Image Viewer with Proxy Pattern

import java.util.HashMap;

import java.util.Map;

public class ProxyPatternExample {

public static void main(String[] args) {

System.out.println("Image Viewer Application");

System.out.println("-----------------------");

Image image1 = new ProxyImage("nature.jpg");

Image image2 = new ProxyImage("cityscape.jpg");

Image image3 = new ProxyImage("nature.jpg");

System.out.println("\nImages created but not loaded yet");

System.out.println("\nDisplaying image1 (first time):");

image1.display();

System.out.println("\nDisplaying image2 (first time):");

image2.display();

System.out.println("\nDisplaying image1 (second time - cached):");

image1.display();

System.out.println("\nDisplaying image3 (should use cached nature.jpg):");

image3.display();

}

interface Image {

void display();

}

static class RealImage implements Image {

private final String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromServer();

}

private void loadFromServer() {

System.out.println("Loading " + filename + " from remote server... (expensive operation)");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

public void display() {

System.out.println("Displaying " + filename);

}

}

static class ProxyImage implements Image {

private final String filename;

private static final Map<String, RealImage> imageCache = new HashMap<>();

public ProxyImage(String filename) {

this.filename = filename;

}

@Override

public void display() {

// Lazy initialization and caching

RealImage realImage = imageCache.get(filename);

if (realImage == null) {

realImage = new RealImage(filename);

imageCache.put(filename, realImage);

}

realImage.display();

}

}

}

Key Features:

1. **Lazy Loading**: Images are only loaded when first displayed
2. **Caching**: Subsequent requests for the same image use the cached version
3. **Virtual Proxy**: Proxy stands in for the real object until needed
4. **Protection Proxy**: Can add access control if needed
5. **Remote Proxy**: Simulates loading from a remote server

Proxy Pattern Benefits:

1. **Performance Optimization**: Avoids expensive operations until needed
2. **Memory Efficiency**: Only loads resource-intensive objects when required
3. **Caching**: Reuses previously loaded objects
4. **Access Control**: Can add permission checks (not shown in this example)
5. **Location Transparency**: Client code works the same with proxy or real object

Output

