**3. What are the TestNG Listeners you used in your project?**

**What are Listeners?**

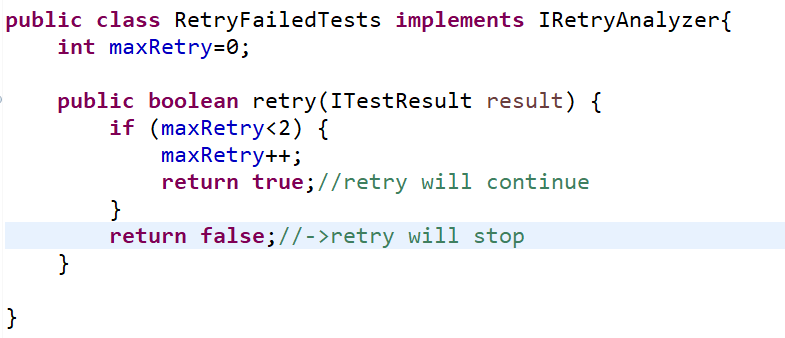
- Listeners are TestNG annotations that “listen” to the events in a script and modify TestNG behavior accordingly. These listeners are applied as interfaces in the code. There are various listeners like IAnnotationTransformer, ITesttListener , IReporter.

- For example, in my project I have used IAnnotationTransformer and IRetryAnalyzer to retry the failed testcases by adding listener tag in my testing.xml file.

**Listener used in my project:**

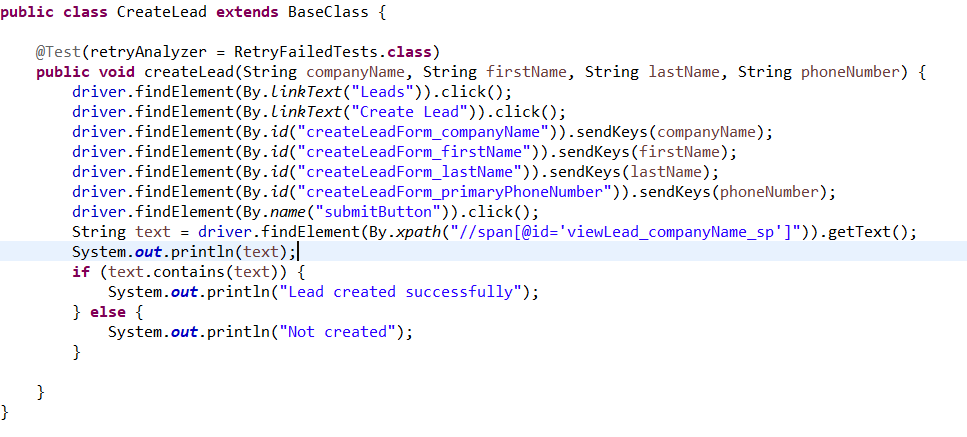
- I have employed Listeners to handle situations where test cases fail due to random browser issues, browser unresponsiveness or unexpected server delays causing issues with the server response. In such cases, we can implement test case rerun using the IAnnotationTransformer Listener.

- To implement this, I first created a RetryFailedTests class which is a user-defined class, which implements the IRetryAnalyzer interface. This interface contains the retry method where we define the logic for retrying failed test cases.

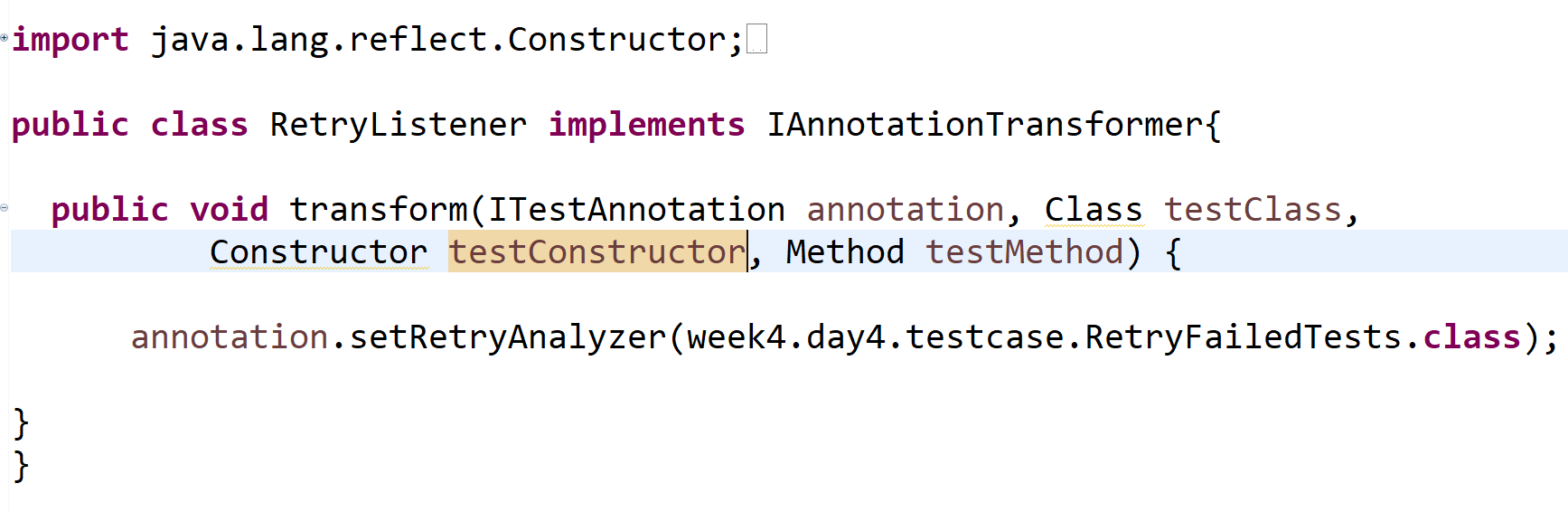
­­

- This retry logic, allows a test to be retried a certain number of times if it fails. The retry method determines whether the test should be retried based on the retry count and the maximum retry count specified.

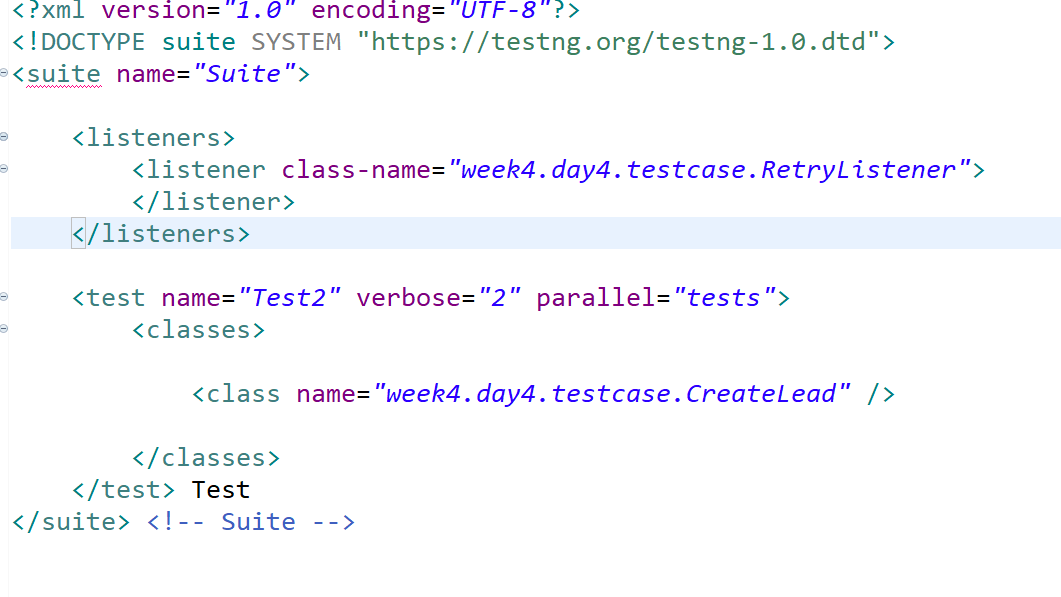
- Then using the retryAnalyzer attribute in the @Test annotation of the test method, I can specify the RetryFailedTests class. But in this case, mentioning the RetryFailedTests class in each @Test is a tedious process since we will have n number of testcases in our project.



- Hence, I utilized the IAnnotationTransformer interface which is a TestNG Listener, which includes the transform method. We can override transform method defined in the IAnnotationTransformer interface.

- Inside the transform method, the setRetryAnalyzer method is called on the annotation object (which represents the test annotation). It sets the retry analyzer for the test annotation to RetryFailedTests.class. 

Finally, I configured the listeners in the XML file using the <Listeners> tag.

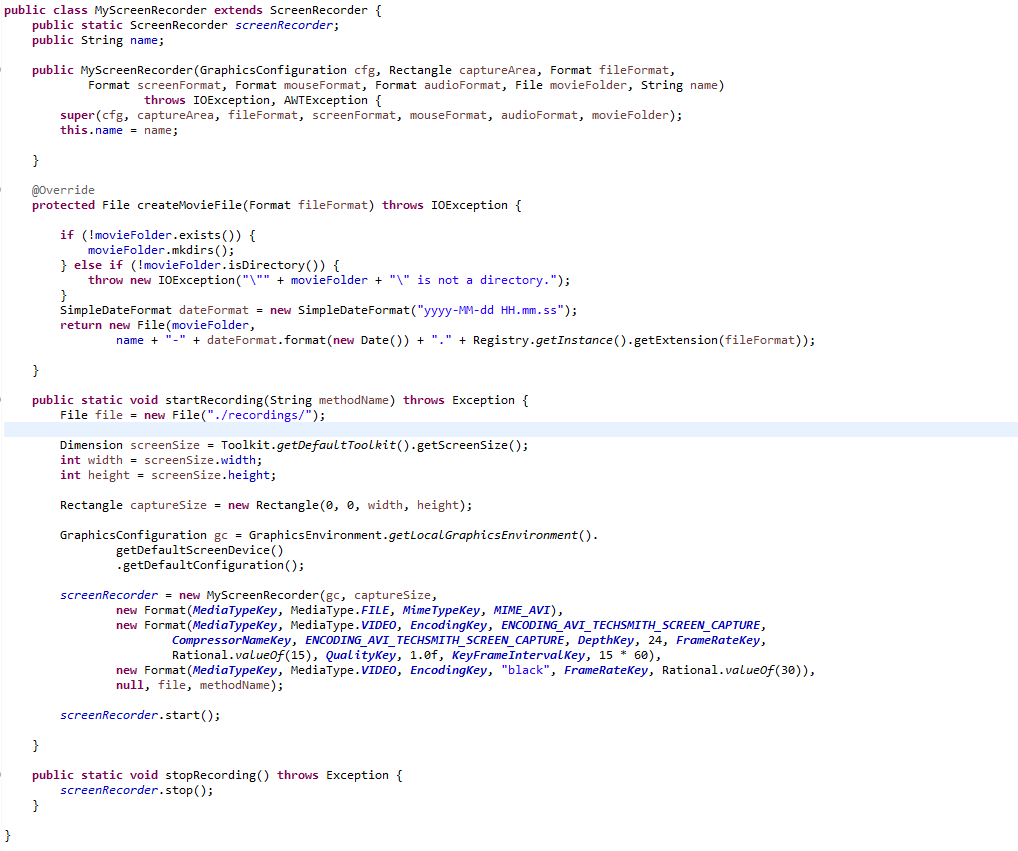


- When we define the <listeners> tag in TestNG XML file and include listener classes within it, TestNG will use these listeners to monitor and respond to events that occur during the test execution like failure of a test.

**ITestListener:**

- I have faced another scenario like In our project the client wants to record the failed testcases, especially when they fail. This helps us visualize what exactly went wrong during a failed test, aiding in debugging and understanding the issue.

- I have integrated a screen recording functionality using a library called "monte-screen-recorder"." I have a user-defined class named MyScreenRecorder which extends ScreenRecorder (which comes from monte.screenRecorder.ScreenRecorder). This class sets up the screen recording parameters and handles the creation of movie files for the recordings.



- I have a user-defined class called TestListener which implements ITestListener Interface where we can override the required methods like onTestStart(), onTestSkip(), onTestSuccess, onTestFailure.

- When a test starts (onTestStart method in TestListener), MyScreenRecorder.startRecording method will be called and provide a unique name for the recording (in this case, the test method name). This method initializes the screen recording process.

- If the test is successful (onTestSuccess), MyScreenRecorder.stopRecording method will be called which stops the screen recording. Additionally, I will check if the file path contains the name of the test that just succeeded (result.getName()).If the condition is true, it deletes the file using delete() which could also clean up the recorded file to maintain a clean and organized recording directory.

- If the test fails (onTestFailure), we also stop the screen recording using the same method.



- The recording includes the entire screen and is saved in the "recordings" directory. We use specific formats and configurations for the recording (e.g., AVI format, frame rate, etc.) to ensure the recording quality.

- So, this integration helps us capture screen recordings specifically during failed test cases, giving us valuable insights into what happened during the test and aiding in identifying the cause of failure.