# Assessment A1: product

This assessment is your **individual assessment** of the final version of the software built by the group.

Fill in this checklist **on your own**, not with multiple group members together.

## identification

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| name | registration number | group (one letter) and name of project | | date |
|  |  |  |  |  |

Fill in this checklist for the completed software. This is the final version of software on the assigned group repository on git.wur.nl at the end of the course. If that repository contains multiple branches, use only branch **main** for this checklist.

Below, in the cells of the first column, use +, ±, – for scoring (don’t leave blank!).

Where the text of an item indicates “all”, “each”, or “none”, you can still specify “+” if you find an occasional exception (which you still have to specify). See the guide for these checklists (separate document) for further instructions – e.g., about “explain ... why”.

Red = still needs to be done, urgently

Yellow = we have to check it at the end of the project

Green = it’s in the script

## interesting enough from a technical perspective

|  |  |  |
| --- | --- | --- |
| score | description | specification / explanation |
| [ ] | The software applies inheritance of own classes.  Specify subclass and superclass of one example (or explain why not). |  |
| [ ] | The software applies polymorphism.  Specify defining superclass/interface, implementing (sub)class, and polymorphic method of one example (or explain why not). |  |
| [ ] | The software uses functionality from one or more non-standard libraries (i.e. which had to be added to the class path).  Specify class name and jar-file of one example (or explain why not). |  |
| [ ] | Programmer-defined objects serve as parameters of methods.  Specify class, method, and parameter of one example (or explain why not). |  |
| [ ] | Programmer-defined objects serve as result type of methods.  Specify class and method of one example (or explain why not). |  |
| [ ] | The software creates objects dynamically.  Specify class, method and line number of one example (or explain why not). |  |

## functionalities implemented

|  |  |  |
| --- | --- | --- |
| score | description | specification / explanation |
| [ ] | The program does something observable when started. [Crashing and system error messages do not count.]  Specify the main class to start the program. |  |
| [ ] | Outcomes of the program are not *obviously* wrong. |  |
| [ ] | The program gives an appropriate error message when something is wrong; give an example. |  |
| [ ] | All unit tests pass  (or explain which ones don’t and why). |  |
| [ ] | All acceptance tests pass  (or explain which ones don’t and why). |  |
| [ ] | The unit tests cover all relevant functionalities  (or explain what is missing and why). |  |
| [ ] | Unit tests cover some error situations as well.  Specify which unit tests (or explain why not). |  |
| [ ] | Acceptance tests cover non-exceptional functionalities at least. |  |

## overall design

|  |  |  |
| --- | --- | --- |
| score | description | specification / explanation |
| [ ] | The state of the system is stored in appropriate classes. |  |
| [ ] | State is stored only once, i.e. no duplication  (or indicate where and why). |  |
| [ ] | Functionalities (responsibilities) are assigned to appropriate classes. |  |
| [ ] | The class model is navigable from the main class. |  |
| [ ] | The class model contains no cycles. |  |
| [ ] | The way the system uses inheritance – or does not use it – is appropriate (“is-a” vs “has-a”). |  |

## class design

|  |  |  |
| --- | --- | --- |
| score | description | specification / explanation |
| [ ] | All methods are either accessor or mutator  (or indicate which are not and why). |  |
| [ ] | The program contains no static methods (other than main) and no static fields (other than constants) [except in case of a singleton pattern]. |  |
| [ ] | The program does not contain duplicate code. |  |
| [ ] | The program contains private auxiliary methods  (or explain why they are not needed). |  |
| [ ] | The program applies proper encapsulation. |  |

## coding

|  |  |  |
| --- | --- | --- |
| score | description | specification / explanation |
| [ ] | The program contains Javadoc comments for at least classes, interfaces, and public methods. |  |
| [ ] | All code adheres to one standard layout. |  |
| [ ] | All code adheres to the standard Java naming conventions. |  |
| [ ] | The program introduces symbolic constants for fixed numbers and strings. |  |
| [ ] | Variables are local when they can. |  |
| [ ] | The program uses while- and for-loops appropriately. |  |
| [ ] | The program prevents exceptions where reasonably possible; and handles them if prevention is impossible or costly. |  |