Besprechung: 06.12.2019

Übungen zur Vorlesung Software Engineering — WS 19/20

Übungsblatt 3

1. **Modelling Behavoir: Use-Case Diagramm**

   Da Sie damit beauftragt wurden die neue Bildbearbeitungssoftware IntelliPhoto zu implementieren, führten Sie eine Umfeldanalyse durch. In dieser haben Sie wertvolle Informationen über verschiedene Nutzergruppen sammeln können.

   So erfuhren Sie, dass Casual User und Einsteiger die Software hauptsächlich für kurze Aufgaben wie das Zusammenschneiden von Bildern, das Ändern der Bildauflösungen und dem Drehen von Bildern benutzen wollen. Außerdem möchten die Casual User die Software dazu benutzen um bestimmte Regionen in einem Bild zu retuschieren.

   Eine weitere Nutzergruppe, die freiberuflichen Fotografen, hingegen möchten neben der Bildretusche auch eine Reihe an Korrekturwerkzeugen, wie der *Helligkeit/Kontrast*, *Farbton/Sättigung* und den *Gradationskurven*, als auch Auswahlwerkzeuge und verschiedene Pinsel haben.

   Die letzte Gruppe von potentiellen Benutzern, die 3D Künstler, wünschen sich eine Schnittstelle für den Import von gängigen 3D-Dateien. Auch soll es für sie möglich sein, einfache geometrische 3D-Objekte direkt im Bild zu erzeugen.

   Jede Nutzergruppe gab an, dass sie sich eine Ebenendarstellung in der Software vorstellen können und benutzen würden.

   Fassen Sie die beschriebenen Ergebnisse in einem *UML-Use-Case-Diagramm* zusammen.

2. **UML Structure: UML-Klassendiagramm**

   Modellieren Sie ein Unternehmen als UML-Klassendiagramm, welches weltweit beliebig viele Standorte besitzt.

   Dabei setzt sich ein Standort aus mindestens einem Gebäude inklusive Adresse zusammen.

   Ein Gebäude besitzt mehrere Büros und exakt eine Mensa. Die Büros haben Nummern sowie ein Namensschild an der Tür.

   In den Büros sitzen Angestellte, welche entweder der Chef, das Management oder der Arbeiterschaft zugeordnet sind.

   Zu beachten ist, dass einem Standort ein Chef und 3–8 Personen aus dem Management zugeordnet sind sowie mindestens 5 Arbeiter haben. Gekennzeichnet sind die Angestellten durch eine ID.

   Die Berufsgruppen haben zudem eigene Aufgabenfelder: Der Chef kontrolliert das Management, welches wiederum die Arbeiter überwacht, welche wiederum die Arbeit verrichten.

   Das Unternehmen stellt verschiedene Produkte (PCs, Laptops, Server) her.

3. **Adapter Pattern: Deque**

   Implementieren Sie die Datenstruktur Deque in Java mit Hilfe des Adapter Patterns.

   Die Operationen der Deque sind:

   - `push` und `pop`: Einfügen und Entnehmen eines Elementes am hinteren Ende der Deque.
   - `put` und `get`: Einfügen und Entnehmen am vorderen Ende der Deque.
   - `first` und `last`: Lesen des ersten oder letzten Elements, ohne es zu entfernen.

   Achten Sie bei Ihrer Implementierung auf Java Generics und bauen Sie Ihre Lösung auf eine bestehende "Collection", wie z.B. `java.util.ArrayList`, auf. Vergewissern Sie sich, dass Ihre neue Klasse *keine* weiteren Methoden als die oben genannten ausführen kann (z.B. `add()` oder `clear()`).

4. **Visitor Pattern: Termausgabe**

   Wir betrachten Terme über die Rechenarten *op* $\in \{+, \cdot\}$, die folgendermaßen rekursiv definiert sind:

   - jedes Literal ist ein Term, z.B. 4
   - ist t ein Term, so ist (t) ein Term
   - sind t_1, t_2 Terme so ist t_1 *op* t_2 ebenso ein Term

   Beispiele für gültige Terme: $4 + 8$, $4 \cdot 8$ oder $4 + (4 \cdot 8)$.

   a) Implementieren Sie die entsprechenden Klassen `Expression`, `Literal`, `Brackets`, `BinaryExpression`, `Addition` und `Multiplication` im Sinne des Visitor Patterns.

   b) Implementieren Sie danach die Visitor Klassen `EvalVisitor` und `PrettyPrintVisitor`.

      `EvalVisitor`: Evaluiert bzw. berechnet den gegebenen Term und hält das Ergebnis

      `PrettyPrintVisitor`: Gibt einen Term in leserlicher Form aus.

      Prüfen Sie Ihre Implementierung durch geeignete Tests!

Hinweise zur Abgabe

- Die Lösungen sind zur entsprechenden Übung mitzubringen.
- Während der Übung werden zufällige Personen ausgelost, die ihre Lösung vorstellen.