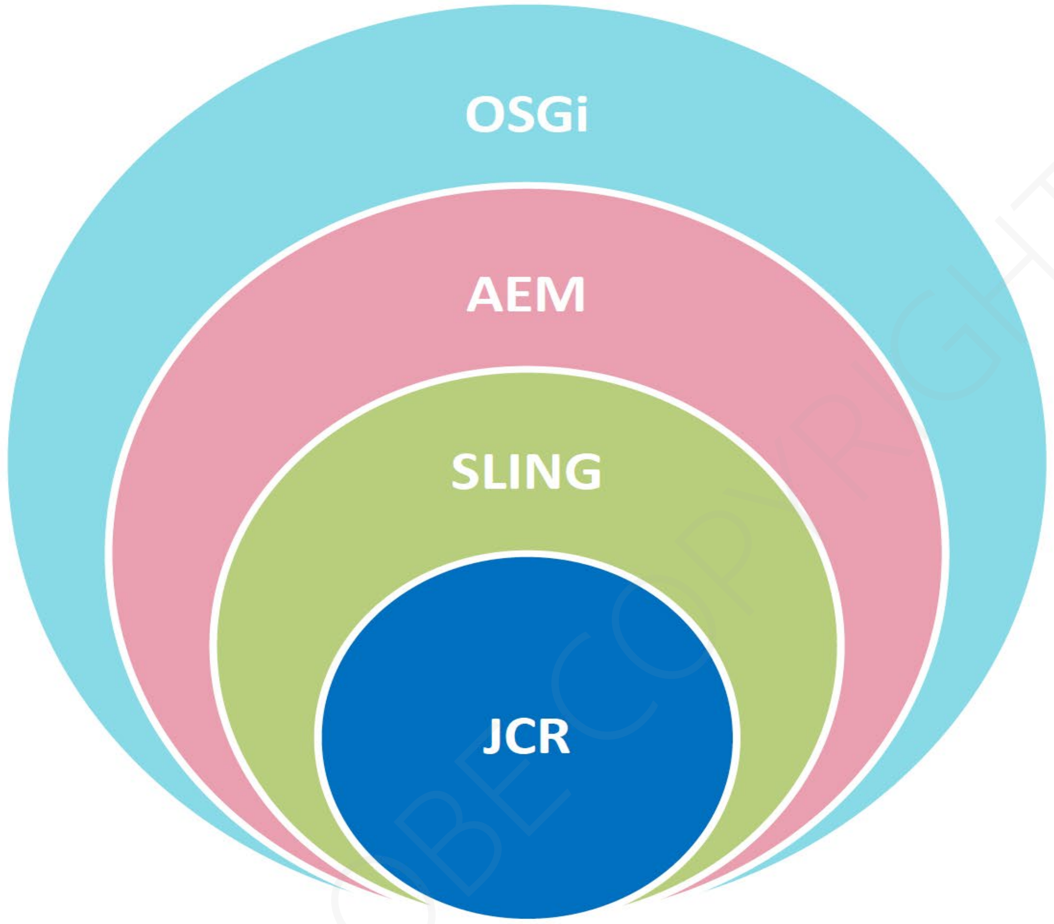
Writing Junit Tests

Agenda:

- Describe the testing framework
- Explain the different types of testing
- Run unit tests and functional tests on your project
- Create unit tests using Mockito
- Create unit tests using Sling Mocks
- Create unit tests using AEM Mocks

Unit Testing AEM applications



1. JCR testing tools
2. Sling API level
3. OSGi API level
4. AEM application level

The JUnit Framework

- JUnit is a testing framework for Java that is used to implement unit testing.
  - o Multiple tests can be run at the same time with no human intervention to interpret results.
- Running JUnit tests
  - o IDE integrations
  - o Maven:
    - ❑ mvn clean test
    - ❑ mvn –Dtest=MyTestsCase, MyOtherTestCase test
- JUnit5 is used in the modern AEM Archetype

Mock Testing

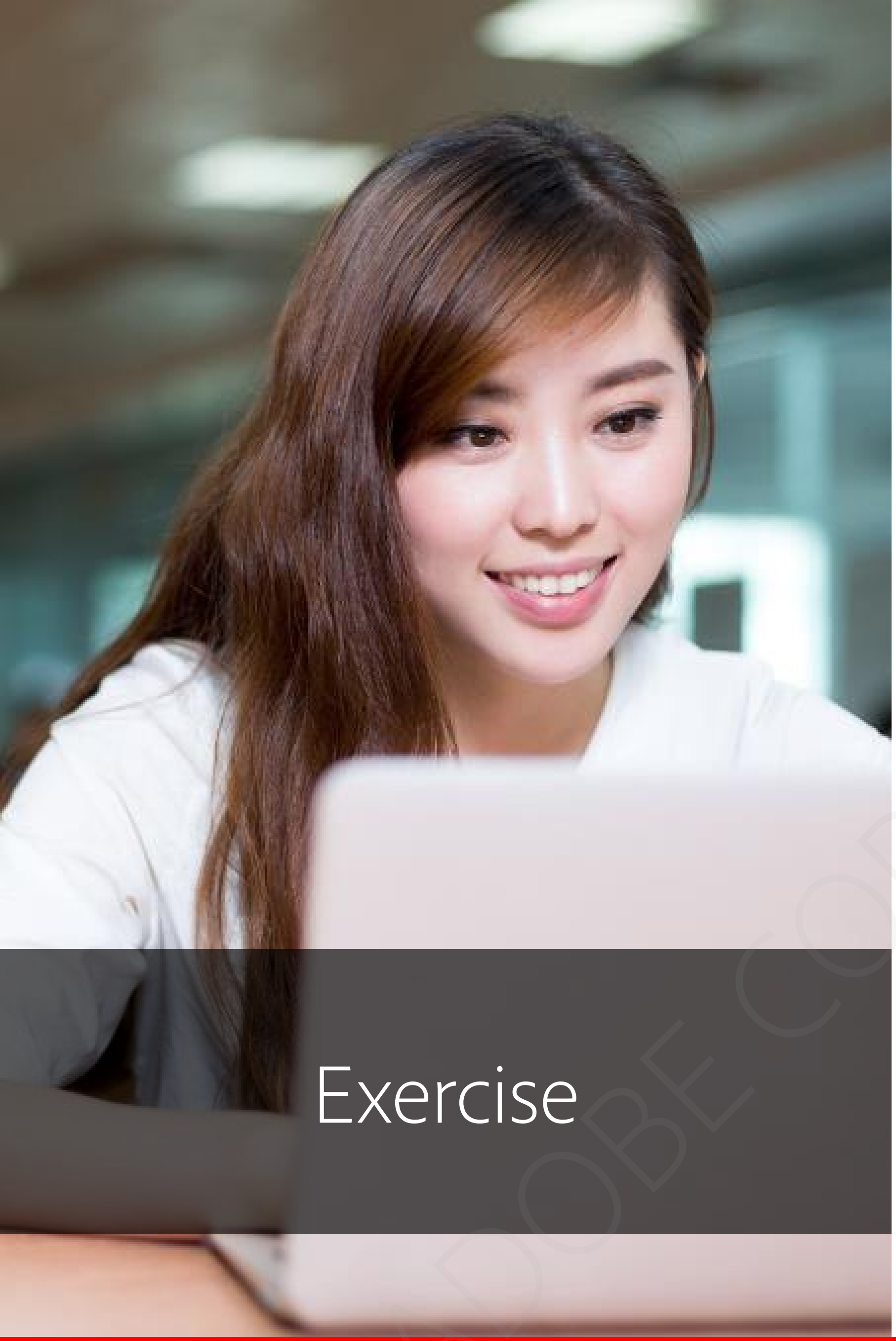
- Method that simulates the behavior of a real method/object in controlled ways
- Mock objects are used in unit testing
- Mock Frameworks to use with AEM:
    - Mockito – Generic mocking framework
    - Sling Mocks – Quickly mock sling objects
    - AEM Mocks – Quickly mock AEM objects

AEM mocks is the most ideal framework because it contains:

- AEM Mock Objects
- OSGi, Sling, and JCR objects as well

Mockito Framework

- Open source testing framework for mock objects
- Useful for generic logic not applicable to AEM
- Features of Mockito:
  - Mocks concrete classes as well as interfaces
  - Verification errors are clean. Stack trace is always clean.
  - Allows flexible verification in order
  - Supports exact-number-of-times and at-least-once verification
  - Flexible verification or stubbing using argument matchers
  - Allows creating custom argument matchers or using existing hamcrest matchers



Exercise 1: Create unit tests using Mockito

In this lab exercise, you will create a unit test and test the Stockplex java class you created earlier. This is a basic unit Test outside the server. If you do not have the Stockplex.java class implemented, then you will need to go back and perform those exercises first.

Sling Mocks Framework

- Mocks for selected Sling APIs for easier testing
- Features
  - Mock Sling Resource API
  - Easy resource creation
  - Import resources from JSON files
  - Simulate request, capture response
  - Support for Sling Models
  - Supports AEM 6.0 – 6.4

Some Examples: SlingContext JUnit Rule

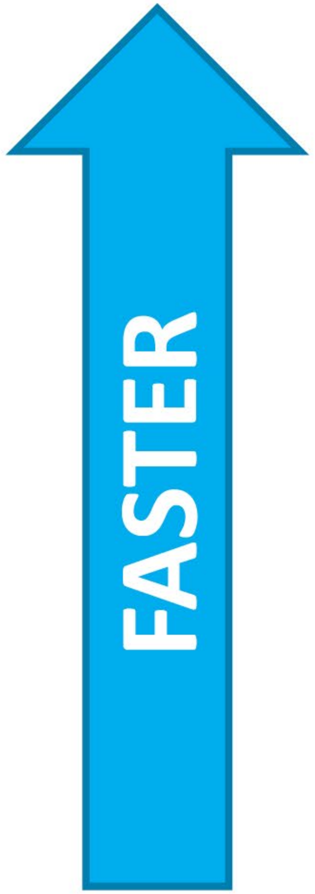
```
@Rule
public final SlingContext context = new SlingContext();

@Test
public void testSomething() {
 Resource resource = context.resourceResolver().getResource("/content/sample/en");
 // further testing
}
```

new SlingContext(
1. RESOURCERESOLVER_MOCK
2. JCR_MOCK
3. NONE
4. JCR_JACKRABBIT
5. JCR_OAK

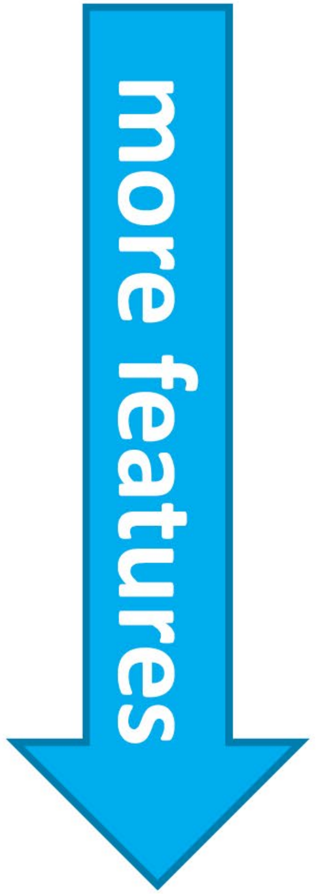
)

org.apache.sling : org.apache.sling.testing.sling-mock

Some Examples: Choose Resource Resolver implementation



Resource Resolver Type	Sling API	JCR API	Node Types	Obser-vation	JCR Query	Lucene Fulltext
RESOURCE RESOLVER_MOCK	✓	✗	✗	✗ (Sling only)	✗	✗
JCR_MOCK	✓	✓	✗	✗	✗ (mocked)	✗
JCR_OAK	✓	✓	✓	✓	✓	✗



Some Examples: Create or Import test data

```
ContentLoader contentLoader = new ContentLoader(resolver);
contentLoader.json("/sample-data.json", "/content/sample/en");

// Import binary data from file in classpath
ContentLoader contentLoader = new ContentLoader(resolver);
contentLoader.binaryFile("/sample-file.gif", "/content/binary/sample-file.gif");

ContentBuilder contentBuilder = new ContentBuilder(resolver);
contentBuilder.resource("/content/test1", ImmutableMap.<String, Object>builder()
      .put("prop1", "value1").put("prop2", "value2").build());
```

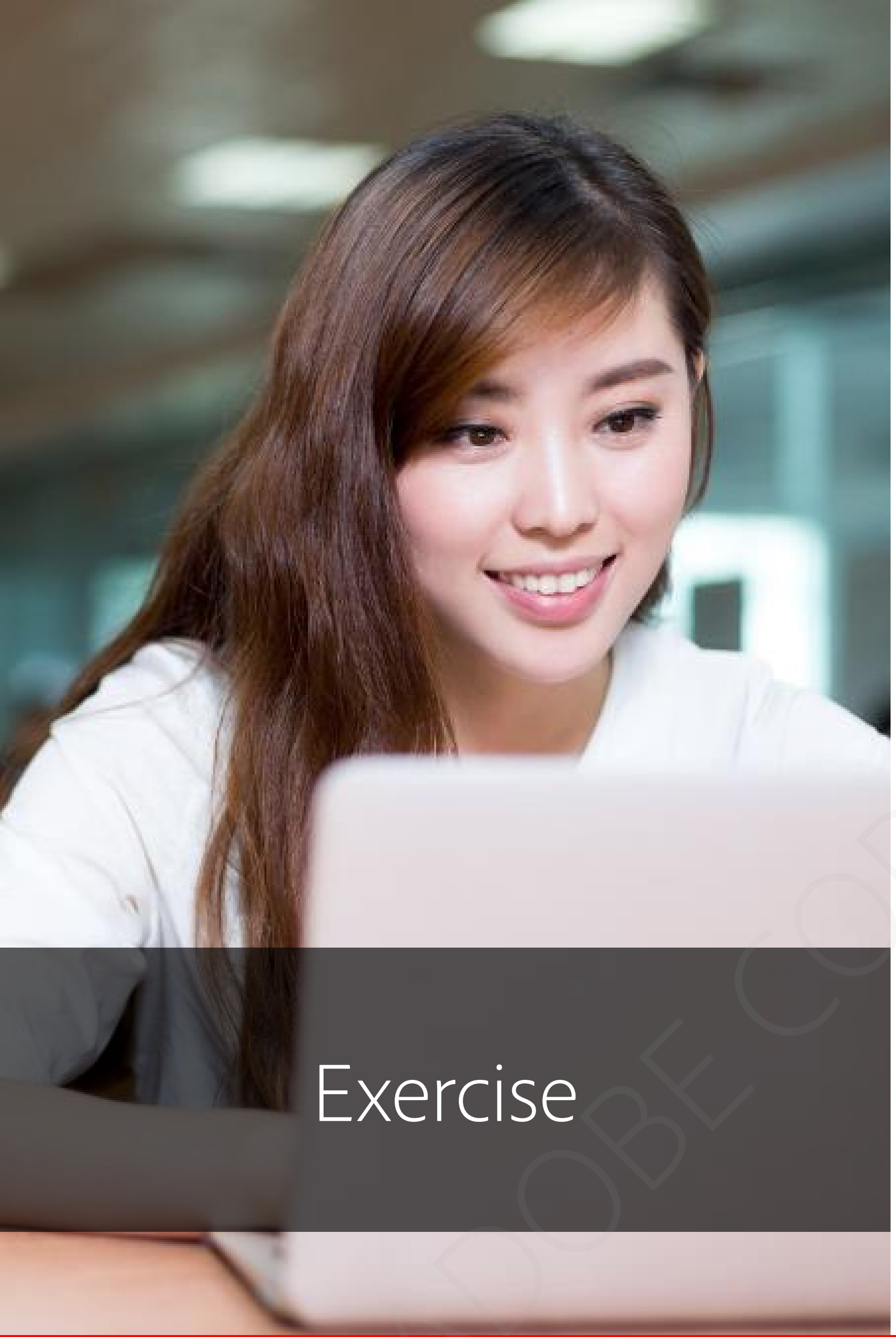
Some Examples: ResourceBuilder integration

```
context.build().resource("/content/site1", "prop1", "value1")
               .resource("en")
               .siblingsMode()
               .resource("page1", "jcr:title", "My title")
               .resource("page2");
```

```
/content
    /site1
      @prop1 = "value1"
        /en
            /page1
              @jcr:title = "My title"
            /page2
```

Some Examples: Sling Helper, Request and Response

```
SlingScriptHelper scriptHelper = MockSling.newSlingScriptHelper();
SlingHttpServletRequest request = scriptHelper.getRequest();

// get service
MyService object = scriptHelper.getService(MyService.class);

MockSlingHttpServletRequest request = new MockSlingHttpServletRequest(resourceResolver);
request.setQueryString("param1=aaa&param2=bbb");
request.setResource(resourceResolver.getResource("/content/sample"));

MockSlingHttpServletResponse response = new MockSlingHttpServletResponse();
```



Exercise 2: Create unit tests using Sling Mocks

In this lab exercise, you will create unit tests using Sling mocking framework.

- Task 1: Add the sling-mock dependency
- Task 2: Create a unit test with sample data
- Task 3: Run the test

AEM Mocks Framework

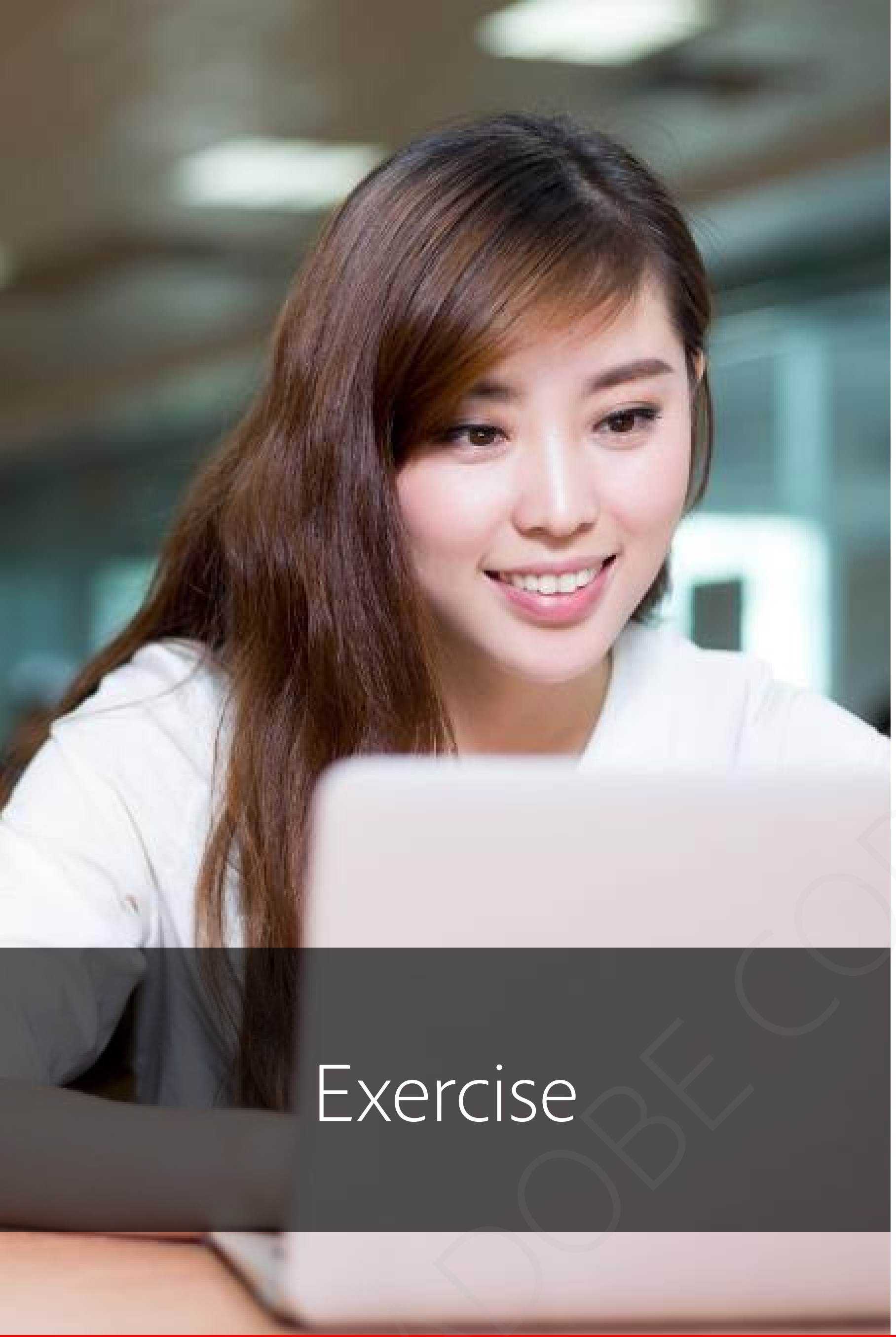
- AemContext for AEM mocks
  - Takes care of all initialization and cleanup tasks required
  - Provides quick context/setup of AEM objects like pages, templates, assets, and so forth…
- AemContext object also provides access to other implementations
  - OSGi Component Context
  - OSGi Bundle Context
  - Sling Resource Resolver
  - Sling Request
  - Sling Response
  - Sling Script Helper

AEM Mocks Framework

- Features of AEM Mocks Framework
  - Access to mocked OSGi, mocked JCR, and mocked Sling environment
  - Resource access using different resource resolver types
  - Implementation of AEM API Objects
  - Easy access to all context objects. Import and create test content for unit tests
  - Registers OSGi services and adapter factories
  - Full support for Sling Models
  - Setting run modes

AEM API Mock Objects

- PageManager
- Page, Template
- ComponentManager
- Component
- TagManager
- Tag
- Designer
- ComponentContext
- EditContext
- EditConfig
- Asset
- Rendition



Exercise 3: Create unit tests using AEM Mocks

In this lab exercise, you will create unit tests using AEM mocking framework.

- Task 1: Add the AEM-mock dependency
- Task 2: Create a unit test with sample data
- Task 3: Run the test

Key Takeaways

Key takeaways from this module:

- Mockito Framework
  - Mockito is an open-source testing framework that allows the creation of mock objects in automated unit tests.

- Continuous Integration
  - Continuous integration is a process where all development work is integrated to a system at a predefined time and is automatically tested and built.