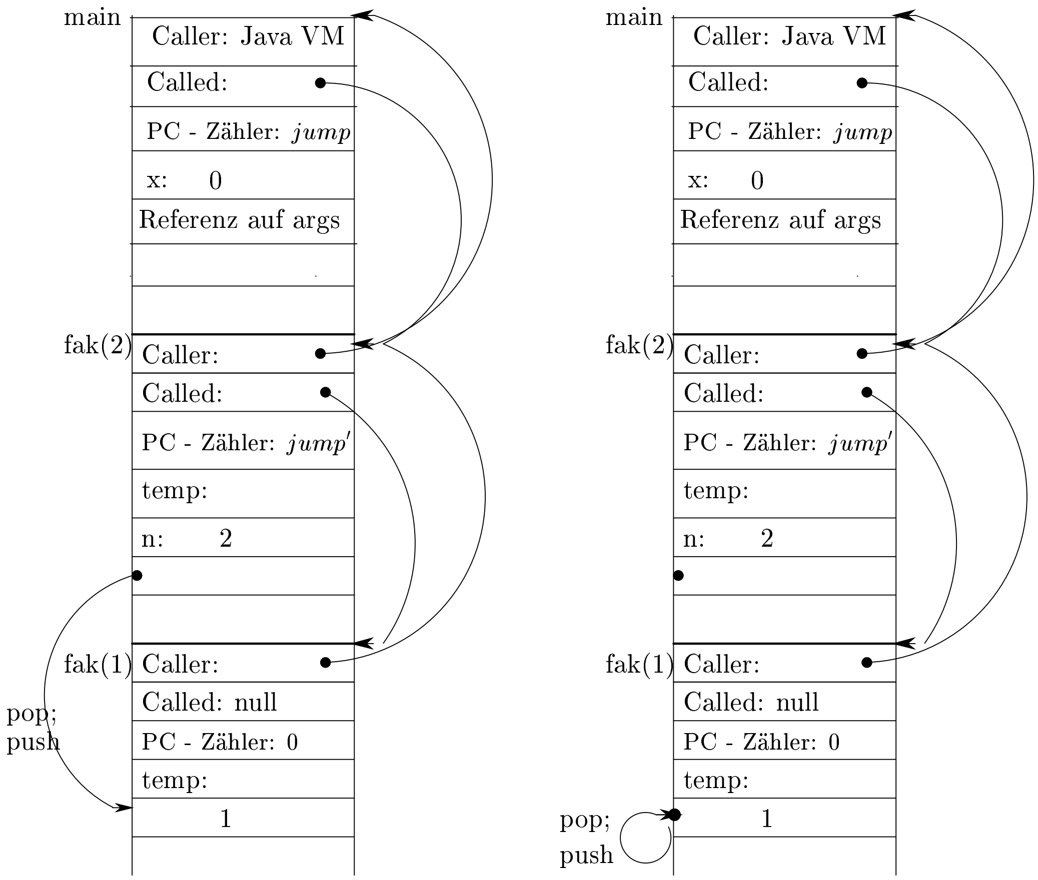

Figure 6: Stack für `fak (1)`

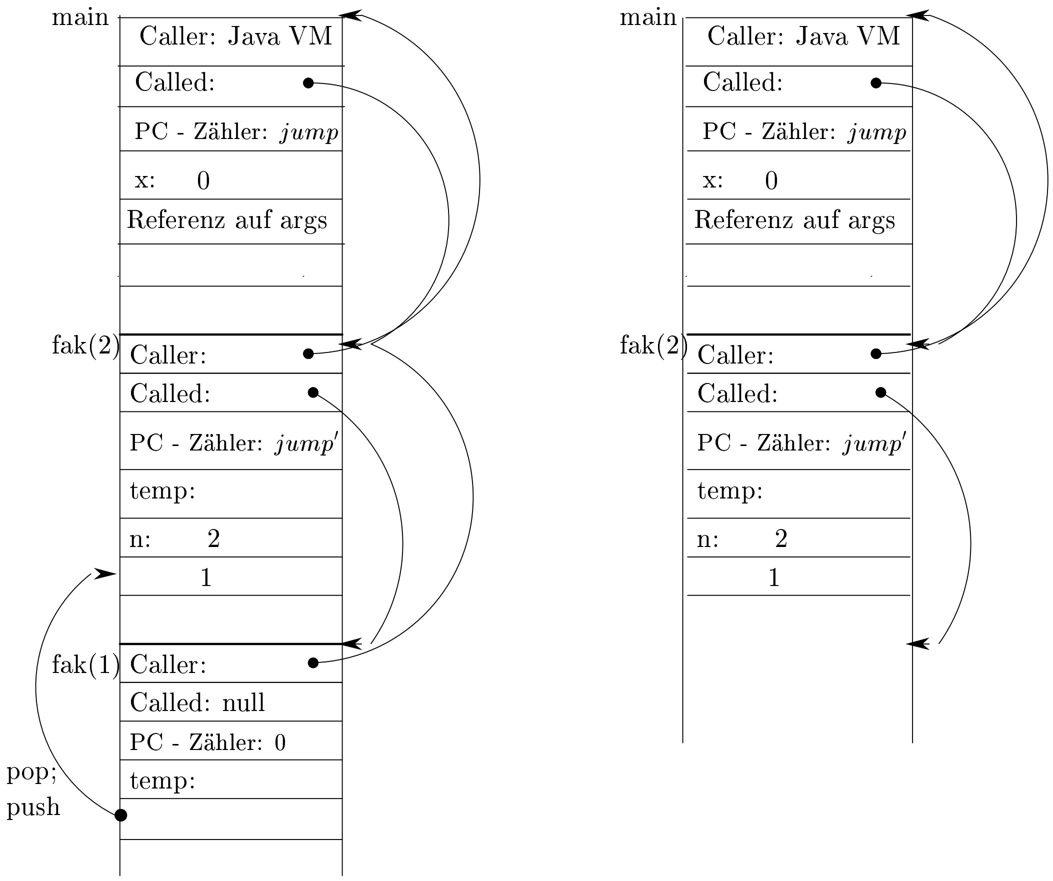

Figure 7: Rücksprung von `fak (1)`

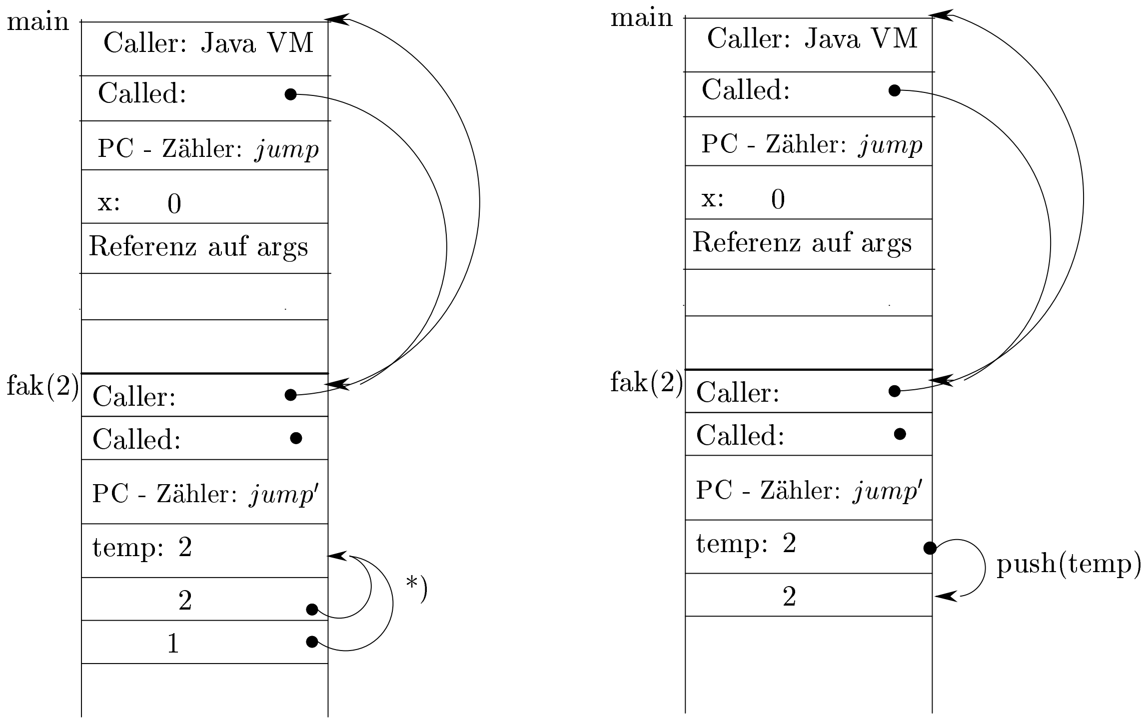

Figure 8: Ablauf von `fak (2)`

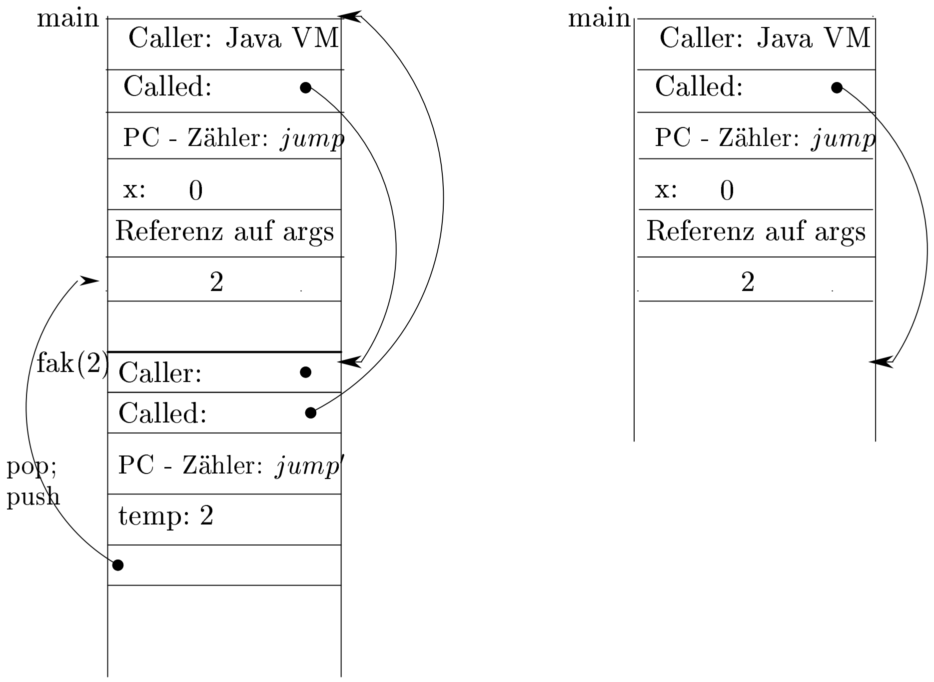

Figure 9: Rücksprung zur main-Methode