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.

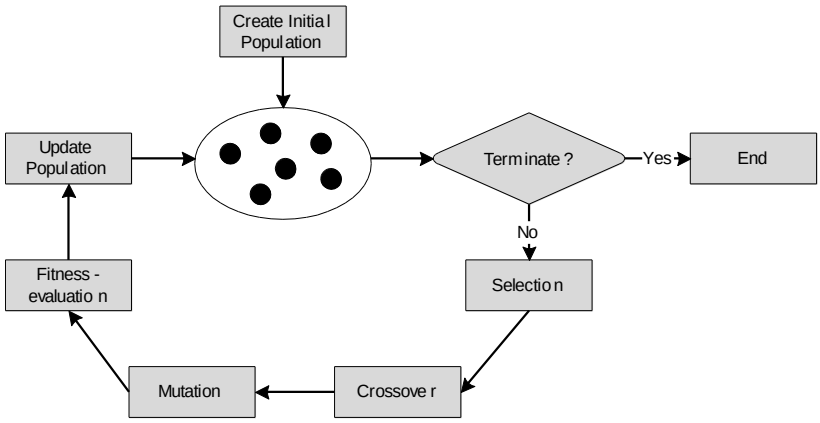
Furthermore, implement another aspect *BinomialCache* that caches calculated (intermediate) results and calls *Calculate* only if the required result is not known yet.

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.