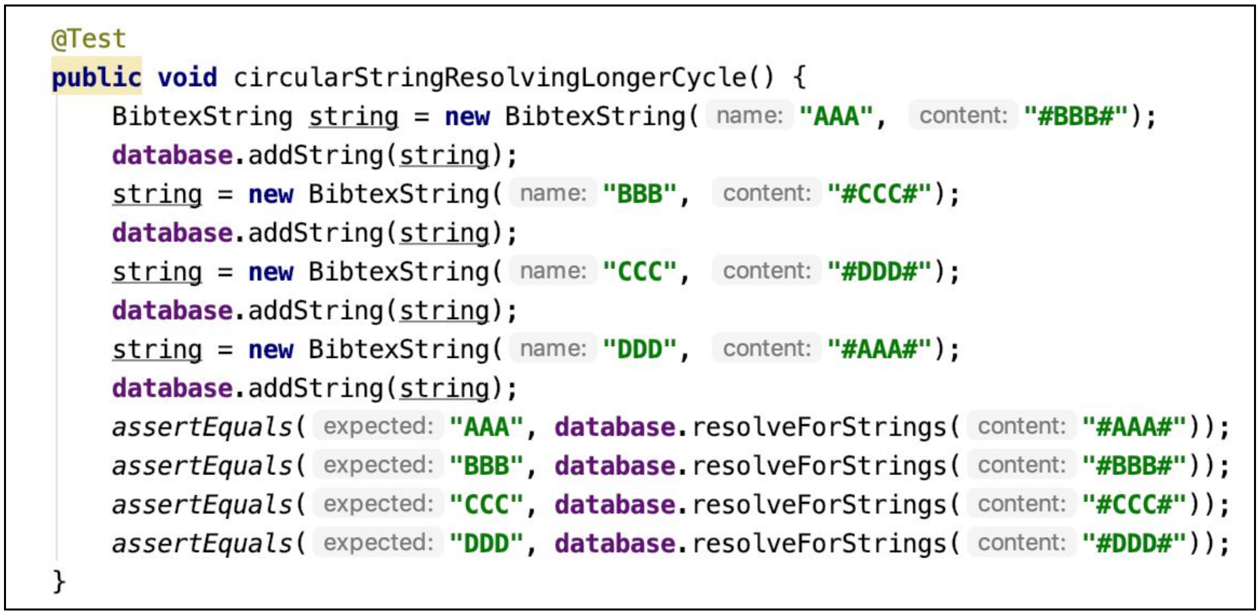
Existing Test Cases

Using the Statistic plugin for IntelliJ, we realised JabRef has 376 existing test cases. These test cases were available in additional categories such as DatabaseTest, FetcherTest and GUITest apart from the traditional Test category. Testing is mainly conducted through ***jUnit 4*** which is a Java framework for Unit tests. The developers advise contributors to stick on not using the Test prefix as this is addressed in the class name. We also learnt that the test case name should have the entire description of the test. Also, each test case should strictly address only a part of the method in the class and test single functionality.

Test Case #1



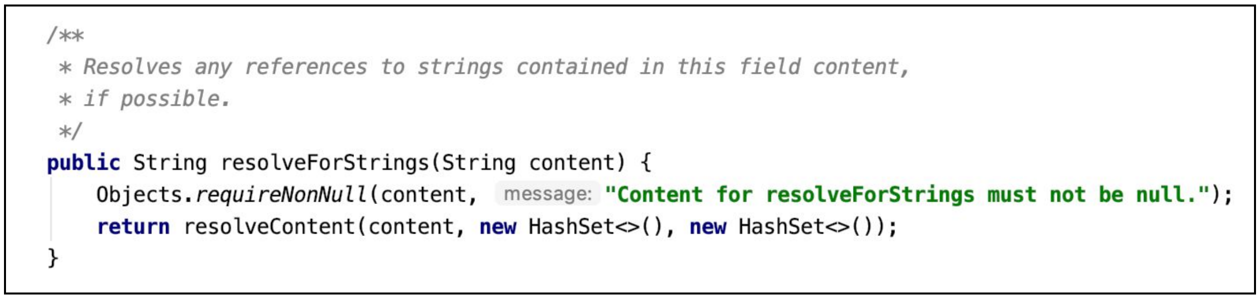
```
@Test
public void circularStringResolvingLongerCycle() {
    BibtexString string = new BibtexString( name: "AAA",  content: "#BBB#");
    database.addString(string);
    string = new BibtexString( name: "BBB",  content: "#CCC#");
    database.addString(string);
    string = new BibtexString( name: "CCC",  content: "#DDD#");
    database.addString(string);
    string = new BibtexString( name: "DDD",  content: "#AAA#");
    database.addString(string);
    assertEquals( expected: "AAA", database.resolveForStrings( content: "#AAA#"));
    assertEquals( expected: "BBB", database.resolveForStrings( content: "#BBB#"));
    assertEquals( expected: "CCC", database.resolveForStrings( content: "#CCC#"));
    assertEquals( expected: "DDD", database.resolveForStrings( content: "#DDD#"));
}
```

Code Snippet (1): circularStringResolvingLongerCycle (BibDatabaseTest)

We started our search by looking into the intricacies of BibDatabaseTest, as for obvious reasons, this is the key to store data for our reference management tool. We noticed a number of mainstream test cases to insert, check, remove entries or to update the string values which were rather straightforward to understand.

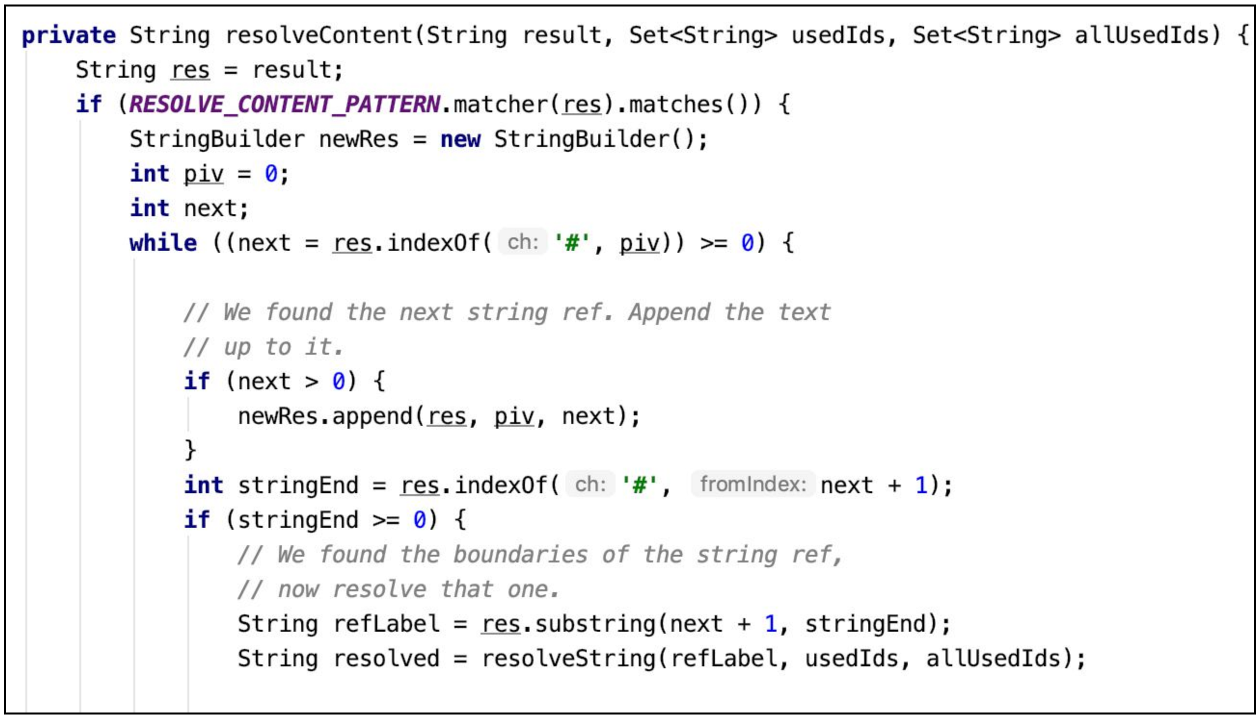
Upon looking for interesting test cases, we noticed the test method shown in Code Snippet(1), it basically defines a BibtexString whose constructor takes values for name and content of the string. This BibtexString is then added to the database. Similarly, the strings are added to the database such that the content forms a circular relation. In the end of the test, we look for assertEquals “AAA” to the content of “#AAA#” given by the resolveForStrings method.

We **learnt** that there is a way the content is being resolved if given within the hash signs and that the name of BibtexString does not matter in that case as long as the content is being provided.



```
/**
 * Resolves any references to strings contained in this field content,
 * if possible.
 */
public String resolveForStrings(String content) {
    Objects.requireNonNull(content, message: "Content for resolveForStrings must not be null.");
    return resolveContent(content, new HashSet<>(), new HashSet<>());
}
```

Code Snippet (2): resolveForStrings method (BibDatabase)

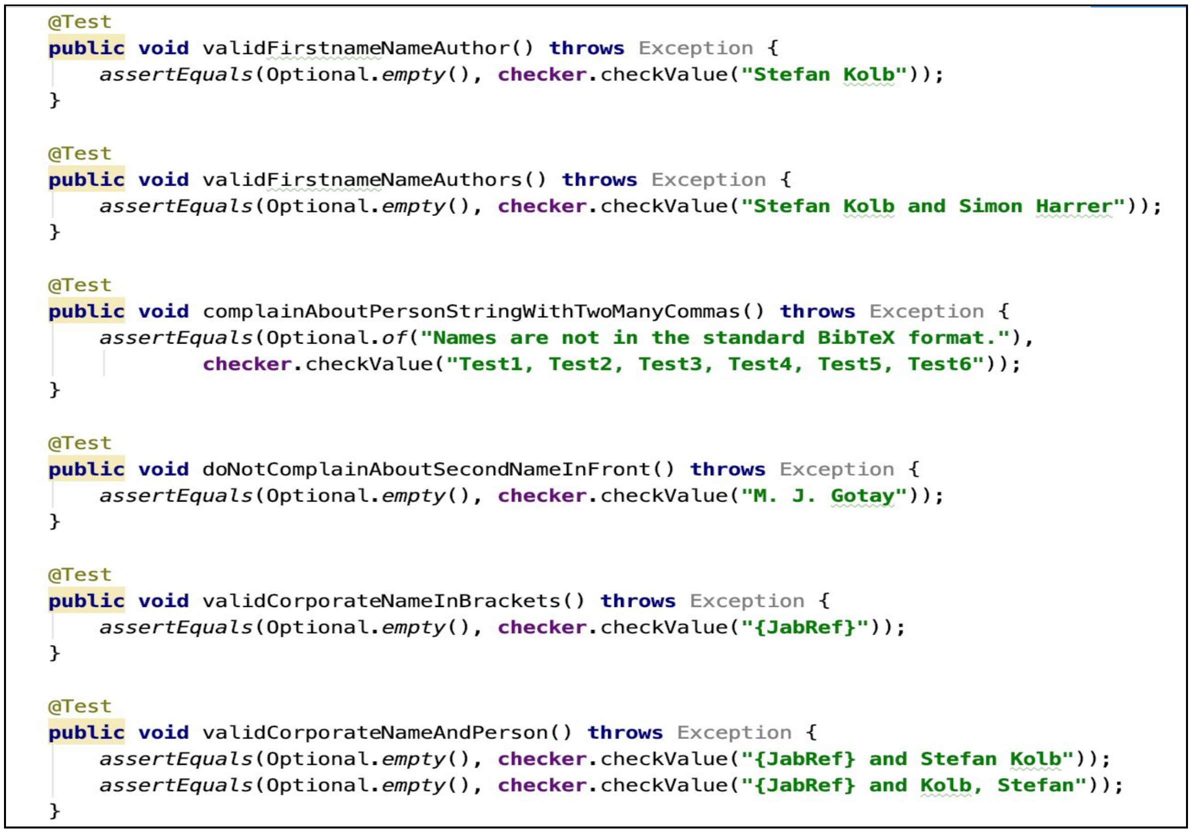


```
private String resolveContent(String result, Set<String> usedIds, Set<String> allUsedIds) {
    String res = result;
    if (RESOLVE_CONTENT_PATTERN.matcher(res).matches()) {
        StringBuilder newRes = new StringBuilder();
        int piv = 0;
        int next;
        while ((next = res.indexOf( ch: '#', piv)) >= 0) {

            // We found the next string ref. Append the text
            // up to it.
            if (next > 0) {
                newRes.append(res, piv, next);
            }
            int stringEnd = res.indexOf( ch: '#', fromIndex: next + 1);
            if (stringEnd >= 0) {
                // We found the boundaries of the string ref,
                // now resolve that one.
                String refLabel = res.substring(next + 1, stringEnd);
                String resolved = resolveString(refLabel, usedIds, allUsedIds);
```

Code Snippet (3): resolveContent method (BibDatabase)

To understand the underlying details behind this resolution, we took a look at the resolveForStrings method (as shown in Code Snippet (2))which takes in the content string and calls resolveContent method with the content string and two new empty hash sets, which inturn returns a string. The resolveContent (as shown in Code Snippet (3)) helped us in understanding that the "#" are removed and are appended as such to form the string content and the hash sets are used in storing the string content retrieved after removing the hash signs and hence the name of the string is irrelevant.

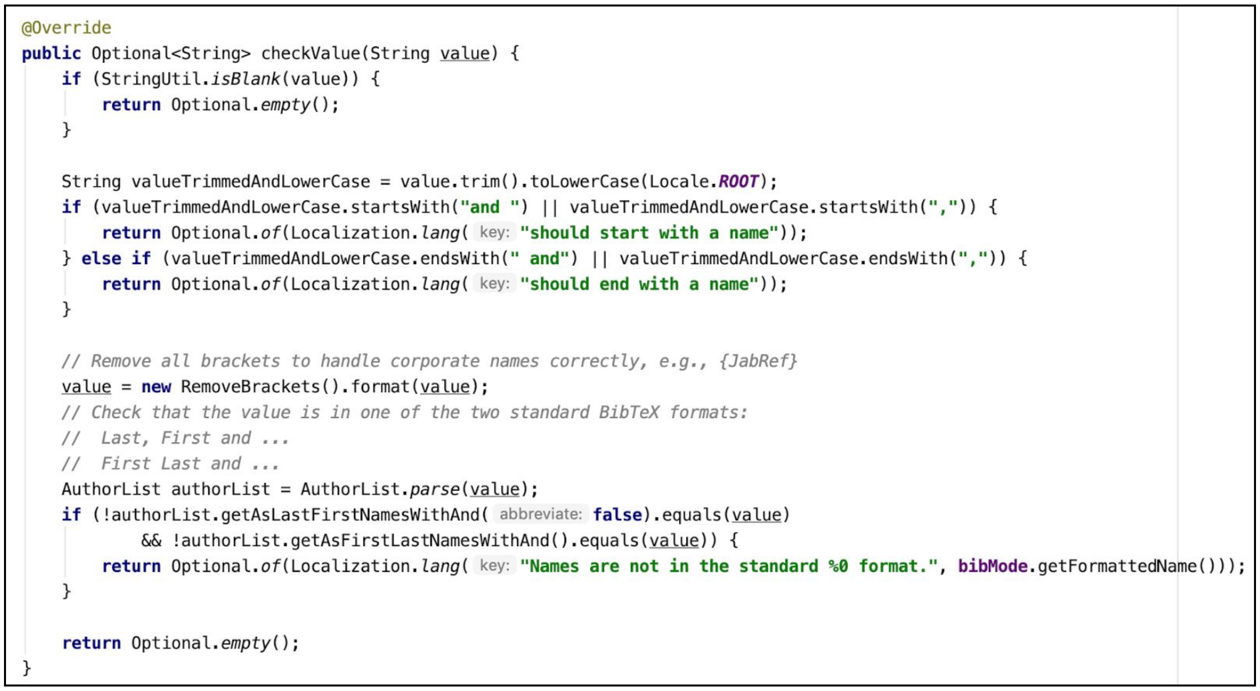
Test Case #2


```java
@Test
public void validFirstnameNameAuthor() throws Exception {
    assertEquals(Optional.empty(), checker.checkValue("Stefan Kolb"));
}

@Test
public void validFirstnameNameAuthors() throws Exception {
    assertEquals(Optional.empty(), checker.checkValue("Stefan Kolb and Simon Harrer"));
}

@Test
public void complainAboutPersonStringWithTwoManyCommas() throws Exception {
    assertEquals(Optional.of("Names are not in the standard BibTeX format."),
            checker.checkValue("Test1, Test2, Test3, Test4, Test5, Test6"));
}

@Test
public void doNotComplainAboutSecondNameInFront() throws Exception {
    assertEquals(Optional.empty(), checker.checkValue("M. J. Gotay"));
}

@Test
public void validCorporateNameInBrackets() throws Exception {
    assertEquals(Optional.empty(), checker.checkValue("{JabRef}"));
}

@Test
public void validCorporateNameAndPerson() throws Exception {
    assertEquals(Optional.empty(), checker.checkValue("{JabRef} and Stefan Kolb"));
    assertEquals(Optional.empty(), checker.checkValue("{JabRef} and Kolb, Stefan"));
}
```

Code Snippet (4) : PersonNamesCheckerTest

One of our changes initiated had to do with the fact that title would throw an error upon having two sentences where the second sentence would start with a capital letter. This led us to looking for test cases in the integrity folder mostly. This made us analyse the PersonNamesCheckerTest class as the author names are crucial entries for a reference management tool and we wanted to look for any similar errors as in the title case.

In Code Snippet (4), we realise a number of interesting test cases to check for the authors names and the key is the checkValue function which is called with the help of checker that is an object of the PersonNamesChecker class. One of the main concepts we were drawn to is the Optional return type used for the checkValue. Optional.empty() was returned with names and a string stating the wrong format was returned else, which is a string. We also noticed “throws Exception” attached to the method, even though no exception can be thrown from these assertions. The above test cases with its entire package **gave us the idea** that the author names can be separated by “and” and their first and last names by commas or space. It also permits a second name to be in the front with dot delimiters. It also suggested from the final test case that the corporate names are given in the braces format.

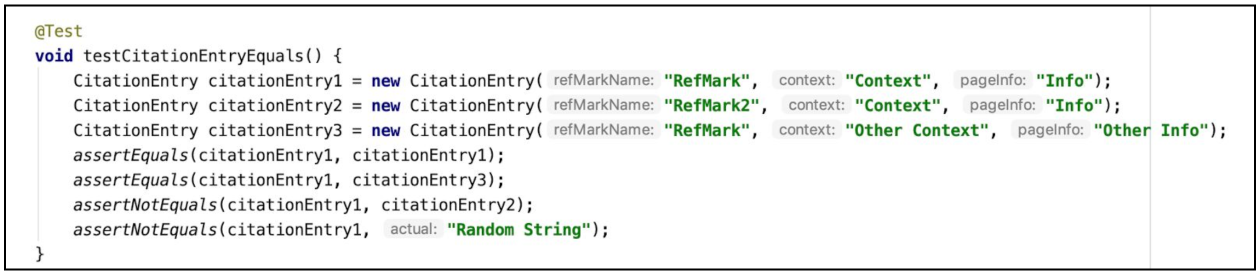


```java
@Override
public Optional<String> checkValue(String value) {
    if (StringUtil.isBlank(value)) {
        return Optional.empty();
    }

    String valueTrimmedAndLowerCase = value.trim().toLowerCase(Locale.ROOT);
    if (valueTrimmedAndLowerCase.startsWith("and ") || valueTrimmedAndLowerCase.startsWith(",")) {
        return Optional.of(Localization.lang( key: "should start with a name"));
    } else if (valueTrimmedAndLowerCase.endsWith(" and") || valueTrimmedAndLowerCase.endsWith(",")) {
        return Optional.of(Localization.lang( key: "should end with a name"));
    }

    // Remove all brackets to handle corporate names correctly, e.g., {JabRef}
    value = new RemoveBrackets().format(value);
    // Check that the value is in one of the two standard BibTeX formats:
    //  Last, First and ...
    //  First Last and ...
    AuthorList authorList = AuthorList.parse(value);
    if (!authorList.getAsLastFirstNamesWithAnd( abbreviate: false).equals(value)
            && !authorList.getAsFirstLastNamesWithAnd().equals(value)) {
        return Optional.of(Localization.lang( key: "Names are not in the standard %0 format.", bibMode.getFormattedName()));
    }

    return Optional.empty();
}
```

Code Snippet (5) : checkValue method (PersonNamesChecker)

To understand the underlying functionalities, we looked into the checkValue method, as shown in Code Snippet (5), where we were fully able to comprehend the Optional return statements. The main function used here is the method parse of the AuthorList class which plays a crucial role in separating the author names by delimiters and the keyword "and" which is not obvious from the test case.

Test Case #3

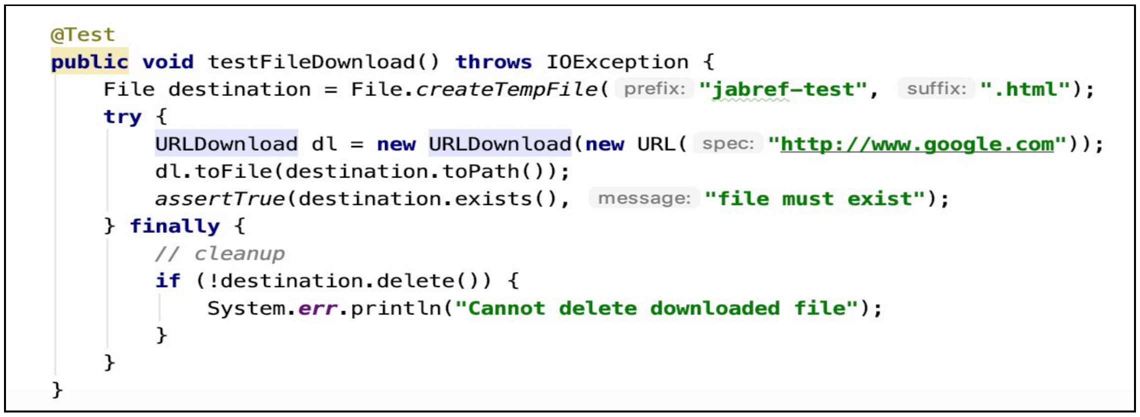


```java
@Test
void testCitationEntryEquals() {
    CitationEntry citationEntry1 = new CitationEntry( refMarkName: "RefMark",  context: "Context",  pageInfo: "Info");
    CitationEntry citationEntry2 = new CitationEntry( refMarkName: "RefMark2",  context: "Context",  pageInfo: "Info");
    CitationEntry citationEntry3 = new CitationEntry( refMarkName: "RefMark",  context: "Other Context",  pageInfo: "Other Info");
    assertEquals(citationEntry1, citationEntry1);
    assertEquals(citationEntry1, citationEntry3);
    assertNotEquals(citationEntry1, citationEntry2);
    assertNotEquals(citationEntry1,  actual: "Random String");
}
```

Code Snippet (6) : testCitationEntryEquals (CitationEntryTest)

One of the first things we noticed was that the word test is used in the name of the method which according to the latest coding style conventions for test cases suggests that we should not use the prefix. This suggests that the test case is one of the earliest written cases before enforcing strict standards. The particular test **suggests** that we can have different context and Info as in citationEntry1 and citationEntry3 but if the refMarkName is the same, these are considered as the same entry for citations. Whereas if we have the same context and Info as in citationEntry1 and citationEntry2 but different refMarkName, it is considered different citation entries. Hence, this again suggests that the essence lies with the refMarkName which is counterintuitive as the context and info can be different.

Test Case #4



```
@Test
public void testFileDownload() throws IOException {
    File destination = File.createTempFile( prefix: "jabref-test",  suffix: ".html");
    try {
        URLDownload dl = new URLDownload(new URL( spec: "http://www.google.com"));
        dl.toFile(destination.toPath());
        assertTrue(destination.exists(),  message: "file must exist");
    } finally {
        // cleanup
        if (!destination.delete()) {
            System.err.println("Cannot delete downloaded file");
        }
    }
}
```

Code Snippet (7) : testFileDownload (URLDownloadTest)

Finally, we were interested in looking for test files concerning downloads from the internet as one of the main functions is to be able to link the reference management tool to online platforms for effective download of files. To serve this purpose, we looked at test files under the net folder.

The URLDownloadTest class has a number of methods to test various functions, like to test the success of download of http, https and ftp. It has tests to check for temporary download and whether the file retains the name of the downloaded link, and checks for the MimeType. Here, we noticed that using URL objects enforces the test cases to handle IOException and hence is necessary to use compared to the previous test cases where it wasn't required. One of the interesting test cases here was the testFileDownload (as shown in Code Snippet(7)), we understood that test cases need not be simple and can have try-catch-finally blocks. This test case creates a destination file with .html suffix. Then this path is used later to download the file using the URLDownload object. We also understood the importance of cleaning up the file, even while managing test cases.