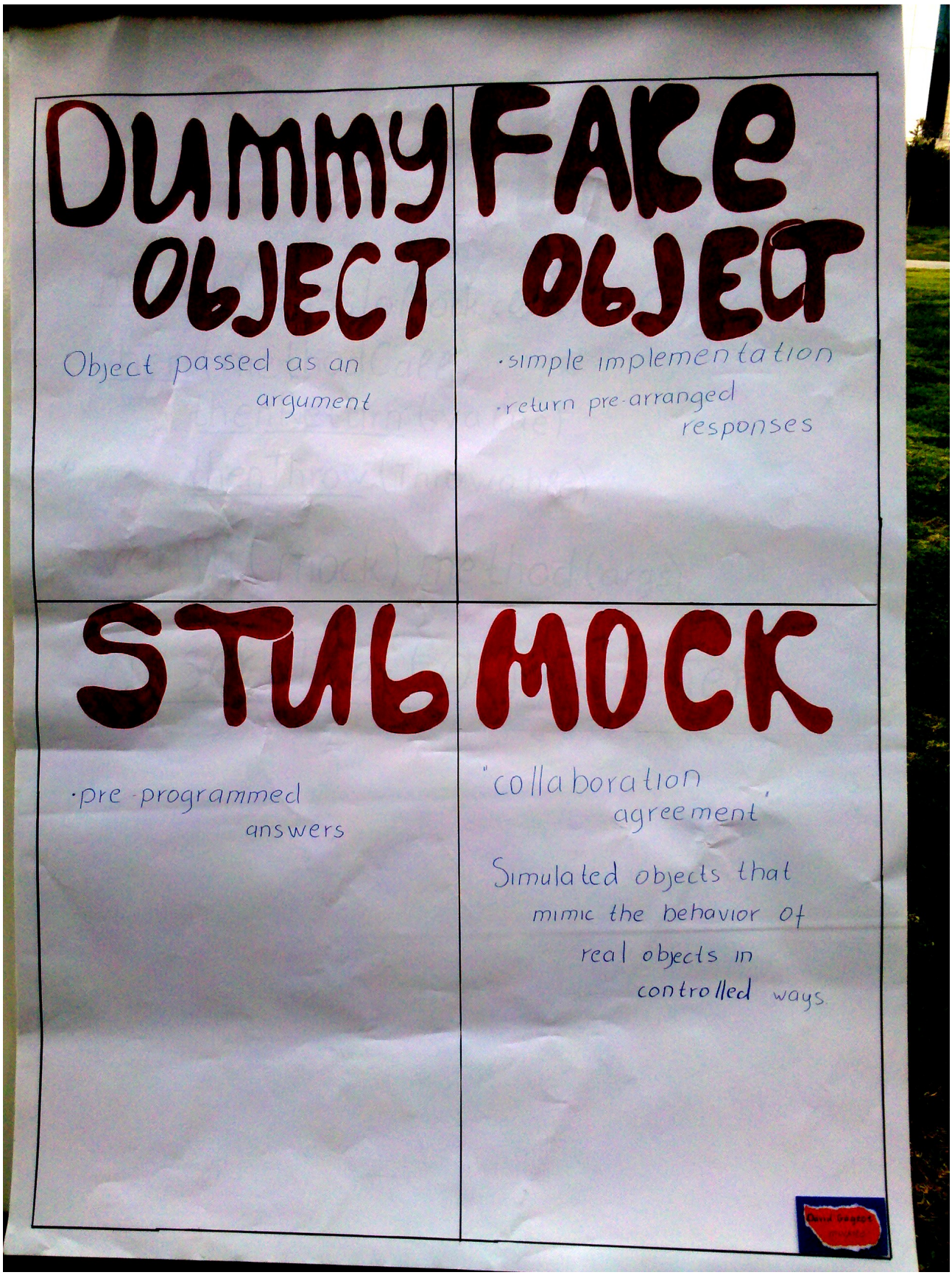
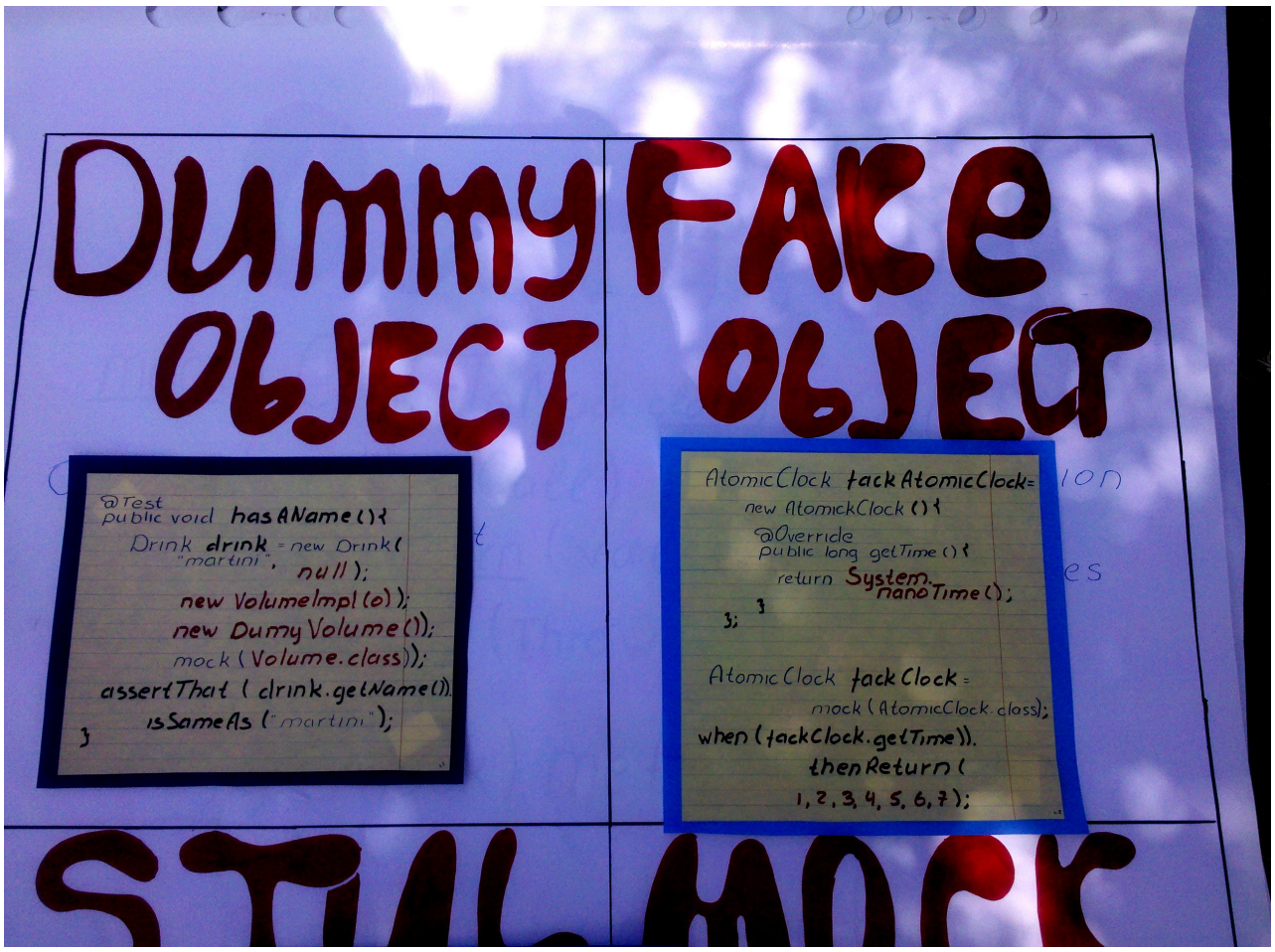
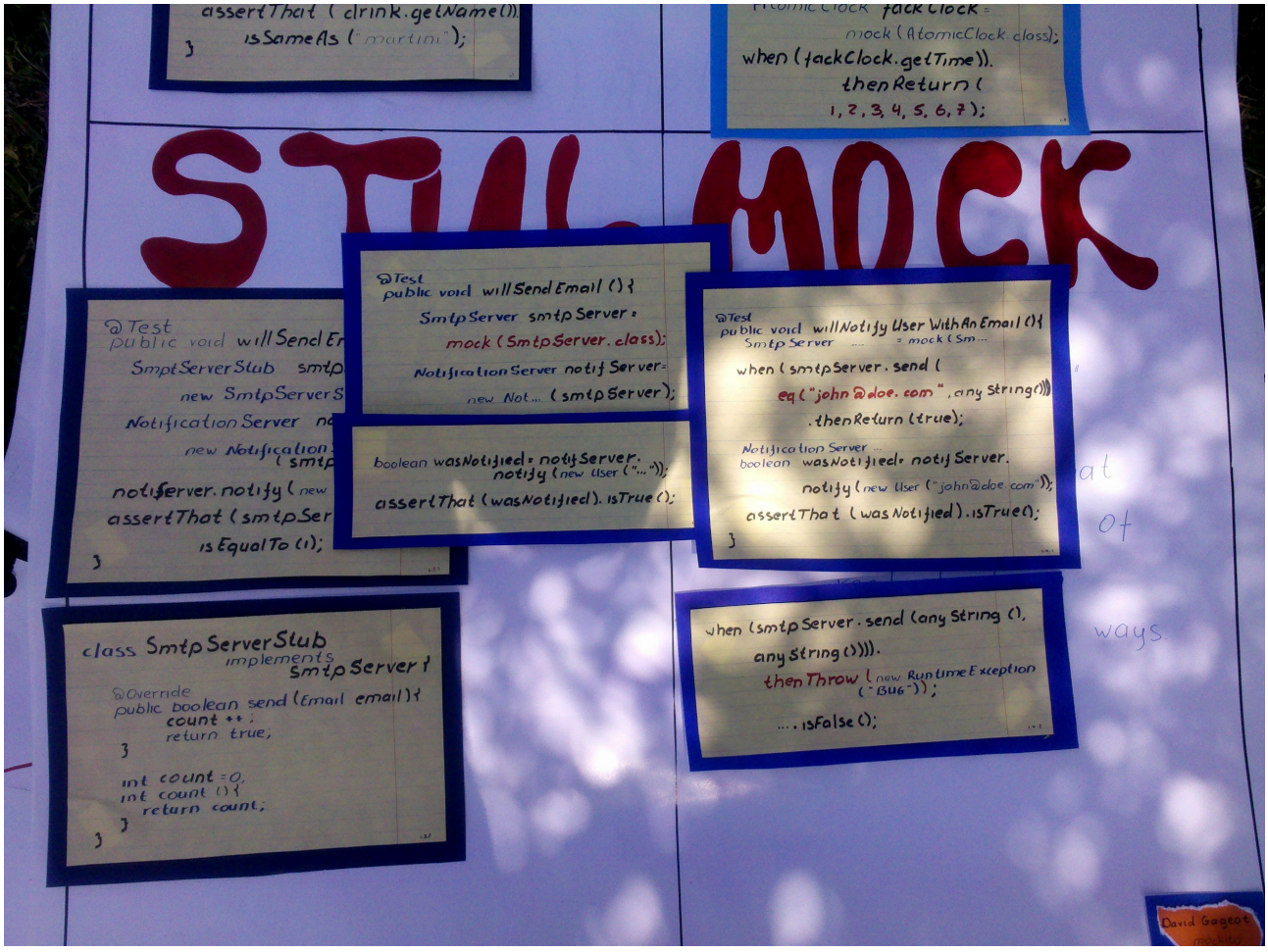
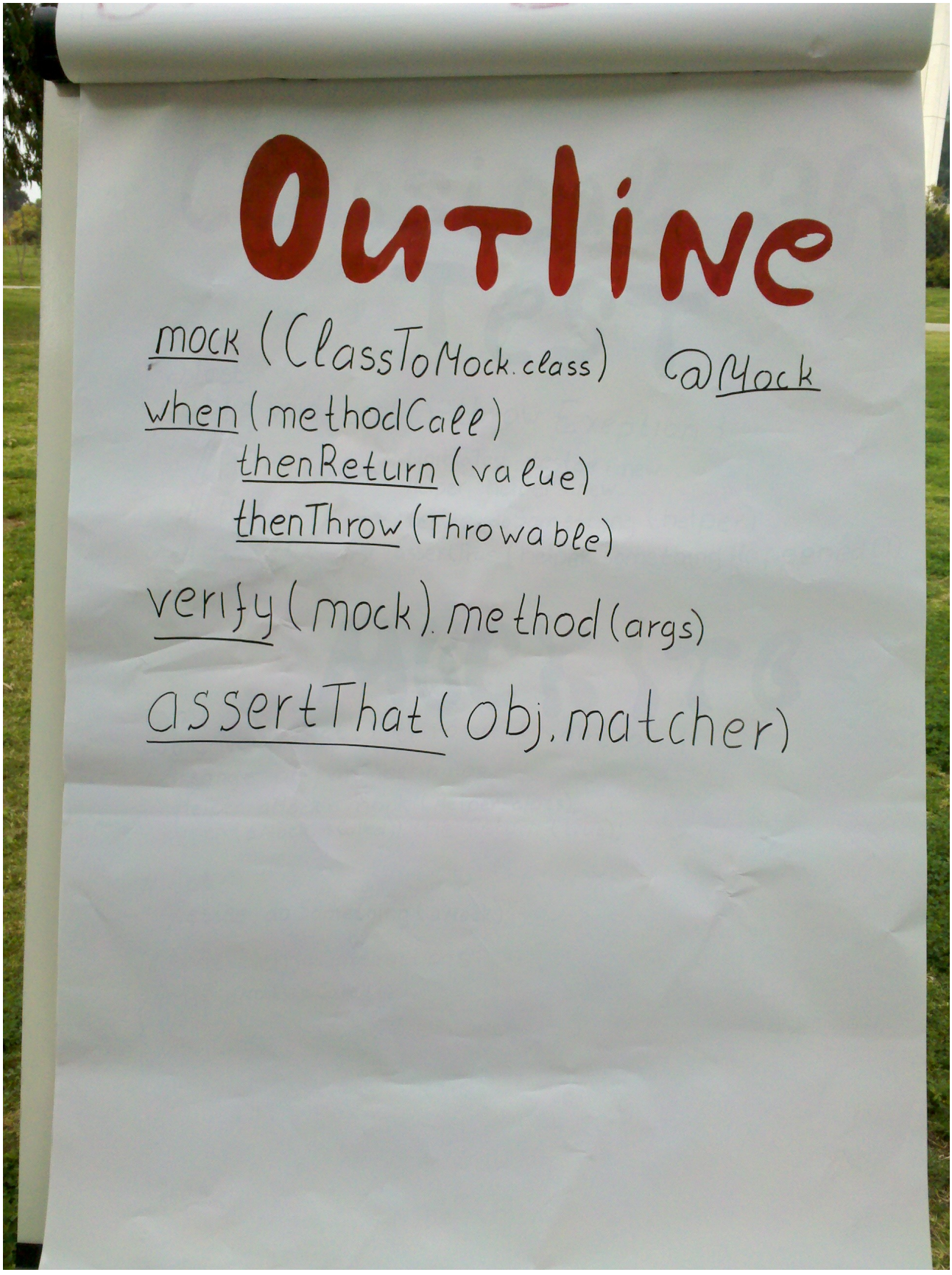
MOCKUP


DUMMY OBJECT
Object passed as an argument
FAKE OBJECT
·simple implementation
·return pre-arranged responses
STUB
·pre-programmed answers
MOCK
"collaboration agreement"
Simulated objects that mimic the behavior of real objects in controlled ways


DUMMY OBJECT
FAKE OBJECT
@Test
public void hasAName(){
Drink drink = new Drink(
"martini", null);
new VolumeImpl(0));
new DummyVolume());
mock(Volume.class));
assertThat(drink.getName())
isSameAs("martini");
}
AtomicClock fackAtomicClock=
new AtomickClock(){
@Override
public long getTime(){
return System.
nanoTime();
}
};
AtomicClock fackClock =
mock(AtomicClock.class);
when(fackClock.getTime()).
thenReturn(
1,2,3,4,5,6,7);
STUB MOCK


assertThat(drink.getName())
isSameAs("martini");
}
mock(AtomicClock.class);
when(fackClock.getTime()).
thenReturn(
1,2,3,4,5,6,7);
STUB MOCK
@Test
public void willSendEmail(){
SmtpServer smtpServer =
mock(SmtpServer.class);
NotificationServer notifServer=
new Not... (smtpServer);
boolean wasNotified = notifServer.
notify(new User("..."));
assertThat(wasNotified).isTrue();
@Test
public void willNotifyUserWithAnEmail(){
SmtpServer ... = mock(Sm...
when(smtpServer.send(
eq("john@doe.com"), anyString()))
.thenReturn(true);
NotificationServer ...
boolean wasNotified= notifServer.
notify(new User("john@doe.com"));
assertThat(wasNotified).isTrue();
}
class SmtpServerStub
implements
SmtpServer{
@Override
public boolean send(Email email){
count++;
return true;
}
int count=0;
int count(){
return count;
}
}
when(smtpServer.send(anyString(),
anyString())).
thenThrow(new RuntimeException
("BUG"));
....isFalse();
David Gageot


Outline
mock(ClassToMock.class) @Mock
when(methodCall)
thenReturn(value)
thenThrow(Throwable)
verify(mock).method(args)
assertThat(obj, matcher)

Classical 3A Test

```
@Test
public void test() throw Exeption {
    //Arrange → UnitToTest testee = new ...
                Helper helper = new ...
    // Act    → testee.doSomething(helper)
    // Assert → assertTrue(helper.somethingHappened())
}
```

MOCKITO

```
@Test
public void test() throw Exeption {
// Arrange, prepare behavior
    Helper aMock = mock(Helper.class);
    when(aMock.isCalled()).thenReturn(true);

// Act
    testee.doSomething(aMock);

// Assert - verify interactions
  verify(aMock).isCalled();
}
```

API

```
import static org.mockito.Mockito.*;

//mock creation
List mockedList = mock(List.class);

//using
mockedList.add("one")
  -"-      .clear();

//verification
verify(mockedList).add("one");
  -"-              .clear();
  -"-              .get(anyInt());
```

```
//Stubbing
when(mockedList.get(0)).thenReturn("first");
```

thenThrow(new RuntimeException())

```
//Argument matchers
when(mockedList.get(anyInt())).thenReturn
                              ("element");
```

mockedList.contains(argThat(isValid()))

```
//Verifying exact number of invocations
mockedList.add("once");
  -"-      .add("twice");
  -"-      .add("twice");

verify(mockedList, times(1)).add("once");
       -"-       ,times(2)).add("twice");
```

=(mockedList).add("once")

never()
atLeastOnce()
atLeast(2)
atMost(5)

```
//Verification in order
List firstMock = mock(List.class);
List secondMock = mock(List.class);

//using
firstMock.add("was called first");
secondMock.add("was called second");

InOrder inOrder = inOrder(firstMock, secondMock);

inOrder.verify(firstMock).add("was called first");
inOrder.verify(secondMock).add("was called second");
```

```
OutputStream mock = mock(OutputStream.class);
OutputStreamWriter osw = new OutputStreamWriter(mock);
osw.write('a');
osw.flush();

BaseMatcher arrayStartingWithA = new BaseMatcher() {
  @Override
  public void describeTo(...)...
  @Override
  public boolean matches(Object item) {
    byte[] actual = (byte[]) items;
    return actual[0] == 'a';
  }
};

verify(mock).write(arg(arrayStartingWithA),
    eq(0), eq(1));
}
```

```
Iterator i = mock(Iterator.class);
when(i.next()).thenReturn("Hello").thenReturn
String result = i.next() + " " + i.next();  ("word");
assertEquals("Hello World", result);

---

Comparable c = mock(Comparable.class);
when(c.compareTo(anyInt())).thenReturn(-1);
assertEquals(-1, c.compareTo(5));

---

@Test(expected = IOException.class)
public void OutputStreamWriter() throws IOException {
  OutputStream mock = mock(OutputStream.class);
  OutputStreamWriter osw = new OutputStreamWriter(mock);
  doThrow(new IOException()).when(mock).close();
  osw.close();
}
```

Limitations

- Needs java 1.5+
- Cannot mock final classes
- Cannot mock static methods
- Cannot mock final methods
- Cannot mock private methods
- Cannot mock equals(), hashCode()
- Cannot mock toString() (can stub it)
- Spying on real methods where real implementation references outer class via OuterClass.this is impossible.

Mockito's spy() method and Spring

```
public class LegacyHelper {
    // Various attributes and functions
    ...

    // One big method that uses external resources (very bad for Unit Testing)
    public int callUrl() {...}
}

public class MyTest {
    // This I don't want to test but my class uses it
    private LegacyHelper helper;

    @BeforeMethod
    public void setUp() {
        helper = spy(new LegacyHelper());
        when(helper.callUrl()).thenReturn(0);
    }

    @Test
    public void testCall() {
        // Now I can use helper without it really calling anything
        helper.callUrl();
        ...
    }
}
```

```
public class SpyFactoryBean {
    // Real or spied object
    private Object real;

    public void setReal(Object obj)..

    public boolean isSingelton()
        { return false; }

    public Class getObjectType(){
        return real.getClass(); }

    public Object getObject(){
        return Mock.spy(real);
}}
```

context file:

```
<?xml version="1.0" .... ?>
<beans>
  <bean id="legacyHelper" class="LegacyHelper" />
  <bean id="mockHelper" class="SpyFactoryBean"
        dependency-check="objects">
        <property name="real" ref="legacyHelper"/>
  </bean>
</beans>
```

Nicolas Frankel

Mockito with Spring MVC Ajax Iteraction

```
@RequestMapping( value = "answer/new", method = RequestMethod.POST)
public ResponseEntity<String> newAnswer ( @RequestParam (
        value = "answerSeverity", required: true) ...
{
    Severity sev = Severity.valueOf (requiresReason);
    SurveyAnswer answer = ...
    answer.setRequiresReason (requiresReason);
    if(requiresReason)
    {
        ....
        answer.addReason (reason);
    }
    answer = surveyService.persist (answer);
    this.getAnswer(sev).add (answer);
    return createJsonResponse (answer);
}

private ResponseEntity<String> createJsonResponse (Object o)
    { ... }

when ( surveyService.persist (any ( SurveyAnswer.class))). thenAnswer (
        new Answer <SurveyAnswer>()
        {
            @Override
            public SurveyAnswer answer (InvocationOnMock inv) throws
                                                        Throwable
            {
                object[] args = inv.getArguments();
                return (SurveyAnswer) args[0];
            }
        });
when ( surveyService.findReasonsByKey (anyCollectionOf (Long.class))).
                thenReturn (getReasons());

@Test
public void testNewAnswerWithReasons() {
    ResponseEntity<String> response = controller.newAnswer (answerSeverity.name(), ....);
    assertEquals ("application/json", response.getHeaders().get ("Content-Type").get(0));
    ...
}
```