Demonstration of Competence — Module 226A

Class-based Implementation (with Focus on Delegation)

Concept for Demonstrating Competences

As an entry in this module you will be assessed according to your basic knowledge of the module 404 („object-based programming“). This basic knowledge is assessed by a discussion or a set of questions. When you have passed this basic test, you can start with this competence grid.

The development of your competences is shown through a set of „learning fields“. Within these „learning fields“ you can develop your own ideas and use them to demonstrate your knowledge, according to the demanded competences. The competence grid is the basis for assessing your expertise throughout this module. There is a list of themes (tasks) you can do in order to show that you know the competence.

Each „learning field“ relates to a specific competence or combines competences. You assess your work and make sure it relates to the given goals in this module (refer to the official Module Identification). Your teacher can assist you.

Demonstration of competence: Each student shows the described competences during the module to the teacher. If the teacher assess this as fit, the competence is signed off accordingly. By working with „learning fields“ you can combine certain competences. Only your personal work can be assessed. If you use other sources, you must declare and comment this.

Grading: Each competence field 0.25. The test counts as 1/3 to the module.

You can demonstrate the achieved competence in following manner:

- **Expert discussion**
  The student shows that he has achieved the competence by a small expert talk
- **Product**
  The student has a concrete product to show (code, document, diagram, etc.)
- **Report**
  The student keeps a log on his/her activities

You document your progress and your results….

- in a portfolio (ie. repository), whereby they are structured and accessible.
- in a log that you write throughout the module duration.
- if necessary through a planning tool, where you define your next steps (ie for a larger application).

Class-based Implementation (with Focus on Delegation)

Competence grid for Module 226A **Name:**_____

Action Goal	A 3.0	B 4.0	C 5.0-6.0*
1 OO Design & UML-Notation (UML 2.5)	*Hz 1.1; 2.1* The student can translate requirements into a design and show how classes cooperate. He/she knows how to design with a UML class diagram and can visually represent the difference between a general association and composition from an implemented code example.	*Hz 1.2; 2.2* Requirements are translated into a useful design. The design shows differences between forms of associations. The student can show the benefits of use cases (and the difference to unit tests), and show the difference between a static and dynamic view of an application (class diagram vs. sequence-diagram).	*Hz* Your own larger application has a detailed design with a UML class diagram and sequence diagram. The design shows the progress (planned implementation vs final status).
2 Implementing a Class-based design	*Hz 3.1* The student knows the syntax of OO (class, object, references) and can show this with examples. Different collections can be used accordingly. Exception-Handling is used correctly and extended (ie. own exceptions). Threads are known and can be shown by examples.	*Hz 3.3* Your own code examples show the principle of Information Hiding. Your examples demonstrate how classes cooperate and they show how delegation works (and its benefits). Further usage of collections are included. Possible usage of Lambda functions (and why they can help).	*Hz 3.2* Your own larger application shows a very good grasp of oo-concepts (with focus on delegation and aggregation, composition). Encapsulation is shown in a sound manner. Classes and methods are well structured.
3 Testing & Documenting	*Hz 4.1, 4.4* The student can write useful test cases for the code, thereby showing the connection between use cases and testing. The difference between blackbox / whitebox testing is shown. Your code is commented.	*Hz 4* The student knows how test-driven development works and they can show unit tests for their own examples. Testprotocols are included for the examples. JavaDoc is used in addition to the basic commenting in code. Mock-ups are used when necessary.	*Hz 4.1;4.3* Your own application has a large unit test coverage (90% of functionality). You also include security tests. Test cases are part of your project documentation.
4 Method competence & Learning Progress	*Hz* You work with a version control tool for code and documentation. You show your learning progress by keeping a log. You share tips and tricks with others in the class by making them public (ie. presentation).	*Hz* The student plans his next steps. This is published and documented. Collaborative work is visible in the version control tool (commits, etc.). A blog is kept documenting your work and progress.	*Hz* The project is well-documented (with MarkDown) and shows how you worked in a team. Agreements (deadlines, milestones) during coding are kept and documented.

Hz = Handlungsziel (the official action goals in the module ID document)

*grading is according to scope and complexity of your own application. There is a separate checklist as a help.

Class-based Implementation (with Focus on Delegation)

Name:_____

Additional Information to the competences

Action Goal	A 3.0	B 4.0	C 5.0-6.0
1 OO Design & UML-Notation (UML 2.5)			
	Class diagrams are shown in various levels of detail (starting from very basic designs to a fine-level design).	Static and dynamic UML diagrams are shown by examples. Forms of associations are interprated in a correct way.	Your design should be detailed and reflect the actual code. Use your initial version as a base and produce the final version for comparison. (note: do not generate the diagrams with a tool!)
	➔ Use the selection of **themes** as a starting point for designs.		
2 Implementing a class-based design			
	Take your basic knowledge of Java as a starting point to show examples. Explore further themes like exception-handling (own classes), threads, usage of APIs, etc.	Delegation is one of the main themes in this module. Show in your examples how classes collaborate. Also show why sometimes inheritance is a bad solution. What can you do with Lambda functions?	You own larger application covers the main aspects of oo-programming (ie clear structure of what classes should do, delegation of tasks, collaborative work between classes). Inheritance is not a priority here, but can be used if necessary.
	➔ Use the selection of **themes** as a base (or point of inspiration). You are free to choose your own topics for applications / examples.		
3 Testing & Documenting			
	Use unit-tests to test main features of your examples. Also explore if mockups are necessary.	Show how test-driven development works by using a concrete example.	Your application is thoroughly tested and documented.
4 Method competence& Learning Progress			