Sling Models for component logic

Agenda:

- Describe Sling Models
- Discuss the Sling Model Exporter in AEM
- Extend a core component
- Create a Sling Model

Working With Sling Models

Sling Models allow for object mapping to the JCR

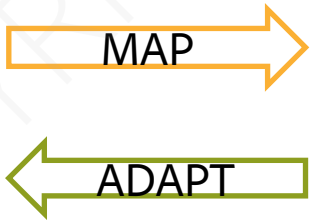
They have the following objectives:

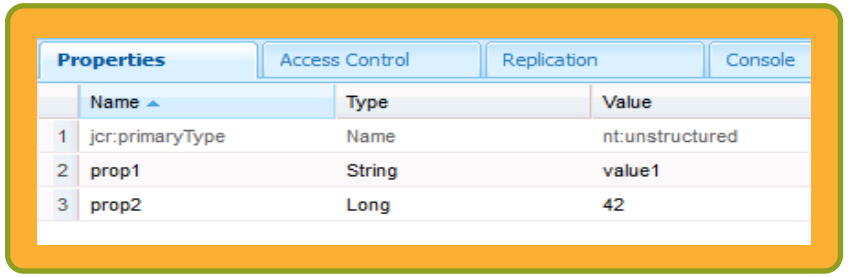
- Entirely annotation-driven
- Use standard annotations where possible
- Pluggable
- Out-of-the-box (OOTB) support resource properties, SlingBindings, OSGi services, and request attributes
- Adapt multiple objects
- Support both classes and interfaces
- Work with existing Sling infrastructure

Using Sling Models for an AEM Project

- The primary purpose of Sling Models – in a modern AEM implementation – is component business logic:
  - Easy mapping of content nodes to a single object
  - Model ease of use within HTL
  - Allows for natural progression into a headless implementation

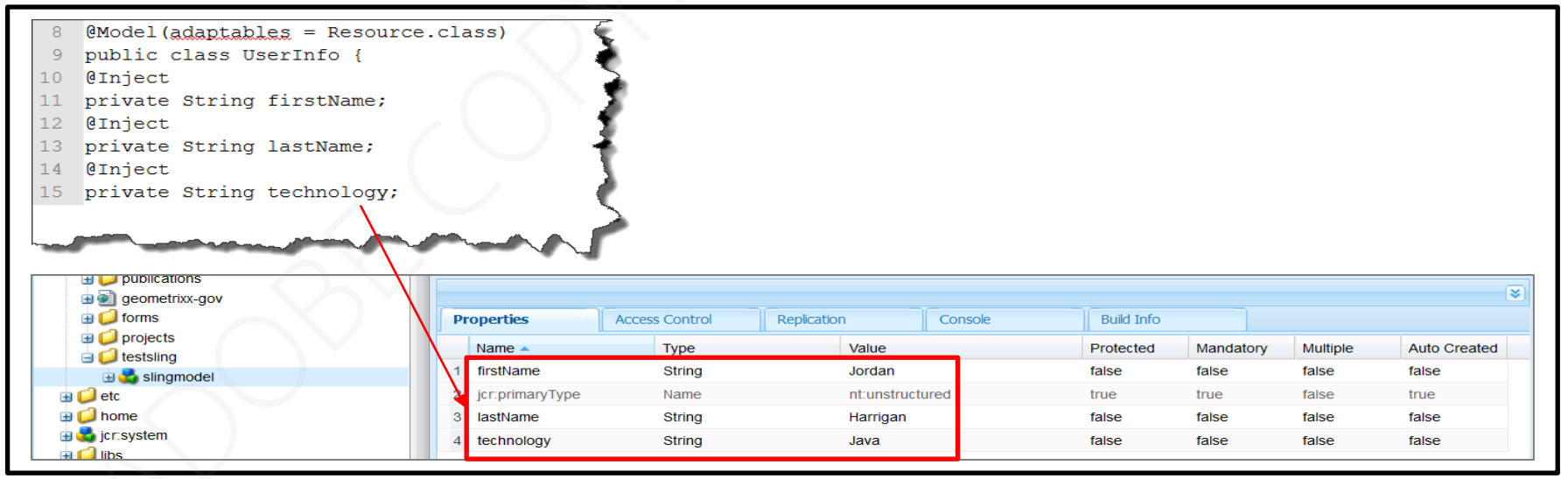
```
@Model(adaptables=Resource.class)
public class MyModel {

  @Inject
    private String prop1;

  @Inject
    private Long prop2;

  public String getProp1() {
        return prop1;
  }
  public Long getProp2() {
        return prop2;
  }
}
```





Understanding the Sling Model

- A Sling Model is implemented as an OSGi bundle.
- A Java class annotated with @Model and the adaptable class
- Java Object that represents data from the adaptable class
  - For example: Object representing nodes/properties from the JCR
- Getter methods are used to return values
- @PostConstruct is used to initialize the model



Benefits of Sling Models

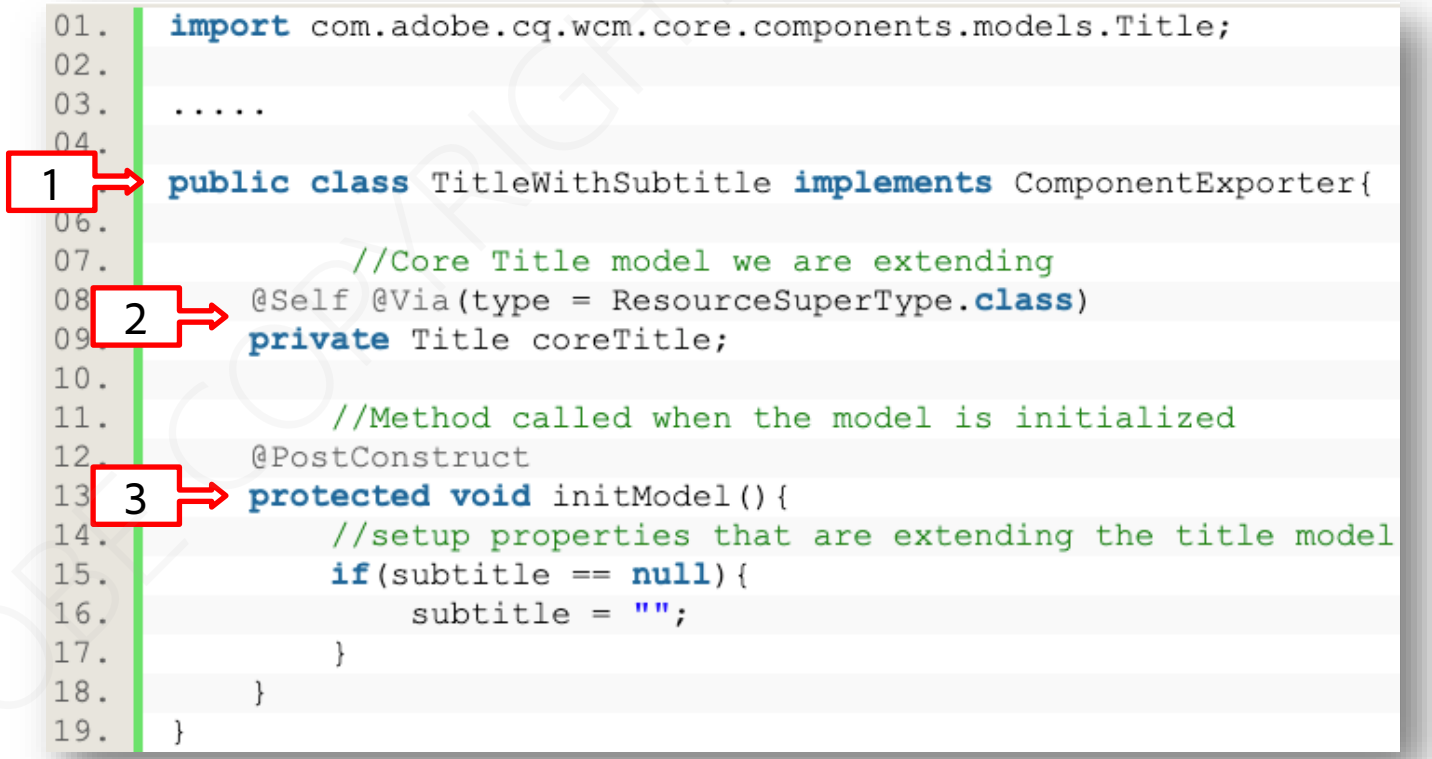
- Save time on creating your own adapters (avoiding boilerplate code)
- Support both classes and interfaces.
- Adapt other sling objects
  - SlingHttpServletRequest – should be adapted for component logic
- Works with existing Sling infrastructure
- You can mock dependencies with tools like Mockito, Sling Mocks, and AEM Mocks
- Easily exported to JSON via the Sling Model Exporter
- Access AEM global objects with @ScriptVariable annotation

AEM Sites Core Components

- Set of components developed by Adobe
- Open source on GitHub
- Highly configurable
- Production-ready
- Versionable
- Use Sling Models for business logic
- Exportable via JSON
- Built for extensibility (presentation and business logic)

Extending a Core Component logic

1. Create a new Sling model
2. Get the core component model
3. Set up any new/extended values




```
01. import com.adobe.cq.wcm.core.components.models.Title;
02.
03. .....
04.
06. public class TitleWithSubtitle implements ComponentExporter{
07.
            //Core Title model we are extending
08.     @Self @Via(type = ResourceSuperType.class)
09.     private Title coreTitle;
10.
11.         //Method called when the model is initialized
12.     @PostConstruct
13.     protected void initModel(){
14.         //setup properties that are extending the title model
15.         if(subtitle == null){
16.             subtitle = "";
17.         }
18.     }
19. }
```

Extending a Core Component logic (Cont.)

- Create **getters** for the original core model
- Create **getters** for all new/extended values
- Exports all getters as JSON
  - Only core cmp properties specified get outputted

```java
public String getTitle() {
    return coreTitle.getText();
}

public String getLinkURL() {
    return coreTitle.getLinkURL();
}

public String getSubtitle() {
    return subtitle;
}
```

JSON exported

```
"titlewithsubtitle": {
    "subtitle": "A true story explained",
    "empty": false,
    "linkURL": "/content/training/us/en/sling-models-explained.html",
    "title": "Sling Models are Awesome!",
    ":type": "training/components/content/titlewithsubtitle"
}
```

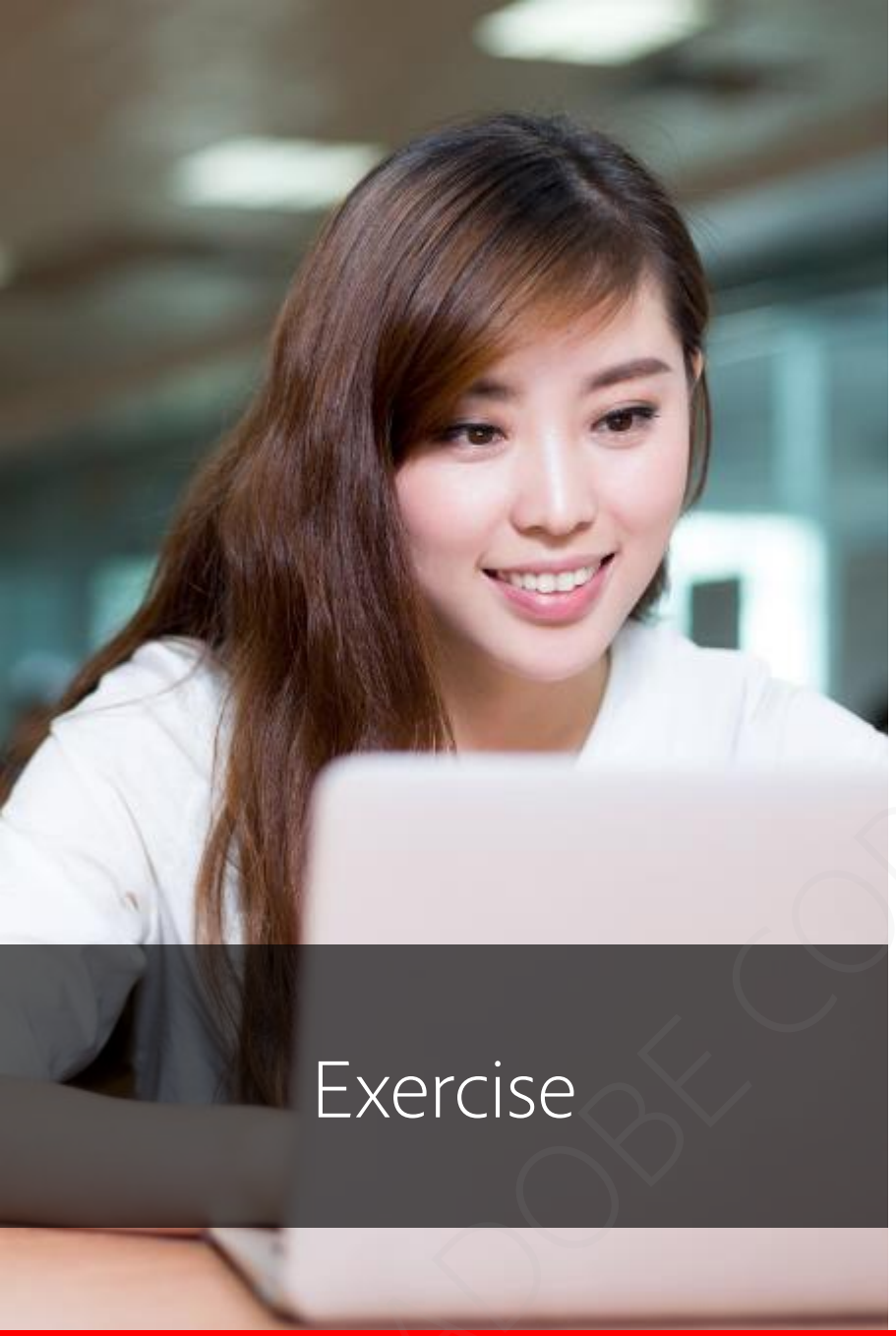
Extending a Core Component logic (Cont.)

- data-sly-resource can render original component
- Custom properties can be added if there is a getter

```
public String getTitle() {
    return coreTitle.getText();
}

public String getLinkURL() {
    return coreTitle.getLinkURL();
}

public String getSubtitle() {
    return subtitle;
}
```

Used in HTL

```
<sly data-sly-use.model="com.adobe.training.core.models.TitleWithSubtitle"
    data-sly-use.template="core/wcm/components/commons/v1/templates.html"
    data-sly-test.hasContent="${!model.empty}">

    <!--/* Inserts the core title component so we don't have to re-code the HTL */-->
    <sly data-sly-resource="${'.' @ resourceType = 'core/wcm/components/title/v2/title'}" ></sly>

    <!--/* Add a new custom property for our TitleWithSubtitle component */-->
    <h2 class="cmp-titlewithsubtitle__subtitle" data-sly-test="${model.subtitle}">${model.subtitle}</h2>
```



Exercise 1: Extend a core component

In this exercise you will learn how to properly extend a sling model that is the business logic for a core component. You will also upload and observe an extended component using your new Sling Model.

- Task 1: Install the Java code
- Task 2: Install the content package
- Task 3: Test the component

Breaking down a Sling Model

For Component logic:

- Must adapt the SlingHttpServletRequest.class
- Optional property injection strategy is the most flexible
- resourceType binds this model to the component

```
@Model(adaptables=SlingHttpServletRequest.class,
        adapters= {ComponentExporter.class},
        defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL,
        resourceType = {"training/components/content/stockplex"})
@Exporter(name="jackson", extensions = "json")
public class Stockplex implements ComponentExporter{
```

Breaking down a Sling Model (Cont.)

To Export via JSON:

- adapter must be the ComponentExporter.class
- @Exporter annotation must declare the type of export
  - Apache Jackson is supported OOTB, but it is extensible to other extensions
- Implement the ComponentExporter

```
@Model(adaptables=SlingHttpServletRequest.class,
        adapters= {ComponentExporter.class},
        defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL,
        resourceType = {"training/components/content/stockplex"})
@Exporter(name="jackson", extensions = "json")
public class Stockplex implements ComponentExporter{
```

Breaking down a Sling Model (Cont.)

Using Global Objects:

- Use @ScriptVariable annotation
- Name the object the exact same as the global object
  - For example: resource, currentPage, currentNode, request, currentStyle, and so forth
  - See Helpx > HTL Global Objects

```
@Exporter(name="jackson", extensions = "json")
public class Stockplex implements ComponentExporter{

    @ScriptVariable
    private Resource resource;
```

Breaking down a Sling Model (Cont.)

Using properties from a dialog:

- Use @ValueMapValue annotation
- Name the object the exact same as the “name” property from the dialog
- Return the dialog property with a getter

```
@Exporter(name="jackson", extensions = "json")
public class Stockplex implements ComponentExporter{

    @ValueMapValue
    private String symbol;

    public String getSymbol() {
        return symbol;
    }
```

Breaking down a Sling Model (Cont.)

Performing business logic:

- Use @PostConstruct annotation
- Can use any method name
- Called when the model is initiated in HTL
- Do not perform heavy calculation in getter methods

```
@Exporter(name="jackson", extensions = "json")
public class Stockplex implements ComponentExporter{

    @PostConstruct
    public void constructDataMap() {
        //perform any heavy coding/calculation
    }
```

Sling Models and HTL

- Call a Sling Model with data-sly-use
- Declare an object for the model (stockplex)
- Use the model getters to return properties to HTL (stockplex.summary)

```
<div data-sly-use.template="core/wcm/components/commons/v1/templates.html"
    data-sly-test.symbol="${properties.symbol}"
    data-sly-use.stockplex="com.adobe.training.core.models.Stockplex">
        <h3>${'Summary:' @ i18n} ${stockplex.summary}</h3>
</div>
```

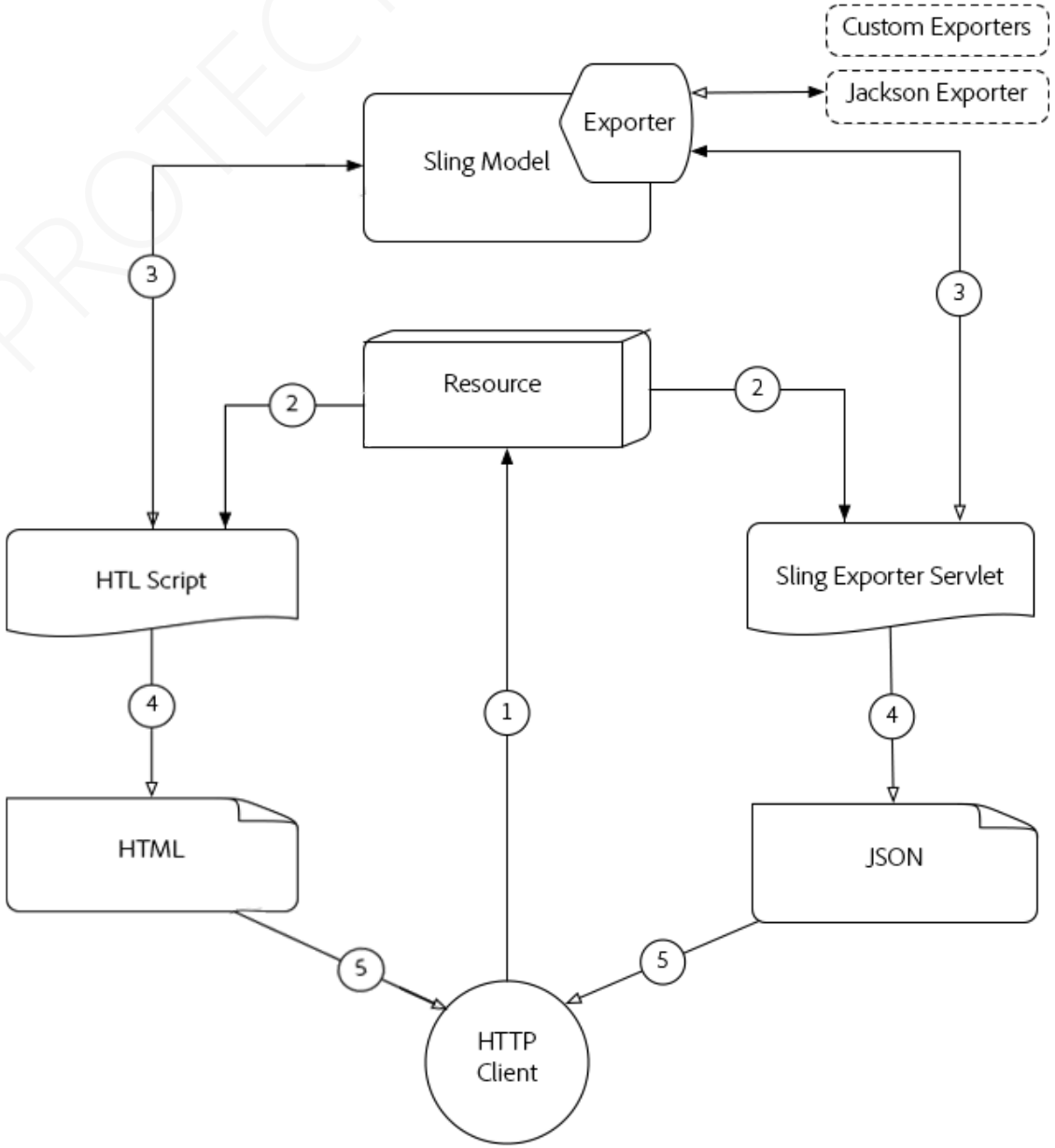
Sling Model Exporter

Enables new annotations to be added to Sling Models that define how the Model can be exported as JSON.

With Sling Model Exporter, you can obtain the same properties as a JSON.

Sling Model uses the Jackson Exporter.

The Sling Model Exporter can be used as a Web service or as a REST API.



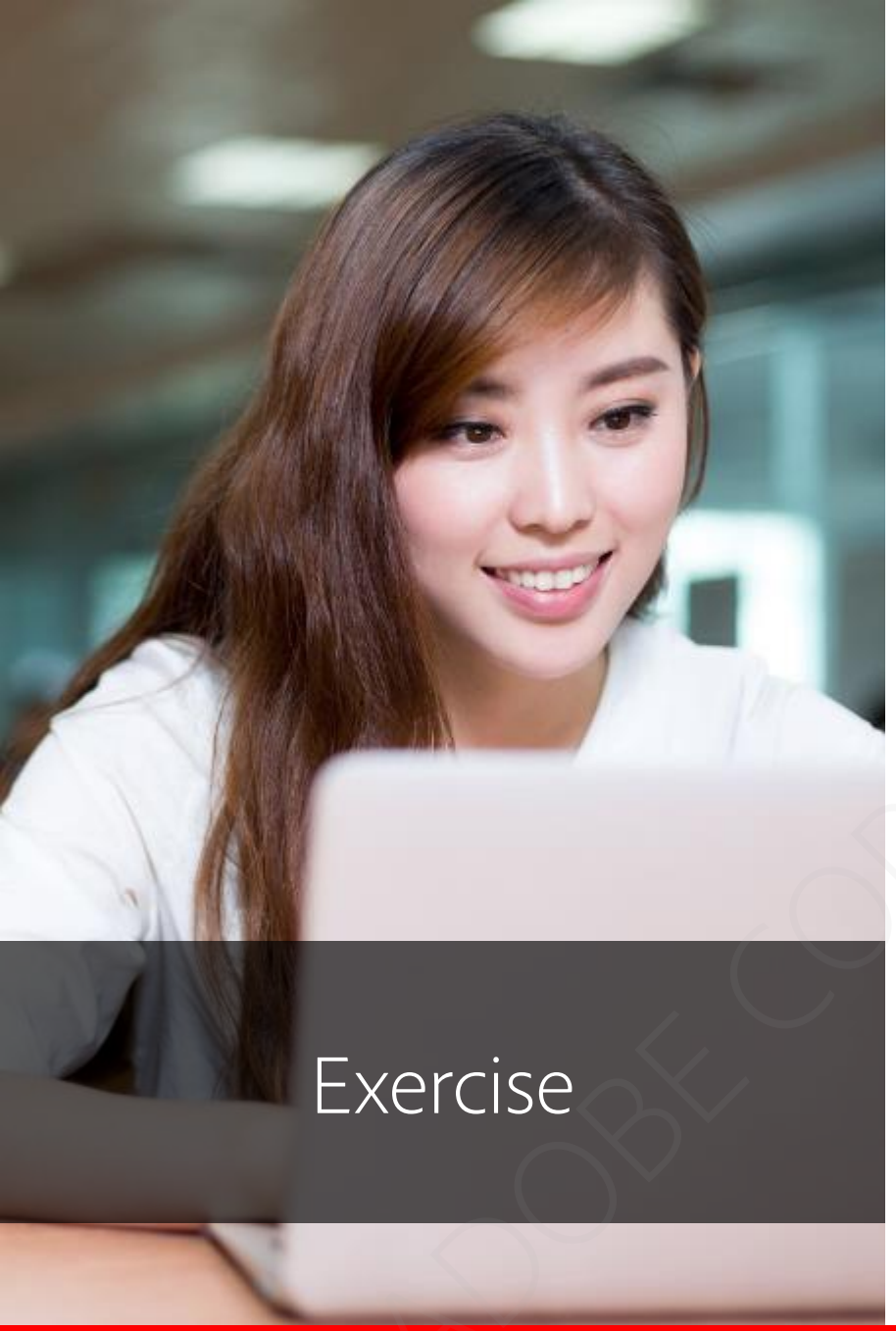
Exporting as JSON

http://localhost:4502/path/to/page/with/resource.model.json

http://localhost:4502/path/to/resource/stockplex.model.json

- Can export the entire page
- Can export a single resource
- Exports all getter methods of the Sling Model
- Exports the resourceType


```
{
    "symbol": "ADBE",
    "summary": "Summary for Adobe Systems",
    "showStockDetails": "true",
    "data": {
        "Company": "Adobe Systems Incorporated",
        "52 Week High": 233.1713,
        "Range High": 229.39,
        "Request Date": "Fri May 4, 2018",
        "Open Price": 223.89,
        "Volume": 1662128,
        "Sector": "Technology",
        "YTD % Change": 0.28593134496342154,
        "UpDown": 2.46,
        "Range Low": 223.39,
        "52 Week Low": 130.82,
        "Request Time": "04:00 PM EDT"
    },
    "currentPrice": "228.51",
    ":type": "training/components/content/stockplex"
}
```



Exercise 2: Create a custom Sling Model

In this exercise, you will upload and install a stockplex component. This will be the front-end HTL code for your component. You will then write a Sling Model to support the business logic.

- Task 1: Create a Sling Model
- Task 2: Install the stockplex component
- Task 3: Test the component

Key Takeaways

Key takeaways from this module:

- Sling Models
  - Used when you have a model object (as a Java class or interface).
  - Used with AEM easily, without creating your own adapters.
- Sling Models with AEM components
  - Allows the content to be exported to JSON
- Sling Model Exporter
  - Allows new annotations to be added to Sling Models