TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN - BỘ MÔN CÔNG NGHỆ PHẦN MỀM

Topic 5:

Build Management—Maven

Giảng viên: Phạm Thị Thương – Bộ môn CNPM – Khoa CNTT
Email: ptthuong@ictu.edu.vn

Main Content

1. **Maven – Main Features**
2. Maven – POM
3. Maven – Build Life Cycle
4. Maven – Repositories
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. Maven & Jenkins – Examples for testing & static code analysis

1. Maven – Các đặc trưng chính

- Apache Maven
  - is an innovative software project management tool. It uses a project object model (POM) file to manage project's build, dependencies, reporting and documentation.
  - Install Maven on Windows, Ubuntu and MacOS
    - Kiểm tra kết quả cài đặt: gõ lệnh

      mvn -version

1. Maven – Các đặc trưng chính

a. Thiết lập dự án đơn giản, tuân theo các cách tiếp cận thực tế tốt nhất

- Maven tránh tối đa việc phải cấu hình nhiều bằng cách cung cấp các project templates (named *archetypes*)

b. Quản lý phụ thuộc

- Các phụ thuộc được update, downloading và validating tự động.

1. Maven – Các đặc trưng chính

c. Cô lập các phụ thuộc dự án & các plugin

- Các phụ thuộc dự án được trích rút từ *dependency repositories*, các plugins lại được trích rút từ *plugin repositories*

  => *tránh* xung đột khi plugin khởi tạo & download các phụ thuộc thêm vào.

d. Build dự án dựa trên mô hình:

- Cho phép build nhiều dự án với các định dạng như *.jar, .war, .ear, ejb, .pom, .maven-plugin, .metadata, …*

1. Maven – Các đặc trưng chính

e. Sinh site cho dự án:

- Cho phép sinh ra các tài liệu dạng website&PDF cho dự án

f. Quản lý phát hành:

- Tự động tích hợp với source control system và quản lý release của dự án mà không cần cấu hình thêm.

g. Hệ thống kho thành phẩm phong phú:

- Các phụ thuộc dự án được download, đẩy lên các kho Maven

Main Content

1. Maven – Các đặc trưng chính
2. **Maven – POM**
3. Maven – Build Life Cycle
4. Maven – Repositories
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. Maven & Jenkins – Examples for testing & static code analysis

2. Maven – POM

- POM (Project Object Model):
  - ~ file XML nằm tại thư mục base/root của dự án (pom.xml).
    - Chứa thông tin về dự án + các goals + plugins + các phụ thuộc + các chi tiết cấu hình, … được Maven sử dụng để builds project(s).

2. Maven – POM

- 3 trường bắt buộc phải có trong POM khi khai báo thông tin của dự án:
  - Project group (groupId),
  - Project name (artifactId)
  - Project version.
  - ⇒ xác định tính duy nhất của dự án: groupId:artifactId:version
    - Ví dụ: com.company.bank:comsumer-banking:1.0

2. Maven – POM

- POM.xml không y.cầu phải viết thủ công.
  - Maven cung cấp một số archetype plugins để tạo projects => *tương ứng project structure & pom.xml file được sinh ra.*
  - Danh sách các archetype?
    - Xem hình (dưới)

New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs | Configure...

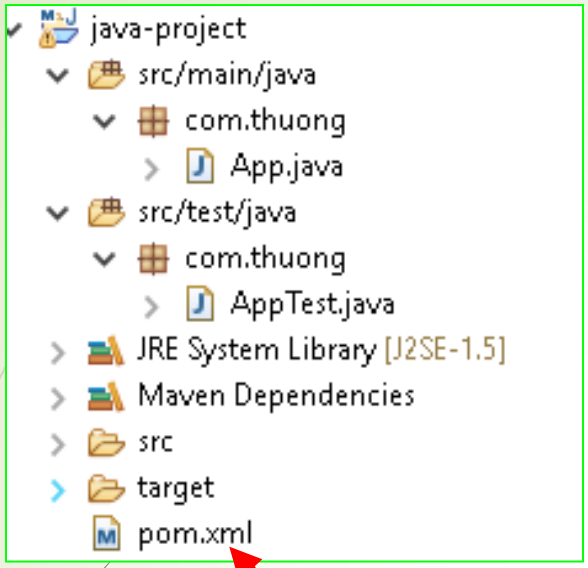
Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-archetype	1.0
org.apache.maven.archetypes	maven-archetype-j2ee-simple	1.0
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven project.

2. Maven – POM

- Ví dụ:
  - Tạo dự án **"java-project"** sử dụng archetype "**maven-archetype-quickstart"** tại path: **"E:\Namhoc_2020\HockyI_2020_2021\2_Baotri_PM__4.Demo_Projects\5.TestMaven":**
    - Mở DOS command line
      - cd đến path trên,
      - Type: mvn archetype:generate -DgroupId=com.thuong -DartifactId=java-project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
    - Kết quả: hình (dưới)




```
mvn archetype:generate -DgroupId=com.thuong -DartifactId=java-project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

E:\Namhoc_2020\HockyI_2020_2021\2_Baotri_PM__4.Demo_Projects\5.TestMaven\Testmore\java-pr...

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/200
2    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/mav
3    <modelVersion>4.0.0</modelVersion>
4    <groupId>com.thuong</groupId>
5    <artifactId>java-project</artifactId>
6    <packaging>jar</packaging>
7    <version>1.0-SNAPSHOT</version>
8    <name>java-project</name>
9    <url>http://maven.apache.org</url>
10   <dependencies>
11     <dependency>
12       <groupId>junit</groupId>
13       <artifactId>junit</artifactId>
14       <version>3.8.1</version>
15       <scope>test</scope>
16     </dependency>
17   </dependencies>
18 </project>
19
```

2. Maven – POM

- **Maven Archetype Plugin**
  - Allows the user to create a Maven project from an existing template (~ archetype)
    - It also allows the user to create an archetype from an existing project.
      - Ex., See figure (below)
    - Note:
      - Archetype Plugin is embedded in IDEs (Eclipse, NetBeans, IDEA).

Main Content

1. Maven – Các đặc trưng chính
2. Maven – POM
3. **Maven – Build Life Cycle**
4. Maven – Repositories
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. Maven & Jenkins – Examples for testing & static code analysis

3. Maven – Build Life Cycle

- Maven có 3 standard build lifecycles:
  - Clean: pre-clean, **clean**, post-clean
  - Default (or Build): **validate**, initialize, generate-sources, process-sources, generate-resources, process-resources, **compile**, process-classes, generate-test-resources, process-test-resources, test-compile, process-test-classes, **test**, prepare-package, **package**, pre-integration-test, **integration-test**, post-integration-test, **verify**, **install**, **deploy**
  - site: pre-site, **site**, post-site, site-deploy

3. Maven – Build Life Cycle

- Ví dụ: Type: **mvn install**
  - Kết quả:

```
E:\Namhoc_2020\HockyI_2020_2021\2_Baotri_PM\__4.Demo_Projects\5.TestMaven\Testmore\java-project>mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.thuong:java-project >----------------------
[INFO] Building java-project 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ java-project ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory E:\Namhoc_2020\HockyI_2020_2021\2_Baotri_PM\__4.Demo_Projects\5.TestMaven\Tes
tmore\java-project\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ java-project ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ java-project ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory E:\Namhoc_2020\HockyI_2020_2021\2_Baotri_PM\__4.Demo_Projects\5.TestMaven\Tes
tmore\java-project\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ java-project ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ java-project ---
[INFO] Surefire report directory: E:\Namhoc_2020\HockyI_2020_2021\2_Baotri_PM\__4.Demo_Projects\5.TestMaven\Testmore\jav
a-project\target\surefire-reports

-------------------------------------------------------
 T E S T S
```

3. Maven – Build Life Cycle

- Lưu ý:
  - Khi một giai đoạn được gọi = lệnh Maven,
    - => chỉ các giai đoạn trước đó đến giai đoạn đó được thực thi.
    - Ví dụ: **mvn clean**

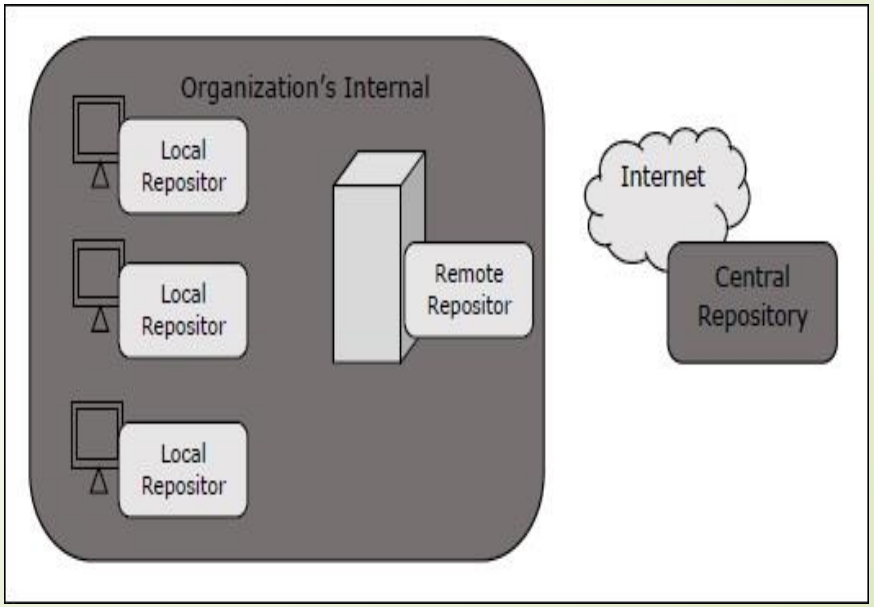
      => pre-clean, **clean** được thực thi, **post-clean** không được thực thi.

Main Content

1. Maven – Các đặc trưng chính
2. Maven – POM
3. Maven – Build Life Cycle
4. **Maven – Repositories**
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. Maven & Jenkins – Examples for testing & static code analysis

4. Maven – Repositories

- Repository (maven):
  - Ba kiểu Maven repository:
    a. **Local,**
    b. **Central,**
    c. **Remote.**



a. Local Repository

- Được tạo khi chạy lệnh maven lần đầu tiên.
  - Kho đặt trên máy tính người dùng:
    - Mặc định, kho được đặt tại: **${user.home}/.m2/repository**
    - Ví dụ:
      - Unix/Mac OS X: **~/.m2/repository**
      - Windows: **C:\Users\{your-username}\.m2\repository**

a. Local Repository

- Tác dụng:
  - Lưu giữ mọi dependencies của dự án (library jars, plugin jars ...).
    - When we compile a Maven project, Maven will download all the project's dependency and plugin jars into the Maven local repository,

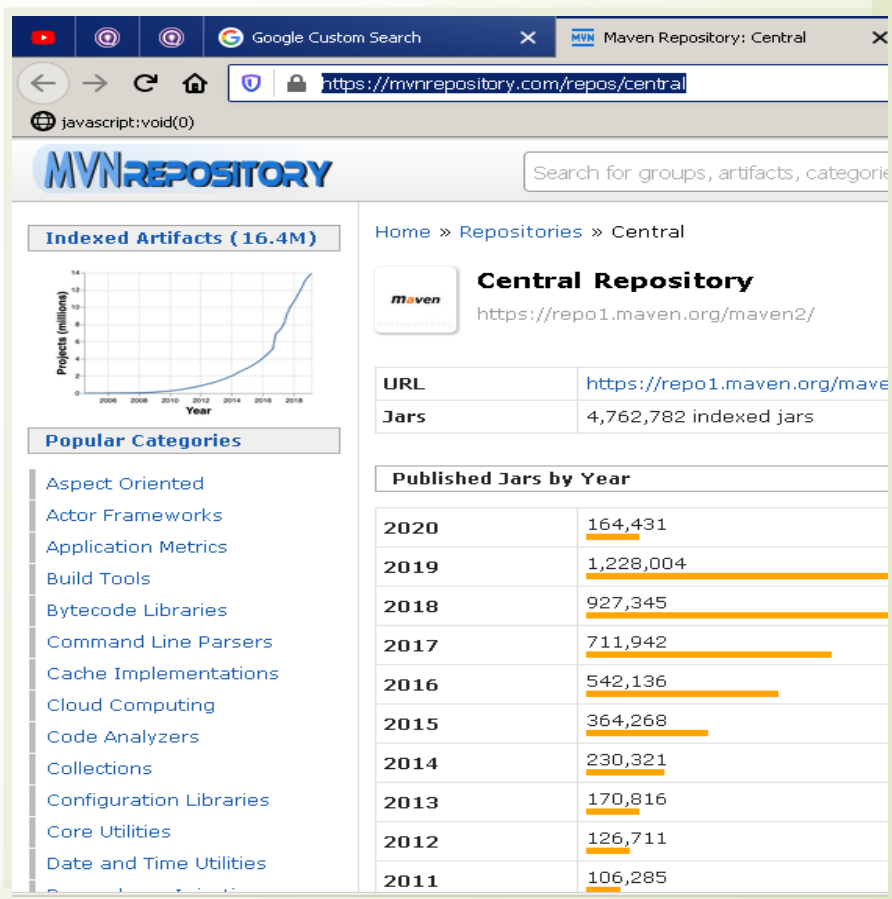
    => Save time for next compilation.

b. Central Repository

- Central Repository: ↑ bởi Maven community.
  - Chứa một số lớn các thư viện dùng chung (commonly used libraries).
    - ⇒ Khi Maven không tìm được dependency trong local repository, nó sẽ bắt đầu tìm kiếm trong central repository:
      - Maven Central Repository URL: https://repo.maven.apache.org/maven2
      - Maven Central Repository Search: https://search.maven.org/

b. Central Repository

- Địa chỉ kho:
  - https://mvnrepository.com/repos/central
  - See Fig:



c. Remote Repository

- **<u>Lưu ý:</u>**
  - Khi Maven không tìm thấy dependency trong central repository.
    - ⇒ Nó sẽ dừng tiến trình build process và ném ra error message đến console.
    - Để ngăn chặn tình huống này, Maven cung cấp **Remote Repository**
      - ⇒ Đây là kho tùy biến riêng của các nhóm developers.

c. Remote Repository

- Ví dụ:
  - The `org.jvnet.localizer` chỉ có tại kho Java.net repository (remote repository):
    - Trong POM.xml file ta khai báo:

```
<dependency>
    <groupId>org.jvnet.localizer</groupId>
    <artifactId>localizer</artifactId>
    <version>1.8</version>
</dependency>
```

⇒ Khi build dự án Maven này, nó sẽ trả về lỗi không tìm thấy phụ thuộc này!

c. Remote Repository

- Ví dụ:
  - Khắc phục lỗi: Declare remote repository (java.net) in the POM.xml file:

```
<repositories>
<repository>
    <id>java.net</id>

<url>https://maven.java.net/content/repositories/public
/</url>
</repository>
    </repositories>
```

The `org.jvnet.localizer` is now available in Maven center repository.

c. Remote Repository

- Cách tạo kho Maven remote?
  - Bài tập: sinh viên thực hiện:
    - Tạo 1 kho remote Maven cho team của mình trên Nexus
    - Đưa các artifact của team lên kho vừa tạo
    - Tham chiếu đến các artifacts này trong dự án Maven cụ thể.

4. Maven – Repositories

- Lưu ý:
  - Khi thực hiện lệnh build, Maven sẽ bắt đầu tìm kiếm các dependences theo trình tự sau:
    - Search local rep → Search Central Repository → Search Remote Repository → nếu không thấy, ném ra lỗi.

Main Content

1. Maven – Các đặc trưng chính
2. Maven – POM
3. Maven – Build Life Cycle
4. Maven – Repositories
5. **Maven – Plugins**
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. Maven & Jenkins – Examples for testing & static code analysis

5. Maven – Plugins

- Maven plug-ins thường dùng để:
  - create jar file,
  - create war file,
  - compile code files,
  - unit testing of code,
  - create project documentation,
  - create project reports.
  - …

5. Maven – Plugins

- Mỗi plugin cung cấp 1 tập các goals,
  - Mỗi goal được thực thi sử dụng cú pháp:

    mvn [plugin-name]:[goal-name]

    - Ví dụ:
      - Biên dịch dự án Java bằng cách chạy lệnh sau:

mvn compiler:compile

5. Maven – Plugins

- Maven cung cấp 2 kiểu Plugins chính:

Sr.No.	Type & Description
1	**Build plugins** Thực thi trong suốt tiến trình build, cần được cấu hình trong <build/> element của pom.xml.
2	**Reporting plugins** Sinh ra site, cần được cấu hình trong <reporting/> element của file pom.xml.

- Một số Maven plugins thông dụng:
  - Xem bảng (dưới)

Sr.No.	Build plugins & Description
1	**Clean[1]:** Cleans up target after the build. Deletes the target directory.
2	**Compiler[1]:** Compiles Java source files.
3	**Deploy**: Deploy the built artifact to the remote repository.
4	**Jar[1]:** Builds a JAR file from the current project.
5	**War[1]:** Builds a WAR file from the current project.
6	**Failsafe[1]**: Run the JUnit integration tests in an isolated classloader.
7	**Install[1]**: Install the built artifact into the local repository.
8	**Site[1]**: Generate a site for the current project.
9	**Surefire[1]**: Run the JUnit unit tests in an isolated classloader.
10	**Javadoc[2]:** Generates Javadoc for the project.
11	**surefire-report[2]**: Generate a report based on the results of unit tests.
12	**Changelog[2]**: Generate a list of recent changes from your SCM.
13	**project-info-reports[2]**: Generate standard project reports

Đều là plugin mã nguồn mở, cho phép tùy biên

See link for more plugins:https://maven.apache.org/plugins/index.html

5. Maven – Plugins

- Ví dụ: Javadoc plugin có 16 goals:

No.	Goal	Description
1	javadoc:javadoc	generates the Javadoc files for the project.
2	javadoc:test-javadoc	generates the test Javadoc files for the project
3	javadoc:jar	creates an archive file of the generated Javadocs
4	javadoc:test-jar	creates an archive file of the generated Test Javadocs.
…	…	…

Ví dụ: type command:

mvn Javadoc:Javadoc

=> kết quả sites được sinh ra trong đường dẫn:
E:\Namhoc_2020\HockyI_2020_2021\2_Baotri_PM__4.Demo_Projects\5.TestMaven\java-project\target\site\apidocs

Main Content

1. Maven – Các đặc trưng chính
2. Maven – POM
3. Maven – Build Life Cycle
4. Maven – Repositories
5. Maven – Plugins
6. **Maven – Build, Test and Run Maven project**
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. Maven & Jenkins – Examples for testing & static code analysis

6. Maven – Build, Test & Run Project

- Package:
  mvn clean package
- Run it as Jar:
  java -jar target/java-project-1.0-SNAPSHOT.jar

Note**: mvn package**

~ run all goals: validate → compile → test → package

Main Content

1. Maven – Các đặc trưng chính
2. Maven – POM
3. Maven – Build Life Cycle
4. Maven – Repositories
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. **Maven – Build Automation**
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. Maven & Jenkins – Examples for testing & static code analysis

7. Maven – Build Automation

- Build tự động: Bài tập
  - SV thực hiện xây dựng 1 kịch bản ở đó tiến trình build các dự án được bắt đầu khi các dự án chúng phụ thuộc được thay đổi và build thành công.
  - Ví dụ:
    - Team A đang phát triển dự án bus-core-api và 2 dự án khác là app-web-ui và app-desktop-ui là các dự án phụ thuộc (được phát triển bởi 2 team B và C)
      - 2 team B, C y.cầu tiến trình build của họ tự động khởi chạy bất cứ khi nào dự án bus-core-api thay đổi?

Main Content

1. Maven – Các đặc trưng chính
2. Maven – POM
3. Maven – Build Life Cycle
4. Maven – Repositories
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. **Maven – Manage Dependencies**
9. Maven – Tích hợp với Jenkins
10. Maven & Jenkins – Examples for testing & static code analysis

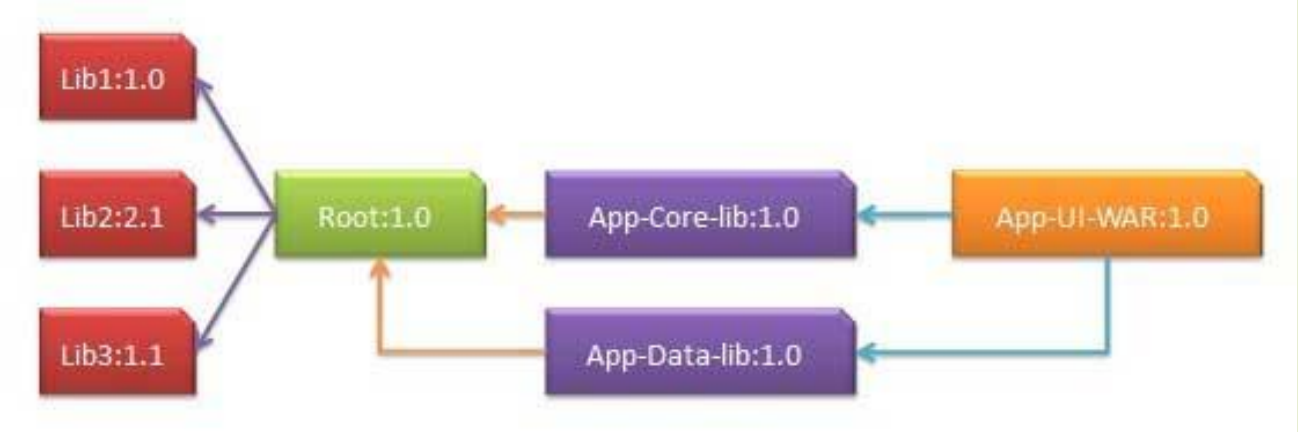
8. Maven – Manage Dependencies

- Quản lý phụ thuộc
  - Là 1 trong các đặc trưng cốt lõi của Maven
    - Quản lý phụ thuộc là một nhiệm vụ khó khăn khi ta làm việc với dự án có nhiều mô đun (hàng trăm module/sub-project)
      - Maven là một giải pháp tốt cho vấn đề này.

8. Maven – Manage Dependencies

- Nếu 1 tập dự án nằm dưới 1 dự án chung
  - Nên tạo một file POM chung chứa tất cả các phụ thuộc của chúng, file này là cha của các file POM thuộc các dự án con.
  - Ví dụ:
    - Xét đồ thị phụ thuộc dự án như sau
      - Xem (hình dưới)

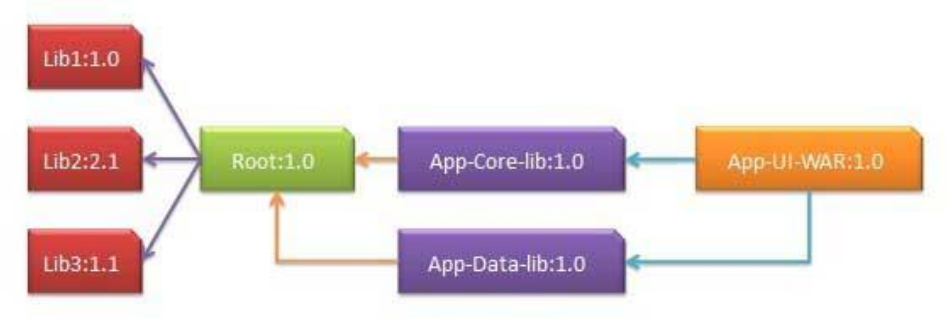
8. Maven – Manage Dependencies



- Trong đó:
  - App-UI-WAR phụ thuộc vào App-Core-lib và App-Data-lib
  - Root là cha của App-Core-lib và App-Data-lib
  - Root xác định lib1, lib2, lib3 là các phụ thuộc trong phần dependence của nó

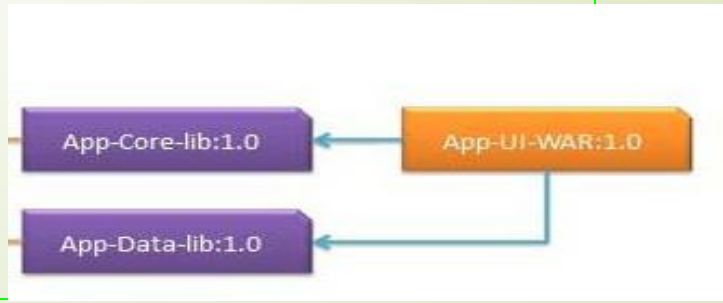
8. Maven – Manage Dependencies

- Khai báo các file POM tương ứng với các phụ thuộc?
  - App-UI-WAR
  - App-Core-lib
  - App-Data-lib
  - Root



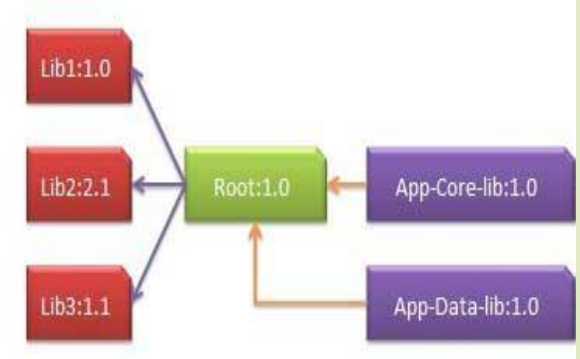
a. POM - App-UI-WAR

```
<groupId>com.companyname.groupname</groupId>
<artifactId>App-UI-WAR</artifactId>
<version>1.0</version>
<packaging>war</packaging>
<dependencies>
  <dependency>
    <groupId>com.companyname.groupname</groupId>
    <artifactId>App-Core-lib</artifactId>
    <version>1.0</version>
  </dependency>
  <dependency>
    <groupId>com.companyname.groupname</groupId>
    <artifactId>App-Data-lib</artifactId>
    <version>1.0</version>
  </dependency>
</dependencies>
```



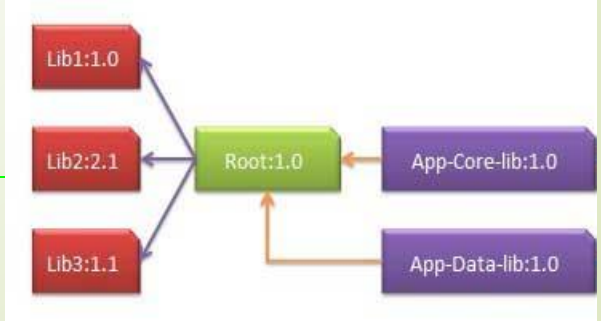
b. POM - App-Core-lib

```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>Root</artifactId>
    <groupId>com.companyname.groupname</groupId>
    <version>1.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.companyname.groupname</groupId>
  <artifactId>App-Core-lib</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
</project>
```



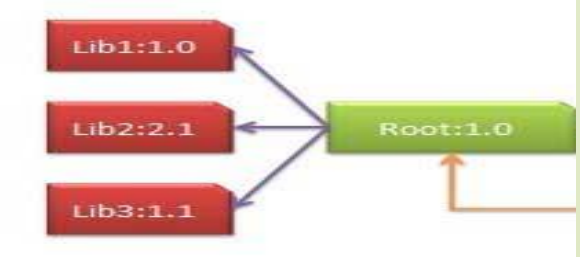
c. POM - App-Data-lib

```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>Root</artifactId>
    <groupId>com.companyname.groupname</groupId>
    <version>1.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.companyname.groupname</groupId>
  <artifactId>App-Data-lib</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
</project>
```



d. POM - Root

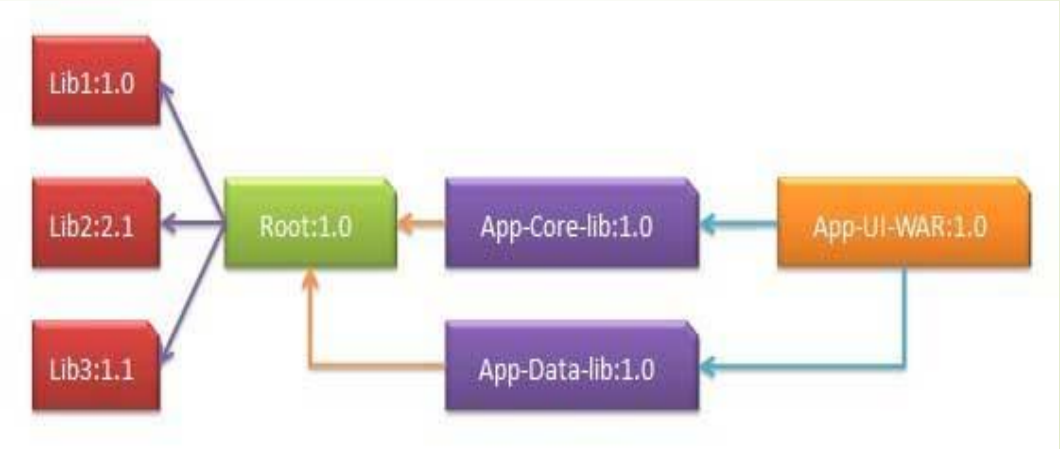
```
<groupId>com.companyname.groupname</groupId>
<artifactId>Root</artifactId>
<version>1.0</version>
<packaging>pom</packaging>
<dependencies>
  <dependency>          <groupId>com.companyname.groupname1</groupId>
      <artifactId>Lib1</artifactId>
      <version>1.0</version>
  </dependency>
    <dependency>        <groupId>com.companyname.groupname2</groupId>
      <artifactId>Lib2</artifactId>
      <version>2.1</version>
  </dependency>
    <dependency>        <groupId>com.companyname.groupname3</groupId>
      <artifactId>Lib3</artifactId>
      <version>1.1</version>
  </dependency>
</dependencies>
```



8. Maven – Manage Dependencies

- Khi build dự án **App-UI-WAR**
  - Maven sẽ khám phá tất cả các phụ thuộc (dependencies) bằng cách duyệt đồ thị phụ thuộc và build ứng dụng

    => ↓ độ phức tạp & tiết kiệm thời gian cho developers



8. Maven – Manage Dependencies

- Tóm lại:
  - Các phụ thuộc chung (Common dependencies) có thể được đặt tại parent pom.
    - Các phụ thuộc của dự án App-Data-lib và App-Core-lib được liệt kê trong dự án Root.
    - Không cần đặc tả Lib1, lib2, Lib3 như các phụ thuộc trong App-UI-WAR.

    => Maven sử dụng cơ chế phụ thuộc **Transitive** để quản lý chúng.

Main Content

1. Maven – Các đặc trưng chính
2. Maven – POM
3. Maven – Build Life Cycle
4. Maven – Repositories
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. **Maven – Tích hợp với Jenkins**
10. Maven & Jenkins – Examples for testing & static code analysis

9. Maven – Tích hợp với Jenkins

- Tích hợp Maven (Build) với Jenkins

  Bước 1: Mở **Manage Jenkins**:

  - Lựa chọn Maven Plugins, tìm Maven và click: install them without the restart option
    - Màn hình (dưới)

localhost:8080/pluginManager/available

Jenkins › Plugin Manager

Maven

- [x] Maven Deployment Linker — 1.5.1
- [ ] Maven Cascade Release Plugin — 1.3.2
  Configure and perform maven release cascade
- [ ] Parasoft Findings — 10.3.2
- [x] Pipeline Maven Integration Plugin — 3.0.0
  This plugin provides integration with Pipeline, configures maven environment to use within a pipeline job by calling sh mvn or bat mvn. The selected maven installation will be configured and prepended to the path.
- [ ] PMD Plug-in — 3.49
  This plug-in generates the trend report for PMD, an open source static code analysis program.
- [x] Maven Repository Server Plugin — 1.3
  This plug-in exposes project builds as a maven repository so the artifacts can be picked up by downstream builds or other systems.
- [ ] Task Scanner Plug-in — 4.52
  This plug-in scans the workspace files for open tasks and generates a trend report.
- [ ] View Job Filters — 1.27
  Manage multiple views and hundreds of jobs much more easily. This plug-in provides more ways to include/exclude jobs from a view, including filtering by SCM path, and by any job or build status type, as well as "chaining" of filters and negating filters.
- [ ] Violation Comments to Bitbucket Server Plugin — 1.54
  Finds violations reported by code analyzers and comments Bitbucket Server (or Stash) pull requests (or commits) with them.
- [ ] Violations plugin — 0.7.11
- [ ] Warnings Plug-in

Install without restart | Download now and install after restart | Update information obtained: 22 hr ago | Check now

9. Maven – Tích hợp với Jenkins

- Tích hợp Maven với Jenkins

  Bước 2. Chọn Global Tool **Configure**

  - Xem hình (dưới)

Cách tạo repository trong Git và Gith | Maven – Download Apache Maven

localhost:8080/manage

Ứng dụng

Jenkins

search | Pham Thi Thuong | log out

Jenkins ENABLE AUTO REFRESH

- New Item
- People
- Build History
- Manage Jenkins
- My Views
- Credentials
- Lockable Resources
- Maven Repository
- New View

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

Manage Jenkins

New version of Jenkins (2.176.2) is available for download (changelog). Or Upgrade Automatically

Warnings have been published for the following currently installed components. Configure which of these warnings are shown

Jenkins 2.176.1 core and libraries:

Multiple security vulnerabilities in Jenkins 2.185 and earlier, and LTS 2.176.1 and earlier

Configure System

Configure global settings and paths.

Configure Global Security

Secure Jenkins; define who is allowed to access/use the system.

Configure Credentials

Configure the credential providers and types

Global Tool Configuration

Configure tools, their locations and automatic installers.

Global Tool Configuration

localhost:8080/configureTools

Start EN 2:32 PM

9. Maven – Tích hợp với Jenkins

- Tích hợp Maven (Build) với Jenkins
  - **Bước 2**. Chọn Global Tool **Configure**

    => Add Maven như hình (dưới):

Cách tạo repository trong Git và Giti | Maven – Download Apache Maven

localhost:8080/configureTools/

Ứng dụng

Jenkins › Global Tool Configuration

Ant installations

Add Ant

List of Ant installations on this system

Maven

Maven installations

Add Maven

Maven

Name: Maven-local

MAVEN_HOME: E:\Year_2019\setup\apache-maven-3.6.1-bin\apache-maven-3.6.1

Install automatically

Delete Maven

Add Maven

List of Maven installations on this system

Docker

Docker installations

Add Docker

List of Docker installations on this system

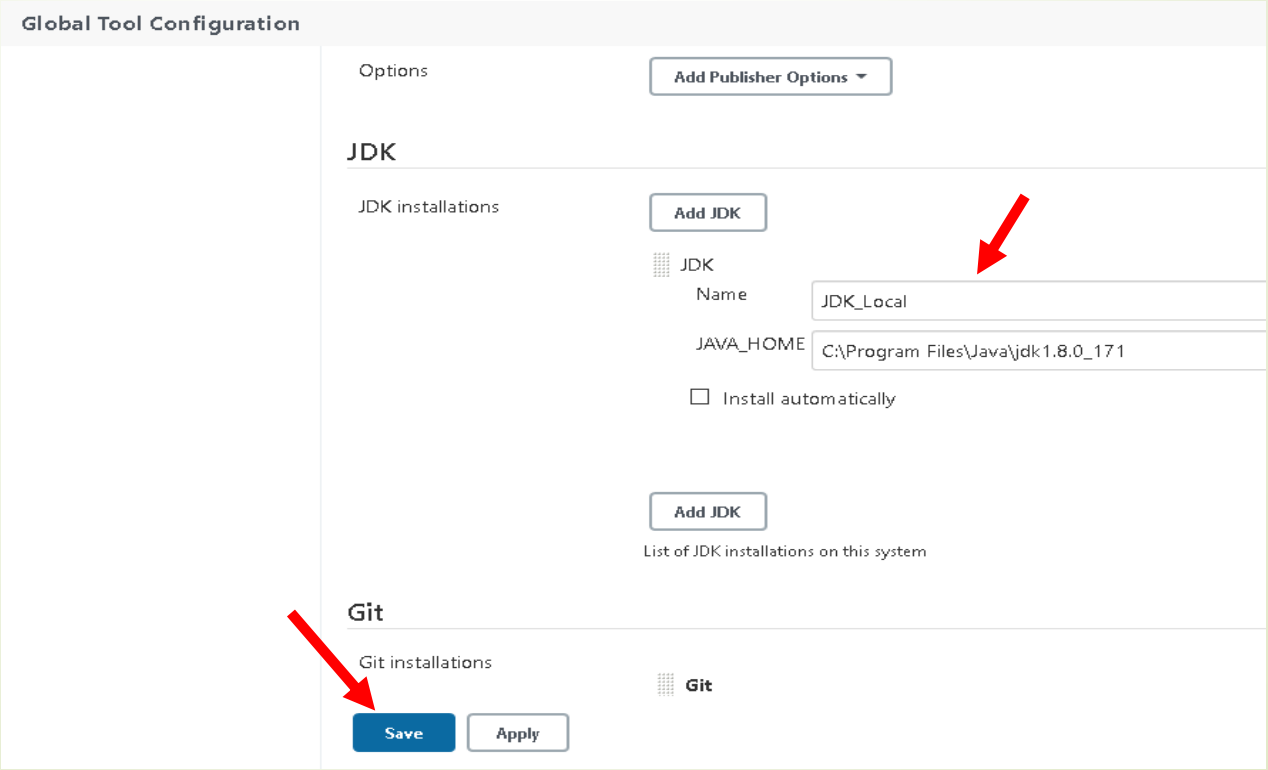
Save | Apply

Page generated: 14:33:14 ICT 22-07-2019 Jenkins ver. 2.176.1

Start | EN | 2:42 PM

9. Maven – Tích hợp với Jenkins

- Tương tự: Add JDK into Jenkins:

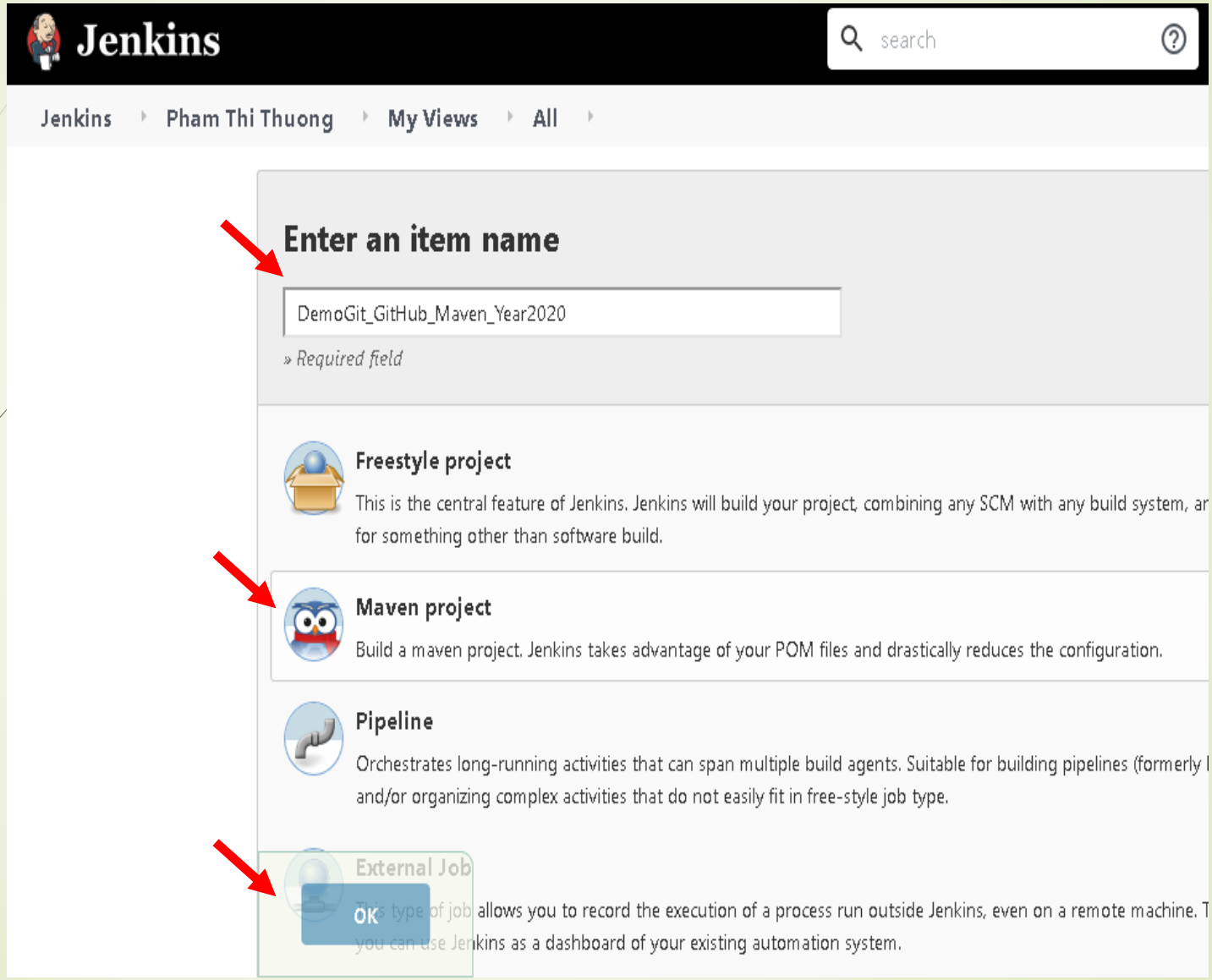


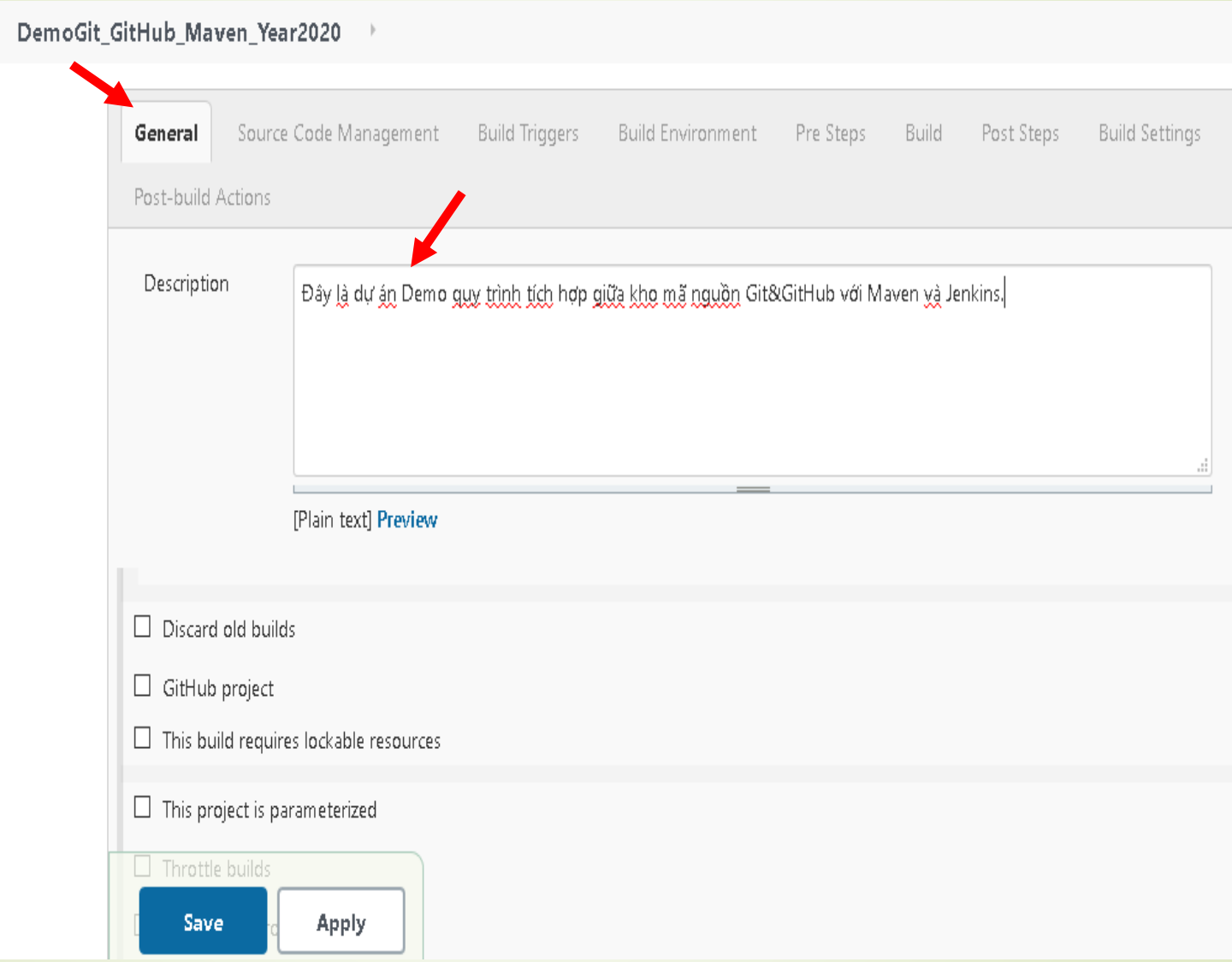
9. Maven – Tích hợp với Jenkins

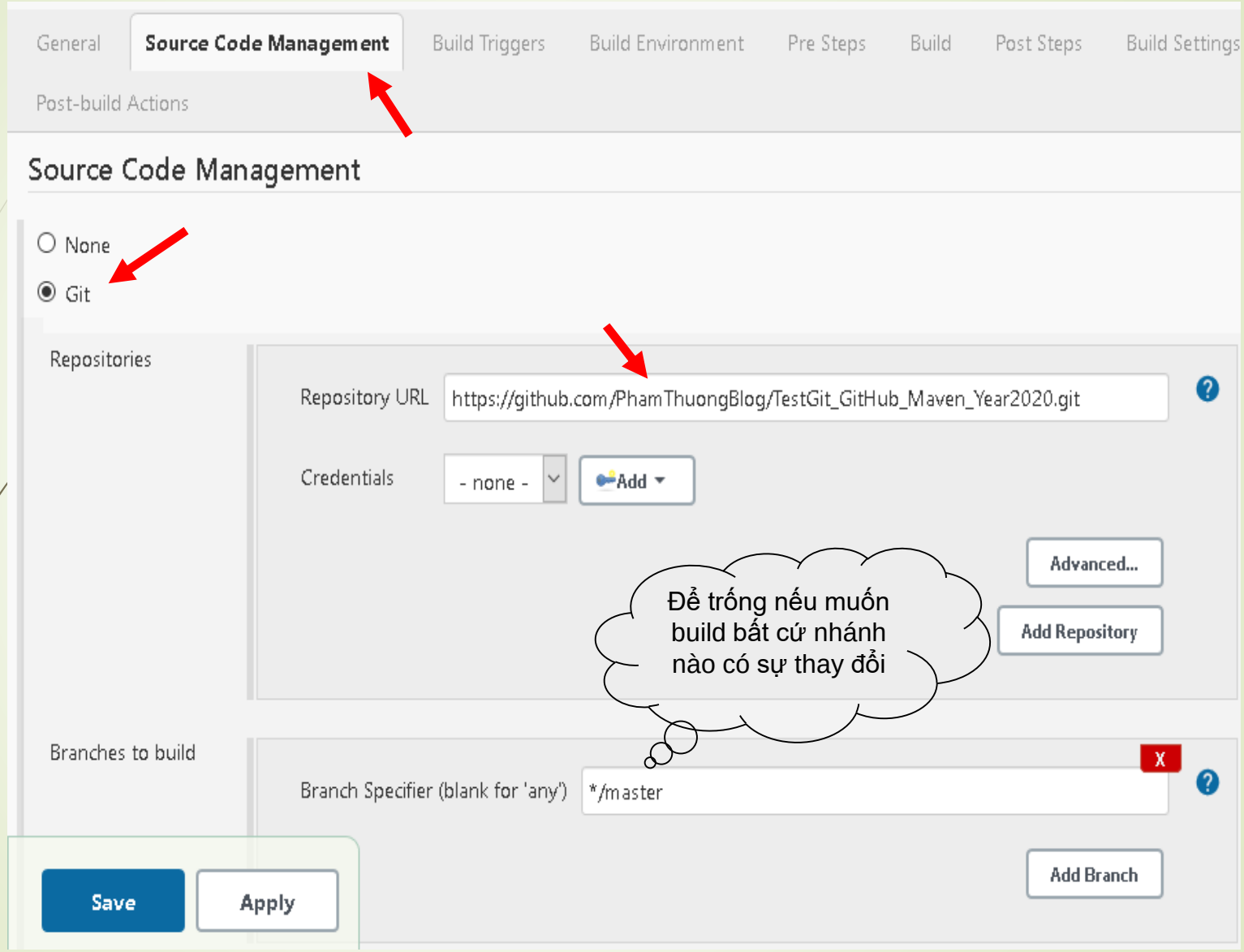
- Demo:
  - Kịch bản: Maven project + Git&GitHub + Jenkins + Maven (build, test, package, install, deploy)

Demo

- **4 Steps:**
  1. Tạo dự án Maven (ví dụ: DemoGit_GitHub_Maven_Year2020)
  2. URL của kho GitHub chứa mã nguồn dự án: https://github.com/PhamThuongBlog/TestGit_GitHub_Maven_Year2020.git
  3. Chạy thử các lệnh của Maven:
     1. compile → test → package → install → deploy
  4. Tạo dự án Maven trong Jenkins (ví dụ: DemoGit_GitHub_Maven_Year2020) & thiết lập cấu hình cho dự án như sau:


Jenkins
search
Jenkins
Pham Thi Thuong
My Views
All
Enter an item name
DemoGit_GitHub_Maven_Year2020
» Required field
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, ar
for something other than software build.
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly
and/or organizing complex activities that do not easily fit in free-style job type.
External Job
OK
allows you to record the execution of a process run outside Jenkins, even on a remote machine.
kins as a dashboard of your existing automation system.


DemoGit_GitHub_Maven_Year2020
General
Source Code Management
Build Triggers
Build Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
Description
Đây là dự án Demo quy trình tích hợp giữa kho mã nguồn Git&GitHub với Maven và Jenkins.
[Plain text] Preview
Discard old builds
GitHub project
This build requires lockable resources
This project is parameterized
Throttle builds
Save
Apply


General
Source Code Management
Build Triggers
Build Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
Source Code Management
None
Git
Repositories
Repository URL
https://github.com/PhamThuongBlog/TestGit_GitHub_Maven_Year2020.git
Credentials
- none -
Add
Advanced...
Add Repository
Để trống nếu muốn build bất cứ nhánh nào có sự thay đổi
Branches to build
Branch Specifier (blank for 'any')
*/master
X
Add Branch
Save
Apply

DemoGit_GitHub_Maven_Year2020

General | Source Code Management | **Build Triggers** | Build Environment | Pre Steps | Build | Post Steps | Build Se

Post-build Actions

Build Triggers

- [x] Build whenever a SNAPSHOT dependency is built
  - [ ] Schedule build when some upstream has no successful builds
- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] Gerrit event
- [ ] GitHub hook trigger for GITScm polling
- [ ] Poll SCM

Build Environment

- [ ] Delete workspace before build starts

Save | Apply

DemoGit_GitHub_Maven_Year2020

General | Source Code Management | Build Triggers | **Build Environment** | Pre Steps | Build | Post Steps | Build S

Post-build Actions

Build Environment

- [x] Delete workspace before build starts

Advanced...

- [ ] Use secret text(s) or file(s)
- [ ] Provide Configuration files
- [ ] Abort the build if it's stuck
- [ ] Add timestamps to the Console Output
- [ ] Define Upstream Maven Repository
- [ ] Inspect build log for published Gradle build scans
- [ ] Maven release build
- [ ] With Ant

Pre Steps

Save | Apply

DemoGit_GitHub_Maven_Year2020

General | Source Code Management | Build Triggers | Build Environment | Pre Steps | **Build** | Post Steps | Build Settings | Post-build Actions

Build

Root POM: pom.xml

Goals and options:

Advanced...

Post Steps

Post Steps

○ Run only if build succeeds ○ Run only if build succeeds or is unstable ◉ Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Execute Windows batch command

Command:

```
java -jar target/java-project-1.2-SNAPSHOT.jar
```

Save | Apply

Thuong › My Views › All › DemoGit_GitHub_Maven_Year2020 ›

General | Source Code Management | Build Triggers | Build Environment | Pre Steps | Build | **Post Steps** | Build Settings | Post-build Actions

Post Steps

○ Run only if build succeeds ○ Run only if build succeeds or is unstable ◉ Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Execute Windows batch command

Command

```
java -jar target/java-project-1.2-SNAPSHOT.jar
```

See the list of available environment variables

Advanced...

Add post-build step

Save | Apply

Demo

- 4 Steps:
  - ⇒ Finish!
- Next:
  - Click Build now để build dự án jenkins: DemoGit_GitHub_Maven_Year2020
    - Kết quả:
      - Xem hình(dưới)

Jenkins › DemoGit_GitHub_Maven_Year2020 ›

- Back to Dashboard
- Status
- Changes
- Workspace
- Build Now
- Configure
- Delete Maven project
- Modules
- Rename

Build History trend

find x

#8 Sep 26, 2020 8:47 AM

Maven project DemoGit_GitHub_Maven_Year2020

Đây là dự án Demo quy trình tích hợp giữa kho mã nguồn Git&GitHub với Maven và Jenkins.

Workspace

Recent Changes

Permalinks

Demo – Kết quả

- Xem tại:
  1. Màn hình Console **(#17)**
     - Hiện thị message build và đóng gói
  2. Kết quả test: → thư mục: Test result
  3. Kết quả đóng gói: → thư mục: Build Artifacts As Maven Repository
  4. Check the project workspace: → thư mục: target

  ⇒ Ví dụ: Xem hình (dưới)

Jenkins › Pham Thi Thuong › My Views › All › DemoGit_GitHub_Maven_Year2020 ›

Back to Dashboard

Status

Changes

Workspace

Build Now

Configure

Delete Maven project

Modules

Rename

Build History trend

find

#9 Sep 26, 2020 8:53 AM

Maven project DemoGit_GitHub_Maven_Year2020

Đây là dự án Demo quy trình tích hợp giữa kho mã nguồn Git&GitHub với Maven và Jenkins.

edit descripti

Disable Project

Workspace

Recent Changes

Latest Test Result (no failures)

Latest Test Result (no failures)

Test Result Trend

Failed Skipped Passed

1 0.8 0.6 0.4 0.2 0

#9

Permalinks

- Last build (#9), 13 min ago
- Last stable build (#9), 13 min ago
- Last successful build (#9), 13 min ago
- Last completed build (#9), 13 min ago

Workspace of DemoGit_GitHub_Maven_Year2020 on master

target /

- classes/com/thuong
- generated-sources/annotations
- generated-test-sources/test-annotations
- maven-archiver
- maven-status/maven-compiler-plugin
- surefire-reports
- test-classes/com/thuong

java-project-1.2-SNAPSHOT.jar	Sep 26, 2020 8:53:43 AM	334.32 KB	view
original-java-project-1.2-SNAPSHOT.jar	Sep 26, 2020 8:53:40 AM	3.35 KB	view

(all files in zip)

Demo – Kết quả

- Lưu ý:
  1. Tab Build trong cấu hình dự án có thể nhập các lệnh maven tại mục Goals and options
     - Ví dụ:

Demo – Kết quả

- Lưu ý:

  2. Vòng đời build mặc định của maven sẽ deploy ứng dụng đóng gói vào kho local maven

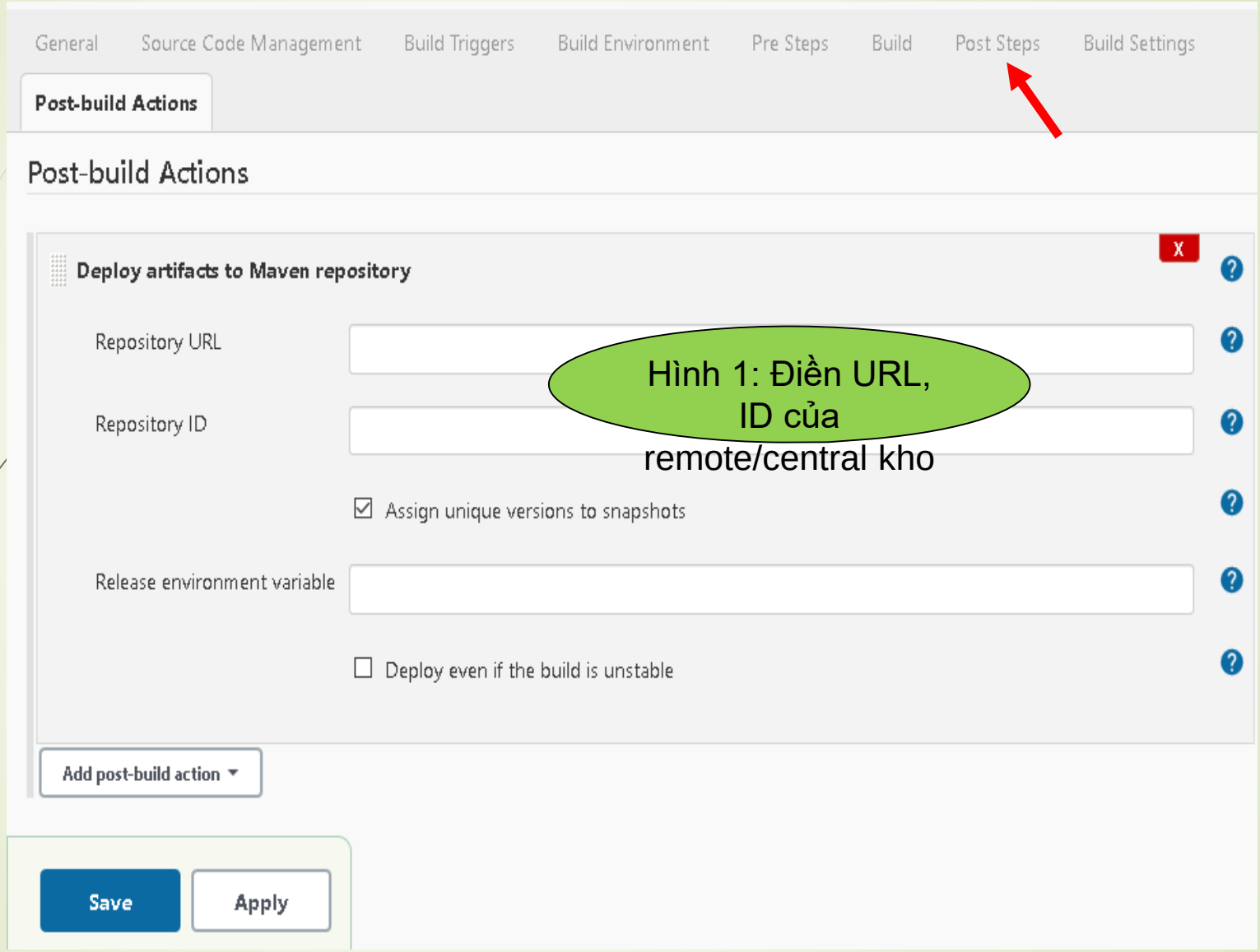
     - Muốn deploy artifacts vào kho Remote Maven

       ⇒ Cấu hình dự án như hình 1

     - Nếu deploy artifacts vào kho Nexus

       ⇒ Cần tài khoản và kho trên Nexus.

     - Nếu muốn thiết lập trạng thái build lên GitHub commit => Cấu hình dự án như hình 2



Hình 1: Điền URL, ID của remote/central kho

Thuong › My Views › All › DemoGit_GitHub_Maven_Year2020 ›

General | Source Code Management | Build Triggers | Build Environment | Pre Steps | Build | Post Steps | **Build Settings** | Post-build Actions

Build Settings

☐ E-mail Notification

Hình 2: Commit kết quả build lên GitHub

Post-build Actions

Set build status on GitHub commit [deprecated]

Build status message

Content: Build successfully!

Result on failure: FAILURE

Add post-build action

Save | Apply

Views › All › DemoGit_GitHub_Maven_Year2020 › #15

```
SNAPSHOT.jar to C:\Users\Administrator\.m2\repository\com\thuong\java-project\1.2-SNAPSHOT\java-project-1.2-SNAPSHOT.jar
[INFO] Installing C:\Users\Administrator\.jenkins\workspace\DemoGit_GitHub_Maven_Year2020\dependency-reduced-pom.xml to
C:\Users\Administrator\.m2\repository\com\thuong\java-project\1.2-SNAPSHOT\java-project-1.2-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  7.988 s
[INFO] Finished at: 2020-09-26T09:36:59+07:00
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving C:\Users\Administrator\.jenkins\workspace\DemoGit_GitHub_Maven_Year2020\dependency-reduced-pom.xml to
com.thuong/java-project/1.2-SNAPSHOT/java-project-1.2-SNAPSHOT.pom
[JENKINS] Archiving C:\Users\Administrator\.jenkins\workspace\DemoGit_GitHub_Maven_Year2020\target\java-project-1.2-
SNAPSHOT.jar to com.thuong/java-project/1.2-SNAPSHOT/java-project-1.2-SNAPSHOT.jar
channel stopped
[DemoGit_GitHub_Maven_Year2020] $ cmd /c call C:\Users\ADMINI~1\AppData\Local\Temp\jenkins1960562143190694531.bat

C:\Users\Administrator\.jenkins\workspace\D                                    arget/java-project-1.2-SNAPSHOT.jar
------------------------------------
Tong cua 10 + 20 = 30
------------------------------------


C:\Users\Administrator\.jenkins\workspace\DemoGit_GitHub_Maven_Year2020>exit 0
[Set GitHub commit status (universal)] SUCCESS on repos [] (sha:e23fb8b) with context:DemoGit_GitHub_Maven_Year2020
Finished: SUCCESS
```

Kết quả: màn hình Console:

https://github.com/PhamThuongBlog/TestGit_GitHub_Maven_Year2020

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

master | 1 branch | 0 tags | Go to file | Add file | Code

ptthuongECJ This is the first commit — e23fb8b 5 hours ago — 1 commits

.settings	This is the first commit	5 hours ago
src	This is the first commit	5 hours ago
.classpath	This is the first commit	5 hours ago
.gitignore	This is the first commit	5 hours ago
.project	This is the first commit	5 hours ago
dependency-reduced-pom.xml	This is the first commit	5 hours ago
pom.xml	This is the first commit	5 hours ago

This is the first commit

master

ptthuongECJ committed yesterday 0 parents commit e23fb8b

Showing 8 changed files with 396 additions and 0 deletions.

26 .classpath

```
@@ -0,0 +1,26 @@
+ <?xml version="1.0" encoding="UTF-8"?>
+ <classpath>
+       <classpathentry kind="src" output="target/classes" path="src/main/java">
+               <attributes>
+                       <attribute name="optional" value="true"/>
+                       <attribute name="maven.pomderived" value="true"/>
+               </attributes>
+       </classpathentry>
+       <classpathentry kind="src" output="target/test-classes" path="src/test/java">
+               <attributes>
+                       <attribute name="optional" value="true"/>
+                       <attribute name="maven.pomderived" value="true"/>
+               </attributes>
+       </classpathentry>
+       <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.
```

Main Content

1. Maven – Các đặc trưng chính
2. Maven – POM
3. Maven – Build Life Cycle
4. Maven – Repositories
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. **Maven & Jenkins – Examples for testing & static code analysis**

10. Maven & Jenkins – Examples for testing & static code analysis

a. **Unit Test: Maven + Junit**
   - How to run unit test with Maven
   - Maven – How to skip unit test
b. **Code Coverage:**
   - Maven – JaCoCo code coverage example
   - Maven – PITest mutation testing example
c. **Static Code Analysis:.**
   - Maven – SpotBugs example
   - Maven – PMD example

a. Unit Test: Maven + JUnit + Jenkins.

- **How to run unit test with Maven**
  - We can use **mvn test** to run unit test with Maven
  - Examples:

```
# Run all the unit test classes.
$ mvn test

# Run a single test class.
$ mvn -Dtest=TestApp1 test

# Run multiple test classes.
$ mvn -Dtest=TestApp1,TestApp2 test

# Run a single test method from a test class.
$ mvn -Dtest=TestApp1#methodname test
```

How to run unit test with Maven

- Example:

  $ git clone https://github.com/mkyong/maven-examples.git

  $ cd maven-examples/maven-unit-test

  $ mvn test => Run all test classes.

  $ mvn -Dtest=TestMessageBuilder test => Run a single test class TestMessageBuilder

  $ mvn -Dtest=TestMessageBuilder#testHelloWorld test

  Run a single test method testHelloWorld() from the test class TestMessageBuilder

⇒ See: Path:
E:\Namhoc_2020\HockyI_2020_2021\2_Baotri_PM__4.Demo_Projects\5.TestMaven\maven-examples\maven-unit-test

- Cấu hình dự án Jenkins (**Git_GitHub_Maven_Year2020_UnitTest**):
  - Xem các hình (dưới)

Git_GitHub_Maven_Year2020_UnitTest

General | Source Code Management | Build Triggers | Build Environment | Pre Steps | **Build** | Post Steps | Build Settings | Post-build Actions

Build

Root POM: maven-unit-test/pom.xml

Goals and options: test

Advanced...

Post Steps

○ Run only if build succeeds ○ Run only if build succeeds or is unstable ◉ Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Add post-build step

Build Settings

Save | Apply

Git_GitHub_Maven_Year2020_UnitTest › #4

```
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.mkyong.examples.TestMagicBuilder
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.03 s - in com.mkyong.examples.TestMagicBuilder
[INFO] Running com.mkyong.examples.TestMessageBuilder
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 s - in com.mkyong.examples.TestMessageBuilder
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[JENKINS] Recording test results
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.018 s
[INFO] Finished at: 2020-10-01T16:19:40+07:00
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving C:\Users\Administrator\.jenkins\workspace\Git_GitHub_Maven_Year2020_UnitTest\maven-unit-test\pom.xml
to com.mkyong.examples/maven-unit-test/1.0-SNAPSHOT/maven-unit-test-1.0-SNAPSHOT.pom
channel stopped
Finished: SUCCESS
```

Kết quả

10. Maven & Jenkins – Examples

a. **Unit Test: Maven + Junit**
   - How to run unit test with Maven
   - **Maven – How to skip unit test**

b. **Code Coverage:**
   - Maven – JaCoCo code coverage example
   - Maven – PITest mutation testing example

c. **Static Code Analysis:**
   - Maven – SpotBugs example
   - Maven – PMD example

a. Unit Test: Maven + JUnit + Jenkins.

- **Maven – How to skip unit test**
  - By default, when building project, Maven will run the entire unit tests automatically. If any unit tests is failed, it will force Maven to abort the building process. In real life, we may **STILL** need to build your project even some of the cases are failed.

    => a few ways to skip the unit test:

Maven – How to skip unit test

- **Cách 1**: Define a system property **-Dmaven.test.skip=true**

```
$ mvn package –Dmaven.test.skip=true
#no test
```

- **Cách 2:** Define this system property in the POM file:

```
<properties>
    <maven.test.skip>true</maven.test.skip>
</properties>
```

```
Terminal
```

```
$ mvn package
#no test
```

10. Maven & Jenkins – Examples

a. **Unit Test: Maven + Junit**
   - How to run unit test with Maven
   - Maven – How to skip unit test
b. **Code Coverage:**
   - **Maven – JaCoCo code coverage example**
   - Maven – PITest mutation testing example
c. **Static Code Analysis:**
   - Maven – SpotBugs example
   - Maven – PMD example

b. Code Coverage: Maven + Code coverage report integration.

- **Maven – JaCoCo code coverage**
  - How to use a JaCoCo Maven plugin to generate a code coverage report for a Java project
    - JaCoCo is used to measure how many code lines are tested
  - **Steps?**

Maven – JaCoCo code coverage

- **Steps:**
  1. Khai báo JaCoCo plugin trong file POM:
     - See Fig
       - More details: See here

     => It will run the JaCoCo 'report' goal during the Maven test phase.

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.2</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <!-- attached to Maven test phase -->
        <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Maven – JaCoCo code coverage

- **Steps:**

  2. Unit Test

     - Create a simple Java code:
       - See Fig
       - More details:
         - See here

MessageBuilder.java

```java
package com.mkyong.examples;

public class MessageBuilder {

    public String getMessage(String name) {

        StringBuilder result = new StringBuilder();

        if (name == null || name.trim().length() == 0) {

            result.append("Please provide a name!");

        } else {

            result.append("Hello " + name);

        }
        return result.toString();
    }

}
```

Maven – JaCoCo code coverage

- **Steps:**
  2. Unit Test
     - Create a Unit test for above class
       - See Fig
       - More details:
         - See here

TestMessageBuilder.java

```java
package com.mkyong.examples;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class TestMessageBuilder {

    @Test
    public void testNameMkyong() {

        MessageBuilder obj = new MessageBuilder();
        assertEquals("Hello mkyong", obj.getMessage("mkyong"));

    }

}
```

Maven – JaCoCo code coverage

Step 3. Run:

- ***cd maven-code-coverage***
- ***mvn clean test***

  => JaCoCo code coverage report will be generated at target/site/jacoco/*

- Open target/site/jacoco/index.html file, to see results:

maven-code-coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.mkyong.examples		84%		50%	2	4	1	6	0	2	0	1
Total	5 of 32	84%	2 of 4	50%	2	4	1	6	0	2	0	1

Maven – JaCoCo code coverage

Results:

- Green – Code is tested or covered.
- Red – Code is not tested or covered.
- Yellow – Code is partially tested or covered.

MessageBuilder.java

```
1.  package com.mkyong.examples;
2.
3.  public class MessageBuilder {
4.
5.      public String getMessage(String name) {
6.
7.          StringBuilder result = new StringBuilder();
8.
9.          if (name == null || name.trim().length() == 0) {
10.
11.             result.append("Please provide a name!");
12.
13.         } else {
14.
15.             result.append("Hello " + name);
16.
17.         }
18.         return result.toString();
19.     }
20.
21. }
```

Maven – JaCoCo code coverage

- **Step 4**: Improving the Unit Test
  - Adding one more test for the red line:

```
@Test
public void testNameEmpty() {

    MessageBuilder obj = new MessageBuilder();
    assertEquals("Please provide a name!", obj.getMessage(" "));

}
```

  - Review the report again: ***mvn clean test***

⇒ result: index.html

⇒ See Fig (below):

maven-code-coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.mkyong.examples		100%		75%	1	4	0	6	0	2	0	1
Total	0 of 32	100%	1 of 4	75%	1	4	0	6	0	2	0	1

MessageBuilder.java

```
1.  package com.mkyong.examples;
2.
3.  public class MessageBuilder {
4.
5.      public String getMessage(String name) {
6.
7.          StringBuilder result = new StringBuilder();
8.
9.          if (name == null || name.trim().length() == 0) {
10.
11.             result.append("Please provide a name!");
12.
13.         } else {
14.
15.             result.append("Hello " + name);
16.
17.         }
18.         return result.toString();
19.     }
20.
21. }
```

Yellow – Code is partially tested or covered.

Maven – JaCoCo code coverage

- Add one more test for the yellow line:

```
@Test
public void testNameNull() {

    MessageBuilder obj = new MessageBuilder();
    assertEquals("Please provide a name!", obj.getMessage(null));

}
```

  - Review the report again: ***mvn clean test***
    - Results:
      - See Fig (below):

maven-code-coverage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.mkyong.examples		100%		100%	0	4	0	6	0	2	0	1
Total	0 of 32	100%	0 of 4	100%	0	4	0	6	0	2	0	1

MessageBuilder.java

```
1.  package com.mkyong.examples;
2.
3.  public class MessageBuilder {
4.
5.      public String getMessage(String name) {
6.
7.          StringBuilder result = new StringBuilder();
8.
9.          if (name == null || name.trim().length() == 0) {
10.
11.             result.append("Please provide a name!");
12.
13.         } else {
14.
15.             result.append("Hello " + name);
16.
17.         }
18.         return result.toString();
19.     }
20.
21. }
```

Test cases bao phủ toàn bộ các nhánh và các lệnh (100% coverage)

10. Maven & Jenkins – Examples

a. **Unit Test: Maven + Junit**
   - How to run unit test with Maven
   - Maven – How to skip unit test
b. **Code Coverage:**
   - Maven – JaCoCo code coverage example
   - **Maven – PITest mutation testing example**
c. **Static Code Analysis:**
   - Maven – SpotBugs example
   - Maven – PMD example

b. Code Coverage: Maven + Code coverage report integration.

- **Maven – PITest mutation testing**
  - How to use a Maven PIT mutation testing plugin to generate a mutation test coverage report for a Java project?
    - Note:
      - Line coverage tools like JaCoCo is just telling whether the code is tested or covered, while the PITest mutation coverage tries to tell the effectiveness of the test.

Maven – PITest mutation testing

- The mutation testing is used to measure the effectiveness of the test.
  - It use mutators (switching math operators, change the return type, remove call and etc) to mutate / change the code into different mutations (create new code based on mutators), and check if the unit test will fail for the new mutations (mutation is killed).
    - The effectiveness of the tests is measured by "how many mutations are killed".

Maven – PITest mutation testing

- To enable PIT mutation testing, put pitest-maven in POM file:
  - See the POM file (below):
  - POM file (more details: see here)

```
<plugin>
        <groupId>org.pitest</groupId>
        <artifactId>pitest-maven</artifactId>
        <version>1.4.3</version>
        <executions>
                <execution>
                        <id>pit-report</id>
                        <!-- optional, this example attached the goal into mvn test
phase -->
                        <phase>test</phase>
                        <goals>
                                <goal>mutationCoverage</goal>
                        </goals>
                </execution>
        </executions>
        <!-- https://github.com/hcoles/pitest/issues/284 -->
        <!-- Need this to support JUnit 5 -->
        <dependencies>
                <dependency>
                        <groupId>org.pitest</groupId>
                        <artifactId>pitest-junit5-plugin</artifactId>
                        <version>0.8</version>
                </dependency>
        </dependencies>
        <configuration>
                <targetClasses>
                        <param>com.mkyong.examples.*</param>
                </targetClasses>
                <targetTests>
                        <param>com.mkyong.examples.*</param>
                </targetTests>
        </configuration>
</plugin>
```

Maven – PITest mutation testing

- Run the PITest:
  - Manually:

    $ mvn clean org.pitest:pitest-maven:mutationCoverage

    Automatically: **$ mvn clean test**

    (Because pom.xml file attacked the "mutationCoverage" goal to Maven test phase. Now, when we run Maven test, it will trigger the PITest test)

    ⇒ Report will be generated at: /target/pit-reports/202010021010/*

    ⇒ View results by open the index.htlm in above folder

Maven – PITest mutation testing

- Examples: this code:

```
public boolean isPositive(int number) {
        boolean result = false;
        if (number >= 0) {
                result = true;
        }
        return result;
}
```

  - By default, PITest will use different mutators to transform the above code into different mutations (new code) :
    - See (below):

#1 Mutation – Changed conditional boundary (mutator)

```
public boolean isPositive(int number) {
        boolean result = false;
        if (number > 0) {// mutator - changed conditional boundary
            result = true;
        }
        return result;
}
```

#2 Mutation – Negated conditional (mutator)

```
public boolean isPositive(int number) {
        boolean result = false;
        if (false) { // mutator - negated conditional
            result = true;
        }
        return result;
}
```

#3 Mutation – Replaced return of integer sized value with (x == 0 ? 1 : 0) (mutator)

```
public boolean isPositive(int number) {
        boolean result = false;
        if (number > 0) {
            result = true;
        }
        return !result; // mutator - (x == 0 ? 1 : 0)
    }
```

Maven – PITest mutation testing

- A Good unit test:
  - Should fail (kill) all the mutations #1,#2,#3

```
@Test
    public void testPositive() {
        CalculatorService obj = new CalculatorService();
        assertEquals(true, obj.isPositive(10));
    }
```

=> This unit test will kill the mutation #2 and #3 (unit test is failed), but the mutation #1 is survived (unit test is passed)

Maven – PITest mutation testing

- Run the PITest:

  $mvn clean test

  => Results:

  - See Fig (below):

CalculatorService.java

```
1   package com.mkyong.examples;
2
3   public class CalculatorService {
4
5       public boolean isPositive(int number) {
6
7           boolean result = false;
8 2         if (number >= 0) {
9               result = true;
10          }
11 1        return result;
12
13      }
14
15  }
```

Mutations

```
8   1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
11  1. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED
```

Active mutators

- INCREMENTS_MUTATOR
- VOID_METHOD_CALL_MUTATOR
- RETURN_VALS_MUTATOR

Maven – PITest mutation testing

- To fail (kill) this test (mutation)
  - ⇒ We need improving the unit test by testing the number zero:

```
@Test
  public void testPositive() {
      CalculatorService obj = new CalculatorService();
      assertEquals(true, obj.isPositive(10));
      //kill mutation #1
      assertEquals(true, obj.isPositive(0));
  }
```

=> Result of run: $mvn clean test

- See index.htlm file => Fig (below)

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
1	100%	5/5	100%	3/3

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
com.mkyong.examples	1	100%	5/5	100%	3/3

CalculatorService.java

```
1   package com.mkyong.examples;
2
3   public class CalculatorService {
4
5       public boolean isPositive(int number) {
6
7           boolean result = false;
8 2         if (number >= 0) {
9               result = true;
10          }
11 1        return result;
12
13      }
14
15  }
```

Mutations

8 1. changed conditional boundary → KILLED
2. negated conditional → KILLED

11 1. replaced return of integer sized value with (x == 0 ? 1 : 0) → KILLED

10. Maven & Jenkins – Examples

a. **Unit Test: Maven + Junit**
   - How to run unit test with Maven
   - Maven – How to skip unit test
b. **Code Coverage:**
   - Maven – JaCoCo code coverage example
   - Maven – PITest mutation testing example
c. **Static Code Analysis:**
   - **Maven – SpotBugs example**
   - Maven – PMD example

c. Static Code Analysis: Maven + Static code analysis report

- **Maven – SpotBugs**
  - How to use SpotBugs Maven Plugin to find bugs in Java code?

    ⇒ **Steps?**

Maven – SpotBugs

- **Steps:**
  - **Step 1**: Define the spotbugs-maven-plugin in the reporting tag of the POM.xlm file:

```
<reporting>
    <plugins>
        <plugin>
            <groupId>com.github.spotbugs</groupId>
            <artifactId>spotbugs-maven-plugin</artifactId>
            <version>3.1.8</version>
        </plugin>
    </plugins>
</reporting>
```

    - More details: See here
  - **Step 2**: Run: **mvn compile site**

=> The SpotBugs report: will be generated

Maven – SpotBugs

- Example:
  - A simple Java code with:
    - *An unused field 'abc' &*
    - *A performance issue in the "+ string" loop.*

    ⇒ SpotBugs will detect it & showing it on the report

    ⇒ See Java code:

    Fig (below):

```
package com.mkyong.examples;

public class StaticCodeExample {

    private int abc; //Unused field

    private String ip = "127.0.0.1";

    public void test() {

        String[] field = {"a", "b", "c", "s", "e"};

        // nối các strings sử dụng dấu “+” trong vòng lặp
        String s = "";
        for (int i = 0; i < field.length; ++i) {
            s = s + field[i];
        }

        System.out.println(ip);
    }
}
```

Maven – SpotBugs

- Run: ***$ mvn compile site***
  - SpotBugs report will be generated and integrated into the Maven site automatically
    - Review the report at target/site/soptbugs.html
      - See Fig (below):

SpotBugs Bug Detector Report

The following document contains the results of SpotBugs

SpotBugs Version is *3.1.8*

Threshold is *medium*

Effort is *default*

Summary

Classes	Bugs	Errors	Missing Classes
1	2	0	0

Files

Class	Bugs
com.mkyong.examples.StaticCodeExample	2

com.mkyong.examples.StaticCodeExample

Bug	Category	Details	Line	Priority
com.mkyong.examples.StaticCodeExample.test() concatenates strings using + in a loop	PERFORMANCE	SBSC_USE_STRINGBUFFER_CONCATENATION	18	Medium
Unused field: com.mkyong.examples.StaticCodeExample.abc	PERFORMANCE	UUF_UNUSED_FIELD		Medium

c. Static Code Analysis: Maven + Static code analysis report

- List of tools for static code analysis
  - For various program languages
    - See Fig (below)
    - More details:
      - See link:
        - https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#Java

Java [edit]

Tool	Latest release	Free software	Duplicate code	Notes
Checkstyle	2020-01-26	Yes; LGPL	No	Besides some static code analysis, it can be used to show violations of a configured coding standard. Duplicate code detection was removed[8] from Checkstyle.
Coverity	2017-01-19	No; Proprietary		Coverity is a static analysis and Static Application Security Testing (SAST) platform that finds critical defects and security weaknesses in code as it's written before they become vulnerabilities, crashes, or maintenance headaches.
Eclipse	2017-06-28	Yes; EPL	No	Cross-platform IDE with own set of several hundred code inspections available for analyzing code on-the-fly in the editor and bulk analysis of the whole project. Plugins for Checkstyle, FindBugs, and PMD.
FindBugs	2015-03-06	Yes; LGPL		Based on Jakarta BCEL from the University of Maryland. SpotBugs is the spiritual successor of FindBugs, carrying on from the point where it left off with support of its community.
Infer	2017-10-19	Yes; BSD with additional patent clause		Developed by an engineering team at Facebook with open-source contributors. Targets null pointer exceptions, leaks, and thread safety issues.
IntelliJ IDEA	2017-11-30	Yes; ASL 2	Yes	A leading Java IDE with built-in code inspection and analysis. Plugins for Checkstyle, FindBugs, and PMD.
JArchitect	2017-06-11	No; Proprietary		Simplifies managing a complex code base by analyzing and visualizing code dependencies, defining design rules, doing impact analysis, and by comparing different versions of the code.
Jtest	2019-05-21	No; Proprietary	Yes	Testing and static code analysis product by Parasoft.
LDRA Testbed		No; Proprietary		Analysis and testing tool suite.
PMD	2020-01-24	Yes; BSD, ASL 2, LGPL	Yes	A static ruleset based source code analyzer that identifies potential problems.
RIPS	2019-01-07	No; Proprietary		Language-specific source code analysis solution with many integration options for accurate detection of complex security and quality issues.

10. Maven & Jenkins – Examples

a. **Unit Test: Maven + Junit**
   - How to run unit test with Maven
   - Maven – How to skip unit test
b. **Code Coverage:**
   - Maven – JaCoCo code coverage example
   - Maven – PITest mutation testing example
c. **Static Code Analysis:**
   - Maven – SpotBugs example
   - **Maven – PMD example**

c. Static Code Analysis: Maven + Static code analysis report

- **Maven – PMD example**
  - How to use Maven PMD Plugin to analyze the Java code and display the issues in a report?
    - **Steps:**
      - Step1: Define the maven-pmd-plugin in the reporting tag:

```
<reporting>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-pmd-plugin</artifactId>
            <version>3.11.0</version>
        </plugin>
    </plugins>
</reporting>
```

Maven – PMD example

- **Steps:**
  - Step 2: Run: mvn compile site

    ⇒ The PMD report will be generated and integrated into the Maven site automatically
    - Review the report at **target/site.pmd.html**
    - Example: java code (above)

      => Result: Fig 2 (below)

maven-static-code-analysis

Last Published: 2020-10-02 | Version: 1.0-SNAPSHOT

maven-stati

Project Documentation

- Project Information
- Project Reports
  - SpotBugs
  - PMD

Built by: Maven

PMD Results

The following document contains the results of PMD 6.8.0.

Files

com/mkyong/examples/StaticCodeExample.java

Violation	Priority	Line
Avoid unused private fields such as 'abc'.	3	6
Do not hard code the IP address	3	9

10. Maven & Jenkins – Examples

a. **Unit Test: Maven + Junit**
   - How to run unit test with Maven
   - Maven – How to skip unit test

b. **Code Coverage: Maven + Code coverage report integration.**
   - Maven – JaCoCo code coverage example
   - Maven – PITest mutation testing example

c. **Static Code Analysis: Maven + Static code analysis report.**
   - Maven – SpotBugs example
   - Maven – PMD example

YÊU CẦU:

⇒ SV cần thực hành thành thạo việc tạo dự án Maven trên Jenkins và thực hành thành công các chủ đề với testing & static code analysis như đã đề cập ở trên

Summary

1. Maven – Main features
2. Maven – POM
3. Maven – Build Life Cycle
4. Maven – Repositories
5. Maven – Plugins
6. Maven – Build, Test and Run Maven project
7. Maven – Build Automation
8. Maven – Manage Dependencies
9. Maven – Tích hợp với Jenkins
10. Maven – Testing & Static Code Analysis

Discussion

Tài liệu tham khảo

- https://github.com/mkyong/maven-examples.git
- https://mkyong.com/maven/maven-jacoco-code-coverage-example/
- https://github.com/mkyong/maven-examples
- https://medium.com/@chayathilakumarai/how-to-integrate-your-selenium-project-with-jenkins-and-git-6ab9f8b492ad
- https://mkyong.com/maven/how-to-create-a-java-project-with-maven/
- https://mkyong.com/tutorials/maven-tutorials/
- https://blog.sonatype.com/maven-deploy-to-nexus
- https://medium.com/@kasunbg/maven-git-nexus-jenkins-automate-maven-releases-with-jenkins-65fbad0023d0