2.5.2 Regarding solving the exercise

Be sure to read the exercise so you solve the right exercise...

These exercises are typically of type "identify the pattern that can solve this problem." If you do not immediately see which one it is, then do not panic. It is better to use the 3-1-2 process and compositional design principles to come up with a flexible compositional solution to the exercise and then we can probably discuss our way towards a concrete pattern during the examination.

Remember to demonstate your ability to express the theory in concrete code by developing code fragments and diagrams.

2.6 Regarding Compositional design

2.6.1 Regarding the theory

The central topics are FRS Chapter 16, 3-1-2 and the compositional design principles, and Chapter 15, behaviour, responsibility, roles, and protocol.

It will be natural to connect to the themes of design patterns, frameworks, and variability mangement.

2.6.2 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 solution.

2.7 Regarding Frameworks

2.7.1 Regarding the theory

Framework theory is basically the 3-1-2 process and compositional design principles but with some additional/supplementary terminology. So, be sure to include concepts like *hotspot, frozen spot, and inversion of control*, as well as understand the characteristics of frameworks. The TEMPLATE METHOD is a central pattern to understand, in both its two variants - *separation and unification.*

Time permitting in your exam planning, have a look at the source code for MiniDraw (guided by the slides and FRS 30) and review how the concepts are applied in practice.

It will be natural to connect to the themes of design patterns, compositional design principles, and variability mangement.

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.