Path to OpenJDK9 and beyond

DevConf, Feb 2014

Jiří Vaněk

Path to OpenJDK9 and beyond

DevConf, Feb 2014

- **Bit of history**
- **JDK8**
- **Features, jeps, projects and icedtea**
- **JDK9**

Index

One page of history

devconf 2012 - http://rvokal.fedorapeople.org/devconf2012/jvanek-jdk8.pdf

1992 - started at Sun labs as project Oak

1996 - Version 1 publicly released

Cca **1998** - plugin, JIT, GNU classpath

Cca **2001** - HotSpot, javaws

Cca **2006** - JDK6, OpenJDK, IcedTea,

Cca **2009** - acquisition by Oracle, IcedTea-Web

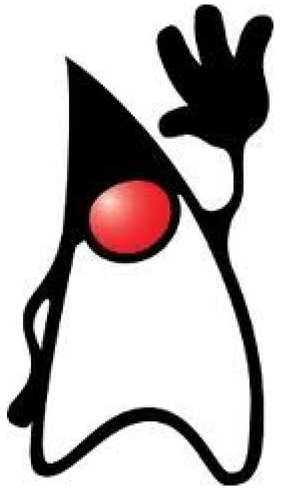
Cca **2011** - OpenJDK7,

- merging Of IcedTea to OpenJDK, jigsaw to JDK9

Middle of **2013** – should beOpenJDK8....

- December 2013 forked JDK9

March **2014** Release of OpenJDK8 ?



Well.. two of them

- OpenJDK7 – released in time, July 2011
    - JVM support for **dynamic languages (invoke dynamic)**,
    - Small language changes (grouped under a project named **Coin**):
    - Concurrency utilities (**fork/join framework**)
    - New file I/O library
    - Care is taken of **community**
    - It **was** conservative change at the end!

OpenJDK8

- JDK7 released in time (mid 2011)
    - Some (mayor) features dropped **after** development freeze
    - Very conservative
    - Nearly in time;)
- This lead to set JDK lifecycle to two years release cycle
    - Some doubts in community
    - Dropping features during 2013
    - Several delaying
    - Errors found in RC
        - 18.3 2014?
            - Cloned form 7 2012/04/26
- Its better then it look likes

OpenJDK8 – sources and build

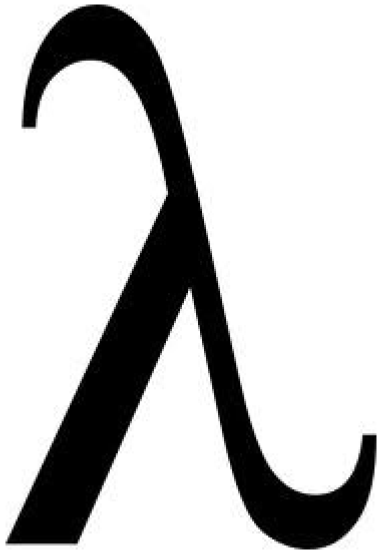
- Bundles available at http://jdk8.java.net/download.html,
- Packed in most distros
- Or you can get bleeding edge by mercurial:
    - *hg clone* http://hg.openjdk.java.net/jdk8/jdk8
        - *+ sh ./get_source.sh*
    - Or *hg clone http://hg.openjdk.java.net/jdk8/jdk8/* {jdk,corba,hotspot,jaxp, jaxws, langtools,nashorn}
        - All mayor projects already merged in
    - *./configure; make*
        - Usage of autotools was greate mprovement in jdk8
        - (jdk7 needed to compile it)
    - http://hg.openjdk.java.net/jdk8/build/raw-file/tip/README-builds.html
- For updates branches you can use
    - http://hg.openjdk.java.net/jdk8u/jdk8u

OpenJDK8

- Conservative release again?
- Project Lambda
- Rest of project coin
- NIO2 (connected with lambda!)
- New build system
- Various (full)filled jeps
    - Not all made it in!
    - But many are going to be backported to 7

Lambda – what it is?

- What is lambda?
  - Already reached documentation
  - http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- An anonymous function
  - Parametrize behavior
  - Treat behavior as data
  - Provides closure mechanism
- Provides:
  - More effective code
  - Parallelism



Lambda

- -> operator to declare lamdas
    - (int x, int y) -> x+y
    - () -> 42
        - Never instantiated (invoke dynamic)
- Defender methods in interfaces
- Collections have ***stream*** method, which provide access to lamdas
    - filter
    - map
    - ForEach
        - Which works with closures
- Parallelism from jdk itself
    - Spliterator
        - http://download.java.net/jdk8/docs/api/java/util/Spliterator.html
- Much more in my last year presentation ;)
    - http://jvanek.fedorapeople.org/underTheRoofOfJDK8.pdf

Small Features of OpenJDK8 – Miscellaneous

Date and Time API

- Based on JodaTime

Base64 Encoding/Decoding

- Unify all 5(!) internal implementations to one improved java.util public api

Limited doPrivlidged

- Enable asserted code to run without full access-control stack walk to check for permissions
- Possible security impact?

Concurrency Updates

- Scalable updatable variables, cache-oriented enhancements to the
  - ConcurrentHashMap API, ForkJoinPool improvements,

Network Interference Aliases Events and Defaults

- Make java SE to work on devices with multiple network (or hierarchical) interfaces
  - Listen to changes **of** the devices configuration
  - Select device

Small Features of OpenJDK8 - JavaDoc

DocTree API

- Enable access to syntactic elements of javadoc

Javadoc to javax.tools

- Allowing execution of javadoc via api
  - Instead of new process “javadoc”

DocLint

- Detect errors in javadoc in compile time
  - Bad syntax, Bad html, Bad annotations, Bad targets, ...

Small Features of OpenJDK8 - built

Autoconf-Based built system

- Introduce autoconf (./configure-style) build setup, refactor the Makefiles to remove recursion
  - Increase build speed radically
  - Simplify build-system source code (Makefiles, etc.)
  - Simplify work for developers
- Result of M4 compilation (generated ./configure script) **will be checked** to repository

Smal VM

- Support the creation of a small VM that is no larger than 3MB.
  - Make necessary modifications so that we can optionally build a small VM no larger than 3MB. (now client and server VMs are around 6 and 9MB)

Compact profiles

- Specify profiles, so java aps will not need to load whole JDK
  - Eg no-gui app will no longer load swing from rt.jar
  - Already doen in some distributions
- Jigsaw?
    - Compact1 java.{io,lang,math,nio,security,text,util,crypto,net}
    - Compact2 java.{rmi,sql,transaction,xml}, org.w3c.{sax,dom}
    - Compact3 java.{lang,management,naming,security,sql,util,xml,tools, org.ieft.jgss}
      - The same packages in profiles have mostly empty intersection of subpackages

Small Features of OpenJDK8 – Lang

Annotation on Java Types

- Now annotations are allowed for
  - Classes
  - methods

- **Access to Parameter Names at Runtime**
- Java have access to all names except parameters' ones

Repeating annotations

- Now annotations are allowed for language member just once time per annotation

Parallel Array sorting and another Lambda collected features

- New methods to Arrays class like
  - *public static parallelSort*
- Dependence and similarity on Lambda with Fork and Join framework
  - Implementation based on ParallelArray framework

Small Features of OpenJDK8 – Lang - JVM

Remove permanent generation

- Part of jrockit and hotspot conversion
  - Jrockit customers do not need to tune permanent generation
  - There should be no need for it in JVM too

Reduce Core-Library Memory Usage

- Reducing heap size occupied by core libraries without lost of performance
  - Eg Reduce base Object size

Reduce Class Metadata Footprint

- Reduce HotSpot’s class metadata memory footprint in order to improve performance on small devices
  - Many manual actions
  - Better usage of automated optimization

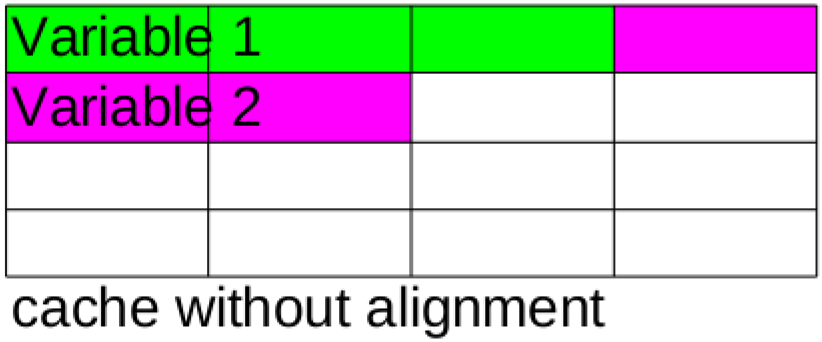
Small Features of OpenJDK8 – Lang - JVM

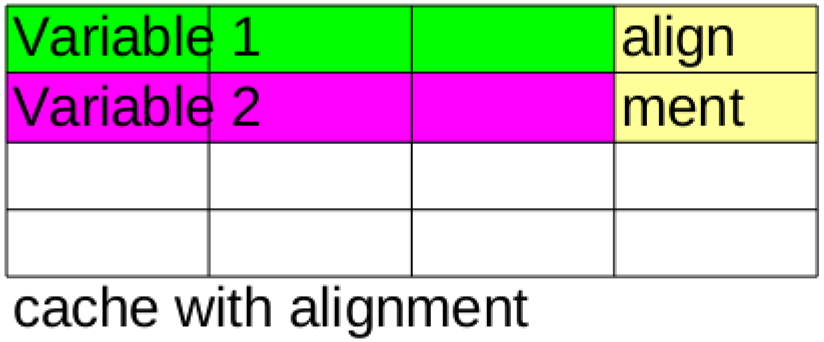
Enhanced Verification Errors

- Bytecode is verified in JVM, but in case of failure exceptions are to vague or misleading
- Although rare, those will be enhanced

Reduce Cache Contention on Specified Fields

- Find way how to specify fields which can spread across multiple cores or share lines in caches
- By aligning the fields





Small Features of OpenJDK8 – Cryptography

Configurable Secure Random-Number generation

- */dev/random* and is blocking until enough entrophy
- */dev/urandom* can provide good randomnes without blockin
- Except */dev/** also custom algorithms

Overhaul JKS-JCEKS-PKCS12 Keystore

- Migrate current keystore format to standard PKCS#12

Stronger Algorithms for Password-Based Encryption

- Lot of current algorithms i in JDK are legacy
  - DES,RC2
- Need to add new ones
  - SHA-2, PBE

Various suites

- **AEAD Cipher suite**
- **NSA Suite B**
- **SHA-224**
- **PKCS#11 crypto provider for Windwos64b (finally!)**

Small Features of OpenJDK8 – dropped features

Collections Enhancements from Third-Party Libraries

- Goal si not to eliminate v3rd parties, but to learn form them and use what can be used
- Dropped 14.1.2013

G1 GC: Reduce need for full GCs

- Enhance G1 so that it does not r4ely on full GCs to perform class unloading or any other critical operation
  - Shorter pauses during GC
  - There should be no need for it in JVM too
- Dropped 14.1.2013

New HTTP Client

- Current URLConnection done with Legacy (ftp, gopher) protocols in mind
- Goal si to made new extensible Api
- Based on NIO
- Keep EE in mind
- Dropped 14.1.2013

Projects, Jeps and IcedTea

- Projects
  - **Larger then JEPs**
    - Are voted for
  - Continuous
    - work over JDKs
    - transition across versions
    - (continuous) Merging
  - Everybody can propose - http://openjdk.java.net/projects/
    - Icedtea... quite a different project
    - Updates - even more different project
  - Ports
  - Needs to be merged
    - **May never ends in JDK**
- Jeps
  - **Smaller features (Jdk Enchancement Propose)**
    - Are decided
    - **Are going tobe in JDK**
  - Are about to be completed and done
  - Everybody can propose - http://openjdk.java.net/jeps/1
  - Are strictly targeted
  - **Are done in live branches**

JDK9

- Forked December 2013
  - Not much done yet...
  - JEP **acceptance** in **progress**
  - Many **projects** already did merging
    - Into jdk7
    - Into jdk9

JEPs – all are canidates

- **Process API Updates** - core/libs
  - Improve the API for controlling and managing operating system processes
- **Collections Enhancements from Third-Party Libraries** - core/libs
  - Evolve the Java Collections Framework by adopting common and popular functionality from third-party libraries.
  - 8?
- **New HTTP Client –** core/net
  - Replace legacy HttpURLConnection
  - 8?
- **Additional Unicode Constructs for Regular Expressions** – core/libs
  - Unicoded regexps without pain
- **Network Interface Aliases, Events, and Defaults** – core/net
  - allow Java SE to work well in devices with multiple network interface types (e.g., both wifi and cellular)
  - 8?

JEPs – all are canidates

- **More-prompt finalization –** vm/gc
  - Improve the promptness of finalization by use of multiple finalizer threads and/or aggressive management of the finalizer queue
- **Increase the Client VM's Default Heap Size** – vm/gc
  - Increase the default maximum heap size of the client JVM so that most client applications can run without tuning
  - :)
- **Improve Contended Locking** – vm/rt
  - Significantly improve contended-locking performance in HotSpot.
- **Reduce GC Latency for Large Heaps** – vm/gc
  - Improve the performance of applications that require large heaps, of up to 32GB, by reducing garbage-collector latency
    - Shenandonah

JEPs – all are canidates

- **Cache Compiled Code –** vm
  - Save and reuse compiled native code from previous runs in order to improve the startup time of large Java applications
- **Improve Fatal Error Logs** – vm
  - Improve HotSpot’s fatal error logs (hs_err files) by including additional historical information and also some context-dependent information
- **Crypto Operations with Network HSMs** - vm
  - improve support for Hardware Security Modules
- **G1 GC: Reduce need for full Gcs** – vm
  - Enhance G1 so that it does not rely on full GCs to perform class unloading or any other critical operations.
  - 8?
- **Unified JVM Logging –** vm
  - Introduce a common logging system for all components of the JVM.

JEPs – all are canidates

- **G1 GC: NUMA-Aware Allocation** – vm/gc
  - And
- **Enable NUMA Mode by Default When Appropriate –** vm
  - To improve the out-of-the-box performance on non-uniform memory accesses
- **Compiler Control –** vm/comp
  - Add the possibility of changing the option sets during run time.
- **Policy for Retiring javac -source and -target Options** – core/lang
  - In JDK 9 and going forward, javac will use a “one + three back” policy of supported source and target options
    - process class files of all previous JDKs, going back to version 45.3 class files generated by JDK 1.0.2, which first shipped in 1996
- **Collection Literals** – core/lang
  - Arrays support for lists
  - Eg List<Integer> list = #[1, 2, 3];

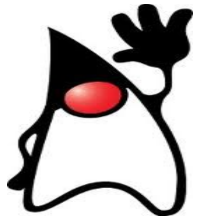
JEPs – all are canidates

- **Serialization 2.0 –** core/lang
  - Research/posted
  - Current approach
    - Security issues
    - Underestimate objects
- **PowerPC/AIX Port**
  - Funded!
  - Add Linux/PowerPC64 and AIX/PowerPC64 to the set of supported OpenJDK platforms.
- **Shenandoah: An Ultra-Low-Pause-Time Garbage Collector** – vm/gc
  - Reduce GC pause times on extremely large heaps by doing evacuation work concurrently with Java threads and making pause times independent of heap size.
  - with a heap of 20GB or less or if you are running with fewer than eight cores, you are probably better off with one of the current GC algorithms.

Projects - icedtea

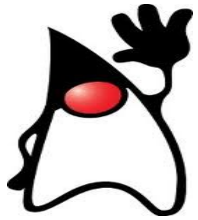
- Actually wrapping project around jdk
    - All Linux distros have java wrapped by IcedTea
- Was founded in 2007 by **RedHat** as reaction to unhappy state of OpenJDK6
    - To make **OpenJDK buildable** without proprietary blobs
        - Used GNU Classpath, **rewrote** proprietary **blobs**
        - To be usable in **pure-opensource** projects as eg Fedora, Ubuntu, Debian, OpenSuse...
    - Make it **easily build-able**
    - Support **alternative JVMS** (Zero,Shark, CACAO, ...)
    - Support for **new plugin**
    - Make OpenJDK easily extensible and fixable
    - Serve as **bridge** between **community** and **upstream**

Projects - icedtea



- The experiment is **successful**:
  - Bootstrap with **GNU Classpath/OpenJDK**
  - **Support** for multiple architectures via **alternative VM** (Zero,Shark, CACAO, JamVM...)
  - Huge number of **excellent patches** (often Linux-ones)
  - Members of IcedTea have pushed numerous patches to upstream
  - It is really easy to make contributions
- Rewriting of **plugin/Javaws** lead to **IcedTea-Web** open-project
- **VisualVM** profiling and debugging tool is replaced by **JVMTI** and **Thermostat** (and some more tools from JBoss family, eg. Byteman)

Projects - icedtea



- However
    - It seems that IcedTea is no longer needed
- Jdk6 (for which it was designed) is dead
    - JDK7, although do exists only thanks to icedtea is spread as icedtea
- OpenJDK build itself cleanly
- IcedTea-Web is separate “project”
- Thermostat is separate “project”
- Most IcedTea “alternative ways” are turned into regular projects
- Its quite easy and strigt forward to contribute
    - Mostly unwillingness of few individuals is keeping icedtea alive
- Icedtea do not have jeps and projects

JDK 9 projects

- Shenadonah
  - Previous lecture
- Nashorn
  - Rewritten javascript engine
  - Replaced rhino
- Sumatra
  - Api for using GPU
- Jigsaw + penrose
  - Penrose is tracking jigsaw and osgi compatibility
- Development changes
  - Handling of updates
  - Handling of community :)

JDK 9 projects

- **ports**

- Macosx
- Ppc64
- Aarch64
  - Aarch64 simulator
    - crosscompialtion
  - Aarch64 virtual machine

- OpenJFX
  - Javafx opensource
- Coin
  - Remaining tasks from openjdk8
- Graal
  - Java compiler written in java

JDK 9 projects - ports

- **Zero**
  - Long time living **not-a-port**
    - Really running everywhere where gcc is
  - Why ports?
    - Zero is C/C++ template VM
    - Much slower then assembly language tempaltes
    - Shark – zero's jit – is long-term broken
- **Macosx**
  - License issues with parts of JDK
    - Awt
  - Many many different parts in MacOS
- **Ppc64**
  - IBM's iniciative
  - Real hotspot assembly templates
- **Aarch64**
  - Aarch64 simulator used **before** the actuall harware protoypes
    - http://hg.code.sourceforge.net/p/smallaarch64sim
    - Cross compilation
  - Later Aarch64 virtual machine, and later real HW. Always new bugs :)
  - Real hotspot assembly templates
- **BDS, Hiku, MIPS....**

JDK 9 projects - Nashorn

- lightweight high-performance JavaScript runtime in Java with a native JVM.
- This Project intends to enable Java developers embedding of JavaScript in Java applications via JSR-223
- standing JavaScript applications using the jrunscript command-line tool.
- utilize the MethodHandles and InvokeDynamic

JDK 9 projects - Sumatra

- Take advantage of
    - graphics processing units (GPUs)
    - accelerated processing units (APUs)
- whether they are discrete devices or integrated with a CPU--to improve performance.
- leveraging the new Java 8 Lambda language
- provide guidance for other JVM-hosted languages such as JavaScript/Nashorn, Scala and JRuby.

JDK 9 projects - OpenJFX

- Oracle announced that it would **donate** the JavaFX toolkit to the open source community and by November 2011 the OpenJDK Community had **agreed to take it** on.
- The project intends to file a JSR in the Java SE 9 timeframe and hopes to eventually be part of the JDK proper.
- The goal of OpenJFX is to build the next-generation Java client toolkit.
- Finally get rid of java-plugin
    - IcedTea-Web?

JDK 9 projects - Grail

a quest for the JVM to leverage its own J

- to expose VM functionality via Java APIs.
  - write in Java a dynamic compiler and interpreter for a language runtime
  - highly extensible dynamic compiler uses features of Java
  -
- Multi-language interpreter framework
  - Java will be just one member in the family of supported languages.
- Performance

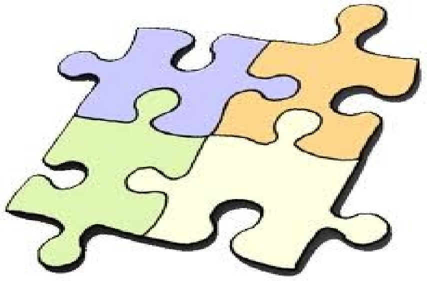
IcedTea-Web

- Is not an project
  - Hosted on classpath.org
  - Initiated as part of IcdTea
- Only known opensource java-plugin
  - Awt-less plugin for macos and mobile devices comming from comunity
- Only known opensource and alive javaws client
  - Ligting tallk!
- Is trying to be project
  - Icedtea only patches
    - Making some private stuff protected
  - Upstreaming unsuccessful
  - Jdk9 – new api for plugins?
  - Jdk8 the original patch?

Thermostat

- Is not an project
    - Again hosted on classpath.org
- Labs yesterday!
- Monitoring and instrumentation tool for the Hotspot like JVMs, with support for monitoring multiple JVM instances on multiple hosts, optionally in a cloud environment.
    - We want a tool that allows users of IcedTea/OpenJDK to monitor running JVMs, especially remote JVMs.
    - We have openjdk sources
    - We have kernel sources
    - ...?
- Replacement and improvement for
    - VisualVM
    - JConsole
- pluginable
- eclipse plugin

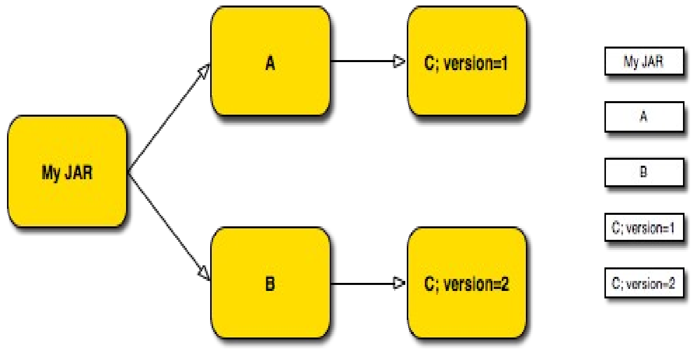
Project Jigsaw



- **modularisation of Java platform**
- Continuous integration with OSGI via project penrose (approved Jan/Feb 2012)
- Current JDK is **monolithic** and **huge** (more then 100MB)
  - “Hello world” in Maven can have up to 4MB
- Modules **will replace class path** (unix and Maven like approach)
  - Eg. by Maven - Build-time, install-time, test-time and run-time
  - Eg. from packages – shared versions and modules
  - Inspired and compatible with OSGI
- Modularization of native-binary parts of JDK is not sure even for JDK9 :(

Project Jigsaw

- What it should solve:
  - JAR hell
    - Too many **transitive** references
    - Dependence on **multiple versions**



    - Unmanaged Dependencies (only via **classloaders** hierarchy) => ServiceLoader API
    - Stomping – **name clash** in jars
    - Use of **private code** – no longer possible?

Project Jigsaw

- Platform fragmentation
    - Will allow unification of SE x ME (and EE?)
        - There are complicated license issues for EE
    - **No more rt.jar** (separate jars for separate technologies – swing, xml, language...)
- **Startup** performance
    - (**pre**)loading only what needed (**pre-downloading**?)
    - Already JDK6 have lazy loading of parts of RT (but still whole RT must be available)
- Integration with **native packaging** systems
    - Rpm/deb... inspiration <-> compatibility
        - Windows will get shorten? O:)
    - Support for better cooperation with native modules also in JDK9?
- Package granularity
    - Libraries consisting from more and more jars?
    - Can lead to “new” “**modules hell**” ?

      (lot of work done to not so)
- What is module?

Project Jigsaw

- Descriptors are **plain-text .java** files “inside” module/jar
- Module declaration:

```
module a.b @ 1.0 {
 requires        c.d @ /* Use v2 or above */ >= 2.0 ;
 requires service e.f;

 provides        g.h @ 4.0;
 provides service i.j with k.l;
 exports  m.n;
 permits  o.p;
 class    cc.dd;

 view a.b.c {
  provides        q.r @ 1.0;
  provides service s.t with u.v;
  exports  w.x;
  permits  y.z;
  class    aa.bb;
 }
}
```

Maven ---> (pom compatibility)

---->jar (classical, classpath re-usable jar)

---->jmod

---->rpm

---->deb

---->war,ear (JDK 9?)

Project Jigsaw - build

- hg clone http://hg.openjdk.java.net/jigsaw/jigsaw
- cd jigsaw
- bash get_sources.sh
- ./configure
- make all

Project Jigsaw - build

- Result
  - build/linux-{i586 amd64}/jdk-module-image”.
- In bin are new tools
  - jmod
  - jpkg
- The ”lib/modules”
  - folder contains a myriad of modules.
  - The JDK is no longer this huge ”rt.jar” with a gravity of JARs around
  - it is now a set of modules.
  - Each module contains (except classes)
    - index
    - metadata

Project Jigsaw – first module

- **mkdir** -p *sources/fact/fact*
- **mkdir** *modules*

- *fact/Factorial.java*

```
package fact;

public class Factorial {
  public static int factorial(int n) {
    if (n <= 0) { return 1; }
    else        { return n * factorial(n - 1); }
  }
}
```

- *fact/module-info.java*

```
module fact @1.0 {
   exports fact}
```

- javac -d modules -modulepath modules -sourcepath sources \

```
    sources/fact/module-info.java \
    sources/fact/fact/Factorial.java
```

Project Jigsaw – first dependent module

- *hello/Main.java*

```
package hello;

import static fact.Factorial.factorial;

      public class Main {
        public static void main(String... args) {
          System.out.println(factorial(10));
        }
      }
```

- *hello/module-info.java*

```
module hello @1.0 {
  requires fact @1.0;
  class hello.Main;
}
```

- javac -d modules -modulepath modules -sourcepath sources \
  sources/fact/* sources/hello/*
- java -m hello

Project Jigsaw – deploy and run

- **jmod** create -L repo
- **jmod** install modules hello fact -L repo
    - **find** repo/
        - repo/
        - repo/fact
        - repo/fact/1.0
        - repo/fact/1.0/index
        - repo/fact/1.0/info
        - repo/fact/1.0/classes
        - repo/%jigsaw-library
        - repo/hello
        - repo/hello/1.0
        - repo/hello/1.0/config
        - repo/hello/1.0/index
        - repo/hello/1.0/info
        - repo/hello/1.0/classes
- **java** -L repo -m hello
    - 3628800
- Modules can be used also directly from **modules** dir where were built

Project Jigsaw – deploy and run

- Make jmod packages
  - **jpkg** -m modules/fact jmod fact
  - **jpkg** -m modules/hello jmod hello
    - fact@1.0.jmod hello@1.0.jmod
- Make linux packages
  - jpkg -m modules/ deb hello
  - jpkg -m modules/ rpm fact
    - fact_1.0_x86_64.deb hello-1.0.x86_64.rpm
- Install module back from a jmod package
  - jmod install -L repo hello@1.0.jmod
  - java -L repo -m hello
    - 3628800

Project Jigsaw – little bit under the hood

- Declaration

  module foo{}

  module foo @1.0 {}

  - Version is optional
  - Name is qualified java identifier
  - No annotations

- Exports

```
module foo{
        exports foo;
}
```

  - Exports all public types in foo, but not in subpackages
  - Name convention

```
module foo{
        exports foo;
        exports foo.bar;
        exports foo.baz;
}
```

  - No private members export ever!

Project Jigsaw – little bit under the hood

- Requires

```
module bar{
        requires foo;
}
```

  - **foo** and **bar** will have **different classloaders**
  - Do **not** export foo's classes
  - Optional version constraints

```
module bar{
        requires foopa @ >=1.0;
        requires foot  @ <2.3a;
}
```

- Re-exports

```
module bar{
        requires public foo;
}
```

  - Reexports foo's classes (otherwise same)

Project Jigsaw – little bit under the hood

- Services

```
module bar{
    provides service servers.Server with myServers.MyServerImpl;
}
```

  - Provides implementation of service

```
module bar{
    requires service servers.Server
}
```

  - Is requiring implementation(s) of service
  - will got myServers.MyServerImpl in this case
- Enhanced ServiceLoader API with possibility of select the impl
- Services creation:

```
Class<Foo> serviceInterface = ...;
ClassLoader serviceConsumer = ...;
// Lazy,  No service instances are instantiated
Iterable<Foo> services = ServiceLoader.load(serviceInterface, serviceConsumer);
// Instantiation occurs on each call to Iterator.next()
for (Foo service : services) { if (service.isCapableOf(...)) {
    return service;}}
return new DefaultFoo();
```

Project Jigsaw – little bit under the hood

- Permits

  ```
  module foopa{
      permits bar;
  }
  ```

  - **foopa** can be required **only** by bar
  - Otherwise same
- Local dependence

  ```
  module bar{
      requires local foopa;
  }
  ```

  - **foopa** must explicitly permits bar
  - **foo** and **bar** will have **same classloaders**
    - **The only case of shared classlaoder**
    - Multi-module packages
- Optional dependence

  ```
  module foopa2{
      requires optional bar2;
  }
  ```

  - Must be ready to work without it

Project Jigsaw – little bit under the hood

- Entry point

  ```
  module foo{
      class foo.Main;
  }
  ```

  - Alternative to manifest entry with main method
  - Java -m foo

- Base module

  Jdk itself - *java.base*

  If module is not requiring exact version, then platform default is added

Project Jigsaw – little bit under the hood

- Aliases

```
module foo{
    provides bar;
}
```

  - Renaming of bar?
  - Necessary for renaming of known packages to new modules

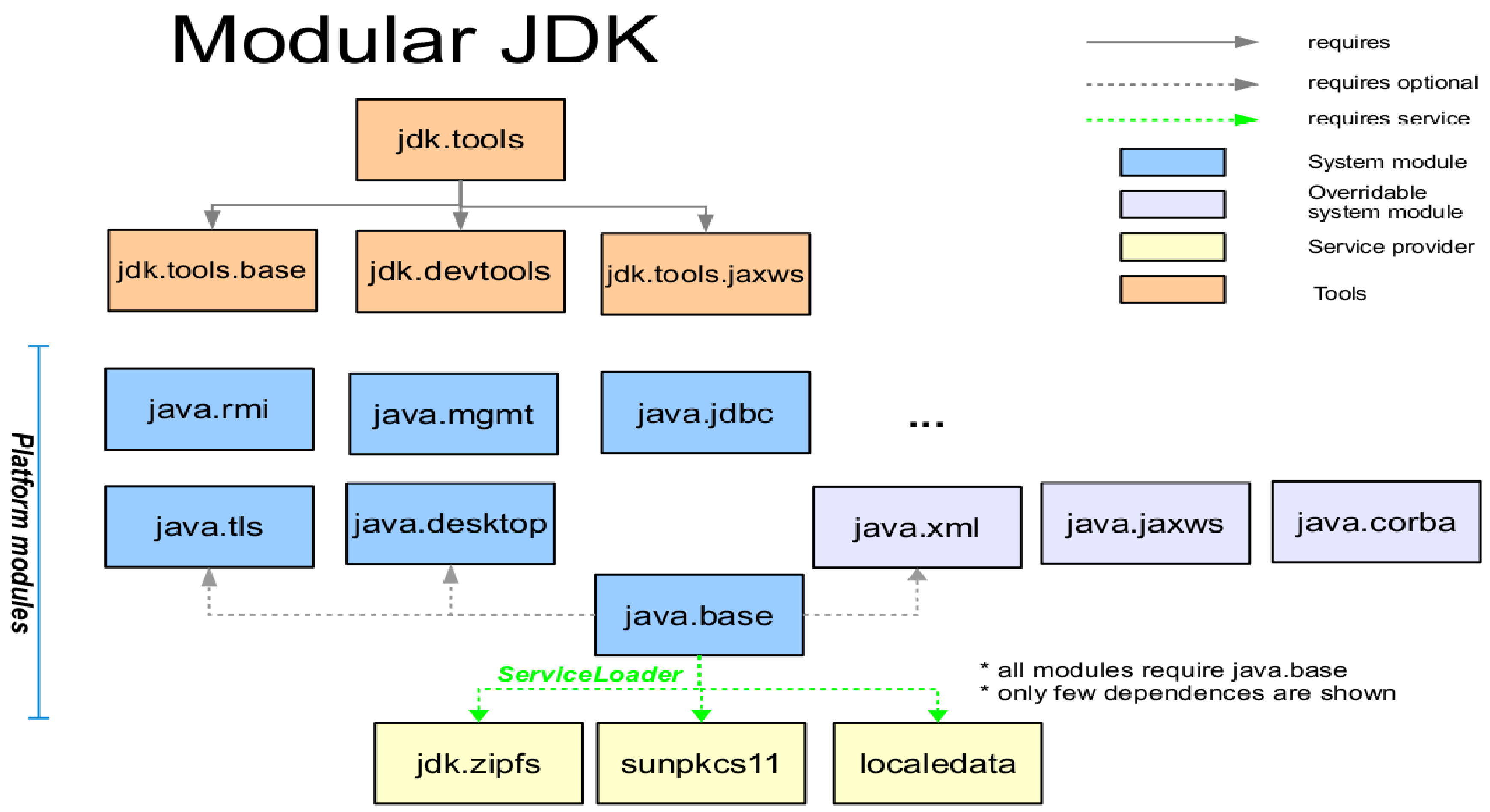
- view

```
module bar{
    requires foo;
    exports bar
    view bar.internal {
        permits baz;
        exports bar.private;
    } view cat {
        class org.foo.Cat
    } view ls{
        class org.foo.List
    }
}
```

- java -m cat   x   java -m ls

Project Jigsaw – modular jdk

- Aliased by *java.base*



Project Jigsaw – classlaoders

- Class.getClassLoader() will never be null
  - There will be classlaoder(s) for *java.base*
  - Replacement for bootclassloader
- Each module will have its own classlaoder
  - Except multi-module package
- No possibility to access private classes of other modules
  - Some hackish way to get its classlaoder and then access via some new reflection tricks??

Project Jigsaw – byte code

- The module-info.java is compiled into module-info.class
- New ClassFile.access_flag ACC_MODULE (0x80000) added on byte code level
- Major/minor version limitation (>= 53.0, jdk 9)
- No implicit reexports – just expanded
- Also views are expanded
- Dependencies, exports and services are tables with indexes to constant pool

Conclusion

- Oracle have fulfill some of his promises
  - Community is taken care about
  - Lambda is going in
  - Most of the small changes are going in
- Dropping of jigsaw in October is sad but probably worthy
  - Modularisation of binary parts?
  - Convergence of java ME?
- Dropping of some of some in January 2013 ..
  - Well smells like problems
- At least it is still evolution and not revolution

Questions?

- http://openjdk.java.net/projects/jdk8/
- http://wiki.eclipse.org/JDT_Core/Java8
- http://openjdk.java.net/projects/jigsaw/doc/module-class-loading.pdf
- http://openjdk.java.net/jeps/0 (all the JEPs of Small features)
- http://julien.ponge.info/notes/building-openjdk8-with-jigsaw/
- http://openjdk.java.net/projects/jigsaw/
- Http://openjdk.java.net/projects/jigsaw/doc/openjdk-jigsaw-modular-services.pdf
- http://icedtea.classpath.org/shenandoah/
- http://openjdk.java.net

Thank you for your attention!