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: ??? METHOD 15 min

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

Aufgabe 2: ??? METHOD 15 min

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: ??? METHOD 15 min

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 30 min

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

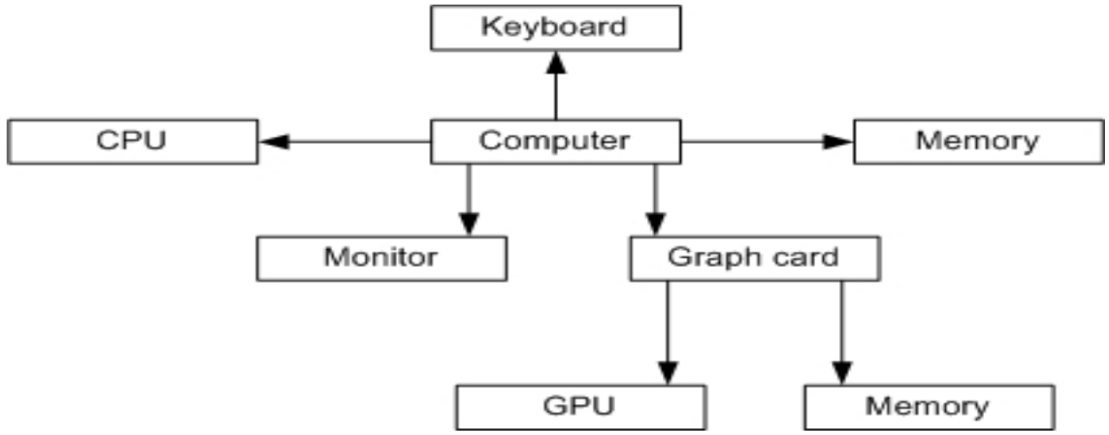
Aufgabe 1: COMPOSITE 30 min

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!

Aufgabe 2: COMPOSITE 30 min

Modelliere einen Computer mithilfe des Kompositum-Musters. Gegeben ist eine erste Realisierung ohne das Muster in der Klasse `CompositeComputerExample` . Nun soll jede Komponente einen Preis besitzen. Bei zusammengesetzten Komponenten ergibt sich der Preis auf Basis der Preise der Einzel-/Subkomponenten (den Teilen). Berechne den Preis für den Computer, die CPU, die Grafikkarte und die GPU.



Prüfe das Ganze mit folgenden Aufrufen, experimentiere ein wenig:

```
public static void main(final String[] args)
{
      Computer computer = new Computer("MY-COMPI");
      computer.add(new ComputerPart("Keyboard", 65));
      computer.add(new ComputerPart("Monitor", 450));
      ComputerPart cpu = new ComputerPart("CPU", 550);
      computer.add(cpu);
      computer.add(new ComputerPart("Memory", 250));

      // Graphics Card
      // TODO

    // TODO: printPrice
}
```

PART 4: Verhaltensmuster

Aufgabe 1: PROXY 15 min

Analysiere im beigefügten Projekt die falsche Implementierung `ProxyPatternExample` im Package `proxy_wrong`. Schaue als Hilfestellung ggf., wie die Fehler in der korrigierten Implementierung behoben wurden. Vollziehe die Probleme und Abhilfen durch Programmstart nach.

Aufgabe 2: PROXY / DYNAMIC PROXY 30 min

Analysiere die Klasse `RestrictedAccessMap` und vor allem, wie aufwändig die Realisierung werden kann: Die Implementierung aller Methoden und die ständigen Aufrufe von `ensureAccessGranted()` machen die Implementierung dieser Klasse recht unübersichtlich, ungelenk und lang. Wie kann man dies durch einen Dynamic Proxy elegant lösen?

Prüfe das Ganze mit folgenden Aufrufen, experimentiere ein wenig:

```
public static void main(final String[] args)
{
    LoggedInUserService.INSTANCE.setLoggedInUser("ADMIN");

    var origValues = Map.of("Mike", 50, "Tim", 50, "Peter", 42);
    var modifiableMap = new HashMap<>(origValues);
    var restrictedMap = createRestricetdAccessMap(modifiableMap);

    restrictedMap.size();
    restrictedMap.entrySet();
    restrictedMap.put("LDM", 6);
    restrictedMap.putAll(Map.of("ALF", 6, "Micha", 47));
}
```

BONUS: Aktiviere zusätzlich den `LoggingInvocationHandler` und verbessere dort die Darstellung der Aufrufe mit und ohne Parameter.

Aufgabe 3: ITERATOR 30 min

A) Implementiere einen `Odd/Even-Pos-Iterator` für Listen (oder Arrays), der jeweils die Elemente an ungerader/gerader Position durchläuft und nutze diesen zur Ausgabe.

B) Implementiere einen `EveryNthListIterator`, der jedes n-te Element durchläuft:

1,2,3,4,5,6,7,8,9,10,11,12 => **3rd** 1,4,7,10 // **5th** 1, 6,11

C) Implementiere einen `EveryNthWithOffsetListIterator`, der jedes n-te Element durchläuft, aber die ersten m Elemente überspringt.

D) Implementiere einen `RandomIterator`, der die Elemente einer Liste in zufälliger Reihenfolge durchläuft.

BONUS

Every n-th Multi Iterator: 1,2,3,4,5,6,7,8,9,10,11,12 => **3rd** 1,4,7,10,2,5,8,11,3,6,9,12
5th 1,6,11,2,7,12,3,8,4,9,5,10 //

Aufgabe 4: STRATEGY Time Table 30 – 45 min

Wir haben bereits bei der Besprechung von OCP die Train Table kennengelernt. Nun kommen nach erfolgreicher Installation – wie in der Praxis auch – neue Wünsche und Anforderungen auf. Diese sind im Dokument «OCP - Exercise - Train Table - Teil 2» beschrieben. Setze dies um.

Aufgabe 5: STRATEGY Filtern 30 – 45 min

Das Dokument «Übung OCP-Strategy-Filtern» zeigt die etwas hemdsärmelige Realisierung einer Filterung von Werten. Gestalte diese Funktionalität mithilfe des Strategy Patterns wartbarer und erweiterbarer.

Aufgabe 6: PATTERN-O-HOLICS 10 min

Das Dokument «patternoholics» zeigt ein komplexeres Gebilde. Analysiere die Abläufe und was ist das Ergebnis? Wie bewertest du das Design?

Aufgabe 7: 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.

Aufgabe 8: OBSERVER 15 min

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

Aufgabe 9: MEDIATOR 30 min

Vervollständige die Klasse `AircraftSimulation` sowie die beteiligten Klassen, die gemeinsam eine Kommunikation zwischen mehreren Flugzeugen und zwei Towers modellieren sollen. Dabei sollen die Flugzeuge Nachrichten an den Tower senden können und diese dann alle anderen Flugzeuge, die beim Tower registriert sind, weitergegeben werden. Zudem kann der Tower Nachrichten an alle Flugzeuge senden.

Realisiere folgende Abfolge:

1) Tower NY: `"Stormy weather, danger of blizzards"`
2) Tower PH: `"change altitude + 1000"`
3) Flug 2: `"Mayday"`

PART 5: Design Patterns mit Java 8

Aufgabe 1: Strategy Pattern Filtern mit Java 8 30 – 45 min

Das Dokument «Übung OCP-Strategy-Filtern» zeigt die etwas hemdsärmelige Realisierung einer Filterung von Werten. Gestalte diese Funktionalität basierend auf den Ideen des Strategy Patterns wartbarer und erweiterbarer. Nutze dabei Lambdas, Functional Interfaces und insbesondere Predicate in Kombination mit Streams.

Aufgabe 2: Execute Around Pattern 15 min

In den Folien haben wir das Execute Around Pattern kennengelernt. Wende dieses an, um eine Funktionalität zur Freigabe von Lock namens `withLock()` zu entwickeln.

Aufgabe 3: Factory Pattern 15 min

Modifiziere die `ProductFactory` so, dass sie Java 8 Mittel nutzt (u.a. Methodenreferenzen und einen Supplier)

Aufgabe 4: Strategy Pattern 15 min

Modifiziere die `ValidationExample` so, dass sie Java 8 Mittel nutzt (u.a. Lambdas)