Introduction

What is the tool that you are testing?

We are testing the tool called jsoup. Our Github link is https://github.com/duke326/SWE261.

What is its purpose?

jsoup is a Java library for working with real-world HTML. It provides a very convenient API for fetching URLs and extracting and manipulating data, using the best of HTML5 DOM methods and CSS selectors.

jsoup implements the WHATWG HTML5 specification, and parses HTML to the same DOM as modern browsers do.

- scrape and parse HTML from a URL, file, or string
- find and extract data, using DOM traversal or CSS selectors
- manipulate the HTML elements, attributes, and text
- clean user-submitted content against a safe-list, to prevent XSS attacks
- output tidy HTML

jsoup is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; jsoup will create a sensible parse tree.

See **jsoup.org** for downloads and the full API documentation.

Any other aspects that are relevant

Lines of code: 32887

In ubuntu system, I used this command to calculate the number of code except the comments.

```
find . -type f -name '*.java' | xargs cat | wc -l
```

Number of java files: 132

In ubuntu system, I used this command to calculate the number of java files.

```
find . -name '*.java' | wc -l
```

Language: 100% of Java

Set Up

Fork & Add

Fork your project into GitHub. Add all your team members to the forked project, add Prof. Jones and TA Maruf as collaborators.

Our link is here: https://github.com/duke326/SWE261.

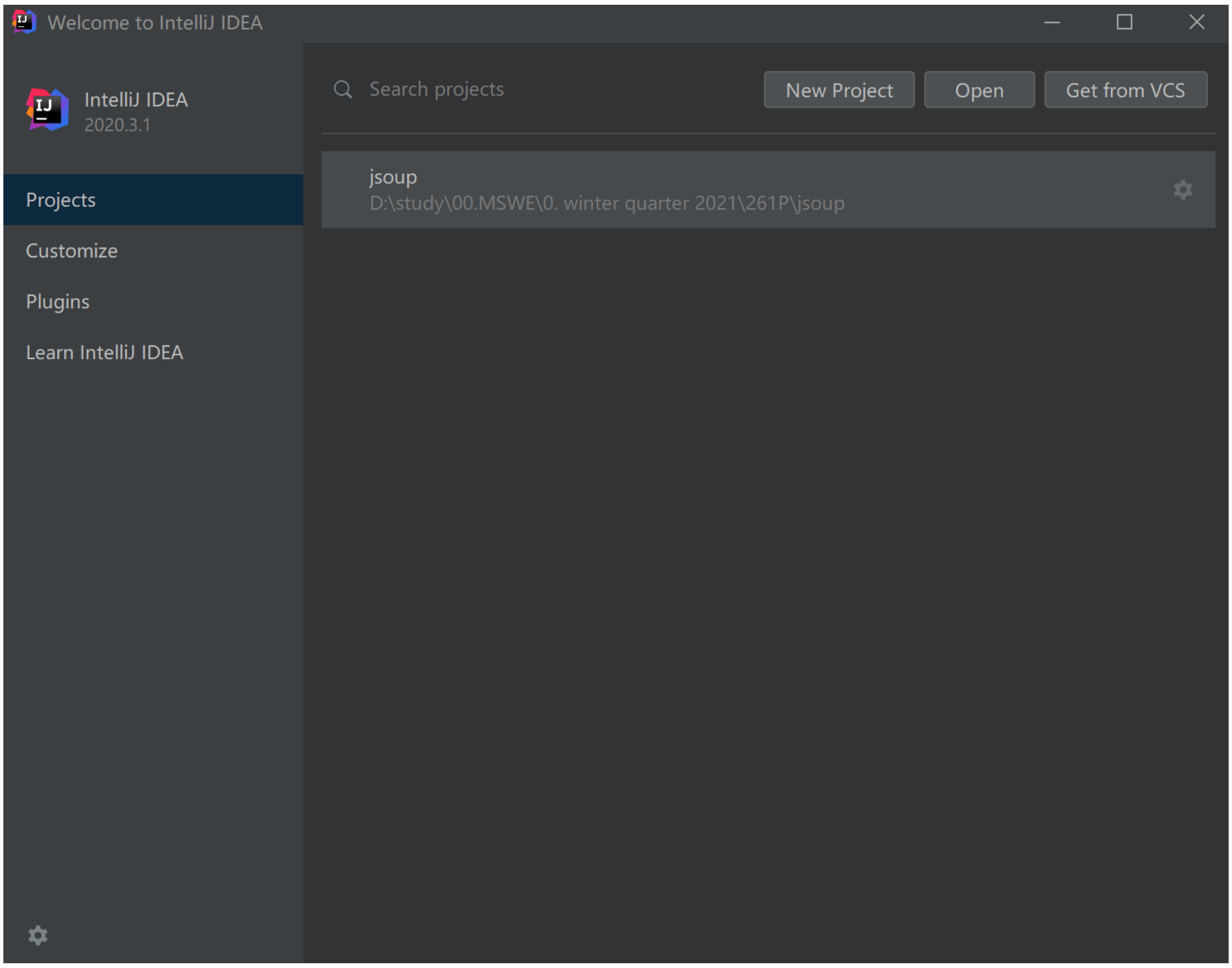
Build

Document its build. What did you need to do to get it built and running?

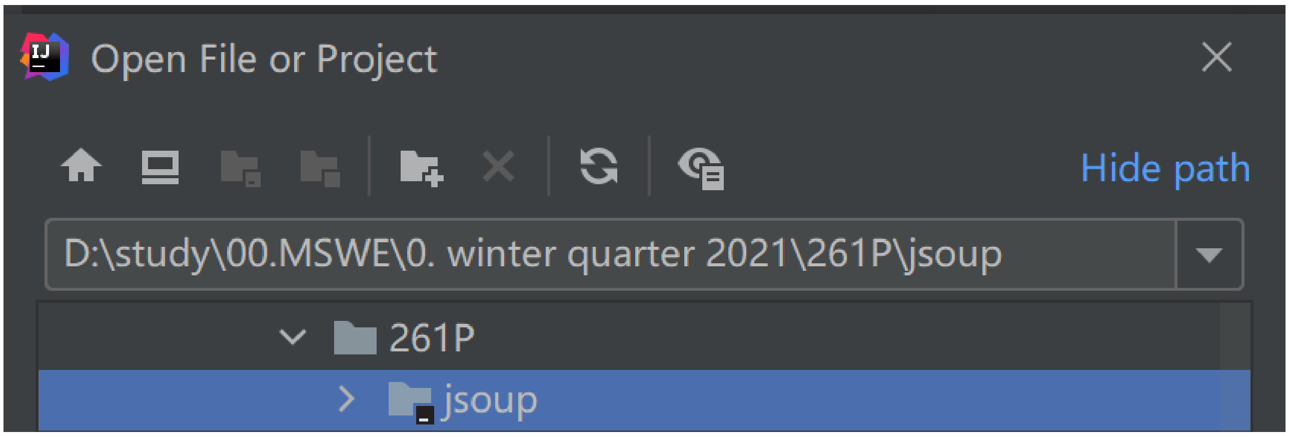
This project used Maven as the build infrastructure for Java projects. Maven uses conventions and patterns to provide a uniform build system. All Maven projects use a shared set of plugins that are retrieved from the Maven repository and Maven executes a defined series of tasks as part of the lifecycle when building the project.

Using IntellJ to open Maven project

(Sample Environment: Windows 10)

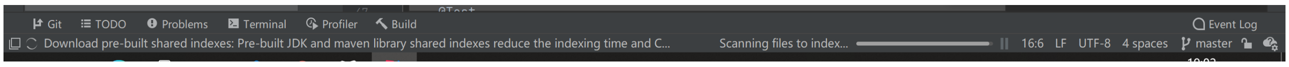


jsoup
D:\study\00.MSWE\0. winter quarter 2021\261P\jsoup



D:\study\00.MSWE\0. winter quarter 2021\261P\jsoup

Then your idea will automatically download the dependency.



Download pre-built shared indexes: Pre-built JDK and maven library shared indexes reduce the indexing time and C...

Scanning files to index... 16:6 LF UTF-8 4 spaces master

How to run test cases?

When the progress bar is full, you can run any test case in the test folder.

Setup for using JUnit 5

Usage of JUnit 5 with Maven

To use Maven you have to use updated version for your build plug-ins and add several dependencies.

The following example shows how to use JUnit 5 with Maven.

```
    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <!--1 -->
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-failsafe-plugin</artifactId>
                <version>2.22.2</version>
            </plugin>
        </plugins>
    </build>

    <!--2 -->
    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.6.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.7.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

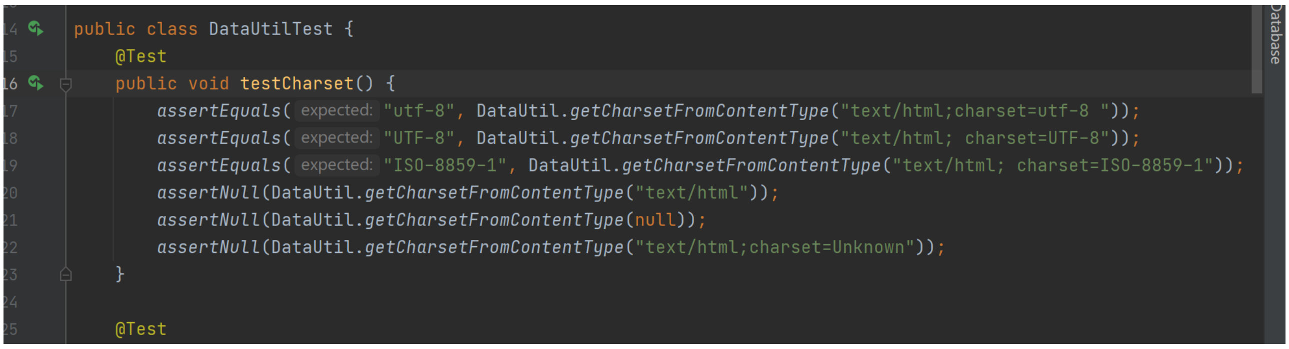
To make it easier for your to apply this, a full pom is displayed, in your project the groupId, artifactId and version would be different.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.vogella</groupId>
    <artifactId>com.vogella.junit5</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <!--1 -->
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.22.2</version>
            </plugin>
            <plugin>
                <artifactId>maven-failsafe-plugin</artifactId>
                <version>2.22.2</version>
            </plugin>
        </plugins>
    </build>

    <!--2 -->
    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.6.0</version>
```

```
                <scope>test</scope>
            </dependency>
            <dependency>
                <groupId>org.junit.jupiter</groupId>
                <artifactId>junit-jupiter-engine</artifactId>
                <version>5.7.0</version>
                <scope>test</scope>
            </dependency>
        </dependencies>
</project>
```

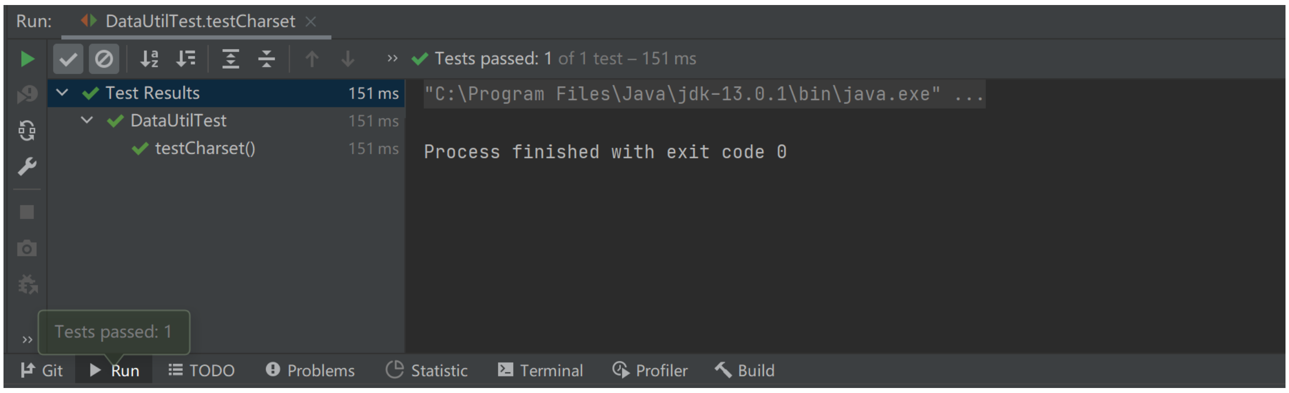
Sample to run

For example, in `test/java/ org.json/helper/DataUtilTest.java`



```
public class DataUtilTest {
    @Test
    public void testCharset() {
        assertEquals( expected: "utf-8", DataUtil.getCharsetFromContentType("text/html;charset=utf-8 "));
        assertEquals( expected: "UTF-8", DataUtil.getCharsetFromContentType("text/html; charset=UTF-8"));
        assertEquals( expected: "ISO-8859-1", DataUtil.getCharsetFromContentType("text/html; charset=ISO-8859-1"));
        assertNull(DataUtil.getCharsetFromContentType("text/html"));
        assertNull(DataUtil.getCharsetFromContentType(null));
        assertNull(DataUtil.getCharsetFromContentType("text/html;charset=Unknown"));
    }

    @Test
```

Press the little green run button, the output would be like this:



```
Run: DataUtilTest.testCharset
Tests passed: 1 of 1 test – 151 ms

Test Results        151 ms    "C:\Program Files\Java\jdk-13.0.1\bin\java.exe" ...
  DataUtilTest      151 ms
    testCharset()   151 ms    Process finished with exit code 0

Tests passed: 1
```

Test cases

Document the existing test cases (JUnit or otherwise). This should be a study of the existing testing practices and frameworks that are used already in the system. (This section might evolve as we learn more throughout the quarter.) How do you run them?

Firstly, the existing test case is **black box testing**. In IEEE, black box testing means that one kind of 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.

Black box testing is also known as functional testing. Black box testing derive sets of inputs that will fully exercise all of the functional requirements of a system.

In our project, it used **JUnit**. *JUnit 5* is the next generation of JUnit. The goal is to create an up-to-date foundation for developer-side testing on the JVM. This includes focusing on Java 8 and above, as well as enabling many different styles of testing.

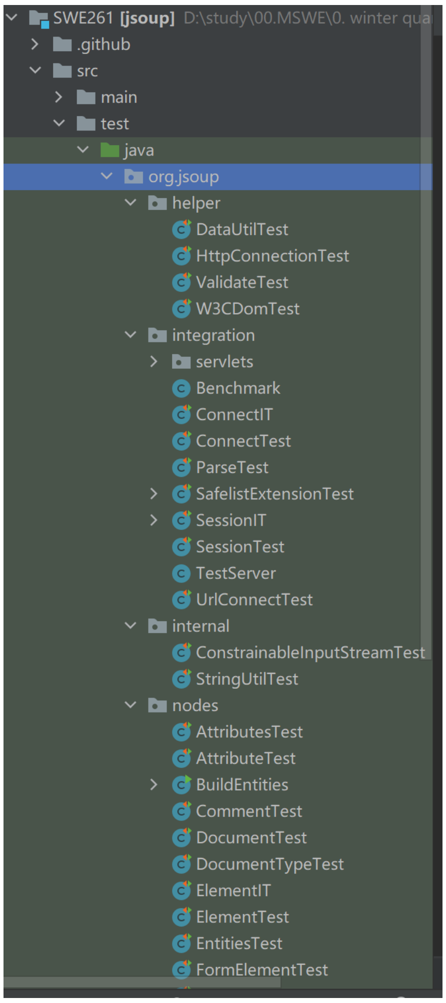
Introduction into JUnit

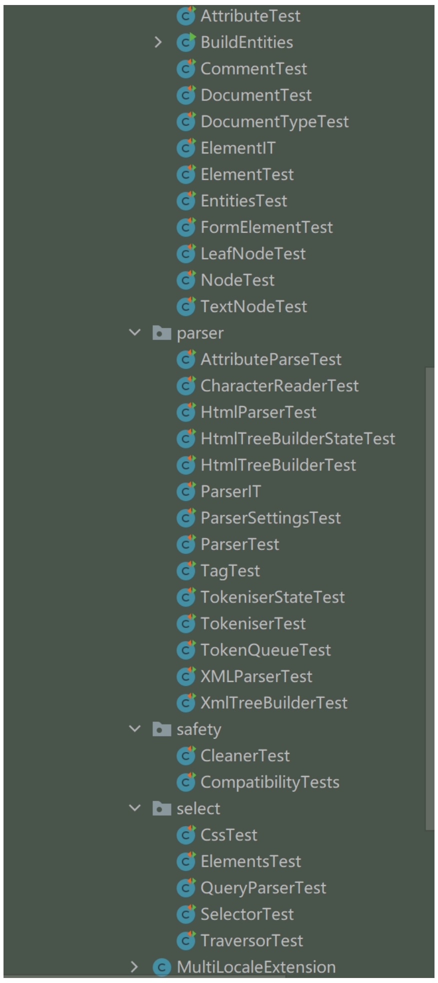
A JUnit *test* is a method contained in a class which is only used for testing. This is called a *Test class*. To define that a certain method is a test method, annotate it with the `@Test` annotation.

This method executes the code under test. You use an *assert* method, provided by JUnit or another assert framework, to check an expected result versus the actual result. These calls are typically called *asserts* or *assert statements*.

Assert statements typically allow to define messages which are shown if the test fails. You should provide here meaningful messages to make it easier for the user to identify and fix the problem. This is especially true if someone looks at the problem, who did not write the code under test or the test code.

Existing test cases


```
SWE261 [jsoup] D:\study\00.MSWE\0. winter qua
  .github
  src
    main
    test
      java
        org.jsoup
          helper
            DataUtilTest
            HttpConnectionTest
            ValidateTest
            W3CDomTest
          integration
            servlets
            Benchmark
            ConnectIT
            ConnectTest
            ParseTest
            SafelistExtensionTest
            SessionIT
            SessionTest
            TestServer
            UrlConnectTest
          internal
            ConstrainableInputStreamTest
            StringUtilTest
          nodes
            AttributesTest
            AttributeTest
            BuildEntities
            CommentTest
            DocumentTest
            DocumentTypeTest
            ElementIT
            ElementTest
            EntitiesTest
            FormElementTest
```


```
            AttributeTest
            BuildEntities
            CommentTest
            DocumentTest
            DocumentTypeTest
            ElementIT
            ElementTest
            EntitiesTest
            FormElementTest
            LeafNodeTest
            NodeTest
            TextNodeTest
          parser
            AttributeParseTest
            CharacterReaderTest
            HtmlParserTest
            HtmlTreeBuilderStateTest
            HtmlTreeBuilderTest
            ParserIT
            ParserSettingsTest
            ParserTest
            TagTest
            TokeniserStateTest
            TokeniserTest
            TokenQueueTest
            XMLParserTest
            XmlTreeBuilderTest
          safety
            CleanerTest
            CompatibilityTests
          select
            CssTest
            ElementsTest
            QueryParserTest
            SelectorTest
            TraversorTest
          MultiLocaleExtension
```

Sample test case

In `org/jsoup/parser/parsesSimpleDocument.java`

```
    @Test
    public void parsesSimpleDocument() {
        String html = "<html><head><title>First!</title></head>
<body><p>First post! <img src=\"foo.png\" /></p></body></html>";
        Document doc = Jsoup.parse(html);
        // need a better way to verify these:
        Element p = doc.body().child(0);
        assertEquals("p", p.tagName());
        Element img = p.child(0);
        assertEquals("foo.png", img.attr("src"));
        assertEquals("img", img.tagName());
    }
```

This method mainly test the parser. This function parse HTML(String) to the **Document** object. **Document** object contains attributes that can be used to analysis the **Elements** in the HTML(String). This `parsesSimpleDocument.java` test method mainly test whether the result from parse method is actually match the real data using `assertEquals`. In other word, to test the validation of function parse.

In `org/jsoup/nodes/testTitles.java`

```
    @Test
    public void testTitles() {
        Document noTitle = Jsoup.parse("<p>Hello</p>");
        Document withTitle = Jsoup.parse("<title>First</title>
<title>Ignore</title><p>Hello</p>");

        assertEquals("", noTitle.title());
        noTitle.title("Hello");
        assertEquals("Hello", noTitle.title());
        assertEquals("Hello",
noTitle.select("title").first().text());

        assertEquals("First", withTitle.title());
        withTitle.title("Hello");
        assertEquals("Hello", withTitle.title());
        assertEquals("Hello",
withTitle.select("title").first().text());
```

```
        Document normaliseTitle = Jsoup.parse("<title>
Hello\nthere   \n   now   \n");
        assertEquals("Hello there now", normaliseTitle.title());
    }
```

This method mainly test the validation of **Document** object and its attribute. There are several assertion test in this test unit, including `noTitle` and `withTitle`. They tested the **title** attributes and validate the accuracy of the method parse.

In `org/jsoup/parser/TagTest.java`

```
  @Test
    public void isCaseSensitive() {
        Tag p1 = Tag.valueOf("P");
        Tag p2 = Tag.valueOf("p");
        assertNotEquals(p1, p2);
    }

  @MultiLocaleTest
    public void canBeInsensitive(Locale locale) {
        Locale.setDefault(locale);

        Tag script1 = Tag.valueOf("script",
ParseSettings.htmlDefault);
        Tag script2 = Tag.valueOf("SCRIPT",
ParseSettings.htmlDefault);
        assertSame(script1, script2);
    }
@Test
    public void equality() {
        Tag p1 = Tag.valueOf("p");
        Tag p2 = Tag.valueOf("p");
        assertEquals(p1, p2);
        assertSame(p1, p2);
    }
```

This Test class contains many unit test to validate the **Tag** and test the functionality of tag attributes in the **Document** object. The above code mainly test the case sensitive, equal and the Insensitive of different **Tag**.

Sample JUnit annotations

NOTATION	DESCRIPTION
`@Test`	Identifies a method as a test method.
`@Disabled("reason")`	Disables a test method with an option reason.
`@BeforeEach`	Executed before each test. Used to prepare the test environment, e.g., initialize the fields in the test class, configure the environment, etc.
`@AfterEach`	Executed after each test. Used to cleanup the test environment, e.g., delete temporary data, restore defaults, cleanup expensive memory structures.
`@DisplayName("<Name>")`	that will be displayed by the test runner. In contrast to method names the name can contain spaces to improve readability.
`@RepeatedTest(<Number>)`	Similar to `@Test` but repeats the test a of times
`@BeforeAll`	Annotates a method which is executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as `static` to work with JUnit.
`@AfterAll`	Annotates a method which is executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this annotation need to be defined as `static` to work with JUnit.
`@TestFactory`	Annotates a method which is a Factory for creating dynamic tests
`@Nested`	Lets you nest inner test classes to force a certain execution order
`@Tag("<TagName>")`	Tags a test method, tests in JUnit 5 can be filtered by tag. E.g., run only tests tagged with "fast".
`@ExtendWith`	Lets you register an Extension class that adds functionality to the tests

Partitioning

In class, we talked about six general engineering principles. One of them is **partition**, which means that divide and conquer.

Motivate the need for systematic functional testing and partition testing.

In our class, "Functional testing" usually implies systematic testing. We use functional testing instead of random or uniform testing is **to find needles and remove them from hay.** In detail, failures are sparse in the space of possible inputs, but dense in some parts of the space. If we systematically test some cases from each part, we will include the dense parts.

Describe these concepts.

Functional testing: Deriving test cases from program specifications.

Systematic testing: Try to select inputs that are especially valuable. Usually by choosing representatives of classes that are apt to fail often or not at all.

Partition testing: separates the input space into classes whose union is the entire space — "Equivalence Partition".

Then, select a feature that allows for partitioning.

In jsoup project, we have XML parser and HTML parser, but in test cases, jsoup only has HTML parser test. Therefore, we decided to choose XML parser and test the functions in XML parser.

Specify your partitions (and boundaries when appropriate) in English — describe them.

In this project, we select XML parse as our partition feature.

In test case, we try to figure out whether Jsoup can process the correct xml string as well as incorrect xml string using `assertEquals` function.

For boundaries, we talked about Leap Year in class, and its boundaries are range of years and certain set of years, like 2000. For our case, we don't have ranges of xml. Then we figure out that we can use the length of each xml file as ranges. Also, we try boundaries case like xml with large tag name case.

Then, write new test cases in JUnit, and describe and document those test cases and how they run.

Test whether Jsoup could get elements from the tag we assign:

```
@Test
public void parsesDocumentSize() {
    Document doc = Jsoup.parse(xml);
    Elements name = doc.getElementsByTag( "name" );
    assertEquals(2, name.size());
}
```

Test whether Jsoup could get element from the elements array:

```
@Test
public void parsesSimpleDocumentElement() {
    Document doc = Jsoup.parse(xml);
    Elements name = doc.getElementsByTag( "name" );
    Element element = name.get( 0 );
    String text = element.text();
    assertEquals("tom", text);
}
```

Test whether Jsoup could return correct answer when we input a wrong tag:

```
@Test
public void parsesNullExistDocumentElement() {
    Document doc = Jsoup.parse(xml);
    Elements id = doc.getElementsByTag( "id" );
    assertEquals(0, id.size());
}
```

Test whether Jsoup could get xml's attribution:

```
@Test
public void parsesSimpleDocumentElementAttr() {
    Document doc = Jsoup.parse(xml);
    Elements name=doc.getElementsByTag("student");
    Element element = name.get( 0 );
    //System.out.println(element.text());
    String studentNumber=element.attr("number");
    //System.out.println(studentNumber);
    assertEquals("0001", studentNumber);
}
```

Test whether a very large tag name can be parse by Jsoup:

```
@Test
public void handleSuperLargeTagNames() {
    // unlikely, but valid. so who knows.

    StringBuilder sb = new StringBuilder(maxBufferLen);
    do {
        sb.append("LargeTagName");
    } while (sb.length() < maxBufferLen);
    String tag = sb.toString();
    String xml = "<" + tag + ">One</" + tag + ">";
    Document doc =
Parser.xmlParser().settings(ParseSettings.preserveCase).parseInput(
xml, "");
    Elements els = doc.select(tag);
    assertEquals(1, els.size());
    Element el = els.first();
    assertNotNull(el);
    assertEquals("One", el.text());
    assertEquals(tag, el.tagName());
}
```

Team Members

Sun Yu(http://github.com/duke326/)

Lai Wang(https://github.com/laiwang2020/)

Xinyi Hu(https://github.com/samaritanhu)

Reference

https://www.vogella.com/tutorials/JUnit/article.html