Lernzettel

Objektorientierte Analyse, Design und Entwurfsmuster

Universität: Technische Universität Berlin
Kurs/Modul: Softwaretechnik und Programmierparadigmen
Erstellungsdatum: September 19, 2025

Lernzettel: Objektorientierte Analyse, Design und Entwurfsmuster

(1) Grundlagen der OOA und OOD.
Objektorientierte Analyse (OOA) dient der domänenbezogenen Erfassung von Objekten, ihren Eigenschaften und Beziehungen, um Anforderungen in Modelle abzubilden.
Objektorientierter Entwurf (OOD) überführt diese Modelle in eine umsetzbare Softwarestruktur durch Klassen, Schnittstellen und Entwurfsmuster.
Kernkonzepte: Objekt, Klasse, Instanz; Kapselung (Information Hiding); Abstraktion; Vererbung; Polymorphie; Schnittstellen; Delegation; Komposition, Aggregation, Assoziationen; Diagrammnotationen (z. B. UML).
Ziel: klare, modulare Strukturen, einfache Erweiterbarkeit und gute Testbarkeit ermöglichen.

(2) Analyse- versus Designprozess.
Analysephase: Domänenverständnis, Ubiquitous Language, Use Cases, Domänenmodell.
Designphase: Klassen-/Objektendesign, Grenzobjekte (Boundary Objects), Entitäten (Entities), Steuerobjekte (Controllers), Mapper/Repository; Festlegung von Schnittstellen und Verantwortlichkeiten.
Modellierung: Klassendiagramme, Sequenzdiagramme, Verhaltensdiagramme; Iteratives, inkrementelles Vorgehen mit schrittweiser Verfeinerung.
Qualitätsziele: Wartbarkeit, Erweiterbarkeit, Testbarkeit, Nachvollziehbarkeit des Designs.

(3) OOA vs OOD – Überblick.
OOA fokussiert auf das Abbilden der fachlichen Domäne in Objekte und deren Beziehungen;
OOD übersetzt diese Modelle in konkrete Strukturen (Klassen, Schnittstellen) und Entwurfsmuster, die in der Implementierung genutzt werden.
Gleichzeitig helfen Entwurfsmuster, eventuelle Kollisionen von Anforderungen zu vermeiden und Wiederverwendung zu fördern.

(4) Entwurfsmuster – Kategorien.
Muster werden grob in drei Kategorien eingeteilt:
- Erzeugungsmuster (Factory Method, Abstract Factory, Builder, Prototype) – schaffen Objekte.
- Strukturmuster (Adapter, Facade, Composite, Decorator, Proxy, Bridge) – definieren Beziehungen und Strukturen.
- Verhaltensmuster (Observer, Strategy, Command, State, Visitor, Mediator, Template Method) – regeln das Verhalten von Objekten und deren Kommunikation.

(5) Wichtige Muster – kurze Beschreibungen.
- Factory Method: Erzeugt Objekte über eine Schnittstelle, lässt Unterklassen entscheiden, welche Klasse instanziiert wird.
- Abstract Factory: Erzeugt Familien verwandter Objekte, ohne deren konkrete Klassen festzulegen.
- Builder: Schrittweises Erzeugen komplexer Objekte; trennt Aufbau von Repräsentation.
- Singleton: Gewährleistet, dass eine Klasse genau eine Instanz hat und global zugänglich ist.
- Adapter: Wandelt Schnittstelle einer Klasse in eine gewünschte Schnittstelle um.
- Facade: Stellt eine einfache API für ein komplexes Subsystem bereit.
- Decorator: Dynamisch zusätzliche Verantwortlichkeiten zu Objekten hinzufügen.

- Proxy: Stellvertreterobjekt, das Zugriff steuert oder vorgelagerte Logik kapselt.
- Composite: Behandlung einzelner Objekte und Objektgruppen als eine Einheit.
- Observer: Eine Änderung eines Objekts benachrichtigt abhängige Objekte automatisch.
- Strategy: Wahl eines Algorithmus zur Laufzeit, austauschbar, ohne Klienten zu ändern.
- Command: Ausführung von Operationen als Objekte; ermöglicht Undo/Redo, Queueing.
- State: Objekt verändert sein Verhalten, wenn sich sein interner Zustand ändert.
- Bridge: Trennung von Abstraktion und Implementierung, um beides unabhängig zu entwickeln.
- MVC: Architekturtrennung von Modell, View und Controller (zentrale Orientierung an UI-Systemen).

(6) SOLID-Prinzipien – Kurzüberblick.
- Single Responsibility Principle (SRP): Eine Klasse hat eine einzige Verantwortlichkeit.
- Open/Closed Principle (OCP): Softwarekomponenten sollen offen für Erweiterung, but geschlossen für Veränderung sein.
- Liskov Substitution Principle (LSP): Unterklassen sollen überall dort verwendbar sein, wo Basisklassen verwendet werden.
- Interface Segregation Principle (ISP): Viele spezialisierte Schnittstellen statt einer großen.
- Dependency Inversion Principle (DIP): Abhängigkeiten von Abstraktionen, nicht von konkreten Implementierungen.

(7) UML-Grundlagen in OOA/OOD.
- Klassen- und Objektdiagramme: Klassenname, Attribute, Operationen; Beziehungen (Assoziationen, Aggregationen, Komposition, Vererbung).
- Sequenzdiagramme: Akteure, Lebenslinien, Nachrichtenfluss, Zeitordnung.
- Aktivitätsdiagramme: Ablaufprozesse, Entscheidungen, Parallelität.
Nutze UML als gemeinsame Sprache, um Domänen- und Entwurfskonzepte zu kommunizieren.

(8) Qualitätsmerkmale und Messung.
- Kopplung vs. Kohäsion: geringe Kopplung, hohe Kohäsion erhöhen Wartbarkeit.
- Änderungsaufwand, Erweiterbarkeit, Testbarkeit, Verständlichkeit.
- Metriken: z. B. Kopplungsgrad, Kohäsionsgrad, Zyklo-Index, Wartbarkeitsindex; Einsatz von Tests (Unit-Tests), statische Analyse, Hoare-Kalkül als theoretischer Beleg für Korrektheit in Teilsystemen.
- Design Patterns tragen zur Qualität durch Wiederverwendbarkeit und Verständlichkeit bei.

(9) Beispielhafter Einsatz eines Musterflowers.
Stellen Sie sich eine Software vor, die Logging, Konfiguration und Benachrichtigungen koordiniert.
- Eine Factory erzeugt Logger-Objekte unterschiedlicher Typen (Datei, Konsole, Netzwerk) anhand der Konfiguration.
- Ein Observer-Muster ermöglicht, dass Komponenten auf Ereignisse wie Fehler oder Warnungen reagieren, ohne eng gekoppelt zu sein.
- Das Strategy-Muster erlaubt das Austauschen von Algorithmen zur Formatierung oder Ausgabe der Logs zur Laufzeit.
Dieses Beispiel illustriert die Trennung von Verantwortlichkeiten, Wiederverwendbarkeit und Erweiterbarkeit durch Muster.