SWE_261P
Project_Part_1: Introduction. Set Up. Functional Testing and Partitioning.
Team Members:
- Yuxin Huang: yuxinh20@uci.edu
- Changhao Liu: liuc50@uci.edu
- Ruokun Xu: ruokunx@uci.edu
Team Name: OffersAreHere
Github link: https://github.com/yx-hh/commons-io

# Apache-commons-io

## Introduction:

Apache Commons IO is the component of the Apache Commons which is derived from Java API and covers extensive use cases for common operations of file input and output. Apache Commons IO Library implements various classes, such as utility classes which provide file and string comparison, filter classes which provide a way to filter files based on logical criteria rather than string-based comparisons, and file monitor classes which provide ways to track changes in a target file or folder and allow actions to be taken on the changes(Apache Commons IO, tutorialspoint.com).

**Additional Info:**

- Language: Java
- Lines of Code (git ls-files | xargs cat | wc -l): 92774
- Number of classes (git ls-files | wc -l): 211
- Javadoc: https://commons.apache.org/proper/commons-io/apidocs/

## Building and Running instruction

- Prerequisite: maven and java environment required

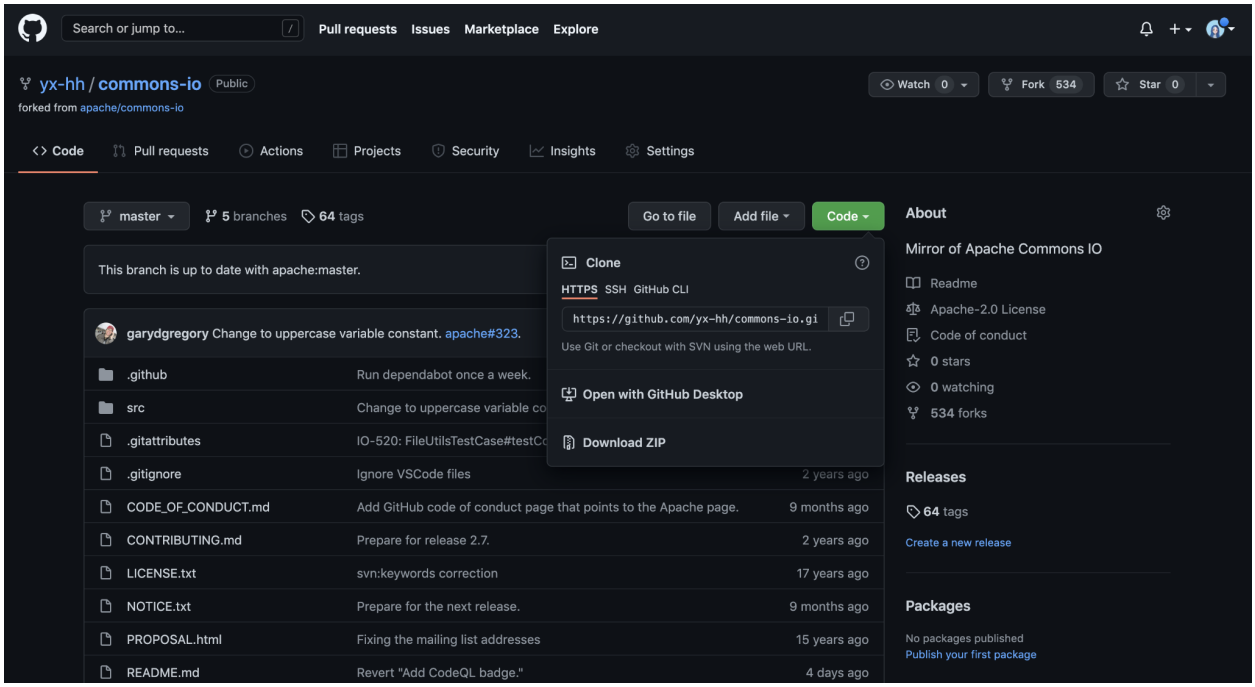
### Building Instruction & Package Instruction

- Use maven dependency: copy and paste the following dependency into pom.xml in your projects and recompile.

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
    <groupId>commons-io</groupId>
```

```
    <artifactId>commons-io</artifactId>
    <version>2.11.0</version>
</dependency>
```

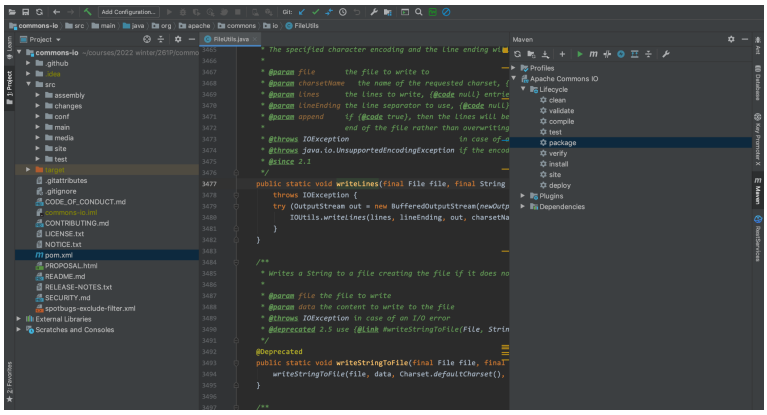
- Use jar: Download the code from GitHub repo: GitHub - yx-hh/commons-io: Mirror of Apache Commons IO




1. Open with IntelliJ Idea and set as a MAVEN project
2. Use **GUI** on **IntelliJ** or the command line in the console to compile and package the project as a jar file.

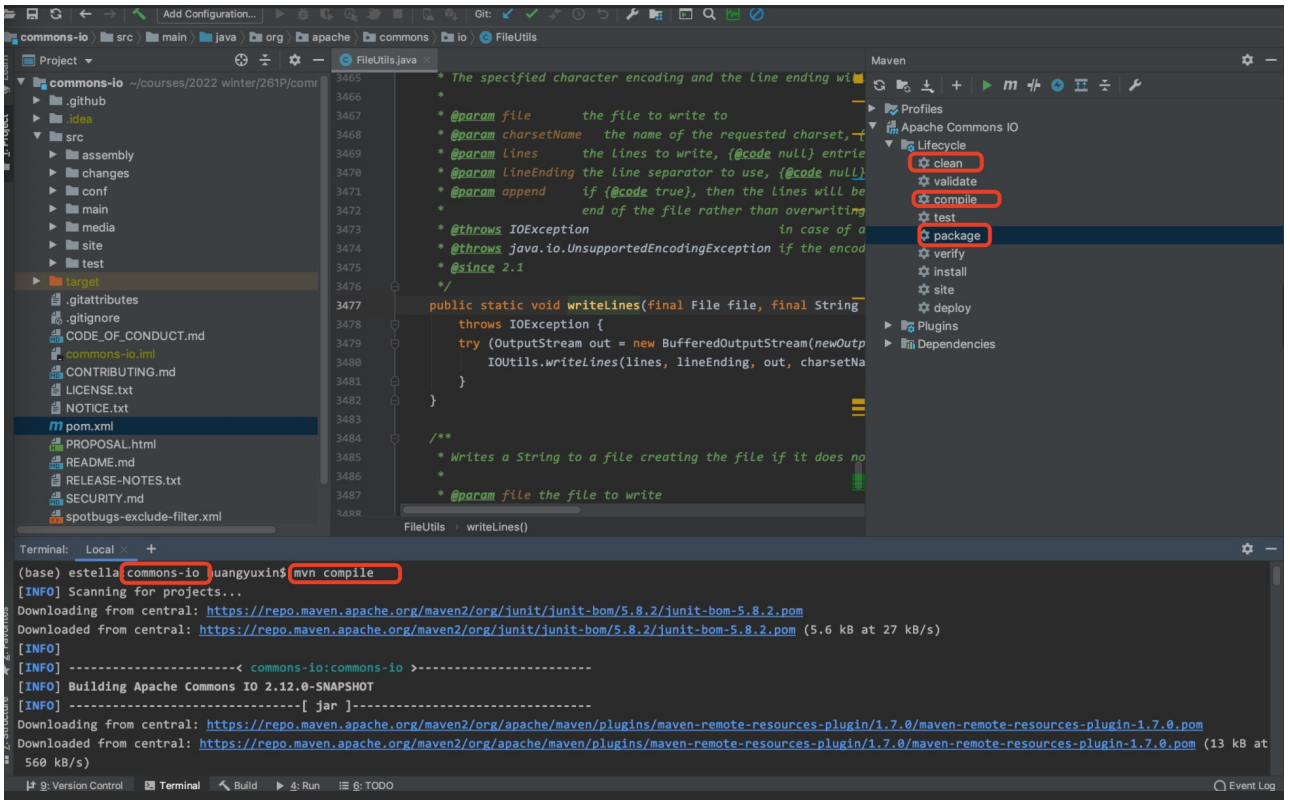
   **GUI**:

   Find the maven table on the right side and click clean, compile and package in order. The picture instruction is as follows.

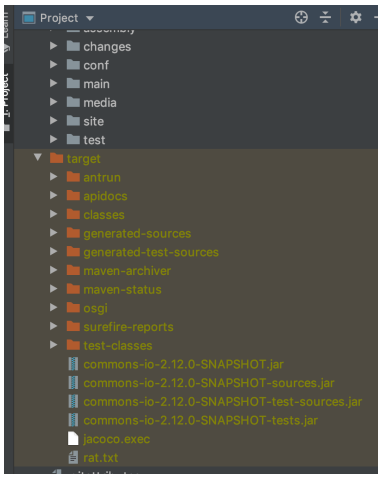


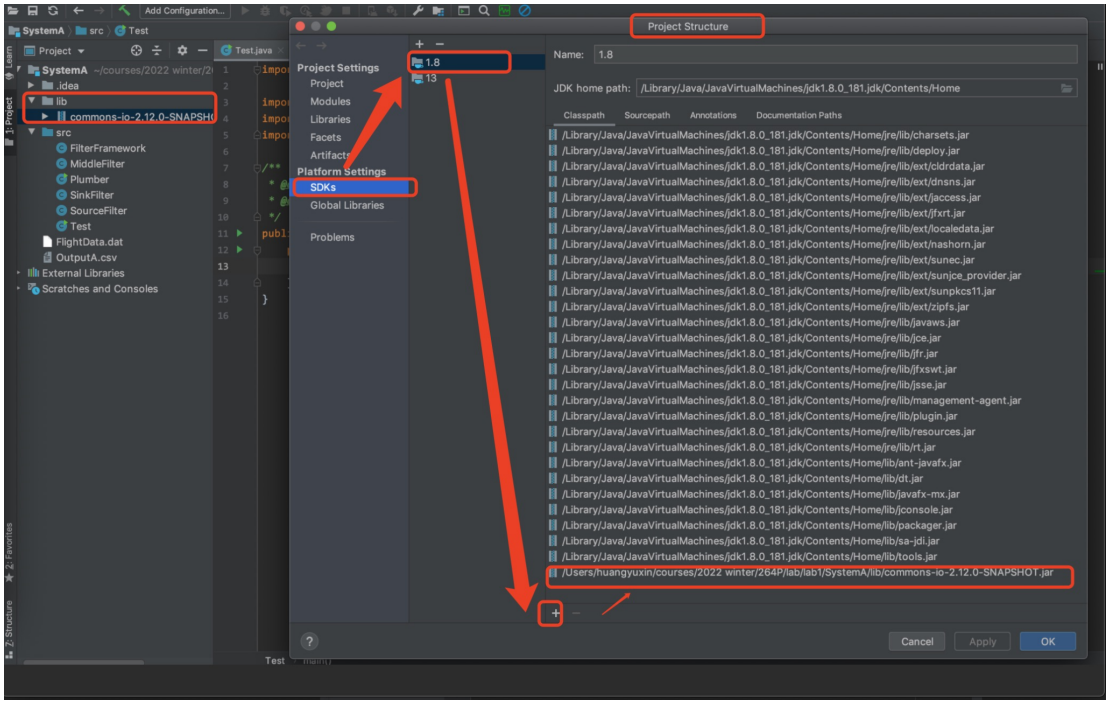
**Command line**:

Under the folder of commons-io, use command line `mvn clear`, `mvn compile`, `mvn package`. In your computer, it may be `maven clean`, `maven compile`, `maven package`, it depends on your maven setting.



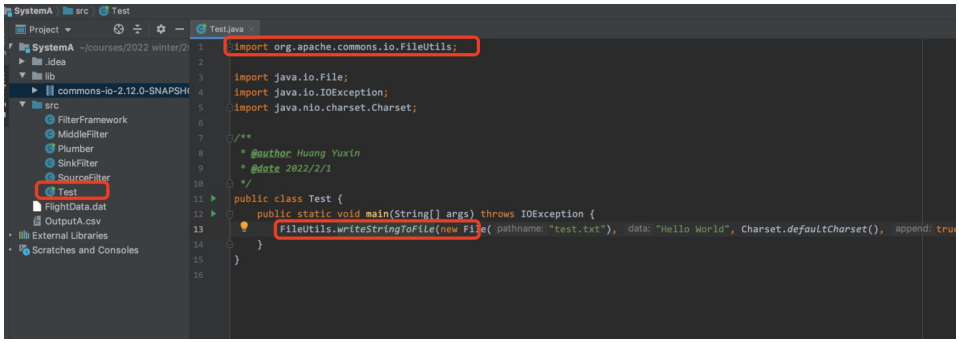

3. Find the jar file under the folder of the target, copy the commons-io-2.12.0-SNAPSHOT.jar to a normal java project, create a lib folder and import this file to java lib. This jar file is an executable file that contains the source code.






## Running instruction

- Use API from commons-io like the following, and static methods can be used directly.

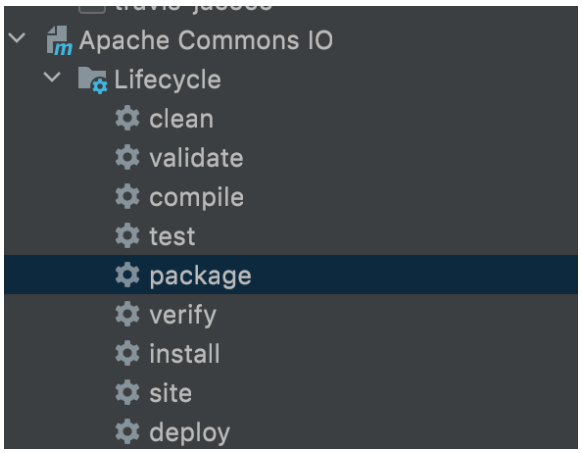



## Existing Test Cases:

Commons IO utilizes a variety of testing techniques that we learned during class, including smoke testing, black box testing, partition, etc. Those techniques play an essential role in helping developers and clients to maintain a great coverage of the codebase and to detect bugs at an early stage.

**Smoke Testing**:

Commons IO has utilized smoke testing, which automatically runs after each “build” to make sure all the main features work well. To understand the role of smoking testing, we should first learn what “build” is: build is to take all the source codes, compile them into machine-readable codes and convert them into software artifacts. Smoking testing can verify the build and check whether the essential features are working. If one of the test cases breaks, it will reject the build and stop running all the further tests to save some time and bandwidth. As you can see from the example below, when we start the build, Commons IO begins to run some smoke tests such as FileUtilsTest to confirm that the main functionalities are working. If any test cases fail, it will reject the build and stop running the rest of the test cases.




```
Downloading from central: https://repo.maven.apache.org/maven2/org/assertj/assertj-parent-pom/2.1.9/assertj-parent-pom-2.1.9.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/assertj/assertj-parent-pom/2.1.9/assertj-parent-pom-2.1.9.pom (15 kB at
  kB/s)
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running org.apache.commons.io.FileUtilsTest
[WARNING] Tests run: 159, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 2.939 s - in org.apache.commons.io.FileUtilsTest
[INFO] Running org.apache.commons.io.FileUtilsDirectoryContainsTest
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.221 s - in org.apache.commons.io.FileUtilsDirectoryContainsTest
```

**Black Box Testing**:

Common IO also utilizes black box testing. According to IEEE, black box testing is “testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions”. In short, testers do not understand the internal system but write test cases to validate some functionalities. Therefore, black box testing mainly focuses on the inputs and expected outputs.

Here is one of the examples: In this screenshot, this specific test case 1. creates a new file 2. writes a string in the file 3. checks the input stream’s first character is ‘H’.

```java
@Test
public void test_openInputStream_exists() throws Exception {
    final File file = new File(tempDirFile,  child: "test.txt");
    TestUtils.createLineBasedFile(file, new String[]{"Hello"});
    try (FileInputStream in = FileUtils.openInputStream(file)) {
        assertEquals( expected: 'H', in.read());
    }
}
```

**Boundary Checking**:

Boundary checking is quite essential in checking whether the program is able to run successfully in some corner cases such as inputs being null, negative numbers,  zero, or positive numbers. Commons IO utilizes this technique to check valid inputs as well as invalid inputs, looking for some potential bugs that may cause the program to break. For example, here are two cases to test the boundary values for openOutputStream. The first test case checks when the file exists and is valid, while the second one tests when the file is a directory, which is an invalid input. Both two tests use boundary values for detecting some potential bugs.

```java
@Test
public void test_openOutputStream_exists() throws Exception {
    final File file = new File(tempDirFile,  child: "test.txt");
    TestUtils.createLineBasedFile(file, new String[]{"Hello"});
    try (FileOutputStream out = FileUtils.openOutputStream(file)) {
        out.write( b: 0);
    }
    assertTrue(file.exists());
}

@Test
public void test_openOutputStream_existsButIsDirectory() {
    final File directory = new File(tempDirFile,  child: "subdir");
    directory.mkdirs();
    assertThrows(IllegalArgumentException.class, () -> FileUtils.openOutputStream(directory));
}
```

**Testing Framework and how to run tests**:

The testing framework of Commons IO is JUnit, which is a popular framework for Java developers. There are multiple ways to run JUnit test cases for the entire suite, a single class, or a single test case.

- To run all the test cases, you can execute “mvn clean test” in the terminal.

```
changhao@changhaoliusmbp commons-io % mvn clean test
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< commons-io:commons-io >-----------------------
[INFO] Building Apache Commons IO 2.12.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ commons-io ---
[INFO] Deleting /Users/changhao/Desktop/style-winter/commons-io/target
[INFO]
```

- To run test cases for a specific class, use “mvn clean test -Dtest=xxxxTest” such as “mvn clean test -Dtest=FileSystemTest”

```
changhao@changhaoliusmbp commons-io % mvn clean test -Dtest=FileSystemTest
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< commons-io:commons-io >-----------------------
[INFO] Building Apache Commons IO 2.12.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ commons-io ---
[INFO] Deleting /Users/changhao/Desktop/style-winter/commons-io/target
```

- To run one single test case, execute “`mvn clean test -Dtest=xxxxTest#testA`” such as “`mvn clean test -Dtest=FileSystemTest#testSorted`”

```
changhao@changhaoliusmbp commons-io % mvn clean test -Dtest=FileUtilsTest#test_openInputStream_notExists
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< commons-io:commons-io >-----------------------
[INFO] Building Apache Commons IO 2.12.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ commons-io ---
[INFO] Deleting /Users/changhao/Desktop/style-winter/commons-io/target
[INFO]
[INFO] --- maven-enforcer-plugin:3.0.0:enforce (enforce-maven-version) @ commons-io ---
[INFO]
[INFO] --- maven-enforcer-plugin:3.0.0:enforce (enforce-maven-3) @ commons-io ---
[INFO]
[INFO] --- maven-enforcer-plugin:3.0.0:enforce (enforce-maven) @ commons-io ---
[INFO]
[INFO] --- apache-rat-plugin:0.13:check (rat-check) @ commons-io ---
[INFO] Enabled default license matchers.
[INFO] Will parse SCM ignores for exclusions...
[INFO] Parsing exclusions from /Users/changhao/Desktop/style-winter/commons-io/.gitignore
[INFO] Finished adding exclusions from SCM ignore files.
```

- If you prefer to use GUI, you can simply click the green arrow of a test class or a single test case:



# Partitioning

## Systematic functional testing

Systematic functional testing is a type of black-box testing. When testing the system, QA will test the software as a whole, and the software system will be divided into different functionalities. Importantly, QA developers write test cases to those functions individually without looking into the source code. The test functions under the specifications and sets input data and expected output. After testing all the functions as a system, testers will validate the system's complete and integrate functions together to make sure the system runs smoothly.

### Systematic functional testing example

An automated company manufactures a computer system that allows users to get access to a variety of car features such as video player, GPS, voice control, and climate control. Without an understanding of how the inner system was built, testers would write test cases to all of these features individually without checking the system internally, and “they must also test them as a complete system to ensure interoperability and a good user experience.” [15 Functional Testing Types with Examples - Applause](https://www.applause.com/blog/functional-testing-types-examples)

### Systematic Functional testing typically involves seven steps

1. Divide the software system into different functions
2. Identify the expected behaviors of each function
3. Create input data based on each function's specifications
4. Confirm the expected and valid output based on the function's specifications
5. Run the test cases with those inputs
6. Compare the expected output and actual output to ensure that they are equal
7. Finish running all the test cases to ensure that the entire system works as expected

[Functional Testing - Wikipedia]

## Partitioning Test

Partitioning test is a type of testing that separates input values into a variety of cases that cover all situations, reducing the number of test cases and decreasing testing time. This method is quite essential in defining test cases to cover all the potential errors and to reduce the time of writing test cases, as there is no need to write every combination thanks to this technique. [cite from [Equivalence partitioning - Wikipedia]]

Partitioning Test Example

Here is an example of an online shopping site that utilizes partition tests. With the unique id and name of each product, we are able to acquire the specific product either by using the product's name or ID.

| Product | ID |
|---|---|
| Belt | 111 |
| Glove | 121 |
| Monitor | 88 |
| Mouse | 30 |
| Butter | 20 |
| Cable | 1 |

If a user enters invalid product ID, the application will show an error page. If the user enters a valid product ID. For example, enter 111 for Belt, the application will show a valid result. [example cited from [Equivalence Partitioning Method - GeeksforGeeks]]

Partition Test typically involves four steps

1. divide test data into equivalence partitions according to the specifications
2. Select representatives
3. create test specifications
4. Produce and execute actual tests

## Commons-IO partitioning test case

In Commons-IO project, we have FileUtils, IOUtils, CopyUtils, etc features to deal with all kinds of problems related to IO Operation. In this part, we chose FileUtils feature for the partitioning test. For partition boundaries, we can use a variety of file size, charset types, and timestamps.

For example, in FileUtils class, we have a method named: public static boolean isFileOlder(final File file, final long timeMillis). In its specification noted the time reference measured in milliseconds since the epoch (00:00:00 GMT, January 1, 1970). When we have input such as timeMillis is null or input file is invalid which does not match the partition principle, we will get error feedback. Test cases related to this example as following:

## New test cases in JUnit

In IntelliJ choose method and right click run test case

### Test whether refFile's last modified time is older than the input time

```
   @Test
   public void testIsFileOlderTrue() throws Exception {
      final File refFile = TestUtils.newFile(tempDirFile, "FileUtils-reference.txt");
      TestUtils.generateTestData(refFile, 1);
      long time = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2100-01-01
01:01:01").getTime();
      assertTrue(FileUtils.isFileOlder(refFile, time), "Old File - Older - File");
   }
```

Test whether refFile's last modified time is newer than the input time

```
@Test
public void testIsFileOlderFalse() throws Exception {
    final File refFile = TestUtils.newFile(tempDirFile, "FileUtils-reference.txt");
    TestUtils.generateTestData(refFile, 1);
    long time = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2001-01-01
01:01:01").getTime();
    assertFalse(FileUtils.isFileOlder(refFile, time), "Old File - older - File");
}
```

Test NullPointerException Throw when refFile is invalid

```
@Test
public void testIsFileOlderFileInvalid() throws Exception {
    assertThrows(NullPointerException.class, () -> FileUtils.isFileOlder(null, Instant.now()));
}
```

Test IllegalArgumentException when time data is invalid

```
@Test
public void testIsFileOlderFileTimeBoundaryInvalid() throws Exception {
    final File refFile = TestUtils.newFile(tempDirFile, "FileUtils-reference.txt");
    TestUtils.generateTestData(refFile, 1);
    final File invalidFile = TestUtils.newFile(tempDirFile, "FileUtils-invalid.txt");
    // Invalid reference File
    assertThrows(IllegalArgumentException.class, () -> FileUtils.isFileNewer(refFile,
invalidFile));
}
```

# Reference


Apache Commons IO - Overview. (n.d.). Retrieved February 2, 2022, from https://www.tutorialspoint.com/commons_io/commons_io_overview.htm

15 Functional Testing Types with Examples - Applause, from https://www.applause.com/blog/functional-testing-types-examples

Equivalence Partitioning Method - GeeksforGeeks, from https://www.geeksforgeeks.org/equivalence-partitioning-method/

Equivalence partitioning - Wikipedia, from
https://en.wikipedia.org/wiki/Equivalence_partitioning

Functional Testing - Wikipedia, from
https://en.wikipedia.org/wiki/Functional_testing

Also cite all the slides from SWE_261P