University of Zurich
Institut für Informatik

s.e.a.l. software evolution & architecture lab
RERG

Design Patterns

Summary

1 Creational patterns

Abstract factory Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Factory method Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Builder Separate the construction of a complex object from its representation so that the same construction process can create different representations.

Prototype Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Singleton Ensure a class only has one instance, and provide a global point of access to it.

2 Structural patterns

Adapter Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Bridge Decouple an abstraction from its implementation so that the two can vary independently.

Composite Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Decorator Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality.

Facade Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

Flyweight Use sharing to support large numbers of fine-grained objects efficiently.

Proxy Provide a surrogate or placeholder for another object to control access to it.

3 Behavioral patterns

Chain of responsibility Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Command Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Interpreter Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Iterator Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Mediator Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

Memento Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

Observer Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

State Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

Strategy Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Template method Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Visitor Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

Literatur

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.