More Quick Patterns

See http://101companies.org/wiki/Contribution:javaExorcism for examples
Object Adapter -- Problem

A given object often has about the right capabilities for a specific client but not necessarily the precise interface expected. Accordingly, an interface adaptation can be achieved by an extra wrapper around the given object.
Adapter -- Solution (object adapters)
Objekt-Adapter

SimpleList<X>

- add(X):boolean
- remove(X):boolean

SimpleLinkedList<X>

*inner : java.util.LinkedList<X>

+ iterator:Iterator<X>

java.util.LinkedList<x>

-Mitgliedsname

viel viele Methoden

Adapter

Target

Service

org.softlang.company.features.util
Adapter -- Solution (class adapters)
Template Method -- Problem

Some algorithms or strategies in a system may be very similar and hence it may be desirable to capture their commonalities in a sort of template so that specifics can be expressed through refinement giving rise to a high degree of reuse.
Strategie: Reduktion/Faltung

Fold (higher-order function)

From Wikipedia, the free encyclopedia
(Redirected from Reduce (higher-order function))

In functional programming, fold – also known variously as reduce, accumulate, aggregate, compress, or inject – refers to a family of higher-order functions that analyze a recursive data structure and recombine through use of a given combining operation the results of recursively processing its constituent parts, building up a return value. Typically, a fold is presented with a combining function, a top node of a data structure, and possibly some default values to be used under certain conditions. The fold then proceeds to combine elements of the data structure's hierarchy, using the function in a systematic way.
Reduce in 101Companies

• Die Strategie zum Aufsummieren der Gehälter:
  • Besuche alle Knoten rekursiv (visit)
  • Extrahiere das Gehalt (zero)
  • Verknüpfe die extrahierten Werte durch Addition (append)

• Strategiemethode reduce in abstrakter Klasse Reducer mit abstrakten Methoden visit, zero, append
Template Method -- Solution

Musterstrategie verwendet primitiveMethode?

AbstractClass

ConcreteClassA

ConcreteClassB

ConcreteClassC


(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Total extends Reduce

Reduce

- reduce
- visit
- zero
- append

Total

- visit
- zero
- append

AbstractClass

ConcreteClass

Code dazu: javaTemplate
Strategie Walker implementiert
Map (Higher order funtion)

Map (higher-order function)
From Wikipedia, the free encyclopedia

This article relies largely or entirely upon a single source. Relevant discussion may be found on the talk page. Please help improve this article by introducing citations to additional sources. (November 2012)

In many programming languages, map is the name of a higher-order function that applies a given function to each element of a list, returning a list of results. It is often called apply-to-all when considered in functional form. This is an example of functoriality.

For example, if we define a function square as follows:

\[
\text{square } x = x \times x
\]

Then calling map square \([1,2,3,4,5]\) will return \([1,4,9,16,25]\), as map will go through the list and apply the function square to each element.
Two general class of algorithms can be identified:

- Queries such as 101feature:Total
- Transformations such as 101feature:Cut
Abstract Factory -- Problem

Object models often give rise to variation. For instance, there may be multiple GUI libraries subject to different widget hierarchies. Whenever components want to abstract from the specific choice, then substantial efforts are required. For instance, the construction of objects must be tunneled through a factory.
Abstract Factory -- Solution

http://sourceMaking.com/design_patterns/abstract_factory

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
/**
 * The factory interface for constructing all possible company objects
 */

public interface Factory {
    Company mkCompany();
    Department mkDepartment();
    Employee mkEmployee();
}

/**
 * A factory that constructs (a kind of) "beans" as opposed to POJPs
 */

public class BeanFactory implements Factory {
    public Company mkCompany() { return new CompanyImpl(); }
    public Department mkDepartment() { return new DepartmentImpl(); }
    public Employee mkEmployee() { return new EmployeeImpl(); }
}

/**
 * A factory that constructs POJPs
 */

public class PojoFactory implements Factory {
    public Company mkCompany() { return new CompanyImpl(); }
    public Department mkDepartment() { return new DepartmentImpl(); }
    public Employee mkEmployee() { return new EmployeeImpl(); }
}
Abstract Factory

DEMO

- Two alternative object models for companies.
  - POJOs
  - Some sort of beans
- Make the choice of the object model configurable.
Singleton

• Das Singleton-Pattern stellt sicher, dass es nur ein einziges Objekt dieser Klasse gibt (Einzelstück)
• Druckerspooler, Logging, etc.

Zeiger auf die einzige Instanz
Privater(!) Konstruktor
getInstance():
  if (instance==null) instance=new Singleton();
  return instance;
** Singleton **

```java
/**
 * An associative operation "append" with its unit "unit".
 */
public interface Monoid<R> {
    R unit();
    R append(R x, R y);
}
```

```java
/**
 * A monoid for addition
 */
public final class AddDoubles implements Monoid<Double> {
    private static AddDoubles instance = null;
    private AddDoubles() {}
    public static AddDoubles getInstance() {
        if (instance == null)
            instance = new AddDoubles();
        return instance;
    }
    public Double unit() { return 0.0; }
    public Double append(Double x, Double y) { return x+y; }
}