# Klassenbasiert (ohne Vererbung) implementieren

**Konzept für den Kompetenznachweis des Moduls 226A**

Für den Einstieg in dieses Modul wird eine minimale Grundkompetenz aus dem Modul 404 verlangt. Diese Grundkenntnisse werden mittels einer schriftlichen Prüfung (oder in einem Fachgespräch mit Fragen) gewährleistet. Siehe dazu die Vorgaben, wie man sich auf diesen Test vorbereiten kann.

Die Kompetenzentwicklung wird im Modul 226 mit einzelnen Aufgabenstellungen nachgewiesen. Als Grundlage der Bewertung dient ein Kompetenzraster, welches einer Matrix von 4 Zeilen und 3 Spalten entspricht.

Jeder Aufgabenstellung ist ein Kompetenzprofil zugeordnet. Die Lehrperson übernimmt dabei die Rolle des Coaches und stellt den Bezug zu den Handlungszielen sicher.

**Nachweis einer Kompetenz:** Der einzelne Lernende hat die beschriebenen Kompetenzen im Verlaufe des Moduls nachzuweisen. Ist die entsprechende Kompetenz nachgewiesen worden, so wird dies durch das Visum des Coachs und des Lernenden im dazugehörigen Feld bestätigt.
Bewertet wird grundsätzlich nur die vom Lernenden erbrachte Eigenleistung. Fremde Quellen sind zu kennzeichnen und zu kommentieren.

**Notengebung:** Prüfung 33%, Kompetenzraster 66%

Folgende Formen sind für den Nachweis einer Kompetenz denkbar:

- **Fachgespräch**
  In einem Gespräch zeigt der Lernende, dass er über die Kompetenz verfügt.
- **Konkretes Produkt**
  Der Lernende zeigt die Kompetenz in Form eines entsprechenden Produkts (Programm, Dokumentation, erstelltes Diagramm, usw.)
- **Bericht**
  Der Lernende hält seine Erkenntnisse in einem selbst erstellten Bericht fest.
- **Sonstiges**
  Es sind auch andere Formen denkbar. Im Fokus steht immer der Nachweis der entsprechenden Teilkompetenz anhand von Eigenleistungen.

Der persönliche Lernfortschritt und die eingebrachten Nachweise sind:

- in einem **ePortfolio** zu *sammeln*, zu *strukturieren* und *zugänglich* zu machen
- im **Kompetenzjournal** festzuhalten.
- vorausschauend zu **planen mit einem geeigneten Planungswerkzeug** (siehe Zeitbedarf).

**Kompetenzraster für das Modul 226A** **Name:**_____

| Handlungsziel | A 3.0 | B 4.0 | C 5.0 – 6.0* |
|---|---|---|---|
| **1 OO Design & UML-Notation** | *Hz 1.1; 2.1*<br>Der Lernende kann Anforderungen in ein Design umsetzen und dabei zeigen, wie Klassen miteinander arbeiten. Er kann das Design mittels einem UML-Klassendiagramm umsetzen und kann Assoziation und Komposition darstellen. | *Hz 1.2; 2.2*<br>Die Anforderungen werden mit einem sinnvollen Design umgesetzt. Das Design zeigt Assoziation, Komposition und Aggregation. Der Lernende kann auch den Nutzen / Unterschied zwischen UseCases und statischen (Klassendiagramm) / dynamischen (Sequenzdiagramm) aufzeigen. | *Hz*<br>Die eigene Anwendung hat ein ausgereiftes Design mit geeigneten UML-Diagrammen (Use Cases, Klassendiagramm, Sequenzdiagramm). Das Design ist Teil eines Projekts und wird in der Implementation widerspiegelt. |
| **2 Klassenbasiertes Design implementieren** | *Hz 3.1*<br>Der Lernende kennt die Syntax der OO-Sprachelemente (Klasse, Objekt, Exceptions, Referenzen) und kann diese an Beispielen zeigen. Dabei können auch Collections eingesetzt werden und allgemein Bibliotheks-Pakete verwendet werden. Die Verwendung von static ist bekannt. | *Hz 3.3*<br>Anhand von erweiterten, eigenen Beispielen wird das Konzept der Datenkapselung aufgezeigt (Information Hiding). Das Zusammenspiel verschiedener Klassen ist ersichtlich und das Prinzip der Delegation kann angewendet werden. Weitere Formen von Collections werden angewendet. (z.Bsp. Map) | *Hz 3.2*<br>Die eigene Anwendung zeigt ein vertieftes Verständnis der OO-Konzepte Assoziation, Aggregation und Komposition. Dabei wird die Datenkapselung sehr gut umgesetzt, u.a. in dem Klassen und Methoden klar strukturiert sind. |
| **3 Testen & Dokumentieren** | *Hz 4.1, 4.4*<br>Der Lernende kann Testfälle / Testszenarien für seine Beispiele erstellen und ausführen. Der Unterschied Blackbox / Whitebox-Test ist bekannt und kann demonstriert werden. Der Quellcode wird kommentiert. | *Hz 4*<br>Der Lernende kann Unit-Tests zu den eigenen Beispielen aufzeigen. Testprotokolle und ein Testfazit werden für die Beispiele aufgeführt. JavaDoc erweitert das gewöhnliche Kommentieren im Code. | *Hz 4.1;4.3*<br>Die eigene Anwendung umfasst ausführliche Testfälle (pro Bereich) und deckt sinnvoll die Funktionalität ab. Es können SecurityTests (Grenzwerte, Pfade, Wertebereichte) umgesetzt werden. Die Testfälle sind in der Projekt-Dokumentation enthalten. |
| **4 Methoden- Sozialkompetenz & Selbstlernkompetenz** | Der Lernende zeigt, dass er das Gelernte auch anderen Lernenden mitteilen kann (mit Publikation, Vorträgen, etc.). Er kann eigene Verantwortung für das Lernen aufbringen und neue Erkenntnisse anderen erklären und vorstellen. | Der eigene Lernfortschritt wird selbst geplant und auch geprüft. Das wird entsprechend dokumentiert. Ergebnisse werden kritisch hinterfragt. Der Austausch in Gruppen / Klasse ist ebenfalls nachvollziehbar (mittels Journals, einem Blog oder durch ein Repository Tool). | Im Projekt sind Vorteile (oder auch Nachteile) der Zusammenarbeit erkennbar. Ein Teamverhalten wird an konkreten Zeichen erkennbar. Abmachungen (Termine, produktiver Code, etc.) werden eingehalten. Die eigene Anwendung umfasst eine detaillierte Dokumentation. |

Hz = entspricht den Handlungszielen in der Modulidentifikation

*Notengebung erfolgt nach Umfang und Komplexität der eigenen Anwendung. Siehe separate Checkliste als Hilfe.

**Name:**_____

**Ergänzungen zum Kompetenzraster**

| Handlungsziel | A 3.0 | B 4.0 | C 5.0 – 6.0 |
|---|---|---|---|
| **1 OO Design & UML-Notation** | | | |
| | Klassendiagramme, Sequenzdiagramme können interpretiert und mit einem UML-tool richtig dargestellt werden.<br>Planung, Empfehlungen; Erkenntnisse sind in einer geeigneten Struktur dokumentiert (z.B. Kompetenzjournal, ePortfolio, usw.) | Einsatz von statischen und dynamischen UML-Diagrammen an konkreter Aufgabenstellung, richtige Darstellung und Interpretation von Assoziationen | kleinere Aufgabenstellung (ca. sechs Fach-Klassen) mit geeigneten UML-Diagrammen als Vorlage für eine Implementierung ausgereift dargestellt. |
| | geeignete Aufgabenstellungen: Lagerverwaltung; Buchverwaltungen; CD-Verwaltung | | |
| **2 Klassenbasiertes Design implementieren** | | | |
| | Grundlegende Kompetenzen in Java- und insbesondere OOP-Sprachelemente können an praktisch gelösten Beispielen nachgewiesen werden. (Klasse, Zugriffspezifizierer, Konstruktor, Java-Sprachelemente, usw.) | Praktische Kompetenzen über Datenkapselung, Delegation, Aufteilung in Subsysteme können mit einer Liste von gelösten Aufgabenstellungen nachgewiesen werden | Implementation stimmt mit einem vorherigen UML-Design überein. OO-Konzepte werden richtig eingesetzt und können begründet werden. Die Applikation hat keine ersichtlichen und logischen Fehler. |
| | Aufgabenstellungen können selber zusammengestellt werden; Auf dem BSCW gibt es zu den meisten Themen kleine Theorien und Aufgabenstellungen, auch das Durcharbeiten von Tutorien kann sinnvoll sein. Siehe auch die Unterlangen „Themes“ mit einzelnen Themenschwerpunkten zu den Kompetenzen. | | |
| **3 Testen & Dokumentieren** | | | |
| | Testplanung, Testfälle, Testprotokollierung, Testfazit kann an einem konkret durchgeführten Beispiel nachgewiesen werden.<br>Eine gelöste Aufgabenstellung kann sinnvoll und verständlich dokumentiert werden | Der Einsatz von Test-tools wie JUnit kann mittels einer umfassenden sinnvollen Übung (Modul-Systemtest) nachgewiesen werden.<br>Die praktischen Kompetenzen für den richtigen Einsatz von JavaDoc können nachgewiesen werden | Die gewählte Aufgabenstellung wird vollständig und fachgerecht getestet.<br>Die Dokumentation ist vollständig und beschreibt den Entwicklungsprozess |
| **4 Methoden- Sozialkompetenz & Selbstlernkompetenz** | | | |
| | Der Lernende zeigt, dass er das Gelernte auch anderen Lernenden mitteilen kann (mit Publikation, Tipps, etc.). Er kann eigene Verantwortung für das Lernen aufbringen. Er zeigt seinen Entwicklungsprozess (wie z.B. aus gemachten Fehlern neue Einsichten entstehen). | Der eigene Lernfortschritt wird selber geplant und auch geprüft. Das wird entsprechend dokumentiert. Ergebnisse werden kritisch hinterfragt. Der Austausch im Tandem / Gruppen ist ebenfalls nachvollziehbar (mittels Journal, Gesprächen, etc.). | Im Projekt sind Vorteile (oder auch Nachteile) der Zusammenarbeit erkennbar. Ein Teamverhalten wird an konkreten Zeichen erkennbar. Abmachungen (Termine, produktiver Code, etc.) werden eingehalten. |