Using services and servlets

Agenda:

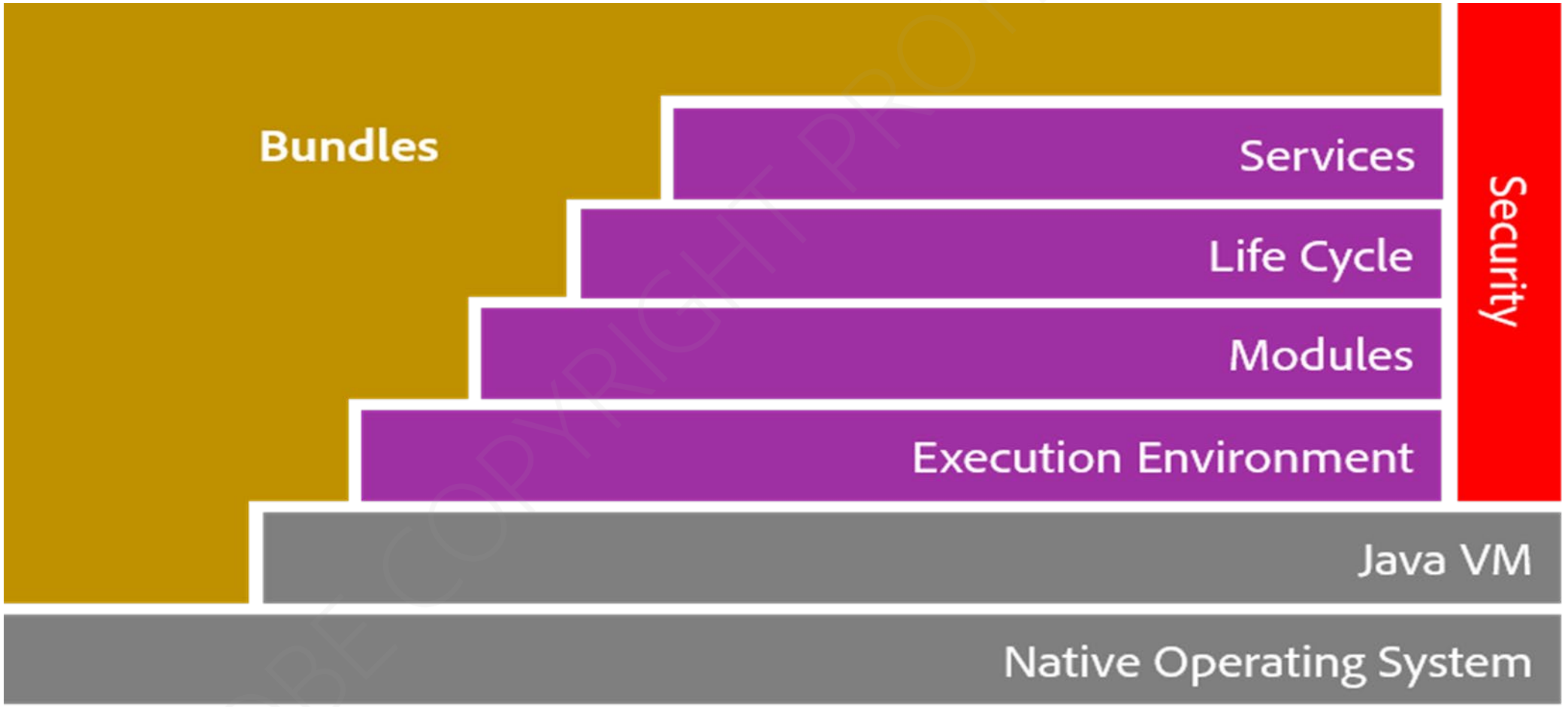
- Describe OSGi architecture
- Define OSGi annotations
- Implement OSGi configurations
- Describe Apache Sling
- Describe the Sling resolution process
- Create Sling servlets

OSGi

- A fundamental layer in the Adobe Experience Manager stack.
  - Used to control the composite bundles of Adobe Experience Manager and their configurations.
  - Enables applications to be created from smaller, reusable, collaborative components.

- An OSGi application is a collection of bundles that interact using service interfaces:
  - Each bundle contains one or many OSGi Components.
  - Bundles are independently developed and deployed.
  - Bundles can be installed, started, or stopped individually.
  - Bundles and their services may appear or disappear at any time.
  - Interdependencies between modules are automatically handled.

OSGi Architecture



Bundles

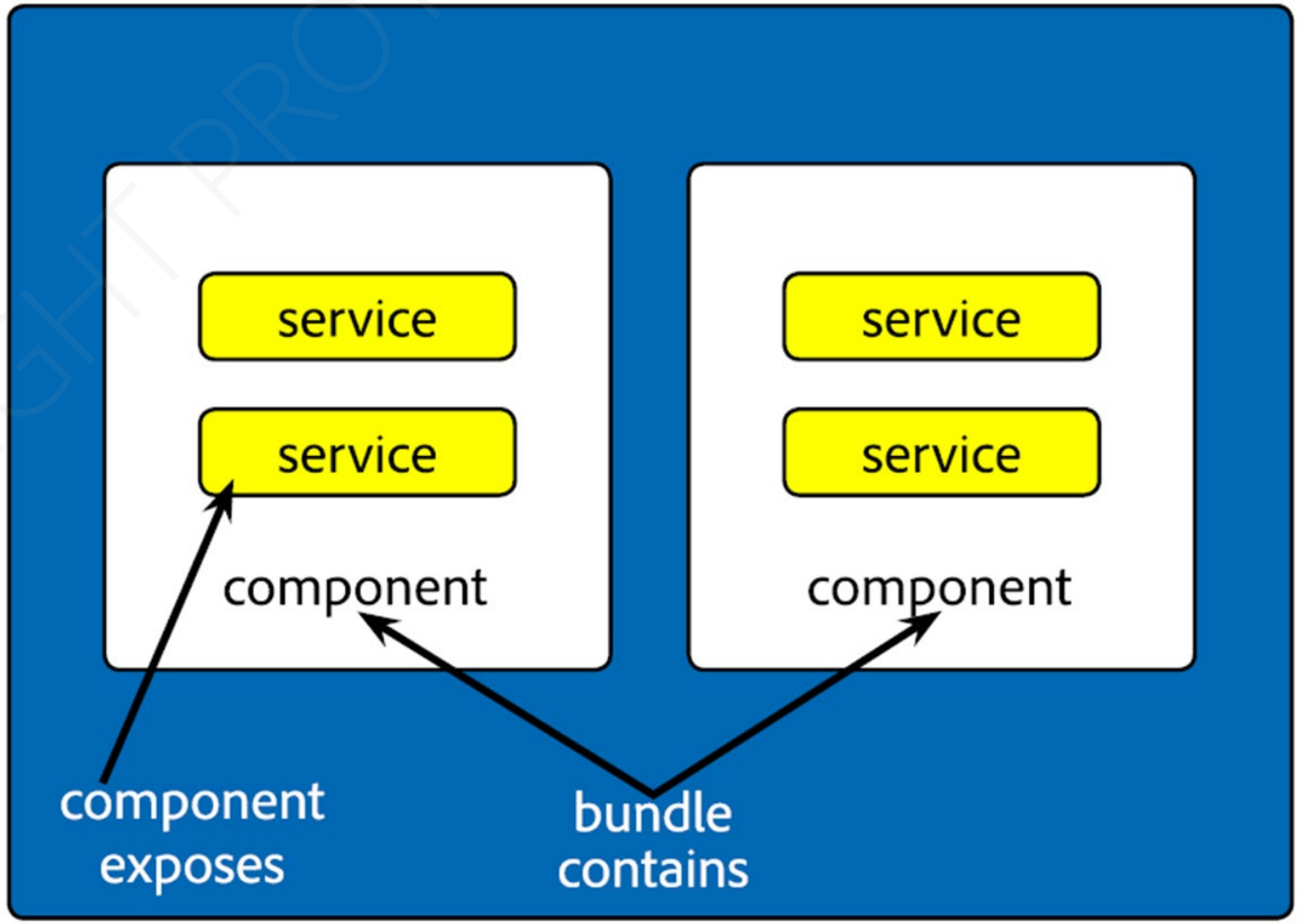
- A bundle is a JAR file, with additional metadata added to the manifest. This metadata may include:
  - Bundle name(s)
  - Bundle version
  - The list of services imported and exported by this bundle
  - Optional information such as:
    - Minimum Java version required
    - Vendor
    - Copyright statement
    - Contact address, and so on

Bundles (Cont.)

- Bundles are loosely packaged.
- It includes package imports and exports with versions.
- Dependencies are independent from the bundle organization.
- “Someone” provides the self-describing package.
- OSGi provides error management for unresolved bundles.
- Modular thinking is required during application design.
- It requires proper metadata and consistent version management.

Bundles contain Components

- Components are Java classes
- Components can implement Services
- A Service is a Java interface

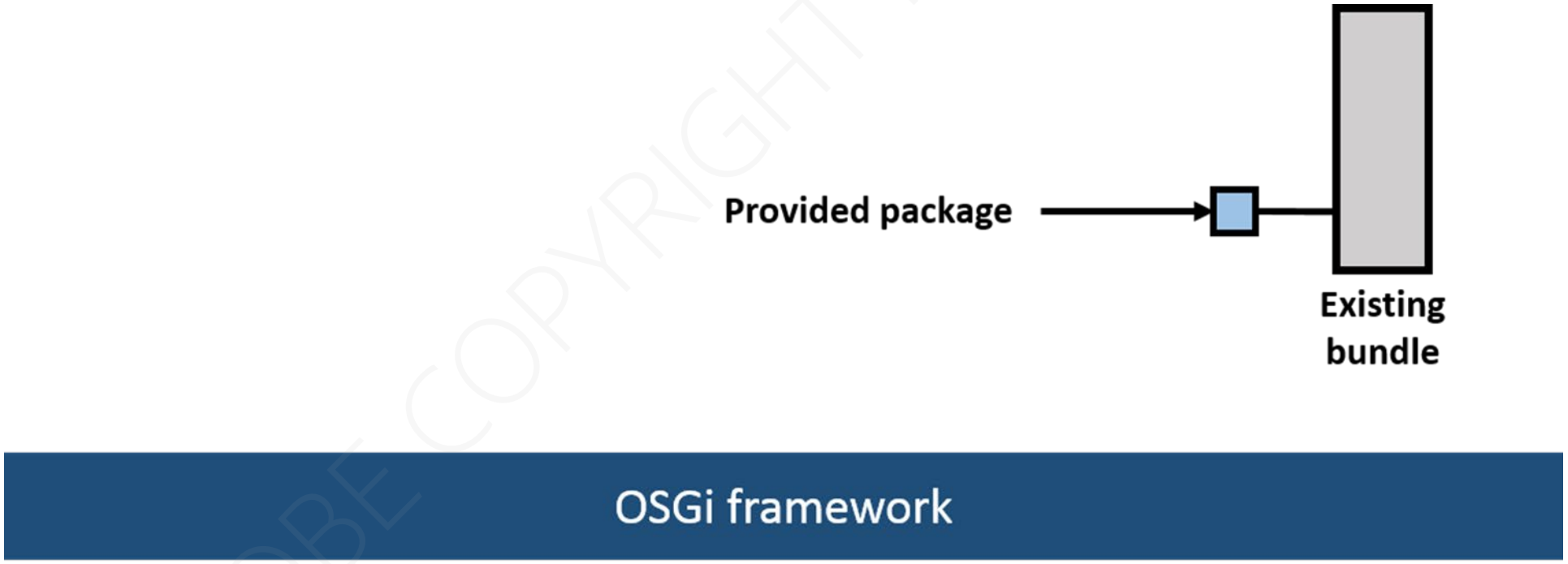


Components and Services

- Components are the main building blocks for OSGi applications. They are provided by a bundle.
- Bundles will contain and provide one or more components.
- A component is a:
  - Piece of software managed by an OSGi container
  - Java object created and managed by a container
- The OSGi container manages component configuration and the list of consumed services.
  - A component can provide a service.
  - A component can implement one or more Java interfaces as services.

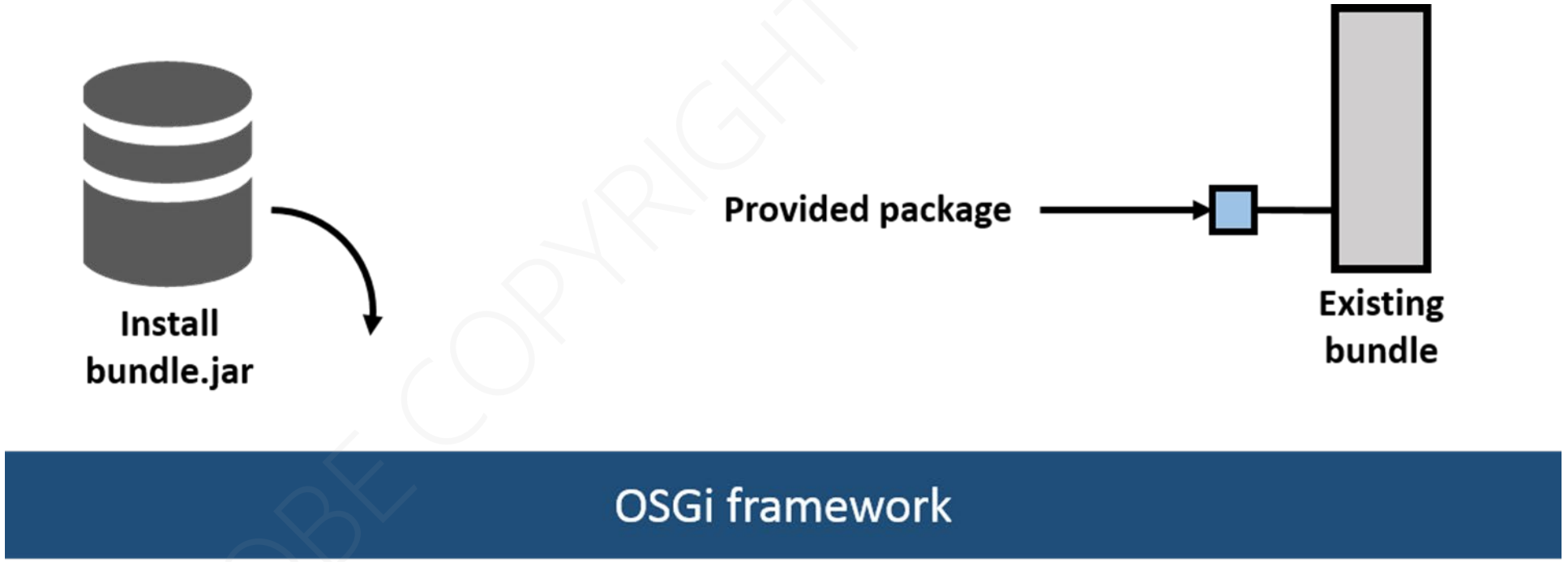
Dependency Management Resolution

- A bundle is present in the container.



Dependency Management Resolution

- When someone provides a new bundle, it is installed into the OSGi container.

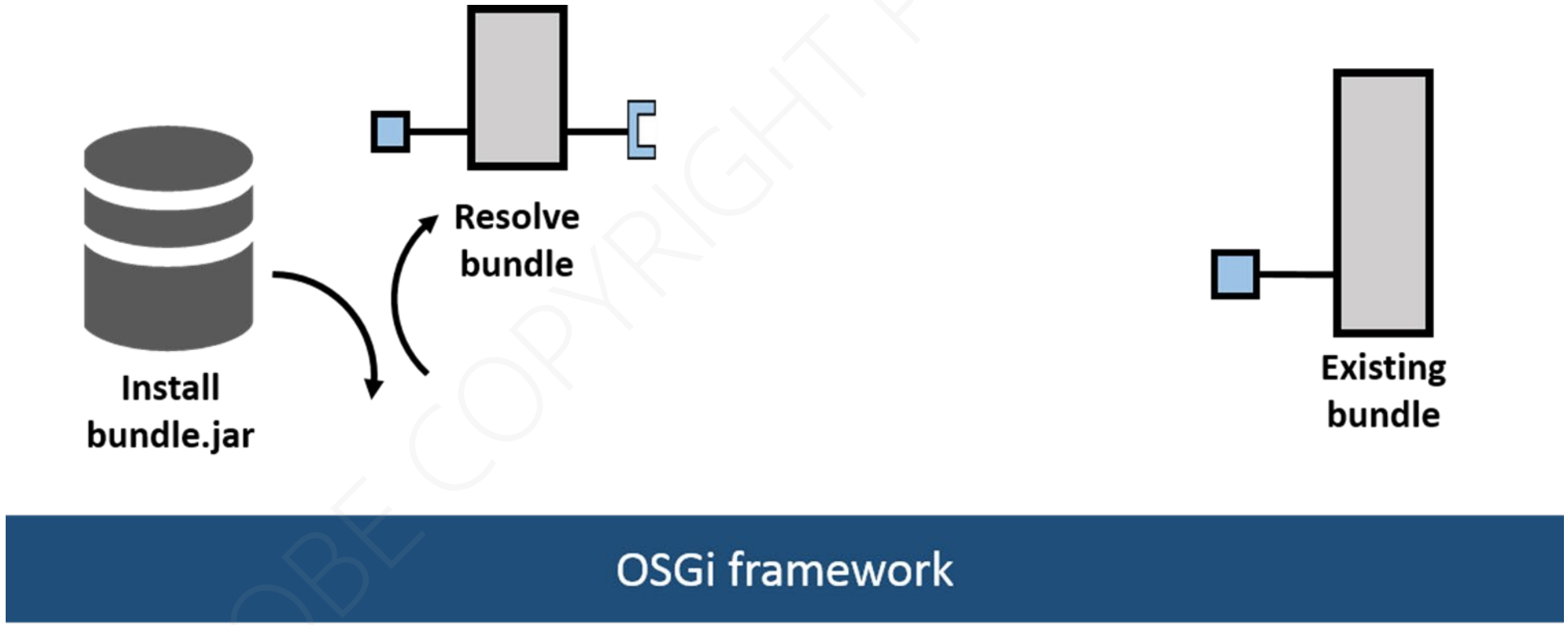


Dependency Management Resolution (Cont.)

- When someone provides a new bundle, it is installed into the OSGi container.

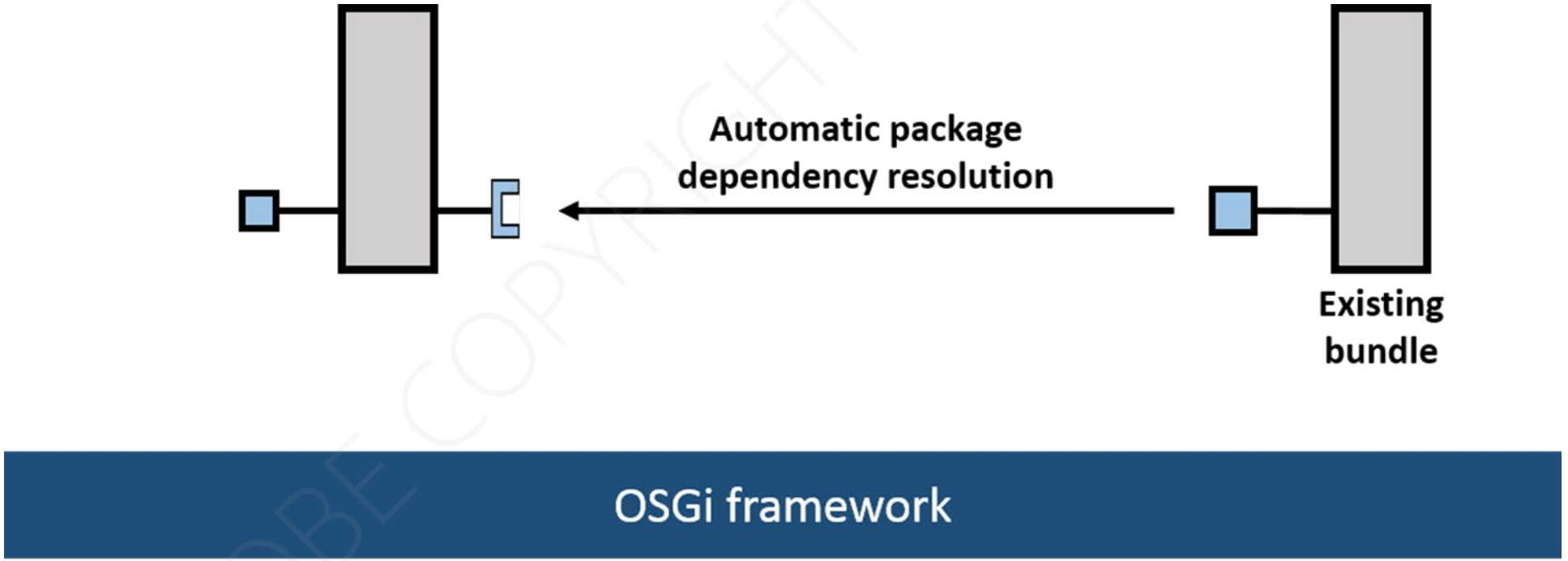
Dependency Management Resolution (Cont.)

- The OSGi container resolves the new bundle.



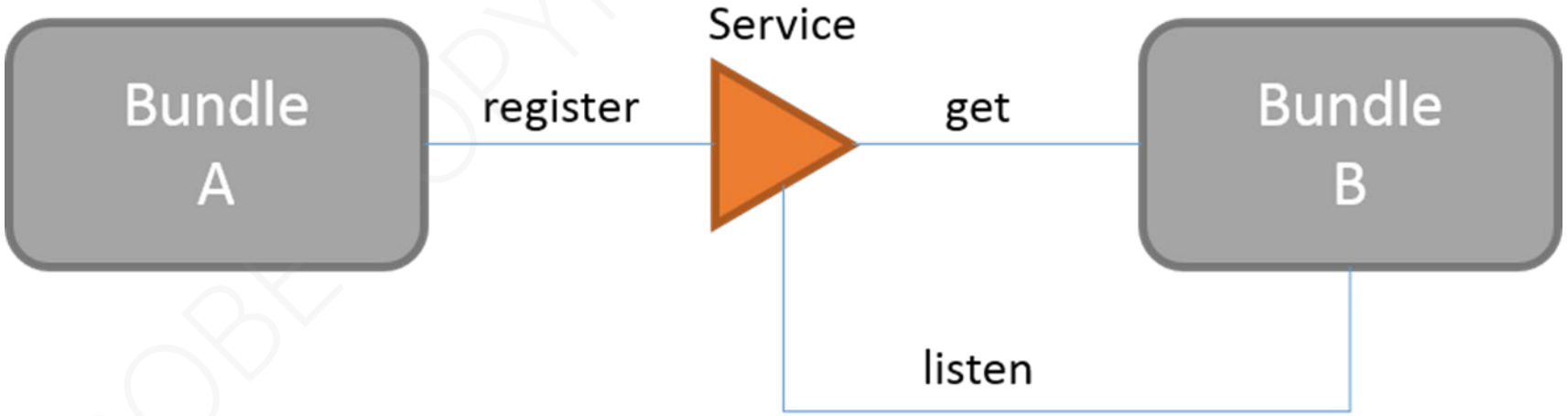
Dependency Management Resolution (Cont.)

- The container provides automatic dependency resolution.

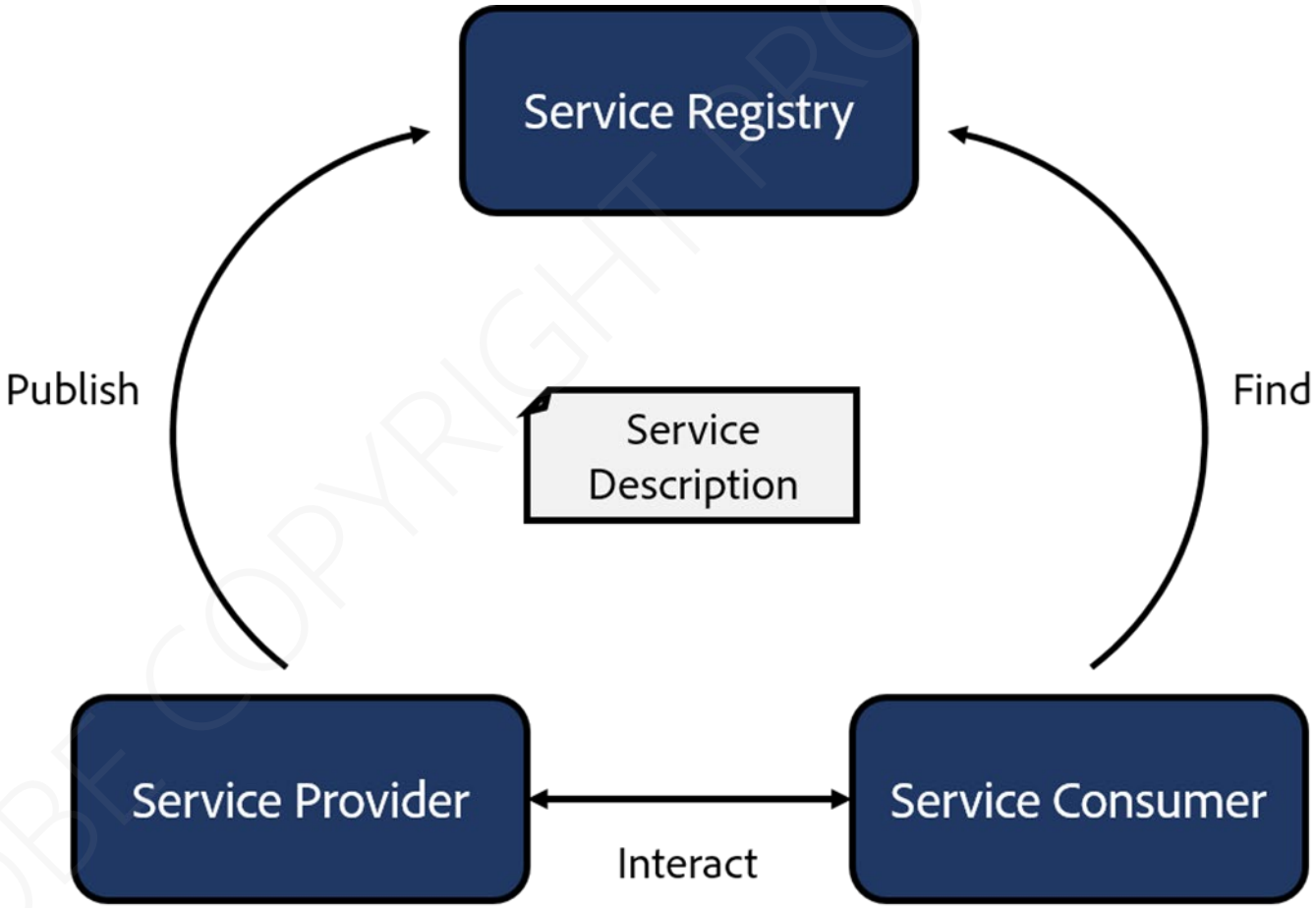


Bundles contain Services

- A bundle can:
  - create an object and register it with the OSGi service registry under one or more interfaces.
  - register a service, get a service, and listen for a service to appear or disappear.
- Any number of bundles can register the same service type, and any number of bundles can get the same service.

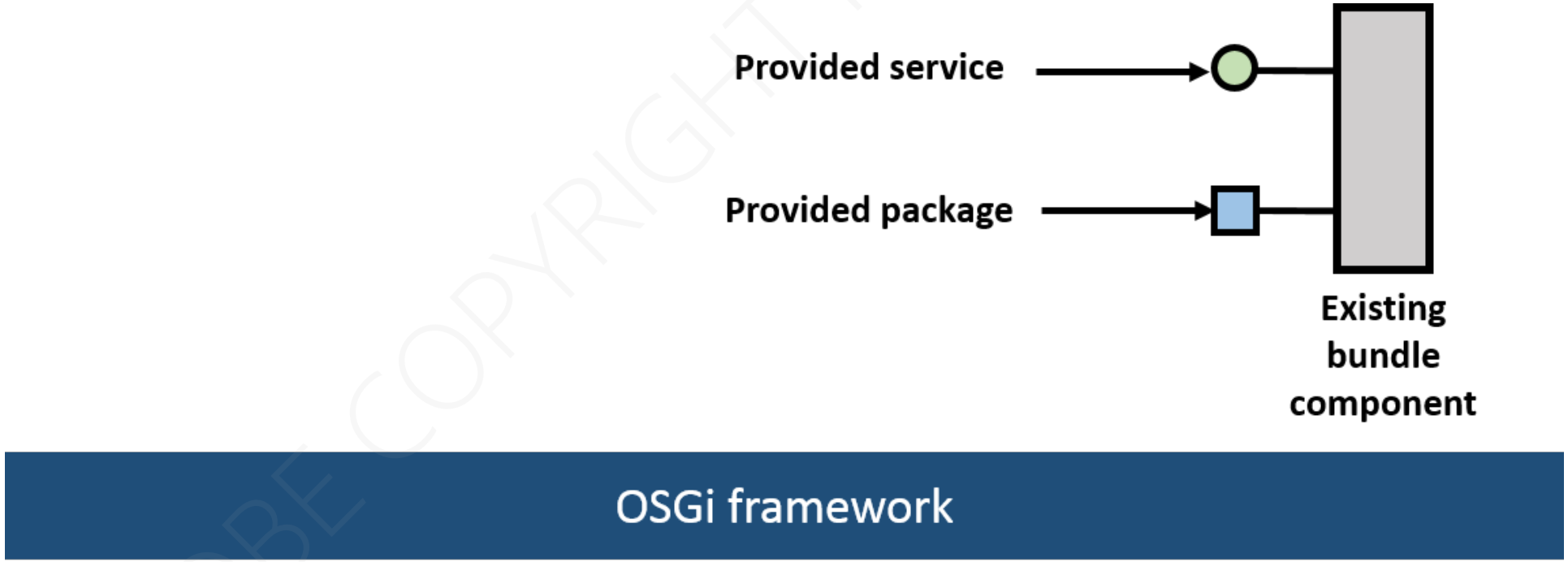


Service Registry Model



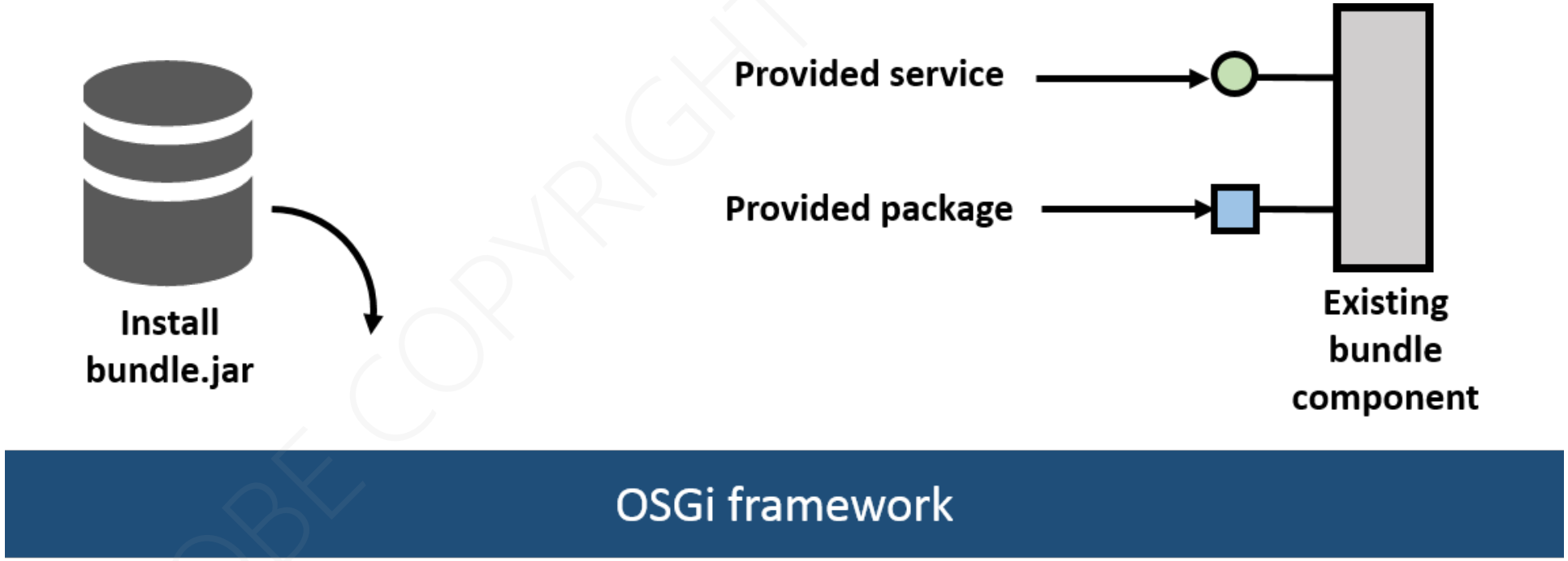
Dynamic Service Lookup

- Existing service



Dynamic Service Lookup (Cont.)

- Install new bundle

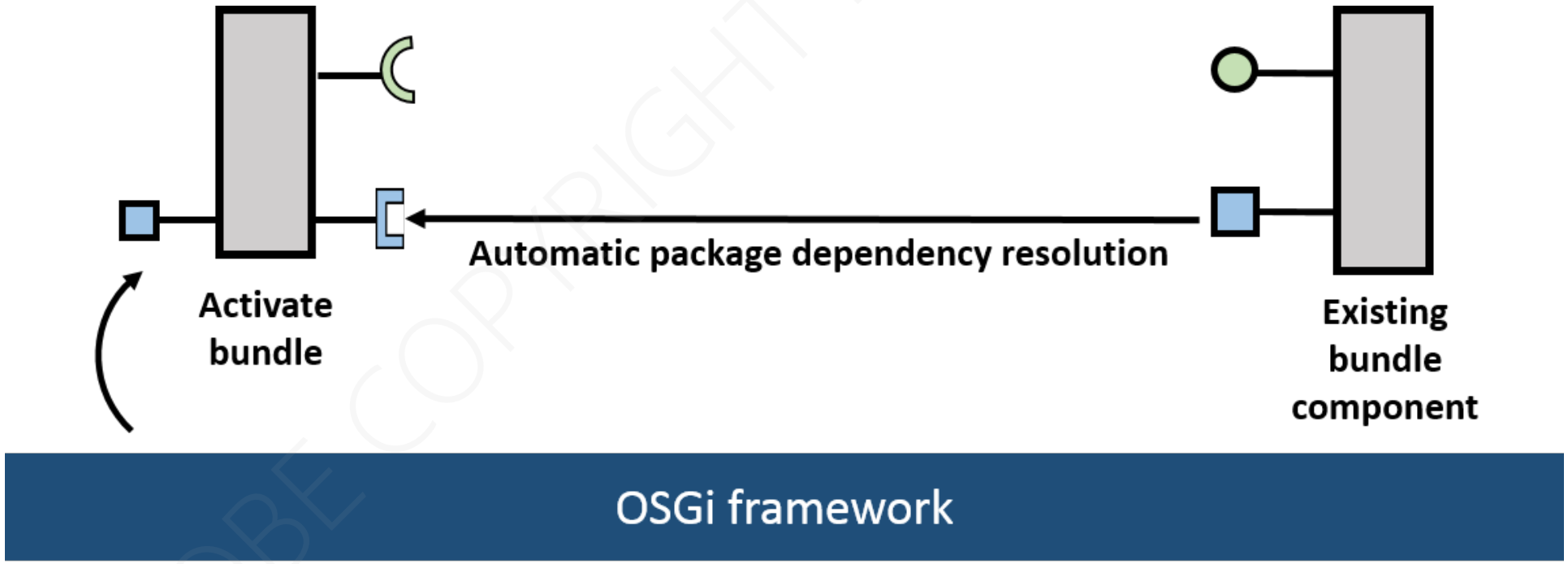


Dynamic Service Lookup (Cont.)

- Activate new bundle

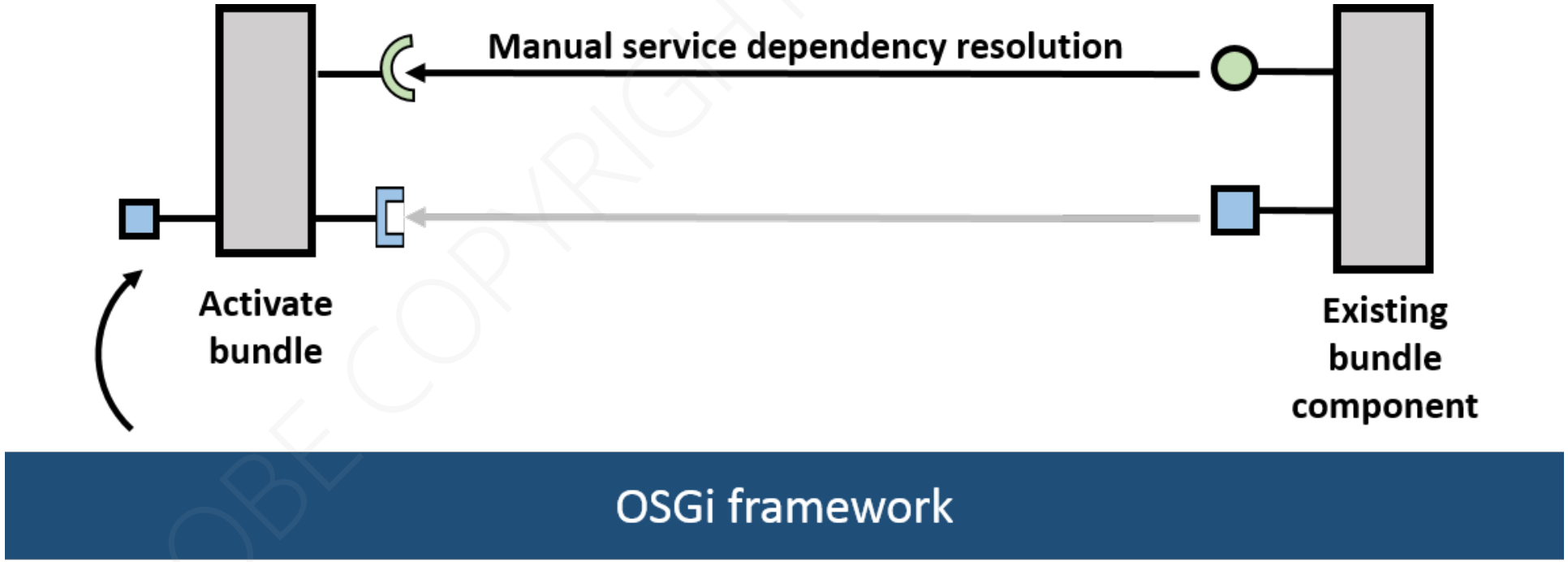
Dynamic Service Lookup (Cont.)

- Automatic resolution of package dependency



Dynamic Service Lookup (Cont.)

- Manual service dependency resolution

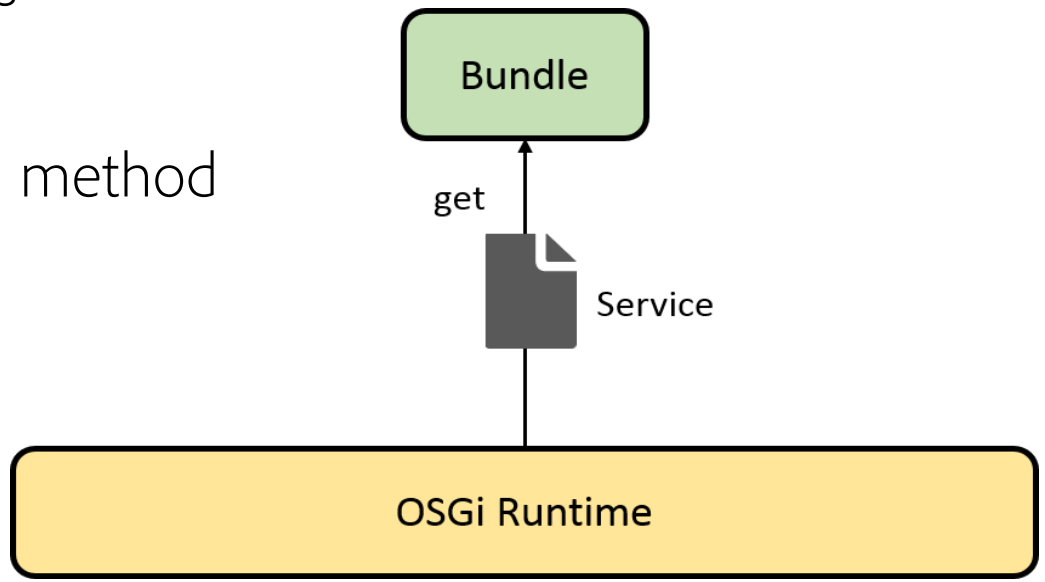


OSGi Declarative Services

- Declarative Services are a part of the OSGi container and simplify the creation of components that publish and/or reference OSGi Services.
- Features:
  - No need to write explicit code to publish or consume services.
  - Components that publish services are delayed, meaning that the service implementation class is not loaded or instantiated until the service is actually requested by a client.
  - Components have their own lifecycle, bounded by the lifecycle of the bundle in which they are defined.
  - Components can automatically receive configuration data from Configuration Admin.

OSGi Declarative Services

- Components are declared using XML configuration files contained in the respective bundle.
- Declarative Services resolve the bundle through the XML configuration files.
- Through the XML configuration, the container:
  - registers the bundle's services
  - keeps track of dependencies among the bundles
  - starts and stops services
  - invokes the optional activation and deactivation method
  - provides access to bundle configuration



OSGI Core Release annotations

- Recommended method for Declarative Services
- R6 Annotations are provided by the org.osgi.service.component.annotations
- The following annotations are supported:
  - Component
  - Activate
  - Deactivate
  - Modified
  - Reference

https://docs.osgi.org/specification/osgi.core/7.0.0/

@Component

- The @Component annotation allows the OSGi Declarative Services to register your component for you.
  - This is the only required annotation.
  - This annotation is used to declare the <component> element of the component declaration.

```
1.  package com.adobe.osgitraining.impl;
2.  import org.osgi.service.component.annontations.Component
3.  @Component
4.  public class MyComponent {
5.  }
```

@Component Modifiers

- Immediate – determines if the component is immediately activated on bundle start
- Service – types under which to register this component as a service
- Name – name of the component
- Property – property entries for this component
- Factory – specifies this component as a factory

Example:

```
1.  @Component (service=WorkflowProcess.class,
2.          name=”My Custom Workflow",
3.      property={“description=Workflow process to set approval status”,
4.          “process.label=Approval Status Writer”})
```

@Activate, @Deactivate, and @Modified

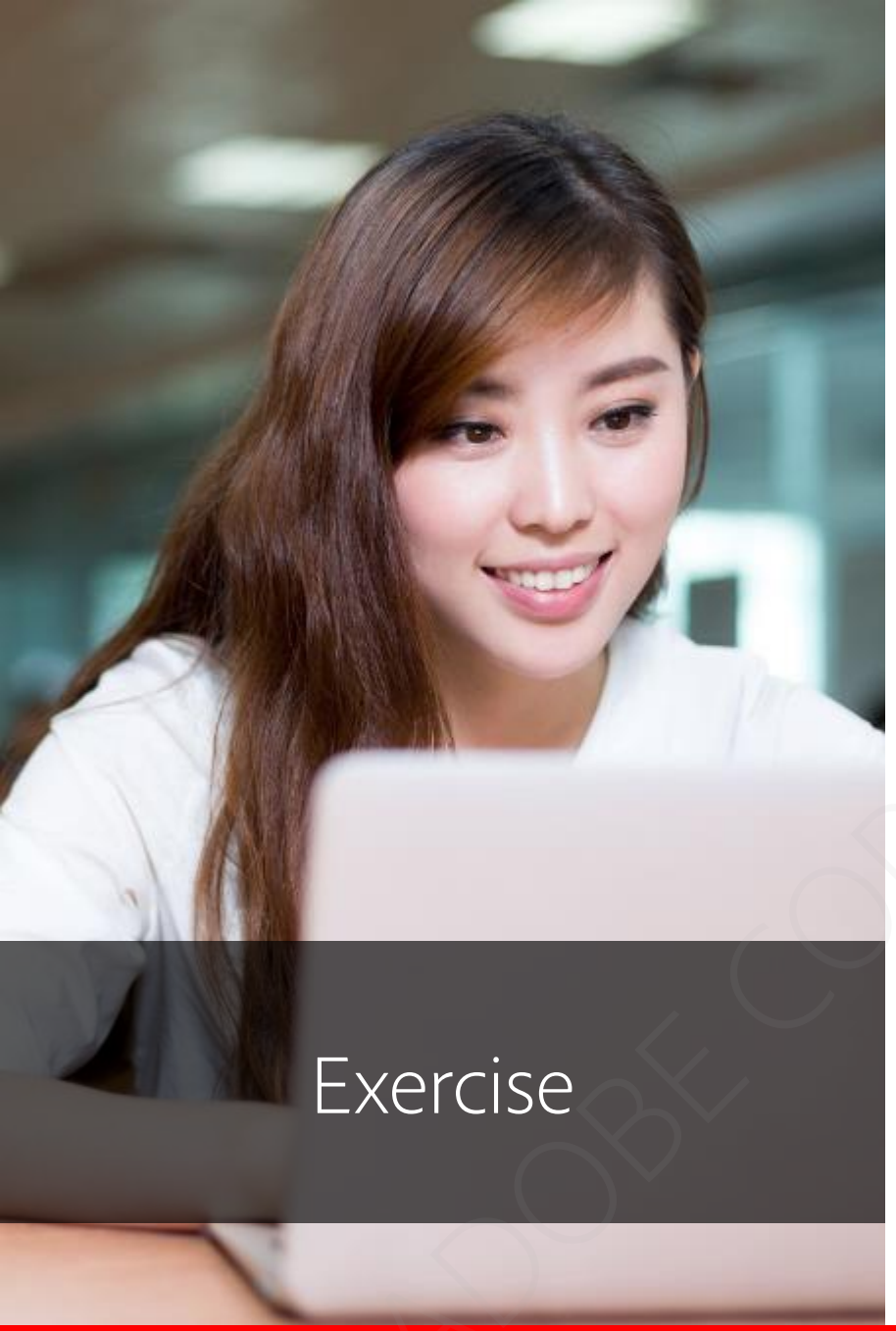
- Start, stop, and modified. These annotations and methods allow us to configure the components based upon input to the method.

```
1.  package com.adobe.osgitraining.impl;
2.  import org.osgi.service.component.annontations.Component;
3.  import org.osgi.service.component.annontations.Activate;
4.  import org.osgi.service.component.annontations.Deactivate;
5.  @Component
6.  public class MyComponent {
7.      @Activate
8.      protected void activate(configuration config) { // do something }
9.      @Deactivate
10.     protected void deactivate(Configuration config) { // do something }
11. }
```

@Reference

- The @Reference annotation defines references to other services in OSGi
- This annotation:
  - is used to declare the <reference> elements of the component declaration.
  - may be declared on a Class level or on any Java field to which it might apply. Depending on where the annotation is declared, the parameters may have different default values.

```
1.  package com.adobe.osgitraining.impl;
2.  import org.osgi.service.component.annontations.Component;
3.  import org.osgi.service.component.annontations.Reference;
4.  import org.apache.sling.api.resource.*;
5.  @Component
6.  public class MyComponent {
7.     @Reference
8.     Private ResourceResolverFactory resourceFactory;
9.  }
```



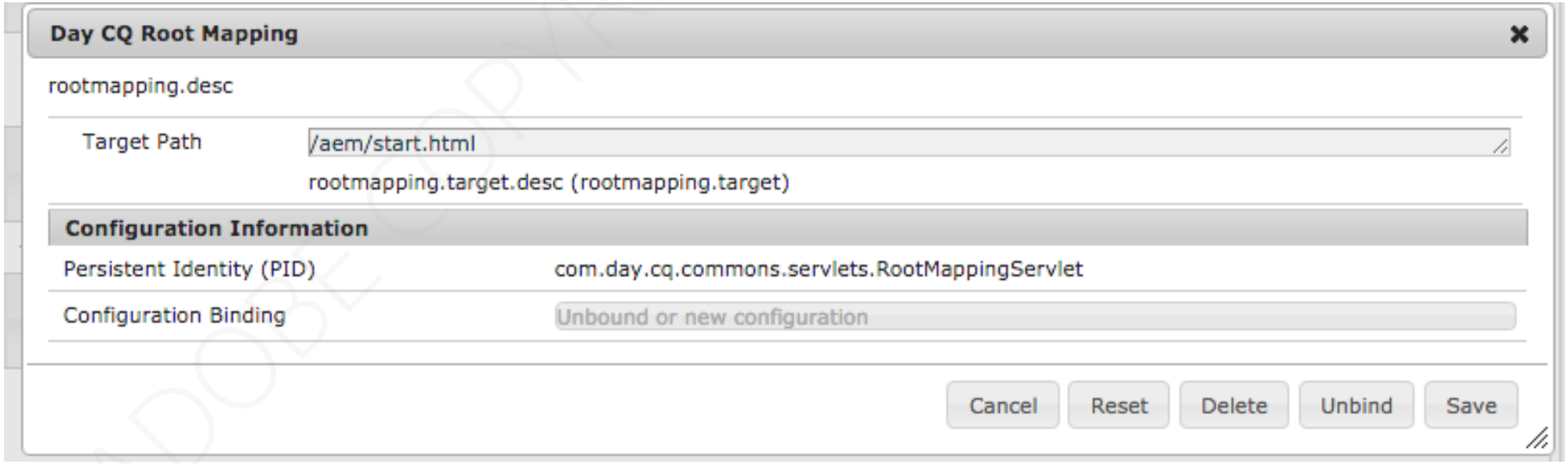
Exercise 1: Create and use a custom service

In this exercise, you will create and use a custom service.

- Task 1: Create a service and an implementation
- Task 2: Deploy the bundle and expose the service in HTL
- Task 3: Test the service

OSGi configuration properties

- Components can have configuration properties
- Allow a component to be configurable per environment by an admin
- Access via:
  - Web Console > OSGi > Configuration
  - http://localhost:4502/system/console/configMgr



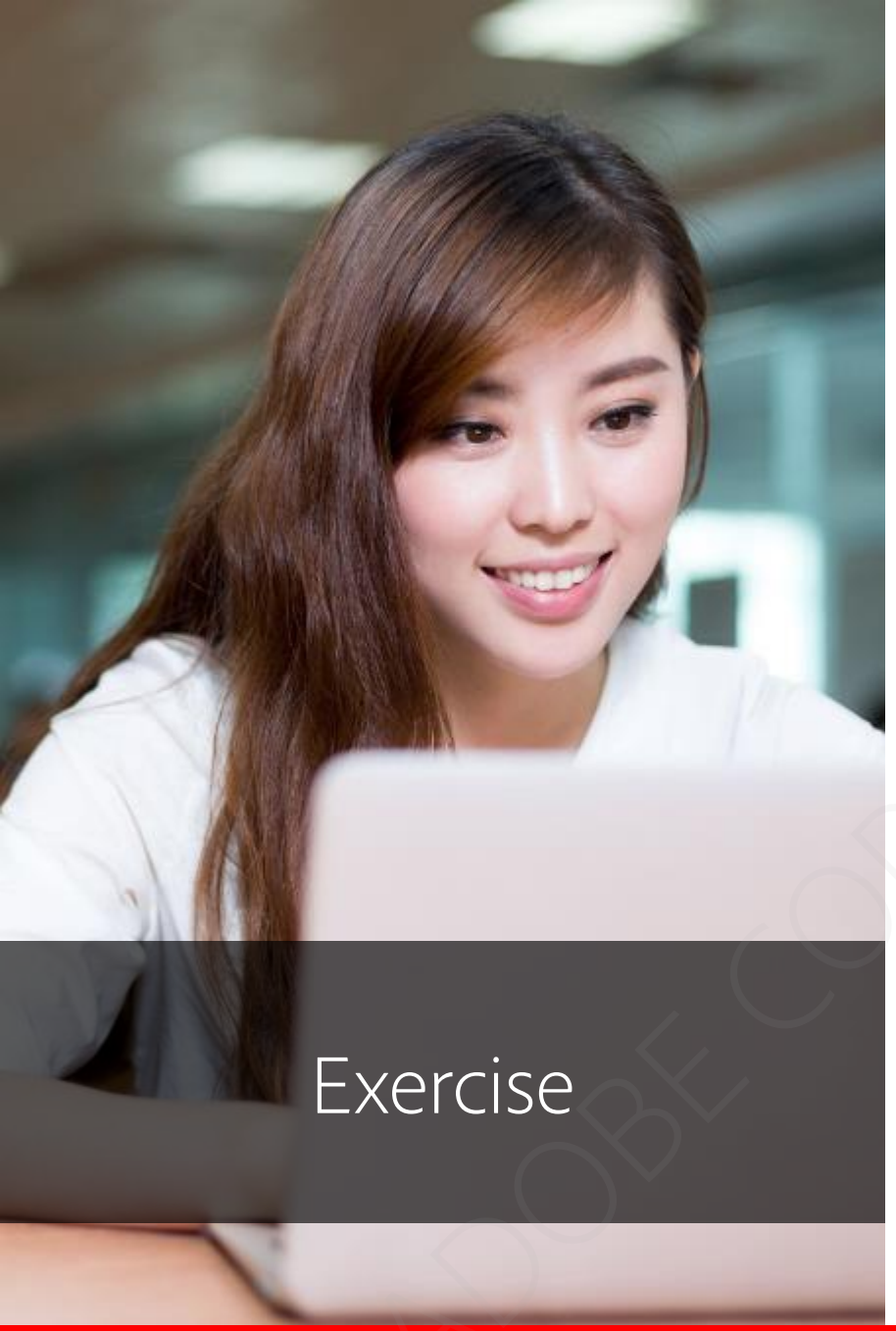
Configuration Service interface

```
1.  import org.osgi.service.metatype.annotations.ObjectClassDefinition;
2.  import org.osgi.service.metatype.annotations.AttributeDefinition;
3.  import org.osgi.service.metatype.annotations.AttributeType;
4.  @ObjectClassDefinition(name = My Configuration Service)
5.  public @interface MyConfigInterface{
6.
7.      @AttributeDefinition(
8.          name="Enter a String",
9.          description="Your description",
10.         type=AttributeType.STRING
11.      )
12.      String myconfig_property() default "";
13. }
```

Use a Configuration Interface

- @Designate is in package org.osgi.service.metatype.annotations
- The @Designate annotation defines the interface for the OSGi configuration interface
  - Can set it as a configuration factory

```
1.  import org.osgi.service.component.annontations.Component;
2.  import org.osgi.service.metatype.annotations.Designate;
3.  @Component
4.  @Designate(ocd=MyConfigInterface.class, factory=true)
5.  public class MyComponent {
6.     private myString
7.    @Activate
8.     protected void activate(MyConfigInterface config) {
9.             myString = config.myconfig_property()
10.    }
11. }
```



Exercise 2: Code OSGi configurations

In this exercise, you will code the custom OSGi configurations for an OSGi component. This will allow an administrator to configure the component as we did earlier. These configurations can then be further targeted by using run modes.

- Task 1: Create a service with configurations
- Task 2: Create JSON configuration file
- Task3: Install via Command Line
- Task 4: Test the service

Understand Sling Resolution

Sling is resource-oriented. Resources are

- Usually mapped to a JCR node, but can also be mapped to a file system or database
- Maintained in the form of a virtual tree

The following items are the common properties that a resource can have:

- Path
- Name
- Resource Type

URL ➔ virtual resource tree ➔ JCR node

Resource Resolver Object

Resolves incoming requests to actual or virtual resources

Abstracts:

- The path resolution
- Access to the persistence layer(s)

Commonly used in code to

- Get resources for CRUD operations

```
216     Resource trade = resourceResolver.getResource(tradePath);
217     //Test if stock import folder exists, otherwise create it
218     Resource stockFolder = ResourceUtil.getOrCreateResource(resourceResolver, stockPath, "", "", false);
```

- Get application managers for AEM object manipulation

```
41  protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
42      //Use pageManagerFactory to get page manager from the request.resourceResolver
43      PageManager pm = pageManagerFactory.getPageManager(request.getResourceResolver());
44      //Use the PageManager to find the containing page of the resource (component)
45      Page curPage = pm.getContainingPage(request.getResource());
```

Resource Resolver | Retrieving the Object

Use @Reference annotation and a service user

```
95    @Reference
96    private ResourceResolverFactory resourceResolverFactory;
184   //Get the service user (training-user) that belongs to the training.core:training subservice
185   Map<String, Object> serviceParams = new HashMap<>();
186   serviceParams.put(ResourceResolverFactory.SUBSERVICE, "training");
187   ResourceResolver resourceResolver = resourceResolverFactory.getServiceResourceResolver(serviceParams);
```

Retrieving from the request/response

```
41    protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
42        //Use pageManagerFactory to get page manager from the request.resourceResolver
43        PageManager pm = pageManagerFactory.getPageManager(request.getResourceResolver());
44        //Use the PageManager to find the containing page of the resource (component)
45        Page curPage = pm.getContainingPage(request.getResource());
```

Adapting a JCR Session or another Object

- Generally this should be avoided if the other method are achievable

Processing Requests

HTTP resources typically expose content nodes in the JCR

Content nodes need to be rendered in the browser

Rendering logic is based on

- Properties on the content node
- The HTTP method used to make the request
- Naming conventions within the URL

For every URL request, these steps are performed for rendering

- Decompose the URL.
- Search for a file indicated by the URL.
- Resolve the Resource.
- Resolve the rendering script/servlet.
- Create a rendering chain.
- Invoke a rendering chain.

Sling URL Decomposition

Example:

- URL: http://myhost/tools/spy.printable.a4.html/a/b?x=12

protocol	Host	Content path	Selector(s)	Extension		Suffix		Param(s)
http://	myhost	tools/spy	.printable.a4	html	/	a/b	?	x=12

Sling Servlets

Java servlets process HTTP requests in a RESTful way

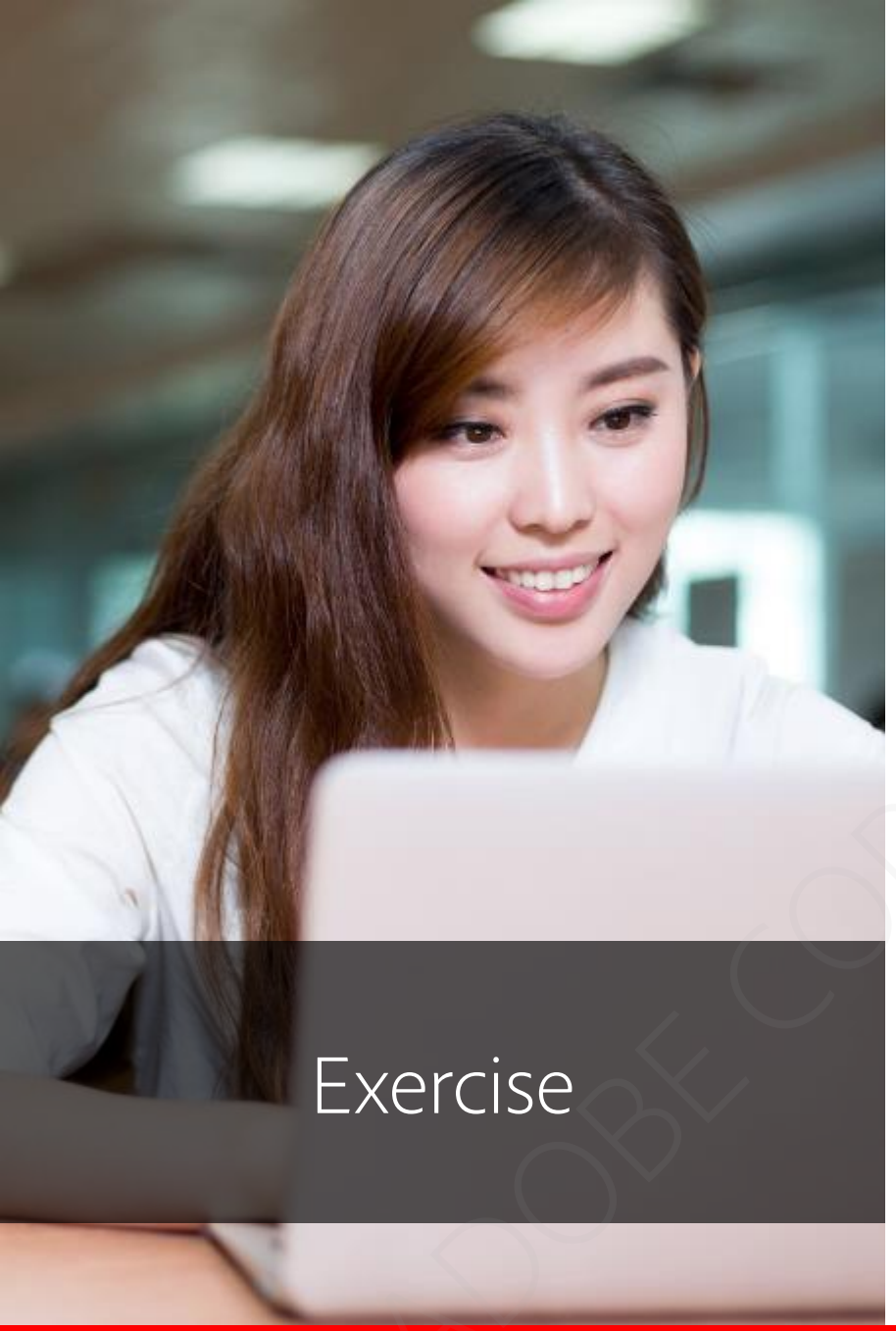
Resources can resolve to a component script or a Sling Servlet

- Servlets take precedence over component scripts

Servlets can be triggered by

- resourceType
- Method
- Selector
- Extension

```
27  @Component(service = { Servlet.class })
28  @SlingServletResourceTypes(
29          resourceTypes=TitleSlingServlet.RESOURCE_TYPE,
30          selectors="foobar",
31          extensions="html")
32  public class TitleSlingServlet extends SlingSafeMethodsServlet {
33      protected static final String RESOURCE_TYPE = "training/components/title";
34
35      @Override
36      protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
```

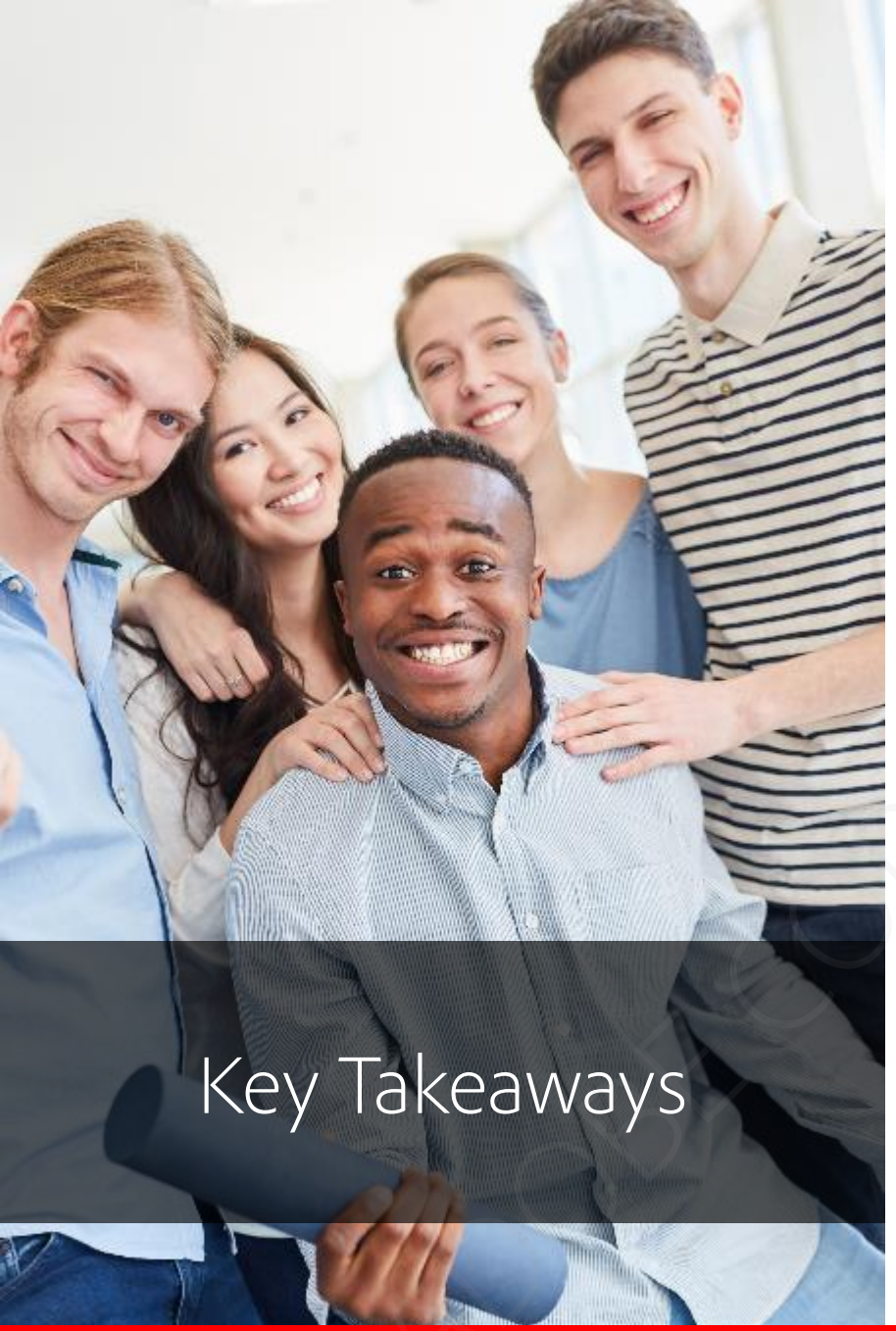


Exercise 3: Create a Sling servlet

In this exercise, you will write a servlet that serves only doGet requests with a specific selector or resource type. The response shall contain the JCR repository properties as JSON.

- Task 1: Create a servlet with a resource type
- Task 2: Use a selector to trigger the servlet

Q&A session



Key Takeaways

Summary:

- OSGi:
  - Open Services Gateway Initiative (OSGi) is one of the fundamental layers in the AEM technology stack.
  - An OSGi application is a collection of bundles that interact using service interfaces.
- Bundles
  - A bundle is a JAR file, with additional metadata added to the manifest.
- Components
  - Components are the main building blocks for OSGi applications. They are provided by a bundle.
  - Bundles will contain and provide one or more components.