Lesson: SDET(Software Development Engineer using AI in Testing)

a). Software: An interconnected collection of programs run in an electronic device like computers, mobiles and other IoT(Internet of Things) devices, is called as software.

b). Quality software: To be considered of high quality, software should:
--> Meet functional requirements specified by the customer (features or functionalities).
--> Satisfy non-functional expectations, such as:
Performance, User-friendliness, Security, Compatibility, etc.
--> Be reasonably priced.
--> Be released on time.

c) Purpose of Testing: The primary purpose of testing is to ensure the release of quality software to Customer(Project) or to Market(Product).

d) Manual Testing vs. Test Automation:

Manual Testing:
--> Performed by a tester (Software Test Engineer, Quality Engineer, Quality Assurance Engineer, or Test Analyst).
--> Involves testing the software without the use of any tools or automation scripts.
--> The tester manually interacts with the software using a mouse and keyboard.

Test Automation:
--> Conducted by a tester (SDET - Software Development Engineer in Test, or Test Automation Engineer).
--> Involves writing and executing automation test scripts (programs) to validate the software.
--> Use AI plugins or AI IDEs or AI Agentic platforms to generate automation test scripts (programs) to validate the software.

e. Why Test Automation?

--> Use AI to develop Automation scripts on fly.

--> By automating repetitive tasks(Re & Regression), testers can focus on more complex issues, leading to higher software quality.

--> Test automation is important because it makes the testing process faster, more consistent, and more reliable by running scripts in 24X7 cycle.

--> It allows for more extensive testing, helps catch bugs early, saves time and effort, and supports faster release cycles.

f. Modern Testing Approaches in Job Market

1. Manual Testing:

- Performed without automation tools, focusing on exploratory and usability testing.
- Still essential for non-repetitive, user-centric testing scenarios but less in demand for repetitive tasks.

2. Traditional Test Automation:

- Involves writing and executing test scripts using tools like Playwright, Selenium, Appium, ..etc
- Highly in demand for reducing testing time and ensuring reliability.
- Requires programming skills and CI/CD pipeline integration.

3. GenAI-Driven Test Automation:

- Uses Generative AI tools to generate and execute test scripts.
- Significantly accelerates script creation, reducing manual coding efforts.
- Requires expertise in AI tools, automation frameworks, and prompt engineering.

Key Trend:

--> While manual testing is foundational, traditional automation and AI-powered testing dominate the job market.

--> Skills in both areas are crucial for staying competitive and future-proofing our career.

g). Testing stages with Test automation:-

Testing Stage	Source	By	Techniques
Documents testing	Docs like BRS, SRS (User Stories), HLD, LLDs, ...etc	PO,SM, BA and SHs	Walkthrough, Inspection and Peer Review manually(static test)
Unit and Integration Testing	individual programs and integration of those programs	Dev Team	White-box techniques manually (or) automation using Junit for Java, NUnit for C#.net, pytest for Python, ...etc
Software Testing (Functional and Non-Functional Test)	-> Functional testing on partial or complete software. -> But non-functional testing on complete software only. So, it is called as "System Testing".	QA(Testers) Team	1. Black-box techniques for functional testing via manually or via automation using Selenium-Java, UFT, Cypress, Postman, RA-Java, Appium-Java, Playwright-TypeScript...etc 2. No techniques for Non-functional tests but conducting manually or via automation using Loadrunner, Jmeter, Burb Suite, ...etc

UAT(User Acceptence Test)	collect final feedback from client site people on final software	PO, SM, BA, SHs, Dev team, QA team	α, β, α techniques via manually or via automation using TestRail, Zephyr, TestLink, ...etc
Release/Port Testing (ensuring that a software release is of high quality and ready for production use)	on final software	A deployment team of individuals and may involve various roles like dev & QA	Jenkins is a widely used CI/CD automation server that can help automate the deployment of software releases.
Maintanance Testing (address changing requirements or technology updates to ensure the software's continued success in client site)	on final software in Clientsite / production	Operations (Ops) Team (OR) CCB-Change Control Board team	Repeat previously executed automation test scripts on modified software, called as Regression test. (Automation only to save time)

--> Now, Most of the Software companies are maintaining separate testers(independent testing) for Software testing stage(Functional + Non-Functional) only.

--> This separate team of testers uses "Test Automation" for complete Functional testing in Software testing.

--> Most of the Non-functional tests will be conducted by manually. Few non-functional tests like Performance, Compatability and Security tests will be conducted via "Test automation".

✅ Automation Importance in Testing Stages

Testing Stage	Automation Role	Importance
1. Requirement/Document Testing	Not common – mostly manual reviews, GenAI may help	❌ Low
2. Unit Testing	Highly automated (via JUnit, NUnit, etc.)	✅ Very High
3. Integration Testing	Automated API calls, service stubs, mocks	✅ High
4. Functional Testing	UI/API automation of key flows	✅ Very High
5. Regression Testing	Automation is critical to speed up releases	✅ Extremely High
6. Non-Functional Testing	Performance, security, accessibility tools used	✅ Medium to High
7. UAT (User Acceptance)	Rarely automated; done manually by end users	❌ Low
8. Release/Maintenance	Automation validates hotfixes, patches, upgrades	✅ High

(Job of Dev)

(Software testing by separate Testers)

(Support)

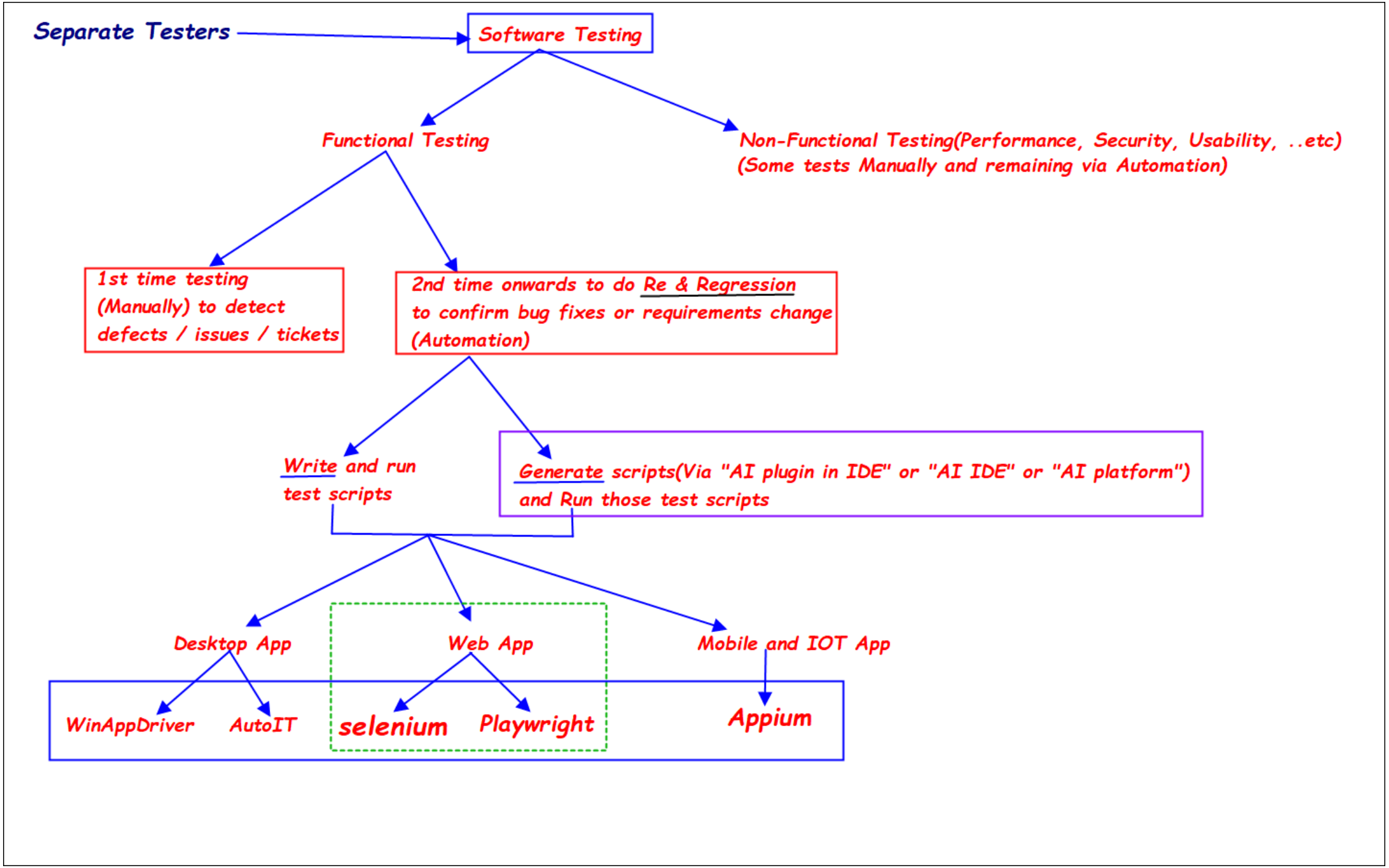
In Summary:

-> First-time Functional Testing is usually done "manually" to explore, validate UI, and catch edge cases.

-> Regression Testing (repeating old test cases after bug fixes) is "automated" to save time and ensure stability.

-> Use these automation scripts when software changed(after code changes) due to customer business changes.

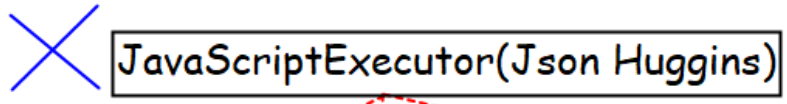
-> That's the industry-standard approach in most QA teams today.


Separate Testers
Software Testing
Functional Testing
Non-Functional Testing(Performance, Security, Usability, ..etc)
(Some tests Manually and remaining via Automation)
1st time testing
(Manually) to detect
defects / issues / tickets
2nd time onwards to do Re & Regression
to confirm bug fixes or requirements change
(Automation)
Write and run
test scripts
Generate scripts(Via "AI plugin in IDE" or "AI IDE" or "AI platform")
and Run those test scripts
Desktop App
Web App
Mobile and IOT App
WinAppDriver
AutoIT
selenium
Playwright
Appium

h. Why Selenium?

--> To automate functional testing on web sites(UI) in computer, we use Selenium.
--> It is a open source (no license cost).
--> It is available in various languages like Java, Python, C#, Ruby and JavaScript.
--> Selenium-Java is more popular due to platform independent and huge online community.

i. Selenium Components:



1 Selenium IDE

-> Developed by Shinya Kastani
-> IDE stands for "Integrated Development Envioronment"
-> Record and playback tool
-> Not acceptable to dynamic web site testing.
-> This component is outdated.

Selenium Remote Control

2 Selenium WebDriver

-> Developed by Simon Stewert
-> Coding library and allows us to write code (Generate code using GenAI if needed)
-> Easy to update code for dynamic web site testing

Selenium Grid 3

-> Developed by Paul Hammant
-> To run our scripts in multiple platforms parallel.

j. Why Playwright?

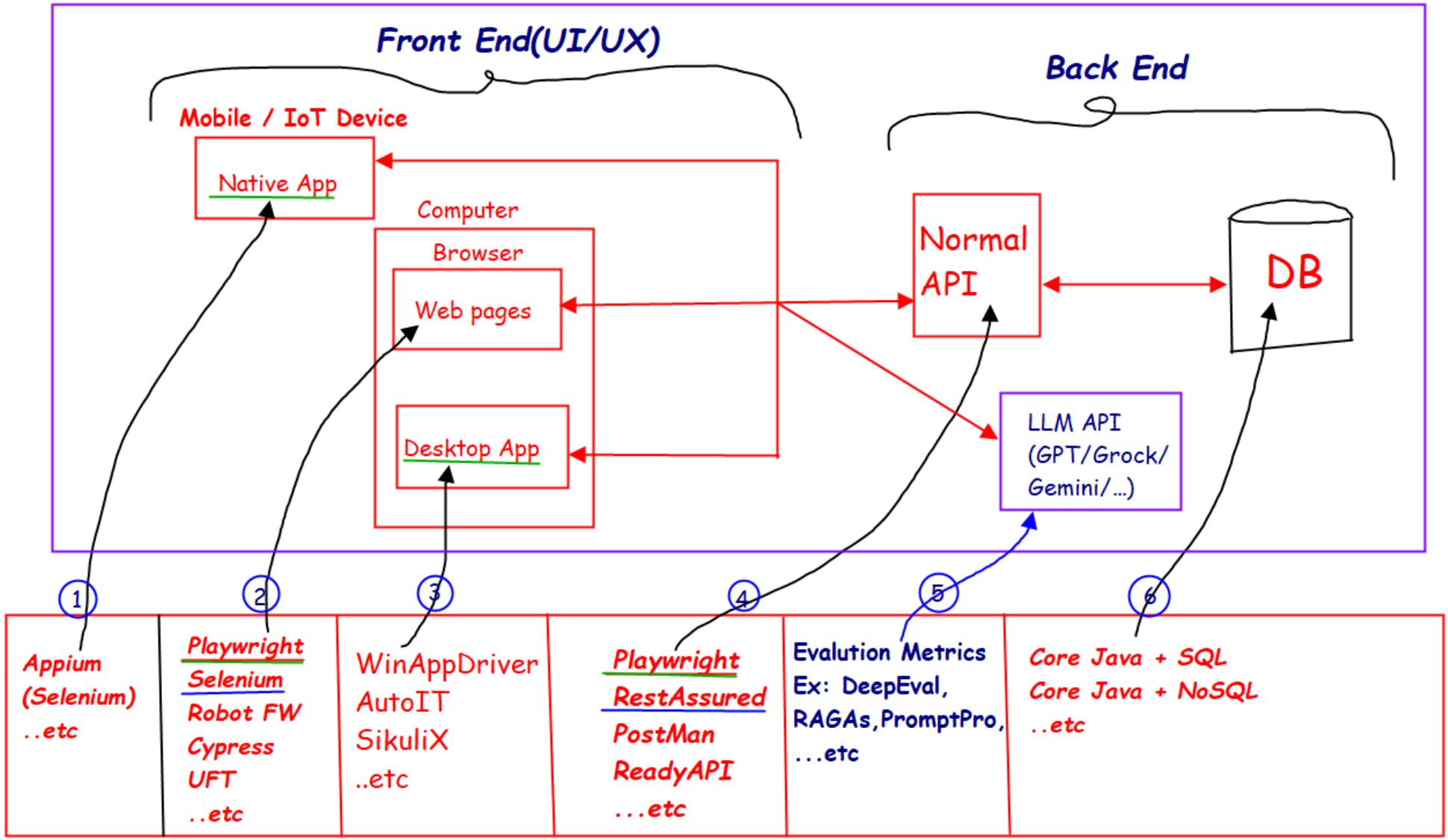
-->To automate end-to-end testing(Web UI+API) of modern web applications across "desktop and mobile browsers", we use "Playwright".
--> It is fully open-source and backed by "Microsoft", with no license cost.
--> Playwright supports multiple languages: Java, Python, C#, and JavaScript/TypeScript.
--> Playwright-Java is gaining popularity for its fast execution, stable automation, and first-class support for modern web features.

k. Playwright Components:

1. Playwright Core API
-> Enables writing tests using Browser, Context, and Page objects.
-> Offers features like auto-wait, rich locators, and headless testing.

2. Playwright Test Runner
-> Comes built-in for JavaScript/TypeScript.
-> Supports "parallel execution", "retries", and "fixtures".
-> For Java, use with "JUnit" or "TestNG".

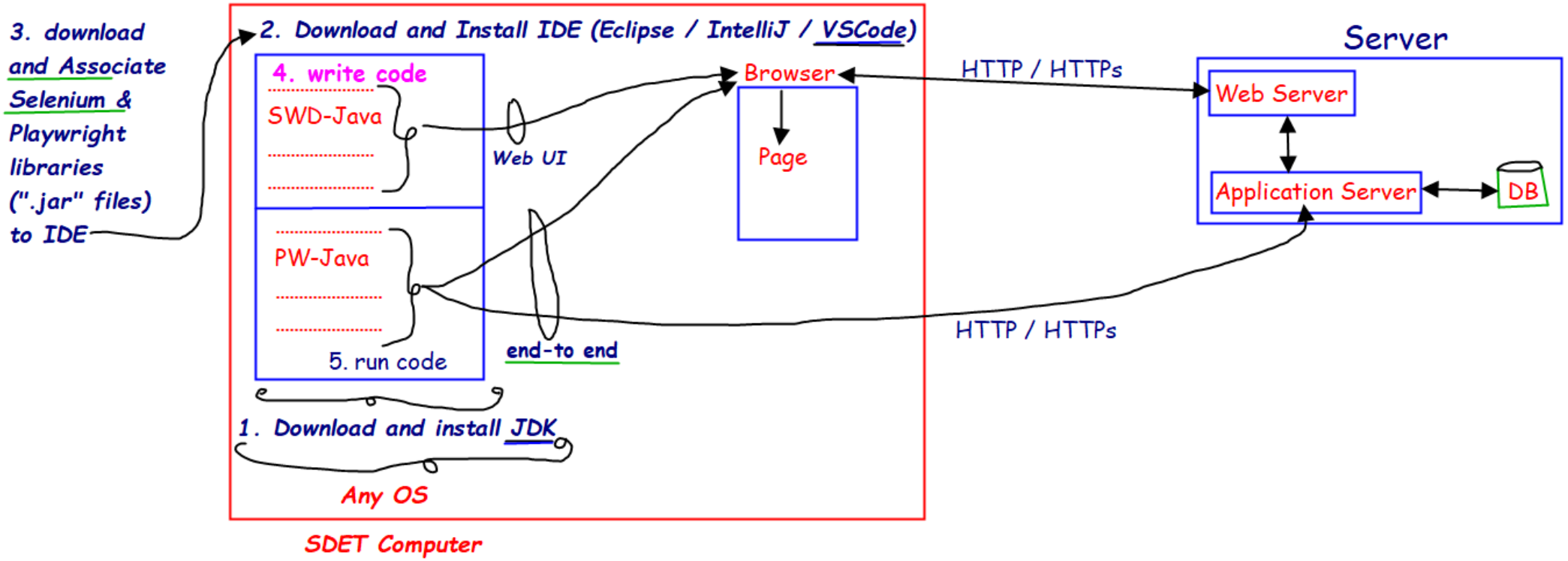
l. List of Tools in Functional Test Automation:



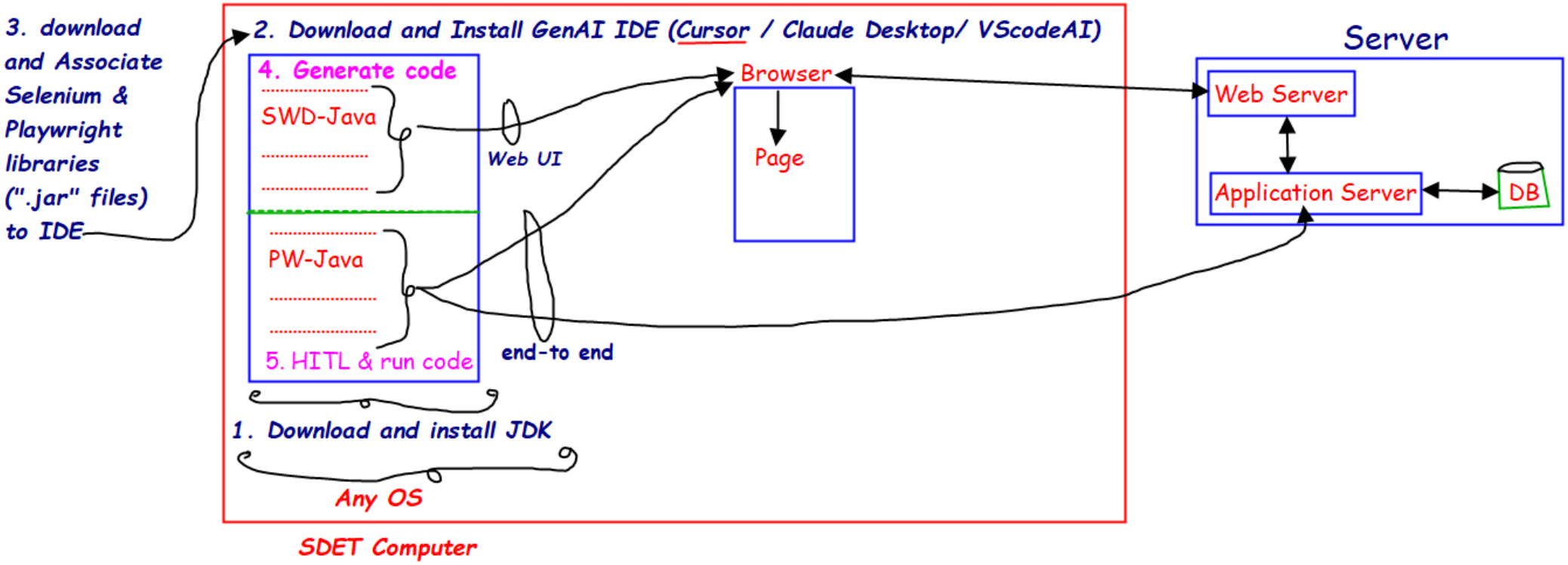
SDET Technology stack for Functional Tesing

Lesson: Playwright - Java

a. Prerequisites to use Playwright or Selenium:

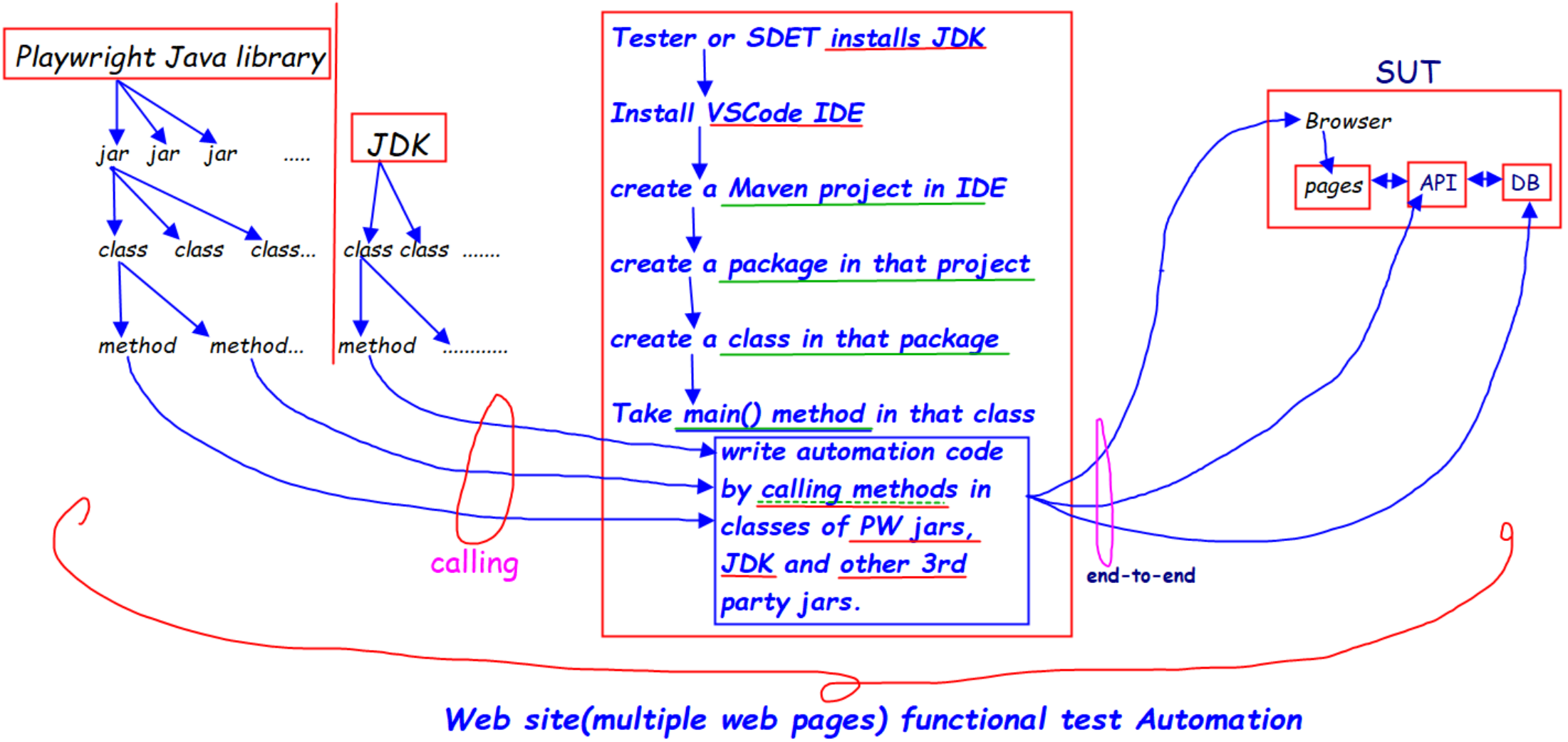


b. Prerequisites to use Playwright or Selenium with GenAI:



HITL -> Human In The Loop

c. Implement Playwright - Java code:



d). Download, install, and configure JDK(11 or 17 or 20 or 21 or 22 or other):-

Step-1: Create a new folder in our computer as a single word name, ex: E:\batch271

Step-2: Goto google site -> enter "JDK23 download" and hit enter -> goto oracle site -> click on jdk link as per OS and bitsize (ex: jdk-23.0.2_windows-x64_bin.exe) -> accept license agreement -> click on download -> login to oracle site (or) create account and login to oracle site to start download -> paste that downloaded jdk file in previously created folder (ex: E:\batch271)

Step-3: dbl click on that file -> Click on "yes" -> click on "next" until "close"

Step-4: Goto "C:\Program Files\Java\jdk-23" folder and copy path -> right click on "Thic PC" -> properties -> Advanced system settings -> environment variables -> Goto system variables -> click on "New" -> enter "JAVA_HOME" as variable name and "C:\Program Files\Java\jdk-23" as value -> click on OK -> select "Path" variable -> edit -> click on new -> paste "C:\Program Files\Java\jdk-23\bin" -> click OK -> -> click OK -> click OK -> Close properties window.

Note-1: To cross check the success of JDK's installation, we have to run below cmds at cmd prompt:

```
javac
```

```
java -version
```

Note-2: Download and Configure Maven software

1. Go to the official site: "https://maven.apache.org/download.cgi" -> Download the "binary zip archive" (ex: apache-maven-3.9.9-bin.zip) -> Paste that downloaded zip file in our personal folder (ex: E:\batch271) -> Extract that zip file using "right click" with "extract here" -> Goto that extracted folder and copy path(ex: E:\batch271\apache-maven-3.9.9)

2. Right click on "Thic PC" -> properties -> Advanced system settings -> environment variables -> Goto "system" variables -> click on "New" -> enter "MAVEN_HOME" as variable name and "E:\batch271\apache-maven-3.9.9" as value -> click on OK

3. Select "Path" variable -> edit -> click on new -> paste "E:\batch271\apache-maven-3.9.9\bin" -> click OK -> -> click OK -> click OK -> Close properties window.

4. Open CMD prompt and run below cmd for verification:

```
mvn -version
```

e). Download and install VSCode:

Step 1: Go to the VSCode Download Page: "https://code.visualstudio.com/" -> Click on "Download for Windows" -> Paste that downloaded ".exe" file in our local folder (ex: E:\batch271) -> Dbl Click on that ".exe" file -> Accept License agreement -> Click on Next -> Select "Create Desktop Icon" checkbox -> Click on install. (Once installed, open VSCode by double-clicking on the desktop icon or from our applications menu)

Step 2: Install Required Extensions in VSCode
To make VSCode ready for Java and Maven projects, we need to install a few essential extensions. Press "Ctrl+Shift+X" to open the Extensions panel -> In the search bar, type "Java Extension Pack" -> Select the extension pack from the list and click Install(This pack includes essential Java tools like language support, Maven integration, Junit/TestNG tests runner and debugging features)

f). Create a Maven project:
Press "Ctrl+Shift+P" to open the Command Palette in VSCode -> Type "Maven: New Project" and select the option that appears -> Choose an archetype (for a simple project, select "No archetype") -> Enter company name as Group ID and project name as Artifact ID (Ex: Group ID: org.magnitia and Artifact ID: playwright-project1) -> Select Location to Save the Project -> Choose(ex: E:\batch271) or create a folder where the new Maven project will be saved.(VSCode will automatically generate a new Maven project with a basic structure and a pom.xml file)

g). Create a package:
Right-click on the java folder (under src/test/java) -> Select New Folder -> Enter the name of our new package. For example, we can name it "tests".

h). Create a class with main() method:

To create a class with a "main()" method under the "tests" package in our Maven project, follow these steps:

1. Right-click on the "tests" package folder -> Select New File and name the file "Test1.java" -> Write Code with the "main()" Method like shown below:

```
public class Test1 {
    public static void main(String[] args) {
        ------------------
    }
}
```

This class contains a `main()` method, which is the entry point for running the Java application. Finally, press "Ctrl+S" to save the file.

i). Associate Playwright Jars and write Code:

--> Associating Playwright JARs is essential for using the Playwright API in our Maven project.

--> Maven makes it easy to add and manage Playwright dependencies by simply adding the dependency in our "pom.xml" file.

--> Playwright JARs enable you to interact with browsers programmatically, and Maven will handle downloading and versioning for us.

--> Maven Repository (http://www.mvnrepository.com) is a central online repository that stores Java libraries (JAR files) and dependencies, making it easier for developers to find and include external libraries in their projects. It allows Maven to automatically download the required libraries for our project by specifying them in the "pom.xml" file.

Note-1:

Goto "mvnrepository.com" site -> serch for "Playwright java" -> click on search -> click on Playwright link -> click on stable version link (ex: 1.52.0) -> copy XML dependency code -> back to VSCode -> open "pom.xml" file -> paste that copied dependency code under "<dependencies>" tag under "<project>" tag like shown below:

```
<project ---------------------------------------->
    ------------------------------------
    ------------------------------------
  <dependencies>
    <dependency>
        <groupId>com.microsoft.playwright</groupId>
        <artifactId>playwright</artifactId>
        <version>1.52.0</version>
    </dependency>
  </dependencies>
</project>
```

-> click on save -> wait until 100% complete download -> Goto "Dependencies" under our maven project under "Maven" section in VSCode to observe all downloaded and associated jars.

Note-2: About Maven

--> Build management tool.

--> Useful to download required plugins and libraries for our daily work. For example, it downloads Playwright jars and other required jars automatically through internet.

--> Alternatives for Maven are Gradle, SBT, Ivy, Grape, Leiningen, Buildr, ...etc

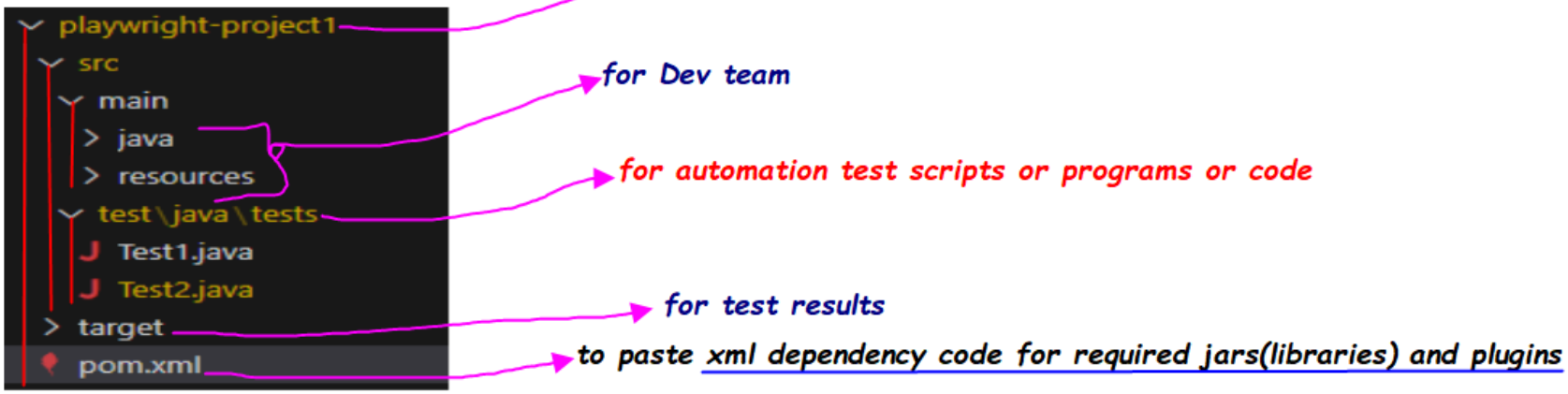
--> To say what we want, we have to use XML dependency code for Maven.

--> To get readymade xml code for required libraries and plugins, we have to visit "http://mvnrepository.com" site. Here, we have to search for xml code by clicking on stable version.

--> We have to paste copied XML code under <dependencies> tag under <project> tag in "pom.xml" file in our maven project.

--> Here, pom stands for "Project Object Model".

Note-3: About Maven Project



Note-4:

--> If we did not get right folder structure in our Maven project like shown in previous diagram, we have to Refresh Project in VSCode. To refresh the Maven project and ensure all the dependencies and folder structure are correct, Open the Command Palette (Ctrl+Shift+P) -> Search for Maven -> Reload Project and select it to reload the Maven project and sync any changes.

--> Alternatively, we can also close and reopen the project in VSCode to ensure everything is reloaded.

Note-5: Naming conventions in Java language

--> In Maven project, group id is company name in lowercase as single word(no space).

--> In Maven project, artifact id is project name in lowercase as single word(no space).

--> In Maven project, package is module name in lowercase as single word(no space).

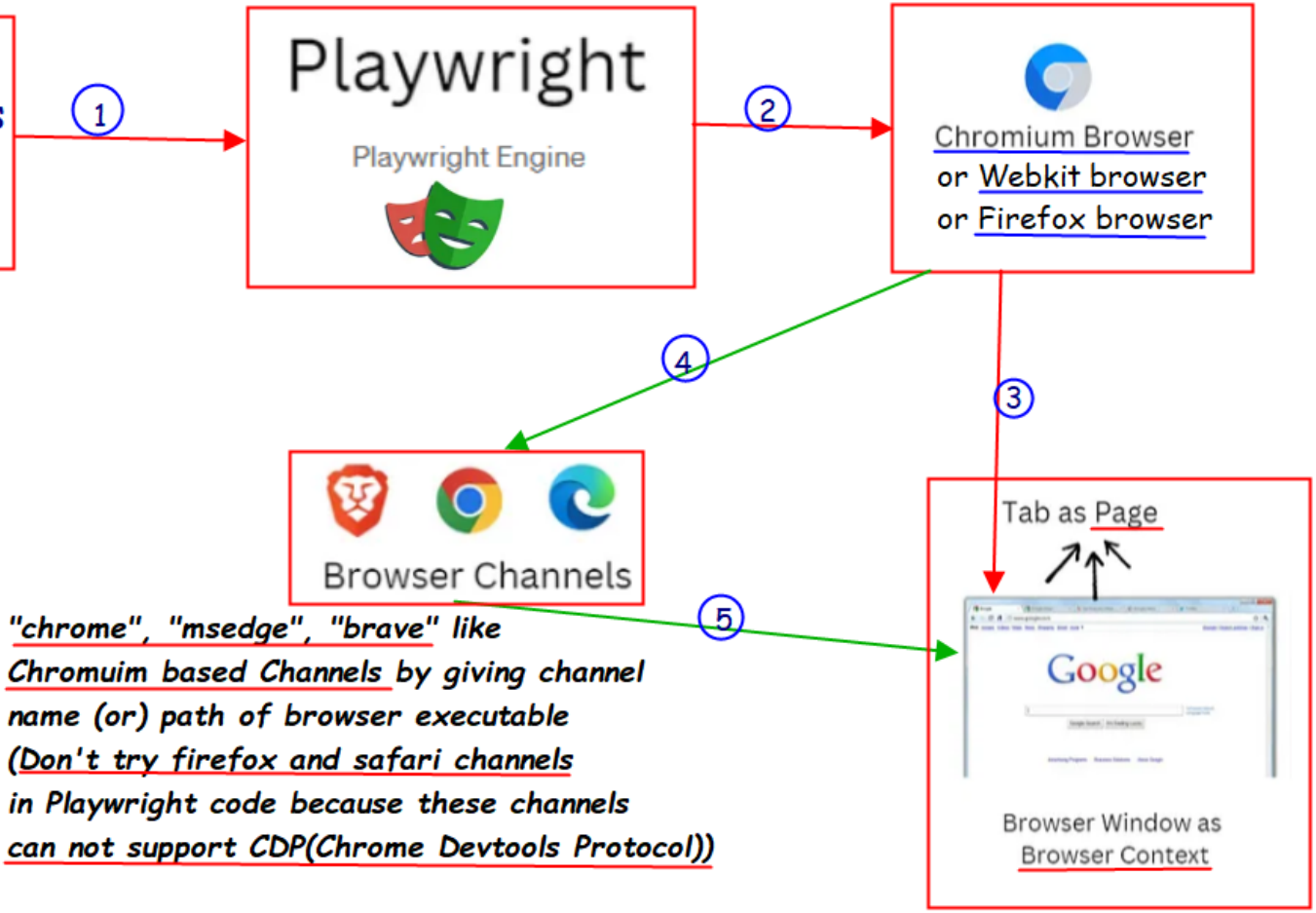
--> In Maven project, class is any name but start with upper case, as single word(no space), and allow upper case in middle. ex: "PlaywrightContext" is a class name

--> In Maven project, method is any name but start with lower case, as single word(no space), and allow upper case in middle. ex: "getText()" , getX(), getY(), ...etc are method names.

--> "Camel case" means we are able to use upper case in middle of the name of classes and methods.

Note-6: Playwright Execution flow

Run Playwright-Java code in main() method in a class in src/test/java/tests package in maven project



"chrome", "msedge", "brave" like Chromuim based Channels by giving channel name (or) path of browser executable (Don't try firefox and safari channels in Playwright code because these channels can not support CDP(Chrome Devtools Protocol))

Flow-1: 1 -> 2 -> 3 to run code on a browser

Flow-2: 1 -> 2 -> 4 -> 5 to run code on a browser chennel

Note-6:

1. Activate Playwright Engine:

-> The foundation layer of the automation stack.

-> In Java, it is initialized via:

```
Playwright pw=Playwright.create();
```

-> This engine handles launching browsers, communicating via WebSocket/Chrome DevTools Protocol, Managing execution flow and browser interactions

2. Browser Channels

-> Represents different brands or variants of supported browsers like Google Chrome, Microsoft Edge, Brave, ..etc

-> Each channel refers to a different executable binary.

-> We can choose a browser channel in Java:

```
BrowserType.LaunchOptions opt=new BrowserType.LaunchOptions();
Browser br=pw.chromium().launch(opt);
```

(OR)

```
BrowserType.LaunchOptions opt=new BrowserType.LaunchOptions().setChannel("chrome");
Browser br=pw.chromium().launch(opt);
```

3. Browser Window as Browser Context

-> A "BrowserContext" emulates a fresh browser window with isolated storage.
-> It acts like a new incognito session.
-> Great for parallel test runs and session separation.
-> In Java:

```
BrowserContext bc=br.newContext();
```

4. Tab as Page

-> Inside each "BrowserContext", We can open multiple "Page" instances.
-> A Page is equivalent to a browser tab.
-> It's the main interface for actions:
  * Navigation
  * Element interaction
  * Screenshot capture
-> In Java:

```
Page p=bc.newPage();
p.navigate("https://www.google.com");
```

Ex-1: Playwright Java code performs a basic browser automation flow using Playwright's bundled Chromium:

```
// Bottom Layer: Playwright Engine
Playwright pw=Playwright.create();

// Launch Chromium browser as visible on desktop
BrowserType.LaunchOptions lo=new BrowserType.LaunchOptions();
lo.setHeadless(false);
Browser br=pw.chromium().launch(lo);

// Browser Context: Acts like a new browser window (fresh session)
BrowserContext bc=br.newContext();

// Tab as Page: Each page is a tab within the browser window/context
Page p=bc.newPage();

// Navigate to a URL
p.navigate("https://www.google.com");
```

Ex-2: Playwright Java code performs a basic browser automation flow using Playwright's bundled firefox:

```
// Bottom Layer: Playwright Engine
Playwright pw = Playwright.create();

// Launch firefox browser
BrowserType.LaunchOptions lo=new BrowserType.LaunchOptions();
lo.setHeadless(false); // Run with UI
Browser br=pw.firefox().launch(lo);

// Browser Context: Acts like a new browser window (fresh session)
BrowserContext bc=br.newContext();

// Tab as Page: Each page is a tab within the browser window/context
Page p=bc.newPage();

// Navigate to a URL
p.navigate("https://www.google.com");
```

Ex-3: Playwright Java code performs a basic browser automation flow using Playwright's bundled webkit:

```
// Bottom Layer: Playwright Engine
Playwright pw = Playwright.create();

// Launch webkit browser
BrowserType.LaunchOptions lo=new BrowserType.LaunchOptions();
lo.setHeadless(false); // Run with UI
Browser br=pw.webkit().launch(lo);

// Browser Context: Acts like a new browser window (fresh session)
BrowserContext bc=br.newContext();

// Tab as Page: Each page is a tab within the browser window/context
Page p=bc.newPage();

// Navigate to a URL
p.navigate("https://www.google.com");
```

Ex-4: Playwright Java code launches system-installed Google Chrome via the channel mechanism instead of the default bundled Chromium by giving channel name.

```
// Bottom Layer: Playwright Engine
Playwright pw = Playwright.create();

// Browser Channels: Launch Chrome browser specifically via channel
BrowserType.LaunchOptions lo = new BrowserType.LaunchOptions();
lo.setHeadless(false);                    // Run with UI
lo.setChannel("chrome");                  // Specific browser channel (chrome, msedge, etc.)
Browser br = pw.chromium().launch(lo);

// Browser Context: Acts like a new browser window (fresh session)
BrowserContext bc = br.newContext();

// Tab as Page: Each page is a tab within the browser window/context
Page p = bc.newPage();

// Navigate to a URL
p.navigate("https://www.google.com");
```

Ex-5: Chrome channel by giving executable path of chrome

```
// Bottom Layer: Playwright Engine
Playwright pw = Playwright.create();
//Change path as per required
String chromePath = "C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe";
// Browser Channels: Launch Chrome browser specifically via channel with executable path
BrowserType.LaunchOptions lo = new BrowserType.LaunchOptions();
lo.setHeadless(false);                          // Run with UI
lo.setExecutablePath(Paths.get(chromePath)); // Use system Chrome
// Launch the browser using the specified path
Browser br = pw.chromium().launch(lo);

// Browser Context: Acts like a new browser window (fresh session)
BrowserContext bc = br.newContext();

// Tab as Page: Each page is a tab within the browser window/context
Page p = bc.newPage();

// Navigate to a URL
p.navigate("https://www.google.com");
```

Ex-6: by giving channel name like "msedge"

```
// Bottom Layer: Playwright Engine
Playwright pw=Playwright.create();

// Browser Channels: Launch Edge browser specifically via channel
BrowserType.LaunchOptions lo=new BrowserType.LaunchOptions();
lo.setHeadless(false);                    // Run with UI
lo.setChannel("msedge");                  // Specific browser channel (chrome, msedge, etc.)

// Launch the browser in system
Browser br=pw.chromium().launch(lo);

// Browser Context: Acts like a new browser window (fresh session)
BrowserContext bc=br.newContext();

// Tab as Page: Each page is a tab within the browser window/context
Page p=bc.newPage();

// Navigate to a URL
p.navigate("https://www.google.com");
```

Ex-7: Cross-browser(if-else if-else)

```
//1. Take Browser name from keyboard
Scanner sc=new Scanner(System.in);  // System.in represents Keyboard
System.out.println("Enter a bundelled browser name or channel name");
String bn=sc.nextLine();
sc.close();
//2. Bottom Layer: Playwright Engine
Playwright pw=Playwright.create();
//3. Declare an object to "Browser" as Null
Browser br=null;
//4. Cross-browser code
if (bn.equalsIgnoreCase("Chromium")) {
            br = pw.chromium().launch(new BrowserType.LaunchOptions().setHeadless(false));
} else if (bn.equalsIgnoreCase("chrome")) {
            br = pw.chromium().launch(new BrowserType.LaunchOptions()
                        .setHeadless(false)
                        .setChannel("chrome")); // Launch system-installed Google Chrome
} else if (bn.equalsIgnoreCase("msedge")) {
            br = pw.chromium().launch(new BrowserType.LaunchOptions()
                        .setHeadless(false)
                        .setChannel("msedge")); // Launch system-installed Microsoft Edge
}
```

```
        else if (bn.equalsIgnoreCase("Firefox"))
        {
            br = pw.firefox().launch(
                    new BrowserType.LaunchOptions().setHeadless(false));
         }
        else if (bn.equalsIgnoreCase("Webkit"))
        {
            br = pw.webkit().launch(
                    new BrowserType.LaunchOptions().setHeadless(false));
         }
         else
         {
            System.out.println("Invalid browser name! Use one of: chromium, chrome, firefox, msedge, webkit.");
            System.exit(0); // 0 means stop execution forcibly
         }
         // Step-5: Perform basic navigation
         Page p = br.newPage();
         p.navigate("https://www.google.com");
         System.out.println("Page Title in " + bn + ": " + p.title());
         // Step-6: Close Browser and Scanner
         br.close();
    }
}
```

Ex-8: Cross-browser(switch case)

```
// Step 1 : Take browser input from user
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter browser name (chromium, chrome, firefox, msedge, webkit): ");
        String bn = sc.nextLine();
        sc.close();
        // Step-2: start Playwright engine
        Playwright pw = Playwright.create();
        // Step-3: Declare Browser object as Null
        Browser br = null;
        // Step-4: Cross browser code
        switch (bn) {
            case "chromium":
                br = pw.chromium().launch(new BrowserType.LaunchOptions().setHeadless(false));
                break;
            case "chrome":
                br = pw.chromium().launch(new BrowserType.LaunchOptions()
                        .setHeadless(false).setChannel("chrome")); // Launch system-installed Chrome
                break;
            case "msedge":
                br = pw.chromium().launch(new BrowserType.LaunchOptions()
                        .setHeadless(false).setChannel("msedge")); // Launch system-installed Microsoft Edge
                break;
```

```
            case "firefox":
                br = pw.firefox().launch(new BrowserType.LaunchOptions().setHeadless(false));
                break;
             case "webkit":
                br = pw.webkit().launch(new BrowserType.LaunchOptions().setHeadless(false));
                break;
             default:
                System.out.println(
                        "Invalid browser name! Use one of: chromium, chrome, firefox, msedge, webkit.");
                System.exit(0);
        }
        // Step 2: Perform basic navigation
        Page p = br.newPage();
        p.navigate("https://www.google.com");
        System.out.println("Page Title in " + bn + ": " + p.title());
        // Close site
        br.close();
    }
}
```

CaseStudy-1:

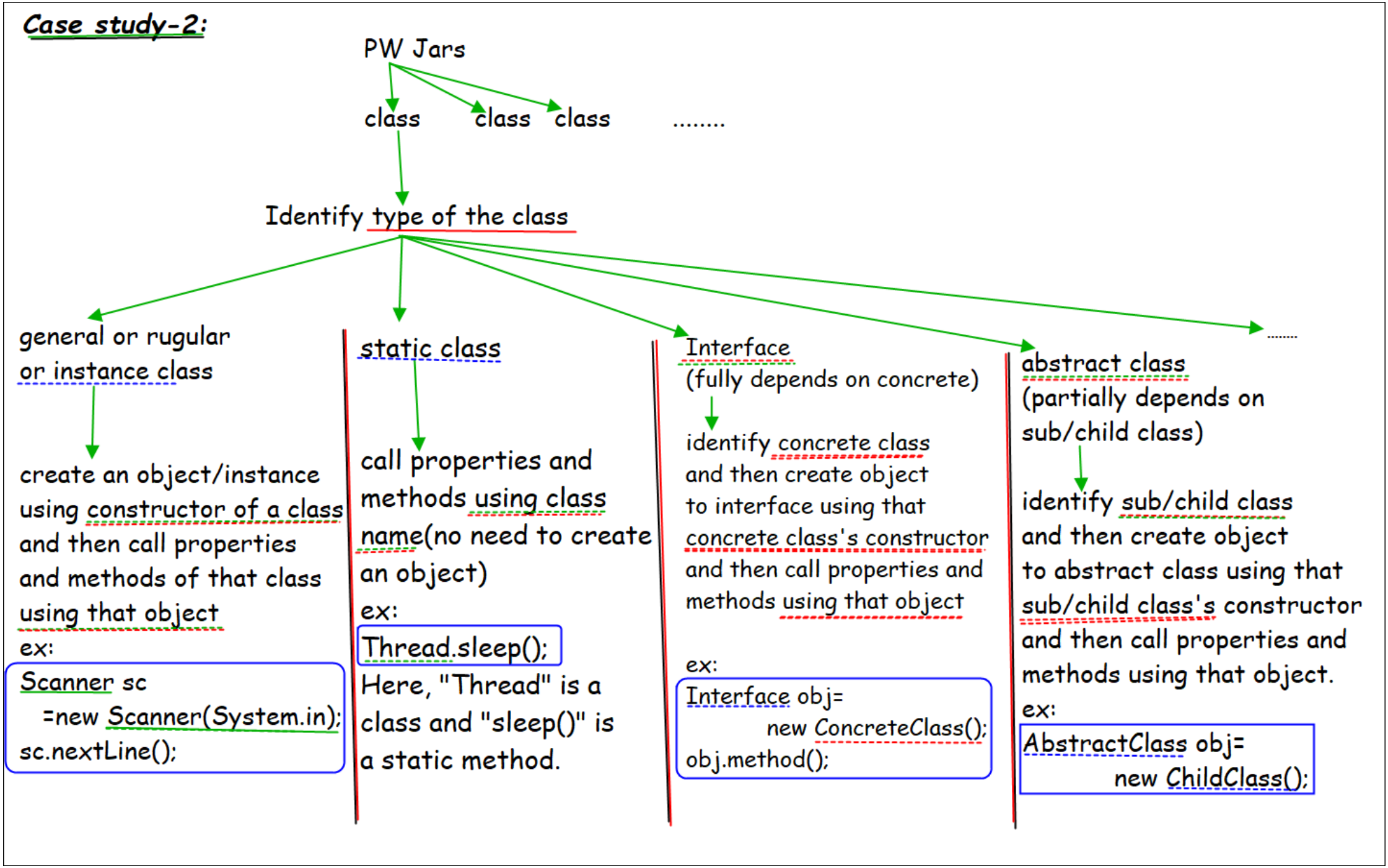
--> As per our earlier discussion, PW jars are having classes but in various types like described below.

--> While testing a website, we call methods of classes of PW jars in our daily automation scripts/code.

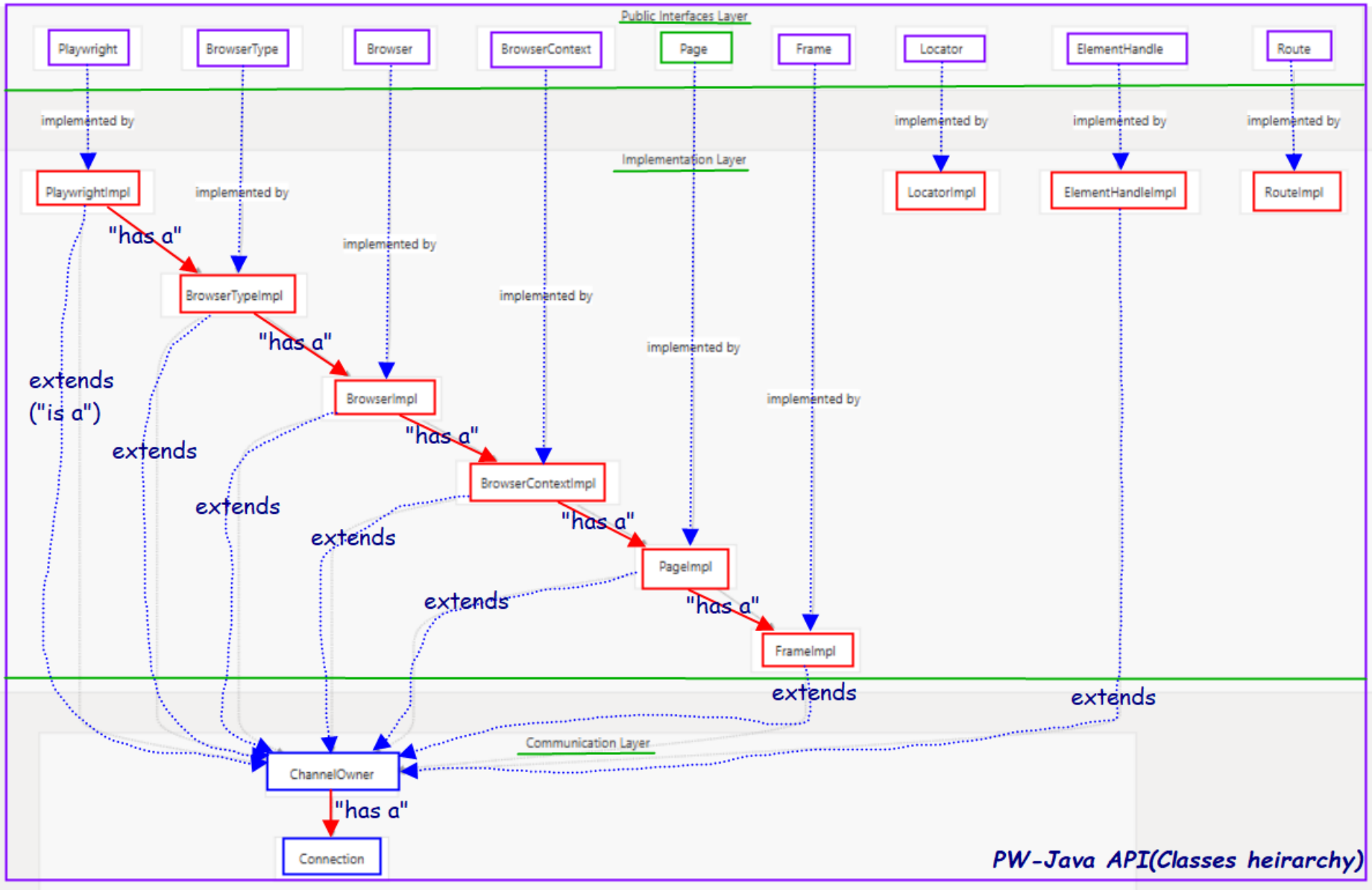
--> To call methods of a class, first of all we have to know the type that class.

1. *Instance Class: A blueprint for creating objects, containing fields(properties or data members), methods, and constructors.*
2. *Abstract Class: A class that cannot be instantiated and may contain abstract methods(without bodies) that subclasses must implement.*
3. *Interface: A reference type in Java, used to specify methods that a class must implement, with no method bodies or implementations.*
4. *Enum: A special class that represents a group of constants (unchangeable variables).*
5. *Anonymous Class: A class without a name, defined and instantiated in a single statement.*
6. *Local Class: A class defined within a method or a block, accessible only within that scope.*
7. *Static Nested Class: A static class defined inside another class, which can be instantiated without an instance of the enclosing class.*
8. *Inner Class: A non-static class defined inside another class, which can access members of the enclosing class.*
9. *Singleton Class: A class designed to ensure that only one instance of the class is created and provides a global access point to that instance.*
10. *Final Class: A class that cannot be subclassed (extended).*
11. *...etc*

Case study-2:


PW Jars
class
class
class
.....
Identify type of the class
general or rugular or instance class
create an object/instance using constructor of a class and then call properties and methods of that class using that object
ex:
Scanner sc =new Scanner(System.in);
sc.nextLine();
static class
call properties and methods using class name(no need to create an object)
ex:
Thread.sleep();
Here, "Thread" is a class and "sleep()" is a static method.
Interface
(fully depends on concrete)
identify concrete class and then create object to interface using that concrete class's constructor and then call properties and methods using that object
ex:
Interface obj= new ConcreteClass();
obj.method();
abstract class
(partially depends on sub/child class)
identify sub/child class and then create object to abstract class using that sub/child class's constructor and then call properties and methods using that object.
ex:
AbstractClass obj= new ChildClass();
.....

Case Study-3: list of important interfaces and classes in PW-Java to automate Web Pages



Case Study-4: What is a Class?

-> A "class" is like a "template" or "blueprint" for creating things (called "objects"). Think of it like this:

1. A class is like a "design" for a house.
2. An object is the "actual house" built from that design.

Ex:

```
class Student {
    // Properties (data)
    String name;
    // Action (method)
    void study() {
        System.out.println(name + " is studying.");
    }
}
```

(class like a template
ex: all classes in PW jars)

```
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();  // creating an object
        s1.name = "Kalam"; //access property
        s1.study();  // calling the action(method)
    }
}
```

(class is runnable because
it has main() method. Here
we have to call properties and
methods of class(like a template)
by using an object.
ex: tests written by us)

In Simple Words:

Term	Real-life Example
Class	Recipe or design (e.g., "Student")
Object	Actual thing made (e.g., s1)
Variable	Information (e.g., name, ...etc)
Method	Action it can do (e.g., study, ...etc)

4.1.What is an Interface?

-> An "interface" is like a "promise" or a "contract". It says:
-> "Any class that uses me must implement these actions". But it "doesn't say how" — just "what" should be done.
-> Think of a "TV Remote Interface": It says every remote must have:

```
turnOn()
turnOff()
```

But the "actual remote"(Sony, LG, Samsung...) decides "how" those actions work.
-> Step 1: Define an Interface

```
interface Remote {
    void turnOn();
    void turnOff();
}
```

This says:

> Any class that uses Remote must have `turnOn()` and `turnOff()` methods.

Step 2: A Class Implements It

```
class TVRemote implements Remote {
    public void turnOn() {
        System.out.println("TV is ON");
    }
    public void turnOff() {
        System.out.println("TV is OFF");
    }
}
```

```
class ACRemote implements Remote {
    public void turnOn() {
        System.out.println("AC is ON");
    }
    public void turnOff() {
        System.out.println("AC is OFF");
    }
}
```

Step 3: Use It

```
public class Main {
    public static void main(String[] args) {
        Remote r1 = new TVRemote();  // interface reference
        r1.turnOn();
        Remote r2 = new ACRemote();  // interface reference
        r2.turnOff();
    }
}
```

--> In general, interfaces are having non-static methods witout bodies/implementations.
--> Few interfaces are having static methods also with bodies as manditory.
Ex:

```
// Source code is decompiled from a .class file using FernFlower decompiler.
public interface Playwright extends AutoCloseable {
   //non-static methods without bodies/implementations
   BrowserType chromium();
   BrowserType firefox();
   APIRequest request();
   Selectors selectors();
   BrowserType webkit();
   void close();
   //static methods with bodies/implementations
   static Playwright create(CreateOptions options) {
      return PlaywrightImpl.create(options);
   }
   static Playwright create() {
      return create((CreateOptions)null);
   }
}
```

--> So, call non-static methods with the help of concrete class. But call static methods via interface name direcctly like:
Playwright pw=Playwright.create();

Summary

Term	Meaning
interface	List of methods with no body (just rules)
implements	Keyword used by class to follow the interface
Remote r = new TVRemote();	Object using interface reference

4.2. What is an Abstract Class?

--> An abstract class is like a half-built class.
--> It gives some common features, but also leaves some parts for the child class to complete.

ex:
Imagine a class called Animal. We know:

All animals eat
All animals make sounds, but each makes it differently

So we can write:
✅ a method for eating (common for all)
❌ but leave sound method undefined

Step 1 : Abstract Class

```
abstract class Animal {
    // Common method
    void eat() {
        System.out.println("This animal eats food.");
    }

    // Abstract method (no body)
    abstract void makeSound();
}
```

Step 2: Child class Completes the Missing Part

```
class Dog extends Animal {
    void makeSound() {
        System.out.println("Dog barks: Woof!");
    }
}
```

Step 3: Use It

```
public class Main {
    public static void main(String[] args) {
        // Animal is abstract, cannot be created directly
        Animal a = new Dog();
        a.eat();              // uses concrete method
        a.makeSound();        // uses Dog?s version
    }
}
```

Output:

```
This animal eats food.
Dog barks: Woof!
```

Summary:

Term	Meaning
abstract class	A class that cannot be fully used directly
abstract method	A method with "no body", must be completed by child
extends	Child class uses the abstract class

Use an "abstract class" when:

-> You want to share some code
-> But "force" child classes to complete certain methods

4.3 "extends" vs "implements"

In Java, extends and implements are two different keywords used for inheritance, but they serve different purposes depending on whether you're working with a class or an interface.

extends: For "class-to-class" or "interface-to-interface" inheritance

Ex-1. When a class extends a class:

-> Used for single inheritance (Java doesn't support multiple inheritance with classes).

-> The child class inherits fields and methods (except private ones) from the parent class.

```
class Animal {
    void sound() {
        System.out.println("Some sound");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("Bark!");
    }
}
```

```
class X{
    ----------
}
class Y{
    ----------
}
class Z extends X, Y{
 -----------
}
```

Ex-2. When an interface extends another interface:

-> One interface can extend multiple interfaces.

-> Used to build a more complex interface by combining others.

```
interface Walkable {
    void walk();
}
interface Runnable extends Walkable {
    void run();
}
```

```
interface X{
    ----------
}
interface Y{
    ----------
}
interface Z extends X, Y{
 -----------
}
```

implements: For "class-to-interface" inheritance

-> A class implements an interface to provide concrete definitions(bodies) for its abstract methods.

-> A class can implement multiple interfaces (Java supports multiple interface inheritance).

```
interface Flyable {
    void fly();
}
class Bird implements Flyable {
    public void fly() {
        System.out.println("Flying...");
    }
}
```

```
interface X{
    ----------
}
interface Y{
    ----------
}
class Z implements X, Y{
  ----------
}
```

Comparison Table

Feature	`extends`	`implements`
Used with	Class → Class OR Interface → Interface	Class → Interface(s)
Multiple allowed?	❌ (for classes) / ✅ (for interfaces)	✅ Multiple interfaces
Purpose	Inherit class or interface behavior	Implement behavior of interfaces
Method implementation	Optional (abstract class) or inherited	Mandatory (unless abstract class)
Example	`class A extends B`	`class A implements C, D`

Case study-5: (About Constructor method in Java language)

--> If our class in JDK or in jars is an "instance class", we have to create an object to that class.
--> This type of class contains concrete methods(with bodies), fields(properties), and constructors.
--> Here, constructors are useful only to create an object to corresponding class.
--> Ex:

```
public class Student{
        //1. Properties or fields
        public int rollno;
        public String name;
        public int age;
        public char gender;   //ex: M or F
        ---------------
        ---------------
        //2. Constructor method
        public Student(){
           ---------------------
           ---------------------
        }
        //3. General methods
        ---------------
        ---------------
}
```

```
public class Runner {
    public static void main(String[] args){
        Student obj=new Student();
    }
}
```

--> If no constructor method in a class, JRE provides default constructor to create an object to corresponding class.

--> For example,

```
public class Sample
{
    int x;
    public void method1(){
        System.out.println("hi");
    }
}
```

```
public class Runner {
    public static void main(String[] args){
        Sample obj=new Sample(); //default constructor
    }
}
```

--> If a class has a constructor but it is not public, then we have to find a child or sub class to create an object to corresponding class using that child or sub class's constructor method.

--> For example,

```
RemoteWebDriver driver=new FirefoxDriver();
```

class name — object name (any name) — keyword to create an object — constructor method of child/sub class

--> If we want to create an object to an interface(having methods without bodies), we have to find a concrete class (which provides bodies to methods of interface) to create an object to interface using that concrete class's constructor method.

--> For example,

```
WebDriver driver=new ChromeDriver();
```

--> If we want to create an object to an interface(having methods without bodies), we have to find a concrete class (which provides bodies to methods of interface) to create an object to interface using that concrete class's constructor method. But the constructor method of concrete class is not public.

--> For example,

```
Playwright obj=new PlaywrightImpl();
```

"PlaywrightImpl()" constructor in "PlaywrightImpl" class is not public

--> But "Playwright" interface has few public static methods with bodies to create an object to "Playwright" itself.

```
Playwright pw = Playwright.create();
```

Here, the static methods with bodies in "Playwright" interface are not showing "public" keyword.

🧠 The Truth: You Are Seeing the **Final Source After Compilation Adjustments**

✨ Microsoft Playwright Java library is built using:

- JDK tools, then packaged into `.jar`
- In the final `.class` file, the `create()` methods are marked `public static`
- You're seeing a cleaned-up version or a custom build tool output
  Ex: FernFlower decompiler

CaseStudy-6: Access Modifiers

🔷 1. Top-Level Classes and Interfaces

Modifier	Accessible Within Same Package	Accessible Outside Package
`public`	✅ Yes	✅ Yes
no modifier (default/package-private)	✅ Yes	❌ No

> ⛔ Only `public` or *no modifier* is allowed for **top-level** classes/interfaces.
> `private` and `protected` are **not allowed**.

🔷 2. Inner Classes / Interface Members (Fields and Methods)

Modifier	Same Class	Same Package	Subclass (other package)	Other Classes
`public`	✅	✅	✅	✅
`protected`	✅	✅	✅	❌
no modifier (default)	✅	✅	❌	❌
`private`	✅	❌	❌	❌

--> In general in classes, Dev uses all types of access modifiers as per requirements:

Member Type	`public`	`protected`	*no modifier*	`private`
Fields	✅	✅	✅	✅
Methods	✅	✅	✅	✅
Constructors	✅	✅	✅	✅
Inner Class	✅	✅	✅	✅

--> In general in intefaces, Dev uses below types of access modifiers as per requirements:

Member Type	Modifier	Default if None Specified	Notes
Field	`public static final`	✅ Implicit	Constants only
Method (abstract)	`public abstract`	✅ Implicit	Must be implemented
Method (default)	`public`	❌ Must be explicit	Can have body
Method (static)	`public`	❌ Must be explicit	Can have body
Method (private)	`private`	❌ Must be explicit	Internal helper

CaseStudy-7: Scope of Variable or object in Java

--> If variables declared inside a method, block, or constructor, then those variables are only accessible within that specific method or block where they are declared. Here, the variable exists from the point of declaration until the method or block is exited.

ex-1:

```
void myMethod(){
    int x = 5;  // x has local scope
    System.out.println(x);  // Can be accessed here
}
// x cannot be accessed here
----------
----------
```

ex-2:

```
int x; // x has global scope
void myMethod() {
    x = 5;
    System.out.println(x);  // Can be accessed here
    -----------
}
// x can be accessed here
System.out.println(x);
----------
```

CaseStudy-8: "is a" vs "has a"

```
Class X
{
   int a;
}
```

```
class Y extends X
{
   int b;
}
```

```
Y obj=new Y();
obj.b;
obj.a;
```

"Is a" relation
(Inheritance)

```
Class X
{
   int a;
}
```

```
class Y
{
   X obj1;
   int b;
}
```

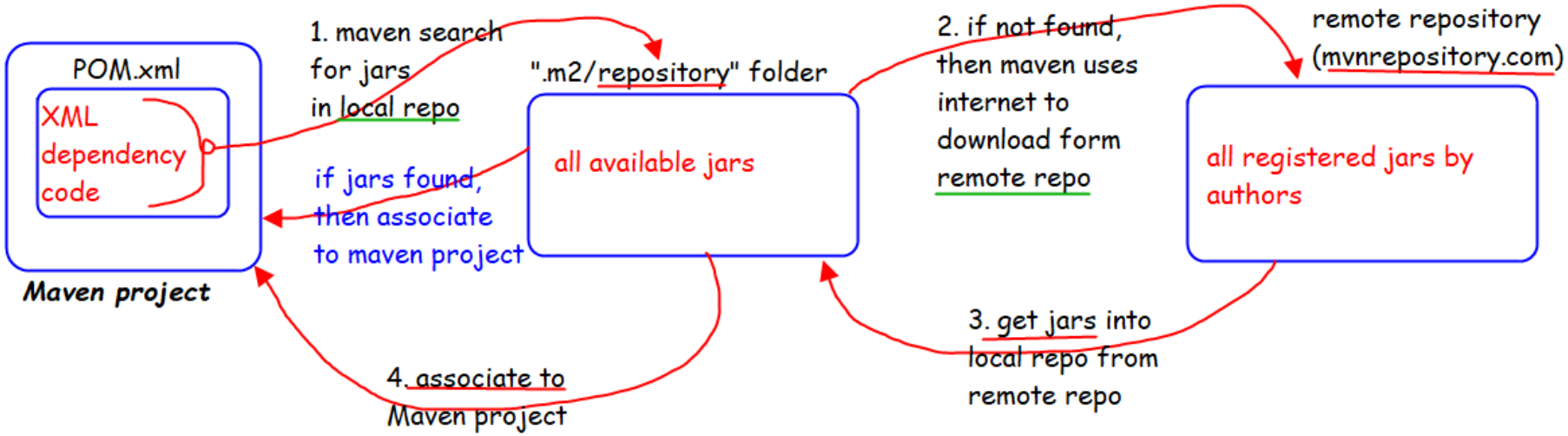
```
Y obj2=new Y();
obj2.b;
obj2.obj1.a;
```

```
class Z
{
   Y obj3;
}
```

```
Z obj4=new Z();
obj4.obj3.obj1.a;
```

"Has a" relation

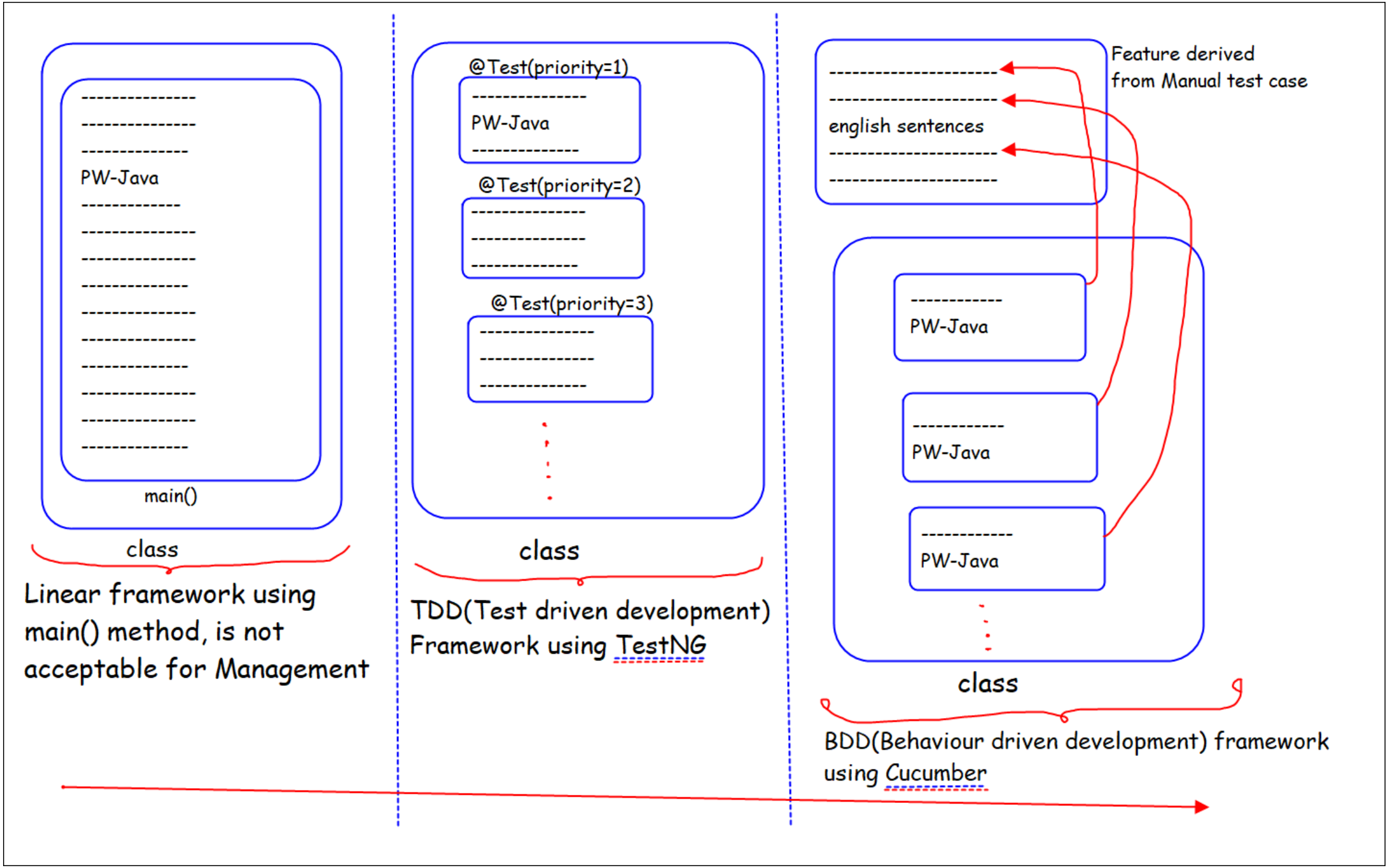
Case Study-9: Job of Maven software

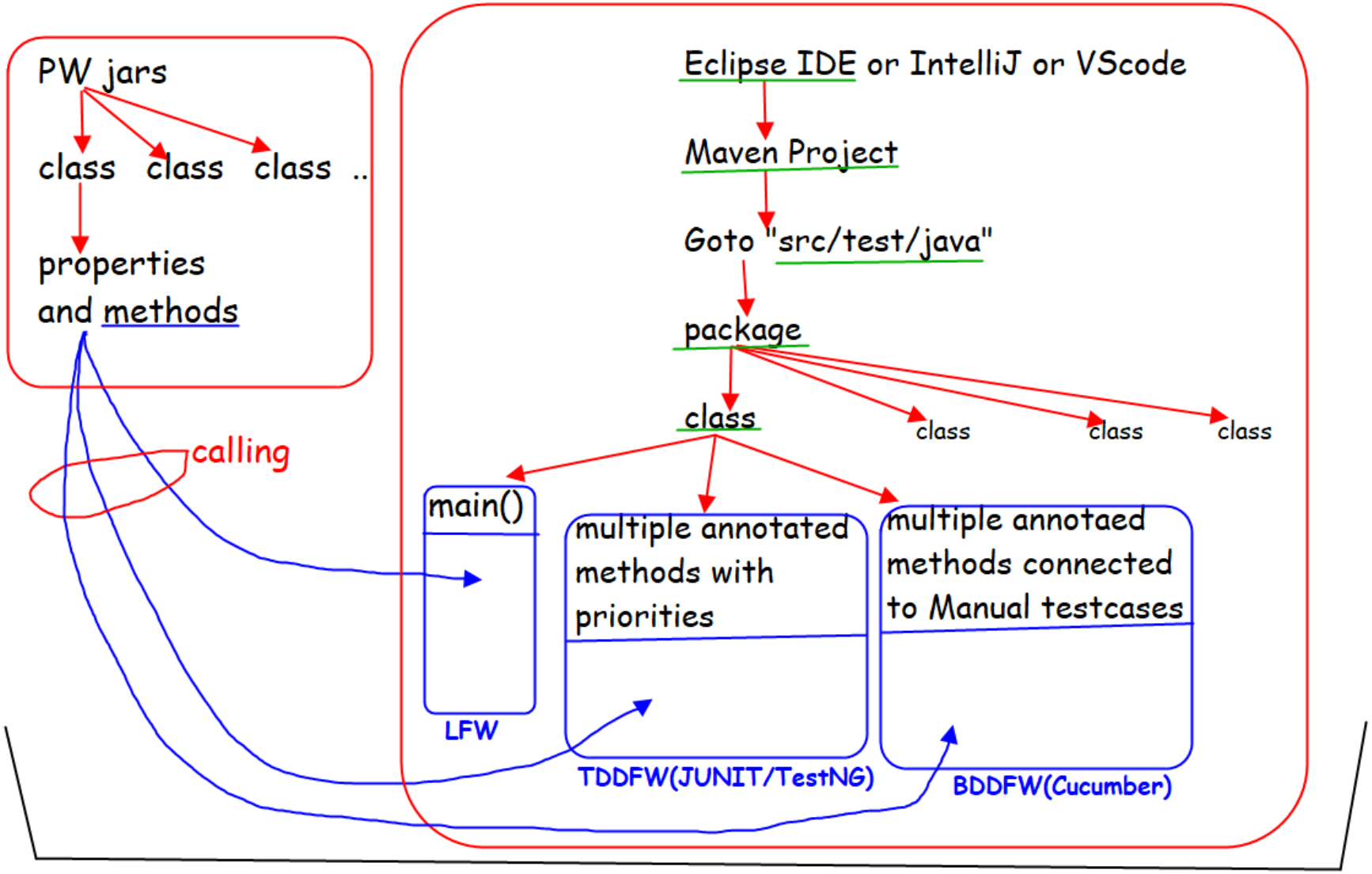


--> If jars found in local repo("C:\users\xxxxx\.m2\repository"), Maven can not connect to Remote repo

Case Study-10: Framework:

--> For code modularity(multiple small pieces of code for easy tounderstand), code reusability(without writing same code for more than one time), and code maintainability(easy to modify by others in future)
--> TDD(Test), KWDD(Keyword) and BDD(Behaviour Driven Development) are available frameworks.
--> Hybrid(TDD+BDD) framework is more useful now for web site testing using SWD-Java.


PW-Java
main()
class
Linear framework using main() method, is not acceptable for Management
@Test(priority=1)
PW-Java
@Test(priority=2)
@Test(priority=3)
class
TDD(Test driven development) Framework using TestNG
Feature derived from Manual test case
english sentences
PW-Java
PW-Java
PW-Java
class
BDD(Behaviour driven development) framework using Cucumber



Test automation in a Framework

j. "Page" interface methods:

--> "Page" is an interface in Playwright jar files. It is one of the core classes used for interaction in test automation.
-> "PageImpl" is a concrete class for that "Page" interface in Playwright jar files.

--> Below code can not work to create an object to "Page" inteface because PageImpl() constructor in not public:

```
Page p=new PageImpl();
```

--> Follow below code to create an object to "Page" inteface:

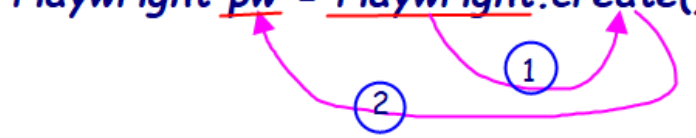
```
Playwright pw = Playwright.create();
BrowserType.LaunchOptions lo = new BrowserType.LaunchOptions();
lo.setHeadless(false); // Run with UI
Browser br = pw.chromium().launch(lo);
Page p = br.newPage();
```

--> Here, "Page" interface's object "p" represents a single tab or window in a browser.
--> Here's a categorized list of commonly used "Page" interface methods in Playwright (Java version).
These methods allow us to interact with and test our application effectively:

j.1. 📄 Navigation

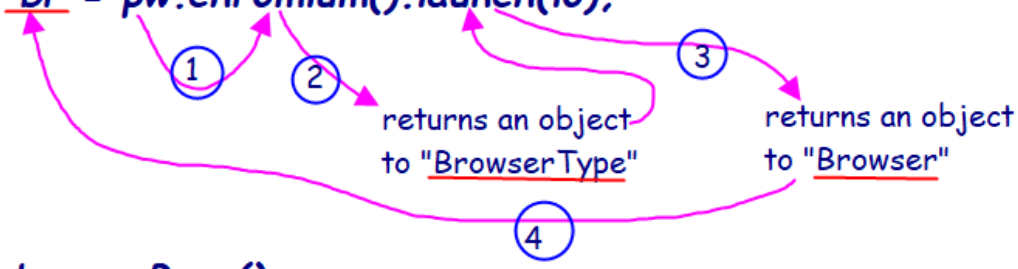
Method	Description
`navigate(String url)`	Navigates to the specified URL.
`goBack()`	Navigates to the previous page.
`goForward()`	Navigates to the next page.
`reload()`	Reloads the current page.

```
Playwright pw = Playwright.create();
```

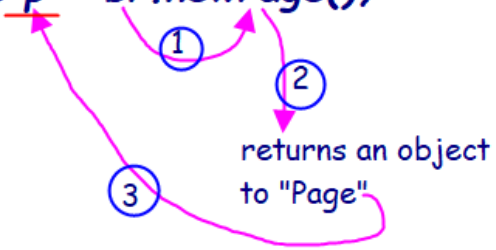



```
BrowserType.LaunchOptions lo = new BrowserType.LaunchOptions();
lo.setHeadless(false); // Run with UI
```

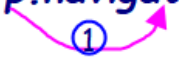

```
Browser br = pw.chromium().launch(lo);
```




```
Page p = br.newPage();
```




```
p.navigate("https://www.google.com");
```



j.2. 🖱 Actions (Click, Fill, etc.)

Method	Description
`click(String selector)`	Clicks the element matching the selector.
`dblclick(String selector)`	Double-clicks the element.
`fill(String selector, String value)`	Fills the input element with the given value.
`type(String selector, String text)`	Types text into the input field.
`press(String selector, String key)`	Simulates pressing a key (e.g., Enter).

Case study-11 : about Selectors:

--> In Playwright, selectors are used to identify elements on a web page so that actions (like click(), fill(), type(), etc.) can be performed on them.

--> Playwright supports multiple types of selectors, each with unique syntax and advantages.

--> Here's the refined list of all Playwright Java selectors along with their correct Java syntax using the Playwright Java API (Locator, FrameLocator, etc.).

#	Selector Type	Java Syntax
1	ID Selector	`page.locator("#my-button");`
2	Class Name Selector	`page.locator(".submit-button");`
3	Text Selector	`page.locator("text=Submit");`

part of CSS selector

Locate an element using text directly is a Playwright's inbuilt feature

4 (default)	CSS Selector	`page.locator("css=button#id");`
5	XPath Selector	`page.locator("xpath=//button[text()='Submit']");`
6	Data-test-id Selector	`page.locator("[data-testid='submit-button']");`
7	Role Selector	`page.getByRole(AriaRole.BUTTON, new Page.GetByRoleOptions().setName("Submit"));`
8	Combining Selectors	`page.locator("div#container >> text=Login");`
9	Chaining Selectors	`page.locator("form").locator("input[type='text']");`
10	Frame Locators	`page.frameLocator("#my-frame").locator("button");`
11	Attribute Selector	`page.locator("[placeholder='Email']");`
12	nth-match Selector	`page.locator("ul > li").nth(2); // 3rd li`
13	SVG Selectors	`page.locator("svg text");`
14	Shadow DOM Selectors	`page.locator("custom-element >> shadow=button");`

all CSS selector expressions(around 15)
("css=" is optional because of default)

all xpath expressions
("xpath=" is mandatory)

Locate an element using "data-testid" directly is a Playwright's inbuilt feature

Locate an element using getByRole() directly is a Playwright's inbuilt feature

* Locate an element by combining different locator types

Hierarchical

use frameLocator() to work with frames

Locate an element by giving attribute with value only (no * or tagname)

use nth() to choose one element when multiple elements were matched

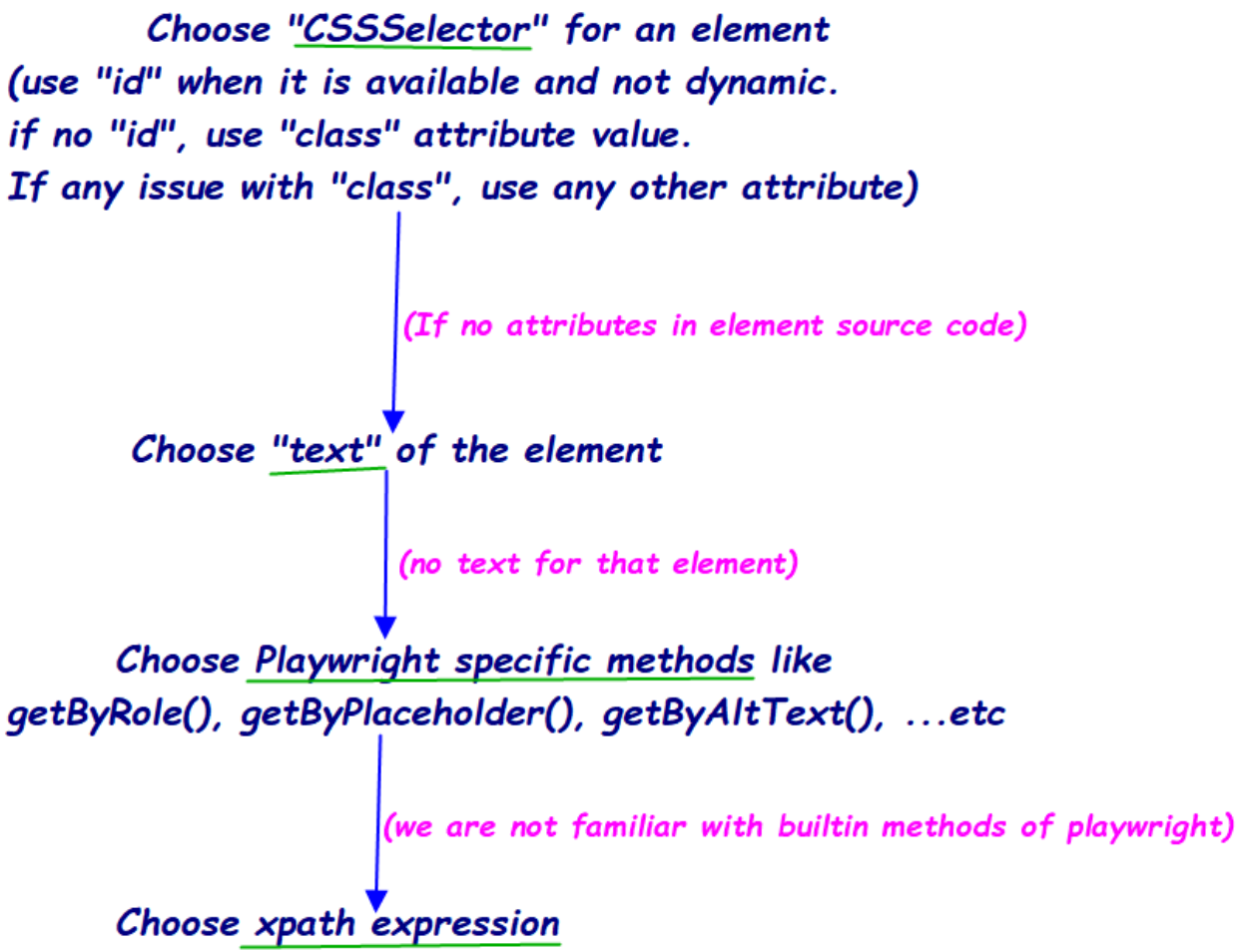
part of CSS selector

Using Combining Selectors

✅ Playwright Locator Types (Selector Engines)

🔢	Selector Type	Syntax Example	Description
1	CSS Selector	`"button.submit"`	Default selector type in `page.locator(...)`
2	XPath Selector	`"xpath=//div[@class='item']"`	Must start with `xpath=`
3	Text Selector	`"text=Login"`	Finds element by visible text
4	Role Selector	`getByRole("button", new Page.GetByRoleOptions().setName("Submit"))`	Based on ARIA roles
5	Label Selector	`getByLabel("Email")`	Matches form labels like `<label for="email">`
6	Placeholder Selector	`getByPlaceholder("Search here")`	Matches input placeholder text
7	Alt Text Selector	`getByAltText("Company Logo")`	For `<img alt="...">`
8	Title Selector	`getByTitle("Close popup")`	For `title="..."` attributes
9	Test ID Selector	`getByTestId("submit-button")`	For attributes like `data-testid="..."`
10	Custom Selectors	`:has(...)`, `:nth-match(...)`, `>>`	Playwright-specific enhancements
11	Frame Locators	`page.frameLocator("#my-frame").locator("button");`	To work with elements under frames

--> In general, we have to follow below heirarchy of selectors or locators of elements to run our playwright test scripts in less time:



Case study-12: "CSSSelector" Expressions:

--> "CSS" stands for Cascading Style sheets

--> "cssSelectors" is an expressions language or a parsing language to parse specific element source in given page source(DOM).

--> In Playwright Java code, we use "cssSelectors" expressions as selectors or locators for finding elements.

--> "cssSelectors" are syntax based expressions like described below:

Syntax-1:

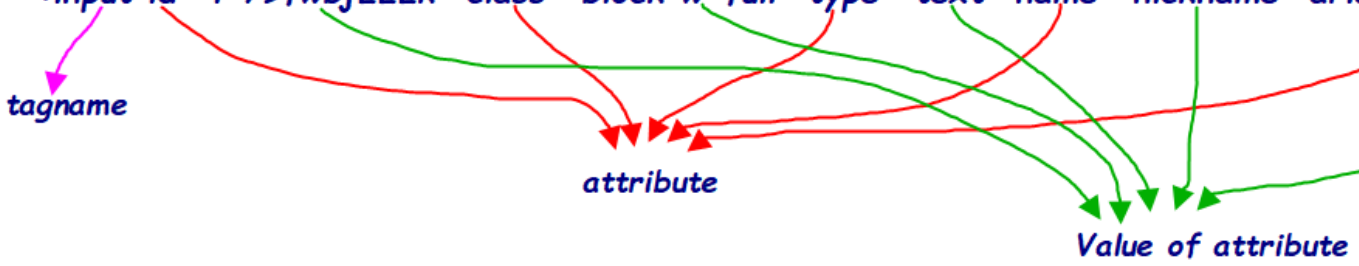
```
tagname[attribute=value]
tagname[attribute='value']
```

-> Quotes are optional in all CSSSelctor expressions.

-> Use "*"(refers any tag in HTML) in the place of tagname.

ex:

```
<input id="i-79fwbj222k" class="block w-full" type="text" name="nickname" aria-required="false">
```




```
input[name=nickname] -> "name" attribute value is "nickname" (exact match)
input[type=text]
input[id='i-79fwbj222k']
--------------------
```

Syntax-2:

```
tagname[attribute*='part of value']
```

```
input[name*='nick']  -> "name" attribute value contains "nick" (partial match)
input[name*='ckna']
input[name*='me']
```

Syntax-3: `tagname[attribute^='starting part of value']`

input[name^='nick'] -> "name" attribute value starts with "nick" (partial match)

Syntax-4: `tagname[attribute$='ending part of value']`

input[name$='me'] -> "name" attribute value ends with "me" (partial match)

Syntax-5:

```
tagname[id='value of id']
tagname#idvalue
#idvalue
```

input[id='i-79fwbj222k'] or input#i-79fwbj222k or #i-79fwbj222k

Syntax-6:

```
tagname[class='value of class']
tagname.classvalue
.classvalue
```

Ex-1 :

`<input class="button" ...../>`

input[class=button]
input.button
.button

Ex: - `<input class="abc xyz pqr" ...../>`

input[class='abc xyz pqr']
input.abc.xyz.pqr
.abc.xyz.pqr
--> when "class" attribute value is multi word, we have to use "." in the place of every space.

Syntax-7:

```
tagname[attribute='value'][attribute = 'value'] ….
tagname#idvalue[attribute='value'] ….
tagname.classvalue[attribute='value'] ….
```

Ex: - `<input type="text" class="inputtext _55r1 _6luy" name="email" id="email ">`

input[id=email][name=email]
input#email[name=email]
#email[name=email]

input[class='inputtext _55r1 _6luy'][name=email]
input.inputtext._55r1._6luy[name=email]
.inputtext._55r1._6luy[name=email]

Syntax-8: `tagname[attribute='value'],tagname[attribute='value']` → comma indicates "OR"

Ex: A toggle button with "OK" or "Cancel"
button[name=OK],button[name=Cancel]
[name=OK],[name=Cancel]

Syntax-9: `tagname[attribute='value']>tagname`

- `tagname[attribute='value']` → Parent element locator
- `tagname` → child element's tagname

Syntax-10: `tagname[attribute='value'] tagname`

Space means parent to descendant(child, grandchild, ...etc)

Syntax-11: using "first-of-type". It will select the first tag element. For example:

`ul>li:first-of-type`

selects first matched "li" element under "ul"

`ul>*:first-of-type`

selects first child element in any type under "ul"

```
<ul>
    <input ----------/>
    <li ---------------/>
</ul>
```

Syntax-12: using "last-of-type". It will select the last tag element. For example:

`ul>li:last-of-type`

selects last matched "li" under "ul"

`ul>*:last-of-type`

selects last element in any type under "ul"

Syntax-13: using tag: nth-of-type(n). It will select the nth tag element of the list. For example:

`ul>li:nth-of-type(3)`

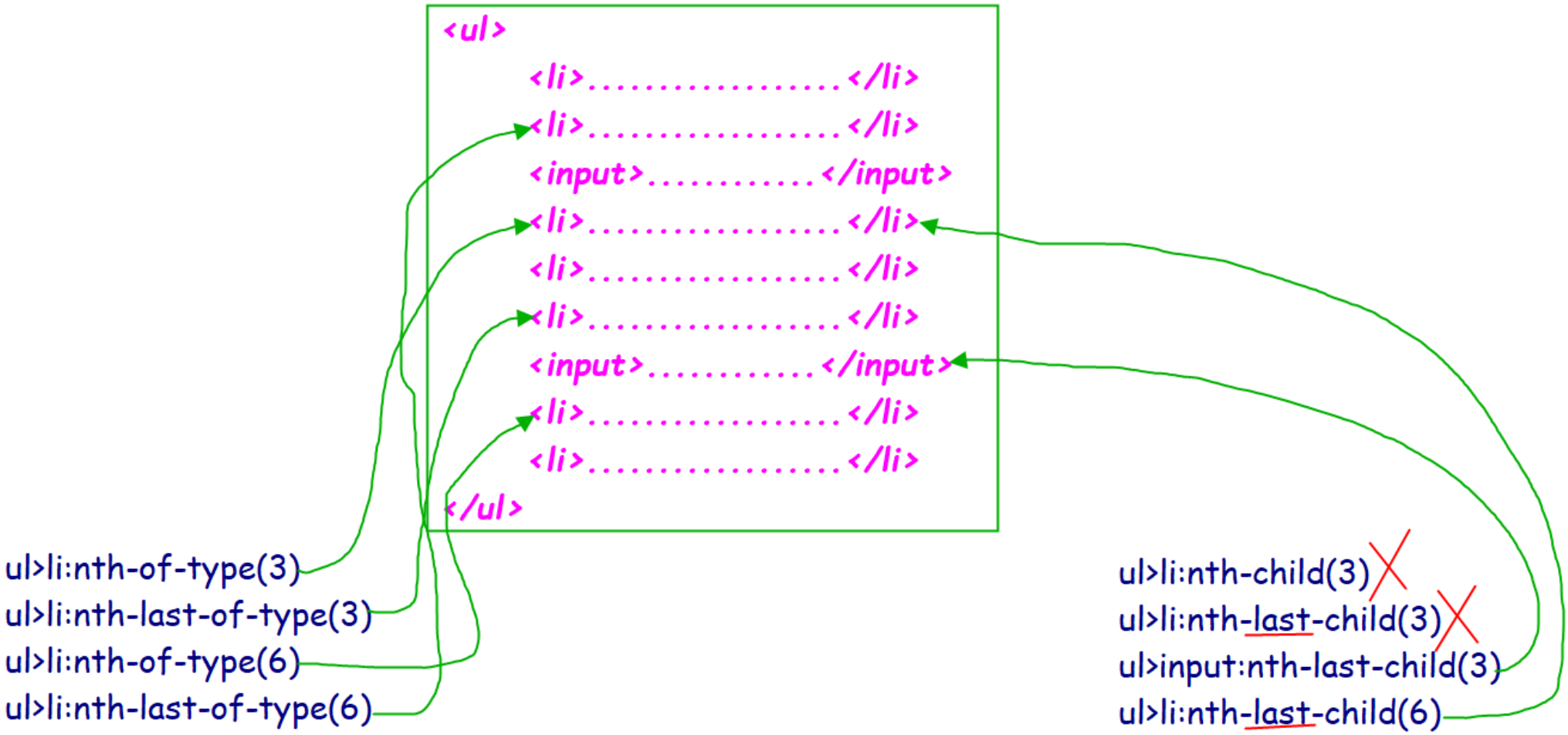
selects 3rd matched "li" under "ul"
(Here, index starts with "1")

`ul>*:nth-of-type(3)`

selects 3rd element in any tag under "ul"
(Here, index starts with "1")

✅ Related Variants

Selector	Description
`:nth-child(n)`	Selects the n-th child from the **start**
`:nth-last-child(n)`	Selects the n-th child from the **end**
`:nth-of-type(n)`	Selects the n-th element **of the same tag** from the start
`:nth-last-of-type(n)`	Selects the n-th element **of the same tag** from the end



--> In "-type", see the type first and then count of that type only.

--> In "-child", Do count first and then see the type.

Syntax-14: We can use "+" operator to locate following sibling of an element. For Example:

```
<ul id="Cars">
    <li id="mercedes">Mercedes made in Germany!</li>
    <li>BMW</li>
    <li>Porsche</li>
</ul>
```

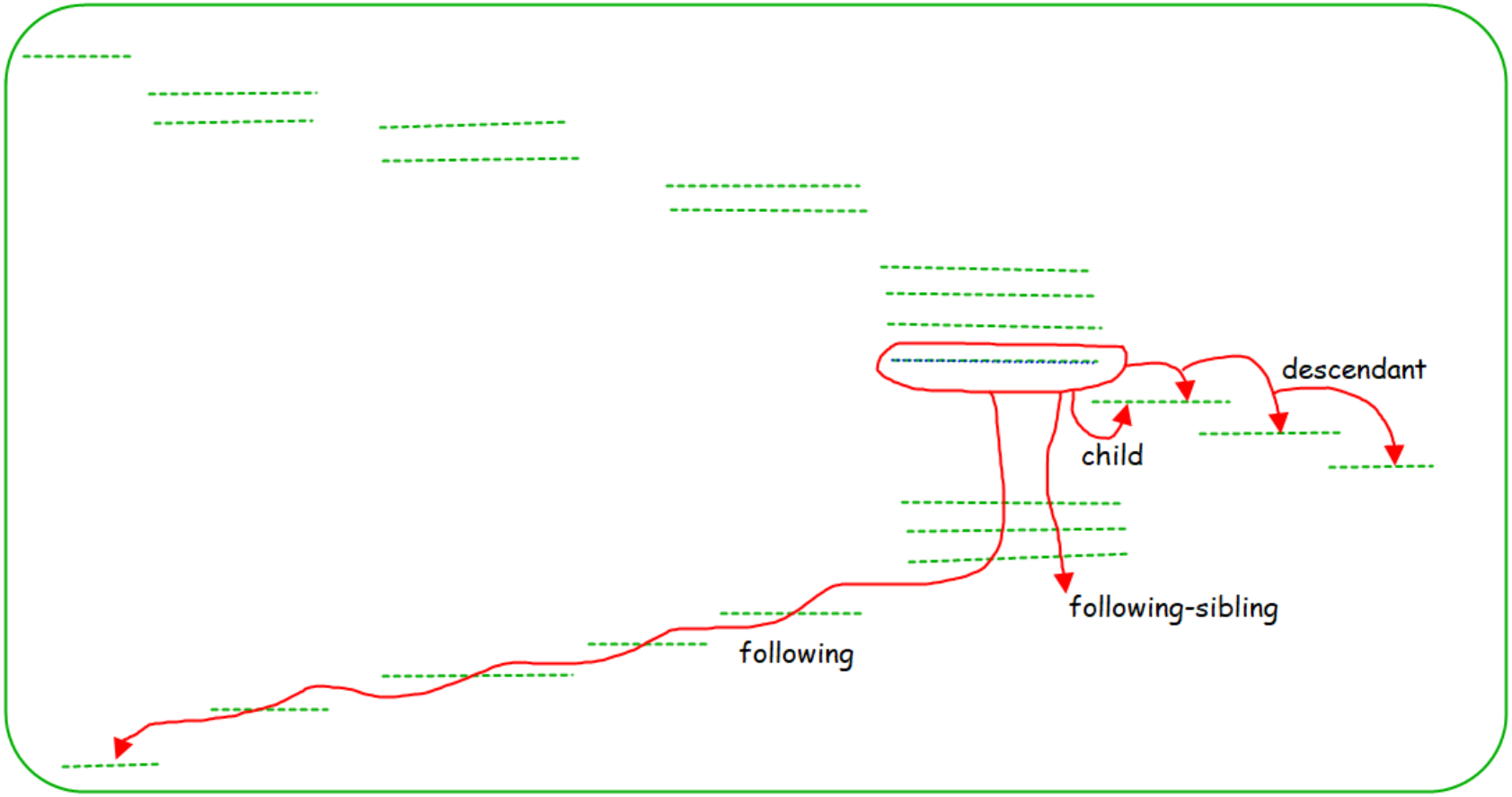
```
li#mercedes+li
```

first go to "li" element with id 'mercedes' and then select its adjacent "li" which is 'BMW' list item.
Note: If we want all following sibling, use "~" in the place of "+"

Syntax-15:- We can find an element by negating given details

`li:not(#mercedes)`	locate "li" but that "li" element's id is not "mercedes".
`input:not([name='kalam'])`	locate "input" tag but that element's name is not "kalam"
`ul>li:not(:first-child)`	locate All "li" child elements except first any type child
`ul>li:not(:first-of-type)`	locate All "li" child elements except first "li" type child

Page elements DOM(Document Object Model)



In CSSSelector,

parent to child -> ">"

parent to descendant -> space

following-sibling -> +

```
tagname [attribute = 'value']

tagname [attribute * = 'part of the value']

tagname [attribute ^ = 'starting part of value']

tagname [attribute $ = 'ending part of value']

tagname # idvalue

tagname.classvalue

tagname.classvalue[attribute = 'value']

tagname [attribute ='value'],tagname[attribute='value']

tagname [attribute = 'value']>tagname

tagname [attribute = 'value']  tagname

tagname>tagname:first-of-type

tagname>tagname:last-of-type

tagname>tagname:nth-of-type(index)

tagname[attribute='value']+tagname

tagname:not([attribute='value'])
```

```
*      -> any tag
=      -> Exact Match
*=     -> Contains(Partial Match)
^=     -> Starts with
$=     -> Ends with
|=     -> Exact match or starts with
#      -> id attribute of an element
.      -> class attribute of an element
,      -> OR
>      -> Parent to child
space-> Parent to Descendant
+      -> Adjacent sibling
~      -> All Following siblings
not    -> Negation
:      -> for first-of-type, last-of-type and
          nth-of-type
[][]   -> AND
```

Syntax-16:-

tagname:only-child

-> Only child of parent

ex:

h1>span:only-child -> "span" is only child to "h1", here no other child to "h1".

Syntax-17:-

tagname:only-of-type

-> Only element of its type

ex:

h1>span:only-of-type -> "span" is only child to "h1" in that "span" type, may be "h1" has other type childs.

Syntax-18:-

tagname:empty

-> No children (text or tags)

ex:

span:empty -> "span" tagged element without any child and text also

Syntax-19:-

input:enabled

-> Enabled form fields

Syntax-20:-

input:disabled

-> Disabled form fields

Syntax-21:-

```
input[type=checkbox]:checked
```

-> Checked checkboxes

Syntax-22:-

```
input[type=radio]:checked
```

-> Checked radio button

Syntax-23:-

```
input:focus
```

-> Element in focus

Syntax-24:-

```
a:hover
```

-> Element on hover

Syntax-25:-

```
input[required]
```

-> Has required attribute

Syntax-26:-

```
span::after
```

-> After pseudo-element

Syntax-27:-

```
input::placeholder
```

-> Placeholder text styling