CSCI 253

Object Oriented Design:
Façade Pattern
George Blankenship

Overview

Creational Patterns
- Singleton
- Abstract factory
- Factory Method
- Prototype
- Builder

Structural Patterns
- Composite
- Façade
- Proxy
- Flyweight
- Adapter
- Bridge
- Decorator

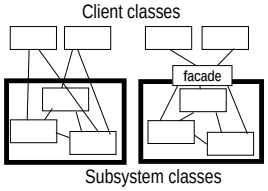
Behavioral Patterns
- Chain of Respons.
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

The Elements of a Design Pattern

- A pattern name
- The problem that the pattern solves
  - Including conditions for the pattern to be applicable
- The solution to the problem brought by the pattern
  - The elements (classes-objects) involved, their roles, responsibilities, relationships and collaborations
  - Not a particular concrete design or implementation
- The consequences of applying the pattern
  - Time and space trade off
  - Language and implementation issues
  - Effects on flexibility, extensibility, portability

The Facade Pattern: The Problem

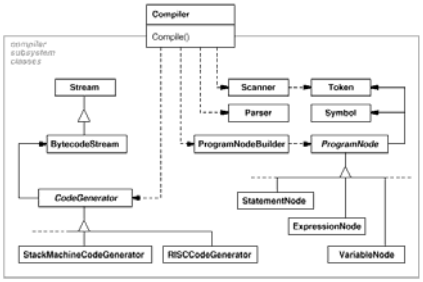
Provide a unified higher level interface to a set of interfaces in a subsystem to make the subsystem easier to use



- provide a simple interface to a complex subsystem
- decouple a subsystem from clients and other subsystems
- create layered subsystems by providing an interface to each subsystem level

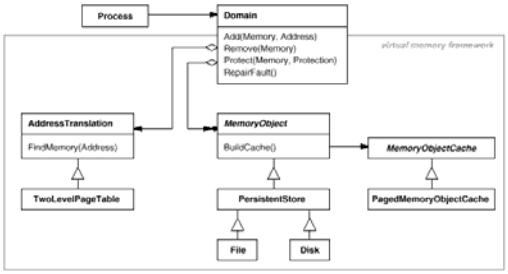
Facade Pattern George Blankenship 4

Compiler



Facade Pattern George Blankenship 5

Virtual Memory



Facade Pattern George Blankenship 6

ServerSocket

```
• public ServerSocket(int port, String portType, HomeTelehealth engine, MessageEngine messageEngine) {
• public void onClose(int transportId) {
• public void onError(int errorId, String errorText) {
• public void onMessage(int transportId, String message) {
        HL7Connection connection = HL7Connection.findConnectionByTransport(transportId);
                if(connection!=null) {
                        traceMessage(message,"server received from "+connection.getAddressA()+" on "+portType);
                        try {messageEngine.parseMessage(message,null,connection);
                        } catch (ChameleonException e) {
                                Trace.exception(CODE_FILE,e," (onMessage) exception while parsing");
                                messageEngine.messageError(207,"Message processing internal error");
                                engine.sendAcceptACK(connection);
                                return;
                        } if(connection.hasActiveFSM()) {
                                connection.getFSM().execute(Constants.FSM_MESSAGE_RECEIVED);
                        } else {engine.sendApplicationACK(connection);}
                } else {traceMessage(message,"server received for server transport ("+String.valueOf(transportId)+")");}
    }
• public Hl7ClientSocket onNewConnection(int transportId) {
• public void onTransportError(int transportId, int errorId,String errorText) {
```

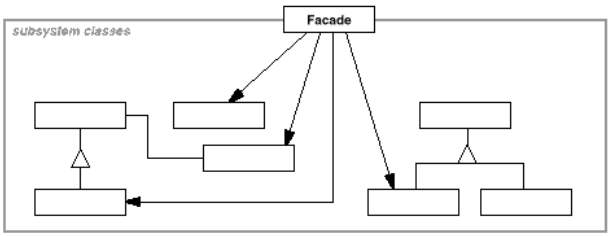
Facade Pattern George Blankenship 7

The Facade Pattern Participants & Collaboration

- Facade:
  - Knows which subsystem classes are responsible for a request
  - Delegates clients requests to appropriate subsystem objects
- Subsystems Classes
  - Implement the subsystem functionality
  - Handle work assigned by the Facade object
  - Have no knowledge of the Facade object; i.e. keep no reference to it
- Clients send their request to Facade which will then forward them to the appropriate subsystem object

Facade Pattern George Blankenship 8

The Façade Pattern: Structure



Facade Pattern George Blankenship 9

The Facade Pattern Consequences

- + Shields clients from subsystem components thereby reducing the number of objects that clients deal with thus making the subsystem easier to use
- + Promotes weak coupling between the subsystem and its clients allowing components of the subsystem to change without affecting the clients
- + Reduces compilation dependencies in large software systems
- + Does not prevent applications from using subsystem classes if they need to

Facade Pattern George Blankenship 10

The Facade Pattern Implementation

- The coupling between clients and subsystems can be reduced even further:
  - by making Facade an abstract class with concrete subclasses for different implementations of a subsystem; this abstract coupling keeps clients from knowing which implementation of a subsystem it is they use
  - by configuring a Facade object with different subsystem objects; to customise simply replace one or more of the subsystem objects
- Public versus Private subsystem classes: the façade is a part of the public interface of a subsystem together with the subsystem classes some clients have to access directly (very few OO languages support the notion of private subsystem classes although it would be a very useful feature)

Facade Pattern George Blankenship 11