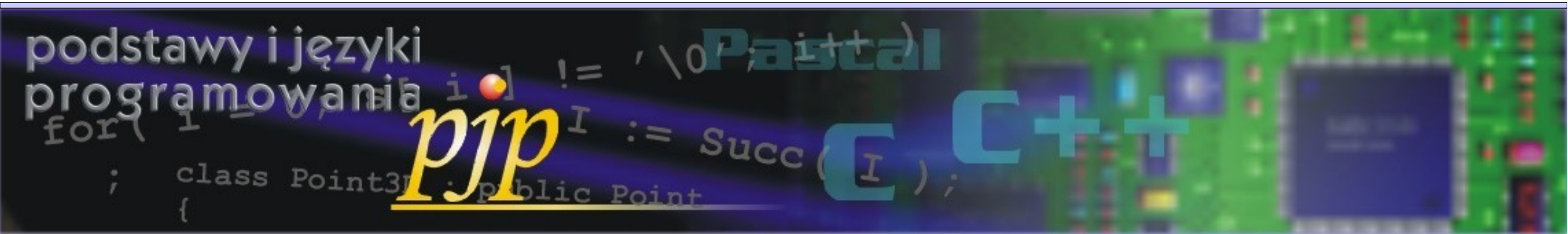
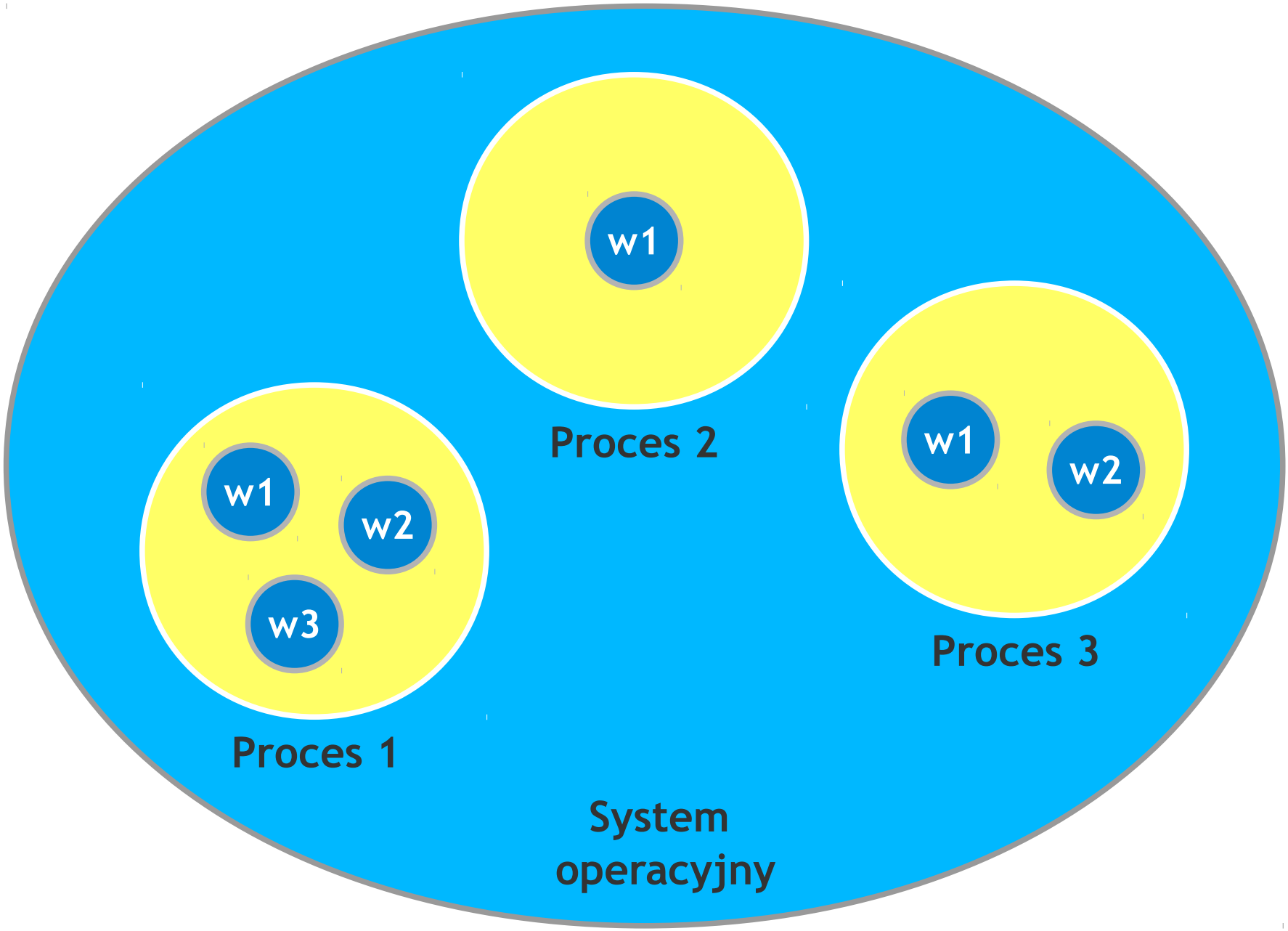
Projektowanie i programowanie obiektowe

Roman Simiński
roman.siminski@us.edu.pl
roman@siminskionline.pl
programowanie.siminskionline.pl

Programowanie wielowątkowe
Wprowadzenie



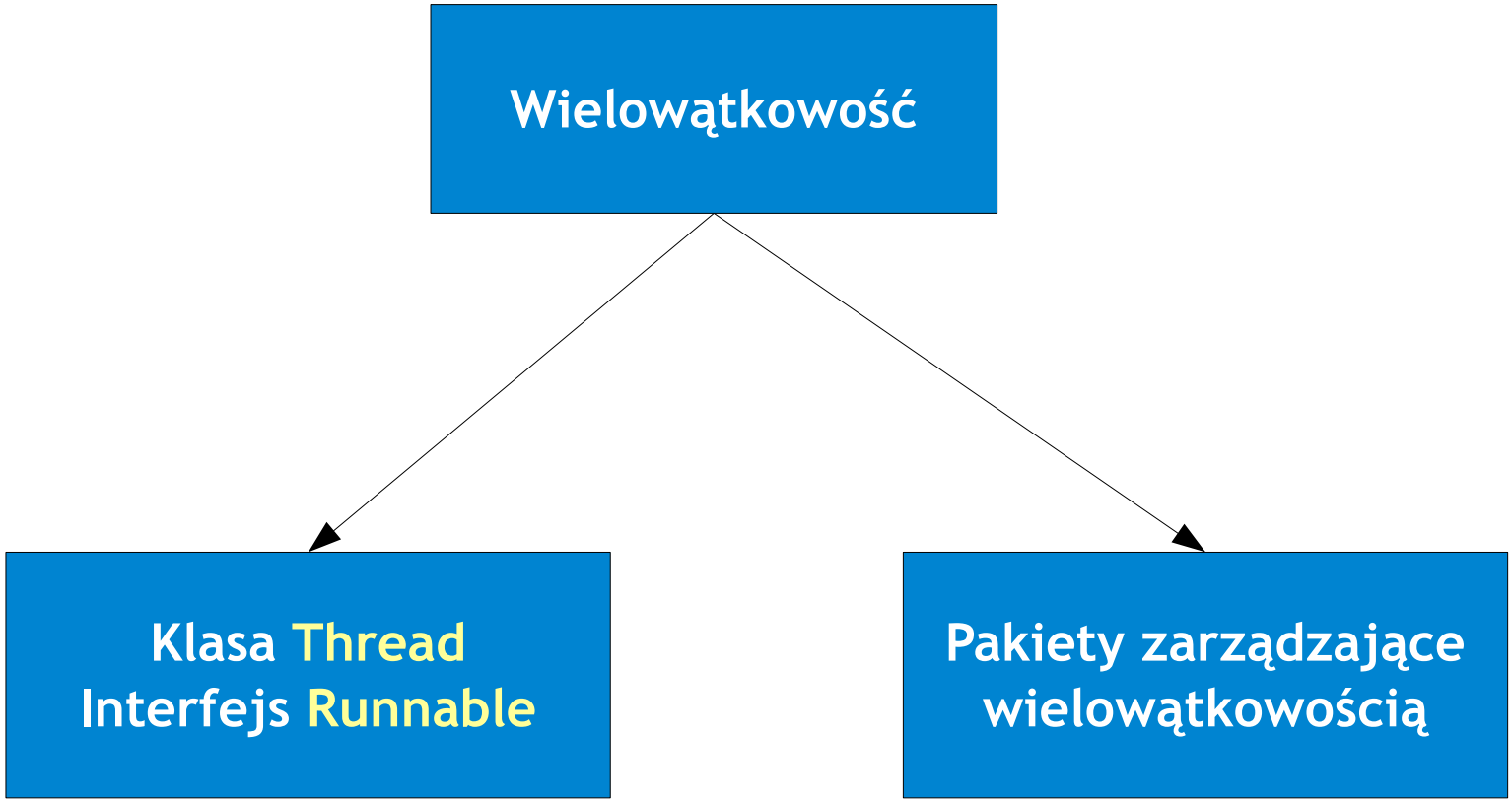
Proces a wątek



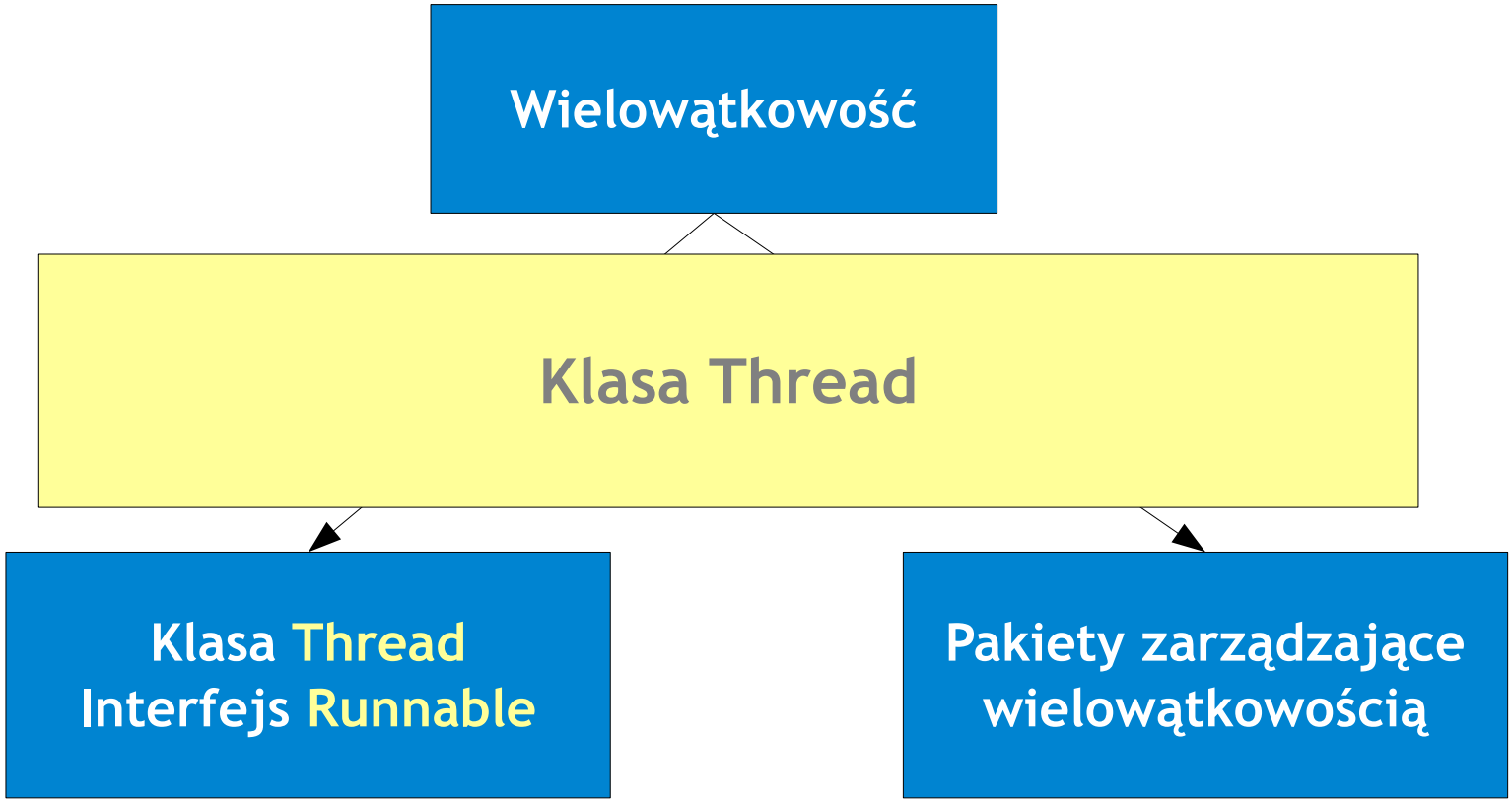
Proces a wątek

- ***Proces*** – wykonujący się program, procesowi są przydzielane zasoby (np. pamięć operacyjna, standardowe strumienie), każdy proces ma własną przestrzeń adresową.
- Systemy wielozadaniowe pozwalają na – teoretycznie – równoległe (jednoczesne) wykonywanie programów (procesów).
- ***Wątek*** - to wyodrębniona sekwencja operacji, która może wykonywać się równolegle z innymi sekwencjami operacji w ramach (programu).
- Wątki w ramach procesu – przynajmniej teoretycznie – wykonują się równolegle.
- Wątki działają w ramach procesu, działające proces posiada przynajmniej jeden, główny wątek.
- Można też powiedzieć, że wątek to *podproces* wydzielony przez *program*, który wykonuje się niezależnie od reszty tego programu.

Podejście do wielowątkowości w języku Java



Podejście do wielowątkowości w języku Java

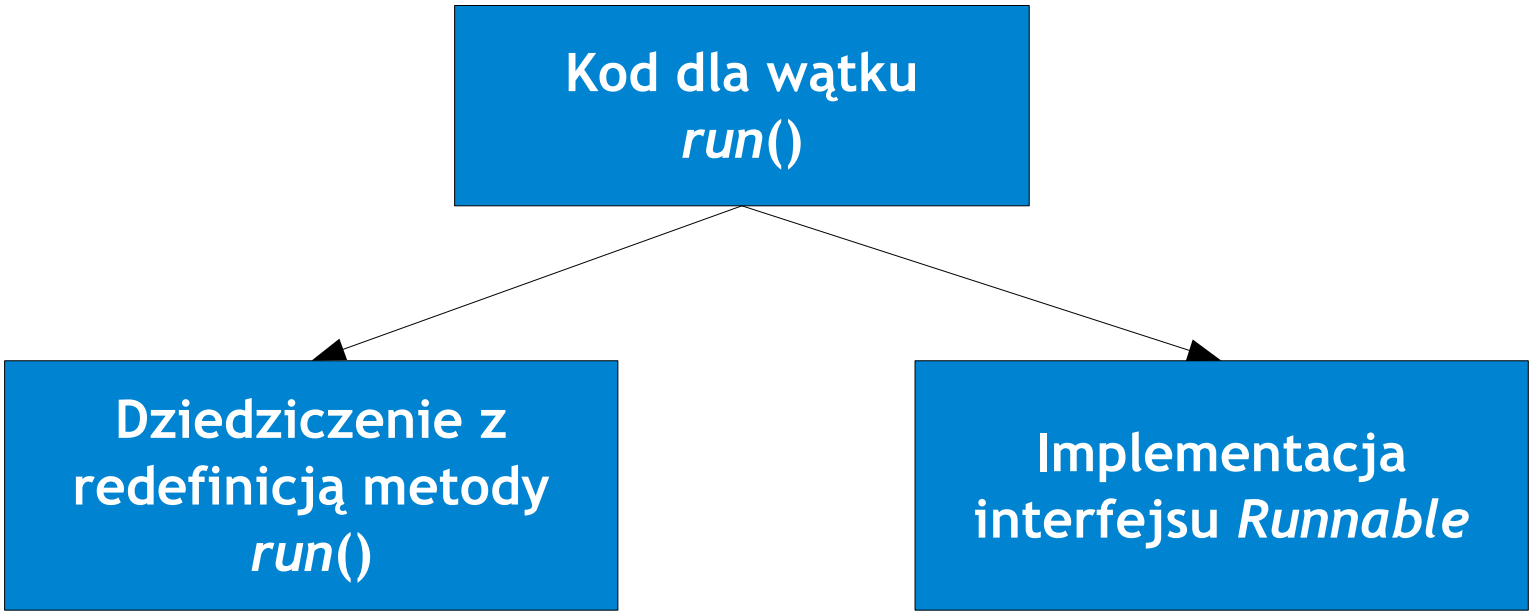


Klasa Thread

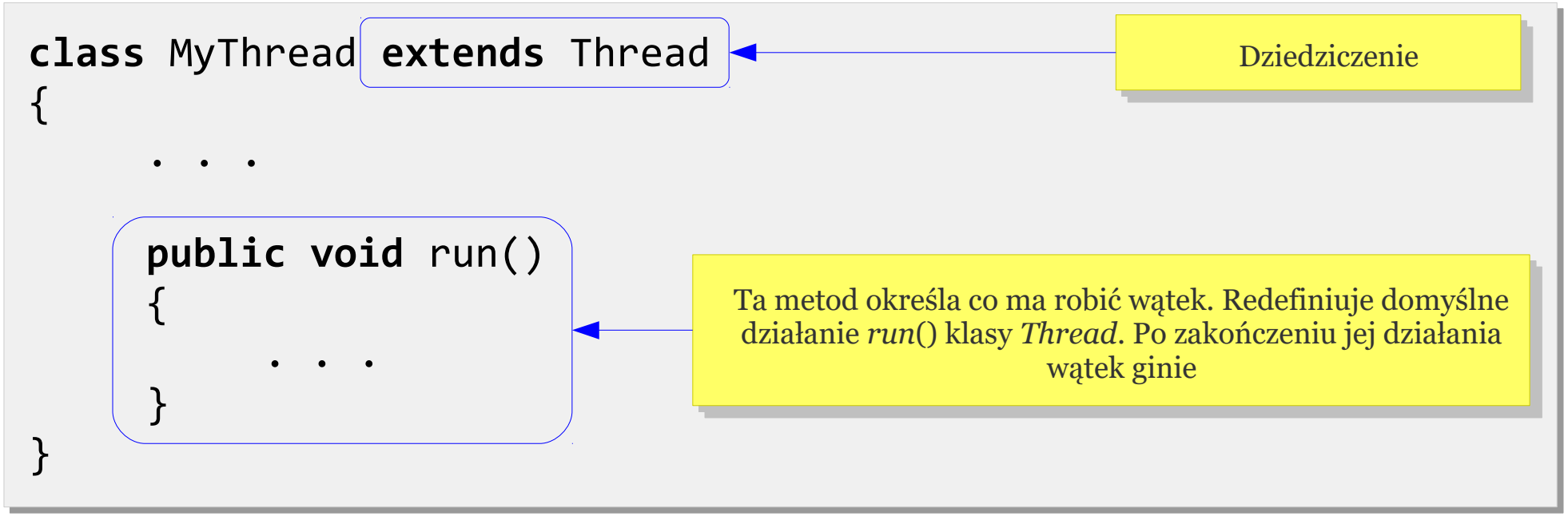
Aby uruchomić wątek należy:

- Utworzyć nowy obiekt klasy *Thread*.
- Uruchomić wątek metoda *start*().

Każdy wątek powinien coś robić, trzeba zdefiniować kod dla wątku – metoda run().



Definiowanie wątku poprzez dziedziczenie z klasy Thread



Utworzenie obiektu wątku i uaktywnienie go

```
class MyThread extends Thread
{
      . . .

      public void run()
      {
            . . .
      }
}
```

```
MyThread t = new MyThread();

t.start();
```

Metoda *start*() uruchamia wykonanie wątku, spowoduje to wywołanie – niekoniecznie natychmiastowe – metody *run*()

Zdefiniowanie interfejsu Runnable

```
class MyThreadRun implements Runnable
{
      . . .

      public void run()
      {
            . . .
      }
}
```

Tworzymy interfejs określający to, co ma robić przyszły wątek – metoda *run*()

Ta metod określa co będzie robić wątek

Utworzenie wątku i wykorzystanie interfejsu

```
class MyThreadRun implements Runnable
{
      . . .

      public void run()
      {
            . . .
      }
}
```

Tworzymy interfejs określający to, co ma robić przyszły wątek – metoda *run*()

Ta metod określa co będzie robić wątek

```
MyThreadRun r = new MyThreadRun();
Thread t = new Thread( r );

t.start();
```

Od tego momentu działanie wątku *t* określa metoda *run*() interfejsu *r*

Dziedziczenie z Thread vs implementacja Runnable

```
MyThread t = new MyThread();

t.start();
```

```
MyThreadRun r = new MyThreadRun();
Thread t = new Thread( r );

t.start();
```

- W języku Java nie występuje dziedziczenie *wielobazowe*. Tworzenie wątku poprzez zdefiniowanie klasy pochodnej od *Thread* blokuje możliwość dziedziczenie po innej klasie.
- Implementacja interfejsu *Runnable* nie blokuje takiej możliwości. Podejście to jest również elastyczniejsze i bardziej rozszerzalne.

Uwaga – nie wywołujemy jawnie metody *run*() obiektu wątku, nie wywołujemy jawnie metody run() implementowanego interfejsu.

Używamy metody *start*() aby uaktywnić wątek.

Wątki można nazywać oraz pobierać ich nazwę

```
class MyThread extends Thread
{
     MyThread( String name )
     {
        super( name );
     }
     public void run()
     {
        System.out.printf( \nWątek \"%s\", getName() );
        . . .
     }
}

. . .
MyThread t = new MyThread( "Mój wątek" );

t.start();
```

```
Wątek "Mój wątek"
```

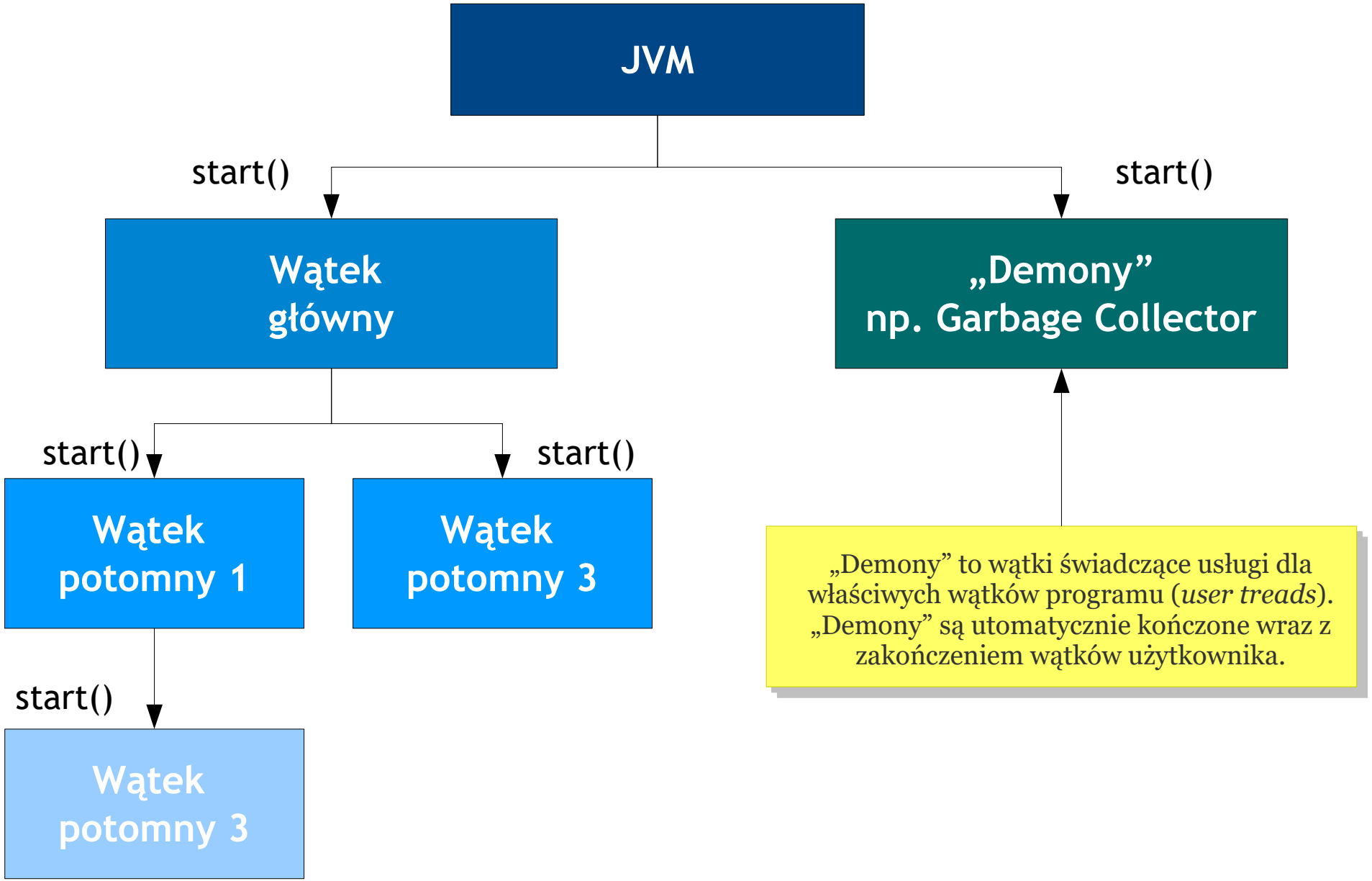
```
MyThreadRun r = new MyThreadRun( r, "Mój wątek" );
Thread t = new Thread( r );

t.start();
System.out.printf( "\nWątek \"%s\", t.getName() );
```

```
Wątek "Mój wątek"
```

Kto zarządza uruchamianiem wątków, kto wywołuje metodę *run*()?

- ***Thread scheduler*** – jest częścią JVM, to on decyduje który wątek powinien być uruchomiony.
- W obrębie pojedynczego procesu tylko jeden wątek może być aktywny.
- Nie należy zakładać *niczego* odnośnie kolejności aktywowania i wykonywania wątków.
- Zarządzanie wielowątkowością może być realizowane z wykorzystaniem dwóch strategii:
  - *preemptive scheduling* – wątki o najwyższym priorytecie są wykonywane aż do momentu ich zawieszenia lub zakończenia lub pojawienia się wątku o wyższym priorytecie.
  - *time slicing* – wątek wykonuje się w ciągu predefiniowanego wycinka czasu, po jego zakończeniu wraca do puli aktywnych wątków i czeka na powtórny przydział okna czasowego. Planista określa kolejny wątek do uaktywnienia.

Co wiadomo o kolejności wykonania wątków?



Kilka wątków i wątek główny

```
class MyThread extends Thread
{
    MyThread( String name )
    {
        super( name );
    }

    public void run()
    {
        System.out.printf( "\nWątek: %s działa...", getName() );
    }
}
. . .
public static void main( String[] args )
{
    System.out.printf( "\nWątek główny: %s",
                       Thread.currentThread().getName() );

    Thread [] threads = new Thread[ 10 ];

    for(int i = 0; i < 10; i++)
        threads[ i ]  = new MyThread( "" + i );

    for(int i = 0; i < 10; i++)
        threads[ i ].start();
}
```

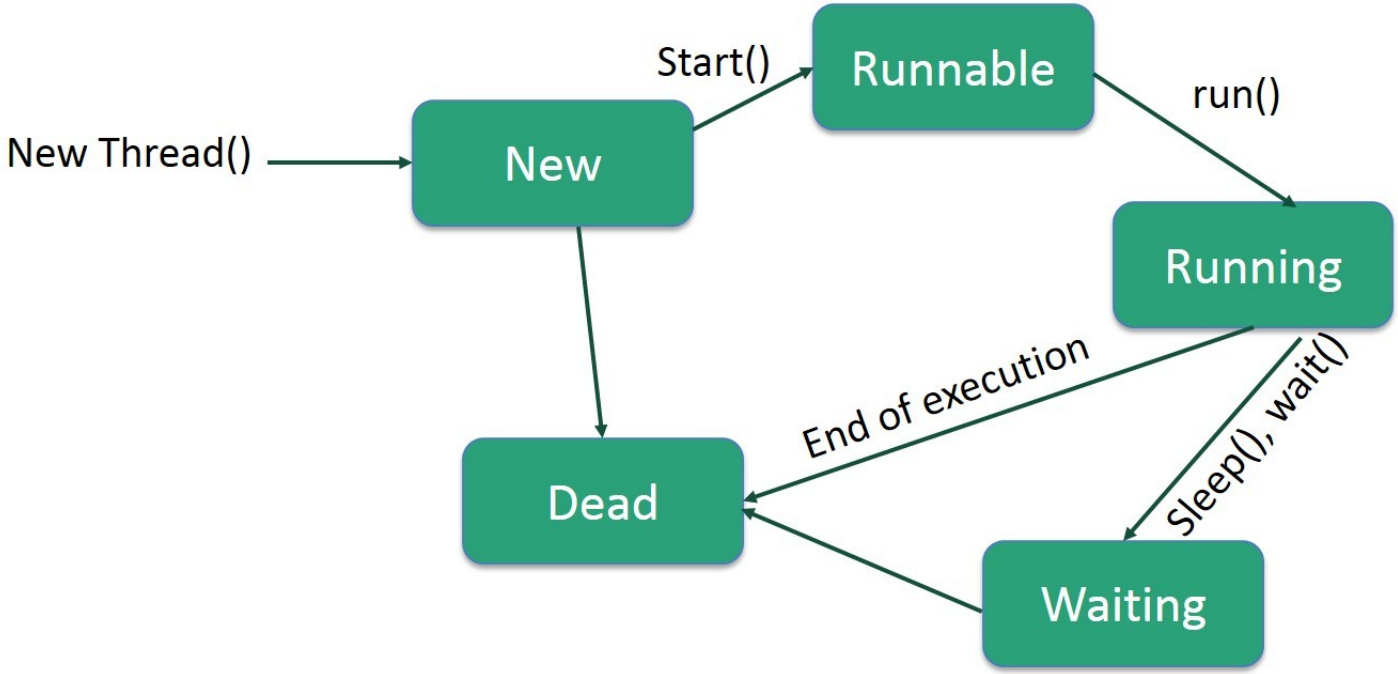
```
Wątek główny: main
Wątek: 0 działa...
Wątek: 1 działa...
Wątek: 2 działa...
Wątek: 3 działa...
Wątek: 4 działa...
Wątek: 5 działa...
Wątek: 6 działa...
Wątek: 7 działa...
Wątek: 8 działa...
Wątek: 9 działa...
```

Ciekawostka, anonimowe wątki

```
public static void main( String[] args )
{
  System.out.printf( "\nWątek główny: %s", Thread.currentThread().getName() );

  for( int i = 0; i < 10; i++ )
  {
    new Thread( "" + i )
    {
      public void run()
      {
          System.out.printf( "\nWątek: %s działa...", getName() );
      }
    }.start();
  }
}
```

```
Wątek główny: main
Wątek: 0 działa...
Wątek: 1 działa...
Wątek: 2 działa...
Wątek: 3 działa...
Wątek: 4 działa...
Wątek: 5 działa...
Wątek: 6 działa...
Wątek: 7 działa...
Wątek: 8 działa...
Wątek: 9 działa...
```

Cykl życia wątku



- *New* – wątek „urodzony”, zacznie działać, gdy program go „wystartuje”.
- *Runnable* – wątek „wystartowany”, wykonuje swoje działania.
- *Waiting/Blocked* – wątek w stanie oczekiwania, np. do odblokowania (synchronizacja) albo do momentu uaktywnienia go przez inny wątek.
- *Timed Waiting* – czasowe „uśpienie”, zawieszenie wykonania wątku.
- *Terminated* – wątek zakończył swoje działanie.

Cykl życia wątku, przykład

```
class MyThread1 extends Thread {
    int counter = 0;
    final int MAX_COUNT = 10;

    MyThread1( String name ) { super( name ); }

    @Override
    public void run() {
        System.out.printf( "\nWątek: %s uruchomiony", getName() );

        for( counter = 0; counter < MAX_COUNT; ++counter ) {
            try {
                switch( counter ) {
                    case 3 : Thread.sleep( 1000L );
                             break;

                }
                System.out.printf( "\nWątek: %s, iteracja: %d stan: %s", getName(),
                                   counter, getState() );
            }
            catch( InterruptedException e ) {}
        }
    }
}
```

Cykl życia wątku

```
public class ThreadExample02
{
    public static void main(String args[])
    {
        MyThread1 thread = new MyThread1( "Mój wątek" );

        System.out.printf( "\nWątek: %s, stan: %s", thread.getName(),
                           thread.getState() );
        thread.start();

        System.out.printf( "\nWątek: %s, stan: %s", thread.getName(),
                           thread.getState() );

        while( thread.isAlive() )
        {
            System.out.printf( "\nWątek: %s, stan: %s", thread.getName(),
                               thread.getState() );
            if( thread.counter >= thread.MAX_COUNT - 3 )
                System.out.printf( "\nWątek: za chwilę się skończy" );
        }

        System.out.printf( "\nWątek: %s, stan: %s\n", thread.getName(),
                           thread.getState() );
    }
}
```

Cykl życia wątku

```
Wątek: Mój wątek, stan: NEW
Wątek: Mój wątek, stan: RUNNABLE
[...]
Wątek: Mój wątek, stan: RUNNABLE
Wątek: Mój wątek uruchomiony
Wątek: Mój wątek, stan: BLOCKED
Wątek: Mój wątek, stan: RUNNABLE
Wątek: Mój wątek, stan: RUNNABLE
Wątek: Mój wątek, iteracja: 0 stan: RUNNABLE
Wątek: Mój wątek, iteracja: 1 stan: RUNNABLE
Wątek: Mój wątek, iteracja: 2 stan: RUNNABLE
Wątek: Mój wątek, stan: RUNNABLE
Wątek: Mój wątek, stan: TIMED_WAITING
[...]
Wątek: Mój wątek, stan: TIMED_WAITING
Wątek: Mój wątek, stan: BLOCKED
[...]
Wątek: Mój wątek, stan: BLOCKED
Wątek: Mój wątek, iteracja: 3 stan: RUNNABLE
Wątek: Mój wątek, stan: BLOCKED
Wątek: Mój wątek, iteracja: 4 stan: RUNNABLE
Wątek: Mój wątek, stan: RUNNABLE
Wątek: Mój wątek, iteracja: 5 stan: RUNNABLE
Wątek: Mój wątek, stan: RUNNABLE
Wątek: Mój wątek, iteracja: 6 stan: RUNNABLE
Wątek: Mój wątek, stan: RUNNABLE
Wątek: Mój wątek, iteracja: 7 stan: RUNNABLE
Wątek: za chwilę się skończy
Wątek: Mój wątek, iteracja: 8 stan: RUNNABLE
Wątek: Mój wątek, stan: BLOCKED
Wątek: za chwilę się skończy
Wątek: Mój wątek, stan: BLOCKED
Wątek: za chwilę się skończy
Wątek: Mój wątek, iteracja: 9 stan: RUNNABLE
Wątek: Mój wątek, stan: BLOCKED
Wątek: za chwilę się skończy
Wątek: Mój wątek, stan: TERMINATED
```