Workshop:
Design Patterns
Übungen

Ablauf

Dieser Workshop gliedert sich in mehrere Vortragsteile, die den Teilnehmern die Thematik Design Patterns näherbringen. Im Anschluss daran sind jeweils einige Übungsaufgaben von den Teilnehmern – idealerweise in Gruppenarbeit – am Rechner zu lösen.

Voraussetzungen

1) Aktuelles JDK 11, idealerweise auch JDK 14/15, installiert
2) Aktuelles Eclipse installiert (Alternativ: NetBeans oder IntelliJ IDEA)

Teilnehmer

- Entwickler und Architekten mit Java-Erfahrung, die ihre Kenntnisse zu Design Patterns vertiefen möchten

Kursleitung und Kontakt

Michael Inden
Derzeit freiberuflicher Buchautor und Trainer
E-Mail: michael.inden@hotmail.com
Blog: https://jaxenter.de/author/minden

Weitere Kurse (Java, Unit Testing, ...) biete ich gerne auf Anfrage als Inhouse-Schulung an.

Design Patterns

PART 2: Erzeugungsmuster

Aufgabe 1: CREATE METHOD

Analysiere die Klasse `PizzaCreator`. Mit welchem Entwurfsmuster kann der Konstruktionsprozess vereinfacht werden? Wende dieses an.

Aufgabe 2: FACTORY METHOD

Analysiere die Klasse `GameCharacterGenerator`. Nutze zunächst eine Create Method und überlege mit welchem Entwurfsmuster der Konstruktionsprozess weiter abstrahiert werden kann. Wende dieses an.

Aufgabe 3: FACTORY METHOD

Es soll eine Klasse `WorkoutProposalGenerator`. Implementiert werden. Diese soll verschiedene Workouts, je nach Nutzerwunsch erstellen, etwa Übungsvorschläge für Ausdauer oder Unter- bzw. Oberkörper. Dazu ist bereits ein Interface `Workout` und ein enum `WorkoutType` definiert. Mit welchem Entwurfsmuster sollte der Konstruktionsprozess gestaltet werden? Wende dieses Muster an und passe die Implementierung entsprechend an.

Aufgabe 4: BUILDER

Analysiere die Klassen `ComputerExample` und `BankAccountExample`. Was sind die Schwierigkeiten bei den jeweiligen Objektkonstruktionen und wie kann man diese durch Einsatz des geeigneten Entwurfsmusters lösen? Wende das passende Muster an und passe die Implementierung entsprechend an.

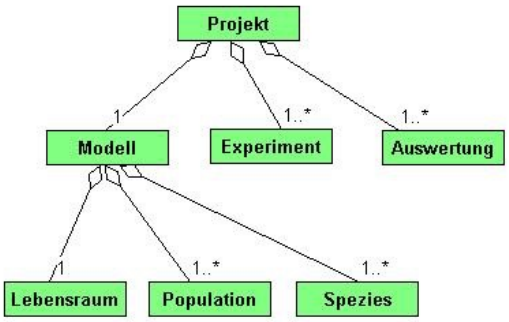
PART 3: Strukturmuster

Aufgabe 3: COMPOSITE 30 min

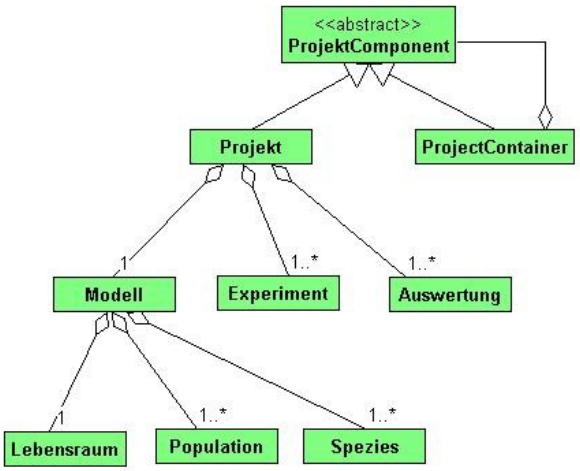
Modelliere eine Projektverwaltung mit UML:

1. Ein Projekt besteht aus einem Modell und mehreren Experimenten und Auswertungen. Modelle enthalten Populationen, Spezies und einen Lebensraum.
2. Ein Projekt kann beliebig viele Sub-Projekte enthalten! Verwende das Kompositum-Muster!

Composite – Beispiellösung Teil 1



Composite – Beispiellösung Teil 2



PART 4: Verhaltensmuster

Aufgabe 5: OBSERVER 15 min

Diskutieren Sie in 2-3er Gruppen, welche Vorteile und Nachteile sich aus der Push/Pull-Variante ergeben.

- Bei der Pull-Variante dient die update()-Methode lediglich als Hinweis, dass eine Änderung stattgefunden hat. Zur Ermittlung und zum Abgleich von Zustandsinformationen werden durch die jeweiligen Beobachter in der Regel diverse get()-Methoden zur Zustandsabfrage des ObservedSubjects aufgerufen. Dies bedingt allerdings, dass je- der Beobachter eine Rückreferenz auf das beobachtete Subjekt besitzt. Im Idealfall ist dies lediglich ein Interface, das aus einem Übergabeparameter der update()-Methode stammt und nicht im Beobachter gespeichert wird (vgl. Diskussion im Absatz »Mythos lose Kopplung«). Ein Vorteil des Pull-Verfahrens ist, dass der Zustand nur bei Bedarf durch den Beobachter auch tatsächlich nachgefragt wird. Für gewisse Darstellungen kann es bereits ausreichend sein, zu wissen, dass überhaupt eine Änderung stattgefunden hat, um etwa ein '*' für eine veränderte Datei anzuzeigen.
- Bei der Push-Variante werden der update()-Methode alle benötigten (relevanten) Zustandsinformationen als Parameter mitgegeben. Dadurch werden keine Rückrufaktionen durch die Beobachter erforderlich, wodurch die Kopplung gelöst wird: Beobachter müssen sich lediglich bei »ihrem« Subjekt als Interessent anmelden, dessen sonstige Schnittstelle jedoch nicht kennen. Nachteil dieser Lösung ist allerdings, dass man alle möglicherweise benötigten Informationen ermitteln und versenden muss. Da keine Annahmen über Beobachter und von diesen benötigte Anteile des Subjektzustands im Voraus bekannt sind, muss immer der komplette Zustand mitgeteilt werden. Das zu übertragende Datenvolumen kann unter Umständen recht umfangreich sein.

Aufgabe 5: Design Patterns Im JDK 15 – 30 min

Finde im JDK Beispiele für einige der zuvor kennengelernten Design Patterns. Erstelle eine Liste mit rund 10 verschiedenen Mustern, etwa Singleton, Create Method, Observer und zugehörigen Klassen.

CREATIONAL

- Singleton
  - `java.lang.Runtime`
  - `java.awt.Desktop`
- Static factory method
  - `java.util.Calendar`
  - `java.text.NumberFormat`
  - `java.nio.charset.Charset`
- Abstract factory
  - `javax.xml.parsers.DocumentBuilderFactory`
  - `javax.xml.transform.TransformerFactory`
  - `javax.xml.xpath.XPathFactor`

STRUCTURAL

(adapter, decorator, flyweight)

- Flyweight
  - `java.lang.Integer`
  - `java.lang.Boolean`
- Adapter
  - `java.io.InputStreamReader`
  - `java.io.OutputStreamWriter`
- Decorator
  - `java.io.BufferedInputStream`
  - `java.io.DataInputStream`
  - `java.io.BufferedOutputStream`
  - `java.util.zip.ZipOutputStream`
  - `java.util.Collections#checkedList()`

BEHAVIORAL

- Chain of responsibility
  - `javax.servlet.FilterChain`
- Command
  - `java.lang.Runnable`
  - `java.util.concurrent.Callable`
  - `java.awt.event.ActionListener`
- Iterator
  - `java.util.Iterator`
  - `java.util.Enumeration`
- Strategy
  - `java.util.Comparator`
  - `javax.servlet.Filter`
- Templatemethod
  - `java.io.InputStream`
  - `java.io.OutputStream`
  - `java.io.Reader`
  - `java.io.Writer`
- Observer
  - `java.awt.event.ActionListener`
  - `java.util.Observer` / `java.util.Observable`