```
  1 //=============================================================================
  2 //      * Letsch Informatik *      www.LetsInfo.ch      CH-8636 Wald
  3 //      Beratung, Ausbildung und Realisation in Software-Engineering
  4 //=============================================================================
  5 // Project   : Master of Advanced Studies in Software-Engineering / 2008
  6 // Modul     : Java - Advanced Concepts
  7 // Title     : Übung 3: "Referenzen und AspectJ" / Aufgabe 1: "Referenzen"
  8 // Author    : Thomas Letsch
  9 // Tab-Width : 2
 10 /*///===========================================================================
 11 * Description: Hard-, Soft- und Weak-Referenzen, Referenz-Qeues und
 12 *              OutOfMemoryError.
 13 * History    : 23.03.07: Initial Version.
 14 *            : 24.01.08: Kosmetik.
 15 * CVS        : $Revision: 1.5 $  $Date: 2008/01/24 20:32:11 $
 16 /*///===========================================================================
 17 //     1         2         3         4         5         6         7         8
 18 //345678901234567890123456789012345678901234567890123456789012345678901234567890
 19 //=============================================================================
 20
 21 import java.lang.ref.ReferenceQueue;
 22 import java.lang.ref.SoftReference;
 23 import java.lang.ref.WeakReference;
 24 import java.util.ArrayList;
 25 import java.util.List;
 26 import java.util.Scanner;
 27
 28 import static java.lang.System.out;
 29
 30 class TestObject {
 31
 32   private byte[] mByteArr;
 33   private int mObjNr;
 34   private static int mNrOfObjs;
 35
 36   TestObject(int pSize) {
 37     out.format("TestObject::TestObject(%,d):\n", pSize);
 38     out.flush();
 39     mByteArr = new byte[pSize];
 40     mObjNr = ++mNrOfObjs;
 41     if (getClass() == TestObject.class) print();
 42   }
 43
 44   public void finalize() {
 45     out.println("========================");
 46     out.println("TestObject::finalize():");
 47     print();
 48     out.flush();
 49   }
 50
 51   public void print() {
 52     out.format("Object-Nr: %,13d\n", mObjNr);
 53     out.format("Size:      %,13d\n", mByteArr.length);
 54     out.flush();
 55   }
 56
 57   public int getNr() {
 58     return mObjNr;
 59   }
 60
 61 } // End of class TestObject
 62
 63
 64 class LinkedTestObject extends TestObject {
 65
 66   private TestObject mNextObject;
 67
 68   LinkedTestObject(int pSize, TestObject pNextObject) {
 69     super(pSize);
 70     mNextObject = pNextObject;
 71     if (getClass() == LinkedTestObject.class) print();
 72   }
 73
 74   public void print() {
 75     super.print();
 76     out.print("Next Object: ");
 77     Object next = mNextObject == null ? "none" : mNextObject.getNr();
 78     out.format("%11s\n", next);
 79     out.flush();
 80   }
 81
 82   public void setNextObject(LinkedTestObject pNextObject) {
 83     mNextObject = pNextObject;
 84   }
 85
 86   public void finalize() {
 87     super.finalize();
 88     mNextObject = null;
 89   }
 90
 91 } // End of class LinkedTestObject
 92
 93
 94
 95
 96
 97 public class References {
 98
 99   public static void main(String[] args) {
100
101     if ((args.length == 1) && (args[0].equals("memory"))) {
102       testOutOfMemoryException(args[0]);
103     }
104     else {
105       testCircularList();
106     }
107
108   }
109
110
111
```

```
112|
113|   static void testCircularList() {
114|     printMemoryUsage();
115|     final int SIZE = (int)150e6; // 150 MB
116|     out.println("\nTest with Circular-List: 3 Objects with "+SIZE/(int)1e6+" MB\n")
   +;
117|     LinkedTestObject l1 = new LinkedTestObject(SIZE, null);
118|     printMemoryUsage();
119|     pause("\nHit RETURN for allocation of second Object:");
120|     LinkedTestObject l2 = new LinkedTestObject(SIZE, l1);
121|     printMemoryUsage();
122|     pause("\nHit RETURN for allocation of third Object:");
123|     LinkedTestObject l3 = new LinkedTestObject(SIZE, l2);
124|     printMemoryUsage();
125|     pause("\nHit RETURN to connect Object Nr.1 with Object Nr.3:");
126|     l1.setNextObject(l3);
127|     l1.print();
128|     pause("\nHit RETURN for Disconnecting all Strong-References.");
129|     l1 = l2 = l3 = null;
130|     printMemoryUsage();
131|     pause("\nRun gc():");
132|     System.gc();
133|     try {Thread.sleep(1000);} catch (InterruptedException e) {}
134|     printMemoryUsage();
135|     pause("\nHit RETURN for Instantiation of TestObject(100MB) \nto provocate real
   +Deallocation of Memory of Object Nr. 1 to 3 ! ;-)\n");
136|     TestObject t = new TestObject((int)100e6);
137|     printMemoryUsage();
138|     pause();
139|   }
140|
```



```
141|
142| /* Session-Log:
143|
144| $ java -Dcom.sun.management.jmxremote -Xms200m -Xmx500m References
145| ============================
146| maxMemory():     520'290'304
147| totalMemory():   208'142'336
148| freeMemory():    207'882'800
149| ============================
150|
151| Test with Circular-List: 3 Objects with 150 MB
152|
153| TestObject::TestObject(150'000'000):
154| Object-Nr:             1
155| Size:        150'000'000
156| Next Object:        none
157| ============================
158| maxMemory():     520'290'304
159| totalMemory():   208'142'336
160| freeMemory():     57'882'784
161| ============================
162|
163| Hit RETURN for allocation of second Object:
164|
165| TestObject::TestObject(150'000'000):
166| Object-Nr:             2
167| Size:        150'000'000
168| Next Object:           1
169| ============================
170| maxMemory():     520'290'304
171| totalMemory():   419'094'528
172| freeMemory():    118'629'424
173| ============================
174|
175| Hit RETURN for allocation of third Object:
176|
177| TestObject::TestObject(150'000'000):
178| Object-Nr:             3
179| Size:        150'000'000
180| Next Object:           2
181| ============================
182| maxMemory():     520'290'304
183| totalMemory():   520'290'304
184| freeMemory():     69'515'800
185| ============================
186|
187| Hit RETURN to connect Object Nr.1 with Object Nr.3:
188|
189| Object-Nr:             1
190| Size:        150'000'000
191| Next Object:           3
192|
193| Hit RETURN for Disconnecting all Strong-References.
194|
195| ============================
196| maxMemory():     520'290'304
197| totalMemory():   520'290'304
198| freeMemory():     69'515'800
199| ============================
200|
201| Run gc():
202|
203| ========================
204| TestObject::finalize():
205| Object-Nr:             3
206| Size:        150'000'000
207| Next Object:           2
208| ========================
209| TestObject::finalize():
```

```
Object-Nr:             2
Size:        150'000'000
Next Object:           1
=======================
TestObject::finalize():
Object-Nr:             1
Size:        150'000'000
Next Object:           3
===========================
maxMemory():     520'290'304
totalMemory():   520'290'304
freeMemory():     68'869'384
===========================

Hit RETURN for Instantiation of TestObject(100MB)
to provocate real Deallocation of Memory of Object Nr. 1 to 3 ! ;-)


TestObject::TestObject(100'000'000):
Object-Nr:             4
Size:        100'000'000
===========================
maxMemory():     520'290'304
totalMemory():   520'290'304
freeMemory():    419'516'168
===========================
Hit RETURN

*/

```



```

  static void printMemoryUsage() {
    long maxMemory = Runtime.getRuntime().maxMemory();
    long freeMemory = Runtime.getRuntime().freeMemory();
    long totalMemory = Runtime.getRuntime().totalMemory();
    final String LINE = "===========================";
    out.println(LINE);
    out.format("maxMemory():   %,13d\n", maxMemory);
    out.format("totalMemory(): %,13d\n", totalMemory);
    out.format("freeMemory():  %,13d\n", freeMemory);
    out.println(LINE);
  }


  static void pause() {
    pause("Hit RETURN");
  }


  static void pause(String pMessage) {
    out.println(pMessage);
    if (System.getProperty("NoPause") != null) {
      return;  // just for Tests (no waiting ;-)
    }
    new Scanner(System.in).nextLine();
  }

```

```
267|
268|   static void testOutOfMemoryException(String pKind) {
269|
270|     out.println("\nDemonstration of Soft- and WeakReferences, ReferenceQueues and M
   +emory-Exception");
271|     out.println("-> SoftReferences are not cleared as long as possible!\n");
272|
273|     final int SIZE = (int)100e3;  // 100 KB
274|
275|     List<byte[]> strongRefList = new ArrayList<byte[]>();
276|     List<SoftReference<byte[]>> softRefList = new ArrayList<SoftReference<byte[]>>(
   +);
277|     ReferenceQueue<byte[]> softQueue = new ReferenceQueue<byte[]>();
278|     List<WeakReference<byte[]>> weakRefList = new ArrayList<WeakReference<byte[]>>(
   +);
279|     ReferenceQueue<byte[]> weakQueue = new ReferenceQueue<byte[]>();
280|     int softRefsCleared = 0;
281|     int weakRefsCleared = 0;
282|     int loops = 0;
283|     try {
284|       while (true) {
285|         loops++;
286|         int i = 0;
287|         for (; i < 100; i++) {
288|           strongRefList.add(new byte[SIZE]);
289|           softRefList.add(new SoftReference<byte[]>(new byte[SIZE], softQueue));
290|           weakRefList.add(new WeakReference<byte[]>(new byte[SIZE], weakQueue));
291|         }
292|         while((softQueue.poll()) != null) {
293|           softRefsCleared++;
294|         }
295|         while((weakQueue.poll()) != null) {
296|           weakRefsCleared++;
297|         }
298|         out.format("Strongly allocated: %,3d MB    Cleared References ->  Soft: %4d
   +   Weak: %4d\n",
299|                    (int)(loops * i * SIZE / 1e6), softRefsCleared, weakRefsCleared)
   +;
300|       }
301|     }
302|     catch(Throwable throwable) {
303|       System.out.println(throwable);
304|     }
305|   } // End of testOutOfMemoryException()
306|
307|
```



```
308|
309| /* Session-Log:
310|
311|
312| $ java -Xms100m -Xmx100m References memory
313|
314| Demonstration of Soft- and WeakReferences, ReferenceQueues and Memory-Exception
315| -> SoftReferences are not cleared as long as possible!
316|
317| Strongly allocated:  10 MB    Cleared References ->  Soft:    0   Weak:   83
318| Strongly allocated:  20 MB    Cleared References ->  Soft:    0   Weak:  188
319| Strongly allocated:  30 MB    Cleared References ->  Soft:    0   Weak:  293
320| Strongly allocated:  40 MB    Cleared References ->  Soft:    0   Weak:  377
321| Strongly allocated:  50 MB    Cleared References ->  Soft:    0   Weak:  468
322| Strongly allocated:  60 MB    Cleared References ->  Soft:  516   Weak:  579
323| Strongly allocated:  70 MB    Cleared References ->  Soft:  516   Weak:  684
324| Strongly allocated:  80 MB    Cleared References ->  Soft:  773   Weak:  794
325| Strongly allocated:  90 MB    Cleared References ->  Soft:  773   Weak:  896
326| Strongly allocated: 100 MB    Cleared References ->  Soft: 1000   Weak:  999
327| java.lang.OutOfMemoryError: Java heap space
328|
329| */
330|
331| /*
332| Resultat:
333| - soft-referenzierte Objekte werden so lange wie möglich 'gehalten'.
334| - nach dem 5.Loop haben die strong- und soft-referenzierten Objekte die 100MB
335|   aufgebraucht.
336|   Nun muss der Garbage-Collector auch beginnen die soft-referenzierten Objekte
337|   zu löschen.
338| - Anstelle den ReferenceQueue's hätte man auch über die Listen mit den Referenzen
339|   iterieren können und dann jeweils mit Reference.get() feststellen können, ob
340|   das referenzierte Objekt bereits freigegeben wurde.
341|
342| Bemerkung:
343| Je nach VM und System können die Resultate etwas differieren.
344|
345| */
346|
347|
348|
349| } // End of class References
```