Exercise 6: Implementing the Proxy Pattern

Objective

To use the Proxy Pattern in an image viewer app for: Lazy initialization (load only when needed)

Caching (reuse already loaded image)

Java Code – ProxyPatternExample.java

// Step 1: Subject Interface interface Image {

void display();

}

// Step 2: Real Subject Class class RealImage implements Image {

private String filename;

public RealImage(String filename) { this.filename = filename; loadFromServer();

}

private void loadFromServer() {

System.out.println("Loading image from remote server: " + filename);

}

@Override

public void display() { System.out.println("Displaying: " + filename);

}

}

// Step 3: Proxy Class

class ProxyImage implements Image { private String filename; private RealImage realImage;

public ProxyImage(String filename) { this.filename = filename;

}

@Override

public void display() {

if (realImage == null) {

realImage = new RealImage(filename); // Lazy loading

} else {

System.out.println("Image loaded from cache: " + filename);

}

realImage.display();

}

}

// Step 4: Test Class

public class ProxyPatternExample {

public static void main(String[] args) {

Image img1 = new ProxyImage("nature.jpg");

// First time: loads from server img1.display();

// Second time: uses cached version img1.display();

// Another image

Image img2 = new ProxyImage("city.png"); img2.display();

}

}

Sample Output

Loading image from remote server: nature.jpg Displaying: nature.jpg

Image loaded from cache: nature.jpg Displaying: nature.jpg

Loading image from remote server: city.png Displaying: city.png

Key Benefits of Proxy Pattern

Feature Description

Lazy Initialization Loads resource only when needed

Caching Support Reuses loaded images without reloading from server Performance Boost Reduces network/server calls for repeated access Access Control (optional) Can restrict or control access to real object