# Wielowątkowość

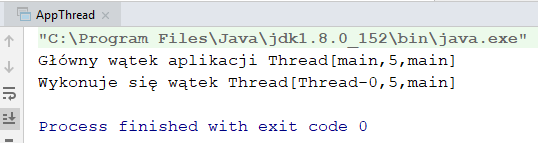
## Wstęp

Proces to każdy działający program w pamięci. Proces ma kawałek pamięci dostępny tylko dla siebie. W każdym procesie istnieje co najmniej jeden wątek zwany wątkiem głównym. Z tego wątku możemy tworzyć wątki poboczne. Wszystkie wątki w obrębie jednego procesu współdzielą ze sobą pamięć. Dzięki temu komunikacja między wątkami jest bardzo szybka a to współdzielenie pamięci jest źródłem wszystkich problemów związanym z programowaniem wielowątkowym.

## Rozszerzenie klasy Thread

**public class** MyThread **extends** Thread {  
 @Override  
 **public void** run(){  
 System.***out***.println(**"Wykonuje się wątek "**+MyThread.*currentThread*());  
 }  
}

**public class** AppThread {  
 **public static void** main(String[] args) {  
 System.***out***.println(**"Główny wątek aplikacji**+Thread.*currentThread*());  
 Thread thread=**new** MyThread();  
 thread.start();  
 }  
}



Nazwy main i Thread-0 otrzymamy jeśli wyświetlanie nazwy wątku zapiszemy jako

MyThread.*currentThread*().getName());

### A jak ustalić własną nazwę dla wątku?

Należy napisać konstruktor i odpowiednio utworzyć obiekt naszego wątku

**public** MyThread(String name) {  
 **super**(name);  
}

Thread thread=**new** MyThread(**"mój wątek"**);

### Metoda start i run

// częste pytanie na rozmowach kwalifikacyjnych

Metoda **start** uruchamia nowy wątek i uruchamia metodę **run**. Gdybyśmy uruchomili metodę run dla wątku, to nasz kod **nie wykona się w osobnym wątku, ale w wątku głównym aplikacji**.

Aktualną wersję plików znajdziesz na

<https://github.com/idzikdev/LearnJava/blob/master/src/main/java/watki/MyThread.java>

<https://github.com/idzikdev/LearnJava/blob/master/src/main/java/watki/AppThread.java>

## Implementacja interfejsu Runnable

**public class** MyRunnable **implements** Runnable {  
 @Override  
 **public void** run() {  
 System.***out***.println(**"Główny wątek aplikacji"**+Thread.*currentThread*());  
 }  
}

Runnable runnable=**new** MyRunnable();  
Thread thread3=**new** Thread(runnable,**"Wątek nr 3"**);  
thread3.start();

### Czy trzeba tworzyć osobną klasę MyRunnable?

Nie, wykorzystamy anonimową klasę.

Runnable runnable2=**new** Runnable() {  
 @Override  
 **public void** run() {  
 IntStream.*rangeClosed*(0,100).forEach(i->System.***out***.println

(i+**" Wykonuje się "**+MyThread.*currentThread*().getName()));  
 }  
};

I ten obiekt (runnable2) można teraz przekazać jako parametr do inicjalizacji Threada.

Thread thread2=**new** Thread(runnable12,**"Wątek nr 2"**);

// częste pytanie na rozmowach kwalifikacyjnych

Ten sposób wstrzykiwania runnable2 do konstruktora nazywa się **wzorcem strategii**.

### Można jeszcze wszystko uprościć stosując wyrażenie lambda.

Thread thread5=**new** Thread**(()->IntStream.*rangeClosed*(0,100)  
 .forEach(i->System.*out*.println(i+" Wykonuje się "+MyThread.*currentThread*().getName()))**,**"Wątek nr 5"**);

## To extends Thread czy implements Runnable

Lepiej będziemy stosować implementację interfejsu Runnable, bo

1. Nie zamykamy sobie drogi do dziedziczenia
2. Można stosować lambdę (mało kodu i przejrzystość)

## Thread.sleep(czas)

Usypia wątek na tyle milisekund ile podamy w nawiasie. Jednak nie ma pewności, że po tym czasie wątek wróci do działania. Ten czas może być większy, gdy procesor jest czymś zajęty.

Zamiast sleep() można użyć

TimeUnit.***SECONDS***.sleep(1);

(Wprowadzono to od Javy 1.8)

## Join()

Aktualny wątek czeka na wątek na którym uruchomimy tę metodę.

thread4.start();  
**try** {  
 thread4.join();  
} **catch** (InterruptedException e) {  
 e.printStackTrace();  
}  
thread5.start();

thread5 poczeka zatem na zakończenie thread4

W join można podać maksymalną liczbę ms przez którą będzie czekał następny wątek na zakończenie pierwszego wątku.

Aktualną wersję plików znajdziesz na

## Executor Service

Jest to Framework zarządzający wielką liczbą wątków a nie kilkoma ;)