Fujaba Modeling & Design Patterns
Intense Course, 4th-8th May 2009

Ruben Jubeh
ruben.jubeh@uni-kassel.de

Department of Software Engineering
Wilhelmshöher Allee 73
34121 Kassel
Germany

Overview

- Introduction
- General Design Principles & Design Patterns Introduction
- Modeling with Fujaba4Eclipse
- **More Design Patterns...**
  - With live coding
  - More modeling techniques (behaviour)
- **Exercises ...** Implement and model patterns yourself

About Points...

- Be here 3 times and do 3 exercises in class - 5points
  - 4 exercises, 6points
  - 5 exercises, 7points
- Bonus homework 1: TextualFileTree (until Friday) - 1point
- Bonus homework 2: SwingFileTree (until Friday) - 1point
- Bonus homework 3: SyncAssocs (until Friday) - 1point
- Friday‘s exercise (can send me your solution as **zipped workspace** up to Monday, 11th May: ruben.jubeh@uni-kassel.de) : 2p
- 10pts at max!

What we have learned yesterday

- Design Patterns
  - Template Method
  - Observer / Listener
  - Factory Method
  - Abstract Factory
  - Prototype
- Fujaba4Eclipse
  - PropertyChange-Mechanism

Exercise 4

- Extend your FileSystem implementation:
  - Use the Factory Method Pattern to create Directory/RegularFile.
  - Hint: If you implemented the delegation to java.io.File, one Factory Method with a Parameter might be enough!
- Add the Sensor class to the HouseAlarm Example and implement the Observer Example, but
  - Use the Java-Terminology
  - Start with plain java if you‘re unsure
  - Model it with Fujaba4Eclipse ...
  - Use the <<JavaBean>> Stereotype on Sensor!

Model of Exercise 4:

Observer / PropertyChange...

with HouseAlarm

Timetable 8th of May

- Design Patterns:
  - Proxy
  - Command
  - Inversion of Control
  - Dependency Injection
- Implementing Plugins
- Break
- Exercise 5: Implement a 2D-Draw-Application

Proxy Pattern

- Similar to Decorator
  - Expose the same interface and delegate...
- Known Uses:
  - Remote Objects
  - Logging
  - Authentication & Authorization
  - Lazy Loading (Virtual Proxy)
- Example
  - Resource Management

Implementation of the Proxy Pattern

Command Pattern

- Aka Operation (in Eclipse)
- Encapsulate a Operation in a Object instance
  - Decouple invocation and execution
- Stored for undo/redo
- Known Uses:
  - Programs with Graphical User Interface, e.g. Eclipse
- Example:
  - GUI - Button - Action
  - Increment a Counter

Implementation

of the Command Pattern

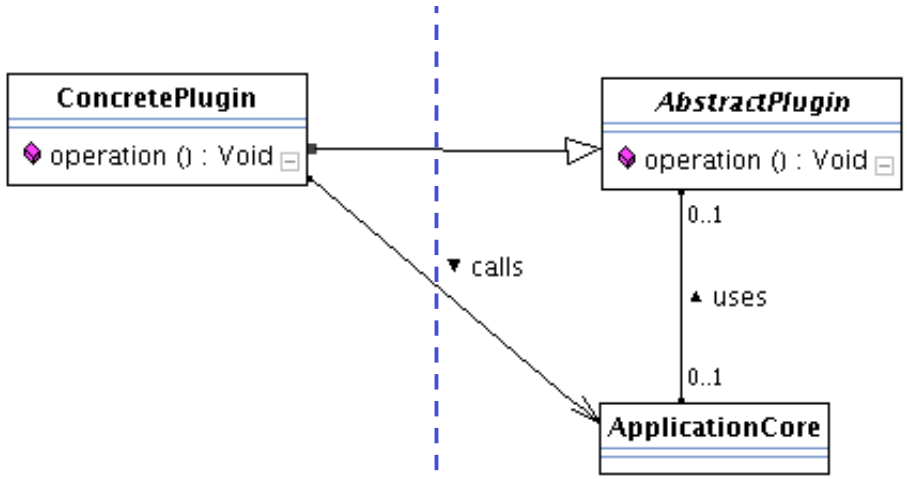
Inversion of Control / Dependency Injection

- Aka Hollywood Principle: „Don‘t call us, we call you!“
- Occurrences:
  - Observer Pattern, Callbacks, or even Plugins
  - Component based Frameworks
- Example: Enterprise Business Logic Component
  - Framework defines lifecycle
  - Framework injects DataSource, coupled Objects

Implementation of the Business Logic Example

Plugins

- A Plugin adds functionality to a application core => Plugin depends core
- The core should have no compile-time-depenceny to it‘s plugin! => Different compilation units



- Plugin calls Core -> Simple
- Core calls Plugin?
  - Only through an Interface!
  - Needs some reflective initialization code...

Design Patterns recapitulation

- Creational Patterns
  - Singleton
  - Factory Method
  - Prototype
- Structural Patterns
  - Adapter
  - Composite
  - Decorator
  - Proxy
- Behavioral Patterns
  - Strategy
  - Chain of Responsibility
  - Hook
  - State
  - Visitor
  - Template Method
  - Command
  - Observer
  - Iterator

Recapitulation of all the Rest

- Fujaba4Eclipse
  - Class Diagrams
  - SDM / Activity Diagrams
  - Code Generation...
  - PropertyChange Mechanism
- Extra Patterns:
  - Inversion of Control
    - Dependency Injection
- „Plugin Pattern“

Exercise 5 - Part 1

- Look at the Code in *dpexamples.day5.drawapp*-Package:
  - Start the DrawApplication
  - Draw Lines by dragging
  - Draw Rectangles with the Alt-Key pressed

- Identify Patterns in the Code:
  - add a Comment saying „This is Pattern X“, maybe explain what pieces of the code participate (Methods, Classes etc.)
  - Hint: There are at least 3... One from Day1, another from Day3 and finally one from Day4

Exercise 5 - Part 2

- Refactor Code, apply Patterns
  - Find at least one place in the Code, where the Strategy or State Pattern can be applied
  - Change that: Implement State or Strategy there!
- Extend Functionality
  - Simple: By pressing Shift, it should be possible to draw Ovals.
  - Advanced: Implement a OvalGraphicsPlugin. The existing Code must not have any dependency to the plugin code, except one String reference is allowed!

Exercise 5 - Part 3

- Implement a simple Select Mechanism:
  - When clicking on the DrawPanel, collect one (easy) or all (advanced) AbstractGraphics instances under the Cursor with a Visitor
  - Remember the selected Objects in DrawApplication...
- Implement Actions and Operations for the five Toolbar-Buttons: Cut, Copy, Paste, Undo, Redo, each one as seperate Class
  - Use the Command Pattern, sublass *AbstractOperation*
  - Create Command instances with the FactoryMethod Pattern
- Basic: Implement a ActionDecorator for logging Actions: Use *System.out.println(...)*
- Advanced: Group your Operations using the Composite Pattern:
  - Multi-selection operation should result in one undo/redo step!

Exercise 5 - Part 4 and Hints

- Replace the whole model by Classes modeled with Fujaba4Eclipse!
- Advanced: Model Operations as well...

- **Email me your zipped workspace up to Monday!**
  - ruben.jubeh@uni-kassel.de
- Part 1, 2 and either 3 or 4 : 2 Points
- Part 1, 2 : 1 Point

Thank you for

Listening & Attending!