# Project Setup Using Karta

1. Pre-Requisites:
   1. Java Setup:

As per the OS platform available, Install Java(version>=17) as steps described from the link: <https://docs.oracle.com/en/java/javase/17/install/overview-jdk-installation.html>

* 1. Maven Setup:

Install Apache Maven as per the step described from the link:

<https://maven.apache.org/install.html>

* 1. Git Bash:

Install Git bash (If the existing terminal doesn’t support git commands) and pair your SSH keys into your GitHub account. The detailed steps are described: <https://docs.github.com/en/authentication/connecting-to-github-with-ssh>

* 1. IDE setup:

Install suitable IDE for the development activity. In following steps, we used IntelliJ for our Project. Link to refer on IntelliJ installation: <https://www.jetbrains.com/help/idea/installation-guide.html>

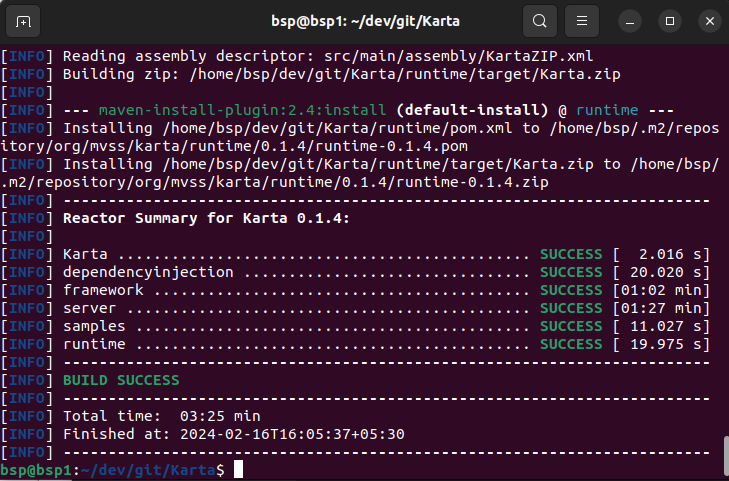
* 1. Build Karta Dependencies:
     1. Clone Karta Project: Go to desired directory, where you need to clone to project and run the below command in the command prompt/terminal.

git clone git@github.com:ManianVSS/Karta.git

* + 1. Download Karta dependency: Open the terminal onto the directory where project is downloaded and run below command.

mvn clean install

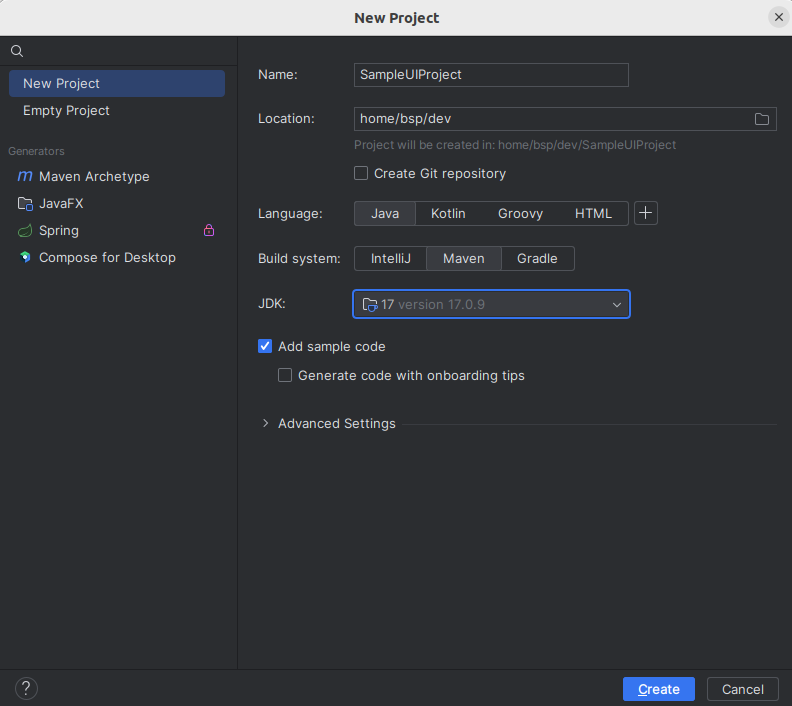
Post running the above command, the build should be successful.



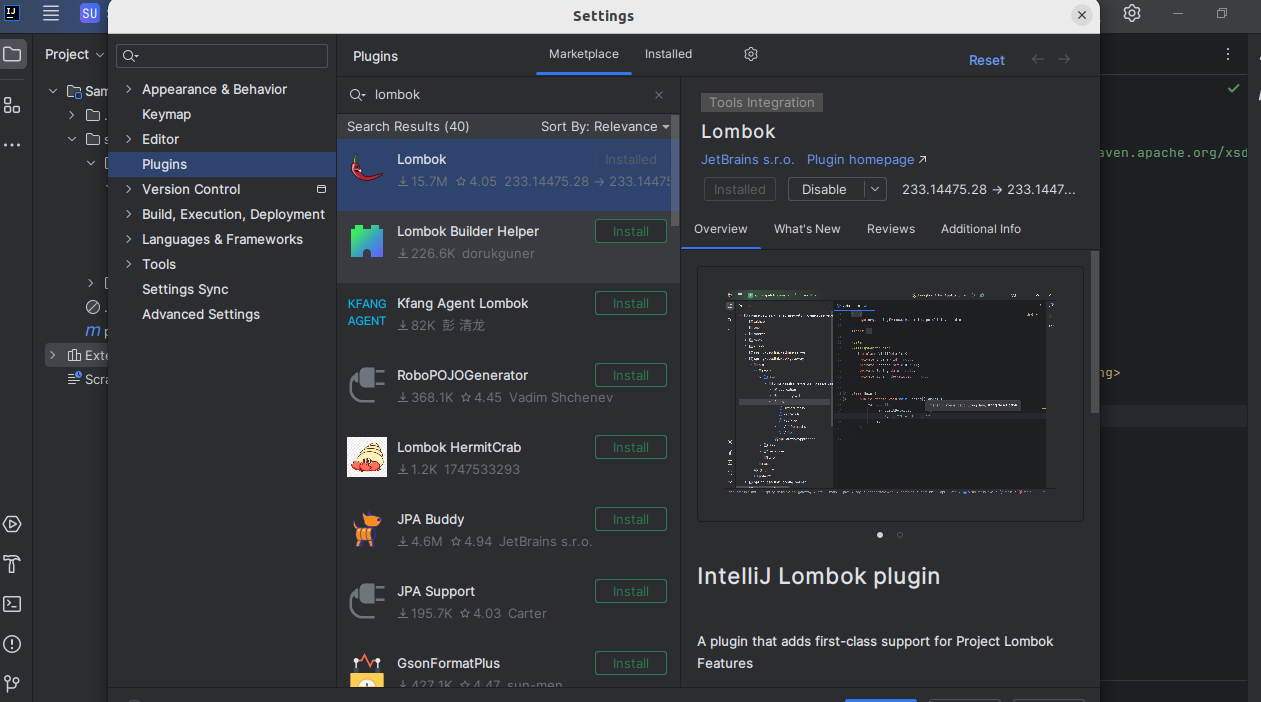
1. New Project Setup:

We are setting up a Sample Project using the Karta Dependencies. Make sure all the pre-requisites are fulfilled. Follow the below steps on creating new **SampleUIProject**.

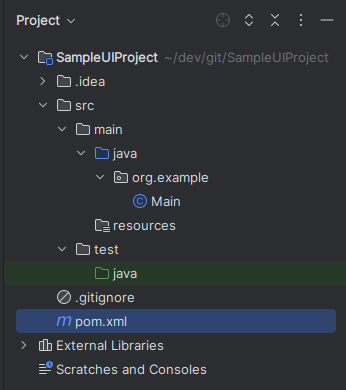
1. Launch the IntelliJ IDE, click on New Project and provide the options as shown below:



1. Click on create and wait till the indexing is completed.
2. Click on settings icon in the right top of IntelliJ and click on Plugins sections and search for lombok in the searchbar. Install lombok plugin.



1. Restart the IDE
2. Project will be created with folder structure as shown below:

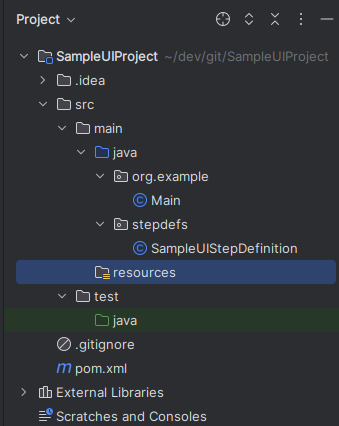


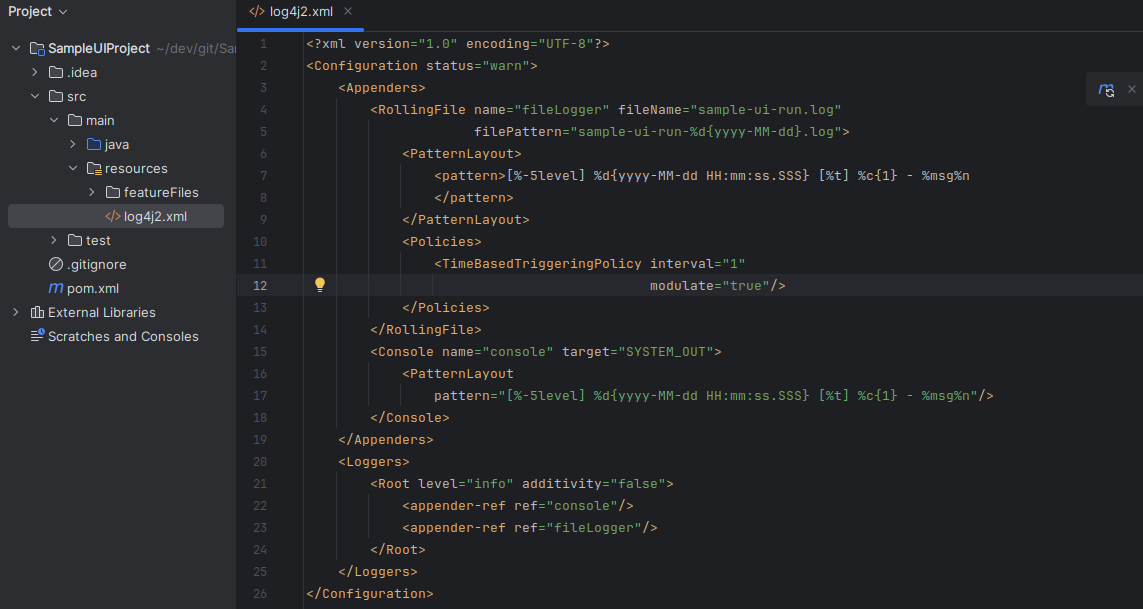
1. Open *pom.xml* and add the Karta dependencies.

<dependencies>  
  
 <dependency>  
 <groupId>org.mvss.karta</groupId>  
 <artifactId>framework</artifactId>  
 <version>0.1.4</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.30</version>  
 <scope>provided</scope>  
 </dependency>  
</dependencies>



* 1. Create new Packages:

1. Right click on the folder ***java*** and click on create new package. Mention the package name as ***stepdefs.***
2. Right click on the ***stepdefs*** and click on new class. Mention the class name as ***SampleUIStepDefinition.***(This is where the BDD steps are implemented)
3. The project structure looks as below:
   1. Logging configuration:
4. Create a file with ***log4j2.xml*** under the ***resources*** folder and copy the contents as below:

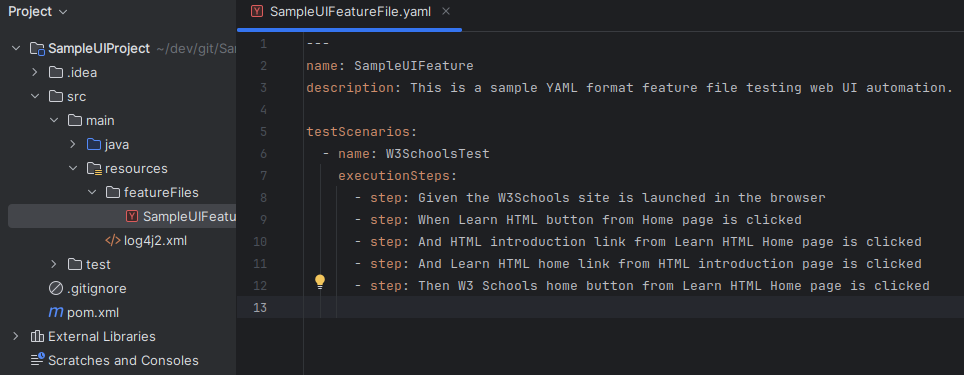


<?xml version="1.0" encoding="UTF-8"?>  
<Configuration status="warn">  
 <Appenders>  
 <RollingFile name="fileLogger" fileName="sample-ui-run.log"  
 filePattern="sample-ui-run-%d{yyyy-MM-dd}.log">  
 <PatternLayout>  
 <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n  
 </pattern>  
 </PatternLayout>  
 <Policies>  
 <TimeBasedTriggeringPolicy interval="1"  
 modulate="true"/>  
 </Policies>  
 </RollingFile>  
 <Console name="console" target="SYSTEM\_OUT">  
 <PatternLayout  
 pattern="[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n"/>  
 </Console>  
 </Appenders>  
 <Loggers>  
 <Root level="info" additivity="false">  
 <appender-ref ref="console"/>  
 <appender-ref ref="fileLogger"/>  
 </Root>  
 </Loggers>  
</Configuration>

* 1. Feature File creation:

1. Under ***resources*** folder, create a directory ***featureFiles*** and create a file ***SampleUIFeatureFile.yaml*** under it.

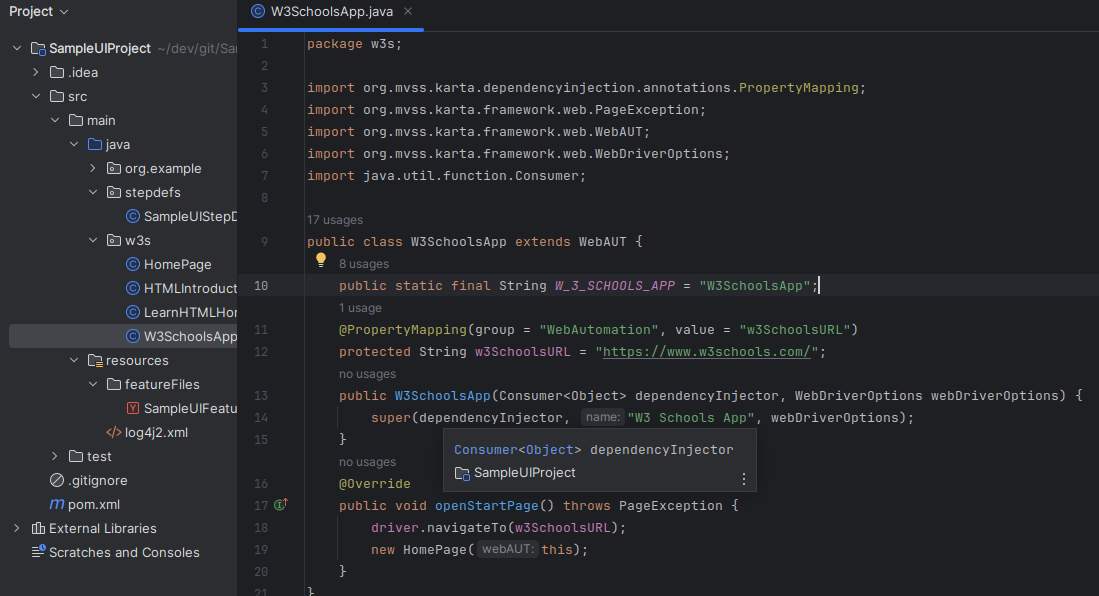
---  
name: SampleUIFeature  
description: This is a sample YAML format feature file testing web UI automation.  
  
testScenarios:  
 - name: W3SchoolsTest  
 executionSteps:  
 - step: Given the W3Schools site is launched in the browser  
 - step: When Learn HTML button from Home page is clicked  
 - step: And HTML introduction link from Learn HTML Home page is clicked  
 - step: And Learn HTML home link from HTML introduction page is clicked  
 - step: Then W3 Schools home button from Learn HTML Home page is clicked



* 1. POM class creation:

1. Create an WebPage used in your application. Create an package ***w3s*** and create an java file ***W3SchoolsApp.java*** under it.

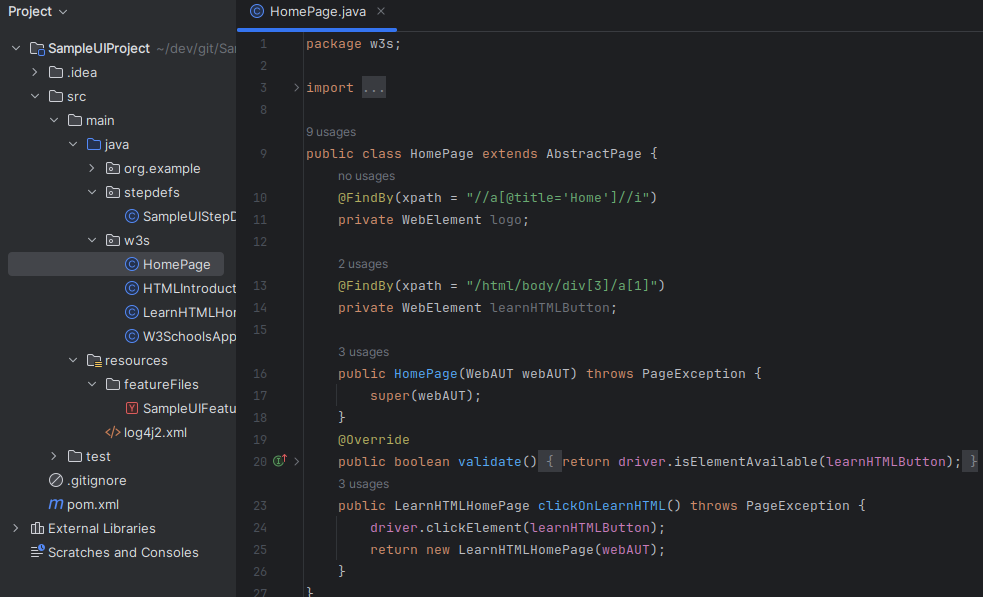
package w3s;  
  
import org.mvss.karta.dependencyinjection.annotations.PropertyMapping;  
import org.mvss.karta.framework.web.PageException;  
import org.mvss.karta.framework.web.WebAUT;  
import org.mvss.karta.framework.web.WebDriverOptions;  
import java.util.function.Consumer;  
  
public class W3SchoolsApp extends WebAUT {  
 public static final String *W\_3\_SCHOOLS\_APP* = "W3SchoolsApp";  
 @PropertyMapping(group = "WebAutomation", value = "w3SchoolsURL")  
 protected String w3SchoolsURL = "https://www.w3schools.com/";  
 public W3SchoolsApp(Consumer<Object> dependencyInjector, WebDriverOptions webDriverOptions) {  
 super(dependencyInjector, "W3 Schools App", webDriverOptions);  
 }  
 @Override  
 public void openStartPage() throws PageException {  
 driver.navigateTo(w3SchoolsURL);  
 new HomePage(this);  
 }  
}



.

1. Create a create class for each of the WebPage, which contains its DOM elements and its actionable methods are mentioned under it. Sample POM class for (W3Schools **Homepage.java**) is shown below:

package w3s;  
  
import org.mvss.karta.framework.web.AbstractPage;  
import org.mvss.karta.framework.web.PageException;  
import org.mvss.karta.framework.web.WebAUT;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
  
public class HomePage extends AbstractPage {  
 @FindBy(xpath = "//a[@title='Home']//i")  
 private WebElement logo;  
  
 @FindBy(xpath = "/html/body/div[3]/a[1]")  
 private WebElement learnHTMLButton;  
  
 public HomePage(WebAUT webAUT) throws PageException {  
 super(webAUT);  
 }  
 @Override  
 public boolean validate() {  
 return driver.isElementAvailable(learnHTMLButton);  
 }  
 public LearnHTMLHomePage clickOnLearnHTML() throws PageException {  
 driver.clickElement(learnHTMLButton);  
 return new LearnHTMLHomePage(webAUT);  
 }  
}

1. Similarly create Dom for Other webpage which your application with interact. Here we have created for ***HTMLIntroductionPage*** & ***LearnHTMLHomePage***.
   1. Hook steps:
2. Hooks class helps in running special methods/stepdefinitions based on the tags configured.

Eg: In our case, we need a browser to be launched with the URL as per the WebApp(W3SchoolsApp), which was created before. The properties to the web browser are updated using the annotation ***@PropertyMapping*** which fetches the values from the YAML defined under ***TestProperties*** folder.

The special methods are annotated and run as per the lifecycle of its execution defined. Few such annotations are @BeforeRun, @BeforeFeature(*FilePattern)*, @BeforeScenario(*TagName*), @ScenarioFailed(*TagName*), @AfterScenario(*TagName*), @AfterFeature(*FilePattern*) & @AfterRun

package stepdefs;  
  
import lombok.extern.log4j.Log4j2;  
import org.mvss.karta.Constants;  
import org.mvss.karta.dependencyinjection.KartaDependencyInjector;  
import org.mvss.karta.dependencyinjection.annotations.KartaAutoWired;  
import org.mvss.karta.dependencyinjection.annotations.PropertyMapping;  
import org.mvss.karta.framework.annotations.ScenarioFailed;  
import org.mvss.karta.framework.models.result.ScenarioResult;  
import org.mvss.karta.framework.models.test.PreparedScenario;  
import org.mvss.karta.framework.models.test.TestFeature;  
import org.mvss.karta.framework.plugins.impl.kriya.\*;  
import org.mvss.karta.framework.runtime.KartaRuntime;  
import org.mvss.karta.framework.web.WebDriverOptions;  
import org.mvss.karta.framework.web.WebDriverWrapper;  
import w3s.W3SchoolsApp;  
import java.io.IOException;  
import java.util.ArrayList;  
  
@Log4j2  
public class Hooks {  
 @KartaAutoWired  
 private KartaRuntime kartaRuntime;  
  
 @KartaAutoWired  
 private KartaDependencyInjector kartaDependencyInjector;  
  
 @PropertyMapping(group = "WebAutomation", value = "webDriverOptions")  
 private WebDriverOptions webDriverOptions = new WebDriverOptions();  
  
 @PropertyMapping(group = "Kriya", value = "stepDefinitionPackageNames")  
 private ArrayList<String> stepDefinitionPackageNames = null;  
  
 @BeforeRun  
 public void beforeRun(String runName) {  
 *log*.info("@BeforeRun Kriya tag check " + runName);  
 }  
  
 @BeforeFeature("cy.\*")  
 public void beforeCycleFeatures(String runName, TestFeature feature) {  
 *log*.info("@BeforeFeature Kriya tag check " + runName + " " + feature.getName());  
 }  
  
 @BeforeScenario(Constants.*UI*)  
 public void beforeUIScenarios(String runName, String featureName, PreparedScenario scenario) {  
 *log*.info("@BeforeScenario load web driver " + runName + Constants.*SPACE* + featureName + Constants.*SPACE* + scenario.getName());  
 W3SchoolsApp w3SchoolsApp = new W3SchoolsApp(kartaDependencyInjector::inject, webDriverOptions);  
 scenario.getContextBeanRegistry().put(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*, w3SchoolsApp);  
 }  
  
 @ScenarioFailed(Constants.*UI*)  
 public void takeScreenshotForUIScenarioFailure(String runName, String featureName, PreparedScenario scenario, ScenarioResult scenarioResult) {  
 *log*.info("@ScenarioFailed called for " + runName + Constants.*SPACE* + featureName + Constants.*SPACE* + scenario.getName());  
  
 W3SchoolsApp w3SchoolsApp = (W3SchoolsApp) scenario.getContextBeanRegistry().get(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*);  
  
 *log*.info("Scenario result is " + scenarioResult);  
  
 if (w3SchoolsApp != null) {  
 WebDriverWrapper driver = w3SchoolsApp.getDriver();  
 if (driver != null) {  
 try {  
 driver.takeSnapshot("FailureScreenshot-");  
 } catch (IOException e) {  
 *log*.warn("Could not take Screen shot", e);  
 }  
 }  
 }  
 }  
  
 @AfterScenario(Constants.*UI*)  
 public void afterUIScenarios(String runName, String featureName, PreparedScenario scenario) {  
 *log*.info("@AfterScenario close web driver " + runName + Constants.*SPACE* + featureName + Constants.*SPACE* + scenario.getName());  
 try {  
 W3SchoolsApp w3SchoolsApp = (W3SchoolsApp) scenario.getContextBeanRegistry().get(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*);  
 if (w3SchoolsApp != null) {  
 w3SchoolsApp.close();  
 }  
 } catch (Exception e) {  
 *log*.error("", e);  
 }  
 }  
  
 @AfterFeature("cy.\*")  
 public void afterCycleFeatures(String runName, TestFeature feature) {  
 *log*.info("@AfterFeature Kriya tag check " + runName + " " + feature.getName());  
 }  
  
 @AfterRun  
 public void afterRun(String runName) {  
 *log*.info("@AfterRun Kriya tag check " + runName);  
 }  
  
}

* 1. Map BDD to StepDefinition:

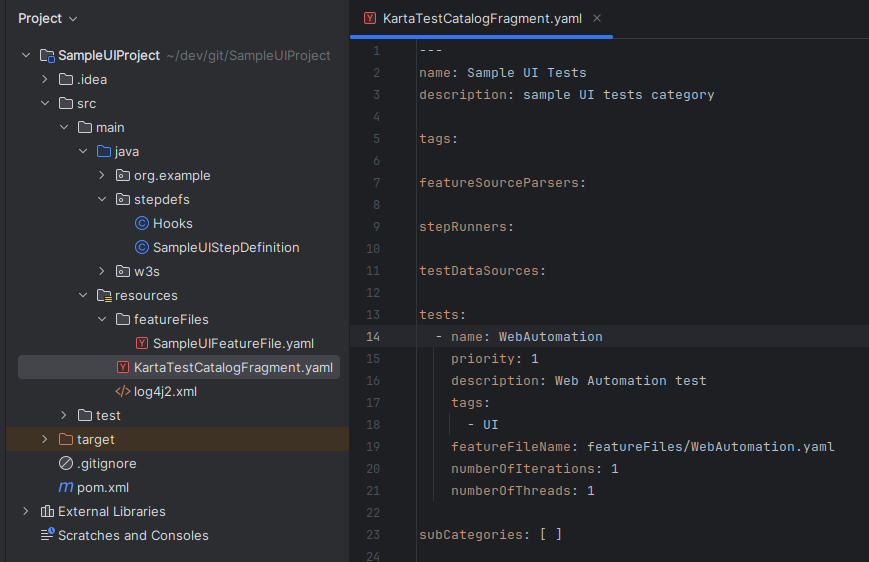
1. Implement the bdd steps to its implemention in the file ***SampleUIStepDefinition.java*** under ***stepdefs*** package.
2. Each of the methos will be associated to the BDD step(associated in Feature File or in Hooks)
3. Contents of SampleUIStepDefinition.java contents are shared below:

package stepdefs;  
  
import lombok.extern.log4j.Log4j2;  
import org.mvss.karta.framework.annotations.ContextBean;  
import org.mvss.karta.framework.annotations.StepDefinition;  
import org.mvss.karta.framework.web.PageException;  
import w3s.HTMLIntroductionPage;  
import w3s.HomePage;  
import w3s.LearnHTMLHomePage;  
import w3s.W3SchoolsApp;  
import java.io.IOException;  
  
@Log4j2  
public class SampleUIStepDefinition {  
  
 @StepDefinition("the W3Schools site is launched in the browser") //  
 public void the\_w3schools\_site\_is\_launched\_in\_the\_browser(@ContextBean(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*) W3SchoolsApp w3SchoolsApp)  
 throws PageException {  
 w3SchoolsApp.init();  
 }  
  
 @StepDefinition("close W3Schools page")  
 public void close\_the\_admin\_console(@ContextBean(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*) W3SchoolsApp w3SchoolsApp) throws PageException {  
 w3SchoolsApp.close();  
 }  
  
 @StepDefinition("take W3Schools screenshot")  
 public void take\_admin\_console\_screenshot(@ContextBean(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*) W3SchoolsApp w3SchoolsApp) throws IOException {  
 w3SchoolsApp.getDriver().takeSnapshot("Screenshot-");  
 }  
  
 @StepDefinition("Learn HTML button from Home page is clicked")  
 public void learn\_html\_button\_from\_home\_page\_is\_clicked(@ContextBean(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*) W3SchoolsApp w3SchoolsApp)  
 throws PageException {  
 HomePage homePage = (HomePage) w3SchoolsApp.getCurrentPage();  
 homePage.clickOnLearnHTML();  
 }  
  
 @StepDefinition("HTML introduction link from Learn HTML Home page is clicked")  
 public void html\_introduction\_link\_from\_learn\_html\_home\_page\_is\_clicked(@ContextBean(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*) W3SchoolsApp w3SchoolsApp)  
 throws PageException {  
 LearnHTMLHomePage learnHTMLHomePage = (LearnHTMLHomePage) w3SchoolsApp.getCurrentPage();  
 learnHTMLHomePage.goToHTMLIntroductionPage();  
 }  
  
 @StepDefinition("Learn HTML home link from HTML introduction page is clicked")  
 public void learn\_html\_home\_link\_from\_html\_introduction\_page\_is\_clicked(@ContextBean(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*) W3SchoolsApp w3SchoolsApp)  
 throws PageException {  
 HTMLIntroductionPage htmlIntroductionPage = (HTMLIntroductionPage) w3SchoolsApp.getCurrentPage();  
 htmlIntroductionPage.goToLearnHTMLHome();  
 }  
  
 @StepDefinition("W3 Schools home button from Learn HTML Home page is clicked")  
 public void w3\_schools\_home\_button\_from\_learn\_html\_home\_page\_is\_clicked(@ContextBean(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*) W3SchoolsApp w3SchoolsApp)  
 throws PageException {  
 LearnHTMLHomePage learnHTMLHomePage = (LearnHTMLHomePage) w3SchoolsApp.getCurrentPage();  
 learnHTMLHomePage.goToW3SchoolsHome();  
 }  
  
 @StepDefinition("fluent navigation on W3 Schools site is demonstrated")  
 public void fluent\_navigation\_on\_w3\_schools\_site\_is\_demonstrated(@ContextBean(W3SchoolsApp.*W\_3\_SCHOOLS\_APP*) W3SchoolsApp w3SchoolsApp)  
 throws PageException {  
 assert ((HomePage) w3SchoolsApp.init()).clickOnLearnHTML().goToHTMLIntroductionPage().goToLearnHTMLHome().goToW3SchoolsHome()  
 .clickOnLearnHTML().goToHTMLIntroductionPage().goToW3SchoolsHome().validate();  
 }  
  
}

* 1. Test Catalog file creation

1. Create a file ***KartaTestCatalogFragment.yaml*** under the resources
2. Feature Files to be run are defined in this catalog file and its type of test along with other configuration to run are defined here.

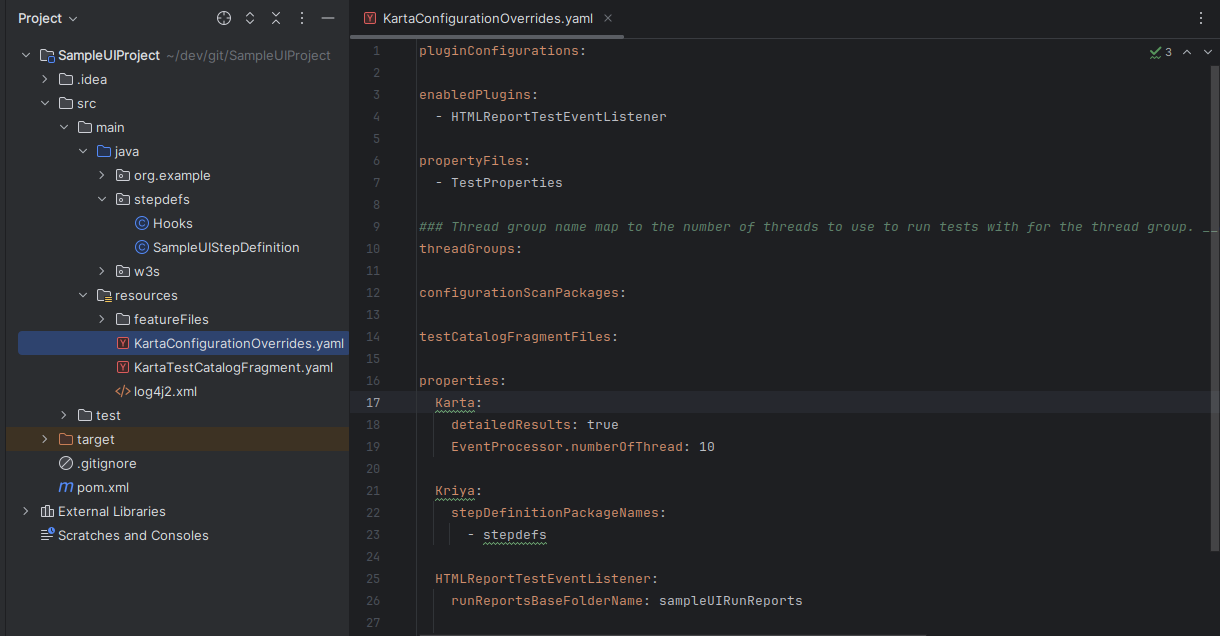
---  
name: Sample UI Tests  
description: sample UI tests category  
  
tags:  
  
featureSourceParsers:  
  
stepRunners:  
  
testDataSources:  
  
tests:  
 - name: WebAutomation  
 priority: 1  
 description: Web Automation test  
 tags:  
 - UI  
 featureFileName: featureFiles/WebAutomation.yaml  
 numberOfIterations: 1  
 numberOfThreads: 1  
  
subCategories: [ ]



* 1. Configuration Override file

1. Create file ***KartaConfigurationOverrides.yaml*** and update the configs whichever is applicable.
   1. Update the value of ***propertyFiles*** as ***TestProperties*** folder which we created
   2. Update the value of ***Properties-> Kriya-> stepDefinitionPackageNames*** as ***stepdefs*** (where our stepdefinition for SampleUIProject are available)
   3. Update the value of ***Properties->HTMLReportTestEventListener->runReportsBaseFolderName***as ***sampleUIRunReports***(where our Test result post execution are present)
   4. The Template is shared for reference.

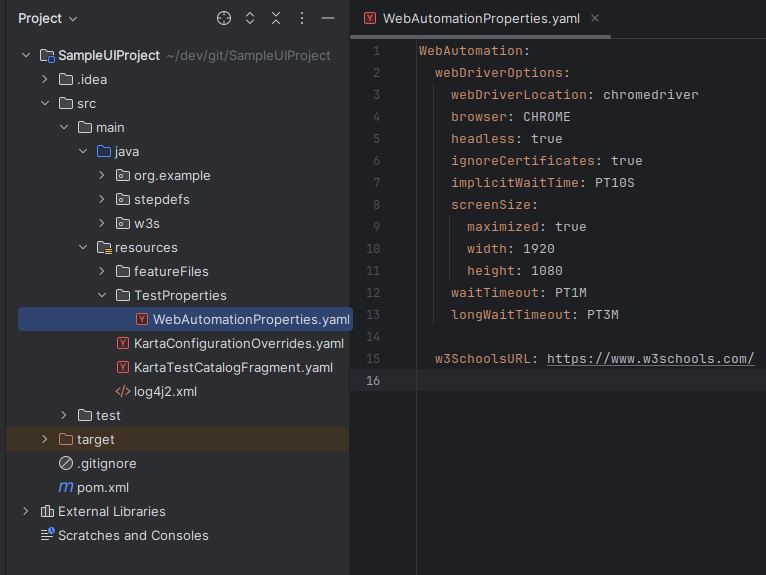
pluginConfigurations:  
  
enabledPlugins:  
 - HTMLReportTestEventListener  
  
propertyFiles:  
 - TestProperties  
  
*### Thread group name map to the number of threads to use to run tests with for the thread group. \_\_default\_\_ is the name of the default thread group*  
threadGroups:  
  
configurationScanPackages:  
  
testCatalogFragmentFiles:  
  
properties:  
 Karta:  
 detailedResults: true  
 EventProcessor.numberOfThread: 10  
  
 Kriya:  
 stepDefinitionPackageNames:  
 - stepdefs  
  
 HTMLReportTestEventListener:  
 runReportsBaseFolderName: sampleUIRunReports



* 1. Test Properties folder for web browser

1. Create a folder as ***TestProperties*** under ***resources*** folder.
2. Create a file as ***WebAutomationProperties.yaml*** under the ***TestProperties*** folder created.
3. Add the properties which are used in your execution with the group name mentioned. Same properties can be used by using the annotation ***@PropertyMapping***
4. The properties used in our Sample UI Projects are shared here for reference.

WebAutomation:  
 webDriverOptions:  
 webDriverLocation: chromedriver  
 browser: CHROME  
 headless: true  
 ignoreCertificates: true  
 implicitWaitTime: PT10S  
 screenSize:  
 maximized: true  
 width: 1920  
 height: 1080  
 waitTimeout: PT1M  
 longWaitTimeout: PT3M  
  
 w3SchoolsURL: https://www.w3schools.com/

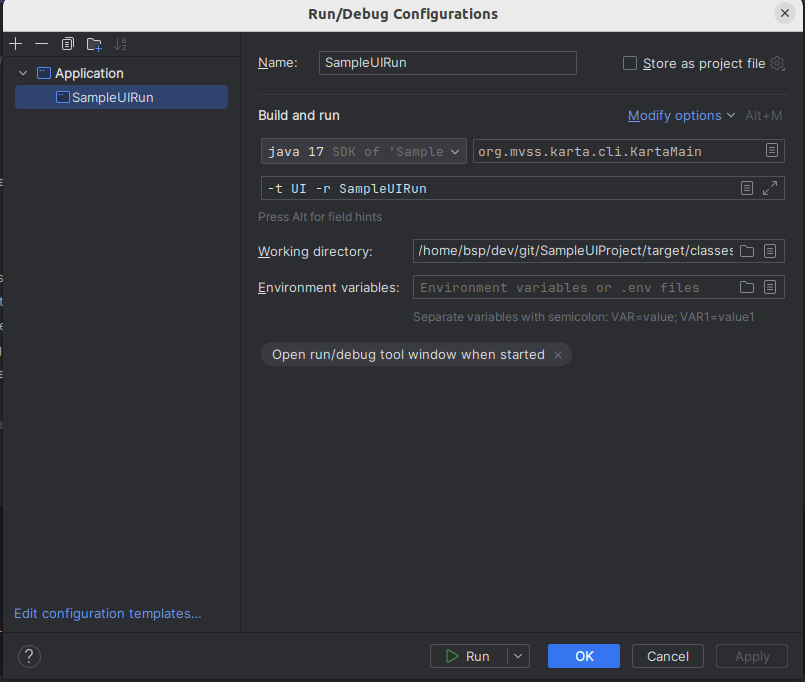


* 1. Download Chromedriver

1. Download the appropriate chrome webdriver based on the Chrome version available.
2. Link to download driver-<https://googlechromelabs.github.io/chrome-for-testing/>
3. Place the downloaded chromedriver under the ***resources*** folder
   1. Run configuration
4. Open the Terminal, where the project is been created.
5. Run the below command

mvn clean install

1. Launch IntelliJ IDE, Click dropdown next to **Run button** & click on ***Edit Configuration***
2. Click on + on the window opened
3. Provide the name for this run config as ***SampleUIRun***
4. Choose the SDK as **java 17**
5. Under main class, provide the value as ***org.mvss.karta.cli.KartaMain***
6. Add the program Arguments with the value as **-t UI -r SampleUIRun.** Where **-t** refers to the tag to be run and **-r** is the runName given to this particular execution
7. Set the Working directory pointing to the classes directory under the target folder.
8. Reference of the Run configurations is shared below.



1. Click on Run
2. The reports of the execution are present in the folder ***sampleUIRunReports***(mentioned in the section 2.8), which is under target->classes folder

