Proxy Pattern

Scenario:

Imagine you're developing a document editor that works with large files or external resources like images stored on a remote server. Loading these resources directly might be time-consuming or resource-intensive. You want to manage access to these resources efficiently, perhaps by loading them only when they're actually needed, or by controlling access based on user permissions.

Purpose:

The Proxy Pattern provides a surrogate or placeholder for another object to control access to it. The proxy controls access to the real object, allowing you to perform actions like lazy initialization, access control, logging, or caching.

When to Use:

- When you need to control access to an object, such as for lazy initialization, logging, or access control.
- When you want to defer the full cost of object creation or initialization until it's actually needed.
- When you need a more sophisticated reference to an object than a simple pointer.

Types of Proxies:

- Virtual Proxy: Controls access to an object that is expensive to create by deferring its creation until it's needed.
- Protection Proxy: Controls access to an object based on user permissions or roles.
- Remote Proxy: Represents an object that exists in a different address space, such as on a remote server.
- Caching Proxy: Provides temporary storage of the results of expensive operations, so that

repeated requests can be served more quickly.

How It Works:

1. Subject Interface: Defines the common interface for the real object and its proxy.
2. Real Subject: The actual object that the proxy represents and controls access to.
3. Proxy: Implements the same interface as the real subject and controls access to the real subject, often adding additional behavior like lazy loading, access control, or logging.
4. Client: Interacts with the subject through the proxy, unaware of whether it is dealing with the real subject or the proxy.

Implementation Example:

Here is how the implementation might look in Java:

```
// Subject Interface
interface Document {
    void displayContent();
}

// Real Subject
class RealDocument implements Document {
    private String content;

    public RealDocument(String content) {
        this.content = content;
        loadFromDisk();
    }
    private void loadFromDisk() {

        System.out.println("Loading document from disk...");

    }


    @Override

    public void displayContent() {

        System.out.println("Document Content: " + content);

    }

}


// Proxy

class DocumentProxy implements Document {

    private String content;

    private RealDocument realDocument;


    public DocumentProxy(String content) {

        this.content = content;

    }


    @Override

    public void displayContent() {

        if (realDocument == null) {

            realDocument = new RealDocument(content); // Lazy initialization

        }

        realDocument.displayContent();
    }
}


// Client Code
public class DocumentEditor {
    public static void main(String[] args) {
        Document document = new DocumentProxy("Design Patterns in Java");


        // Document content is loaded and displayed only when needed
        System.out.println("First display:");
        document.displayContent(); // Triggers loading from disk


        System.out.println("\nSecond display:");
        document.displayContent(); // Uses already loaded content
    }
}
```

Key Points to Remember:

- Proxy Pattern provides a way to control access to an object, adding an additional layer between the client and the real object. This layer can be used for various purposes like lazy initialization, logging, or access control.

- Different Types of Proxies: Depending on the use case, the proxy can serve different purposes (e.g., virtual, protection, remote, caching).

Advantages:

- Controlled Access: The proxy controls access to the real object, allowing you to manage when and how the object is accessed.
- Performance Optimization: By deferring expensive operations until they are needed (e.g., loading large files), the proxy can optimize the performance of the application.
- Security: The proxy can control access to the object, ensuring that only authorized clients can interact with it.

Disadvantages:

- Increased Complexity: Introducing a proxy adds another layer of complexity to the code.
- Potential Performance Overhead: Depending on the implementation, the proxy might introduce a slight performance overhead, especially if it adds logging or access control.