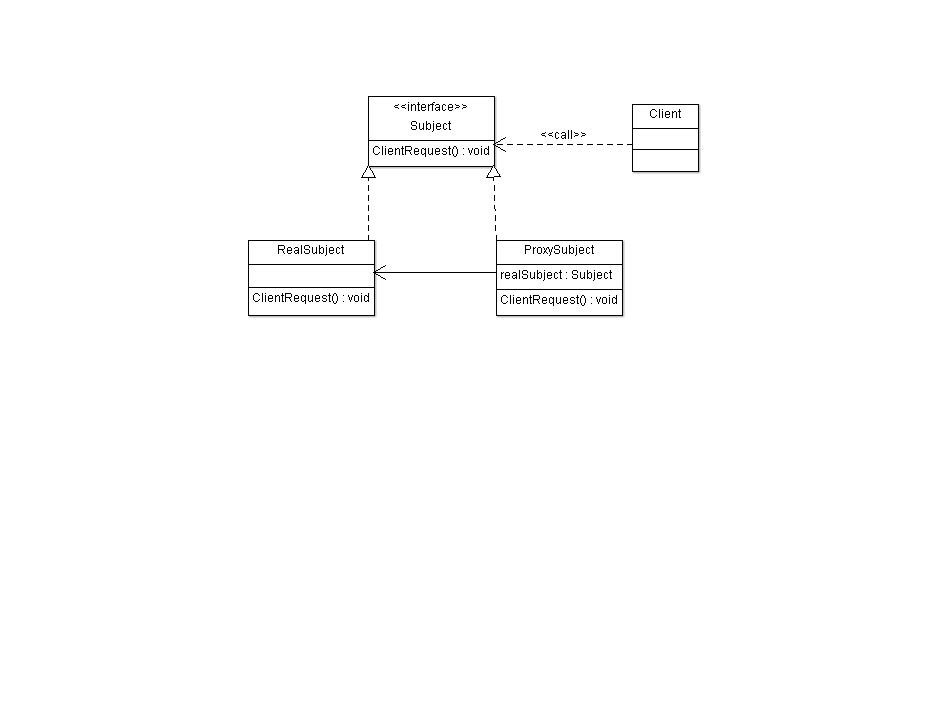
OODP - Object Oriented Design and Programming

Exercise 3

Sachith Dassanayake (12712980)

**Proxy Design Pattern**

The proxy pattern is a software design pattern which can be used to control access to heavy & costly objects by creating a wrapper class as the proxy. The proxy class can also add additional functionalities without changing the real class. The proxy design pattern provides a common interface to the proxy and the real object so that the proxy can be used when the real object is expected.



UML Diagram for Proxy Design Pattern

**Subject**

The interface for RealSubject. The interface must be implemented by the proxy subject so it can be used in any location where the RealSubject is expected.

**ProxySubject**

This class implements the same interface used by the RealSubject. It keeps a reference to the RealSubject so it can be substituted for the Realsubject. The proxy controls the access to the RealSubject and it can also add other functionalities without changing the RealSubject.

**RealSubject**

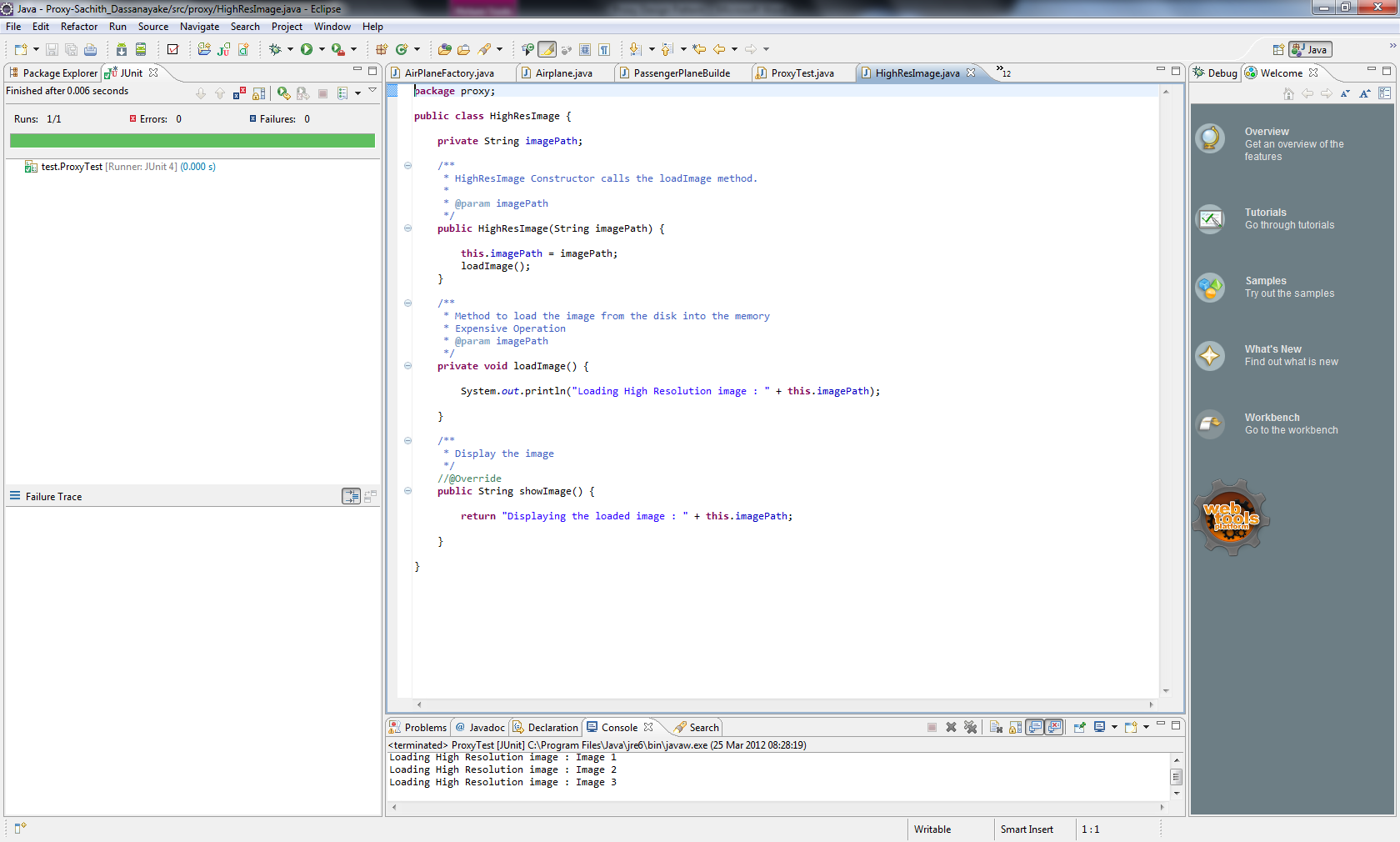
The real object with heavy and costly operations.

Let’s look at an example of the proxy design pattern - a simple picture viewer application. The application should list all the images on the disk. Let’s assume the images on the disk are high resolution and large files.

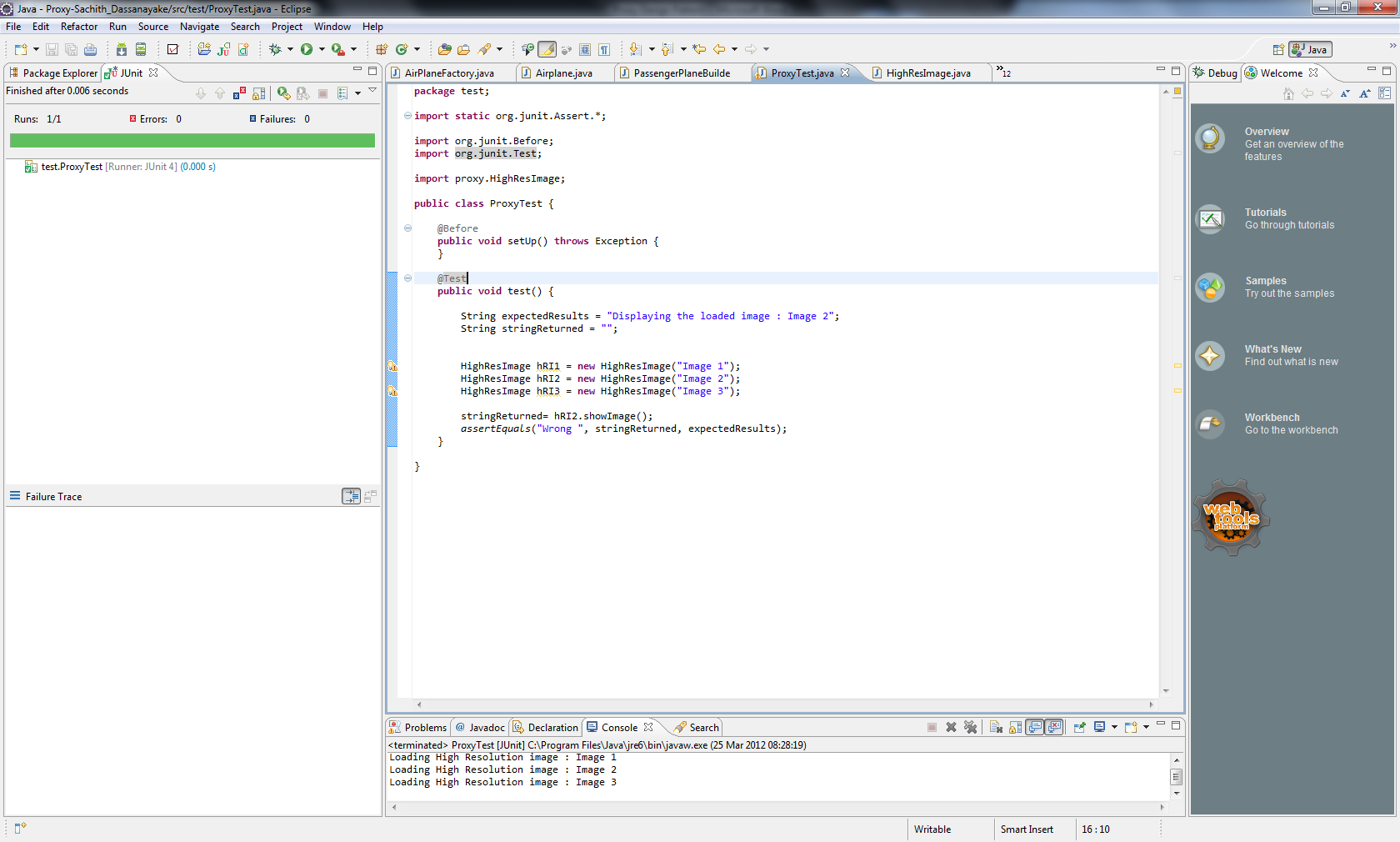
The user needs to see the names of all the files available on the disk. He may only view the files he wishes to open.

To load all the images on to the memory can be very costly in terms of resources. So it’s better if the program can display only the file names first and then display the image when the user selects it.

First we will consider doing it without using a design pattern. We create a class to initialize and load each and every image on the disk.



Class to load the images from the disk and display - HighResImage.java



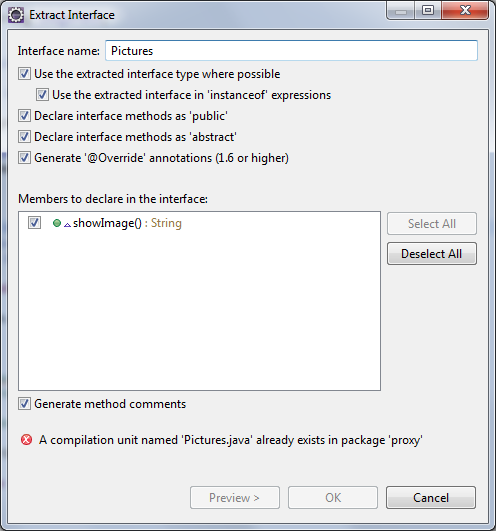
Test case to test the HighResImage class – ProxyTest.java

This is a simple demonstration so the actual coding to load the image from the disk is not implemented instead it gives a console output to alert the user what it does.

As you can see for each and every image on the disk there will be an object loaded on to the memory. This can be very heavy process.

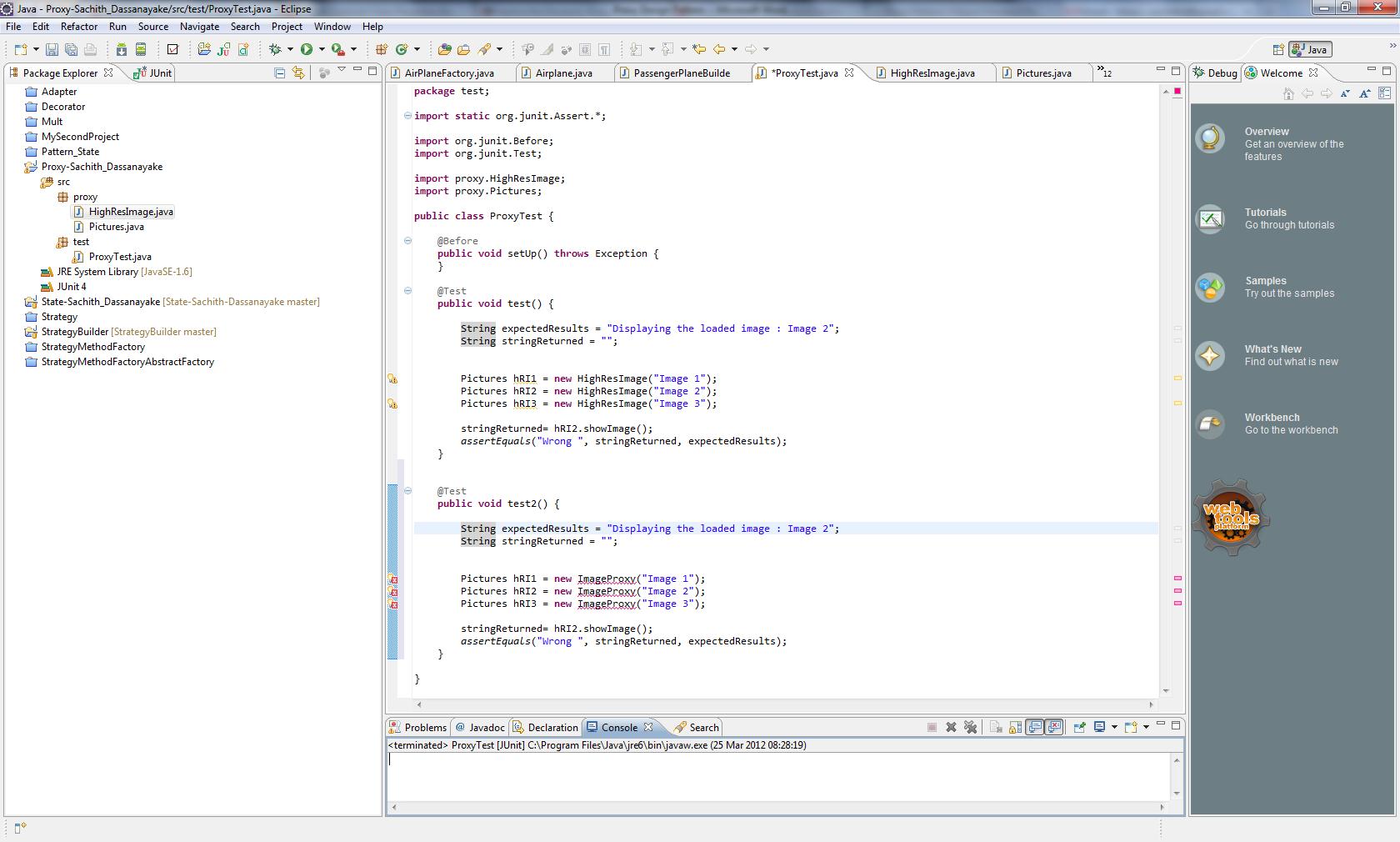
Let’s see how we can introduce the Proxy design pattern to overcome this issue by refactoring our code.

Extract the interface from HighResImage class Pictures. On Eclipse IDE right click on the class name -> Refactor -> Extract Interface

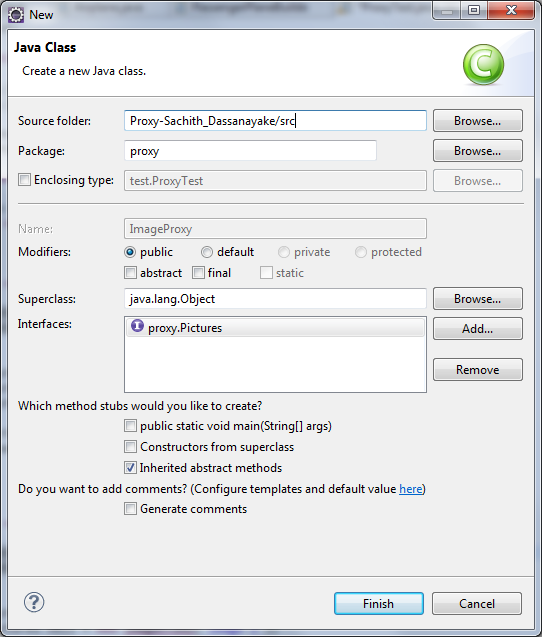


Name the interface as Pictures and we chose showImage() method to declare in the interface.

Create a new test case for the proxy class.

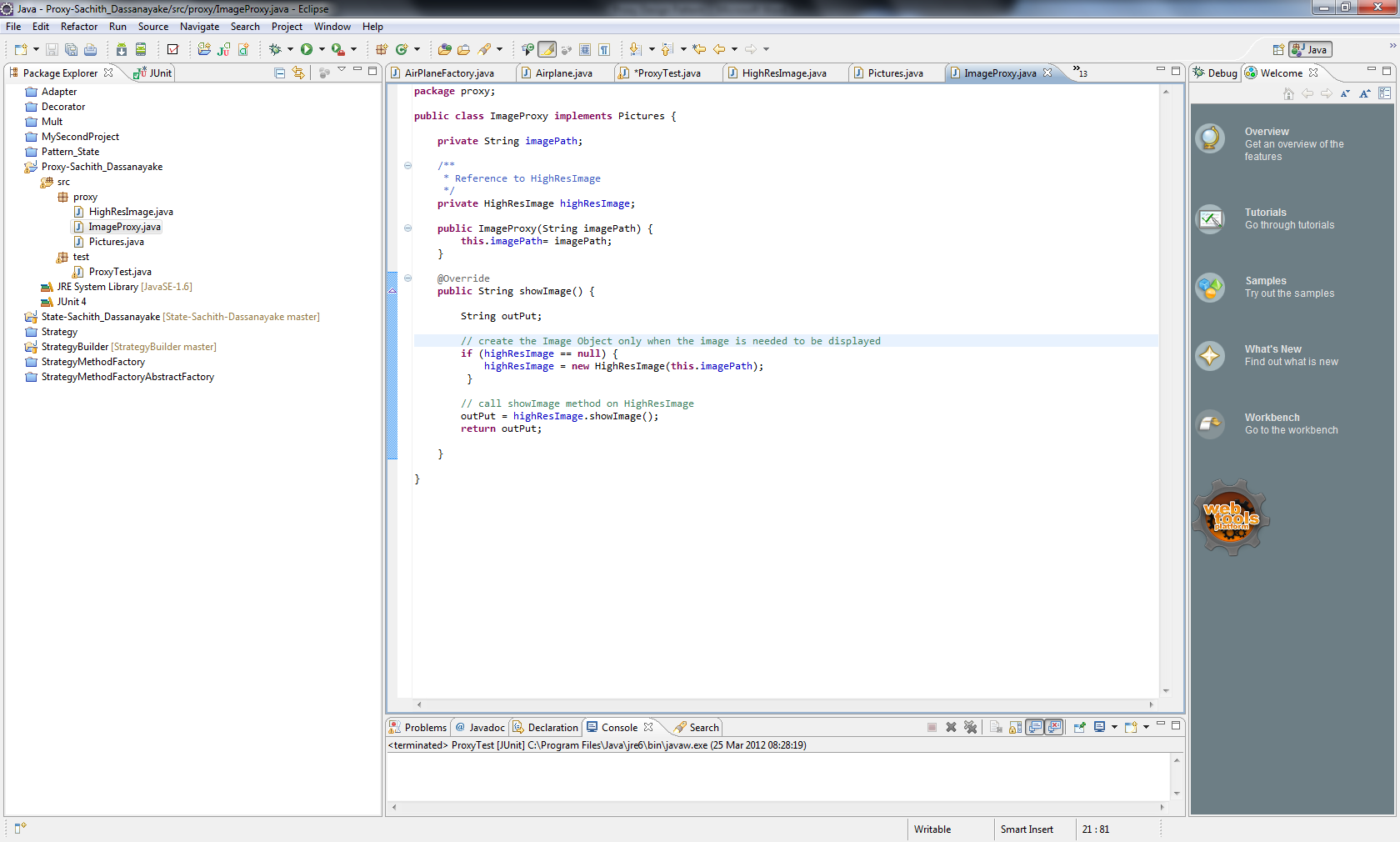


Resolve the errors by creating a new class. Right click on the error and select create class ImageProxy.

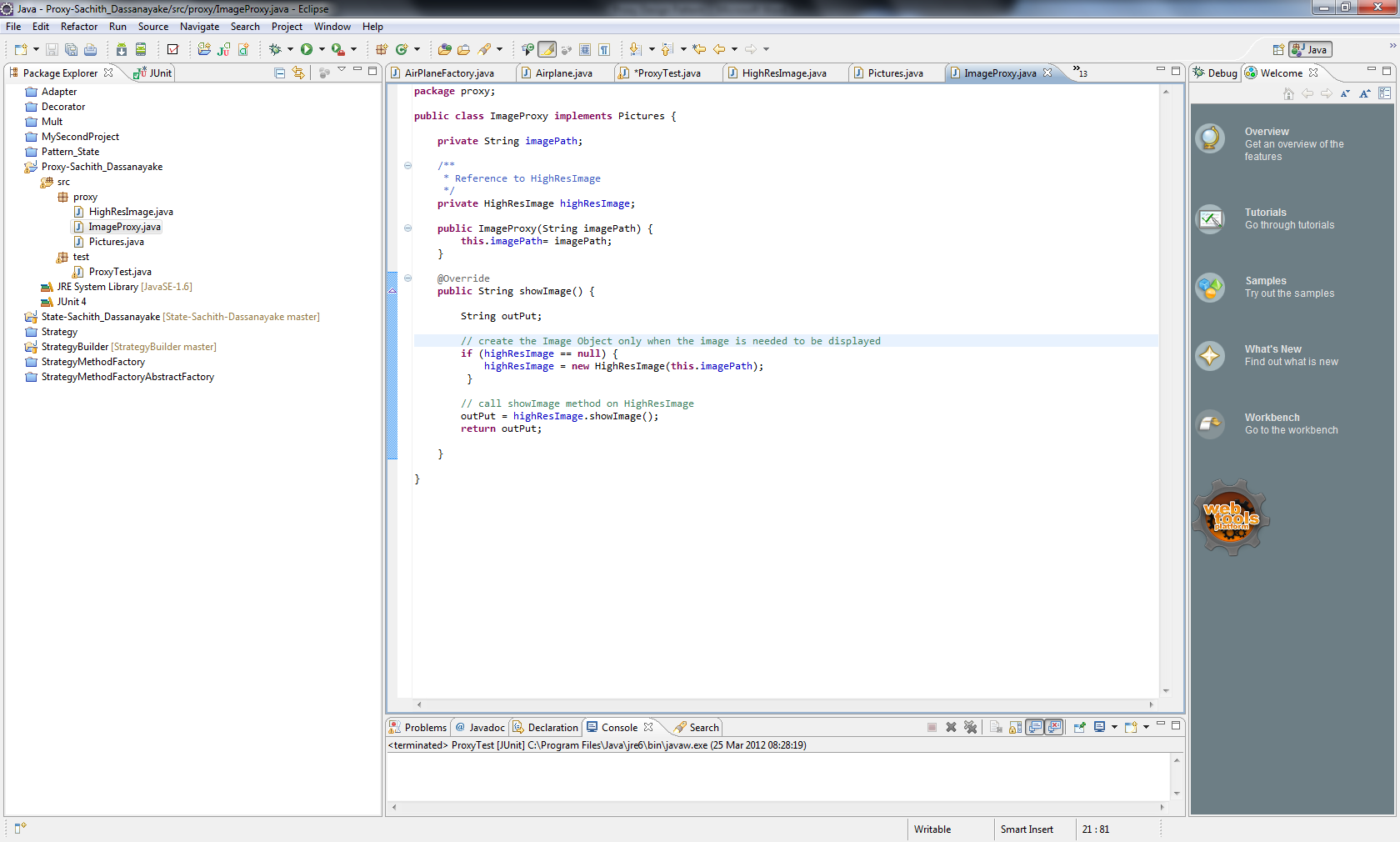


Choose the correct package name and make sure it implements the Pictures interface.

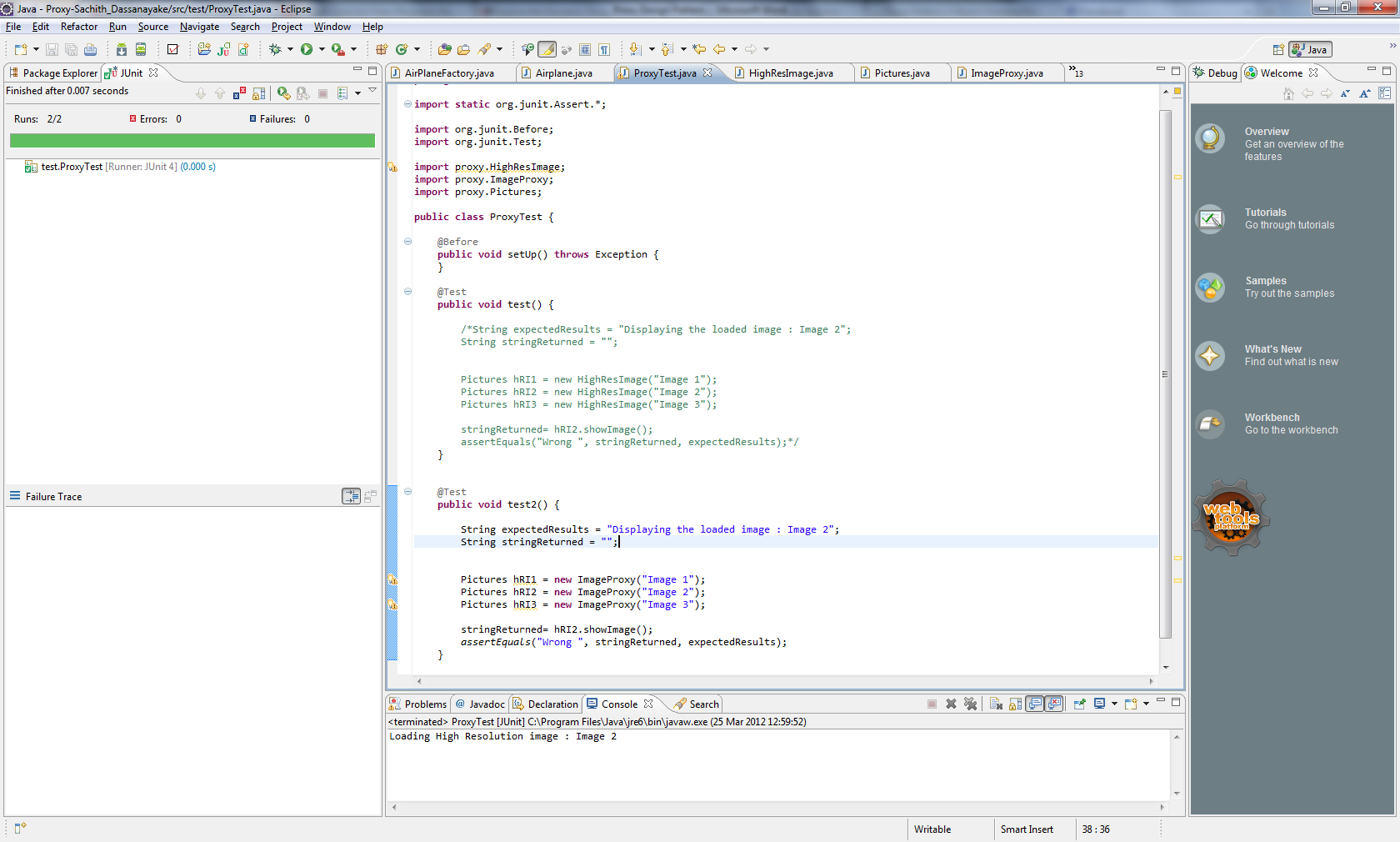
Implement the class ImageProxy as follows.



As you can see the proxy class creates an object of the real class and loads the images only when the showImage() method is called.



Let’s run the test case to see how the refactored code works.



There are 3 images initialized but we want to show only one image. In earlier example we initialized 3 image variables and all 3 images were loaded on to the memory despite we wanted to display only one. But after creating the proxy class even though we created 3 imageProxy objects only the image which needs to be displayed was loaded on to the memory from the disk.

The modification was done without changing any of the client code and the program will work more efficiently because now it loads an image from the disk only when it’s required.

‘Proxy’ design pattern together with another design pattern