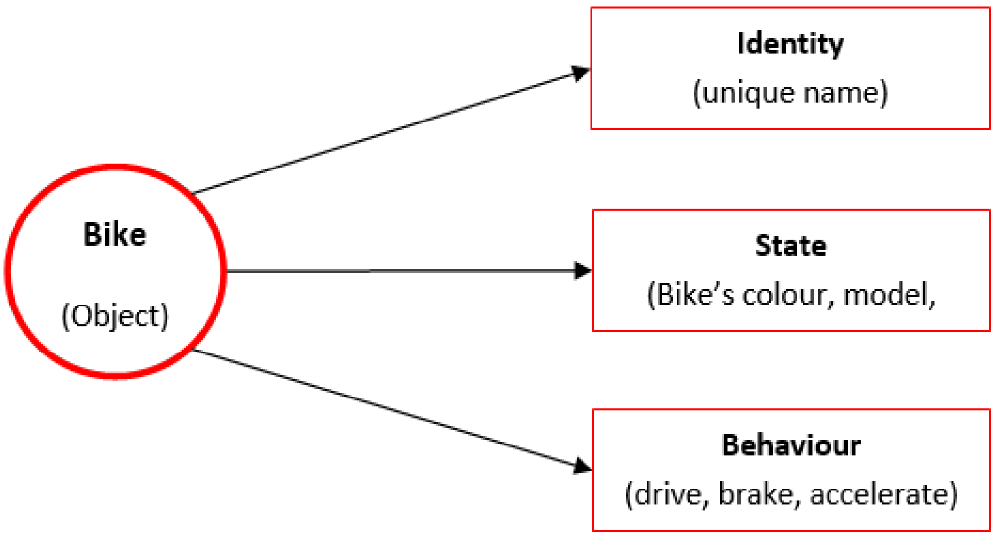
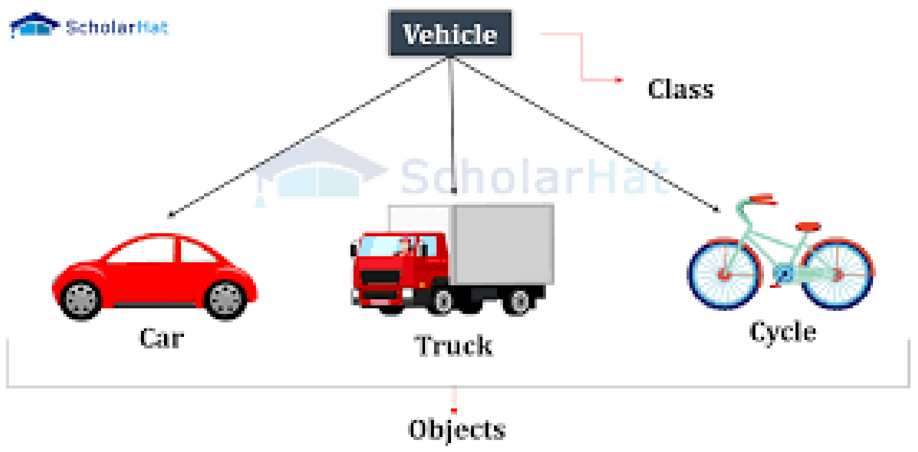
1. Object in Java

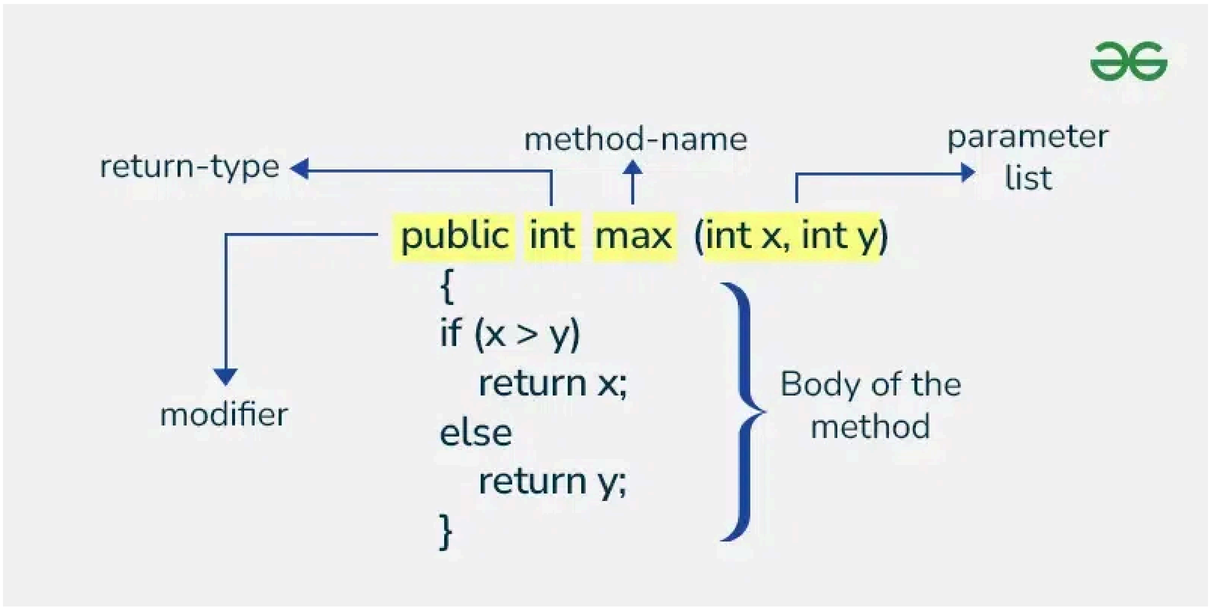
An entity that has state and behaviour is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible).





2. Method in Java

Method is a way to perform some task. Similarly, the method in Java is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using methods.



```
public int max (int x, int y)
{
if (x > y)
    return x;
else
    return y;
}
```

3. String Methods

charAt()	Returns the character at a specified index in a string	String str = "Hello"; char ch = str.charAt(1); // 'e' (index starts at 0)

equals()	Compare two strings for equality. It is case-sensitive.	String str1 = "Hello";
		String str2 = "hello";
		boolean isEqual = str1.equals(str2); // false

equalsIgnoreCase()	Compares two strings for equality, ignoring case differences	String str1 = "Hello";
		String str2 = "hello";
		boolean isEqual = str1.equalsIgnoreCase(str2); // true

isEmpty()	Checks if a string is empty (length is 0)	String str = ""; boolean isEmpty = str.isEmpty(); // true
length()	Returns the length (number of characters) of a string.	String str = "Hello"; int len = str.length(); // 5
toLowerCase()	Converts all characters in a string to lowercase.	String str = "Hello"; String lower = str.toLowerCase(); // "hello"
toUpperCase()	Converts all characters in a string to uppercase.	String str = "Hello"; String upper = str.toUpperCase(); // "HELLO"

4. Array Methods

equals()	Checks for reference equality(Memory Location) when used with arrays	int[] arr1 = {1, 2, 3};
		int[] arr2 = {1, 2, 3};
		boolean isEqual = arr1.equals(arr2);
		// false (different references)

Arrays.equals()	Check for content equality	int[] arr1 = {1, 2, 3};
		int[] arr2 = {1, 2, 3};
		boolean isEqual = Arrays.equals(arr1, arr2);
		// true (compares content)

length	Length is a property (not a method) that returns the size (number of elements) of the array.	int[] arr = {1, 2, 3, 4, 5}; int size = arr.length; // 5

5. Array List Methods

An **ArrayList** in Java is a **resizable array** from the **Java Collections Framework,** Can grow or shrink dynamically as we add or remove elements.

ArrayList<String> list = new ArrayList<>();

add()	Adds the specified element to a collection	List<String> list = new ArrayList<>();
		list.add("Apple");
		// Adds "Apple" to the list

contains()	Checks if the collection contains the specified element	List<String> list = new ArrayList<>();
		list.add("Apple");
		boolean hasApple = list.contains("Apple"); // true

forEach()	Iterates over each element in the collection and performs the specified action	List<String> list = List.of("Apple", "Banana", "Cherry"); list.forEach(item -> System.out.println(item)); // Prints each element

get()	Retrieves the element at the specified index	List<String> list = new ArrayList<>();
		list.add("Apple");
		String item = list.get(0); // "Apple"

isEmpty()	Checks if the collection is empty.	List<String> list = new ArrayList<>(); boolean empty = list.isEmpty(); // true

iterator()	Returns an Iterator to traverse the collection.	List<String> list = List.of("Apple", "Banana", "Cherry");
		Iterator<String> iterator = list.iterator();
		while (iterator.hasNext())

		{
		    System.out.println(iterator.next());
		}

listIterator()	Returns a ListIterator for bi-directional traversal of a List	List<String> list = List.of("Apple", "Banana", "Cherry");
		ListIterator<String> listIterator = list.listIterator();
		while (listIterator.hasNext())
		{
		
		System.out.println(listIterator.next());
		}

size()	Returns the number of elements in the collection	List<String> list = List.of("Apple", "Banana"); int size = list.size(); // 2

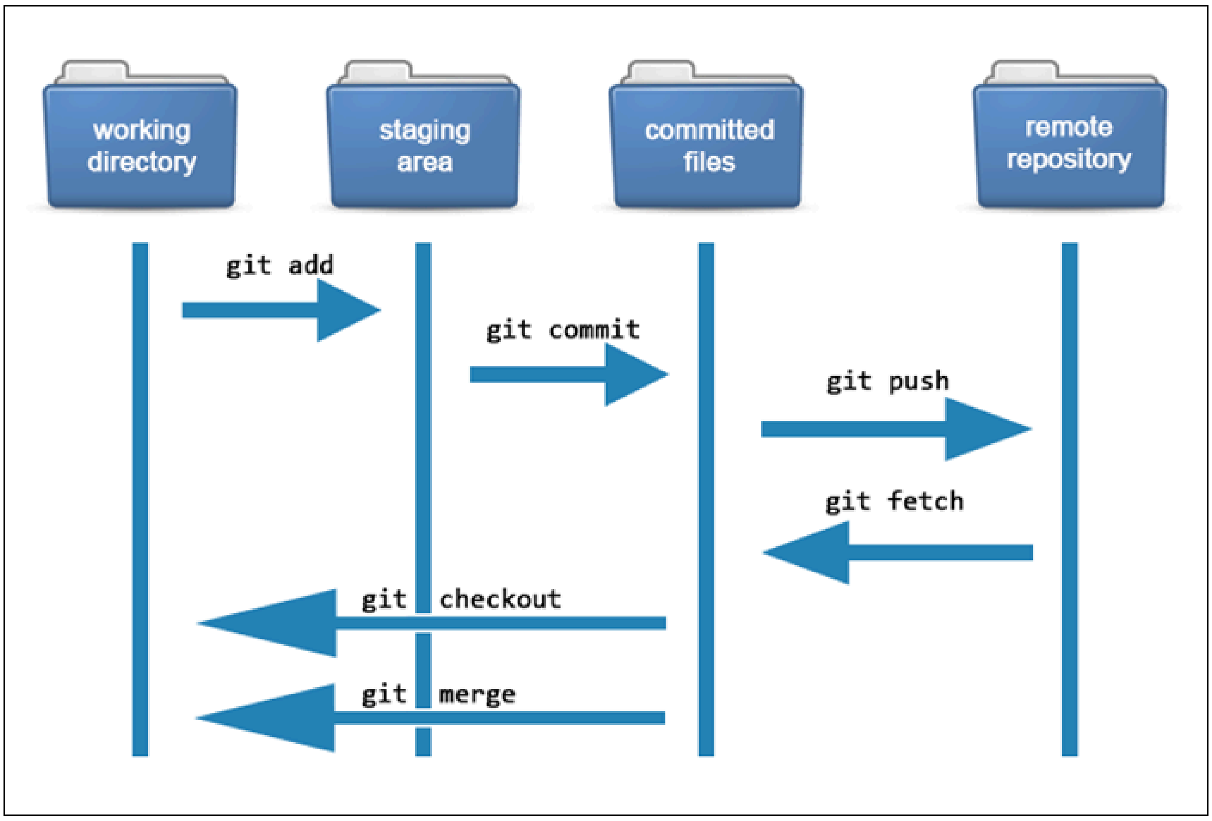
toArray()	Converts the collection into an array	List<String> list = List.of("Apple", "Banana");
		Object[] array = list.toArray();
		System.out.println(array[0]); // "Apple"

2. Git Basics

2.1. What is a Git repository?

A Git repository is a storage space where the project's files and their entire version history are tracked. It can be hosted locally on our computer or remotely on platforms like GitHub. It allows us to manage, collaborate on, and track changes in a project's files over time.

2.2. How does Git work?



- **Tracking Changes:** Git monitors file changes in our project.
- **Staging Changes:** Changes are added to a staging area to prepare them for a commit.
- **Committing:** Changes in the staging area are saved as a commit, creating a snapshot of our project at a point in time.
- **Branching and Merging:** It allows multiple branches for independent work that can later merge.
- **Distributed Workflow:** Developers work on local repositories and sync changes with a remote repository using push and pull.

2.3. Briefly explain the Git commands below

Git add	Adds changes in the working directory to the staging area, preparing them for the next commit.	git add file.txt - To add a file git add . -To add all the files
Git push	Uploads local commits to a remote repository	git push origin main
Git status	Shows the current state of the working directory and staging area, indicating any changes or untracked files.	git status
Git commit	Saves the staged changes to the repository with a descriptive message.	git commit -m "Add new feature"

2.3.1. Differences between commit and push

	Git Commit	**Git Push**
Where it happens	Local repository	Remote repository
Purpose	Saves changes locally	Sends changes to the remote repository
Visibility	Visible only to me (locally)	Visible to others (in the remote repo)
Dependency	Can be done independently of a push	Requires previous commits to push

2.3.2. Differences between fetch and pull in git

(https://youtu.be/GOrhB6eYASU?si=s-A2FkcggL2hUIar)

	Git Fetch	**Git pull**
Purpose	Downloads updates from the remote repository but does not modify the working directory.	Downloads updates from the remote repository **and merges them** into the current branch.
Effect on Files	Updates only the remote-tracking branches in the local repository.	Updates the remote-tracking branches **and applies the changes** to the working directory.

Scope	Non-destructive; does not affect the working directory.	Destructive if there are uncommitted changes or conflicts in the working directory.
Use Case	To check for updates without integrating them.	To fetch and integrate updates in a single step
Merging	No automatic merge; must manually merge or rebase the fetched updates.	Automatically merges (or rebases if configured) updates after fetching them.
Command Output	Lists remote changes without modifying the working branch.	Applies changes to the working branch and may require conflict resolution.
Common Syntax	`git fetch <remote>` `(e.g., git fetch origin)`	`git pull <remote> <branch>` (e.g., `git pull origin main`)

2.3.3. Differences between fetch and status in git

	Git Fetch	**Git Status**
Purpose	Fetches updates from the remote repository to the local repository.	Shows the current state of the local working directory and staging area.
Effect on Files	Updates local metadata (remote-tracking branches) but does not change working directory or local branch.	Does not alter any files; just displays the current status.
Scope	Works with the remote repository to fetch updates.	Focuses on local repository and working directory.
Command Output	Lists changes, commits, or branches available on the remote but not yet merged locally.	Shows modified, staged, and untracked files, and whether branch is ahead or behind the remote.
Use Case	To check for new commits, branches, or tags on the remote repository before merging or pulling them.	To check the current status of local files and decide what to stage, commit, or push.
Common Syntax	`git fetch <remote>` (e.g., `git fetch origin`)	git status

2.4. What is branching in Git?

Branching allows the creation of separate lines of development within a project. Each branch is an independent version of the project where we can make changes without affecting the main branch until merge.

Command: `git branch <branch-name>`

2.5. What is a conflict in Git?

A conflict occurs when Git cannot automatically merge changes because different edits were made to the same part of a file in separate branches. Developers must resolve these conflicts manually by choosing which changes to keep.

2.6. What is merge in Git?

Merging integrates changes from one branch into another. For example, we can merge a feature branch into the main branch to incorporate the new changes.

Command: `git merge <branch-name>`

2.7. What is a remote in Git?

A remote is a version of a repository that is hosted on a server, such as GitHub, GitLab, or Bitbucket. It allows collaboration by enabling multiple users to push and pull changes.
Example: origin is the default name for a remote repository.

2.8. What is the difference between git fetch and git pull?

Git fetch:
Downloads updates from a remote repository but does not automatically integrate them into the local branch.

Command: `git fetch origin`.

git pull:
Fetches updates from a remote repository and automatically merges them into the current branch.

Command: `git pull origin main`.

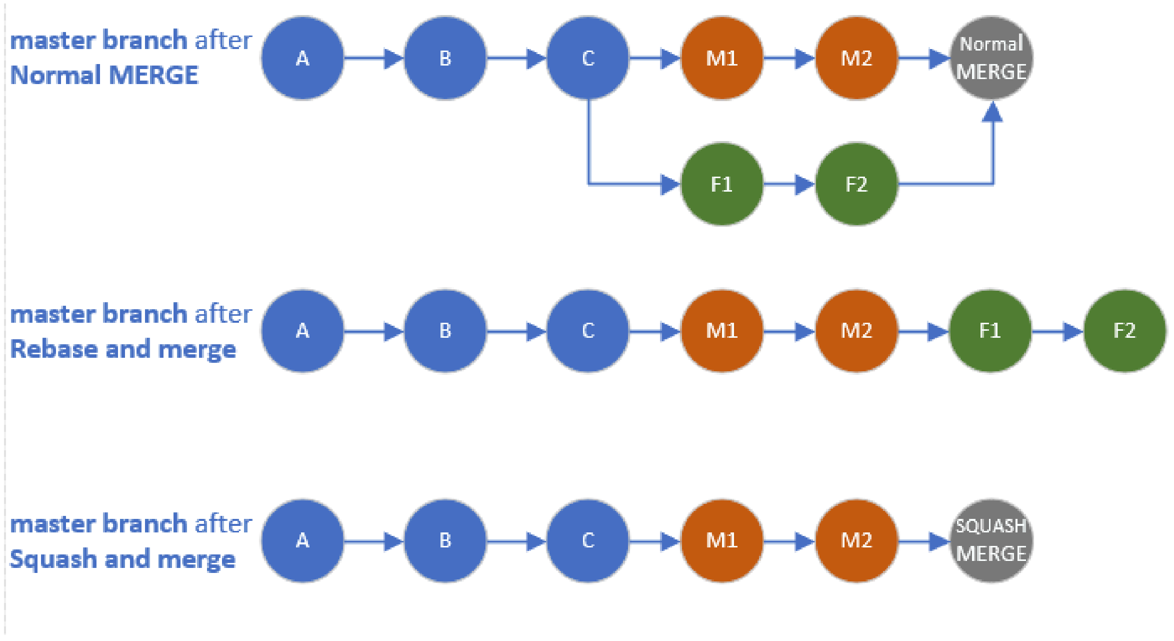
2.9. How do you revert a commit that has already been pushed and made public?

1. Use the git revert command to create a new commit that undoes the changes of the previous commit.
   Example: git revert <commit-hash>.

2. Push the revert commit to the remote repository: git push.

2.10. What is git rebase, git merge and squash

Rewrites history to produce a linear history without merge commits.


master branch after Normal MERGE
A
B
C
M1
M2
Normal MERGE
F1
F2
master branch after Rebase and merge
A
B
C
M1
M2
F1
F2
master branch after Squash and merge
A
B
C
M1
M2
SQUASH MERGE

1. Introduction to Automation Testing

(https://www.functionize.com/automated-testing)

Automation testing involves using specialized tools and frameworks to execute test cases, compare actual results with expected results, and report outcomes. Unlike manual testing, automation testing eliminates human intervention, making the process faster, more consistent, and efficient.

2. Benefits of Automation Testing

- **Time Efficiency**: Executes tests faster than manual testing.
- **Improved Accuracy**: Reduces the risk of human error.
- **Reusability**: Test scripts can be reused across multiple test cycles.
- **Cost-Effectiveness**: Saves effort and cost over the long term for repetitive tests.
- **Continuous Testing**: Supports continuous integration and delivery pipelines.
- **Scalability**: Can handle large volumes of test cases across different environments.

3. When to Automate Tests?

- **Tests are Repetitive**: Like regression tests or smoke tests.
- **Test Coverage Needs to be Maximized**: When many test cases must run in limited time.
- **Tests are Data-Driven**: When testing the same functionality with multiple datasets.
- **hi**: Ensure essential functionalities always work.

4. Which Test Cases to Automate?

- **Regression Tests**: Frequent code changes need validation.
- **Smoke Tests**: Ensures the basic functionalities work after builds.
- **Data-Driven Tests**: Verifying different inputs for the same functionality.
- **API Tests**: Testing backend APIs for functionality and integration.
- **Performance and Load Tests**: Simulating high-traffic scenarios.
- **High-Risk Areas**: Testing critical modules that affect overall functionality.

5. Types of Automation Tests

- **Business Critical test cases**

  Business Scenarios and all critical flows which needs to be perfectly tested and function.

- **Smoke Tests**

Smoke Tests are good candidates for Automation.

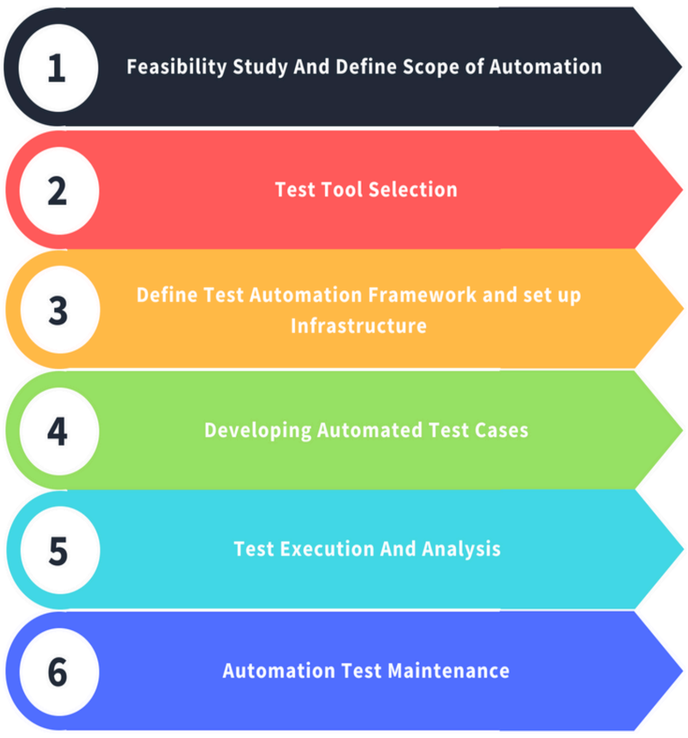
- **Test Cases that are very difficult to perform manually**

  Tasks that involve complex calculation, tedious steps are good candidates for automation.

- **Test Cases or Modules which are stable enough**

  If the application or module is not stable enough, it is not worth automation.

6. Automation Test Process


1 Feasibility Study And Define Scope of Automation
2 Test Tool Selection
3 Define Test Automation Framework and set up Infrastructure
4 Developing Automated Test Cases
5 Test Execution And Analysis
6 Automation Test Maintenance

7. When Test Automation is not the best solution?

- Exploratory Testing: Requires human intuition and creativity.
- Short-Term Projects: Automation setup may not be cost-effective.
- UI Changes Frequently: High maintenance cost for test scripts.
- Ad Hoc Testing: Unstructured testing doesn’t suit automation.
- Small Test Cases: If running tests manually is quicker and easier.

4.1. Introduction

An **automation tool** is software or a framework designed to automate repetitive tasks, workflows, or processes that would otherwise be performed manually. These tools are commonly used across various domains, including software testing, data processing, IT operations, and business processes, to increase efficiency, accuracy, and productivity.

4.2. Advantages of Automation tools

1. Enhanced Test Coverage
2. Reusability
3. Better Accuracy
4. Faster Feedback Cycles

4.3. Types of Tools

- **Functional Testing Tools**
  Functional testing tools verify that software behaves as expected by testing specific functions or features of an application.

Tool	Description
Selenium	● Parallel testing with Selenium Grid ● Supports Mac, Windows, and Linux ● Works with multiple browsers (Firefox, Chrome, Safari, etc.) ● Integrates with TestNG, JUnit, Jenkins ● Community support only ● Does not support mobile or desktop app testing
Appium	● Supports Java, Python, C#, and more ● Tests on real devices, emulators, and simulators ● Integrates with various testing frameworks and CI/CD tools ● Free and open-source

Katalon	● Record and playback for UI testing ● Integrates with JIRA, Jenkins, Azure DevOps ● Supports Groovy scripting ● Various testing methodologies (BDD, keyword-driven, etc.)

- **Mobile Testing Tools**
  Mobile testing tools are designed to automate testing for mobile apps across various devices, operating systems, and screen sizes.

Tools	Description
Appium	● Supports Java, Python, C#, and more ● Tests on real devices, emulators, and simulators ● Integrates with various testing frameworks and CI/CD tools ● Free and open-source
Espresso	● Tests user interfaces only ● Requires Java (JDK), Android Studio, and Gradle ● Does not support Kotlin or iOS testing

- **Continuous Integration Tools**

  Continuous integration tools help developers automate the process of integrating and testing code regularly, reducing integration issues.

Tools	**Description**
Jenkins	● Open-source automation server for CI/CD pipelines. ● Highly customizable with plugins. ● Supports building, deploying, and testing.

GitLab CI/CD	● Integrated within GitLab. ● Provides version control and CI/CD capabilities in one platform. ● Easy to set up and use.

4.4. Challenges to tools

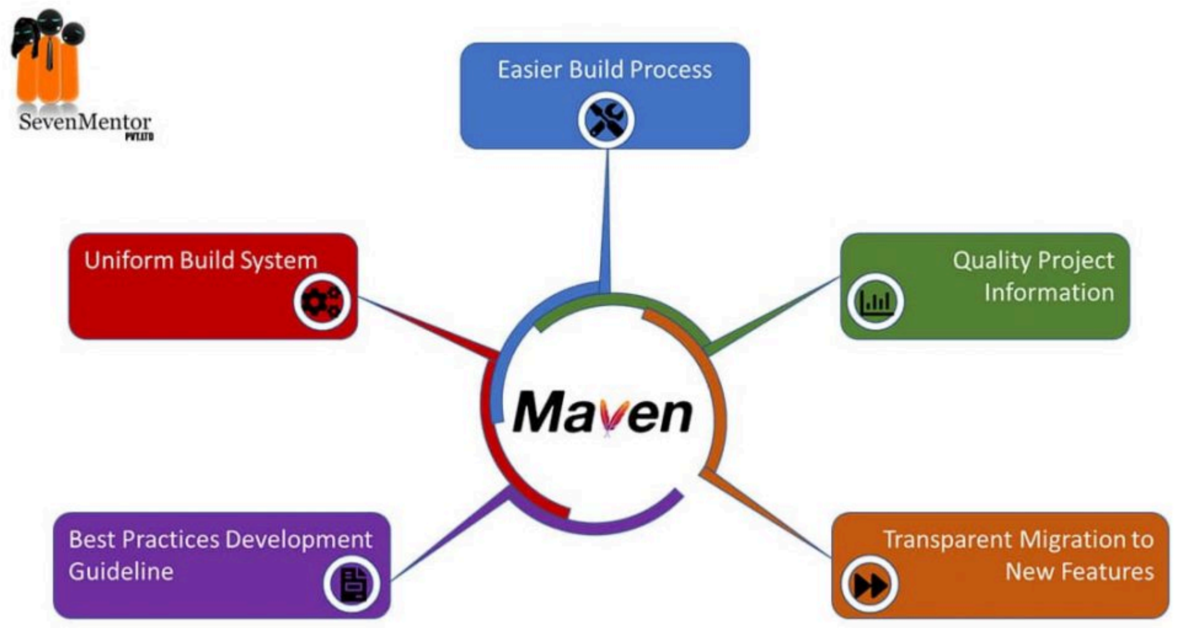
- Initial Setup costs
- Maintenance overhead
- Not a replacement for manual testing

5. Basics of maven

5.1. Introduction to maven

(https://www.sevenmentor.com/maven-and-its-features)

Maven is an open-source **build automation and project management tool** primarily used for **Java-based projects**. It helps developers automate the building, testing, and deploying of applications and manage project dependencies efficiently.


SevenMentor
Easier Build Process
Uniform Build System
Quality Project Information
Maven
Best Practices Development Guideline
Transparent Migration to New Features

5.1.1. Features of maven

Dependency Management	Maven simplifies the process of including external libraries (dependencies) by automatically downloading and integrating them from a central repository.
Standardized Project Structure	It enforces a uniform project layout, making it easier for developers to understand and collaborate on projects.
Extensibility with Plugins	Maven supports numerous plugins to add functionality, like testing frameworks, report generation, or code quality checks.
Build Automation	Maven automates repetitive tasks such as compiling code, running tests, and packaging the application into deployable formats (e.g., `.jar` or `.war` files).

5.2. Why do we need maven?

- **Dependency Management**
  - ❖ Why it's needed: Modern projects often rely on numerous external libraries or dependencies. Manually managing these can be time-consuming and error-prone.
  - ❖ What Maven does: Maven automatically downloads, updates, and resolves dependencies from a central repository, ensuring you always use the correct versions.

- **Standardized Project Structure**
  - ❖ Why it's needed: Different developers or teams may follow inconsistent folder structures, making collaboration and onboarding challenging.
  - ❖ What Maven does: Maven enforces a convention over configuration approach with a standard directory layout, simplifying the organization and understanding of projects.

- **Build Automation**
  - ❖ Why it's needed: Compiling code, running tests, packaging, and deploying software manually are repetitive and prone to errors.
  - ❖ What Maven does: With a single command (`mvn clean install`), Maven automates the build lifecycle, including compilation, testing, and packaging.

- **Integration with Tools**
  - ❖ Why it's needed: Integration with tools like Jenkins for CI/CD, IDEs, and other software can be cumbersome without a standardized approach.
  - ❖ What Maven does: Maven integrates well with development tools, making it easier to set up and maintain workflows.

5.3. Project Object Model (POM)

- What it is: The central configuration file (`pom.xml`) in a Maven project.
- Role:
  1. Defines project details (name, version, description).
  2. Manages dependencies, plugins, build configurations, and more.
- Structure: Written in XML format with tags like `<groupId>`, `<artifactId>`, `<dependencies>`, etc.
- Importance: Acts as the blueprint for the Maven build process.

6. Basics of Selenium

(https://www.youtube.com/watch?v=xhVs-h1ik00&list=PLgbf4L0WvebciyGW9bKfMLr-TGUp_u6K5)

6.1 What is selenium

- Selenium is a free (open-source) automated testing framework used to validate web applications across different browsers and platforms.
- We can use multiple programming languages like Java, C#, Python, etc to create Selenium Test Scripts.
- Testing done using the Selenium testing tool is usually referred to as Selenium Testing

6.2 Key features

- **Cross-Browser Support:** Works with major browsers like Chrome, Firefox, Safari, and Edge.
- **Cross-Platform Compatibility**: Can run tests on Windows, macOS, and Linux.
- **Multiple Language Support**: Write scripts in Java, Python, C#, JavaScript, Ruby, and more.
- **Integration**: Easily integrates with tools like Maven, Jenkins, and CI/CD pipelines

6.3 Selenium Components

- **Selenium IDE**: A record-and-playback tool for creating quick tests. The IDE records the user actions on the browser and exports them as a reusable script in different languages such as Java, C# and JavaScript.

- **Selenium WebDriver**: A programming interface for driving the web browsers. It allows writing tests in different programming languages to interact with the web elements, simulate user interactions and perform assertions.

- **Selenium Grid**: Is a tool that is used for concurrent execution of test cases on different browsers, machines, and operating systems simultaneously. It allows parallel test execution making it faster to run large test suites.

- **Selenium RC (Deprecated)**: The original version of Selenium; replaced by WebDriver.

6.4 How to set up PC for run selenium test script

- Install the JDK and note the installation directory.
- Set the JAVA_HOME environment variable
- Download Eclipse IDE for Java Developers
- Install and launch Eclipse
- Install maven and set the environment variables
- Create a maven project
- Add Selenium and TestNG dependencies to pom.xml file

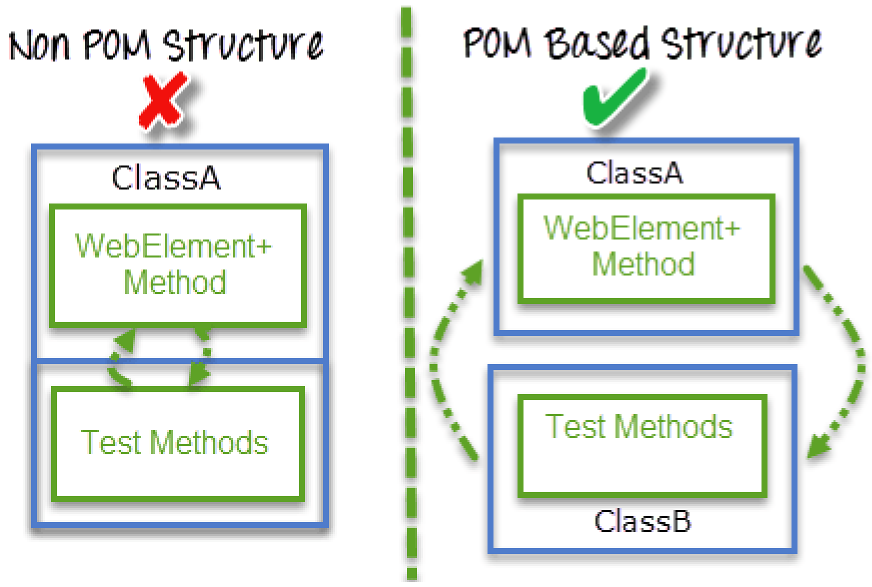
6.5 Locators in selenium

Locator	Description	Syntax (in Java)
ID	Identify the WebElement using the ID attribute	driver.findElement(By.id(“IdValue”));
Name	Identify the WebElement using the Name attribute	driver.findElement(By.name(“nameValue”));
ClassName	Use the Class attribute for identifying the object	driver.findElement(By.className(“classValue”));
LinkText	Use the text in hyperlinks to locate the WebElement	driver.findElement(By.linkText(“textofLink”));
Partial LinkText	Use a part of the text in hyperlinks to locate the desired WebElement	driver.findElement(By.partialLinkText(“PartialTextof Link”));
TagName	Use the TagName to locate the desired WebElement	driver.findElement(By.tagName(“htmlTag”));
CssSelector	CSS used to create style rules in web page is leveraged to locate the desired WebElement	driver.findElement(By.cssSelector(“cssValue”));
XPath	Use XPath to locate the WebElement	driver.findElement(By.xpath(“xpathValue”));

6. POM in Selenium

6.1. Introduction to POM

Page Object Model, also known as POM, is a that creates an object repository for storing all web elements. It helps reduce code duplication and improves test case maintenance.
In Page Object Model, consider each web page of an application as a class file. Each class file will contain only corresponding web page elements. Using these elements, testers can perform operations on the website under test.



6.2. Key Concepts use in POM

- **Page Classes**

  In POM, each web page or a specific section of a web page is represented by a dedicated class. This class acts as the interface between your test scripts and the actual web page. The page class contains locators for the web elements on the page (such as buttons, input fields, and links) and methods that represent actions a user can perform

- **Locators in POM**

  Iin POM, locators are used to identify and interact with web elements.

  Selenium offers several strategies to locate elements on a web page, such as:

  - ID: Locates an element by its unique ID.

- Class Name: Locates an element by its CSS class.
- XPath: Provides a flexible way to locate elements based on their structure in the DOM.
- CSS Selector: Locates elements using CSS rules.

- **Methods in POM**

The methods in a page class are designed to encapsulate interactions with the web elements. These methods should be intuitive, making it easier to write and understand your test cases. For example, methods might simulate user actions like filling out a form, clicking a button, or retrieving text from an element.

6.3. Implementation POM

Setting Up the Environment

Steps to set up your environment:

1. Install Selenium WebDriver: Ensure you have the WebDriver installed for your programming language (e.g., Java, Python). WebDriver allows Selenium to interact with the web browser.
2. Install a Testing Framework: Choose a testing framework like JUnit or TestNG (for Java) or PyTest (for Python). These frameworks help organize and run your tests effectively.
3. Download Browser WebDrivers: Obtain browser-specific WebDrivers, such as ChromeDriver or GeckoDriver for Firefox. These are necessary for running tests in different browsers.
4. Configure your IDE: Use an Integrated Development Environment (IDE) like IntelliJ IDEA, Eclipse, or Visual Studio Code to manage your project and keep it organized.

Creating Page Classes

1. Identify Web Elements: In each page class, you need to define the web elements that your test will interact with, such as input fields, buttons, and links. These elements are often located using attributes like ID, class name, or XPath.
2. Create Methods for User Actions: Define methods in the page classes to handle common interactions, such as entering text, clicking a button, or navigating to a particular section. These methods simplify how your test scripts interact with the page, improving the overall clarity of the tests.

Writing Test Cases with POM

1. Set Up a Test Class: Create a test class that will contain your test cases. This is typically done using your chosen testing framework (e.g., JUnit, TestNG, PyTest).
2. Use Page Classes in Tests: In the test class, instantiate the page classes that represent the web pages involved in the test. This allows you to use the methods from those page classes to perform actions like logging in, submitting forms, or clicking links.
3. Write Test Logic: Instead of focusing on low-level interactions (like clicking buttons or filling fields), your test cases can now focus purely on the test logic. For example, a login test case can simply call methods like “enter username” and “click login,” leaving the detailed page interactions to the page class.
4. Add Assertions: Add assertions to verify expected outcomes, such as checking that a login was successful or a specific page is displayed after an action. The testing framework you use will provide built-in assertions for validating these conditions.

6.4. Page Factory

Page Factory is a class provided by Selenium WebDriver to support Page Object Design patterns. In Page Factory, testers use @FindBy annotation. The initElements method is used to initialize web elements.

1. @FindBy: An annotation used in Page Factory to locate and declare web elements using different locators. Below is an example of declaring an element using @FindBy

   ```
   @FindBy(id="elementId") WebElement element;
   ```

2. initElements(): initElements is a static method in Page Factory class. Using the initElements method, one can initialize all the web elements located by @FindBy annotation.

6.5. Advantages of POM

- **Easy Maintenance**: POM is useful when there is a change in a UI element or a change in action. An example would be: a drop-down menu is changed to a radio button. In this case, POM helps to identify the page or screen to be modified. As every screen will have different Java files, this identification is necessary to make changes in the right files. This makes test cases easy to maintain and reduces errors.
- **Code Reusability:** As already discussed, all screens are independent. By using POM, one can use the test code for one screen, and reuse it in another test case. There is no need to rewrite code, thus saving time and effort.

- **Readability and Reliability of Scripts:** When all screens have independent java files, one can quickly identify actions performed on a particular screen by navigating through the java file. If a change must be made to a specific code section, it can be efficiently done without affecting other files.

6.6. Disadvantages POM

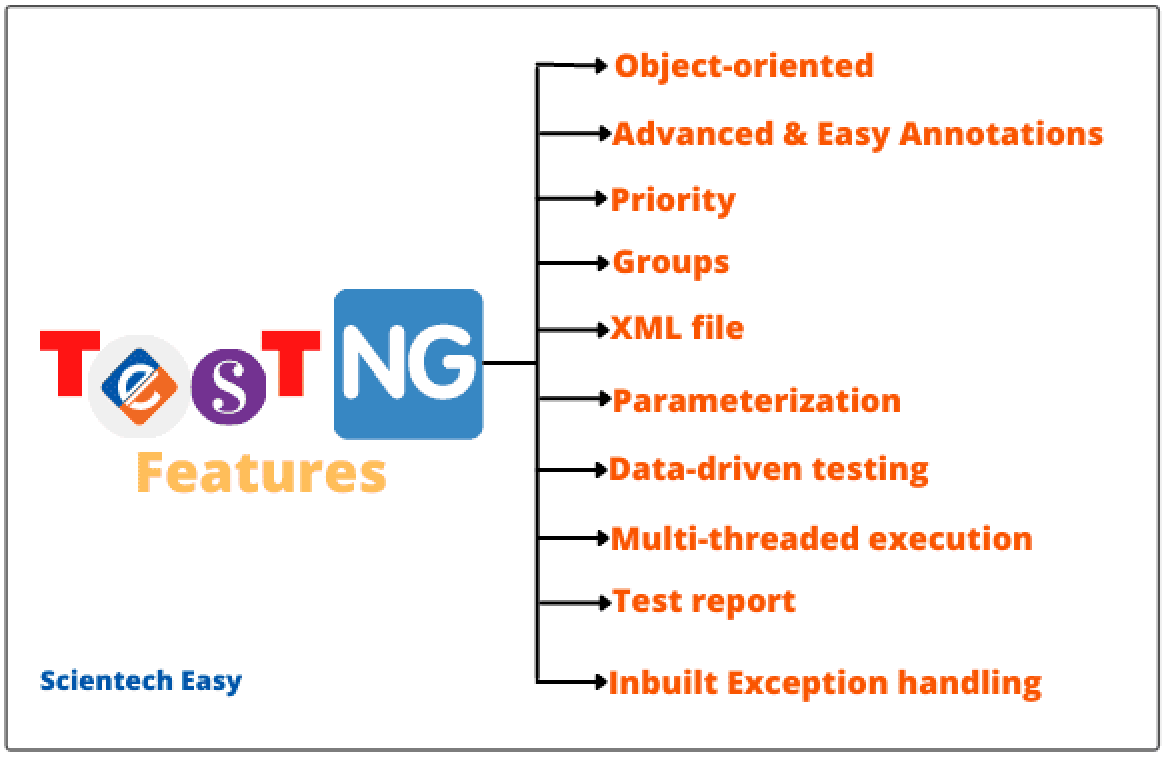
- Initial Setup Time

- Increased Complexity
- Shared File Risk
- Complex Page Objects

7. TestNG Framework

7.1. What is TestNG?

- TestNG stands for "Test Next Generation."
- It is a testing framework inspired by JUnit and NUnit.
- Designed to simplify testing in Java with flexibility and powerful features.
- Supports unit, functional, end-to-end, and integration testing.



7.2. Key Features

- Annotations for better code readability.
- Parallel test execution.
- Test dependency management.
- Built-in support for data-driven testing.
- Detailed reports for analysis.
- Integration with build tools (Maven, Gradle) and CI/CD pipelines.

7.3. Installation and Setup

7.3.1. Add TestNG to Your Project:

- For Maven: Add dependency to pom.xml:

```xml
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.8.0</version>
    <scope>test</scope>
</dependency>
```

- · For Gradle: Add to build.gradle:

```
testImplementation 'org.testng:testng:7.8.0'
```

7.3.2. Install TestNG Plugin (Optional):

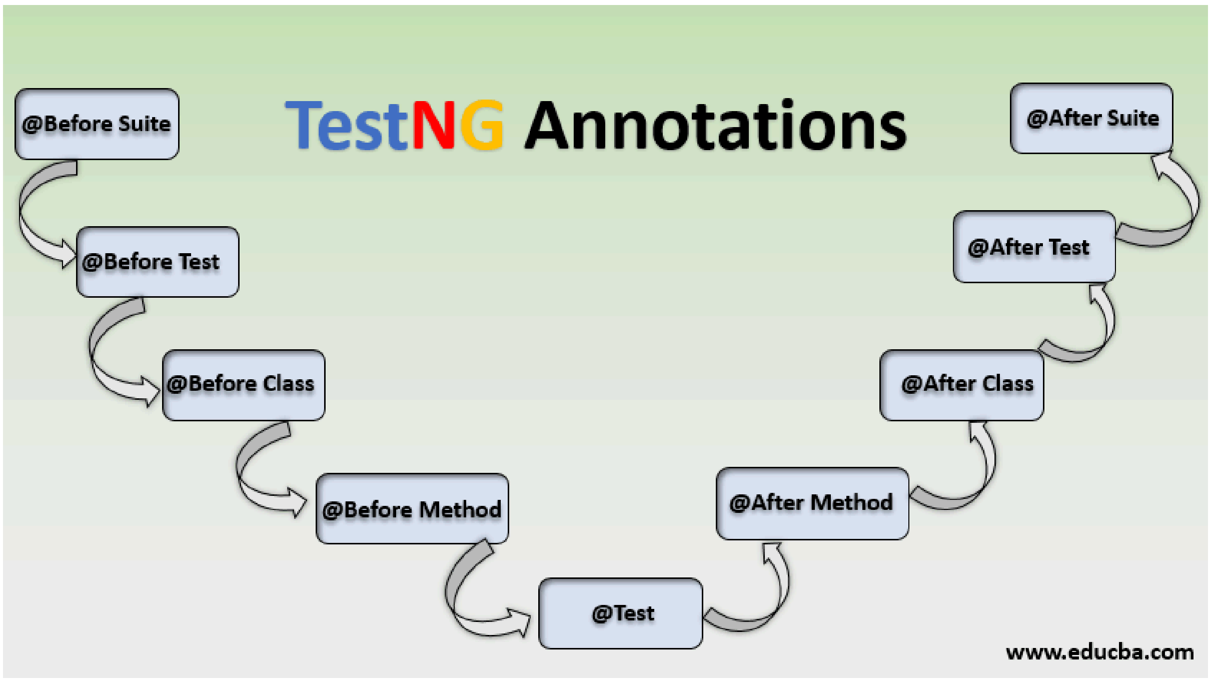
- For IDEs like Eclipse or IntelliJ IDEA.

Verify Setup:

- · Create a simple TestNG test and run it.

7.4. Annotations in TestNG

Annotation	Description
@Test	Marks a method as a test case.
@BeforeSuite	Executes before the entire suite.
@AfterSuite	Executes after the entire suite.
@BeforeMethod	Executes before each test method.
@AfterMethod	Executes after each test method.
@DataProvider	Provides data for data-driven testing.
@Parameters	Injects values into test methods from XML.



7.5. TestNG XML Configuration

- Central configuration file (testng.xml) to manage test execution.

```xml
<suite name="Test Suite">
    <test name="Test Case Group">
        <classes>
            <class name="com.example.MyTest" />
        </classes>
    </test>
</suite>
```

7.6. Benefits:

- Define test groups.
- Set test dependencies.
- Control parallel execution

7.7. Running TestNG Tests

- From IDE: Right-click and select "Run as TestNG Test."
- Using Command Line:

```
java -cp testng.jar org.testng.TestNG testng.xml
```

- With Maven:

```
mvn test
```

7.8. Reporting in TestNG

- Default Reports:
  - HTML and XML reports are automatically generated.
  - Location: test-output folder.
- Custom Reporting:
  - Use listeners like ITestListener or Reporter for customized reports.

7.9. Assertions in TestNG

- Assertions are used to validate test conditions.
- Examples:
  - Assert.assertEquals(actual, expected);
  - Assert.assertTrue(condition);
  - Assert.assertFalse(condition);
  - Assert.assertNull(object);

- Benefits:
  - Ensures test validation.
  - Stops execution if an assertion fails.

7.10. Advantages of TestNG

- Flexible annotations and configurations.
- Supports test prioritization and grouping.
- Parallel test execution reduces runtime.
- Data-driven testing with @DataProvider.
- Compatible with multiple tools and frameworks

8.Test Reporting Tools

8.1 What is a Test Reporting Tool?

- A test reporting tool is software that collects and presents data from test executions.
- Helps teams analyze test outcomes, track issues, and improve software quality.

Purpose

- To provide a comprehensive view of test results.
- To facilitate decision-making based on test data

8.2 Importance of Test Reporting Tools

- **Improved Visibility -** Offers clear insights into test progress and outcomes.
- **Efficiency -** Reduces manual effort in analyzing test results.
- **Collaboration -** Enhances communication between QA teams, developers, and stakeholders.
- **Quality Assurance -** Helps identify issues early and ensures better software quality.

8.3 Common Features

- **Data Aggregation**: Collects data from various test executions.
- **Visual Representation**: Generates charts, graphs, and dashboards for easy understanding.
- **Customizability**: Allows creating tailored reports based on project needs.
- **Integration**: Works seamlessly with CI/CD pipelines and testing frameworks.
- **Artifact Management**: Supports logs, screenshots, and error traces.

8.4 Types of Test Reporting Tools

Built-in-Reports

- Examples: TestNG Default Reports, JUnit Default Reports.
- Best for basic reporting needs.

Framework-Specific Reports

- Examples: Allure Reports, Extent Reports.
- Ideal for detailed, visually appealing reports.

Real-Time Reporting Tools:

- Example: Report Portal.
- Tracks distributed tests and provides real-time insights.

Custom Reports:

- Created using tools like Apache POI, FreeMarker, or Python libraries.
- Best for projects requiring specific formats or branding.

8.5 Test Reporting Tools

8.5.1 Extent Reports

A rich HTML reporting library for Selenium.

Provides visually appealing, interactive reports.

Use Cases - When detailed, visually interactive reports are needed.

Key Features

- Supports screenshots, logs, and steps.
- Categorizes tests based on tags.
- Integrates with TestNG, JUnit, and Cucumber

8.5.2 Allure Reports

A modern reporting tool for Selenium and other test frameworks.

Focuses on step-by-step execution details.

Use Cases - For comprehensive reporting with dashboards.

Key Features

- Supports attaching screenshots, logs, and other artifacts.
- Provides graphs and dashboards.
- Integrates with TestNG, JUnit, Pytest, and others.

8.5.3 TestNG Default Reports

Built-in reporting mechanism for TestNG.

Generates basic HTML and XML reports.

Use Cases - Quick reporting without external libraries.

Key Features

- Simple and lightweight.
- Includes test results, failures, and skipped tests.
- Can be customized using listeners (e.g., IReporter).

8.5.4 Report Portal

A real-time reporting and analytics tool.

Supports distributed test execution tracking.

Use Cases - For large-scale testing environments with CI/CD pipelines.

Key Features

- Provides dashboards and analytics.
- Integrates with CI/CD pipelines and multiple test frameworks.
- Supports log aggregation and real-time updates.

8.6 Comparison Of Reporting Tools

Tool	Ease of Use	Customization	Integration	Visual Appeal
Extent Reports	High	Moderate	TestNG, Junit,Cucumber	High
Allure Reports	Moderate	High	Multiple Frameworks	High
TestNG Default	High	Low	TestNG Only	Low
Report Portal	Moderate	Moderate	CI/CD + multiple tools	High

8.7 Choosing the Right Tool

- **Project Size**: For small projects, default reports may suffice; for large projects, tools like Allure or Report Portal are better.

- **Integration Needs**: Extent Reports and Allure offer seamless integrations with popular frameworks.

- **Customization:** Use custom HTML reports when specific branding or structure is required.

- **Team Collaboration**: Report Portal is ideal for distributed teams.

9. Data-Driven Testing

9.1 Introduction to Data-Driven Testing

· A testing approach where test logic is separated from test data.

· Test scripts are executed multiple times with varying input data sets.

· Purpose: Validate application behavior across different inputs.

Key Principle: “Reuse the same script, vary the data.”

9.2 Benefits of Data-Driven Testing

- **Efficiency**: Reuse a single script for multiple data sets.
- **Scalability**: Add more test cases by updating test data.
- **Maintainability**: Centralized test data management.
- **Comprehensive Testing**: Cover a wide range of scenario

9.3 How It Works in Selenium

1. Prepare test data (e.g., Excel, CSV, database).
2. Read the test data using libraries (e.g., Apache POI, OpenCSV).
3. Parameterize the test script to accept dynamic inputs.
4. Run the test script iteratively for each data set.

9.4 Tools and Frameworks

- **TestNG**: Supports parameterization and data providers.
- **Apache POI**: Reads/writes Excel files.
- **JXL**: Lightweight library for Excel file handling.
- **OpenCSV**: Processes CSV files.
- **Database**: Use SQL queries to fetch test data.

9.5 Example Workflow

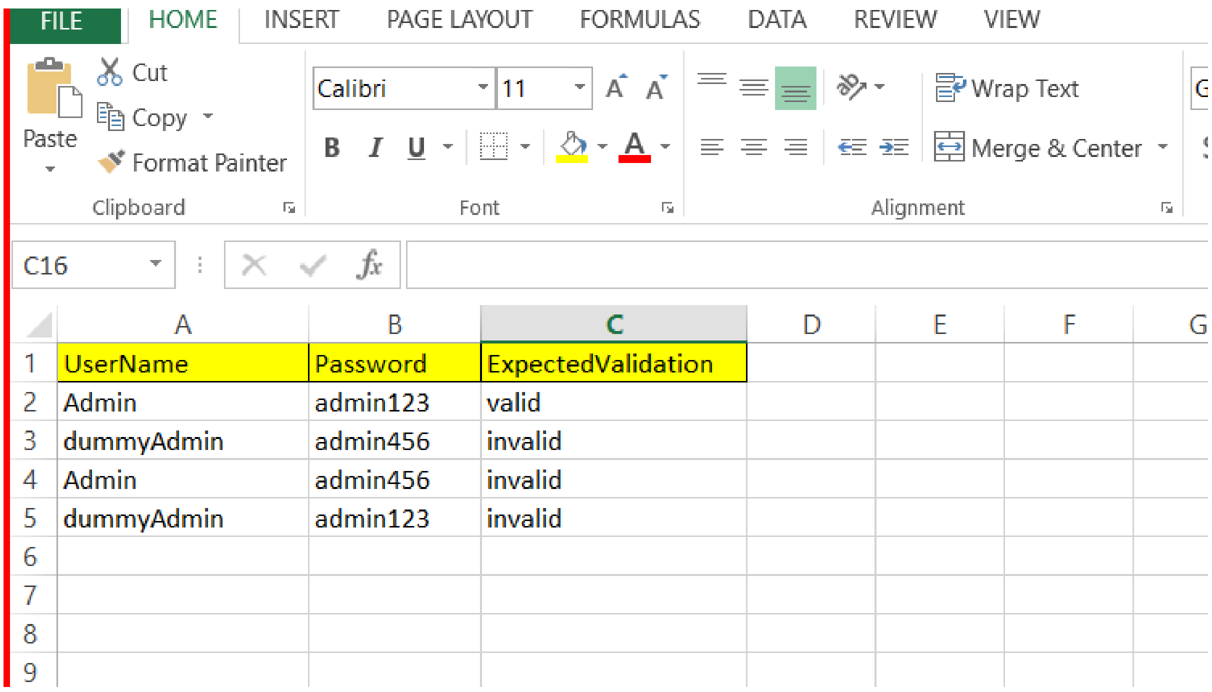
1. Create an Excel file with test data.
2. Use Apache POI to read the Excel file in your Selenium script.
3. Use TestNG’s DataProvider to pass data to the test script.
4. Execute the test for each row in the Excel file.

9.6 Code Demonstration

Dependencies to be added in pom.xml

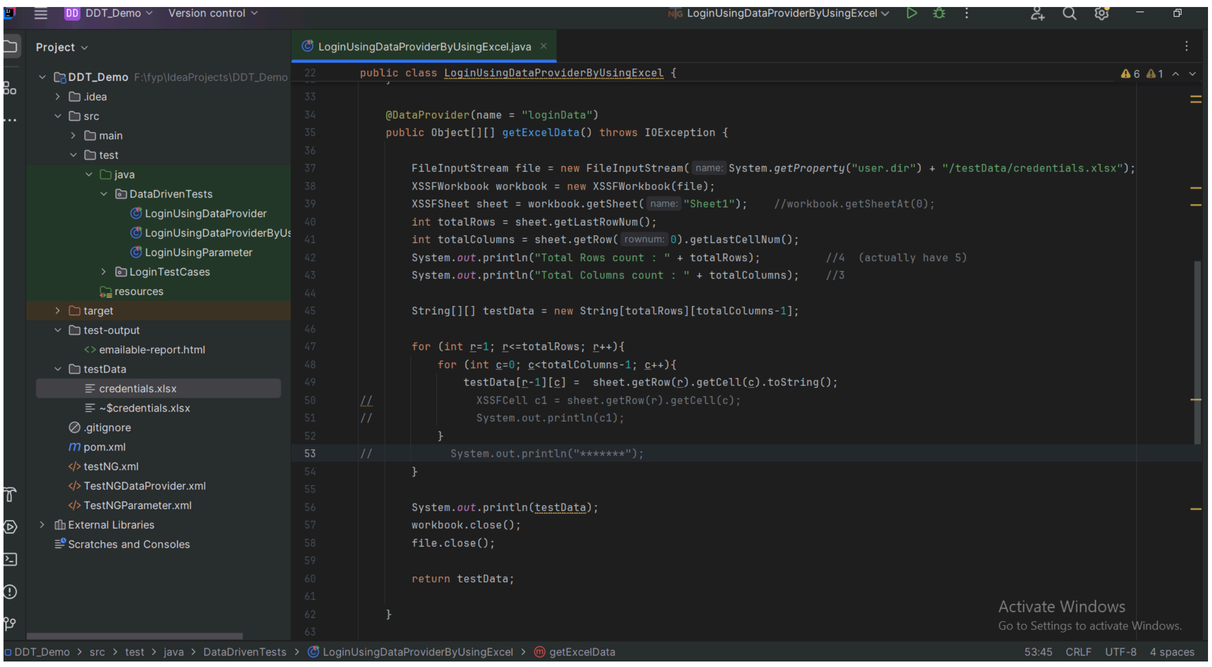
```xml
<!-- https://mvnrepository.com/artifact/org.apache.poi/poi -->

<dependency>

   <groupId>org.apache.poi</groupId>

   <artifactId>poi</artifactId>

   <version>5.2.5</version>

</dependency>


<!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->

<dependency>

   <groupId>org.apache.poi</groupId>

   <artifactId>poi-ooxml</artifactId>

   <version>5.3.0</version>

</dependency>
```

Excel File Sample



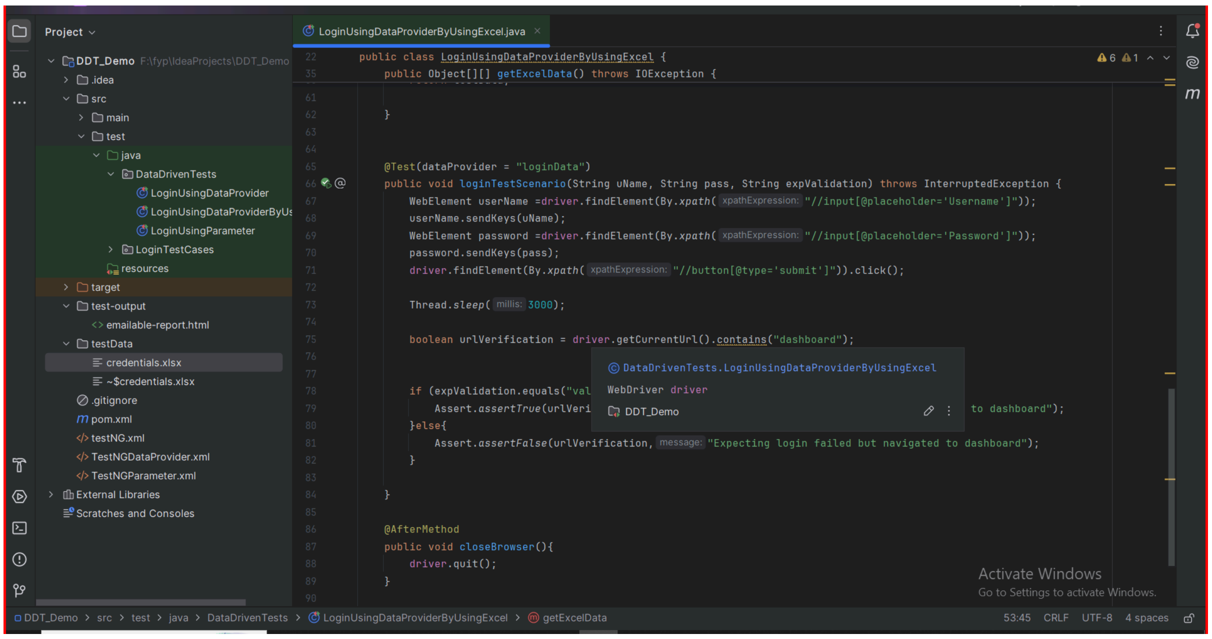
	A	B	C
1	UserName	Password	ExpectedValidation
2	Admin	admin123	valid
3	dummyAdmin	admin456	invalid
4	Admin	admin456	invalid
5	dummyAdmin	admin123	invalid

Read Excel File



```java
public class LoginUsingDataProviderByUsingExcel {

    @DataProvider(name = "loginData")
    public Object[][] getExcelData() throws IOException {

        FileInputStream file = new FileInputStream(System.getProperty("user.dir") + "/testData/credentials.xlsx");
        XSSFWorkbook workbook = new XSSFWorkbook(file);
        XSSFSheet sheet = workbook.getSheet("Sheet1");    //workbook.getSheetAt(0);
        int totalRows = sheet.getLastRowNum();
        int totalColumns = sheet.getRow(0).getLastCellNum();
        System.out.println("Total Rows count : " + totalRows);          //4  (actually have 5)
        System.out.println("Total Columns count : " + totalColumns);    //3

        String[][] testData = new String[totalRows][totalColumns-1];

        for (int r=1; r<=totalRows; r++){
            for (int c=0; c<totalColumns-1; c++){
                testData[r-1][c] =  sheet.getRow(r).getCell(c).toString();
//                XSSFCell c1 = sheet.getRow(r).getCell(c);
//                System.out.println(c1);
            }
//            System.out.println("*******");
        }

        System.out.println(testData);
        workbook.close();
        file.close();

        return testData;

    }
```

Selenium Test



```java
public class LoginUsingDataProviderByUsingExcel {
    public Object[][] getExcelData() throws IOException {

    }


    @Test(dataProvider = "loginData")
    public void loginTestScenario(String uName, String pass, String expValidation) throws InterruptedException {
        WebElement userName =driver.findElement(By.xpath( xpathExpression: "//input[@placeholder='Username']"));
        userName.sendKeys(uName);
        WebElement password =driver.findElement(By.xpath( xpathExpression: "//input[@placeholder='Password']"));
        password.sendKeys(pass);
        driver.findElement(By.xpath( xpathExpression: "//button[@type='submit']")).click();

        Thread.sleep( millis: 3000);

        boolean urlVerification = driver.getCurrentUrl().contains("dashboard");
                                  DataDrivenTests.LoginUsingDataProviderByUsingExcel
                                  WebDriver driver
        if (expValidation.equals("val
            Assert.assertTrue(urlVeri     DDT_Demo                              to dashboard");
        }else{
            Assert.assertFalse(urlVerification, message: "Expecting login failed but navigated to dashboard");
        }

    }

    @AfterMethod
    public void closeBrowser(){
        driver.quit();
    }
```

9.7 Challenges of DDT

- Managing large test data files.
- Test script complexity increases with dynamic data handling.
- Debugging issues when test data is incorrect.
- Performance impact with large data sets.

9.8 Best Practices of DDT

- Keep test data files clean and well-organized.
- Validate test data before running the tests.
- Use descriptive test case IDs for better tracking.
- Leverage frameworks (e.g., TestNG) for efficient data management.

10.Behaviour-Driven Development

(https://smartbear.com/product/testleft/features/behavior-driven-development/)

10.1 Introduction to Behavior-Driven Development

Behavior-Driven Development (BDD) is a collaborative and agile software development methodology that combines the principles of Test-Driven Development (TDD) with clear, user-focused communication.

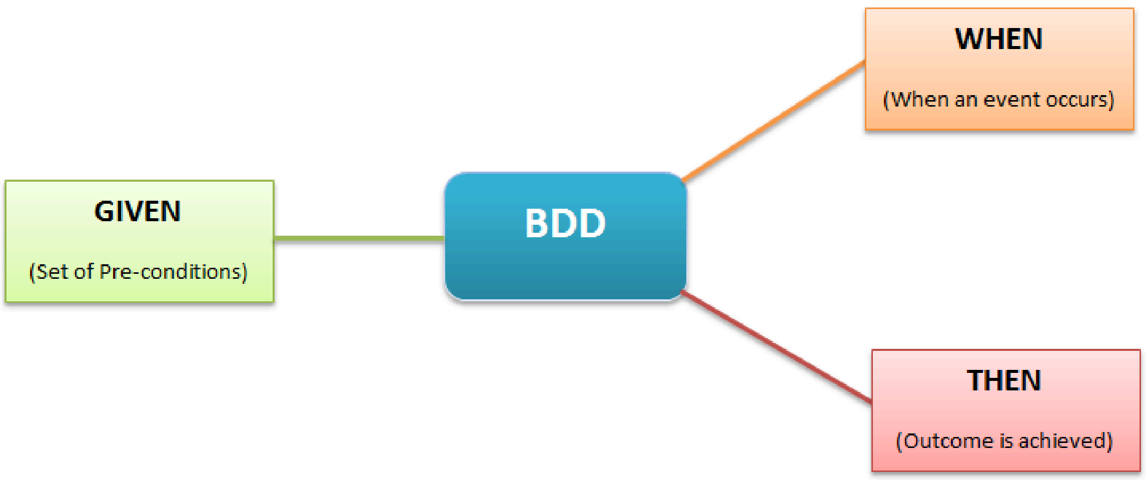
It emphasizes understanding the business goals of the software and aligning development efforts to meet those objectives.

10.2 Key Features of BDD

User-Centric Approach:
BDD starts by understanding what the end user expects the system to do. Scenarios are written in plain language to reflect real-world use cases.

Gherkin Syntax:
The Given-When-Then format allows scenarios to describe:

- **Given:** The initial setup or preconditions.
- **When:** The action performed by the user.
- **Then:** The expected outcome or result.



Collaboration:
BDD fosters collaboration among **developers, testers, and business stakeholders** to ensure shared understanding and alignment of goals.

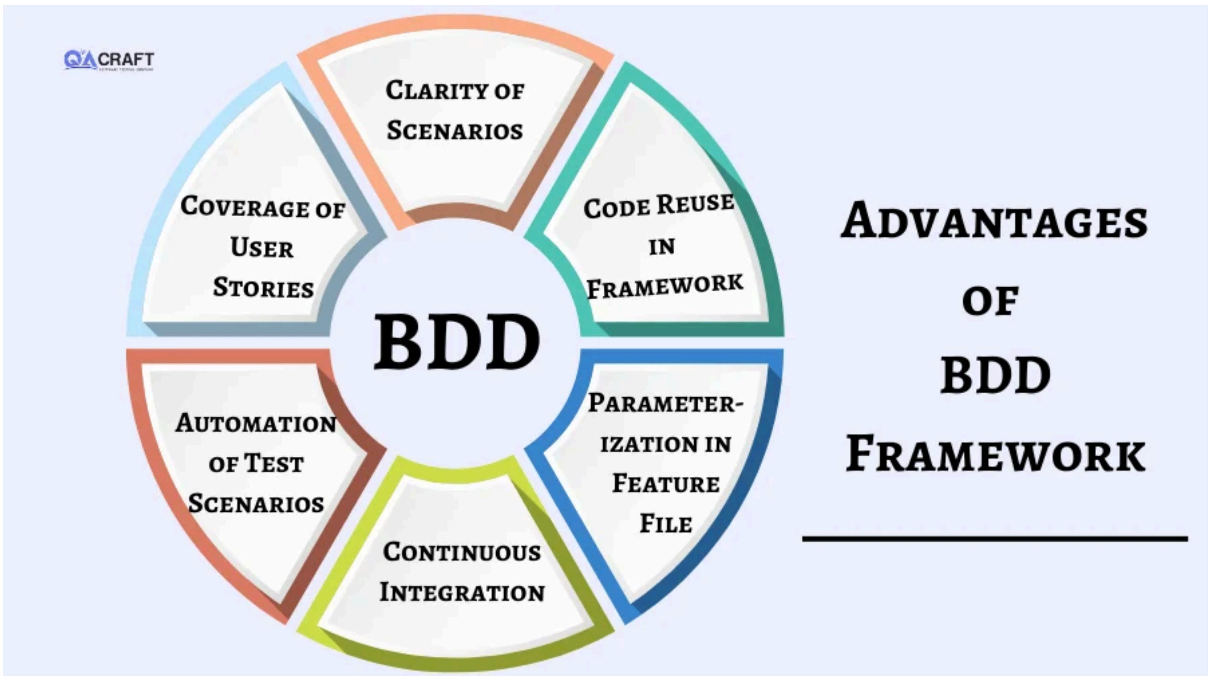
Living Documentation:
The scenarios serve as documentation that evolves with the system, making it easier to maintain and understand over time.

Automation of Tests:
Tools like **Cucumber, SpecFlow, JBehave, and Behave** integrate with programming languages to turn scenarios into automated tests, reducing manual testing efforts.

Improved Feedback Loop:
By involving all team members early and continuously, BDD identifies potential issues, misunderstandings, or gaps in requirements sooner.

10.3 Benefits of BDD

- · Encourages **shared ownership** of requirements and tests.
- · Reduces **ambiguities** in understanding business requirements.
- · Ensures the final product meets **user expectations**.
- · Simplifies **test maintenance** and aligns development with business needs.



10.4 When to Use BDD

- Collaboration is needed among developers, testers, and stakeholder
- The project involves frequent changes or evolving requirements.
- Ensuring the software meets business goals and user expectations is a priority.
- Requirements need to be written in plain, understandable language.

11. Xpath

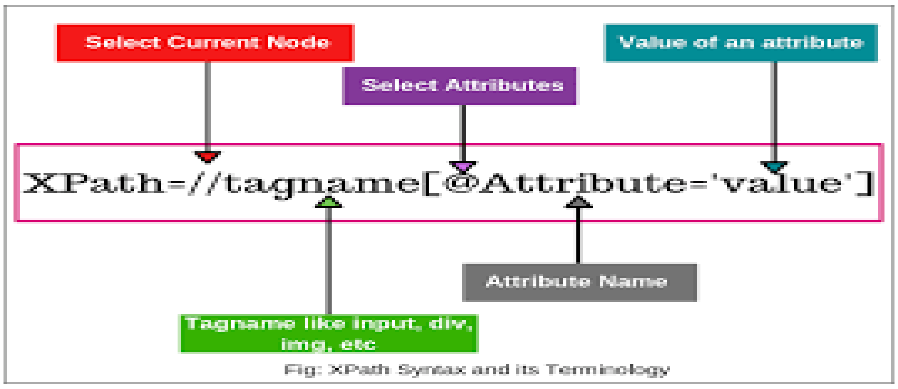
(https://learn-automation.com/how-to-write-dynamic-xpath-in-selenium/)

11.1. Why do we need XPath? Why not other locators?

- It allows locating elements even when other attributes like ID or name are missing or dynamically generated.
- It supports traversing DOM hierarchies, enabling finding child, parent, or sibling elements.

11.2. What is XPath?

- XPath (XML Path Language) is a query language for navigating and identifying elements within an XML or HTML document.
- It provides a syntax to locate elements based on their hierarchy, attributes, and text.


Select Current Node
Value of an attribute
Select Attributes
XPath=//tagname[@Attribute='value']
Attribute Name
Tagname like input, div, img, etc
Fig: XPath Syntax and its Terminology

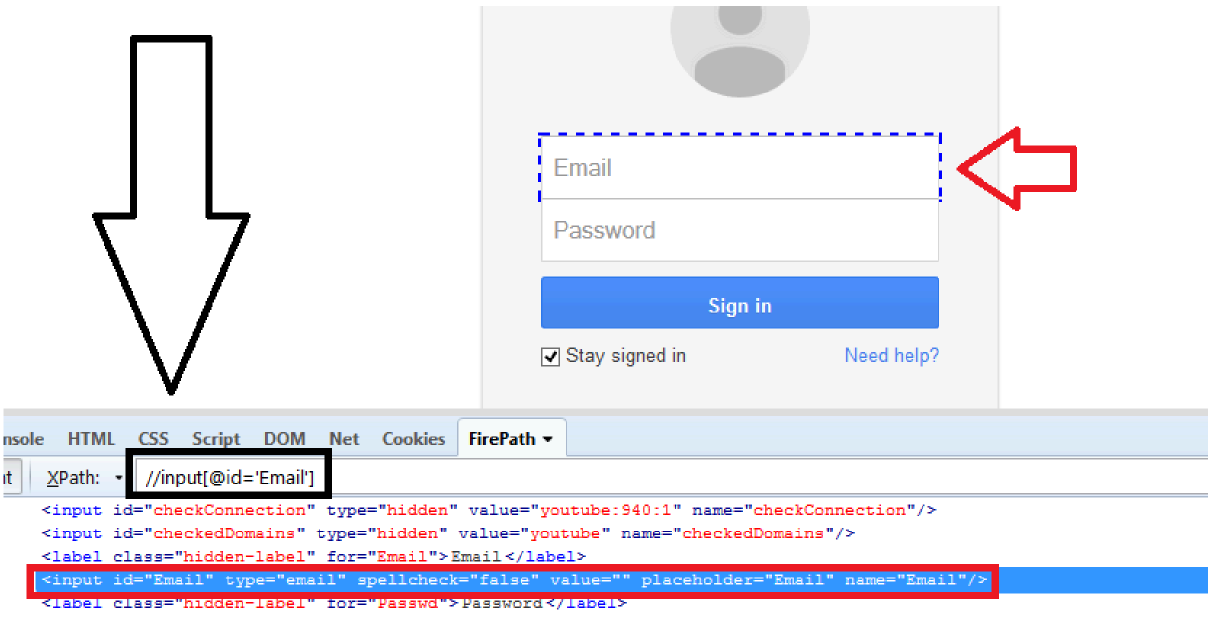
11.3. Types of XPath

Feature	Absolute XPath	Relative XPath
Syntax	Starts with `/` `/html/body/div/table/tr[1]/td[2]`	Starts with `//` `//div[@class='header']`
Dependence	Depends on the entire DOM structure	Independent of the full DOM hierarchy
Robustness	Breaks with slight changes in DOM	More robust and flexible

11.4. Identification Strategies Using XPath

A. Locating Elements with Respect to TagName and Attributes

Locating Elements with Known Attributes
Xpath - `//input[@id='username']`



```
//input[@id='Email']
<input id="checkConnection" type="hidden" value="youtube:940:1" name="checkConnection"/>
<input id="checkedDomains" type="hidden" value="youtube" name="checkedDomains"/>
<label class="hidden-label" for="Email">Email</label>
<input id="Email" type="email" spellcheck="false" value="" placeholder="Email" name="Email"/>
<label class="hidden-label" for="Passwd">Password</label>
```

1. Locating Elements with Known Tag Name and Attributes
   Xpath - `//button[@type='submit']`
2. Locating Elements with Multiple Attributes
   Xpath - `//input[@id='username' and @type='text']`

B. Locating Elements Using XPath Functions

Locating Elements with Known Visible Text (Exact Match)
Xpath - `//button[text()='Login']`

1. Locating Elements When Part of the Visible Text is Known (Partial Match)
   Xpath - `//button[contains(text(), 'Log')]`
2. Locating Elements When Starting Visible Text is Known
   Xpath - `//button[starts-with(text(), 'Log')]`
3. Locating Elements with Dynamic Attribute Values

Use `contains()` for dynamic values:
Xpath - `//input[contains(@id, 'user')]`

C. Locating XPath with `and` and `or` Operators

Using `and`:
Xpath - `//input[@id='username' and @type='text']`

Using `or`:
Xpath - `//input[@id='username' or @name='user']`

D. Locating Elements with Chained XPath

Combine multiple conditions to traverse the DOM:
Xpath - `//div[@class='header']//span[@class='logo']`

12. Test Framework

12.1. Introduction to Test Automation Framework

A testing framework is a set of guidelines or rules used for creating and designing test cases. A framework is a combination of practices and tools that are designed to help QA professionals test more efficiently.

12.2. Key Components

1. Test Data Management**:** Externalizes test data in formats like Excel or JSON to enable data-driven testing.

2. Test Script Repository: Houses automated test cases, organized and managed with tools like Selenium or Appium.

3. Object Repository: Central storage for UI locators to decouple scripts from UI changes.

4. Utilities & Helpers: Common functions (e.g., file handling, screenshots) to simplify testing.

5. Test Execution Engine**:** Manages script execution using tools like TestNG or JUnit.

6. Reporting & Logging: Provides detailed execution reports using tools like Allure.

7. CI/CD Integration: Automated testing as part of the build pipeline with Jenkins or GitHub Actions.

12.3. Types of Test Automation Frameworks

(https://smartbear.com/learn/automated-testing/test-automation-frameworks/)

- Linear Automation Framework
- Modular Based Testing Framework
- Library Architecture Testing Framework
- Data-Driven Framework
- Keyword-Driven Framework
- Hybrid Testing Framework

12.4. Benefits of Test Automation Frameworks

- Saving time
- Improved team performance and productivity
- Cost savings
- Test scaling

12.5. Steps to Design a Test Automation Framework

(https://vmsoftwarehouse.com/9-steps-to-create-a-test-automation-framework)

1. Define the scope and objectives.

Defining objectives: The first step in designing test automation is to decide which tests to automate. Criteria should be set to define these (e.g., faster feedback, better test coverage) depending on the organization's requirements.

2. Choose the right tools.

Evaluate your options: Consider factors such as project requirements, team experience, and technology stack. Determining the scope of testing also includes deciding on the test environments, platforms, and devices that will be used for automation.
The frameworks should be primarily:

- Compatible with the technology stack
- Easy to use and supportive of various test types
- Considerate of community support and documentation

3. Design the architecture of the framework.

The architecture includes configuring test environments, integrating with CI/CD, and selecting test automation tools. A well-designed architecture ensures the scalability, maintainability, and performance of automated tests.

4. Configure the environment.

The test environment should mimic the production environment. It includes hardware, software, network, and data configurations.

5. Implement the test framework.

- Create a project structure that can be transferred between projects and is convenient to use
- Implement utility classes for common activities such as logging, configuration handling, etc.
- Develop the basic components, such as:
    1. Configure configuration files for environment-specific settings.
    2. Implement initialization and removal of drivers (for Selenium/Appium).
    3. Create a base class that other test classes can extend.

6. Create and organize test cases.

- Write test scripts based on identified test scenarios.
- Use POM (Page Object Model) to extract web page details from test scripts.
- Test against multiple data sets

7. Implement reporting and logging.

- Integrate reporting tools (e.g., Allure, Extent Reports) to generate detailed test reports.
- Implement logging mechanisms to obtain detailed information about test runs (e.g., Log4j).

8. Perform testing and validation.

- Run tests locally to make sure they work as expected.
- Configure tests to run automatically as part of continuous CI/CD integration.
- Identify and fix any problems and optimize tests for stability and performance.

9. Maintain a testing framework.

- Update the framework and dependencies regularly.
- Perform periodic code refactoring to maintain quality.
- Constantly add new test cases to improve test coverage.

13. Jenkins

13.1. What is Jenkins?

- Jenkins is a Continuous Integration/Continuous Deployment (CI/CD) tool.
- It automates repetitive tasks in the software development process.
- It supports multiple plugins to integrate with various tools like Git, Maven, Docker, etc.

13.2. Key Features of Jenkins

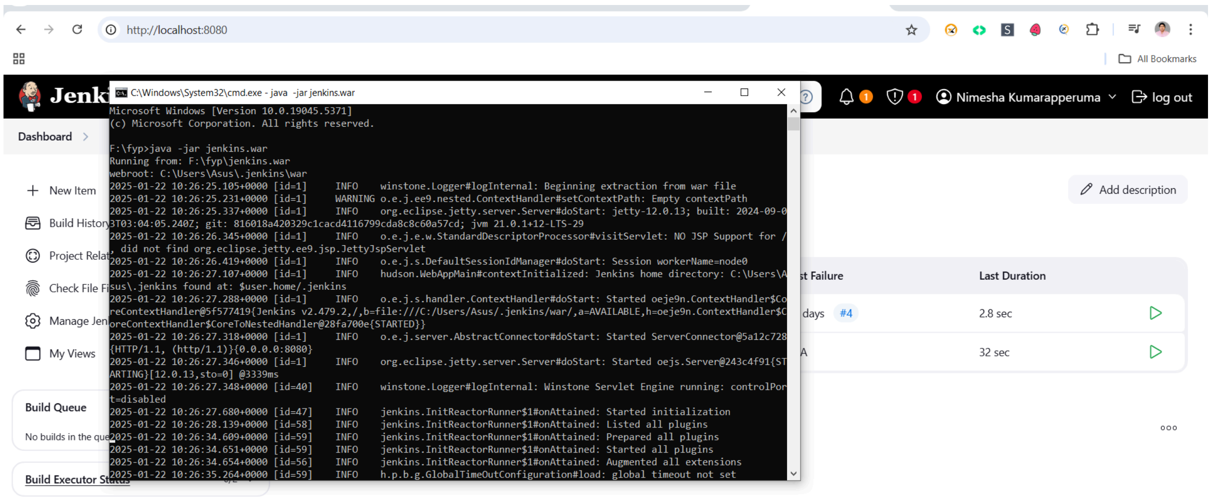
- Open-source and free to use.
- Cross-platform (supports Windows, Linux, macOS).
- Rich plugin ecosystem.
- Distributed builds with master-agent architecture.
- Easy integration with version control systems.

13.3.Installing Jenkins

- Install Java
- Download Jenkins war file

13.4. Access Jenkins

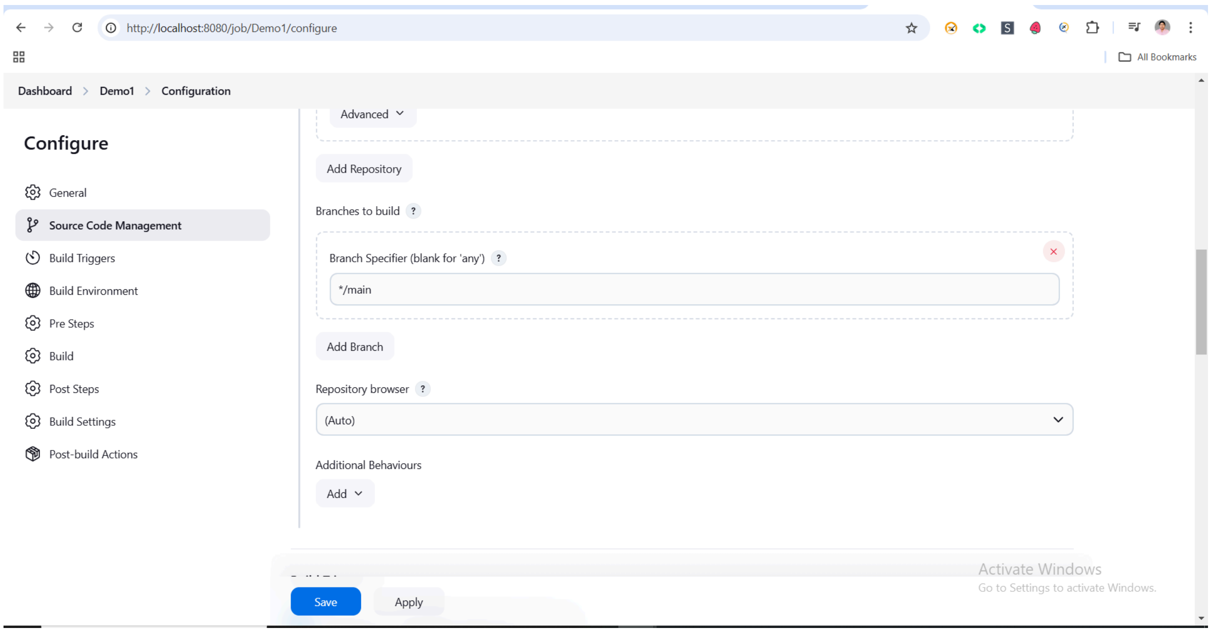
1. Run **Java -jar jenkins.war** in cmd of the folder that include the jenkins.war file
2. Open a web browser.
3. Go to `http://localhost:8080` (default port).



```
Microsoft Windows [Version 10.0.19045.5371]
(c) Microsoft Corporation. All rights reserved.

F:\fyp>java -jar jenkins.war
Running from: F:\fyp\jenkins.war
webroot: C:\Users\Asus\.jenkins\war
2025-01-22 10:26:25.105+0000 [id=1]     INFO    winstone.Logger#logInternal: Beginning extraction from war file
2025-01-22 10:26:25.231+0000 [id=1]     WARNING o.e.j.ee9.nested.ContextHandler#setContextPath: Empty contextPath
2025-01-22 10:26:25.337+0000 [id=1]     INFO    org.eclipse.jetty.server.Server#doStart: jetty-12.0.13; built: 2024-09-0
3T03:04:05.240Z; git: 816018a420329c1cacd4116799cda8c8c60a57cd; jvm 21.0.1+12-LTS-29
2025-01-22 10:26:26.345+0000 [id=1]     INFO    o.e.j.e.w.StandardDescriptorProcessor#visitServlet: NO JSP Support for /
, did not find org.eclipse.jetty.ee9.jsp.JettyJspServlet
2025-01-22 10:26:26.419+0000 [id=1]     INFO    o.e.j.s.DefaultSessionIdManager#doStart: Session workerName=node0
2025-01-22 10:26:27.107+0000 [id=1]     INFO    hudson.WebAppMain#contextInitialized: Jenkins home directory: C:\Users\A
sus\.jenkins found at: $user.home/.jenkins
2025-01-22 10:26:27.288+0000 [id=1]     INFO    o.e.j.s.handler.ContextHandler#doStart: Started oeje9n.ContextHandler$Co
reContextHandler@5f577419{Jenkins v2.479.2,/,b=file:///C:/Users/Asus/.jenkins/war/,a=AVAILABLE,h=oeje9n.ContextHandler$C
oreContextHandler$CoreToNestedHandler@28fa700e{STARTED}}
2025-01-22 10:26:27.318+0000 [id=1]     INFO    o.e.j.server.AbstractConnector#doStart: Started ServerConnector@5a12c728
{HTTP/1.1, (http/1.1)}{0.0.0.0:8080}
2025-01-22 10:26:27.346+0000 [id=1]     INFO    org.eclipse.jetty.server.Server#doStart: Started oejs.Server@243c4f91{ST
ARTING}[12.0.13,sto=0] @3339ms
2025-01-22 10:26:27.348+0000 [id=40]    INFO    winstone.Logger#logInternal: Winstone Servlet Engine running: controlPor
t=disabled
2025-01-22 10:26:27.680+0000 [id=47]    INFO    jenkins.InitReactorRunner$1#onAttained: Started initialization
2025-01-22 10:26:28.139+0000 [id=58]    INFO    jenkins.InitReactorRunner$1#onAttained: Listed all plugins
2025-01-22 10:26:34.609+0000 [id=59]    INFO    jenkins.InitReactorRunner$1#onAttained: Prepared all plugins
2025-01-22 10:26:34.651+0000 [id=59]    INFO    jenkins.InitReactorRunner$1#onAttained: Started all plugins
2025-01-22 10:26:34.654+0000 [id=56]    INFO    jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
2025-01-22 10:26:35.264+0000 [id=59]    INFO    h.p.b.g.GlobalTimeOutConfiguration#load: global timeout not set
```

13.5. Setting Up First Jenkins Job

1. Log in to Jenkins.
2. Click on **New Item**.
3. Enter a job name and select a job type (e.g., Maven Project).
4. Configure the job:
    - **Source Code Management**: Link your Git repository.
    - **Build Triggers**: Choose triggers like Poll SCM or Build Periodically.
5. Save and click **Build Now** to run your job.



http://localhost:8080/job/Demo1/configure

Dashboard > Demo1 > Configuration

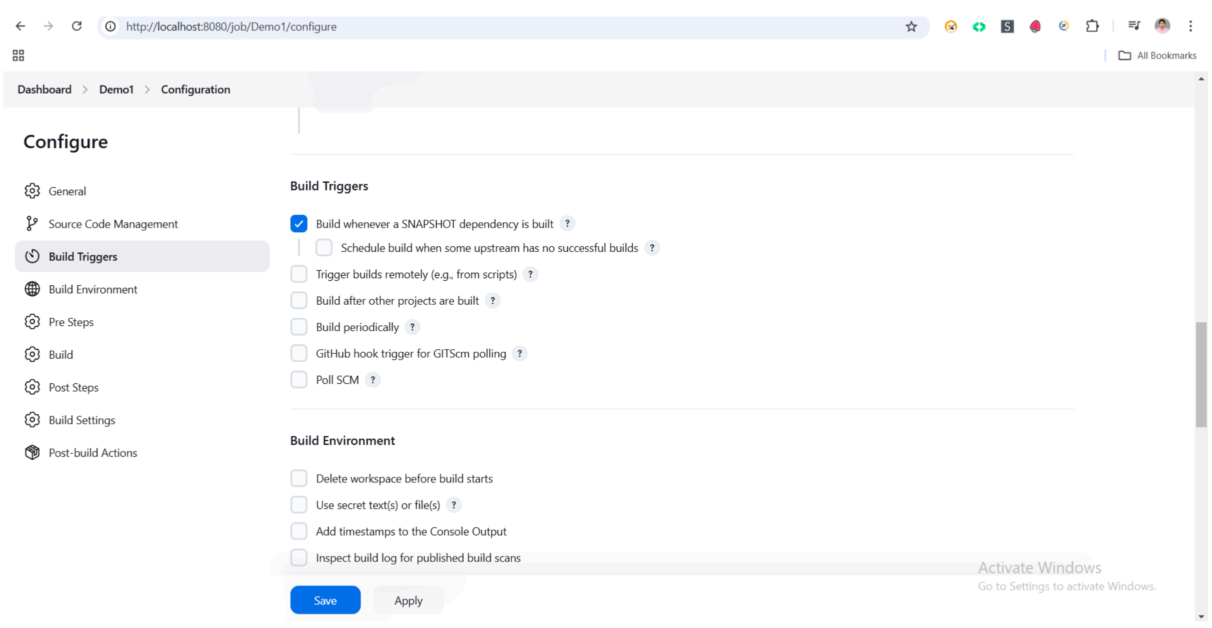
Configure

Branches to build

Branch Specifier (blank for 'any'): */main

Repository browser: (Auto)

Additional Behaviours



http://localhost:8080/job/Demo1/configure

Dashboard > Demo1 > Configuration

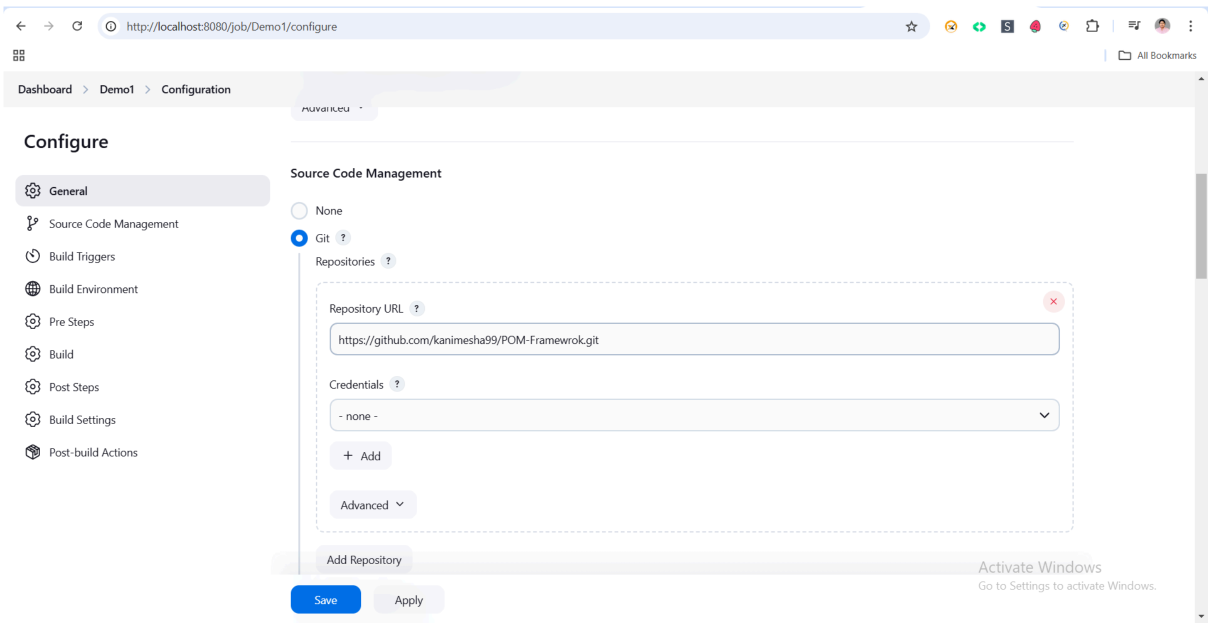
Configure

Build Triggers

- [✓] Build whenever a SNAPSHOT dependency is built
    - [ ] Schedule build when some upstream has no successful builds
- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] GitHub hook trigger for GITScm polling
- [ ] Poll SCM

Build Environment

- [ ] Delete workspace before build starts
- [ ] Use secret text(s) or file(s)
- [ ] Add timestamps to the Console Output
- [ ] Inspect build log for published build scans



http://localhost:8080/job/Demo1/configure

Dashboard > Demo1 > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Pre Steps
- Build
- Post Steps
- Build Settings
- Post-build Actions

Source Code Management

None [✓ Git]

Repositories

Repository URL: https://github.com/kanimesha99/POM-Framewrok.git

Credentials: - none -

Add

Advanced

Add Repository

Save Apply

Activate Windows
Go to Settings to activate Windows.