Informatik II (D-ITET)
Frühjahrssemester 2014
Prof. F. Mattern
C. Beckel (beckel@inf.ethz.ch)

ETH Zürich
Institut für Pervasive Computing
Gruppe für Verteilte Systeme
http://vs.inf.ethz.ch

Übungsserie Nr. 4

Ausgabe: 19. März 2014
Abgabe: 26. März 2014

Hinweise

Für diese Serie benötigen Sie das Archiv
http://vs.inf.ethz.ch/edu/FS2014/I2/downloads/u4.zip.

1. Aufgabe: (8 Punkte) Ein wachsender Stack

Bei der Verwendung des Stacks auf Seite 83 (Folie 205) im Skript muss man im Vorfeld wissen, wie viele Elemente später maximal auf dem Stack benötigt werden. Manchmal weiss man das allerdings nicht. Eine einfache, wenn auch aufwändige Möglichkeit, dynamisch wachsende Stacks zu ermöglichen, ist, beim Erreichen der Kapazitätsgrenze einen neuen Puffer mit doppelter Grösse anzulegen und alle Werte vom alten in den neuen Puffer zu kopieren. Die Klasse *u4a1.Stack* soll diese Idee umsetzen.

(1a) *(1 Punkt)* Implementieren Sie den Konstruktur der Klasse.

(1b) *(2 Punkte)* Implementieren Sie die Methode *toString* unter Verwendung von *StringBuffer*.

(1c) *(2 Punkte)* Implementieren Sie die Methode *grow*. Am Ende dieser Methode besitzt der Stack einen Puffer doppelter Grösse, der mit den bisherigen Werten gefüllt ist.

(1d) *(3 Punkte)* Implementieren Sie die Funktionen *push*, *pop*, *peek*, *empty*, *size* und *capacity* entsprechend der vorhandenen Javadoc-Dokumentation.

2. Aufgabe: (8 Punkte) Stackimplementierung der Ackermann-Funktion

Die Ackermann-Funktion $A(n, m)$ ist eine extrem schnell wachsende, mathematische Funktion, die insbesondere in der theoretischen Informatik eine bedeutende Rolle spielt[1]. Sie ist für alle $n, m \in \mathbb{N}_0$ definiert durch:

$$\begin{aligned} A(0, m) &= m + 1 \\ A(n + 1, 0) &= A(n, 1) \\ A(n + 1, m + 1) &= A(n, A(n + 1, m)) \end{aligned}$$

(2a) *(2 Punkte)* Die Berechnung von $A(1, 1)$ führt zu folgenden rekursiven Teilberechnungen und -ergebnissen:

```
A(1,1)
  A(1,0)
    A(0, 1)
    <- 2
  <- 2
  A(0, 2)
  <- 3
<- 3
```

Geben Sie entsprechend der obigen Form die Berechnung von $A(2, 1)$ an.

(2b) *(3 Punkte)* Beschreiben Sie in Pseudocode einen *iterativen* Algorithmus, der die Ackermann-Funktion mit Hilfe eines Stacks berechnet. Verwenden Sie dazu die Stackmethoden *push*, *pop* und *size*.

Hinweis: Eine Iteration nimmt die Werte für n und m vom Stack und hinterlässt entweder das Ergebnis oder die Parameter für die nächste Iteration auf dem Stack. Eine Iteration entspricht dabei einem rekursiven Aufruf.

(2c) *(3 Punkte)* Implementieren Sie die Methode *u4a2.IterativeAckermann.A* mittels Ihres Algorithmus aus Teilaufgabe *b*. Verwenden Sie dabei Ihre Stackimplementierung aus Aufgabe 1.

Hinweis: Geben Sie am Anfang jeder Iteration die Stringrepräsentation des Stacks auf der Standardausgabe aus. Dann erscheinen diese Schnappschüsse auf der Console und Sie können damit ggf. Fehler analysieren.

[1] `http://de.wikipedia.org/wiki/Ackermannfunktion`

3. Aufgabe: (7 Punkte) Java-Bytecode

(3a) *(2 Punkte)* Geben Sie den Bytecode der Methode *u4a2.RecursiveAckermann.A* an.

Hinweis: Verwenden Sie den Disassembler *javap* mit der Option *-c*. Analog zum Starten der JVM müssen sie dabei die zu analysierende Klasse und nicht die *.class*-Datei angeben.

(3b) *(3 Punkte)* Geben Sie für die folgenden Ausschnitte aus dem Javacode von *u4a2.RecursiveAckermann.A* die korrespondierenden Zeilen aus dem Bytecode aus Teilaufgabe a an.

- `if (n==0)`
- `return m + 1`
- `if (m==0)`
- `return A(n-1,1)`
- `return A(n-1, A(n, m-1))`

Hinweis: Falls Sie für einzelne Mnemonics die Bedeutung nicht kennen, schlagen Sie diese in der Dokumentation[2,3] nach.

(3c) *(2 Punkte)* Vergleichen Sie den Bytecode aus Teilaufgabe a mit Ihrem Javacode aus Aufgabe 2c. Was fällt Ihnen auf?

[2] `http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-6.html#jvms-6.5`
[3] `http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings`