3. Wątki i procesy

Tworzenie wątków umożliwia podział aplikacji na niezależne podprogramy, wykonujące współbieżnie określone zadania. Taki podział programu zwiększa wydajność aplikacji (pobieranie danych, odtwarzanie muzyki, obróbka grafiki itd.)

Tworzenie wątków

Wątki w Javie reprezentowane są przez obiekty klasy `Thread` bądź też obiekty z niej dziedziczące. Kod wykonywany jako wątek zawarty jest w ciele metody `run()` klasy `Thread` (bądź też obiektów z niej dziedziczących) lub klasy implementującej interfejs `Runnable`.

Wykorzystanie klasy `Thread` sprowadza się do utworzenia klasy pochodnej, nadpisania metody `run()`

```
class Watek extends Thread {
  void run() {
        // ciało metody run
  }
}
```

oraz stworzenia obiektu nowo utworzonej klasy.

```
Watek nowyWatek = new Watek();
```

Tworzenie wątków za pomocą interfejsu `Runnable` to stworzenie klasy implementującej ten interfejs

```
class Watek implements Runnable {
  void run() {
        // ciało metody run
  }
}
```

oraz utworzenie obiektu tej klasy i przekazanie go do konstruktora klasy `Thread`.

```
Watek obiekt = new Watek();
Thread nowyWatek = new Thread(obiekt);
```

Dodatkowo tworząc wątki możemy nadawać im nazwy (wykorzystujemy wtedy odpowiednie konstruktory klasy `Thread` lub metodę `setName()`). Do odczytu nazwy wątku przeznaczona jest metoda `getName()`.

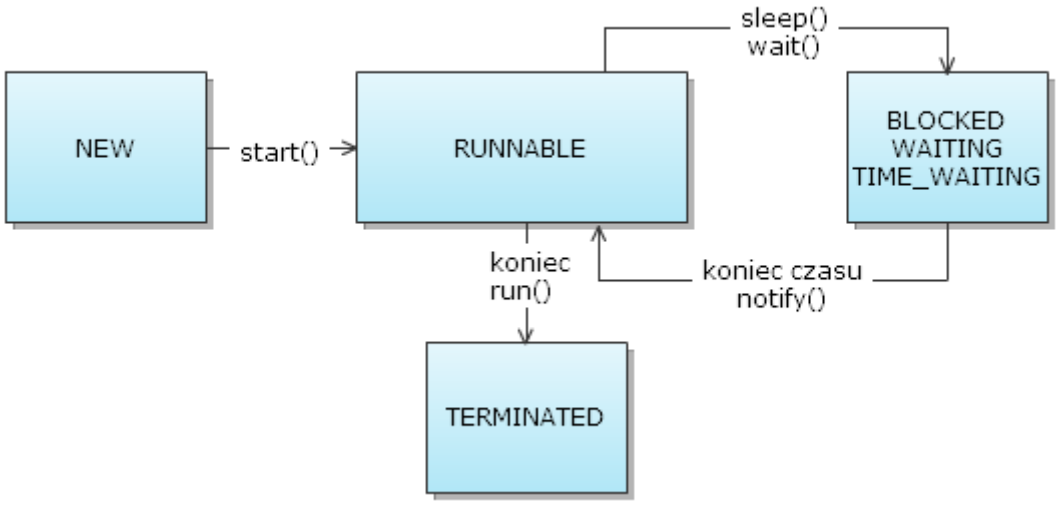
Najczęściej stosowanym sposobem tworzenia wątków jest wykorzystanie interfejsu `Runnable` (dziedziczenie w Javie jest jednokrotne). Warto również pamiętać, że główny program jest także wątkiem.

Cykl życia wątku

Każdy wątek może znajdować się w następujących stanach:

- **NEW** - wątek utworzony, ale jeszcze nie uruchomiony,
- **RUNNABLE** - wątek uruchomiony (może działać lub też nie),
- **BLOCKED** - wątek zablokowany,
- **WAITING** - wątek oczekujący na inny wątek,
- **TIMED_WAITING** - wątek oczekujący na inny wątek (do czasu),
- **TERMINATED** - wątek zakończony.

Cykl życia wątku czy też powiązania pomiędzy jego stanami przedstawia poniższy diagram.



Aktualny stan wątku możemy zawsze sprawdzić za pomocą metody `getState()`.

Uruchamianie i zatrzymywanie wątków

Uruchomienie wątku to wywołanie metody `start()`. Powoduje ona uruchomienie kodu zawartego w metodzie `run()` jako osobne zadanie. Poniżej przykładowa aplikacja wykorzystująca klasę `Thread`.

```
class Napis extends Thread {

 private String napis;
 private int pauza;

 public Napis(String napis, int pauza) {
        this.napis = napis;
        this.pauza = pauza;
 }

 public void run() {
        for (int i = 0; i < 10; i++) {
               System.out.print(napis);
               try {
                      sleep(pauza);
               } catch(InterruptedException e){}
        }
 }
}

public class Test {
 public static void main(String [] args) {
        Napis napis1 = new Napis("Rower", 1000);
```

```
        Napis napis2 = new Napis("Narty", 800);

        napis1.start();
        napis2.start();
  }
}
```

Ta sama aplikacja z wykorzystaniem interfejsu `Runnable`.

```
class Napis implements Runnable {
  private String napis;
  private int pauza;

  public Napis(String napis, int pauza) {
        this.napis = napis;
        this.pauza = pauza;
  }

  public void run() {
        for (int i = 0; i < 10; i++) {
              System.out.print(napis);
              try {
                    Thread.sleep(pauza);
              } catch(InterruptedException e){}
        }
  }
}

public class Test {
  public static void main(String [] args) {
        Napis napis1 = new Napis("Rower", 1000);
        Napis napis2 = new Napis("Narty", 800);

        Thread watek1 = new Thread(napis1);
        Thread watek2 = new Thread(napis2);

        watek1.start();
        watek2.start();
  }
}
```

Wynik działania aplikacji:

```
RowerNartyNartyRowerNartyRowerNartyRowerNartyRowerNartyNartyRowerNartyRowerNartyRower
```

Programista ma możliwość zmiany stanu wątków poprzez metody `sleep()` oraz `yield()`. Pierwsza z nich usypia dany wątek na określoną liczbę milisekund, a druga wstrzymuje wykonywanie bieżącego wątku (daje możliwość dostępu do czasu procesora innych oczekującym wątkom - podział czasu jest wtedy bardziej równomierny).

Każdy wątek może mieć również określony priorytet. Wątki o wyższym priorytecie będą wykonywane szybciej niż wątki o priorytecie niższym. Do zmiany i odczytu priorytetu służą metody `setPriority()` oraz `getPriority()`. Ze względu na zróżnicowanie środowisk, w których może zostać uruchomiona aplikacja najczęściej do określenia priorytetu wykorzystuje się stałe dostępne w klasie `Thread` (`MIN_PRIORITY`, `MAX_PRIORITY`, `NORM_PRIORITY`).

Znacznym ułatwieniem w pracy z wątkami jest umożliwienie wątkowi oczekiwania na zakończenie pracy przez inny wątek. Wykorzystywana do tego celu jest metoda `join()`. Poniżej prosty przykład jej wykorzystania.

```
class Watek extends Thread {

  public void run() {
```

```
        for (int i = 0; i < 10; i++) {
               System.out.print("...");
               try {
                      sleep(500);
               }
               catch(InterruptedException e){}
        }
        System.out.println();
  }
}

public class Test {
  public static void main(String [] args) {
        Watek nowyWatek = new Watek();

        nowyWatek.start();

        System.out.println("Czekam na zakończenie wątku");

        try {
               nowyWatek.join();
        } catch (InterruptedException e) {}

        System.out.println("wykonuje dalszą część programu");
  }
}
```

Wynik działania aplikacji.

```
Czekam na zakończenie wątku
...............................
wykonuje dalszą część programu
```

Wątek przestaje działać z chwilą zakończenia instrukcji zawartych w metodzie `run()`. Istnieje możliwość przerwania pracy wątku poprzez wywołanie metody `interrupt()`. Przechwycenie wyjątku `InterruptedException` umożliwia obsługę tego sposobu przerwania pracy wątku.

```
try {
   Thread.sleep(10000);
} catch(InterruptedException e) {
   System.out.println("Zostałem przerwany!");
}
```

Metoda `isAlive()` sprawdza czy dany wątek wykonuje się dalej czy też zakończył już swoją pracę.

Wątki demony

Wątki demony, to wątki usługowe. Ich zadaniem jest służenie innym wątkom. Zazwyczaj są to wątki związane z czasem (wątki zegara), lub wątki czyszczące zbędne dane (standardowy *garbage collection* wbudowany w Javę). Podstawową różnicą w stosunku do zwykłych wątków jest sposób kończenia pracy. Wątki demony działają przez cały czas życia aplikacji. Aplikacja może zostać zakończona nawet jeśli w tle pracują wątki demony (w przypadku zwykłych wątków aplikacja czeka na ich zakończenie). Aby wątek stał się wątkiem demonem musi zostać wywołana metoda `setDaemon(true)`. Wywołanie tej metody musi nastąpić przed uruchomieniem wątku. Zawsze można sprawdzić czy dany wątek jest demonem czy też nie (wywołanie metody `isDaemon()`).

Poniższy kod tworzy wątek demon i sprawdza jego działanie.

```
class WatekDemon extends Thread {

    public WatekDemon() {
        setDaemon(true);
```

```
    }

    public void run() {
        while (true) {
            System.out.println("Tutaj wątek demon");
            try {
                sleep(1000);
            } catch (InterruptedException e) {
            }
        }
    }
}

public class TestDemona {
    public static void main(String[] args) {

        new WatekDemon().start();

        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {}

        System.out.println("Aplikacja kończy swoje działanie");
    }
}
```

Wynik działania programu:

```
Tutaj wątek demon
Tutaj wątek demon
Tutaj wątek demon
Tutaj wątek demon
Tutaj wątek demon
Tutaj wątek demon
Tutaj wątek demon
Tutaj wątek demon
Tutaj wątek demon
Tutaj wątek demon
Aplikacja kończy swoje działanie
```

Procesy

Procesy umożliwiają uruchamiane zewnętrznych aplikacji systemu operacyjnego. Umożliwia to wykorzystanie poleceń systemowych wewnątrz własnych aplikacji. Są uruchamiane niezależnie od innych procesów i nie mają dostępu do danych występujących w pozostałych procesach. Zasoby związane z procesem są przydzielane z poziomu systemu operacyjnego. Programista nie ma na nich wpływu. Standardowa aplikacja Javy jest uruchamiana jako jeden proces, który może posiadać wiele wątków (często wątki nazywane są lekkimi procesami). Poniżej przykładowe utworzenie procesu uruchamiającego systemowy notatnik.

```
try {
    Process p = new ProcessBuilder("notepad.exe", "").start();
} catch (Exception e) {
    System.out.println("Błąd: " + e.getMessage());
}
```

Zadania

1. Napisz klasę `Watek` rozszerzającą możliwości klasy `Thread` oraz umożliwiającą tworzenie wątków z nazwą (odpowiedni konstruktor `Watek(String)`). Zadaniem wątku będzie pięciokrotne wyświetlenie na ekranie łańcucha tekstowego „Watek: ” oraz nazwy wątku. Po każdym wyświetleniu wątek powinien zostać uśpiony na czas jednej sekundy. W celu przetestowania aplikacji utwórz 10 elementową tablicę wątków i wypełnij ją obiektami klasy `Watek` z odpowiednimi nazwami (W1, W2 itd.). Po uzupełnieniu tablicy uruchom wszystkie wątki. Zaobserwuj wyniki na ekranie.

```
Watek[] tablicaWatkow = new Watek[10];
```

2. Zmodyfikuj zadanie pierwsze tak, by klasa `Watek` implementowała interfejs `Runnable`, a nie dziedziczyła z klasy `Thread`. Jakie problemy napotykasz? Jak stworzyć wątki z nazwą? W jaki sposób odczytać nazwę bieżącego wątku?

```
Thread[] tablicaWatkow = new Thread[10];
```

3. Dodaj do zadania pierwszego lub drugiego obsługę metody `interrupt()`. Po uruchomieniu aplikacji przerwij działanie wątku czwartego. Sprawdź poprawność wyników.

4. Napisz klasę `Watek` dziedziczącą z klasy `Thread`, która będzie miała następujący konstruktor:

```
public Watek(String tekst, int liczbaTabulacji)
```

   gdzie `tekst` to łańcuch tekstowy związany z danym wątkiem, a `liczbaTabulacji` to ilość znaków `\t`, które pojawią się podczas wypisywania danych na ekranie (utwórz od razu w klasie pole `napis` typu `String` przechowujące odpowiednią liczbę tabulacji wraz z tekstem).

   Klasa ma mieć metodę `run()`, która w pętli od 1 do 500:

   - wykona poniższy kod:

```
for(int x=0; x < napis.length(); x++) System.out.print (napis.charAt(x));
System.out.println (numerIteracji);
```

   - wstrzyma swoje działanie na czas pomiędzy 0, a 1 sekundą.

   Utwórz klasę Testową oraz kilka obiektów klasy `Watek`. Uruchom wszystkie wątki. Np.:

```
new Watek("Marek", 1).start();
new Watek("Kasia", 2).start();
new Watek("Andrzej", 3).start();
new Watek("Natalia", 4).start();
```

   Przykładowy początkowy fragment wyniku działania aplikacji:

```
Marek1
Marek2
    Kasia1
        Andrzej1
Marek3
        Andrzej2
            Natalia1
    Kasia2
            Natalia2
    Kasia3
```

```
         Andrzej3
  Marek4
              Natalia3
   .....................
```

Usuń metodę wstrzymującą działanie wątku. Sprawdź poprawność wyświetlanych informacji. Gdzie jest błąd? Co zmieni dodanie metody `yield()` po wyświetleniu każdej iteracji?