

72. Dokumentationswerkzeuge

1

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und
Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
<http://st.inf.tu-dresden.de>
Version 12-0.2, 31.01.13

- 1) Aufgaben
- 2) Templategesteuerte Dokumentationswerkzeuge
- 3) Elucidative Werkzeuge

In 2012/13 weggelassen
nur zur Info



46.1 Aufgaben der Dokumentationswerkzeuge

2

http://en.wikipedia.org/wiki/Software_documentation



Ziel der Softwaredokumentation

3

- ▶ Dokumente dienen dem Know-how-Transfer zwischen AG und AN sowie innerhalb des Entwicklerteams
 - Sie retten und bewahren das Wissen über Programme, sind aktuell zu halten und dienen als Quelle für Personen, die die Software weiterentwickeln und warten müssen.
 - Sie dienen als Basis für die Durchführung von Prüfungen (Tests, Audits usw.)
 - Dienen zur Messung des Projektfortschritts (Terminerefüllung, Meilensteine)

- ▶ Basis der Anwendung von Regeln für die Einhaltung von Standards und der Revisionsicherheit.

Man unterscheidet:

- ▶ **Separate Dokumentation**, die nicht direkt Bestandteil der Programme ist. Meist von Hand geschrieben, und schnell veraltet (documentation aging)
- ▶ **Generierte Dokumentation**, die aus dem Programm und seinen Spezifikationen erzeugt wird
- ▶ **Integrierte Dokumentation**, die im Programm, z. B. als Kommentare enthalten ist. Mit Werkzeugen (z. B. JavaDoc) können diese Informationen extrahiert werden und in einer Code-Dokumentation zusammengefasst werden.
- ▶ **Elucidative Dokumentation**, die beides miteinander verbindet und konsistent hält

Zur Gestaltung von Softwaredokumentationen geben folgende Standards Anleitung:

- ▶ auf nationaler Ebene DIN 66230, 66231, 66232, 66270(1998);
- ▶ auf internationaler Ebene ISO/IEC 6592(2000), ISO/IEC 18019(2004)

Aufgaben der Dokumentationswerkzeuge

4

- ▶ Erstellung von **Dokumenten** aus den Ergebnissen bzw. Objekten der Software-Entwicklung in Textform, Graphik, Quellcode u.a.
 - Ermöglichen des Single-Source-Prinzips
 - **Generierbarkeit** der Dokumente aus verschiedenen Quellen (Repository, Quellprogramme u.a.)
 - **Verknüpfung (Verbinden) der Dokumentation** zu Code und Modellen (model mappings)
- ▶ **Strukturierung** von Dokumenten entsprechend Aufbau des Softwaresystems bzw. von Dokumentationsvorschriften
 - Berücksichtigung vorgegebener **Dokumentationsstandards** (z.B. DoD-STD-2167A)
 - Variable **Gestaltbarkeit** der Dokumente nach Struktur, Inhalt, Layout, unterschiedlichen DTP-Formaten (PDF, Postscript, MS-Word,..)
- ▶ **Wiederverwendbarkeit** zur Dokumentenproduktion auch unterschiedl. für Projekte

- ▶ **Benutzerdokumentation** erklärt dem Anwender die Benutzung des Programms
 - Bedienhandbuch
 - Online-Dokumentation
 - Hilfe-Handbuch
- ▶ **Systemdokumentation** zur Installation, Spezifikation der Systemtestfälle und zur Wartung, einschließlich Code-Dokumentation
- ▶ **Projektdokumentation**
 - Entwicklerdokumentation
 - Projektführungsdokumentation (Auftrag, Projektplan, Statusberichte, Abnahmedok.)
- ▶ **Qualitätsdokumentation**
 - Test-, Audit- bzw. Review-Dokumentationen
 - Nachweis der Qualitätsparameter
- ▶ **Prozessdokumentation**
 - Dokumente der Prozessgestaltung (Vorgehensmodelle, Standards, Richtlinien)

Quelle: [24 S. 245 ff.]

46.2 Generative, Templategesteuerte Dokumentationswerkzeuge

6

.. funktionieren mit templategesteuerter
Codegenerierung

Dokumentationswerkzeug *JavaDoc*

7

- ▶ Erstellt aus dem mit Kommentaren versehenem Java-Quelltext automatisch HTML-Dokumentationsdateien
 - aus HTML-Schablonen (templates)
- ▶ Das Werkzeug verwendet die öffentlichen Klassen-, Interface- und Methodendeklarationen und
 - fügt zusätzliche Informationen aus evtl. vorhandenen Dokumentationskommentaren hinzu.
 - Zu jeder Klassendatei xyz.java wird eine HTML-Seite xyz.html generiert.
- ▶ Über zusätzlich generierte Index- und Hilfsdateien wird das Navigieren über verschiedene Links und Querverweise mit anderen Seiten der Dokumentation unterstützt.
- ▶ Über die Steuerung mittels Tags (@) können Metadaten zur Aufbereitung der Dokumente verwendet werden.
- ▶ Das Layout kann nur schwer beeinflusst werden. Inhalt und Struktur der Dokumente werden 1:1 vom Programmquelltext übernommen.
 - Über Doclets ist eine weitere Konvertierung auch in RTF, XML, PDF, Framemaker und Windows Help möglich.
- ▶ JavaDoc gibt es mittlerweile für alle anderen Programmiersprachen



- ▶ Zusammenarbeit mit MS Word-Templates
 - Hardcopy von Diagrammen mit OLE in ein Word-Dokument
 - Der SELECT Document Generator kann ein Dokumentationsgerüst in MS Word generieren
- ▶ Inhalt der generierten Dokumentation:
 - Diagramme des SELECT-Modells, genaue Objektbeschreibungen, leere Abschnitte für Analysen, Berichte oder Zusammenfassungen
 - Auswahl aus vorgefertigten Dokumentvorlagen
 - Auswahl der einzufügenden Abschnitte aus der Vorlage
 - Nachträgliches Einfügen und Entfernen von Abschnitten möglich
 - Aktualisierung des Inhalts durch selektive Synchronisierung mit dem SELECT-Modell
- ▶ Das Layout lässt sich anpassen durch Bearbeiten der Dokumentvorlagen oder des generierten Dokumentes
- ▶ Navigation in den generierten Abschnitten der Dokumentation direkt aus dem Document Generator heraus

- ▶ Aus den Modellspezifikationen sowie den weiteren Informationen (Textdateien, Grafiken) wird eine Modelldokumentation generiert.
- ▶ Modellspezifische Dokumentationsvorlagen
 - Struktur und Layout nach Wunsch bestimmbar.
- ▶ Die im Modellbrowser selektierten Dokumentationsinhalte werden über einen Dokumentationsgenerator erzeugt und in die eigentliche Dokumentation überführt.

- ▶ Ausgabeart:
 - Voranzeigefenster (StandardEinstellung in eigenem Format);
 - PostScript-Dokument;
 - Word für Windows Dokument;
 - ASCII-Text Dokument;
 - XML-Datei
 - Druck-Repository für Druckaufbereitung.

Beispiel Dokumentationsvorlage

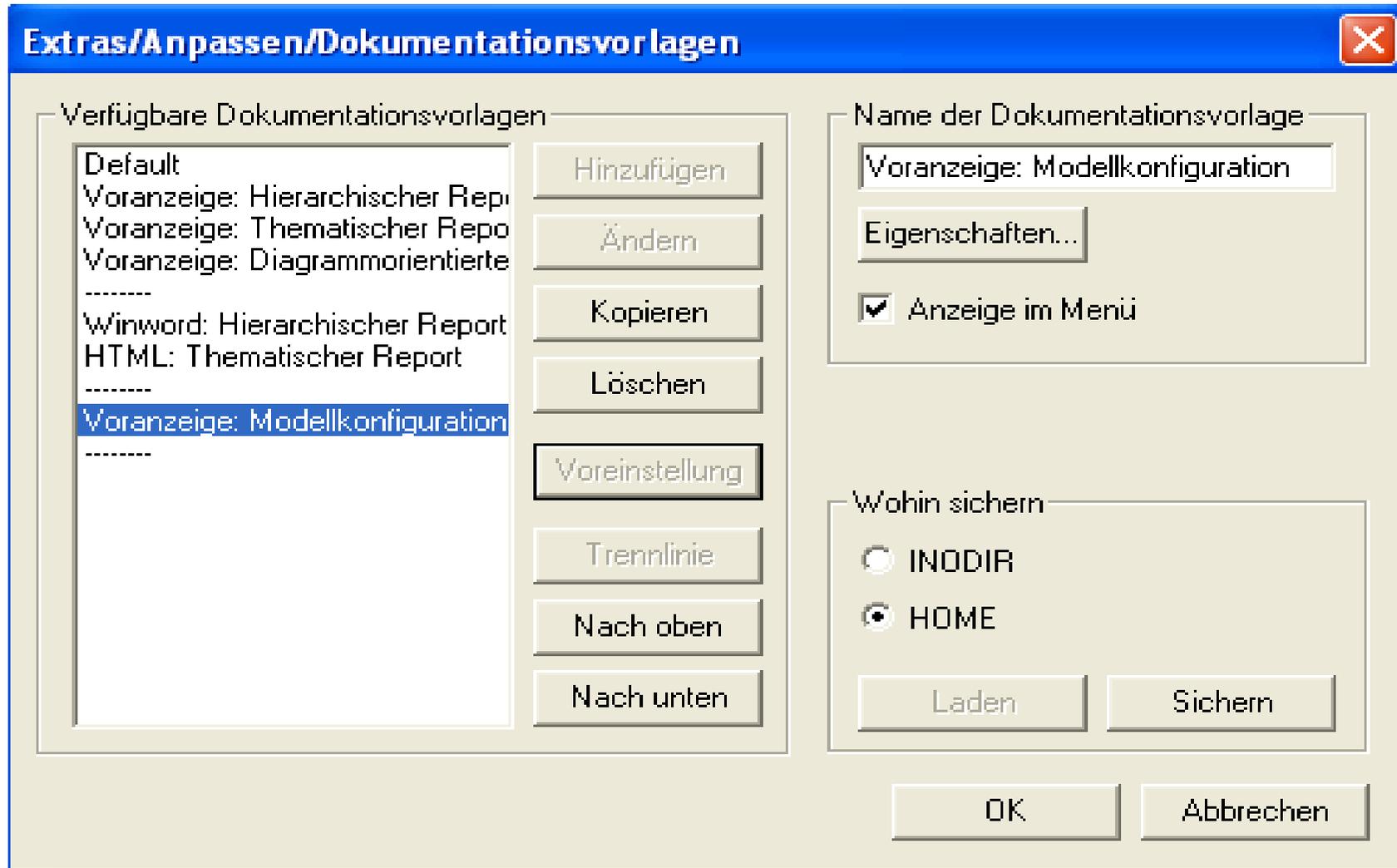
10

The screenshot displays the INNOVATOR Doku-Repository interface. The title bar reads "Doku-Repository 'd:\Programme\Innovator8\inodir\de_de\inodocu.dr' - INNOVATOR". The menu bar includes "Datei", "Bearbeiten", "Ansicht", "Struktur", "Wechseln", "Extras", and "Hilfe". The toolbar contains various icons for file operations and viewing. The main window shows a tree structure for the directory "d:\Programme\Innovator8\inodir\de_de\inodocu.dr".

- Titelseite**
 - Beispiel-Titelseite
 - Beispiel-Titelseite für HTML
 - Beispiel-Titelseite für PHB
- Kopfzeile**
- Fußzeile**
- Kapitel**
- Einstellungen**
- Struktur**
 - Struktur SA/SD**
 - Struktur ERM/SERM**
 - Struktur UML**
 - de_de Diagrammorientiert
 - de_de Hierarchisch (wie INNOVATOR 7)
 - de_de Modellkonfiguration
 - de_de Thematisch
 - de_de Thematisch, HTML optimiert
 - Struktur GPM**
 - Struktur PHB**
 - de_de Projekthandbuch
 - Struktur Meta**
- Auswahl**

Anpassen Dokumentationsvorlagen

11



Beispieldokumente Innovator

12

Voranzeige c:\templ\dr21912

Datei Wechseln Optionen Hilfe

- i -

Inhaltsverzeichnis

1. externes Kapitel	1
1.1. Unterkapitel1	1
1.2. Unterkapitel2	1
1.3. Unterkapitel3	1
2. Doku	2
2.1. systemModel	3
2.1.1. use case system	3
2.1.1.1. Anwendungsfalldiagramm UseCaseDiagram	3
2.1.1.2. Paketdiagramm Create Defaults for Use Cases	4
3. Index	5

innovator - Inhaltsverzeichnis Seite 1 von 1 - Zoom-Faktor: 100,0%

Start | Micr... | 2 N... | 2 h... | Doku... | Wind...

Deckblatt -
Rahmengliederung
als Standard
vorgegeben

Voranzeige c:\templ\dr21914

Datei Wechseln Optionen Hilfe

Seite 3 von 5

2.1. systemModel

2.1.1. use case system

2.1.1.1. Anwendungsfalldiagramm UseCaseDiagram

```
graph LR; Akteur --- UC((UseCase)); UC --- Akteur_1;
```

innovator - Haupttext Seite 3 von 5 - Zoom-Faktor: 100,0%

Start | Micr... | 2 N... | 2 h... | Doku... | Wind... | f Doku... | 2 I... | DE | 15:14

Einbinden eines Use Case Diagramms
unter Punkt 2.1.1.1.

Indexverzeichnis ebenfalls standard-
mäßig erstellt.

46.3 Elucidative Dokumentationswerkzeuge

13

- They link code, models and documentationi by **model mapping**

Tutorial: Beispiel

14



```
public class SalesPointApplication extends Shop {  
  
    public SalesPointApplication() {  
        super();  
    }  
  
    public void start() {  
        super.start();  
        try {  
            Log.setGlobalOutputStream(new FileOutputStream("logfile.log",true));  
        }  
        catch (IOException ioex) {  
            System.err.println("Unable to create log file.");  
        }  
    }  
}
```

Verhalten beim Schließen

Wenn die Anwendung geschlossen wird, öffnet SalesPoint automatisch einen Speichern-Dialog um den aktuellen Status der Anwendung zu sichern. Wenn das nicht gewünscht wird lässt sich dieses Verhalten leicht ändern. Es ist die `quit()`-Methode des Shops zu überschreiben.

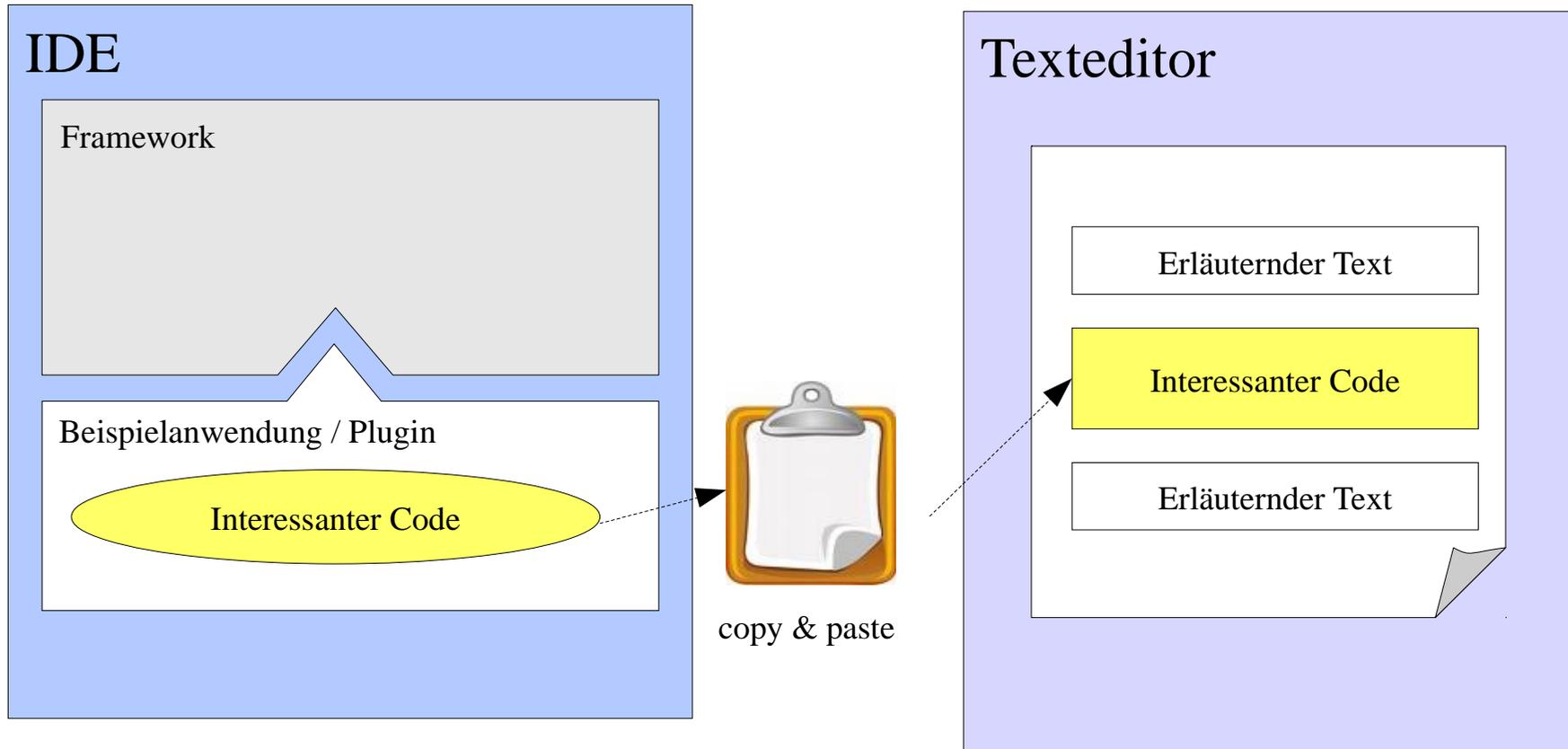
Mit `shutdown(boolean)` wird versucht, alle laufenden Prozesse zu beenden und das Shopfenster zu schließen. Hatte dies Erfolg wird `true` zurückgeliefert. Als Argument wird erwartet, ob der Speicherdialog erscheinen soll oder nicht. Es ist zu beachten, dass das Programm nach dem Schließen des Shops noch nicht beendet ist. Die `exit`-Methode ist manuell aufzurufen.

```
import sale.Shop;  
import log.Log;  
import java.io.*; //IOException, FileOutputStream  
  
public class SalesPointApplication extends Shop {  
  
    public SalesPointApplication() {  
        super();  
    }  
  
    public void start() {
```

Adblock Fertig

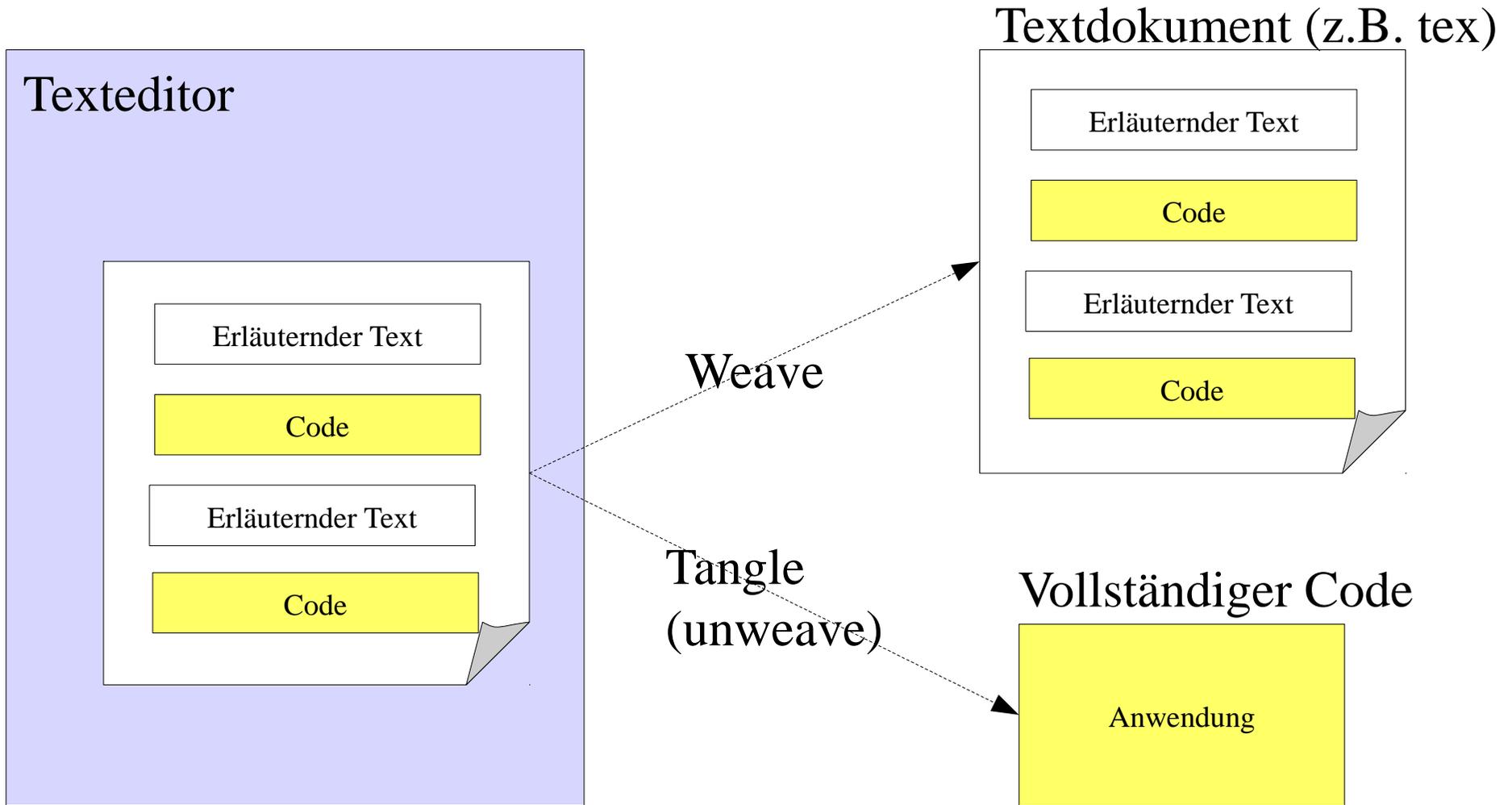


Manuelles Tutorialschreiben



Literate Programming by Code Unweaving

16



Literate Programming

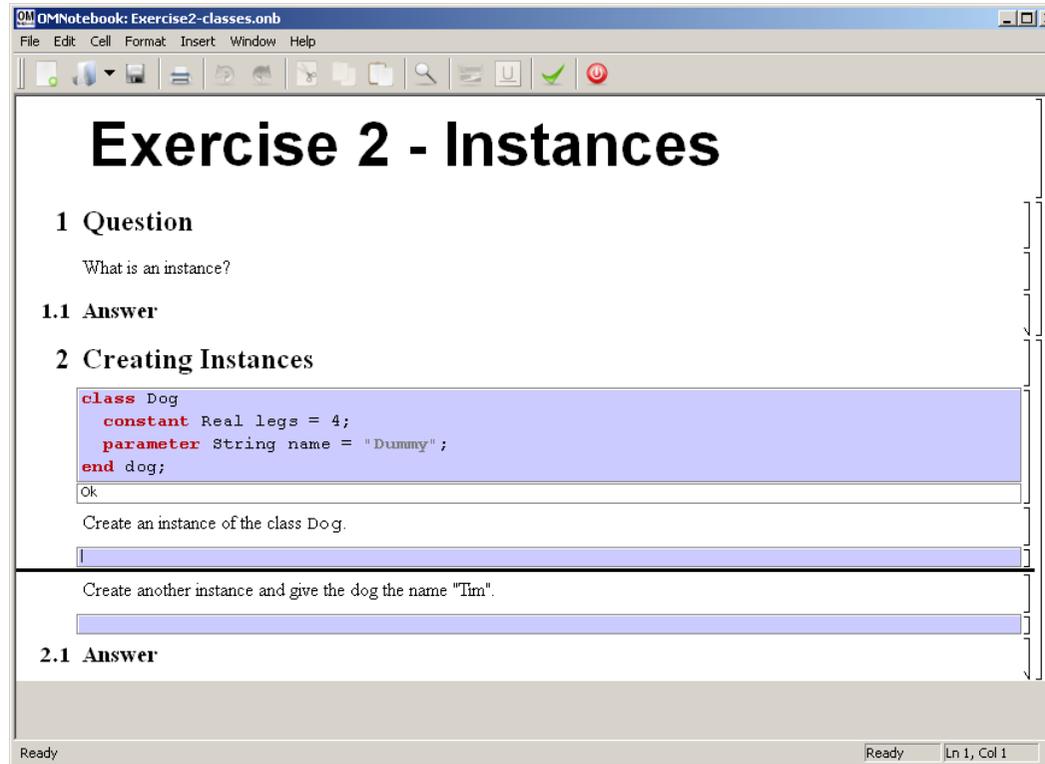
17

[[The program text below specifies the “expanded meaning” of ‘⟨Program to print . . . numbers 2⟩’; notice that it involves the top-level descriptions of three other sections. When those top-level descriptions are replaced by their expanded meanings, a syntactically correct PASCAL program will be obtained.]]

```
⟨Program to print the first thousand prime
  numbers 2⟩ ≡
program print_primes(output);
  const m = 1000;
  ⟨Other constants of the program 5⟩
  var ⟨Variables of the program 4⟩
  begin ⟨Print the first m prime numbers 3⟩;
  end.
```

aus Literate Programming
von Donald E. Knuth

- ▶ Überblick: <http://www.literateprogramming.com/>
- ▶ OMNotebook/DrModelica: <http://www.modelica.org/tools>



The screenshot shows the OMNotebook application window titled "OMNotebook: Exercise2-classes.onb". The window contains the following content:

Exercise 2 - Instances

1 Question

What is an instance?

1.1 Answer

2 Creating Instances

```
class Dog
  constant Real legs = 4;
  parameter String name = "Dummy";
end dog;
```

Ok

Create an instance of the class Dog.

Create another instance and give the dog the name "Tim".

2.1 Answer

Ready Ready Ln 1, Col 1

- ▶ Verlinkte Dokumente mit interaktiven Übungen, teilweise ausführbar
- ▶ Inspiriert von DrScheme und DrJava, Lernwerkzeugen für Scheme bzw. Java

Elucidative Programming Links Documentation with Queries to Code

19

IDE

Framework

Beispielanwendung / Plugin

Interessanter Code

DEFT

Tutorial-Kern

Erläuternder Text

Code Link

Erläuternder Text

Generiertes Tutorial

Erläuternder Text

Interessanter Code

Erläuternder Text

Link

Automatisches Einfügen



Elucidative Programming

20

The screenshot shows the Elucidative Programming environment. At the top, there are several icons for document manipulation and a menu with 'time' and 'general' options. Below the icons, there are three sections: '1 Introduction', '2 The solution', and '3 Post Scriptum'. The '1 Introduction' section is selected and highlighted in blue. It contains the following text:

1 Introduction
In this introductory section we first discuss time system formats and a function in Scheme which returns the current time. Then we discuss the issue of normalization and two possible ways to attack the problem.

1.1 Time systems and functions
1.2 The plan of attack

1.1 Time systems and functions
There are several different standards for representation of time on a Computer. *Universal*

The right pane shows the corresponding Scheme code:

```
;;  
;; Fixed second counts and calendar facts.  
(define seconds-in-a-normal-year 31536000)  
(define seconds-in-a-leap-year 31622400)  
(define seconds-in-a-week 604800)  
(define seconds-in-a-day 86400)  
(define seconds-in-an-hour 3600)  
(define base-year 1970)  
(define month-length-normal-year  
  (vector 31 28 31 30 31 30 31 31 30 31 30 31))
```

- ▶ <http://www.cs.aau.dk/~normark/elucidative-programming/>
- ▶ <http://deftproject.org>

Development Environment For Tutorials (DEFT www.deftproject.org)

21

The screenshot displays the DEFT development environment with three main components:

- Projektübersicht (Project Explorer):** Shows a tree view of the 'Fahrkartenautomat' project, including folders for 'Chapters', 'Code files', 'Code snippets', 'Images', and 'Tutorials'. The 'Code files' folder is expanded to show various source files like 'FahrkartenAutomat.cs', 'Fahrschein.cs', 'SecurityIO.cs', etc.
- Texteditor:** Displays the source code for 'FahrkartenAutomat.cs'. A code fragment is highlighted in yellow:

```
try
{
    FileStream fs = new FileStream("test.log",
        FileMode.Create);
    Log.Log.GlobalOutputStream = new
        System.IO.BinaryWriter(fs);
}
catch (IOException)
{
    System.Console.Error.WriteLine("Unable to
        create Log file.");
    return ;
}
```

The text below the code explains: 'Es wird versucht, die Datei test.log zu erzeugen. Falls sie schon existiert, wird sie überschrieben. Der FileStream fs kann Daten (Bytes) vom Programm in die Datei schreiben. Byteweises Schreiben von Informationen ist allerdings sehr umständlich. Ein BinaryWriter kapselt den FileStream und bietet Methoden zum Schreiben von Strings, Zahlen, und Anderem. Der globalen Log-Klasse der Anwendung, Log.Log, wird dieser BinaryWriter zugewiesen. Alle Logzugriffe erfolgen von nun an über ihn und damit in die Datei test.log.'
- AST-Fenster (Code Outline):** Shows the Abstract Syntax Tree (AST) for the project, listing elements like 'UsingDirectives', 'FahrkartenAutomat', 'TICKETS', 'MONEY', 'DefaultMenuSheet', 'FahrkartenAutomat(string)', 'createDisplayManager()', 'Main(string[])', 'FahrscheinVerkaufenAction', and 'WartungAction'.

Prof. U. Alßmann, Softwareentwicklungswerkzeuge (SEW)



Development Environment For Tutorials

22

- ▶ Eclipse RCP-Anwendung
- ▶ Sprachunabhängig
- ▶ Verwaltung von Text und Code unter einem Dach
- ▶ Automatisches Prettyprinting von Codefragmenten
- ▶ Hilfe bei der Aktualisierung
 - Automatische Aktualisierung von eingebetteten Codefragmenten
 - Benachrichtigung bei veränderten Codefragmenten

Generiertes HTML Tutorial

23

Prof. U. Alßmann, Softwareentwicklungswerkzeuge (SEW)

Start der Anwendung

In der Klasse `FahrkartenAutomat` befindet sich die `Main`-Methode, mit der sich das Programm starten lässt. Dort werden Daten initialisiert und der `FahrkartenAutomat` instanziiert.

Logging

Der erste Schritt ist die Konfiguration des Loggings. Das `SalesPoint`-Framework bietet Funktionen und Datentypen an, mit denen Aktionen geloggt werden können. Es gibt GUI-Komponenten, mit denen die Inhalte des Logs wieder nutzerfreundlich angezeigt werden können. Eine Anzeige des Logs ist derzeit nicht im `FahrkartenAutomaten` implementiert, geloggt wird aber trotzdem schon.

Um Logging zu verwenden, muss ein Stream auf die Logdatei geöffnet werden.

```
try
{
    FileStream fs = new FileStream("test.log", FileMode.Create);
    Log.Log.GlobalOutputStream = new System.IO.BinaryWriter(fs);
}
catch (IOException)
{
    System.Console.Error.WriteLine("Unable to create Log file.");
    return ;
}
```

Es wird versucht, die Datei `test.log` zu erzeugen. Falls sie schon existiert, wird sie überschrieben. Der `FileStream fs` kann Daten (Bytes) vom Programm in die Datei schreiben. Byteweises Schreiben von Informationen ist allerdings sehr umständlich. Ein `BinaryWriter` kapselt den `FileStream` und bietet Methoden zum Schreiben von Strings, Zahlen, und Anderem. Der globale Log-Klasse der Anwendung, `Log.Log`, wird dieser `BinaryWriter` zugewiesen. Alle

```
{
}

protected override DisplayManager createDisplayManager()
{
    Size d = System.Windows.Forms.Screen.PrimaryScreen.Bounds.Size;
    Point tempAux = new Point((d.Width - 100) / 2, (d.Height - 80) / 2);
    Point tempAux2 = new Point(5, 5);
    return new AWTDisplayManager(this, ref tempAux, ref tempAux2);
}

[STAThread]
public static void Main(string[] args)
{
    //System initialisieren
    try
    {
        FileStream fs = new FileStream("test.log", FileMode.Create);
        Log.Log.GlobalOutputStream = new System.IO.BinaryWriter(fs);
    }
    catch (IOException)
    {
        System.Console.Error.WriteLine("Unable to create Log file.");
        return ;
    }

    // Kataloge anlegen

    // Fahrscheinkatalog
    Catalog cTickets = Catalog.forName(TICKETS);

    cTickets.addItem(new Fahrschein("Einzelfahrt", 300));
    cTickets.addItem(new Fahrschein("Sammel Fahrchein", 1500));
    cTickets.addItem(new Fahrschein("ermäßigte Einzelfahrt", 150));
}
```



The End