Design Patterns

Software modeling

What is a design pattern?

- Introduced by “The Gang of Four”:
  - Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm
- Proposes a **reusable** solution to a **common** problem
- Based on much experience
- Template that works in many solutions
- Supports software developers to reach a good solution fast

Types of Design

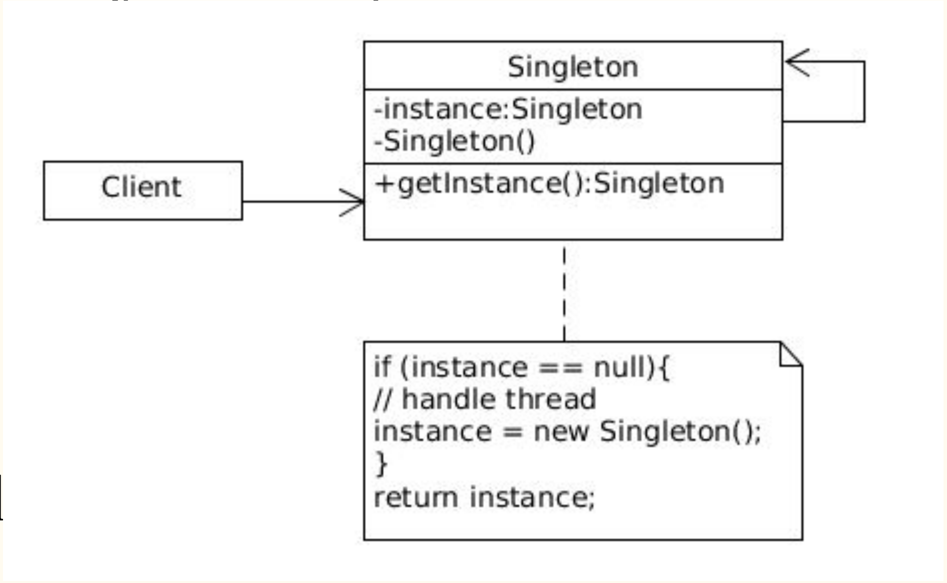
- Creational
    - How to create objects
    - About control, maintainability, extensibility
- Structural
    - How to form structures
    - Manage complexity, efficiency
- Behavioral
    - How to distribute responsibility across objects
    - Communication, flexibility

What do Design Patterns Describe?

- Explains
  - The problem it solves
  - Structured Classes that solve the problem
  - Explanation of how it solves the problem
- Provides examples
  - Pseudocode
- Describes consequences

Singleton - Creation

- When a single instance of an object is required.
  - i.e. Database, Log
- Private constructor
- Static creation method
  - Use private constructor
  - Save in static var
  - All calls return same val



```
if (instance == null){
// handle thread
instance = new Singleton();
}
return instance;
```

Classic Singleton Example

```
Public class ClassicSingleton {
    private static ClassicSingleton instance = null;
    protected ClassicSingleton() {
        // to prevent instantiation.}
    public static ClassicSingleton getInstance() {
        if(instance == null) {
           instance = new ClassicSingleton();
        }
        return instance;
      }
}
```

Test the Singleton

```
import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;
import org.junit.Assert;
import junit.framework.TestCase;

public class SingletonTest extends TestCase {
  private ClassicSingleton singleton1 = null, singleton2 = null;
  private static Logger logger = Logger.getRootLogger();
  public SingletonTest(String name) {
     super(name);
     BasicConfigurator.configure();}
```

Test singleton cont.

```
 public void setUp() {
      logger.info("get singleton 1 instance");
      singleton1 = ClassicSingleton.getInstance();
      logger.info("The singleton: " + singleton1);
      logger.info("get singleton 2 instance");
      singleton2 = ClassicSingleton.getInstance();
      logger.info("The singleton: " + singleton2);}
  public void testUnique() {
      logger.info("check that singletons are equal");
      Assert.assertEquals(true, singleton1 == singleton2);}
}
```

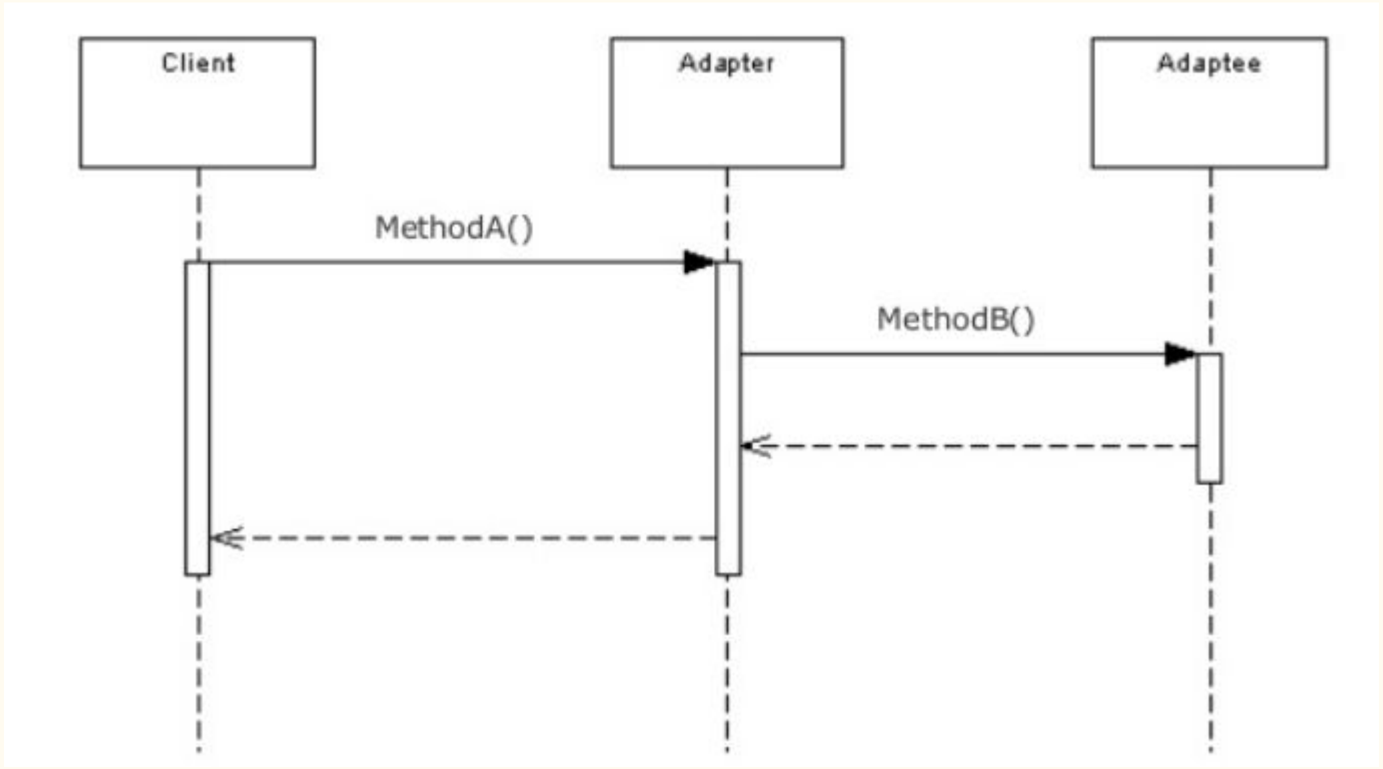
Result of test:

```
[main] INFO root  - get singleton 1 instance

[main] INFO root  - The singleton: ClassicSingleton@30946e09

[main] INFO root  - get singleton 2 instance

[main] INFO root  - The singleton: ClassicSingleton@30946e09

[main] INFO root  - check that singletons are equal
```

Structural design patterns

- Proposes structural approaches
  - Adapter -- enabling incompatible classes to work together - glue
  - Proxy - a class acting on behalf of another
  - Facade -- one class representing many others

Adapter



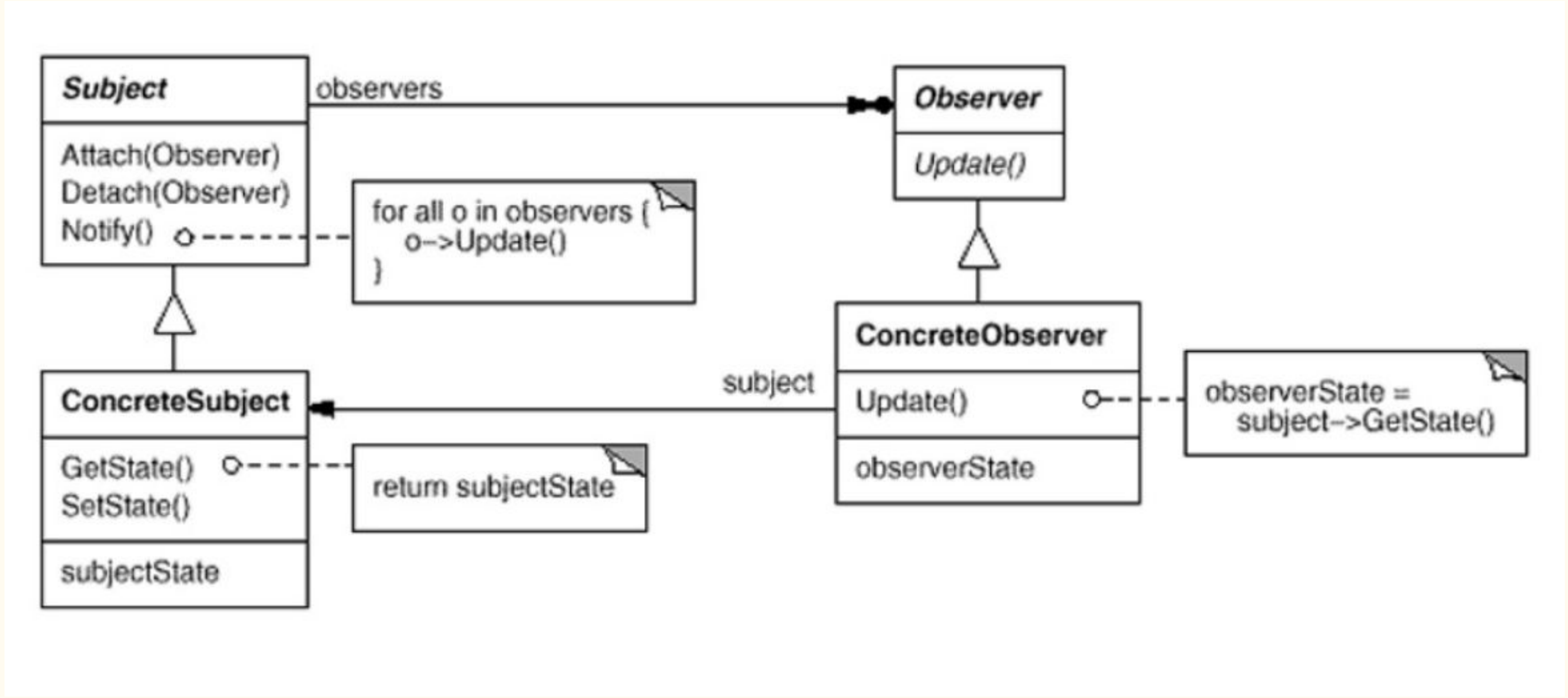
Behavioral Patterns

- Observer -- Notify specific changes (aware of each other)
- Publication / Subscription - Sub/Pub   -- Subscribe to changes and get notifications (not aware of each other)
- Interpreter -- Add language aspects
- Iterator -- Iterate over collection
- Mediator -- Simplify communication
- State -- Alter state
- Many more....

Observer

- Notify all dependents of a change
- Broadcast
- State change triggers behavior in other objects
- Subject -- the object being observed
- Observers -- the object observing the subject

Observer Design



Useful Links

- https://sourcemaking.com/design_patterns
- https://refactoring.guru/design-patterns/
- Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Ralph Johnson, John Vlissides, Richard Helm