Programming in Java

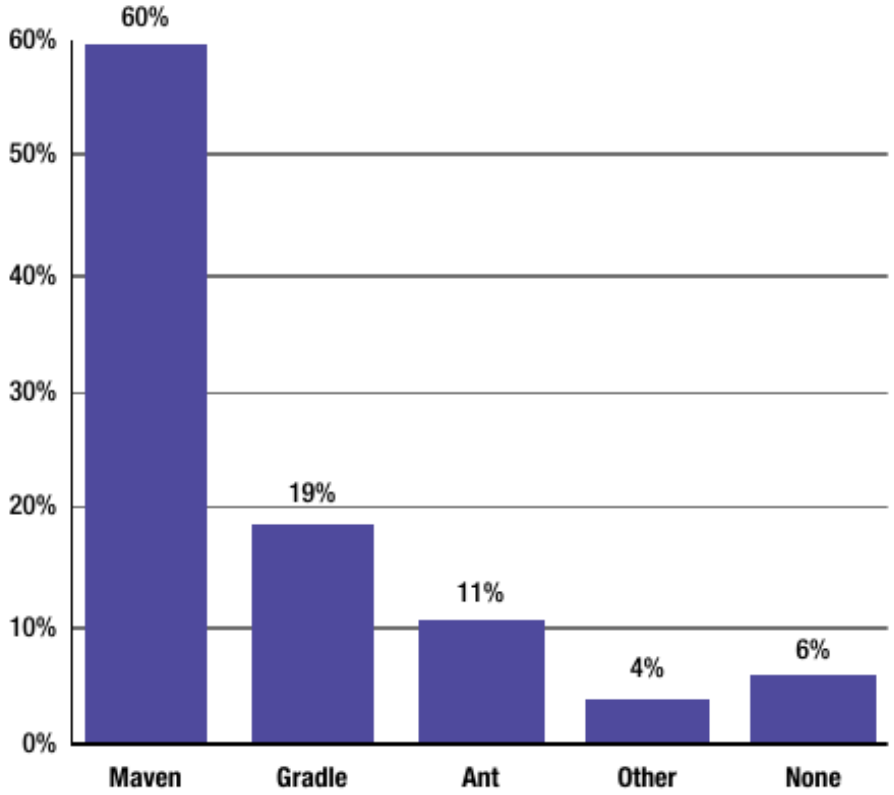
4. Maven

Maven

- Apache Maven
  - A build automation and dependency management tool for Java projects
  - Free and open source – supported as an Apache project
- Convention-over-configuration approach
  - This means that it has a set of defaults which are conventionally used in Java projects
  - But these can be overridden with user defined configuration settings
- Widely used in the Java ecosystem
  - Simplifies project builds
  - Manages dependencies automatically
  - Standardizes project structure

Build Tools

- Build task automation
  - Compile, clean, test, package and deploy
- Variety of build tools available in the Java world
  - Apache Ant is the oldest and crankiest to use
  - Gradle is an up to date version of Ant using Kotlin or Groovy scripting instead of XML
  - The most common is Maven, derived for the Apache Jakarta project



The Role of Maven

- Project Management:
  - Handles builds, documentation, and reporting
- Build Lifecycle Management:
  - Compiles, tests, packages, installs, and deploys
- Dependency Resolution:
  - Downloads and updates libraries from central repositories
- Team Collaboration:
  - Standard project setup ensures easier onboarding

Maven Lingo

- You execute goals via plugins
  - Done over the different phases of the build lifecycle, to generate artifacts.
- Examples of artifacts are jars, wars, and ears.
  - Artifacts have an artifactId, a groupId, and a version.
  - Together, these are called the artifact's "coordinates."
  - The artifacts are stored in repositories.
  - Artifacts are deployed to remote repositories and installed into local repositories.
- A POM (Project Object Model) describes a project.

Standard Structure

- Maven defines a standard project structure
- Benefits of standard structure:
  - Tools can integrate easily
  - Fewer configuration hassles
- Has been adopted as a de facto informal standard in the Java community

```
my-app/
  src/
    main/
      java/
      resources/
    test/
      java/
      resources/
  target/  (build output)
  pom.xml
```

Maven

- CICD integration
  - Maven integrates with various CICD pipeline tools like Jenkins
  - Supported by almost all IDEs
- Archetypes
  - Pre-configured project templates used to generate new projects
  - Archetypes generate all of the folders and files needed to start the project
  - Customized as needed
- Maven has created de facto standards
  - Standard directory layout: now used by other tools like Gradle
  - Artifact naming: Using a set of specific “coordinates”
  - Java dependency repositories: did not exist prior to Maven - now standarized

Maven Lifecycles

- Maven has a default life lifecycle made up of phases
  - **validate**: check if all information necessary for the build is available
  - **compile**: compile the source code
  - **test-compile:** compile the test source code
  - **test: run unit tests**
  - **package:** package compiled source code into the distributable format (jar, war, …)
  - **integration-test**: process and deploy the package if needed to run integration tests
  - **install:** install the package to a local repository
  - **deploy**: copy the package to the remote repository
- If any phase is run (maven package) for example then all of the phases prior to that are also run
  - When executing a Maven command, the “target” is one of more of the phases above
  - **maven compile** executes the Maven target “compile”

The pom file

- A maven project is described by a pom.xm file
  - Contains all the information needed to build and execute the project
- Dependency management
  - The pom file defines all the dependencies the project needs
  - Automatically fetches and caches those dependencies locally when running a build
- Build configuration
  - The operations in the build cycle have minimal functionality
  - The project is customized using plugins
  - These add specific functionality a build phase
  - There exist a large library of plugins that perform common specific tasks

Maven Coordinates

- Defines the properties of the project
  - * indicates required fields
  - *Group ID - the organization name
  - *Artifact ID - the name of the app
  - Name - the display name of app
  - Description - Doc string
  - *Version - the version of this app
- The group is like the top level package
  - Often the reverse organization url is used as the prefix for uniqueness
  - Eg. For the IBM bootcamp, a standard groupID could be `com.ibm.bootcamp`
- The artifactID is the application name

Maven Options

- As seen in the screenshot, there are a number of options available for a pom file
  - We won't be covering these in class
  - They are documented in the Apache maven documentation

```
<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.
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.lq</groupId>
    <artifactId>HelloWorld</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>Hello world App</name>
    <url>https://helloworld.com</url>
    <developers>
        <developer>
            <id>Beck</id>
            <name>Kent Beck</name>
            <email>kent@beck.com</email>
            <properties>
                <active>true</active>
            </properties>
        </developer>
        <developer>
            <id>Fowler</id>
            <name>Martin Fowler</name>
            <email>martin@flowler.com</email>
            <properties>
                <active>true</active>
            </properties>
        </developer>
    </developers>
</project>
```

Lab 4-1

Maven Basics

Maven Archetypes

- Archetypes are project templates that define the basic layout and configuration of a Maven project.
  - Automate the process of setting up a project with a consistent directory structure and a pre-configured pom.xml file.
- Why use Archetypes?
  - Save time in setting up projects
  - Ensure consistency across multiple projects and teams
  - Encourage best practices in project layout
- You invoke an archetype when you run:
  - mvn archetype:generate
  - You'll be prompted to select an archetype from a list (or specify one by ID) and provide values for groupId, artifactId, version, and package.

Maven Archetypes

- Archetypes are project templates that define the basic layout and configuration of a Maven project.
  - Automate the process of setting up a project with a consistent directory structure and a pre-configured pom.xml file.
- Why use Archetypes?
  - Save time in setting up projects
  - Ensure consistency across multiple projects and teams
  - Encourage best practices in project layout
- You invoke an archetype when you run:
  - mvn archetype:generate
  - You'll be prompted to select an archetype from a list (or specify one by ID) and provide values for groupId, artifactId, version, and package.

Maven Archetypes

- Popular Archetypes:
  - maven-archetype-quickstart: Simple Java application
  - maven-archetype-webapp: Web application with servlet structure
  - maven-archetype-archetype: Template to create your own archetypes
  - maven-archetype-j2ee-simple: Simple J2EE application project
  - maven-archetype-site-simple: Simple project site generation
  - maven-archetype-plugin: Template for creating Maven plugins


Lab 4-2
Maven Archetype Project

Dependency Management

- Maven manages external libraries (dependencies) that a project needs.
  - Dependencies are declared in pom.xml and downloaded from central or custom repositories.
  - Maven ensures the correct version is fetched and added to the classpath.
- Transitive dependencies:
  - If a dependency depends on another library,
  - Maven resolves and downloads those dependencies too.
- Dependencies are stored in the local repository (~/.m2/repository) to avoid repeated downloads.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13.2</version>
  <scope>test</scope>
</dependency>
```

Dependency Scope

- compile (default):
  - Available in all classpaths. Used for libraries required to compile and run the code.
- provided:
  - Required at compile-time but assumed to be provided by the runtime environment (e.g., servlet API in a web container).
- runtime:
  - Not required for compilation but needed at runtime (e.g., JDBC drivers).
- test:
  - Available only during test compilation and execution.
- system:
  - Similar to provided, but you have to explicitly provide the JAR path.

Dependency Management

- Dependency Conflicts:
  - If multiple versions of the same dependency are found, the nearest one in the dependency tree is used
  - Versions can be overridden explicitly in the pom.xm for transitive dependencies
- Tools for analyzing dependencies:
  - mvn dependency:tree: Shows resolved dependencies and hierarchy.
  - mvn dependency:list: Lists all resolved dependencies.
  - mvn dependency:analyze: Finds unused or undeclared dependencies.

Plugin Management

- Customized behavior in each phase can be added via plug-ins
  - Plugins are tools that perform tasks during the build process
  - Maven's functionality is implemented by plugins: both built-in and custom.
- Built-in Plugins Examples:
  - maven-compiler-plugin: Compiles Java source code.
  - maven-surefire-plugin: Runs unit tests.
  - maven-jar-plugin: Packages compiled code into a JAR file.
  - maven-clean-plugin: Cleans the target directory.
  - maven-install-plugin: Installs built artifacts into the local repository.
  - maven-deploy-plugin: Uploads the artifact to a remote repository.

Plugin Configuration

- Plugins are maintained by a large plugin development community
  - Collection curated by the Apache Maven project
  - https://maven.apache.org/plugins/
- Plugins are configured in the <build> section of pom.xml
- Plugins may require configuration parameters
  - In the example, the Java version is specified

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.10.1</version>
      <configuration>
        <source>17</source>
        <target>17</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Lab 4-3

Maven Dependencies and Plugins


Java™