Testing JabRef

In JabRef, we mainly rely on basic JUnit unit tests to increase code coverage.

General hints on tests

Imagine you want to test the method `format(String value)` in the class `BracesFormatter` which removes double braces in a given string.

- *Placing:* all tests should be placed in a class named `classTest`, e.g. `BracesFormatterTest`.
- *Naming:* the name should be descriptive enough to describe the whole test. Use the format `methodUnderTest_ expectedBehavior_context` (without the dashes). So for example `formatRemovesDoubleBracesAtBeginning`. Try to avoid naming the tests with a `test` prefix since this information is already contained in the class name. Moreover, starting the name with `test` leads often to inferior test names (see also the Stackoverflow discussion about naming).
- *Test only one thing per test:* tests should be short and test only one small part of the method. So instead of

  ```
  void format() {
      assertEqual("test", format("test"));
      assertEqual("{test", format("{test"));
      assertEqual("test", format("test}}"));
  }
  ```

  we would have five tests containing a single `assert` statement and named accordingly (`formatDoesNotChangeStringWithoutBraces`, `formatDoesNotRemoveSingleBrace`, , etc.). See JUnit AntiPattern for background.
- Do *not just test happy paths*, but also wrong/weird input.
- It is recommended to write tests *before* you actually implement the functionality (test driven development).
- *Bug fixing:* write a test case covering the bug and then fix it, leaving the test as a security that the bug will never reappear.
- Do not catch exceptions in tests, instead use the `assertThrows(Exception.class, () -> doSomethingThrowsEx())` feature of junit-jupiter to the test method.

Coverage

IntelliJ has build in test coverage reports. Choose “Run with coverage”.

For a full coverage report as HTML, execute the gradle task `jacocoTestReport` (available in the "verification" folder in IntelliJ). Then, you will find <build/reports/jacoco/test/html/index.html> which shows the coverage of the tests.

Lists in tests

Instead of

```
assertTrue(actualList.isEmpty());
```

use

```
assertEquals(List.of(), actualList);
```

Similarly, to compare lists, instead of following code:

```
assertEquals(2, actualList.size());
assertEquals("a", actualList.get(0));
assertEquals("b", actualList.get(1));
```

use the following code:

```
assertEquals(List.of("a", "b"), actualList);
```

BibEntries in tests

- Use the `assertEquals` methods in `BibtexEntryAssert` to check that the correct BibEntry is returned.

Files and folders in tests

If you need a temporary file in tests, use the `@TempDir` annotation:

```
class TestClass{


  @Test
  void deletionWorks(@TempDir Path tempDir) {
  }
}
```

to the test class. A temporary file is now created by `Files.createFile(path)`. Using this pattern automatically ensures that the test folder is deleted after the tests are run. See https://www.geeksforgeeks.org/junit-5-tempdir/ for more details.

Loading Files from Resources

Sometimes it is necessary to load a specific resource or to access the resource directory

```
Path resourceDir = Paths.get(MSBibExportFormatTestFiles.class.getResource("MsBibExportFormatTest1.bib").
```

When the directory is needed, it is important to first point to an actual existing file. Otherwise the wrong directory will be returned.

Preferences in tests

If you modify preference, use following pattern to ensure that the stored preferences of a developer are not affected:

Or even better, try to mock the preferences and insert them via dependency injection.

```
@Test
public void getTypeReturnsBibLatexArticleInBibLatexMode() {
     // Mock preferences
     PreferencesService mockedPrefs = mock(PreferencesService.class);
     GeneralPreferences mockedGeneralPrefs = mock(GeneralPReferences.class);
     // Switch to BibLatex mode
     when(mockedPrefs.getGeneralPrefs()).thenReturn(mockedGeneralPrefs);
     when(mockedGeneralPrefs.getDefaultBibDatabaseMode())
        .thenReturn(BibDatabaseMode.BIBLATEX);

     // Now test
     EntryTypes biblatexentrytypes = new EntryTypes(mockedPrefs);
     assertEquals(BibLatexEntryTypes.ARTICLE, biblatexentrytypes.getType("article"));
}
```

To test that a preferences migration works successfully, use the mockito method `verify`. See `PreferencesMigrationsTest` for an example.

Database tests

PostgreSQL

To quickly host a local PostgreSQL database, execute following statement:

```
docker run -d -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=postgres -e POSTGRES_DB=postgres -p 5432:54
```

Set the environment variable `DBMS` to `postgres` (or leave it unset)

Then, all DBMS Tests (annotated with `@org.jabref.testutils.category.DatabaseTest`) run properly.

MySQL

A MySQL DBMS can be started using following command:

```
docker run -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=jabref -p 3800:3307 mysql:8.0 --port=3307
```

Set the environment variable `DBMS` to `mysql`.

Fetchers in tests

Fetcher tests can be run locally by executing the Gradle task `fetcherTest`. This can be done by running the following command in the command line:

```
./gradlew fetcherTest
```

Alternatively, if one is using IntelliJ, this can also be done by double-clicking the `fetcherTest` task under the `other` group in the Gradle Tool window (`JabRef > Tasks > other > fetcherTest`).

“No matching tests found”

In case the output is “No matching tests found”, the wrong test category is used.

Check “Run/Debug Configurations”

Example

```
:databaseTest --tests "org.jabref.logic.importer.fileformat.pdf.PdfMergeMetadataImporterTest.pdfMetadata
```

This tells Gradle that `PdfMergeMetadataImporterTest` should be executed as database test. However, it is marked as `@FetcherTest`. Thus, change `:databaseTest` to `:fetcherTest` to get the test running.

Advanced testing and further reading

On top of basic unit testing, there are more ways to test a software:

Type	Techniques	Tool (Java)	Kind of tests	Used In JabRef
Functional	Dynamics, black box, positive and negative	JUnit-QuickCheck	Random data generation	No, not intended, because other test kinds seem more helpful.
Functional	Dynamics, black box, positive and negative	GraphWalker	Model-based	No, because the BibDatabase doesn’t need to be tests

Type	Techniques	Tool (Java)	Kind of tests	Used In JabRef
Functional	Dynamics, black box, positive and negative	TestFX	GUI Tests	Yes
Functional	Dynamics, black box, negative	Lincheck	Testing concurrent algorithms	No
Functional	Dynamics, white box, negative	PIT	Mutation	No
Functional	Dynamics, white box, positive and negative	Mockito	Mocking	Yes
Non-functional	Dynamics, black box, positive and negative	JETM, Apache JMeter	Performance (performance testing vs load testing respectively)	No
Structural	Static, white box	CheckStyle	Constient formatting of the source code	Yes
Structural	Dynamics, white box	SpotBugs	Reocurreing bugs (based on experience of other projects)	No