

# Nowości w świecie programowania współbieżnego w Java 7

Pakiet `java.util.concurrent` (jsr 166y)

**Jacek Laskowski**

[jacek@japila.pl](mailto:jacek@japila.pl)

<http://www.JacekLaskowski.pl>

wersja 1.0, 07.11.2011

# Ja...cek Laskowski

- **Entuzjasta** Java EE, OSGi, SCA oraz programowania funkcyjnego (Clojure)
- Założyciel i lider **Warszawa JUG**
- Organizator **Javarsovia**, **Confitura** oraz **warsjava**
- Członek zespołu **NetBeans DreamTeam**
- Blogger na **<http://JacekLaskowski.pl>** oraz **<http://blog.japila.pl>**
- Twittuje jako **@jaceklaskowski**
- Członek zespołów **Apache Geronimo** i **Apache OpenEJB**
- Specjalista produktów **IBM WebSphere** w **IBM Polska**



# Pakiet jsr 166y = Fork/Join et al

- Pakiet **jsr 166y** - nowe klasy w `java.util.concurrent` w Java 7
- Część zmian w **java.util.concurrent** już w Java 5
- <http://gee.cs.oswego.edu/dl/concurrency-interest/index.html>
- <http://gee.cs.oswego.edu/dl/jsr166/dist/jsr166ydocs/>
- Do użycia w Java 6 z `-Xbootclasspath/p:jsr166.jar`

# Motywacja jsr 166y

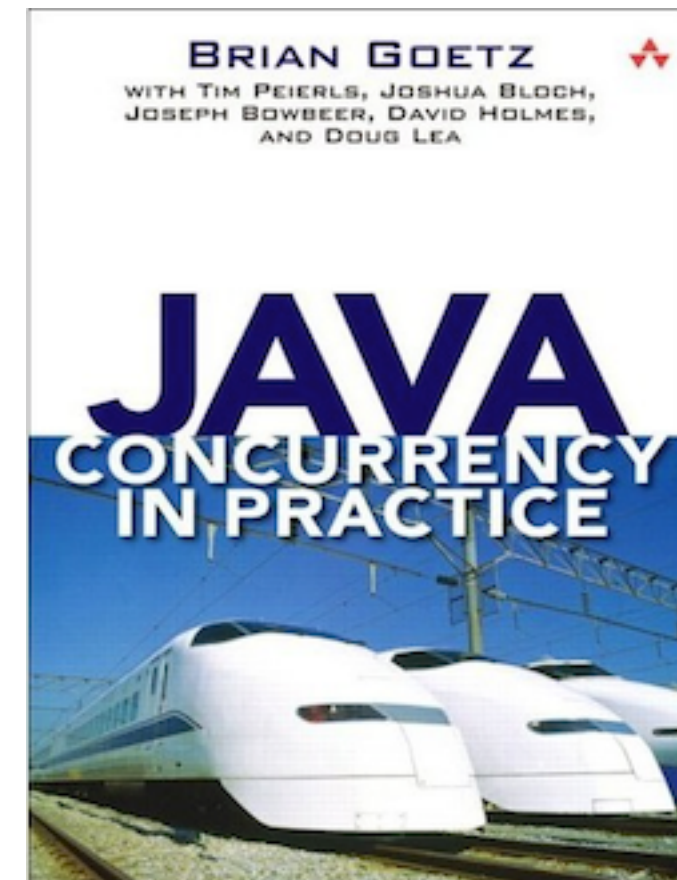
- Uproszczenie **programowania współbieżnego** na platformie Java SE na podobieństwo wymiaru zmian wokół pakietu *Collections* do obsługi struktur danych
- *“Such facilities are notoriously hard to get right and even more difficult to optimize. The concurrency facilities written by application programmers are often incorrect or inefficient.”*  
2.3 What need of the Java community will be addressed by the proposed specification? <http://www.jcp.org/en/jsr/detail?id=166>
- Konstrukcje wyższego poziomu (niż *synchronized*, *volatile*, *Object.wait()* oraz *Object.notify()*)
- Pule wątków, kolejki blokujące

# Kody źródłowe jsr 166y

- Repozytorium <http://gee.cs.oswego.edu/cgi-bin/viewcvs.cgi/jsr166>
- Building JSR-166 version 1.7.0
  - jacek:~/oss/jsr166  
\$ ant jsr166ydist
  - jacek:~/oss/jsr166  
\$ ant test

# JSR 166 Concurrency Utilities

- <http://jcp.org/aboutjava/communityprocess/final/jsr166/index.html>
- Prowadzący: **Doug Lea**
- W grupie również Brian Goetz, Josh Bloch, Cliff Click, Tim Peierls
- Autorzy *Java Concurrency in Practice*, Addison-Wesley, 2006  
<http://www.informit.com/store/product.aspx?isbn=0321349601>



# Co nowego w `java.util.concurrent`?

- Szkielet **Fork/Join**
  - klasa *`java.util.concurrent.ForkJoinPool`*
- **ThreadLocalRandom**
- **Phaser**

# Fork/Join

- Algorytm “podkradania zadań”
- balansowanie obciążenia równoległe pracujących wątków w celu zwiększenia efektywności wykorzystywania procesorów/rdzeni
- Wątek bez zadań podkrada zadania zajęтым wątkom
- Współbieżna wersja dla algorytmów “*dziel i zwyciężaj*”

# Interfejs Executor

- **java.util.concurrent.Executor**
- Dostępny od wersji Java 5
- **void execute(Runnable zadanie)**
- Wykonuje zarejestrowane zadanie
- Zadanie może być wykonane w nowym wątku, w wątku z puli wątków lub w bieżącym wątku w zależności od rodzaju Executor
- Oddzielenie “co” (zadanie) od “jak” (planowanie wykonania)

# Executor z nowym wątkiem

```
class ThreadPerTaskExecutor implements Executor {  
    public void execute(Runnable r) {  
        new Thread(r).start();  
    }  
}
```

# Executor z wątkiem z puli wątków

```
class ThreadPoolExecutor implements Executor {  
    public void execute(Runnable r) {  
        Worker w = ... // pobierz wątek z puli  
        w.start(r);  
    }  
}
```

# Executor z bieżącym wątkiem

```
class DirectExecutor implements Executor {  
    public void execute(Runnable r) {  
        r.run();  
    }  
}
```

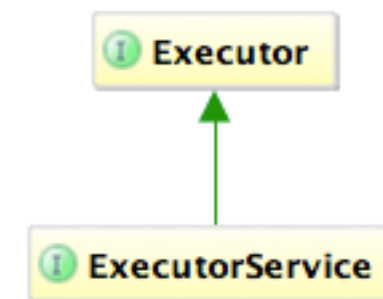
# Executor w akcji

```
Executor executor = ... // pewien Executor  
executor.execute(new RunnableTask1());  
executor.execute(new RunnableTask2());
```

# Interfejs ExecutorService

- **java.util.concurrent.ExecutorService**
- Dostępne od wersji Java 5
- Specjalizowany Executor, który umożliwia:
  - Wyłączenie przyjmowania zadań
  - Zgłaszanie kolekcji zadań
  - Zadania asynchroniczne z *Future*

Diagram for ExecutorService

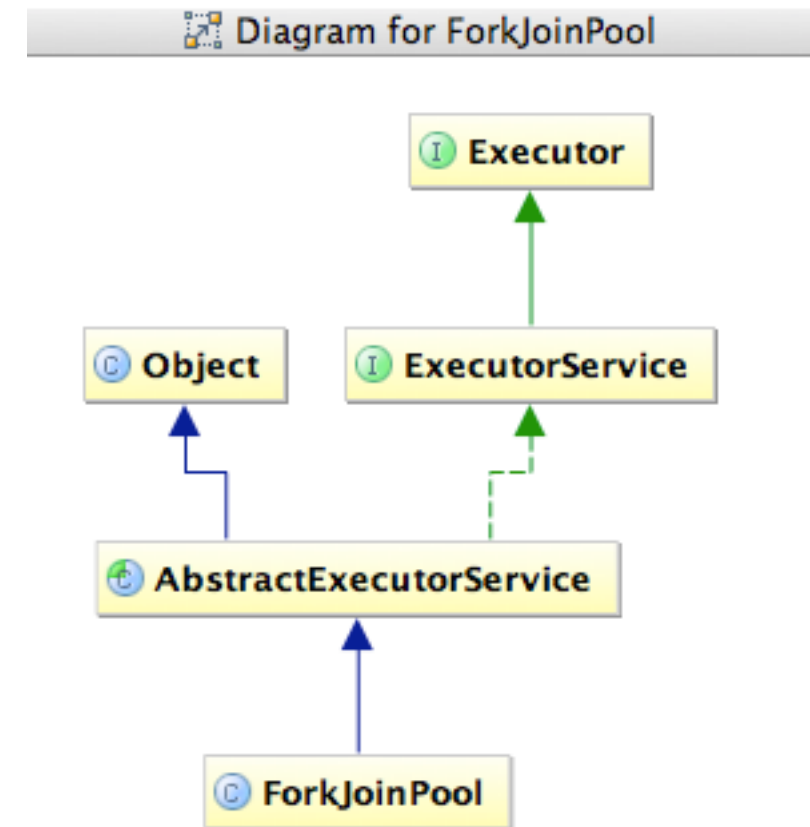


# Typy **Fork/Join**

## Java 7

# Klasa ForkJoinPool

- **java.util.concurrent.ForkJoinPool**  
<http://download.oracle.com/javase/7/docs/api/java/util/concurrent/ForkJoinPool.html>
- ExecutorService dla **ForkJoinTask**



# Klasa ForkJoinTask<V>

- **java.util.concurrent.ForkJoinTask<V>**  
<http://download.oracle.com/javase/7/docs/api/java/util/concurrent/ForkJoinTask.html>
- Implementuje **Future<V>** - wynik operacji asynchronicznej
- Klasa nadrzędna dla zadań w **ForkJoinPool**
- Lżejszy “wątek”
- *“computational tasks calculating pure functions or operating on purely isolated objects”*



# Metody i zasady ForkJoinTask<V>

- **fork()** rozpoczyna asynchroniczne obliczenia
- **join()** oczekuje wyniku obliczenia
- Niezalecane operacje w ramach obliczeń:
  - unikanie *synchronized*
  - unikanie blokowanych operacji, poza join() lub Phasers
  - unikanie blokowanego IO
  - dostęp do zmiennych niewspółdzielonych między zadaniami

# Klasa RecursiveAction

- **java.util.concurrent.RecursiveAction**

<http://download.oracle.com/javase/7/docs/api/java/util/concurrent/RecursiveAction.html>

- Rekurencyjny **ForkJoinTask<Void>** bez zwracania obliczenia
- Metody typu *join()* zawsze zwracają **null**
- Przykład: sortowanie

# Klasa RecursiveTask<V>

- **java.util.concurrent.RecursiveTask<V>**

<http://download.oracle.com/javase/7/docs/api/java/util/concurrent/RecursiveTask.html>

- Rekurencyjny **ForkJoinTask** zwracający wynik obliczenia
- Przykład: obliczanie liczb Fibonacciego

# Trochę praktyki...

- Obliczanie kwadratów liczb w tablicy
- `invokeAll() = fork() + join()`



# Klasa ThreadLocalRandom

- **java.util.concurrent.ThreadLocalRandom**

<http://download.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadLocalRandom.html>

- **Generator liczb pseudolosowych** dedykowany dla wyłącznie jednego wątku lub obliczenia w ForkJoinTask
- **ThreadLocalRandom.current()** - zamiennik dla *Math.random()*
- `ThreadLocalRandom.current().nextX(p1)` dla long, double
- `ThreadLocalRandom.current().nextX(p1, p2)` dla int, long, double

# Phaser

- **java.util.concurrent.Phaser**

<http://download.oracle.com/javase/7/docs/api/java/util/concurrent/Phaser.html>

- Synchronizator (bariera, ang. *barrier*) podobny do *CyclicBarrier* i *CountDownLatch*
- Wiele etapów synchronizacji = wiele faz wykonywanej aktywności
- Liczba synchronizowanych zmienna w czasie
- Phaser można “układać” w strukturę drzewiastą, aby zniwelować współbieganie się o współdzielony zasób

# Zastosowanie Phaser

- Aplikacja do zamówień (za “*Java, The Complete Reference*”, 8th Edition Herberta Schildta, str. 875)
- Krok 1: Kontrola poprawności danych klienta, stanu i płatności
- Krok 2: Wyliczenie ceny oraz płatności dodatkowe (VAT)
- Krok 3: Potwierdzenie płatności i wyliczenie czasu dostawy
- Każdy z kroków to etap dla Phaser, a każde zadanie to wątek

# Phaser w akcji

- Tworzymy instancję Phaser
  - **Phaser()** - 0 zarejestrowanych
  - **Phaser(numParties)** - *numParties* zarejestrowanych
- int **register()** - zwraca numer etapu
- int **arrive()** lub int **arriveAndAwaitAdvance()**
- Phaser przechodzi do kolejnego etapu lub kończy działanie

# Java Concurrent Animated

Java provides the concurrent library that simplifies concurrent programming, but this is hard to learn and visualize. This project is a series of animations each illustrating the coding and usage of a component in the java concurrent library.

<http://sourceforge.net/projects/javaconcurrenta/>

# Pytania?

**Jacek Laskowski**

[jacek@japila.pl](mailto:jacek@japila.pl)

<http://www.JacekLaskowski.pl>