**Exercise 6: Implementing the Proxy Pattern**

**Scenario:**

You are developing an image viewer application that loads images from a remote server. Use the Proxy Pattern to add lazy initialization and caching.

**Steps:**

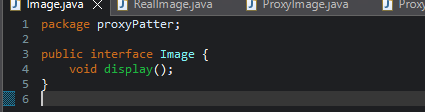
1. **Create a New Java Project:**
   * Create a new Java project named **ProxyPatternExample**.
2. **Define Subject Interface:**
   * Create an interface Image with a method **display()**.
3. **Implement Real Subject Class:**
   * Create a class **RealImage** that implements Image and loads an image from a remote server.
4. **Implement Proxy Class:**
   * Create a class **ProxyImage** that implements Image and holds a reference to RealImage.
   * Implement lazy initialization and caching in **ProxyImage**.
5. **Test the Proxy Implementation:**
   * Create a test class to demonstrate the use of **ProxyImage** to load and display images.

**Solution:**

| **Component** | **Role** | **Purpose** |
| --- | --- | --- |
| **Image** | Subject Interface | Declares method display() used by both real and proxy |
| **RealImage** | Real Subject | Represents the actual image fetched from remote server |
| **ProxyImage** | Proxy | Manages access to RealImage (lazy load + cache) |
| **ProxyTest** | Client Code | Uses Image interface and works with ProxyImage |

**Image.java**

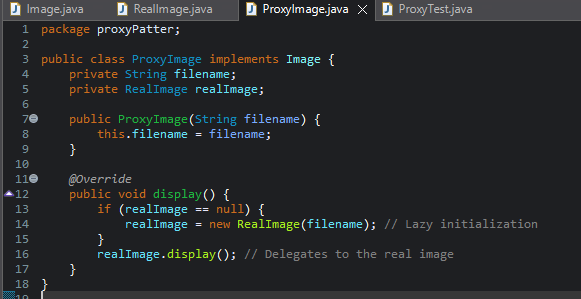
**Purpose:** Defines the common method display() to show the image.  
 Both RealImage and ProxyImage will implement this.



**RealImage.java**

**Purpose**

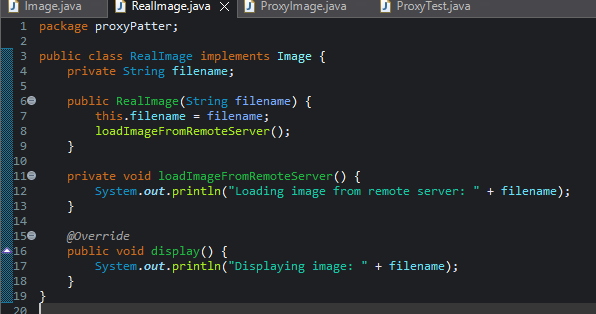
* loadImageFromRemoteServer() runs only when an object is created.
* Simulates heavy operation (image loading) in the constructor.

****

**ProxyImage.java**

**Purpose:**

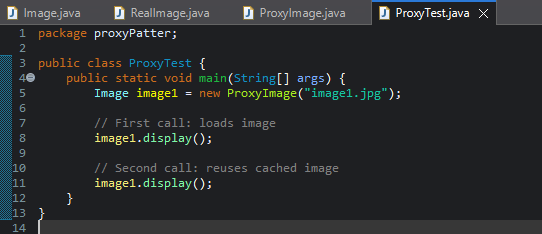
* Implements lazy loading: RealImage is only created when display() is first called.
* Implements caching: On subsequent display() calls, no reloading happens.
* Acts as a stand-in for RealImage.

****

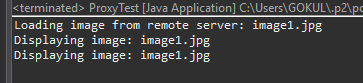
**ProxyTest.java**

**Purpose:**

* Demonstrates how a proxy can delay and control the actual object creation and use.
* Both calls go through the same interface (Image), showing abstraction.

****

**Output:**

****

**Benefits of the Proxy Pattern**.

* ✔ Performance Optimization – Avoids repeated resource loading.
* ✔ Encapsulation – Hides complexity from the client.