2.7.2 Regarding solving the exercise

The typical exercises require you to identify variability points in the problem, and use 3-1-2 and compositional design to provide as flexible framework.

2.8 Regarding Clean Code and Refactoring

2.8.1 Regarding the theory

The central theory of Clean Code is the definition and understanding of the properties outlined by Robert Martin, specied up with those that I have formulated. Bringing a Clean Code sheet to the exam may help to get an overview.

2.8.2 Regarding solving the exercise

The typical exercises contain a code section with unclean code, so solving the exercise is largely a matter of spotting the clean code properties that are not kept (document and explain it using the clean code sheet), and refactor the code into a form which is more clean.

Remember that the theme involves *refactoring*: do not just present a refactored design that you made in the preparation time. We need to understand and see your *refactoring process*—so pick one problematic area in the code base, and then demonstrate the code that replaces that particular problem, and move on to the next problematic part.

One pitfall I have seen students fall into is *writing a new method from scratch* instead of refactoring the provided code example. This is **NOT a proper solution** and will fail the exam. The focus is improving existing code!

2.9 Regarding Broker

2.9.1 Regarding the theory

The central theory of Broker is the architectural pattern, so be sure to understand UML structure, and the roles involved: what each role is responsible for, and what protocol is involved—that is—the call sequence and transformation of data along the call sequence. Also ensure to understand the typical parameters that pass between roles, as this helps you in designing solutions to the exercises—to get the method parameter lists correct.

Passing object “references” from server to client has its own set of methods and techniques that you have to master: The creation of objectId, the use of Name Services, the client having to create ClientProxies based upon returned objectId, and of course the key point—that the server can look up the proper objects using the name service.

2.9.2 Regarding solving the exercise

The typical exercises present some kind of Java interface that allows clients to call methods that are executed on a remote server—and your task is to sketch the design (typically adapting the Broker UML to the concrete exercise) and sketch parts of the Java code for those Broker roles that are not general: ClientProxies and Invokers.

All Broker exercises will involve passing server created objects.

So - be sure to study the TeleMed, the GameLobby, and your own Broker solution code base: understand which roles can be implemented in general and may thus be in a library; and which roles you always must code yourself. Also ensure you can sketch the code that handles the mechanism for transferring server created object to the client: the use of object identities, name services, and proxy creation on the client side.

While you can bring much code that you have made in the preparation, remember to carefully explain the code you show: In the ClientProxy tell us what the individual parameters are (object id, operation name, that X.class, parameters), what they represent; and similar on the Invoker side.

If you get overwhelmed by the exercise, a fall-back strategy is to sketch a design using only the "Broker I" part of Broker—that is ignore the issues of "server created objects". That way, you can ignore the issues of name services, objectId creation, etc., and focus on getting ClientProxy and Invoker code presented but treat the return value as a 'pass by value' object.

To examplify this, consider method `Card getCardInField()` from HotStone. In your presentation, you can just treat Card as an pass-by-value object that GSon can marshall and demarshall for you (just as if the return type was "String" instead).

Of course, you do not score top marks doing this, but rather a confident simple solution than a completely messy full solution.