

Selenium with Java: From Zero to Real Tests

You already know Java. This guide teaches only what is new: Selenium words, objects and methods, and how to use them to test a real web page. Every code example is Java and was run successfully.

IMPORTANT

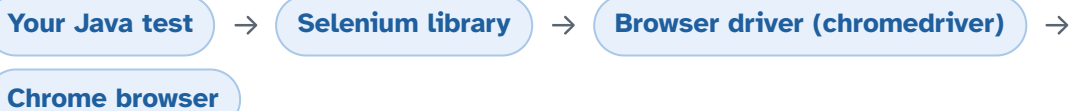
Red boxes and rows: learn these first. They are used in almost every test and asked in almost every interview.

GOOD TO KNOW

Blue boxes and rows: useful, but you can learn them later or only when needed.

1. What is Selenium and how does it work?

Selenium is a Java library (a set of classes in a jar file) that controls a real browser. Your Java code says "open this page", "type here", "click that button", and the browser does it. Then your code checks the result.



Selenium cannot talk to Chrome directly. A small program called the **browser driver** sits in the middle and translates the commands. In Selenium 4, a helper called **Selenium Manager** downloads the correct driver for you automatically.

Words you will hear

WORD	SIMPLE MEANING
WebDriver	The main Java interface. One WebDriver object = one browser you can control.
Driver / ChromeDriver	<code>WebDriver</code> is the interface. <code>ChromeDriver</code> is the class that opens Chrome. <code>FirefoxDriver</code> and <code>EdgeDriver</code> do the same for other browsers.
WebElement	One thing on the page: a button, text box, link, label. Your code holds it in a <code>WebElement</code> variable.

Locator (By)	The way to tell Selenium which element you mean: by id, name, CSS, XPath and so on.
Session	From <code>new ChromeDriver()</code> to <code>driver.quit()</code> . One browser window lifetime.
DOM	The page as a tree of elements. Locators search this tree.
Headless	Running the browser with no visible window. Faster, used on servers.
Selenium Grid	Runs tests on many machines and browsers at once. Good to know only.
Selenium IDE	A browser plugin that records clicks. Not used in real projects.

2. Setup in Java (Maven)

IMPORTANT Dependencies

Create a Maven project, then add these dependencies to `pom.xml`. Maven downloads the jars for you.

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>4.27.0</version>
</dependency>
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.10.2</version>
</dependency>
```

What it does: `selenium-java` is Selenium itself. `testng` is the test framework that runs your tests and gives you `Assert`. Use a recent Selenium version if you see a newer one on Maven Central. A "CDP version" warning in the console is harmless.

Also install Google Chrome. Nothing else is needed, Selenium Manager gets the driver.

Your first test (complete)

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

public class FirstTest {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();           // 1. open Chrome
        driver.get("https://example.com");               // 2. go to a
        page
        String title = driver.getTitle();                // 3. read the
        page title
        System.out.println(title);
        driver.findElement(By.tagName("a")).click();      // 4. find a link
        and click it
        driver.quit();                                   // 5. close the
        browser
    }
}
```

What it does: Five steps that every Selenium test has: open the browser, go to a page, find elements and act on them, check something, close the browser.

IMPORTANT Why WebDriver on the left side?

`WebDriver driver = new ChromeDriver();` The variable type is the interface `WebDriver`, the object is `ChromeDriver`. This is Java polymorphism. To run on Firefox you only change the right side to `new FirefoxDriver()`. Everything else stays the same.

IMPORTANT Never forget quit()

Always call `driver.quit()`. If you forget it, browser windows stay open and eat memory. In a test class, put it in `@AfterMethod` so it runs even when the test fails.

3. All the main classes in one picture

CLASS / INTERFACE	USE

WebDriver	Controls the browser: open pages, find elements, switch windows.
WebElement	One element: click it, type in it, read its text.
By	Describes how to find an element (id, CSS, XPath...).
Select	Helper for <code><select></code> dropdowns.
Actions	Mouse and keyboard actions: hover, double click, drag and drop.
Alert	Browser popups (alert, confirm, prompt).
WebDriverWait + ExpectedConditions	Wait until something is true.
JavascriptExecutor	Run JavaScript inside the page.
Keys	Special keys: ENTER, TAB, BACK_SPACE...
ChromeOptions	Settings for Chrome: headless, window size, notifications off.

4. WebDriver methods (the browser)

These are called on the `driver` object. Red rows you will use daily.

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>get(String url)</code>	Open a web page and wait until it loads.	<code>driver.get("https://shop.com");</code>
IMPORTANT	<code>getTitle()</code>	Returns the page title.	<code>String t = driver.getTitle();</code>
IMPORTANT	<code>getCurrentUrl()</code>	Returns the current address.	<code>driver.getCurrentUrl();</code>
IMPORTANT	<code>findElement(By)</code>	Finds the first matching element. Throws <code>NoSuchElementException</code> if none.	<code>driver.findElement(By.id("email"));</code>

IMPORTANT	<code>findElements(By)</code>	Finds all matching elements as a <code>List</code> . Empty list if none (no error).	<code>List<WebElement></code> <code>rows =</code> <code>driver.findElements(By.tagName("tr"));</code>
IMPORTANT	<code>quit()</code>	Closes every window and ends the session.	<code>driver.quit();</code>
IMPORTANT	<code>close()</code>	Closes only the current window or tab.	<code>driver.close();</code>
IMPORTANT	<code>navigate().back()</code> <code>/ forward() /</code> <code>refresh() /</code> <code>to(url)</code>	Browser back, forward, reload, go to a URL.	<code>driver.navigate().back();</code>
IMPORTANT	<code>manage().window().maximize()</code>	Makes the window full size.	<code>driver.manage().window().maximize();</code>
IMPORTANT	<code>manage().timeouts().implicitlyWait(Duration)</code>	Global wait for finding elements (see waits section).	<code>driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));</code>
IMPORTANT	<code>switchTo().frame(...)</code>	Go inside an iframe. Use name, id, index or <code>WebElement</code> .	<code>driver.switchTo().frame("payFrame");</code>
IMPORTANT	<code>switchTo().defaultContent()</code>	Come back from a frame to the main page.	<code>driver.switchTo().defaultContent();</code>
IMPORTANT	<code>switchTo().window(String handle)</code>	Move to another window or tab.	<code>driver.switchTo().window(handle);</code>
IMPORTANT	<code>switchTo().alert()</code>	Take control of a browser popup.	<code>Alert a =</code> <code>driver.switchTo().alert();</code>
IMPORTANT	<code>getWindowHandle()</code> <code>/</code> <code>getWindowHandles()</code>	Id of the current window / ids of all open windows (a <code>Set<String></code>).	<code>Set<String> all =</code> <code>driver.getWindowHandles();</code>

GOOD TO KNOW	<code>getPageSource()</code>	The full HTML of the page as a String.	<code>driver.getPageSource();</code>
GOOD TO KNOW	<code>switchTo().newWindow(WindowType.TAB)</code>	Open a new tab and move to it.	<code>driver.switchTo().newWindow(WindowType.TAB);</code>
GOOD TO KNOW	<code>switchTo().parentFrame()</code>	Go up one frame level.	<code>driver.switchTo().parentFrame();</code>
GOOD TO KNOW	<code>manage().window().minimize() / fullscreen() / setSize()</code>	Other window controls.	<code>driver.manage().window().setSize(new Dimension(1280, 800));</code>
GOOD TO KNOW	<code>manage().timeouts().pageLoadTimeout(Duration)</code>	How long to wait for a page to load before failing.	
GOOD TO KNOW	<code>manage().getCookies() / addCookie() / deleteAllCookies()</code>	Read, add and clear cookies. <code>deleteAllCookies()</code> gives a fresh start.	<code>driver.manage().deleteAllCookies();</code>

IMPORTANT `close()` vs `quit()`

`close()` closes one window. `quit()` closes everything. Interviewers ask this difference very often.

IMPORTANT `findElement` vs `findElements`

`findElement` gives one element and fails if none is found. `findElements` gives a list and never fails. Use `findElements(...).size() == 0` to check that something is NOT on the page.

5. Locators: how to find an element

A web page is HTML. Each element has attributes like `id`, `name`, `class`. To click a button, you first find it with a locator from the `By` class.

```
<input id="email" name="email" type="email" class="form-control"
placeholder="you@example.com">
<button type="submit" class="btn">Login</button>
```

What it does: Example HTML we will find elements in.

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>By.id("email")</code>	Finds by the id attribute. Fastest and best when the id is unique and stable.	<code>driver.findElement(By.id("email"))</code>
IMPORTANT	<code>By.cssSelector("input#email")</code>	Finds with a CSS selector. Fast and flexible.	<code>By.cssSelector("button[type='submit']")</code>
IMPORTANT	<code>By.xpath("//button[text()='Login']")</code>	Finds with XPath. Most powerful: can use text and parent / child.	<code>By.xpath("//input[@id='email']")</code>
IMPORTANT	<code>By.name("email")</code>	Finds by the name attribute.	<code>By.name("email")</code>
IMPORTANT	<code>By.linkText("Forgot password?")</code>	Finds a link by its full visible text.	<code>By.linkText("Sign up")</code>
GOOD TO KNOW	<code>By.partialLinkText("Forgot")</code>	Link text that contains the given part.	
GOOD TO KNOW	<code>By.className("btn")</code>	Finds by one class name. Many elements often share a class.	<code>By.className("btn")</code>
GOOD TO KNOW	<code>By.tagName("a")</code>	Finds by tag. Used to get lists like all links or all rows.	<code>driver.findElements(By.tagName("a"))</code>
GOOD TO KNOW	<code>RelativeLocator.with(By.tagName("input")).below(By.id("email"))</code>	Selenium 4: find by position (above, below, toLeftOf, toRightOf, near).	

CSS and XPath cheat sheet

WANT TO FIND	CSS SELECTOR	XPATH
By id	<code>#email</code>	<code>//*[@id='email']</code>
By class	<code>.btn</code>	<code>//*[@class='btn']</code>
Tag with attribute	<code>input[type='email']</code>	<code>//input[@type='email']</code>
By exact text	not possible	<code>//button[text()='Login']</code>
By part of text	not possible	<code>//button[contains(text(),'Log')]</code>
Part of an attribute	<code>div[class*='err']</code>	<code>//div[contains(@class,'err')]</code>
Starts with	<code>input[id^='user']</code>	<code>//input[starts-with(@id,'user')]</code>
Child inside parent	<code>#product-3 button</code>	<code>//div[@id='product-3']//button</code>
Second item of many	<code>li:nth-child(2)</code>	<code>(//li)[2]</code>
Next to a label	not possible	<code>//label[text()='Email']/following::input[1]</code>
Two conditions	<code>input[type='radio'] [value='UPI']</code>	<code>//input[@type='radio' and @value='UPI']</code>

IMPORTANT Locator rules

- Order of choice: **id** first, then **CSS**, then **XPath**.
- XPath that starts with `//` is **relative** and safe. XPath like `/html/body/div[2]/div[1]/button` is **absolute** and breaks when the page changes a little.
- Get a locator: right click the element in Chrome, click Inspect, read its attributes. Test your locator in the Console: `$x("//button[text()='Login']")` for XPath and `$("#email")` for CSS.
- If an id changes on every load (like `btn_8374`), use `starts-with()` or `contains()`.

6. WebElement methods (the things on the page)

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>click()</code>	Clicks the element.	<code>driver.findElement(By.id("loginBtn")).click();</code>
IMPORTANT	<code>sendKeys(CharSequence...)</code>	Types text. Also sends special keys. On a file input, sends the file path.	<code>box.sendKeys("hello");</code>
IMPORTANT	<code>clear()</code>	Removes the text of an input.	<code>box.clear();</code>
IMPORTANT	<code>getText()</code>	Returns the visible text of the element.	<code>String msg = el.getText();</code>
IMPORTANT	<code>getAttribute("name")</code>	Returns an attribute or property such as <code>value</code> , <code>href</code> , <code>class</code> , <code>type</code> . To read what a user typed, use <code>getAttribute("value")</code> .	<code>box.getAttribute("value");</code>
IMPORTANT	<code>isDisplayed()</code>	True if the element is visible on the page.	<code>el.isDisplayed();</code>
IMPORTANT	<code>isEnabled()</code>	True if the element is not disabled. Use for buttons.	<code>btn.isEnabled();</code>
IMPORTANT	<code>isSelected()</code>	True if a checkbox, radio button or option is selected.	<code>box.isSelected();</code>
GOOD TO KNOW	<code>getDomAttribute("name")</code>	Selenium 4: the attribute exactly as written in the HTML.	<code>el.getDomAttribute("data-state");</code>
GOOD TO KNOW	<code>getDomProperty("name")</code>	Selenium 4: the live property value of the element.	<code>el.getDomProperty("value");</code>
GOOD TO KNOW	<code>getTagName()</code>	The tag, such as <code>input</code> .	
GOOD TO KNOW	<code>getCssValue("color")</code>	A style value. Used for colour or font checks.	<code>el.getCssValue("color");</code>

GOOD TO KNOW	<code>getSize() / getLocation() / getRect()</code>	Width, height and position on the page.	
GOOD TO KNOW	<code>submit()</code>	Submits the form that contains the element.	
GOOD TO KNOW	<code>findElement(By) / findElements(By)</code>	Search only inside this element (a child search).	<code>card.findElement(By.tagName("button"));</code>
GOOD TO KNOW	<code>getScreenshotAs(OutputType.FILE)</code>	Screenshot of just this element.	

IMPORTANT Common mix-ups

- `getText()` gives text you can SEE. A hidden element returns an empty string.
- `getAttribute("value")` gives what is INSIDE a text box. `getText()` on a text box returns empty. This confuses everyone once.
- `isDisplayed()` throws `NoSuchElementException` if the element is not in the HTML at all. Use `findElements(...).size()` for that case.

7. Special elements

Dropdown: the Select class

Works only for the HTML `<select>` tag. Wrap the element in `Select`.

```
Select country = new Select(driver.findElement(By.id("country")));
country.selectByVisibleText("India");    // choose by the text the user sees
country.selectByValue("USA");            // choose by the value attribute
country.selectByIndex(0);                // choose by position, starts at 0
String chosen = country.getFirstSelectedOption().getText();
```

What it does: Selects an option in three ways and reads the chosen one.

METHOD	WHAT IT DOES	EXAMPLE

IMPORTANT	<code>selectByVisibleText(String)</code>	Choose by the visible text.
IMPORTANT	<code>selectByValue(String)</code>	Choose by the value attribute.
IMPORTANT	<code>getOptions()</code>	List of all options as <code>List<WebElement></code> . <code>select.getOptions().size();</code>
IMPORTANT	<code>getFirstSelectedOption()</code>	The selected option (single dropdown).
GOOD TO KNOW	<code>selectByIndex(int)</code>	Choose by position.
GOOD TO KNOW	<code>isMultiple()</code>	True for a multi-select list.
GOOD TO KNOW	<code>getAllSelectedOptions()</code>	All selected options of a multi-select.
GOOD TO KNOW	<code>deselectAll()</code> / <code>deselectByValue()</code> / <code>deselectByVisibleText()</code>	Only for multi-select lists.

Popups: the Alert interface

```
Alert alert = wait.until(ExpectedConditions.alertIsPresent()); // wait for the
popup
String text = alert.getText(); // read the message
alert.accept(); // press OK
// alert.dismiss(); // press Cancel
// alert.sendKeys("abc"); // type into a prompt box
```

What it does: Waits for a popup, reads it, and presses OK.

IMPORTANT Alert rule

While a popup is open, Selenium cannot do anything else on the page. You will get `UnhandledAlertException` . Always handle the popup first.

Mouse and keyboard: the Actions class

```
Actions actions = new Actions(driver);
actions.moveToElement(menu).perform();           // hover
actions.doubleClick(button).perform();           // double click
actions.contextClick(item).perform();             // right click
actions.dragAndDrop(source, target).perform();    // drag and drop
actions.sendKeys(Keys.ESCAPE).perform();          // press a key
```

What it does: Every Actions chain must end with perform(), or nothing happens.

Special keys

```
box.sendKeys("selenium", Keys.ENTER);           // type and press Enter
box.sendKeys(Keys.BACK_SPACE);                   // delete one character
box.sendKeys(Keys.chord(Keys.CONTROL, "a"));      // Ctrl + A (use
Keys.COMMAND on Mac)
```

What it does: Keys has ENTER, TAB, ESCAPE, BACK_SPACE, ARROW_UP, ARROW_DOWN and more.

JavascriptExecutor

```
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("arguments[0].scrollIntoView(true);", element); // scroll
                                                                    // to element
js.executeScript("arguments[0].click();", element);               // click
                                                                    // with JavaScript
js.executeScript("arguments[0].value = arguments[1];", dateBox, "2000-05-20");
                                                                    // set a value
Object title = js.executeScript("return document.title;");
```

What it does: Runs JavaScript inside the page. arguments[0] is the first value you pass after the script text.

Use it when normal Selenium cannot do the job: date pickers, hidden elements, scrolling, or a click that is blocked by another element. Try normal Selenium first.

Frames, windows and file upload

```
driver.switchTo().frame("payFrame");           // go inside an iframe
driver.switchTo().defaultContent();           // come back to the main page

String main = driver.getWindowHandle();       // remember the main window id
for (String h : driver.getWindowHandles()) {
    if (!h.equals(main)) driver.switchTo().window(h); // move to the new tab
}

driver.findElement(By.id("resume")).sendKeys("/Users/me/resume.pdf"); // file
upload
```

What it does: For an `<input type="file">` you do NOT click. You send the full path of the file as text.

Screenshot

```
File shot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);
Files.copy(shot.toPath(), Path.of("failure.png"));
```

What it does: Saves a picture of the page. Normally done when a test fails.

8. Waits: the most important topic

IMPORTANT Why we need waits

A page needs time to load. If your code looks for a button before it appears, you get `NoSuchElementException`. A **wait** tells Selenium to give the page a little time. Without waits, tests fail randomly.

WAIT	HOW IT WORKS	ADVICE
Implicit	One global rule: for every <code>findElement</code> , keep trying up to N seconds.	Simple, but do not mix with explicit waits.
Explicit (<code>WebDriverWait</code>)	Wait for ONE condition on ONE element (visible, clickable...).	Best choice. Use this.

Fluent	Explicit wait where you also set how often to check and which errors to ignore.	For special cases.
Thread.sleep	Stops for a fixed time.	Avoid. It always waits the full time.

```
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));
```

What it does: Implicit wait: for the whole test, find calls retry for up to 5 seconds.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
WebElement msg =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("msg")));
wait.until(ExpectedConditions.elementToBeClickable(By.id("payBtn"))).click();
```

What it does: Waits up to 10 seconds, but continues the moment the condition is true. If it never becomes true, a `TimeoutException` is thrown.

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	<code>visibilityOfElementLocated(By)</code>	Element is in the page and visible. Returns the element.	
IMPORTANT	<code>elementToBeClickable(By)</code>	Element is visible and enabled, ready to click.	
IMPORTANT	<code>presenceOfElementLocated(By)</code>	Element exists in the HTML (it may still be hidden).	
IMPORTANT	<code>alertIsPresent()</code>	A popup is open.	
GOOD TO KNOW	<code>invisibilityOfElementLocated(By)</code>	Element has gone (for example a loading spinner).	
GOOD TO KNOW	<code>textToBePresentInElementLocated(By, text)</code>	Element contains the given text.	
GOOD TO KNOW	<code>titleContains(text) / urlContains(text)</code>	Page title or URL has the text.	
GOOD TO KNOW	<code>frameToBeAvailableAndSwitchToIt(...)</code>	Frame is ready and Selenium switches into it.	

GOOD TO KNOW	<code>numberOfElementsToBe(By, n)</code>	Exactly n elements found.
GOOD TO KNOW	<code>stalenessOf(element)</code>	The old element is gone from the page.

```
Wait<WebDriver> fluent = new FluentWait<>(driver)
    .withTimeout(Duration.ofSeconds(20))
    .pollingEvery(Duration.ofMillis(500))
    .ignoring(NoSuchElementException.class);
WebElement el = fluent.until(d -> d.findElement(By.id("late")));
```

What it does: Fluent wait: check every 500 ms for up to 20 seconds, and ignore "not found" errors while waiting.

9. Common exceptions

	EXCEPTION	WHY IT HAPPENS	FIX
IMPORTANT	<code>NoSuchElementException</code>	Element not found. Wrong locator, page not loaded, or inside a frame.	Fix the locator, add a wait, switch to the frame.
IMPORTANT	<code>TimeoutException</code>	An explicit wait never became true.	Check condition and locator.
IMPORTANT	<code>StaleElementReferenceException</code>	The page refreshed and your saved element is old.	Find the element again.
IMPORTANT	<code>ElementClickInterceptedException</code>	Another element (banner, popup) covers the button.	Close the cover, scroll, or wait.
GOOD TO KNOW	<code>ElementNotInteractableException</code>	Element exists but is hidden or disabled.	Wait until visible or enabled.
GOOD TO KNOW	<code>NoAlertPresentException</code>	You switched to an alert but none is open.	Wait for <code>alertIsPresent()</code> .
GOOD TO KNOW	<code>UnhandledAlertException</code>	A popup is open and blocks the next action.	Accept or dismiss it first.

GOOD TO KNOW

SessionNotCreate
dException

Browser and driver versions
do not match.

Update Selenium or
Chrome.

10. TestNG: running and checking tests

Selenium only controls the browser. **TestNG** runs your methods as tests and tells you pass or fail. It works like JUnit with extra features.

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	@Test	Marks a method as a test.	
IMPORTANT	@BeforeMethod / @AfterMethod	Run before and after EACH test. Use for opening and closing the browser.	
IMPORTANT	@DataProvider	Gives many rows of data to one test method (data-driven testing).	@Test(dataProv ider = "logins")
GOOD TO KNOW	@BeforeClass / @AfterClass	Run once for the whole class.	
GOOD TO KNOW	@BeforeSuite / @AfterSuite	Run once for the whole run.	
GOOD TO KNOW	@Test(priority = 1, enabled = false, groups = "smoke")	Order, skipping and grouping of tests.	
GOOD TO KNOW	testng.xml	A file to choose which tests to run, run in parallel, or pass parameters.	

	METHOD	WHAT IT DOES	EXAMPLE
IMPORTANT	Assert.assertEquals actual, expected)	Fails if the two values differ. Note the order: actual first, expected second.	Assert.assertEquals als(driver.getTi tle(), "Home");

IMPORTANT	Assert.assertTrue(condition) / assertFalse	Fails if the condition is not true / not false.	Assert.assertTrue(btn.isEnabled());
IMPORTANT	Assert.assertEquals(actual, expected, "message")	The message shows when the test fails. Always add one for clear reports.	
GOOD TO KNOW	Assert.assertNull / assertNull / fail()	Other checks.	
GOOD TO KNOW	SoftAssert	Collects ALL failures and reports them together when you call <code>assertAll()</code> . A normal Assert stops at the first failure.	soft.assertAll();

The base class used in the three cases below

```
1  package selenium;
2
3  import org.openqa.selenium.WebDriver;
4  import org.openqa.selenium.chrome.ChromeDriver;
5  import org.openqa.selenium.chrome.ChromeOptions;
6  import org.openqa.selenium.support.ui.WebDriverWait;
7  import org.testng.annotations.AfterMethod;
8  import org.testng.annotations.BeforeMethod;
9
10 import java.time.Duration;
11
12 public class BaseTest {
13
14     protected WebDriver driver;
15     protected WebDriverWait wait;
16
17     @BeforeMethod
18     public void setUp() {
19         ChromeOptions options = new ChromeOptions();
20         if (Boolean.getBoolean("headless")) {
21             options.addArguments("--headless=new");    // run without
// showing a window
22         }
23         options.addArguments("--window-size=1280,900");
24         driver = new ChromeDriver(options);            // opens Chrome
25         wait = new WebDriverWait(driver, Duration.ofSeconds(5));
26     }
27
28     @AfterMethod(alwaysRun = true)
29     public void tearDown() {
30         if (driver != null) {
31             driver.quit();                            // closes the
// browser even if the test failed
32         }
33     }
34 }
```

LINES	WHAT THIS PART DOES
1-11	Package and imports: classes borrowed from Selenium, TestNG and Java.
12-13	The parent class of all test classes. Other classes say <code>extends BaseTest</code> and get the browser setup for free (inheritance).
14-16	<code>driver</code> controls the browser. <code>wait</code> is a ready explicit wait of 5 seconds. They are <code>protected</code> so child classes can use them.

17-18	TestNG runs this before every test method.
19-23	Settings for Chrome. With <code>-Dheadless=true</code> Chrome runs without a window. The window size is fixed so the layout is the same on every machine.
24	Starts Chrome. Selenium Manager downloads the right chromedriver if needed.
25-27	Creates the wait: up to 5 seconds for any condition.
28-34	Runs after every test, even if the test failed (<code>alwaysRun = true</code>). <code>quit()</code> closes the browser and ends the session.

11. Page Object Model (design pattern)

GOOD TO KNOW POM

Without POM, every test has locators inside it. If a locator changes, you edit many tests. In POM each page gets one class holding its locators and actions. Tests only call these actions. The three cases below keep things simple, so the locators and helper methods sit inside each test class. A real project would move them into page classes like this:

```
public class LoginPage {
    private final WebDriver driver;
    private final By email = By.id("email");
    private final By password = By.id("pass");
    private final By loginBtn = By.id("loginBtn");

    public LoginPage(WebDriver driver) { this.driver = driver; }

    public void loginAs(String user, String pass) {
        driver.findElement(email).sendKeys(user);
        driver.findElement(password).sendKeys(pass);
        driver.findElement(loginBtn).click();
    }
}

// in a test: new LoginPage(driver).loginAs("qa@test.com", "Test@1234");
```

What it does: Locators are private and hidden (encapsulation). If the id changes, you fix one line.

12. Three real automations

Click a case below to open it. In each case you will see: the practice page, where every element is, the manual test cases, the full Java code with line numbers, and the code explained line by line. **Read the manual test cases first, then the code.** That is how real automation is done: manual thinking first, code second.

IMPORTANT Before you start

- Open each practice page in Chrome and test it by hand first. The page links are inside each case below.
- `PAGE_URL` in each test already points to the live practice page on GitHub Pages, so you only need internet. To use your own copy, open the file from `docs/demo-pages/` and put its `file:///` path in `PAGE_URL`.
- Run with Maven: `mvn test -Dtest=PasswordTest`. Remove `-Dheadless=true` to watch the browser work.
- All three cases were run and passed (66 checks) on the pages as given.

▼ Case 1: Password validation (a full validation spec)

Boundary values, equivalence partitions, negative and state testing on one password form.

STEP 1: THE PRACTICE PAGE

Open the page, click around like a manual tester first, then look at the locators below.

Open demo-password.html

The test already points to this live page (`PAGE_URL`), so it runs from any computer with internet. The same file is in the repository at `docs/demo-pages/demo-password.html`.

IMPORTANT What you will learn from this case

- Data-driven tests (one method runs many times with different data)
- Boundary value analysis and equivalence partitioning in real code
- Checking one rule at a time
- Reading attributes and the enabled / disabled state of a button

STEP 2: WHERE EACH ELEMENT IS (LOCATORS)

ELEMENT	HTML ON THE PAGE	BY.ID	CSS SELECTOR	XPATH
Username box	<code><input id="username" type="text"></code>	<code>username</code>	<code>#username</code>	<code>//input[@id='username']</code>
Password box	<code><input id="password" type="password"></code>	<code>password</code>	<code>input[type='password']</code>	<code>//input[@id='password']</code>
Confirm password box	<code><input id="confirmPassword" type="password"></code>	<code>confirmPassword</code>	<code>#confirmPassword</code>	<code>//label[text()='Confirm password']/following::input[1]</code>
Show / Hide button	<code><button id="togglePwd">Show</button></code>	<code>togglePwd</code>	<code>#togglePwd</code>	<code>//button[text()='Show']</code>
One rule line	<code><li id="rule-length" data-state="pending"></code>	<code>rule-length</code>	<code>li[data-state='pass']</code>	<code>//li[@id='rule-length']</code>
Strength text	<code>Not entered</code>	<code>strength</code>	<code>#strength</code>	<code>//span[@id='strength']</code>
Error message	<code><div id="pwdError" class="error"></code>	<code>pwdError</code>	<code>#pwdError</code>	<code>//div[@id='pwdError']</code>
Create password button	<code><button type="submit" id="submitBtn" disabled></code>	<code>submitBtn</code>	<code>button[type='submit']</code>	<code>//button[@type='submit']</code>
Success message	<code><div id="successMsg" hidden></code>	<code>successMsg</code>	<code>#successMsg</code>	<code>//div[contains(text(),'created successfully')]</code>

STEP 3: MANUAL TEST CASES FIRST

Automation follows manual testing. We first decide what to test, and only then write code.

ID	TECHNIQUE USED	INPUT	EXPECTED RESULT
TC01	State check	Nothing typed	Button disabled, strength "Not entered", rules pending
TC02-05	Positive + boundary	Abcd@1234, Abc@1234 (8 chars), Abcdefgh@1234567 (16 chars), Str0ng@Passw0rd	All rules pass, Medium or Strong, button enabled, success message
TC06-07	Boundary value	7 characters, 17 characters	Only the length rule fails
TC08-11	Equivalence partition	No uppercase, no lowercase, no digit, no special character	Only that one rule fails
TC12	Error guessing	A space inside the password	No-space rule fails
TC13	Business rule	Username ubed, password MyUBED@123	No-username rule fails
TC14-16	Error guessing (hostile input)	SQL text, script text, only spaces	Rejected, button disabled, no popup
TC17-19	Negative: confirm password	Different, different only in case, empty	Error message, button disabled
TC20-21	State transition	Valid then invalid then valid; username changed later	Button enabled or disabled to match the data
TC22	UI behaviour	Click Show / Hide	Field type switches between password and text

STEP 4: THE CODE

File: `selenium/PasswordTest.java`

```
1 package selenium;
2
3 import org.openqa.selenium.By;
4 import org.openqa.selenium.WebElement;
5 import org.testng.Assert;
6 import org.testng.annotations.DataProvider;
7 import org.testng.annotations.Test;
8 import org.testng.asserts.SoftAssert;
9
10 public class PasswordTest extends BaseTest {
11
12     // Live practice page hosted on GitHub Pages (works from any
computer with internet)
13     private static final String PAGE_URL =
14         "https://ubedzz49.github.io/qa-interview-prep/demo-pages/demo-  
password.html";
15
16     private static final String[] ALL_RULES =
17         {"length", "upper", "lower", "number", "special",  
"nospace", "nouser"};
18
19     // ----- Locators: all in one place -----
20     private final By username = By.id("username");
21     private final By password = By.id("password");
22     private final By confirm = By.id("confirmPassword");
23     private final By submitBtn = By.id("submitBtn");
24     private final By toggleBtn = By.id("togglePwd");
25     private final By strength = By.id("strength");
26     private final By pwdError = By.id("pwdError");
27     private final By confirmError = By.id("confirmError");
28     private final By successMsg = By.id("successMsg");
29
30     // ----- Helper methods -----
31     private void open() {
32         driver.get(PAGE_URL);
33     }
34
35     private void type(By locator, String text) {
36         WebElement field = driver.findElement(locator);
37         field.clear();
38         if (!text.isEmpty()) {
39             field.sendKeys(text);
40         }
41     }
42
43     private void enter(String pwd, String confirmPwd, String user) {
44         type(username, user);
45         type(password, pwd);
46         type(confirm, confirmPwd);
```

```

46     }
47
48     private void enter(String pwd) {
49         enter(pwd, pwd, "");
50     }
51
52     private String ruleState(String rule) {
53         return driver.findElement(By.id("rule-" +
rule)).getDomAttribute("data-state");
54     }
55
56     private boolean buttonEnabled() {
57         return driver.findElement(submitBtn).isEnabled();
58     }
59
60     // ----- 1. Initial state -----
61     @Test
62     public void tc01_initialState() {
63         open();
64         Assert.assertFalse(buttonEnabled(), "Button must be disabled at
start");
65         Assert.assertEquals(driver.findElement(strength).getText(),
"Not entered");
66         for (String rule : ALL_RULES) {
67             Assert.assertEquals(ruleState(rule), "pending", "Rule " +
rule);
68         }
69         Assert.assertEquals(driver.findElement(pwdError).getText(),
"");
70
Assert.assertFalse(driver.findElement(successMsg).isDisplayed());
71     }
72
73     // ----- 2. Positive cases -----
74     @DataProvider(name = "positive")
75     public Object[][] positive() {
76         return new Object[][]{
77             // id,         technique,           password,
expected strength
78             {"TC02", "Valid typical password",      "Abcd@1234",
"Medium"},
79             {"TC03", "Boundary: 8 chars (min)",      "Abc@1234",
"Medium"},
80             {"TC04", "Boundary: 16 chars (max)",      "Abcdefgh@1234567", "Strong"},
81             {"TC05", "Strong password (12+ chars)",  "Str0ng@Passw0rd", "Strong"},
82         };
83     }

```

```

84
85     @Test(dataProvider = "positive")
86     public void positiveCases(String id, String technique, String pwd,
String expectedStrength) {
87         open();
88         enter(pwd);
89
90         SoftAssert soft = new SoftAssert();                // collect
ALL failures, then report
91         for (String rule : ALL_RULES) {
92             soft.assertEquals(ruleState(rule), "pass", id + " rule " +
rule);
93         }
94         soft.assertEquals(driver.findElement(strength).getText(),
expectedStrength, id + " strength");
95         soft.assertEquals(driver.findElement(pwdError).getText(), "",
id + " error text");
96         soft.assertTrue(buttonEnabled(), id + " button enabled");
97         soft.assertAll();
98
99         driver.findElement(submitBtn).click();
100         Assert.assertEquals(driver.findElement(successMsg).getText(),
"Password created successfully");
101     }
102
103     // ----- 3. Negative cases: break exactly ONE rule at a time -
-----
104     @DataProvider(name = "negative")
105     public Object[][] negative() {
106         return new Object[][]{
107             // id,         technique,
password,                username, broken rule
108             {"TC06", "Boundary: 7 chars (min - 1)",
"Abc@123",                "",        "length"},
109             {"TC07", "Boundary: 17 chars (max + 1)",
"Abcdefgh@12345678",     "",        "length"},
110             {"TC08", "Partition: no uppercase letter",
"abcd@1234",            "",        "upper"},
111             {"TC09", "Partition: no lowercase letter",
"ABCD@1234",            "",        "lower"},
112             {"TC10", "Partition: no number",
"Abcd@efgh",            "",        "number"},
113             {"TC11", "Partition: no special character",
"Abcd12345",            "",        "special"},
114             {"TC12", "Error guessing: space inside",                "Abcd
@1234",                "",        "nospace"},
115             {"TC13", "Business rule: contains username",
"MyUBED@123",            "ubed", "nouser"},
116         };

```

```

117     }
118
119     @Test(dataProvider = "negative")
120     public void negativeCases(String id, String technique, String pwd,
String user, String broken) {
121         open();
122         enter(pwd, pwd, user);
123
124         SoftAssert soft = new SoftAssert();
125         for (String rule : ALL_RULES) {
126             String expected = rule.equals(broken) ? "fail" : "pass";
127             soft.assertEquals(ruleState(rule), expected, id + " rule "
+ rule);
128         }
129         soft.assertEquals(driver.findElement(strength).getText(),
"Weak", id + " strength");
130         soft.assertEquals(driver.findElement(pwdError).getText(),
"Password does not meet all the rules");
131         soft.assertFalse(buttonEnabled(), id + " button must be
disabled");
132         soft.assertAll();
133     }
134
135     // ----- 4. Error guessing: hostile and strange input -----
--
136     @Test
137     public void tc14_sqlInjectionTextIsRejected() {
138         open();
139         enter("'" OR '1'='1");
140         Assert.assertFalse(buttonEnabled());
141
142         Assert.assertFalse(driver.findElement(successMsg).isDisplayed());
143     }
144
145     @Test
146     public void tc15_scriptTextShowsNoPopup() {
147         open();
148         enter("<script>alert(1)</script>");
149         Assert.assertFalse(buttonEnabled());
150         // if the text had run as a script, an alert would be open now
151         try {
152             driver.switchTo().alert();
153             Assert.fail("An unexpected popup appeared");
154         } catch (org.openqa.selenium.NoAlertPresentException expected)
{
155             // good: no popup
156         }
157     }

```

```

158     @Test
159     public void tc16_onlySpacesAreRejected() {
160         open();
161         enter("      ");
162         Assert.assertEquals(ruleState("nospace"), "fail");
163         Assert.assertFalse(buttonEnabled());
164     }
165
166     // ----- 5. Confirm password -----
167     @Test
168     public void tc17_mismatchShowsError() {
169         open();
170         enter("Abcd@1234", "Abcd@12345", "");
171         Assert.assertEquals(driver.findElement(confirmError).getText(),
172 "Passwords do not match");
173         Assert.assertFalse(buttonEnabled());
174     }
175
176     @Test
177     public void tc18_caseDifferenceIsMismatch() {
178         open();
179         enter("Abcd@1234", "abcd@1234", "");
180         Assert.assertEquals(driver.findElement(confirmError).getText(),
181 "Passwords do not match");
182     }
183
184     @Test
185     public void tc19_emptyConfirmKeepsButtonDisabled() {
186         open();
187         enter("Abcd@1234", "", "");
188         Assert.assertFalse(buttonEnabled());
189     }
190
191     // ----- 6. State transitions -----
192     @Test
193     public void tc20_validInvalidValidChangesButton() {
194         open();
195         enter("Abcd@1234");
196         Assert.assertTrue(buttonEnabled());
197
198         type(password, "abcd@1234"); // breaks
199         the uppercase rule
200         Assert.assertFalse(buttonEnabled());
201
202         type(password, "Abcd@1234");
203         Assert.assertTrue(buttonEnabled());
204     }
205
206     @Test

```

```

204     public void tc21_changingUsernameCanBreakValidPassword() {
205         open();
206         enter("Abcd@1234");
207         Assert.assertTrue(buttonEnabled());
208         type(username, "abcd");
209         Assert.assertEquals(ruleState("nouser"), "fail");
210         Assert.assertFalse(buttonEnabled());
211     }
212
213     // ----- 7. UI behaviour -----
214     @Test
215     public void tc22_showHideTogglesFieldType() {
216         open();
217         WebElement field = driver.findElement(password);
218         Assert.assertEquals(field.getDomAttribute("type"), "password");
219         driver.findElement(toggleBtn).click();
220         Assert.assertEquals(field.getDomAttribute("type"), "text");
221         Assert.assertEquals(driver.findElement(toggleBtn).getText(),
222             "Hide");
222         driver.findElement(toggleBtn).click();
223         Assert.assertEquals(field.getDomAttribute("type"), "password");
224     }
225 }

```

STEP 5: EXPLAINED LINE BY LINE

LINES	WHAT THIS PART DOES
1-9	Package and imports. <code>SoftAssert</code> lets one test report all its failures together.
10-12	Extends <code>BaseTest</code> , so <code>driver</code> and <code>wait</code> are ready. Every <code>@Test</code> method gets a fresh browser.
13-14	Address of the practice page. Change <code>PATH/T0</code> to your folder.
15-17	The seven rule names the page shows. We loop over this array instead of writing seven lines each time.
18-28	Every element is found by its <code>id</code> and stored once as a <code>By</code> . If an id changes you fix one line.
29-33	<code>open()</code> loads the page.
34-41	<code>type()</code> finds the box, <code>clear()</code> removes old text, <code>sendKeys()</code> types the new text. Empty text is skipped.

42-51	<code>enter()</code> fills username, password and confirm password. The shorter <code>enter(pwd)</code> below it is an overloaded shortcut.
52-55	Reads the <code>data-state</code> attribute of a rule line. The page sets it to <code>pass</code> , <code>fail</code> or <code>pending</code> .
56-59	Is the Create password button clickable? Uses <code>isEnabled()</code> .
60-72	TC01: before typing anything the button is disabled, strength says "Not entered", every rule is pending and there is no error or success text.
73-84	Data provider with four valid passwords. Two of them are the boundaries 8 and 16 characters. One test method will run four times, once per row.
85-102	Types the password, then checks all rules pass, the strength text, no error and an enabled button. <code>SoftAssert</code> reports every failed check together. Then it clicks the button and checks the success message.
103-118	Eight invalid passwords. Each breaks exactly ONE rule: boundary 7 and 17, missing uppercase / lowercase / number / special, a space, and the username rule.
119-134	For every rule it expects <code>fail</code> only for the broken rule and <code>pass</code> for all others. That proves the page flags the right rule and nothing else. It also checks "Weak", the error text and a disabled button.
135-165	TC14: SQL text. TC15: script text; if it had run, an alert would be open, so <code>switchTo().alert()</code> would NOT throw <code>NoAlertPresentException</code> . TC16: only spaces.
166-188	Different confirm password, a difference only in letter case, and an empty confirm password.
189-212	The button must follow the data: valid, then invalid, then valid again. Changing the username later can also make a valid password invalid.
213-225	The Show / Hide button changes the field <code>type</code> between <code>password</code> and <code>text</code> , and its own text between Show and Hide.

STEP 6: RUN IT

```
mvn test -Dtest=PasswordTest // run this class only
mvn test -Dtest=PasswordTest -Dheadless=true // run without a browser
window
mvn test // run every test
```

▼ Case 2: Full page validation (every kind of input)

Text boxes, date, radio, checkbox, dropdowns, multi-select, slider, textarea, file upload and popups.

STEP 1: THE PRACTICE PAGE

Open the page, click around like a manual tester first, then look at the locators below.

Open demo-form.html

The test already points to this live page (`PAGE_URL`), so it runs from any computer with internet. The same file is in the repository at `docs/demo-pages/demo-form.html` .

IMPORTANT What you will learn from this case

- Every input type and how to handle it
- Dependent dropdowns and multi-select
- Alert and confirm popups
- File upload and boundary testing on file size

STEP 2: WHERE EACH ELEMENT IS (LOCATORS)

ELEMENT	HTML ON THE PAGE	BY.ID	CSS SELECTOR	XPATH
Full name box	<code><input id="fullName" type="text"></code>	<code>fullName</code>	<code>#fullName</code>	<code>//input[@id='fullName']</code>
Date of birth	<code><input id="dob" type="date"></code>	<code>dob</code>	<code>input[type='date']</code>	<code>//input[@id='dob']</code>

Male radio button	<code><input type="radio" name="gender" id="genderMale" value="Male"></code>	genderMale	<code>input[name='gender'][value='Male']</code>	<code>//input[@name='gender' and @value='Male']</code>
Country dropdown	<code><select id="country"></code>	country	<code>select#country</code>	<code>//select[@id='country']</code>
State dropdown (starts disabled)	<code><select id="state" disabled></code>	state	<code>select#state</code>	<code>//select[@id='state']</code>
Skills multi-select	<code><select id="skills" multiple></code>	skills	<code>select[multiple]</code>	<code>//select[@multiple]</code>
English checkbox	<code><input type="checkbox" name="lang" id="langEnglish" value="English"></code>	langEnglish	<code>input[name='lang']:checked</code>	<code>//input[@name='lang' and @value='English']</code>
Experience slider	<code><input type="range" id="experience" min="0" max="20"></code>	experience	<code>input[type='range']</code>	<code>//input[@type='range']</code>
Address textarea	<code><textarea id="address" maxlength="200"></code>	address	<code>textarea#address</code>	<code>//textarea[@id='address']</code>
Resume file input	<code><input type="file" id="resume"></code>	resume	<code>input[type='file']</code>	<code>//input[@type='file']</code>
Error under a field	<code><div class="error" id="err-email"></code>	err-email	<code>#err-email</code>	<code>//div[@id='err-email']</code>
Submit / Reset buttons	<code><button type="submit" id="submitBtn">Submit</button></code>	submitBtn / resetBtn	<code>button[type='submit']</code>	<code>//button[text()='Reset']</code>

STEP 3: MANUAL TEST CASES FIRST

Automation follows manual testing. We first decide what to test, and only then write code.

ID	CONTROL	WHAT IS TESTED	TECHNIQUE
TC01-02	Whole form	Start state; submit empty form shows 11 error messages	State check, required fields
TC03	Name, email, phone (text)	Length 1, 2, 40, 41; letters only; email formats; phone length and first digit	Boundary value, equivalence partition
TC04	Date of birth (date)	Exactly 18 years, one day short, one day over, future date	Boundary value
TC05	Gender (radio)	Only one can be selected at a time	State check
TC06-07	Country and State (dependent dropdowns)	State list changes with country; disabled again when country is cleared	Dependency check
TC08	Skills (multi-select)	1 to 3 allowed; 4 and 0 give errors	Boundary value
TC09	Languages (checkboxes)	Tick and untick, count of ticked boxes	State check
TC10	Experience (slider)	Value display at 12 and at maximum 20	Boundary value
TC11-12	Address (textarea)	Counter, 200 character limit, minimum 10 characters	Boundary value
TC13	Resume (file upload)	PDF, PNG, exactly 2 MB, 2 MB + 1 byte, empty file, cv.pdf.exe	Boundary value, error guessing
TC14	Terms (checkbox)	Must be ticked	Required field
TC15	Submit	Full valid form, alert popup, result text	Positive end-to-end
TC16	Reset	Confirm popup: Cancel keeps data, OK clears it	Popup handling

STEP 4: THE CODE

File: selenium/FormTest.java

```
1  package selenium;
2
3  import org.openqa.selenium.Alert;
4  import org.openqa.selenium.By;
5  import org.openqa.selenium.JavascriptExecutor;
6  import org.openqa.selenium.WebElement;
7  import org.openqa.selenium.support.ui.ExpectedConditions;
8  import org.openqa.selenium.support.ui.Select;
9  import org.testng.Assert;
10 import org.testng.annotations.DataProvider;
11 import org.testng.annotations.Test;
12
13 import java.io.IOException;
14 import java.nio.file.Files;
15 import java.nio.file.Path;
16 import java.time.LocalDate;
17 import java.util.Arrays;
18 import java.util.List;
19 import java.util.stream.Collectors;
20
21 public class FormTest extends BaseTest {
22
23     // Live practice page hosted on GitHub Pages (works from any
computer with internet)
24     private static final String PAGE_URL =
25     "https://ubedzz49.github.io/qa-interview-prep/demo-pages/demo-
26     form.html";
27     private static final int MB = 1024 * 1024;
28
29     // ----- Helper methods -----
30     private void open() {
31         driver.get(PAGE_URL);
32     }
33
34     private void type(String id, String text) {
35         WebElement field = driver.findElement(By.id(id));
36         field.clear();
37         field.sendKeys(text);
38     }
39
40     private void clickSubmit() {
41         driver.findElement(By.id("submitBtn")).click();
42     }
43
44     private String error(String field) {
45         return driver.findElement(By.id("err-" + field)).getText();
46     }
47 }
```

```

46     // Date and range inputs are awkward to type into, so we set them
with JavaScript
47     private void setValueWithJs(String id, String value) {
48         WebElement el = driver.findElement(By.id(id));
49         ((JavascriptExecutor) driver).executeScript(
50             "arguments[0].value = arguments[1];" +
51             "arguments[0].dispatchEvent(new Event('input', {
bubbles: true }));", el, value);
52     }
53
54     // Create a fake file of a given name and size, return its full
path
55     private String makeFile(String name, int sizeInBytes) throws
IOException {
56         Path folder = Files.createTempDirectory("upload");
57         Path file = folder.resolve(name);
58         Files.write(file, new byte[sizeInBytes]);
59         return file.toString();
60     }
61
62     // ----- 1. Initial state -----
63     @Test
64     public void tc01_initialState() {
65         open();
66
67         Assert.assertFalse(driver.findElement(By.id("state")).isEnabled());
68
69         Assert.assertEquals(driver.findElement(By.id("expOut")).getText(),
"0");
70
71         Assert.assertEquals(driver.findElement(By.id("addrCount")).getText(),
"0");
72
73         Assert.assertEquals(driver.findElements(By.cssSelector("input[name='gen
der']:checked")).size(), 0);
74
75         Assert.assertFalse(driver.findElement(By.id("terms")).isSelected());
76
77         Assert.assertFalse(driver.findElement(By.id("result")).isDisplayed());
78     }
79
80     // ----- 2. Empty form: every required field shows an error --
-----
81
82     @Test
83     public void tc02_emptyFormShowsAllErrors() {
84         open();
85         clickSubmit();
86         String[][] expected = {
87             {"fullName", "Full name is required"},

```

```

81         {"email", "Email is required"},
82         {"phone", "Mobile number is required"},
83         {"dob", "Date of birth is required"},
84         {"gender", "Select a gender"},
85         {"country", "Select a country"},
86         {"skills", "Select at least one skill"},
87         {"lang", "Select at least one language"},
88         {"address", "Address is required"},
89         {"resume", "Upload your resume"},
90         {"terms", "You must accept the terms"},
91     };
92     for (String[] row : expected) {
93         Assert.assertEquals(error(row[0]), row[1], "Field " +
row[0]);
94     }
95 }
96
97 // ----- 3. Text inputs: data-driven validation -----
98 @DataProvider(name = "textCases")
99 public Object[][] textCases() {
100     String phoneMsg = "Enter a 10 digit number starting with 6, 7,
8 or 9";
101     String nameLenMsg = "Name must be 2 to 40 characters";
102     return new Object[][]{
103         // field,          value,          expected message,
why
104         {"fullName", "A",          nameLenMsg,
"Boundary: 1 char (min - 1)"},
105         {"fullName", "A1",          "",
"Boundary: 2 chars (min)"},
106         {"fullName", "A".repeat(40), "",
"Boundary: 40 chars (max)"},
107         {"fullName", "A".repeat(41), nameLenMsg,
"Boundary: 41 chars (max + 1)"},
108         {"fullName", "John3",          "Name can have only
letters, space, dot, hyphen and apostrophe", "Invalid: digit in name"},
109         {"fullName", "Mary-Ann O'Neil", "",          "Valid
special characters"},
110         {"email", "abc",          "Enter a valid email
address", "Invalid: no @"},
111         {"email", "abc@mail",          "Enter a valid email
address", "Invalid: no domain extension"},
112         {"email", "a b@mail.com",          "Enter a valid email
address", "Invalid: space"},
113         {"email", "user@mail.com",          "",          "Valid
email"},
114         {"phone", "12345",          phoneMsg, "Invalid:
too short"},

```

```

115         {"phone", "5876543210", phoneMsg, "Invalid:
starts with 5"},
116         {"phone", "98765432101", phoneMsg,
"Boundary: 11 digits"},
117         {"phone", "abcdefghij", phoneMsg, "Invalid:
letters"},
118         {"phone", "9876543210", "", "Valid
mobile number"},
119     };
120 }
121
122 @Test(dataProvider = "textCases")
123 public void tc03_textFieldValidation(String field, String value,
String message, String why) {
124     open();
125     type(field, value);
126     clickSubmit();
127     Assert.assertEquals(error(field), message, why);
128 }
129
130 // ----- 4. Date of birth: boundary around 18 years -----
131 @DataProvider(name = "dobCases")
132 public Object[][] dobCases() {
133     LocalDate today = LocalDate.now();
134     return new Object[][]{
135         {"exactly 18 years today (valid)",
today.minusYears(18).toString(), ""},
136         {"turns 18 tomorrow (invalid)",
today.minusYears(18).plusDays(1).toString(), "You must be at least 18
years old"},
137         {"18 years and 1 day (valid)",
today.minusYears(18).minusDays(1).toString(), ""},
138         {"tomorrow (future date)",
today.plusDays(1).toString(), "Date of birth cannot be in
the future"},
139     };
140 }
141
142 @Test(dataProvider = "dobCases")
143 public void tc04_dateOfBirth(String why, String date, String
message) {
144     open();
145     setValueWithJs("dob", date);
146     clickSubmit();
147     Assert.assertEquals(error("dob"), message, why);
148 }
149
150 // ----- 5. Radio buttons -----
151 @Test

```

```

152     public void tc05_onlyOneRadioCanBeSelected() {
153         open();
154         driver.findElement(By.id("genderFemale")).click();
155
156         Assert.assertTrue(driver.findElement(By.id("genderFemale")).isSelected(
157         ));
158         driver.findElement(By.id("genderOther")).click();
159
160         Assert.assertTrue(driver.findElement(By.id("genderOther")).isSelected(
161         ));
162
163         Assert.assertFalse(driver.findElement(By.id("genderFemale")).isSelected
164         ());
165
166         Assert.assertEquals(driver.findElements(By.cssSelector("input[name='gen
167         der']:checked")).size(), 1);
168     }
169
170     // ----- 6. Dropdowns: single and dependent -----
171     private List<String> optionTexts(Select select) {
172         return
173         select.getOptions().stream().map(WebElement::getText).collect(Collector
174         s.toList());
175     }
176
177     @Test
178     public void tc06_countryChangesStateList() {
179         open();
180         Select country = new
181         Select(driver.findElement(By.id("country")));
182         WebElement stateElement = driver.findElement(By.id("state"));
183
184         country.selectByVisibleText("India");
185         Assert.assertTrue(stateElement.isEnabled());
186         Assert.assertEquals(optionTexts(new Select(stateElement)),
187         Arrays.asList("-- Select state --", "Karnataka",
188         "Maharashtra", "Delhi"));
189
190         country.selectByValue("USA");
191         Assert.assertEquals(optionTexts(new Select(stateElement)),
192         Arrays.asList("-- Select state --", "California",
193         "Texas"));
194
195         country.selectByIndex(0); // back to
196         "-- Select country --"
197         Assert.assertFalse(stateElement.isEnabled());
198     }
199
200     @Test

```

```

187     public void tc07_stateIsRequiredAfterChoosingCountry() {
188         open();
189         new
190         Select(driver.findElement(By.id("country"))).selectByValue("UK");
191         clickSubmit();
192         Assert.assertEquals(error("state"), "Select a state");
193     }
194
195     // ----- 7. Multi-select -----
196     @Test
197     public void tc08_multiSelectAllowsOneToThreeSkills() {
198         open();
199         Select skills = new
200         Select(driver.findElement(By.id("skills")));
201         Assert.assertTrue(skills.isMultiple());
202
203         skills.selectByValue("Selenium");
204         skills.selectByValue("Git");
205         clickSubmit();
206         Assert.assertEquals(error("skills"), "");
207
208         skills.selectByValue("Playwright");
209         skills.selectByValue("SQL"); // now 4
210         selected = max + 1
211         clickSubmit();
212         Assert.assertEquals(error("skills"), "Select at most 3
213         skills");
214
215         skills.deselectAll();
216         clickSubmit();
217         Assert.assertEquals(error("skills"), "Select at least one
218         skill");
219     }
220
221     // ----- 8. Checkboxes -----
222     @Test
223     public void tc09_languageCheckboxes() {
224         open();
225         driver.findElement(By.id("langEnglish")).click();
226         driver.findElement(By.id("langTelugu")).click();
227
228         Assert.assertEquals(driver.findElements(By.cssSelector("input[name='lan
229         g']:checked")).size(), 2);
230
231         driver.findElement(By.id("langEnglish")).click(); // click
232         again = untick
233
234         Assert.assertEquals(driver.findElements(By.cssSelector("input[name='lan
235         g']:checked")).size(), 1);
236     }

```

```

226
227     // ----- 9. Slider -----
228     @Test
229     public void tc10_sliderShowsItsValue() {
230         open();
231         setValueWithJs("experience", "12");
232
233         Assert.assertEquals(driver.findElement(By.id("expOut")).getText(),
234             "12");
235         setValueWithJs("experience", "20"); // max
236
237         Assert.assertEquals(driver.findElement(By.id("expOut")).getText(),
238             "20");
239     }
240
241     // ----- 10. Textarea -----
242     @Test
243     public void tc11_textareaCounterAndLimit() {
244         open();
245         WebElement box = driver.findElement(By.id("address"));
246         box.sendKeys("Hello");
247
248         Assert.assertEquals(driver.findElement(By.id("addrCount")).getText(),
249             "5");
250
251         box.clear();
252         box.sendKeys("x".repeat(205)); // we
253         type 205, the browser stops at 200
254         Assert.assertEquals(box.getAttribute("value").length(), 200);
255
256         Assert.assertEquals(driver.findElement(By.id("addrCount")).getText(),
257             "200");
258     }
259
260     @Test
261     public void tc12_shortAddressIsRejected() {
262         open();
263         type("address", "Short");
264         clickSubmit();
265         Assert.assertEquals(error("address"), "Address must be at least
266             10 characters");
267     }
268
269     // ----- 11. File upload -----
270     @DataProvider(name = "fileCases")
271     public Object[][] fileCases() {
272         return new Object[][]{
273             // why, file name, size,
274             expected message

```

```

264         {"valid PDF", "resume.pdf", 1000,
    ""},
265         {"wrong type (png)", "photo.png", 1000,
    "Only PDF, DOC or DOCX files are allowed"},
266         {"exactly 2 MB (max)", "big.pdf", 2 * MB,
    ""},
267         {"2 MB + 1 byte (max + 1)", "bigger.pdf", 2 * MB + 1,
    "File must be 2 MB or smaller"},
268         {"empty file", "empty.pdf", 0,
    "File is empty"},
269         {"double extension", "cv.pdf.exe", 1000,
    "Only PDF, DOC or DOCX files are allowed"},
270     };
271 }
272
273 @Test(dataProvider = "fileCases")
274 public void tc13_resumeUpload(String why, String name, int size,
    String message) throws IOException {
275     open();
276     // For a file input we do NOT click. We send the file path as
text.
277     driver.findElement(By.id("resume")).sendKeys(makeFile(name,
    size));
278     clickSubmit();
279     Assert.assertEquals(error("resume"), message, why);
280 }
281
282 // ----- 12. Terms checkbox -----
283 @Test
284 public void tc14_termsMustBeAccepted() {
285     open();
286     clickSubmit();
287     Assert.assertEquals(error("terms"), "You must accept the
    terms");
288     driver.findElement(By.id("terms")).click();
289     clickSubmit();
290     Assert.assertEquals(error("terms"), "");
291 }
292
293 // ----- 13. Happy path: whole form, plus the alert popup ----
-----
294 @Test
295 public void tc15_validFormSubmitsAndShowsResult() throws
    IOException {
296     open();
297     type("fullName", "Ravi Kumar");
298     type("email", "ravi@example.com");
299     type("phone", "9876543210");
300     setValueWithJs("dob", "1998-05-20");

```

```

301         driver.findElement(By.id("genderMale")).click();
302
303         new
304         Select(driver.findElement(By.id("country"))).selectByVisibleText("India
305         ");
306         new
307         Select(driver.findElement(By.id("state"))).selectByVisibleText("Karnata
308         ka");
309
310         Select skills = new
311         Select(driver.findElement(By.id("skills")));
312         skills.selectByValue("Selenium");
313         skills.selectByValue("Playwright");
314
315         driver.findElement(By.id("langEnglish")).click();
316         driver.findElement(By.id("langHindi")).click();
317         setValueWithJs("experience", "2");
318         type("address", "12 MG Road, Bengaluru");
319
320         driver.findElement(By.id("resume")).sendKeys(makeFile("ravi.pdf",
321         500));
322         driver.findElement(By.id("terms")).click();
323
324         clickSubmit();
325
326         // The page shows an alert popup. We must handle it before
327         doing anything else.
328         Alert alert = wait.until(ExpectedConditions.alertIsPresent());
329         Assert.assertEquals(alert.getText(), "Application submitted
330         successfully");
331         alert.accept();
332
333         String json = driver.findElement(By.id("result")).getText();
334         Assert.assertTrue(json.contains("\"fullName\": \"Ravi
335         Kumar\""));
336         Assert.assertTrue(json.contains("\"phone\": \"9876543210\""));
337         Assert.assertTrue(json.contains("\"state\": \"Karnataka\""));
338         Assert.assertTrue(json.contains("\"experience\": 2"));
339         Assert.assertTrue(json.contains("\"resume\": \"ravi.pdf\""));
340     }
341
342     // ----- 14. Reset button and the confirm popup -----
343     @Test
344     public void tc16_resetAsksForConfirmation() {
345         open();
346         type("fullName", "Ravi Kumar");
347         new
348         Select(driver.findElement(By.id("country"))).selectByValue("USA");
349
350

```

```

339         driver.findElement(By.id("resetBtn")).click();
340         Alert confirm =
wait.until(ExpectedConditions.alertIsPresent());
341         Assert.assertEquals(confirm.getText(), "Clear the whole
form?");
342         confirm.dismiss();                                     // press
Cancel
343
Assert.assertEquals(driver.findElement(By.id("fullName")).getAttribute(
"value"), "Ravi Kumar");
344
345         driver.findElement(By.id("resetBtn")).click();
346         wait.until(ExpectedConditions.alertIsPresent()).accept();    //
press OK
347
Assert.assertEquals(driver.findElement(By.id("fullName")).getAttribute(
"value"), "");
348
Assert.assertFalse(driver.findElement(By.id("state")).isEnabled());
349     }
350 }

```

STEP 5: EXPLAINED LINE BY LINE

LINES	WHAT THIS PART DOES
1-20	Imports. Note <code>Select</code> , <code>Alert</code> , <code>JavascriptExecutor</code> , <code>LocalDate</code> and <code>file</code> classes. Each is used for one type of input.
21-26	Extends <code>BaseTest</code> . <code>PAGE_URL</code> is the page address. <code>MB</code> is the number of bytes in 1 MB, used for the file size boundaries.
27-46	Small helpers: <code>open()</code> , <code>type()</code> , <code>clickSubmit()</code> and <code>error(field)</code> , which reads the red message under a field (<code>err-<field></code>).
47-54	A date box and a slider are hard to type into. This helper sets the value with JavaScript, then fires an <code>input</code> event so the page reacts as if a user did it.
55-61	Creates a real temporary file with a given name and size. A file input needs a real path on disk.
62-73	TC01: the State dropdown is disabled, counters show 0, no radio selected, terms not ticked, result hidden.
74-96	TC02: click Submit on an empty form. A 2D array holds each field and its expected message, and a loop checks all 11.

97-121	Data provider with 15 rows: boundaries on name length (1, 2, 40, 41), invalid characters, email formats, phone length and first digit. An empty expected message means "no error expected".
122-129	One method for all 15 rows: type the value into the field whose id is <code>field</code> , click Submit, compare the error text.
130-149	Dates are calculated from today with <code>LocalDate</code> , so the test never goes out of date. Boundary: exactly 18 years, one day too young, one day older, and a future date.
150-161	Click a radio, check <code>isSelected()</code> . Picking another one must unselect the first. A CSS selector with <code>:checked</code> counts selected radios.
162-193	<code>Select</code> wraps a <code><select></code> . This is a dependent dropdown: changing the country changes the list of states. <code>getOptions()</code> reads the list; <code>selectByIndex(0)</code> returns to the first option and the State box becomes disabled again.
194-215	<code>isMultiple()</code> confirms the list allows many choices. Select 2 (valid), then 4 (max + 1, error), then <code>deselectAll()</code> (0, error).
216-226	Clicking a checkbox toggles it. Count of ticked boxes uses the CSS <code>:checked</code> .
227-236	The slider value is set with JavaScript and the shown number is checked, including the maximum 20.
237-258	Types "Hello" and checks the counter. Then types 205 characters: the browser keeps only 200 (<code>maxLength</code>). Another test checks the 10 character minimum.
259-281	Six cases with name and size. Boundary: exactly 2 MB passes, 2 MB + 1 byte fails. Also a wrong type, an empty file, and a double extension (<code>cv.pdf.exe</code>).
282-292	Without ticking the terms an error shows. After ticking it, the error clears.
293-331	Fills every control with valid data. After Submit the page shows an alert; <code>alertIsPresent()</code> waits for it, <code>getText()</code> reads it, <code>accept()</code> presses OK. Then the result text is checked.
332-350	Reset shows a confirm popup. <code>dismiss()</code> (Cancel) keeps the data. <code>accept()</code> (OK) clears the form and disables the State box.

STEP 6: RUN IT

```

mvn test -Dtest=FormTest // run this class only
mvn test -Dtest=FormTest -Dheadless=true // run without a browser window
mvn test // run every test

```

▼ Case 3: Complete flow (login to order confirmation)

One user journey: login, search, cart, coupon, checkout, payment and logout.

STEP 1: THE PRACTICE PAGE

Open the page, click around like a manual tester first, then look at the locators below.

Open demo-shop.html

The test already points to this live page (`PAGE_URL`), so it runs from any computer with internet. The same file is in the repository at `docs/demo-pages/demo-shop.html`.

IMPORTANT What you will learn from this case

- An end-to-end test with many steps
- Checking calculations (subtotal, discount, total)
- Negative cases inside a flow
- Finding elements inside a card or table row

STEP 2: WHERE EACH ELEMENT IS (LOCATORS)

ELEMENT	HTML ON THE PAGE	BY.ID	CSS SELECTOR	XPATH
Login email box	<code><input id="email" type="email"></code>	<code>email</code>	<code>#email</code>	<code>//input[@id='email']</code>
Login button	<code><button type="button" id="loginBtn">Login</button></code>	<code>loginBtn</code>	<code>#loginBtn</code>	<code>//button[text()='Login']</code>
Search box	<code><input id="search" placeholder="Search products"></code>	<code>search</code>	<code>input[placeholder='Search products']</code>	<code>//input[@placeholder='Search products']</code>

A product card	<code><div class="product" id="product-2"></code>	<code>product-2</code>	<code>.product</code>	<code>//div[@id='product-2']</code>
Add button inside a card	<code><button id="add-3">Add to cart</button></code>	<code>add-3</code>	<code>#product-3 button</code>	<code>//div[@id='product-3']//button</code>
Cart link	<code><button id="cartLink">Cart (0</button></code>	<code>cartLink</code>	<code>#cartLink</code>	<code>//button[contains(text(),'Cart')]</code>
Quantity of an item	<code>2</code>	<code>qty-1</code>	<code>#cart-row-1 span</code>	<code>//tr[@id='cart-row-1']//span</code>
Remove button	<code><button id="rm-3">Remove</button></code>	<code>rm-3</code>	<code>#rm-3</code>	<code>//tr[@id='cart-row-3']//button[text()='Remove']</code>
Coupon box	<code><input id="couponInput" aria-label="Coupon code"></code>	<code>couponInput</code>	<code>#couponInput</code>	<code>//input[@id='couponInput']</code>
Total amount	<code><strong id="total"></code>	<code>total</code>	<code>#total</code>	<code>//strong[@id='total']</code>
UPI radio button	<code><input type="radio" name="pay" id="payUpi" value="UPI"></code>	<code>payUpi</code>	<code>input[name='pay'] [value='UPI']</code>	<code>//input[@name='pay' and @value='UPI']</code>
Order id	<code><strong id="orderId"></code>	<code>orderId</code>	<code>#orderId</code>	<code>//strong[@id='orderId']</code>

STEP 3: MANUAL TEST CASES FIRST

Automation follows manual testing. We first decide what to test, and only then write code.

STEP	ACTION	EXPECTED RESULT	IDEA

Login (4 cases)	Empty fields, wrong password, unknown email, password in wrong case	Correct error message, shop page stays hidden	Negative testing
1	Login with qa@test.com and Test@1234	Welcome text, 3 products shown	Positive
2	Search "pen", "zzz"; filter by category	1 product, "No products found", counts correct	Search and filter
3	Add Notebook twice and Backpack once	Cart count is 3	State change
4	Open the cart	Notebook qty 2, line total 500, subtotal 1700	Calculation check
5	Click + then -	Quantity 3 then back to 2	State change
6	Coupon ABC, then SAVE10	Invalid message; then discount 170, total 1530	Negative then positive
7	Remove Backpack	Subtotal 500, discount 50, total 450	Recalculation
8	Checkout with empty fields, PIN 123, no payment	Three different error messages	Validation
9	Pay with UPI	Order placed, id like ORD-12345, total 450, cart emptied	End result
10	Logout	Back to login page	Flow end
Extra	Open cart with no items	Checkout button disabled	Rule check

STEP 4: THE CODE

File: selenium/ShopFlowTest.java

```
1 package selenium;
2
3 import org.openqa.selenium.By;
4 import org.openqa.selenium.Keys;
5 import org.openqa.selenium.WebElement;
6 import org.openqa.selenium.support.ui.ExpectedConditions;
7 import org.openqa.selenium.support.ui.Select;
8 import org.testng.Assert;
9 import org.testng.annotations.DataProvider;
10 import org.testng.annotations.Test;
11
12 public class ShopFlowTest extends BaseTest {
13
14     // Live practice page hosted on GitHub Pages (works from any
computer with internet)
15     private static final String PAGE_URL =
16         "https://ubedzz49.github.io/qa-interview-prep/demo-pages/demo-";
17     private static final String EMAIL = "qa@test.com";
18     private static final String PASSWORD = "Test@1234";
19
20     // ----- Helper methods -----
21     private void type(String id, String text) {
22         WebElement field = driver.findElement(By.id(id));
23         field.clear();
24         field.sendKeys(text);
25     }
26
27     private void click(String id) {
28         driver.findElement(By.id(id)).click();
29     }
30
31     private String text(String id) {
32         return driver.findElement(By.id(id)).getText();
33     }
34
35     private void login(String email, String password) {
36         type("email", email);
37         type("pass", password);
38         click("loginBtn");
39     }
40
41     // "₹1530" -> 1530
42     private int money(String id) {
43         return Integer.parseInt(text(id).replace("₹", ""));
44     }
45
46     private int productCount() {
47         return driver.findElements(By.cssSelector(".product")).size();
48     }
49 }
```

```

47     }
48
49     // Empty a text box by pressing Backspace (this fires the key
events the page listens to)
50     private void emptyBox(String id) {
51         WebElement box = driver.findElement(By.id(id));
52         int length = box.getAttribute("value").length();
53         for (int i = 0; i < length; i++) {
54             box.sendKeys(Keys.BACK_SPACE);
55         }
56     }
57
58     // ----- Login: negative cases -----
59     @DataProvider(name = "badLogins")
60     public Object[][] badLogins() {
61         return new Object[][]{
62             {"both empty", "", ""},
63             {"Email and password are required", "Email", "wrong"},
64             {"Invalid email or password", "nobody@test.com", "wrong"},
65             {"password in wrong case", EMAIL, "wrong"},
66             {"test@1234", "Invalid email or password", "wrong"}
67         };
68     }
69
70     @Test(dataProvider = "badLogins")
71     public void loginFails(String why, String email, String password,
72         String message) {
73         driver.get(PAGE_URL);
74         login(email, password);
75         Assert.assertEquals(text("loginError"), message, why);
76
77         Assert.assertFalse(driver.findElement(By.id("shopView")).isDisplayed());
78     }
79
80     // ----- Complete end-to-end flow -----
81     @Test
82     public void endToEndPurchase() {
83         driver.get(PAGE_URL);
84
85         // 1. Login with valid details
86         login(EMAIL, PASSWORD);
87
88         wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("shopView")));
89
90         Assert.assertEquals(text("welcome"), "Welcome, qa@test.com");

```

```

86         Assert.assertEquals(productCount(), 3);
87
88         // 2. Search and filter products
89         type("search", "pen");
90         Assert.assertEquals(productCount(), 1);
91
92         Assert.assertEquals(driver.findElement(By.cssSelector("#product-2
93             .name")).getText(), "Pen Pack");
94
95         emptyBox("search");
96         type("search", "zzz");
97
98         Assert.assertTrue(driver.findElement(By.id("noResults")).isDisplayed())
99         ;
100
101         emptyBox("search");
102         Select category = new
103         Select(driver.findElement(By.id("category")));
104         category.selectByVisibleText("Bags");
105         Assert.assertEquals(productCount(), 1);
106         category.selectByValue("All");
107         Assert.assertEquals(productCount(), 3);
108
109         // 3. Add products to the cart
110         click("add-1"); // Notebook
111         click("add-1"); // Notebook
112         again, so quantity = 2
113         driver.findElement(By.xpath("//div[@id='product-
114             3']//button")).click(); // Backpack, found with XPath
115         Assert.assertEquals(text("cartCount"), "3");
116
117         // 4. Check the cart and the maths
118         click("cartLink");
119         Assert.assertEquals(text("qty-1"), "2");
120         Assert.assertEquals(driver.findElement(By.cssSelector("#cart-
121             row-1 .line-total")).getText(), "₹500");
122         Assert.assertEquals(money("subtotal"), 500 + 1200);
123         Assert.assertEquals(money("discount"), 0);
124         Assert.assertEquals(money("total"), 1700);
125
126         // 5. Change quantity
127         click("inc-1");
128         Assert.assertEquals(text("qty-1"), "3");
129         click("dec-1");
130         Assert.assertEquals(text("qty-1"), "2");
131
132         // 6. Coupon: invalid first, then valid
133         type("couponInput", "ABC");
134         click("applyCoupon");

```

```

127     Assert.assertEquals(text("couponMsg"), "Invalid coupon code");
128     Assert.assertEquals(money("discount"), 0);
129
130     type("couponInput", "SAVE10");
131     click("applyCoupon");
132     Assert.assertEquals(text("couponMsg"), "Coupon SAVE10 applied:
10% off");
133     Assert.assertEquals(money("discount"), 170);
134     Assert.assertEquals(money("total"), 1530);
135
136     // 7. Remove an item and the total updates
137     click("rm-3"); // remove
Backpack
138     Assert.assertEquals(driver.findElements(By.id("cart-row-
3")).size(), 0);
139     Assert.assertEquals(money("subtotal"), 500);
140     Assert.assertEquals(money("discount"), 50);
141     Assert.assertEquals(money("total"), 450);
142
143     // 8. Checkout: validation first
144     click("checkoutBtn");
145     click("placeOrder");
146     Assert.assertEquals(text("checkoutError"), "Name and address
are required");
147
148     type("shipName", "Ravi Kumar");
149     type("shipAddress", "12 MG Road, Bengaluru");
150     type("pin", "123");
151     click("placeOrder");
152     Assert.assertEquals(text("checkoutError"), "Enter a valid 6
digit PIN code");
153
154     type("pin", "560001");
155     click("placeOrder");
156     Assert.assertEquals(text("checkoutError"), "Select a payment
method");
157
158     // 9. Pay and confirm the order
159     click("payUpi");
160     click("placeOrder");
161
    wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("doneVie
w"))));
162     Assert.assertEquals(text("orderStatus"), "Order placed");
163     Assert.assertTrue(text("orderId").matches("ORD-\\d{5}"), "Order
id format");
164     Assert.assertEquals(text("orderTotal"), "₹450");
165     Assert.assertEquals(text("orderPay"), "UPI");

```

```

166         Assert.assertEquals(text("cartCount"), "0");    // cart is
            emptied
167
168         // 10. Logout returns to the login page
169         click("logoutBtn");
170
171         Assert.assertTrue(driver.findElement(By.id("loginView")).isDisplayed())
            ;
172
173         Assert.assertFalse(driver.findElement(By.id("topNav")).isDisplayed());
174     }
175
176     // ----- Empty cart rule -----
177     @Test
178     public void checkoutIsDisabledForEmptyCart() {
179         driver.get(PAGE_URL);
180         login(EMAIL, PASSWORD);
181         click("cartLink");
182
183         Assert.assertTrue(driver.findElement(By.id("emptyCart")).isDisplayed())
            ;
184
185         Assert.assertFalse(driver.findElement(By.id("checkoutBtn")).isEnabled()
            );
186     }
187 }

```

STEP 5: EXPLAINED LINE BY LINE

LINES	WHAT THIS PART DOES
1-11	Imports. <code>Keys</code> is used to press Backspace.
12-18	Extends <code>BaseTest</code> . The page address and the valid login are constants.
19-40	Short helpers so the test reads like a story: <code>type</code> , <code>click</code> , <code>text</code> (visible text by id), and <code>login</code> .
41-49	<code>money("total")</code> turns the text "₹1530" into the number 1530 so we can compare numbers.
50-57	Empties a box by pressing Backspace. This fires the keyboard events the search box listens to.
58-68	Four wrong logins: both empty, wrong password, unknown email, and the password in the wrong letter case.

69-76	Each row logs in with bad data, then checks the exact error text and that the shop page stays hidden.
77-81	One long test is one full user journey. The numbered comments are the steps.
82-87	Log in, wait until the shop page is visible (explicit wait), check the welcome text and 3 products.
88-103	Search "pen" shows 1 product; a CSS selector with a space (<code>#product-2 .name</code>) finds the name inside the card. "zzz" shows "No products found". The <code>Select</code> class filters by category.
104-109	Add the Notebook twice. The Backpack button is found with XPath: the button inside the div whose id is <code>product-3</code> .
110-117	Open the cart and check quantity 2, the line total, and that subtotal $1700 = 500 + 1200$.
118-123	Plus makes quantity 3, minus brings it back to 2.
124-135	A wrong coupon gives an error and no discount. SAVE10 gives 10% off: discount 170, total 1530.
136-142	Remove the Backpack. The row must be gone (<code>findElements</code> size 0) and the totals recalculate: 500, 50, 450.
143-157	Three validation messages in order: empty fields, wrong PIN 123, no payment method.
158-167	Choose UPI and place the order. Check the status, the order id with a regular expression (<code>ORD-</code> and 5 digits), the amount, and that the cart is empty.
168-173	Logout returns to the login page and hides the top menu.
174-183	A separate small test: with an empty cart the Checkout button must be disabled.

STEP 6: RUN IT

```

mvn test -Dtest=ShopFlowTest           // run this class only
mvn test -Dtest=ShopFlowTest -Dheadless=true // run without a browser
window
mvn test                                // run every test

```

13. Quick interview answers

- **IMPORTANT What is Selenium?**

An open-source library to automate web browsers. It supports many languages (Java, Python, C#) and browsers (Chrome, Firefox, Edge, Safari). It does not test desktop or mobile apps (mobile uses Appium).

- **IMPORTANT Why `WebDriver driver = new ChromeDriver()` and not `ChromeDriver driver = ...`?**

WebDriver is an interface. Writing the interface on the left lets you switch to another browser by changing one word on the right.

- **IMPORTANT findElement vs findElements?**

One element and an exception if none, versus a list that is empty if none.

- **IMPORTANT close() vs quit()?**

close() closes the current window. quit() closes all windows and ends the session.

- **IMPORTANT Implicit vs explicit wait?**

Implicit: global, for all elements. Explicit: one condition for one element, with `WebDriverWait`. Prefer explicit, and never mix the two.

- **IMPORTANT Which locator do you prefer?**

id first, then CSS, then relative XPath, because they are fast and stable.

- **IMPORTANT XPath vs CSS?**

XPath can search by text and move to parent elements. CSS is simpler and usually faster.

- **IMPORTANT Absolute vs relative XPath?**

Absolute starts from the root (`/html/...`) and breaks easily. Relative starts with `//` and is safer.

- **IMPORTANT How do you handle a dropdown, an alert, a frame, a new window?**

`Select class;` `switchTo().alert();` `switchTo().frame();` `getWindowHandles() + switchTo().window();`

- **IMPORTANT How do you handle `StaleElementReferenceException`?**

The page changed and the saved element is old. Find the element again.

- **IMPORTANT What is Page Object Model?**

A design pattern: one class per page that holds its locators and actions. It reduces repeated code and makes maintenance easy.

- **IMPORTANT How does TestNG help?**

Runs tests, gives annotations (`@Test` , `@BeforeMethod`), assertions, data providers,

grouping, parallel runs and reports.

- **GOOD TO KNOW** **What is a data provider?**

A method that returns rows of data. The test method runs once per row.

- **GOOD TO KNOW** **How do you upload a file?**

`sendKeys` with the file path on the `<input type="file">`. Do not click it.

- **GOOD TO KNOW** **How do you run tests without a visible browser?**

Headless mode with `ChromeOptions.addArguments("--headless=new")`.

- **GOOD TO KNOW** **What is Selenium Grid?**

Runs tests in parallel on many browsers and machines.

- **GOOD TO KNOW** **What are Selenium limitations?**

Web only, no built-in reports, needs explicit waits, and cannot handle captcha or image checks.

- **GOOD TO KNOW** **Selenium vs Playwright?**

Playwright has built-in auto-waiting, is often faster, and has built-in tracing. Selenium has the bigger community and more language support. Learn both ideas.