Design Patterns with Fujaba
Intense Course, 5th-9th October 2009

Ruben Jubeh
ruben.jubeh@uni-kassel.de

Kassel University
Department of Software Engineering
Wilhelmshöher Allee 73
34121 Kassel
Germany
Timetable

Monday, October 5th:
  10.15-11.45, room 405
  12.00-13.30, room 403
  14.15-15.45, room 405 (Exercise)

Tuesday, October 6th:
  10.15-11.45, room 404
  12.00-13.30, room 404
  14.15-15.45, room 405 (Exercise)

Friday, October 9th:
  10.15-11.45 room 402
  12.00-13.30 room 315
  14.15-15.45 room 315 (Exercise)
How this course is organized

- Interactive :) That means: questions desired
- 5 Points for participating (at least at two days)
  - => Name tags please
- Bring your Laptop etc. to follow the exercises
- Lecture part (50%):
  - Design Patterns
  - Introduction of Fujaba4Eclipse
- Interactive part (25%):
  - Live Coding of Design Patterns in Eclipse
  - Live Modeling in Fujaba4Eclipse
  - DIY simultaneously, or in the exercise
- Exercises (25%):
  - in-class-work, presentation at 15:15-15:45 ...
How this course is organized (2)

- Homework
  - work on homework (and exercises) in pairs!
  - use Eclipse/Java or Fujaba4Eclipse and the supplied workspaces!
  - Me/Artjom are helping you on problems and questions during the exercise session
  - Homework submissions by mail to: artjom85@ut.ee, up to Monday 12th October 12:00 noon
    - Single workspace or project zip per submission containing all Homeworks done
    - Use homework[1-3] as package resp. Class diagram name
    - If done correct, gives another 5 points (1+2+2)

- Lectures and Demos are recorded, will be published together with my workspace after each session!
What you should know before starting with the exercises

- Object Orientation Principles, like inheritance, (delegation), encapsulation etc.
- UML (2.x): Class- and Object-Diagrams
  - Introduced: Story Diagrams (Activity + Object / Graphtransform.)
- Java Programming
  - Basic Collection API: List, Iterator... foreach-loop
  - (Basic Swing API Knowledge)
- Usage of Eclipse
  - Work with Java Projects
- How to Debug
  - Stepping, Variables
Literature

Basics:

⌧ E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns (Elements of Reusable Object-Oriented Software); Addison Wesley (1994)

⌧ Patterns Home Page: http://hillside.net/patterns/

Fujaba (http://www.fujaba.de/):

A. Zündorf: Rigorous Object Oriented Software Development with Fujaba. Draft Version 0.3. Extended version of [Zün01] (2002)

Background:

⌧ Frederick P.\ Brooks: The Mythical Man Month, Addison Wesley 1975 (rather short but amusing)
Overview

- General Design Principles & Design Patterns Introduction
- Modeling with Fujaba4Eclipse
- More Design Patterns... with live coding
  - Strategy Pattern
- Fujaba4Eclipse: Modelling Methods
  - Simple Control Flow, Statement Activities
  - Creating and Deleting Objects
- More Design Patterns:
  - Composite Pattern (with Fujaba4Eclipse)
  - Visitor Pattern (in Java)
- Exercise 1: Strategy Pattern
- Homework 1: Implement Composite and Visitor Pattern in Java or Fujaba4Eclipse
You will be prepared for...

- Design Know How for large Applications > 100 000 LOC
- Basic principles of maintainability and extensibility of software applications
- Use Inheritance and Delegation where it fits best
- Basic knowledge of the most common Design Patterns
  - You will notice them everywhere :)

- Model driven Software Engineering with Fujaba4Eclipse
  - Using generated code
  - Graphical Debugging
Architecture & Design Priciples

- A certain functionality or design aspect should be placed at one place in the code, not distributed
  - only local changes needed
- Local changes should affect only local classes/methods, interfaces, visibility, ... Use loose coupling of components
- Keep the software open & flexible
  - Requirements might change
- Reuseable, changeable and understandable
  - Software / Projects evolve...
- Use inheritance as little as possible, do not overengineer!
Design Patterns

- Origin: Architectural patterns for designing buildings, cities...
  - Christopher Alexander, 1977
- Kent Beck adapted this for software design... 1987
  - Result was the GoF book, 1995
  - Pattern community
- Provide solutions to frequent design problems
- „Designer Experience“
- Like templates, need to be applied to the concrete problem
- Not: ready-to-use classes, components...
What are Design Patterns?

"A description of an object-oriented design technique which names, abstracts and identifies aspects of a design structure that are useful for creating an object-oriented design.

A design pattern identifies classes and instances, their roles, collaborations and responsibilities. Each design pattern focuses on a particular object-oriented design problem or issue.

It describes when it applies, whether it can be applied in the presence of other design constraints, and the consequences and trade-offs of its use."

E. Gamma et al (1995): Design Patterns, Addison-Wesley
Reference Architecture of interactive Systems (e.g. Word, Eclipse, Fujaba...)

- GUI: Input (Commands)
- Data Model
- Repository / Persistency
- Import / Export
- Generators / Interpreters
- GUI: View (Unparsing)
- QVT (Control)
Motivating Example:

- Tree of Objects (various type)
- Realizes hierarchic data structures
- Widely used in almost every software application
- Naive realization leads to unmaintainable code
  - Root, Drives, Folders, Files...
- We will discuss it later in depth:
  - Composite Pattern
- Many variations and extensions of this pattern...
Inheritance

- Substitution principle: Child must do like parents
- One can always use a derived class instead of the parent class
Live Implementation of the Customer Example:

```java
... prod = e.work(t);
...```

...
Delegation:

```
this.alarm();
```

```
:House
:FlashLight
:Siren
:MailAlarm
```
From Fujaba ...

- **Means:** From **UML** to **Java And Back Again**
  - Intention: Code generation and Reverse Engineering

- **Class Diagram Editor**
  - Static / structure of your program
  - Code Generator generates Java, C++, EMF...

- **Story Driven Modeling**
  - Combination of Activity Diagrams and Object Diagrams
  - aka Graph Rewrite System, aka Story/Activity Diagrams
  - Dynamic / Behavioural specification of your program
  - Code Generator generates executable Java sourcecode

- **Fujaba is/was a legacy standalone Swing Application**
  - User programs are difficult to debug
  - Was often used in combination with an IDE

- **Large codebase of extension Plugins (Realtime, Diagram types, MOF)**
SDM Method / Activity Diagram:

- **Person**
  - `full` : Boolean
  - `name` : String
  - `eat` : Void

- **Table**
  - `0..n` on
  - `0..1`

- **Student**

- **Meal**

- **Person**: `eat()`: Void
... towards Fujaba4Eclipse

- Integration in Eclipse gives
  - JDT, Debugger, Compiler, Run environment
- A Fujaba Project/Model is Part of an Eclipse Project, one may have many models in one Project
- Unfortunately, different GUIs exist:
  - GEF-Based Eclipse Editor for Class- and Activity-Diagrams
  - We use: Swing-Adapted GUI
eDOBS

- (eclipse) Dynamic Object Browsing System
  - Shows Heap / Object structures
- Like the Variables View in the eclipse Debugger, but:
  - Graphical representation of Objects as Boxes
  - Pointers are links between objects
- Attributes- and Method-View
  - Attribute values can be changed
  - Methods can be called interactively
- BeanShell-Integration allows scripting & more interaction
Invoke eDOBS (Perspective)
eDOBS Debug Perspective
Singleton Pattern

- Exactly one instance: e.g. FileSystem, Computer, Root Node of a Model
- Globally accessible, each caller should get the same instance
  - => protection against multiple instanciation
- Inheritance should be possible
  - Problem here: Where is the Instance created?
- When using many Singletons, a SingletonRegistry might be useful
Live implementation of the Singleton Pattern in Java
Singleton Pattern Specification

- Known Usage:
  - java.lang.System (not strictly speaking)

- Consequences: (Benefits and Drawbacks)
  - It is tempting to use this pattern when looking up references is difficult...
  - But beware: What if it turns out the singleton is no singleton anymore?
Preparation for Exercises

- Download and unpack Fujaba4Eclipse from
  http://www.se.eecs.uni-kassel.de/fileadmin/se/projects/Fujaba4EclipseKassel/
  - Or use the circulating USB Sticks...
  - Or use the Eclipse update Site in combination with any existing Eclipse 3.5 (unsupported):
    http://www.se.eecs.uni-kassel.de/fileadmin/se/projects/Fujaba4EclipseKassel/update/
    (Ensure that you select Fujaba4Eclipse Kassel Release)

- Download and **unpack** the workspace for this Exercise

- Start Eclipse and select the unpacked workspace
  - You should see a Project „DesignPatternsCourse“ without any errors

- The src-folder contains handwritten code, the generated-folder contains code generated by Fujaba4Eclipse

- Don‘t hesitate to ask if something goes wrong!
Project Usage

- Open the model/DPExamples.ctr
- Switch to the Fujaba4Eclipse Perspective
- Open the Tree under DPExamples.ctr
- Doubleclick a Class-Diagram to open it
- Create a new class: Right-click on the diagram ...
- Use the export button or Menu -> „Import / Export“ -> „Export All Classes to Java“ to regenerate the code
- Verify that your new Class is inside the generated-folder
Back to the simple solution: case distinction

public void alarm ()
{
    if (mode == SIRENE)
    {
        ...
    }
    else if (mode == FLASH)
    {
        ...
    }
    else if (mode == MAIL)
    {
        ...
    }
    ...

Bad! Don‘t do that at home!

Why?
• try to use the alarm in your car
• Add a method like shutdown()
DP #2: Strategy Pattern

```
Run()  |  Alarm()  |  Siren()  |  MailAlarm()
House  |  AlarmDevice  |  FlashLight |  Siren |  MailAlarm
       |            | alarm()    | alarm() | alarm()
```

```
| run() | alarm() |
| House |
|       |         |
```

```
| 1     |          |
|       | AlarmDevice |
|       |             |
```

```
|       |          |
|       | FlashLight |
|       | alarm()   |
|       |           |
```

```
|       |          |
|       | Siren    |
|       | alarm()  |
|       |           |
```

```
|       |          |
|       | MailAlarm|
|       | alarm()  |
|       |           |
```
Live modeling:

adding SilentAlarm
DP #2: Strategy Pattern Specification

- Use inheritance to subclass concrete strategies
- Use delegation to forward a request to a certain strategy

- Consequences:
  - Eliminates if/elsif/else...-chains
  - Strategy can be changed at runtime
  - Adding a new ConcreteStrategy to the design is easy

- Known Usages:
  - Comparator for sorting Lists etc.
  - Swing Layout manager (Flow-, Box-, Grid-Layout)
Modeling methods in Fujaba4Eclipse Demo (with HouseAlarm)
Modeling Methods with Fujaba4Eclipse

```
StartupApplication::main (args: StringArray): Void
```

```
House
  Create / Goto Method Body
    Edit Selected Method ...
    Create / Edit Class ...
    Create / Edit Attributes ...
    Create / Edit Methods ...
  =>
```

```
Successor
  Append Stop
    Append Story Pattern With this Object
    Append Story Pattern With Parameters
    Append Empty Story Pattern
    Append Statement Activity
    Append Stop
```

Fujaba Modeling & Design Patterns
© 2009 Ruben Jubeh, Albert Zündorf, University of Kassel 33
Story / Activity Diagrams

- Start -> Control Flow -> Stop
  - Transitions in between

- Statement Activity for plain Java Code

- Story Pattern (aka Activity) contain Object diagrams
  - Specify a Graph Transformation
    - Black: match
    - Green: create
    - Red: destroy

  - Objects might have modifiers:
    - Create, Destroy
    - “bound“ Objects omit the Type

  - Links between objects
    - Same modifiers - create (implicit), destroy
Advanced Modelling

- Success- and Failure-Transition
- Each-time-Activity for iterations
  - Doubleclick or Edit Story Activity
  - Might use each-time- and end-for-all-Transition
- Collaboration statements
  - Number for Timeflow, enter method
- Constraints

- You can verify whether the method does what you expect by looking at the generated source code (at first)!
Composite & Visitor Pattern - Motivation

- We want to implement the unix command „find“
  - Search all drives, all directories and all files
  - Write the list of files with full path to system output
- Solution with iteration and recursion is easy. Just pass the preceding path on depth-first-recursion
  - But: handling for each type in the tree required!
- Another listFiles() - now with just preceding spaces
  - Copy the method
  - Modify the output...
  - Still easy?
  - Error-prone!
Implementing
Composite Pattern
Live (for FileSystem)
DP #3: Composite Pattern Specification

- Uniform interface for traversal into tree / part-whole structures
- Common superinterface for containers and contained (leaves), which
  - Declares composite methods
  - Leaf operations delegated

- Implementation:
  - Where are children stored?
  - Parent reference?
  - add()/remove() implementation for leaves?

- Known Uses:
  - Widget Library Class Hierarchies (e.g. Swing)
Exercise 1

- Implement the single Alarm / delegation example from slide 16 in Java:
  - Use the singleton pattern for the class House
- Write a method, which instanciates all types of alarms, adds them individually to the house and calls `house.alarm()` each time (Strategy Pattern)
- Model the same example in Fujaba4Eclipse (hint: use a different package name)
- Change your `main()`-Method to use the generated code (or model one)
- Use the eDOBS to visualize your object structure!
  - Can you observe any differences between plain java and generated classes?
Live modeling
of Exercise 1

cardinality here? 1 ? n ?
we start with 1 ...
Homework 1:

- Implement Abstract-, RegularFile and Directory using the Composite Pattern as shown before (either in plain java or as Fujaba4Eclipse model, assume unix-style filesystem, you might omit Root/Drive)

- Delegate RegularFile(String name), child computation, getName() and getSize() to java.io.File in the class RegularFile

- Implement findAll() and countFiles() like shown in the lecture / session 3! How many files are there?

- Implement findAll(String extension) - should list only files with the given file extension. How can this be combined with countFiles()? Find a solution that prints just the number of .java-Files!

- Count the files in the current directory by calling traverse(new RegularFile(“.”, visitor); using the CountFilesVisitor. How many are there?

- Write a SummarizeFileSizesVisitor! What is the summarized size of the files in the current directory?

- Advanced: Add a file extension filter to the CountFilesVisitor. How many *.java-Files are there?