Lectures

- Lecture 1 (10.02) – Introduction to Software Testing
- Lecture 2 (17.02) – Basic Black-Box Testing Techniques: Boundary Value Analysis & Equivalence Class Partitioning
- Lecture 3 (03.03) – BBT advanced: Combinatorial Testing
- Lecture 4 (10.03) – Basic White-Box Testing Techniques: Control-Flow Coverage
- Lecture 5 (17.03) – BBT adv.: State-Transition, Metamorphic, Random Testing
- Lecture 6 (24.03) – Test Levels, Test Tools, Test Automation
- Lecture 7 (31.03) – BBT adv.: Exploratory Testing, Behaviour Testing
- Lecture 9 (14.04) – Security Testing of Mobile Applications
- Lecture 10 (21.04) – WBT adv.: Data-Flow Testing / Mutation Testing
- Lecture 12 (05.05) – Defect Estimation / Test Documentation, Organisation and Process Improvement (Test Maturity Model)
- Lecture 13 (12.05) – Exam Preparation
- Lecture 14 (19.05) – Advanced Topics (optional)
Structure of Lecture 6

• Test Levels
• Test Tools
• Test Automation
• Lab 6
STLC = Software Test Life Cycle

SDLC = Software Development Life Cycle

STLC integrated with SDLC
<table>
<thead>
<tr>
<th>Level</th>
<th>Definition and Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance Testing (AT)</td>
<td>The level of the software testing process where a system is tested for acceptability. <strong>The purpose of AT is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.</strong></td>
</tr>
<tr>
<td>System Testing (ST)</td>
<td>The level of the software testing process where a complete, integrated system/software is tested. <strong>The purpose of ST is to evaluate the system’s compliance with the specified requirements.</strong></td>
</tr>
<tr>
<td>Integration Testing (IT)</td>
<td>The level of the software testing process where individual units are combined and tested as a group. <strong>The purpose of IT is to expose faults in the interaction between integrated units.</strong></td>
</tr>
<tr>
<td>Unit Testing (UT)</td>
<td>The level of the software testing process where individual units/components of a software/system are tested. <strong>The purpose of UT is to verify that each unit of the software performs as designed.</strong></td>
</tr>
</tbody>
</table>
## Testing Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Who and How?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance Testing (AT)</td>
<td><strong>Who:</strong> Product Management, Sales, Customer Support, Customers</td>
</tr>
<tr>
<td></td>
<td>How: Usually, Black Box Testing method is used; often the testing is done ad-hoc and non-scripted</td>
</tr>
<tr>
<td>System Testing (ST)</td>
<td><strong>Who:</strong> Normally, independent Testers perform System Testing</td>
</tr>
<tr>
<td></td>
<td>How: Usually, Black Box Testing method is used</td>
</tr>
<tr>
<td>Integration Testing (IT)</td>
<td><strong>Who:</strong> Either Developers themselves or independent Testers</td>
</tr>
<tr>
<td></td>
<td>How:</td>
</tr>
<tr>
<td></td>
<td>- Any of Black Box, White Box, and Gray Box Testing methods can be used</td>
</tr>
<tr>
<td></td>
<td>- Test drivers and test stubs are used to assist in Integration Testing.</td>
</tr>
<tr>
<td>Unit Testing (UT)</td>
<td><strong>Who:</strong> Developers</td>
</tr>
<tr>
<td></td>
<td>How:</td>
</tr>
<tr>
<td></td>
<td>- White-Box Testing Method</td>
</tr>
<tr>
<td></td>
<td>- UT frameworks (e.g., jUnit), drivers, stubs, and mock/fake objects are used</td>
</tr>
</tbody>
</table>
Test Levels

Unit/Component Level

Component code → Unit Test Component 1
Component code → Unit Test Component 2
Component code → Unit Test Component 3
Component code → ... → Unit Test Component n

Integrated Test

Design specs

System Level

Syst funct. requirements
Other software requirements
Customer requirements specs
User Environment:

Function Test → Performance Test → Acceptance Test → Installation Test

System in USE
Testing OO Code

Unit (intra-class) tests

Integration (inter-class) tests

System tests

Acceptance (Validation) tests
Intra-Class Testing Example (Lecture 5)

State Machine Testing:

- The state of an object is modified by operations (methods)
- Methods can be modeled as state transitions

State machine model can be derived from specification (BBT), code (WBT), or both

- Test cases are sequences of method calls that traverse the state machine model
- In addition: Cover the CFG
Testing in different Process Types

**Waterfall model**

- Programmers
- Testers

**Agile model(s)**

- Tester
- Customer
- Programmer

Idea: Testing in collaboration
Agile development (e.g., SCRUM): Time is fixed, scope may change

20 days to complete iteration or sprint
60 days to complete release 1
60 days to complete release 2
120 days for whole project

NB: Often a sprint is only 1 week (5 days)
Challenges of Testing in Agile Development

• Requirements change all the time
• Specification documents are never final
• Code is never ‘finished’, never ‘ready for testing’
• Limited time to test
• Need for regression testing in each increment
  • Developers always break things
  • How can we trust that the code is not broken?
Approaches to Testing in Agile Development

- Automated regression testing
  - Automated unit testing
  - Test-driven development
  - Daily builds and automated tests

- Stabilisation phase or increment
  - Feature freeze
  - Testing and debugging at the end of the increment or release

- Separate system testing
  - Independent testing
  - Separate testers or test team
A Combined Testing Approach
Test Levels

Unit/Component Level

- Component code → Unit Test Component 1
- Component code → Unit Test Component 2
- Component code → Unit Test Component 3
- Component code → Unit Test Component n

Integrated Test

Design specs

System Level

- Syst funct. requirements
- Other software requirements
- Customer requirements specs

- Function Test
- Performance Test
- Acceptance Test
- Installation Test

System in USE
Test Scaffolding

Driver

Classes to be tested

Stubs

Tool example: JUnit

Tool example: JMockIt

(c) 2008 Mauro Pezzè & Michal Young
Unit Testing

Driver

Classes to be tested

Stubs

Tool example: JUnit

Tool example: JMockIt

(c) 2008 Mauro Pezzè & Michal Young
import org.junit.*;
import static org.junit.Assert.*;
import java.util.*;

public class JunitTest1 {
    private Collection collection;

    @BeforeClass
    public static void oneTimeSetUp() {
        // one-time initialization code
        System.out.println("@BeforeClass - oneTimeSetUp");
    }

    @AfterClass
    public static void oneTimeTearDown() {
        // one-time cleanup code
        System.out.println("@AfterClass - oneTimeTearDown");
    }

    @Before
    public void setUp() {
        collection = new ArrayList();
        System.out.println("@Before - setUp");
    }

    @After
    public void tearDown() {
        collection.clear();
        System.out.println("@After - tearDown");
    }

    @Test
    public void testEmptyCollection() {
        assertTrue(collection.isEmpty());
        System.out.println("@Test - testEmptyCollection");
    }

    @Test
    public void testOneItemCollection() {
        collection.add("itemA");
        assertEquals(1, collection.size());
        System.out.println("@Test - testOneItemCollection");
    }
}
import org.junit.*;
import static org.junit.Assert.*;
import java.util.*;

public class JunitTest1 {
    private Collection collection;

    @BeforeClass
    public static void oneTimeSetUp() {
        // one-time initialization code
        System.out.println("@BeforeClass - oneTimeSetUp");
    }

    @AfterClass
    public static void oneTimeTearDown() {
        // one-time cleanup code
        System.out.println("@AfterClass - oneTimeTearDown");
    }

    @Before
    public void setUp() {
        collection = new ArrayList();
        System.out.println("@Before - setUp");
    }

    @After
    public void tearDown() {
        collection.clear();
        System.out.println("@After - tearDown");
    }

    @Test
    public void testEmptyCollection() {
        assertTrue(collection.isEmpty());
        System.out.println("@Test - testEmptyCollection");
    }

    @Test
    public void testOneItemCollection() {
        collection.add("itemA");
        assertEquals(1, collection.size());
        System.out.println("@Test - testOneItemCollection");
    }
}
From JUnit4 to JUnit5 – JDK Versions

- JUnit4 requires Java 5 or higher
- JUnit5 requires Java 8 or higher
# From JUnit4 to JUnit5 – Annotations

<table>
<thead>
<tr>
<th>FEATURE</th>
<th>JUNIT 4</th>
<th>JUNIT 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare a test method</td>
<td>@Test</td>
<td>@Test</td>
</tr>
<tr>
<td>Execute before all test methods in the current class</td>
<td>@BeforeClass</td>
<td>@BeforeAll</td>
</tr>
<tr>
<td>Execute after all test methods in the current class</td>
<td>@AfterClass</td>
<td>@AfterAll</td>
</tr>
<tr>
<td>Execute before each test method</td>
<td>@Before</td>
<td>@BeforeEach</td>
</tr>
<tr>
<td>Execute after each test method</td>
<td>@After</td>
<td>@Each</td>
</tr>
<tr>
<td>Disable a test method / class</td>
<td>@Ignore</td>
<td>@Disabled</td>
</tr>
<tr>
<td>Test factory for dynamic tests</td>
<td>NA</td>
<td>@TestFactory</td>
</tr>
<tr>
<td>Nested tests</td>
<td>NA</td>
<td>@Nested</td>
</tr>
<tr>
<td>Tagging and filtering</td>
<td>@Category</td>
<td>@Tag</td>
</tr>
<tr>
<td>Register custom extensions</td>
<td>NA</td>
<td>@ExtendWith</td>
</tr>
</tbody>
</table>
class CalculatorTest {
    Calculator calc;
    @BeforeEach
    void init() {
        System.out.println("inside @BeforeEach");
        calc = new Calculator();
    }
    @Test
    void additionTest() {
        System.out.println("inside additionTest");
        assertAll(
            () -> assertEquals(2, calc.add(1,1), "Doesn't add two positive numbers properly"),
            () -> assertEquals(0, calc.add(-1,1), "Doesn't add a negative and a positive number properly"),
            () -> assertNotNull(calc, "The calc variable should be initialized")
        );
    }
    @Test
    void divisionTest() {
        System.out.println("inside divisionTest");
        assertThrows(ArithmeticException.class, () -> calc.divide(2,0));
    }
    @AfterEach
    void afterEach() {
        System.out.println("inside @AfterEach");
    }
    @AfterAll
    static void close() {
        System.out.println("inside @AfterAll");
    }
}

JUnit5 – Test Life-Cycle

Console Output:
inside @BeforeEach
inside divisionTest
inside @AfterEach
inside @AfterEach
inside @AfterEach
inside @AfterAll
class CalculatorTest {
    Calculator calc;

    @BeforeAll
    static void start() {
        System.out.println("inside @BeforeAll");
    }

    @BeforeEach
    void init() {
        System.out.println("inside @BeforeEach");
        calc = new Calculator();
    }

    @Test
    void additionTest() {
        System.out.println("inside additionTest");
        assertAll(
            () -> assertEquals(2, calc.add(1,1), "Doesn't add two positive numbers properly"),
            () -> assertEquals(0, calc.add(-1,1), "Doesn't add a negative and a positive number properly"),
            () -> assertNotEquals(calc, "The calc variable should be initialized")
        );
    }

    @Test
    void divisionTest() {
        System.out.println("inside divisionTest");
        assertThrows(ArithmeticException.class, () -> calc.divide(2,0));
    }

    @AfterEach
    void afterEach() {
        System.out.println("inside @AfterEach");
    }

    @AfterAll
    static void close() {
        System.out.println("inside @AfterAll");
    }
}

OBS: Despite the additionTest() method being declared first, it is not guaranteed that it will be executed first.
class CalculatorTest {
    Calculator calc;
    @BeforeAll
    static void start() {
        System.out.println("inside @BeforeAll");
    }
    @BeforeEach
    void init() {
        System.out.println("inside @BeforeEach");
        calc = new Calculator();
    }
    @Test
    void additionTest() {
        System.out.println("inside additionTest");
        assertAll(
            () -> assertEquals(2, calc.add(1,1), "Doesn't add two positive numbers properly"),
            () -> assertEquals(0, calc.add(-1,1), "Doesn't add a negative and a positive number properly"),
            () -> assertNotNull(calc, "The calc variable should be initialized")
        );
    }
    @Test
    void divisionTest() {
        System.out.println("inside divisionTest");
        assertThrows(ArithmeticException.class, () -> calc.divide(2,0));
    }
    @AfterEach
    void afterEach() {
        System.out.println("inside @AfterEach");
    }
    @AfterAll
    static void close() {
        System.out.println("inside @AfterAll");
    }
}

JUnit5 –
Test Life-Cycle

Must have void return type
Must not be static or private

Console Output:
inside @BeforeAll
inside @BeforeEach
inside divisionTest
inside @AfterEach
inside @BeforeEach
inside additionTest
inside @AfterEach
inside @AfterAll
class CalculatorTest {
    Calculator calc;

    @BeforeAll
    static void start() {
        System.out.println("inside @BeforeAll");
    }

    @BeforeEach
    void init() {
        System.out.println("inside @BeforeEach");
        calc = new Calculator();
    }

    @Test
    void additionTest() {
        System.out.println("inside additionTest");
        assertAll(
            () -> assertEquals(2, calc.add(1,1), "Doesn't add two positive numbers properly"),
            () -> assertEquals(0, calc.add(-1,1), "Doesn't add a negative and a positive number properly"),
            () -> assertNotNull(calc, "The calc variable should be initialized")
        );
    }

    @Test
    void divisionTest() {
        System.out.println("inside divisionTest");
        assertThrows(ArithmeticException.class, () -> calc.divide(2,0));
    }

    @AfterEach
    void afterEach() {
        System.out.println("inside @AfterEach");
    }

    @AfterAll
    static void close() {
        System.out.println("inside @AfterAll");
    }
}

Console Output:
inside @BeforeAll
inside @BeforeEach
inside divisionTest
inside @AfterEach
inside @BeforeEach
inside additionTest
inside @AfterEach
inside @AfterAll
JUnit5

Many examples:
- [https://howtodoinjava.com/junit-5-tutorial/](https://howtodoinjava.com/junit-5-tutorial/)

→ How to use Assertions
→ How to test Exceptions
→ …
Integration Testing

- More than one (tested) unit
- Detecting defects
  - On the interfaces of units
  - Communication between units
- Helps assembling incrementally a whole system
- Non-functional aspects if possible
- Done by developers/designers or independent testers
  - Preferably developers and testers in collaboration
- Often omitted due to setup difficulties
  - Time is more efficiently spent on unit and system tests
Integration Testing – Procedural

Note: This has nothing to do with class hierarchy in OO (inheritence)
What if Class 4 not yet ready or not accessible?
Integration Testing – OO

What if Class 4 not yet ready or not accessible?

What if an external service not yet ready or accessible (or should not be accessed – because we are just testing)?
Integration Testing

Driver

Classes to be tested

Stubs

Tool example: JUnit

Tool example: JMockIt

(c) 2008 Mauro Pezzè & Michal Young
Mockito

Example:
Class RecordService:
stores a given file in a database and a network location with the help of classes DatabaseDao and NetworkDao.

Method `saveFile()` invokes `save()` methods defined in Database Dao and NetworkDao.
Mockito

RecordService

DatabaseDao database

NetworkDao network

saveFile()

...
### Mockito

<table>
<thead>
<tr>
<th>RecordService</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatabaseDao database</td>
</tr>
<tr>
<td>NetworkDao network</td>
</tr>
<tr>
<td>saveFile()</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

### Why mock?

When unit testing methods like `saveFile()`, if we only used JUnit, our tests will depend on the `safe()` methods in other classes.

We want the unit tests to be independent of all other dependencies.
Mockito

RecordService.java

```java
public class RecordService {
    DatabaseDAO database;
    NetworkDAO network;

    //setters and getters

    public boolean saveFile(String fileName) {
        database.save(fileName);
        System.out.println("Saved in database in Main class");
        network.save(fileName);
        System.out.println("Saved in network in Main class");
        return true;
    }
}
```
In the test environment, it is neither possible to access the database nor the network location. We create mocks for both repositories.
Mockito

To test the `RecordService` class, we create a unit test.

```java
import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.MockitoJUnitRunner;
import com.howtodoinjava.demo.mockito.DatabaseDAO;
import com.howtodoinjava.demo.mockito.NetworkDAO;
import com.howtodoinjava.demo.mockito.RecordService;

@RunWith(MockitoJUnitRunner.class)
public class ApplicationTest {

    @InjectMocks
    RecordService recordService;

    @Mock
    DatabaseDAO databaseMock;

    @Mock
    NetworkDAO networkMock;

    @Test
    public void saveFileTest() {
        boolean saved = recordService.saveFile("temp.txt");
        assertEquals(true, saved);

        verify(databaseMock, times(1)).save("temp.txt");
        verify(networkMock, times(1)).save("temp.txt");
    }
}
```
public class ApplicationTest {

    @RunWith(MockitoJUnitRunner.class)
    public class ApplicationTest {

        @InjectMocks
        RecordService recordService;

        @Mock
        DatabaseDAO databaseMock;

        @Mock
        NetworkDAO networkMock;

        @Test
        public void saveFileTest()
        {
            boolean saved = recordService.saveFile("temp.txt");
            assertEquals(true, saved);
            verify(databaseMock, times(1)).save("temp.txt");
            verify(networkMock, times(1)).save("temp.txt");
        }
    }
}
ApplicationTest.java

```java
...

@RunWith(MockitoJUnitRunner.class)
public class ApplicationTest {

    @InjectMocks
    RecordService recordService;

    @Mock
    DatabaseDao databaseMock;

    @Mock
    NetworkDao networkMock;

    @Test
    public void saveFileTest() {
        boolean saved = recordService.saveFile("temp.txt");
        assertEquals(true, saved);
        verify(databaseMock, times(1)).save("temp.txt");
        verify(networkMock, times(1)).save("temp.txt");
    }
}
```

Annotate the test with the `@RunWith(MockitoJUnitRunner.class)` so that mockito can process the annotations.
@RunWith(MockitoJUnitRunner.class)
public class ApplicationTest {

    @InjectMocks
    RecordService recordService;

    @Mock
    DatabaseDao databaseMock;

    @Mock
    NetworkDao networkMock;

    @Test
    public void saveFileTest() {
        boolean saved = recordService.saveFile("temp.txt");
        assertEquals(true, saved);

        verify(databaseMock, times(1)).save("temp.txt");
        verify(networkMock, times(1)).save("temp.txt");
    }
}
`ApplicationTest.java`

```java
...
@RunWith(MockitoJUnitRunner.class)
public class ApplicationTest {
    @InjectMocks
    RecordService recordService;

    @Mock
    DatabaseDao databaseMock;

    @Mock
    NetworkDao networkMock;

    @Test
    public void saveFileTest() {
        boolean saved = recordService.saveFile("temp.txt");
        assertEquals(true, saved);

        verify(databaseMock, times(1)).save("temp.txt");
        verify(networkMock, times(1)).save("temp.txt");
    }
}
```

Annotate the dao fields with the `@Mock` annotation to have a mock object instantiated for each of them.
Call the method to be tested, i.e., `saveFile()`, on the class to be tested, i.e., `recordService`. 

```java
@RunWith(MockitoJUnitRunner.class)
public class ApplicationTest
{
    @InjectMocks
    RecordService recordService;
    
    @Mock
    DatabaseDao databaseMock;
    
    @Mock
    NetworkDao networkMock;
    
    @Test
    public void saveFileTest()
    {
        boolean saved = recordService.saveFile("temp.txt");
        assertEquals(true, saved);
        verify(databaseMock, times(1)).save("temp.txt");
        verify(networkMock, times(1)).save("temp.txt");
    }
}
```
ApplicationTest.java

...  

@RunWith(MockitoJUnitRunner.class)
public class ApplicationTest {
  
    @InjectMocks
    RecordService recordService;

    @Mock
    DatabaseDao databaseMock;

    @Mock
    NetworkDao networkMock;

    @Test
    public void saveFileTest() {
        boolean saved = recordService.saveFile("temp.txt");
        assertEquals(true, saved);

        verify(databaseMock, times(1)).save("temp.txt");
        verify(networkMock, times(1)).save("temp.txt");
    }
}
Mockito – verify()

Simple behavior verification ➔ verify(); using List class as an example

//Let's import Mockito statically so that the code looks clearer
import static org.mockito.Mockito.*;

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

//using mock object
mockedList.add("one");
mockedList.clear();

//verification
verify(mockedList).add("one");
verify(mockedList).clear();

Once created, a mock will remember all interactions. Then you can selectively verify whatever interactions you are interested in.

Interface java.util.List
Mockito – when … then

Simple stubbing ➔ “when() … then()”

//You can mock concrete classes, not just interfaces (see previous slide)
LinkedList mockedList = mock(LinkedList.class);

//stubbing: when – then (like if – else)
when(mockedList.get(0)).thenReturn("first");
when(mockedList.get(1)).thenThrow(new RuntimeException());

//following prints "first"
System.out.println(mockedList.get(0));

//following throws runtime exception
System.out.println(mockedList.get(1));

//following prints "null" because get(999) was not stubbed
System.out.println(mockedList.get(999));

By default, for all methods that return a value, a mock will return either “null”, a primitive value, or an empty collection, as appropriate. For example: “0” for an int and “false” for a boolean.
With stubbing, specific return values can be defined.
Mockito – when … then

Argument matchers …

//stubbing using built-in anyInt() argument matcher
when(mockedList.get(anyInt())).thenReturn("element");

//stubbing using custom matcher (let's say isValid() returns your own matcher implementation):
when(mockedList.contains(argThat(isValid()))).thenReturn("element");

//following prints "element"
System.out.println(mockedList.get(999));

//you can also verify using an argument matcher
verify(mockedList).get(anyInt());

Mockito verifies argument values in natural java style: by using an equals() method. Sometimes, when extra flexibility is required then you might use argument matchers; with verify() you check whether a method has been used.

Compare with previous slide!
Mockito

Annotations: https://howtodoinjava.com/mockito/mockito-annotations/

- **@Mock** is used for mock creation. It makes the test class more readable; alternative: Mockito.mock(classToMock).
- **@Spy** is used to create a spy instance; similar to @Mock but creates a real instance of the mocked class and tracks every interaction with it. Can be used instead of spy(Object) method.
- **@InjectMocks** is used to instantiate the tested object automatically and inject all the @Mock or @Spy annotated field dependencies into it (if applicable).
- **@Captor** is used to create an argument captor

To process annotations, “MockitoAnnotations.initMocks(testClass);” must be used at least once. Alternatively, one can use the built-in runner MockitoJUnitRunner or rule MockitoRule (then use @Rule annotation).
Mockito – Annotations: @Mock

The @Mock annotation is used to create and inject mocked instances. We do not create real objects, rather ask mockito to create a mock for the class.

The @Mock annotation is alternative to Mockito.mock(classToMock). They both achieve the same result. Using @Mock is usually considered “cleaner“, as we don’t fill up the tests with boilerplate assignments that all look the same.

Using the @Mock annotation
• allows shorthand creation of objects required for testing.
• minimizes repetitive mock creation code.
• makes the test class more readable.
• makes the verification error easier to read because field name is used to identify the mock.
Mockito – Annotations: @Mock

@Mock Example:

```java
@Mock
HashMap<String, Integer> mockHashMap;

@Test
public void saveTest()
{
    mockHashMap.put("B", 2);

    Mockito.verify(mockHashMap, times(1)).put("B", 2);
    Mockito.verify(mockHashMap, times(0)).get("B");

    assertEquals(0, mockHashMap.size());
}
```

Here we mock the HashMap class. In real tests, we shall be mocking actual application classes.

We put a key-value pair in the map, then we verify that method invocation was performed on the mocked map instance, then we check its size.
Mockito – Annotations: @Mock

@Mock Example:

```java
@Mock
HashMap<String, Integer> mockHashMap;

@Test
public void saveTest() {
    mockHashMap.put("B", 2);
    Mockito.verify(mockHashMap, times(1)).put("B", 2);
    Mockito.verify(mockHashMap, times(0)).get("B");

    assertEquals(0, mockHashMap.size());
}
```

Here we mock the HashMap class. In real tests, we shall be mocking actual application classes.

We put a key-value pair in the map, then we verify that method invocation was performed on the mocked map instance, then we check its size.

Not equal to 1 – why?

Get() method was never called.
Mockito – Annotations: @Mock

@Mock Example:

```java
@Mock
HashMap<String, Integer> mockHashMap;

@Test
public void saveTest() {
    mockHashMap.put("B", 2);
    Mockito.verify(mockHashMap, times(1)).put("B", 2);
    Mockito.verify(mockHashMap, times(0)).get("B");
    assertEquals(0, mockHashMap.size());
}
```

When using @Mock, Mockito creates a bare-bones shell instance of the Class, entirely instrumented to track interactions with it. This is not a real object and does not maintain the state changes to it.

When using @Spy, Mockito creates a real instance of the class and tracks every interactions with it. It maintains the state changes to it.

➤ Next slide
Mockito – Annotations @Spy

@Spy Example:

```java
@Spy
HashMap<String, Integer> hashMap;

@Test
public void saveTest() {
    hashMap.put("A", 10);
    Mockito.verify(hashMap, times(1)).put("A", 10);
    Mockito.verify(hashMap, times(0)).get("A");

    assertEquals(1, hashMap.size());
    assertEquals(new Integer(10), (Integer) hashMap.get("A"));
}
```

The @Spy annotation is used to create a real object and spy on that real object. A spy helps to call all the normal methods of the object while still tracking every interaction, just as we would with a mock.

In the example, the size of map now equals 1 because we added one key-value pair to it. We are also able to get back the value added to map using it’s key (not possible in mocked instances).
Mockito – Annotations @Captor

@Captor Example:

```java
@Mock
HashMap<String, Integer> hashMap;

@Captor
ArgumentCaptor<String> keyCaptor;

@Captor
ArgumentCaptor<Integer> valueCaptor;

@Test
public void saveTest() {
    hashMap.put("A", 10);
    Mockito.verify(hashMap).put(keyCaptor.capture(), valueCaptor.capture());

    assertEquals("A", keyCaptor.getValue());
    assertEquals(new Integer(10), valueCaptor.getValue());
}
```

The @Captor annotation is used to create an ArgumentCaptor instance which is used to capture method argument values for further assertions.

Note that Mockito verifies argument values using the equals() method of the argument class.
Mockito – How to initialize Annotations?

Option 1: Use @RunWith(MockitoJUnitRunner.class) at the top of unit test class.

```java
@RunWith(MockitoJUnitRunner.class)
public class ExampleTest {

    @Mock
    private List list;

    @Test
    public void shouldDoSomething() {
        list.add(100);
    }
}
```
Mockito – How to initialize Annotations?

**Option 2:** Use MockitoAnnotations.initMocks(this) in @Before method of unit test class.

```java
public class ExampleTest {

    @Mock
    private List list;

    @Before
    public void setup() {
        MockitoAnnotations.initMocks(this);
    }

    @Test
    public void shouldDoSomething() {
        list.add(100);
    }
}
```
Mockito – How to initialize Annotations?

Option 3: Use MockitoJUnit.rule() to create MockitoRule class.

```java
public class ExampleTest {

    @Rule
    public MockitoRule rule = MockitoJUnit.rule();

    @Mock
    private List list;

    @Test
    public void shouldDoSomething() {
        list.add(100);
    }
}
```
Mockito

More possibilities can be found here:

https://static.javadoc.io/org.mockito/mockito-core/2.25.1/org/mockito/Mockito.html#1

For web applications:
use MockWebServer to mock the behavior of another web server
Test Levels

Unit/Component Level

Component code → Unit Test Component 1
Component code → Unit Test Component 2
Component code → Unit Test Component 3
Component code → Unit Test Component n

Integrated Test

Design specs

System Level

Function Test → Performance Test → Acceptance Test → Installation Test

User Environment:
- Syst funct. requirements
- Other software requirements
- Customer requirements specs

System in USE

LTAT.05.006 / Lecture 06 / © Dietmar Pfahl 2022
Test Levels

Unit/Component Level

- Component code
- Unit Test Component 1
- Unit Test Component 2
- Unit Test Component 3
- Integrated Test
- Design specs
- Component code
- Unit Test Component n

System Level

Overview of System Testing Types

- Smoke Testing
- Functional Testing
- Usability Testing
- Security Testing
- Performance Testing
- Regression Testing
- Compliance Testing

+ Exploratory Testing
+ GUI Testing
Structure of Lecture 6

- Test Levels
- Test Tools
- Test Automation
- Lab 6
Tools – the Workbench

- Good for
  - repeating tasks
  - organising data

- Requires training
- Must be introduced incrementally
- No “silver bullet”

Evaluation criteria
- Ease of use
- Power
- Robustness
- Functionality
- Ease of introduction
- Quality of support
- Cost
- Company policies and goals
Test Tools – in the Process

- Requirement specification
  - Architectural design
  - Detailed design
  - Test design tools
  - Static analysis tools
  - Test management tools

- Test execution and comparison tools
  - Code

- Performance simulator tools
  - Integration test

- Acceptance test
  - System test

- Dynamic analysis tools
  - Debugging tools

- Coverage tools
  - Unit test
There is no shortage of Test Tools

http://www.testingtools.com

**Agile Testing**
Agile testing comes with many challenges, involves experimenting and trying new ideas. Get started with agile testing tools to help you improve the quality of your Agile, Scrum or XP projects and build better software faster.

**Load Testing**
To ensure that your web application and website performs flawlessly even with a large number of visitors and users, load testing can be used to verify your application's performance and the capacity of your infrastructure.

**Mobile Testing**
There are various tools and online resources to help you build tests for your mobile devices, record and run automated UI and unit tests for mobile apps and code libraries as well as help you test responsive and fluid web interfaces.

**Test Automation**
Automated software testing and related automated testing tools is becoming more and more import for teams in order to automatically verify key functionality, test for regressions and help teams run a large number of tests in a short period of time.

**Test Management**
Software development teams benefit from adapting a test management or test case management tool. Such tools helps software teams manage their testing efforts, record test results and generate reports to help optimize all testing activities and to provide useful feedback to developers and project leads.

**Usability Testing**
Usability testing is used to observe (usually first-time) users of your application or visitors of your website in order to improve the usability and user experience of your projects. This page lists some of the best resources available to help you get started with your own usability tests.
Example: Functional (Web-)Testing

1. Recorded Scripts
2. Engineered Scripts
3. Data-driven Testing
4. Keyword-driven Testing
5. Model-based Testing
Recorded Scripts

- Unstructured
- Scripts generated using capture and replay tools
- Relatively quick to set up
- Mostly used for regression testing
- Scripts non-maintainable, in practice
  - If the system changes they need to be captured again

Capture Replay Tools
- Record user’s actions to a script (keyboard, mouse)
  - Tool specific scripting language
- Scripts access the (user) interface of the software
  - Input fields, buttons and other widgets
- Simple checks can be created in the scripts
  - Existence of texts and objects in the UI
  - Data of GUI objects
Recorded Scripts

• Example with Selenium IDE

Some web-application to be tested ...

http://opensource.demo.orangehrm.com
Recorded Scripts

- Example with Selenium IDE

Make sure Record button is ON!
Open Base URL in browser
Login using values:
  - Login Name: demo
  - Password: demo
Click ‘Login’ button
Recorded Scripts

- Example with Selenium IDE

... next actions ...

Highlight ‘Welcome demo’ text
Verify that text is present
- command: VerifyTextPresent

Click ‘Logout’

... then stop recording ...
Recorded Scripts

- Example with Selenium IDE

Test Case (=Test Scenario) consists of several Test Steps

**Open Base URL in browser**

Login using values:
- Login Name: demo
- Password: demo

Click Login button

Highlight ‘Welcome demo’ text

Verify that text is present

Click ‘Logout’

Tests can be run (replay) ...

Record Button switched off ...

http://opensource.demo.orangehrm.com

Open Base URL in browser

Login using values:
- Login Name: demo
- Password: demo

Click Login button

Highlight ‘Welcome demo’ text

Verify that text is present

Click ‘Logout’

Tests can be run (replay) ...

Record Button switched off ...

Private settings

Selenium commands

Data (values)

Location on Web-page (Target)

may use xpath, css, id, field, etc.

TCs can be saved and exported into several programming languages (java, python, c#, etc.)
Recorded Scripts

Typical problem: Test fails

What went wrong?

Time is needed to validate the login information…
The test must WAIT for the next web page to load before proceeding.
Selenium IDE Commands (Selenese)

- **type**: Sets the value of an input field, as though you typed it in.
- **open**: Opens a page using a URL.
- **click**: Clicks on a link, button, checkbox or radio button.
- **clickAndWait**: Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call `waitForPageToLoad`.
- **select**: Select an option from a drop-down using an option locator.
- **selectFrame**: Selects a frame within the current window.
- **verifyTitle/assertTitle**: Verifies an expected page title.
- **verifyTextPresent**: Verifies that the specified text pattern appears somewhere on the rendered page shown to the user.
- **verifyElementPresent**: Verifies that the specified element is somewhere on the page.
- **waitForPageToLoad**: Waits for a new page to load. You can use this command instead of the “AndWait” suffixes, “clickAndWait”, “selectAndWait”, “typeAndWait” etc.
- **highlight**: Briefly changes the `backgroundColor` of the specified element yellow. Useful for debugging.
- **pause**: Wait for the specified amount of time (in milliseconds)
- **store**: The name of a variable in which the result is to be stored. This command is a synonym for `storeExpression`.
- **echo**: Prints the specified message into the third table cell in your Selenese tables. Useful for debugging.
- **refresh**: Simulates the user clicking the “Refresh” button on their browser.

**Source**: https://www.selenium.dev
Selenium IDE Commands (Selenese)

- **type**: Sets the value of an input field, as though you typed it in.
- **open**: Opens a page using a URL.
- **click**: Clicks on a link, button, checkbox or radio button.
- **clickAndWait**: Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load (like a link usually does), call waitForPageToLoad.
- **select**: Select an option from a dropdown using an option locator.
- **selectFrame**: Selects a frame within the current window.
- **verifyTitle/assertTitle**: Verifies an expected page title.
- **verifyTextPresent**: Verifies that the specified text pattern appears somewhere on the rendered page shown to the user.
- **verifyElementPresent**: Verifies that the specified element is somewhere on the page.
- **waitForPageToLoad**: Waits for a new page to load. You can use this command instead of the "AndWait" suffixes, "clickAndWait", "selectAndWait", "typeAndWait" etc.
- **highlight**: Briefly changes the backgroundColor of the specified element yellow. Useful for debugging.
- **pause**: Wait for the specified amount of time (in milliseconds)
- **store**: The name of a variable in which the result is to be stored. This command is a synonym for storeExpression.
- **echo**: Prints the specified message into the third table cell in your Selenese tables. Useful for debugging.
- **refresh**: Simulates the user clicking the “Refresh” button on their browser

**Selenium Assertions** can be categorized into three categories:

- **Assert**: When an ‘Assert’ fails, the test is aborted.
- **Verify**: When a ‘Verify’ fails, the test will continue execution, logging the failure.
- **WaitFor**: Wait for some condition to become true. They will succeed immediately if the condition is already true. However, they will fail and halt the test if the condition does not become true within the current timeout setting.

Source: https://www.selenium.dev
Engineered Scripts

- Scripts are well-designed (following a systematic approach), modular, robust, documented, and maintainable
- Separation of common tasks
  - E.g. setup, cleanup/teardown, and defect detection
- Test data is still embedded into the scripts
  - One driver script per test case
- Test code is mostly written manually
- Implementation and maintenance require programming skills which testers (test engineers) might not have
- “Just like any other software development project”
Engineered Scripts – Example
Engineered Scripts – Example

Click on ‘Math Calculators’
Engineered Scripts – Example

We got a list ‘Math Calculators’ and ... clicked on ‘Percent Calculator’
Engineered Scripts – Example

We enter ‘10’ and ‘50’
Engineered Scripts – Example

We click ‘Calculate’ and get the result (‘5’).
How do we test this with ‘Engineered Scripts’?

(this will be done in-depth in Lab 6)
Engineered Scripts

• Example with Selenium WebDriver

```java
package TestNG;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium *
import org.openqa.selenium.Firefox.FirefoxDriver;
import org.openqa.selenium.firefox.FirefoxOptions;
import org.openqa.selenium.WebDriver *
import org.openqa.selenium.WebElement *
import org.openqa.selenium.chrome.ChromeDriver *

public class TestNGClass {
    WebDriver driver = new FirefoxDriver();
    
    @BeforeTest
    public void launachapp() {
        // Put an implicit wait, will wait for 10 seconds before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        // Launch website
        driver.navigate().to("http://www.calculator.net");
        // Open the web page
        // Open the calculator
        // Enter value 10 in the first number of the percent calculator
        // Enter value 50 in the second number of the percent calculator
        // Click on the button to calculate the result
        // Get the Result text based on its xpath
        String result = driver.findElement(By.xpath("//[@id='content']//tr//td[2]//input")).getText();
        // Print a log in message to the screen
        System.out.println("The Result is " + result);
        if(result.equals("5")) {
            System.out.println("The Result is Pass");
        } else {
            System.out.println("The Result is Fail");
        }
    }

    @Test
    public void calculatepercent() {
        // Click on Math Calculators
        driver.findElement(By.xpath("//*[id='menu']//div[3]/a")
        .click();
        // Click on Percent Calculators
        driver.findElement(By.xpath("//*[id='menu']//div[4]/div[3]/a")
        .click();
        // Enter value 10 in the first number of the percent calculator
        // Enter value 50 in the second number of the percent calculator
        // Click Calculate Button
        driver.findElement(By.xpath("//[@id='content']//table//tbody//tr//td[2]//input")
        .click();
        // Get the Result text based on its xpath
        String result = driver.findElement(By.xpath("//[@id='content']//p[2]//span//font//b")
        .getText();
        // Print a log in message to the screen
        System.out.println("The Result is " + result);
        if(result.equals("5")) {
            System.out.println("The Result is Pass");
        } else {
            System.out.println("The Result is Fail");
        }
    }

    @AfterTest
    public void terminatetest() {
        driver.close();
    }
}
```
Engineered Scripts

- Example with Selenium WebDriver
  - Click on 'Math Calculator'

```java
package TestNG;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium *
import org.openqa.selenium.Firefox.FirefoxDriver;
import org.openqa.selenium *
import org.openqa.selenium.Firefox.FirefoxDriver;
public class TestNGClass
{
    WebDriver driver = new FirefoxDriver();
    @BeforeTest
    public void launparty()
    {
        // Puts an implicit wait, will wait for 10 seconds before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        // Launch website
        driver.navigate().to("http://www.calculator.net");
        driver.manage().window().maximize();
    }
    @Test
    public void calculatecopper()
    {
        // Click on Math Calculators
        driver.findElement(By.xpath("//*[@id='menu']/*[3]/a")).click();
        // Click on Percent Calculators
        driver.findElement(By.xpath("//*[@id='menu']/*[4]/a")).click();
        // Enter value 10 in the first number of the percent calculator
        driver.findElement(By.id("par1")).sendKeys("10");
        // Enter value 50 in the second number of the percent calculator
        driver.findElement(By.id("par2")).sendKeys("50");
        // Click Calculate Button
        driver.findElement(By.xpath("//*[@id='content']//table//tr//td//input")).click();
        // on its xpath
        driver.findElement(By.xpath("//*[@id='content']//p[@p]//font[@b]")).getText();
        // the screen
        result = " + result;"
        // Result is Pass"
        result = " + result;"
        // Result is Fail"
    }
}
```
Engineered Scripts

- Example with Selenium WebDriver

Click on 'Percent Calculator'

```java
package TestNG;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.Firefox.FirefoxDriver;
import org.openqa.selenium.Firefox.FirefoxDriver;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class TestNGClass {
    WebDriver driver = new FirefoxDriver();

    @BeforeTest
    public void launchapp() {
        // Puts an implicit wait, will wait for 10 seconds before throwing exception
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        // Launch website
        driver.navigate().to("http://www.calculator.net");
        driver.manage().window().maximize();
    }

    @Test
    public void calculatepercent() {
        // Click on Math Calculators
        driver.findElement(By.xpath("//*[id='MathCalculators']/div[6]/a")).click();
        // Click on Percent Calculators
        driver.findElement(By.xpath("//*[id='MathCalculators']/div[7]/a")).click();
        // Enter value 10 in the first number of the percent calculator
        driver.findElement(By.id("cpar1")).sendKeys("10");
        // Enter value 50 in the second number of the percent Calculator
        driver.findElement(By.id("cpar2")).sendKeys("50");
        // Click Calculate Button
        driver.findElement(By.xpath("//*[id='content']//table/tbody/tr/td[2]/input")).click();
        // Get the Result text based on its xpath
        String result = driver.findElement(By.xpath("//*[id='content']//p[2]/span/font/b")).getText();
        // Print a log In message to the screen
        System.out.println("The Result is "+ result);
        if(result.equals("5")) {
            System.out.println("The Result is Pass");
        } else {
            System.out.println("The Result is Fail");
        }
    }

    @AfterTest
    public void terminateTest() {
        driver.close();
    }
}
```
Engineered Scripts

- Example with Selenium WebDriver

Enter ’10’ – first number
Enter ’50’ – second number

Click on ’Calculate’
Get result with getText() – ’5’
Engineered Scripts

- Example with Selenium WebDriver

```java
package TestNG;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.*;
import org.openqa.selenium.Firefox.FirefoxDriver;
import org.openqa.selenium.annotations.AfterTest;
import org.openqa.selenium.annotations.BeforeTest;
import org.openqa.selenium.annotations.Test;

public class TestNGClass {
    WebDriver driver = new FirefoxDriver();

    @BeforeTest
    public void launchapp() {
        // Puts an implicit wait, will wait for 10 seconds before throwing exception
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        // Launch website
        driver.navigate().to("http://www.calculator.net");
        driver.manage().window().maximize();
    }

    @Test
    public void calculatepercent() {
        // Click on Math Calculators
driver.findElement(By.xpath("*[@id='menu']/div[3]/a")).click();
        // Click on Percent Calculators
driver.findElement(By.xpath("*[@id='menu']/div[4]/div[3]/a")).click();
        // Enter value 10 in the first number of the percent calculator
driver.findElement(By.id("cpar1")).sendKeys("10");
        // Enter value 50 in the second number of the percent calculator
driver.findElement(By.id("cpar2")).sendKeys("50");
        // Click Calculate Button
driver.findElement(By.xpath("*[@id='content']/table/tbody/tr/td[2]/input")).click();
        // Get the Result Text based on its xpath
        String result = driver.findElement(By.xpath("*[@id='content']/p[2]/span/font/b")).getText();
        // Print a Log In message to the screen
        if(result.equals("5")) {
            System.out.println("The Result is Pass");
        } else {
            System.out.println("The Result is Fail");
        }
    }

    @AfterTest
    public void terminateTest() {
        driver.close();
    }
}
```

Checks if result equals '5' (10% of 50)
Selenium-WebDriver API Commands and Operations

https://www.seleniumhq.org/docs/03_webdriver.jsp - selenium-webdriver-api-commands-and-operations

- Fetching a Web-page
- Locating UI Elements (WebElements)
- Getting text values
- User Input - Filling In Forms
- Moving Between Windows and Frames
- …
Selenium-WebDriver API Commands and Operations

https://www.seleniumhq.org/docs/03_webdriver.jsp - selenium-webdriver-api-commands-and-operations

- Fetching a Web-page
- Locating UI Elements (WebElements)

Locating elements in WebDriver can be done on the WebDriver instance itself or on a WebElement. Each of the language bindings exposes a `findElement()` and `findElements()` method.

- **findElement()** returns a WebElement object matching the query, and throws an exception if such an element cannot be found.
- **findElements()** returns a list of WebElements, possibly empty if no DOM elements match the query.

Examples of “By” strategies are shown on the next slides...
Selenium-WebDriver API Commands and Operations

Locating UI Elements

By

- Xpath
- ID
- Link Text

HTML snippet in a web-page:

```html
<input type="text" name="example" />

<INPUT type="text" name="example" />
```

1 tag name and 2 attribute names per line

- The “Find” methods take a locator or query object called “By”
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By XPath

HtmlUnit:
Finds 1 arg for ”input” tag
Finds 0 args for ”INPUT” tag

Because:
- lower-cased
- not showing default args (i.e., ”text” arg of ”type” attribute)

### By XPath

At a high level, WebDriver uses a browser’s native XPath capabilities wherever possible. On those browsers that don’t have native XPath support, we have provided our own implementation. This can lead to some unexpected behavior unless you are aware of the differences in the various XPath engines.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Tag and Attribute Name</th>
<th>Attribute Values</th>
<th>Native XPath Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>HtmlUnit Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
<tr>
<td>Internet Explorer Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>No</td>
</tr>
<tr>
<td>Firefox Driver</td>
<td>Case insensitive</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This is a little abstract, so for the following piece of HTML:

```html
<input type="text" name="example" />
<INPUT type="text" name="other" />
```

```java
List< WebElement> inputs = driver.findElements(By.xpath("//input"));
```

The following number of matches will be found:

<table>
<thead>
<tr>
<th>XPath expression</th>
<th>HtmlUnit Driver</th>
<th>Firefox Driver</th>
<th>Internet Explorer Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>/input</td>
<td>1 (&quot;example&quot;)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/INPUT</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Sometimes HTML elements do not need attributes to be explicitly declared because they will default to known values. For example, the ”input” tag does not require the ”type” attribute because it defaults to ”text”. The rule of thumb when using xpath in WebDriver is that you should not expect to be able to match against these implicit attributes.
Selenium-WebDriver API Commands and Operations

Locating UI Elements

By XPath

**HtmlUnit:**
Finds 1 arg for "input" tag
Finds 0 args for "INPUT" tag

Because:
- lower-cased
- not showing default args (i.e., "text" arg of "type" attribute)
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By XPath

InternetExplorer:
Finds 2 args for "input" tag
Finds 0 args for "INPUT" tag

Because:
- lower-cased
- showing all args of all attributes

At a high level, WebDriver uses a browser’s native XPath capabilities wherever possible. On those browsers that don’t have native XPath support, we have provided our own implementation. This can lead to some unexpected behavior unless you are aware of the differences in the various XPath engines.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Tag and Attribute Name</th>
<th>Attribute Values</th>
<th>Native XPath Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>HtmlUnit Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
<tr>
<td>Internet Explorer Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>No</td>
</tr>
<tr>
<td>Firefox Driver</td>
<td>Case insensitive</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This is a little abstract, so for the following piece of HTML:

```html
<input type="text" name="example" />
<input type="text" name="other" />
```

```java
List<WebElement> inputs = driver.findElements(By.xpath("//input"));
```

The following number of matches will be found:

<table>
<thead>
<tr>
<th>XPath expression</th>
<th>HtmlUnit Driver</th>
<th>Firefox Driver</th>
<th>Internet Explorer Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>/input</td>
<td>1 (&quot;example&quot;)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/INPUT</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Sometimes HTML elements do not need attributes to be explicitly declared because they will default to known values. For example, the "input" tag does not require the "type" attribute because it defaults to "text". The rule of thumb when using xpath in WebDriver is that you should not expect to be able to match against these implicit attributes.
Selenium-WebDriver API Commands and Operations

Locating UI Elements

By XPath

Firefox:
Finds 2 args for "input" tag
Finds 2 args for "INPUT" tag

Because:
- not case-sensitive
- showing all args of all attributes

By XPath
At a high level, WebDriver uses a browser's native XPath capabilities wherever possible. On those browsers that don't have native XPath support, we have provided our own implementation. This can lead to some unexpected behaviour unless you are aware of the differences in the various XPath engines.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Tag and Attribute Name</th>
<th>Attribute Values</th>
<th>Native XPath Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>HtmlUnit Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
<tr>
<td>Internet Explorer Driver</td>
<td>Lower-cased</td>
<td>As they appear in the HTML</td>
<td>No</td>
</tr>
<tr>
<td>Firefox Driver</td>
<td>Case insensitive</td>
<td>As they appear in the HTML</td>
<td>Yes</td>
</tr>
</tbody>
</table>

This is a little abstract, so for the following piece of HTML:

```html
<input type="text" name="example" />
<input type="text" name="other" />
```

```java
List<WebElement> inputs = driver.findElements(By.xpath("//input"));
```

The following number of matches will be found:

<table>
<thead>
<tr>
<th>XPath expression</th>
<th>HtmlUnit Driver</th>
<th>Firefox Driver</th>
<th>Internet Explorer Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>//input</td>
<td>1 (&quot;example&quot;)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>//INPUT</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Sometimes HTML elements do not need attributes to be explicitly declared because they will default to known values. For example, the "input" tag does not require the "type" attribute because it defaults to "text". The rule of thumb when using xpath in WebDriver is that you should not expect to be able to match against these implicit attributes.
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By ID

**By ID**
This is the most efficient and preferred way to locate an element. Common pitfalls that UI developers make is having non-unique id's on a page or auto-generating the id, both should be avoided. A class on an html element is more appropriate than an auto-generated id.

Example of how to find an element that looks like this:

```
<div id="coolestWidgetEvah">...</div>
```

```java
WebElement element = driver.findElement(By.id("coolestWidgetEvah"));
```
Selenium-WebDriver API Commands and Operations

Locating UI Elements
By Link Text

By Link Text
Find the link element with matching visible text.

Example of how to find an element that looks like this:

```java
<a href="http://www.google.com/search?q=cheese">cheese</a>
```

```java
WebElement cheese = driver.findElement(By.linkText("cheese"));
```
Data-Driven Testing

• Data-Driven Testing = Executing the same set of test steps with multiple (different) data

• Test inputs and expected outcomes stored as data
  – Typically in tabular format

• Test data are read from an external data source

• One driver script can execute all of the designed test cases

Note that in previous example test data (‘10’ and ‘50’) was embedded in the test case definition

Example data (new):
- imdb film database
- employees
public class DataProviderExample extends SeleneseTestCase{

@BeforeClass
public void setUp() throws Exception {
    ...'
}

@DataProvider(name = "DP1")
public Object[][] createData1() throws Exception{
    Object[][] retObjArr=getTableArray("test\Resources\Data\data1.xls", "DataPool", "imdbTestData1");
    return(retObjArr);
}

testDataProviderExample(String movieTitle, String directorName, String moviePlot, String actorName) throws Exception{
    //enter the movie title
    selenium.type("q", movieTitle);
    //they keep switching the go button to keep the bots away
    if (selenium.isElementPresent("nb15go_image"))
        selenium.click("nb15go_image");
    else
        selenium.click("xpath=/descendant::button[@type='submit']");
    selenium.waitForPageToLoad("30000");
    //click on the movie title in the search result page
    selenium.click("xpath=/descendant::a[text()="'+movieTitle+'"]");
    selenium.waitForPageToLoad("30000");
    //verify director name is present in the movie details page
    verifyTrue(selenium.isTextPresent(directorName));
    //verify movie plot is present in the movie details page
    verifyTrue(selenium.isTextPresent(moviePlot));
    //verify movie actor name is present in the movie details page
    verifyTrue(selenium.isTextPresent(actorName));
}

@AfterClass
...
public String[][] getTableArray(String xlFilePath, String sheetName, String tableName){
    String[][] tabArray=null;
    try{
        Workbook workbook = Workbook.getWorkbook(new File(xlFilePath));
        Sheet sheet = workbook.getSheet(sheetName);
        int startRow,startCol, endRow, endCol,ci,cj;
        Cell tableStart=sheet.findCell(tableName);
        startRow=tableStart.getRow();
        startCol=tableStart.getColumn();
        Cell tableEnd= sheet.findCell(tableName, startCol+1,startRow+1, 100, 64000, false);
        endRow=tableEnd.getRow();
        endCol=tableEnd.getColumn();
        System.out.println("startRow=",startRow," endRow=",endRow," startCol=",startCol," endCol=",endCol);
        tabArray=new String[endRow-startRow][endCol-startCol];
        ci=0;
        for (int i=startRow+1;i<endRow;i++,ci++){
            cj=0;
            for (int j=startCol+1;j<endCol;j++,cj++){
                tabArray[ci][cj]=sheet.getCell(j,i).getContents();
            }
        }
    }
    catch (Exception e)    {
        System.out.println("error in getTableArray()");
    }
    return(tabArray);
}
Data-Driven Testing

• External test data can be edited without programming skills
  Test design and framework implementation are now separate tasks:
  – design task can be given to someone with the domain knowledge
    (business people, customers) and
  – framework implementation to someone with programming skills.

• Avoids the problems of embedded test data
  – Data are hard to understand in the middle of all scripting details
  – Updating tests or creating similar tests with slightly different test data
    types/structures always requires programming (-> copy-paste scripting)

• Follow this link for a fully worked example of Data-Driven Testing with Selenium:
Data-Driven Testing

For detailed description of the concepts, see:


```python
data = open('testdata.tsv').read()
lines = data.splitlines()[1:]  # [1:] excludes the header row

for line in lines:
    testId, number1, operator, number2, expected = line.split('	')
    # Actual test functionality excluded
```
Keyword-Driven Testing

• Keywords also known as action words
• Keyword-driven testing improves data-driven testing:
  – Keywords abstract the navigation and actions from the script
  – Keywords and test data are read from an external data source
• When test cases are executed keywords are interpreted by a test library (=set of test scripts) which is called by a test automation framework
• Example 1:  
  Login: admin, t5t56y; // 2 args  
  AddCustomers: newCustomers.txt // 1 arg  
  RemoveCustomer: Pekka Pukaro // 1 arg

• Example 2: click, checkbox, undo, …
• More keywords (=action words) can be defined based on existing keywords
• Keyword driven testing ~= domain specific languages (DSL)
• Details: http://doc.froglogic.com/squish/4.1/all/how.to.do.keyword.driven.testing.html
• Another tool: http://code.google.com/p/robotframework/
Keyword-Driven Testing

Similar to data-driven testing
But instead of testing functions directly, handler looks for keywords and based on that derives required test-data
Purpose: increase flexibility and re-usability

Keyword-driven test data file adds one level of abstraction
Keyword-Driven Testing

Several layers of keywords possible

Benefit: can define new keywords using existing ones
Model-based Testing

• System under test is modelled
  – UML-state machines, domain specific languages (DSL)
• Test cases are automatically generated from the model
  – The model can provide also the expected results for the generated test cases
  – More accurate model -> better test cases
• Generate a large amount of tests that cover the model
  – Many different criteria for covering the model
  – Execution time of test cases might be a factor
• Challenges:
  – Personnel competencies
  – Data-intensive systems (cannot be modelled as a state-machine)
• Simple MBT tool http://graphwalker.org/
GraphWalker  [http://graphwalker.org/](http://graphwalker.org/)

Example:
A regression test for the login function of the Spotify Desktop Client

The feature is supposed to work like this:
- In a freshly installed client, and the client is started, the Login dialog is expected to be displayed.
- The user enters valid credentials and the browse view is expected to start.
- If the user quits, or logs out, the Login dialog is displayed once again.
- If the user checks the Remember Me checkbox, and logs in (using valid credentials), the client starts, and, next time the user starts the client, it will start without asking the client for credentials.
Example:
A regression test for the login function of the Spotify Desktop Client

For testing the 2 first steps, a model would look something like this:

1. The **Start** vertex is where the tests starts.
2. In **e_Init**, we remove all cache, and kill any previous client processes.
3. **v_ClientNotRunning** will assert that there is no Spotify client process running.
4. **e_Start** starts the client.
5. **v_LoginPrompted** asserts that the login dialog is displayed and correctly rendered.
6. **e_ValidPremiumCredentials** enters a valid username and password and clicks the Sign In button.
7. **v_Browse** asserts that the Browse view is correctly displayed.
Example:
A regression test for the login function of the Spotify Desktop Client
GraphWalker  [http://graphwalker.org/](http://graphwalker.org/)

Example:
A regression test for the login function of the Spotify Desktop Client

```java
%> java -jar graphwalker.jar offline -m Login.graphml "random(edge_coverage(100))" e_Init v_ClientNotRunning e_StartClient v_LoginPrompted e_InvalidCredentials v_LoginPrompted e_ValidPremiumCredentials v_Browse e_Logout v_LoginPrompted e_Close v_ClientNotRunning ...
```
This command generates a random test sequence achieving 100% edge (branch) coverage.

Example:
A regression test for the login function of the Spotify Desktop Client

```java
%> java -jar graphwalker.jar offline -m Login.graphml "random(edge_coverage(100))"
```

```
e_Init /rememberMe=false;validLogin=true;
e_StartClient [!rememberMe&!validLogin]
ev_LoginPrompted
e_ToggleRememberMe /rememberMe=!rememberMe
```

```
e_InvalidCredentials /validLogin=false;
e_Logout
```

```
e_ValidPremiumCredentials /validLogin=true;
e_Logout
```

```
e_Init
v_ClientNotRunning
e_StartClient
ev_LoginPrompted
e_InvalidCredentials
ev_Logout
e_ValidPremiumCredentials
ev_Browse
e_Logout
e_LoginPrompted
e_Close
ev_ClientNotRunning...
```
Functional Testing Approaches

1. Recorded Scripts
   - Cheap to set up, quick & dirty

2. Engineered Scripts
   - Structured

3. Data-driven Testing
   - Data separation

4. Keyword-driven Testing
   - Action separation, DSL

5. Model-based Testing
   - Modeling & Automatic test case generation
Structure of Lecture 6

• Test Levels
• Test Tools
• Test Automation
• Lab 6
What can be automated?

What can be automated?

- Test generation (test case & script)
  - Test case (steps with data & oracle)
  - Test data (input)
  - Test oracle (expected output)
  - Test verdict (PASS/FAIL decision)
- Test selection & execution
  - E.g., regression testing
- Test reporting
- Debugging
  - Fault localisation (using failure/error information)
What can be automated?

- Test generation (test case & script)
  - Test case (steps with data & oracle)
  - Test data (input)
  - Test oracle (expected output)
  - Test verdict (PASS/FAIL decision)
- Test selection & execution
  - E.g., regression testing
- Test reporting
- Debugging
  - Fault localisation (using failure/error information)

Often, people mean automated test execution when they talk about test automation.
When to automate?

- Test Automation should be used by considering the following:
  - Large and critical projects
  - Projects that require testing the same areas frequently
  - Requirements not changing frequently
  - Accessing the application for load and performance with many virtual users
  - Stable software
  - Availability of time/effort (for set-up, execution, maintenance, etc.)
Test automation promises

1. Efficient regression test
2. Run tests more often
3. Perform difficult tests (e.g. load, outcome check)
4. Better use of resources
5. Consistency and repeatability
6. Reuse of tests
7. Earlier time to market
8. Increased confidence
Common problems

1. Unrealistic expectations
2. Poor testing practice
   "Automatic chaos just gives faster chaos"
3. Expected effectiveness
4. False sense of security
5. Maintenance of automatic tests
6. Technical problems (e.g. Interoperability)
7. Organizational issues
Limits of automated testing

• Does not replace manual testing
• Manual tests find more defects than automated tests
  – Does not improve effectiveness
• Greater reliance on quality of tests
  – Oracle problem
• Test automation may limit the software development
  – Costs of maintaining automated tests
Structure of Lecture 6

• Test Levels
• Test Tools
• Test Automation
• Lab 6
HW 6 – Automated Web App Testing

Lab 6 (week 31: Mar 29 & 30) – Automated Web Application Testing (9 points)

Lab 6 Instructions & Tools

Submission Deadlines:
- Tuesday Labs: Monday, 04 Apr, 23:59
- Wednesday Labs: Tuesday, 05 Apr, 23:59

- Penalties apply for late delivery: 50% penalty, if submitted up to 24 hours late; 100 penalty, if submitted more than 24 hours late
HW 6 – Automated Web App Testing

Lab 6 (week 31: Mar 29 & 30) – Automated Web Application Testing (9 points)

Lab 6 Instructions & Tools

In order to get full marks you must at least submit the following:

- at least 20 test cases in total. This should include at least 5 negative tests.
- The tests should find at least 2 of the seeded bugs, that means 2 of the tests, should be failing tests.

Find defects
Add test code
(positive & negative)
Next Weeks

• Quiz 6 (in Moodle!):
  • Opens at end of lecture – closes on Monday at 17:30am!
• Finish and submit HW 5:
  – Random Testing (with Randoop)
• Lab 6:
  – Automated Web Application Testing (Selenium)
• Lecture 7:
  – Black-Box Testing (advanced): Exploratory Testing / Behaviour Testing