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 homework1:TextualFileTree (until Friday) - 1point
- Bonus homework2:SwingFileTree (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

- More Design Patterns...:
  - DP #5: Adapter Pattern
  - DP #6: Strategy and Hook Pattern
  - DP #7: Chain of Responsibility
  - DP #8: State Pattern
- Fujaba4Eclipse: more on Modeling Methods
- Exercise 3...

Timetable 7th of May

- **Discussion of Exercise 3**
- Design Patterns
  - Template Method (Example?)
  - Observer / Listener
- Break 1
- Design Patterns
  - Factory Method
  - Abstract Factory
  - Prototype
- Break 2
- Fujaba4Eclipse: PropertyChange-Mechanism
- Exercise 4

Exercise 3

- Start with your model of exercise 1 or the given workspace.
- Modify the HouseAlarm to use the Chain-of-Reponsibility-Pattern.
- What could one do if no AlarmDevice is responsible? implement a solution that ensures that the alarm() won‘t get lost
- Let MailAlarm.alarm() throw a *new RuntimeException(“Mail server down!“)*. Implement the state pattern by adding some logic which switches to the Siren then. Where should the try/catch be? Where is the strategy exchange?

Model of Exercise 3:

CoR - „next“ vs to-many

Exception handling

Implementation of Exercise 3: State Pattern in Alarm with eDOBS

Timetable 7th of May

- Discussion of Exercise 3
- Design Patterns
  - Template Method
  - Decorator
  - Observer / Listener
- Break 1
- Design Patterns
  - Factory Method
  - Abstract Factory
  - Prototype
- Break 2
- Fujaba4Eclipse: PropertyChange-Mechanism
- Exercise 4

Template Method Pattern (aka Single-Hook)

- Motivation
  - Algorithms with a fixed outer structure
  - But individual steps might vary
- Example
  - ImageConverter:
    - Collect all Files in a Directory
    - Open, read, convert to Bitmap
    - Transform
    - Write, Compress...

Live implementation

of

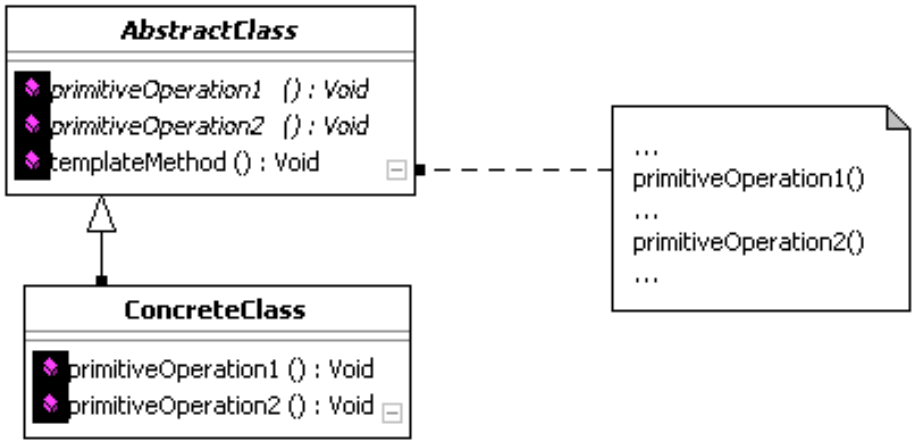
ImageConverter

Using the

Template Method Pattern

Template Method Pattern : Specification

- Outer, general algorithm
- Specialized steps



- **Consequences:** Benefits and drawbacks
  - Helps to have a defined structure
  - No object-level composition
- **Known Uses:** probably any abstract class :)

Decorator Pattern

- Template Method doesn‘t help if we need to mix things
  - Combine different open, transform and write operations
    - (GeneralFileConverter)
- A deeper look into the transform() of the ImageConverter
  - Change Color-depth
  - Add Watermark
  - Adjust Brightness/Contrast
- Idea behind the Decorator:
  - Keep the interface (transform())
  - But delegate to individual objects

Live implementation

of

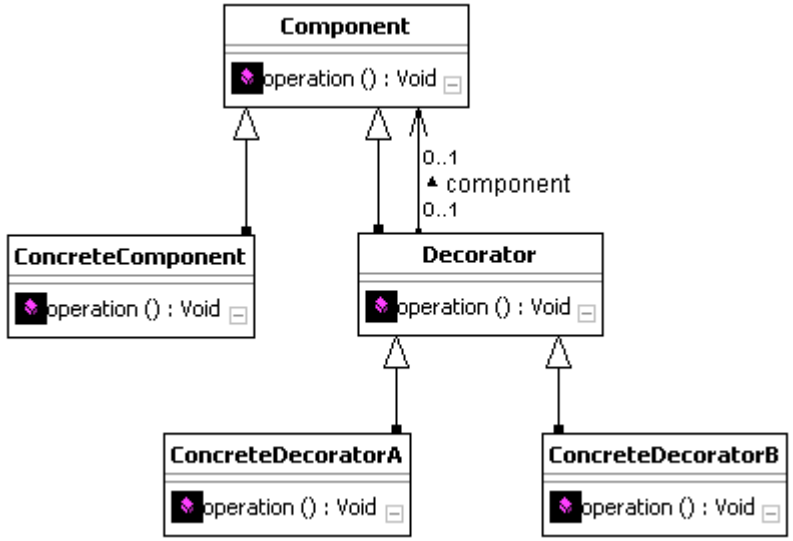
ImageConverter

using the

Decorator Pattern

Decorator Pattern : Specification

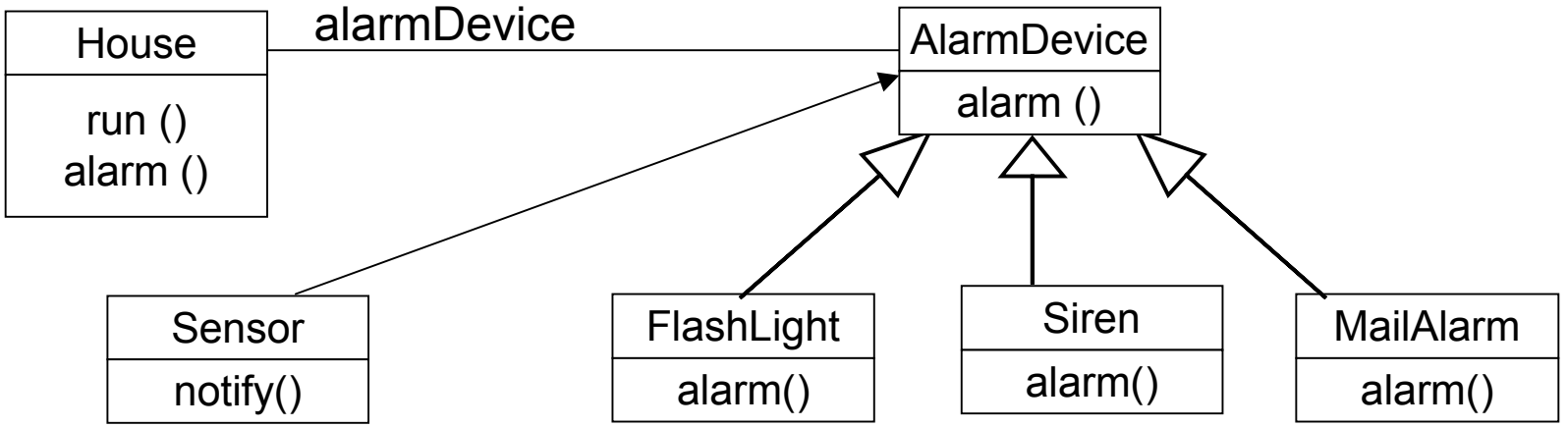
- Delegate to objects
- Based to Strategy (delegation chain of length 1)



- **Consequences:**
  - Runtime flexibility
  - Lots of Objects
- **Known Uses:** java.io.[Input|Output]Stream subclasses

Observer Pattern

- Basic Pattern behind MVC
- To-many relationship between subject and observers
- aka Publish-Subscribe-Protocol
- Example:
  - One Sensor in the House notifies all AlarmDevices



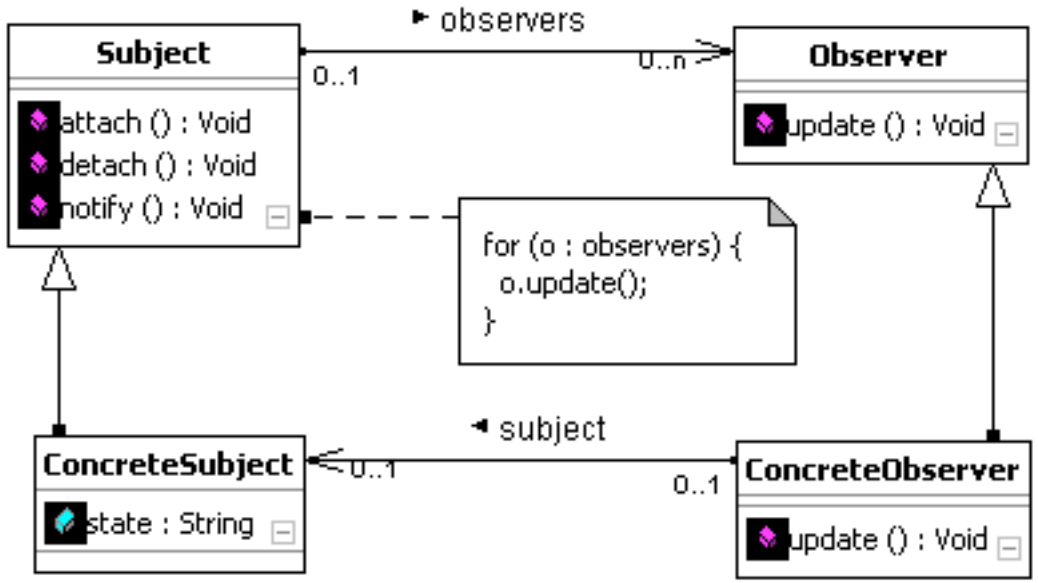
Live implementation

of the Observer Pattern

with HouseAlarm

Observer Pattern : Specification

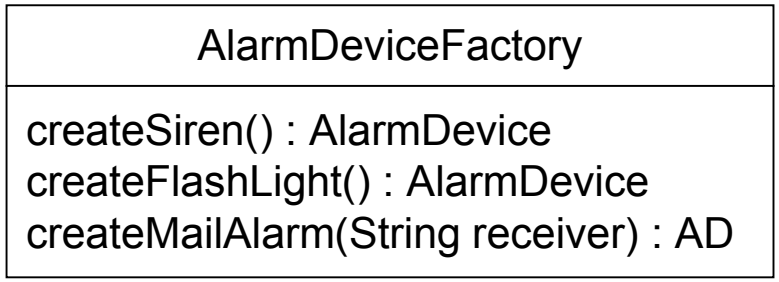
- Observers register at the Subject
- Subjects changes are delivered to all Observers



- Updates: Push- vs. Pull
- **Known Uses:** GUI/Widget Libraries, e.g. various Listeners in Swing

Factory Method Pattern

- Motivation
  - Have a central place where concrete classes are instanciated
- We should use that in the HouseAlarmExample...
  - Currently used is main() or init()



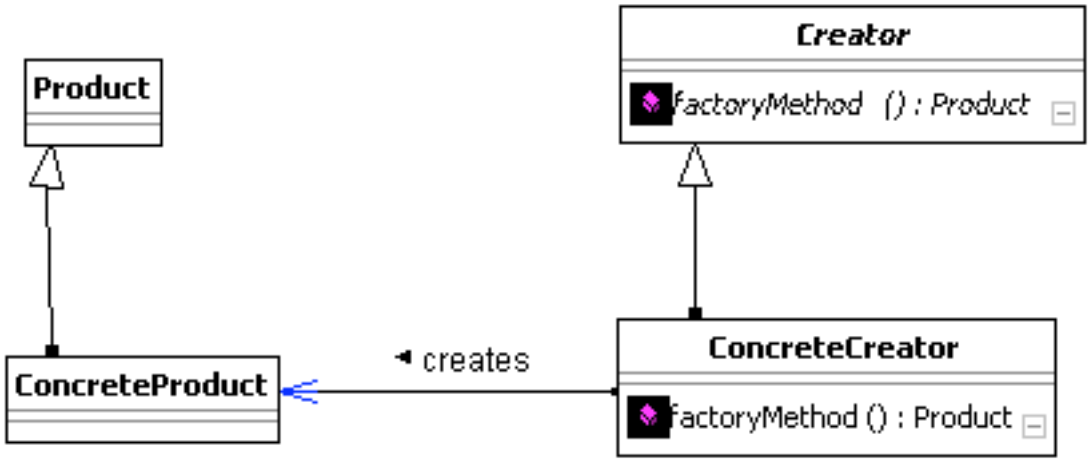
Live implementation

of the FactoryMethod Pattern

with HouseAlarm

Factory Method Pattern : Specification

- One abstract method per concrete class in a (singleton) Factory
- Subclasses create the desired corresponding type



Abstract Factory Pattern

- Extends Factory Method:
  - Create Methods per Type as well
- But the Factory itself is subclassed
  - Which might come from a Factory method... :)
- Example: Different concrete class hierarchies for adapting to platform-specific window / user interface toolkits:
  - MFCCompFactory vs. GTKCompFactory extend WidgetFactory
  - Common: Button, Window, Scrollbar...
- Benefits and drawbacks
  - Consistent, concrete classes isolated
  - Easy to change...
  - Extensions of the Class Hierarchy is difficult

Prototype Pattern

- Lets you create new Objects by copying existing ones
- Clone() Operation required (which is an integral part of Java)
- Alternative to Factory (avoids Factory classes/methods)

- Example / Known uses:
  - Clipart Gallery etc.
  - Matlab/Simulink Block Library

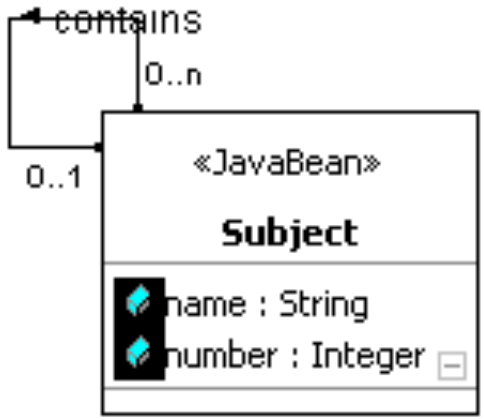
Timetable 7th of May

- Discussion of Exercise 3
- Design Patterns
  - Template Method
  - Observer / Listener
- Break 1
- Design Patterns
  - Factory Method
  - Abstract Factory
  - Prototype
- Break 2
- **Fujaba4Eclipse: PropertyChange-Mechanism**
- Exercise 4

PropertyChange Mechanism

- General JDK mechanism (some default implementation is missing), basically the Observer Pattern
- Slighty different Terminology: Listener, Source, Property
- adopted in the Fujaba Code Generator
  - Uses Constants in the generated Code
  - Robust
- Requires <<JavaBean>> stereotype in each class which should fire events
- All setters fire accordingly...
  - oldValue, newValue

Lets have a look...

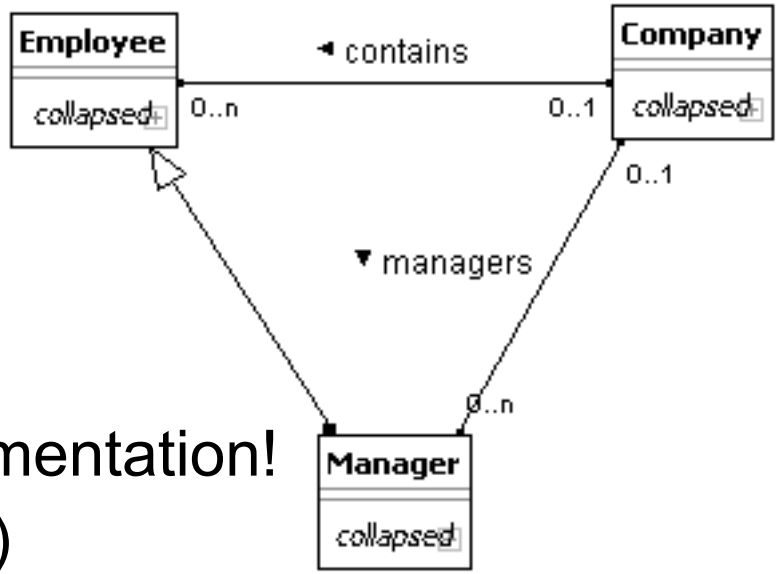


Timetable 7th of May

- Discussion of Exercise 3
- Design Patterns
  - Template Method
  - Observer / Listener
- Break 1
- Design Patterns
  - Factory Method
  - Abstract Factory
  - Prototype
- Break 2
- Fujaba4Eclipse: PropertyChange-Mechanism
- **Exercise 4**

Bonus Homework 3 (1pt)

- Write some code (plain Java) that implements PropertyChangeListener and is responsible for keeping the two associations here in sync:
  - When adding objects to either assoc
  - Also remove!



- Write a JUnit-Test for you implementation!
  - In Fujaba4Eclipse preferred :)

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!