Übung 3: *Referenzen* und *AspectJ*

Aufgabe 1: Referenzen

a) Unerreichbare zirkulär-verhängte Liste

Demonstrieren Sie, dass der *Carbage-Collector* eine unerreichbare (*unreachable*), zirkuläre-verhängte Liste erkennt und aus dem Speicher entfernt (vgl. erste Folie von Kapitel *Referenzen (Folie 83)*: auf der Skizze ist oben rechts eine solche Liste dargestellt).

Allozieren Sie dazu hintereinander drei Objekte welche im Heap mit dem Operator *new* jeweils 150 Mega-Byte Speicher allozieren und verhängen Sie diese zirkulär (entsprechende Klasse erstellen).

Starten Sie dazu die *Virtual-Machine* mit einer initialen Heap-Grösse von 200 und einer maximalen von 500 Mega-Byte
(Eclipse-Menü:*Run>Open Run Dialog…>'Projekt-Name'>*
Tab: (x)=Arguments>VM arguments: `-Xms200m -Xmx500m`).

Setzen Sie im Weiteren zur Visualisierung das Tool ***jconsole*** ein
(*C:\Programs\Java\jdk1.6.2\bin\jconsole.exe*).
Damit die *jconsole* mit der *Virtual-Machine* der Applikation kommunizieren kann muss diese mit einem weiteren, zusätzlichen Argument gestartet werden (wieder bei *VM arguments*): `-Dcom.sun.management.jmxremote`

Hinweis:
Um das Verhalten in der *jconsole* gut sehen zu können tut man am besten in der Applikation zwischen den einzelnen Schritten einen Sleep (z.B. `Thread.sleep(3000);`) einbauen oder von der Konsole User-Input lesen (z.B. `new java.util.Scanner(System.in).nextLine();`).

b) OutOfMemory-Exception mit Soft- und Weak-Referenzen (optional)

Provozieren Sie *OutOfMemory-Exceptions*.
Untersuchen Sie in diesem Zusammenhang das Verhalten resp. die Differenzen bei der Benutzung von *Soft-* resp. *Weak-Referenzen* (werden *Soft-Referenzen* wirklich erst später als *Weak-Referenzen* gelöscht?)

Test-Applikation (Pseudo-Code):

```
start VM with -Xms100m -Xmx100m
forever  // -> while no Memory-Exception ;-)
    for 1 to 100 do
      allocate a strongly referenced 100KB-Object
      allocate a softly referenced 100KB-Object
      allocate a weakly referenced 100KB-Object
    analyse Status of all softly and weakly References
```

Hinweis:
Für die Analyse des Zustandes der *Soft-* und *Weak-Referenzen* (wurden die entsprechenden Objekte vom Garbage-Collector schon freigegeben resp. hat er auf den Referenzen *clear()* schon aufgerufen) könnten z.B. auch *ReferenceQueue's* verwendet werden.

Aufgabe 2: AspectJ

Bei der Applikation *BinarySearchTree* (von der Übung 1 / Aufgabe 2b) soll *Logging* mittels *java.util.Logging* hinzugefügt werden, aber *ohne dass die bestehenden Sourcen verändert werden* !

Setzten Sie dazu *AspectJ* ein.

Es soll pro Methoden-Aufruf eine Zeile in einem Log-File "***logging.txt***" erzeugt werden mit dem Methoden-Namen und den Parameter-Werten.

Beispiel mit einer imaginären Test-Applikation:

```
Test test = new Test();
test.meth1(1, 2, "Hallo MUP");
test.meth2(11, 22, "Hallo MUP zum zweiten");
```

Im Logfile "*logging.txt*":

```
meth1(1, 2, Hallo MUP)
meth2(11, 22, Hallo MUP zum zweiten)
```

Hinweise:

- Testen Sie ihren Aspekt am besten zuerst an einer eigenen, kleinen Test-Klasse (wie in obigem Beispiel).
- Machen Sie die Log-Ausgaben zuerst nur auf *System.err*, und erst ein einem zweiten Schritt mittels *java.util.Logging* in das File „*logging.txt*“.

Vorgehen:

- Erzeugen Sie ein normales Java-Projekt mit der Musterlösung von Übung 1 / Aufgabe 2b (*BinarySearchTree.java, TreePrinter.java, TreePrinterAcc.java*).
- Konvertieren Sie das *Java-Projekt* in ein *AspectJ-Projekt* mit: Im *Packet-Explorer* das Projekt selektieren, dann Context-Menü (rechte Maus): *AspectJ Tools>Convert to AspectJ Project*
- Dann kann mit *New>Other...>AspectJ>Aspect* ein neuer Aspekt erzeugt werden.
- Mit *Run as>AspectJ/Java Application* wird die Java-Applikation in Kombination mit dem Aspekt zur Ausführung gebracht.

