Homework 1

1. Passes all automated tests.
2. The running time on the Moodle server is less than 20 ms.
3. There are references to all sources used (comments at the beginning of the program).

Homework 2

1. Passes all automated tests.
2. The insertion point is found by binary search.
3. The binary insertion sort method is significantly faster than the insertion sort method.
4. The tail of the sorted part of an array is shifted right using the System.arraycopy method - this makes your program much faster than in case you shift elements in loop.
5. At least three measurements of running times are done, average times are put in an Excel table that is used to build a graph.
6. There is only one graph (of type ’line graph’, amount of data on x-axis and running time on y-axis) showing the lines for all methods, format the y-axis to be logarithmic for better readability.
7. The graph is equipped with a legend - which line corresponds to which sorting method.
8. There are references to all sources used (if any).

Homework 3

1. Passes all automated tests.
2. The solution uses a list, not an array. There must be no additional variable to hold the number of elements.
3. Instance variables are described by the keyword ’private’, variable names start with a lowercase letter.
4. The methods pop, op, and tos check by themselves, that there are enough elements on the stack (for example, the method op requires two elements). If not enough, an exception must be thrown (this does not count that an exception occurs in the methods being called, throw a new exception with an appropriate error message).
5. The method op checks, among other things, the admissibility of the operation sign, in case of an error it reflects to the user an incorrect original input in the error message. The op method is completely independent of the interpret method, while the interpret can catch the exceptions thrown by the op (but does not have to, it depends on your solution).
6. All exceptions must have an appropriate and informative error message, that provides context and cause of the error in user terms.
7. The toString method should use a string buffer (e.g., StringBuilder) to collect the result.
8. The method equals must not depend on the method toString.
9. The error handling of the method interpret must detect 4 types of errors in the parameter expression pol:
    a) no expression - the value is null, an empty string or a string consisting of whitespace symbols only,
    b) the expression contains too many numbers - e.g., "5. 2. – 7.",
    c) the expression contains an illegal symbol - e.g., "67. xxx +",
    d) the expression does not contain enough numbers to perform certain operation - for example "3. 4. + - 5." (not enough numbers to perform ’-’).

Be sure to include the string pol that caused the error to an error message in its original form.

10. If any sources are used in the solution, they are referenced.

It is hard to believe that point 4 is appropriate, especially in the case of method op. Why does op have to double check what the pop has already checked, isn't it repetitive and redundant? The answer is "no" because the user of the method op does not need to know that this method calls inside the method pop, it is "kitchen side" and can be implemented differently. An error message, the source of which is pop, confuses such a user: what did he still do wrong that an error occurred somewhere far away from op?

As a general principle, error handling should be treated very carefully and response to the problem must be given as quickly as possible and with context (the real cause of the error). If you delegate error handling to a sub-method, this valuable information is lost - it doesn't help much to know that somewhere deep in the "bear's stomach" IndexOutOfBoundsException occurred for some reason, if the real cause of the error is a misuse of the method op, which can be checked. If you want to take advantage of the pop error handling, do a try-catch - catch the exception caused by the pop and throw a new and more accurate exception in the method op.

Repeating the code is usually not recommended, but in error handling it may help to better locate the error and report the reason to the user in a language that she understands.

Homework 4

1. If any sources are used in the solution, they are referenced. All automated tests pass.

2. The instance variables are defined by the keyword ’private’, the names of the variables and methods start with a lowercase letter.

3. All exceptions shall have an appropriate and informative error message. Error handling of the valueOf method also fully reflects the incorrect input string.

4. The fraction sign is attached to the numerator (the denominator is always positive), the fraction is reduced (cannot be 2/4, it must be 1/2 instead). The number 0 is 0/1 as a fraction.

5. When performing operations, a new object must be created for the result, the operands of the operation cannot be modified.

6. The equals method does not use the comparison of real number approximations of fractions, but the comparison of long integer cross-products.

7. The inverse and divideBy methods check by themselves division by zero and throw the exception with an appropriate error message if necessary.

8. The hashCode method depends on all instance variables. The hash code of equal fractions is the same, but not the other way around (the equality of fractions must not be inferred from the equality of hash codes). In addition, it should not be easy to guess when two fractions have the same hash code (for example, symmetric function is not appropriate because fractions a/b and b/a produce the same hash code).

9. The methods toString and valueOf must be compatible - valueOf correctly interprets all strings that toString outputs and rejects everything else (full error handling needed).

Homework 5

1. If any sources are used in the solution, they are correctly referenced. Do you understand exactly how the source code works? Are you able to recreate your code without any external help?

2. The names of variables and methods start with a lowercase letter, instance variables are private, methods that do not depend on objects are described as class methods (’static’). Your own additional methods are not ’public’.

3. The string buffer should be used to create parenthetic representations and the addition of strings with plus sign should be avoided.

4. Is the error handling of the input pol of the RPN parsing method exhaustive? Do the error messages reflect both the substantive cause of the error (not the "kitchen side") and the illegal input itself in its original form? Error handling must be as good as in homework 3.

Start early this time because the task is twice as labour-intensive as the previous one.

Homework 6

1. The program is based on a given template (classes GraphTask, Graph, Vertex, Arc, ...). Result of the program must be a computer processable object (list of arcs, list of vertices, graph with modified info fields, matrix, etc.) which could be processed further, not only a printout. For example, the path must be presented as a list of arcs (List <Arc>, not String).
2. If the problem requires an undirected graph, then each edge AB of it is represented by two opposite arcs AB and BA. If you need to find a path or a cycle then the result must be a list of arcs (List<Arc>, not string or list of vertices).
3. The program has been successfully tested with at least five examples, incl. example of a graph with 2000+ vertices (only measure the execution time). No junit tests required.
4. The program code is documented using Javadoc.
5. The program has been submitted via the Moodle program link and has received positive feedback.
6. The report shall comply with all formal requirements. Upload your report as a single pdf file. Report must be fully self-explanatory and contain:
   - Description of the problem (background, all definitions, use cases, drawings, etc.)
   - Description of the solution (algorithm)
   - User guidelines (API, what methods and what parameters to use to obtain the result)
   - Testing plan (at least 5 tests compulsory, also a time test for a graph with 2000+ vertices)
   - Used literature
   - Appendix 1. Full text of the solution (code)
   - Appendix 2. Test results, e.g., screenshots (use white background for all screenshots). Drawings are most helpful to understand the results - provide at least two drawings.
7. The report is comprehensive enough and contains all the information for a non-dedicated reader (such as your math teacher). Approximate volume 20 - 30 pages. Provide all the necessary definitions together with the task description.
8. The text of the program is included in the report (as an appendix) and the printout must not have a dark background.

9. The examples have screenshots (light background) and at least two examples also have explanatory drawings to convince the reader of the correctness of the result.
10. The report is presented as a single pdf file via the Moodle report link.

In an IntelliJ environment, check that the project language level is 8, otherwise visibility problems will occur between inner classes.

Homework 7

1. Passes all automated tests in Moodle.
2. In addition to the automated tests, you should go through the long-running tests that are currently available in the test file as lines commented out. They might take a lot of time and should therefore be executed locally on your computer.
3. Puzzle.main(new String [] {"ABCDEFGHIJAB", "ABCDEFGHIJA", "ACEHJBDFGIAC"}); must give two solutions.
4. Puzzle.main(new String [] {"CBEHEIDGEI", "CBEHEIDGEI", "BBBBBBBBBB"}); must not give any solutions.
5. All sources are referenced.