Train yourself to code a given variability point in the three major code based variants: parametric, polymorphic, and compositional. For instance, code HotStone winner strategies using the three techniques.

Express your solution using code fragments written using a not-too-fuzzy code editor.

Often the exercise will ask you to provide both a compositional as well as parametric/polymorphic solution. If pressed for time, focus on the compositional one.

2.4 Regarding Test Doubles and unit/integration testing

2.4.1 Regarding the theory

The central concepts are direct and indirect input, depended-on unit, and the replacement of these, as well as the different types (stubs, fake objects, spies). Next, the levels of testing (unit, integration, and system) from FRS 8.1.5. are relevant.

2.4.2 Regarding solving the exercise

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

Look for the aspects of the exercise that is not under automated test control (random values, sensor values, hardware input, system clock input, etc.)—and use the standard compositional approach for encapsulating the indirect input behind interfaces.

Remember to demonstrate your ability to express the theory in concrete code by developing code fragments/diagrams.

2.5 Regarding Design patterns

2.5.1 Regarding the theory

The central theory is that of compositional design principles as the "engine" inside almost all design patterns. In addition it is obvious to include the definitions of design patterns (there are four in the book), and at least the ones in Chapter 9.

Review the design patterns in the FRS catalogue (or you can find them all in the excellent poster by Simon Kracht / find the link on weekplan 3) and review their Intent section (you can find that in the Design Pattern Index which you can download from baerbak.com under "missing insets in new printing".)

Many patterns have a rather similar structure to STRATEGY but review closely those that are a bit different. E.g. the BUILDER has its `getResult` method only in the delegates and the instance is declared by the concrete type, not by the interface type (why?); PROXY and DECORATOR has a special structure, etc. For top presentations, these details are important.