




## Kompetenzraster

Name: \_\_\_\_\_ Klasse: \_\_\_\_\_ Note: \_\_\_\_\_

Handlungsziel	Kompetenzen				
	1	2	3	4	5
Ein objektorientiertes Design nachvollziehen und durch Einsatz der Vererbung erweitern. <b>HZ1</b>	Ich kann Situationen mit redundanten Informationen in mehreren Klassen durch Einführung einer Basisklasse vermeiden.  ✓ ✓	Ich kann überprüfen, ob in einer gegebenen Situation Vererbung angebracht ist, oder ob es besser wäre mit Komposition zu arbeiten.	Ich kann ein objektorientiertes Programm mit einigen fachlichen und technischen Klassen entwerfen, welches frei von Redundanzen ist.	Ich kann eine switch-artige Struktur durch Anwendung von polymorphen Variablen (Delegation und Methoden überschreiben) eliminieren und fördere so die Erweiterbarkeit meines Codes.	Ich kann einfachere Design-Patterns, wie Composite, Iterator, Observer, Template Methode etc. anwenden.
Die Notation dynamischer und statischer Strukturen einer Anwendung mittels Unified Modeling Language (UML) nachvollziehen. <b>HZ2</b>	Ich kann die statische Struktur eines objektorientierten Programms mit einigen Klassen in einem UML Klassendiagramm dokumentieren.	Ich kann die Objektsituation eines objektorientierten Programms zu einem bestimmten Zeitpunkt mit einem Speicherdiagramm erklären.	Ich kann in einem UML Sequenzdiagramm den Aufruf von überschriebenen Methoden bei Objekten mit unterschiedlicher Klasse, aber identischer Basisklasse, illustrieren. (Illustration des dynamischen Bindens bei polymorphen Variablen)	Ich kenne die Bedeutung von weiteren UML-Elementen, wie Interfaces, Pakete, Namespace (Klassendiagramm) und kombinierte Fragmente für Schleifen und Entscheidungen (Sequenzdiagramm) und kann damit meine Diagramme noch aussagekräftiger machen.	Ich kann mit einem UML-Tool ein objektorientiertes Modell für ein Programm erstellen und daraus Code erzeugen. Den Code kann ich von Hand zu einem lauffähigen Programm vervollständigen und schliesslich das Modell mit dem fertigen Code synchronisieren.

Objektorientiertes Design implementieren.  <b>HZ3</b>	Ich kann mit C# eine Basisklasse und davon abgeleitete Klassen mit Instanzvariablen, und Konstruktoren implementieren. Ich vermeide dabei Redundanz im Code.  <div style="text-align: center;">   </div>	Ich kann Methoden in Basisklassen deklarieren / implementieren und in abgeleiteten Klassen implementieren / überschreiben. Ich verwende dann die Basisklasse zur Deklaration von polymorphen Variablen.	Ich kann ein Listener-Interface implementieren oder eine Adapterklasse erweitern und damit Ereignisse in C# Programmen behandeln.	Ich kann ein lauffähiges objektorientiertes C# Programm mit mehreren Klassen und einer minimalen grafischen Benutzeroberfläche oder mit Konsolenbedienung implementieren.	Ich mache ausgiebigen Gebrauch von weiteren C#-Sprachmitteln und den in der .Net-Bibliothek verfügbaren Klassen.
Fortgeschrittene Testfälle für funktionale Einheiten implementieren, welche durch geeignete Techniken von anderen Systemteilen unabhängig sind.  <b>HZ4</b>	Ich kann für eine gegebene Klasse ein Interface definieren und dieses in einer anderen Klasse an Stelle der ersten Klasse einsetzen. Auf diese Weise kann ich die beiden Klassen entkoppeln.  <div style="text-align: center;">  </div>	Ich kann für ein gegebenes Interface eine Stub-Klasse implementieren, welche beim Aufruf der vom Interface definierten Methoden fest programmierte Resultate liefert.	Ich kann eine Stub-Klasse in einem Unit-Test einsetzen und so die zu testende Klasse unabhängig von anderen Klassen testen.	Ich kann fortgeschrittene StubKlassen implementieren, welche einen Zustand haben. Dieser wird durch die aufgerufenen Methoden verändert und kann im Unit-Test überprüft werden.	Ich kann in meinen Unit-Tests eine Mocking-Bibliothek, wie moq oder Rhino Mocks, einsetzen und bei damit erzeugten Mock-Objekten die Einhaltung von Testbedingungen überprüfen.

Massstab: Alle Kompetenzen links der roten Linie erfüllt: Note 4.0. Pro fehlender / zusätzlicher Kompetenz - / + 0.25