JAVA

Moduly

Modularizace

- modul
  - explicitně definované co poskytuje i co ***požaduje***

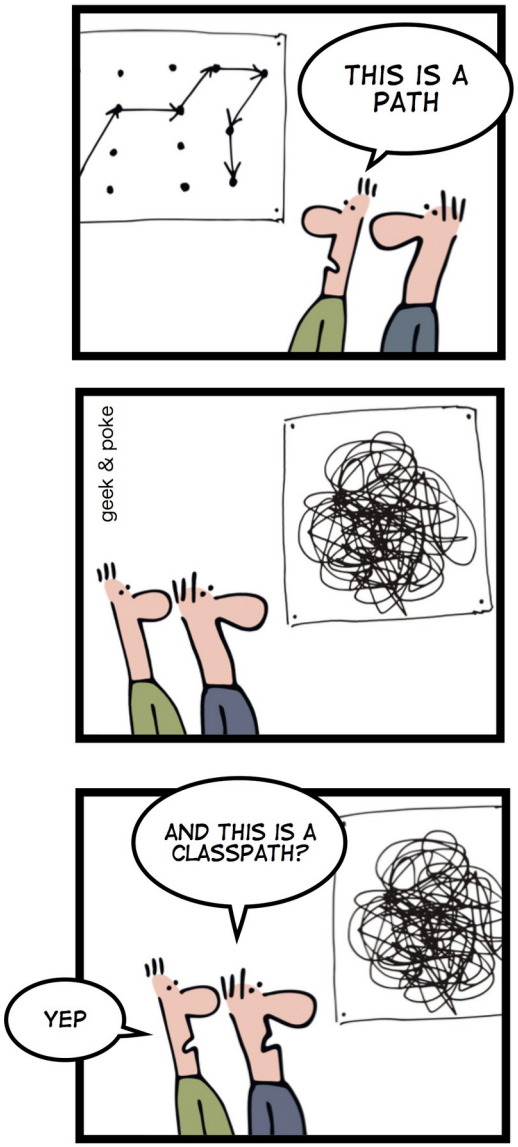
- proč
  - koncept *classpath* je „křehký“
  - chybí zapouzření

Modu

- modul
  - exp

ožaduje

- proč
  - kon
  - chyl

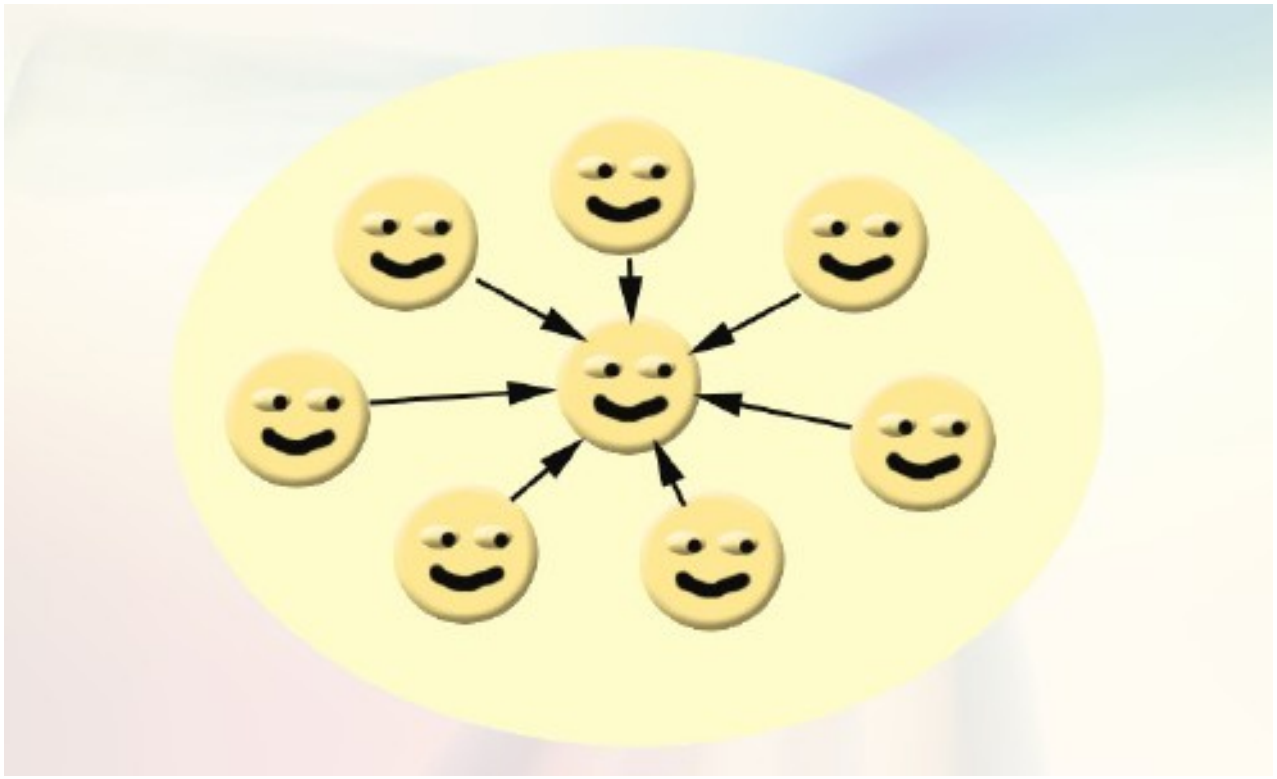


Modulární aplikace – motivace

- proč
  - aplikace více a více komplexní
  - skládání aplikací
  - vývoj v distribuovaných týmech
  - komplexní závislosti
  - dobrá architektura programu
    - ví o svých závislostech
    - spravuje závislosti

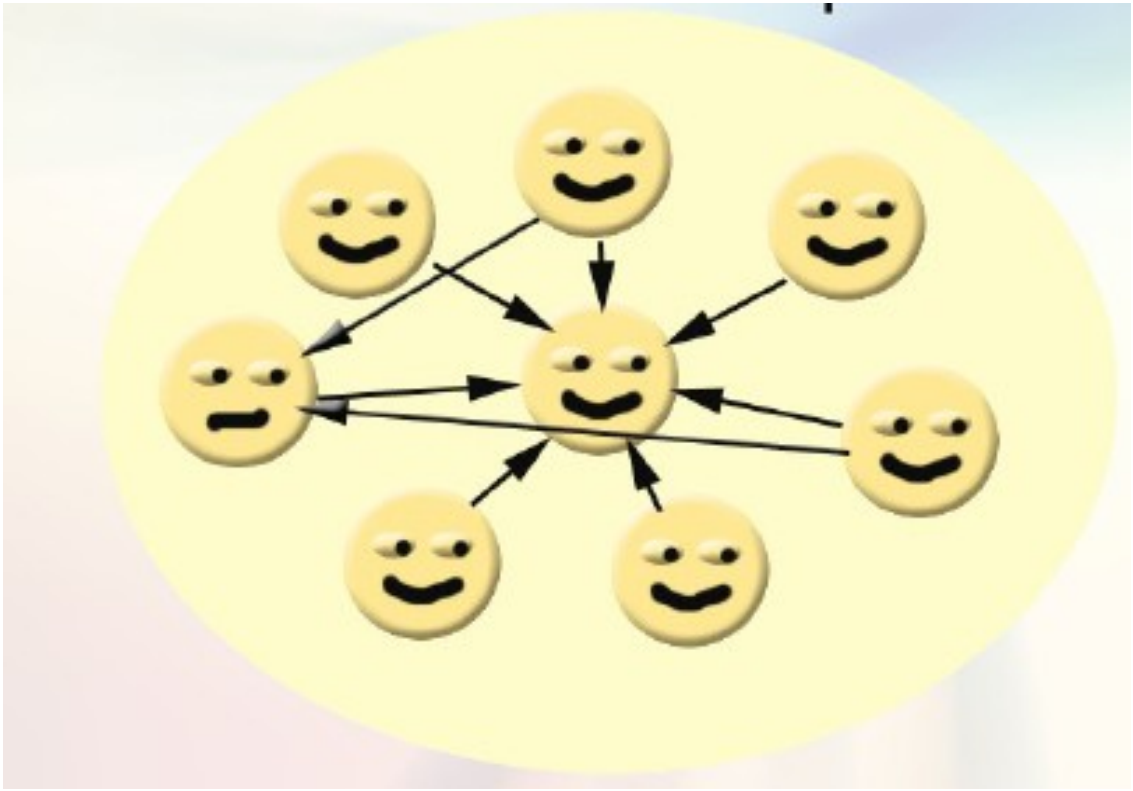
Modulární aplikace – motivace

- Verze 1.0 – vše dobře navrženo



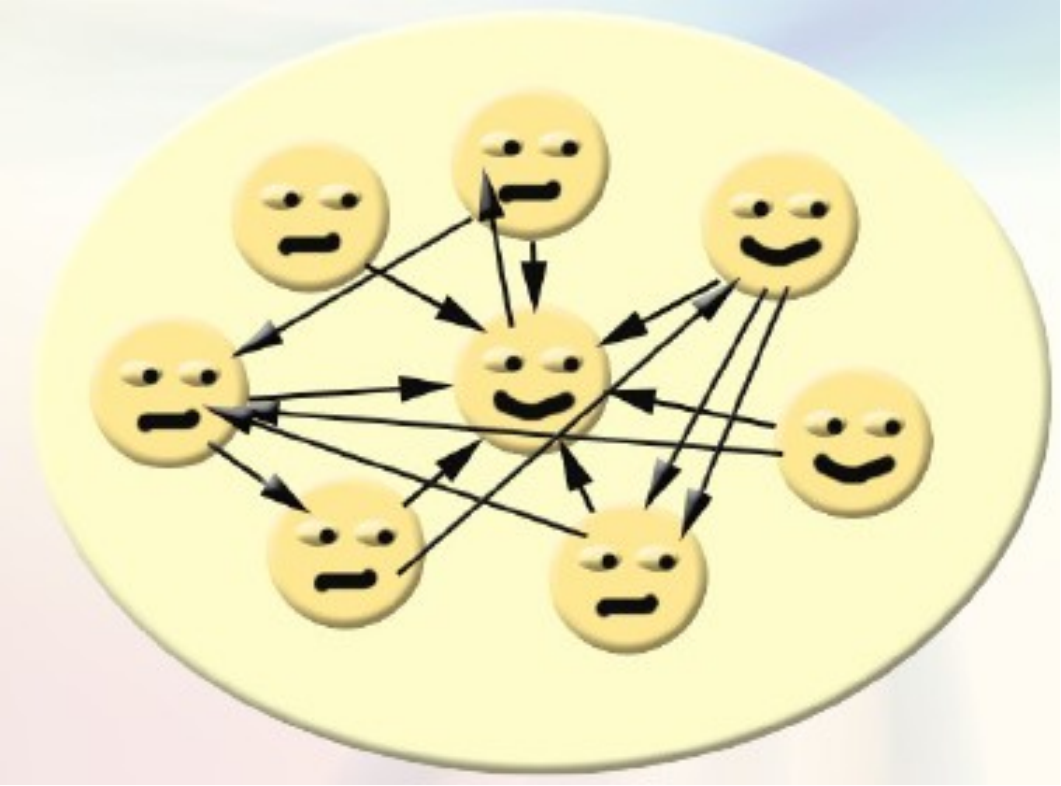
Modulární aplikace – motivace

- Verze 1.1...několik „vynálezavých hacků“...vyčistíme to ve 2.0



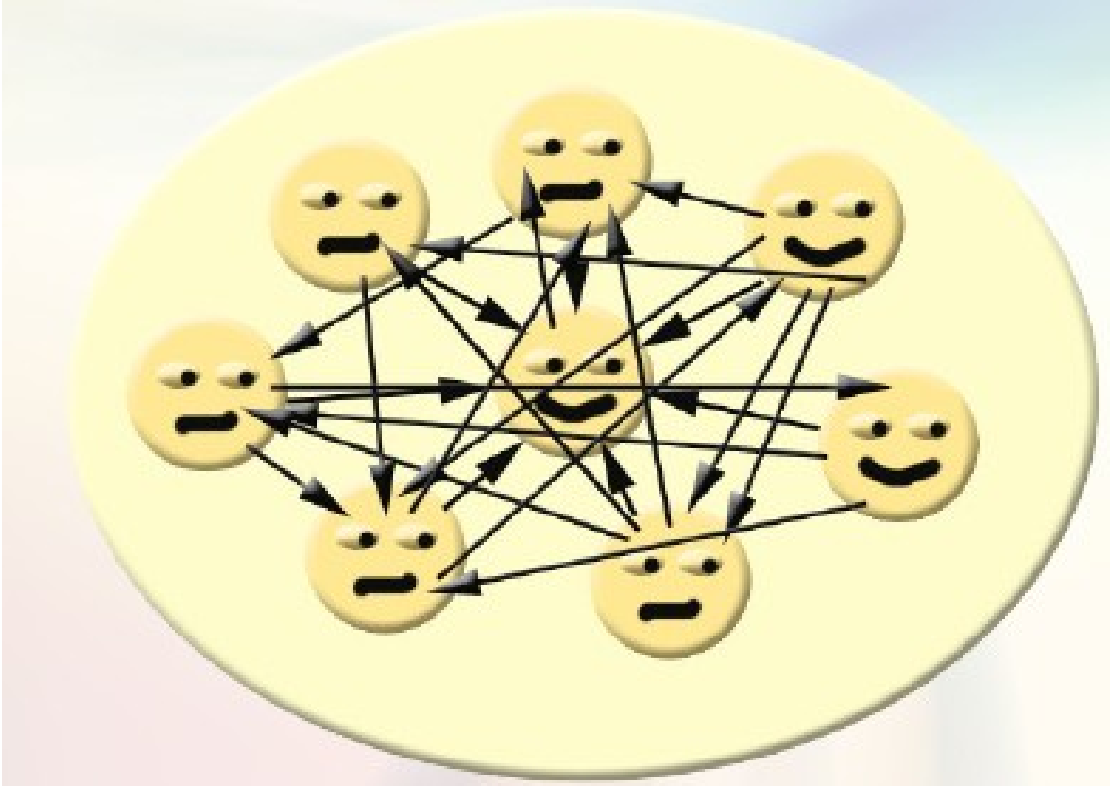
Modulární aplikace – motivace

- Verze 2.0...oops...ale...funguje to!



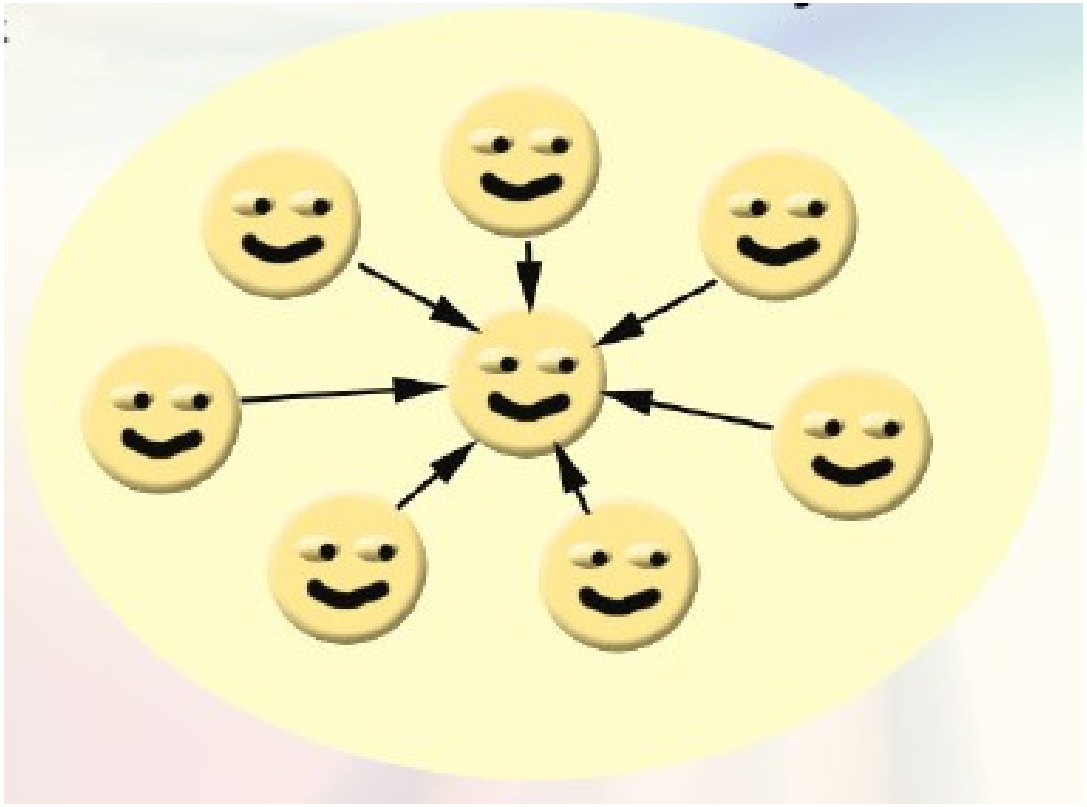
Modulární aplikace – motivace

- Verze 3.0 – Pomoc! Oprava jakékoliv chyby přinese dvě další chyby!



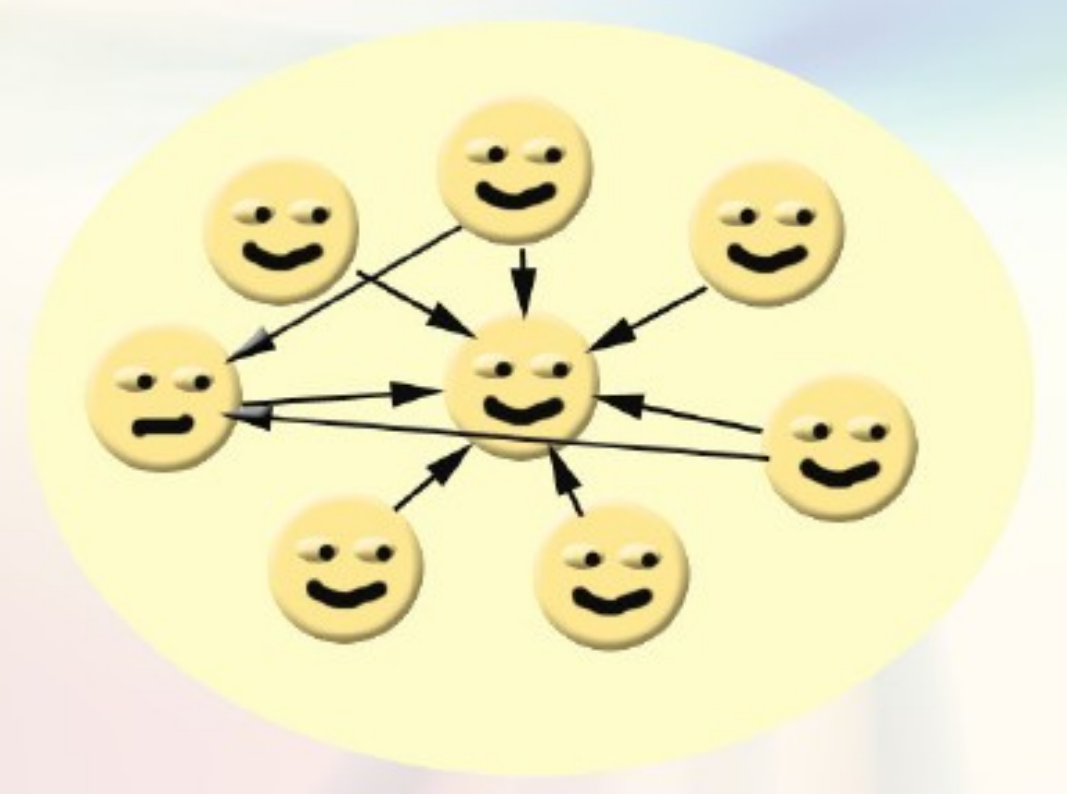
Modulární aplikace – motivace

- Verze 4.0 – vše dobře navrženo. Kompletně přepsano, trvalo to o rok delé, ale funguje to...



Modulární aplikace – motivace

- Version 4.1...tohle vypadá povědomě...



Deklarace modulu

- module-info.java

```
module com.foo.bar {
    requires com.foo.baz;
    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;
}
```

- modular artifact
  - modularní JAR – JAR obsahující module-info.class
  - nový formát JMOD
    - ZIP s třídami, nativním kódem, konfigurací,...

Moduly a JDK

- standardní knihovna JDK také modularní
  - java.base – vždy „required“

```
module java.base {
    exports java.io;
    exports java.lang;
    exports java.lang.annotation;
    exports java.lang.invoke;
    exports java.lang.module;
    exports java.lang.ref;
    exports java.lang.reflect;
    exports java.math;
    exports java.net;
    ...
```

Module readability & module path

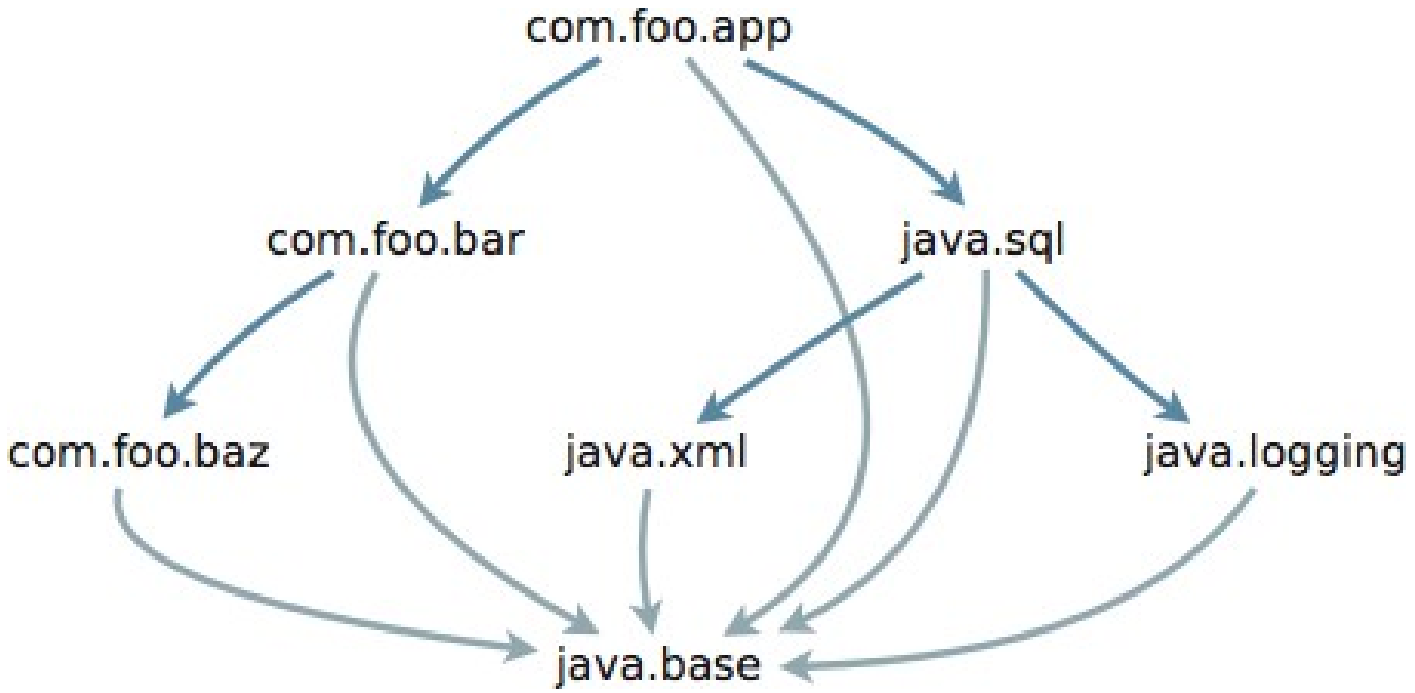
- Pokud modul přímo závisí na jiném modulu

  Modul ***čte (reads)*** jiný modul (nebo, jinak, druhý modul je ***čitelný (readable)*** prvním modulem)

- ***Module path*** – ekvivalent ke classpath
  - ale pro moduly
    - -p, --module-path
  - spuštění aplikace
    java -p <module_path> name_of_module/name_of_class

Module graph

```
module com.foo.app {
    requires com.foo.bar;
    requires java.sql;
}
```



Dostupnost (Accessibility)

- Pokud jsou dva typy S a T definovány v různých modulech a T je public, potom kód v S může přistupovat (access) k T pokud:
  - modul s S čte modul s T, a zároveň
  - modul s T exportuje balíček s T

Implied readability

- Readability není tranzitivní
  - příklad:

in java.sql

```
java.sql.Driver {
    java.util.Logger getParrentLogger();
    ...
```

in java.logging

```
module java.sql {
  requires public java.logging;
  requires public java.xml;
  exports java.sql;
  exports javax.sql;
  exports javax.transaction.xa;
}
```

Services & ServiceLoader

```
module com.mysql.jdbc {
  requires java.sql;
  requires org.slf4j;
  exports com.mysql.jdbc;
  provides java.sql.Driver with com.mysql.jdbc.Driver;
}
```

```
module java.sql {
  requires public java.logging;
  requires public java.xml;
  exports java.sql;
  exports javax.sql;
  exports javax.transaction.xa;
  uses java.sql.Driver;
}
```

Qualified exports

- module java.base {
  ...
  exports sun.reflect ***to***
    java.corba,
    java.logging,
    java.sql,
    java.sql.rowset,
    jdk.scripting.nashorn;
  }

- není určeno pro běžné používání

requires static

- vyžadován během překladu, ale volitelný při spuštění

```
module com.foo.bar {
    requires static com.foo.baz;
}
```

- VAROVÁNÍ
  - kód vyžadující balíček přes `required static` musí být připraven na nedostupnost

opens, open

- před Java 9, cokoliv je dostupné přes reflection
  - i privátní elementy
- v Java 9+, i reflexe dodržuje pravidla pro moduly
- ale – balíčky lze „otevřít“

```
module com.foo.bar {
    opens com.foo.bar.alpha;
}
```

- typy v „otevřeném“ balíčku jsou dostupné při běhu

```
open module com.foo.bar { }
```

  - otevírá všechny svoje balíčky

opens to

- **opens** package **to** list-of-modules
  - otevírá balíček jen pro vybrané moduly

Reflection

```
package java.lang.reflect;

public final class Module {
  public String getName();
  public ModuleDescriptor getDescriptor();
  public ClassLoader getClassLoader();
  public boolean canRead(Module source);
  public boolean isExported(String
                    packageName);
  ...
}
```

Vrstva (layer)

- layer – instance grafu modulů při běhu programu
- mapuje každý modul v grafu na jedinečný classloader

- vrstvy lze vrstvit přes sebe
  - nová vrstva může být vytvořena nad existující
    - graf modulů vrstvy – obsahuje (jako reference) grafy modulů všech vrstev níže
- boot layer
  - vytvořen VM při startu

- vrstvy – určeny pro aplikační servery, IDE,...

Kompatibilita se „starou“ Javou

- Classpath stále podporováno
  - v podstatě jsou moduly „volitelné“

- Nepojmenovaný modul
  - cokoliv mimo jakýkoliv modul
    - „starý“ kód
  - čte jakýkoliv jiný modul
  - exportuje všechny svoje balíčky pro všechny jiné moduly

Automatic module

- implicitně definovaný pojmenovaný module
  - nemá deklaraci modulu

- „obyčejný“ JAR umístěný na module path místo na classpath
  - JAR bez module-info.java

Scripting API

Přehled

- podpora skriptovacích jazyku přímo z Javy
  - integrace skriptů do Java programu
  - volání skriptů
  - používání Java objektů ze skriptu
    - a obraceně
  - …
- od Java 6 přímo součástí JDK
  - součástí JDK je JavaScript engine
    - Java 6-7 Mozilla Rhino engine
    - Java 8 Nashorn engine
    - implementace JavaScript jazyka v Javě
    - od Java 11 – Nashor označen deprecated
      - bude odstraněn bez náhrady
        - ale Scripting API zůstává
  - existuje mnoho implementací pro další jazyky
    - použití přes ServiceLoader

Proč

- jednotné rozhraní pro všechny skriptovací jazyky
  - dříve si každá implementace řešila rozhraní po svém
- snadné požívání skr. jazyků
  - proměnné „bez“ typů
  - automatické konverze
  - …
  - programy není nutno kompilovat
    - existence „shelů“
- použití
  - složitější konfigurační soubory
  - rozhraní pro „administrátora“ aplikace
  - rozšiřování aplikace (pluginy)
  - skriptování v aplikaci
    - obdoba jako JS v prohlížeči, VBScript v office,...

Použití

- balíček javax.scripting
- ScriptEngineManager
  - základní třída
  - nalezení a získání instance skript. enginu
- základní použití
  - instance ScriptEngineManageru
  - nalezení požadovaného enginu
  - spuštění skriptu pomocí metody eval()

Hello world

```
public class Hello {
  public static void main(String[] args) {
    ScriptEngineManager manager =
                                        new ScriptEngineManager();
    ScriptEngine engine =
                      manager.getEngineByName("JavaScript");
    //ScriptEngine engine =
                         manager.getEngineByExtension("js");
    //ScriptEngine engine =
    manager.getEngineByMimeType("application/javascript");
    try {
      engine.eval("println( \"Hello World!\");" );
      System.out.println(
                    engine.eval( " 'Hello World again!' "));
    } catch(ScriptException e) { ... }
  }
}
```

Přehled funkčnosti

- skript
  - řetězec nebo znakový stream (reader)
  - vyhodnocení přes ScriptEngine.eval()
- interface Compilable
  - jeho implementace volitelná
    - otestovat – instanceof Compilable
  - kompilace skriptu do byte-code
- interface Invocable
  - jeho implementace volitelná
    - otestovat – instanceof Invocable
  - volání metod a funkcí ze skriptů
- Bindings, ScriptContext
  - prostředí pro vykonávání skriptů
    - mapování proměnných sdílených mezi Javou a skriptem

Získání enginu (1)

- ScriptEngineManager.getEngineFactories()
  - seznam všech ScriptEngineFactory

```
for (ScriptEngineFactory factory :
                              engineManager.getEngineFactories()) {
  System.out.println("Engine name: " + factory.getEngineName());
  System.out.println("Engine version: " +
                                        factory.getEngineVersion());
  System.out.println("Language name: " +
                                         factory.getLanguageName());
  System.out.println("Language version: " +
                                       factory.getLanguageVersion());
  System.out.println("Engine names:");
  for (String name : factory.getNames()) {
    System.out.println("   " + name);
  }
  System.out.println("Engine MIME-types:");
  for (String mime : factory.getMimeTypes()) {
    System.out.println("   " + mime);
  }
}
```

Získání enginu (2)

- ScriptEngineFactory.getEngine()

- nebo přímo

- ScriptEngineManager.getEngineByName()
- ScriptEngineManager.getEngineByExtension()
- ScriptEngineManager.getEngineByMimeType()

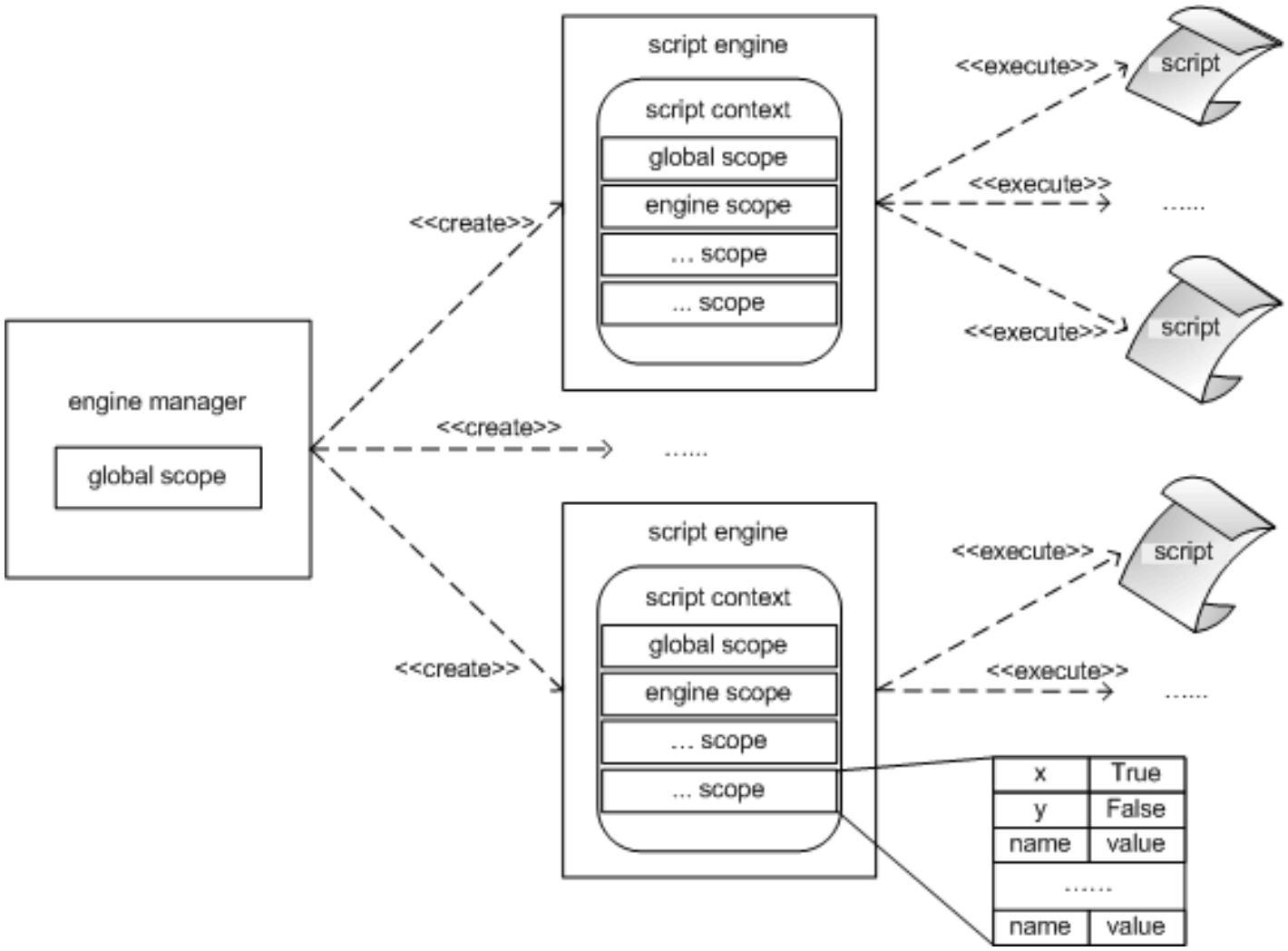
Skripty

- vykonání skriptu
  - Object ScriptEngine.eval(String s, ...
  - Object ScriptEngine.eval(Reader r, …
- předávání proměnných (základní varianta)
  - void ScriptEngine.put(String name, Object value)
  - Object ScriptEngine.get(String name)
  - POZOR na konverze typů!

Předávání proměnných

- interface Bindings
  - extends Map<String,Object>
  - základní implementace – SimpleBindings
- interface ScriptContext
  - prostředí, ve kterém se skripty vykonávají
  - základní implementace – SimpleScriptContext
  - obsahuje scopes
    - scope = Binding
  - speciální scopes
    - ENGINE_SCOPE – lokální pro ScriptEngine
    - GLOBAL_SCOPE – globální pro EngineManager
  - getAttribute(..) / setAttribute(..) odpovídají getBindings(..).get / put
  - lze nastavit standardní Reader a Writery (vstup a výstup) pro skript

Předávání proměnných



zdroj obrázku: http://www.javaworld.com/javaworld/jw-04-2006/jw-0424-scripting.html

Volání funkcí/metod

- interface Invocable
  - volitelná funkčnost, je třeba testovat (instanceof)
  - poskytuje
    - volání funkcí skriptu z Java kódu
    - volání metod objektů skriptu z Java kódu, pokud je skriptovací jazyk objektový
    - implementace Java interface funkcemi (metodami) skriptu

```
ScriptEngine engine = manager.getEngineByName("javascript");
Invocable inv = (Invocable) engine;

engine.eval("function run() { println( 'funkce run'); };");
Runnable r = inv.getInterface(Runnable.class);
(new Thread(r)).start();

engine.eval("var runobj = { run: function()
                                { println('metoda run'); } };");
o = engine.get("runobj");
r = inv.getInterface(o, Runnable.class);
(new Thread(r)).start();
```

JavaScript engine v JDK (1)

- některé funkce odstraněny (nebo nahrazeny)
  - převážně z důvodů bezpečnosti
- vestavěné funkce pro import Java balíčků
  - importPackage(), importClass()
    - balíčky přístupné přes Packages.JmenoBalíčku, pro nejpoužívanější balíčky jsou definované zkratky (proměnné): java (ekvivalentní Packages.java), org, com, ..
    - java.lang není importován automaticky (možné konflikty objektů Object, Math, ..)
    - od Java 8 nutno nejdříve použít `load("nashorn:mozilla_compat.js");`
  - objekt JavaImporter
    - pro ukrytí importovaných prvků do proměnné (předchází konfliktům)

```
var imp = new JavaImporter( java.lang, java.io);
```

JavaScript engine v JDK (2)

- Java objekty v js
  - vytváří se stejně jako v Javě
  - var obj = new Trida(...)
- Javovské pole v js
  - vytvoříme přes Java Reflection
  - var pole = java.lang.reflect.Array.newInstance(..)
  - dále pracujeme běžně: pole[i], pole.length, …

```
var a = java.lang.reflect.Array.newInstance( java.lang.String, 5);
a[0] = "Hello"
```

- anonymní třídy
  - anonymní implementace Java rozhraní

```
var r  = new java.lang.Runnable() {
    run: function()
    {
        println( "running...");
    }
};
var th = null;
th = new java.lang.Thread( r);
th.start();
```

JavaScript engine v JDK (3)

- anonymní třídy (pokrač.)
  - autokonverze funkce na rozhraní s jednou metodou

```
function func() {
     print("I am func!");
};
th = new java.lang.Thread( func);
th.start();
```

JavaScript engine v JDK (4)

- přetížené Java metody
  - připomenutí
    overoading se děje při překladu (javac)
  - při předání JavaScriptových proměnných Java metodám vybere script engine správnou variantu
  - výběr můžeme ale ovlivnit
    - objekt[“název_metody(typy parametrů)”](parametry)
    - pozor! řetězec bez mezer!

Další enginy

- existuje velké množství hotových enginů
  - awk, Haskell, Python, Scheme, XPath, XSLT, PHP,...

- vytvoření vlastního enginu
  - implementace API
    - nutno implementovat alespoň
      - ScriptEngineFactory
      - ScriptEngine
  - deklarace implementování javax.script.ScriptEngineFactory
    - pro ServiceLoader