Project Descriptions

This chapter describes the projects that you have to carry out in the course of the module Software Systems. The Design Project is due in week 4, the Programming Project is due in week 10.

Design

Make a design for a medium-sized system. It is expected that you do this project assignment in teams of four students.

In this project you make a design by means of UML diagrams for a software system for parking houses. There are multiple business processes to be analysed, a number of use cases to be identified and described, and the project entails a nontrivial data model. Most of the information needed is described in the case history below. However, some additional information information is to be acquired by means of an interview (in week 2).

During the first three weeks you have one afternoon to work on the project, in week 4 you have two more full days to complete it.

What to hand in

The final project deliverable should contain the following items

- A brief report, describing the contents of further documents and, if applicable, anything you want to communicate (e.g. why you have made particular design choices). In an appendix you should mention who the team members are and what everybody's contribution to the final deliverable is.
- A report about the interview conducted in week 2. It should briefly describe the facts (whom you interviewed, when, who conducted the interview, etc.) and give a complete account of the relevant information that you extracted from the interview.
- Appropriate[1] activity diagrams describing relevant business processes (at least 2), following the style of lecture 1 / recommended exercise 1.
- A glossary.
- A complete requirements list (with requirements and use cases as in Bennett et al. p. 170).
- Use case diagrams for the complete system.
- An actor list.
- Complete (brief) use case descriptions (as in Bennett et al. pp. 171–174).
- Extended use case descriptions of representative use cases (at least 4) (as in Bennett et al. p. 176).
- A complete analysis class diagram (as in Bennett et al. p. 231).
- Appropriate state machine diagrams (at least 2), following the style of lecture 6 / recommended exercise 6.

[1] Several items in this list of deliverables mention that you should make *appropriate* choices. What is appropriate? In this context it means that you should choose processes/objects worth modelling, because they are not entirely trivial, and having the model on paper helps to understand exactly what the system should do. In other words, as a design effort it makes sense to model them.
An example of what *not* to do: For two state machine diagrams one could model the barrier at the entrance of the car park (having two states: *up* and *down*), and the barrier at the exist of the car park (*idem*). This would not be regarded as a meaningful contribution to the design.

- Appropriate sequence diagrams of relevant system processes (at least 2).

You may submit the report in English or Dutch.

Case Description: Barchester City Council Parking System

Barchester City Council operates seven car parks in the centre of Barchester[2]. The Council has a requirement for a new system to control its car parks. This system must provide for the day-to-day operation of each car park—issuing tickets, handling payment and controlling barriers—and the management of car parks—recording problems, issuing season tickets and monitoring service level agreements with the security company that guards the car parks.

The car park operational system controls entry to and exit from a car park and payment for car parking.

There are two types of users: ordinary customers, who pay for their use of each car park at the time they use it, and season ticket holders, who pay a fixed amount in advance for parking for three, six or twelve months in a specific car park. Season ticket holders are allocated parking spaces in designated areas that are not available to ordinary customers from Monday to Friday. Season tickets are for weekdays only; the designated spaces are available to all customers at week-ends. No more than 10% of the spaces in a car park are allocated to season ticket holders.

Entry to the car park

When a car approaches an entry barrier, its presence is detected by a sensor under the road surface, and a ‘Press Button’ display is flashed on the control pillar

The ordinary customer must press a button on the control pillar, and a ticket is printed and issued. The ticket must be printed within five seconds. A ‘Take Ticket’ display is flashed on the control pillar. If the car park is full, no ticket is issued, and a ‘Full’ display is flashed on the control pillar. If a vehicle leaves the car park, then the ‘Press Button’ display is activated again when there is a vehicle waiting.

When the customer pulls the ticket from the control pillar, the barrier is raised.

The season ticket holder does not press the button, but inserts his or her season ticket into a slot on the control pillar. A check is made that the season ticket is valid for this car park and has not expired, that it is a weekday and that the season ticket holder is not recorded as having already entered this car park and not left. If all these checks are passed, then the barrier is raised. The checks must take no longer than five seconds. A record is made of the time of entry for that season ticket holder.

A sensor on the other side of the barrier detects when the car has passed and the barrier is lowered.

The ticket issued to each ordinary customer has a bar code on it. The bar code has a number on it and the date (ddmmyyyy) and time (hhmmss) of entry to the car park. The number, date and time of entry are also printed on the ticket in human readable form.

The details of the ticket are stored: ticket no., issue date, issue time, issuing machine.

The number of vehicles in the car park is incremented by 1 and a check is made against the capacity of the car park. If the car park is full, then a display near the entrance is switched on to say ‘Car Park Full’, and no further tickets are issued until a vehicle leaves the car park.

Payment

When the ordinary customer is ready to leave, he or she must go to a pay station to pay. The ticket is inserted into a slot, and the bar code is read. The ticket bar code information is compared with the stored information. If the dates or times are not the same, the ticket is ejected, and the customer is told (via an LCD display) to go to the office. In the office, the attendant has a bar code reader and can check a ticket. Typically the problem is damage to the bar code on the ticket, and the attendant can use the office system to calculate the charge, take payment and validate the ticket (see below).

At the pay station, if the ticket dates and times are the same as the bar code dates and times, then the current date and time are obtained, and the duration of the stay in the car park is calculated. From this the car park charge is calculated and displayed on the LCD display. Calculation and display of the charge must take no more than two seconds.

[2]This case description was written by Bennett, McRobb, and Farmer, as additional case study material to their book. The case description in this manual has some minor modifications compared to the original version.

There are two tariffs: a short-stay tariff and a long-stay tariff. These include the rates for weekdays from 8.00 am to 6.00 pm, and lower rates for entry after 6.00 pm and at week-ends.

If no change is available, this information is displayed on the LCD display. The customer must then insert notes or coins to at least the amount of the charge. Each note or coin is identified as it is inserted and the value added to an accumulated amount and displayed on the LCD display. Invalid notes are ejected from the note slot. Invalid coins are dropped through into the return tray. A message is displayed on the LCD display.

As soon as the amount accumulated exceeds the charge, the ticket is validated. The current date and time are added to the stored data for that ticket (payment date, payment time).

If the amount entered exceeds the charge and change is available, then the amount of change is calculated and that amount of change is released into the return tray. Otherwise, no change is given. In either case, a message is displayed on the LCD display.

The ticket has the payment date and time printed on it and is ejected from the ticket slot.

A message is displayed telling the customer to press the ‘Receipt’ button if they need a receipt. If they press this button, a receipt is printed and ejected into the receipt tray. The receipt shows the Council address, address of the car park, VAT number, date and amount paid.

A message is displayed for the customer telling them to take the ticket back to their car and leave the car park within 15 minutes.

Leaving the car park

When the customer drives up to the exit barrier, the car is detected by a sensor, and an ‘Insert Ticket’ display is flashed on the control pillar. The customer must insert the ticket. The bar code is read and a check is made that no more than 15 minutes have elapsed since the payment time for that ticket. If more than 15 minutes have elapsed, an intercom in the control pillar is activated and connected to the attendant in the car park office. The customer can talk to the attendant, and the attendant can view the details of the ticket on his or her computer. The attendant can activate the barrier remotely, for example if there is a queue to get out and the customer is likely to have been reasonably delayed.

If no more than 15 minutes have elapsed, the barrier is raised. A sensor on the other side of the barrier detects when the car has passed and the barrier is lowered.

The number of vehicles in the car park is decremented by 1 and a check is made against the capacity of the car park. If the car park was full, then the display near the entrance is switched to say ‘Spaces’, and a check is made to see if any vehicles are waiting. If they are, then the control pillar for the first waiting vehicle is notified. If the driver of the vehicle waiting there does not press the button (for example, because they have backed out and left), then the control pillar for the next waiting vehicle is notified.

At any time, the attendant can view the status of a pay station or a barrier control pillar. Once a connection is made, the status is updated every 10 seconds.

Season ticket holders do not have to go to the pay station, when they are ready to leave the car park, they go to the exit and insert their season ticket into a slot on the exit barrier control pillar. The barrier is raised and a record is made of the time at which the season ticket holder left.

Security visit recording

The City Council has a contract with security companies to visit the car parks at regular intervals. The contract specifies the number of visits per day to each car park and the minimum duration of each visit. Each car park has an office to which the security guards have access. In the office is a card reader similar to the one used for reading season tickets in the control pillars. When a security guard arrives in a car park, he or she puts a card into the card reader and the date and time of arrival is recorded. When the security guard leaves, he or she puts the card in again, and the departure time is recorded. (This card also allows security guards to enter and leave the car park in the same way as season ticket holders. However, this is not used to record the arrival and departure of security guards, as they may not be able to enter with a vehicle if there is a queue of cars at the barrier.)

Currently, the City Council uses two security companies, but could use more or only one in the future. Each security company is issued with a specific number of cards, depending on the number of car parks they are responsible for. Each security company is responsible for specific car parks.

Management functionality

All requirements for operational use have been described above. The City Council would like to obtain further management information from the system. It is not included in this document because the requirements were not finalized at the date of writing. The responsible person from the City Council has invited the IT company to visit him to discuss this.

Submission and grading

Packaging and submission

- Package all the files you want to submit in a single zip file. Please include the group number in the name of the zip file.
  The main document should be recognizable by its file name.
  Somewhere near the top the main document should contain a table of submitted files and their contents.

- Submit it by means of the Blackboard assignment.

Important Dates

~~Thu~~ Sun 23:59 CET, week 4	Submission deadline for maximal project grade
Fri 23:59 CET, week 10	Submission deadline for maximum 6.0 grade

Grading

If your submission satisfies what is requested (i.e. it contains all the items mentioned under "what to hand in") and your models are roughly OK (not necessarily perfect) you can expect at least a 6.0. In determining the grade, the following quality criteria will be taken into account.

Good quality, increasing your grade (from more important to less important):

- Completeness – you didn't overlook any requirements.
- Appropriate explanations – if you have made particular design choices, can you tell us what and why.
- Appropriate use of specification techniques (like inclusions and extensions in use case diagrams, generalization in class diagrams). Use them where they help, but be aware that too much structure does not make diagrams easier to understand.
- Appropriate choice of elements to specify (activity diagrams, extended case descriptions, sequence diagrams, state diagrams). There is no point in adding them if they are trivial.
- Appropriate choice of names for classes, use cases, etc.

Bad quality, decreasing your grade (from more important to less important):

- Missing stuff – e.g. diagrams or tables that were asked for are not included.
- Requirements are ignored or misrepresented in the design
- Errors in the use of UML – a missing arrow head will not cost you points; systematically forgetting cardinalities in a class diagram will.
- Unclear explanations and badly structured text.
- The deliverable is not packaged as described above.

Not part of the grading criteria:

- Choice of language.
- Grammatical correctness (as long as it does not degrade the readability of the report).
- Graphic quality of the diagram (as long as they are clearly readable).
- Lay-out of the documents.

In the unfortunate case that your group has to resubmit an improved version, you will receive a list with specific requests that have to be satisfied to acquire a 6.

Programming

Develop a distributed client/server program to play a board game, and describe the design in a report. It is expected that you make this project assignment in pairs.

This section describes the requirements for the programming project. Purpose of the project is to demonstrate the programming and design skills you have acquired during this module. You will demonstrate this by developing a client/server application to play a board game.

The application should at least provide the following functionality:

- a game server which offers the possibility to play the game;
- enforcement of the game rules;
- leaderboard functionality to maintain a high score list for the game;
- support to play the game over the network;
- support for 2–4 players per game;
- a graphical user interface;
- means to securely authenticate clients to servers; and
- support for computer players.

During week 5, 7 and 8, there are special sessions dedicated to the project. Participation in these sessions is *mandatory*. During the session of week 5, you will discuss the global design of the application; during the session of week 7 you will design the communication protocol; and during the session of week 8 you will design the authentication protocol for better security. More information about the purpose of these sessions is given in the chapters about the different weeks.

The last practical programming assignments of weeks 5, 6, 7 and 8 will ask you to start developing individual components of the project. Make sure that you indeed use your time to do this; if you wait until the last two weeks of the module before starting on the development of the game, you will probably not have enough time. Most of these week, during the Friday afternoon a peer feedback session is planned where you can discuss your solution with other members of your tutorial group.

Below, we will describe the following:

- the rules of the board game to be implemented;
- a global description of the game application;
- the functional requirements to the application, as they should be implemented;
- guidelines for the report;
- the minimal requirements for
  - the implementation; and
  - the report;
- instructions about
  - planning;
  - the peer review process; and
  - delivery of the code and the report;
- possible extensions to the application;
- an overview of the grading criteria for this project.

Rules of the Board Game

The rules of the board game to be implemented are available on Blackboard.

Global Description of the Game Application

After starting a client GUI, a user of the client can enter the IP address and port number of the server, and his own name. After entering this information, the client will log on to the server. The client will be waiting for the server to signal that there is another client logged on to the server. When a second client has logged on, the game can start, or the clients can decide to wait for more players, up to a maximum of four. When the game is ready to be started, the server can assign arbitrary colours to the players, or the players can choose their colour in order of arrival. After the game has started, the server should remain waiting for incoming requests from new clients that would like to play the game. Thus, it should be possible to play several games simultaneously on the same server.

The game itself could proceed as follows. The player whose turn it is, enters a move, taking into account the rules of the game. The client checks the move, and when it is legal, it sends it to the server.

The server also checks legality of the move, and if it is, it will send this information to all clients who can then update their internal game state. The turn then moves to the next player, who should also enter a legal move. This procedure proceeds until the rules of the game indicate the game has finished.

Communication Protocol

Your client application should be able to communicate with server applications from other students within your tutorial group, and vice versa. As a consequence, all students within the same tutorial group should use the same *protocol* for client/server communication. The protocol describes which data will be exchanged between the client and the server, and in which order. This data will consist of the moves in the game, and inspection requests for the leader board.

The protocol will be determined during a tutorial group meeting in week 7. More information about how the protocol can be defined is given in Section 7.4.2.
Remark
It is possible that your tutorial group decides on a different protocol than the one discussed above, with a different distribution of responsibilities between client and server.

Leaderboard

Additionally, the server maintains a leaderboard, providing scoring information about all the games played on the server. It should be possible to retrieve this information in several different ways (*e.g.*, overall high score, best score of the day, best score of a particular player).

Security

To prevent players from cheating the server provides a way for clients to securely authenticate themselves (using a PKI-based approach) to the server and vice-versa.

Functional Requirements of the Application

Your application should implement a collection of requirements.

For the *server*, the following requirements should be implemented

1. When the server is started, a port number should be entered that the server will listen to. When the server has a GUI, it is allowed to enter the port number via the GUI.
2. If the port number already is in use, an appropriate error message is returned, and a new port number can be entered.
3. A server should be able to support multiple instances of the game that are played simultaneously by different clients.
4. If the server has a GUI, all communication message should be shown in a `(J)TextArea`. Additionally, the GUI should be scalable. If the server has a command line interface, all communication messages should be written to `System.out`.
5. The server should respect the protocol as defined for the tutorial group during the project session in weeks 7 and 8, *i.e.*, the server should be able to communicate with all other clients from the tutorial group.
6. The server maintains a leaderboard, providing different views of the scores obtained so far. The leaderboard should at least provide: the overall high score; the best scores of the day; and the best scores of a particular player.
7. The server uses the services of the provided Authentication Server to allow clients to authenticate themselves.

For the *client*, the following requirements should be implemented

1. The client should have a scalable GUI, which provides several options to the user (*e.g.*, possibility to a enter port number and IP address) to request a game at the server.
2. The client should support human players, and computer players with (some) artificial intelligent behaviour.
3. The thinking time of the computer player (and thus the power of the artificial intelligence) should be a parameter that can be changed via the client GUI.
4. The client provides a *hint* functionality. This shows a human player a possible move, as indicated by the computer player. The move may only be proposed, the human player should have the possibility to decide whether to play this move, or make a different one.
5. After the game is finished, the player should be able to start a new game.
6. If a player quits the game before it has finished, closes the GUI, or the client crashes, the other players should be informed. In this case, the other players should be allowed to register again with the server to play the game.
7. A server might at all times disconnect. The clients should react to this in a decent way, closing all open connections *etc.*
8. The client should respect the protocol as defined for the tutorial group during the project session in weeks 7 and 8, *i.e.*, the client should be able to communicate with all other servers from the tutorial group.
9. The client GUI should provide functionality to inspect the leaderboard information stored on the server.
10. The client authenticates itself to the server using the services of the provided Authentication Server.

Finally, as a *global requirement*, the client and the server should always be in the same game state.

Warning
It might be possible to find an implementation of the game on the Internet. Copying such an implementation will be considered as fraud, and will be communicated to the exam committee.

Therefore, at any moment in time, you should be able to explain your own solution to the lecturers.

Guidelines for the Report

The report may be written in English or Dutch.

Discussion of the Overall Design
The design of the application describes the overall structure of the system, in terms of the classes and their relationships. It should describe the following:

1. Class diagrams, with an explanation, for example per package and globally. Make sure your layout reflects the structure of your application, *e.g.*, by repositioning classes, and removing unnecessary details. Feel free to add extra labels and notes.
2. A systematic overview of which part of the requirements is implemented by which classes. If you did not manage to implement all requirements, this should be indicated here.
3. The use of the *Observer* and *Model-View-Controller* patterns.
4. Formats for data storage and communication protocols. It is not necessary to repeat the protocol as defined for the tutorial group, you can simply refer to it.

The *target group* of this report is a *software maintainer*, *i.e.*, somebody that did not necessarily write the code himself, but at a later occasion should extend or improve it, and therefore should be able to understand the overall design of your application.

Discussion per Class
Every class in the application should be discussed by summarising:

1. the role of the class in the system;
2. the responsibilities of the class;
3. the other classes that are used by this class to achieve its responsibilities;
4. if there are any special cases in the class's contract;

5. any precautions taken to fulfill the preconditions in the contract of the server classes.

The *target group* of this report is a *software maintainer*, *i.e.*, somebody that did not necessarily write the code himself, but at a later occasion should extend or improve one or more classes, and therefore should be able to understand the purpose and functionality of a class.

Test Report
As always, thorough testing of the application is an integral part of its development. You are expected to discuss both unit tests and system tests.

If you discovered errors in your implementation during testing, and had no time to fix these errors, you should indicate these results in this part of the report.

The *target group* of this report is a *software maintainer*, *i.e.*, somebody that did not necessarily write the code himself, but at a later occasion should extend or improve one or more classes, and then should be able to check whether at least the existing tests still pass.

Unit Testing Unit tests test the functionality of a class in isolation. Various techniques exist for this purpose:

- Using dedicated test classes, as you have learned during this module. This is the most thorough testing method for unit tests.
- Communication over a network can be tested by simulating a part of the system manually via `telnet`.
- Add a `main` method that creates several instances of the class under test, and then invokes several methods on these objects.
- Visual inspection of a GUI.

The report on the unit tests should describe for each class separately the following information:

1. how the class has been tested (in isolation, together with other classes, not at all);
2. which test technique has been applied;
3. if appropriate, test programs that have been developed;
4. test results and expected results; and
5. how much percent of each method are covered by tests (the coverage should be determined using Emma). Possibly discuss reasons and consequences of low coverage in a class.

You are allowed to use JUnit for unit testing, but you should clearly document how it has been used.

The test report should provide sufficient information for the reader to repeat the tests. Test programs should be included in the implementation.

System tests System tests will try the complete functioning application as a whole. The functionality requirements will serve as the basis for these tests, as the implementation should fulfill those. Each requirement may be tested separately by creating one or several test executions with different usage scenarios (both good and bad usage).

The system tests should also be described in the report. Again, describe which aspects of the system have been tested and how. In particular, you should describe:

1. which functionality has been tested;
2. the test execution, in such a way that the reader is able to repeat the test;
3. the expected behaviour of the system; and
4. the actual behaviour of the system (if different from the expected behaviour).

Reflection on Planning
Before the beginning of week 9, you were asked to make a planning for the project. During the last two weeks of the module, you should keep track of how your planning corresponds with your actual progress, and adapt your planning if necessary.

In your report you should reflect on this, and in particular you should describe the following:

1. How was your planning influenced by your experiences with the planning and time writing during week 4 of the module?

2. To what extent did your planning correspond to the actual progress during the project weeks? What made you deviate from your planning? For example:
    - Were there tasks within the project that took significantly more or less work than you had expected? If so, how come?
    - Were there significant losses of time or momentum in your projects (one or more periods in which you just did not seem to make the progress you had intended)? Looking back, how can you explain this?
3. Which countermeasures did you take to compensate deviation from your original planning? What was the impact of this on the intended scope or quality of the project?
4. What did you learn from this experience for your next (project) planning? Take your answers for answers for question 2 and 3 into account and ask yourself how you would want to prevent this or deal with this a next time.
5. Imagine you are next year's student assistant for this project. Please describe at least two dos and don'ts that you intend to tell your students to help them with their planning.

Minimal Requirements for the Project

The project should be graded with at least a 6.0. This paragraph provides a check list with minimal requirements for your implementation and your report. If you fulfill these requirements, your project will be graded with at least a 6.0.

Minimal Requirements for the Implementation

The implementation should satisfy the following requirements:

- The application should implement all the functional requirements discussed in the previous paragraph.
- The program should compile without any problems.
- The implementation should make use of the *Model-View-Controller* pattern and the *Observer* pattern.
- All self-defined classes must be in self-defined packages, the classes should be organized in at least three different packages.
- The code must follow a good coding style, including layout and variable naming. Specifically, it must not produce any checkstyle warnings (see below for details).
- Result values of methods should not be used to encode error states, but Exceptions and Exception handling should be used instead; all Exceptions explicitly thrown in own code should be self-defined.
- For the three most complex classes in the server (can also be classes that are shared with the client; see below for a definition of "most complex"): the classes and all their methods should be documented in Javadoc, with JML pre- and postconditions and class invariants (JML specifications must type-check with OpenJML).
- The server should be multi-threaded to support multiple concurrent games being played.
- There should be test classes and test runs. The tests should try different situations described in the functional requirements and game rules (e.g., the specified port for the server is free or in use; different number of players; or playing an illegal move)
- For the three most complex server classes (see below), the test coverage should at least be 50%, determined using Emma.

You should download the checkstyle configuration from Blackboard (go to COURSE MATERIAL and find the checkstyle configuration file as well as an instruction for importing it in the `Tools` item). Each violation of a check in the configuration will be reported in the Eclipse PROBLEMS view as a warning. As type of the warning "Checkstyle Problem" will be specified. Your project should not contain any such warnings produced by checkstyle. Be aware that that Eclipse does not show all warnings, should their number be too large. In this case, the PROBLEMS view will show a line like "warnings (100 of 200 items)". Make sure that you see all checkstyle warnings in such a case, e.g., you can open the menu of the PROBLEMS view, select CONFIGURE CONTENTS and uncheck USE ITEM LIMITS.

Extensive documentation and test coverage (as detailed above) should be provided for the three most complex, self-defined classes in the server application. The server application is formed by all classes that are necessary to run the server, some of these classes may be shared with the client application. To determine the complexity of classes, you should use the Eclipse Metrics tool and in particular the provided metric "Weighted Methods per Class" (WMC).

The implementation must be handed-in as a single ZIP archive. To create this archive, use EXPORT... from the FILE menu in Eclipse. Next, select ARCHIVE FILE from the GENERAL category. Select your project and press finish. The ZIP archive should have the following content and structure:

- All self-defined classes should be stored in a single directory hierarchy.
- There should be a `README` file with information about installation and starting the game, indicating for example which directories and files are necessary, and conditions for the installation. After reading this file, a user should be able to install and execute the game without any problem. The `README` file should be located in the rood folder of the project.
- Documentation of all self-defined classes (as Javadoc-generated HTML) in a directory hierarchy separate from the source files.
- Any non-standard predefined classes should be included in bytecode format.

Typical causes that make the installation and compilation procedure fail are names and paths or hard-coded URLs. *Test this before submitting your project.*

If your program contains references to the file system, e.g., to load image files, make sure to be platform independent. If may be that your application will be executed under a different operating system than you used during development. For further information, see the documentation for `java.lang.File` and in particular the constants `pathSeparator` and `separator` defined in this class.

Minimal Requirements for the Report

The report should describe the following:

- A discussion of the overall design of the application:
  - Document the use of the Model-View-Controller pattern: which classes and packages play the roles of model, view and controller?
  - A class diagram of the contents of the package holding your model classes (according to the Model-View-Controller pattern). The class diagram should show all public methods and fields, fields with a type that is contained in the class diagram should be represented as association.
  - For each functional requirements of the server application, the report should contain a discussion how it is implemented and in which class(es).
- Formats for data storage and communication protocols are described clearly.
- For each self-defined class:
  - The responsibilities of the class in the system. I.e., which functional requirements does it implement? If applicable, which rule(s) does it implement? Which role does it play in the implementation of the Model-View-Controller or Observer pattern (if any)?
  - For all classes that it refers to (i.e., as a field type, super type, argument type, local variable type or by calling its constructor) you should describe the purpose of using this class.
- For each of the three most complex server classes (see the description of the minimal requirements for the implementation):
  - Which precautions must be taken to fulfill the preconditions in the contract of the class.
- A report of the unit tests for each of the three most complex server classes (see again the description of the minimal requirements for the implementation):
  - Besides the class under test, which other self-defined classes are executed during each test.
  - The expected results for each test together with an explanation of the expected result (e.g., refer to the game rules or functional requirements), as well as the actual result when running the test.
  - A discussion of the test coverage of the class under test. Explain whether you consider the reached level of coverage high or low. For those parts of the code that are not covered, you should also discuss why they are not covered (e.g., why was it difficult to write tests that cover them).
- A reflection on your planning and self-management during the project:
  - How was your planning influenced by your experiences with the planning and time writing during week 4 of the module?
  - How much did your planning correspond to the actual progress during the project weeks?

- What changes did you make to your planning?
- What did you learn from this experience for your next (project) planning?

It is *not* necessary to copy your source code or your Javadoc-generated documentation in the report.

Project Activities

Planning To successfully complete the project, it is important to make a planning for it. You should at least plan during which periods you are going to work on *designing*, *implementing*, *documenting code*, *testing*, and *writing the report*. At least for the tasks implementation and report writing you should think of sub-tasks and plan them as well. The planning should be shown to a student assistant and signed off at the latest during the project session on Monday (6th and 7th hour) in week 9. The student assistant might give some suggestions how to adjust your planning, if he thinks it is unrealistic. The final goal of the project is to create a working program and a decent report. The student assistant has also done this during his first year, therefore it is important to take his feedback into account.

During the last two weeks of the module, the student assistant can help you to adjust your planning.

Impress Me Session Thursday of week 9, during the 3rd and 4th hour, there will be an *impress me* session where in all rooms several lecturers will be present to look at your intermediate results. All project teams are required to show what their application can do so far, during this session. Participation in this session is mandatory.

Peer Reviews At the end of week 9, you have to submit a preliminary version of your report, to receive feedback from your fellow students. The grouping for the peer reviews will be announced via Blackboard. You should send your report to your reviewers on Friday evening. The peer reviews should be discussed and signed off during the project session on Monday in week 10.

Remember what you learned about giving feedback during Module 1. Try to judge all components, and where appropriate provide additional details to support your judgement. It is strongly recommended that you discuss the feedback face-to-face.

Remember that fellow students are also doing this for you, so take your review job seriously. Moreover, understanding the approach from other people might also help your own implementation.

The peer review form can be downloaded from Blackboard.

Tournament Wednesday afternoon in week 10, a tournament will be organised where the different computer players will compete against each other. Bonus points can be gained by winning the tournament.

Submitting The report and code should be submitted via Blackboard. The code should all be in a single zip-file, and all classes should be there that are necessary to compile the application.

Important Dates

Thu 6–7, week 5	Tutorial group meeting on overall design
Thu 7–8, week 7	Tutorial group meeting on communication protocol
Thu 7–8, week 8	Tutorial group meeting on authentication protocol
Mon 6–7, week 9	Sign off project planning
Thu 3–4, week 9	Impress me session with lecturers
Fri, week 9	Send preliminary version of the report to peer reviewers
Mon 6–7, week 10	Peer review discussions and signing off
Wed 6–9, week 10	Tournament
Wed 23:59 CET, week 10	Submission deadline for maximal project grade
Fri 23:59 CET, week 10	Submission deadline for maximum 6 grade

Possible Extensions

If you have sufficient time, you are welcome to extend your game application. You could consider the following functionalities.

Chat Box

When the game application is developed as described above, it can only be used to play the game. It can be fun to have the possibility to communicate with your opponents during the game. Therefore, a possible extension is to add a field to the client GUI where the user can enter texts. After pressing the return button, the message is sent to the server, who then sends it to the other clients participating in the game, after which the message is shown on all client GUIs.

Challenge option

Up to now, clients are connected in order of logging on to the server. It would be nicer if a client could choose from the registered clients against who to play. To achieve this, the following changes will be necessary:

- a client should know which other clients are registered;
- a client should be able to choose his opponents;
- a client should be able to refuse a game.

Some of the extensions also require extending the server and the communication protocol. You should make sure that trying to use the extended functionality functionality with a server from a different team does not lead to a crash.

These extensions will only be graded if the application respects the basic functional requirements.

Grading

Both the implementation and the report will be graded separately with a grade between 0.0 and 10.0. The final grade for the project is the average of the two grades.

The minimal requirements as sketched above are necessary to score 6.0 points overall for the project assignment. If your implementation and/or report does better than the minimum requirements (e.g., you documented more parts, provide proper loop-invariants, provide more extensive tests, or discuss the design of larger parts of your system in the report), you can get up to 10.0 points.

If you submit your project on Friday, and if it fulfills the minimal requirements, you will score 6.0 points for the project. In this case, no bonus points will be awarded.

Rules for Bonus Points

There are several ways to earn bonus points. Bonus points will be added to the average grade of the implementation and report (see above). It is not possible to earn more than 10.0 points in total.

- The protocol as agreed upon in the tutorial group will be maintained and kept up-to-date by a single student. If this task is fulfilled properly, the student will get a bonus of 1.0 points.
- The winner of the tournament within an tutorial group will get 1.0 bonus points, the runner up will get 0.5 bonus points.
- For every extension to the basic application, such as chat and challenge functionality, at most 0.5 bonus points can be obtained.

All bonus points are added to the overall project result.

Grading Criteria

Finally, we give the list of grading criteria that will be used to grade the project assignment. You can use this list to judge your own progress with the project.

Code

1. The program has all the requirement components.
2. There is a `README` file with installation and execution instructions.

3. The program compiles without errors.
4. The program executes without errors.
5. The program has been sufficiently documented, with Javadoc, and JML pre- and postconditions, and class invariants.
6. The implementation of large and/or complex methods has been documented internally, including the use of loop invariants.
7. The program layout is understandable and accessible.
8. Packages and accessibility is used in a sensible way.
9. The program has a GUI.
10. Bytecode of predefined, external, non-standard Java classes is submitted with the code.
11. All used test classes and test executions are submitted with the code.

Design
1. The overall design is logical.
2. The implementation corresponds to the design in the report.
3. The Model-View-Controller and Observer patterns have been used well.
4. The program is divided in a logical way into classes.
5. All classes have detailed documentation.

Programming Style
1. Names of classes, variables and methods are well-chosen and clear.
2. Code is efficient and neatly implemented.
3. The program is easily maintainable (use of constants, variable names etc.)
4. The tutorial group's communication protocol has been properly implemented.
5. The exception mechanism is used appropriately.
6. The GUI looks decent and is scalable.
7. Concurrency constructs are used properly.
8. Artificial intelligence for a computer player has been implemented.

Testing
1. Appropriate unit tests are provided.
2. Appropriate system tests are provided.
3. Sufficient test coverage is reached.
4. Tests are well documented.
5. All classes in the system have been tested by unit testing.

Bonus Points
1. The chat functionality is correctly implemented and documented.
2. The challenge functionality is correctly implemented and documented.

Report Contents
1. The report has all the required components.
2. The general design is described clearly.
3. The class diagrams have been explained clearly.
4. The choices made in the overall design are well-documented and motivated.
5. The general design is written for the appropriate target group.
6. There is an overview of which functionality is implemented by which classes.
7. It is described how the Observer and Model-View-Controller patterns are used.
8. Formats for data storage and communication protocols are described clearly.
9. The role and responsibilities of classes is described clearly.
10. The choices in class implementations are well-documented and motivated.
11. The class implementations have been written for the appropriate target group.
12. The connections between classes are described clearly.
13. Special cases in the class's contract are clearly described.
14. Test activities are described clearly.
15. The test activities cover the whole application.
16. The test results have been analysed.
17. The results are written for the appropriate target group.
18. There is a reflection on the planning and its necessary re-adjustments.

Structure
1. The user can quickly see what the text is about.

2. The text is logically structured.
3. The necessary information can be found easily.

Language

1. The language in the report is understandable for the target group.
2. Spelling is correct and consistent.
3. Sentences are well-constructed.