Examination
Software Analysis and Design
DIT185

Software Engineering and Management
Chalmers | University of Gothenburg

Wednesday June 1, 2022

Time	08:30-12:30
Location	Lindholmen
Responsible teacher	Daniel Strüber (mobile: 0760475434)
Total number of pages	**7** (including this page)
Teacher visits exam hall:	At circa 09:30 and at circa 11:30
Exam (4.5 HEC)	Max score: 100 pts
Grade limits (4.5 HEC)	3: at least 50 pts
	4: at least 70 pts
	5: at least 85 pts

ALLOWED AID:

- English dictionary
- **NOT ALLOWED:** Anything else not explicitly mentioned above (including additional books, other notes, previous exams, or any form of electronic device: dictionaries, agendas, computers, mobile phones, etc.)

PLEASE OBSERVE THE FOLLOWING:

- Start each question on a new paper;
- Sort your answers in order (by question and sub-task) before handing them in;
- Write your student code on each page and put the number of the question on **every** paper;
- In several tasks, you will be asked to create a UML diagram. If any aspects of your created diagrams are ambiguous or potentially unclear without further explanation, please provide a textual description that explains these aspects.
- We set up this exam in such way that you can realistically complete it in four hours. In particular, the diagrams are generally smaller than the ones from the assignments.
- This exam is composed by eight exam tasks, some of them with further sub-tasks. Tasks 1-6 are based on the same case, a microblogging service (described on page 2).
- Points are denoted for each task and sub-task. The point distribution can give you an indicator of how much time to spend on each task and sub-task.

System description for Tasks 1-6:

The exam case is a microblogging service. A microblogging service is a social platform (phone app and web application) that allows users to write and share short postings – think Twitter. In our case, we want to develop a microblogging service where the maximal posting length is 240 characters.

In order for users to write postings, users need to be registered – that is, they need an account. Users can create an account, for which they specify a username, e-mail address and password. Accounts are initially inactive after registration. To activate the account, the user needs an activation code, which is sent to their e-mail address. Accounts can be banned by administrators, in response to inappropriate user behavior. Banned accounts can be unbanned by an administrator. In addition, bans can be temporary, in case of which the account can become active again after a specified penalty time has passed.

Every registered user owns a publicly accessible user page. The user page lists the user's postings. The postings are sorted in chronological order; latest postings first. Users also can share the postings of other users on their user page. Shared postings appear in the sequence according to the time when they were shared. User can also configure their page: they can enter a place of residence, a website, as well as a brief description.

Users can "follow" other users, that is, subscribe to their postings. The postings of all followed users are displayed in a special page that is custom-generated for each registered user (the so-called *feed*). On the user page, the users who follow the user as well as those who follow the user are listed. Users have a setting to control whether they individually need to confirm new followers. Users who follow each other can send each other private messages.

Task 1: Requirement modeling (12 pt.)

a) Create a use case diagram for the microblogging service. Your use case diagram should include all important use cases, and only use cases that are actually important for the app. Make sure that your use-case diagram is well-structured: It should have a system boundary, at least two actors outside that boundary, relations from the actors to all use cases (and labeled extensions where appropriate). Where appropriate, align use cases in a hierarchy (<<extend>>, <<include>> relationships). (8 pt.)

b) Provide correct description (according to the Cockburn template) for one central use case from your use case diagram. Include at least one extension and one sub-variation. You can find the Cockburn template on the last page of this examination sheet. (4 pt.)

Deliverables:

- Task 1a: Use case diagram
- Task 1a: Textual description (optional)
- Task 1b: Use case description

Task 2: Domain modelling (12 pt.)

a) Create a domain model, in the form of a class diagram, for the microblogging service. Recall that a domain model depicts the concepts from the real-world domain that need representation in the system. Develop your domain model in a systematic way by applying guidelines and heuristics, such as those from the lectures. Create classes with important associations (including multiplicities, where applicable) and attributes. Create at least one generalization (inheritance relationship). (8 pt.)

b) Provide a textual description that explains how you developed the domain model, including specific explanations of how you used guidelines and heuristics. If any aspect of your domain model is not self-explanatory, explain it here. (4 pt.)

Deliverables:

- Task 2a) Domain model (class diagram)
- Task 2b) Textual description of how you developed domain model (half a page)

Task 3: Behavior modelling: state machine diagram (12 pt.)

a) Create a design model, in the form of a state machine diagram, that models the different states of accounts in the microblogging service. Include at least three states that accounts can be in, and the possible transitions between the states. (8 pt.)

b) Illustrate your state chart diagram with two object diagrams. Choose a transition from your state machine diagram, and create two object diagrams showing snapshots of the system directly *before* and *after* the transition. It's okay to show a small excerpt of the system – you can focus on those parts that are affected by the transition. (4 pt.)

Deliverables:

- Task 3a) Design model (state machine diagram)
- Task 3b) Two object diagrams, name of transition

Task 4: Behavior modelling: sequence diagram (12 pt.)

Create a sequence diagram that models the interaction between two users that follow each other and then send each other messages (one private message each). Use a white-box modeling approach: that is, your model should include at least two interacting objects of the microblogging service. You can assume that there is a class diagram that contains the relevant classes for that (you don't need to draw the class diagram).

Deliverables:

- Task 4: Sequence diagram, modeled in white-box way

Task 5: Architecture modelling (12 pt.)

a) Create a component diagram for the microblogging service. For this, choose an architectural style and create a component diagram for the application based on it. For this task, assume that the microblogging service should be supported by different user interfaces, e.g., through several smart phone apps and through a website. Model at least three interacting components together with their interactions (either through interfaces or dependencies). (8 pt.)

b) Explain the design rationale for your component diagram: explain the reasons for choosing the architectural style, and make references to relevant design principles. (4 pt.)

Deliverables:

- Task 5a) Architecture model (component diagram)
- Task 5b) Textual description of rationale

Task 6: Consistency (10 pt.)

1. Come up with two small examples for inconsistencies in UML diagrams, that is, diagrams in which consistency rules such as those seen in the lecture are violated. You can use the same pair of diagram types (e.g., class diagrams vs. sequence diagrams) for both examples, but you are not allowed to use the same type of violation (same rule violated) several times. For each example, draw two diagrams and name the consistency rule that is violated. (5 pt.)

2. Consider your class diagram from Task 2 and your sequence diagram from Task 3. The class diagram is probably not consistent with the sequence diagram yet, because

it is a domain model. However, it's possible to make the class diagram consistent, by making a few changes to it, turning it into a design model.
Which changes are required? Please give concrete steps with details (e.g., names of concrete elements added) to explain how the class diagram can be made consistent with the sequence diagram.
You can find the relevant consistency rules on the last page of this examination sheet. (5 pt.)

Deliverables:

- Task 6a: Two examples, including a pair of models and a violated consistency rule
- Task 6b: Description of steps for establishing consistency.

Task 7: Applying design patterns to three cases (15 pt.)

Below, you will be presented with three small example cases. For each of them, provide the design based on an appropriate design pattern. Use a different design pattern for each case. For each case, name the design pattern and create a class diagram showing how the implemented design pattern looks like for the case. (5 points per case)

Case 1: Distributed supermarket cash desk. In this case, we want to create a software system for cash desks in a supermarket. All information about articles and prices is stored in a central database. The individual cash desks store a local copy of the information from that database. When the price of existing articles is updated, all cash desks have to be informed about the new price, and update their copies of the database with the new price.
Please use a design pattern for handling the updates of information about new prices.

Case 2: Logger. A logger is an application that takes information about a running software system and writes it into a log, either a file or a database. Since there might be many things going on in the system in parallel, it's important that the logging happens in a coordinated way: A single object should be responsible for handling all logging.
Please use a design pattern for making sure that there is indeed a single object handling all the logging.

Case 3: Blocks in a sandbox video game. A sandbox game is a game where players can freely design the world, by building it from blocks. In their most basic form, blocks are just generic building blocks. However, blocks can have many additional features: for example, they can glow in the dark, be semi-transparent, or be radioactive. Different kinds of blocks can be created by combining several of these features.
Please use a design pattern that supports the flexible creation of blocks from combination of features.

Deliverables:

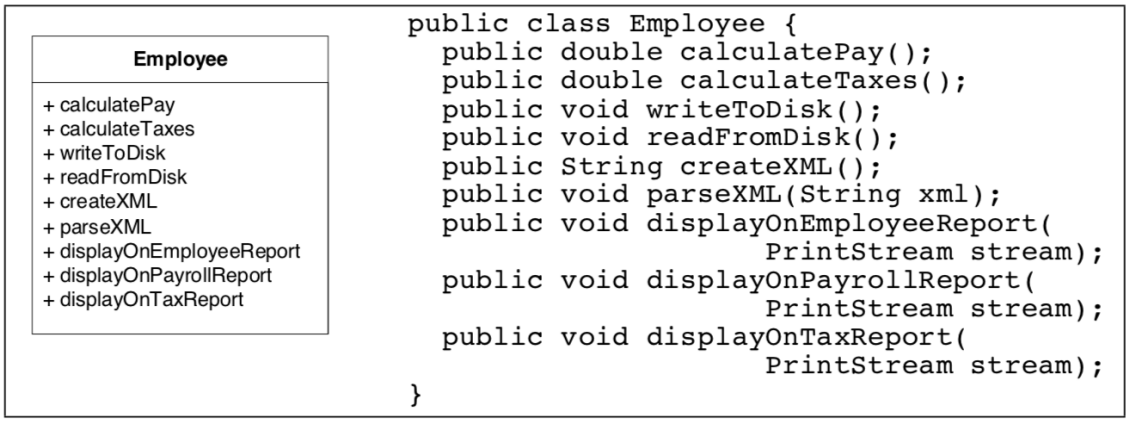
- Task 7: For each case, a class diagram and the name of design pattern

Task 8: Improving software quality via design models (15 pt.)

a) Critically discuss the software quality of the class diagram below. Explicitly address at least two design principles.

b) Create an improved version of the class diagram.

c) Discuss the rationale for your improved class diagram: why does it lead to an improvement? Explicitly address the design principles again that you previously addressed in task 8a).




```
public class Employee {
  public double calculatePay();
  public double calculateTaxes();
  public void writeToDisk();
  public void readFromDisk();
  public String createXML();
  public void parseXML(String xml);
  public void displayOnEmployeeReport(
                        PrintStream stream);
  public void displayOnPayrollReport(
                        PrintStream stream);
  public void displayOnTaxReport(
                        PrintStream stream);
}
```

Explanations: Methods `calculatePay` and `calculateTaxes` calculate the salary and the taxes of the employee. Method `writeToDisk` saves the information about an employee to a file in the file system, `readFromDisk` reads the information about an employee from a previously saved file in the file system. The methods `createXML` and `parseXML` create a String representation of an employee's information (in XML format) and read an employee's information from the same type of String representation, respectively. The methods `displayOnEmployeeReport`, `displayOnPayrollReport`, and `displayOnTexReport` determine how the employee information is represented in different types of reports: the created representation will be added to the provided `PrintStream`.

Deliverables:

- Task 8a: Textual discussion of software quality (one paragraph)
- Task 8b: Class diagram
- Task 8c: Textual discussion of rationale (one paragraph)

Appendix

For Task 2: Cockburn Template (relevant excerpt, to be filled out by you)

Use case name:	<the name should be the goal as a short active verb phrase>
Goal in context:	<a longer statement of the goal>
Primary Actor:	<a role name for the primary actor, or description>
Precondition:	<what we expect is already the state of the world>
Success End Condition:	<the state of the world upon successful completion>
Failed End Condition:	<the state of the world if goal abandoned>
Trigger:	<the action upon the system that starts the use case, may be time event>
Main Success Scenario:	<put here the steps of the scenario from trigger to goal delivery, and any cleanup after> <step #> <action description>
Extensions:	<put here the extensions, one at a time, each referring to the step of the main scenario> <step altered> <condition>: <action or sub use case> <step altered> <condition>: <action or sub use case>
Sub-Variations:	<put here the variations that will cause eventual bifurcation in the scenario> <step or variation #> <list of variations> <step or variation #> <list of variations>

For Task 6: Consistency Rules for Class Diagrams vs. Sequence Diagrams

1. The type of a lifeline (type of the connectable element of the lifeline) in a sequence diagram must not be an interface nor an abstract class.
2. If a message in a sequence diagram is referring to an operation, that operation must not be abstract.
3. If a message in a sequence diagram refers to an operation, through the signature of the message, then that operation must belong, as per the class diagram, to the class that types the target lifeline of the message.
4. Interactions between objects in a sequence diagram, specifically the numbers of types of interacting objects, must comply with the multiplicity restrictions specified by the class diagram (e.g., association end multiplicities).
5. In order for objects to exchange messages in a sequence diagram, the sending object must have a handle to the receiving object as specified in the class diagram. Another way of saying this is that the sender must have visibility to the receiver. A specific case of this situation is when the sending object's class has an association (possibly inherited) to the receiving object's class.
6. The behavioral semantics of a composition or aggregation association in the class diagram must be inferred in sequence diagrams. For instance, in a whole-part (composition) relation, the part should not outlive the whole.
7. Each public method in a class diagram triggers a message in at least one sequence diagram.
8. Each class in the class diagram must be instantiated in a sequence diagram.
9. No operation can be used in a message of a sequence diagram if this breaks the visibility rules of the class diagram (public, protected, private).