Object-Oriented Programming with Java - Advanced Course

Project Description

Robin Müller-Bady

Winter 2019/2020

1 Introduction

The examination of this module is in form of a group project. The resulting program has to be fully functional and documented. Each team should come together and work autonomously and is responsible for distributing tasks fair and equal among all team members. The task must be solved by the team. Help of external people or any fraudulent activities, e.g. copying source code from different sources without a reference will result in, at least, disqualification or legal consequences. The amount of own work must significantly exceed the amount of code copied from various sources even if referenced. This does not apply for the use of external libraries for extending own program features in a reasonable ratio of own code vs. external code. Any suspicious activity of that kind will be reported to the faculty examination board for further investigation. All further conditions of the examination regulation apply.

The following constraints apply to the module examination:

Start Date December 9th 2019, 00:01 AM

Submission Deadline February 9th 2020, 11:59 PM via Moodle platform

Team size 3-4 team members

Topic The presented project from Section 3

Submission Details The following things are required for a successful submission

1. A working program as runnable *.jar file including all necessary libraries and the source code (be careful with the discussed jar-in-jar problem) and the (fully functional and configured) zipped project (e.g. eclipse or netbeans) including all necessary files e.g. project files for importing. The source code must be commented as mentioned in the clean code lecture. Make sure that your program runs at least under Linux with JDK 11. Note that the machines in the lab have an installed JDK 12 version.

Please notice that a dysfunctional program, i.e., a program that does not start or execute properly under the given requirements, will be marked 0 points!

2. Documentation as separate document (PDF file) including technical details of the program, user documentation and presentation of the individual team member performance. A milestone document has to be submitted every 3 weeks (after week 3, after week 6). Preferably, the milestone document is one developing document over the whole project run time which is finally submitted as project documentation.
3. The declaration of authorship (German: "Ehrenwörtliche Erklärung") must be signed by each team member and added to the submitted documentation. You find an example text below.
4. Submit the software and required documents as archive in *.zip/*.tar.gz format via the moodle platform in the specific section (will be announced in the exercises). The archive must contain the runnable *.jar file, the full source code, the documentation, the milestone documents, and the signed/scanned declaration of authorship.

Presentation After the submission each individual (person) has to present the team solution and the participation to it in a group presentation. The presentation should take around 15 to 20 minutes per group.

Milestone presentations All groups have to give a short presentation (around 10 minutes) about each individual milestone during the exercises. Details will be announced during the semester.

Hints It is also possible to implement only parts of the given requirements. In case you do that, please make sure that your program is running even in that case. If parts of the specified program are missing, points will be deducted.

It is necessary to successfully submit every required item by the closing date. In addition, a successful participation in the final presentation is necessary.

Declaration of Authorship

Example for a declaration of authorship ("Ehrenwörtliche Erklärung"):

I hereby declare that the submitted project is my own unaided work or the unaided work of our team. All direct or indirect sources used are acknowledged as references.

I am aware that the project in digital form can be examined for the use of unauthorized aid and in order to determine whether the project as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future projects submitted. Further rights of reproduction and usage, however, are not granted here.

This work was not previously presented to another examination board and has not been published.

First and last name	City, date and signature

2 Grading Criteria

The grading of the projects is based on the given lectures and the individual autonomous learning of the topic. The module is passed with a total of 50 points (50%). See Table 1 for details of the grading structure. For a detailed view of the necessary requirements see Table 2.

Table 1: Points vs. Grades

Grade	5,0	4,0	3,7	3,3	3,0	2,7	2,3	2,0	1,7	1,3	1,0
Points	$<$50	50	55	60	65	70	75	80	85	90	95^{+}

Table 2: Requirements

Feature	**Points**	**Description**
Runnable Program	30	The running program is covering the described basic functionalities as described in Section 3.
Milestones	20	Milestones are submitted as required. The requirements of the milestones are specified in Section 3.
Object-Orientation	5	The program was developed using an adequate object orientation. At least one abstract method and one interface must be developed and used.
Collections	5	Information will be stored in suitable Java Collections or appropriate alternative data structures.
Error Handling	5	The program contains adequate error handling. At least one own exception (extending any type of Java Exception/Throwable) must be present.
Streams and Files	5	The program works with streams and files. At least one reading and writing file access has to be made.
Threads	5	The program contains at least two threads (from which one is “main”).
Clean Code	20	The program is developed using the clean code standard(s) as presented in the lecture.
Documentation	5	The documentation inside and outside the code is present as described in Section 3. Appropriate logging is provided using java.util.Logging or comparable.
Additional Features	5	The program can contain additional features to gather extra points. Additional features are to be discussed with the lecturer in advance in order to be graded.
Total	**105**	

3 Project - Communication Network Analysis

3.1 Programming Task

The task is to develop a tool for the basic analysis of graph-based communication network models. Nowadays, communication networks grow in importance and maintenance becomes an increasingly challenging task. In order to provide an overview over the network infrastructure for, e.g., network administrators or CEOs, it is helpful to aggregate and display information appropriately. In this project, a basic analysis of a given network model has to be implemented, from the input of a model over the appropriate processing of the model data to the output of the information. The final Java program works in 3 main steps:

1. Reading communication network information from a network model file (multiple test files of different sizes are provided) into appropriate Java data structures.
2. Processing the network model data and acquisition of the necessary information in accordance with the requirements defined.
3. Output the results of the processing into the specified file(s) and/or the user interface (command line interface (CLI) or graphical user interface (GUI)).

The basic functionalities that have to be covered are as follows:

- It is possible to start the program from the CLI or GUI and provide an input file for processing using the following or similar interface:

```
$> java myJavaProgram inputfile.graphml
```

- The input file format is *.graphml[1] (an XML-based format[2]). Example files of different sizes are provided in the moodle course. The files contain the node ID, described by node key *v_id*, the egde ID, described by the edge key *e_id*, and the edge weighting, described by the edge key *e_weight*.
- Information of the *.graphml file is transferred into appropriate data structures, i.e., class structures with respect to clean OOP-principles and the information required for the processing.
- From the created data structures, it is possible to output the following *graph* properties of the provided network graph on the system output stream:
  1. **Number of nodes**
  2. **Number of edges**
  3. The **vertex names** or **IDs**
  4. The **edge names** or **IDs**

[1] http://graphml.graphdrawing.org/
[2] https://www.w3schools.com/xml/xml_whatis.asp

5. **Connectivity**, i.e., whether the the graph is connected
6. **Diameter** of the graph, i.e., the maximum distance between any two nodes considering the shortest path between them

- From the created data structures, it is possible to output the following *node* and *edge* properties of the provided network graph on the system output stream:
  1. The **shortest path** between two vertices according to the Dijkstra algorithm[3], where both vertices can be provided on the command line using the following or similar interface:

```
$> java myJavaProgram inputfile.graphml -s 1 16
```

     where the *-s* switch starts the shortest path calculation of the program with *1* and *16* being the IDs of the vertices as parameters for the switch.

  2. Let $G = (V, E)$ be a network graph model with $v \in V$ being the nodes and $\{v_i, v_j\} \in E$ being the edges of the graph. Calculate the **betweenness centrality measure** for a selected node, where this measure is defined as:

$$g(v_x) = \sum_{v_i \neq v_x \neq v_j} \frac{\sigma_{v_i v_j}(v_x)}{\sigma_{v_i v_j}} \tag{1}$$

     with $\sigma_{v_i v_j}$ being the total number of shortest paths from node v_i to v_j and $\sigma_{v_i v_j}(v_x)$ being the number of shortest paths that pass through node v_x. This measure indicates how central a specific node is for all existing shortest path between all nodes in a given graph. The syntax of the betweenness calculation may be as follows using a CLI:

```
$> java myJavaProgram inputfile.graphml -b 1
```

     where the *-b* switch starts the betweenness calculation of the program and *1* is the ID of the selected node as parameter for the switch.

- Calculate all above graph and node/edge properties and output them into a new *.graphml file using the following or similar interface:

```
$> java myJavaProgram inputfile.graphml -a outputfile.graphml
```

  where the *-a* switch starts the calculation of all above mentioned properties and has a filename (*outputfile.graphml*) as parameter where to place the output. In case one or multiple properties require a parameter, e.g., a specific node, the calculation has to be applied to all possible input values. As an example, betweenness has to be calculated for all possible nodes in the graph and stored in each node of the output *.graphml file (the selection of the key name is your choice but must be documented), while shortest paths have to be calculated from each node to any other node in the graph. In case the properties are already present in the input file, existing information is overwritten and replaced with the new calculations.

[3] https://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_en.html

3.2 Documentation

The requirements for the documentation are

- JavaDoc is applied for each public method inside a class except getter and setter methods, unless they do something else than getting/setting an attribute of a class which requires documentation.
- Other source code documentation/comments are applied as necessary.
- An appropriate logging of information is provided using the java.util.Logging or a comparable library. Logging must at least be applied to each exceptional program flow, e.g., when throwing exceptions or handling error code but it is recommended to also log debug information, e.g., when handling data or reading in files etc. Logging should replace calls to "System.out." completely.
- Documentation outside the code is submitted as PDF file containing user- and technical documentation having adequate descriptions, graphics, diagrams etc. Following elements must be present:
  - User handbook (how to use the program)
  - Technical description
  - UML diagram(s)
  - Work distribution among team members preferably as table
  - Milestone documents
- Milestone documents containing the current state of the project and the participation of the team members. It has to be submitted separately every 3 weeks, i.e., after the 3rd and the 6th week, via elearning as one document in PDF format. Submitting the milestones is mandatory. The submission deadline the milestones for each time frame is Sundays at 11:59 PM (23:59) via the moodle course. Deadlines are also shown in the moodle course. Submission is closed after the deadline and no further submission is possible.