14. Demonstrate integration of Selenium with Ant

Steps 1: Installing Ant

- Ant is already installed in your Practice lab. Refer QA to QE bal guide for Phase 2 for more information.

Steps 2: Writing a code for build.xml

build.xml is the most important component of the Ant build tool. For a Java project, all learning, setup, compilation and deployment related tasks are mentioned in this file in an XML format. When we execute this XML file using a command line or any IDE plugin, all instructions written into this file will get executed in a sequential manner.

Let's understand the code within a sample build.xml.

- Project tag is used to mention a project name and basedir attribute. The basedir is the root directory of an application.

```
<project name="YTMonetize" basedir=".">
```

- Property tags are used as variables in the build.xml file to be used in further steps.

```
<property name="build.dir" value="${basedir}/build"/>
                <property name="external.jars" value=".\resources"/>
    <property name="ytoperation.dir" value="${external.jars}/YTOperation"/>
<property name="src.dir"value="${basedir}/src"/>
```

- Target tags are used as steps that will execute in a sequential order. The Name attribute is the name of the target. You can have multiple targets in a single build.xml.

```
    <target name="setClassPath">
```

- **path** tag is used to bundle all the files logically which are in the common location.

```
<path id="classpath_jars">
```

- **pathelement** tag will set the path to the root of the common location where all the files are stored.

```
<pathelement path="${basedir}/"/>
```

- **pathconvert** tag is used to convert paths of all the common file inside the path tag to the system's classpath format.

```
<pathconvert pathsep=";" property="test.classpath" refid="classpath_jars"/>
```

- **fileset** tag is used to set the classpath for different third-party jars in our project.

```
<fileset dir="${ytoperation.dir}" includes="*.jar"/>
```

- **Echo** tag is used to print the text on the console.

```
<echo message="deleting existing build directory"/>
```

- **Delete** tag will clean the data from the given folder.

```
<delete dir="${build.dir}"/>
```

- **mkdir** tag will create a new directory.

```
<mkdir dir="${build.dir}"/>
```

- **javac** tag is used to compile the java source code and move the .class files to a new folder.

```
<javac destdir="${build.dir}" srcdir="${src.dir}">
    <classpath refid="classpath_jars"/>
</jalvac>
```

- **jar** tag will create a jar file from the .class files.

```
<jar destfile="${ytoperation.dir}/YTOperation.jar" basedir="${build.dir}">
```

- **manifest** tag will set your main class for execution.

```
<manifest>
        <attribute name="Main-Class" value="test.Main"/>
</manifest>
```

- 'depends' attribute is used to make a target dependent on another target.

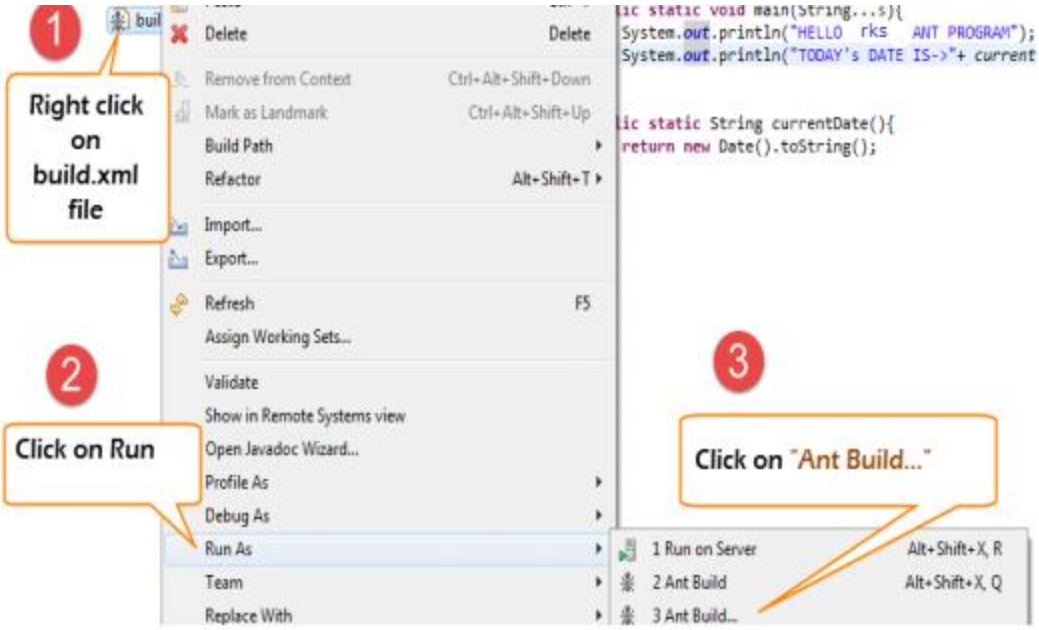
```
<target name="run" depends="compile">
```

- **java** tag will execute the main function from the jar created in the compile target section.

```
<java jar="${ytoperation.dir}/YTOperation.jar" fork="true"/>
```

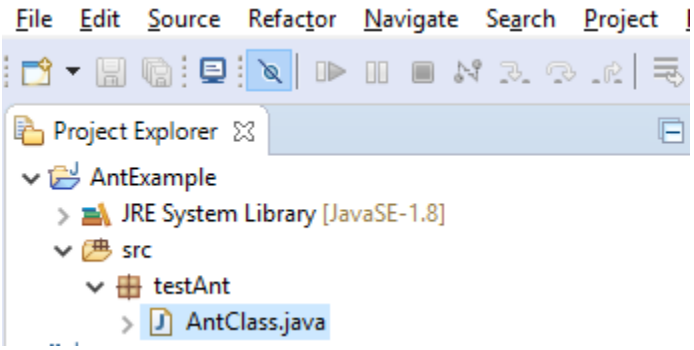
Steps 3: Running Ant using Eclipse plugin

- To run Ant from Eclipse, go to build.xml file -> right click on the file -> Run as… -> Build file.



Steps 4: Writing a code to implement the functionality of Ant

- We will take a small sample program that will explain the Ant functionality very clearly. Our project structure will look something like –



- Here in this example, we have 4 targets:

1. Set the classpath for external jars.
2. Clean the previously compiled code.
3. Compile the existing java code.

4. Run the code.

```
AntClass.class

package testAnt;

import java.util.Date;


public class AntClass {

   public static void main(String...s){

                    System.out.println("HELLO ANT PROGRAM");


                    System.out.println("TODAY's DATE IS->"+ currentDate() );


}

public static String currentDate(){

      return new Date().toString();

      }

}
```

Steps 5: Writing a build.xml file

```
<?xml version="1.0" encoding="UTF-8"  standalone="no"?>


<!--Project tag used to mention the project name, and basedir attribute will be
the root directory of the application-->



<project name="YTMonetize" basedir=".">
```

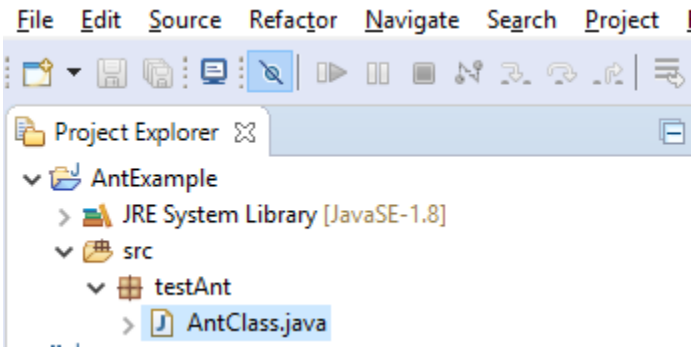
```
   <!--Property tags will be used as variables in build.xml file to use in further steps-->


     <property name="build.dir" value="${basedir}/build"/>


   <property name="external.jars" value=".\resources"/>


     <property name="ytoperation.dir" value="${external.jars}/YTOperation"/>

<property name="src.dir"value="${basedir}/src"/>

<!--Target tags used as steps that will execute in sequential order. name attribute will be the name  of the target and < a name=OLE_LINK1 >'depends' attribute used to make one target to depend on another target -->

          <target name="setClassPath">

                         <path id="classpath_jars">


                                  <pathelement       path="${basedir}/"/>


                         </path>

<pathconvert              pathsep=";"property="test.classpath" refid="classpath_jars"/>

</target>

     <target name="clean">

                <!--echo tag will use to print text on console-->

     <echo message="deleting existing build directory"/>


      <!--delete tag will clean data from given folder-->

                <delete dir="${build.dir}"/>
```

```
        </target>
    <target name="compile" depends="clean,setClassPath">

    <echo message="classpath:${test.classpath}"/>

                <echo message="compiling....."/>

    <!--mkdir tag will create new director-->

    <mkdir dir="${build.dir}"/>
        <echo message="classpath:${test.classpath}"/>

        <echo message="compiling....."/>
 <!--javac tag used to compile java source code and move .class files to a new folder-->
<javac destdir="${build.dir}" srcdir="${src.dir}">
```

```
                <classpath refid="classpath_jars"/>

    </javac>
    <!--jar tag will create jar file from .class files-->
    <jar destfile="${ytoperation.dir}/YTOperation.jar"basedir="${build.dir}">

        <!--manifest tag will set your main class for execution-->
    <manifest>
    <attribute name="Main-Class" value="testAnt. AntClass"/>
    </manifest>
```

```
            </jar>

             </target>

            <target name="run" depends="compile">


<!--java tag will execute main function from the jar created in compile target section--
>

<java jar="${ytoperation.dir}/YTOperation.jar"fork="true"/>

</target>

            </project>
```

Steps 6: Executing the TestNG code using Ant



- Here we will create a class with the TestNG methods and set the classpath for Testing in build.xml.
- Now to execute the TestNG method, we will create another testng.xml file and call this file from the build.xml file.

Step 1) We create an **"AntClass.class"** in package **testAnt.**

```
AntClass.class
```

```
package testAnt;

import java.util.Date;

import org.testng.annotations.Test;

public class AntClass {

   @Test

        public void AntTestNGMethod(){


                System.out.println("HELLO ANT PROGRAM");


                System.out.println("TODAY's DATE IS->"+ currentDate() );


        }

        public static String currentDate(){


                return new Date().toString();


        }

}
```

Step 2) Create a target to load this class in build.xml.

```
<!-- Load testNG and add to the class path of application -->

        <target name="loadTestNG" depends="setClassPath">

<!—using taskdef  tag we can add a task to run on the current project. In below  line, we are adding testing task in this project. Using testing task here now we    can run testing code using the ant script -->

        <taskdef resource="testngtasks" classpath="${test.classpath}"/>
```

```
</target>
```

Step 3) Create testng.xml

```
testng.xml

<?xml version="1.0"encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="YT"thread-count="1">

                                <test name="TestNGAnt">

                                <classes>

                                  <class name="testAnt. AntClass">

        </class>

</classes>

</test>

</suite>
```

Step 4) Create a Target in build.xml to run this TestNG code.

```
<target name="runTestNGAnt" depends="compile">

<!-- testng tag will be used to execute testng code using corresponding
testng.xml    file. Here classpath attribute is setting classpath for testng's jar
to the project-->

            <testng classpath="${test.classpath};${build.dir}">

<!—xmlfileset tag is used here to run testng's code using testing.xml file.
Using

includes tag we are mentioning path to testing.xml file-->

             <xmlfileset dir="${basedir}" includes="testng.xml"/>

</testng>
```

Step 5) The complete build.xml will look like:

```
<?xml version="1.0"encoding="UTF-8"standalone="no"?>

<!--Project tag used to mention the project name, and basedir attribute will be the root directory of the application-->

                        <project name="YTMonetize" basedir=".">

<!--Property tags will be used as variables in build.xml file to use in further steps-->

                        <property name="build.dir"value="${basedir}/build"/>

<!-- put  testng related jar in the resource  folder -->

                <property name="external.jars" value=".\resource"/>

                        <property name="src.dir" value="${basedir}/src"/>

<!--Target tags used as steps that will execute in sequential order. Name attribute will be the name of the target and 'depends' attribute used to make one target to depend on another target-->

<!-- Load testNG and add to the class path of application -->

       <target name="loadTestNG"depends="setClassPath">

            <taskdef resource="testngtasks"classpath="${test.classpath}"/>

                        </target>

                        <target name="setClassPath">

                             <path id="classpath_jars">

                        <pathelement path="${basedir}/"/>

                                    <fileset dir="${external.jars}" includes="*.jar"/>

        </path>

       <pathconvert
pathsep=";"property="test.classpath"refid="classpath_jars"/>
```

```
    </target>
    <target name="clean">
  <!--echo tag will use to print text on console-->
        <echo message="deleting existing build directory"/>
   <!--delete tag will clean data from given folder-->
        <delete dir="${build.dir}"/>
         </target>
<target name="compile"depends="clean,setClassPath,loadTestNG">
        <echo message="classpath:${test.classpath}"/>
        <echo message="compiling....."/>
            <!--mkdir tag will create new director-->
            <mkdir dir="${build.dir}"/>
         <echo message="classpath:${test.classpath}"/>
                <echo message="compiling....."/>
<!--javac tag used to compile java source code and move .class files to a new folder-->
       <javac destdir="${build.dir}"srcdir="${src.dir}">
          <classpath refid="classpath_jars"/>
           </javac>
 </target>
<target name="runTestNGAnt"depends="compile">
<!-- testng tag will be used to execute testng code using corresponding testng.xml file -->
    <testng classpath="${test.classpath};${build.dir}">
  <xmlfileset dir="${basedir}"includes="testng.xml"/>
```

```
        </testng>

</target></project>
```

Steps 7: Integrating Ant with Selenium WebDriver

- We have learned that by using Ant we can put all the third party jars in a particular location in the system and set their path for our project.
- Using this method, we are setting all the dependencies of our project in a single place and making it more reliable for compilation, execution, and deployment.
- Similarly, for our testing projects, using Selenium, we can easily mention Selenium dependency in build.xml and we don't need to add a classpath for it manually in our application.
- So, now you can ignore the below-mentioned traditional way to set classpaths for our project.

Steps 8: Modifying previous examples

We are going to modify the previous example now.

Step 1) Set the property Selenium.jars to the Selenium related jar in the resource folder.

```
<property name="selenium.jars" value=".\selenium"/>
```

Step 2) In the target **setClassPath**, add the Selenium files.

```
<target name="setClassPath">
        <path id="classpath_jars">
            <pathelement path="${basedir}/"/>
            <fileset dir="${external.jars}" includes="*.jar"/>
          <!-- selenium jar added here -->
                <fileset dir="${selenium.jars}" includes="*.jar"/>
    </path>
```

Step 3) Complete build.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>



<!--Project tag used to mention the project name, and basedir attribute
will be the root directory of the application-->

                                <project name="YTMonetize" basedir=".">

                <!--Property tags will be used as variables in build.xml file to
use in further steps-->

          <property name="build.dir" value="${basedir}/build"/>

      <!-- put  testng related jar in the resource  folder -->

                <property name="external.jars" value=".\resource"/>

<!-- put  selenium related jar in resource  folder -->

     <property name="selenium.jars" value=".\selenium"/>

          <property name="src.dir" value="${basedir}/src"/>

<!--Target tags used as steps that will execute in sequential order. name
attribute will be the name of the target and 'depends' attribute used to
make one target to depend on another target-->

      <!-- Load testNG and add to the class path of application -->

       <target name="loadTestNG" depends="setClassPath">

          <taskdef resource="testngtasks" classpath="${test.classpath}"/>

                        </target>

<target name="setClassPath">

                <path id="classpath_jars">

                                                <pathelement path="${basedir}/"/>

          <fileset dir="${external.jars}" includes="*.jar"/>

                                <!-- selenium jar added here -->

                    <fileset dir="${selenium.jars}"includes="*.jar"/>
```

```
      </path>

  <pathconvert pathsep=";" property="test.classpath"
refid="classpath_jars"/>

</target>

<target name="clean">

<!--echo tag will use to print text on console-->

           <echo message="deleting existing build directory"/>

                    <!--delete tag will clean data from given folder-->

                              <delete dir="${build.dir}"/>

                                        </target>

<target name="compile" depends="clean,setClassPath,loadTestNG">

       <echo message="classpath:${test.classpath}"/>

            <echo message="compiling....."/>

      <!--mkdir tag will create new director-->

              <mkdir dir="${build.dir}"/>

                    <echo message="classpath:${test.classpath}"/>

                    <echo message="compiling....."/>

<!--javac tag used to compile java source code and move .class files to
new folder-->

   <javac destdir="${build.dir}"srcdir="${src.dir}">

          <classpath refid="classpath_jars"/>

        </javac>

        </target>

        <target name="runTestNGAnt" depends="compile">
```

```
<!-- testng tag will be used to execute testng code using corresponding testng.xml file -->
                                <testng classpath="${test.classpath};${build.dir}">
            <xmlfileset dir="${basedir}" includes="testng.xml"/>
                        </testng>
        </target>
</project>
```

Step 4) Now change the previously created class **AntClass.java** with the new code.

AntClass.java:

```
package testAnt;

import java.util.List;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.Test;


public class AntClass {

    @Test

                public void TestNGMethod(){

        WebDriver driver = new FirefoxDriver();
```

```
                driver.get("http://demo.guru99.com/test/home/");
 List<WebElement> listAllCourseLinks =
driver.findElements(By.xpath("//div
[@class='canvas-middle']//a"));


        for(WebElement webElement : listAllCourseLinks) {
                System.out.println(webElement.getAttribute("href"));
                  }
                }
}
```

Step 5) Now, you can run your test.

Steps 9: Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files.

cd <folder path>

Initialize your repository using the following command:

git init

Add all the files to your git repository using the following command:

git add .

Commit the changes using the following command:

git commit . -m “Changes have been committed.”

Push the files to the folder you initially created using the following command:

git push -u origin master