Vorlesung Software-Projekt

Sommersemester 2014

Prof. Dr. Rainer Koschke

5. Übungsblatt

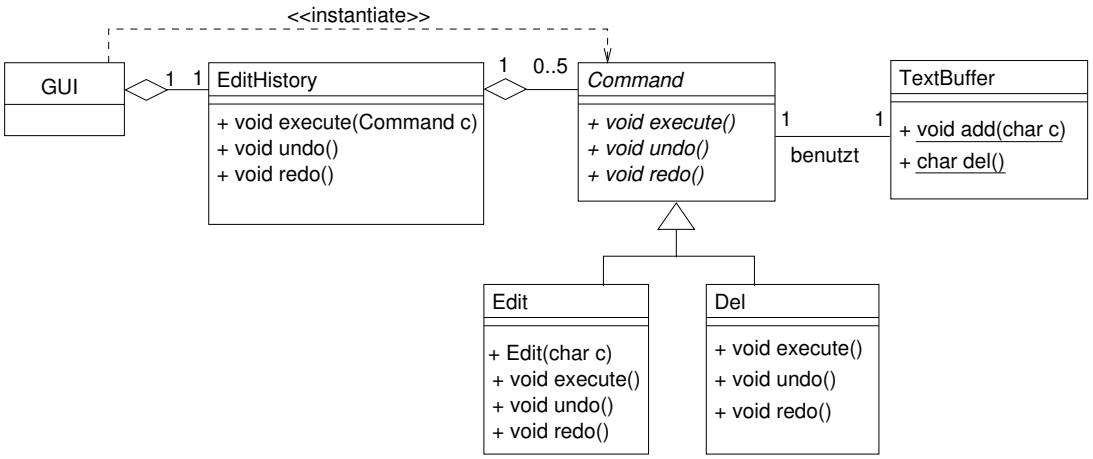
Dieses Übungsblatt ist spätestens am 27. Juli 2014 23:59 Uhr (MESZ) über MEMS abzugeben.

Aufgabe 1 (20 Punkte)

Gegeben sei ein einfacher Texteditor, der zwei simple Kommandos zur Verfügung stellt:

- Eine Edit-Operation, mit deren Hilfe ein einzelnes Zeichen eingegeben werden kann. Diese Operation sei mit Hilfe der Klasse `Edit` implementiert.
- Eine Delete-Operation, mit der das zuletzt eingegebene Zeichen gelöscht werden kann. Diese Operation sei mit Hilfe der Klasse `Del` implementiert.

Der Texteditor soll über eine Undo-Funktionalität verfügen, die mit Hilfe des Entwurfsmusters `Command` implementiert wird. Der Sachverhalt wird im folgenden Klassendiagramm nochmals skizziert:



In der Klasse `EditHistory` werden alle bisher ausgeführten Kommandos aus der GUI abgespeichert. Die Klasse `TextBuffer` speichert den eingegebenen Text. Sie hat zwei statische Methoden:

- `static void add(char c)` speichert das Zeichen `c` im Textpuffer;
- `static char del()` löscht das zuletzt eingegebene Zeichen des Textpuffers und liefert dieses gelöschte Zeichen als Ergebnis zurück.

Implementieren Sie die Klassen `Command`, `Edit` und `Del` in Java.
Hinweis: Das ist eine alte Klausuraufgabe. Das bedeutet, Sie müssen auch in der Klausur in der Lage sein, Java zu schreiben, und zwar ohne die Hilfen, die eine moderne IDE, wie Eclipse, bietet.

Aufgabe 2 (20 Punkte)

Eine Bücherei verwaltet Bücher und Buchserien. Eine Buchserie besteht aus mehreren Büchern (mindestens ein Buch). Ein Buch kann höchstens Teil einer Buchserie sein. Es gibt auch Bücher, die keiner Buchserie zugeordnet sind. Bücher und Buchserien haben einen Titel und einen Verlagsnamen. Bücher haben darüber hinaus eine Autorenliste mit mindestens einem Autoren.
Geben Sie ein UML-Klassendiagramm an, das diesen Sachverhalt beschreibt. Setzen Sie dieses Klassendiagramm dann in Java-Datenstrukturen um. Ergänzen Sie den Code um die notwendigen Setter und Getter für die Attribute (die alle als `private` deklariert sein sollen).

Aufgabe 3 (20 Punkte)

Diese Aufgabe baut auf der vorherigen Aufgabe 2 auf.
Ein Leser kann einzelne Bücher mit einem bis fünf Sternen bewerten (Buchserien können nicht selbst bewertet werden). Die Signatur der entsprechenden Methode lautet wie folgt:

```
void setRating(int rating); // 1 <= rating <= 5
```

Die Methode `setRating()` wird bei jeder einzelnen Bewertung durch einen Leser aufgerufen. Sie kann also im Verlaufe des Programms mehrfach aufgerufen werden.
Darüber hinaus gibt es eine Operation, die sowohl für Bücher als auch Buchserien die durchschnittliche Bewertung durch die Leser liefert. Ihre Signatur lautet wie folgt:

```
double getRating(); // 0.0 <= result <= 5.0
```

Im Falle eines Buches liefert `getRating()` die durchschnittliche Bewertung, die mittels `setRating()` vorher angegeben wurde, also über alle vorherigen Aufrufe dieser Methode hinweg. Der Wert 0 wird genau dann zurückgeliefert, wenn noch keine Bewertung stattgefunden hat.
Im Falle einer Buchserie liefert `getRating()` entsprechend das gewichtete arithmetische Mittel der Bewertungen aller bereits bewerteten Bücher der Serie und 0 genau dann, wenn noch kein einziges Buch der Buchserie bewertet wurde. Die Gewichtung einer Buchbewertung ist die Anzahl der abgegebenen Bewertungen. Besteht eine Buchserie S zum Beispiel aus drei Büchern A, B und C, wobei A die durchschnittliche Bewertung 3,3 von 5 Lesern und B die durchschnittliche Bewertung 1,2 von 2 Lesern habe und C noch gar nicht bewertet sei, dann errechnet sich die Bewertung von S wie folgt: $(3,3 \cdot 5 + 1,2 \cdot 2)/(5 + 2)$.

Erweitern Sie Ihr Java-Programm aus der vorherigen Aufgabe, indem Sie an passender Stelle in Ihren Klassen die oben genannten beiden Operationen einfügen. Wenden Sie bei der Implementierung von `getRating()` das Entwurfsmusters *Composite* an.

Aufgabe 4 (20 Punkte)

Gegeben ist das folgende zu testende Programm, das bestimmt, von welchem Typ ein Viereck ist:

```
1  public class Viereck {
2
3      /**
4       * Bestimmt den speziellsten Typ eines Vierecks anhand gegebener Innenwinkel wie folgt:
5       *
6       *  - Rechteck: alle vier Innenwinkel betragen 90 Grad
7       *  - Parallelogramm: zwei Paare gleich grosser gegenueberliegender Innenwinkel
8       *  - Trapez: zwei Paare benachbarter Supplementwinkel (das heisst, die Winkel ergaenzen
9       *        sich zu 180 Grad)
10      *  - Sehnenviereck: die beiden gegenueberliegenden Innenwinkel betragen in der
```

```
11      *         Summe 180 Grad
12      *  - sonstiges Viereck: die Summe der Innenwinkel betraegt 360 Grad
13      *  - kein Viereck: sonst
14      *
15      * @param alpha erster Winkel, gegenueberliegender Winkel von gamma
16      * @param beta  zweiter Winkel, gegenueberliegender Winkel von delta
17      * @param gamma dritter Winkel, gegenueberliegender Winkel von alpha
18      * @param delta vierter Winkel, gegenueberliegender Winkel von beta
19      *
20      * @return Rechteck => 1, Parallelogramm => 2, Trapez => 3, Sehnenviereck => 4,
21      *       sonstiges Viereck => 5,
22      * @throws Exception wird geworfen, wenn kein Viereck vorliegt
23      */
24     public static int typ(double alpha, double beta, double gamma, double delta)
25             throws Exception {
26         if (alpha + beta + gamma + delta != 360.0) {
27             throw new Exception("kein Viereck");
28         }
29         if (alpha <= 0.0 || beta <= 0.0 || gamma <= 0.0 || delta <= 0.0) {
30             throw new Exception("kein Viereck");
31         }
32         if (alpha == beta && beta == gamma && gamma == delta) {
33             int result = 2;
34             if (alpha == 90.0) {
35                 result = 1;
36             }
37             return result;
38         } else if (alpha == gamma && beta == delta) {
39             return 2;
40         } else if (alpha + beta == 180.0 && gamma + delta == 180.0) {
41             return 3;
42         } else if (alpha + delta == 180.0 && gamma + beta == 180.0) {
43             return 3;
44         } else if (alpha + gamma == 180.0 || beta + delta == 180.0) {
45             return 4;
46         } else {
47             return 5;
48         }
49     }
50 }
```

a) Was ist ein vollständiger Test?

b) Ihr Chef bittet Sie um eine Stellungnahme zum Aufwand für einen Test. Seine Frage ist, ob es möglich ist, die Methode `typ()` innerhalb eines Monats *vollständig* zu testen, wenn jeder Eingabeparameter der Methode `typ()` auf den Wertebereich $[1.0 \ldots 128.0]$ beschränkt wird? Wie beantworten Sie diese Frage? Begründen Sie Ihre Antwort.

c) Wie viele Testfälle bräuchten Sie höchstens für einen **vollständigen Äquivalenztest**, wenn Sie aus jeder Äquivalenzklasse genau einen Testfall auswählen? Begründen Sie Ihre Antwort, indem Sie dazu explizit für jede Ihrer Äquivalenzklassen genau einen Testfall (mit Eingabe und erwarteter Ausgabe) aufführen.

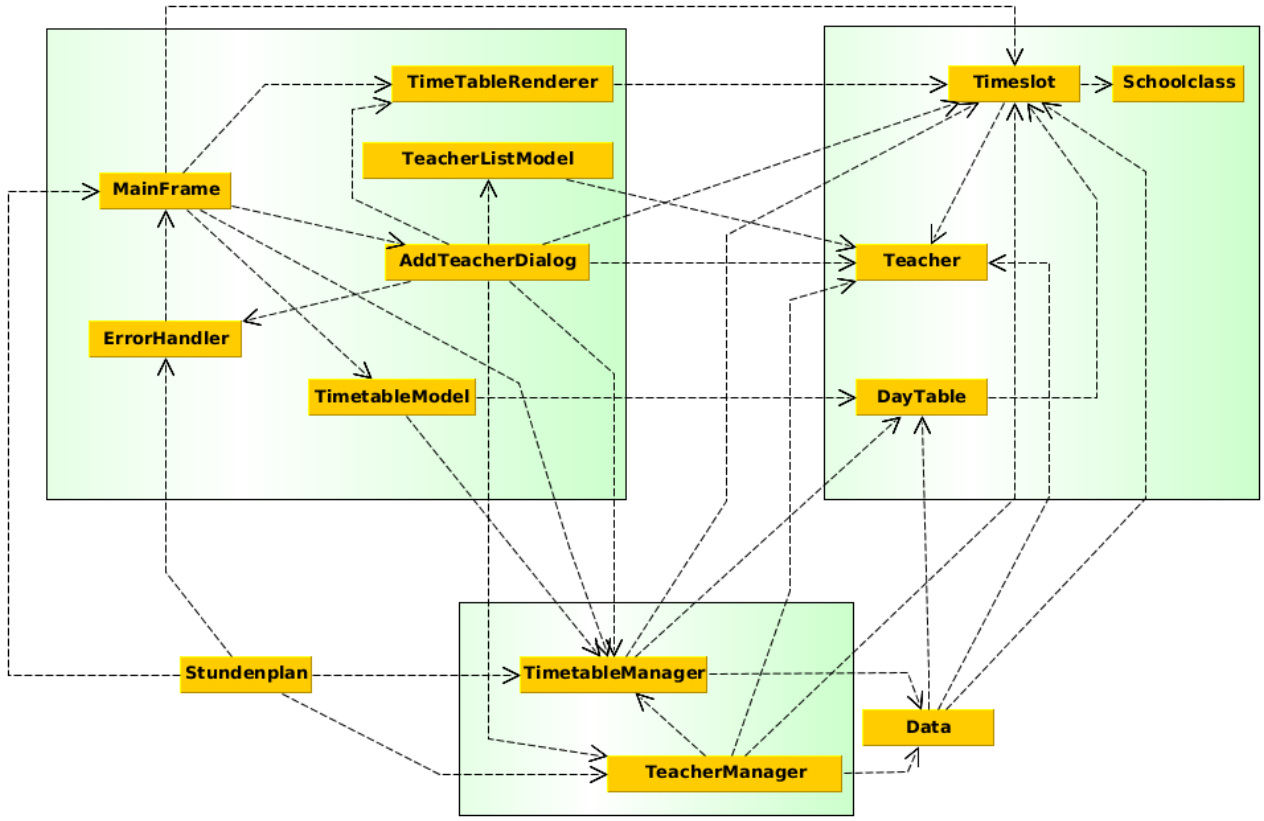
d) Berechnen Sie die **Zweigabdeckung** für die folgenden beiden Aufrufe:

- `typ(80.0, 80.0, 100.0, 100.0)`
- `typ(0.0, 0.0, 0.0, 0.0)`

Begründen Sie Ihre Antwort. Wenn Sie möchten, können Sie dazu den Kontrollfluss als Aktivitätsdiagramm darstellen und dort die Wege der Testfälle einzeichnen; es reicht jedoch auch eine wörtliche Erläuterung mit passenden Argumenten mit Verweis auf die Programmzeilen.

Aufgabe 5 (20 Punkte)

Ein Klassenabhängigkeitsgraph ist ein Graph, dessen Knoten die Klassen eines Programms und dessen gerichteten Kanten die statischen Abhängigkeiten zwischen je zwei Klassen beschreiben. Eine Klasse A ist abhängig von einer anderen Klasse B, wenn im Code der Klasse A auf Eigenschaften (Methoden oder Attribute) von Klasse B direkt zugegriffen wird.
Nachfolgend sehen Sie den Klassenabhängigkeitsgraphen für den Prototypen zu unserem *SWP-Stundenplanprojekt*.



Für diese Aufgabe sind die Cross-Cutting-Concerns mit den Paketen `config` und `exceptions` nicht zu berücksichtigen.
Geben Sie die Reihenfolge an, in der Sie bei einem *minimal inkrementellen* Integrationstest nach der Bottom-Up-Strategie die Klassen integrieren wollen. Beim minimal inkrementellen Integrationstest kommt in jedem Schritt im Integrationstest genau eine Klasse, die noch nicht getestet wurde, zur Menge der bereits zusammen getesteten Klassen hinzu. Geben Sie dabei alle Testrümpfe und -treiber an, die bei diesem inkrementellen Test notwendig sind.
Hinweis: Der Prototyp ist auf http://gitlab.informatik.uni-bremen.de verfügbar. Sie können das Projekt über HTTPS oder SSH auschecken.

Via HTTPS:
`git clone https://gitlab.informatik.uni-bremen.de/koschke/stundenplan.git stundenplan`

Via SSH:
`git clone git@gitlab.informatik.uni-bremen.de:koschke/stundenplan.git stundenplan`

Für SSH ist dabei Folgendes zu beachten: `http://gitlab.informatik.uni-bremen.de/help/ssh`.
Der oben gezeigte Klassenabhängigkeitsgraph wurde mittels Sonar und dem Maven-Goal `mvn sonar:sonar` erzeugt.