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

What we have learned yesterday

- What are Design Patterns
- Fujaba4Eclipse
  - How to draw Diagrams
  - Generate Code from the Model
- Design Patterns Part One: Singleton
- Interactive Debugging with eDOBS
- Exercise 1 : Implement Singleton and Delegation

Timetable 5th of May

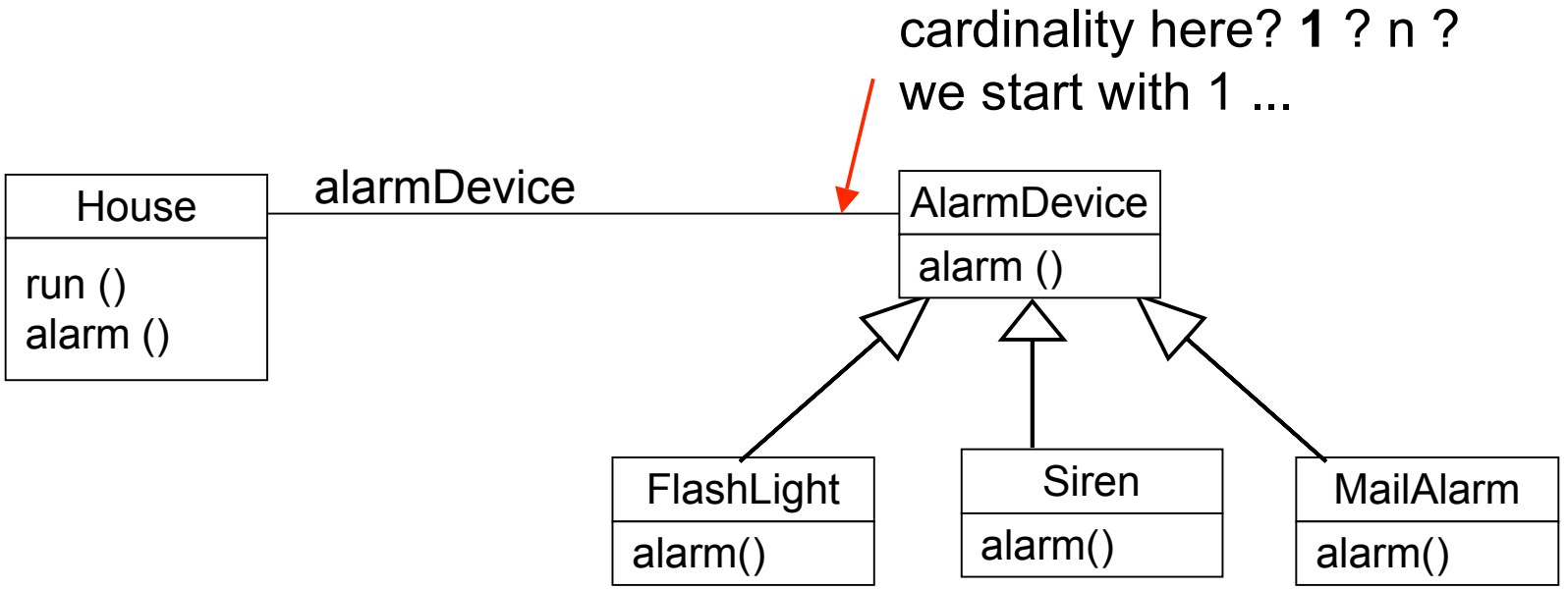
- **Solution of Exercise 1**
- Design Pattern #2: Strategy Pattern
- Fujaba4Eclipse: Modelling Methods
  - Simple Control Flow, Statement Activities
  - Creating and Deleting Objects
- JUnit - short recapitulation
- Fujaba4Eclipse: Writing JUnit-Tests
- *Break 1*
- More Design Patters:
  - DP#3: Composite Pattern (with Fujaba4Eclipse)
  - DP#4: Visitor Pattern (in Java)
- *Break 2*
- Exercise 2: Implement Composite and Visitor in Java or Fujaba4Eclipse

Exercise 1

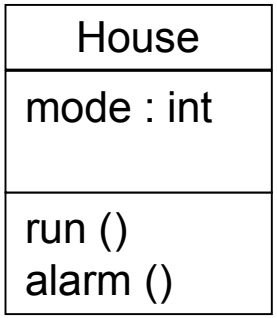
- Implement the Alarm example from slide 14 in Java:
  - Use the singleton pattern for the class House
- Write a method, which instanciates all types of alarms, adds them individually to the house and calls *house.alarm()* each time
- Model the Alarm example in Fujaba4Eclipse (hint: use a different package name)
- Change your *main()*-Method to use the generated code
- In the Review with the Tutor: Use the eDOBS to visualize your object structure!

Live modeling

of Exercise 1



Back to the simple solution: case distinction

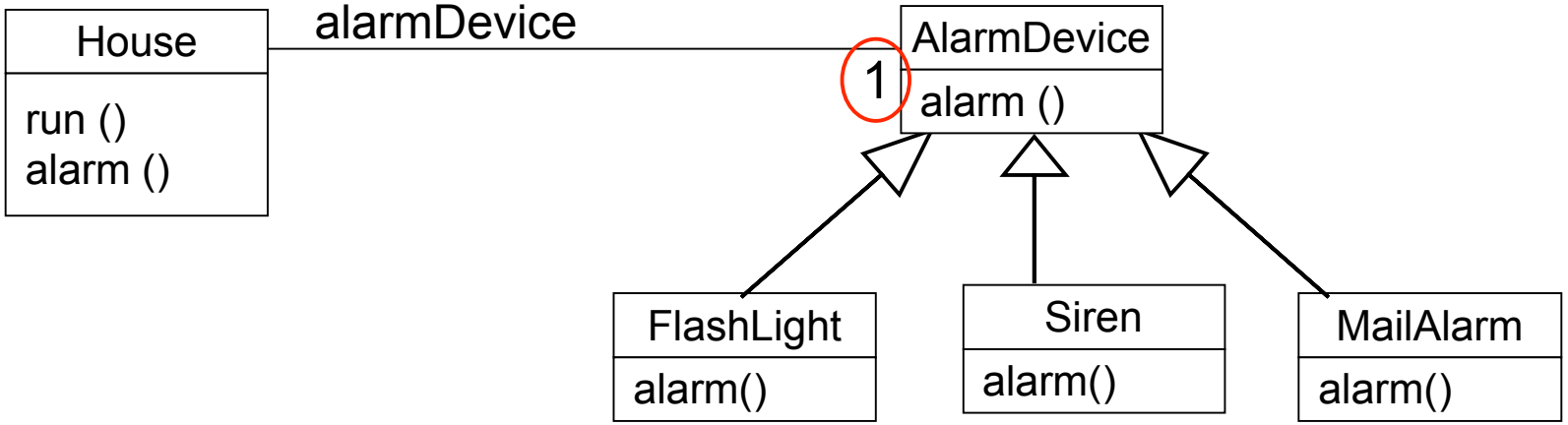



```
public void alarm ()
{
  if (mode == SIRENE)
  { …}
  else if (mode == FLASH)
  { …}
  else if (mode == MAIL)
  { …}
  …
```

Bad! Don‘t do that at home!

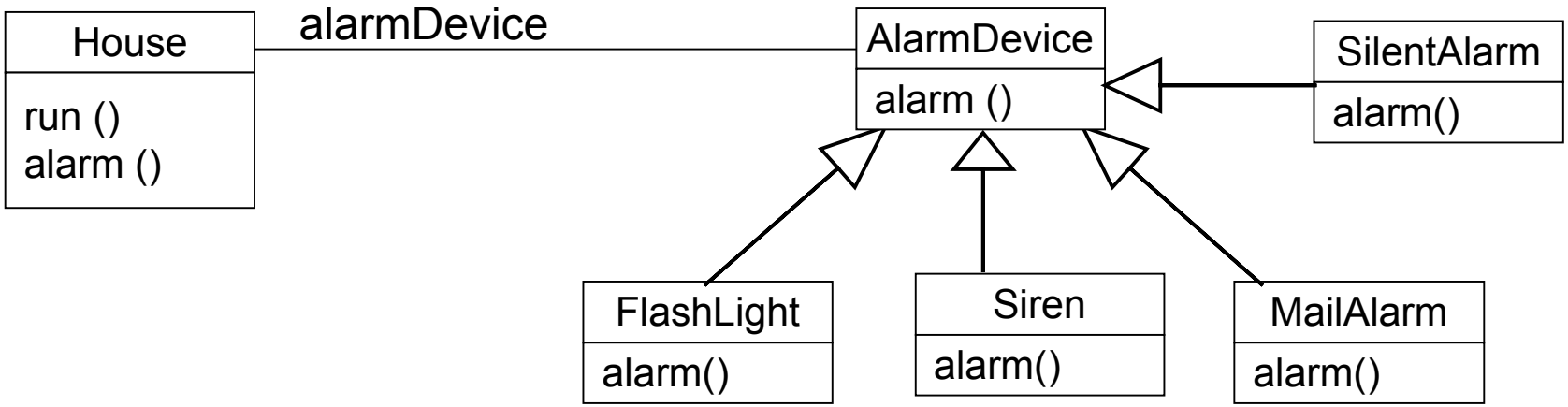
Why?
- try to use the alarm in your car
- Add a method like shutdown()

DP #2: Strategy Pattern



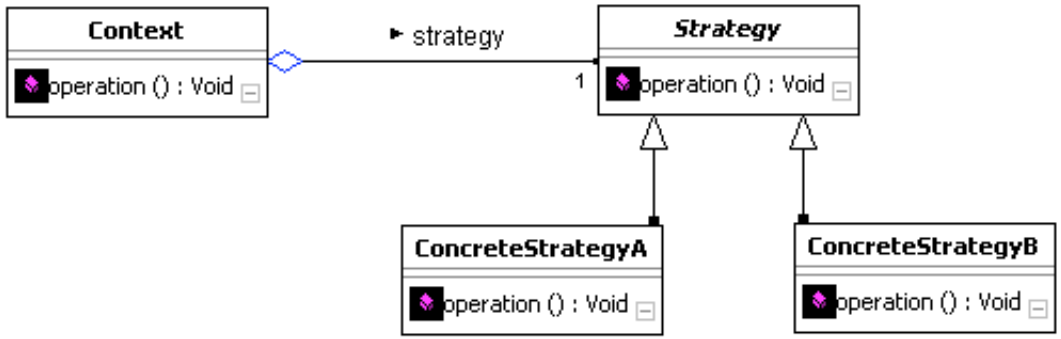
Live modeling:

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/elseif/else...-chains
  - Strategy can be changed at runtime
  - Adding a new ConcreteStrategy to the design is easy
- Known Usages:
  - Comparator for sorting sorting Lists etc.
  - Swing Layout manager (Flow-, Box-, Grid-Layout)

Timetable 5th of May

- Solution of Exercise 1
- Design Pattern #2: Strategy Pattern
- **Fujaba4Eclipse: Modelling Methods**
  - Simple Control Flow, Statement Activities
  - Creating and Deleting Objects
- JUnit - short recapitulation
- Fujaba4Eclipse: Writing JUnit-Tests
- *Break 1*
- More Design Patters:
  - DP#3: Composite Pattern with Fujaba4Eclipse
  - DP#4: Visitor Pattern in Java
- *Break 2*
- Exercise 2: Implement Composite and Visitor in Java or Fujaba4Eclipse

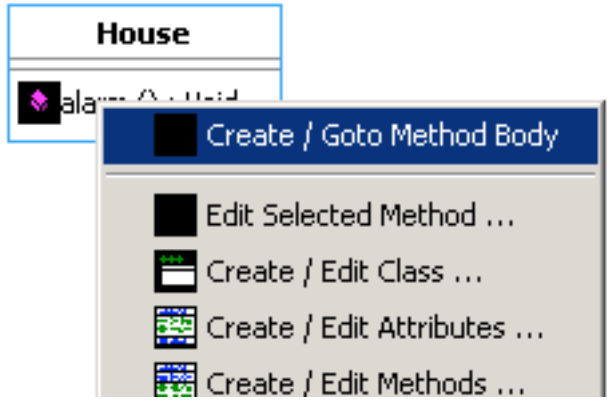
Modeling methods

in Fujaba4Eclipse

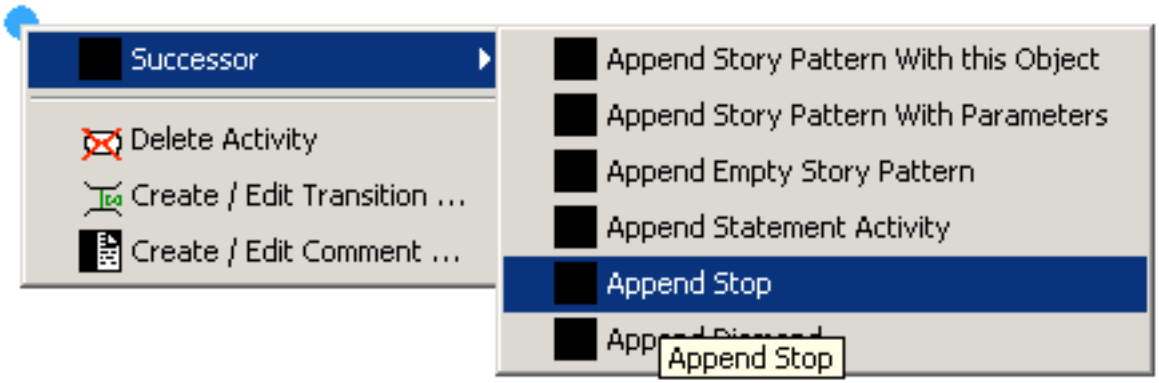
Demo

(with HouseAlarm)

Modeling Methods with Fujaba4Eclipse



StartupApplication::main (args: StringArray): Void



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

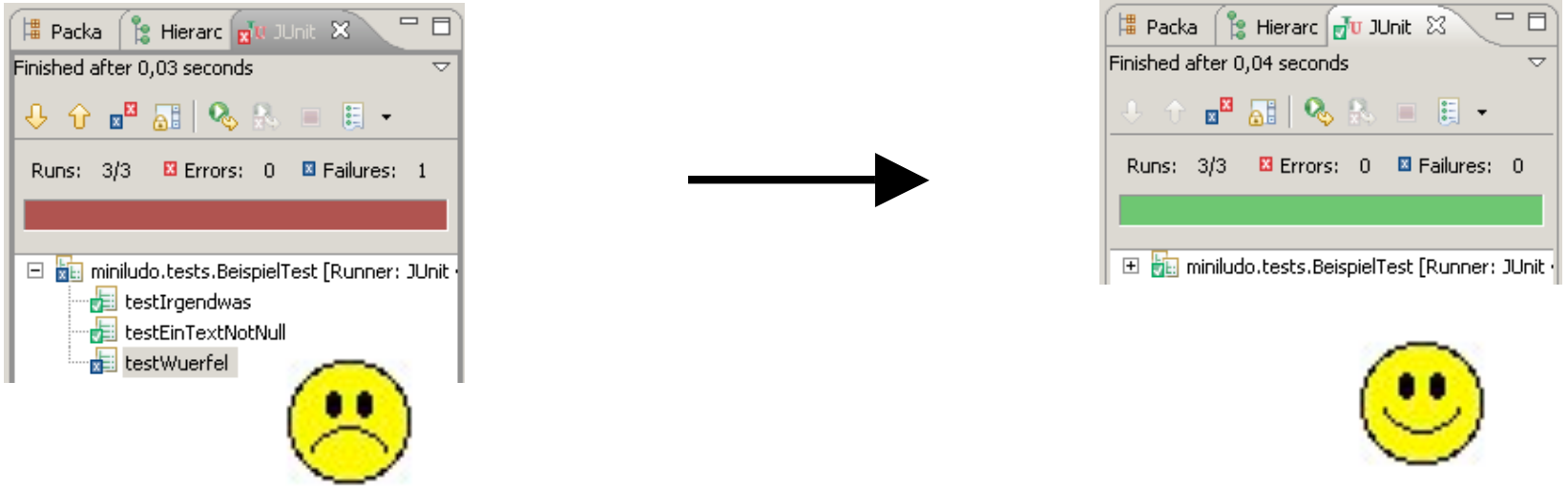
- You can verify whether the method does what you expect by looking at the generated source code!

Timetable 5th of May

- Solution of Exercise 1
- Design Pattern #2: Strategy Pattern
- Fujaba4Eclipse: Modelling Methods
  - Simple Control Flow, Statement Activities
  - Creating and Deleting Objects
- **JUnit - short recapitulation**
- Fujaba4Eclipse: Writing JUnit-Tests
- *Break 1*
- More Design Patters:
  - DP#3: Composite Pattern with Fujaba4Eclipse
  - DP#4: Visitor Pattern in Java
- *Break 2*
- Exercise 2: Implement Composite and Visitor in Java or Fujaba4Eclipse

JUnit Framework (1)

- Free Open Source Framework
  - http://www.junit.org/
  - We use: 3.8.1, current version: 4.5 (utilizes annotations)
- Authors: Kent Beck & Erich Gamma
- Basic principle:
  - No Feature is realized without a test!



Why JUnit - Tests (here)

- We‘re testing, whether our example does what it should
  - Programatically
  - Reproduceable
  - Self-evaluating
- Fix run-and-look-at-results programatically
- Write test first!
  - Define external interface of your classes first
  - Define a example situation
    - Helps with debugging
  - Implement the classes-to-test until the test succeeds!

A TestCase

- A separate Class, but in the same Package
- inherits *junit.framework*|*org.junit.TestCase*, which provides *assert*-Methods
- Contains several Test Methods (tests): *public void testXYZ() throws Exception*
- Should test the basic usecase with an example situation
  - Build example situation (or use *setUp()* for that)
    - Maybe remember it‘s state
  - Call Component to test
    - One or more Method calls
  - Evaluate the situation afterwards
    - Return value or modifications of the example situation
    - Handle errors / exceptions, or pass them along
- Should test situations that lead to an error
  - Of course, this needs to be handled. Don‘t forget *fail()*
- Should not test trivia

Assertion-Methods

- *assertEquals(expected, actual)*
  - Object, primitive Types as boolean, int, long, ...
- *assertTrue(boolean)*
- *assert[Not]Null(Object), assert[Not]Same(obj1, obj2)*
- *fail()* (in conbination with Control Flow)
- On failed assertion or *fail()*:
  - throw *junit.framework.AssertionFailedError*
- All Assertion-Methods also with ‚*String message*‘ Parameter

Timetable 5th of May

- Solution of Exercise 1
- Design Pattern #2: Strategy Pattern
- Fujaba4Eclipse: Modelling Methods
  - Simple Control Flow, Statement Activities
  - Creating and Deleting Objects
- JUnit - short recapitulation
- **Fujaba4Eclipse: Writing JUnit-Tests**
- *Break 1*
- More Design Patters:
  - DP#3: Composite Pattern with Fujaba4Eclipse
  - DP#4: Visitor Pattern in Java
- *Break 2*
- Exercise 2: Implement Composite and Visitor in Java or Fujaba4Eclipse

Live implementing

and modeling

a JUnit test

(Company-Example)

Timetable 5th of May

- Solution of Exercise 1
- Design Pattern #2: Strategy Pattern
- Fujaba4Eclipse: Modelling Methods
  - Simple Control Flow, Statement Activities
  - Creating and Deleting Objects
- JUnit - short recapitulation
- Fujaba4Eclipse: Writing JUnit-Tests
- *Break 1*
- **More Design Patters:**
  - DP#3: Composite Pattern with Fujaba4Eclipse
  - DP#4: Visitor Pattern in Java
- *Break 2*
- Exercise 2: Implement Composite and Visitor in Java or Fujaba4Eclipse

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!

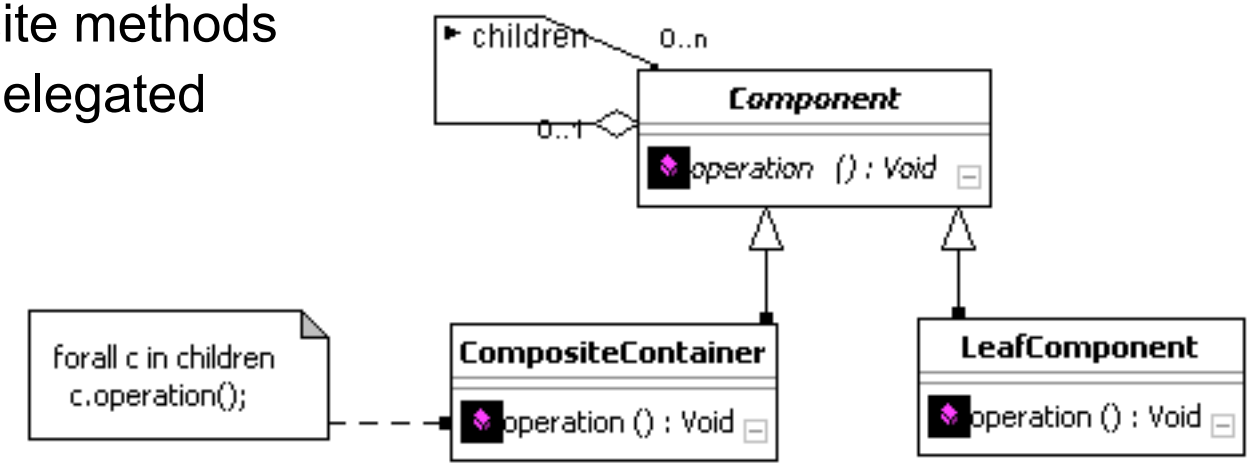
Implementing

Composite Pattern

Live (for FileSystem)

DP #3: Composite Pattern Specification

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



- Implementation:
  - Where are childs stored?
  - Parent reference?
  - add()/remove() implementation for leaves?
- Known Uses:
  - Widget Library Class Hierarchies (e.g. Swing)

Bonus homework: 3rd find() - print a textual Tree

- List files like this (try it at home with or without visitor):

```
C:/
 |- windows/
 |    |- system32/
 |    |    *- hardwaredriver.sys
 |    *- notepad.exe
 *- temp/
      *- editorcontent.tmp
```

- (Will give 1 Point for easily understandable, elegant solution :))

Visitor motivation

- Now we want to implement some operations:
  - Count all files recursively
  - Summarize the file size of all files ...
  - Find empty directories ...
  - Difficult with the current approach!!!

- Decouple tree traveral from operation (e.g. System.out.print)
- Allow different operations, group then in classes
- Operations might be interested only in a certain type of the traveral elements

Implementing

Visitor Pattern

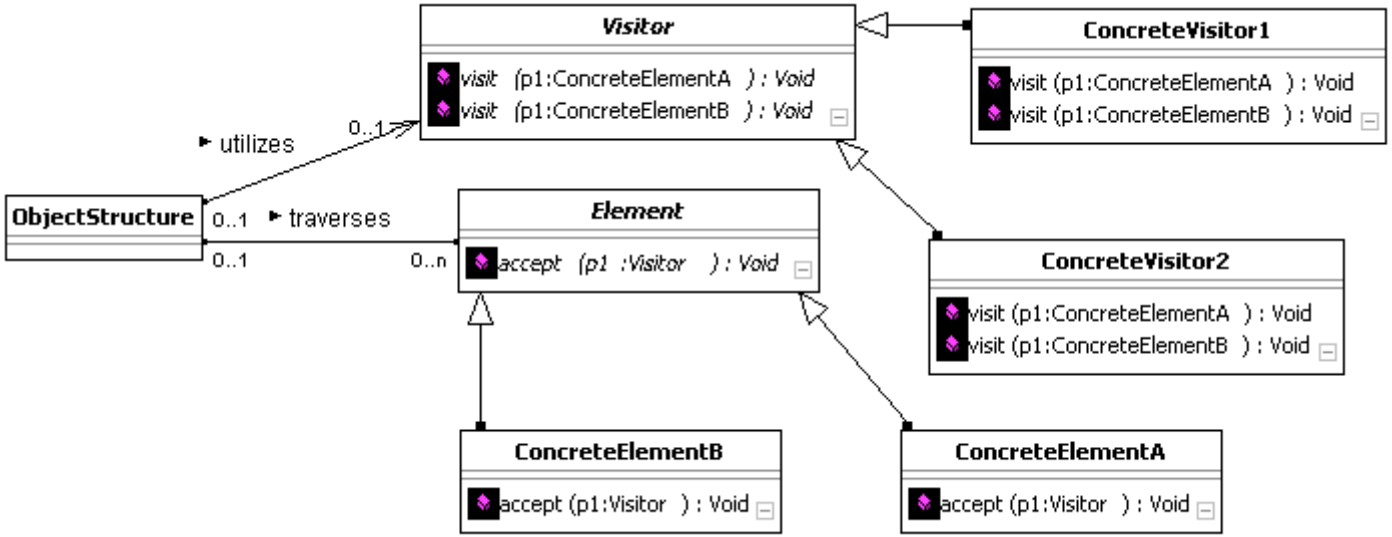
Live

(CountFilesVisitor and

FindEmptyDirectoriesVisitor)

DP #4: Visitor Pattern - Specification

- Pass a Visitor to all Elements while iterating/traversal
- Elements call corresponding visit()-Method



- **Consequences**: Benefits and Drawbacks
  - Separation of concern
  - Adding new ConcreteElement is hard, operations not
  - Encapsulation? **Known Usages**: Parser, e.g. javacc

Timetable 5th of May

- Solution of Exercise 1
- Design Pattern #2: Strategy Pattern
- Fujaba4Eclipse: Modelling Methods
  - Simple Control Flow, Statement Activities
  - Creating and Deleting Objects
- JUnit - short recapitulation
- Fujaba4Eclipse: Writing JUnit-Tests
- *Break 1*
- More Design Patters:
  - DP#3: Composite Pattern with Fujaba4Eclipse
  - DP#4: Visitor Pattern in Java
- *Break 2*
- **Exercise 2: Implement Composite and Visitor in Java or Fujaba4Eclipse**

Exercise 2:

- Implement *Abstract-, RegularFile* and *Directory* using the Composite Pattern as shown before (either in plain java or as Fujaba4Eclipse model, assume unix-style filesystem, omit Root/Drive)
- Delegate *RegularFile(String name)*, child computation, *getName()* and *getFileSize()* to *java.util.File* in the class RegularFile
- Implement the CountFilesVisitor as shown before
- Count the files in the current directory by calling *traverse(new RegularFile("."), visitor);* using the CountFilesVisitor. How many are there?
- Write a SummarizeFileSizesVisitor !
- Advanced:
  - Modify your program so it can traverse into JAR-Files (which are essentially ZIP-Files). How many files to you get now with the CountFilesVisitor?
  - Add a file extension filter to the CountFilesVisitor. How many *.java-Files are there?