1. Different Flavours of Selenium

Selenium has evolved into a **suite of tools**, each serving a specific purpose in automation:

Flavour	Purpose	Example Scenario
Selenium IDE	Record and playback tool for quick automation scripts.	QA records a login test flow in the browser for rapid bug verification.
Selenium RC *(Retired)*	Legacy remote control tool for cross-browser automation using HTTP requests.	Used in older projects before WebDriver.
Selenium WebDriver	Modern, object-oriented API for automating browsers programmatically.	Writing Java code to automate an e-commerce checkout flow.
Selenium Grid	Run tests in parallel on different browsers & OS combinations.	Running the same test suite on Chrome (Windows) and Safari (macOS) simultaneously.
RemoteWebDriver	Run automation tests on a remote machine or cloud (e.g., BrowserStack, Selenium Grid).	Executing tests in a Dockerized Selenium Grid environment.

---

2. Selenium WebDriver – Introduction

Selenium WebDriver is a browser automation API that interacts directly with the browser without a middle server (unlike RC).

- **Key Features**:
    - Supports multiple programming languages (Java, Python, C#, Ruby, JS).
    - Controls browsers via browser-specific drivers.
    - Handles modern web technologies like HTML5, AJAX, React apps.

Example:

```
WebDriver driver = new ChromeDriver();
driver.get("https://example.com");
System.out.println(driver.getTitle());
driver.quit();
```

---

3. Selenium WebDriver Architecture

High-Level Flow:

1. **Selenium Test Script** (Java/Python/C#/etc.)
2. **Selenium Client Library** (e.g., `selenium-java` jar)
3. **JSON Wire Protocol / W3C WebDriver Protocol**
4. **Browser Driver** (e.g., `chromedriver.exe`)
5. **Browser** (e.g., Chrome, Firefox, Edge)

Working:

- The script sends commands (in JSON format) to the browser driver.
- The browser driver translates commands into native browser actions.
- The browser executes actions and returns responses.

Diagram:

```
[Your Code] → [Selenium Library] → [JSON/W3C Protocol] → [Browser Driver] →
[Browser]
```

---

4. Software Required for the Course

1. **JDK 1.8 or above** – For compiling Java code.
2. **Eclipse IDE for Java Developers** – Recommended IDE.
3. **Selenium WebDriver JARs** – To connect Java with Selenium.
4. **Browser Drivers** – ChromeDriver, GeckoDriver (Firefox), EdgeDriver.
5. **WebDriverManager** *(optional)* – Auto-downloads browser drivers.
6. **Demo Application for Testing** – Example: https://opensource-demo.orangehrmlive.com

---

5. Installations and Pre-requisites

Step 1: Install **JDK**

- Download from: https://jdk.java.net/
- Verify:

```
java -version
javac -version
```

Step 2: Install **Eclipse IDE**

- Download from: https://www.eclipse.org/downloads/

Step 3: Download **Selenium JAR Files**

- From: https://www.selenium.dev/downloads/
- Add JARs to Eclipse project → `Right click Project → Properties → Java Build Path → Add External JARs`

Step 4: Download **Browser Drivers**

- Chrome: https://chromedriver.chromium.org/
- Firefox: https://github.com/mozilla/geckodriver/releases
- Edge: https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

---

6. Configuring Selenium WebDriver in Eclipse

1. Create **Java Project** in Eclipse.
2. Create a `lib` folder → Place Selenium JARs + Drivers.
3. Add JARs to **Build Path**.
4. Create a **Java Class** with `main` method.

---

7. Creating First Test Script in Selenium WebDriver

Example: Open Google and Search

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class GoogleSearchTest {
    public static void main(String[] args) {
        // Set driver path
        System.setProperty("webdriver.chrome.driver",
"C:\\drivers\\chromedriver.exe");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open Google
        driver.get("https://www.google.com");

        // Find Search Box and enter text
        WebElement searchBox = driver.findElement(By.name("q"));
        searchBox.sendKeys("Selenium WebDriver");
        searchBox.submit();

        // Print page title
        System.out.println("Page Title: " + driver.getTitle());

        driver.quit();
    }
}
```

8. Locators & Object Identification

Selenium Locators help find HTML elements.

1. By ID

```
driver.findElement(By.id("username")).sendKeys("admin");
```

Scenario: Login page where input fields have unique `id`.

---

2. By Name

```
driver.findElement(By.name("password")).sendKeys("admin123");
```

Scenario: Forms where elements are identified by `name` attribute.

---

3. By Class Name

```
driver.findElement(By.className("btn-login")).click();
```

Scenario: Buttons or labels styled with a specific class.

---

4. By Tag Name

```
List<WebElement> links = driver.findElements(By.tagName("a"));
System.out.println("Total Links: " + links.size());
```

Scenario: Fetch all hyperlinks on a page.

---

5. By XPath

```
driver.findElement(By.xpath("//input[@id='username']")).sendKeys("admin");
```

Scenario: Complex DOM elements without IDs, needing relative or absolute paths.

---

6. By CSS Selector

```
driver.findElement(By.cssSelector("input#username")).sendKeys("admin");
```

Scenario: Faster and cleaner than XPath for styling-related selectors.

□ Example Test Script with All Locators

```
WebDriver driver = new ChromeDriver();
driver.get("https://opensource-demo.orangehrmlive.com");

driver.findElement(By.id("txtUsername")).sendKeys("Admin");
driver.findElement(By.name("txtPassword")).sendKeys("admin123");
driver.findElement(By.className("button")).click();
driver.findElement(By.tagName("a")).click();
driver.findElement(By.xpath("//a[text()='Logout']")).click();
driver.findElement(By.cssSelector("input[name='search']")).sendKeys("Test");
driver.quit();
```