Einführung in Software Engineering WS 10/11

Fachbereich Informatik

Dr. Michael Eichberg
eichberg@informatik.tu-darmstadt.de

Assistent: Ralf Mitschke
mitschke@st.informatik.tu-darmstadt.de

TECHNISCHE UNIVERSITÄT DARMSTADT

Übungsblatt 11 (10 Punkte): Observer Pattern

Abgabeformat: Reichen Sie ihre Lösung per SVN ein. Jede Übung muss in einem eigenen Ordner ***ex<Number>*** *(<Number> = 01, 02, ...)* in Ihrem Gruppenverzeichnis eingereicht werden. Während der Übungsbearbeitung können Sie Ihre Lösungen beliebig oft in das SVN hochladen (per Commit). Wir prüfen die Zeit der Einreichung Ihrer Lösungen unter der Benutzung des SVN Zeitstempels.

Erstellen Sie für Lösungen der Aufgaben, die keinen Quelltext erfordern, eine PDF-Datei mit dem Dateinamen ***solution.pdf***. Dies gilt auch für alle UML-Diagramme, die Sie erstellen. Die Basisanwendung wird als Eclipse-Projekt vorgegeben. Ihr eigener Code muss entsprechend in den dafür vorgesehenen Verzeichnissen (***/src*** oder ***/test***) erstellt werden.

Abgabetermin: 09.02.2011 - 24:00 Uhr

Aufgabe 1 (10 Punkte)

Ziel: *Anwendung des Observer Pattern*

Derzeit besteht das primäre Fenster (`FlashcardsWindow`) unserer Flashcards-Anwendung im Wesentlichen aus einigen Steuerungselementen und einer Auflistung der Flashcards der aktuellen Serie. In dieser Aufgabe sollen Sie das Fenster erweitern. Alle Anzeigen sollen sich automatisch während des Lernens anpassen, sobald sich die Daten der Flashcard ändern.

Nutzen Sie im Folgenden das Observer Pattern, um die entsprechenden Klassen über Änderungen an den Daten zu informieren.

a) Anzeige von Metadaten (2 Punkte)

Die Oberfläche soll aufgeteilt werden, um Platz für ein Infofenster zu schaffen (siehe Code Vorschlag für `infoPanel`). In diesem Infofenster sollen die Metadaten der ausgewählten Flashcard angezeigt werden. Abb. 1 zeigt einen Vorschlag, wie dies in der GUI umgesetzt werden kann.

Folgende Metadaten sollten mindestens angezeigt werden: Erstellungsdatum, wann die Karte das letzte Mal nicht erfolgreich und wann sie erfolgreich gelernt wurde und wie oft sie bereits angezeigt wurde.

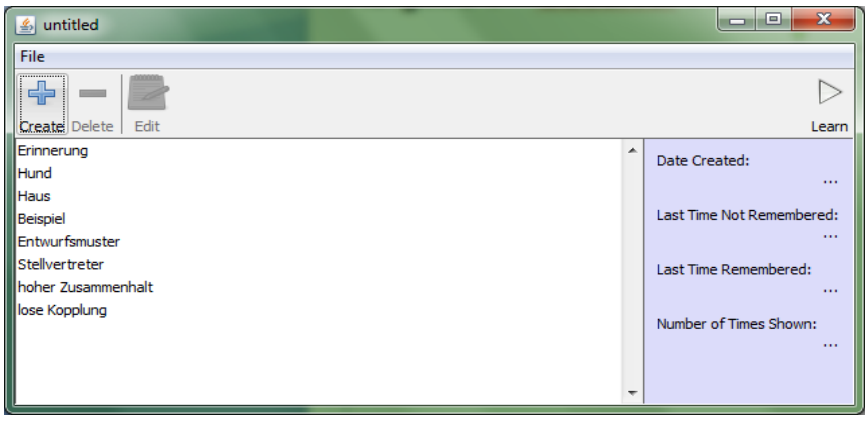

Abb. 1 Beispiel Flashcards-Anwendung mit Anzeige von Metadaten

b) Farbliche Markierung der Flashcards (2 Punkte)

In Abhängigkeit davon wie gut eine Flashcard gelernt wurde, soll diese in der Liste farblich hinterlegt werden. Dabei bedeutet rot, dass man sich an die Antwort der Flashcard beim letzten Mal nicht erinnern konnte. Grün, dass sie die letzten beiden Male gewusst wurde. Gelb, dass man bereits einmal richtig lag. Abb. 2 zeigt einen Vorschlag, wie dies in der GUI umgesetzt werden kann.

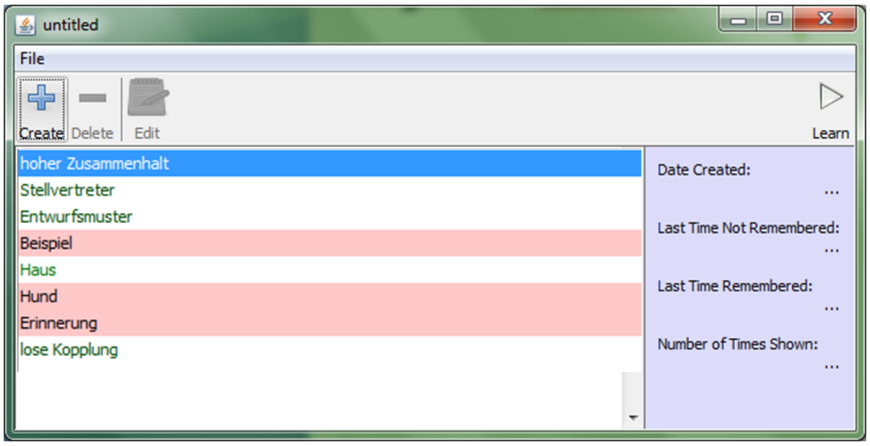

Abb. 2 Beispiel Flashcards-Anwendung mit Farblicher Markierung

c) Nutzerwarnung ungespeicherter Änderungen (1 Punkt)

Falls Änderungen an den Daten (Flashcard, FlashcardSeries oder Metadaten) stattfanden und diese nicht gespeichert wurden, soll der Benutzer beim Beenden der Anwendung gewarnt werden. Bevor die Anwendung beendet wird erhält der Nutzer dann die Möglichkeit diese Änderungen abzuspeichern. Dies kann z.B. über einen Dialog geschehen, der erscheint sobald die Anwendung geschlossen wird und nicht gespeicherte Daten vorhanden sind.

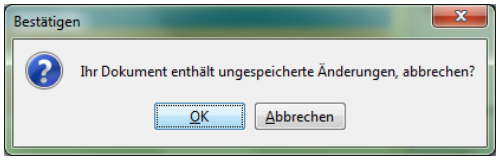

Abb. 3 Beispieldialog zur Warnung über nicht gespeicherte Daten

d) Dokumentation des Designs (3 Punkte)

Erstellen Sie ein UML Klassendiagramm welches Ihre Implementierung des Observer Patterns dokumentiert. Die Klassen sollten alle für das Pattern relevanten Methoden zeigen. Weitere Methoden sind nicht aufzuführen. Notieren Sie welche Klassen welche Rollen des Patterns inne haben (entweder im Diagramm, siehe „Design Pattern – Introduction“ Folie 27, oder extern in Ihrem Lösungsdokument).

e) Design des Auslösens von Benachrichtigungen (1 Punkt)

Im Observer Pattern verlassen sich die Observer darauf benachrichtigt zu werden, wenn sich ein Subject ändert. Es gibt allerdings unterschiedliche Optionen wer die Verantwortlichkeit bekommt, das Schicken einer Benachrichtigung auszulösen. Welche Variante haben Sie in Ihrer Implementierung gewählt. Erläutern Sie kurz den Vor- und den Nachteil dieser Variante.

f) Design des Informationsaustauschs bei Änderungen (1 Punkt)

Ein Observer benötigt meist nicht nur die Information, dass eine Änderung stattfand, sondern erwartet Daten über die Art und den Umfang der Änderung. Zur Übertragung dieser Daten gibt es zwei Grundsätzliche Modelle. Welches Modell haben Sie in Ihrer Implementierung gewählt. Erläutern Sie kurz für Ihre Implementierung welche Methoden von welcher Klasse aufgerufen werden, um Daten auszutauschen. Erläutern Sie kurz den Vor- und den Nachteil dieses Modells.

Code Beispiel für die Konfiguration der GUI

Sie können Folgendenes Codebeispiel zur Implementierung des `infoPanel` verwenden:

```
public FlashcardsWindow(FlashcardSeries flashcards) {
[…]
    JPanel infoPanel = new JPanel();
    // configure the infoPanel
    infoPanel.setOpaque(true);
    infoPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
    infoPanel.setBackground(new Color(220, 220, 250));
    infoPanel.setLayout(new BoxLayout(infoPanel, BoxLayout.PAGE_AXIS));
[…]
    // create labels for metadata
    dateCreatedLabel = createInfoPaneLabel(infoPanel, "Date Created:");
    lastTimeNotRememberedLabel =
        createInfoPaneLabel(infoPanel, "Last Time Not Remembered:");
    lastTimeRememberedLabel = createInfoPaneLabel(infoPanel, "Last Time Remembered:");
    numberOfTimesShownLabel = createInfoPaneLabel(infoPanel, "Number of Times Shown:");
    numberOfTimesRememberedInARowLabel =
        createInfoPaneLabel(infoPanel,Number of Times Remembered in a Row:");
[…]
// create the split pane with list on left and info on right
    JSplitPane splitPane = new JSplitPane();
    splitPane.setResizeWeight(1.0d);
    splitPane.setContinuousLayout(true);
    splitPane.setDividerSize(1);
    splitPane.setLeftComponent(listScrollPane);
    splitPane.setRightComponent(infoPanel);
[…]
    // add the split pane to the window frame
    frame.setJMenuBar(menuBar);
    frame.getContentPane().add(splitPane);
    frame.getContentPane().add(toolbar, BorderLayout.NORTH);
    frame.setSize(640, 480);
    frame.setLocationByPlatform(true);
[…]
}

private JLabel createInfoPaneLabel(JPanel panel, String title) {

    JLabel titleLabel = new JLabel(title);
    titleLabel.setFont(UIManager.getFont("TableHeader.font"));
    Box titleBox = Box.createHorizontalBox();
    titleBox.add(titleLabel);
    titleBox.add(Box.createHorizontalGlue());

    JLabel contentLabel = new JLabel("...");
    contentLabel.setHorizontalTextPosition(SwingConstants.RIGHT);
    contentLabel.setHorizontalAlignment(SwingConstants.RIGHT);
    contentLabel.setFont(UIManager.getFont("List.font"));
    Box contentBox = Box.createHorizontalBox();
    contentBox.add(Box.createHorizontalGlue());
    contentBox.add(contentLabel);

    panel.add(titleBox);
    panel.add(contentBox);
    panel.add(Box.createVerticalStrut(15));

    return contentLabel;
}
```