# webMethods
# Integration Workshop – Day 9



# Recap – Day8

## Learnt So Far  (Recap & assessment)

- Basic concepts of Web services
- Producers & Consumers
- WSDL, SOAP, HTTP
- How to create web service connectors
- web service Descriptors
- How to test web services



# Introduction

- This course will help participants to understand the basics about Java services.
- Participants will get hands on basic creation java services in webMethods.
- This course will help in building the basic knowledge and hands on IData, handling documents, DataTypes in webMethods.



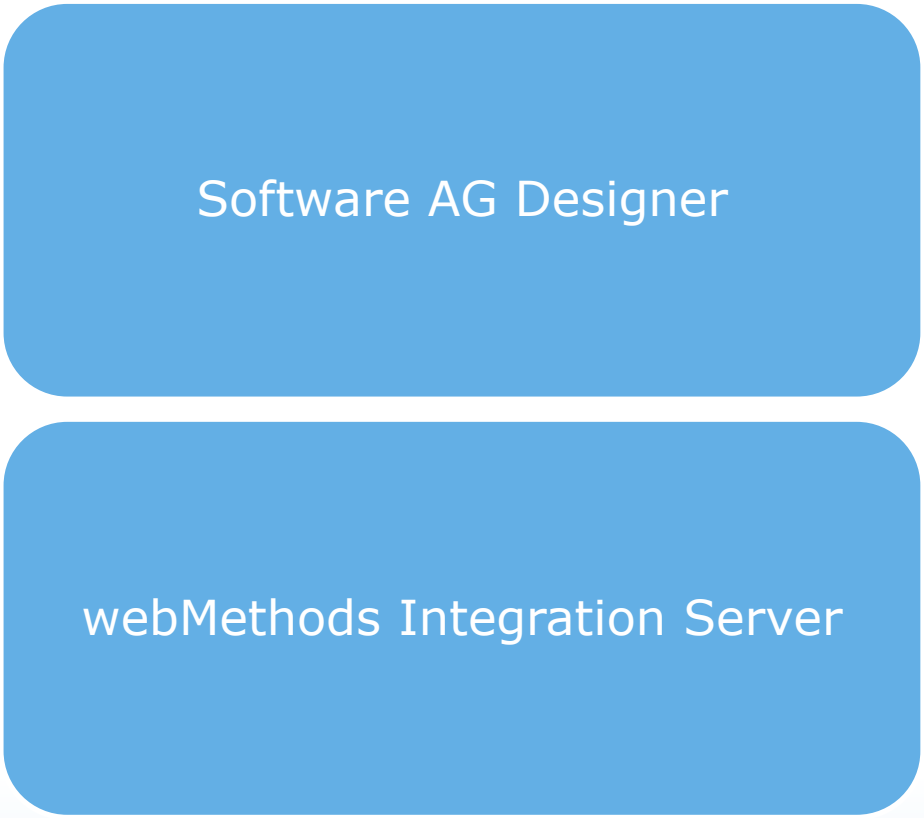
# Objectives

- Get hands on experience java service development
- Get knowledge on IData, Data handling in java services
- How to handle documents in Input
- How to handle different DataTypes, Code generation, logic.
- Get knowledge on Shared resources
- How java services are getting stored in Integration Server.



# Software versions

- This class focuses on the webMethods  suite



# Chapters

**Day 9**

- Java services - Pipeline
- IData & Service Signature
- Coding Java service
- Cursors
- Handle DataTypes & Input Output
- Integration Server Code Namespace



# What is Pipeline

- An IData Object
  - It is an IData object that contains an ordered collection of name/value pairs on which a service operates
  - It can contain any number of elements of any valid Java object, including other IData objects
  - Instantiated by server when the service is run
  - Dropped when service completes



# Input / Output & IData

- A Service takes one, and only one, input variable – a Pipeline.
  - The service extracts the actual input values it needs from the elements of an IData object
- A Service returns output by inserting it into the pipeline (IData)
- Multiple invokes within a service manipulate the same Pipeline IData object



# Service Signature

calculateOrders

```
package Training.Feb2014.javaservices;

import com.wm.data.*;

public final class calculateOrders_SVC

{

    /**
     * The primary method for the Java service
     *
     * @param pipeline
     *             The IData pipeline
     * @throws ServiceException
     */
    public static final void calculateOrders(IData p
            throws ServiceException {

        |
    }

    // --- <<IS-BEGIN-SHARED-SOURCE-AREA>> ---


    // --- <<IS-END-SHARED-SOURCE-AREA>> ---
}
```

**public**

can be invoked by other services

**static**

use a single (static) instance for all invokes – maximize throughput, minimize memory load

**final**

do not extend – improves performance

**void**

does not return an object

# Service Signature

calculateOrders

```
package Training.Feb2014.javaservices;

import com.wm.data.*;

public final class calculateOrders_SVC

{

    /**
     * The primary method for the Java service
     *
     * @param pipeline
     *            The IData pipeline
     * @throws ServiceException
     */
    public static final void calculateOrders(IData pipeline)
            throws ServiceException {

    }

    // --- <<IS-BEGIN-SHARED-SOURCE-AREA>> ---


    // --- <<IS-END-SHARED-SOURCE-AREA>> ---
}
```

**calculateOrders**

service name is the method name

**( IData pipeline )**

Takes IData object named “pipeline” as input

**throws ServiceException**

failure on invoke produces a ServiceException object

# IData Interface and Cursors

**IData operations come in two parts**

- **Position the cursor**
- **get/set the data**

- The IData interface looks like this:

```
public interface IData
{
        public IDataCursor getCursor();
        public IDataCursor getSharedCursor();
}
```



# IData in Action

**http post**

Data arguments:

input1 = Hello

input2 = World


**Code: Get Value**

| | |
|---|---|
| *input 1* | Hello |
| *input 2* | World |
| | |

String myInput = (String) Cursor.getValue();

# Cursor Types

| Cursor Type | Description |
| --- | --- |
| IDataCursor | the basic one, for all cursor manipulations. |
| IDataSharedCursor | for advanced cursor manipulation<br>*Not discussed in this class* |



# Using a Cursor

| IDataCursor | Useful Methods |
| --- | --- |
| Moving / Searching | first(), first(String key), next(), next(String key), previous(), previous(String key), last(), last(String key) |
| Get Data | getValue(), getKey() |
| Set Data | insertAfter(String key, value), insertBefore(String key, value), setValue(), setKey() |

# Example Code

```
public static final void stringCompare( IData pipeline ) throws
ServiceException
{

// pipeline
IDataCursor pipelineCursor = pipeline.getCursor();

String  string1 = IDataUtil.getString( pipelineCursor, "string1" );
String  string2 = IDataUtil.getString( pipelineCursor, "string2" );
pipelineCursor.destroy();

int result = string1.compareTo (string2);

// pipeline
IDataCursor pipelineCursor_1 = pipeline.getCursor();
IDataUtil.put( pipelineCursor_1, "result", "result" );
pipelineCursor_1.destroy();

    return;
```

create Cursor

get string1 from pipeline

get string2 from pipeline

do work

output results

# IData & Records

- Document objects are also IData objects 
- To manipulate them inside a service, use IDataFactory to instantiate an IData object
- IDataFactory is a factory class which has static create () methods.
- With this technique the user code is not directly aware of the concrete Implementation
- IData is an interface
- The IData implementation does not have a public constructor – you cannot use "new"

# Instantiating Documents in Code


**addrDoc IData object**

| | |
|---|---|
| ***companyName*** | **Jones, Ltd.** |
| ***addr1*** | **12 High St.** |
| ***city*** | **London** |

```
static IData addrDoc;
        static
        {
          addrDoc = IDataFactory.create();
          IDataCursor addrCursor = addrDoc.getCursor();
          addrCursor.last();
          addrCursor.insertAfter("companyName", "Jones, Ltd.");
          addrCursor.insertAfter("addr1", "12 High St.");
          addrCursor.insertAfter("city", "London");
        }
```

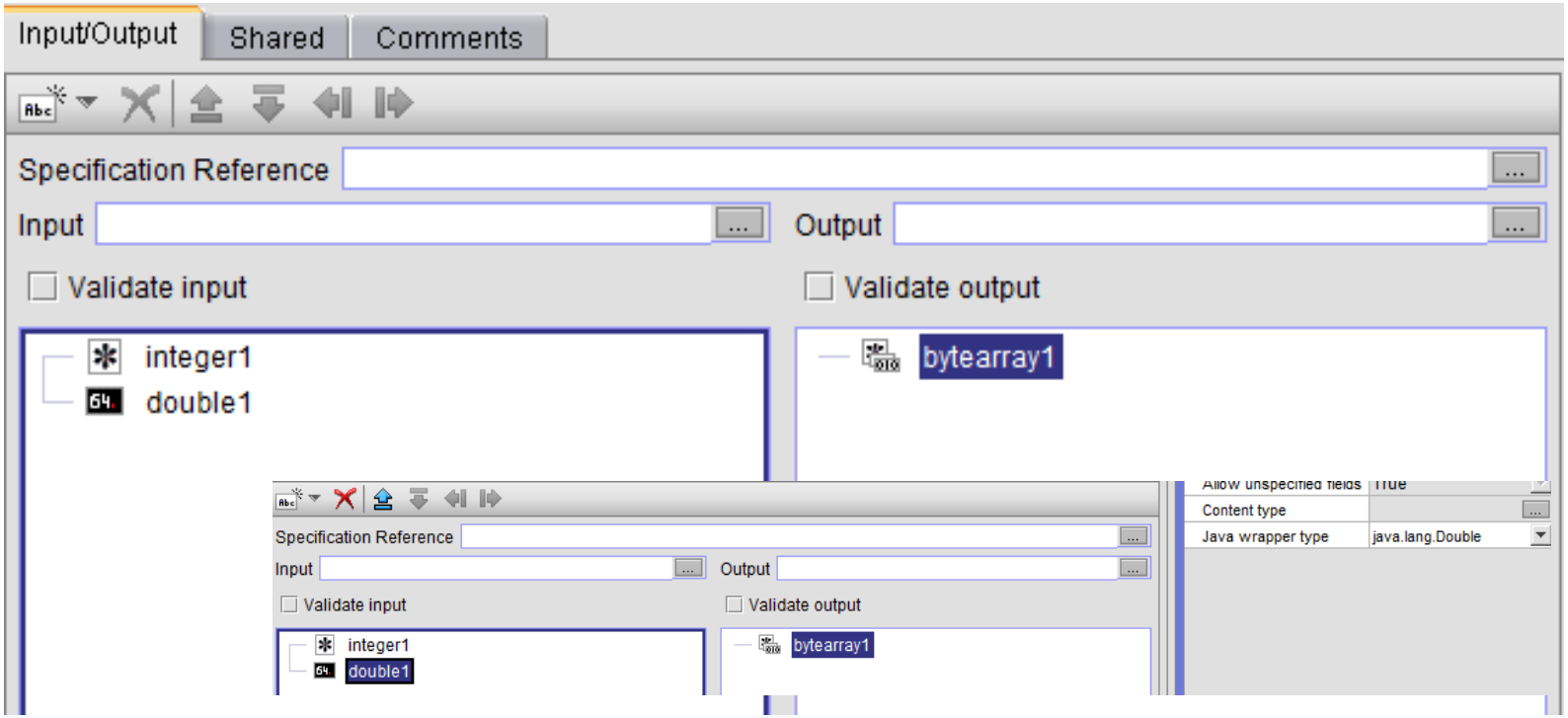
# Pipeline Object Types and Java

| | |
|---|---|
| String | java.lang.String |
| String List | java.lang.String [] – a string array |
| String Table | Two-dimensional String array |
| Document | Structure containing various data types IData |
| Document List | Same as Document, but an array IData [] |
| Object | Doesn't fit one of the already listed types<br>Any subclass of **java.lang.Object**<br>**(java.util.InputStream)** |
| Object List | Same as object, but array type.<br>e.g. **java.util.InputStream [ ]** |



# Java Object Data Types

- Declare Objects, Object Lists
- Under Properties, apply Constraints by selecting a Java class type






Cognizant

# Coding using Developer

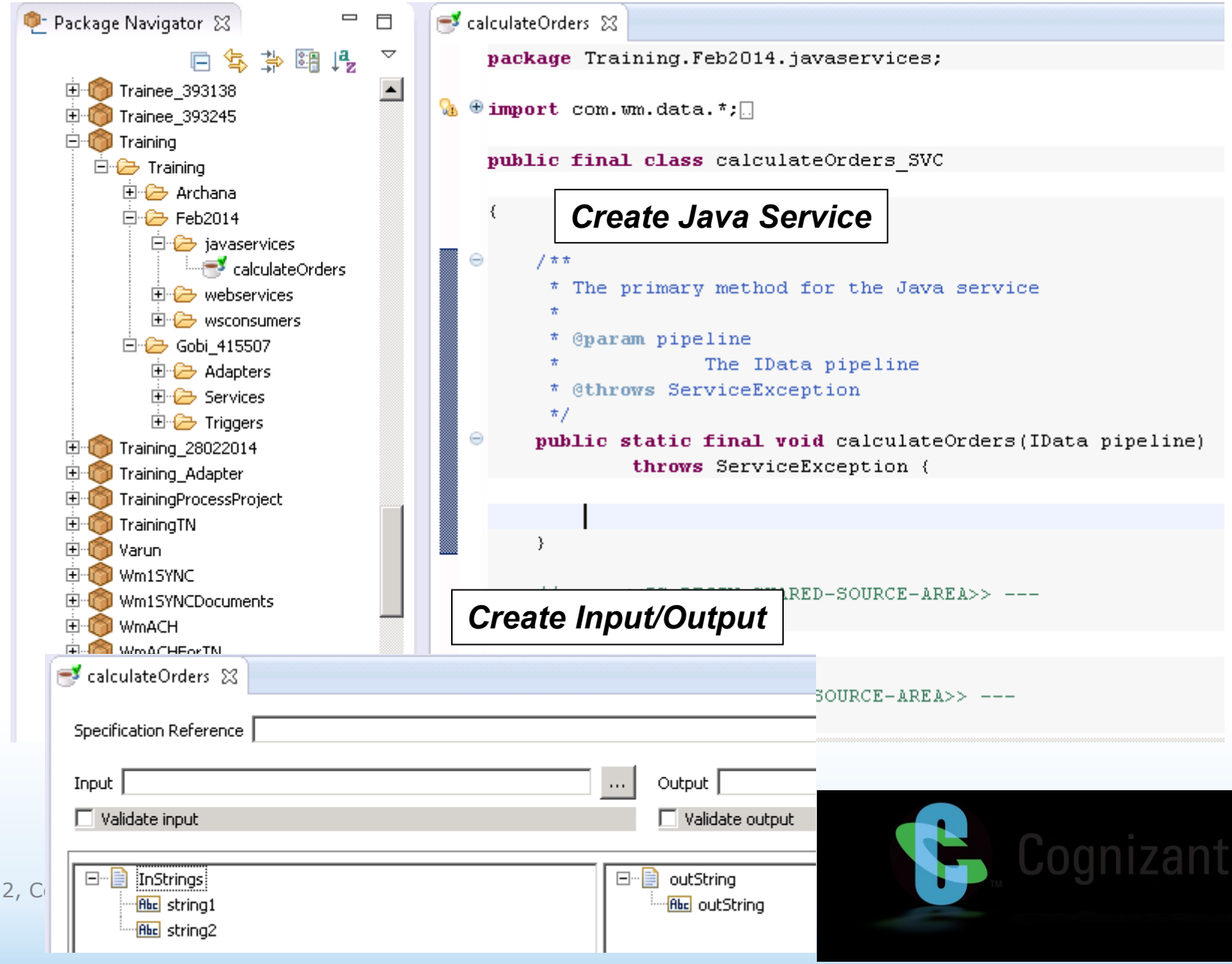
- Create an empty Java service
- Specify the input/output
  - Use the "Input/Output" tab in the Service Tab area
- (Optional) Generate code for implementing this service, and paste it in the source block
- Specify packages to be imported
  - Use the "Imports" section of the "Shared" tab
- Type your source code
  - Use the "Source" tab in the Service Tab area
- (Optional) Type code to be shared by all services in this folder
  - Use the "Source" section of the "Shared" tab

# Third Party Jars

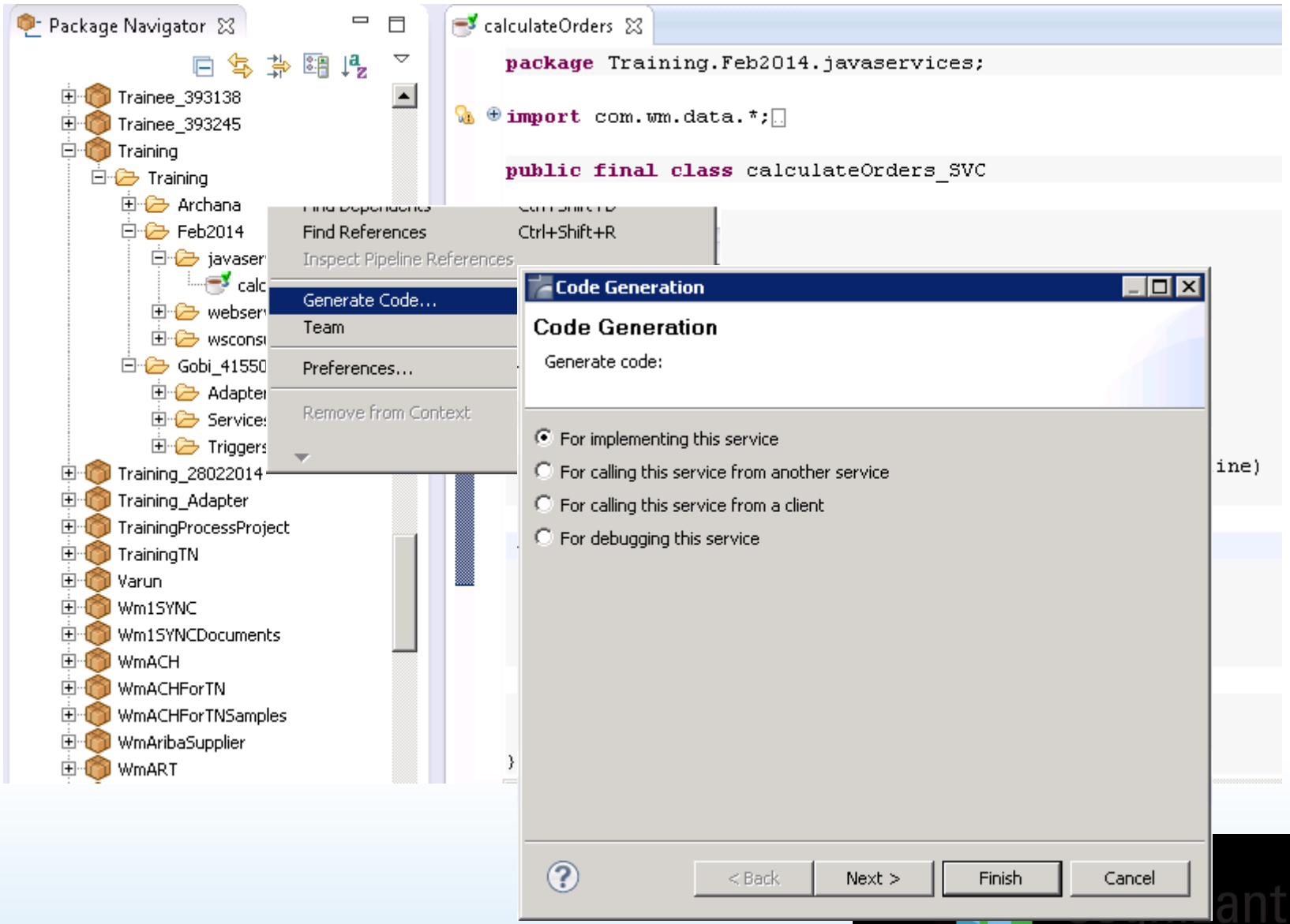
- Third party jars
  - Java service can use third party jars
  - Jars need to be placed in
    - <wM Installation Dir>/IntegrationServer/packages/<PackageName>/code/jars folder
  - Once the jars are in place, the package need to be reloaded before referencing them in the service
    - Using imports tab
    - In code



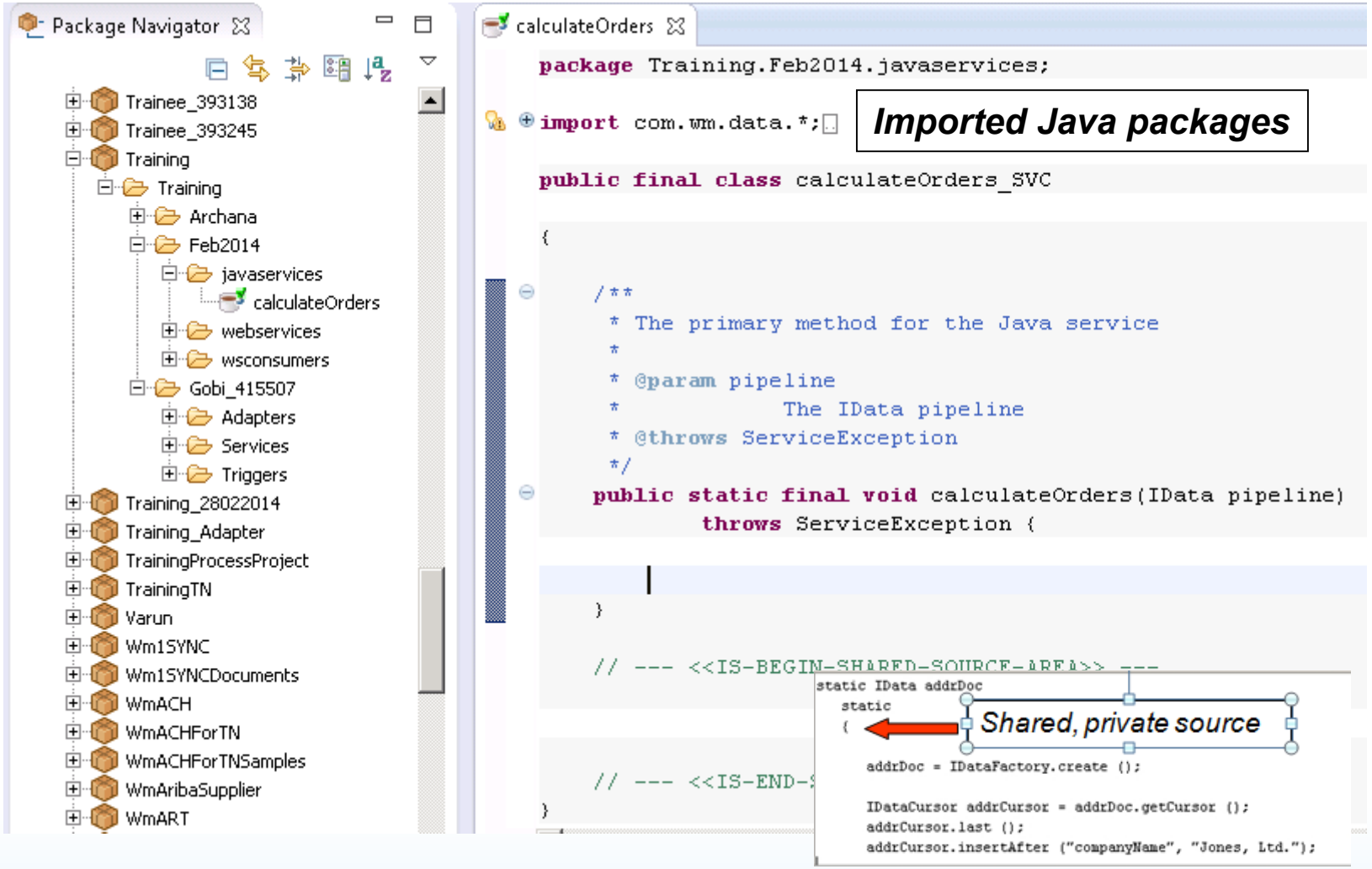
# Creating Java service



# Creating Java service



# Creating Java service - Shared



# Creating Java Service - Source

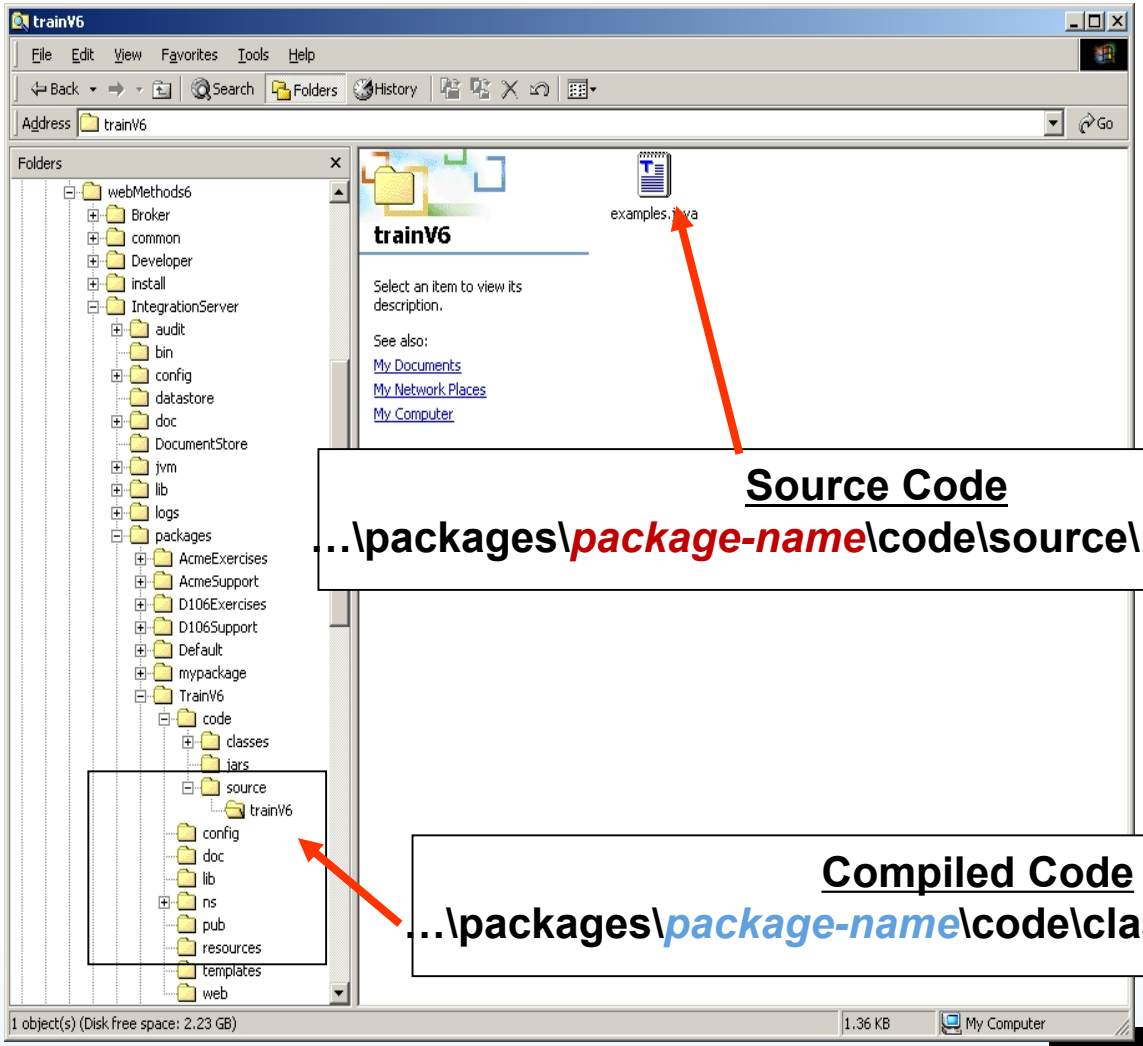
```
public static final void stringCompare( IData pipeline ) throws
ServiceException
{

// pipeline
IDataCursor pipelineCursor = pipeline.getCursor();

String  string1 = IDataUtil.getString( pipelineCursor, "string1" );
String  string2 = IDataUtil.getString( pipelineCursor, "string2" );
pipelineCursor.destroy();

int result = string1.compareTo (string2);

// pipeline
IDataCursor pipelineCursor_1 = pipeline.getCursor();
IDataUtil.put( pipelineCursor_1, "result", "result" );
pipelineCursor_1.destroy();
```

Input

Your Logic Here

Output

# Integration Server Code Namespace




**Source Code**

**…\packages*package-name*\code\source*folder\folder\…***

**Compiled Code**

**…\packages*package-name*\code\classes*folder\folder\…***

# Summary

## What have we learnt today ?

- Pipeline
- IData
- Cursors, DataTypes
- Code Generation
- Shared Resources
- Coding Java service
- Integration Server Code namespace



# Q & A

- What is a Pipeline ? What is IData ?
- How are packages and folders in webMethods represent in Java code ?
- How do we get data from the input ?
- How do we set data for the output in java services ?
- When do we go for IDataFactory?
- What is the equivalent representation of DocumentList dataType in java?
- Where do we place the external jars which needs to be referenced for the java service ?
- What is Integration Server Code Namespace ?



# Thank you