**Objective**: Set up a CI pipeline in Jenkins that automates the build and test process for a sample Java project hosted on GitHub.

**Prerequisites**:

1. Jenkins installed and running.
2. A GitHub account.
3. Java and Maven installed on your Jenkins server.

**Steps**:

1. **Create a Sample Java Project**:
   * Create a simple Java project with a few classes and tests. You can initialize a new Maven project using the following command:

arduino

* + mvn archetype:generate -DgroupId=com.example -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
  + Add some code and unit tests to the project. Ensure that the tests are written using a testing framework like JUnit.

1. **Push the Project to GitHub**:
   * Create a new repository on GitHub to host your sample Java project.
   * Push the project code to the GitHub repository.
2. **Set Up Jenkins Job**:
   * Log in to your Jenkins dashboard.
   * Click on "New Item" to create a new Jenkins job.
   * Enter a name for the job (e.g., "My Java CI Pipeline") and select "Freestyle project".
   * Click "OK" to create the job.
3. **Configure Source Code Management**:
   * In the job configuration page, scroll down to the "Source Code Management" section.
   * Choose "Git" and enter the URL of your GitHub repository.
   * Optionally, configure credentials if your repository requires authentication.
4. **Set Up Build Triggers**:
   * In the job configuration page, scroll down to the "Build Triggers" section.
   * Select "Poll SCM" and specify a schedule (e.g., \* \* \* \* \*) to poll for changes in your GitHub repository.
5. **Configure Build Steps**:
   * In the job configuration page, scroll down to the "Build" section.
   * Click on "Add build step" and choose "Invoke top-level Maven targets".
   * Enter the Maven goals to execute (e.g., clean test) and specify the path to the pom.xml file.
6. **Save Job Configuration**:
   * Save the job configuration by clicking "Apply" or "Save".
7. **Run the Pipeline**:
   * Trigger the job manually or wait for Jenkins to poll for changes in your GitHub repository.
   * Jenkins will clone the repository, build the project using Maven, and execute the tests.
8. **View Pipeline Results**:
   * Monitor the job's progress from the Jenkins dashboard.
   * If the build fails or any tests fail, Jenkins will provide detailed logs to help diagnose the issue.
9. **Optional: Add Post-Build Actions**:
   * Configure post-build actions to archive artifacts, generate reports, or trigger downstream jobs based on the build result.
10. **Iterate and Improve**:
    * Iterate on your Jenkins job configuration to add additional build steps, integrate with more tools (e.g., SonarQube for code quality analysis), and improve automation and error handling.

By completing this exercise, you'll have set up a basic CI pipeline in Jenkins that automates the build and test process for your sample Java project hosted on GitHub. You can further expand and customize this pipeline to suit your specific project requirements.