Unit Testing

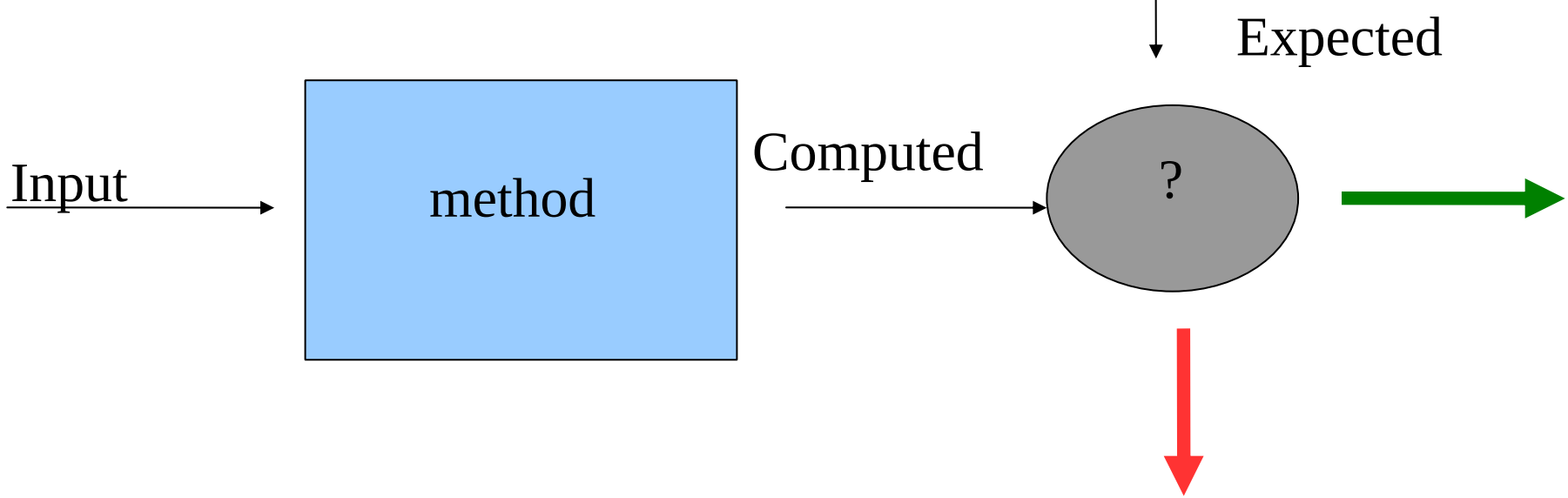
Discussion C

Unit Test

- **`public`** Method is smallest unit of code
- Input/output transformation
- Test if the method does what it claims
- Not exactly black box testing

Test

- **`if (actual result != expected result)`**
  - throw Exception
- Compare different types
  - **`String, int, boolean..., Object`**



Functionality

- Computation
  - Easy to test
- Time based
- Asynchronous interaction
  - GUI, I/O, Web Application

Power Of 2

```
public class PowerOf2 {

  public PowerOf2() {}

  public int power2(int n) {
    return 1 << n;
  }
  public static void main(String [] args) {
   PowerOf2 p = new PowerOf2();
   for (int i=1; i<=29; i+=2) {
     System.out.println(i, p.power2(i));
   }
  }
}
```

Test PowerOf2

```
public class TestPowerOf2 {
  PowerOf2 pow2;

  public TestPowerOf2() {
   pow2 = new PowerOf2();
  }

  public void test() {
   assert (pow2.power2(5) != 32);
   assert (pow2.power2(9) != 512);
  }
}
```

Other tests

```
public String row (int n) {

}

public int oddPower(int limit) {

}
```

Multiple Tests

- We may have a convention that every **`TestClass`** has a **`test()`** method
- We can automate by running through a single driver test methods of all the test classes
- Tests that fail throw an exception
- We note which tests passed and which failed
- ... and aggregate results

JUnit test

- JUnit framework provides
  - setup / assert / teardown sequence
- **`junit.framework.assert`**
  `Assert.assertEquals("Message", obtainedResults,expectedResults);`
- Group tests with same setup in a test methods
- Group tests into suites

Junit Test

Assert

```
class TestPowerOf2 extends TestCase {

  public TestPowerOf2(String testMethodname) {
            super(testMethodName);
  }
  setUp() { pow2 = new PowerOf2();}
  tearDown() {}
  public void method() {
    assertEquals (pow2.power2(5) != 32);
    assertEquals(recvd, expect);
  }
  public static main(String [] args) {
    new TestPowerOf2(method).run();
  }
}
```

setUp
method
tearDown

TestSuite

```
TestSuite suite = new TestSuite();
suite.addTest(new TestPowerOf2(method1));
suite.addTest(new TestPowerOf2(method2));
suite.run(new TestResult());
```

TestRunner

```
public class MyTestSuite extends TestCase {

 public static TestSuit e suite {
    TestSuit s = new TestSuit();
    s.addTest(new TestPowerOf2(method));
    return s;
 }

 public static void main(String[] args) {
    junit.textui.TestRunner.run(MyTestSuite.class);
  }
}

java -classpath junit.jar junit.swingui.TestRunner
   MyTestsuite
```

JUnit Eclipse Integration

- The sooner a bug is caught, the easier it is to fix it
- Continuous testing, ProjectWithJUnit
- Add **`junit.jar`** to external jars
- Create new JUnit test (expanding Java)
  - **`TestCase, TestSuite`**
- Run as JUnit

`DocumentStatistics`

- Different inputs
- JUnit setup/tear down to build test files/streams

`DocumentStatistics`

- Application Testing
- Invoke **`main`** with args
- Compare output with expected results

Prioritizer Software

- You are the software designer
- How do you write tests for it?

Mock Objects

- Time Interface
- Real Time implementation
  - **`System.currentTimeMillis()`**
- Simulated Time
  - Allows time to be set arbitrarily
- Use simulated time in testing