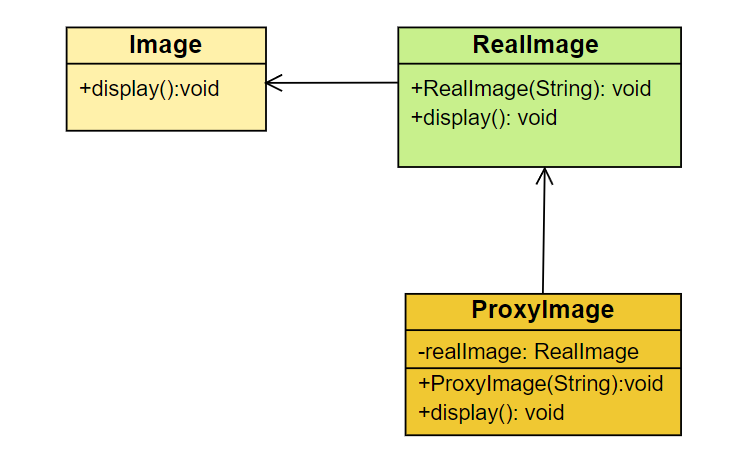
**Exercise 6: Implementing the Proxy Pattern**

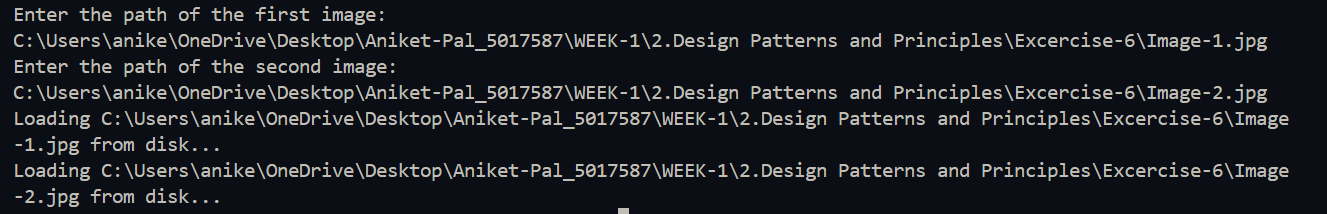
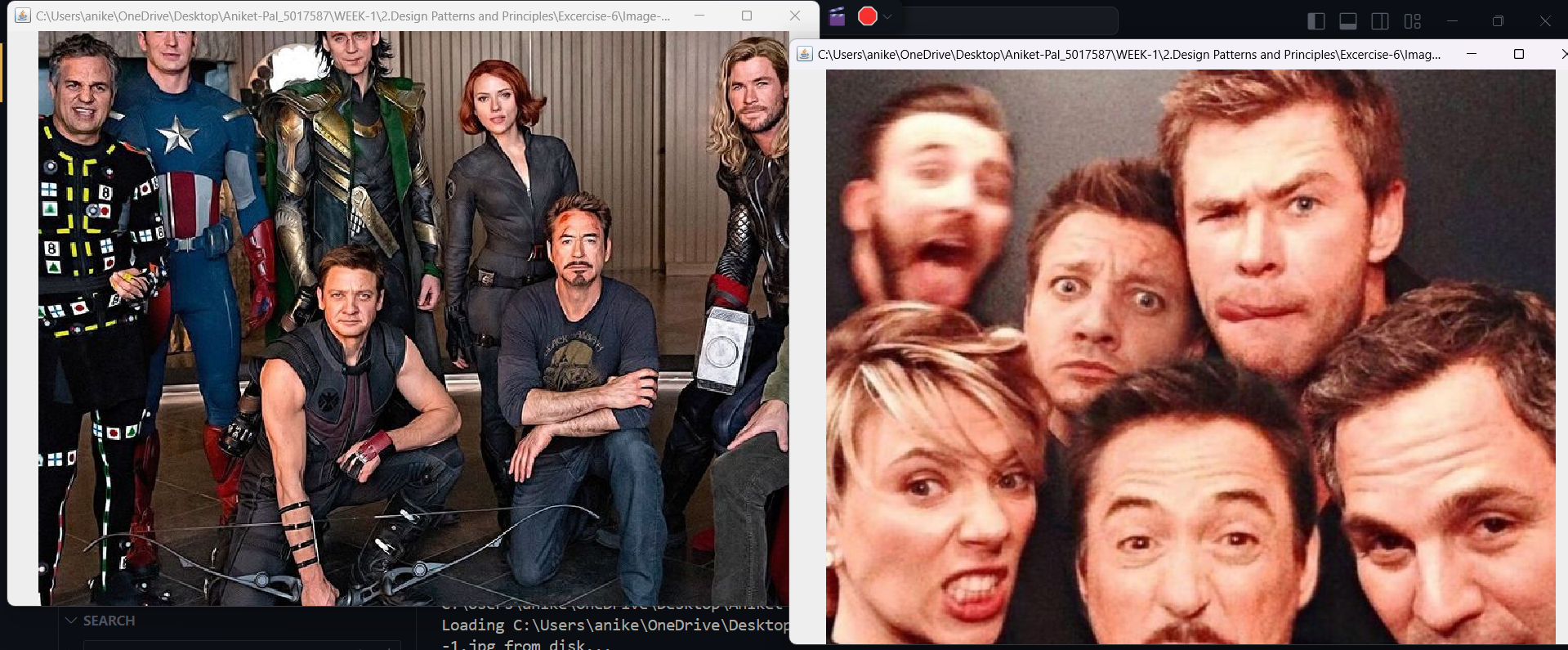
* **Create a New Java Project**: Set up a new project environment.
* Open the Integrated Development Environment (IDE). In this Exercise I use VS Code
* Create a new project and name it *ProxyPatternExample*.
* Set up the project structure according to your IDE's conventions
* **Define Subject Interface:** Establish a common interface that both the real subject and the proxy will implement.
* Create an interface named *Image* with a method *display().*
* This interface defines the contract for displaying images, ensuring both the real image and proxy classes implement this method.
* **Implement Real Subject Class:** Create the actual object that performs the core functionality (loading and displaying images).
* Implement a class named *RealImage* that implements the *Image* interface.
* Include a constructor that takes a file path as an argument.
* Implement the *display()* method to show the image.
* Simulate loading an image from a remote server (in real applications, this would involve network operations).
* **Implement Proxy Class:** Create a proxy that controls access to the real subject, adding lazy initialization and caching.
* Implement a class named *ProxyImage* that also implements the *Image* interface.
* Include a constructor that takes a file path as an argument.
* Hold a reference to an instance of *RealImage*.
* Implement lazy initialization in the *display()* method, where the *RealImage* is only created when needed (when *display()* is called for the first time).
* Implement caching by reusing the *RealImage* instance for subsequent calls to *display().*
* **Test the Proxy Implementation:** Demonstrate the functionality and advantages of the proxy pattern.
* Create a test class, such as *ProxyPatternExample*.
* Instantiate *ProxyImage* objects with paths to image files.
* Call the *display()* method on these objects.
* Observe the *behavior*: the first call to *display()* should load the image (simulating a delay), and subsequent calls should display the image immediately (using the cached instance).

Here the class diagram of the program for better understanding:

* **Image Interface**: This is an interface with a method display().*RealImage* and *ProxyImage* implement this interface.
* **RealImage Class**: Implements the Image interface. Contains the actual logic to load and display the image. Has a constructor *RealImage*(String filePath) and a method *display().*
* **ProxyImage Class**:

Implements the *Image* interface. Contains a reference to a *RealImage* object. Has a constructor *ProxyImage*(String filePath) and a method display(). Implements lazy initialization and caching by creating the *RealImage* object only when needed.

* **Here is the code** – [link](https://github.com/Hyperstrom/Aniket-Pal_5017587/tree/main/WEEK-1/2.Design%20Patterns%20and%20Principles/Excercise-6)
* **Here is the Output of the code –**



This is how the code output looks like