An Introduction to Unit Testing

- **A 'unit' is a small piece of your code**
  - A small piece of functionality
- **A unit test verifies the correctness of that unit of code**
  - A purist might say that in a well-written unit test, only a single 'thing' should be able to fail
  - Generally accepted rule-of-thumb: a unit test should take less than a second to complete

Why Write Unit Tests?

- **Unit testing provides verification that your code is functioning correctly**
- **Much faster than testing your entire program each time you modify the code**
  - Fastest MapReduce job on a cluster will take many seconds
    - Even in pseudo-distributed mode
  - Even running in LocalJobRunner mode will take several seconds
    - LocalJobRunner mode is discussed later in the course
  - Unit tests help you iterate faster in your code development

Why MRUnit?

- **JUnit is a popular Java unit testing framework**
- **Problem: JUnit cannot be used directly to test Mappers or Reducers**
  - Unit tests require mocking up classes in the MapReduce framework
    - A lot of work
- **MRUnit is built on top of JUnit**
  - Works with the mockito framework to provide required mock objects
- **Allows you to test your code from within an IDE**
  - Much easier to debug

JUnit Basics (1)

- **`@Test`**
  - Java annotation
  - Indicates that this method is a test which JUnit should execute
- **`@Before`**
  - Java annotation
  - Tells JUnit to call this method before every `@Test` method
    - Two `@Test` methods would result in the `@Before` method being called twice

JUnit Basics (2)

- **JUnit test methods:**
  - `assertEquals()`, `assertNotNull()` etc
    - Fail if the conditions of the statement are not met
  - `fail(msg)`
    - Explicitly fails the test with the given error message
- **With a JUnit test open in Eclipse, run all tests in the class by going to Run ➔ Run**
- **Eclipse also provides functionality to run all JUnit tests in your project**
- **Other IDEs have similar functionality**

JUnit: Example Code

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

import org.junit.Before;
import org.junit.Test;

public class JUnitHelloWorld {
  protected String s;
  @Before
  public void setup() {
    s = "HELLO WORLD";
  }
  @Test
  public void testHelloWorldSuccess() {
    s = s.toLowerCase();
    assertEquals("hello world", s);
  }

  @Test
  public void testHelloWorldFail() {
    assertEquals("hello world", s);
  }
}
```

Using MRUnit to Test MapReduce Code

- **MRUnit builds on top of JUnit**
- **Provides a mock `InputSplit` and other classes**
- **Can test just the Mapper, just the Reducer, or the full MapReduce flow**

MRUnit: Example Code – Mapper Unit Test (1)

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.junit.Before;
import org.junit.Test;

public class TestWordCount {
  MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;

  @Before
  public void setUp() {
    WordMapper mapper = new WordMapper();
    mapDriver = new MapDriver<LongWritable, Text, Text, IntWritable>();
    mapDriver.setMapper(mapper);
  }

  @Test
  public void testMapper() {
    mapDriver.withInput(new LongWritable(1), new Text("cat dog"));
    mapDriver.withOutput(new Text("cat"), new IntWritable(1));
    mapDriver.withOutput(new Text("dog"), new IntWritable(1));
    mapDriver.runTest();
  }
}
```

MRUnit: Example Code – Mapper Unit Test (2)

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.junit.Before;
import org.junit.Test;

public
  MapD

  @Bef
  publ
    Wo
    mapDriver = new MapDriver<LongWritable, Text, Text, IntWritable>();
    mapDriver.setMapper(mapper);
  }

  @Test
  public void testMapper() {
    mapDriver.withInput(new LongWritable(1), new Text("cat dog"));
    mapDriver.withOutput(new Text("cat"), new IntWritable(1));
    mapDriver.withOutput(new Text("dog"), new IntWritable(1));
    mapDriver.runTest();
  }
}
```

Import the relevant JUnit classes and the MRUnit `MapDriver` class as we will be writing a unit test for our Mapper.

MRUnit: Example Code – Mapper Unit Test (3)

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.junit.Before;
import org.junit.Test;

public class TestWordCount {
  MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;

  @Bef
  publ
    Wo
    ma
    mapDriver.setMapper(mapper);
  }

  @Test
  public void testMapper() {
    mapDriver.withInput(new LongWritable(1), new Text("cat dog"));
    mapDriver.withOutput(new Text("cat"), new IntWritable(1));
    mapDriver.withOutput(new Text("dog"), new IntWritable(1));
    mapDriver.runTest();
  }
}
```

`MapDriver` is an MRUnit class (not a user-defined driver).

MRUnit: Example Code – Mapper Unit Test (4)

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.junit.Before;
import org.junit.Test;

public class TestWordCount {
  MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;

  @Before
  public void setUp() {
    WordMapper mapper = new WordMapper();
    mapDriver = new MapDriver<LongWritable, Text, Text, IntWritable>();
    mapDriver.setMapper(mapper);
  }

  @Te
  pub
    ma
    ma
    ma
    mapDriver.runTest();
  }
}
```

Set up the test. This method will be called before every test, just as with JUnit.

MRUnit: Example Code – Mapper Unit Test (5)

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mrunit.mapreduce.MapDriver;
import org.junit.Before;
import org.junit.Test;

public class TestWordCount {
  MapDriver<LongWritable, Text, Text, IntWritable> mapDriver;

  @Before
  public
    Word
    mapD
    mapD
  }

  @Test
  public void testMapper() {
    mapDriver.withInput(new LongWritable(1), new Text("cat dog"));
    mapDriver.withOutput(new Text("cat"), new IntWritable(1));
    mapDriver.withOutput(new Text("dog"), new IntWritable(1));
    mapDriver.runTest();
  }
}
```

The test itself. Note that the order in which the output is specified is important – it must match the order in which the output will be created by the Mapper.

MRUnit Drivers (1)

- **MRUnit has a `MapDriver`, a `ReduceDriver`, and a `MapReduceDriver`**
- **Methods to specify test input and output:**
  - `withInput`
    - Specifies input to the Mapper/Reducer
    - Builder method that can be chained
  - `withOutput`
    - Specifies expected output from the Mapper/Reducer
    - Builder method that can be chained
  - `addOutput`
    - Similar to `withOutput` but returns `void`

MRUnit Drivers (2)

- **Methods to run tests:**
  - `runTest`
    - Runs the test and verifies the output
  - `run`
    - Runs the test and returns the result set
    - Ignores previous `withOutput` and `addOutput` calls
- **Drivers take a single (key, value) pair as input**
- **Can take multiple (key, value) pairs as expected output**
- **If you are calling `driver.runTest()` or `driver.run()` multiple times, call `driver.resetOutput()` between each call**
  - MRUnit will fail if you do not do this

Running Unit Tests From Eclipse