Advanced Software Engineering:

Design Principles

- **Location 1:** **Single Responsibility Principle (SRP) Violation.**
- **Location 2:** **Dependency Inversion Principle (DIP) & Open/Closed Principle (OCP) Violations.**
- **Location 3:** **Single Responsibility Principle (SRP) Violation.**
- **Location 4:** **Single Responsibility Principle (SRP) Violation.**
- **Location 5:** **Interface Segregation Principle (ISP) Violation.**

Location 1 violations:

❖ Source Code Location

apache-tomcat-6.0.29-src\apache-tomcat-6.0.29-src\java\org\apache\tomcat\util\http\cookies

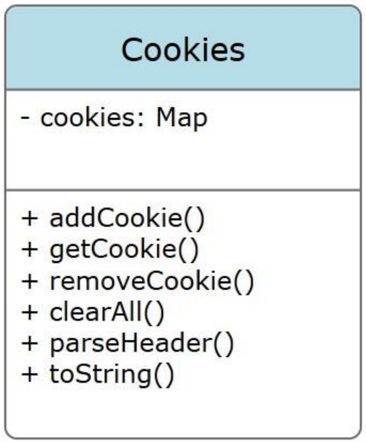
❖ Introduction

The Cookies class violates the **Single Responsibility Principle (SRP)** because it handles multiple responsibilities:

1. Managing cookies in memory.
2. Parsing HTTP headers for cookies.
3. Logging cookie-related operations.

➔Each test requires multiple unrelated setups, also changes to one responsibility risk breaking others.

❖Original Class Diagram



❖ Proposed Updated Design

To overcome the SRP violation, the responsibilities are divided into three distinct classes:

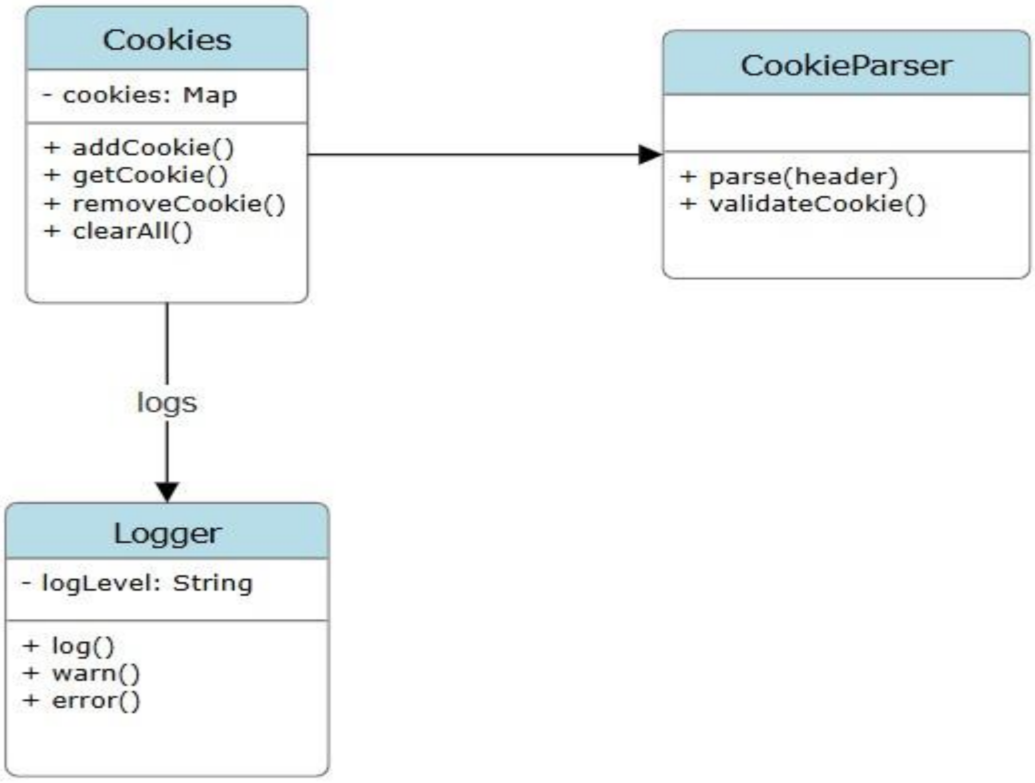
○ CookieManager: Handles the management of cookies in memory. ○ CookieParser:

Focuses on parsing cookies from HTTP headers.

○ Logger: Dedicated to logging cookie-related operations.

➔ Each responsibility (cookie management, parsing, and logging) is isolated, enabling independent testing, adding new parsing rules or logging methods doesn't affect cookie management, CookieParser can be reused for other HTTP header parsing tasks.

❖Updated Class Diagram



Location 2 violations:

❖Source Code Location

apache-tomcat-6.0.29-src\apache-tomcat-6.0.29-src\java\org\apache\tomcat\util\log

❖Identified Design Flaw

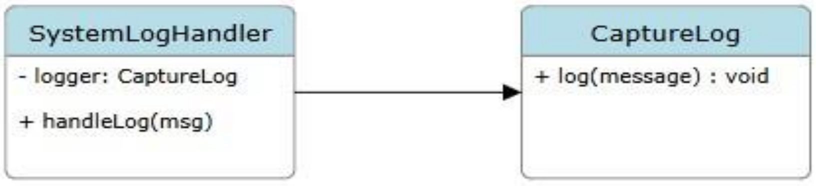
The design flaw in the code violates **Dependency Inversion Principle (DIP)** and **Open/Closed Principle (OCP).**

- **DIP Violation**:
  The SystemLogHandler class directly depends on the concrete class CaptureLog. This tight coupling makes it hard to modify or replace the logging mechanism without altering the SystemLogHandler code.

- **OCP Violation**:
  To introduce a new logging mechanism, you would need to modify the existing SystemLogHandler class, violating the principle of keeping existing code closed to modification but open to extension.

➔ Making it hard to maintain, reducing flexibility and reusability.

❖Original Class Diagram

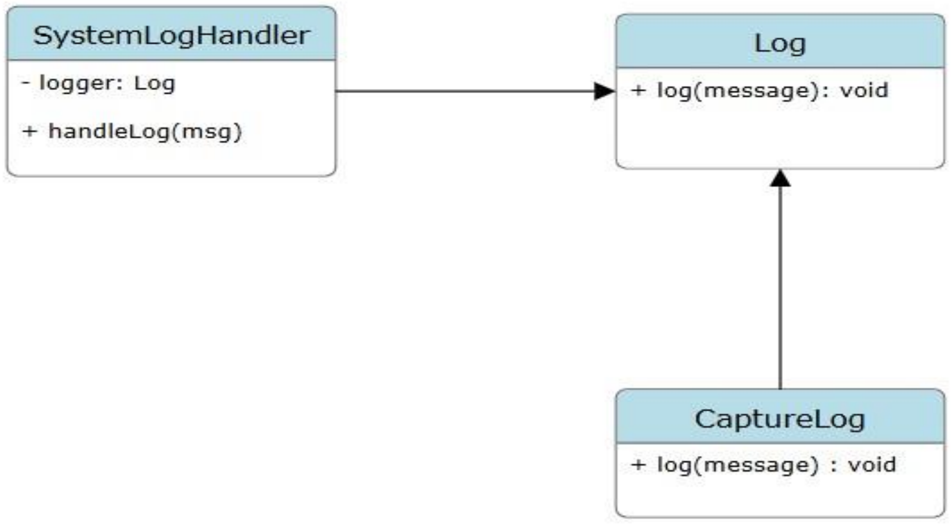


❖ Proposed Updated Design

1. Introduce an interface Log that defines the logging contract with a method saveLog(String message).
2. Modify CaptureLog to implement the Log interface.
3. Update SystemLogHandler to depend on the Log interface instead of the concrete CaptureLog class.

➔ Adding new loggers is straightforward. For example, you can add a DatabaseLog implementation without touching SystemLogHandler, also bug fixes or enhancements in one logger don't propagate into others, and by depending on a Log interface, SystemLogHandler avoids unnecessary coupling.

❖Updated Class Diagram



Location 3 violations:

❖Identified Design Flaw

The AJP implementation in the AjpProtocol class violates the **Single Responsibility Principle (SRP).** The class performs multiple responsibilities:

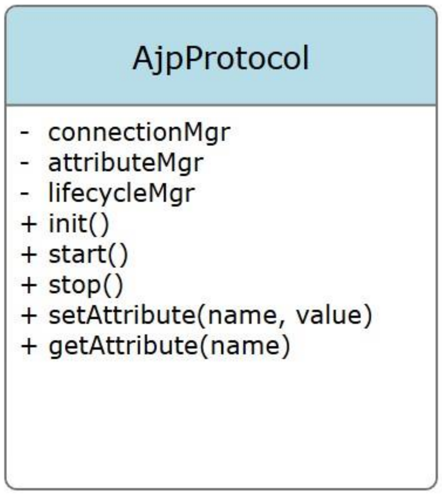
1. Managing connections.
2. Managing attributes associated with requests/responses.
3. Dealing with lifecycle management of connections.

➔ This tightly coupled design makes the class difficult to maintain, test, extending one functionality risks breaking others, and bugs in one area affect the entire class.

❖Source Code Location

apache-tomcat-6.0.29-src\apache-tomcat-6.0.29-src \java\org\apache\coyote\ajp

❖ Original Class Diagram



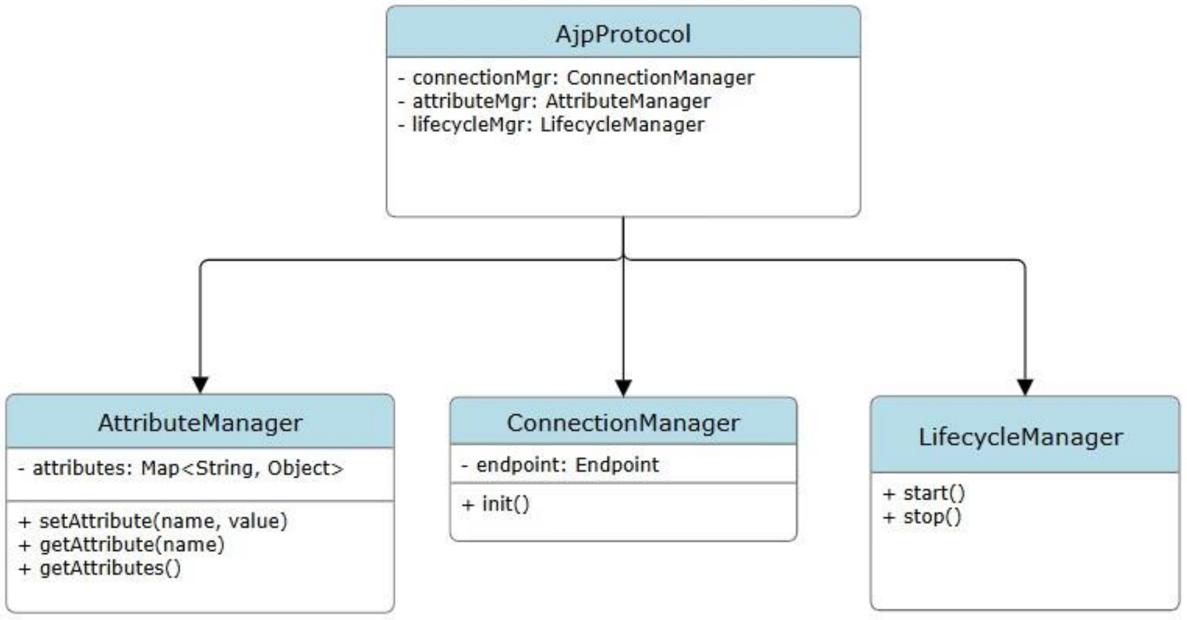
❖ Proposed Updated Design

The updated design separates the responsibilities into distinct components. This aligns with the SRP by ensuring that each class is responsible for a single functionality:

- ConnectionManager: Manages connections and sockets.
- AttributeManager: Handles request and response attributes.
- LifecycleManager: Manages the lifecycle of AJP components (start/stop logic).

➔ Each class focuses on a single responsibility, reducing the risk of bugs, **Enhanced Test Coverage** for specific responsibilities, improving reliability, and Adding new lifecycle operations or connection types doesn't require modifying unrelated code.

❖Updated Class Diagram


AjpProtocol
- connectionMgr: ConnectionManager
- attributeMgr: AttributeManager
- lifecycleMgr: LifecycleManager
AttributeManager
- attributes: Map<String, Object>
+ setAttribute(name, value)
+ getAttribute(name)
+ getAttributes()
ConnectionManager
- endpoint: Endpoint
+ init()
LifecycleManager
+ start()
+ stop()

Location 4 violations:

❖Identified Design Flaw

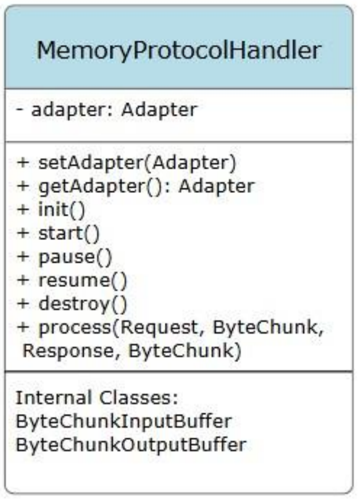
The MemoryProtocolHandler class is responsible for multiple functionalities, making it a clear violation of **SRP**:
1. Protocol lifecycle management (init, start, pause, resume, destroy).
2. Request and response handling via adapters.
3. Input and output buffering using internal nested classes (ByteChunkInputBuffer, ByteChunkOutputBuffer).

➔ This design tightly couples protocol lifecycle and I/O handling, making the class difficult to modify or extend without altering its existing code, which also violates OCP

❖Source Code Location

apache-tomcat-6.0.29-src\apache-tomcat-6.0.29-src\java\org\apache\coyote\memory

❖Original Class Diagram



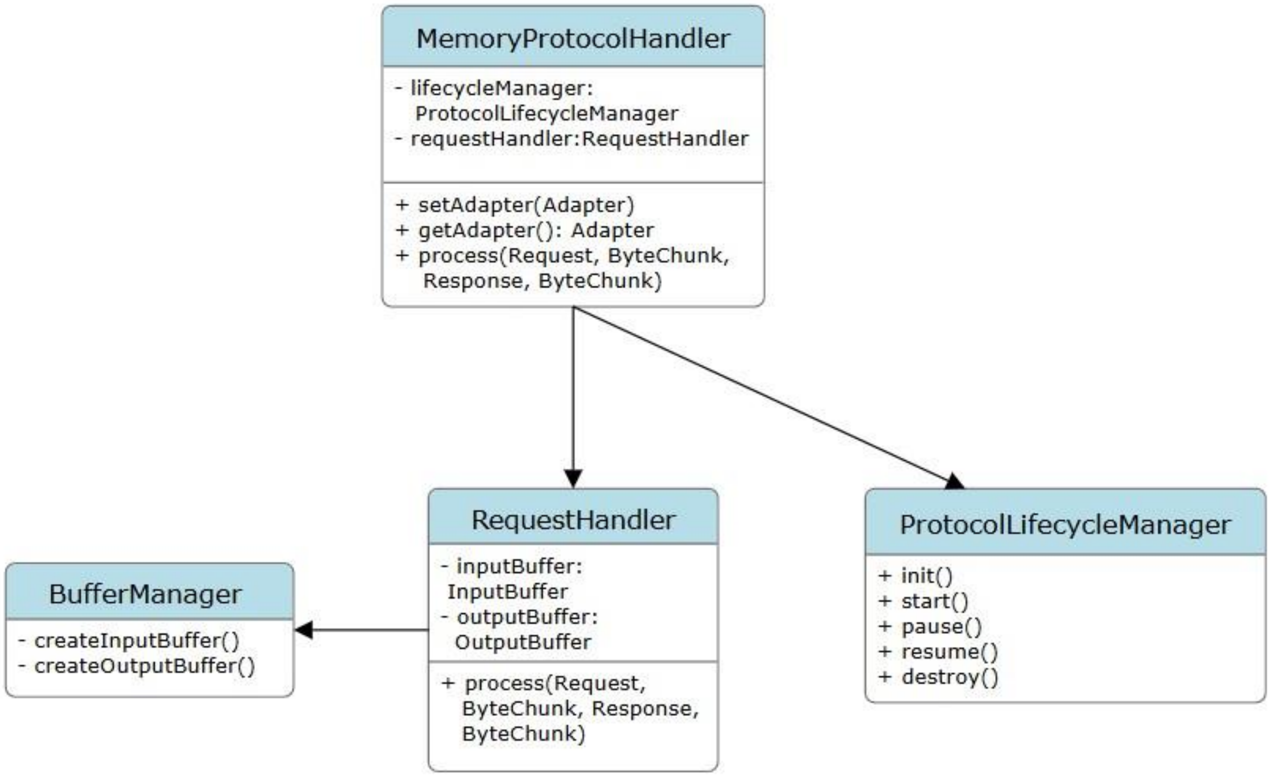
❖Proposed Updated Design

The class should be broken into specialized components:

1. ProtocolLifecycleManager: Manages the lifecycle of the protocol.
2. RequestHandler: Handles request and response processing.
3. BufferManager: Manages input and output buffering.

➔ This separation ensures SRP compliance, as each component focuses on a single responsibility. Furthermore, if a new buffering mechanism is required, we can extend the BufferManager without modifying the existing MemoryProtocolHandler.

❖Updated Class Diagram


MemoryProtocolHandler
- lifecycleManager: ProtocolLifecycleManager
- requestHandler:RequestHandler
+ setAdapter(Adapter)
+ getAdapter(): Adapter
+ process(Request, ByteChunk, Response, ByteChunk)
RequestHandler
- inputBuffer: InputBuffer
- outputBuffer: OutputBuffer
+ process(Request, ByteChunk, Response, ByteChunk)
ProtocolLifecycleManager
+ init()
+ start()
+ pause()
+ resume()
+ destroy()
BufferManager
- createInputBuffer()
- createOutputBuffer()

Location 5 violations:

❖Identified Design Flaw

The design flaw in VariableMapperFactory lies in its violation of the **Interface Segregation Principle (ISP).**

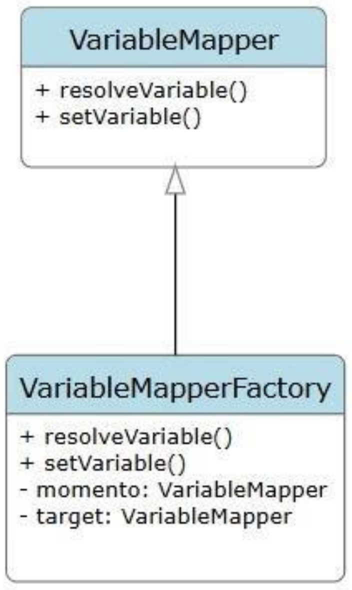
VariableMapperFactory inherits from VariableMapper, which defines two methods: resolveVariable and setVariable. Where VariableMapperFactory has no use for the setVariable method.

➔ The issue violates the Interface Segregation Principle, forcing VariableMapperFactory to implement methods it doesn't need, leading to fragility, runtime exceptions, and reduced maintainability.

❖Source Code Location

apache-tomcat-6.0.29-src\apache-tomcat-6.0.29-src \java\org\apache\el\lang\ VariableMapperFactory

❖Original Class Diagram

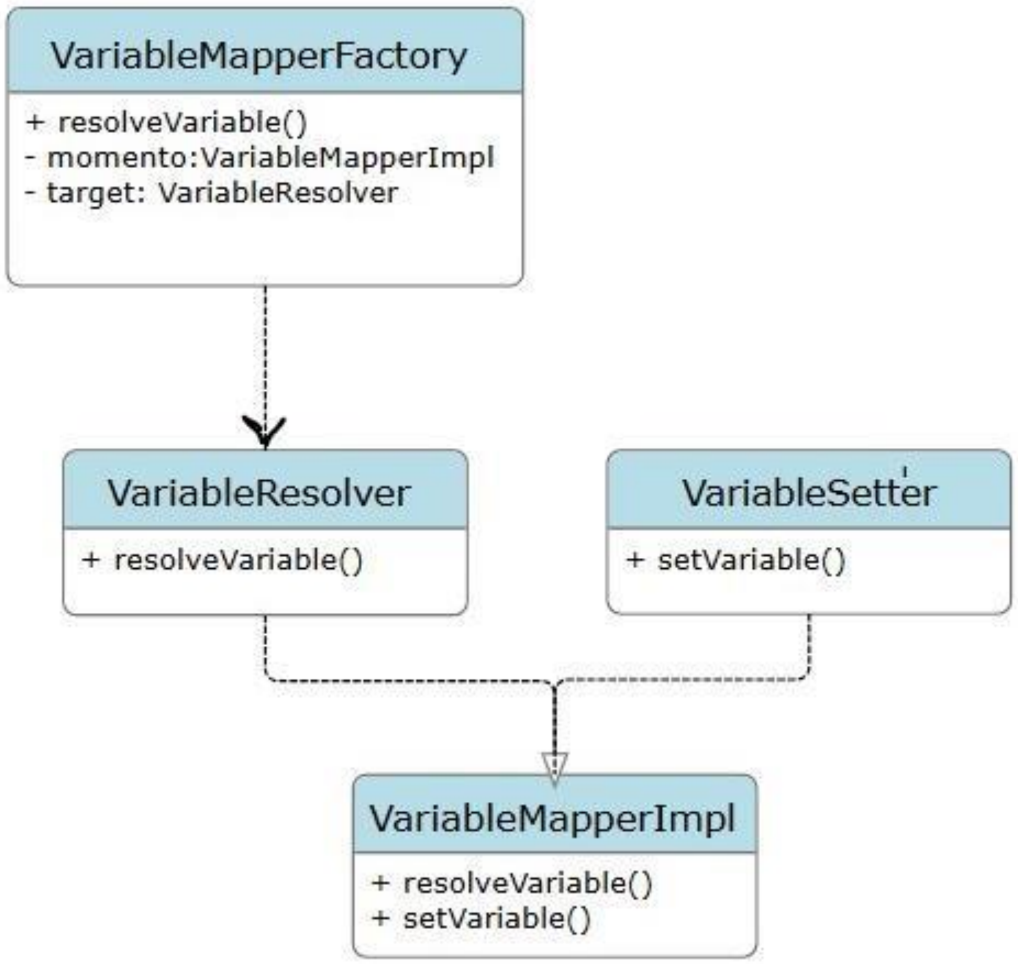


❖Proposed Updated Design

Create smaller, focused interfaces like VariableResolver and VariableSetter. VariableMapperFactory will implement only VariableResolver, Classes that need both can implement multiple interfaces, and refactor VariableMapper to avoid enforcing unused methods on its subclasses.

➔ The solution simplifies the design by splitting interfaces, ensuring each class implements only the methods it requires, improving robustness, scalability, and compliance with SOLID principles.

❖ Updated Class Diagram


VariableMapperFactory
+ resolveVariable()
- momento:VariableMapperImpl
- target: VariableResolver
VariableResolver
+ resolveVariable()
VariableSetter
+ setVariable()
VariableMapperImpl
+ resolveVariable()
+ setVariable()