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

Automatisch generierte Beschreibung

**Entwurf und Implementierung  
Adresskonverter**

Studiengang  
Informatik / Informationstechnik  
An der Dualen Hochschule Baden-Württemberg Stuttgart  
Lent Blattner, Gregori Daiger, Panagiotis Fotiadis

Abgabe von: Gregori Daiger Lenz Blattner Panagiotis Fotiadis

Matrikelnummer: 2395489 5611072 5962782

Inhalt

[**1. Einleitung** 3](#_Toc137159818)

[**2. Installation, Setup und Ausführung des Java-Source-Codes** 3](#_Toc137159819)

[**2.1 Installation und Setup** 3](#_Toc137159820)

[**2.2 Richtige Erstellung der Konfigurationsdatei** 5](#_Toc137159821)

[**3.** Konzeptionelle Entscheidungen und Projektstruktur 6](#_Toc137159822)

[**3.1 Pipes and Filters Architektur** 6](#_Toc137159823)

[**3.2 FilterFactory und Reflection** 7](#_Toc137159824)

[**3.3 Erweiterbarkeit und Modularität** 7](#_Toc137159825)

# **1. Einleitung**

Das Ziel dieses Java-Projekts ist es, eine konfigurierbare Anwendung zu erstellen, die in der Lage ist, Adressen von Personen aus Dateien verschiedener Formate zu lesen (JSON, XML und YAML), sie zu verarbeiten und das Ergebnis in einer Datei des gewünschten Formats zu speichern. Mithilfe der Anwendung können Listen von Personen in den oben genannten Formaten durch Konverter von einem Eingangs-Datei-Format in ein Ausgangs-Datei-Format transformiert werden oder bestimmte Teile der Datei extrahiert werden. Durch die Formatierungs-Modules werden die Werte hinter den Eigenschaften verändert, wobei die Struktur unverändert bleibt.   
Beim Entwurf und der Implementierung der Java-Applikation wurde besonderes Augenmerk auf die Aspekte Modularität, Erweiterbarkeit und Wartbarkeit gelegt, sodass es möglich ist einfach und ohne Veränderung des Codes weitere Konverter- und Formatierungs-Module hinzuzufügen.

Um diese Ziele zu erreichen, wurde das Architekturmuster "Pipes and Filters" angewendet. Durch die Verwendung dieses Musters ist die Anwendung in der Lage, verschiedene Operationen (oder "Filter") auf die Daten anzuwenden und das Ergebnis durch "Pipes" an die nächste Operation weiterzuleiten. Jeder Filter ist eine unabhängige Einheit, die ihre Aufgabe unabhängig von den anderen Filtern erfüllt, was zu einer hohen Modularität und Erweiterbarkeit der Anwendung führt.

Die Implementierung der Anwendung basiert auf Java und verwendet eine Reihe von Bibliotheken, darunter Jackson für die Arbeit mit JSON, XML und YAML-Dateien und JUnit für Tests.

# **2. Installation, Setup und Ausführung des Java-Source-Codes**

## **2.1 Installation und Setup**

In dieser kurzen Anleitung wird erklärt, wie der Source-Code des Projekts „SWEProjekTGit“ aus

Nach dem Herunterladen des Projekts aus der Abgabe, muss im ersten Schritt im Terminal oder in der Kommandozeile zu dem Verzeichnis, das das Projekt enthält, navigiert werden.  
  
Um das Projekt zu kompilieren und auszuführen, ist die Verwendung von Gradle erforderlich, einem leistungsstarken Build-Tool, das für das Bauen, Testen und Verwalten von Softwareprojekten verwendet wird. Das Projekt selbst enthält bereits eine Gradle-Wrapper-Datei (gradlew oder gradlew.bat), die die spezifische Version von Gradle automatisch herunterlädt und installiert. Daher ist keine separate Gradle-Installation erforderlich.

Für die Ausführung des Projekts im Terminal muss im ersten Schritt zu dem Verzeichnis navigiert werden, in dem das Projekt „SWEProjekTGit“ sich befindet

Um das Projekt zu bauen, muss der Befehl **./gradlew build** in der Befehlszeile oder im Terminal ausgeführt werden. Dieser Befehl kompiliert den Quellcode, führt Tests aus und erstellt eine JAR-Datei, die das kompilierte Programm enthält.

In diesem Projekt wird das Gradle-Plugin **shadowJar** verwendet, um eine "fette" JAR-Datei zu erstellen. Dies bezeichnet eine JAR-Datei, die neben dem eigentlichen Programmcode auch alle seine Abhängigkeiten enthält. So wird sichergestellt, dass das Programm auf jedem System, das eine Java-Laufzeitumgebung besitzt, ausgeführt werden kann, ohne dass zusätzliche Bibliotheken installiert werden müssen.

Nachdem das Projekt erfolgreich mit **./gradlew build** gebaut wurde, befindet sich die resultierende JAR-Datei im Verzeichnis **build/libs**.

Um das Programm zu starten, muss der folgende Befehl ausgeführt werden:

**java -jar build/libs/SWEProjekTGit-1.0-SNAPSHOT-all.jar C:\Pfad\Zur\Configdatei\config.txt**

Dieser Befehl gibt an, dass eine JAR-Datei ausgeführt werden soll (**java -jar**), gibt den Pfad zur JAR-Datei an (**build/libs/SWEProjekTGit-1.0-SNAPSHOT-all.jar**) und übergibt den Pfad zur Konfigurationsdatei (**C:\Pfad\Zur\Configdatei\config.txt**) als Argument „args“ an das Programm.

Es ist wichtig zu beachten, dass die Pfadangaben in der Konfigurationsdatei und im Befehl zum Ausführen des Programms entsprechend der tatsächlichen Dateistruktur angepasst werden müssen.

Zusätzlich zur Bereitstellung des Quellcodes kann die erstellte JAR-Datei zusammen mit einer Beispielkonfigurationsdatei und einer Beispieleingabedatei zur Verfügung gestellt werden, um die Ausführung des Programms zu erleichtern.

Die **build.gradle.kts** Datei definiert die Abhängigkeiten des Projekts und die Konfiguration für das Build-System. Die Abhängigkeiten werden automatisch von Gradle verwaltet und bei Bedarf heruntergeladen. Die wichtigsten Abhängigkeiten sind:

* **junit-jupiter** für Unit-Tests
* **jackson-databind**, **jackson-dataformat-xml** und **jackson-dataformat-yaml** für die Arbeit mit JSON, XML und YAML.

Um das Projekt zu bauen, muss im Terminal der Befehl **gradle build** ausgeführt werden. Gradle kompiliert den Code, führt Tests aus und erstellt ein ausführbares JAR, das dann ausgeführt werden können.

Um das erstellte JAR auszuführen, kann der Befehl **java -jar** gefolgt vom Namen des JARs. Das JAR enthält alle Abhängigkeiten und kann auf jedem System mit einer kompatiblen Java-Laufzeitumgebung ausgeführt werden.

Die Konfiguration der Anwendung erfolgt durch eine Konfigurationsdatei im txt.Format, diese wird dann in der Main als Argument übergeben. Diese Konfigurationsfile enthält dann Informationen darüber, welche Filter angewendet werden sollen (Reihenfolge in richtiger Reihenfolge angeben) und enthält den Pfad zum File, auf das dann der erstgenannte Filter angewendet wird.

## **2.2 Anleitung zur Ersellung der Konfigurationsdatei**

Um das Programm zu konfigurieren, wird eine Konfigurationsdatei benötigt. Die Konfigurationsdatei ist eine Textdatei, die bestimmte Schlüssel-Wert-Paare enthält, die das Verhalten des Programms steuern.

Es gibt zwei wichtige Einträge in der Konfigurationsdatei:

* **StartFile**; Hier sollte der Pfad zur zu verarbeitenden Eingabedatei eingetragen werden. Beispielsweise könnte dies **C:\Pfad\zur\Konfigurationsdatei\personen.json** sein, wenn sich die Datei "personen.json" im angegebenen Verzeichnis befindet. Es ist wichtig, dass dieser Pfad korrekt ist und dass die Datei existiert und vom Benutzer gelesen werden kann, der das Programm ausführt. Außerdem muss darauf geachtet werden die Dateiendung klein zu schreiben also “json”, “xml” statt “Json”, “Xml”.
* **Filters**: Dieser Eintrag listet die Filter auf, die in der Pipeline verwendet werden sollen. Filter sind bestimmte Operationen, die auf die Eingabedatei angewendet werden. Sie werden in der Reihenfolge angegeben, in der sie angewendet werden sollen, getrennt durch Kommata.

Ein Beispiel für einen vollständigen **Filters**-Eintrag könnte folgendermaßen aussehen:

Filters: filter.converter.JsonToJsonEmailConverter, filter.converter.JsonToXmlConverter, filter.formatter.EncryptFormatter, filter.formatter.DecryptFormatter

In diesem Beispiel werden vier Filter verwendet:

1. **JsonToJsonEmailConverter**: Konvertiert eine JSON-Datei, die eine Liste von Personen enthält, in eine JSON-Datei, die nur die E-Mail-Adressen dieser Personen enthält.
2. **JsonToXmlConverter**: Konvertiert eine JSON-Datei in eine XML-Datei.
3. **EncryptFormatter**: Verschlüsselt den Inhalt der Datei.
4. **DecryptFormatter**: Entschlüsselt den Inhalt der Datei.

Diese Filter werden in der angegebenen Reihenfolge angewendet. Es ist möglich, beliebig viele Filter anzugeben, solange sie durch Kommata getrennt sind und der vollqualifizierte Klassenname verwendet wird (also der Paketname und der Klassenname, getrennt durch Punkte).

Es ist zu beachten, dass der genaue Pfad und die genauen Filteranweisungen für die individuellen Anforderungen angepasst werden müssen. Es ist wichtig, dass die Filter in einer Reihenfolge angegeben werden, die Sinn ergibt (zum Beispiel wäre es nicht sinnvoll, zuerst zu entschlüsseln und dann zu verschlüsseln), ansonsten werden Fehler bei der Ausführung auftreten, die durch entsprechende Exceptions abgefangen werden.

Mit diesem Setup ist man in der Lage, das Projekt zu bauen und auszuführen. Der nächste Abschnitt der Dokumentation wird sich mit den Konzepten und der Architektur der Anwendung befassen, gefolgt von einer detaillierten Beschreibung der einzelnen Komponenten und ihrer Interaktionen.

# **3.** Konzeptionelle Entscheidungen und Projektstruktur

In diesem Abschnitt wird erläutert, warum bestimmte Konzepte gewählt wurden und wie das Projekt strukturiert ist.

## **3.1 Pipes and Filters Architektur**

Die Architektur "Pipes and Filters" wurde für dieses Projekt ausgewählt, weil sie eine natürliche Wahl für Anwendungen ist, die auf die Verarbeitung von Datenströmen ausgerichtet sind. Sie ermöglicht es, komplexe Aufgaben durch die Kombination einfacher, unabhängiger Komponenten zu bewältigen.

Die Hauptkomponenten in einem Pipes-and-Filters-System sind die Filter, die auf die Daten operieren, und die Pipes, die den Datenfluss zwischen den Filtern ermöglichen. In unserem Fall sind die Filter Java-Klassen, die eine bestimmte Operation auf einer Datei durchführen, wie zum Beispiel Konvertieren, Formatieren oder Verschlüsseln. Die Pipes sind in diesem Kontext metaphorisch und repräsentieren die Reihenfolge, in der die Filter auf die Daten angewendet werden.

Die Hauptklasse **Application** liest eine Konfigurationsdatei und erzeugt auf der Grundlage dieser Konfigurationsdatei eine Liste von Filtern. Anschließend wird eine neue Pipeline erstellt und die Filter werden dieser Pipeline hinzugefügt. Das Ausführen der Pipeline entspricht dann dem Durchlaufen der Filter in der angegebenen Reihenfolge.

Die Verwendung der Pipes-and-Filters-Architektur hat mehrere Vorteile. Sie erlaubt es, die Verarbeitung von Daten auf hoher Ebene zu steuern, ohne sich um die Details der einzelnen Operationen kümmern zu müssen. Sie bietet auch eine hohe Flexibilität, da verschiedene Filter einfach hinzugefügt, entfernt oder neu angeordnet werden können, um unterschiedliche Ergebnisse zu erzielen.

## **3.2 FilterFactory und Reflection**

Eine Herausforderung bei der Implementierung der Pipes-and-Filters-Architektur war die Erzeugung von Filter-Instanzen basierend auf den in der Konfigurationsdatei angegebenen Klassennamen. Dies wurde durch die Verwendung der Entwurfsmethode Factory in Verbindung mit der Java-Reflection-API gelöst.

Die Factory-Methode ist ein Entwurfsmuster, das zur Erzeugung von Objekten verwendet wird, ohne dass die genaue Klasse der zu erzeugenden Objekte im Voraus bekannt sein muss. In unserem Fall wird eine Factory-Methode in der **FilterFactory**-Klasse verwendet, um Filter-Objekte zu erzeugen.

Die Java-Reflection-API ermöglicht es, zur Laufzeit auf Informationen über Klassen und Objekte zuzugreifen. In diesem Fall wird sie verwendet, um eine Instanz einer Filter-Klasse basierend auf ihrem Klassennamen zu erzeugen.

Die Methode **createFiltersFromConfig** in der **FilterFactory**-Klasse liest die Namen der Filterklassen aus der Konfigurationsdatei, lädt die entsprechenden Klassen zur Laufzeit mit Hilfe der Reflection-API und erzeugt dann Instanzen dieser Klassen.

Die Verwendung der Factory-Methode in Kombination mit Reflection erlaubt eine hohe Flexibilität bei der Definition der Pipeline. Neue Filter können einfach hinzugefügt werden, indem die entsprechende Klasse implementiert und ihr Name in der Konfigurationsdatei angegeben wird, ohne dass der Hauptcode geändert werden muss.

## **3.3 Erweiterbarkeit und Modularität**

Eine der Schlüsselentscheidungen bei der Gestaltung dieses Projekts war, es von Anfang an so zu konzipieren, dass es leicht erweiterbar und modular ist. Das bedeutet, dass neue Filter und Funktionen problemlos hinzugefügt werden können, ohne bestehende Code-Teile zu beeinträchtigen.

Durch die Verwendung des Pipes-and-Filters-Musters, zusammen mit der Factory-Methode und Reflection, konnte ein System erstellt werden, das leicht erweitert und angepasst werden kann. Ein neuer Filter kann einfach als neue Klasse hinzugefügt und in der Konfigurationsdatei referenziert werden. Das System wird dann automatisch diesen neuen Filter in die Datenverarbeitungspipeline aufnehmen. Wie man weitere Filter-Module hinzufügen kann wir in einem späteren Abschnitt erläutert.

Diese Flexibilität erleichtert die Anpassung an neue Anforderungen und die Einbindung neuer Funktionen, was dieses Projekt zu einer robusten und zukunftssicheren Lösung macht.

Formularbeginn

# **4. Projektarchitektur und Design: Diagramme**

## **4.1 Klassendiagramm**

Folgendes Klassendiagramm zeigt die wichtigsten Klassen und die wichtigsten Interaktionen, sowie nach außen bereitgestellte Methoden.  
Formularbeginn

**HINWEIS: Die Klassen aus dem addressmodell package, die als Klassen dienen, um die Personen und deren Adressen in Java Objekte abzubilden, ist in diesem Diagramm nicht integriert. Ein eigenes Diagramm zeigt die Struktur des addressmodell und die Methoden der Klassen.**

**Ein Bild, das Text, Screenshot, Diagramm, Software enthält.

Automatisch generierte Beschreibung**

Abbildung 1: Klassendiagramm: addressmodell (Java-Klassen für Adressendarstellung)

**Ein Bild, das Text, Screenshot, Diagramm, Plan enthält.

Automatisch generierte Beschreibung**

Abbildung 2: Klassendiagramm des Projekts

## **4.2 Aktivitätsdiagramme**

Folgendes Aktivitätsdiagramm, zeigt vereinfacht die Aktivitäten und Fälle die auftreten können, wenn der JsonToXmlConverter ausgeführt wird:

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

Automatisch generierte Beschreibung

Abbildung 3: Aktivitätsdiagramm zu JsonToXmlConverter

Das Diagramm zeigt den JsonToXmlConverter. Es beginnt mit dem Start der **process()** Methode. Dann wird die Eingabedatei gelesen. Wenn die Datei existiert, wird der Listentyp bestimmt und die JSON-Daten werden in Java-Objekte konvertiert. Danach wird der Ausgabedateipad generiert und die Informationen werden in die XML-Ausgabedatei geschrieben. Schließlich wird der Ausgabedateipfad zurückgegeben.Wenn die Datei nicht existiert ,wird eine IOException ausgelöst.

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

Automatisch generierte Beschreibung

Das Diagramm zeigt den Ablauf des Hauptprogramms Application. Es beginnt mit dem Start der **main()** Methode. Dann wird überprüft, ob Argumente vorhanden sind. Wenn Argumente vorhanden sind, wird der Pfad zur Konfigurationsdatei gesetzt, Filter aus der Konfigurationsdatei erstellt, eine Pipeline erstellt und die Filter hinzugefügt. Anschließend wird die Konfigurationsdatei gelesen und der Pfad zur Startdatei gefunden. Wenn ein Startdateipfad gefunden wird, wird die Pipeline auf die Datei angewendet und das Ergebnis ausgegeben. Wenn kein Startdateipfad gefunden wird, wird eine Fehlermeldung ausgegeben. Wenn keine Argumente vorhanden sind, wird eine Fehlermeldung ausgegeben und das Programm beendet.

1. Hinzufügen weitere Konverter oder Formatter.

Erweiterung der Anwendung um zusätzliche Converter und Filter

Unsere Anwendung unterstützt die Erweiterung durch zusätzliche Converter und Filter. Sie können eigene Dateiformatkonverter und -filter in dem Verzeichnis src/main/java/filter/external hinzufügen. Hier sind die Schritte, die Sie befolgen sollten, um diese Funktionalität zu nutzen:

1. Erstellen Sie einen neuen Converter oder Filter:

1.1. Erstellen Sie eine neue Klasse im Verzeichnis src/main/java/filter/external. Diese Klasse sollte die Filter-Schnittstelle implementieren.

1.2. In der Klasse sollten Sie die process Methode überschreiben, die einen Dateipfad als Eingabe annimmt und den Pfad zur bearbeiteten Ausgabedatei zurückgibt.

1.3. In der process Methode können Sie Utils-Klasse verwenden, um die Datei zu lesen, die Daten zu verarbeiten und die Ergebnisse in eine neue Datei zu schreiben.

1.4. Achten Sie darauf, dass Ihre Klasse alle benötigten Ausnahmen richtig behandelt.

2. Hinzufügen zum Projekt:

Nachdem Sie Ihre Converter- oder Filterklasse erstellt haben, können Sie sie zu Ihrem Projekt hinzufügen und sie wie gewünscht verwenden.

Beachten Sie bitte, dass die neuen Converter oder Filter auf die gleiche Weise funktionieren sollten wie die vorhandenen. Sie sollten die Methode process überschreiben und sorgfältig testen, um sicherzustellen, dass sie korrekt funktioniert.

Bitte beachten Sie, dass alle erstellten Klassen den im Projekt definierten Codierungsstandards entsprechen sollten. Falls Sie Hilfe benötigen, steht Ihnen die Dokumentation zur Verfügung oder Sie können uns direkt kontaktieren.

3. Falls noch andere Dateiformate unterstützt werden sollen:  
3.1 Erstellen Sie ein neues Utils:

3.2. Erstellen Sie eine neue Utils-Klasse für den gewünschten Dateityp im Verzeichnis src/main/java/utils. Diese Klasse sollte die IUtil-Schnittstelle implementieren.

1.2. Die Utils-Klasse sollte Methoden enthalten, um Daten aus einer Datei in Java-Objekte umzuwandeln (toJava) und um Java-Objekte in eine Datei zu schreiben (fromJava).

1.3. Sie können die vorhandenen Utils (JsonUtils, XmlUtils, YamlUtils) als Beispiele verwenden.