Build Tools

The `DemoDev` repository examples are built on Java 8. Here is a list of resources used to build most of the examples: Eclipse and STS include Java source and API documentation.

Tool	Version	Download	Notes
Java SDK	java version "1.8.0_231"	*SDK*: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html *JCE Policy File*: http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html *Java Docs*: http://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html *Changes*: https://blogs.oracle.com/thejavatutorials/tags/jdk8	Set environment variable JAVA_HOME
Spring Tool Suite	STS/GGTS 3.9.9.RELEASE	*Product*: https://spring.io/tools/sts/all *Setup*: http://www.petermartinblog.com/2013/01/17/springsource-tool-suite/ (STS includes its own version of Maven; use *Window>Preferences>Installations* to view)	Includes Eclipse and many plugins, configure to use your installed SDK
Maven	Apache Maven 3.6.2	http://maven.apache.org/download.cgi (includes install instructions)	Set environment variable M2_HOME
Ant	Apache Ant(TM) version 1.10.6	http://ant.apache.org/bindownload.cgi	Set environment variable ANT_HOME

Most of the examples use JUnit as the testing framework. Maven accesses the required JUnit components through dependencies referencing repositories in project POMs. However, the reference site is http://www.JUnit.org and instructions are at https://github.com/junit-team/junit/wiki/Download-and-Install. Useful information about using Maven with STS/Eclipse is found in this post: http://stackoverflow.com/questions/2341004/eclipse-sts-maven.

Build Related Environment Variables

You build system will need these environment variables defined (this is a Windows example):

- ANT_HOME=C:\bin\apache-ant-1.9.4
- JAVA_HOME=C:\Java\jdk1.8.0_20
- M2_HOME=C:\bin\apache-maven-3.2.2
- Path=. . .;% M2_HOME%;. . .

Maven Background

Maven is a powerful and complex build tool that assumes very specific project directory layouts, and assumes that you will "buy into" the notion that a project produces a single build artifact (e.g., Jar file.) Initial Maven projects may be generated using *Archetypes* (a project template) and by the Eclipse M2 plugin. Maven has a "declarative" build process tied to "phases" of the build. Maven components called "plugins" are executed at various build phases to achieve goals (e.g., compile, test, and create a jar.) Since Maven can be used to invoke Ant for specialized build tasks, Ant is included in the tools list as well.

There are three popular levels of testing in TDD: unit, integration, and system tests; and all are expressed in a testing framework (e.g., JUnit.) Unit tests have no dependencies on external components; integration tests depend only on their immediate collaborators; and system tests invoke generally complete applications.

JUnit tests can cause a build problem while developing because component modifications often cause them to temporarily fail. In addition, not all tests should be run all of the time. One can always add a statement in a test to bypass executing that test, or an annotation to a test method to suppress the test, but these are not always efficient or convenient. Maven plugins may suppress testing as follows:

- Property **skipTests=true**: skips *running* the tests.
- Property **maven.test.skip=true**: skips *compiling* the tests (note, this is honored by Surefire, Failsafe and the Compiler Plugin.
- Property **skipITs =true**: skips integration tests (test names ending in IT)

The Surefire plugin operates in the project *test phase* and is preferred for unit tests. The Failsafe plugin operates in the verify phase and is preferred for integration tests. Both plugins allow test execution selection based on class naming conventions. As you may already know, Integration testing offers its own challenges, so please be sure to see http://maven.apache.org/surefire/maven-failsafe-plugin/integration-test-mojo.html for details.

Additional Maven Resources

Tutorials: http://www.tutorialspoint.com/maven/

- Integration Testing Overview: http://www.agile-engineering.net/2011/06/seperating-maven-unit-integration-tests.html
- Integration Testing with JUnit: http://www.java-tutorial.ch/maven/maven-integration-test
- From Spring: http://spring.io/blog/2011/01/17/green-beans-getting-started-with-maven-and-spring/
- Best Practices: http://books.sonatype.com/mvnref-book/reference/pom-relationships-sect-pom-best-practice.html

Testing Plugin Comparison:
https://confluence.atlassian.com/display/CLOVER/Using+with+Surefire+and+Failsafe+Plugins