Keanu Pastuschka

Objektorientierte Programmierung zusammenfassung 2024

Inhaltsverzeichnis

[**1. Versionierung** 1](#_Toc157286561)

[1.1 Was ist die Versionierung ? 1](#_Toc157286562)

[2.2 Was sagt eine Versionsnummer aus ? 2](#_Toc157286563)

[1.2 Lokale und Zentrale und verteilte Versionsverwaltung 2](#_Toc157286564)

[1.3 Was ist das Sandbox-Prinzip ? 2](#_Toc157286565)

[1.4 Was ist eine Revision ? 2](#_Toc157286566)

[1.5 Optimistic Locking & Pessimisitc Locking 2](#_Toc157286567)

[1.6 Was ist einchecken und auschecken ? 2](#_Toc157286568)

[1.7 Beispiel zu Auslieferung von Softwareversionen 3](#_Toc157286569)

[1.8 Beispielbranches aus der Vorlesung 3](#_Toc157286570)

[1.9 Git-Befehle 3](#_Toc157286571)

[1.10 Vor- und Nachteile einer Versionsverwaltung 4](#_Toc157286572)

[**2. Model View Controller ( MVC )** 4](#_Toc157286573)

[2.1 Was ist MVC ? 4](#_Toc157286574)

[2.2 Trennungsdefinition von MVC 5](#_Toc157286575)

[2.3 Reinfolge der Abrufe des MVC 5](#_Toc157286576)

[2.4 Klassendiagramm-Beispiele für einem MCV 6](#_Toc157286577)

[**3. Fabrik-Methode** 7](#_Toc157286578)

[3.1 Was ist die Fabrik-Methode ? 7](#_Toc157286579)

[3.2 Codebeispiel 7](#_Toc157286580)

[3.3 Vor- und Nachteile der Fabrikmethode 8](#_Toc157286581)

[3.4 Weitere Methoden um die Fabrik-Methode auszubauen ( Klassendiagramm ) 9](#_Toc157286582)

[4. Singleton 10](#_Toc157286583)

[4.1 Was ist Singleton 10](#_Toc157286584)

[4.2 Wie sieht die Struktur als Klassendiagramm aus ? 10](#_Toc157286585)

[4.3 Codebeispiel 10](#_Toc157286586)

# **1. Versionierung**

## 1.1 Was ist die Versionierung ?

Versionierung ermöglicht es Teams von Entwicklern oder Mitarbeitern, gemeinsam an Projekten zu arbeiten, Änderungen zu verfolgen, Konflikte zu lösen und die Integrität von Dateien oder Software über verschiedene Iterationen hinweg zu erhalten. Sie bietet auch eine Möglichkeit, auf frühere Versionen zurückzugreifen oder bestimmte Änderungen rückgängig zu machen, falls erforderlich.

## 2.2 Was sagt eine Versionsnummer aus ?

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

## 1.2 Lokale und Zentrale und verteilte Versionsverwaltung

Lokal

* Es wird nur eine Datei versioniert
* Die Versionen eines Dokuments werden in der Datei des dokuemtsn selbst gespeichert
* Lokale Versionierung war damals die erste Art der Versionierung, aber für Softwareprojekte ungeeinget.

Zentrale und verteile Versionsverwaltung

* Wird als Client-Server-System aufgebaut
* Durch Rechteverwaltung wird dafür gesorgt, dass nur gewisse Personen zugreifen können
* Die Versionsgeschichte ist nur im Repository vorhanden

## 1.3 Was ist das Sandbox-Prinzip ?

* Hier wird eine Kopie auf das lokale Repository ausgecheckt und damit wird dann lokal gearbeitet.
* Alle Änderungen bleiben lokal, bis sie wieder eingecheckt werden.

## 1.4 Was ist eine Revision ?

Beim Einchecken wird eine neue Revision erzeugt: Revisionsnummer, Zeitstempel .. wird gespeichert.

## 1.5 Optimistic Locking & Pessimisitc Locking

Optimistic Locking

* Auschecken einer Datei ist immer möglich, beim Einchecken muss ggf. mit zwischenzeitlich erfolgten Veränderungen abgeglichen werden.

Pessimistic Locking

* Entwickler kann Datei nur auschecken, wenn kein anderer Entwickler die Datei ausgechecked hat.

## 1.6 Was ist einchecken und auschecken ?

* Einchecken = Hochladen
* Auschecken = Runterladen

## 1.7 Beispiel zu Auslieferung von Softwareversionen

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

## 1.8 Beispielbranches aus der Vorlesung

Ein Bild, das Diagramm, Reihe, Screenshot, Text enthält.

Automatisch generierte Beschreibung

## 1.9 Git-Befehle

* **Push:** "Push" bezieht sich auf das Hochladen lokaler Änderungen aus deinem Arbeitsverzeichnis auf einen entfernten Repository-Server wie GitHub. Du verwendest "push", um deine lokalen Commits mit anderen zu teilen.
* **Pull:** "Pull" bezieht sich darauf, dass du die Änderungen von einem entfernten Repository auf deinen lokalen Arbeitsbereich herunterlädst. Dies ermöglicht es dir, die neuesten Änderungen von anderen Entwicklern zu erhalten und sie in deinem lokalen Repository zu integrieren.
* **Clone:** "Clone" bezieht sich darauf, dass du eine Kopie eines Repositorys von einem entfernten Server wie GitHub auf deinen lokalen Computer herunterlädst. Dadurch erhältst du eine vollständige Kopie des Repositorys, einschließlich aller Branches und Commits.
* **Commit:** "Commit" bezieht sich auf das Speichern von Änderungen an deinem lokalen Repository. Mit einem Commit bestätigst du deine Änderungen und fügst sie zur Versionsgeschichte deines Projekts hinzu.

## 1.10 Vor- und Nachteile einer Versionsverwaltung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

# **2. Model View Controller ( MVC )**

## 2.1 Was ist MVC ?

* MVC ist ein klassenbasiertes Strukturmuster.
* MVC wird auch als Architektur Pattern eingesetzt.

## 2.2 Trennungsdefinition von MVC

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

## 2.3 Reinfolge der Abrufe des MVC

Ein Bild, das Text, Diagramm, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

## 2.4 Klassendiagramm-Beispiele für einem MCV

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

# **3. Fabrik-Methode**

## 3.1 Was ist die Fabrik-Methode ?

Die Fabrikmethode (Factory Method) ist ein Entwurfsmuster aus dem Bereich der Softwareentwicklung. Es gehört zur Kategorie der sogenannten "Erzeugungsmuster" (Creational Patterns) und wird verwendet, um die Erzeugung von Objekten zu kapseln.

## 3.2 Codebeispiel

// Produkt

**interface** Product {

**void** operation();

}

// Konkretes Produkt

**class** ConcreteProduct **implements** Product {

@Override

**public** **void** operation() {

System.***out***.println("Operation in ConcreteProduct ausgeführt.");

}

}

// Erzeuger (abstrakte Klasse)

**abstract** **class** Creator {

**abstract** Product factoryMethod();

}

// Konkreter Erzeuger

**class** ConcreteCreator **extends** Creator {

@Override

Product factoryMethod() {

**return** **new** ConcreteProduct();

}

}

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Creator creator = **new** ConcreteCreator();

Product product = creator.factoryMethod();

product.operation();

}

}

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

Automatisch generierte Beschreibung

## 3.3 Vor- und Nachteile der Fabrikmethode

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

## 3.4 Weitere Methoden um die Fabrik-Methode auszubauen ( Klassendiagramm )

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

# 

# **4. Singleton**

## 4.1 Was ist Singleton

Singleton stellt sicher, dass nur genau eine Instanz einer Klasse erzeugt wird.

## 4.2 Wie sieht die Struktur als Klassendiagramm aus ?

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

## 4.3 Codebeispiel

**public** **class** Singleton {

// Die einzige Instanz der Klasse

**private** **static** Singleton *instance*;

// Private Konstruktor, um die Instanziierung von außerhalb zu verhindern

**private** Singleton() {

// Hier können Initialisierungen durchgeführt werden

}

// Methode, um die Instanz der Klasse zu erhalten

**public** **static** Singleton getInstance() {

// Wenn die Instanz noch nicht erstellt wurde, erstelle sie

**if** (*instance* == **null**) {

*instance* = **new** Singleton();

}

// Gib die einzige Instanz zurück

**return** *instance*;

}

// Beispiel Methode der Singleton-Klasse

**public** **void** showMessage() {

System.***out***.println("Hallo! Ich bin eine Singleton-Klasse.");

}

**public** **static** **void** main(String[] args) {

// Beispiel zur Verwendung der Singleton-Klasse

Singleton singletonInstance = Singleton.*getInstance*();

singletonInstance.showMessage();

}

}