**1.Creating Jar file and using that Jar file in selenium project**

|  |  |
| --- | --- |
|  | Use [Eclipse Export Wizard](http://help.eclipse.org/juno/index.jsp?topic=/org.eclipse.jdt.doc.user/tasks/tasks-33.htm). While exporting, select "Create Runnable Jar" and select the class which is entry point (which contains main method) of your project. |

**2. create exception handling class into this project**

**Main.java**

package testproject.main;

import testproject.testmywebsite.\*;

public class Main {

public static void main(String[] args) {

TestMyWebsite tmw = new TestMyWebsite();

tmw.testMyWebsite();

tmw.stopTest();

}

}

**Testmy website.java**

import org.openqa.selenium.\*;

import org.openqa.selenium.chrome.ChromeDriver;

import java.io.\*;

import testproject.testmywebsite.tools.\*;

import testproject.testmywebsite.login.\*;

public class TestMyWebsite {

private WebDriver driver;

public TestMyWebsite() {

setUp();

}

private void setUp() {

// Create a new instance of the Chrome driver

driver = new ChromeDriver();

driver.manage().window().maximize();

}

public void testMyWebsite() {

testLogin();

}

public void testLogin() {

TestLogin tl = new TestLogin(driver);

tl.testLogin();

}

public void stopTest() {

//Close the browser

driver.quit();

}

}

**3. create Reading data base classes**

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface XmlParameters {

String[] value();

}

@Test(dataProvider = "XMLFileLoader")

@XmlParameters({"username", "password"})

public void testSomething(String username, String password) {

// implementation omitted for brevity

}

@DataProvider(name = "XMLFileLoader")

public static Object[][] getDataFromXmlFile(final Method testMethod) {

XmlParameters parameters = testMethod.getAnnotation(XmlParameters.class);

String[] fields = parameters.value();

//load just the fields you want

return new Object[][] { { "user1", "pass1" } };

}

**4. TestNG**

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use, such as:

* Annotations.
* Run your tests in arbitrarily big thread pools with various policies available (all methods in their own thread, one thread per test class, etc...).
* Test that your code is multithread safe.
* Flexible test configuration.
* Support for data-driven testing (with @DataProvider).
* Support for parameters.
* Powerful execution model (no more TestSuite).
* Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc...).
* Embeds BeanShell for further flexibility.
* Default JDK functions for runtime and logging (no dependencies).
* Dependent methods for application server testing.

**package** selenium;

**import** org.testng.annotations.BeforeClass;

**import** org.testng.annotations.Test;

**public** **class** Testing {

@BeforeClass

**public** **void** setUp() {

// code that will be invoked when this test is instantiated

}

@Test(groups = { "fast" })

**public** **void** aFastTest() {

System.***out***.println("Fast test");

}

@Test(groups = { "slow" })

**public** **void** aSlowTest() {

System.***out***.println("Slow test");

}

}

**5. junit**

JUnit is a unit testing framework for Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit, that originated with JUnit.

public class MessageUtil {

private String message;

//Constructor

//@param message to be printed

public MessageUtil(String message){

this.message = message;

}

// prints the message

public String printMessage(){

System.out.println(message);

return message;

}

}

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class TestJunit {

String message = "Hello World";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage() {

assertEquals(message,messageUtil.printMessage());

}

}

import org.junit.runner.JUnitCore;

import org.junit.runner.Result;

import org.junit.runner.notification.Failure;

public class TestRunner {

public static void main(String[] args) {

Result result = JUnitCore.runClasses(TestJunit.class);

for (Failure failure : result.getFailures()) {

System.out.println(failure.toString());

}

System.out.println(result.wasSuccessful());

}

}