Java Lecture 3

Maven; useful external libraries

What is Maven?

https://maven.apache.org/what-is-maven.html

“a tool that can now be used for building and managing any Java-based project”

“something that will make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project”

Why use Maven?

It automates Java tasks such as

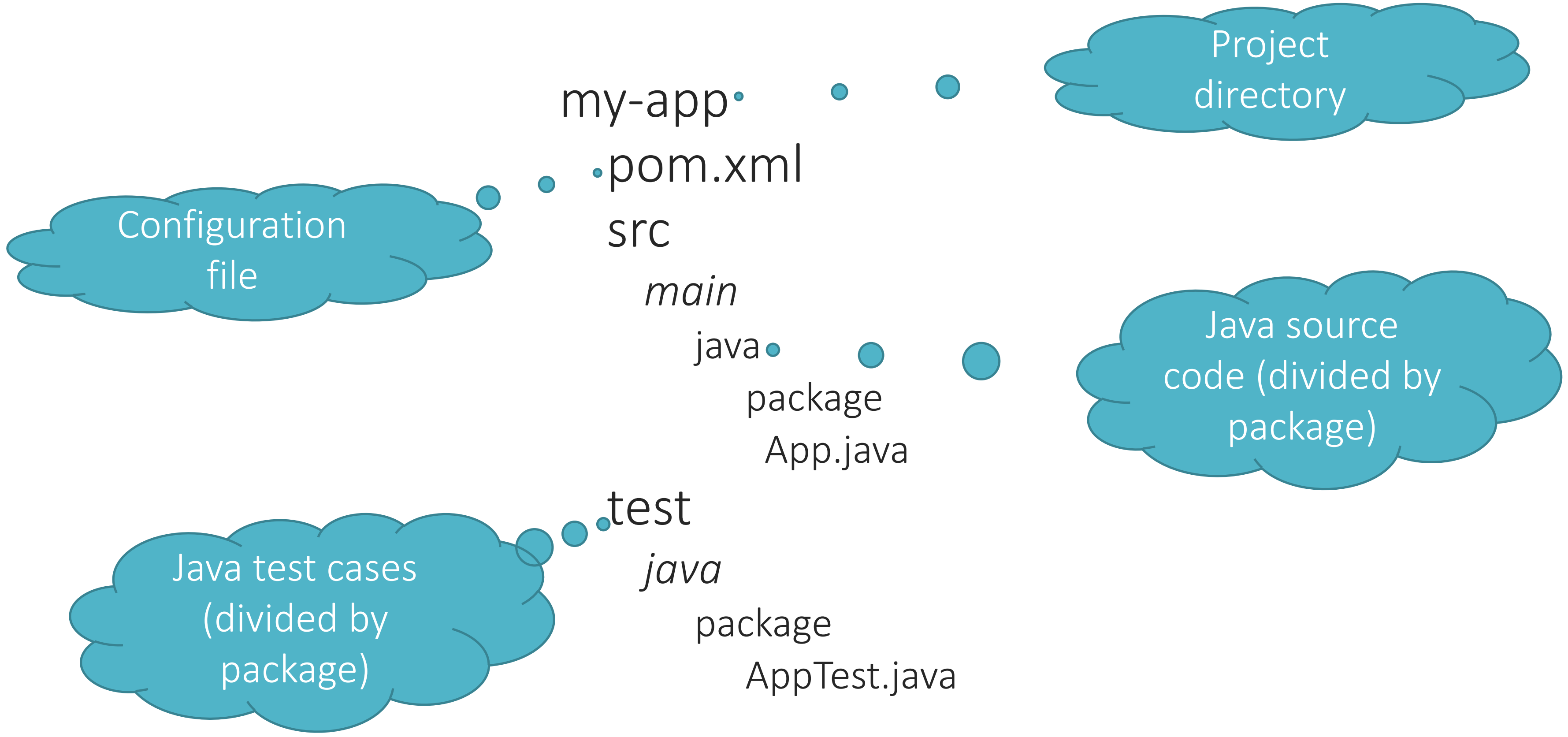
- Downloading dependencies
- Adding dependencies to build path
- Compiling source code into binary code
- Running tests
- Packaging code into “deployable artifacts” (JAR, WAR, ZIP files)
- Deploying artifacts to a server or repository

https://www.baeldung.com/maven

How is Maven different from Eclipse?

- In addition to being a code editor, Eclipse also lets you
  - Compile .java files to .class files automatically
  - Run test cases using JUnit
  - Add external libraries (you need to download them yourself and add them to the build path)
- Maven lets you do all of that, plus …
  - Download, install, link against, and update external libraries automatically
  - Package up versions of your code to install/run elsewhere
- (Note: you can run Maven in Eclipse and you will do so in the lab later)

Directory structure for a Maven project



```
my-app
  pom.xml
  src
    main
      java
        package
          App.java
  test
    java
      package
        AppTest.java
```

Project directory
Configuration file
Java source code (divided by package)
Java test cases (divided by package)

Maven configuration file: pom.xml (POM: “Project Object Model”)

```
<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">
     <modelVersion>4.0.0</modelVersion>

     <!-- content -->

</project>
```

This part is the same in every POM file

Maven “coordinates”

Project naming information:

- groupId: unique in an organisation or project (e.g., **org.apache.maven** for Maven built-in projects, **uk.ac.glasgow.dcs** for Glasgow School of CS, etc)
- artifactId: the name the project is known by (within the organisation)
- version: an identifier for the current version

```
<groupId>org.codehaus.mojo</groupId>
<artifactId>my-project</artifactId>
<version>1.0</version>
```

Dependencies

- Most projects depend on other projects to build and run correctly
- Dependencies are specified as
  - Coordinates (groupId, artifactId, version)
  - Type: how it is packaged (usually **jar**, this is the default)
  - Scope: where it is used in the project
    - ***Compile:*** *available everywhere in the project (building, running, testing, etc)*
    - ***Runtime:*** *not required for compilation, but is required for running and testing*
    - ***Test:*** *only needed for compiling and running tests*
  - You can also indicate whether dependencies are optional (e.g.., only needed for running some parts of a project)

Dependencies example

```
<dependencies>
     <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>4.12</version>
          <type>jar</type>
          <scope>test</scope>
          <optional>true</optional>
     </dependency>
     ...
</dependencies>
```

Properties

Used to configure aspects of how Maven is run (e.g., Java language version, source encoding)

```
<properties>
      <maven.compiler.source>1.7</maven.compiler.source>
      <maven.compiler.target>1.7</maven.compiler.target>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
</properties>
```

Configuring plugins

- Plugins are used by Maven to control aspects of the building, testing, etc processes
- Details of plugin and configuration are also specified in the pom.xml file
  - Under the <build> element, within the <plugins> element
- Plugins specified with
  - Coordinates
  - Additional configuration information relevant to the specific plugin

Plugin example

```
<build>
...
 <plugins>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <version>2.6</version>
    <configuration>
     <classifier>test</classifier>
    </configuration>
    <dependencies>...</dependencies>
    <executions>...</executions>
   </plugin>
  </plugins>
</build>
```

A note about Java 9+

Maven by default uses an old compiler that is not compatible with Java 9+

To fix this, you need to add the following to your **pom.xml**:

```
<properties>
    <maven.compiler.release>12</maven.compiler.release>
</properties>

<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
            </plugin>
        </plugins>
    </pluginManagement>
</build>
```

Main phases in the Maven lifecycle

compile: compile the source code of the project

test: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

package: take the compiled code and package it in its distributable format, such as a JAR.

verify: run any checks to verify the package is valid and meets quality criteria

install: install the package into the local repository, for use as a dependency in other projects locally

deploy: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

clean: cleans up artifacts created by prior builds

site: generates site documentation for this project

How to run Maven (command line*)

1. Go to the top-level directory (i.e., the directory containing **pom.xml** and **src** and **target** subdirectories)
2. Type **mvn** with the appropriate phase name(s); e.g.,
   1. **mvn package** to build the package
   2. **mvn clean** to clean up everything created by prior builds
   3. **mvn test** to test the compiled sources

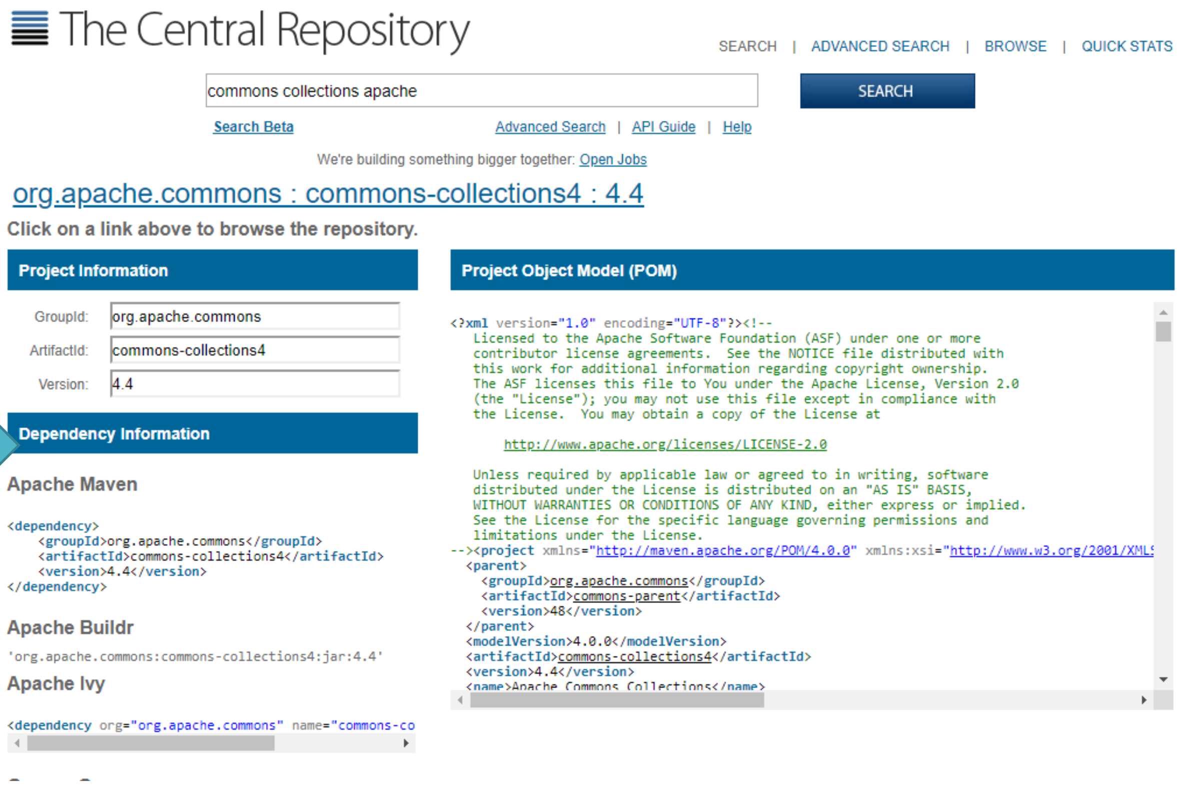
Useful external libraries

External libraries

One big advantage of building with Maven: it is extremely easy to add external libraries (i.e., libraries that aren’t built into Java) to your project

Repository of packages: Maven Repository at https://search.maven.org/

Library details from central repository



The Central Repository
SEARCH | ADVANCED SEARCH | BROWSE | QUICK STATS
commons collections apache
SEARCH
Search Beta
Advanced Search | API Guide | Help

We're building something bigger together: Open Jobs

org.apache.commons : commons-collections4 : 4.4

Click on a link above to browse the repository.

Project Information

GroupId:	org.apache.commons
ArtifactId:	commons-collections4
Version:	4.4

Dependency Information

Apache Maven

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-collections4</artifactId>
    <version>4.4</version>
</dependency>
```

Apache Buildr

```
'org.apache.commons:commons-collections4:jar:4.4'
```

Apache Ivy

```
<dependency org="org.apache.commons" name="commons-co
```

Project Object Model (POM)

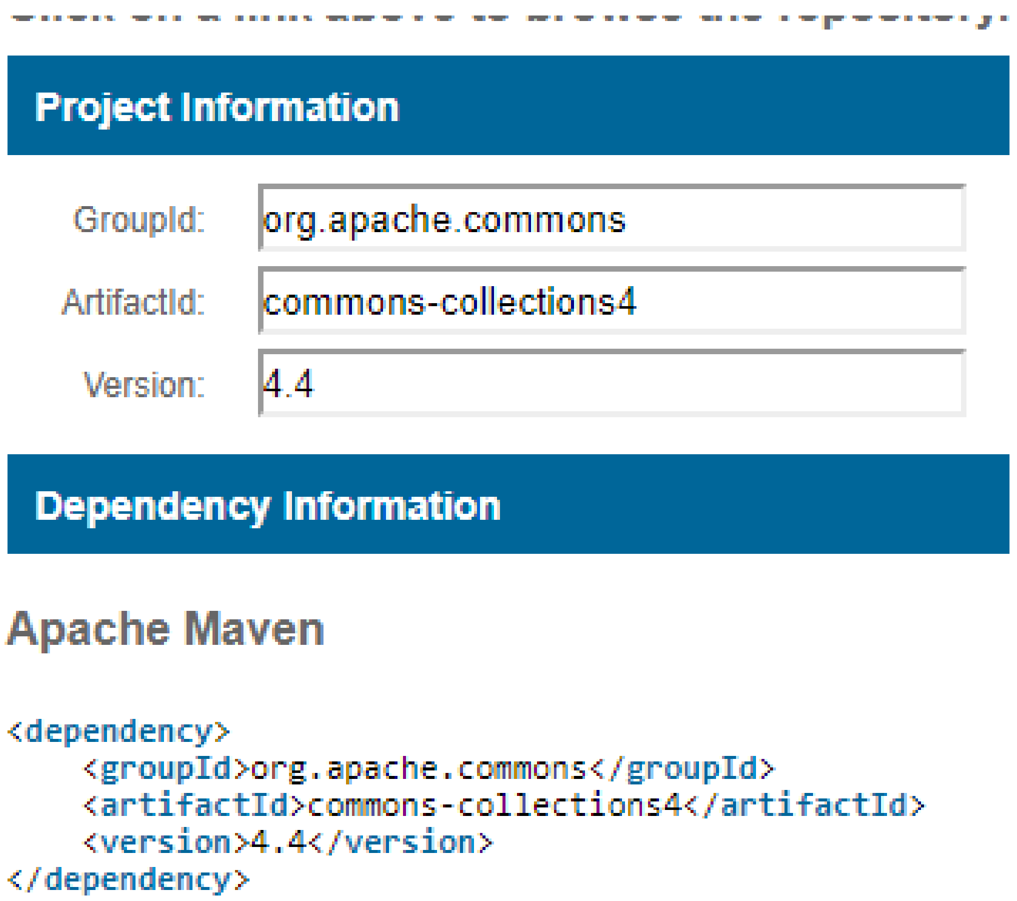
```
<?xml version="1.0" encoding="UTF-8"?><!--
   Licensed to the Apache Software Foundation (ASF) under one or more
   contributor license agreements.  See the NOTICE file distributed with
   this work for additional information regarding copyright ownership.
   The ASF licenses this file to You under the Apache License, Version 2.0
   (the "License"); you may not use this file except in compliance with
   the License.  You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
--><project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLS
  <parent>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-parent</artifactId>
    <version>48</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>commons-collections4</artifactId>
  <version>4.4</version>
  <name>Apache Commons Collections</name>
```

Zoomed in …

Add this information to your **pom.xml** and you can use this library in your project!



Project Information

GroupId:	org.apache.commons
ArtifactId:	commons-collections4
Version:	4.4

Dependency Information

Apache Maven

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-collections4</artifactId>
    <version>4.4</version>
</dependency>
```

Some useful external libraries

(List adapted from https://dzone.com/articles/20-useful-open-source-libraries-for-java-programme)

Logging libraries: Log4j, SLF4j, LogBack

JSON parsing libraries: Jackson, Gson

Unit testing: JUnit, Mockito, PowerMock

General purpose libraries (more useful versions of built-in classes): Apache Commons, Google Guava

Enhanced Collections: Commons Collections, Goldman Sachs collections, Google collections, Trove, FastUtil

Example: including Google Collections in a project

```
<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">
          <modelVersion>4.0.0</modelVersion>

          <groupId>uk.ac.glasgow.dcs</groupId>
          <artifactId>my-project</artifactId>
          <version>1.0</version>
          <dependencies>
                    <dependency>
                              <groupId>com.google.guava</groupId>
                              <artifactId>guava</artifactId>
                               <version>28.1-jre</version>
                    </dependency>
          </dependencies>
</project>
```

Then, in your code ...

```
import com.google.common.collect.Multiset;
import com.google.common.collect.HashMultiset;

public class GuavaTester {


   public static void main(String args[]) {


      //create a multiset collection

      Multiset<String> multiset = 
HashMultiset.create();


      multiset.add("a");

      multiset.add("b");

      multiset.add("c");

      multiset.add("d");

      multiset.add("a");

      multiset.add("b");

      multiset.add("c");


      //print the occurrence of an element

      System.out.println("Occurrence of 'b' : 
"+multiset.count("b"));


      //print the total size of the multiset

      System.out.println("Total Size : 
"+multiset.size());
```