Unit-тестирование в Java

План лекции

- Зачем писать модульные тесты
- Семейство Xunit
- JUnit
- TestNG
- Mock Objects
- Stub
- Spy

Зачем писать модульные тесты

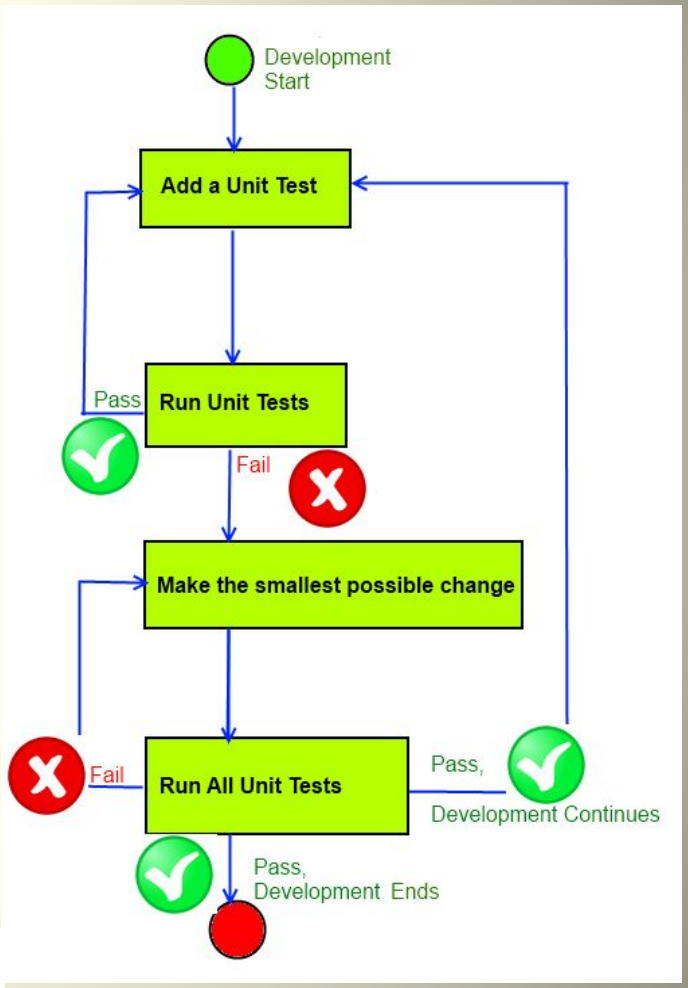
Преимущества

- Игнорируются при сборке
- Показывает, что отдельные части программы работоспособны
- Упрощает поиск ошибок
- Позволяет проводить рефакторинг будучи уверенным, что модуль работает корректно

Недостатки

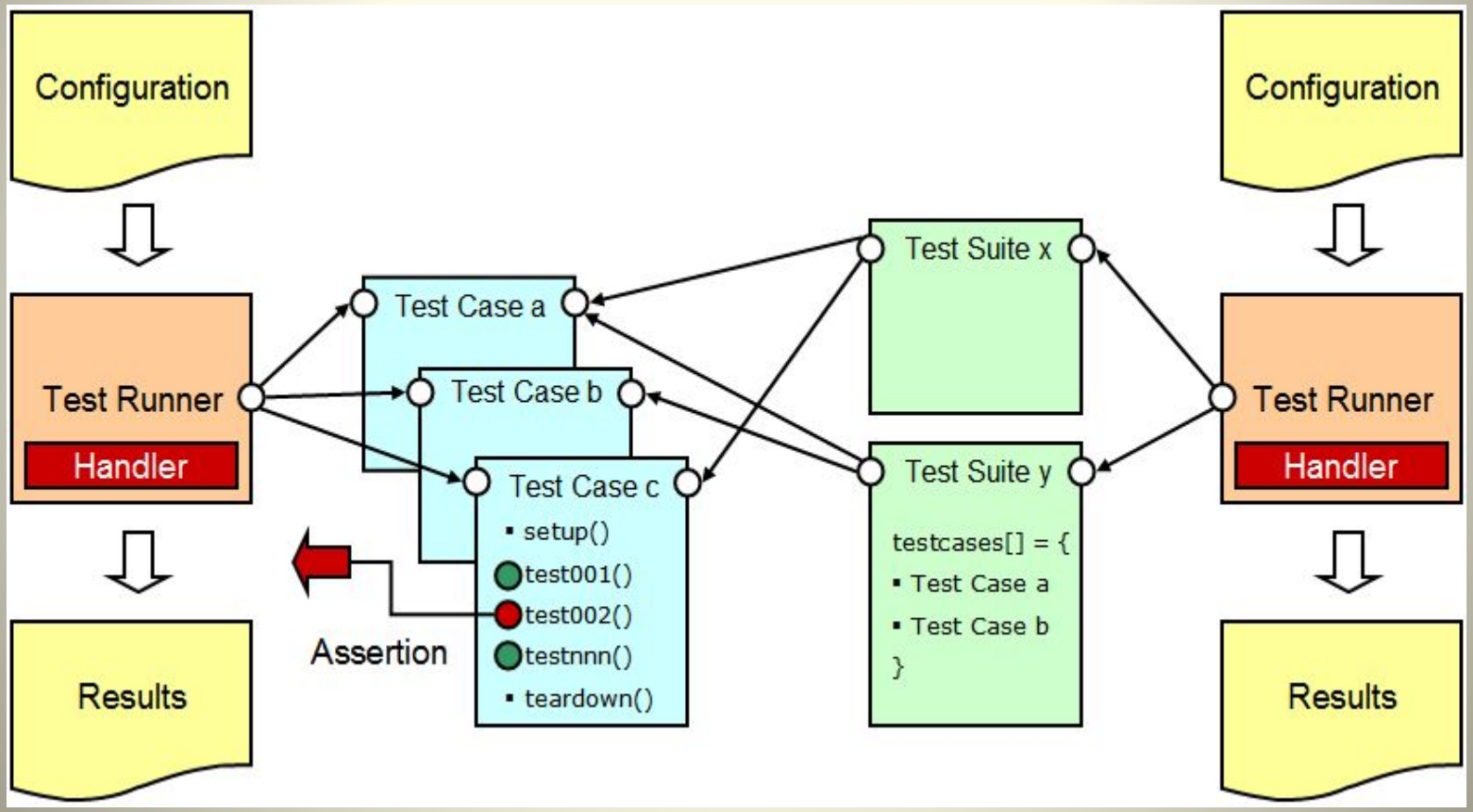
- Написание лишнего кода (иногда тесты больше самого модуля)

Мифы тестирования



Семейство xUnit

- Архитектура xUnit



Пример типичного теста

- setup ();

/* Подготовка*/

- testExampleMethod();

/*

Тело теста - Здесь мы выполняем все тесты

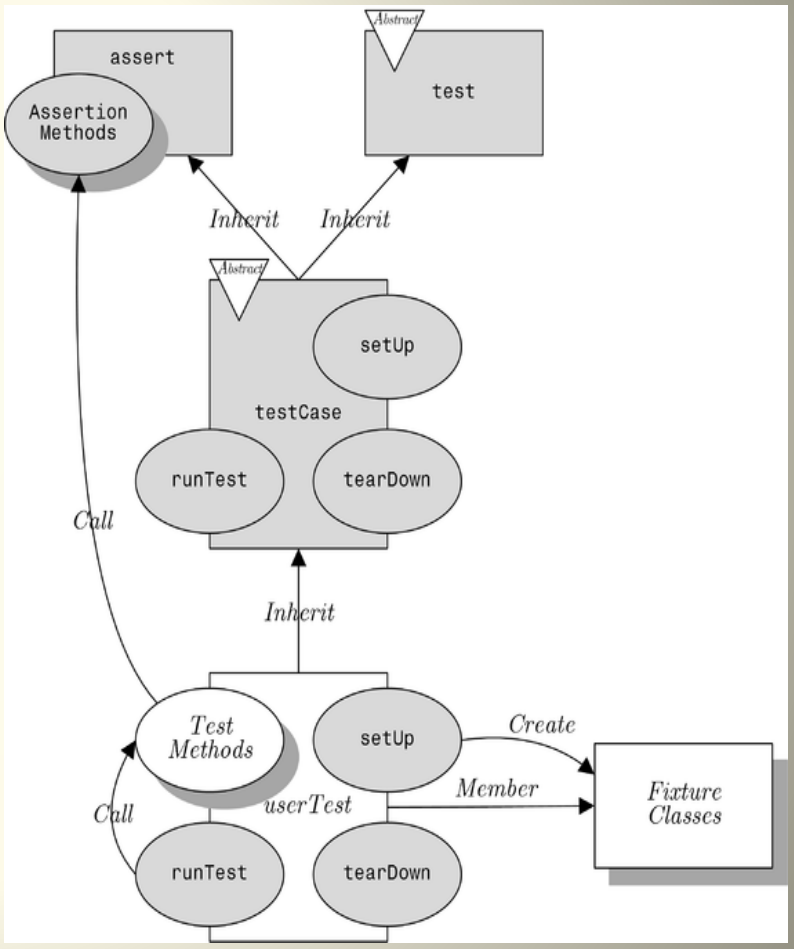
*/ ...

- teardown ();

/* Приведение в первоначальное состояние*/

Пример теста JUnit

```
import junit.framework.*;
public class JavaTest extends TestCase
{
     protected int value1, value2;
     // assigning the values
    protected void setUp()
   {
      value1=3; value2=3;
    }
    // test method to add two values
    public void testAdd() {
       double result= value1 + value2;
       assertTrue(result == 6);
    }
}
```



Пример теста TestNG

```
public class TestNGSimpleTest {
   int testInt;
   @BeforeMethod
   public void setUp() {
      testInt = 0;
   }
   @Test
   public void addTest() {
      testInt++;
      assert (testInt == 1);
      System.out.println("add test");
   }
   @Test
   public void subtractTest() {
      testInt--;
      assert (testInt == -1);
      System.out.println("subtract test");
   }
}
```

- Особенности

- Маркировка тестового класса или методов с помощью аннотации @Test.

- Обозначение методов пред- и пост-условий с помощью аннотаций @Before* и @After*.

- Проверочные методы класса Assert.

Пример набора тестов TestNG

```
<suite name="Suite1" verbose="1" >
 <test name="Nopackage" >
  <classes>
    <class name="NoPackageTest" />
  </classes>
 </test>

 <test name="Regression1">
  <classes>
    <class name="test.sample.ParameterSample"/>
    <class name="test.sample.ParameterTest"/>
  </classes>
 </test>
</suite>
```

Сравнение Junit/TestNG

Functionality - JUnit 4 vs TestNG

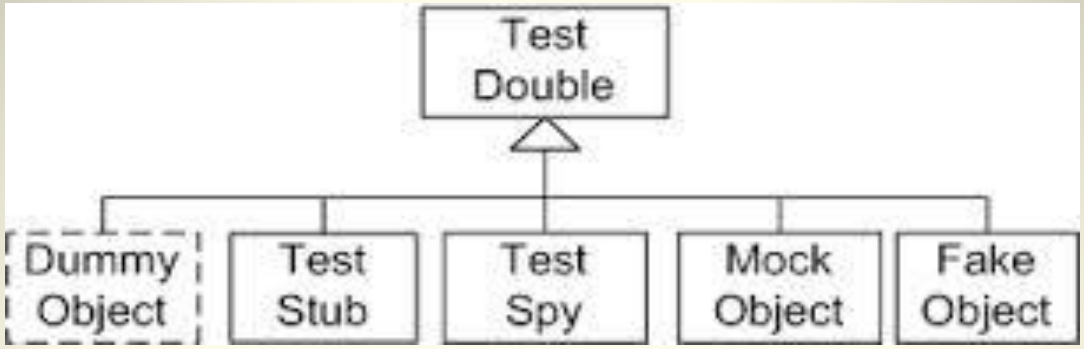
	Annotation Support	Exception Test	Ignore Test	Timeout Test	Suite Test	Group Test	Parameterized (primitive value)	Parameterized (object)	Dependency Test
TestNG	✓	✓	✓	✓	✓	✓	✓	✓	✓
JUnit 4	✓	✓	✓	✓	✓	✗	✓	✗	✗

TestNG имеет уникальную концепцию "Grouping", каждый метод привязывается к группе, он может классифицировать тесты в соответствии с особенностями.

JUnit не поддерживает зависимостей теста на данный момент. TestNG использует "dependOnMethods" для реализации тестирования зависимостей.

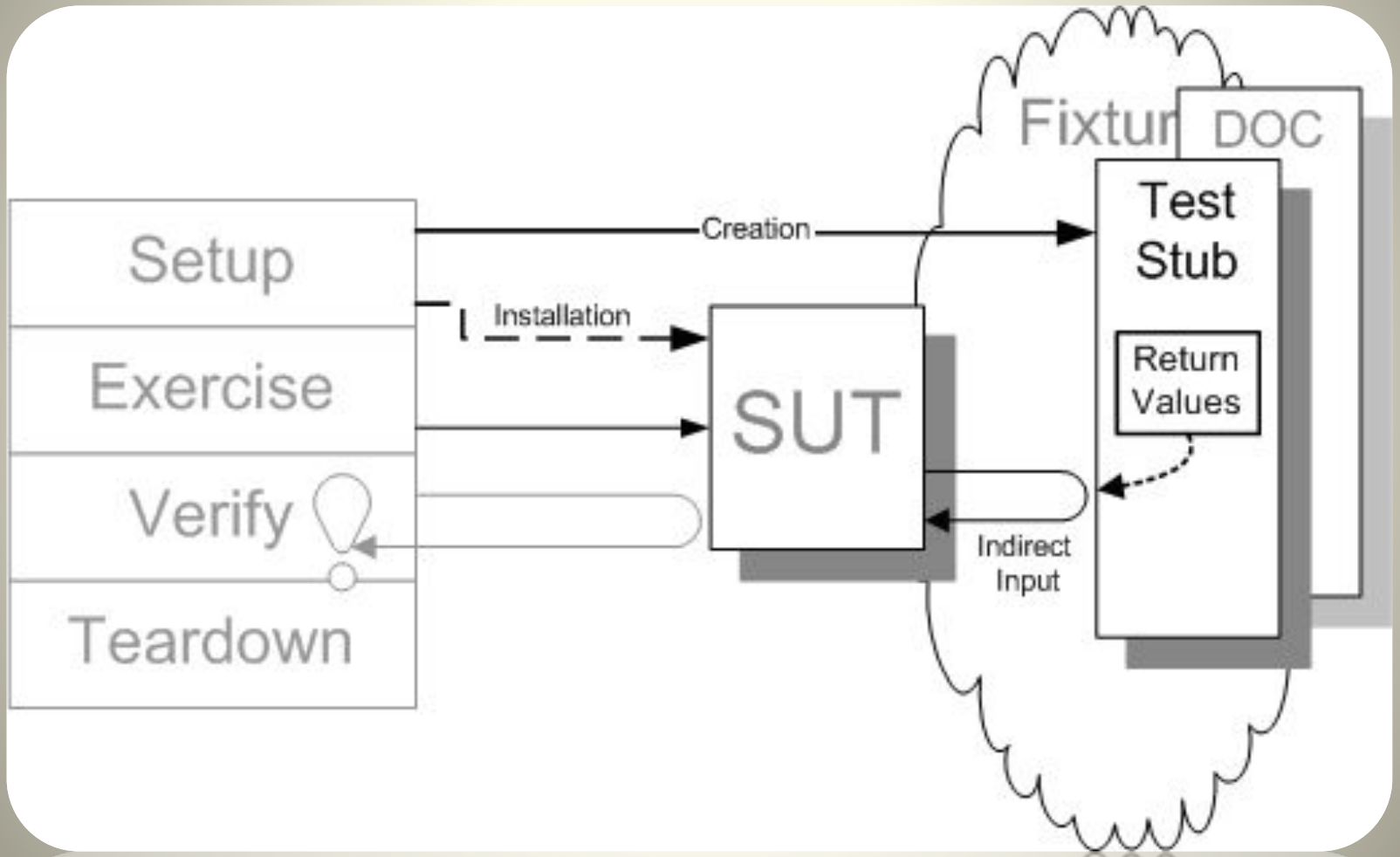
- http://testng.org/
- http://junit.org/

Дублёры (Test Double)



Тип Test Double	Описание
Stub (Заглушка)	Используем stub'ы для замены настоящих объектов, от которых зависит SUT (System Under Test), и получения возможности неявной передачи данных системе из теста
Spy (Шпион)	Усовершенствованная версия test stub. Ведет журнал обращений от SUT к себе. После выполнения шага Exercise данные используются для проверки корректности работы SUT
Mock (Прототип)	Используется для проверки неявных выходных данных и взаимодействий SUT по мере работы системы. Mock Object включает в себя обязанности Test Stub в том плане, что он возвращает какие-то данные системе

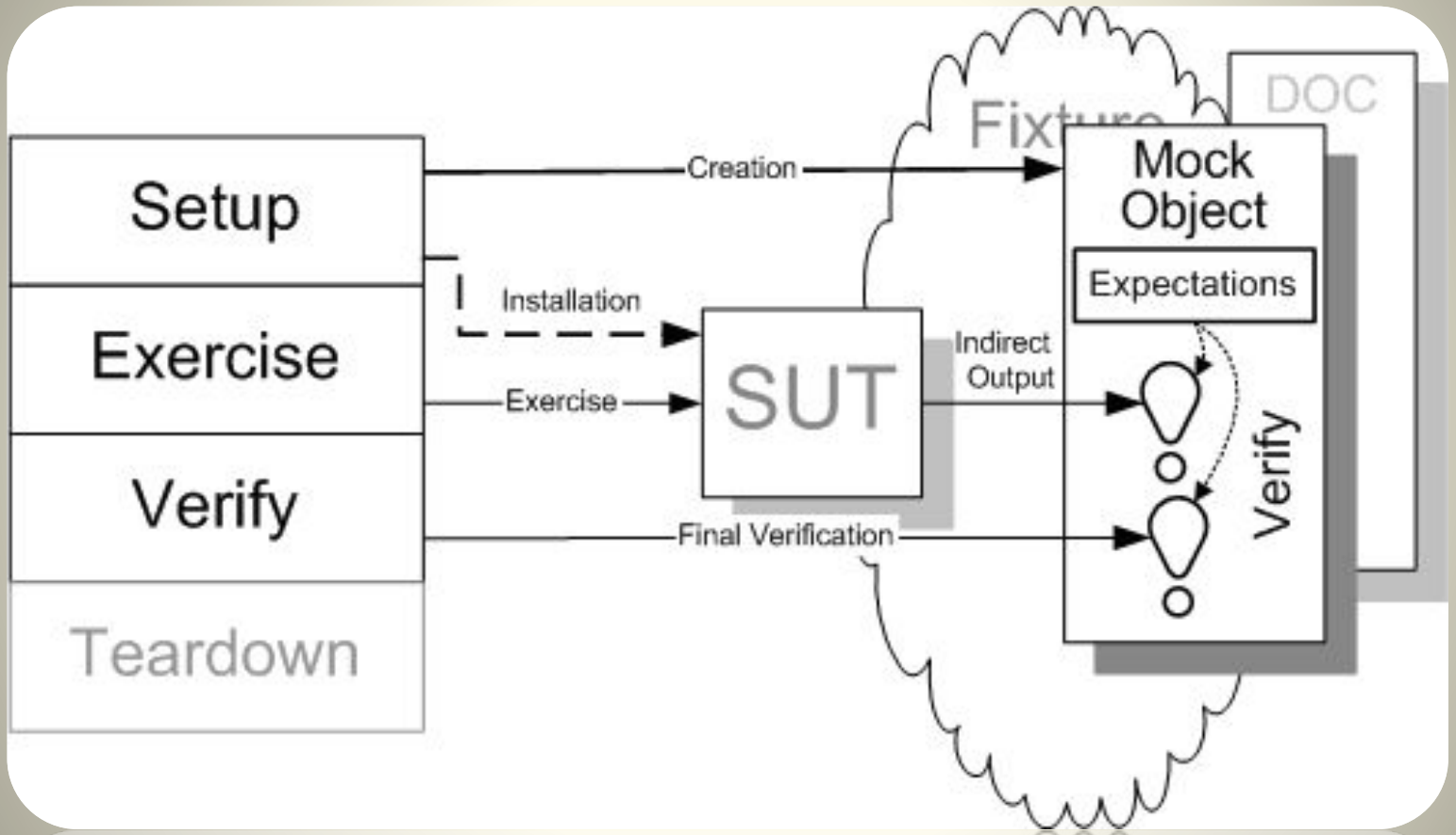
Test Stub



Пример Stub

```
//Вы можете создавать mock для конкретного класса, не только для интерфейса
LinkedList mockedList = mock(LinkedList.class);

//stub'инг
when(mockedList.get(0)).thenReturn("first");
when(mockedList.get(1)).thenThrow(new RuntimeException());

//получим "first“
System.out.println(mockedList.get(0));

//получим RuntimeException
System.out.println(mockedList.get(1));

//получим "null" ибо get(999) не был пределен
System.out.println(mockedList.get(999));
```

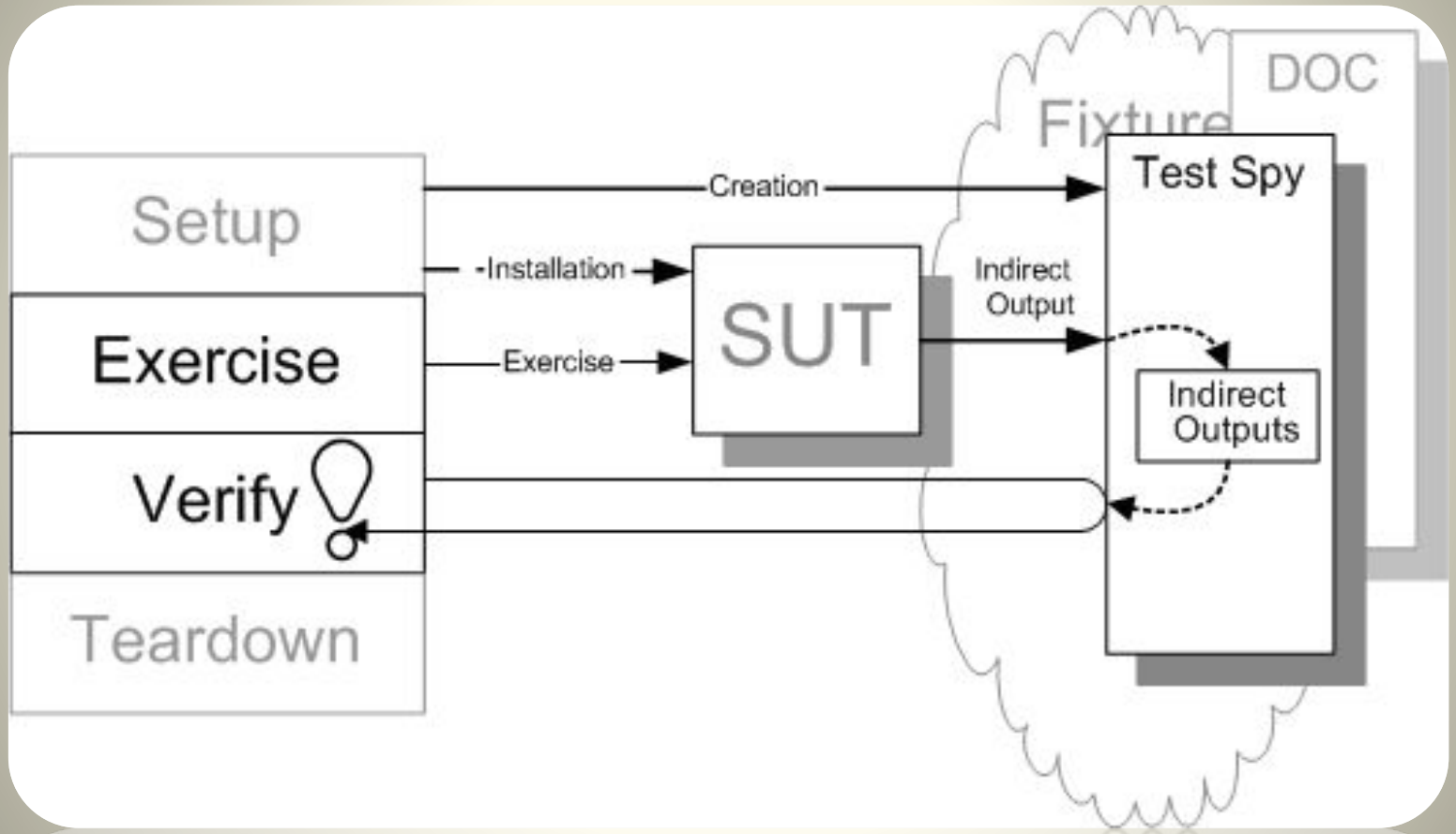
Mock Object


Setup
Exercise
Verify
Teardown
Creation
Installation
Exercise
Final Verification
SUT
Indirect Output
Fixture
DOC
Mock Object
Expectations
Verify

Пример Mock Object

```
//статически импортируем методы (для красоты и легкости кода)
import static org.mockito.Mockito.*;

//вот он - mock-объект (заметьте: List.class - это интерфейс)
List mockedList = mock(List.class);

//используем его
mockedList.add("one");
mockedList.clear();

//проверяем, были ли вызваны методы add с параметром "one" и
//clear
verify(mockedList).add("one");
verify(mockedList).clear();
```

Test Spy


DOC
Fixture
Setup
Exercise
Verify
Teardown
Creation
Installation
Exercise
SUT
Indirect
Output
Test Spy
Indirect
Outputs

Пример Spy

```
List list = new LinkedList();
List spy = spy(list);
//опционально, определяем лишь метод size()
when(spy.size()).thenReturn(100);
//используем реальные методы
spy.add("one");
spy.add("two");
//получим "one"
System.out.println(spy.get(0));
//метод size() нами переопределён - получим 100
System.out.println(spy.size());
//можем проверить
verify(spy).add("one");
verify(spy).add("two");
```

Популярные mock фреймворки в Java