

Co nowego w JDK 7

Marcin Kalas
marcin.kalas@gmail.com



IT Project Management

PHP

.NET & C#

Java

March 26th, 2010 in Poznan, Poland

JDK 7 – kiedy? Czy warto się interesować?

Wprowadzenie Euro



Euro w Polsce



J2SE 5.0

Open Source JDK

J2SE 1.4
2738 klas i interfejsów

J2SE 6.0

J2SE 7.0
Ponad 4000 klas i interfejsów

J2EE beta

Dla prawie 75% programistów
Java podstawowym językiem

JVM zainstalowany
na ponad 550 milionach
komputerów

Konferencja 4Developers
w Krakowie



Euro 2012

Pierwsza konferencja JavaOne



2001

2003

2005

2007

2009

2011



IT Project Management

PHP

.NET & C#

Java

March 26th, 2010 in Poznan, Poland

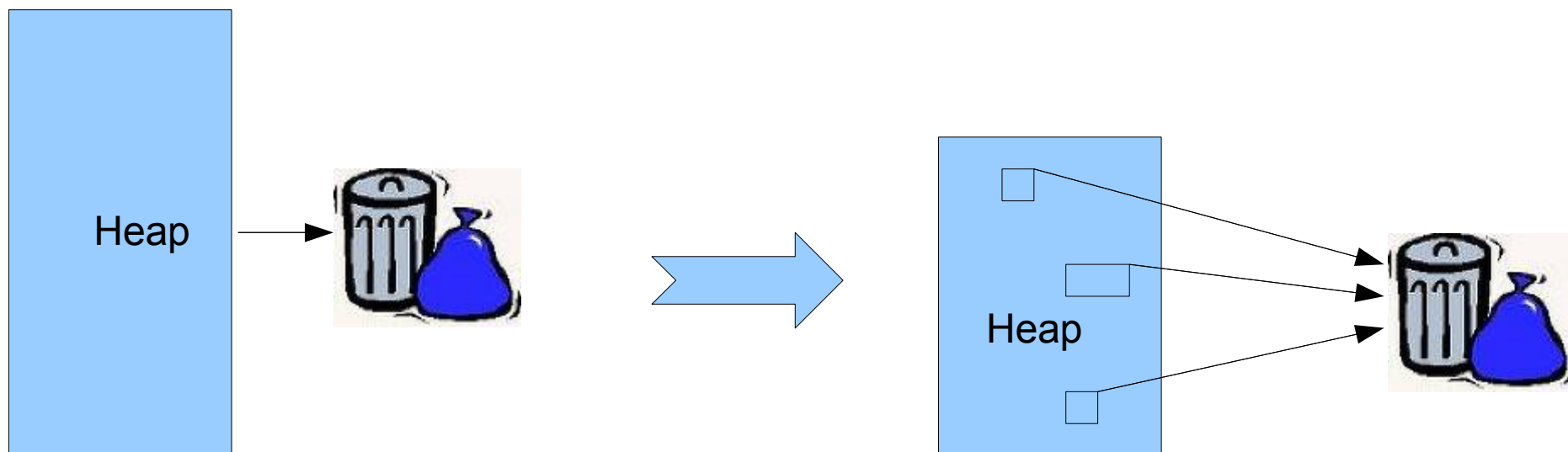
Zmiany planowane w JDK 7

- Wydajność:
 - Garbage First – nowy garbage collector
 - Skompresowany 64-bitowy wskaźnik
- Rozwój języka – udogodnienia dla programistów
 - Projekt Coin (niewielkie ale ułatwiające programowanie zmiany)
 - Udoskonalone API do operacji I/O (JSR 203)
 - Udoskonalone anotacje typów (JSR 308)
 - Domknięcia
 - Algorytmy równoległe (pakiet `java.util.concurrent.forkjoin`, `ParallelArray`, JSR 166)
- Modularyzacja (JSR 294, projekt Jigsaw)
- Udoskonalone wsparcie dla języków dynamicznych (projekt da Vinci; nowy bytecode `invokedynamic`, JSR 292)



JDK 7 – wydajność

- Skompresowany 64-bitowy wskaźnik
- Nowy Garbage Collector - G1



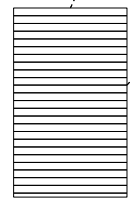
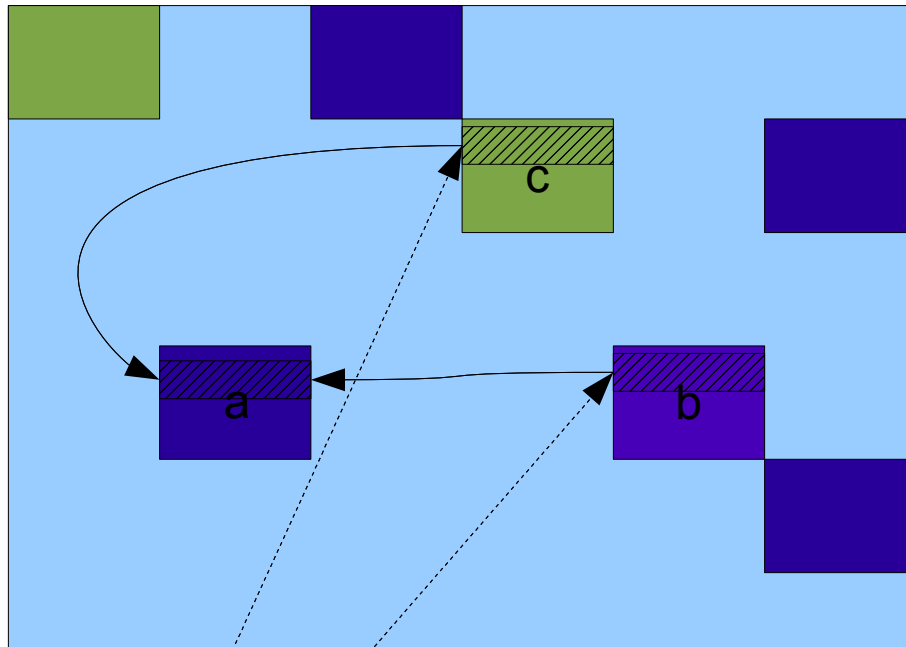
Garbage First – nowy garbage collector

- Wszystkie strategie garbage collector (HotSpot JVM, J2SE 5+) wykorzystują pokolenia
 - Większość obiektów charakteryzuje się krótkim czasem życia
 - Niewiele obiektów o długim czasie życia posiada referencję do obiektów młodych
 - Opłacalne jest zastosowanie różnych strategii dla obu typów obiektów:
 - Podział sterty na 3 obszary: young generation, old generation, perm
 - Większość cykli dotyczy tylko obszaru młodych obiektów
 - Wykorzystanie kolektora kopiującego
 - Obszar obiektów o długim czasie życia odświeżany jest kolejno w fazach mark-sweep-(compact)
- Garbage First – charakterystyka:
 - Następca kolektora CMS
 - Równoległy
 - Współbieżny
 - Wykorzystujący pokolenia
 - Kompaktujący stertę (konsolidacja wolnych obszarów ułatwiająca alokację nowych obiektów)
 - Kompaktowanie to najdroższa i najdłuższa operacja
 - Łatwiejszy w strojeniu

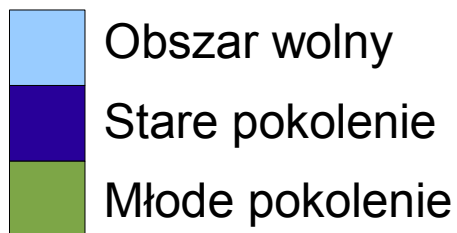


Garbage first – nowy garbage collector

Sterta przed fazą **Evacuation**



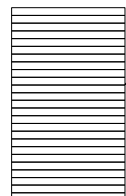
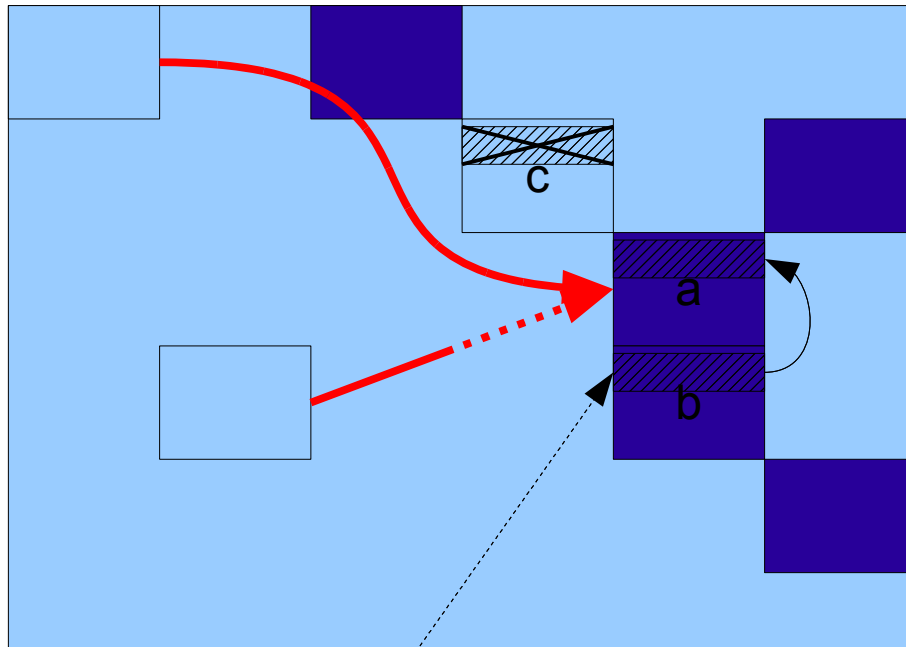
Remembered set regionu a



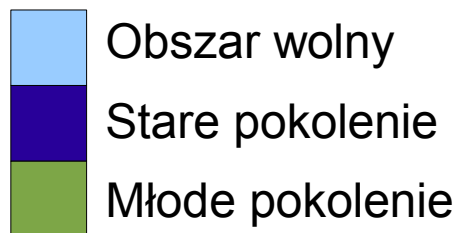
- Podział sterty na obszary 1 MB
 - Młode pokolenie – zbiór obszarów
 - Stare pokolenie – zbiór obszarów
- **Evacuation** - odśmiecanie obiektów (zarówno młode jak i stare pokolenie)
 - Wykonywane w jednym cyklu Stop-The-World
 - Równoległe
 - Kopiowanie obiektów z jednego obszaru do drugiego
- *Remembered set*
 - Zawiera informacje o obszarach zawierających referencje do obiektów danego obszaru
 - Umożliwia odśmiecenie danego regionu bez potrzeby skanowania całej sterty
 - Tworzony dla każdego obszaru (łączny narzut to około 5% rozmiaru sterty)

Garbage first – nowy garbage collector

Sterta po fazie **Evacuation**



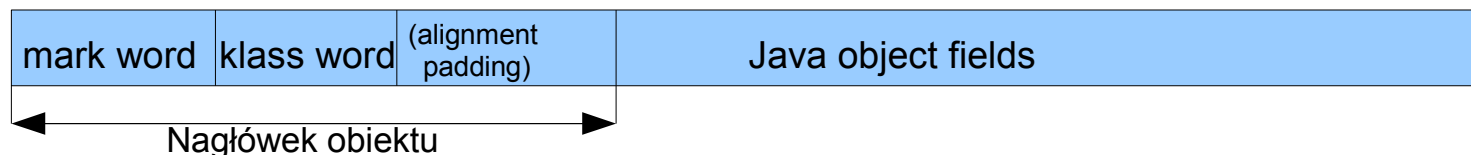
Remembered set obszaru a



- Concurrent mark
 - Wykorzystanie techniki snapshot-at-the-beginning
 - Snapshot sterty może być wykonany podczas ewakuacji
 - Krótkie pauzy STW by zapewnić poprawny stan sterty
 - Faza Remark
 - Faza Clean-up

Skompresowany 64-bitowy wskaźnik

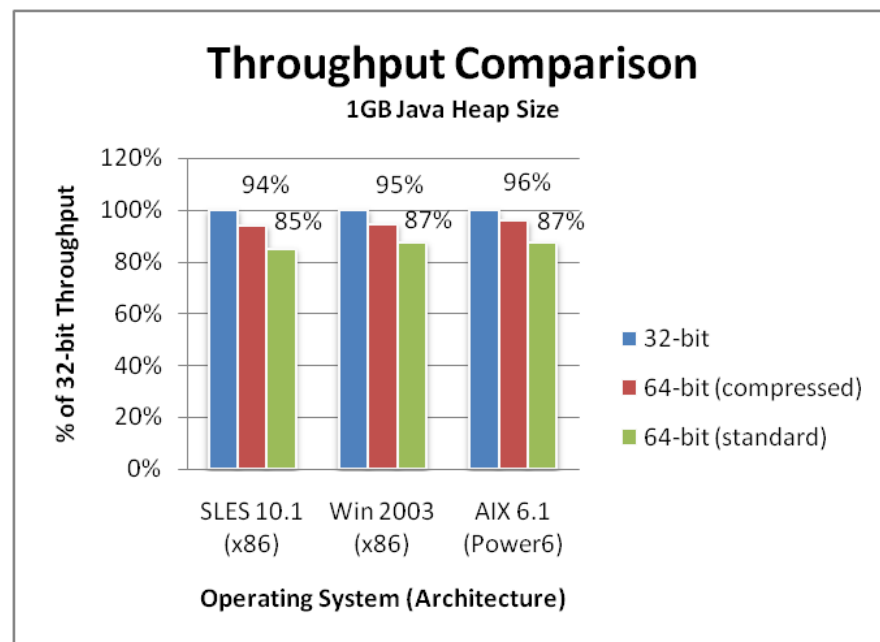
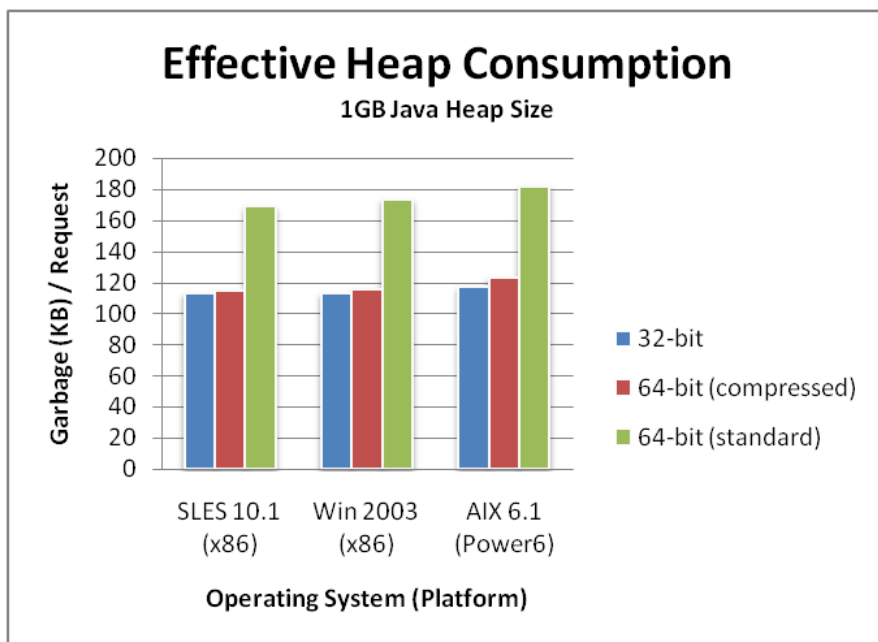
- 32-bitowa maszyna wirtualna ma ograniczony rozmiar sterty
 - Do dyspozycji mamy $2^{32} = 4$ [GB] przestrzeń adresową podzieloną na przestrzeń użytkownika oraz przestrzeń jądra
 - Maksymalny rozmiar sterty zależy od systemu operacyjnego
- 64-bitowa maszyna wirtualna nie ma takich ograniczeń (są one tak duże, że możemy je bezpiecznie pominąć)
 - Wykorzystanie 64-bitowej maszyny wirtualnej powoduje jednak zwiększenie zużycia pamięci
 - Średnio wzrost ten wynosi około 40%
 - Zwiększenie rozmiaru nagłówków obiektów
 - Zwiększenie rozmiaru referencji do innych obiektów
 - Przykład:
 - obiekt typu Integer + 6 referencji, które na niego wskazują



Reprezentacja	Heap bits	Reference bits (N=6)	Łącznie bitów	Porównanie
int	0	192	192	1
Object, 32-bit	128	192	320	1,67
Object, 64-bit	256	384	640	3,33

Skompresowany 64-bitowy wskaźnik

- Skompresowany wskaźnik pozwala na reprezentację oop (ordinary object pointer) z wykorzystaniem 32 bitów
- Wykorzystanie skompresowanego wskaźnika umożliwia przejście na platformę 64-bitową bez wzrostu zużycia pamięci
- Jednocześnie narzut dodatkowych instrukcji związanych z zdekodowaniem/kodowaniem skompresowanego wskaźnika nie jest duży



** testy przeprowadzone z wykorzystaniem maszyny wirtualnej IBM*



Projekt Coin

- Wprowadzenie niewielkich zmian w składni języka ułatwiających programistom codzienne zadania
 - Propozycje zmian można było zgłaszać od 27 lutego 2009 do 30 marca 2009
 - Zgłoszono 70 propozycji, które zostały poddane dyskusji
- Założenia projektu
 - Rozważne wybieranie zmian w składni języka
 - Brak nowych słów kluczowych



String w instrukcjach switch

• **Autor:** Joseph D. Darcy

• **Korzyści:**

- Możliwość wykorzystania stringów w instrukcji *switch* analogicznie do typów prostych (byte, short, char oraz int)
- Bardziej czytelny i przejrzysty kod (łatwiejsza interpretacja stringów niż liczb całkowitych)
- Łatwiejsze zrozumienie działania instrukcji *switch* gdy pojawi się w debuggerze (lub gdy musimy zdebuggować aplikację na podstawie logów)

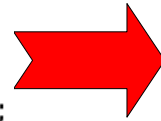
• **Implementacja:**

- String w instrukcji *switch* można zaimplementować z wykorzystaniem funkcji zwracającej wartość całkowitą (integer) na podstawie stringa
- dobry kandydat na taką funkcję to *String.hashCode* lub *((long)s.hashCode<<32 + s.length())* w przypadku kolizji
- Technika podobna do wykorzystania enum w instrukcji switch

• **Przykłady wykorzystania**

NetBeans powiadomił nas o błędzie kompilacji.
W wersji 6.8 NetBeans nie wspiera projektu Coin (wsparcie pojawi się w kolejnych wersjach).

```
String conferenceName = "4DeveloPERS";  
if (JAVA_ONE.equals(conferenceName)) {  
    System.out.println("Jestemy na konferencji: " + JAVA_ONE);  
} else if (FOR_DEVELOPERS.equals(conferenceName)) {  
    System.out.println("Jestemy na konferencji: " + FOR_DEVELOPERS);  
} else if (DEVOXX.equals(conferenceName)) {  
    System.out.println("Jestemy na konferencji: " + DEVOXX);  
}
```

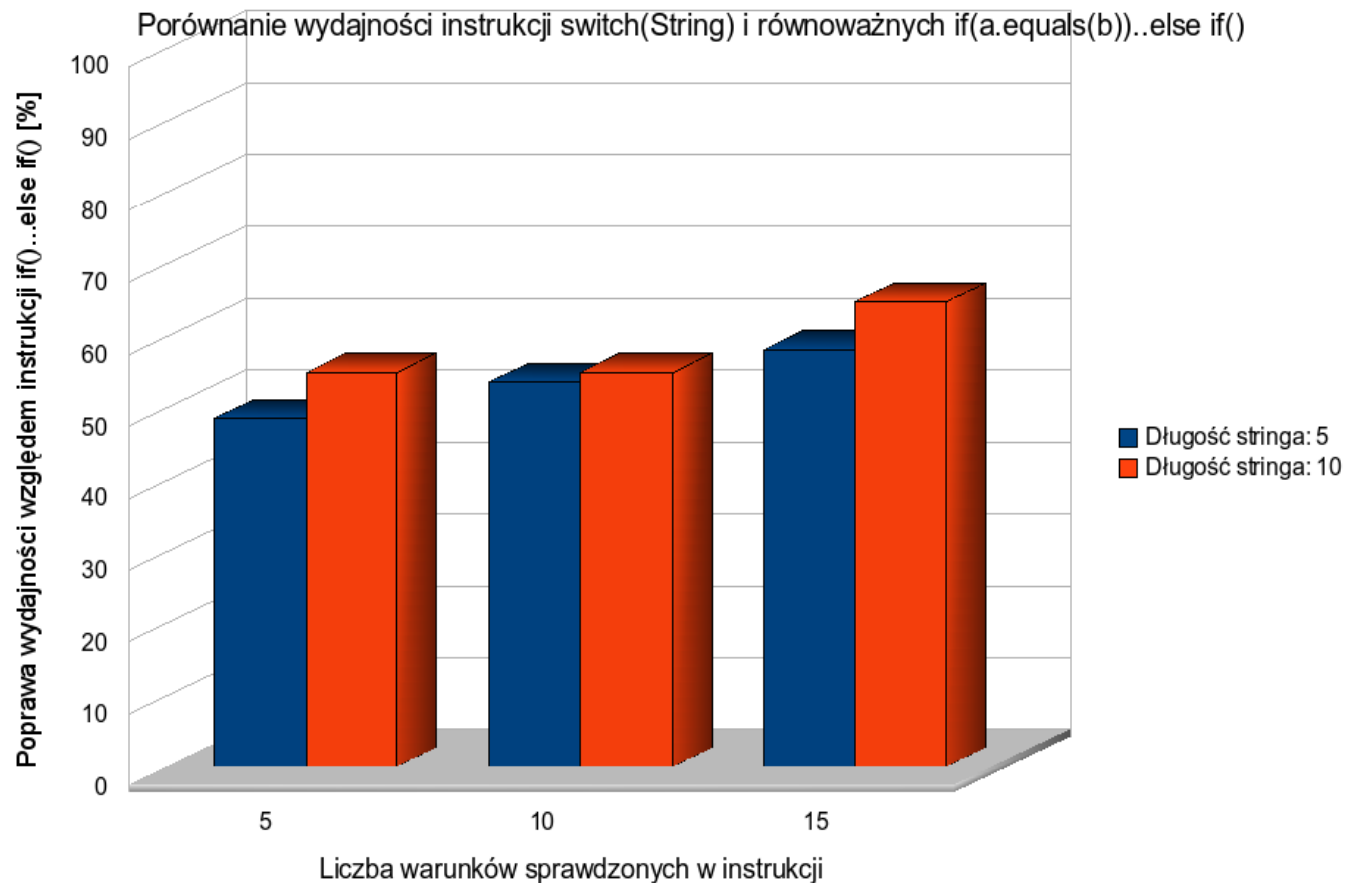


```
switch (conferenceName) {  
    case FOR_DEVELOPERS:  
        System.out.println("Jestemy na konferencji: " + FOR_DEVELOPERS);  
        break;  
    case JAVA_ONE:  
        System.out.println("Jestemy na konferencji: " + JAVA_ONE);  
        break;  
    case DEVOXX:  
        System.out.println("Jestemy na konferencji: " + DEVOXX);  
        break;  
}
```



String w instrukcjach switch – wydajność

- Metoda `String.equals()` może być stosunkowo droga
 - Wykorzystanie instrukcji `switch(String)` zamiast równoważnych `if(a.equals(b))...else if(c.equals(b))...else()` może dać potencjalne korzyści
 - Porównanie instrukcji warunkowych – brak dodatkowej logiki biznesowej



Automatyczne zarządzanie zasobami

- Autor Przykład ze strony <http://java.sun.com/j2se/1.5.0/docs/guide/>

- Motywacja

- Manualne zarządzanie zasobami jest bardzo częstym błędem w bazach danych

– W 60%

- W 60% przypadków

- Kolejny problem utrudnia je

- Implementacja

- Automatyczne zarządzanie (jeden lub więcej zasobów)

- Po zakończeniu pracy zasób musi być zamknięty

- Zasób będzie musiał implementować określony interfejs (*Disposable*)

- Przykłady

```
try {
    URL a = new URL(args[0]);
    URLConnection urlc = a.openConnection();
    is = conn.getInputStream();
    int ret = 0;
    while ((ret = is.read(buf)) > 0) {
        processBuf(buf);
    }
    // close the inputstream
    is.close();
} catch (IOException e) {
    try {
        respCode = ((URLConnection)conn).getResponseCode();
        es = ((URLConnection)conn).getErrorStream();
        int ret = 0;
        // read the response body
        while ((ret = es.read(buf)) > 0) {
            processBuf(buf);
        }
        // close the errorstream
        es.close();
    } catch (IOException ex) {
        // deal with the exception
    }
}
```

} jest kłopotliwe i
). połączenia do

och)

że skutkować jej

rwotny wyjątek i

owane są zasoby

padku normalnego



Automatyczne Zarządzanie Zasobami

```
22 public static void main(String[] args) {  
23  
24     String path = "/home/grome/";  
25     BufferedReader br = null;  
26     try {  
27         br = new BufferedReader(new FileReader(path));  
28         br.readLine();  
29     } catch (IOException ex) {  
30         Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
31     } finally {  
32         try {  
33             br.close();  
34         } catch (IOException ex) {  
35             Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);  
36         }  
37     }
```

Hmmmm....NetBeans jeszcze nie pozwala skompilować takiej konstrukcji.

Zadeklarowany zasób (zasoby) zostanie automatycznie zamknięty (nawet w przypadku zgłoszenia wyjątku)

```
private void testAutomaticResourceManagement() throws IOException {  
    String path = "/home/grome/";  
    try (BufferedReader br = new BufferedReader(new FileReader(path)) {  
        br.readLine();  
    }  
}
```

Udoskonalone literały liczb całkowitych

- Korzyści:
 - W niektórych zastosowaniach wykorzystanie binarnej reprezentacji liczb ułatwia implementacje
 - Implementacja protokołów
 - Wykonanie operacji bitowych (przesunięcie)
 - Zapis liczb (np. maski bitowej) w formie szesnastkowej wymaga od programisty dodatkowej pracy
 - Zapis szesnastkowy mało intuicyjny i czytelny
 - np. `int BITMASK = 0x1E; // 0000 1111`
- Przykłady:

```
public static void main(String[] args) {  
    // Better integer literals  
  
    int bitMask = 0b00001111;  
    int bitMask01 = 0b0000_0010;  
    long testValue = 2535421;  
    long testValue1 = 2_535_421;  
  
    // Bit operation examples:  
  
    // Shift example (multiple by 2)  
    int testValue2 = 4;  
    int test = testValue2 * 0b0010;  
    System.out.println("New value is: " + test);  
}
```



Udoskonalona inferencja typów (operator **Dimond**)

- **Autor:** Jeremy Manson
- Korzyści:
 - Przykład:

```
Map<String, List<Integer>> map = new HashMap<String, List<Integer>>();
```

- Parametry przekazane w konstruktorze mogą być bezpiecznie pominięte – typy mogą być ustalone na podstawie kontekstu:

```
Map<String, List<Integer>> map = new HashMap<>();
```

- Przykłady

```
20 Map<String, List<String>> map = new HashMap<> (); // saves typing
21
22 private void testDimondOperator() {
23
24     map.put("Test", new ArrayList<>()); // compilation error
25
26     ArrayList<String> tempList = null;
27     map.put("Test", tempList = new ArrayList<>());
28     tempList.add("test");
29
30 }
```

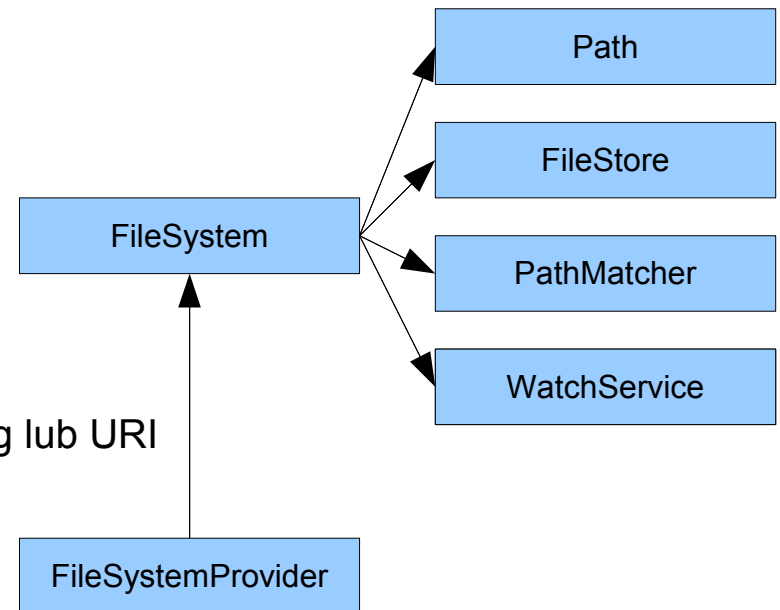
Udoskonalone API do operacji I/O (JSR 203)

- New I/O: Nowy interfejs do systemu plików
- New I/O: Wsparcie dla asynchronicznych operacji I/O (zarówno dla plików jak i socketów)
- New I/O: Udoskonalony SocketChannel



Udoskonalone API dostępu do systemu plików

- Problemy z klasą *java.io.File*:
 - Wiele metod zwracających rezultat true/false zamiast wyjątków, kiedy operacja się nie powiodła
 - Problemy podczas pracy z linkami symbolicznymi
 - Ograniczony dostęp do atrybutów plików
 - Problemy wydajnościowe:
 - Kosztowne zapytania o atrybuty plików (každorazowe odwołanie do systemu plików)
 - Obsługa katalogów – problemy wydajnościowe z dużymi katalogami
 - Brak metod do kopiowania lub przenoszenia plików
- Nowy pakiety: *java.nio.file*, *java.nio.file.attribute*
 - *FileSystem*: interfejs do systemu plików
 - *FileRef*: uchwyt do pliku lub katalogu
 - Definicja metod pracy na plikach/katalogach
 - *Path* (implementuje *FileRef*): uchwyt do pliku
 - Tworzona poprzez klasę *FileSystem* na podstawie String lub URI
 - *FileStore*: magazyn plików (urządzenie, partycja dysk etc.)



Udoskonalone API dostępu do systemu plików

- Tworzenie/odczyt/zapis plików z wykorzystaniem *java.nio.Path*:
 - Stream I/O: *newInputStream(OpenOption... options)*, *newOutputStream(OpenOption... options)*
 - Metody kompatybilne z pakietem *java.io*
 - Channel I/O: *java.nio.channels.SeekableByteChannel*
 - Kanał pozwalający na kontrolę pozycji w odczytywanym pliku
 - Utworzenie kanału:
 - *Path.newByteChannel(Set<? extends OpenOption> options, FileAttribute<?>... attrs)*
 - Jeżeli wykorzystamy domyślny dla danego systemu operacyjnego *FileSystem* to metoda zwróci obiekt *java.nio.channels.FileChannel*
 - Kontrola pozycji w pliku:
 - *position()*, *position(newPosition)*, *size()*
 - *Path.newByteChannel()* - tworzenie/odczyt plików
 - Możemy przekazać dodatkowe atrybuty: *READ*, *WRITE*, *APPEND*, *TRUNCATE_EXISTING*, *CREATE_NEW*, *CREATE*, *DELETE_ON_CLOSE*, *SYNC*, *DSYNC*
- Kopiowanie plików:
 - *copyTo(Path target, CopyOption... options)*
 - Dodatkowe opcje pozwalają na nadpisanie istniejącego pliku oraz na skopiowanie atrybutów pliku
- Przenoszenie plików:
 - *moveTo(Path target, CopyOption... options)*
 - Dodatkowe opcje: nadpisanie pliku, operacja atomowa



Udoskonalone API dostępu do systemu plików

- Obsługa linków symbolicznych:

- Tworzenie linków symbolicznych
- skopiowanie

- Direct

```
PosixFileAttributes attr = Attributes.readPosixFileAttributes(testPath);  
String mode = PosixFilePermissions.toString(attr.permissions());  
System.out.println(mode); // "rwxr-x--x"  
  
Set<PosixFilePermission> perm = PosixFilePermissions.fromString("rwxr-xr--");  
Attributes.setPosixFilePermissions(testPath, perm);
```

file

Symbolicznych)

- BasicFileAttributes** – dostarcza informacje o plikach

- Dostarczony we ws
- Atrybuty: *isDirectory()*, *isFile()*, *isSymbolicLink()*, *lastAccessTime()*

- Specyfikacja definiuje inne

- PosixFileAttributes** – *group()*, *owner()*, *permissions()*
- DosFileAttributes** – *isArchive()*, *isHidden()*, *isReadOnly()*, *isSystem()*

Closures

- Wyrażenie lambda (definiowanie domknięć)
- Typy funkcyjne
- Funkcje jako obiekty pierwszego rzędu
- Konwersja domknięć

Closures

- Gorąca dyskusja czy Java powinna wspierać domknięcia
- Domknięcia pozwoliłyby na:
 - Zastąpienie klas anonimowych lepszym i wygodniejszym mechanizmem
 - Weliminowanie sytuacji w których jako parametr metody przekazujemy implementację interfejsu z jedną metodą, która zostanie następnie wykonana
 - np. komparator, który przekazujemy do TreeMap
 - Uproszczenie implementacji równoległych programów (wykorzystujących ParallelArray)
- Plan JDK 7 został przesunięty na październik 2010 – dało to więcej czasu na implementację między innymi domknięć (Mark Reinhold, Devovx 2009)
- Prace nad domknięciami prowadzone w ramach projektu Lambda
 - Jeżeli projekt da pozytywne rezultaty powstanie JSR
- Do tej pory powstały 3 różne propozycje domknięć: BGGA, CICE oraz FCM
 - Projekt Lamda najbardziej zbliżony do FCM (np. wykorzystanie operatora #)
 - W obecnej formie jeszcze nieimplementowalny...



Clousers – propozycja Straw-Man

- Wyrażenie lambda - #
 - `#()(42)` – funkcja bez parametrów zwracająca 42
 - `#(int x, int y)(x * y)` – funkcja, która przyjmuje dwa parametry i zwraca ich iloczyn
- Typ funkcyjny
 - `#int() fortyTwo = #()(42);`
 - `#int(int,int) multiplier = #(int x, int y)(x * y);`
- Konwersja domknięć:

```
Thread th = new Thread(new Runnable() {  
    public void run() {  
        doSomeStuff();  
        doMoreStuff();  
    }  
});
```



```
Thread th = new Thread(  
    #(){ doSomeStuff(); doMoreStuff(); }  
);  
#() doStuff = #(){ doSomeStuff(); doMoreStuff(); }  
Thread th2 = new Thread(doStuff);
```

Modularyzacja JDK



„JAR/Classpath hell is dead”
Mark Reinhold, JavaOne 2009

Modularyzacja JDK

- JDK jest bardzo dużym, niepodzielnym, monolitycznym pakietem
 - JDK 1.1 – 7 głównych pakietów, około 200 klas i interfejsów
 - JDK 7 – duża liczba pakietów, ponad 4000 klas i interfejsów
 - ~2.035.060 linii kodu (kod, komentarze)
 - ~7000 plików źródłowych w ~480 katalogach
 - Sposób rozwoju JDK spowodował powstanie silnych zależności pomiędzy różnymi częściami platformy
 - Aplikacja HelloWorld potrzebuje obecnie 300 klas do inicjalizacji
 - Nie ma możliwości wykorzystania tylko części platformy (np. na mniejszych urządzeniach)
- JAR/Classpath hell
 - Brak definiowania zależności
 - Brak wersjonowania
 - Brak możliwości enkapsulacji wewnętrznych klas i interfejsów (w obrębie danego modułu)
- Brak dobrze zdefiniowanych zależności z systemem instalacji programów/pakietów w systemie operacyjnym
- Wydajność: start-up time, zużycie pamięci



Modularyzacja JDK

- Modularyzacja platformy oraz modularyzacja kodu aplikacji
 - Zależności
 - Lista zależności od innym modułów
 - Wersjonowanie
 - Wersjonowanie aplikacji i zależności
 - Enkapsulacja
 - Ukrycie wewnętrznych klas i interfejsów modułu
- System modularyzacji
 - JSR 294 – wprowadzenie zmian w języku oraz maszynie wirtualnej, które będą wspierały modularyzację
 - Jigsaw – system modularyzacji w JDK7, nie jest częścią JSR
 - Implementacja referencyjna (nie będzie oficjalną częścią Java SE 7)
 - System modularyzacji (np. Jigsaw) oraz OSGi będą korzystały ze zmian wprowadzonych w JSR 294
 - Dlaczego nie wykorzystać sprawdzonego OSGi?
 - OSGi wykorzystuje maszynę wirtualną podczas gdy celem projektu Jigsaw jest modularyzacja nie tylko aplikacji ale i JDK
 - System modularyzacji powinien być dobrze zintegrowany z maszyną wirtualną oraz językiem

Informacje statyczne:

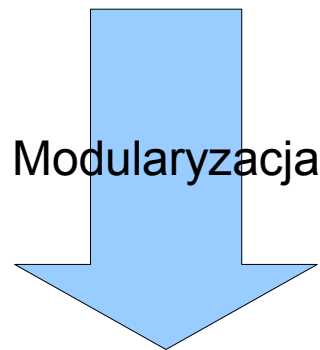
- wczesne wykrycie problemów, które pojawiają się po uruchomieniu
- środowisko w trakcie kompilacji odzwierciedla środowisko uruchomieniowe



Modularyzacja JDK – modularyzacja kodu

/home/grom/src/

conference/poznan/forDevelopers/Main.java



/home/grom/src/

conference/poznan/forDevelopers/Main.java

module-info.java

Modularyzacja JDK – modularyzacja kodu

Plik module-info.java

```
module conference.poznan.forDevelopers {
```

```
    system jigsaw;
```

```
    requires module jdk.base @ 7.*;
```

```
    requires module jdk.boot @ 7.*;
```

```
    requires module rome;
```

```
    ....
```

```
    requires module test @ =1.0;
```

```
    requires module java-joda @ [1.2, 2.5];
```

```
    ....
```

```
    provides module forDevelopers;
```

```
    class conference.poznan.forDevelopers.Main.java
```

```
}
```

Nazwa modułu

System modularyzacji

Logiczny classpath

Co dany moduł eksportuje

Klasa główna modułu

Modularyzacja JDK – modularyzacja kodu

- Kompilacja kodu

```
javac -modulepath /home/grom module-info.java  
conference/poznan/forDevelopers/*.java
```

- Uruchomienie aplikacji

```
java -modulepath /home/grom -m forDevelopers
```

- Przygotowanie pakietu instalacyjnego

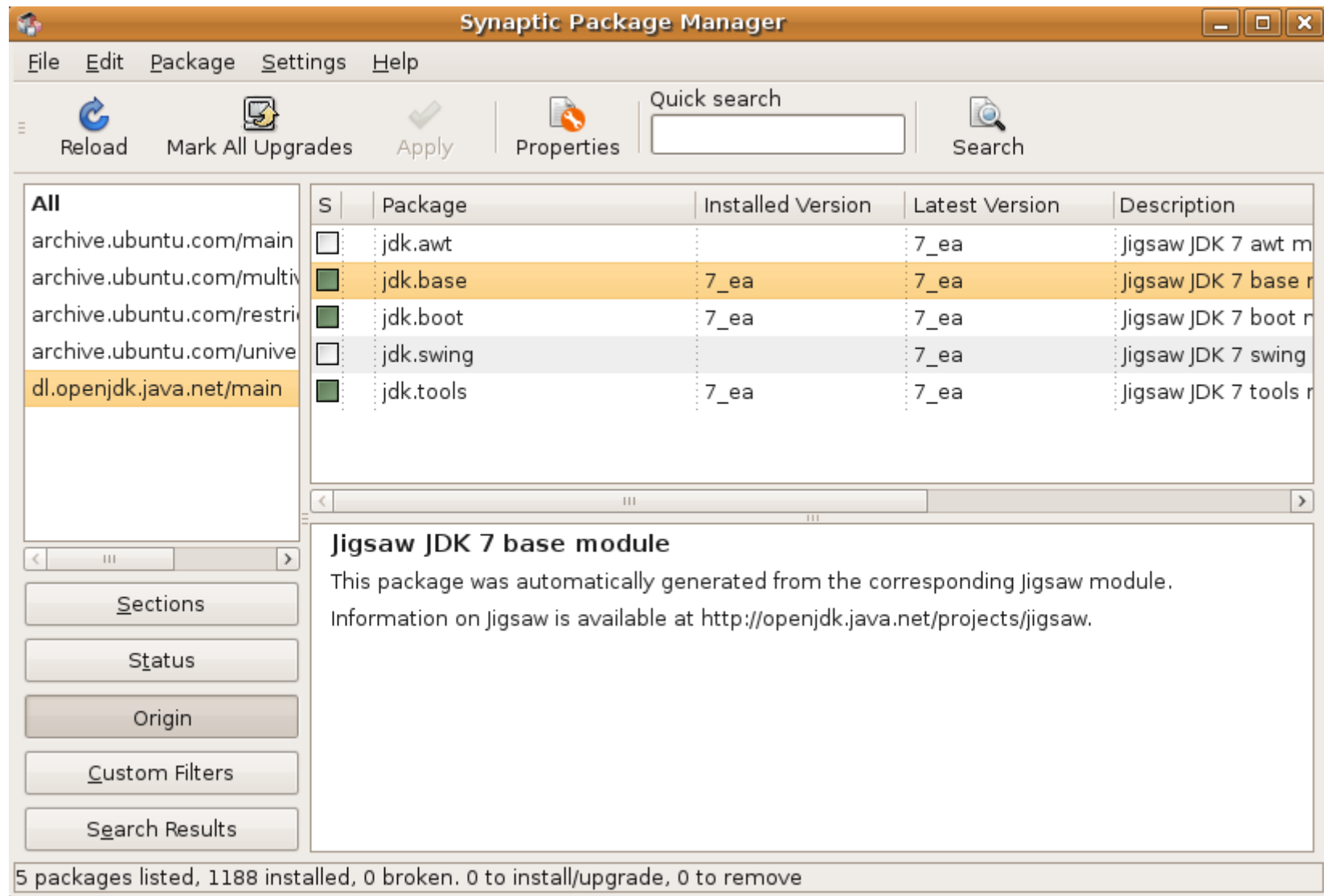
```
jpkg -m /home/grom deb conference.poznan.forDevelopers
```

- Dodatkowe narzędzia

- jmod – opcje: config, install, list, preinstall, show



Modularyzacja JDK - Demo



I wiele innych zmian w JDK7...

- JSR 308 – udoskonalone anotacje typów
 - Między innymi anotacje typów generycznych, anotacje rzutowania:
 - *Map<@NonNull String, @NonEmpty List<@ReadOnly Document>> files;*
 - *@ReadOnly Document [][] docs1 = new @ReadOnly Document [2] [12]; // array of arrays of read-only documents*
 - *Collection<? super @Existing File>*
 - *myString = (@NonNull String) myObject;*
 - Checker Framework – framework pozwalający na wykrycie potencjalnych błędów na podstawie anotacji
 - Możliwość pisania własnych pluginów
- Algorytmy równoległe: `java.util.concurrent.forkjoin`, `ParallelArray`, JSR 166



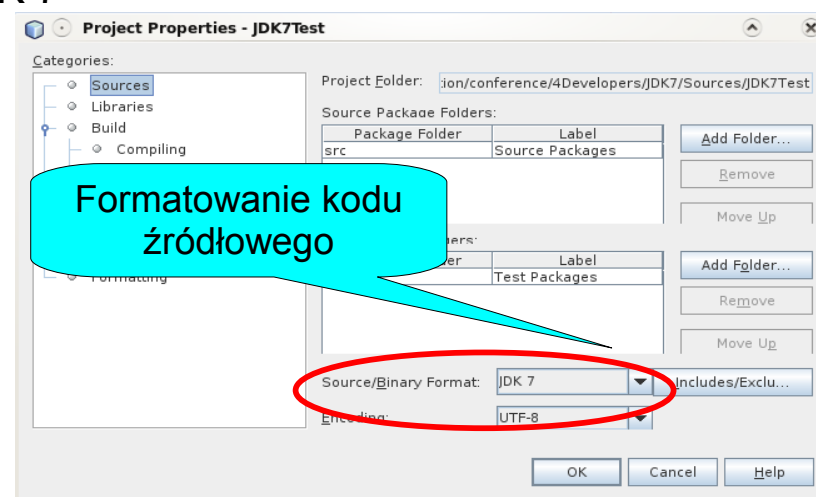
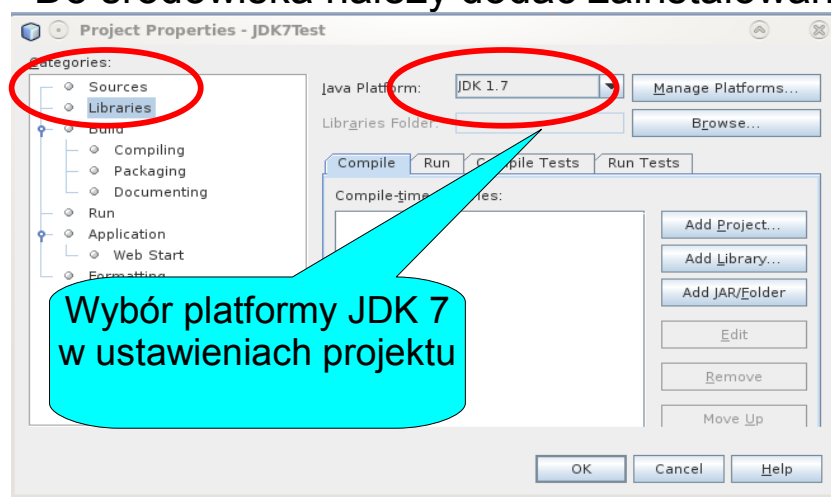
Jak rozpocząć pracę z JDK 7?

I. Instalacja JDK

- Źródła oraz binaria dostępne poprzez [Java Early Access Downloads](#)
- Obecnie dostępny **Milestone 5** (każdy milestone składa się z 5 do 9 buildów; kolejny build wykonywany jest co tydzień)
 - Formalnie nie będą pojawiały się wersje *beta*

II. Przygotowanie IDE

- W wersji 6.8 NetBeans wspiera JDK 7 (brakuje jednak wsparcia między innymi dla projektu *Coin*, które pojawi się w późniejszych wersjach)
- Do środowiska należy dodać zainstalowane JDK 7



module-info.java

```
module conference.forDevelopers.JDK7.pytania {  
    requires module solidna.prezentacja @ 7.0;  
    requires module przyklady @ 7.0;  
    requires module zakonczyc.o.czasie @ 7.0;  
  
    provides module odpowiedzi;  
}
```



Referencje

- Strona projektu OpenJDK:
<http://openjdk.java.net/projects/jdk7/milestones/>
- Blog Marka Reinholda:
<http://blogs.sun.com/mr/>

