Domain Modeling sowie Java Records und Enums

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de , Raum 149B
Version: 1.0.2.3

Eine kurze Einführung, um das Entwickeln von kleinen Projekten zu erleichtern.

Kontrollfragen: kontrollfragen.de.md.html
Folien: https://delors.github.io/prog-adv-java-domain-modeling/folien.de.md.html
https://delors.github.io/prog-adv-java-domain-modeling/folien.de.md.html.pdf
Fehler melden: https://github.com/Delors/delors.github.io/issues
KI Verwendung: Bei der Erstellung der Unterlagen wurden KI Assistenten (insbesondere Claude aber ggf. auch ChatGPT, Ollama mit Gemma/LLama/Qwen oder OpenCode mit Kimi/Qwen/Deepseek...) unterstützend eingesetzt. Dies erfolgte insbesondere zur Unterstützung bei der Generierung von Grafiken (d. h. SVG Dateien), oder um sich Übersichtstabellen generieren zu lassen. Weiterhin wurde KI zur allgemeinen Qualitätssicherung eingesetzt. Inhalte, die ggf. von der KI vorgeschlagen wurden, wurden im Falle der Übernahme explizit validiert und angepasst.

1. Implementierung einer einfachen Domänenklasse

Wiederholung etablierter Konzepte

✎ Übung

1.1. Implementieren einer Klasse für einfache Telefonbucheinträge

1. Entwickeln Sie eine Klasse `Telefonbucheintrag` mit den Attributen:
    1. `Integer telefonnummer`
    2. `String vorname`
    3. `String nachname`

sowie geeigneten Konstruktoren und allen passenden *get*- und *set*-Methoden. Setzen Sie strikte *Encapsulation* um.

2. Implementieren Sie weiterhin eine Methode `public String toString()`, die den Eintrag in der Form „Vorname Nachname, Telefonnummer“ zurückgibt.
3. Implementieren Sie eine Methode `public boolean equals(Object obj)`, die zwei Einträge (d. h. zwei Objekte mit dem dynamischen Typ `Telefonbucheintrag`) als gleich betrachtet, wenn die Telefonnummern gleich sind.

   Prüfen Sie - durch das Studium der Dokumentation der Methode `java.lang.Object.equals()` - ob Sie die Methode korrekt implementiert haben.
4. Implementieren Sie eine Methode `public int hashCode()`, die einen `int` Wert zurückgibt und einen Eintrag halbwegs sinnvoll repräsentiert. Prüfen Sie ob Ihre Implementierung dem Kontrakt der Methode `java.lang.Object.hashCode()` entspricht.
5. Schreiben Sie eine Methode, die drei `Telefonbucheintrag`-Objekte erzeugt. Zwei davon sollen die gleichen Inhalte haben. Prüfen Sie dann, ob die Methoden `equals()` und `hashCode()` korrekt implementiert sind.

2. (Domain) Modeling [Larman2001]

Was ist Domain Modeling?

Warum:	Die Domänenmodellierung hilft uns, die relevanten Konzepte und Ideen einer Domäne zu identifizieren.
Wann:	Immer dann, wenn wir die (weiteren) Konzepte in einem Bereich verstehen müssen.
Beteiligte:	Entwickler, Domänenexperten(, Anwender)
Leitlinie:	Erstellen Sie nur für die anstehenden Aufgaben ein Domänenmodell.

Domain Model ≙ *Analysemodell oder auch Konzeptmodell*

> Curtis'law: [...] Good designs require deep application knowledge.[1]
>
> —[EndresRombach2003]

---

[1] Sinngemäß: „Ein guter Entwurf benötigt ein tiefgreifendes Verständnis des Einsatzgebiets der zu entwickelnden Software."

Das Domänenmodell

- Das Domänenmodell wird erstellt, um die Domäne in Konzepte oder Objekte in der realen Welt aufzuschlüsseln.
- Das Modell sollte die Menge der konzeptionellen Klassen identifizieren.
  (Das Domänenmodell wird iterativ vervollständigt.)
- *Es ist die Grundlage für den Entwurf der Software*.

Erstellen eines Domänenmodells

? Frage

Was sind die relevanten Konzepte/Objekte der Domäne?

Vorgehen

1. Identifizieren Sie die relevanten Konzepte/Objekte.

   (Dies kann zum Beispiel durch das Studieren von existierenden Modellen passieren[Fowler1997] oder durch die Analyse von fachlichen Dokumenten, die die Domäne beschreiben.)
2. Identifizieren Sie die Attribute der Konzepte/Objekte.
3. Identifizieren Sie die Beziehungen zwischen den Konzepten/Objekten.
4. Erstellen Sie ein Klassendiagramm.

!! Wichtig

Verwenden Sie das Vokabular der Domäne; z. B. sollte ein Modell für eine Bibliothek Namen wie „Ausleiher“ anstelle von „Kunde“ verwenden.

Modellierungsaspekte

01

? Frage

Wann sollte ich etwas als Attribut oder als Klasse modellieren?

✓ Antwort

Faustregel: Wenn wir uns ein Konzept X in der realen Welt nicht als Zahl, Datum oder Text vorstellen können, dann sollte X wahrscheinlich mit Hilfe einer Klasse modelliert werden und ist kein Attribut.

02

Die Attribute in einem Domänenmodell sollten vorzugsweise „primitive“ Datentypen in Bezug auf die Domäne sein.

Sehr häufige Datentypen sind: Booleans, Datum, Zahl, Zeichen, String, Adresse, Farbe, Telefonnummer,...

03

Erwägen Sie die Modellierung von Mengen als Klassen, um Einheiten zuordnen zu können.

⧉ Beispiel

Der Datentyp des Attributs „Betrag“ einer Zahlung sollte auch die Währung angeben.

✎ Übung

2.1. Domänenmodell für ein Kassensystem

Erstellen Sie ein Domänenmodell (d. h. ein UML Klassendiagramm) für ein einfaches Kassensystem basierend auf der folgenden Beschreibung und Ihrem Domänenwissen:

> Verkauf abwickeln: Ein Kunde kommt an der Kasse an und möchte einen Artikel kaufen. Der Kassierer verwendet das Kassensystem, um jeden Artikel zu erfassen. Das System zeigt eine laufende Summe und Details zu den einzelnen Positionen an. Der Kunde gibt die Zahlungsinformationen ein, die das System prüft und aufzeichnet. Das System aktualisiert den Warenbestand. Der Kunde erhält eine Quittung vom System und verlässt dann das Geschäft mit den Artikeln.

⁂ Hinweis

Denken Sie daran, dass wir im Domänenmodell nur die relevanten Konzepte modellieren sollten; d. h. Klassen und deren Attribute und Beziehungen. Methoden sind hier nicht relevant.

3. Java records[JEP395]

Java Records - Überblick

- Java Records sind eine spezielle Form von Klassen, die dazu dienen, unveränderliche Daten zu modellieren.
- Java Records sind häufig hervorragend geeignet, um Klassen aus Domänenmodellen, die insbesondere der Datenhaltung dienen, zu modellieren.
- Java Records sind seit Java 16 verfügbar.

Beispiel: Implementierung einer Klasse `Point`

Traditioneller Ansatz

```java
class Point {
    private final int x;
    private final int y;

    Point(int x, int y) {
        this.x = x; this.y = y;
    }

    int x() { return x; }
    int y() { return y; }

    public boolean equals(Object o) {
        if (!(o instanceof Point)) return false;
        Point other = (Point) o;
        return other.x == x && other.y == y;
    }

    public int hashCode() {
        return Objects.hash(x, y);
    }

    public String toString() {
        return String.format(
            "Point[x=%d, y=%d]", x, y
        );
    }
}
```

Implementation mit Java `record`

```java
record Point(int x, int y) {}
```

Verwendung von Java Records

```
jshell> record Point(int x, int y) {}
|  created record Point

jshell> var p = new Point(1,2);
p ==> Point[x=1, y=2]
```

Deklaration und Initialisierung

```
jshell> var x = p.x();var y = p.y()
x ==> 1
y ==> 2

jshell> System.out.println(p.toString() + " #" + p.hashCode() )
Point[x=1, y=2] #33
```

Verwendung

```
jshell> new Point(1,2).hashCode();
33

jshell> new Point(1,2) == p
false

jshell> new Point(1,2).equals(p);
true
```

Die Getter und Setter heißen bei Records *Component Methods*. Ein direkter Zugriff auf die Attribute ist nicht möglich:

```
jshell> p.x
|  Error:
|  x has private access in Point
|  p.x
|  ^-^
```

Java Records - Technische Besonderheiten

- Java Records erben immer von `java.lang.Record`.
- Die Klasse ist (implizit) `final` (und notwendigerweise nicht abstrakt).
- Die Felder, die die Komponenten eines Records sind, sind `final` und `private`.
- Ein Record kann keinen weiteren (veränderlichen) Zustand haben.
- Verschachtelte/Lokale Records sind möglich sind jedoch immer `static`.
- Records sind *serializable*.
- Sie haben eine *equals()*- und *hashCode()*-Methode, die auf allen Attributen basiert.
- Sie haben eine *toString()*-Methode, die alle Attribute ausgibt.
- Sie haben *getter*-Methoden für alle Attribute.
- Sie haben einen Konstruktor, der alle Attribute initialisiert.
- Sie können *static* und *non-static* Methoden haben.
- Sie können *interfaces* implementieren.
- Sie können *annotations* haben.

Konstruktoren von Record Klassen

- Ein Record hat immer einen kanonischen Konstruktor, der alle Attribute initialisiert.

(Ein Record hat nie einen parameterlosen Standardkonstruktor.)

- Ein Record kann weitere Konstruktoren haben, die jedoch den kanonischen Konstruktor aufrufen müssen.

Beispiel

```
record Point(int x, int y) {
    public Point(int x) { this(x,x); }
}
```

- Es ist möglich einen kompakten kanonischen Konstruktor zu definieren. Der Code zur Initialisierung der Attribute (z. B. `this.x = x;`) wird dann implizit am Ende generiert.

Beispiel

```
record Point(int x, int y) {
    public Point {
        if (x < 0 || y < 0)
            throw new IllegalArgumentException(
                "Negative Koordinaten sind nicht erlaubt.");
    }
}
```

- Der primäre Zweck von zusätzlichen Konstruktoren ist es, die Validierung oder Normalisierung der Attribute zu ermöglichen.

✎ Übung

3.1. Ein einfacher Telefonbucheintrag mit Java Records

Entwickeln Sie eine Klasse Telefonbucheintrag mit den Attributen:

```
1. int telefonnummer
2. String vorname
3. String nachname
```

verwenden Sie dazu ein Java Record. Führen Sie ggf. eine Normalisierung der Attribute durch (löschen von Leerzeichen am Anfang und Ende). Validieren Sie die übergebenen Werte (Telefonnummer muss (hier) größer 0 sein und die Namen müssen mind. einen Buchstaben enthalten. Instanziieren Sie drei Objekte und prüfen Sie, ob die Validierung korrekt funktioniert und die Vergleichbarkeit der Objekte korrekt implementiert ist.

4. Aufzählungen

Modellierung von Aufzählungen mit Java Enums

Eine enum-Deklaration spezifiziert eine neue Enum-Klasse, eine eingeschränkte Art von Klasse, die eine kleine Menge von benannten Klasseninstanzen definiert.

Beispieldeklaration

```
public enum Arbeitstag { MONTAG, DIENSTAG, MITTWOCH, DONNERSTAG, FREITAG }
```

Beispielverwendung (Arbeitstag w = Arbeitstag.FREITAG)

```
jshell> switch(w) {
          case FREITAG -> System.out.println("gleich ist Wochenende");
          default -> System.out.println("noch viel zu tun");
        }
gleich ist Wochenende

jshell> w.values();
==> Arbeitstag[5] { MONTAG, DIENSTAG, MITTWOCH, DONNERSTAG, FREITAG }

jshell> Arbeitstag.valueOf("FREITAG").ordinal()
==> 4
```

Java Enum Konstanten können eigene Eigenschaften haben

Beispiel

```java
public enum Arbeitstag {
    MONTAG(1), DIENSTAG(2), MITTWOCH(3), DONNERSTAG(4), FREITAG(5);

    private final int tag;

    Arbeitstag( int tag ) {
        if (tag < 1 || tag > 5)
            throw new IllegalArgumentException("Ungültiger Tag: " + tag);
        this.tag = tag;
    }
}
```

Enum Konstanten können eigene Klassenbodies deklarieren

Deklaration

```
enum Operation {
    PLUS {
        double eval(double x, double y) { return x + y; }
    },
    DIVIDED_BY {
        double eval(double x, double y) { return x / y; }
    };

    abstract double eval(double x, double y);
}
```

Verwendung

```
jshell> double x = 2;
jshell> for(var op : Operation.values()) {
            System.out.println(op.name() + " " +op.eval(x,x));
        }
PLUS 4.0
DIVIDED_BY 1.0
```

Java enums - Technische Besonderheiten

- Enums sind `final` oder `sealed` (falls es innere Klassen gibt)
- geschachtelte Enums sind (implizit) `static`
- der Supertyp alle Enums ist *java.lang.*`Enum`

  (`extends` wird für Enums nicht unterstützt.)
- Klonen von Enums (*clone*()) ist nicht möglich.
- Es können keine Instanzen der Enum Klasse (z. B. der Klassen Wochentag erzeugt werden.)

✎ Übung - Java Enums

4.1. Enum für Währungen

Deklarieren Sie eine Enum (`Currency`) für Währungen (Euro, Pfund etc.).

Es soll möglich sein für eine Währung, das Währungssymbol zu erhalten.

Für jede Währung soll es weiterhin möglich sein, die verfügbaren Stückelungen (🇺🇸 *denominations*) zu erhalten.

Schreiben Sie eine kleine `main()`-Methode, um Ihr Enum zu testen.

> ◆ Bemerkung
>
> Falls Sie die Stückelungen in einem Array zwischenspeichern sollten, dann stellen Sie sicher, dass das Array nicht verändert werden kann, wenn der Nutzer die verfügbaren Stückelungen abfragt.

Bibliography

[Larman2001] Craig Larman; Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process; Prentice Hall, 2001

[EndresRombach2003] Albert Endres and Dieter Rombach; A Handbook of Software and Systems Engineering; Addison Wesley, 2003

[Fowler1997] Martin Fowler; Analysis Patterns: Reusable Object Models; Addison-Wesley, 1997

[JEP395] JEP 395: Records; https://openjdk.java.net/jeps/395; zuletzt aktualisiert am 3.2.2024