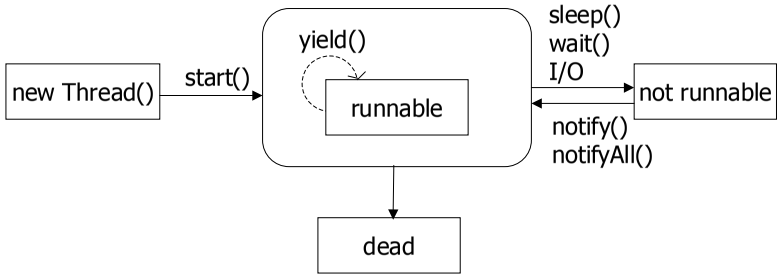
Wątki

Wątki

- *Lekkie* procesy
- Implementowane w JVM (w miarę możliwości) jako wątki systemowe
- Java dostarcza mechanizmów do:
  - zarządzania cyklem życia wątków (uruchamianie, zatrzymywanie itp.)
  - synchronizacji operacji na współdzielonych danych
  - grupowania wątków
  - sterowania priorytetem
- Kod wątku uruchamia się za pomocą metody **`public void run()`** klasy dziedziczącej z klasy **`Thread`** lub implementującej interfejs **`Runnable`**

Cykl życia wątku



Uruchomienia wątków - przykład

```
public class SimpleThread extends Thread {
    public SimpleThread(String str) {super(str);}
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                Thread.sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
    public static void main (String[] args) {
        new SimpleThread("Java").start();
        new SimpleThread("C++").start();
    }
}
```

Oczekiwanie na zakończenie wątków - przykład

```
public class SimpleThread extends Thread {
    ...
   public static void main (String[] args) {
        Thread t1 = new SimpleThread("Java");
        t1.start();
        t1.join();
        new SimpleThread("C++").start();
   }
}
```


Przerywanie wątków - przykład 1

```
public class SimpleThread extends Thread {
     static Thread t1;
    public SimpleThread(String str) {super(str);}
    public void run() {
        for (int i = 0; i < 10 & this==t1; i++) {
            System.out.println(i + " " + getName());
            try {
                Thread.sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
   public static void main (String[] args) {
       t1 = new SimpleThread("Java");
       t1.start();
       t1=null;
   }
}
```


Przerywanie wątków - przykład 2

```
public class SimpleThread extends Thread {
 public SimpleThread(String str) {super(str);}
  public void run() {
       for (int i = 0; i < 10 & ! isInterrupted(); i++) {
           System.out.println(i + " " + getName());
           try {
               Thread.sleep((long)(Math.random() * 1000));
           } catch (InterruptedException e) {
                    interrupt();
           }
       }
       System.out.println("DONE! " + getName());
  }
   public static void main (String[] args) throws Exception{
       Thread t1 = new SimpleThread("Java");
       t1.start();
       t1.interrupt();
       Thread.sleep(1000);
       System.out.println("Is t1 alive: " + t1.isAlive());
   }
}
```


Priorytety

- Umożliwiają sterowanie przydziałem zasobów systemu (czas procesora) poszczególnym wątkom
- Przydział dokonywany za pomocą *fixed priority scheduling*, czyli w danym momencie czasu system rozważa wątki z najwyższą wartością priorytetu, innym wątkom może być przydzielony czas procesora tylko jeżeli wątki o wyższym priorytecie są w stanie *not runnable*
- Ustawiane w zakresie Thread.MIN_PRIORITY i Thread.MAX_PRIORITY (normalny: Thread.NORM_PRIORITY)

Priorytety

```
public class SimpleJob implements Runnable{
    public void run() {
        int tick=0;
        while(tick<10000000) tick++;
        System.out.println("DONE! " +
              Thread.currentThread().getName());
    }
    public static void main (String[] args) {
        Thread t1 = new Thread(new SimpleJob(), "Java");
        t1.start();
        t1.setPriority(Thread.MAX_PRIORITY);
        Thread t2 = new Thread(new SimpleJob(), "C++");
        t2.start();
    }
}
```


Grupy wątków

- Hierarchiczne zarządzanie wątkami
- Przydzielenia wątku do grupy odbywa się za pomocą odpowiedniego konstruktora *Thread()*
- W predefiniowanej grupie *main* uruchamiany jest wątek metody *static public void main(String[] s)*
- Wszystkie wątki dla których nie zdefiniowano grupy są przydzielane do grupy *main*
- Istnieje możliwość zdefiniowania polityki bezpieczeństwa dotyczącej manipulacji wątkami przez inne wątki (SecurityManager)

Grupy wątków - przykład

```
public class SimpleThread extends Thread {
 public SimpleThread (ThreadGroup group, String name){
       super(group, name);
 }
 ...
 public static void main (String[] args) throws Exception{
   ThreadGroup mainGroup =
       Thread.currentThread().getThreadGroup();
   ThreadGroup javaGroup=new ThreadGroup(mainGroup,
                                         "JavaGroup");
    ThreadGroup cppGroup=new ThreadGroup(mainGroup, "C++Group");

   for(int i=0; i<10; i++){
       new SimpleThread(javaGroup, "Java"+i).start();
       new SimpleThread(cppGroup, "Java"+i).start();
   }
   cppGroup.interrupt();
 }
}
```


Synchronizacja wątków

- Fragment kodu, który odwołuje się do tego samego obiektu z równoległych wątków nazywa się sekcją krytyczną
- Java umożliwia wskazanie sekcji krytycznej (metody lub bloku) za pomocą słowa kluczowego *synchronized*
- Rozpoczęcie wykonywania sekcji krytycznej oznacza zablokowanie obiektu
- Metody dziedziczone z klasy Object: *wait*, *notify* i *notifyAll* umożliwiają implementację nieaktywanego oczekiwania na zdjęcie blokady

Synchronizacja wątków: przykład (Producent)

```
public class Producer extends Thread {
    private CubbyHole cubbyhole;
     public Producer(CubbyHole c) {
         cubbyhole = c;
    }

    public void run() {
         for (int i = 0; i < 10; i++) {
             cubbyhole.put(i);
             System.out.println("Producer put: " + i);
             try {
                 sleep((int)(Math.random() * 100));
             } catch (InterruptedException e) { }
         }
    }
}
```


Synchronizacja wątków: przykład (Konsument)

```
public class Consumer extends Thread {
    private CubbyHole cubbyhole;

    public Consumer(CubbyHole c) {
         cubbyhole = c;
    }

    public void run() {
         int value = 0;
         for (int i = 0; i < 10; i++) {
             value = cubbyhole.get();
             System.out.println("Consumer got: " + value);
         }
    }
}
```


Synchronizacja wątków: przykład (CubbyHole)

```
public class CubbyHole {
    private int contents;
    private boolean available = false;
    public synchronized int get() {
        while (available == false) {
            try {wait();
            } catch (InterruptedException e) { }
        }
        available = false;
        notify();
        return contents;
    }
    public synchronized void put(int value) {
        while (available == true) {
            try {wait();
            } catch (InterruptedException e) { }
        }
        contents = value;
        available = true;
        notify();
    }
}
```


Synchronizacja wątków: przykład (main)

```
public class ProducerConsumerTest {
    public static void main(String[] args) {
        CubbyHole c = new CubbyHole();
        Producer p1 = new Producer(c);
        Consumer c1 = new Consumer(c);

        p1.start();
        c1.start();
    }
}
```