Lab 5: Build a reusable Shared Library for Maven build/test stages and use it in a pipeline

## 🎯 Objective

Move from **hardcoded Jenkinsfiles** (Labs 3 & 4) to **Reusable Shared Libraries**.
This enables **standardization, reusability, and enterprise-scale pipeline governance**.

---

### Step 1 — Why Shared Libraries?

- DRY principle → no duplicate `build/test/package` code across projects.
- Centralized updates → fix once, all pipelines benefit.
- Enforce standards → coding guidelines, security checks, approvals.
- Extensible → teams can add project-specific steps but reuse the core.

💡 *Industry Example*: At Netflix, all pipelines use common library functions for build/test/deploy, ensuring consistent quality gates.

---

### Step 2 — Library Repository Setup

Create a new repo: `jenkins-shared-libs`

GITHUB refrrence: GitHub - cloud-dev-user/jenkins-shared-libs

```
jenkins-shared-libs/
 ├── vars/
 │    ├── mavenBuild.groovy
 │    ├── mavenTest.groovy
 │    ├── mavenPackage.groovy
 │    ├── parallelMavenTests.groovy
 │    └── notifySlack.groovy   (optional, for alerts)
 └── src/                      (for advanced helper classes)

```

---

### Step 3 — Core Library Functions

🔹 **vars/mavenBuild.groovy**

```
def call(String goals = 'clean install -DskipTests') {
    stage('Maven Build') {
        echo "Executing: mvn ${goals}"
        sh "mvn ${goals}"
    }
}

```

🔹 **vars/mavenTest.groovy**

```
def call() {
    stage('Maven Test') {
        sh 'mvn test'
        junit '**/target/surefire-reports/*.xml'
    }
}

```

🔹 **vars/mavenPackage.groovy**

```
def call() {
    stage('Maven Package') {
        sh 'mvn package -DskipTests'
        archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
    }
}

```

🔹 **vars/parallelMavenTests.groovy**

```
def call(Map modules = [:]) {
    stage('Parallel Maven Tests') {
        parallel modules.collectEntries { module, path ->
            ["${module}" : {
                dir(path) {
                    sh 'mvn test'
                    junit 'target/surefire-reports/*.xml'
                }
            }]
        }
    }
}

```

💡 This function lets teams run tests for `core`, `tax`, `app` in **parallel** just by calling:

```
parallelMavenTests([
    "Core" : "core",
    "Tax"  : "tax",
    "App"  : "app"
])

```

🔹 **vars/notifySlack.groovy**

```
def call(String message, String channel = '#devops') {
    stage('Notify Slack') {
        echo "Slack → ${channel}: ${message}"
        // Uncomment when Slack plugin configured:
        // slackSend channel: channel, message: message
    }
}

```

---

### Step 4 — Configure Jenkins

1. **Manage Jenkins → Configure System → Global Pipeline Libraries**
   - Name: `my-shared-lib`
   - Default Version: `main` (or a tag like `v1.0`)
   - SCM: Git (GitHub/GitLab/Bitbucket).
2. Pipelines can now load it with:

```
@Library('my-shared-lib') _

```

---

### Step 5 — Using the Library in a Jenkinsfile

**Jenkinsfile**

```
@Library('my-shared-lib') _

node {
    stage('Checkout') {
        checkout scm
    }

    // Reusable stages
    mavenBuild()
    parallelMavenTests([
        "Core" : "core",
        "Tax"  : "tax",
        "App"  : "app"
    ])
    mavenPackage()

    notifySlack("Build Completed ✅ for ${env.JOB_NAME} #${env.BUILD_NUMBER}")
}

```

---

### Step 6 — Validation

- Commit & push `jenkins-shared-libs`.
- Run pipeline → confirm **stages come from library** (look for `Loading library my-shared-lib` in logs).
- Verify parallel tests execute correctly.
- Confirm Slack/notification messages (if configured).

---

### Optional Challenges (Stretch Goals)

1. **SonarQube Quality Gate**
   - Create `vars/sonarScan.groovy` to integrate static code analysis.
2. **Code Coverage Enforcement**
   - Add `jacocoReport.groovy` that fails build if coverage < 80%.
3. **Dynamic Notifications**
   - Enhance `notifySlack` to detect failure/success and send different messages.
4. **Versioned Libraries**
   - Use version pinning:

     ```
     @Library('my-shared-lib@v2.0') _

     ```

5. **Multi-Language Support**
   - Add `npmBuild.groovy` for Node.js projects, showing how one library can serve **polyglot pipelines**.

---

✅ By completing this lab, your team will:

- Master **Shared Libraries** (core Jenkins skill at enterprise scale).
- Learn **parallel test orchestration** with reusable code.
- Build **extensible, versioned, and standardized pipelines**.