Stage 1 - Java

Stage 2 - Web automation (Selenium) & AutoIT (Windows GUI automation)

Stage 3 - Framework 1 (TestNG, Data Driven Framework, Page Object Model, Keyword Driven Framework)

Stage 4 - BDD Framework

Stage 5 - Git and Jenkins

Java Concept - https://github.com/balaji-githubstore/selenium-java-concept-equiniti-sep-2025.git

Selenium - <https://github.com/balaji-githubstore/selenium-java-concept-equiniti-sep-2025.git>

Framework 1 - <https://github.com/balaji-githubstore/java-hybrid-framework-equiniti-sep-2025.git>

BDD Framework - https://github.com/balaji-githubstore/java-bdd-framework-equiniti-sep-2025.git

Selenium

1. Automates only the web application
2. Open Source
3. Language Independency - Java, C#, Python, Ruby, Javascript

Selenium - A suite of tools

1. Selenium IDE
   1. No Programming language knowledge required
   2. Record and playback
   3. Plugin - chrome, edge, firefox
   4. Used only simple scripting or exploratory testing
2. Selenium RC - Depreciated
   1. Programming knowledge is required
   2. Architecture

Source code (Java+Selenium RC jar) → RC server (Turn ON/OFF) → Browser

1. Selenium WebDriver
   1. Programming knowledge is required
   2. Architecture

Source code (Java+Selenium WebDriver jar) → Browser

1. Selenium Grid

If you want to scale by distributing and running tests on several machines and manage multiple environments from a central point

### Java Programming

1. Installation
   1. JDK - Java Development Kit (above 11)
      1. It also installs the JRE

<https://builds.openlogic.com/downloadJDK/openlogic-openjdk/17.0.16+8/openlogic-openjdk-17.0.16+8-windows-x64.zip>

1. IDE
   1. Eclipse
   2. IntelliJ
2. Architecture

Source code (.java) → Byte code (.class) → O/P

Compiler → helps to convert souce to byte code

JVM → Helps the operating system to understand the byte code

Compile time ⇒ Source code to byte code

Runtime → Bytecode to O/P

1. UpperCamelCase → MyFirstProject

lowerCamelCase → myFirstProject

1. Structure of an eclipse

Workspace

Project - UpperCamelCase

Package - lowercase (com.companyname.purpose)

Class - UpperCamelCase

Methods/variables - lowerCamelCase

1. Datatypes
   1. Primitive/Pre-defined datatypes
   2. Non- Primitive/Non-predefined/user defined datatypes
      1. String
      2. Array
      3. User defined datatypes
2. When we give a whole number, it will be considered as an int

When we store a decimal value, it will considered as double

1. Debugging
   1. Resume
   2. Terminate
   3. Step over
   4. Step into
2. Conditional statement
   1. If
   2. Switch
3. Iterative statements
   1. For
   2. Advance for loop
   3. While
   4. Do while
4. Methods - Building block of the program

Reusability

Maintenance

* 1. Static methods
     1. How to create a static method and call it?

//accessmodifer static returntype methodName(arguments)

To call it

classname.methodName()

* 1. Non-static methods
     1. How to create non-static method and call it?
     2. How to call non-static method?
        1. Create object
        2. Use objref.methodName()

1. Varaiable
   1. Static variable
   2. Non-static variable
2. Object
   1. Declaration
   2. Instantiation - new
   3. Initialization
3. Class - A class is a blueprint or type or template from which objects are created
4. Object
   1. An object is an instance of class
   2. Every object has its own state (non-static variable) and behaviour (non-static method)
5. Access modifier
   1. Private - accessible within the class
   2. Default - accessible within the package
   3. Protected - accessible within the package and also to the inherited classes
   4. Public - accessible anywhere
6. Constructors - Prerequisite of the object
   1. Constructor name and class name should be same without any return type
   2. It is kind of method and it gets called whenever new objects is created
   3. There will be always a default constructor and it helps to load all the non-static variable with default values.
   4. We can override the default constructor by creating explicitly our own constructor
      1. With argument
      2. Without argument
   5. If the class contains constructor with argument then we need to call that only during object creation
7. this
   1. Helps to distinguish between non-static variable and then local variable
   2. this - will point to the current object
8. Constructor overloading / Compile time polymorphism / Static polymorphism

The constructor to be called is resolved during compile time

Can create multiple constructors by changing in

1. Number of arguments
2. Sequence of arguments
3. Datatype of arguments
4. Method overloading / Compile time polymorphism / Static polymorphism

The method to be called is resolved during compile time

Can create multiple methods with the same name by changing in

1. Number of arguments
2. Sequence of arguments
3. Datatype of arguments
4. Collections
   1. Non-generic collections
      1. List
      2. HashMap
   2. Generic collections
      1. List
      2. HashMap
5. Inheritance
   1. Reuasblity and maintenance
   2. When we want to reuse methods and variable, then we can use inheritance concepts

### Selenium WebDriver

1. Create Java project
2. Configure the Selenium jar
3. Launch browser
4. Navigate to url, getTitle, getCurrentUrl, getPageSource
5. Click, type, Select
6. To inspect the element → tagname, attribute, text, or not
7. Basic locator
   1. id
   2. name
   3. className
   4. tagName
   5. linkText
   6. partialLinkText

findElement → when there are duplicate locators then findElement picks the first webelement

1. Advance locators
   1. XPath
   2. CSS
2. Synchronization
   1. Unconditional wait (from java)
      1. Thread.*sleep*(5000); → not recommended
   2. Conditional wait
      1. Implicit wait
         1. Default Implicit wait - 0s
         2. Applicable for all **findElement and findElements** methods
         3. Example: Implicit wait - 30s
            1. If the element is not present, it will check for 30 seconds and then throw an exception
            2. If the element is present, it will do the operation immediately.
            3. Polling time - 0.5s
      2. Explicit wait
         1. Exact condition
         2. Polling time - 0.5s
      3. Fluent wait
3. Dropdown
   1. With select tag
      1. selectByVisibleText()
      2. selectByValue()
      3. selectByIndex() → starts at 0
   2. Without select tag
4. Click → element should be present and visible
5. Frame, multiple tabs/windows, alert - switchTo()
6. Multiple tabs/windows
7. List vs Set
   1. List - can contain duplicates
   2. Set - cannot contain duplicates
8. Close vs quit
   1. Close - close the current tab/session
   2. Quit - close the current browser/all session and also it kills the process associated with it
9. Frame - embed one html into another html
   1. Even though the locator is correct, we get Exception in thread "main" org.openqa.selenium.NoSuchElementException:
   2. Check for frame or iframe tag
   3. Switch to frame
      1. Using index
      2. Using name or id as a string
      3. Using WebElement
10. Alert
    1. Javascript alert
11. Upload
12. Actions
    1. May not throw proper exception
    2. May not work on headless
    3. Do not disturb the mouse/keyboards

Keyboard

Modifier keys - ctrl, alt, shift → keyDown() & keyUP()

Other keys → sendKeys()

1. CSS Selector
2. Javascripts
   1. Click on hidden elements
   2. Type on readonly textbox
   3. Scroll page

Options 1 → Javascript - click & type -

document.querySelector("div[id='book-flight'] input[name='dateDeparture']").click()

document.querySelector("div[id='book-flight'] input[name='dateDeparture']").value='20 Sep 2025'

Option 2 → Javascript and webelement

**var** ele1= driver.findElement(By.*xpath*("//div[@id='book-flight']//input[@name='dateDeparture']"));

js.executeScript("arguments[0].value='20 Sep 2025'",ele1);

1. Shadowroots
2. Chromeoptions - Pre-setting to the browser
3. Dynamic table

Git - Git is a free and open source distributed version control system

Architecture

Project (in local machine) → local repository (in local machine) → remote repository (github, aws code commit, gitlab, bitbucket)

Concepts →

Modified → staging → commit

Git concepts to update code to remote repo (github or gitlab)

1. git init → initialize local repo
2. git add . → staging
3. git commit -m "first commit" → update the local repo
4. git remote add origin1 <https://github.com/balaji-githubstore/selenium-java-concept-equiniti-sep-2025.git> → registering the remote url with name origin1
5. git push -u origin1 master

Framework 1 - Hybrid Framework

1. Unit Test Framework - TestNG
2. Data Driven Framework - Separating the test methods and test data
3. Page Object Model - Page class and Page methods

Keyword Driven Framework - for effectively handling the webdriver keywords

Build Management Tools - Maven

1. Pom.xml → Project Object Model
   1. Helps to configure the jars and dependent jars easily
   2. Can run the project in the pipeline using goals
   3. Can configure the test to run easily
   4. Migrate the project easily

Packages

[com.eq](http://com.eq).test → Test Classes and Test methods

[com.eq](http://com.eq).base → Browser and report configurations

[com.eq](http://com.eq).pages → Page class and Page methods

[com.eq](http://com.eq).utilities → Reusable for handling excel, json, properties

Eclipse: Install testng for eclipse

Steps to create a framework

1. Create a Maven project
   1. Add group ID and artifact ID
2. Add dependencies
   1. TestNG - <https://mvnrepository.com/artifact/org.testng/testng/7.11.0>
   2. Selenium - <https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java/4.35.0>
3. Create a Test package, test class and test methods (@Test)
4. Every @Test method should have minimum one verification/assertions. Assertions decides whether the @Test is pass or fail
5. @Test method runs based on ASCII keycode order
6. TO change the order using priority. @Test method without priority will be given higher preference
7. Annotation
   1. @BeforeMethod → runs before each @Test
   2. @AfterMethod → runs after each @Test (even though @Test fails)
8. Page object model - Design pattern
   1. Reuse
   2. Maintenance
   3. Readability

To implement page object model

1. For each web page in the application, we need to create a class - Page class
2. Operations/Actions will happen through the methods
3. Create object repository (locators) for handling the webelements
4. @DataProvider → helps to run the @Test method with a different set of test data
   1. Create @Test with arguments
   2. Create a method that return two dimensional array and also need to provide @DataProvider
   3. Connect the @Test with @DataProvider
5. Excel - Read
   1. poi - https://mvnrepository.com/artifact/org.apache.poi/poi/5.4.1
   2. Poi-ooxml - https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml/5.4.1
6. Location - Read/Write
7. Open
8. Sheet
9. Row
10. Cell

.xlsx → XSSFWorkbook

.xls → HSFWorkbook

1. Excel with @DataProvider
2. TestNG XML - Suite file
   1. Run the project using xml
   2. Parameterization using xml
   3. Cross-browser testing
   4. Parallel execution
   5. Group the test and run

BDD Framework - Behaviour Driven Development

Actual BDD - Understanding the requirement

Modified BDD - ATDD - Acceptance test-driven development - writing the acceptance testing

BDD framework

1. Java/Javascript - Cucumber
2. C# - reqnroll
3. Python - jbehave

Architecture of cucumber

Feature file (.feature) → Step definition (.java)

Steps to create a BDD framework

1. Create maven project
2. Add dependencies
   1. Cucumber-java - https://mvnrepository.com/artifact/io.cucumber/cucumber-java
   2. Cucumber-testng - https://mvnrepository.com/artifact/io.cucumber/cucumber-testng
   3. Selenium
3. Create a feature file
   1. Add the feature header
   2. Add feature description
      1. Unformatted description
      2. Formatted description

In order to [feature goal]

As a [role]

I would like to [visible change in the application]

1. Add Scenario
   1. Provide scenario title
   2. Scenario should contain minimum one given, when, then

Given [pre-requisite]

When [operation/actions]

Then [verifications]

1. Create a runner file to configure feature and step defn
2. Step parameterization
   1. Code reuse
   2. Reduce duplicates in step defn
3. Scenario outline
   1. one scenario - multiple sets of test data
   2. Reduce duplicate in feature file
4. Background
   1. Use it mostly when you have repeated given otherwise avoid it
5. Cucumber features
   1. dryRun - to find the missing step defn
   2. publish
   3. Plugin
   4. Tags - and, or, not
6. Datatable
7. Scenario outline with Datatable
8. Hooks
   1. @Before - runs before each scenario
   2. @After - runs after each scenario. Even though fails
9. Hooks to configure the browser launch and close
10. Configuring the dependency injection for object handling
    1. Add dependencies - <https://mvnrepository.com/artifact/io.cucumber/cucumber-picocontainer/7.28.2>
    2. Configure constructor to get the object for whatever classes created in cucumber project

Concepts of BDD - Improve the reusablity, readability

1. Step parameterization - code reuse
2. Scenario outline - one scenario - multiple sets of test data
3. Background - repeated steps in all scenarios (repeated given)
4. Datatable - tabular data to step defn

—--

1 GB - 1024 MB

1 MB - 1024 KB

1 KB - 1024 B

1 B - 8 bits

Exceptions:

1. NoSuchDriverException:
2. NoSuchElementException:
3. Exception in thread "main" org.openqa.selenium.ElementClickInterceptedException:- target element is hidden by some other element
4. ElementNotInteractableException → element is present and not visible
5. NoSuchWindowException
6. org.openqa.selenium.NoAlertPresentException: no such alert