CSCI 253

Object Oriented Design:

Bridge Pattern

George Blankenship

Overview

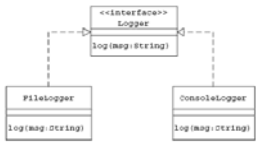
Creational Patterns	Structural Patterns	Behavioral Patterns
🖫 Singleton	🗐 Composite	🖫 Chain of Respons.
🗐 Abstract factory	🗐 Façade	🖫 Command
🗐 Factory Method	🗐 Proxy	🖫 Interpreter
🗐 Prototype	🗐 Flyweight	🗐 Iterator
🗐 Builder	🖫 Adapter	🖫 Mediator
	🖫 Bridge	🖫 Memento
	🖫 Decorator	🗐 Observer
		🗐 State
		🗐 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 Bridge Pattern: The Problem

Decouple interface from implementation so that the two can be changed independently



–Logging feature may be put into a separate class.
–The resulting class is an abstraction of the message logging functionality.
–A message can be logged to different types of destinations such as a file, console and others.

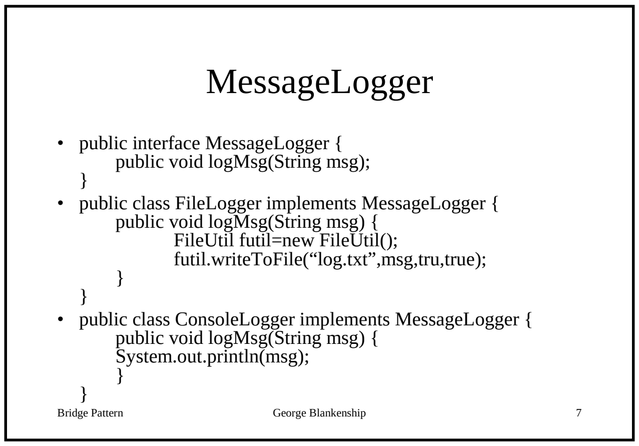
Also known as : Handle/Body

Design Problems

- After this design is implemented, let us suppose that an application object needs to log messages in a different format (e.g., in an encrypted form).
- The existing messaging logging functionality design is not sufficient without either:
  - Modifying different implementers
  - Extending the entire class hierarchy
- Having to modify the existing code in order to extend the functionality is not advisable and violates the basic object-oriented open-closed principle.
- Subclassing the class hierarchy for every different type of message format is also not recommended as it could result in an exponential number of subclasses and soon there will be an exploding class hierarchy.

Bridge Pattern Advantages

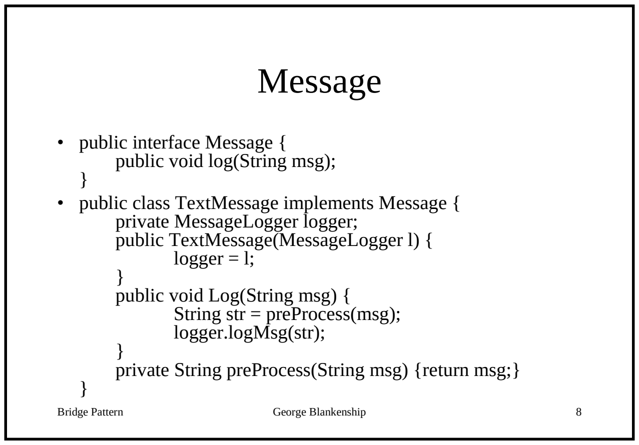
- The Bridge pattern can be used in this case to provide the ability to add new message formats and new types of implementations to the logger abstraction.
- The Bridge pattern separates the interface and implementations into two separate class hierarchies so that they both can be modified without affecting each other.



MessageLogger

```
• public interface MessageLogger {
        public void logMsg(String msg);
  }
• public class FileLogger implements MessageLogger {
        public void logMsg(String msg) {
                FileUtil futil=new FileUtil();
                futil.writeToFile(“log.txt”,msg,tru,true);
        }
  }
• public class ConsoleLogger implements MessageLogger {
        public void logMsg(String msg) {
        System.out.println(msg);
        }
  }
```

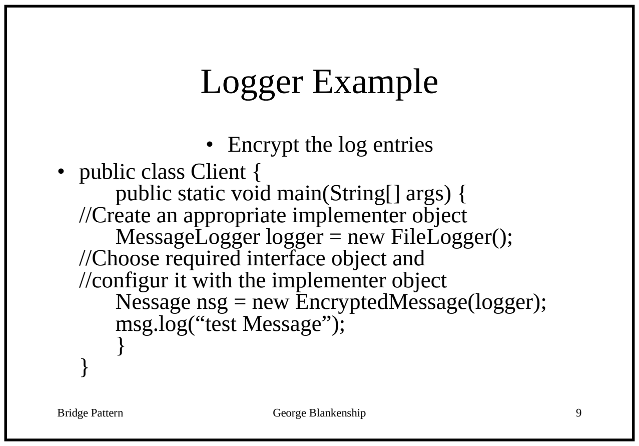
Bridge Pattern George Blankenship 7



Message

```
• public interface Message {
        public void log(String msg);
  }
• public class TextMessage implements Message {
        private MessageLogger logger;
        public TextMessage(MessageLogger l) {
                logger = l;
        }
        public void Log(String msg) {
                String str = preProcess(msg);
                logger.logMsg(str);
        }
        private String preProcess(String msg) {return msg;}
  }
```

Bridge Pattern George Blankenship 8



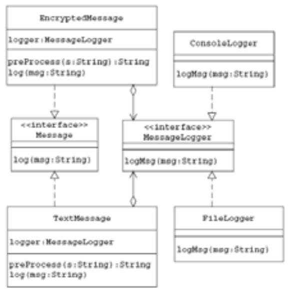
Logger Example

• Encrypt the log entries

```
• public class Client {
        public static void main(String[] args) {
  //Create an appropriate implementer object
        MessageLogger logger = new FileLogger();
  //Choose required interface object and
  //configur it with the implementer object
        Nessage nsg = new EncryptedMessage(logger);
        msg.log(“test Message”);
        }
  }
```

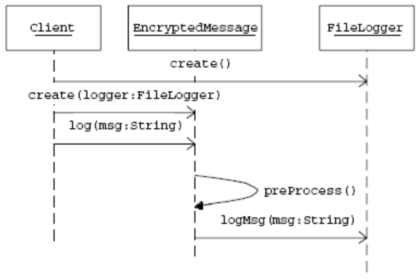
Bridge Pattern George Blankenship 9

Logger Structure



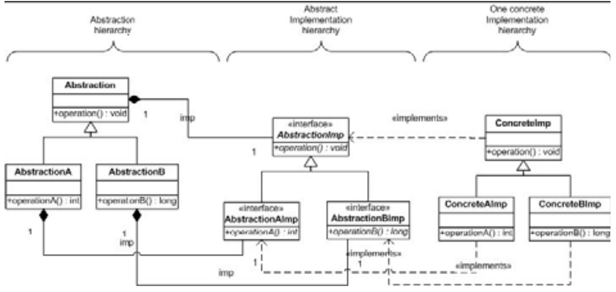
Bridge Pattern George Blankenship 10

Processing Flow



Bridge Pattern George Blankenship 11

The Bridge Pattern: Structure

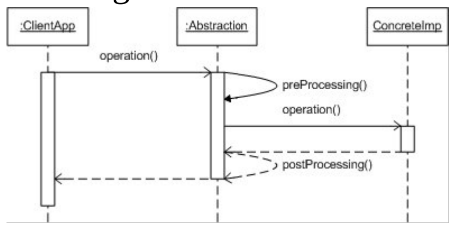


Bridge Pattern George Blankenship 12

The Bridge Pattern: Participants

- Abstraction
  - Class or Class hierarchy
  - Defines the interface
- Implementation
  - Class or Class hierarchy
  - Defines actual interface
  - Defines underlying behavior
- Concrete implementation
  - Instantiation of implementation classes

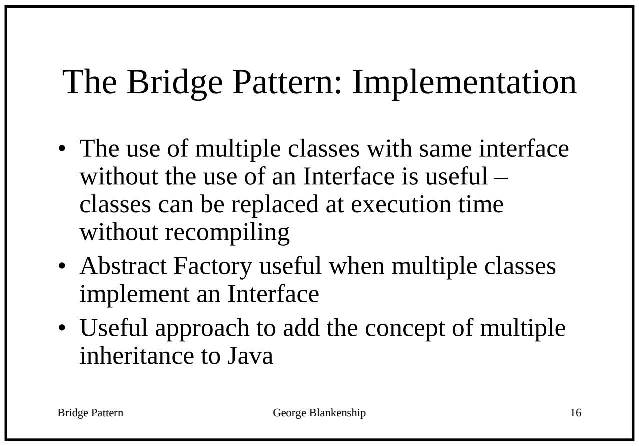
The Bridge Pattern: Collaboration



- Client references abstraction interface
- Abstraction interface invokes proper instantiated object

The Bridge Pattern: Consequences

- Decoupling of interface and implementation
  - Implementation is configured at run-time
  - Places library behind an interface
- Hierarchy of interface and implementation are independent
- Implementation is hidden from clients



The Bridge Pattern: Implementation

- The use of multiple classes with same interface without the use of an Interface is useful – classes can be replaced at execution time without recompiling
- Abstract Factory useful when multiple classes implement an Interface
- Useful approach to add the concept of multiple inheritance to Java

Bridge Pattern George Blankenship 16