Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

InIT Institut für Angewandte Informations-technologie

Lernaufgabe

V4 – Framework Design

Lernziel

Sie sind in der Lage, die Prinzipen des Entwurfs von Frameworks und der Anwendung von Frameworks zu erläutern und in eigenen Applikationen einzusetzen.

Einleitung

Sie möchten ein Framework entwickeln, das Ihnen langweilige Programmierarbeit abnimmt. Ihre Idee ist es, dass Anwendungsentwickler anstelle von Klassen nur Interfaces deklarieren und dass dann Ihr Framework automatisch die Klassen dazu erstellt.

Zuerst sollen die Attribute in Klassen automatisch erzeugt werden, gesteuert über die Namen der Interface Methoden. Wenn Ihr Framework eine getter- oder setter-Methode in der Interface Deklaration entdeckt, dann soll in der automatisch erzeugten implementierenden Klasse ein dazu passendes Attribut erzeugt werden. Interface-Default-Methoden sollen wie erwartet in die Klasse übernommen werden. Zudem soll es möglich sein, dass eine leere Interface-Methode auf eine statische Methode delegiert.

Weitere Features sind denkbar, z.B. Support für PropertyChangeEvents oder für das Visitor-Pattern bis zu einer eigenen Persistenz-Lösung.

Von der Präsentation - Folien 53 und 54 – inspiriert haben Sie die möglichen Java-Technologien dazu recherchiert:

- Es ist möglich, zur Laufzeit über Object.getClass() den Typ eines Objekts zu erhalten. Darauf sind dann alle Annotationen, Methoden und Attribute («Field») abfragbar. Dieser Bereich von Java wird auch «Reflection» genannt.
- Es gibt die Möglichkeit, den Compiler sozusagen zu erweitern. Dazu muss eine Klasse geschrieben werden, die das Interface javax.annotation.processing.Processor implementiert. So eine Klasse wird auch Annotation-Processor genannt. Ihre Entwicklungsumgebung hat die Möglichkeit, dass eine Jar-Datei mit dem Annotation-Processor dem Java-Compiler hinzugefügt werden kann. Dieser Annotation-Processor kann sich für gewisse Annotationen registrieren und wird dann vom Java-Compiler mit den Sprachelementen aufgerufen, die so annotiert wurden. Anschliessend kann er Fehlermeldungen ausgeben sowie Source- oder Bytecode erzeugen. Der Source-Code wird dann anschliessend vom Java-Compiler noch übersetzt. Die Sprachelemente, die der Annotation-Processor erhält, umfassen wie bei der Reflection fast alle verfügbaren Informationen, aber in einer anderen Form. Es sind dies Klassen aus der Package javax.lang.model und ihren Subpackages.
- Der Java-Compiler übersetzt den Quelltext in Bytecode. Bytecode kann mit Maschinencode für eine Stack-Maschine verglichen werden. Das Java JDK selbst hat keine Klassen, um Bytecode zu erzeugen, aber es gibt mehrere Bibliotheken, die eine mehr oder weniger komfortable Erzeugung von Bytecode ermöglichen.
- In der Standardklassenbibliothek gibt es die Klasse java.lang.reflect.Proxy, die es ermöglicht, zur Laufzeit ein beliebiges Interface zu implementieren. Das

resultierende Objekt leitet alle Methodenaufrufe auf ein java.lang.reflect.InvocationHandler Objekt weiter.

Aufgabe V4.1: Domänenmodell

Erstellen Sie das Domänenmodell für das in der Aufgabenstellung beschriebene Fachgebiet.

Vorgehen

1. Investieren Sie ein paar Minuten, um sich mit Hilfe der Javadoc und/oder anderen Beiträgen über die obigen Technologien kundig zu machen.
2. Identifizieren Sie Konzepte der Fachdomäne mit ihren Attributen.
3. Setzen Sie die Konzepte in Beziehung zueinander.
4. Zeichnen Sie ein (vereinfachtes) UML Klassendiagramm mit den Konzepten und Assoziationen.

Hinweise, Tipps

- Da das Fachgebiet technisch ist, werden Sie natürlich auch entsprechende Konzepte finden.
- Legen Sie besonderen Wert darauf, dass die verschiedenen Varianten der Untersuchung von Annotationen und die Varianten der Klassenerzeugung gut ersichtlich sind.

Ergebnis

Domänenmodell

Zeit: 30‘

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

InIT Institut für Angewandte Informations-technologie

Aufgabe V4.2: Analyse bestehender Code für das Framework

Ermutigt von den verfügbaren Technologien haben Sie die Anforderungen noch genauer definiert.

Sie legen fest, dass Interfaces, die automatisch implementiert werden, mit einer Annotation @AutoImpl markiert werden, das vom Framework zur Verfügung gestellt wird. Die getter- und setter-Methoden müssen nicht markiert werden, sondern werden nur über den Namen erkannt.

Interface-Default-Methoden sollen wie erwartet funktionieren und Interface-Methoden, die mit der Annotation @MethodDelegationTo markiert sind, delegieren zu der in den Annotations-Parameter definierten Methode. Die statische Methode erwartet als ersten zusätzlichen Parameter das aufrufende Objekt (vom Typ des annotierten Interface), damit in der statischen Methode die Objekt Methoden aufgerufen werden können.

In einer ersten Iteration haben Sie sich dazu entschieden, die Annotationen zur Laufzeit zu detektieren und mit dieser Information Klassen basierend auf java.lang.reflect.Proxy zu erzeugen. Natürlich haben Sie die Architektur so angelegt, dass die weiteren Varianten ohne grössere Umstände noch hinzugefügt werden können.

Ihre Arbeitskollegen sind an Ihrer Arbeit interessiert und bitten Sie, Ihnen die erstellte Software zu erklären. Dazu zeichnen Sie ein Package-Diagramm, das die Teilsysteme und Ihre Abhängigkeiten visualisiert und noch Klassen- und/oder Interaktionsdiagramme für interessante Bereiche, wo Sie bekannte Design Patterns eingesetzt haben.

Analysieren Sie das bestehende Framework. Auf dem ZHAW GitHub finden Sie das Repository: https://github.zhaw.ch/SWEN1-LP2019/SWEN1_V4_Framework-Design.

Vorgehen

1. Verschaffen Sie sich einen Überblick über die vorhandenen Packages und Klassen.
2. Führen Sie die Beispiele und Testfälle durch.
3. Finden Sie heraus, welche Packages zur Domänenlogik gehören und welche wiederverwendbar wären.
4. Identifizieren Sie die Abhängigkeiten der Packages.
5. Zeichnen Sie mit diesen Informationen ein Package-Diagramm.
6. Finden Sie die Anwendung von Design Patterns. Viele davon sind bereits im Klassennamen ersichtlich.

Hinweise, Tipps

- Bei diesem Framework gibt es zwar kein UI, aber dennoch kann Domänenlogik und unterstützender, potenziell wiederverwendbarer Code unterschieden werden.
- Die Abhängigkeiten der Packages sind nur über die darin enthaltenen Klassen ersichtlich.
- Design Patterns sind oft bereits im Klassennamen ersichtlich.

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

InIT Institut für Angewandte Informations-technologie

Ergebnis

Stichwortartiger Text und geeignete UML-Diagramme, die die Softwarearchitektur und den Einsatz von Design Patterns visualisieren.

Zeit: 30‘