Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Prof. Dr. Sascha Alda
Dr. Rita Cornely, Moritz Balg

18. Dezember 2024

Übungen zur Vorlesung Software Engineering I
WS 2024 / 2025

Übungsblatt Nr. 9
(Abgabe bis: Freitag, den 10. Januar 2025, 21:00 Uhr)

Aufgabe 1 (Modellierung und Umsetzung einer Dokumentstruktur, 20 Punkte):

Sie haben die Aufgabe, eine *hierarchische* Dokumentstruktur zu modellieren und zu implementieren. Folgende Anforderungen sollen dabei umgesetzt werden.

Eine hierarchische Dokumentstruktur besteht allgemein aus Dokumenten. Ein Dokument (ein „`Document`-Objekt“) kann ein komplexes Dokument („`ComplexDocument`“) oder ein einfaches Dokument („`CoreDocument`“) sein. Ein `ComplexDocument` kann eine beliebige Anzahl von `Document`-Objekten (also `CoreDocument` *oder* `ComplexDocument`) aufnehmen. Aus einem `ComplexDocument` heraus sollen zudem auch wieder `Document`-Objekte gelöscht werden können.

Ein `CoreDocument` ist als *abstrakt zu* interpretieren, wovon es aber zwei konkrete Ausprägungen (SubTypen) gibt: `TextDocument` und `GraficDocument`.

Ein `TextDocument` wird instanziiert mit einem Inhalt (hier: vom Typ `String`) sowie einem Encoding (zu implementieren als ein internes `enum`). Das `TextDocument` sollte drei Encoding-Typen unterstützen: UTF-8, UTF-32 sowie UTF-16. Ein `TextDocument`-Objekt z.B. mit einem Encoding UTF-8 *muss* demnach wie folgt über einen Konstruktor erzeugt werden können:

```
... = new TextDocument("Ein Text!", TextDocument.Encoding.UTF8  );
```

wobei in der Konstanten UTF8 ein entsprechender String belegt sein muss, der dann bei der Byte-Berechnung des Strings (siehe unten) verwendet werden kann. Hinweis: recherchieren sie hierzu die Methode `getBytes` aus der Klasse `String`. Eine gute Erläuterung findet sich hier:

https://stackoverflow.com/questions/4385623/bytes-of-a-string-in-java
(Last Access: 08.01.2024)

Es soll also die Anzahl der Bytes berechnet werden, die benötigt werden, um einen String in einem bestimmten Encoding darzustellen.

Ein `GraficDocument`-Objekt wird nur über die Angabe einer URL erzeugt, die als Inhalt des Objekts *angenommen* werden sollte:

```
... = new GraficDocument( "localhost:8080" );
```

Werden konkrete `Document`-Objekte erzeugt, dann muss eine beliebige ID von *außen* über eine entsprechende `setter`-Methode gesetzt werden. Auch das Auslesen einer gesetzten ID über eine `getter`-Methode soll möglich sein.

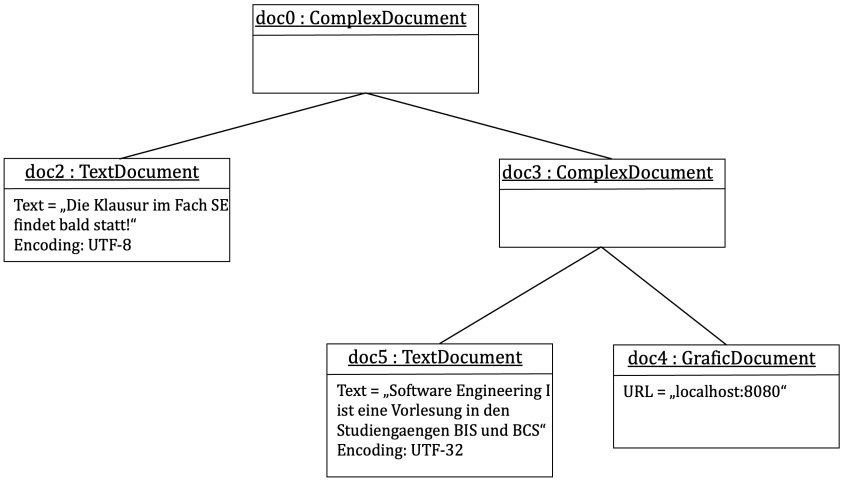
Es soll zudem möglich sein, innerhalb einer gegebenen Hierarchie über alle `Document`-Objekte hinweg mittels einer Traversierungs-Operation die gesamte Größe (ausgedrückt in der Einheit *Bytes*) der hierarchischen Dokumentstruktur zu ermitteln. Diese ist bei `TextDocument`-Objekten abhängig vom gewählten Encoding. Bei `GraficDocument`-Objekten können sie von einem konstanten Wert von 1200 Bytes ausgehen.

1.)
Modellieren sie diese Struktur mit Hilfe der UML sowie unter der Anwendung eines entsprechenden Entwurfsmusters. Modellieren sie die resultierende Struktur inklusive der Methoden (inkl. Parameter und Sichtbarkeiten) und der Attribute (inkl. Sichtbarkeiten) anhand eines Klassendiagramms *exakt*. Ein Client, welche die Dokument-Struktur verwendet, brauchen sie nicht zu modellieren.

Das enum Encoding sollten sie in ihrem Modell als eigenständige Klasse mit einem Stereotype <<enumType>> interpretieren, welches mit der Klasse `TextDocument` assoziiert ist. Hier ist im Modell zu berücksichtigen, dass die Klasse `TextDocument` auf das Encoding navigieren kann, umgekehrt aber nicht.

2.)
Liefern sie auch eine Java-Implementierung ihrer identifizierten Klassen und Interfaces. Für die Methoden sollten sie eine einfache Implementierung angeben. Entwickeln sie auch eine Klasse `TestClient`, in der die hierarchische Dokumentstruktur, wie in der unteren Abbildung dargestellt, *exakt* implementiert wird. Errechnen sie darin auch die gesamte Größe in Bytes über alle Dokumente hinweg. IDs können sie beliebig setzen. Anstelle einer Test-Klasse können sie natürlich auch einen JUNIT-Test entwickeln.

3.)
Modellieren sie die Interaktionen bei der Erstellung und Auswertung (Traversierung) der Byte-Größe einer Objekt-Hierarchie mittels eines *konkreten* Sequenzdiagramms. Die Interaktionen müssen dabei als *synchron* modelliert werden. Die Parameter im Konstruktor der Dokumente (z.B. Text oder Encoding) müssen nicht berücksichtigt werden. Starten sie die Modellierung ausgehend von ihrer Test-Klasse. Beschränken sie ihre beispielhafte Modellierung auf die Erstellung und Auswertung der Substruktur ausgehend von `doc3` aus der unteren Abbildung (insgesamt also drei Objekte in der Hierarchie).



Aufgabe 2 (Verifikationen von Rechnungen, 10 Punkte)

Der Betreiber des Legacy-Systems Reise-Anbieter aus der Aufgabe 8-1 bittet um ihre Mithilfe! Das System (repräsentiert durch eine Klasse `ReiseAnbieterController`, welche das Interface `ReiseAnbieter` implementiert) soll über eine Möglichkeit zur Verifikation von Buchungen gemäß internationalem Reporting-Standard erweitert werden. Ihr Auftraggeber sieht dabei aktuell drei verschiedene Verifikationen vor, die jeweils als eigenständige Algorithmen im System bereitgestellt werden sollen und von einer Klasse `GlobalConfig` flexibel *gesetzt* werden kann:

- Swiss GAAP FER
- IFRS
- US-GAAP

Da das Unternehmen in Zukunft plant, in weitere Länder zu expandieren, möchte man zudem offen sein für die Integration weiterer Algorithmen zur Anbindung weiterer länderspezifischer Standards. Alle Algorithmen verfügen über ein gemeinsames Interface, wobei das Interface über eine Methode verfügt:

```
verfiyBooking ( b : Buchung ) : Status
```

Die Klasse `ReiseAnbieterController` soll von den Algorithmen entkoppelt sein, d.h. die Klasse darf die Klassen der Algorithmen nicht kennen.

Ihre Aufgabe:

a.)

Welches Entwurfsmuster (*Singular*!) würden sie für dieses Entwurfsproblem verwenden? Modellieren sie mit Hilfe des identifizierten Entwurfsmusters das Klassendiagramm entsprechend. Modellieren sie die Signaturen der Methoden ihrer identifizierten Klassen mit Hilfe der Notationsmöglichkeiten durch die UML. Modellieren sie auch die wichtigsten Attribute. Die Klassen `Buchung` und `Status` brauchen sie nicht zu berücksichtigen. Die

Abhängigkeiten ausgehend von der Klasse `GlobalConfig` sollten sie explizit berücksichtigen.

b.)
Modellieren sie ein konkretes Szenario mit Hilfe eines UML-basierten Sequenzdiagram. Verwenden sie dazu *asynchrone* Interaktionen. Gehen sie davon aus, dass die Policy zu Beginn des Szenarios den Algorithmus „IFRS“ erzeugt und in den Context einsetzt. Darauf kann der Client (hier: `ReiseAnbieterController`) die weiteren Interaktionen initiieren. Bei Callbacks können sie eine Callback-Funktion `return` annehmen, welche z.B. den Status übergibt (Beispiel: `return ( s : Status )`). Die Objekte `Buchung` und `Status` brauchen sie nicht explizit erzeugen, können sie aber als Argumente auf den Nachrichten annehmen.

c.)
Die Klasse `ReiseAnbieterController` verwendet auch eine Klasse BuchungDAO, die Buchungs-Objekte verwaltet. Modellieren Sie die Klasse BuchungDAO mittels eines UML-basierten Klassendiagramms (Hinweis: nur die Klasse BuchungDAO modellieren). Die packet-privaten Methoden der Klasse sollen gemäß dem CRUD-Mechanismus aufgebaut sein. Für jede Kategorie aus CRUD sollte mindestens eine Methode vorhanden sein. Entwickeln Sie zudem einen exemplarischen Java-Code. Die Methoden selber brauchen Sie nicht zu implementieren, geben Sie nur die Signaturen der Methoden an.