Session: 6

Introduction to Messaging

For Aptech Centre Use Only

Objectives

- Describe the messaging concept and its architecture
- Describe Java Messaging Service API
- Describe various messaging models supported by JMS
- Explain the working of Message-driven beans
- Describe how to create and configure a Message-driven bean in Java EE application

For Aptech Centre Use Only

Introduction

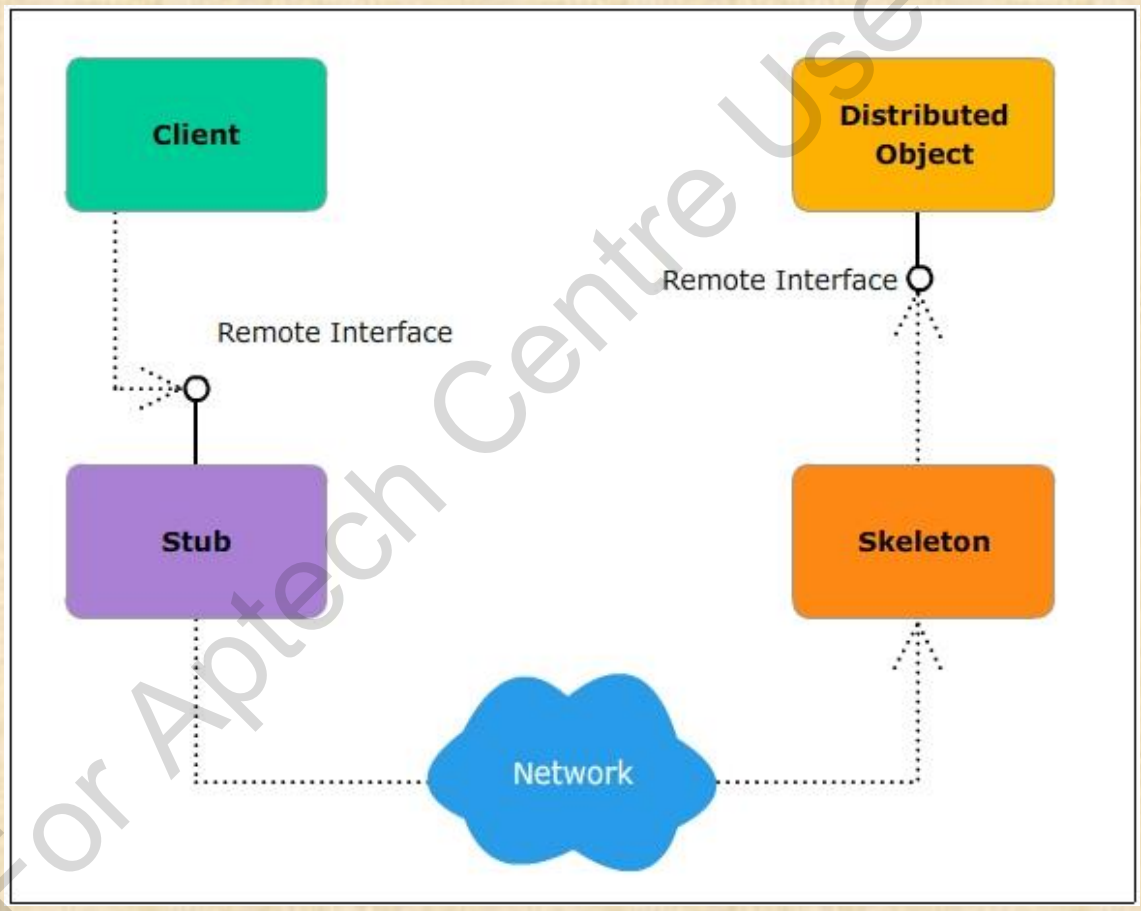
- Java application architecture is loosely coupled.
- Components are independent of each other.

- **How the components communicate?**
  - Technologies such as:
    - Remote Method Invocation (RMI)
    - Common Object Request Broker Architecture (CORBA)
    - Component Object Model (COM) or Distributed Component Object Model (DCOM)
  - Allow communication between distributed components or objects.

For Aptech Centre Use Only

Remote Method Invocation

- Following figure shows the implementation of RMI:



For Aptech Centre Use Only

Distributed Communication in Enterprise Applications 1-2

- Sun Microsystems introduced RMI mechanism which provides a native way to communicate distributed objects running in the different JVMs.
- RMI is an extension of Java Remote Invocation over Internet Inter-ORB Protocol (RMI-IIOP).
  - The use of RMI-IIOP resulted in the interoperability between the applications based on CORBA architecture.
  - Later on, RMI-IIOP became the protocol for EJBs to communicate and hence, served as a foundation for EJB.

For Aptech Centre Use Only

Distributed Communication in Enterprise Applications 2-2

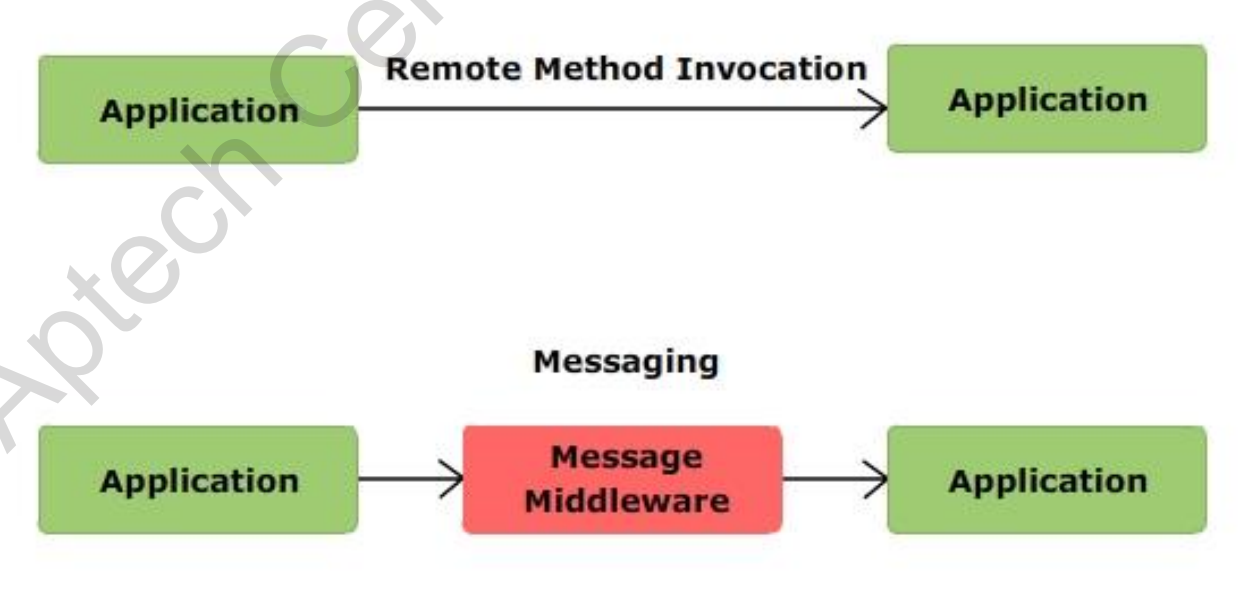
- Following are the features of RMI-IIOP which make it unsuitable for usage with EJB components:
  - Cannot support asynchronous communication.
  - Client and server are not decoupled.
  - Does not provide service reliability.
  - Cannot support communication between a client and multiple servers.

Since RMI-IIOP lacks the mentioned features, messaging was introduced for EJB applications.

For Aptech Centre Use Only

Messaging 1-2

- Messaging is based on the concept of Message Oriented Middleware (MOM) which serves as a middleman placed between the client and the server.
- Messaging implements asynchronous message processing.
- Following figure compares the implementation of RMI against Messaging:



For Aptech Centre Use Only

Messaging 2-2

- Messaging implements the following features to overcome the short comings of RMI-IIOP:
  - Asynchronous processing.
  - Decoupling of message senders and consumers.
  - Reliable message delivery system based on MOM.
  - Support for multiple message producers and consumers.

For Aptech Centre Use Only

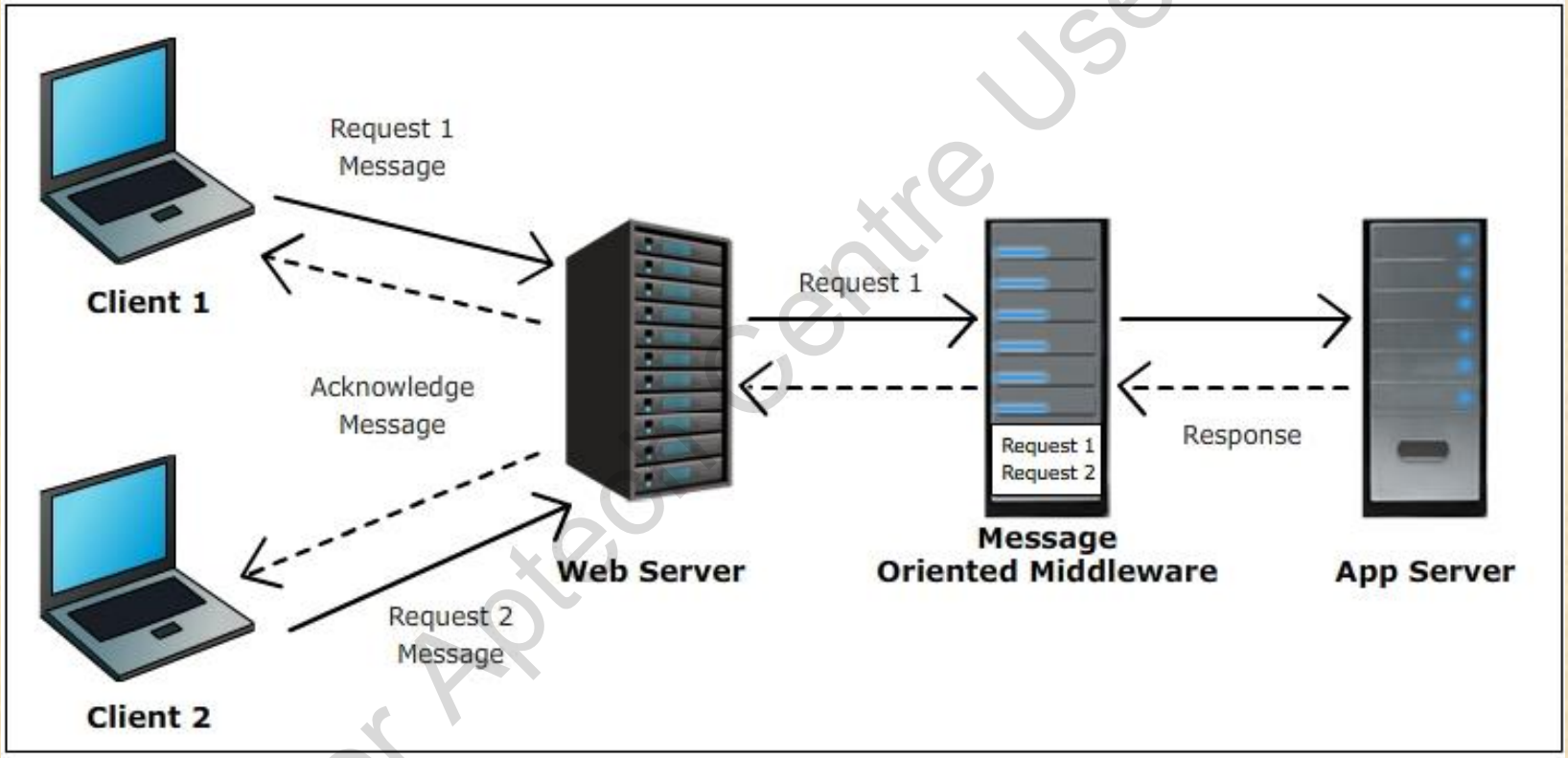
Message Oriented Middleware (MOM) 1-4

- Refers to an infrastructure that supports messaging.
- Can be defined as a software that enables asynchronous message exchange between system components.
- The software stores the message in the location specified by the sender and acknowledges immediately.
- The message sender is known as the **producer**.
- The location where the message is stored is known as the **destination**.
- The software components can retrieve the stored messages and are known as message **consumers**.

For Aptech Centre Use Only

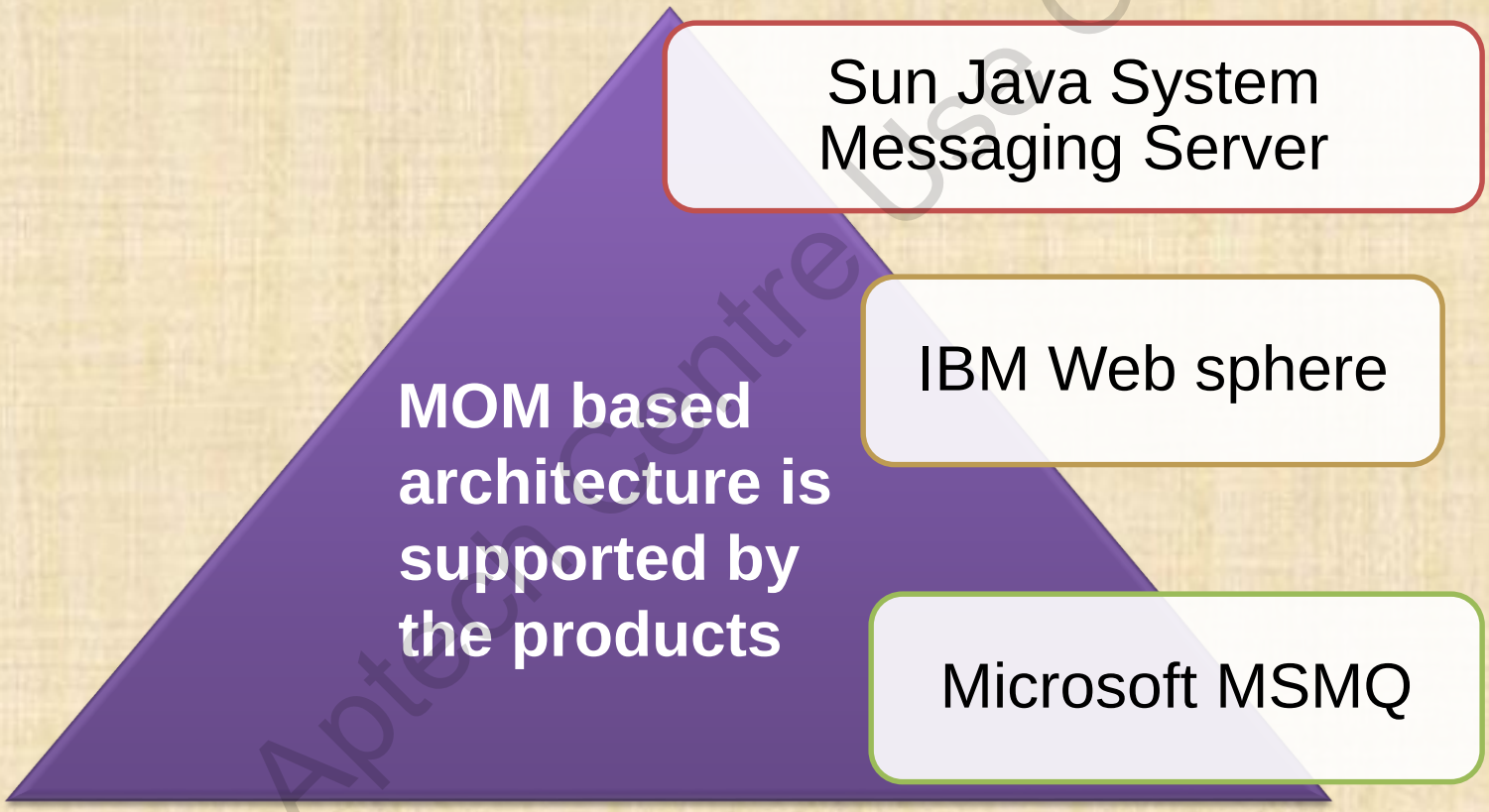
Message Oriented Middleware (MOM) 2-4

- Following figure shows the functioning of the MOM:



For Aptech Centre Use Only

Message Oriented Middleware (MOM) 3-4



For Aptech Centre Use Only

Message Oriented Middleware (MOM) 4-4

- MOM implementation has the following disadvantages:
  - Any server supporting MOM-based middleware has its own APIs which is specific to a vendor.
  - MOM APIs are not portable to other messaging systems.
  - Developers have to learn the proprietary messaging API to incorporate messaging functionality.

The Java EE platform provides a vendor-neutral API known as Java Messaging Service (JMS).

For Aptech Centre Use Only

Java Messaging Service (JMS) 1-2

- JMS supports messaging among the application components on the Java EE platform.
- JMS eliminates the need to learn the vendor-specific MOM-based APIs to perform communication between the components in the enterprise applications.
- JMS allows asynchronous communication between the components through messages.

For Aptech Centre Use Only

Java Messaging Service (JMS) 2-2

JMS API is divided into two parts:

JMS API

- Provides the functionality of creating, sending, receiving, and reading messages among the application components through a set of interfaces.

Service Provider Interface (SPI)

- Used as a plug-in for the MOM implementation on the server.

For Aptech Centre Use Only

JMS Communication Models

Based on the pattern of communication there are two messaging models supported by JMS:

- Publish-Subscribe model
- Point-to-Point messaging model

For Aptech Centre Use Only

Publish-Subscribe Model 1-4

- Multiple message producers can communicate with multiple message consumers.
- Messages are sent through a virtual channel called **Topic**.

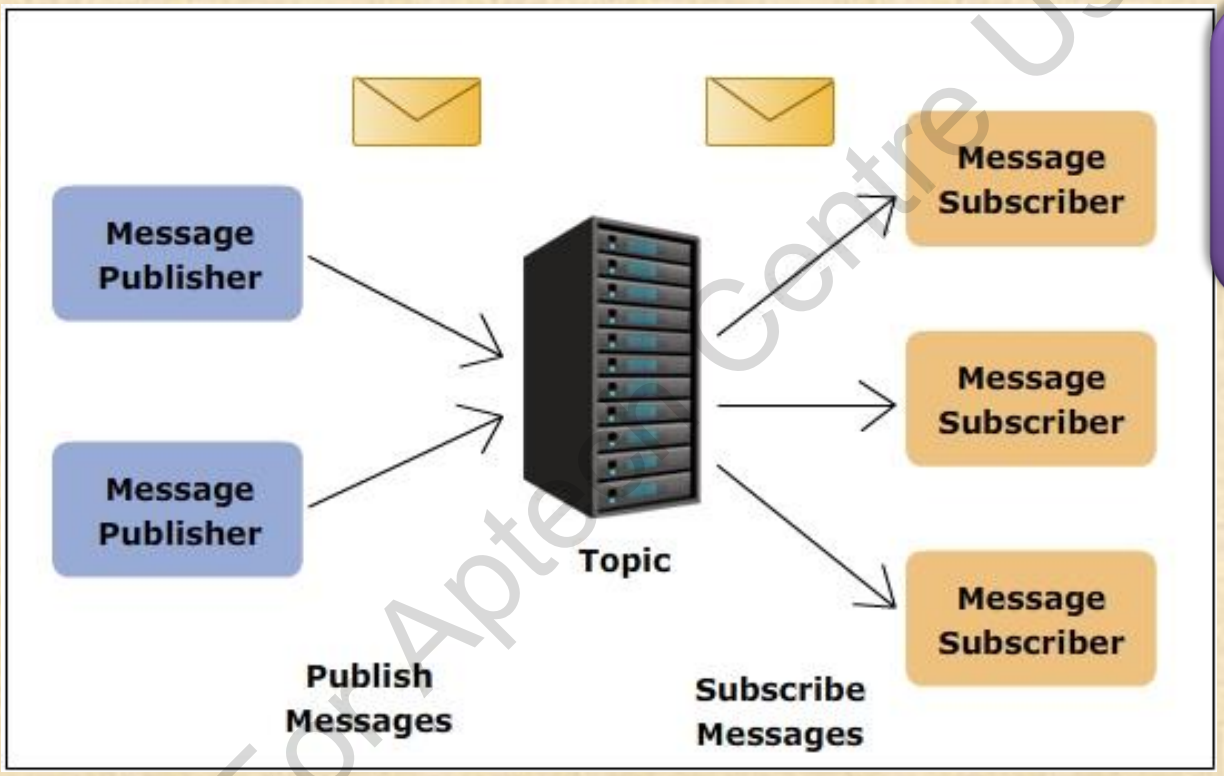
Working of Publish-Subscribe Model

- **Topic** is a logical destination object which contains messages from different sender components.
- The components which is sending the message to the topic is said to be a **Publisher**.
- In order to access messages from the topic, the components should subscribe to the topic.
- **Subscriber** is the term used to refer to a message consumer component which has subscribed to receive all the messages from topic destination.

For Aptech Centre Use Only

Publish-Subscribe Model 2-4

- Following figure shows how different components communicate in a publish-subscribe model:



The MOM system maintains the list of subscribers for each Topic.

For Aptech Centre Use Only

Publish-Subscribe Model 3-4

- The variants of subscriptions are:

Shareable subscriptions	• Message can be received by a set of consumers who have agreed to share the subscription.
Non-shareable subscriptions	• Implies that only one subscriber can receive a message through subscription.

For Aptech Centre Use Only

Publish-Subscribe Model 4-4

- Subscriptions can also be classified as durable or non-durable.

Durable subscriptions	• Messages are retained by the topic for certain time period after all clients consume.
Non-durable subscriptions	• Messages are not retained by the topic after all the subscribed clients receive the message.

For Aptech Centre Use Only

Point-to-Point Model 1-2

- The communication is between a pair of components.
- The destination is called as a **Queue**.

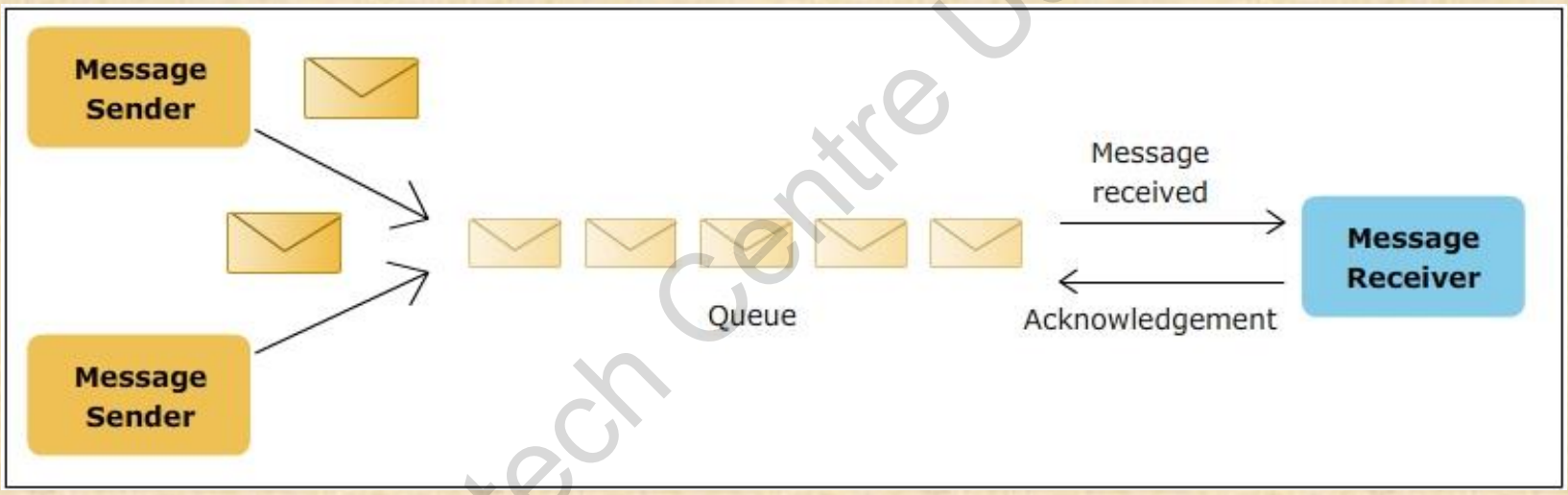
Working of Point-to-Point Model

- There is only a single consumer for each message.
- A consumer can grab the message in the queue, but a given message is consumed only once by a consumer.
- Multiple producers can send a message to the queue.
- Only one consumer can consume a message from the queue.
- Messages are sent by the producers to a centralized queue and are distributed as First In First Out (FIFO).
- Receivers access the queue to retrieve the messages.
- Messages are retained in the queue until the consumer consumes it.

For Aptech Centre Use Only

Point-to-Point Model 2-2

- Following figure shows how different components communicate through point-to-point messaging model:



For Aptech Centre Use Only

Use of JMS API in EJB

Application components can send and receive asynchronous messages

Application components can set up message listeners

JMS messages can be processed through message driven beans

Containers maintain pool of message driven beans to process JMS messages

JMS messages can be used as a part of Java transaction

For Aptech Centre Use Only

Messaging Application Architecture 1-2

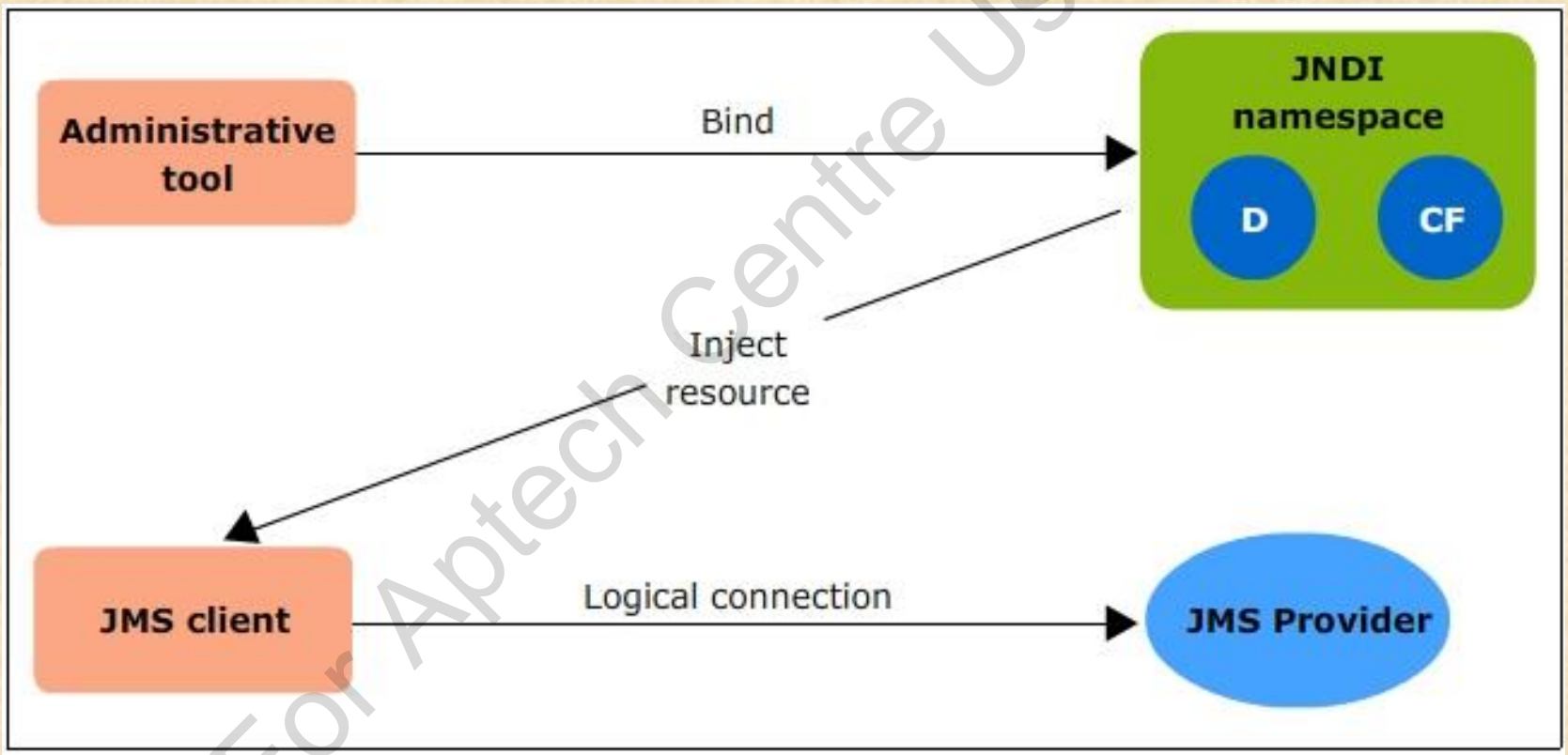
- A typical JMS application has the following components:

JMS Provider	• Routes messages among components. • Implements interfaces provided by API. • Manages messaging infrastructure.
JMS clients	• Are those application components which use the services provided by the JMS provider and exchange messages among themselves.
Administered objects	• Serves as a communication infrastructure for JMS applications. The JMS Connection objects, sessions, and destination are administered objects.
Messages	• Are objects which are exchanged during communication process.

For Aptech Centre Use Only

Messaging Application Architecture 2-2

- Following figure shows how different objects of JMS application interact:



For Aptech Centre Use Only

JMS API Programming Model 1-8

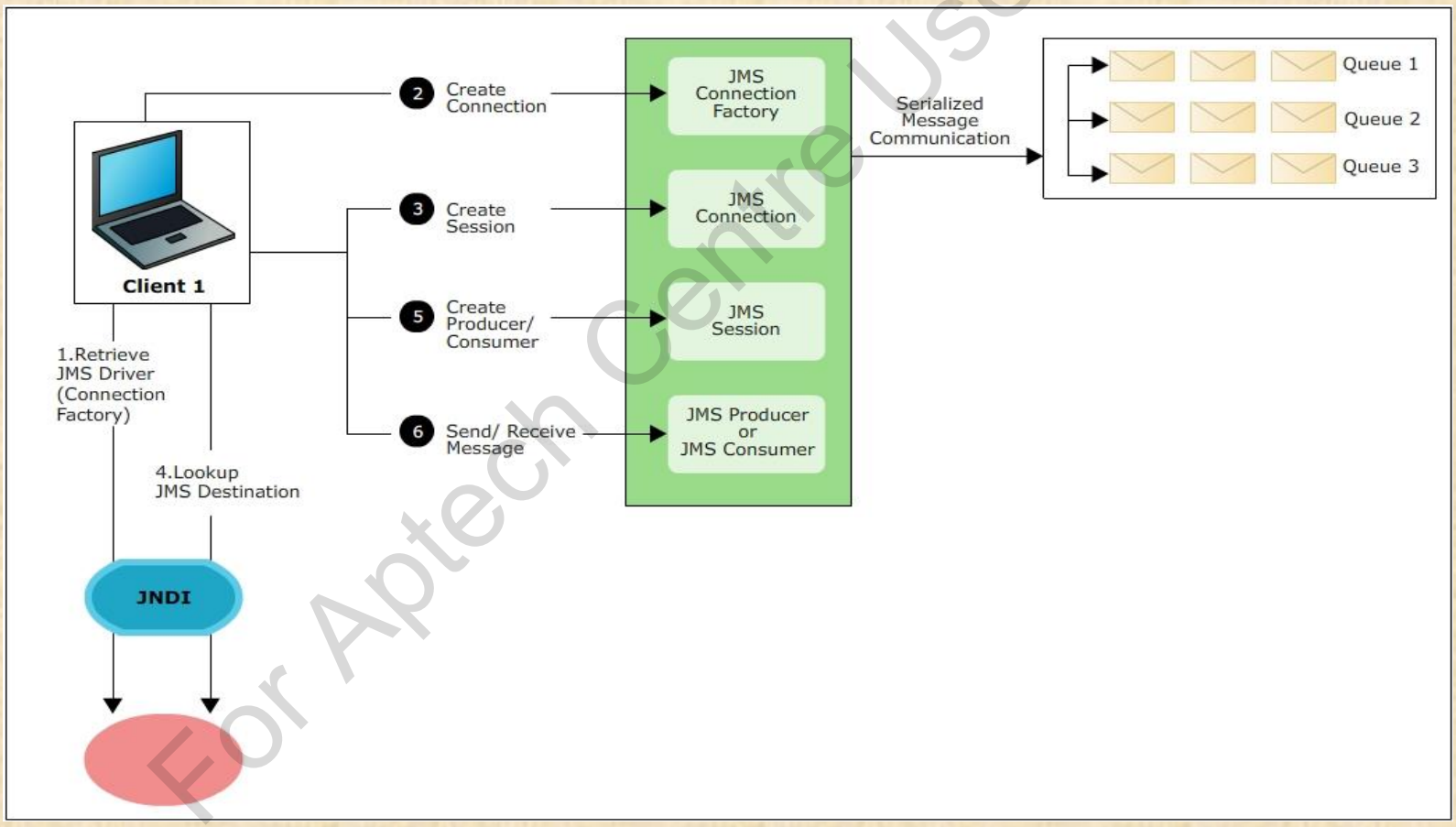
Following are the objects which are part of a JMS application:

- Administered objects
- Connections
- Session
- JMSContext
- JMS Message Producers
- JMS Message Consumers
- Messages

For Aptech Centre Use Only

JMS API Programming Model 2-8

- Following figure shows the JMS API programming model and how different objects interact with each other:



For Aptech Centre Use Only

JMS API Programming Model 3-8

JMS Connection Factories

- Are administered objects.
- Can be an instance of **`ConnectionFactory`**, **`QueueConnectionFactory`**, or **`TopicConnectionFactory`**.
- Connection object when instantiated is a part of **`JMSContext`**.
- The Java EE server provides a JNDI name to access the **`ConnectionFactory`** object.

JMS Destination

- An administered object.
- Either a Queue or Topic.
- Can be created on the application server.

For Aptech Centre Use Only

JMS API Programming Model 4-8

JMSContext objects

- Provide an application environment for JMS applications.
- Are created from a **`ConnectionFactory`** through **`createContext()`** method.
- Are used to create other objects of the applications.
- Are associated with a **`Connection`** and **`Session`** object.

JMS Message Producer objects

- Can be created using **`createProducer()`** method in the **`JMSContext.`**

For Aptech Centre Use Only

JMS API Programming Model 5-8

JMS Message Consumers

- Created in a similar way as message producers along with the **`JMSContext`** instance.
- **`setAutoStart()`**, **`receive()`**, and **`start()`** methods can be used to alter the behavior of the consumer.

Messages

- Object of communication.
- Every message has a standard format.
- Each message is associated with a message header.

For Aptech Centre Use Only

JMS API Programming Model 6-8

- There are three parts of a message – message header, properties, and body.
- Message header holds all the control information of the message.
- Following table shows the interpretation of various fields in the message header:

Header field	Description
JMSDestination	This field defines the destination where the message is destined to. This is set by the `send()` method of message producer.
JMSDeliveryMode	This field defines the mode of message delivery, that is the method of persisting the message. This value is set by the `send()` method.
JMSDeliveryTime	This field of the message defines the time when the message sending process is initiated and also the delay time tolerable. This field is also set by the `send()` method.
JMSExpiration	This field defines the maximum time duration for which the current message is valid. This value is set by the `send()` method.

For Aptech Centre Use Only

JMS API Programming Model 7-8

Header field	Description
JMSPriority	This field defines the priority of the message and is set by the `send()` method of the message producer.
JMSMessageID	It is an unique identifier to identify the message and is set by the `send()` method.
JMSTimeStamp	This field specifies the time when the message was handed over to the JMS provider and is set by the `send()` method.
JMSCorrelationID	This field is set by the client and is used to link one message to the following message in a session.
JMSReplyTo	This field specifies the destination where the reply to the current message has to be sent.
JMSType	JMS messages can have data of different types. This field defines the data type of the message content and is set by the client.
JMSRedelivered	This field is set when the acknowledgement for the message is not received and as a result the message is redelivered.

For Aptech Centre Use Only

JMS API Programming Model 8-8

- Message body implies actual content of the message.
- JMS supports following types of messages:
  - Text message
  - Map message
  - Bytes message
  - Stream message
  - Object message
  - Message

For Aptech Centre Use Only

Implementing JMS 1-3

To create a JMS application, developer has to create JMS resource on the server.

Messages are stored in a JMS resource such as Queue or Topic.

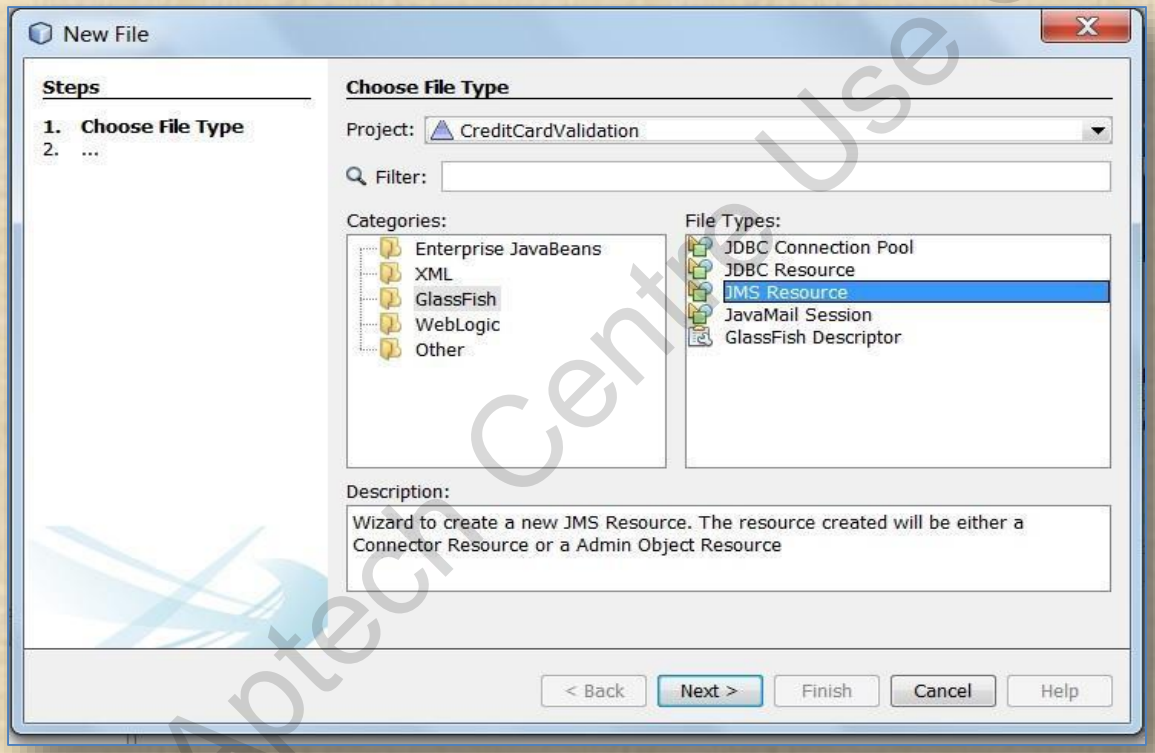
Developers can create JMS resources through 'Administration Console' on GlassFish server.

Applications may require creation of multiple JMS resources.

For Aptech Centre Use Only

Implementing JMS 2-3

- Following figure demonstrates how to create a JMS resource on the server:

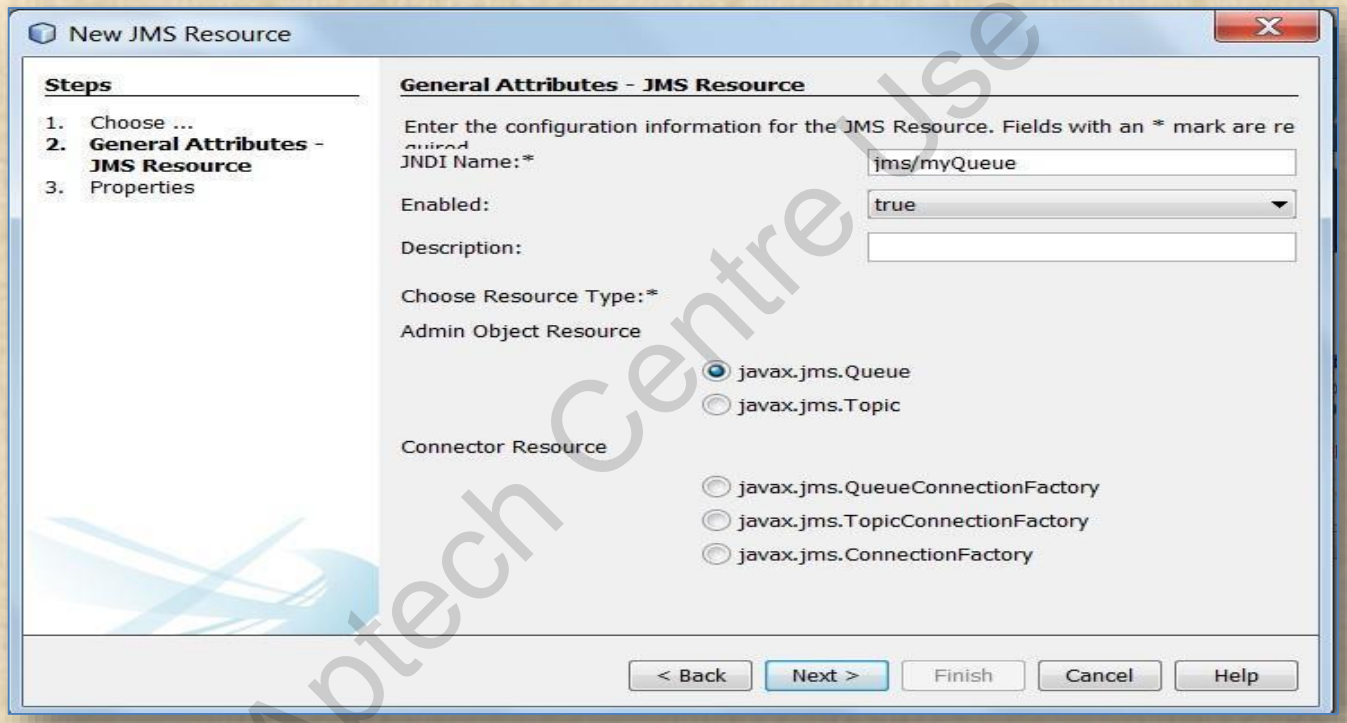


Right-click the project and select '**Other**'. Choose **JMS Resource** from the menu.

For Aptech Centre Use Only

Implementing JMS 3-3

- Following figure shows the wizard to provide the JNDI name of the JMS resource:



- Complete the resource creation process by providing the configuration information.

For Aptech Centre Use Only

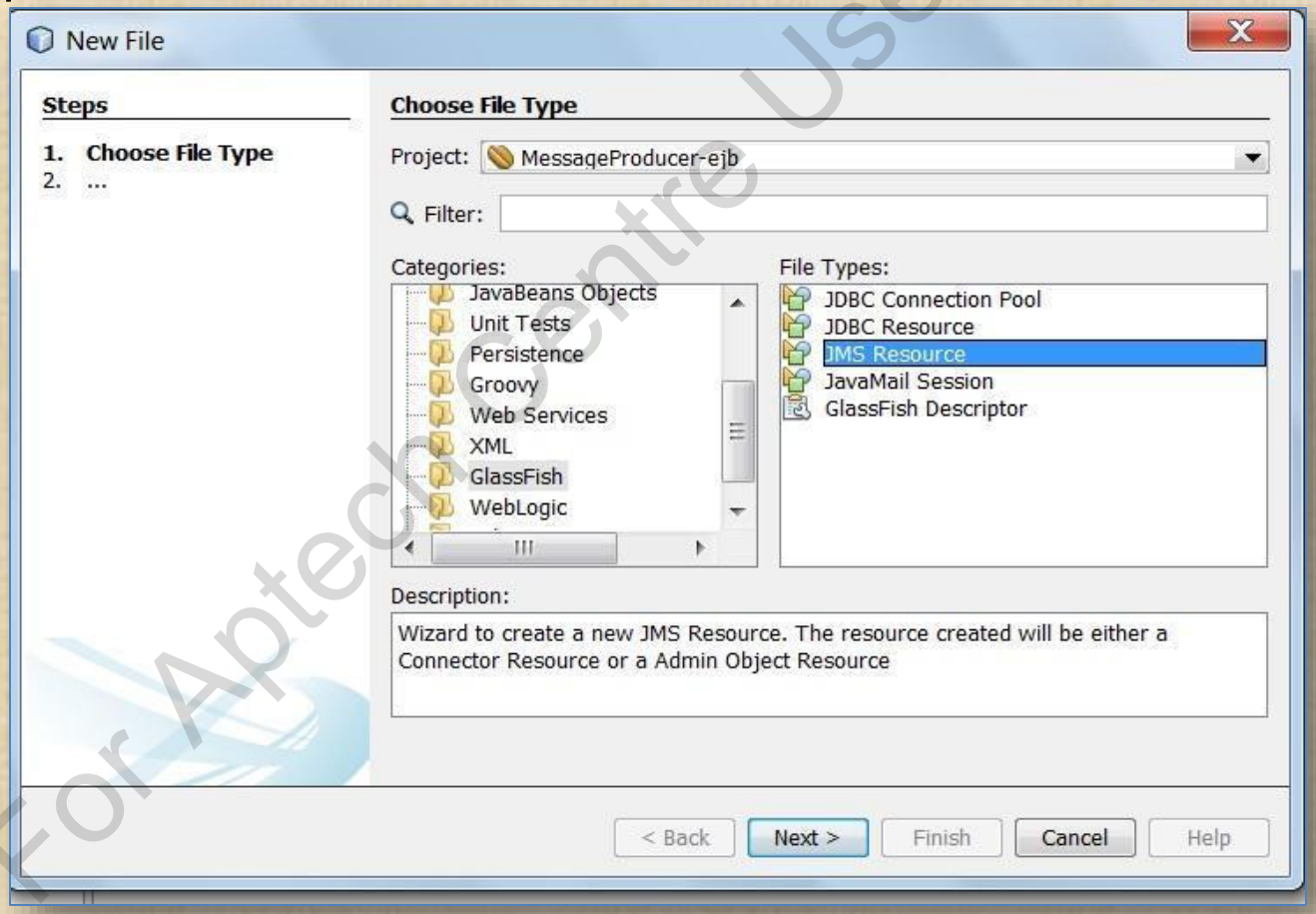
JMS Resources in an Application 1-7

- The process of creating a message producer and message consumer are as follows:
  - Create a message producer application. The message producer enterprise application sends messages to the message destination on the server.
  - Create the '**JMS Resource**' on the GlassFish server. When the message producer application sends messages.
  - The messages are stored on these JMS resources located on the server. The JMS resources created on the server can either be a `JMSQueue` object or `JMSTopic` object.

For Aptech Centre use Only

JMS Resources in an Application 2-7

- Following figure demonstrates creating **JMS resource(Queue/Topic)** in the application:



For Aptech Centre Use Only

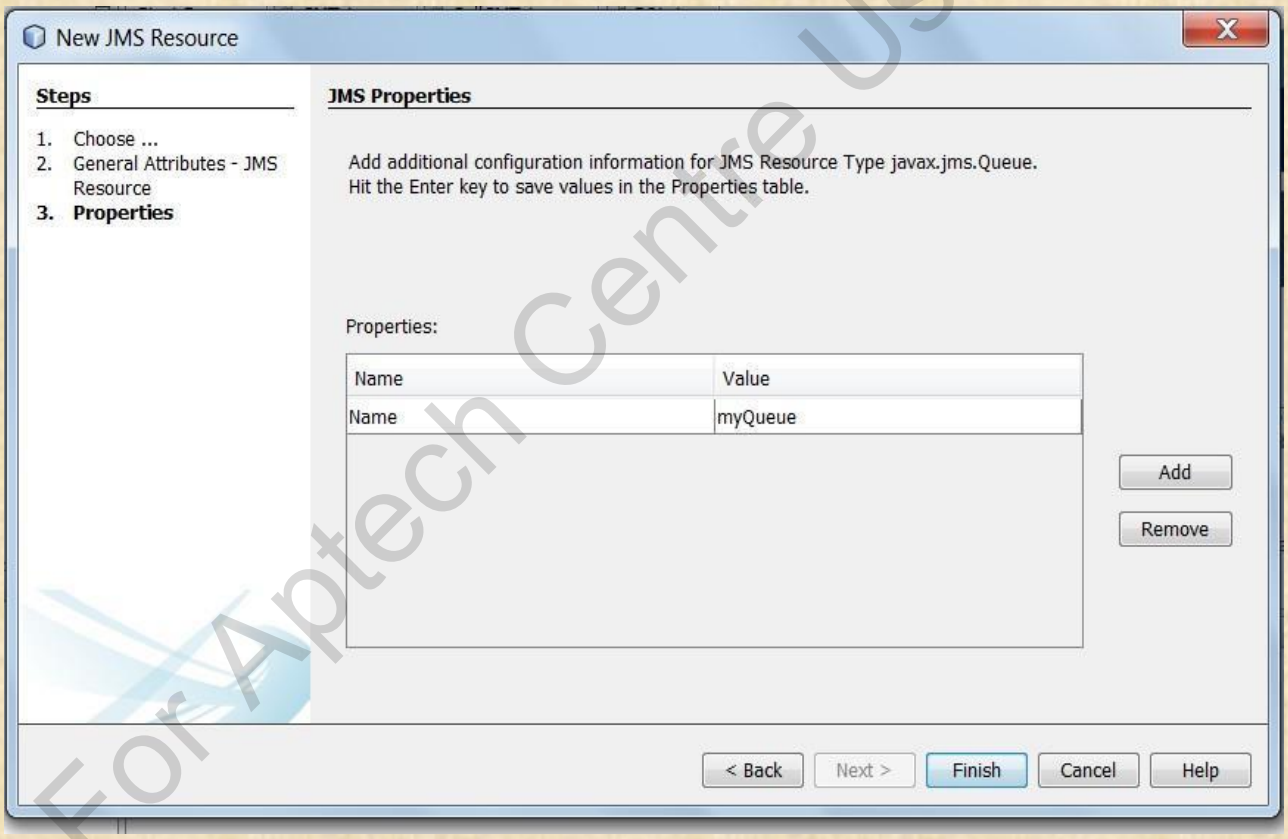
JMS Resources in an Application 3-7

- Following figure demonstrates adding a **`Queue`** object and a **`Connector`** object to the application server through NetBeans IDE:

For Aptech Centre Use Only

JMS Resources in an Application 4-7

- Following figure shows how to provide a name to the JMS resource on the server:



For Aptech Centre Use Only

JMS Resources in an Application 5-7

- The information about **myQueue** is added to the `glassfish-resources.xml` file in the Server Resources folder.

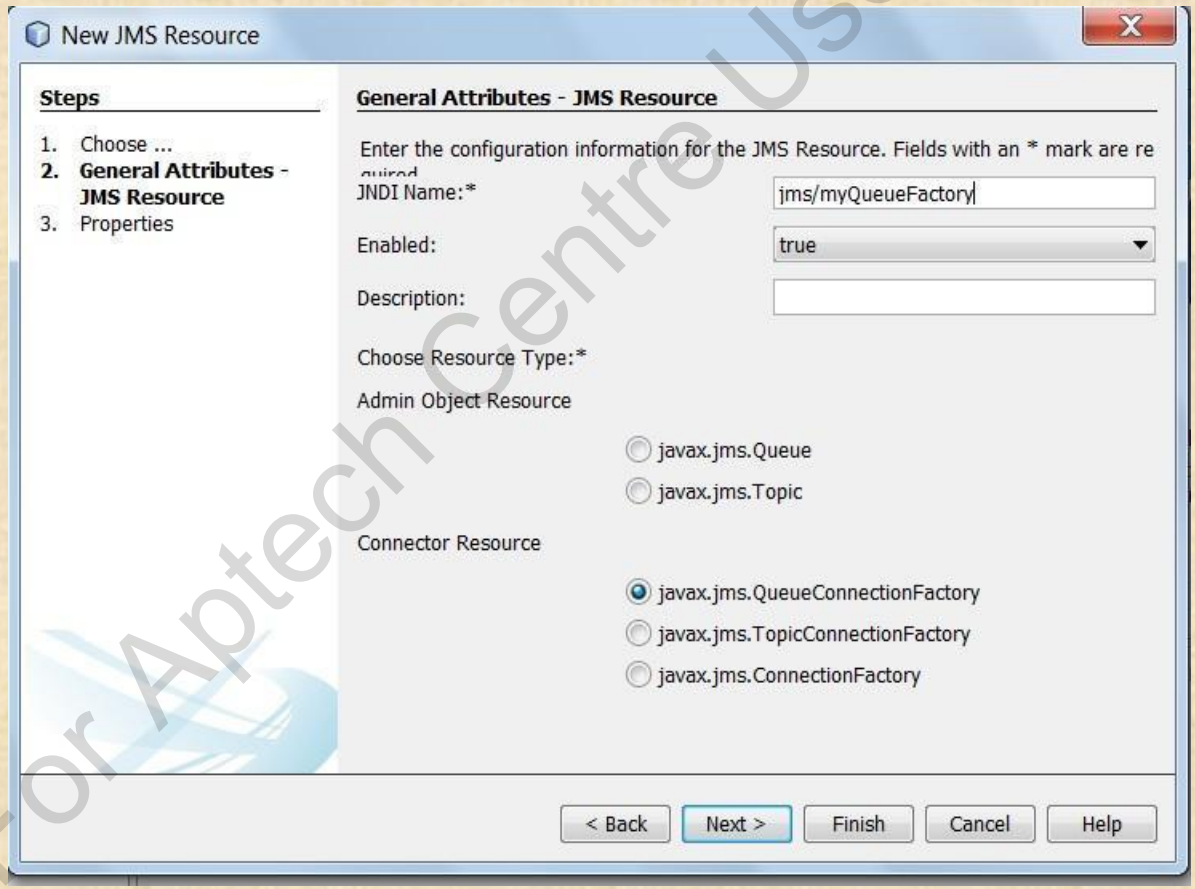
Creating ConnectionFactory

- The `ConnectionFactory` resource is required to make connections between the message producer module and the `Queue` object on the server.
- Right-click the project and select **New → Other → GlassFish JMS Resource** from the New File dialog box.
- Click **Next**.

For Aptech Centre Use Only

JMS Resources in an Application 6-7

- Following figure shows how to create **`QueueConnectionFactory`** and **`Connector`** resources for the application:



For Aptech Centre Use Only

JMS Resources in an Application 7-7

Specify the JNDI Name as '**jms/myQueueFactory**' and select the Connector Resource as '**javax.jms.QueueConnectionFactory**'.

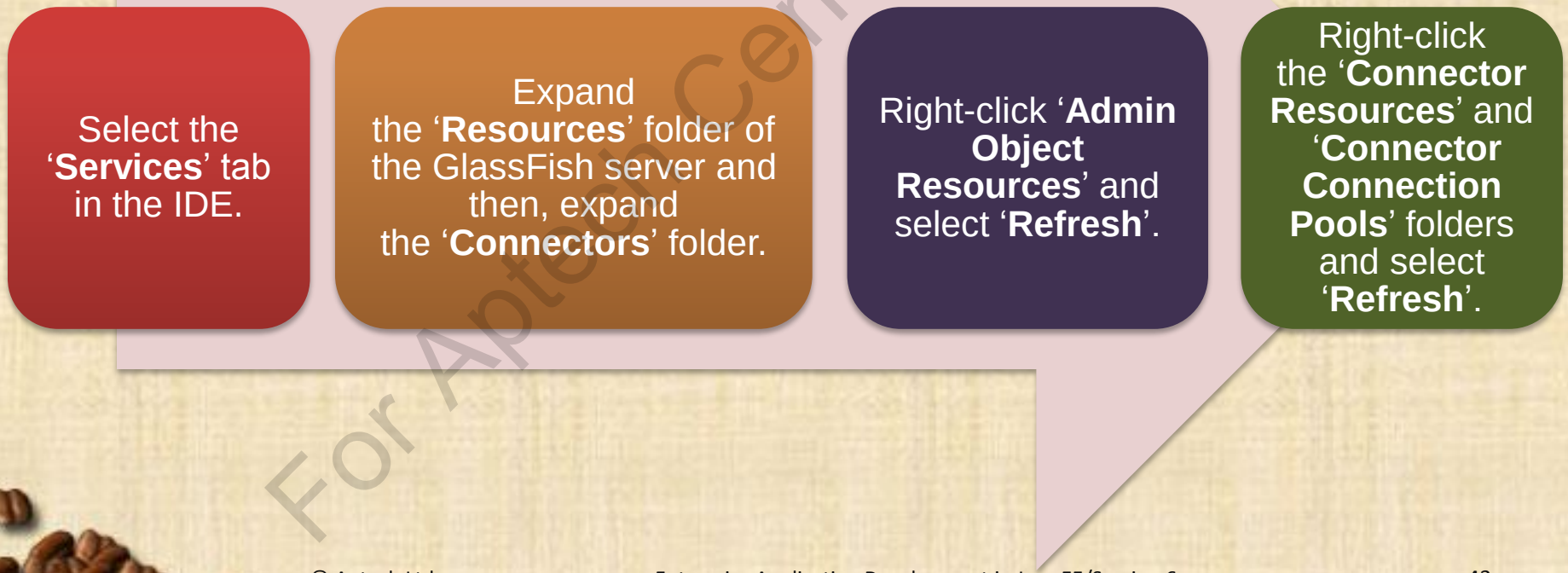
Click '**Next**'. The JMS Properties screen is displayed. Here, you can specify additional configuration information.

Click '**Finish**'. The information about **myQueueFactory** is added to the glassfish-resources.xml file in the Server Resources folder.

For Aptech Centre Use Only

Checking JMS Resources on GlassFish Server 1-2

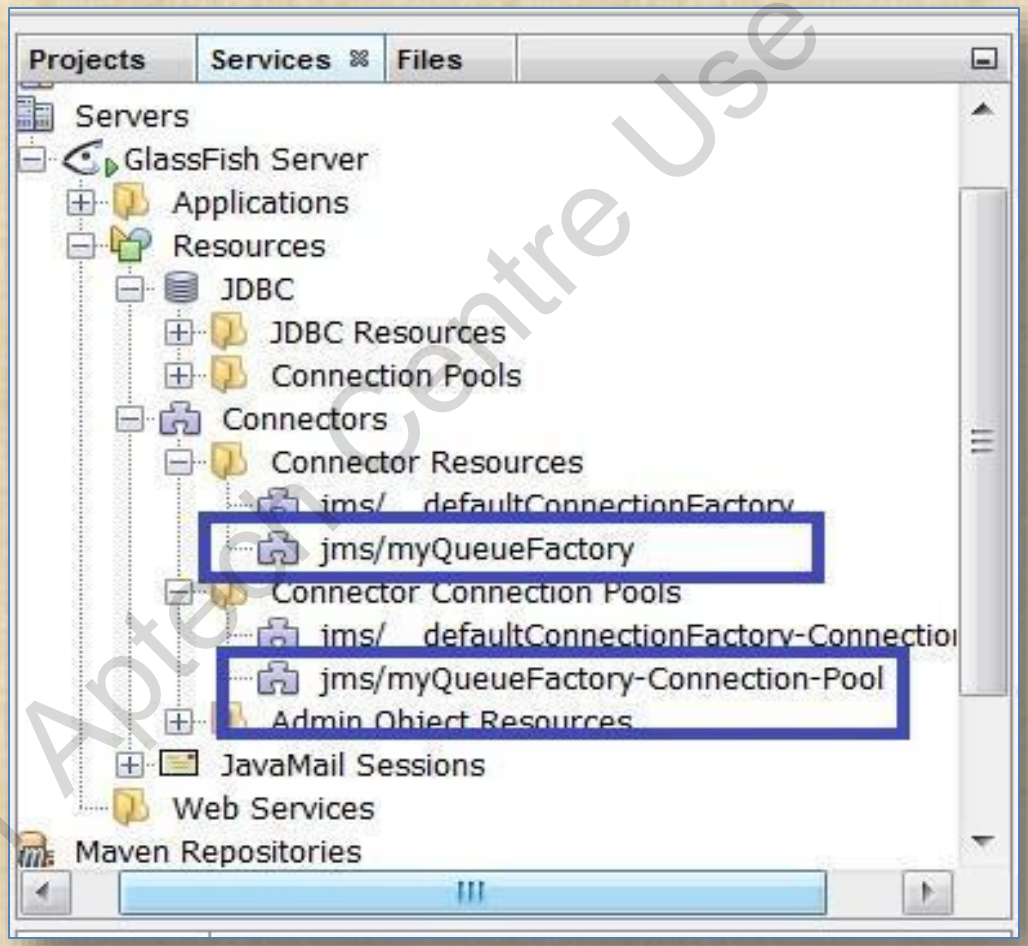
- Following are the steps to be followed to check whether JMS resources are added to GlassFish server:



For Aptech Centre Use Only

Checking JMS Resources on GlassFish Server 2-2

- Following figure shows JMS resources on GlassFish server:



For Aptech Centre Use Only

Creating Bean Objects in MessageProducer Application 1-3

- Following are the steps to create a JSF managed bean which will act as message producer:

Right-click the project name and select **New → Other → JavaServer Faces → JSF ManagedBean** from the New File dialog box.

Click '**Next**'. The New JSF ManagedBean dialog box displays on screen.

Specify the class name of the bean as '**MessageProducerBean**' and click '**Finish**'. The ManagedBean is created.

For Aptech Centers Use Only

Creating Bean Objects in MessageProducer Application 2-3

- Following code snippet shows the code for the managed bean created in **`MessageProducer`** application:

```
package mes;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class MessageProducerBean {

    /**
     * Creates a new instance of MessageProducerBean
     */

}
```

For Aptech Centre Use Only

Creating Bean Objects in MessageProducer Application 3-3

```
 private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
    public MessageProducerBean() {
    }

}
```

- A JSF ManagedBean is created to which the attribute message has been added. The getter and setter methods for this property have also been generated.

For Aptech Centre Use Only

Adding JMS Code to the ManagedBean 1-3

- Following are the steps for adding code to the ManagedBean:

In the editor area of '**MessageProducerBean'**, press '**Alt + Insert**' key to open the NetBeans Code Generator feature and select '**Send JMS Message**' option.

The Send JMS Message dialog box displays on screen. Ensure that Project Destination is set to '**jms/myQueue'** and Connection Factory is set to '**jms/myQueueFactory'**.

Click '**OK**'. NetBeans adds the proper resource declarations to the code for the `Queue` and `ConnectionFactory` instances.

For Aptech Centre Use Only

Adding JMS Code to the ManagedBean 2-3

- Following code snippet shows the JMS code added to the Managed bean:

```
@ManagedBean
@RequestScoped
public class MessageProducerBean {
    @Resource(mappedName = "jms/myQueue")
    private Queue myQueue;
    @Inject
    @JMSConnectionFactory("jms/myQueueFactory")
    private JMSContext context;

    /**
     * Creates a new instance of MessageProducerBean
     */
    private String message;
```

For Aptech Centre Use Only

Adding JMS Code to the ManagedBean 3-3

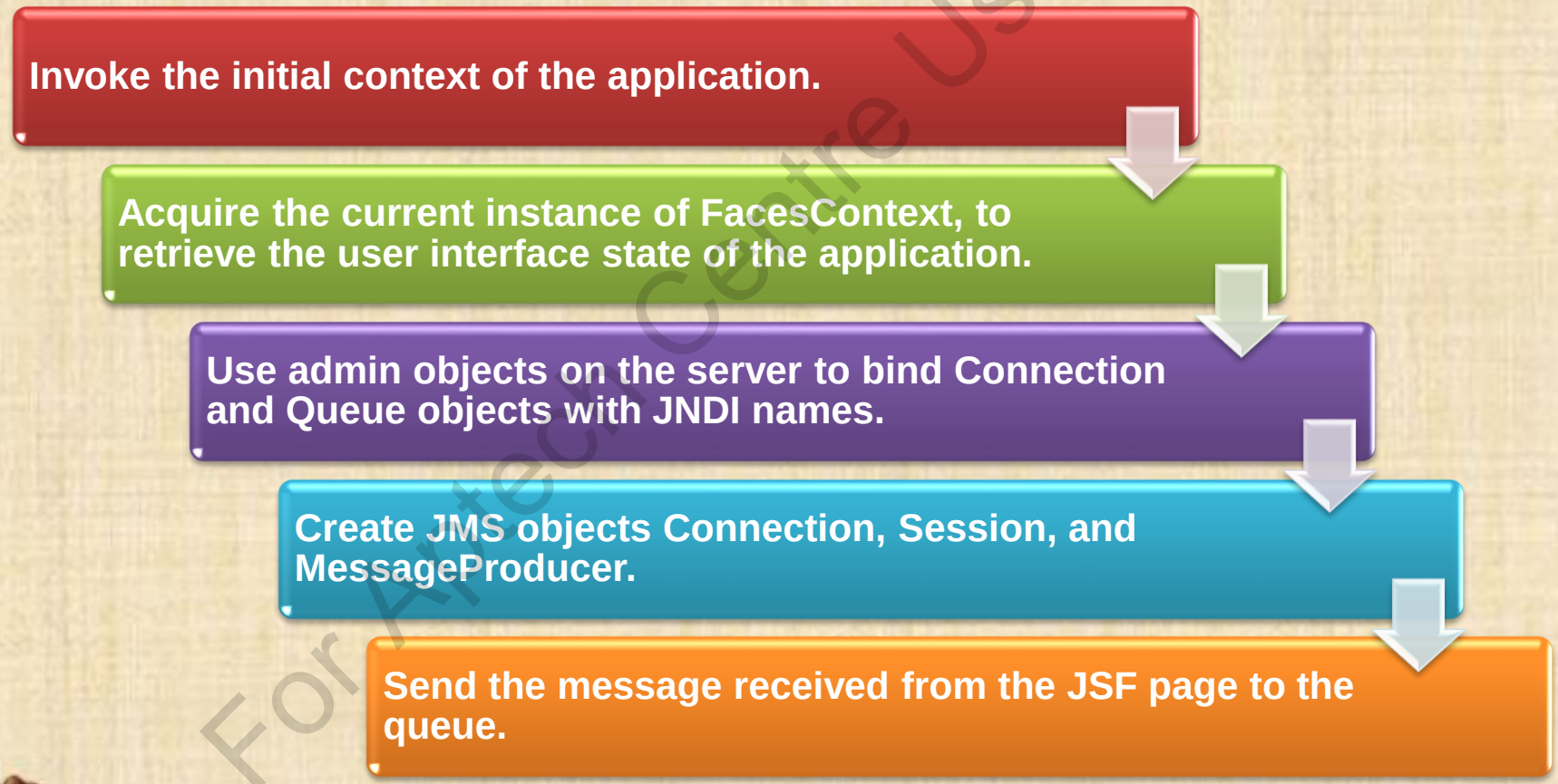
```
public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
    public MessageProducerBean() {
    }

}
```

- A `Queue` object has been added which is meant to be the destination of the message sent from the session bean.
- A `JMSConnectionFactory` object is also added as a resource which enables the message sender to connect to the destination on the application server.

For Aptech Centre Use Only

Creating a send() Method in the MessageProducer Application 1-3

- Following are the steps to add a new **`send()`** method to the application:



For Aptech Centre Use Only

Creating a send() Method in the MessageProducer Application 2-3

- Following code snippet shows the code to be added to the **`send()`** method:

```
. . .
public void send() throws NullPointerException,
NamingException, JMSException {
InitialContext initContext = new InitialContext();

FacesContext facesContext =
FacesContext.getCurrentInstance();

ConnectionFactory factory =(ConnectionFactory)
initContext.lookup("jms/myQueueFactory");

Destination = (Destination)
initContext.lookup("jms/myQueue");
initContext.close();
```

For Aptech Centre Use Only

Creating a send() Method in the MessageProducer Application 3-3

```
//Create JMS objects
Connection connection = factory.createConnection();
Session session = connection.createSession(false, Session.
AUTO_ACKNOWLEDGE);
MessageProducer sender = session.createProducer(myQueue);

//Send messages
   TextMessage msg = session.createTextMessage(message);
   sender.send(msg);
   FacesMessage facesMessage = new FacesMessage("Message sent: " +
message);
   facesMessage.setSeverity(FacesMessage.SEVERITY_INFO);
   facesContext.addMessage(null, facesMessage);
}
```

- In the `send()` method, first lookup of the Queue and Connection objects on the server is performed.
- Once these objects are accessed, their instances are created to send a message to the Queue which is configured on the server. These message objects reside on the server.
- They can be accessed by a consumer application.

For Aptech Centre Use Only

Creating a receive() Method in MessageConsumer Application 1-2

- Create a **`MessageConsumer`** application. The managed bean within this application should have a **`receive()`** method to receive messages.
- Following code snippet shows the code in the **`receive()`** method:

```
public class ReceiverBean implements
javax.ejb.SessionBean
 {
 InitialContext jndiContext;
public String receiveMessage() {
try{
QueueConnectionFactory f = (QueueConnectionFactory)
jndiContext.lookup("java:comp/env/jms/myQueueFactory"
);
jndiContext.lookup("java:comp/env/jms/myQueue");
```

For Aptech Centre Use Only

Creating a receive() Method in MessageConsumer Application 2-2

```
QueueConnection connect = factory.createQueueConneciton();
QueueSession session = connect.createQueueSession(true,0);

QueueReceiver rec = session.createReceiver(myqueue);
TextMessage textMsg = (TextMessage)rec.receive();
connect.close();
return textMsg.getText();
} catch(Exception e) {
    throws new EJBException(e);

  }
}
```

For Aptech Centre Use Only

Introduction to Message-Driven Bean 1-2

Message-driven Bean

- Is an enterprise bean used to process asynchronous messages received from application components.
- Is an EJB component that receives JMS message and other messages.
- Is a stateless, server-side component that is used for processing asynchronous messages delivered using JMS.
- Is invoked by the container and is responsible for processing messages.
- Does not have a remote or local business interface.
- Multiple instances of a message-driven bean can simultaneously process messages.
- Message-driven beans cannot be accessed through interfaces.
- Every message-driven bean has an **`onMessage()`** method to be executed.

For Aptech Centre Use Only

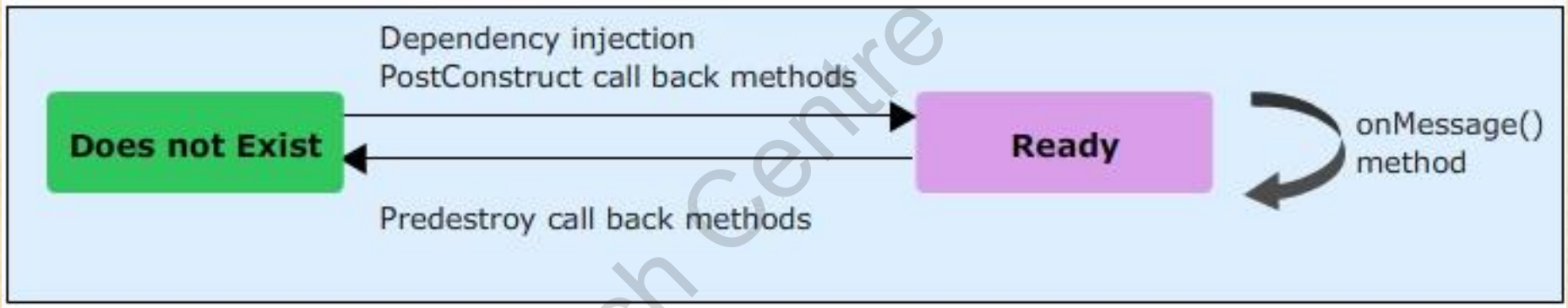
Introduction to Message-Driven Bean 2-2

- Following are the characteristics of message-driven bean:
  - A message-driven bean is executed only upon receiving a message.
  - They are short lived and are invoked asynchronously.
  - Message-driven beans can access and update database, but do not represent the data in the database.
  - Message-driven beans are stateless and transaction aware.

For Aptech Centre Use Only

Life Cycle of a Message Driven Bean

- Following figure shows the life cycle of a Message-Driven Bean:



For Aptech Centre Use Only

Lifecycle Event handlers

- Callback methods which are invoked when the message-driven bean transits from one state to another in the life cycle.
- Message-driven bean can have two types of callback methods:
  - **`PostConstruct`**
  - **`PreDestroy`**
- Callback methods should be appropriately annotated with **`@PostConstruct`** and **`@PreDestroy.`**

For Aptech Centre Use Only

Comparing Session Beans with Message-Driven Beans

Session beans	Message-driven beans
• Can be invoked directly through interfaces. • Maintain state of the bean. • Can be passivated or activated.	• Cannot be invoked directly. • Asynchronously invoked through messages. • Cannot maintain the state of the bean. • Cannot be passivated.

For Aptech Centre Use Only

Creating and Configuring Message-Driven Beans 1-7

- Message-Driven Bean (MDB) is created like session beans in an enterprise application.
- MDB must implement **`javax.jms.MessageListener`** interface.
- They should have definition for **`onMessage()`** method.
- The implementation class of MDB should have a default constructor.

EJB 3.0, it is not compulsory to implement the **MessageDrivenBean** interface.

For Aptech Centre Use Only

Creating and Configuring Message-Driven Beans 2-7

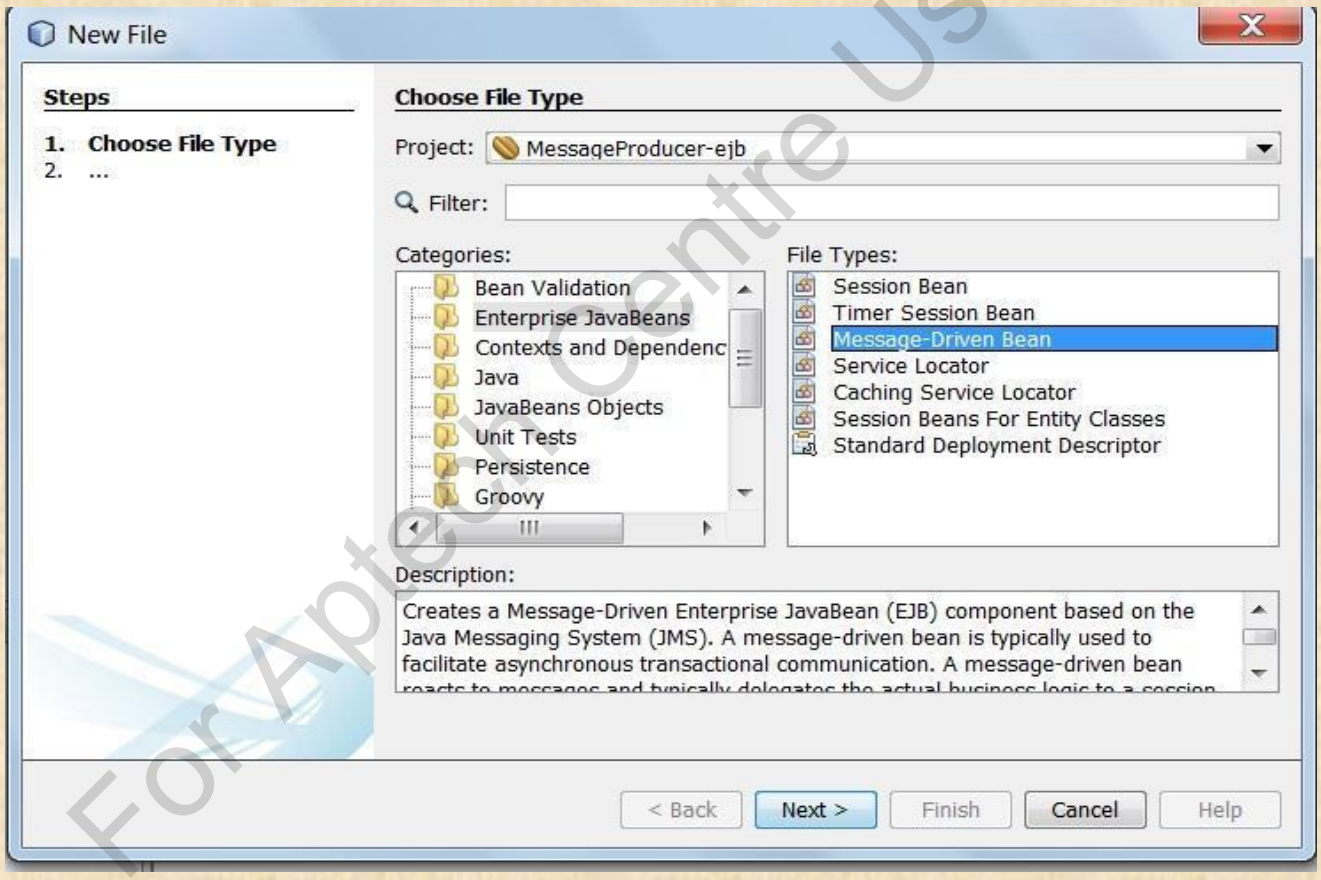
- Following code snippet shows the code used for creating a message listener for a consumer:

```
…..
JMSContext J = connectionFactory.createContext();
JMSConsumer JC = J.createConsumer(destination);
Listener L = new Listener();
JC.setMessageListener(L);
…..
```

For Aptech Centre Use Only

Creating and Configuring Message-Driven Beans 3-7

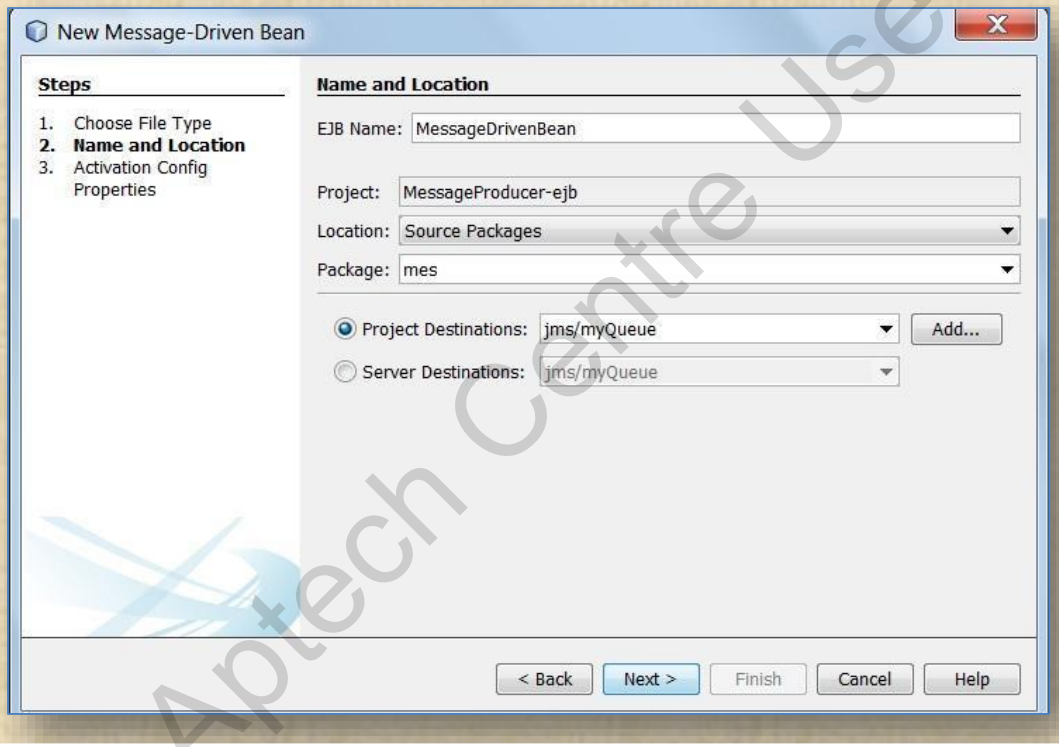
- Following figure shows how to create Message-Driven Beans in NetBeans IDE:



For Aptech Centre Use Only

Creating and Configuring Message-Driven Beans 4-7

- Following figure shows how to configure the destination for the MDB:

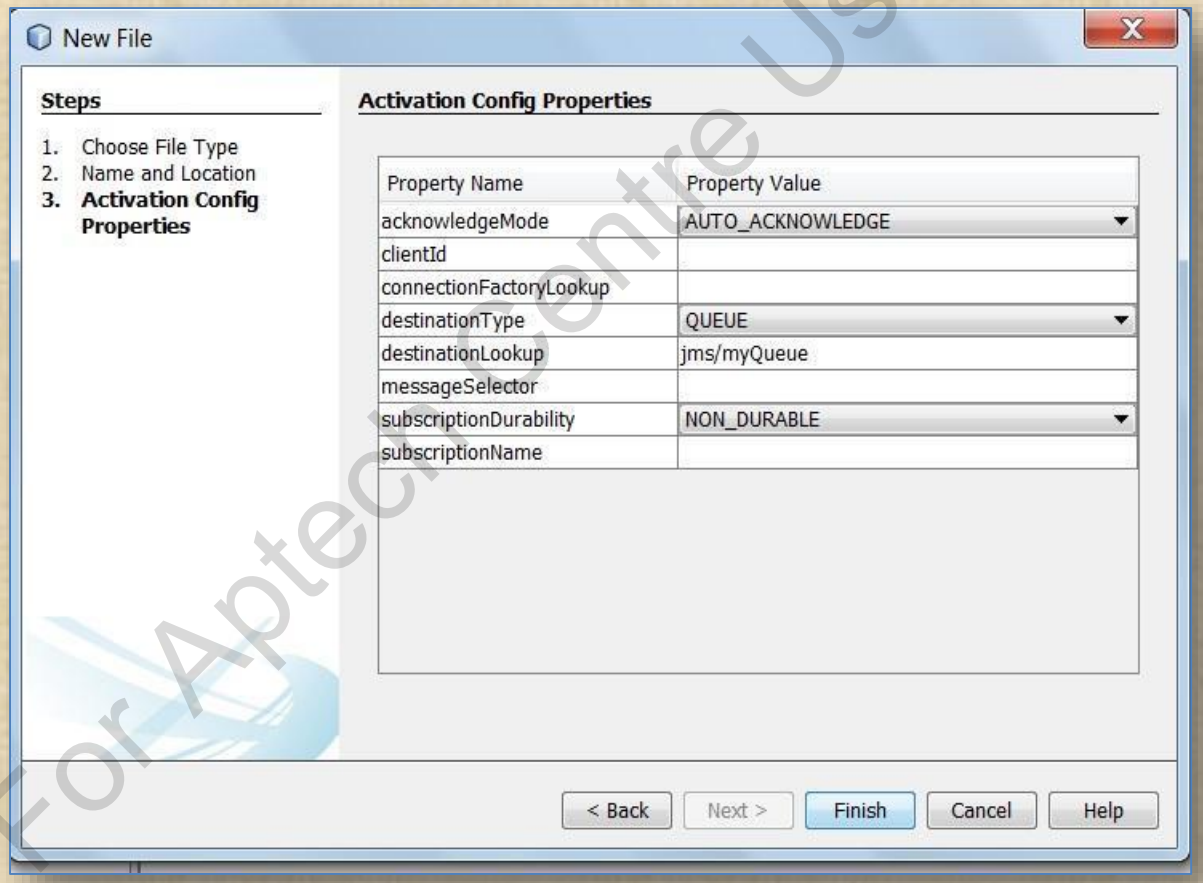


The MDB should process messages received on the destination.

For Aptech Centre Use Only

Creating and Configuring Message-Driven Beans 5-7

- Following figure shows configuring the MDB after selecting the destination:



For Aptech Centre Use Only

Creating and Configuring Message-Driven Beans 6-7

- Following code snippet shows the code for **`onMessage()`** method of the MDB:

```
package mes;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.Message;
import javax.jms.MessageListener;

@MessageDriven(activationConfig = {
@ActivationConfigProperty(propertyName =
"destinationType", propertyValue = "javax.jms.Queue"),

@ActivationConfigProperty(propertyName =
"destinationLookup", propertyValue = "jms/myQueue")
})
```

For Aptech Centre Use Only

Creating and Configuring Message-Driven Beans 7-7

```
public class MessageDrivenBean implements MessageListener
{

    public MessageDrivenBean() {
    }

    @Override
    public void onMessage(Message message) {

        . . .
    }

}
```

For Aptech Centre Use Only

Summary

- JMS API can be used to create communication modules of enterprise applications. Apart from being part of an enterprise application, independent JMS applications can also be created.
- JMS primarily communicates through two messaging models, Publish-Subscribe model, and Point-to-point model.
- The prime focus of JMS API is to provide asynchronous communication.
- Communicating components can define message listeners on the message destinations.
- JMS API Programming model defines how different Java classes are used to implement the JMS application.
- Message-driven beans are defined in Java EE which are invoked on receiving a message on the message destination.

For Aptech Central Use Only