Introduction to OSGi

- Avadhut

Agenda

- What is OSGi?
- What does OSGi offers?
- Specification Process
- Architecture
- Bundles
- MANIFEST.MF
- Life-cycle
- OSGi Implementation Example

What is OSGi?

- Stands for Open Services Gateway initiative
- Is an open standard organization founded in March 1999 that originally specified and continues to maintain OSGi standards
- OSGi defined standards for the development of applications in java programming language
- These standards are referred as OSGi Specifications globally
- Component based framework
- Applications/Components are called as bundles for deployment
- OSGi Specifications have evolved beyong the original focus of service gateways

What does OSGi offers?

- Modular system and a service platform that implements a complete and dynamic component model
- Bundles can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot
- The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly
- Now a days, it’s being used in applications ranging from mobile phones to open-source Eclipse IDE
- Other application areas include automobiles, industrial automation, building automation, PDAs, grid computing, entertainment, fleet management and application servers

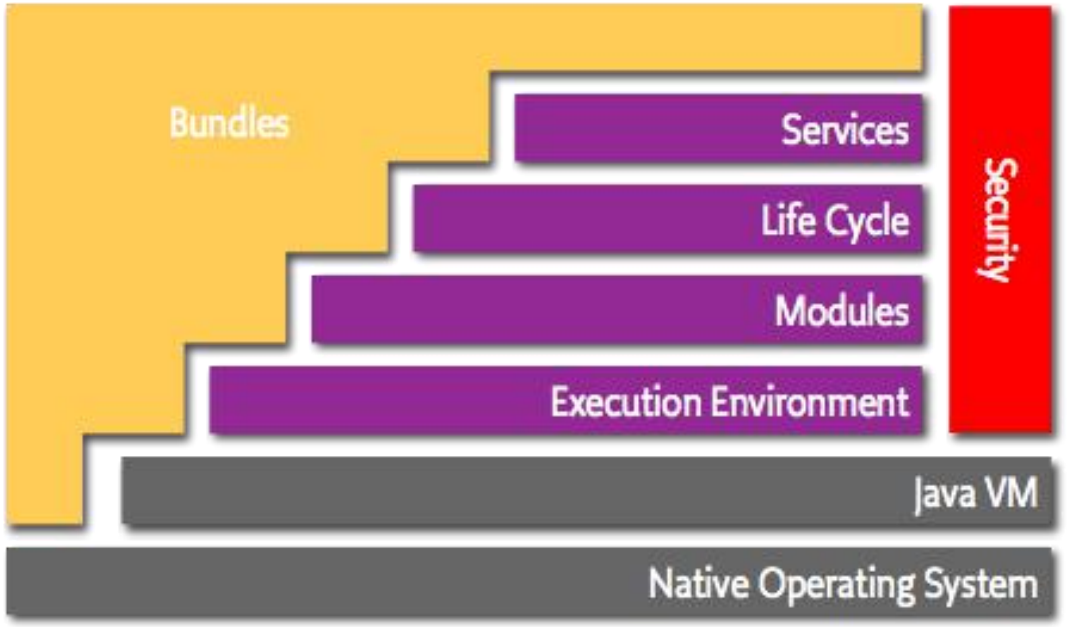
Specification Process

- The OSGi specification is developed by the members in an open process
- It is made available to the public free of charge under the OSGi Specification License.
- The OSGi Alliance has a compliance program that is open to members only.
- Today, there are seven certified OSGi framework implementations.

Implementation	Core Framework
Apache Felix	6
Concierge	5
Eclipse Equinox	7
JBoss	4.2
Hitachi	4.x
Knopflerfish	6
ProSyst	4.2

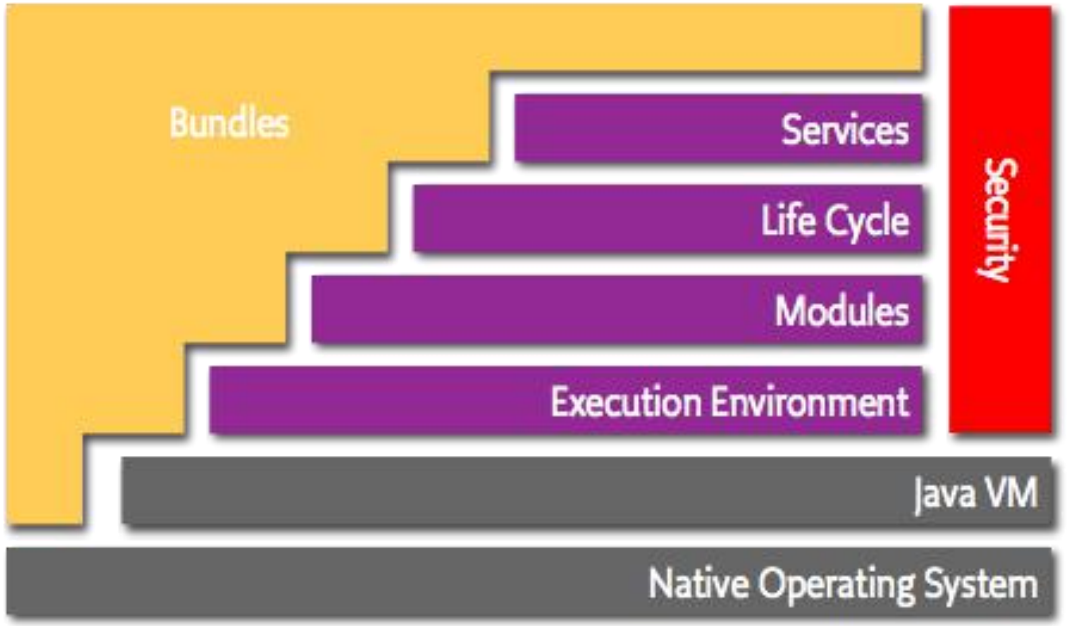
Architecture

- **Bundles** : Bundles are normal JAR components with extra manifest headers.
- **Services** : The services layer connects bundles in a dynamic way by offering a publish-find-bind model for plain old Java interfaces (POJIs) or plain old Java objects (POJOs).
- **Services Registry** : The application programming interface for management services (ServiceRegistration, ServiceTracker and ServiceR eference)
- **Life-Cycle** : The application programming interface for life cycle management (install, start, stop, update, and uninstall) for bundles.



Architecture

- **Modules** : The layer that defines encapsulation and declaration of dependencies (how a bundle can import and export code)
- **Security** : The layer that handles the security aspects by limiting bundle functionality to pre-defined capabilities.
- **Execution environment** : where OSGi implementation can be executed



Bundles

- A bundle is a group of Java classes and additional resources equipped with a detailed manifest MANIFEST.MF file on all its contents
- As well as additional services needed to give the included group of Java classes more sophisticated behaviors, to the extent of deeming the entire aggregate a component
- In simple words, OSGi compliant Java Archive

MANIFEST.MF

- A **manifest file** in computing is a file containing metadata for a group of accompanying files that are part of a set or coherent unit.

- For example, the files of a computer program may have a manifest describing the name, version number, license and the constituting files of the program.

Example Manifest.mf file

```
1 Bundle-Name: Hello World
2 Bundle-SymbolicName: org.kodtodya.helloworld
3 Bundle-Description: A Hello World bundle
4 Bundle-ManifestVersion: 2
5 Bundle-Version: 1.0.0
6 Bundle-Activator: org.kodtodya.Activator
7 Export-Package: org.kodtodya.helloworld;version="1.0.0"
8 Import-Package: org.osgi.framework;version="1.3.0"
```

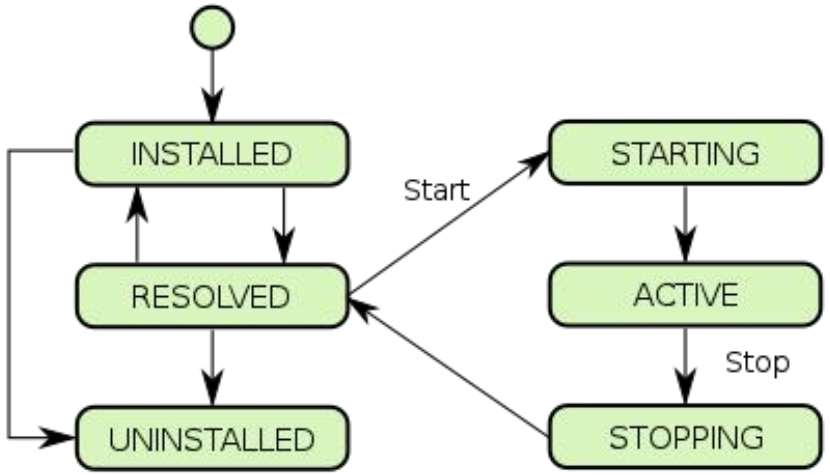
MANIFEST.MF

Example Manifest.mf file

```
1 Bundle-Name: Hello World
2 Bundle-SymbolicName: org.kodtodya.helloworld
3 Bundle-Description: A Hello World bundle
4 Bundle-ManifestVersion: 2
5 Bundle-Version: 1.0.0
6 Bundle-Activator: org.kodtodya.Activator
7 Export-Package: org.kodtodya.helloworld;version="1.0.0"
8 Import-Package: org.osgi.framework;version="1.3.0"
```

- **Bundle-Name:** Defines a human-readable name for this bundle, Simply assigns a short name to the bundle.
- **Bundle-SymbolicName:** The only required header, this entry specifies a unique identifier for a bundle, based on the reverse domain name convention (used also by the java packages).
- **Bundle-Description:** A description of the bundle's functionality.
- **Bundle-ManifestVersion:** Indicates the OSGi specification to use for reading this bundle.
- **Bundle-Version:** Designates a version number to the bundle.
- **Bundle-Activator:** Indicates the class name to be invoked once a bundle is activated.
- **Export-Package:** Expresses which Java packages contained in a bundle will be made available to the outside world.
- **Import-Package:** Indicates which Java packages will be required from the outside world to fulfill the dependencies needed in a bundle.

Life-cycle



Bundle State	Description
INSTALLED	Successfully installed
RESOLVED	All Java classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped.
STARTING	The bundle is being started, the *BundleActivator.start* method has been called but start method has not yet returned. When the bundle has an activation policy, the bundle will remain in the STARTING state until the bundle is activated according to its activation policy.
ACTIVE	Successfully activated and is running; its Bundle Activator start method has been called and returned
STOPPING	The bundle is being stopped. The *BundleActivator.stop* method has been called but the stop method has not yet returned
UNINSTALLED	The bundle has been uninstalled & cannot move to another state.

Sample Example of OSGi Implementation

```
package com.kodtodya;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class Activator implements BundleActivator {
        private BundleContext context;

        @Override
        public void start(BundleContext context) throws Exception {
                System.out.println("Starting Bundle: Hello World...");
                this.context = context;
        }


        @Override
        public void stop(BundleContext context) throws Exception {
                System.out.println("Stopping Bundle: Goodbye Cruel World...");
                this.context = null;
        }
}
```

Thank you …!!!