# EBU6501 - Middleware

## Week 4, Day 1: OSGi Architecture and Messaging Services


## Gokop Goteng & Ethan Lau

Queen Mary
University of London



**Original version created by: Dr. Gokop Goteng**

# Lecture Aim and Outcome

- Aim
  - The aim of this lecture is to present the OSGi architecture and different models of messaging services to students.
- Outcome
  - At the end of this class, students will:
    - Understand component programming concepts using OSGi architecture
    - Know the implementation steps of messaging services
    - Implement service-oriented modules

# Lecture Outline

- Open Services Gateway Initiative (OSGi)
  - OSGi Terminologies
- OSGi Architecture
- OSGi Bundles and Service-Oriented Interactions
- Types of OGSi Implementations
- OSGi Deployment
- Advantages of using OSGi
- Technologies that use OSGi Specifications
- Java Messaging Service (JMS) API Architecture
- Java Messaging Domains

# Open Services Gateway Initiative (OSGi)

- OSGi
  - Modular specifications for the Java programming language
  - A **service-oriented** platform
  - Implements **dynamic and distributed component** model for Java specifications
  - Applications and components bundles are remotely managed and administered (started, stopped, installed, etc.) on the fly without a reboot or refreshing the system or services
  - **Automated detection** of the **status** of **services**
  - The specification is created for **managing distributed and cloud resources and services**

# OSGi Terminologies

- **Component**
  - A web resource, web service, module or software package that communicates with other interacting components using its interface
    - Modular and cohesive implementation of services
- **Bundles**
  - OSGi components implemented by OSGi developers
    - A complete component software that can operate on its own
    - An embedded software that is shipped with hardware or other applications

# OSGi Terminologies

- **Modules**
  - Comes from modular programming concept
  - The implementation of a particular functionality of a program
  - Other modules or parts of the program can call it using its interface if they implement acceptable interface rules to each other
- **Services**
  - Comes from service-oriented programming concept
  - Reusable functionalities of software instances using service-oriented architecture and interfaces

# OSGi Terminologies

- **POJI**
  - Plain Old Java Interface (POJI)
    - The native definition of java interface as used in J2SE (Java 2 Standard Edition)
    - Also used in J2EE (Java 2 Enterprise Edition) to implement interfaces as web services
    - Services in OSGi are offered using POJI

- Example of POJI

```
interface Student {
        public void register();
        public void writeExam();
}
```

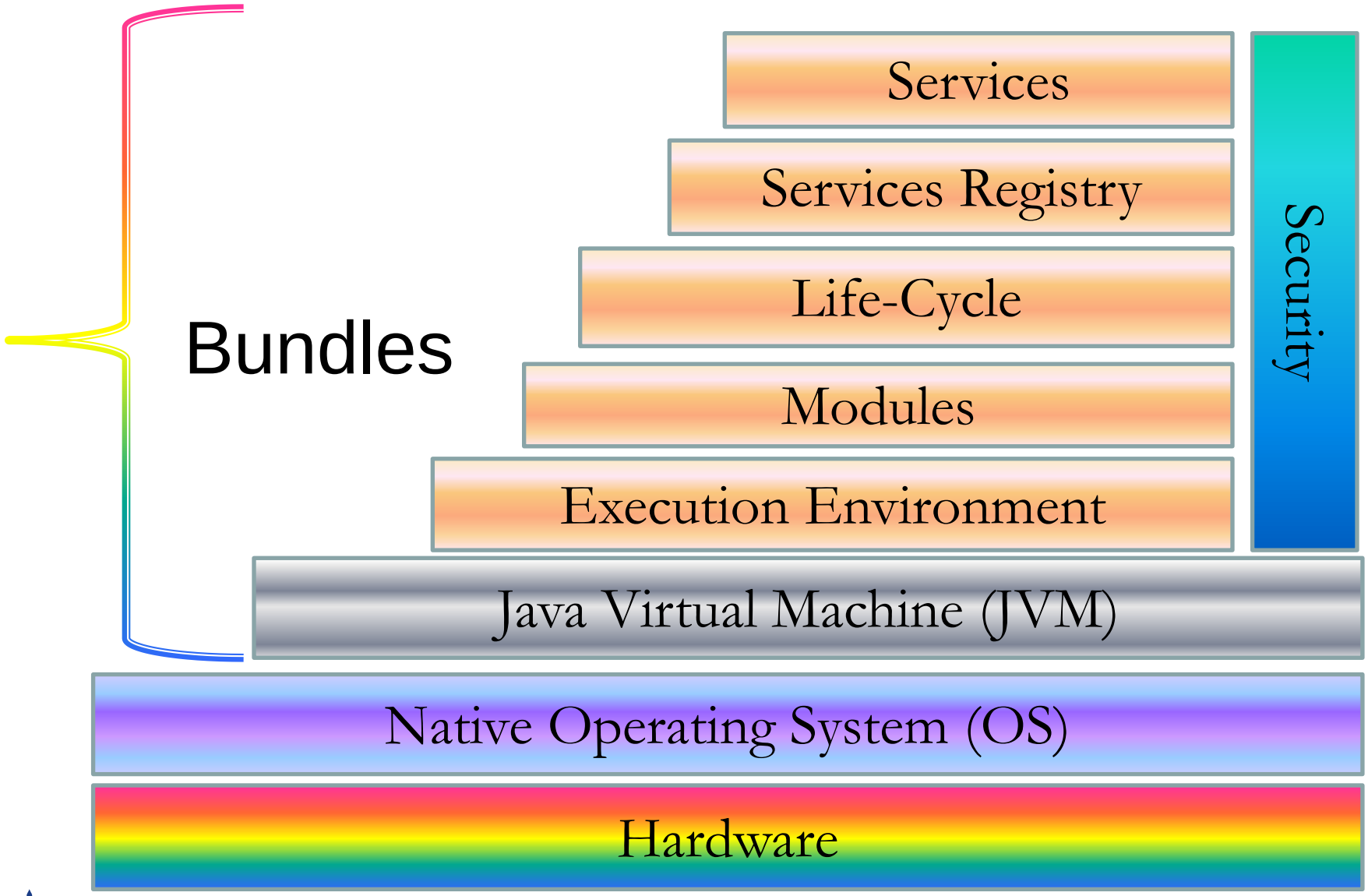
# OSGi Terminologies

- **POJO**
  - Plain Old Java Object (POJO)
    - The native definition of java objects as used in J2SE (Java 2 Standard Edition)
      - It only follows the J2SE model framework
    - It can be used in J2EE (Java 2 Enterprise Edition) to implement objects as part of a web service application
    - It is also used to implement EJB (Enterprise Java Beans) functionalities

# OSGi Architecture

- It is based on a **layered model**
- **Each layer** is a **bundle**
- **Each bundle** implements **interfaces** that **communicates** and **interacts** with its **neighboring bundles**

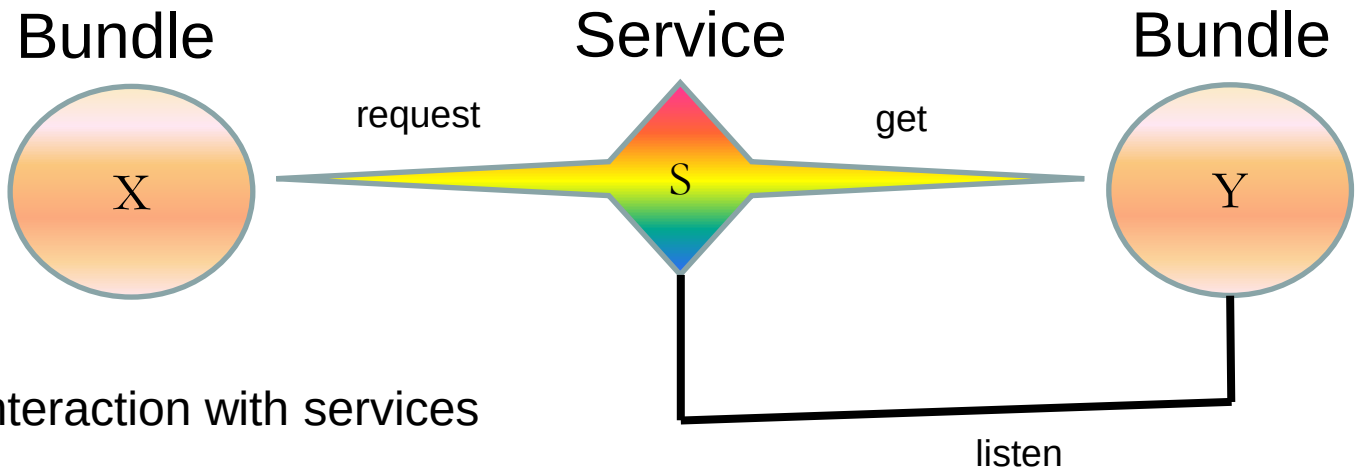
# OSGi Architecture



# OSGi Architecture

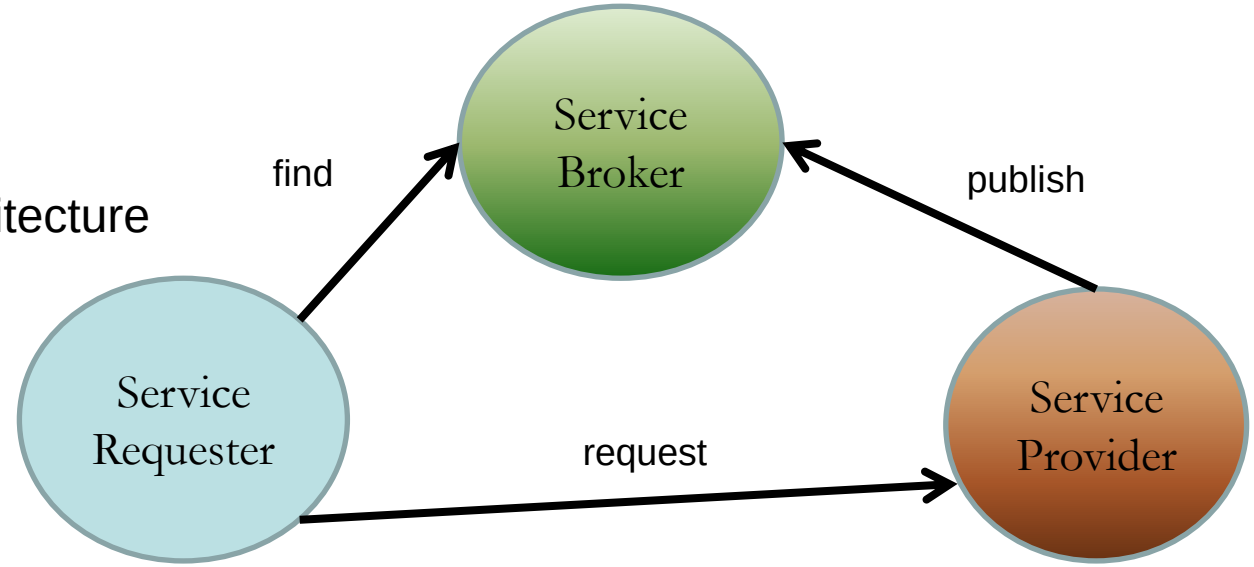
- **Bundles**
  - They are the **OSGi components** written by Java developers based on the OSGi specifications
- **Services**
  - This layer **connects each bundle dynamically** through a publish-find-bind model for POJOs
- **Services Registry**
  - This is the **API for management services** that register services and tracks down services based on their unique service references
- **Life-Cycle**
  - The application programming interface (API) that allows bundles to be **remotely installed, stopped, started, updated, uninstalled** on the fly **without restarting** the services
- **Modules**
  - This layer defines how a bundle can **import and export** codes
- **Security**
  - The security layer that handles **authentication, authorisation** and **protection** of bundles
- **Execution Environment**
  - This layer defines the **methods** and **classes** that are available for a specific platform (JVM, OS, etc.)

# OSGi Bundles and Service-Oriented Interactions




1. OSGi Bundles interaction with services

2. Service-Oriented Architecture



# Types of OSGi Implementations

- There are commercial and open-source implementations of the OSGi specification framework
- There are also different implementations of the OSGi services
- Many projects have implemented software artifacts (products) as OSGi bundles

# OSGi Deployment

- **Bundles** are **deployed** on the **OSGi Framework**
  - OSGi Framework is the collaborative runtime environment of OSGi
  - It is **different** than a Container such as Apache Tomcat Container or Java Application Servers
    - The framework has imports and exports capabilities to start bundles without the need for them to load classes
    - Management of the framework is standardised
- **Bundles run on the same virtual machine (VM)** during deployment, sharing codes coming from modules
- There are management APIs that allow bundles to start, stop, install, uninstall and update other bundles

# Advantages of using OSGi

- **Reduces complexity** for developers as bundles hide their implementation details and developers just need to use their APIs
- The component model of OSGi allows **reuse** of the **modules** and bundles and this **saves the time** of developers as the cost of rewriting codes for the company
- OSGi framework adopts a **dynamic addition** and **removal** of bundles, hence is a **real-world model**
- **Ease** of **deployment** and **dynamic updates**
- It is **secure** for distributed and cloud infrastructures
- Its development and maintenance is **well supported** by companies such Oracle, IBM, Samsung, RedHat and Ericson
  - This means that OSGi will continue to improve/evolve and offer better functionalities in the future.

# Technologies that use OSGi Specifications

- Mobile phones
- Cloud computing
- Grid computing
- Personal digital assistants (PDAs)
- Automobiles
- Industrial automation
- Buildings automation
- Fleet management
- Application servers

# Java Messaging Service (JMS)

## What Is Messaging?

- A **method of communication** between software components or applications.
- A **peer-to-peer** facility: A messaging client can send messages to, and receive messages from, any other client.
- Enables **loosely coupled** distributed communication - sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender. The sender and the receiver need to **know only which message format** and which destination to use.
- Differs from electronic mail (email), which is a method of communication between people or between software applications and people.
- Messaging is used for communication between **software applications** or **software components**.



*Source: https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpp4/index.html*

# Java Messaging Service (JMS)

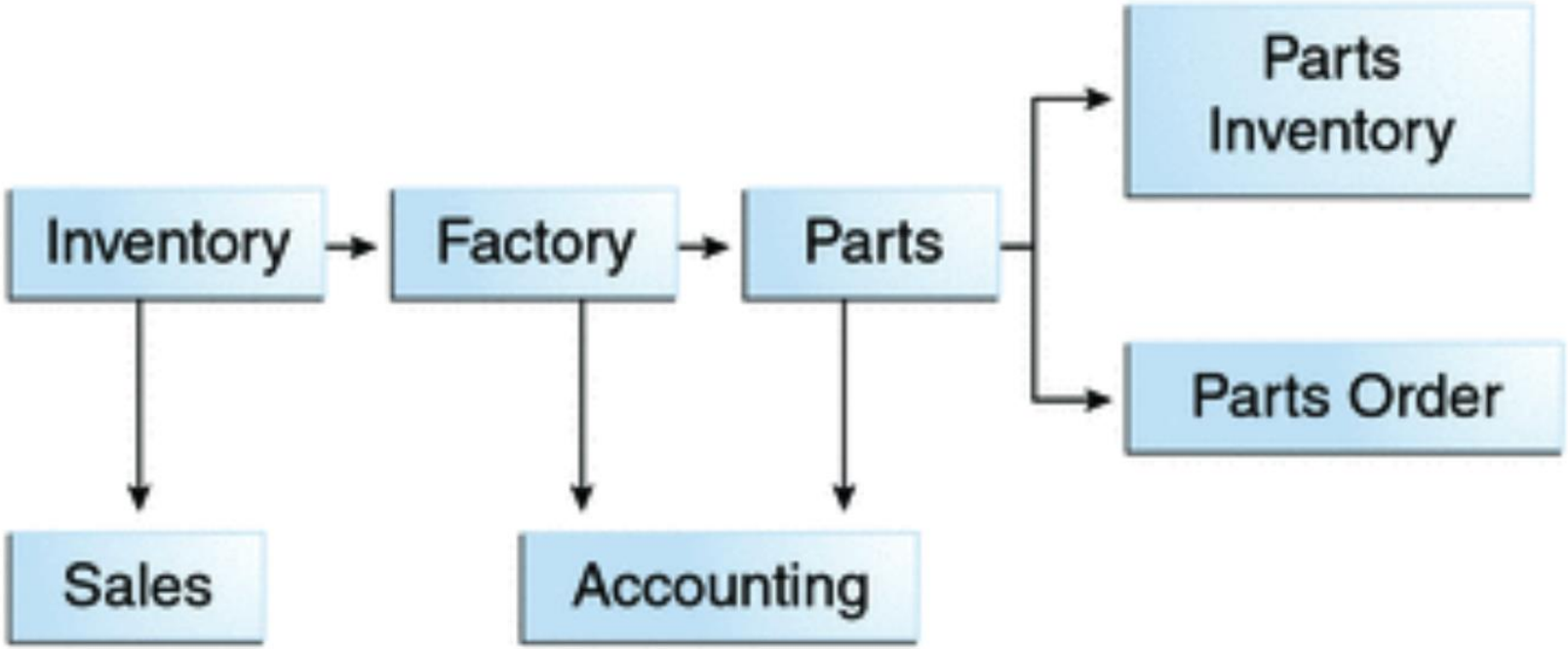
- Is a Java API that allows applications to create, send, receive, and read messages.
- Allow programs written in the Java programming language to **communicate with other** messaging implementations.
- Minimizes the set of concepts a programmer must learn in order to use messaging products but provides enough features to support sophisticated messaging applications.
- JMS API enables communication that is **not only loosely coupled** but also:
- **Asynchronous**: A JMS provider can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them.
- **Reliable**: The JMS API can ensure that a message is delivered once and only once.



*Source: https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpp4/index.html*

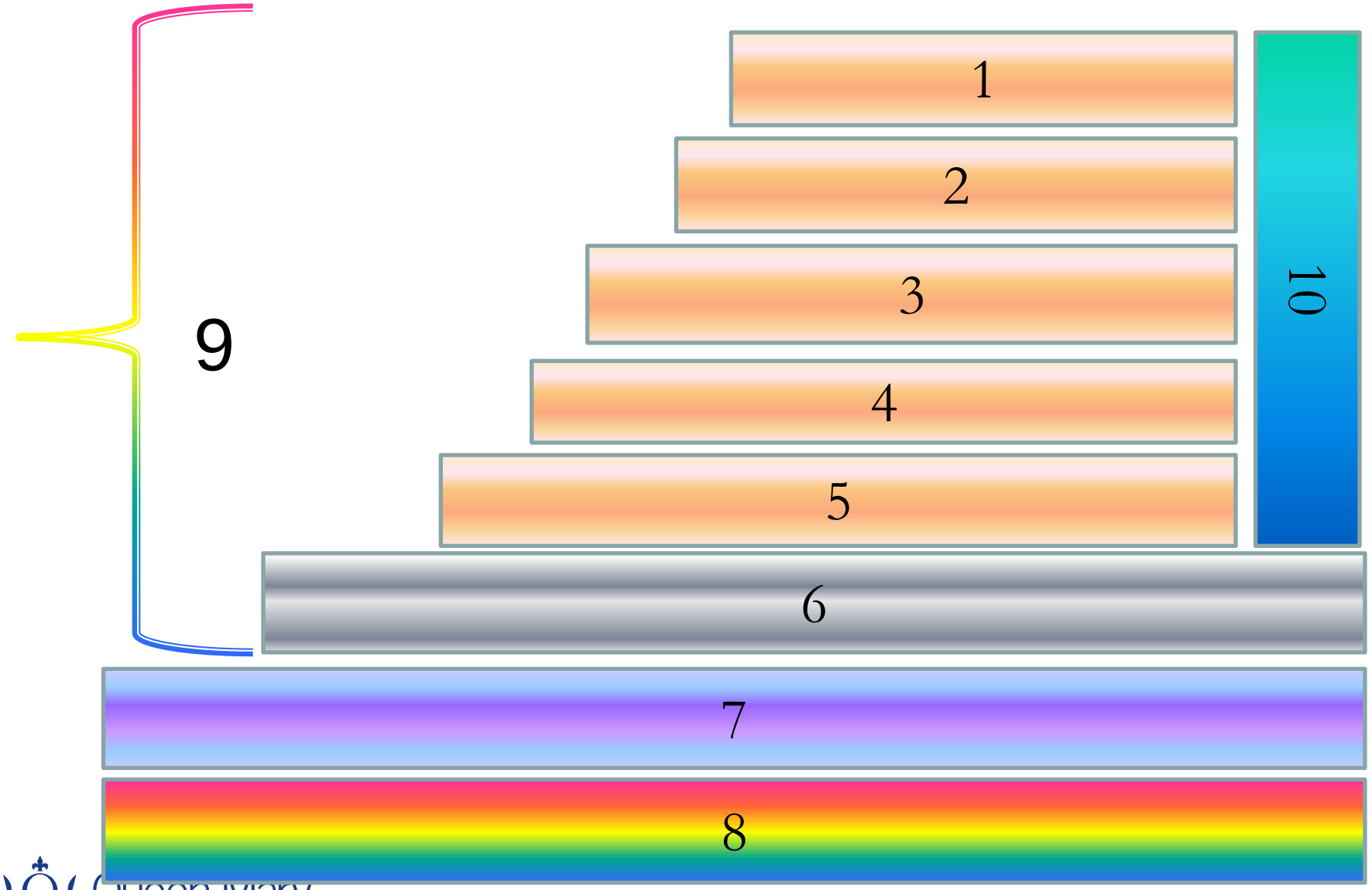
# Java Messaging Service (JMS)

**Example: JMS in enterprise application**




Source: https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpp4/index.html

# Class Task: Close Your Notes and Label The Following



# Class Task: Describe the functions of:

1. Life-Cycle
2. Modules
3. Security