SimpleThreadSafeSeleniumFramework

File■by■File Explanation & Team Handover Guide

This document explains what each file does, why it exists, how it is used, and what you typically customize when adapting the framework to a real project.

1) Framework Overview (How it works end■to■end)

- **Thread safety**: WebDriver is stored in **ThreadLocal** so each parallel test thread gets an isolated browser session.
- **Lifecycle**: **BaseTest** creates the driver in @BeforeMethod and quits it in @AfterMethod.
- **POM**: Page classes contain locators + actions; tests call page methods (tests do not store locators).
- **Reporting**: A TestNG listener creates an Extent test per method (also thread■local) and attaches screenshots on failures.
- **Config driven**: config.properties controls run mode (local/grid), browser, URL, and timeouts; values can be overridden via -D.
- **Data driven**: Example DataProviders read from Excel and from a demo DB (H2) to show both styles.

Project map (high level)

src/main/java → framework reusable code (config/driver/pages/utils)
src/test/java → test code (base, listeners, tests)
src/test/resources → config files (config.properties, log4j2.xml, testdata)
testng.xml → suite definition + parallel settings
pom.xml → dependencies + surefire configuration
Jenkinsfile → CI pipeline skeleton

2) src/main/java (Framework Code)

2.1 ConfigManager.java (Configuration Reader)

Purpose: Central place to read configuration values (browser, URLs, timeouts, grid settings). It loads **config.properties** once and also supports overriding values via JVM system properties (e.g., -Dbrowser=firefox).

What it does internally

- Loads **config.properties** from the classpath using the ClassLoader (so it works in IDE and in Maven).
- Stores values in a Properties object (in memory).
- When you call ConfigManager.get("key"), it first checks **System.getProperty** (CLI override) then falls back to properties file.
- Provides helper **getInt** for numeric values with default fallback.

How to use it

Example usage inside framework code:

```
String baseUrl = ConfigManager.get("base.url"); int wait =
ConfigManager.getInt("explicit.wait.seconds", 10);
```

What you customize

- Add new keys in **src/test/resources/config.properties** (e.g., api.base.url, db.url, users.admin).
- Optionally add typed getters (getBoolean/getLong) if your project needs them.

2.2 DriverManager.java (ThreadLocal WebDriver Holder)

Purpose: Stores and retrieves the WebDriver for the *current* test thread. This is the key component that makes the framework thread■safe in parallel execution.

What it does internally

- Uses **ThreadLocal<WebDriver>** so each thread has its own isolated driver instance.
- Provides **setDriver()** in setup to bind a driver to the running thread.
- Provides **getDriver()** so pages/utils can always fetch the correct driver for that test.
- Provides **unload()** (ThreadLocal.remove) to prevent memory leaks in long runs.

How to use it

In tests and page objects, do not store a static driver. Always use:

```
WebDriver driver = DriverManager.getDriver();
```

What you customize

Usually nothing—this stays small and stable. The main rule is: never bypass it by using static driver variables.

2.3 DriverFactory.java (Creates Local or Grid Drivers)

Purpose: Builds the correct WebDriver based on configuration. It supports **local** execution (Chrome/Firefox/Edge) and optional **Selenium Grid** execution (RemoteWebDriver).

What it does internally

- Reads **run.mode** and **browser** from ConfigManager.
- If run.mode=local → creates a local driver (ChromeDriver/FirefoxDriver/EdgeDriver).
- If run.mode=grid → creates RemoteWebDriver with browser-specific capabilities and connects to grid.url.
- Uses WebDriverManager for local driver binaries (auto-download for demo projects).

How to use it

Used in BaseTest setup:

```
WebDriver driver = new DriverFactory().createDriver(); DriverManager.setDriver(driver);
```

What you customize

- Add browser options (headless, disable notifications, downloads folder, accept insecure certs).
- Add Grid capabilities (platformName, browserVersion, resolution) for your Grid/Cloud provider.
- Add support for additional browsers or mobile (Appium) if needed.
- If your org manages driver binaries centrally, you may remove WebDriverManager and use pre-installed drivers.

2.4 pages package (Page Object Model)

Purpose: Each class represents one screen (or component) of the application under test. Pages contain locators + reusable actions. Tests call page methods, not locators.

LoginPage.java

- Contains locators for username/password/login button/flash message.
- Exposes fluent methods: enterUsername(), enterPassword(), submitLogin().
- Uses WaitUtils (explicit waits) so actions are stable.
- Uses DriverManager.getDriver() so it always uses the correct thread's driver.

How to use LoginPage in a test:

```
LoginPage login = new LoginPage(); SecureAreaPage secure =
login.enterUsername("u").enterPassword("p").submitLogin();
```

SecureAreaPage.java

- Represents the post-login area in the demo app.
- Contains locators for flash message and logout link.
- Provides methods getFlashMessage() and logout().
- Keeps tests clean: test asserts on returned values instead of locating elements.

What you customize in pages

- Create a page class per banking module (DashboardPage, BeneficiaryPage, FundTransferPage, StatementsPage).
- Keep locators private and expose business-level actions as methods.
- Avoid static fields; avoid sharing WebElements across tests.

2.5 utils package (Reusable Helpers)

WaitUtils.java (Explicit Wait Wrapper)

- Centralizes WebDriverWait creation using explicit.wait.seconds from config.
- Provides helper methods like visible(locator) and click(locator).
- Encourages stability by avoiding Thread.sleep in tests/pages.

How to use WaitUtils:

```
WaitUtils.visible(By.id("username")).sendKeys("tom");
WaitUtils.click(By.cssSelector("button[type=submit]"));
```

ScreenshotUtils.java (Screenshots on failure)

- Captures screenshots using TakesScreenshot.
- Stores screenshots in target/screenshots.
- Uses unique file names including test name + thread id + timestamp (prevents overwrites in parallel runs).

How it is used:

Called automatically from ExtentTestNGListener.onTestFailure().

ExcelUtils.java (Excel Data Provider Helper)

- Reads an .xlsx file from classpath resources (src/test/resources).
- Returns Object[][] for TestNG DataProvider (each row becomes one test execution).
- Assumes row0 is header and reads rows starting from row1.

How to use in a DataProvider:

```
@DataProvider(name="excel", parallel=true) public Object[][] data(){ return
ExcelUtils.readSheet("testdata/login-data.xlsx","login"); }
```

DBUtils.java (Database Data Provider Helper — Demo H2)

- Demonstrates DB-driven test data using an embedded H2 in-memory database.
- Initializes schema + sample rows on first use (init()).
- Fetches login datasets and returns Object[][] for DataProvider.
- Shows best practice: do not share a single JDBC Connection across threads; use a pool or per-call connection.

How to customize DBUtils for your org:

- Replace H2 with your real DB URL/driver and credentials (use environment variables or Jenkins credentials binding).
- Prefer a connection pool (HikariCP) or a managed DataSource.
- Keep DB query logic inside DBUtils—not in tests.

3) src/test/java (Test Layer)

3.1 BaseTest.java (Setup/Teardown & Driver Lifecycle)

- Runs before each test method (@BeforeMethod): creates WebDriver using DriverFactory and stores it in DriverManager (ThreadLocal).
- Navigates to base.url and maximizes window.
- Runs after each test method (@AfterMethod): quits the driver and removes ThreadLocal reference.
- Ensures tests are independent and safe for parallel execution.

How you use BaseTest:

```
All test classes should extend BaseTest: public class MyTests extends BaseTest { ... }
```

3.2 listeners package (Reporting + Retry)

ExtentManager.java (ExtentReports Singleton)

- Creates and configures ExtentReports once for the entire run.
- Attaches an ExtentSparkReporter writing to target/extent-report/ExtentReport.html.
- Sets system info metadata shown in the report.

Customize ExtentManager:

Change report name/title/system info inside ExtentManager.getExtent().

ExtentTestNGListener.java (Thread■safe reporting + screenshots)

- Implements TestNG ITestListener to listen to test start/success/failure events.
- Creates one ExtentTest node per test method.
- Stores ExtentTest in ThreadLocal so parallel tests do not mix logs.
- On failure, captures screenshot via ScreenshotUtils and attaches it to the report.
- On finish, flushes the report to write the HTML file.

How it is enabled:

Registered in testng.xml under .

RetryAnalyzer.java (Controlled retry for flaky failures)

- Implements IRetryAnalyzer.
- Retries a failing test up to MAX_RETRY times (set to 1 in this sample).
- Helps reduce noise from transient issues, but should be used sparingly.

How to use RetryAnalyzer:

```
@Test(retryAnalyzer = RetryAnalyzer.class) public void test(){ ... }
```

3.3 tests package (Your test cases)

LoginTests.java (Example tests + DataProviders)

- Shows two tests: one driven by Excel DataProvider and one driven by DB DataProvider.
- Both tests reuse a shared helper method runLoginScenario() for clean code.
- Demonstrates how parallel=true on DataProviders can increase concurrency.
- Demonstrates assertions on business outcomes (flash message).

4) Resources & Root Files (What they do)

4.1 src/test/resources/config.properties

Central configuration file for browser/run mode/grid URL/base URL/timeouts.

Common keys you will customize:

```
run.mode=local|grid browser=chrome|firefox|edge grid.url=http://... base.url=https://...
explicit.wait.seconds=10
```

4.2 src/test/resources/log4j2.xml

Log4j2 configuration: console logging + rolling file logging under target/logs.

Customize pattern and log level here.

*4.3 src/test/resources/testdata/**

Test data files such as login-data.xlsx used by ExcelUtils.

4.4 testng.xml

Defines which tests run and how they run in parallel. Also registers listeners.

Key settings to customize:

```
...
```

4.5 pom.xml

Maven project file: dependencies (Selenium, TestNG, Extent, Log4j2, POI, H2) and Surefire plugin configuration to run testng.xml.

4.6 Jenkinsfile

A starter Jenkins pipeline: checks out code, runs mvn clean test, archives reports/logs/screenshots as artifacts.

Appendix: Best practices for extending this framework

- Keep tests independent for parallel execution (no shared accounts unless data is isolated).
- Never store WebDriver or ExtentTest in static variables—use ThreadLocal patterns.
- Prefer explicit waits and stable locators; keep RetryAnalyzer low (0–1).
- Add a BasePage if your project grows (common actions, common waits).
- Tag tests using TestNG groups (smoke/regression) and run them selectively in Jenkins.