File handling

- **Write/Read excel using apache POI/Maven**
    1. First Go to the https://mvnrepository.com/
    2. Search for “apache poi”
    3. For our course, we used below library

    4. Version 5.2.3 was taken for our example on 23rd of September, 2023, which was the latest version at the moment.
    5. Take the maven dependency and paste it inside your pom.xml file. If there is no tag call <dependancies></dependancies>, create it in your pom.xml and paste your dependency inside dependencies.

```
<!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>5.2.3</version>
</dependency>
```

    6. Follow the source code given you for more understanding.

- **Write/Read json file using jackson-databind ObjectMapper**
    1. Same as above steps. We used the following library and following dependency version on source code.
    Search for “jackson json” on https://mvnrepository.com/


```
<!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.15.2</version>
</dependency>
```

Follow the source code given you for more understanding.

- **Write/Read xml file using jackson-dataformat-xml**
    1. Same as above steps. We used the following library and following dependency version on source code.
       Search for “jackson xml” on https://mvnrepository.com/

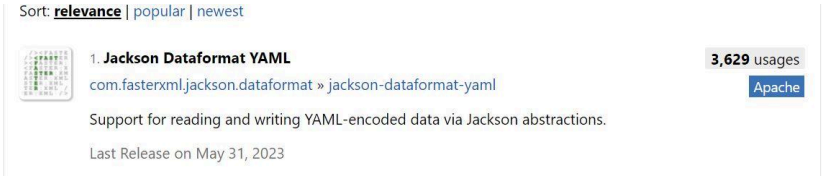

```
<!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-xml -->
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.15.2</version>
</dependency>
```

Follow the source code given you for more understanding.

- **Write/Read YAML file using jackson-dataformat-yaml**

This was given to you to practice at home. You can follow the below steps.

1. Same as above steps. We can use the following library and following dependency version on source code.
   Search for “jackson yaml” on https://mvnrepository.com/




```
<!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-yaml -->
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-yaml</artifactId>
    <version>2.15.2</version>
</dependency>
```

TestNG

- **Why**
  - **Parallel execution**
    - Parallel execution in testing refers to the ability to run multiple test cases or test methods concurrently, or in parallel, instead of sequentially. It allows you to execute tests simultaneously on multiple threads or processes, which can significantly reduce the overall test execution time.

    - Parallel execution is especially beneficial in scenarios where you have a large number of test cases, and running them sequentially would be time-consuming. By running tests in parallel, you can take advantage of modern multi-core processors and distributed computing environments to speed up the testing process.

      Speed: Parallel execution can dramatically reduce the time required to execute a suite of tests. This is particularly important in continuous integration (CI) and continuous delivery (CD) pipelines, where fast feedback is essential.

      Isolation: Tests should be designed to be independent of each other so that they can run in parallel without interfering with each other's results. Shared resources or dependencies between tests should be managed carefully.

      Thread Safety: Parallel execution can introduce concurrency issues, such as race conditions. It's crucial to ensure that your tests and application code are thread-safe when running in parallel.

      Test Framework Support: Many test frameworks and testing tools, including TestNG and JUnit, provide built-in support for parallel test execution. They offer configuration options to specify the degree of parallelism (e.g., running tests in multiple threads or processes).

      Parallelism Strategies: Depending on your testing framework, you can implement parallelism at various levels, such as at the test suite level (running entire test classes in parallel) or at the test method level (running individual test methods in parallel).

      Resource Management: When running tests in parallel, you need to manage resources efficiently, such as database connections, web server instances, or any other shared resources. Resource contention can affect test stability.

Reporting: Parallel test execution can generate consolidated test reports that show the results of all parallel executions. These reports help identify failures and issues across multiple test runs.

Configuration: Parallel execution often requires configuration in the test framework and the testing environment. Test runners need to allocate resources and manage the parallel execution process.

- **Ordered execution**
  - Ordered execution in TestNG refers to the ability to specify and enforce a specific order for executing test methods within a test class. TestNG provides mechanisms for controlling the order in which test methods run, allowing you to define dependencies and sequence in your test cases.

    Dependent Test Methods: You can specify that one test method depends on the successful execution of another test method by using the dependsOnMethods attribute in the @Test annotation. This ensures that the dependent test method runs only if the specified method(s) have passed.

    ```
    @Test
    public void login() {
        // ...
    }

    @Test(dependsOnMethods = "login")
    public void performActionAfterLogin() {
        // This method will run only if "login" method succeeds.
    }
    ```

    Priority Attribute: TestNG allows you to assign priorities to test methods using the priority attribute in the @Test annotation. Test methods are executed in ascending order of their priorities.

    ```
    @Test(priority = 1)
    public void testMethod1() {
        // This method runs first.
    }

    @Test(priority = 2)
    public void testMethod2() {
        // This method runs after testMethod1.
    }
    ```

XML Configuration: In the testng.xml configuration file, you can specify the order in which test methods or test classes should be executed by listing them in a specific sequence within <test>, <classes>, or <methods> elements. This approach provides fine-grained control over test execution order.

```
<test name="MyTest">
  <classes>
    <class name="com.example.MyTestClass">
      <methods>
        <include name="testMethod1"/>
        <include name="testMethod2"/>
      </methods>
    </class>
  </classes>
</test>
```

- **Support data driven testing**
  - Data-driven testing in TestNG is a testing approach in which you execute the same test method with multiple sets of input data. The primary goal is to test a piece of functionality under various data scenarios to ensure that it behaves correctly and produces the expected results. Data-driven testing is particularly useful when you want to verify that your application handles different input data correctly and consistently.

```
import org.testng.annotations.Test;
import org.testng.annotations.DataProvider;

public class DataDrivenTest {

  @Test(dataProvider = "testData")
  public void testWithDifferentData(String input1, int input2) {
    // Your test logic here using the input data
    System.out.println("Input1: " + input1);
    System.out.println("Input2: " + input2);
  }

  @DataProvider(name = "testData")
  public Object[][] testData() {
    // Provide test data in a two-dimensional array
    return new Object[][]{
      {"Data Set 1", 100},
      {"Data Set 2", 200},
```

```
            {"Data Set 3", 300}
        };
    }
}
```

- **Reporting and support third party reporting**

- **TestNG Basic Annotations**

  Follow the given example of the project source to understand the below annotations.

  - BeforeSuite/ AfterSuite
  - BeforeTest/ AfterTest
  - BeforeClass/AfterClass
  - BeforeMethod/ AfterMethod
  - Test
  - BeforeGroup/ AfterGroup
  - DataProvider (Support data driven testing)

- TestNG.xml

  The testng.xml file is a configuration file used in TestNG, a popular testing framework for Java. It serves several important purposes in test automation

  Test Suite Configuration: The primary purpose of testng.xml is to define a test suite. A test suite is a collection of test classes or test methods that need to be executed together. The file specifies which test classes to run and in what order.

  Parallel Execution: testng.xml allows you to specify how tests should be executed in parallel. You can configure parallel execution at various levels, such as running test classes, test methods, or groups of tests concurrently. This is particularly useful for speeding up test execution on multi-core machines.

  Inclusion and Exclusion: You can use testng.xml to include or exclude specific test classes or test methods from the test suite. This provides flexibility in defining what gets tested and what doesn't.

  Test Parameters: The file allows you to define parameters that can be passed to test methods. Parameters can be used to customize test behavior or supply test data.

  Listeners and Reporters: TestNG provides listeners and reporters for customizing test execution and generating detailed test reports. You can

configure these listeners and reporters in testng.xml to capture logs, screenshots, or other artifacts during testing.

Dependencies: You can specify dependencies between test methods or test groups in testng.xml. TestNG ensures that dependent methods or groups are executed in the correct order, based on their dependencies.

Parallel Test Suites: testng.xml also allows you to define multiple test suites. This is useful when you have different sets of tests that you want to execute separately but within the same testing framework.

Parameterization: You can parameterize test data in testng.xml and use it in your test methods. This enables you to run the same test methods with different sets of input data.

Listeners: You can specify listeners in testng.xml to perform actions at various stages of test execution, such as test start, test failure, and test success. This is useful for customizing test behavior or generating custom reports.

- Control the execution flow
- Define parameters
- Include and exclude tests
- Parallel execution