

- Complete some basic exercises with our software tools
 - Maven
 - JUnit

- Maven is a tool for building and managing Java-based projects
- Provides easy access to a large repository of useful java code
 - For example, the apache commons libraries
 - The library that we will use in the exercise today is `commons-cli`¹
- Configuration details are stored in a special xml file called `pom.xml` (project object model)
- Different goals (e.g. compilation, running tests, packaging) are executed using the command `mvn <goal>`

¹<https://commons.apache.org/proper/commons-cli/>

```
<dependencies>
  <dependency>
    <groupId>commons-cli</groupId>
    <artifactId>commons-cli</artifactId>
    <version>1.4</version>
  </dependency>
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-text</artifactId>
    <version>1.2</version>
  </dependency>
</dependencies>
```

- Dependencies in Maven can be pulled into the project by adding the appropriate XML (as shown above)
- At minimum, we need to include the group and artifact ids and the version number

- The structure of a typical maven project is shown below
- `pom.xml` is the *project object model*, the main configuration for your project

```
/Project Root/  
|--/src/  
|   |--/main/  
|       |--/java/  
|       |--/webapp/  
|   |--/test/java/  
|--README.txt  
|--LICENSE.txt  
|--NOTICE.txt  
|--pom.xml
```

- A *unit test* is a check of the smallest piece of functionality in a software project: usually a single method or function
- There are different types of tests that vary in scope
 - Integration Test: run on a set of interacting objects
 - Acceptance Test: run on full systems
- In JUnit, a unit test is just a java class containing annotated methods
- In Maven, unit tests are typically placed under `/src/main/test/`

Creating a simple unit test in JUnit

```
import static org.junit.Assert.*;

import org.junit.Test;
import org.junit.*;

public class CalculatorTest {
    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(12,8);
        assertEquals(20,result,0);
    }
}
```

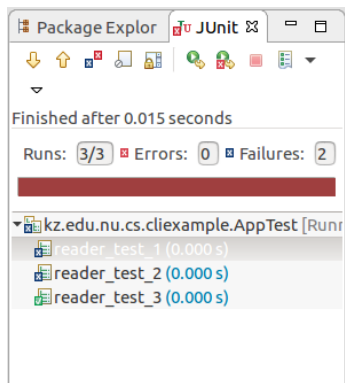
- Tests are marked using @Test
- Results are sent to the environment using different assertion statements
- assertEquals checks for equality of the first two arguments

```
public class CalculatorTest {  
  
    private Calculator calculator;  
  
    @Before  
    public void setUp() throws Exception {  
        calculator = new Calculator();  
    }  
  
    @Test  
    public void testAdd() {  
        // Calculator calculator = new Calculator();  
        double result = calculator.add(12, 8);  
        assertEquals(20,result,0);  
    }  
  
    @Test  
    public void testAdd1() {  
        // Calculator calculator = new Calculator();  
        double result = calculator.add(32, 1);  
        assertEquals(33,result,0);  
    }  
}
```

- Multiple tests that share some common setup
- Define a private instance of the object under test and include the allocation in the setUp method

- `assertArrayEquals("String",A,B)`: checks the equality of the arrays
- `assertSame("String",A,B)`: checks that the objects A and B are the same
- `assertTrue("String",condition)`: asserts that condition is true
- `fail("String")`: always cause a failure, can be used to indicate an incomplete test
- other assertions can be found at the documentation for JUnit
<http://junit.sourceforge.net/javadoc/>

- Shown at right is an example of running a unit-test in Eclipse
- A green bar is displayed when all unit tests pass, a red bar is shown if any of the tests fail
- Unit tests are, themselves, a form of documentation



- You all should have installed the appropriate software by now
- Clone (copy) the repository that is available at the URL
<https://github.com/marks1024/test-cli-repository-361>
- Import the cloned project into your IDE and complete the following 2 tasks
 - Complete the implementation of the static method wordcount so that the provided unit tests all pass
 - Edit the main application (App) so that it accepts a string as a command line argument, passes the string to wordcount, and prints the result to the console
 - For example, given the arguments -s "The cat is orange." the application should output "4" to the console
 - In Eclipse you can set command line arguments in "Run Configurations"
 - Upload to the **Lecture** moodle as a zip file