Rheinische Friedrich-Wilhelms-Universität Bonn, Institut für Informatik III

Vorlesung Aspektorientierte Softwareentwicklung (AOSD)
– Sommersemester 2008 –

Dr. Günter Kniesel, Daniel Speicher

Übungsblatt 3

Abgabe: Dienstag 17.06.2008 23:59:59 per SVN
(Tutorien: Freitag 20.06. + Montag 23.06.)

Hinweis: Der Code auf den sich die einzelnen Aufgaben beziehen ist im SVN zu finden unter https://svn.iai.uni-bonn.de/repos/IAI_Software/se/aosd2008ss/aufgaben/trunk/B03A...

Aufgabe 1 (Observer-Pattern, ***AspectJ***) (6 Punkte)

Das im SVN vorliegende Projekt B03A1 enthält die objektorientierte Implementierung des Model-View-Controller Patterns anhand des bekannten TicTacToe-Beispiels. Dabei teilt das Model seinen Views Änderungen durch den Aufruf von `notifyObservers()` mit.

Die objektorientierte Implementierung soll nun durch die aspektorientierte Variante von Jan Hannemannn und Grergor Kiczales ersetzt werden, die in der Vorlesung vorgestellt wurde:

a) Analysieren Sie das Beispielprogramm und identifizieren Sie die verwendeten Instanzen des Observer-Patterns, ihre Teilnehmer und die Rollen die sie im Pattern spielen.
b) Kopieren Sie das Beispielprojekt (nennen Sie es B03A1-refactored) und implementieren Sie darin Aspekte die die entdeckten Instanzen des Patterns realisieren. Gehen Sie dabei wie folgt vor:
   - Erzeugen Sie den abstrakten Observer-Aspekt (wie in der Vorlesung vorgestellt).
   - Erzeugen Sie zu jeder Pattern-Instanz einen konkreten Aspekt mit konkreten Pointcuts, die die Stellen beschreiben, an denen der Aspekt greifen soll.
   - Löschen Sie aus den Java-Klassen die Teile die durch den Aspekt realisiert werden sollen und testen Sie ob ihr Aspekt wie beabsichtigt wirkt.

Aufgabe 2 (Adapter Pattern, ***AspectJ***) (4 Punkte)

Design Patterns liefern uns Rezepte zur Implementierung bestimmter Objektstrukturen. Oftmals werden sie gebraucht, um Probleme zu lösen, die durch die Grenzen des jeweils verwendeten Programmierparadigmas entstehen. Daher bietet es sich an, dass einige objektorientierten Design Patterns mit Hilfe von AspectJ anders umgesetzt werden als in einer rein objektorientierten Sprache. In dieser Aufgabe beschäftigen wir uns dementsprechend mit den Unterschieden zwischen den GoF-Varianten von OO-Pattern und ihrer AspectJ-Implementierung. Vergleichen Sie daher Ihre Lösung mit der „offiziellen“ GoF-Variante und scheuen Sie sich nicht, bewusst davon abzuweichen.

Das Projekt B03A2 im SVN enthält im Paket *a* zwei Implementierungen von Sensoren (für Temperatur und Strahlung), die weder in einer Vererbungsbeziehung stehen noch einen gemeinsamen Obertyp haben (außer „Object“).

Nun soll für beliebige Sensoren eine Statusabfrage (Methode *getStatus*()) implementiert werden, die die Werte des jeweiligen Sensors ermittelt und in drei Gefahrenkategorien einordnet:

- ok
- critical
- danger

Für die Temperatursensoren ist ein Wert von unter 150 °C ok, ein Wert darüber ist kritisch und ab 200 °C beginnt der Gefahrenbereich. Bei Strahlungssensoren ist der Grenzwert ab dem die Strahlung kritisch wird 0,7 und Gefahr besteht ab einer Strahlung von 1,3.

Nutzen Sie die Fähigkeiten von AspectJ für eine möglichst einfache Umsetzung der obigen Aufgabe aus.
Natürlich sollen die bestehenden Klassen dabei nicht geändert werden!

Aufgabe 3 (Abstract Factory Pattern, ***LogicAJ***) (12 Punkte)

Hanneman und Kiczales haben das Abstract Factory Pattern als mit AspectJ nicht realisierbar eingestuft. In dieser Aufgabe geht es darum es mit Hilfe von LogicAJ umzusetzen.

Im SVN finden Sie im Package abstractfactory eine Anwendung (Klasse Main.java) die auf einer Objektstruktur mittels einer abstrakten Fabrik arbeitet. Die abstrakte Fabrik selbst, sowie die getFactory()-Methode, mittels der man sich eine bestimmte konkrete Fabrik holen kann, sind schon in dem Basiscode enthalten, so dass die Anwendung die entsprechenden Aufrufe benutzen kann. Jede abstrakte oder konkrete Produktfamilie ist in einem eigenen Package implementiert. Die Struktur der Anwendung entspricht dem Schema das in den dem Aufgabenblatt beigefügten Folien (s. Website) erläutert wird.

Ihre Aufgabe ist es

- die Generierung der konkreten Fabrikklassen in einem abstrakten LogicAJ-Aspekt, zu implementieren
- in einem konkreten Unteraspekt die konkreten Pointcuts zu definieren die die Anwendung lauffähig machen

Tipps zum abstrakten Aspekt:

- Definieren Sie einen *abstrakten* Pointcut
  `abstractFactory( ?abstractFactory, ?concreteFamilyPackage )`
  der angibt in welchen Java-Paketen konkrete Produktfamilien für eine bestimmte abstrakte Fabrik enthalten sind.

- Schreiben Sie einen konkreten Pointcut
  `concreteFactory( ?concreteFamilyPkg, ?concreteFactory )`
  der zu einem Package, das eine konkrete Produktfamilie enthält, den vollständig qualifizierten Namen der zugehörigen konkreten Fabrik bestimmt. Denken Sie sich ein passendes Namensschema aus.

- Schreiben Sie eine generische Introduction, die leere konkrete Fabriken erzeugt, die jeweils von der passenden abstrakten Fabrik erben.

- Schreiben Sie einen Pointcut
  ```
  factoryMethodImplem(?abstractFactory, ?concreteFactory,
  ?abstractProduct, ?concreteProduct, ?methodName, ??params)
  ```
  der zu einer konkreten Fabrik deren abstrakte Fabrik bestimmt und zu einer Methode der Abstrakten Fabrik deren komplette Signatur (Ergebnistyp, Name, Parameter) sowie das konkrete Produkt das davon in der konkreten Fabrik erzeugt werden soll.

- Schreiben Sie eine generische Introduction, die jede konkrete Fabrik mit den dazugehörigen konkreten Fabrikmethoden füllt.

- Schreiben Sie einen Advice der sicherstellt, dass beim Aufruf der getFactory()-Methode die passende konkrete Fabrik erzeugt wird.

Erinnern Sie sich auch, dass predicate Pointcuts mehrdirektional funktionieren: je nach dem welche Parameter beim Aufruf schon gebunden sind, werden Bindungen für die verbleibenden freien Parameter berechnet. Z.B. kann typepackage(**?T**,**?P**) genutzt werden um zum gegebenen Typ **?T** sein Package **?P** zu bestimmen oder um zum gegebenen Package **?P** der Reihe nach alle darin enthaltenen Typen an **?T** zu binden.