//jar.camp()

OSGi vs **Jigsaw**

Modularność w Javie

Radek Urbaś

http://urbas.tk/

@rurbas

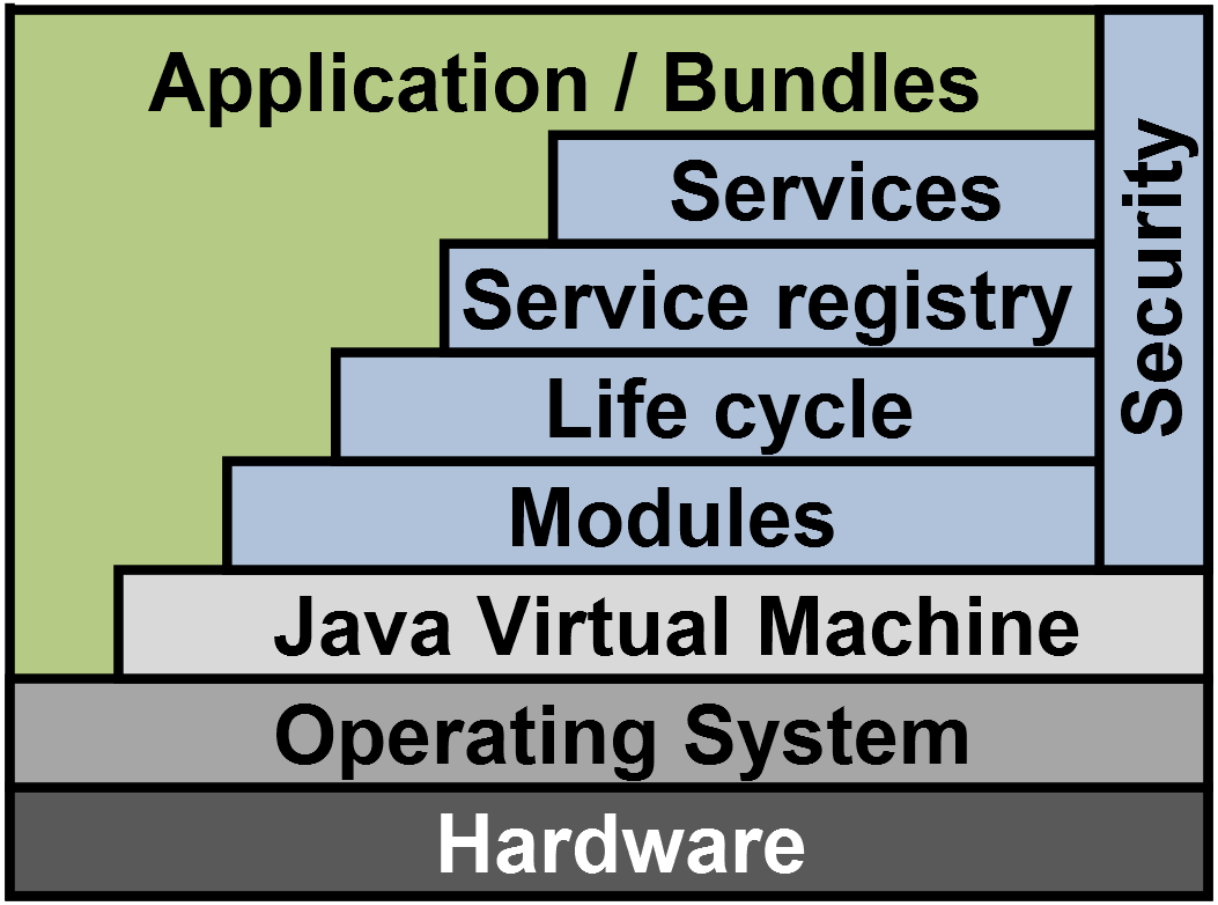
Historia OSGi

- Open Service Gateway initiative
- JSR 8: *Open Services Gateway Specification*: marzec 1999
- OSGi Release 1 (R1): maj 2000
- OSGi Release 2 (R2): październik 2001
- OSGi Release 3 (R3): marzec 2003
- OSGi Release 4 (R4): październik 2005 / wrzesień 2006
- OSGi Release 5 (R5): czerwiec 2012

Historia Jigsaw

- JSR 277*: Java™ Module System*: czerwiec 2005
- JSR 294: *Improved Modularity Support in the Java™ Programming Language*: kwiecień 2006
- Ogłoszenie Project Jigsaw: grudzień 2008
- Pierwotnie planowane do Java 7: lipiec 2011
- Następnie planowane do Java 8: wrzesień 2013
- Według najnowszych informacji przesunięte do Java 9

Architektura OSGi



Aspekty OSGi

- Bundles
  - JAR + manifest
- Services
  - POJOs
- Service Registry
  - API
  - Declarative Services
- Life-Cycle
  - Installed, Resolved, Starting, Active, Stopping, Uninstalled

Aspekty OSGi

- Modules
  - Enkapsulacja
  - Import-Package
  - Export-Package
  - Require-Bundle
- Security
- Execution Environment
  - Na przykład: JavaSE-1.7

OSGi Bundle

- MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: OSGi vs Jigsaw
Bundle-SymbolicName: tk.urbas.osgivsjigsaw
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: tk.urbas.osgivsjigsaw.Activator
Bundle-Vendor: Radek Urbas
Bundle-RequiredExecutionEnvironment: JavaSE-1.7
Import-Package: org.osgi.framework;version="1.3.0"
Export-Package: tk.urbas.osgivsjigsaw
Bundle-ActivationPolicy: lazy
```

OSGi Semantic Versioning

- Version: major . minor . micro (service) . qualifier
- Version range
  - [1.0.0,2.0.0)
  - 1.0.0
- Technical whitepaper http://www.osgi.org/wiki/uploads/Links/SemanticVersioning.pdf
- Eclipse versioning policy http://wiki.eclipse.org/Version_Numbering

Java bez Jigsaw

- Kontrola dostępu do klas, metod, atrybutów:
  - Public
  - Protected
  - Private
  - Default (Package-private)
- JAR
- Brak modularności i wersjonowania modułów
- Classpath Hell
- Pomagają narzędzia typu Maven

Jigsaw

- Pierwotne cele:
  - Rozwiązanie problemów z classpath
  - Wydajność/Start-up (modularyzacja JDK)
  - Wykorzystanie natywnych systemów instalacji modułów
- Szeroka lista wymagań: http://openjdk.java.net/projects/jigsaw/doc/draft-java-module-system-requirements-12
- Na pierwszy rzut oka założenia bardzo podobne do OSGi

Jigsaw - Modules

- Plik **module-info.java**
- Nazwa – zasady analogiczne jak nazwy pakietów
- Najprostsza definicja modułu wg. dokumentacji OpenJDK:

```
module org.astro { }
```

- Moduł z numerem wersji:

```
module org.astro @ 1.2 { }
```

Jigsaw - Exports

- Export:
  - Pakiet
  - Podpakiety nie są eksportowane dopóki nie są wymienione explicite

```
module org.astro @ 1.2 {
    exports org.astro;
}
```

Jigsaw - Requires

- Zależności:
  - Nazwa
  - Wersja (opcjonalnie)

```
module com.greetings @ 0.1 {
    requires org.astro @ 1.2;
}
```

Jigsaw – Entry Point

- Ekwiwalent Main-Class w JAR
- Nazwa klasy implementującej *public static void main*

```
module com.greetings @ 0.1 {
    requires org.astro @ 1.2;
    class com.greetings.Hello;
}
```

Jigsaw – life-cycle

- Compile time
  - Rozwiązanie zależności
  - Kompilacja kodu
  - Pakowanie
- Install time
  - Instalowanie modułu do biblioteki
  - Stworzenie konfiguracji dla modułów *invokable*
- Run time
  - Ładowanie modułu *invokable* do JVM
  - Linkowanie z zależnymi modułami

Jigsaw – kompilacja i uruchomienie

- javac -d <modules> **-modulepath** <modules> -sourcepath <src>
- **jmod**
  - tworzy *module library*
  - instaluje moduły do biblioteki
- **jpkg**
  - Pakuje skompilowane moduły do plików *.jmod
- java **–L <module_library> -m <module_name>**

Jigsaw – re-export

- Zależności mogą być re-exportowane

```
module com.greetings @ 0.1 {
    requires public org.astro @ 1.2;
}
```

- Moduł deklarujący zależność na *com.greetings* może korzystać z klas eksportowanych przez *org.astro*
- Relacja *requires public* jest przechodnia

Jigsaw – versions

- Brak sprecyzowanej wersji
- Dokładne wymaganie
- Wersja minimalna
- Wersja maksymalna

```
module X @ 1.0 {
    requires A;
    requires B @ 1.0;
    requires C >= 1.0;
    requires D < 1.0;
}
```

Jigsaw – permits

- Próba ograniczenia listy modułów które mogą mieć zależność na dany moduł
- Problem re-exportu

```
module org.astro @ 1.2 {
    exports org.astro;
    permits com.greetings;
}
```

Jigsaw – aliases

- Moduł może deklarować inną nazwę pod którą występuje
- Zależne moduły mogą używać aliasu

```
module org.astro @ 1.2 {
    provides alias.name;
}
```

Jigsaw – optional dependencies

- Moduł może używany jeśli nie ma dostępnej zależności (runtime)
- Implementacja modułu musi obsługiwać takie sytuacje
- Zależność musi być dostępna w czasie kompilacji

```
module com.greetings @ 0.1 {
    requires optional org.astro;
}
```

Jigsaw – split-packages

- Powiązanie między modułami
- Wspólny *class-loader*
- Modularyzacja *legacy code*

```
module org.astro @ 1.0 {
    permits com.greetings;
    exports p;
}
module com.greetings @ 1.0 {
    requires local org.astro;
    exports p;
}
```

Jigsaw – views

- To co widzieliśmy do tej pory to *default view*
- Moduł może definiować inne *view* widoczne dla zdefiniowanych modułów

```
module com.greetings @ 1.0 {
    requires org.astro;
    exports com.greetings;
    view com.greetings.internal {
        permits com.anotherconsumer;
        exports com.greetings.another;
    }
}
```

Jigsaw – services

- Moduły mogą rejestrować serwisy
- Nazwa serwisu – interfejs
- Implementacja serwisu
- Brak dodatkowego wersjonowania serwisów

```
module org.astro @ 1.2 {
    exports org.astro;
    provides service org.astro.IWorldService
with org.astro.internal.WorldService;
}
```

Jigsaw – services

- Moduły mogą korzystać z serwisów
- Referencje do serwisów pobiera się przez użycie *java.util.ServiceLoader*

```
module com.greetings @ 0.1 {
    requires org.astro @ 1.2;
    requires service org.astro.IWorldService
    class com.greetings.Hello;
}
```

Jigsaw – services

- Zależności na serwisy mogą być opcjonalne

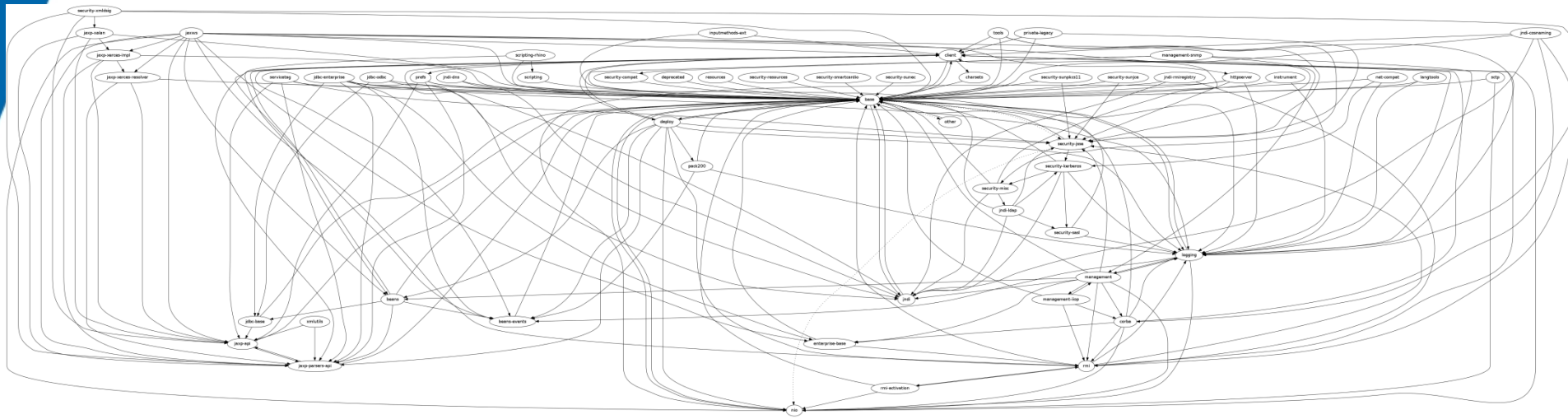
```
module com.greetings @ 0.1 {
    requires org.astro @ 1.2;
    requires optional service org.astro.IWor
    class com.greetings.Hello;
}
```

- Moduły mogą dostarczać wiele implementacji

```
module org.astro @ 1.2 {
    provides service org.astro.IWorldService with
org.astro.internal.WorldService;
    provides service org.astro.IWorldService with
org.astro.internal.AlternativeWorldService;
}
```

Jigsaw – JDK

- JDK 7



Jigsaw – Open JDK 8 b42

- jdk@8-ea
- jdk.auth@8-ea
- jdk.base@8-ea
- jdk.compat@8-ea
- jdk.compiler@8-ea
- jdk.corba@8-ea
- jdk.cosnaming@8-ea
- jdk.crypto@8-ea
- jdk.deploy@8-ea
- jdk.deploy.core@8-ea
- jdk.desktop@8-ea
- jdk.devtools@8-ea
- jdk.httpserver@8-ea
- jdk.instrument@8-ea
- jdk.jaxp@8-ea
- jdk.jaxws@8-ea
- jdk.jdbc@8-ea
- jdk.jdbc.rowset@8-ea
- jdk.jndi@8-ea
- jdk.jre@8-ea
- jdk.jta@8-ea
- jdk.jx.annotations@8-ea
- jdk.kerberos@8-ea
- jdk.logging@8-ea
- jdk.management@8-ea
- jdk.management.iiop@8-ea
- jdk.prefs@8-ea
- jdk.rmi@8-ea
- jdk.scripting@8-ea
- jdk.sctp@8-ea
- jdk.security.acl@8-ea
- jdk.smartcardio@8-ea
- jdk.snmp@8-ea
- jdk.sunec@8-ea
- jdk.sunmscapi@8-ea
- jdk.tls@8-ea
- jdk.tools@8-ea
- jdk.tools.base@8-ea
- jdk.tools.jaxws@8-ea
- jdk.tools.jre@8-ea
- jdk.xmldsig@8-ea
- jdk.zipfs@8-ea
- sun.charsets@8-ea
- sun.localedata@8-ea
- sun.resources@8-ea

Jigsaw - repositories

- jmod
  - add-repo URL
  - del-repo URL
  - repos
- Eksperymentalna funkcjonalność

Project Penrose

- OSGi / Project Jigsaw interoperability
- http://openjdk.java.net/projects/penrose/

Q & A

- Prezentacja i kilka przykładów https://github.com/rurbas/jarcamp_jigsaw