GP 2	Generative Programming	ST 17, Exercise 4

Deadline: 19.05.2017, 10:30

☐ **Gr. 1, J. Karder, MSc.**
☐ **Gr. 2, P. Fleck, MSc.**

Name _____ **Effort in h** _____

Points _____ **Lecturer** _____

1. Tracing (5 Points)

Implement a simple class with at least one data field and two methods; one of these methods is to be called by the other one. In the test program create an object of this class, and call the methods and access the data fields.

Furthermore, implement an aspect *Tracing* that protocols each method call as well as each access to one of the data fields. Please distinguish between the regular ending of a method and its abortion by an exception. The so generated output could for example look like this:

```
Entering addPositiveValue
Accessing last
Accessed last
Entering setPositiveValue
Accessing positiveValues
Accessed positiveValues
Exiting setPositiveValue
Accessing last
Accessed last
Accessing last
Accessed last
Exiting addPositiveValue
Entering setPositiveValue
Exiting setPositiveValue ERROR: value is not positive (java.lang.IllegalArgumentException)
Exiting addPositiveValue ERROR: value is not positive (java.lang.IllegalArgumentException)
```

As we now want to indent outputs according to their interlacing in order to increase the lucidity of the outputs, implement indentation without changing the original *Tracing* aspect.

2. Caching (5 Points)

Develop a class *BinomialCoefficient* that offers the static method *Calculate* for calculating the binomial coefficient. This calculation is defined as follows:

$$bc(n,0) = 1$$
$$bc(n,n) = 1$$
$$bc(n,m) = bc(n-1,m-1) + bc(n-1,m)$$

Implement an aspect *LogRecursiveCalls* that counts the number of recursive method calls and writes it to the console as soon as the execution of the first call of *Calculate* is finished.

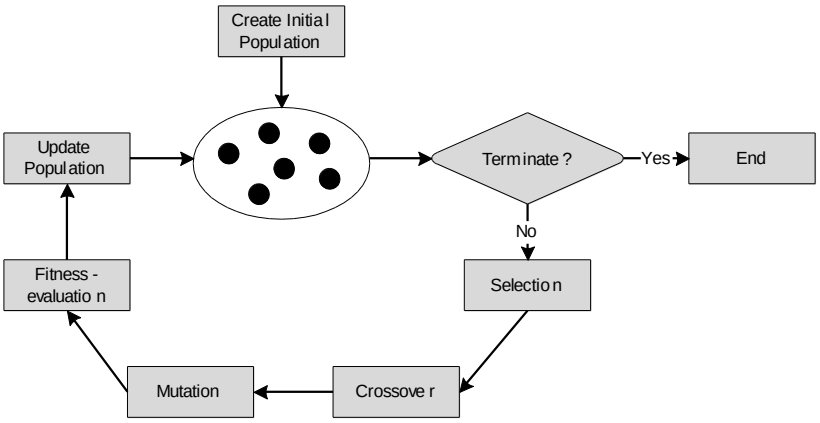
Furthermore, implement another aspect *BinomialCache* that caches calculated (intermediate) results and calls *Calculate* only if the required result is not known yet.

Finally, add an aspect *RuntimeMeasurement* that can be used for measuring and displaying the runtime consumed by a method.

Test all aspects extensively and document, whether the use of *BinomialCache* is reasonable or not.

3. Aspect-Oriented TSP Solver (3 + 3 + 2 + 6 Points)

On elearning you can find a simple program *TSPSolver* that solves the traveling salesman problem (TSP) with genetic algorithms (GAs). GAs are evolutionary algorithms that simulate natural evolution using the adaption of species as optimization technique. The basic workflow of a GA looks like this:



A standard GA with tournament selection (tournament group size $k = 2$), order crossover, inversion mutation and generational replacement is already implemented. Path encoding is used for representing solution candidates, i.e., each roundtrip is encoded as a permutation; e.g., [1 3 2 5 4] represents the roundtrip from city 1 to 3 to … to 4 und back to city 1.

Furthermore, we here already have several aspects that modify the behavior of the program: The aspect *MeasureRuntime* is used for measuring the runtime of an execution of the algorithm, the aspect *RandomSelection* replaces the originally used tournament selection, and the aspects *CyclicCrossover* and *MaximalPreservativeCrossover* replace the respective originally used crossover operator.

Thus, we here see that aspect oriented programming is ideal to enhance (and hopefully improve) the TSPSolver with new concepts. Your task is now to implement the following additional aspects for TSPSolver:

a) Implement an aspect *CountEvaluatedSolutions* that calculates the number of evaluated solutions and eventually writes it to the console as soon as the algorithm has terminated. Based on this implement another aspect *LimitEvaluatedSolutions* that ensures that no next iteration of the algorithm is executed as soon as a predefined maximum number of evaluations is reached.

b) Develop an aspect *Elitism* that adds 1-elitism to the already implemented replacement strategy (generational replacement). I.e., at each generation step the best individual of the parents generation survives and replaces the worst of the new children. Using this aspect we get a monotonous improvement of the quality of the population's best individual.

c) Add an aspect *ProtocolProgress* to TSPSolver that stores best, worst and average qualities of the population and writes it to the console after the algorithm has finished.

d) Unfortunately TSPSolver does not have any graphical visualization. Since we already have a framework for generating SVGs, your final task is now to use this framework for plotting the progress of the best, average and worst quality as well as the best roundtrip found by the algorithm.

1 Tracing

Dieser Abschnitt beschäftigt sich mit der Dokumentation der Aufgabenstellung *Tracing*.

1.1 Lösungsidee

Für das Testen des *tracing application.PositiveValueStore* implementiert, die mehrere verschachtelte Aufrufe sowie das Auslösen einer Ausnahme simuliert. Die Klasse *application.Main* wurde von en Aspekten ausgenommen, damit die implementierten Testmethoden nicht in den *Logs* aufscheinen.

Der Aspekt *TracingAspect traced* alle Konstruktoraufrufe, Methodenaufrufe und Parameterzugriffe von Klassen bevor und nachdem Zugriff. Als *Loggingframework* wird *slf4j* verwendet. Alle Zugriffe werden in die Logdatei *aspectj/logs/tracing-logfile.txt* geschrieben.

Der Aspekt *IndentionLogTrace* realisiert das Einrücken der logs um die Verschachtelung der Methodenaufrufe zu verdeutlichen. Es wird ein *Pointcut* auf alle Methoden der Schnittstelle *org.slf4j.Logger* definiert, und ein *Around JoinPoint* implementiert der den übergebenen Text formatiert, je nachdem ob eine erwarteter Zeichenkette (*Before method, After method, Before constructor, After constructor*) am Anfang des Textes gefunden wurde. Wird eine solche Zeichenkette am Anfang des Textes gefunden wird bei *Before ...* eine definierte Anzahl von Leerzeichen am Anfang des Textes eingefügt und beim Finden des Textes *After ...* einmalig die definierte Anzahl von Leerzeichen am Anfang des Textes entfernt.

1.2 Quelltexte

Folgender Abschnitt enthält die implementierten Klassen, Aspekte und das implementierte Testprogramm.

Listing 1: PositiveValueStore.java

```
1  package application;
2
3  /**
4   * This class represents a positive value store
5   *
6   * @author Thomas Herzog <herzog.thomas81@gmail.com>
7   * @since 05/05/17
8   */
9  public class PositiveValueStore {
10
11     private int[] positiveValues;
12     private int size;
13     private int last = 0;
14
15     public PositiveValueStore(int size) {
16         this.positiveValues = new int[size];
17         this.size = size;
18     }
19
20     public void addPositiveValue(int value) {
21         setPositiveValues(last, value);
22     }
23
24     public void setPositiveValues(int idx,
25                                   int value) {
26         checkIdx(idx);
27         checkValue(value);
28         getPositiveValues()[idx] = value;
```

```
29          last++;
30      }
31
32      public int[] getPositiveValues() {
33          return positiveValues;
34      }
35
36      private void checkValue(final int value) {
37          if (value < 0) {
38              throwExceptionIfIdxInvalid();
39          }
40      }
41
42      private void checkIdx(final int idx) {
43          if (idx >= size) {
44              throwExceptionIfValueInvalid();
45          }
46      }
47
48      private void throwExceptionIfIdxInvalid() {
49          throw new ArrayIndexOutOfBoundsException("Index exceeds size");
50      }
51
52      private void throwExceptionIfValueInvalid() {
53          throw new IllegalArgumentException("Only positive values are supported");
54      }
55  }
```

Listing 2: TracingAspect.aj

```
1   package aspects;
2
3   import application.Main;
4   import org.slf4j.Logger;
5   import org.slf4j.LoggerFactory;
6
7   /**
8    * The aspect for tracing the chained method calls.
9    *
10   * @author Thomas Herzog <herzog.thomas81@gmail.com>
11   * @since 05/05/17
12   */
13  public aspect TracingAspect {
14
15      private static Logger log = LoggerFactory.getLogger(Main.LOGGER_NAME);
16
17      pointcut methodCall():
18              call(* application.*.*(..))
19                      && !within(application.Main);
20
21      pointcut fieldAccess():
22              (get(* application..*.*) || set(* application..*.*))
23                      && !within(application.Main);
24
25      pointcut newObject():
26              call(application.*.new(..));
27
28      before(): methodCall(){
29          log.info("Before method '{}#{}'",
    ↪   thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
30                  thisJoinPointStaticPart.getSignature().getName());
31      }
```

```
32
33      after() returning: methodCall(){
34          log.info("After method '{}#{}'",
 ↪   thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
35                  thisJoinPointStaticPart.getSignature().getName());
36      }
37
38      after() throwing(Throwable t): methodCall(){
39          log.info("After method '{}#{}' / {}#'{}'",
 ↪   thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
40                  thisJoinPointStaticPart.getSignature().getName(),
41                  t.getClass().getSimpleName(),
42                  t.getMessage());
43      }
44
45      before(): fieldAccess() {
46          log.info("Before field '{}#{}'",
 ↪   thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
47                  thisJoinPointStaticPart.getSignature().getName());
48      }
49
50      after(): fieldAccess() {
51          log.info("After field '{}#{}'",
 ↪   thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
52                  thisJoinPointStaticPart.getSignature().getName());
53      }
54
55      before():newObject() {
56          log.info("Before constructor '{}'",
 ↪   thisJoinPointStaticPart.getSignature().getDeclaringType());
57      }
58
59      after():newObject() {
60          log.info("After constructor '{}'",
 ↪   thisJoinPointStaticPart.getSignature().getDeclaringType());
61      }
62  }
```

Listing 3: IndentionLogTrace.aj

```
1   package aspects;
2
3   /**
4    * This class intercepts the log calls within the TRacingAspect for log message indention.
5    *
6    * @author Thomas Herzog <herzog.thomas81@gmail.com>
7    * @since 05/12/17
8    */
9   public aspect IndentionLogTrace {
10
11      private String currentIndent = "";
12      private static final String INDENT = "      ";
13      private static final int MAX_INDENT_IDX = INDENT.length() - 1;
14
15      pointcut logCall(String msg):
16              if(application.Main.logIndentionEnabled)
17                      && call(void org.slf4j.Logger.* (String, ..)) && !within(IndentionLogTrace)
18                      && args(msg, ..)
19                      && within(TracingAspect);
20
21      void around(String msg): logCall(msg){
22          if ((msg.startsWith("After method") || msg.startsWith("After constructor")) &&
 ↪   (currentIndent.length() >= MAX_INDENT_IDX)) {
```

```
            currentIndent = currentIndent.substring(MAX_INDENT_IDX, currentIndent.length() - 1);
        }
        proceed(currentIndent + msg);
        if (msg.startsWith("Before method") || msg.startsWith("Before constructor")) {
            currentIndent = INDENT + currentIndent;
        }
    }
}
```

Listing 4: Main.java

```
package application;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Main class for testing the implemented aspects.
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public class Main {

    public static final String LOGGER_NAME = "aspectj-tracing";
    private static final Logger log = LoggerFactory.getLogger(LOGGER_NAME);
    public static boolean logIndentionEnabled = false;

    public static void main(String args[]) {
        log.info("-------------------------------------------------");
        log.info("testIndentionDisabled()");
        log.info("-------------------------------------------------");
        testIndentionDisabled();
        log.info("-------------------------------------------------");
        log.info("");
        log.info("-------------------------------------------------");
        log.info("testIndentionEnabled()");
        log.info("-------------------------------------------------");
        testIndentionEnabled();
        log.info("-------------------------------------------------");
    }

    private static void testIndentionDisabled(){
        logIndentionEnabled = false;
        PositiveValueStore value = new PositiveValueStore(10);
        try {
            value.addPositiveValue(1);
            value.addPositiveValue(-1);
        } catch (Throwable e) {
            log.error("Error in Main occurred", e);
        }
    }

    private static void testIndentionEnabled(){
        logIndentionEnabled = true;
        PositiveValueStore value = new PositiveValueStore(10);
        try {
            value.addPositiveValue(1);
            value.addPositiveValue(-1);
        } catch (Throwable e) {
            log.error("Error in Main occurred", e);
        }
```

```
52      }
53  }
```

1.3 Tests

Folgender Abschnitt enthält die Tests der Aufgabenstellung in Form der generierten *logs*. Für den originalen *log* siehe Datei *tracing-logfile.txt*.

```
[main] INFO aspectj-tracing - ----------------------------------------------------
[main] INFO aspectj-tracing - testIndentionDisabled()
[main] INFO aspectj-tracing - ----------------------------------------------------
[main] INFO aspectj-tracing - Before constructor 'class application.PositiveValueStore'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After constructor 'class application.PositiveValueStore'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#getPositiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#getPositiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#throwExceptionIfIdxInvalid'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#throwExceptionIfIdxInvalid' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#checkValue' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#setPositiveValues' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] ERROR aspectj-tracing - Error in Main occurred
java.lang.ArrayIndexOutOfBoundsException: Index exceeds size
    at application.PositiveValueStore.throwExceptionIfIdxInvalid(PositiveValueStore.java:49)
    at application.PositiveValueStore.checkValue(PositiveValueStore.java:38)
    at application.PositiveValueStore.setPositiveValues(PositiveValueStore.java:27)
    at application.PositiveValueStore.addPositiveValue(PositiveValueStore.java:21)
    at application.Main.testIndentionDisabled(Main.java:37)
    at application.Main.main(Main.java:22)
[main] INFO aspectj-tracing - ----------------------------------------------------
```

Abbildung 1: Nicht eingerückter *log*

```
[main] INFO aspectj-tracing - Before constructor 'class application.PositiveValueStore'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After constructor 'class application.PositiveValueStore'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing -             Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -             After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#getPositiveValues'
[main] INFO aspectj-tracing -             Before field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing -             After field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#getPositiveValues'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing -             Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -             After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing -             Before method 'PositiveValueStore#throwExceptionIfIdxInvalid'
[main] INFO aspectj-tracing -             After method 'PositiveValueStore#throwExceptionIfIdxInvalid' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#checkValue' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#setPositiveValues' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] ERROR aspectj-tracing - Error in Main occurred
java.lang.ArrayIndexOutOfBoundsException: Index exceeds size
    at application.PositiveValueStore.throwExceptionIfIdxInvalid(PositiveValueStore.java:49)
    at application.PositiveValueStore.checkValue(PositiveValueStore.java:38)
    at application.PositiveValueStore.setPositiveValues(PositiveValueStore.java:27)
    at application.PositiveValueStore.addPositiveValue(PositiveValueStore.java:21)
    at application.Main.testIndentionEnabled(Main.java:48)
    at application.Main.main(Main.java:28)
[main] INFO aspectj-tracing - ----------------------------------------------------
```

Abbildung 2: Eingerückter *log*

2 Caching

Dieser Abschnitt beschäftigt sich mit der Dokumentation der Aufgabenstellung *Caching*.

2.1 Lösungsidee

Der Algorithmus für die Berechnung des Binominialkoeffizienten wird in der Klasse *BinomialCoefficient* implementiert.

Es wird ein abstrakter Aspekt *AbstractAspect* implementiert, der die *Pointcut firstCall*, *allCallsWithArgs* und *innerCalls* definiert, sowie *JoinPoint* für *firstCall*, die abstrakte Methoden aufrufen. Dies soll so strukturiert werden, da alle Aspekte auf den ersten Aufruf der Berechnungsmethode reagieren sollen, um ihre Zustände zu initialisieren und zurückzusetzen. Dazu werden zwei Methoden *beforeFristCall* und *afterFirstCall* zur Verfügung gestellt, die von den konkreten Aspekten überschrieben werden können. Die Methoden *beforeFristCall* und *afterFirstCall* sind mit leeren Methodenrumpf in der Klasse *AbstractAspect* implementiert.

Es wird der Aspekt *LogRecursiveCallsAspect* implementiert, der einen *JoinPoint* definiert, der nur ausgeführt wird wenn das *Logging* aktiviert wurde und die Bedingungen definiert im *PointCut innerCalls* erfüllt sind. Beim ersten Aufruf der Berechnungsmethode wird vor dem Aufruf der Methode der Zähler initialisiert und nach dem Aufruf das Resultat über den *Logger* in die *Logs* geschrieben und der Zähler wieder zurückgesetzt.

Es wird der Aspekt *BinomialCacheAspect* implementiert, der die Berechnungsergebnisse zwischenspeichert um sie bei einem erneuten Auftreten der Variablen *n, m* zurückliefert und so einen weiteren rekursiven Abstiegt verhindert. Es wird die Klasse *BinomMapKey* implementiert, die als Schlüssel in einer *java.util.HashMap* fungiert, die wiederum die berechneten Werte speichert. Das Zwischenspeichern wird nur dann durchgeführt, wenn *Main.CachingEnabled* auf den Wert *true* gesetzt ist.

Es wird der Aspekt *RuntimeMeasurementAspect* implementiert, der die Dauer der gesamten Berechnung misst und über den *Logger* ind *Logs* schreibt. Das Messen der Zeit wird nur dann durchgeführt wenn *Main.LoggingEnabled* auf den Wert *true* gesetzt ist.

2.2 Quelltexte

Folgender Abschnitt enthält die implementierten Klassen, Aspekte und das implementierte Testprogramm.

Listing 5: AbstractAspect.aj

```
1  package aspects;
2
3  /**
4   * This is the base class for providing advice for the first calls and defines all of the used
    ↪  point cut.
5   *
6   * @author Thomas Herzog <t.herzog@curecomp.com>
7   * @since 05/17/17
8   */
9  public abstract aspect AbstractAspect {
10
11     public pointcut firstCall():
12             call(long application.BinomialCoefficient.calculate(..))
13                     && !within(application.BinomialCoefficient)
14                     && !within(aspects.*);
15
```

```
    public pointcut allCallsWithArgs(int n,
                                     int m):
            call(long application.BinomialCoefficient.calculate(int,int))
                    && args(n,m)
                    &&!within(aspects.*);

    public pointcut innerCalls():call(long application.BinomialCoefficient.calculate(..))
            && within(application.BinomialCoefficient)
            && !within(aspects.*);

    before(): firstCall() {
        beforeFirstCall();
    }

    after(): firstCall() {
        afterFirstCall();
    }

    protected void beforeFirstCall() {
        // default does nothing
    }

    protected void afterFirstCall() {
        // default does nothing
    }
}
```

Listing 6: LogRecursiveCallsAspect.aj

```
package aspects;

import application.Main;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * This
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public aspect LogRecursiveCallsAspect extends AbstractAspect {

    private int callCount;

    private static final Logger log = LoggerFactory.getLogger(Main.LOGGER_NAME);

    @Override
    protected void beforeFirstCall() {
        if (Main.LoggingEnabled) {
            callCount = 0;
        }
    }

    @Override
    protected void afterFirstCall() {
        if (Main.LoggingEnabled) {
            log.info("Recursive calls: {}", callCount);
            callCount = 0;
        }
    }

```

```
    after(): if(application.Main.LoggingEnabled)
            && innerCalls() {
        callCount++;
    }
}
```

Listing 7: BinomialCacheAspect.aj

```
package aspects;

import application.Main;
import model.BinomMapKey;

import java.util.HashMap;
import java.util.Map;

/**
 * This is the caching aspect which caches the calculated value for n,m and returns the cached
 ↪  value if already calculated,
 * otherwise calculates it and caches it for the occurrence of n,m.
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public aspect BinomialCacheAspect extends AbstractAspect {

    private Map<BinomMapKey, Long> cache = new HashMap<>(500);

    @Override
    protected void beforeFirstCall() {
        if (Main.CachingEnabled) {
            cache = new HashMap<>(500);
        }
    }

    @Override
    protected void afterFirstCall() {
        if (Main.CachingEnabled) {
            cache = null;
        }
    }

    long around(int n,
                int m): if(application.Main.CachingEnabled) && allCallsWithArgs(n,m ) {
        final BinomMapKey key = new BinomMapKey(n, m);
        Long value;
        if ((value = cache.get(key)) == null) {
            value = proceed(n, m);
            // Will be null after last call, no need to cache anymore
            if (cache != null) {
                cache.put(key, value);
            }
        }

        return value;
    }
}
```

Listing 8: RuntimeMeasureAspect.aj

```java
package aspects;

import application.Main;
import org.apache.commons.lang3.time.StopWatch;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/12/17
 */
public aspect RuntimeMeasureAspect extends AbstractAspect {

    private static final Logger log = LoggerFactory.getLogger(Main.LOGGER_NAME);
    private static StopWatch watch;

    @Override
    protected void beforeFirstCall() {
        if (Main.RuntimeMeasurementEnabled) {
            watch = new StopWatch();
            watch.start();
        }
    }

    @Override
    protected void afterFirstCall() {
        if (Main.RuntimeMeasurementEnabled) {
            watch.stop();
            log.info("Calculation duration: millis={}", watch.getTime());
            watch = null;
        }
    }
}
```

Listing 9: BinomMapKey.java

```java
package model;

/**
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/12/17
 */
public class BinomMapKey {

    private final int n;
    private final int m;

    public BinomMapKey(int n,
                       int m) {
        this.n = n;
        this.m = m;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        BinomMapKey that = (BinomMapKey) o;

        if (n != that.n) return false;
```

```
26          return m == that.m;
27      }
28
29      @Override
30      public int hashCode() {
31          int result = n;
32          result = 31 * result + m;
33          return result;
34      }
35  }
```

Listing 10: BinomialCoefficient.java

```
1   package application;
2
3   /**
4    * This class calculates the binomial coefficient for n and m.
5    *
6    * @author Thomas Herzog <herzog.thomas81@gmail.com>
7    * @since 05/05/17
8    */
9   public class BinomialCoefficient {
10
11      public static long calculate(int n,
12                                   int m) {
13          return (m == 0 || m == n)
14                  ? 1L
15                  : (calculate(n - 1, m - 1) + calculate(n - 1, m));
16      }
17  }
```

Listing 11: Main.java

```
1   package application;
2
3   import org.slf4j.Logger;
4   import org.slf4j.LoggerFactory;
5
6   /**
7    * @author Thomas Herzog <herzog.thomas81@gmail.com>
8    * @since 05/05/17
9    */
10  public class Main {
11
12      public static boolean LoggingEnabled = false;
13      public static boolean CachingEnabled = false;
14      public static boolean RuntimeMeasurementEnabled = false;
15      public static final String LOGGER_NAME = "aspect-caching";
16
17      private static final Logger log = LoggerFactory.getLogger(LOGGER_NAME);
18
19      public static void main(String args[]) {
20          final int n = 45;
21          final int m = 10;
22          log.info("----------------------------------------------------------------");
23          log.info("testAllDisabled()");
24          log.info("----------------------------------------------------------------");
25          testAllDisabled(n, m);
26          log.info("----------------------------------------------------------------");
27          log.info("");
```

```
28          log.info("--------------------------------------------------------------------");
29          log.info("testRuntimeMeasurementEnabled()");
30          log.info("--------------------------------------------------------------------");
31          testRuntimeMeasurementEnabled(n, m);
32          log.info("--------------------------------------------------------------------");
33          log.info("");
34          log.info("--------------------------------------------------------------------");
35          log.info("");
36          log.info("testRuntimeMeasurementAndLoggingEnabled()");
37          log.info("--------------------------------------------------------------------");
38          testRuntimeMeasurementAndLoggingEnabled(n, m);
39          log.info("--------------------------------------------------------------------");
40          log.info("");
41          log.info("--------------------------------------------------------------------");
42          log.info("");
43          log.info("testAllEnabled()");
44          log.info("--------------------------------------------------------------------");
45          testAllEnabled(n, m);
46          log.info("--------------------------------------------------------------------");
47      }
48
49      private static void testAllDisabled(final int n,
50                                          final int m) {
51          LoggingEnabled = false;
52          CachingEnabled = false;
53          RuntimeMeasurementEnabled = false;
54
55          log.info("Starting: measurement={} / cachingEnabled={} / logRecursiveCallsEnabled={}",
    ↪  RuntimeMeasurementEnabled, CachingEnabled, LoggingEnabled);
56          log.info("          n={} / m={}", n,m);
57          log.info("BinomialCoefficient.calculate(45, 10): {}", BinomialCoefficient.calculate(n,
    ↪  m));
58      }
59
60      private static void testRuntimeMeasurementEnabled(final int n,
61                                                         final int m) {
62          LoggingEnabled = false;
63          CachingEnabled = false;
64          RuntimeMeasurementEnabled = true;
65
66          log.info("Starting: measurement={} / cachingEnabled={} / logRecursiveCallsEnabled={}",
    ↪  RuntimeMeasurementEnabled, CachingEnabled, LoggingEnabled);
67          log.info("          n={} / m={}", n,m);
68          log.info("BinomialCoefficient.calculate(45, 10): {}", BinomialCoefficient.calculate(n,
    ↪  m));
69      }
70
71      private static void testRuntimeMeasurementAndLoggingEnabled(final int n,
72                                                                   final int m) {
73          CachingEnabled = false;
74          LoggingEnabled = true;
75          RuntimeMeasurementEnabled = true;
76
77          log.info("Starting: measurement={} / cachingEnabled={} / logRecursiveCallsEnabled={}",
    ↪  RuntimeMeasurementEnabled, CachingEnabled, LoggingEnabled);
78          log.info("          n={} / m={}", n,m);
79          log.info("BinomialCoefficient.calculate(45, 10): {}", BinomialCoefficient.calculate(n,
    ↪  m));
80      }
81
82      private static void testAllEnabled(final int n,
83                                         final int m) {
84          LoggingEnabled = true;
```

```
85          CachingEnabled = true;
86          RuntimeMeasurementEnabled = true;
87
88          log.info("Starting: measurement={} / cachingEnabled={} / logRecursiveCallsEnabled={}",
    ↪   RuntimeMeasurementEnabled, CachingEnabled, LoggingEnabled);
89          log.info("          n={} / m={}", n,m);
90          log.info("BinomialCoefficient.calculate(45, 10): {}", BinomialCoefficient.calculate(n,
    ↪   m));
91      }
92  }
```

2.2.1 Tests

Folgender Abschnitt enthält die Tests der Aufgabenstellung in Form der generierten *logs*. Für den originalen *log* siehe Datei *caching-logfile.txt*.

Während der Tests hat sich gezeigt, das mit aktiviertem *Caching* sich das Laufzeitverhalten deutlich verbessert hat, da die rekursiven Aufrufe deutlich weniger geworden sind und daher auch die Anzahl der Berechnungen sich deutlich verringert hat.

```
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching - testAllDisabled()
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching - Starting: measurement=false / cachingEnabled=false / logRecursiveCallsEnabled=false
[main] INFO aspect-caching -           n=45 / m=10
[main] INFO aspect-caching - BinomialCoefficient.calculate(45, 10): 3190187286
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching -
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching - testRuntimeMeasurementEnabled()
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching - Starting: measurement=true / cachingEnabled=false / logRecursiveCallsEnabled=false
[main] INFO aspect-caching -           n=45 / m=10
[main] INFO aspect-caching - BinomialCoefficient.calculate(45, 10): 3190187286
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching -
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching -
[main] INFO aspect-caching - testRuntimeMeasurementAndLoggingEnabled()
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching - Starting: measurement=true / cachingEnabled=false / logRecursiveCallsEnabled=true
[main] INFO aspect-caching -           n=45 / m=10
[main] INFO aspect-caching - BinomialCoefficient.calculate(45, 10): 3190187286
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching -
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching -
[main] INFO aspect-caching - testAllEnabled()
[main] INFO aspect-caching - ----------------------------------------------------------------
[main] INFO aspect-caching - Starting: measurement=true / cachingEnabled=true / logRecursiveCallsEnabled=true
[main] INFO aspect-caching -           n=45 / m=10
[main] INFO aspect-caching - BinomialCoefficient.calculate(45, 10): 3190187286
[main] INFO aspect-caching - ----------------------------------------------------------------
```

Abbildung 3: *Caching* Test *Logs*

3 Asepct-Oriented TSP Solver

3.1 Lösungsidee

3.2 Quelltexte

3.3 Tests