# Programowanie współbieżne i równoległe

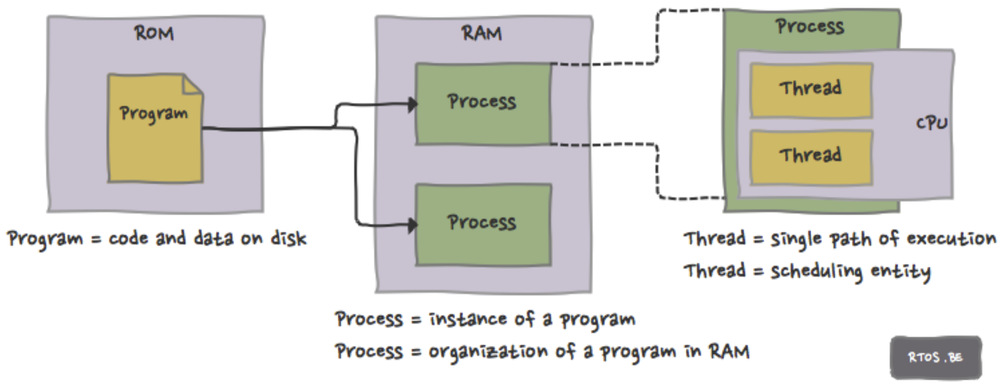
## Wątki, zadania, synchronizacja, monitory, wartości atomowe, kolekcje

dr inż. Aleksander Smywiński-Pohl, Michał Idzik

Część przykładów pochodzi ze strony: http://winterbe.com

## Java + Threads

## Program, proces, wątek


ROM
Program
RAM
Process
Process
Process
Thread
Thread
CPU
Program = code and data on disk
Thread = single path of execution
Thread = scheduling entity
Process = instance of a program
Process = organization of a program in RAM
RTOS.BE

- program - plik lub zestaw plików opisujących w jakis sposób należy przetwarzać dane
- proces - uruchomiony program posiadający własną pamięć oraz licznik instrukcji

- wątek - lekki proces w obrębie działającego programu, posiadający własny stos oraz licznik instrukcji

## Klasa `Thread`

API klasy `Thread`:

- `run()`
- `start()`
- `join()`
- `currentThread()`
- `getName()`
- ...

In [1]:
```
import static java.lang.System.out;
import java.util.List;
import java.util.LinkedList;

List<Thread> threads = new LinkedList<>();

out.println("Wątek główny " + Thread.currentThread().getName());

threads.add(new Thread(() -> out.println("Wątek " + Thread.currentThread().getNa
threads.add(new Thread(() -> out.println("Wątek " + Thread.currentThread().getNa
threads.add(new Thread(() -> out.println("Wątek " + Thread.currentThread().getNa

threads.forEach(Thread::start);
for(Thread thread : threads){
    thread.join();
}
```

```
Wątek główny IJava-executor-0
Wątek Thread-3
Wątek Thread-1
Wątek Thread-2
```

## Interfejs `Runnable`

API interfejsu `Runnable`:

- `run()`

In [3]:
```
class Task implements Runnable {
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }
}

Task task = new Task();
Thread thread = new Thread(task);
thread.start();
// thread.join();
```

```
Thread-5
```

Alternatywnie: **klasa anonimowa**

In [ ]:
```
Runnable task = new Runnable() {
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }
}
```

...co można też zapisać jako **wyrażenie lambda**:

In [ ]:
```
Runnable task = () -> System.out.println(Thread.currentThread().getName());
```

In [6]:
```
import static java.lang.System.out;
import java.util.List;
import java.util.LinkedList;
```

```
List<Runnable> tasks = new LinkedList<>();

tasks.add(() -> out.println("Zadanie " + Thread.currentThread().getName()));
tasks.add(() -> out.println("Zadanie " + Thread.currentThread().getName()));
tasks.add(() -> out.println("Zadanie " + Thread.currentThread().getName()));

List<Thread> threads = new LinkedList<>();
tasks.forEach((task) -> threads.add(new Thread(task)));

threads.forEach(Thread::start);
for(Thread t : threads) {
    t.join();
}
```

```
Zadanie Thread-12
Zadanie Thread-14
Zadanie Thread-13
```

## `Thread#sleep()`

In [8]:
```
import static java.lang.System.out;
import java.util.concurrent.*;

Thread sleepingThread = new Thread(() -> {
    try{
        out.println("Idę spać na 3 sekudny");
        TimeUnit.SECONDS.sleep(3);
        out.println("Godzinę później...");
    } catch (InterruptedException ex) {
        out.println("Sen został przerwany");
    }
});
```

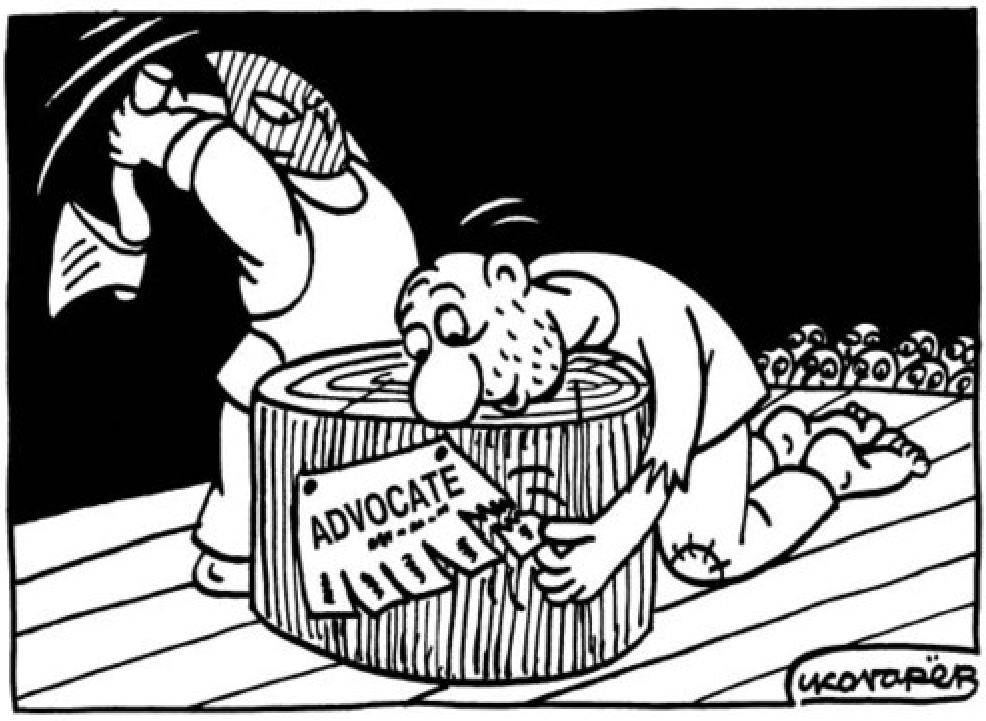
```
sleepingThread.start();
sleepingThread.join();
```

```
Idę spać na 3 sekudny
Godzinę później...
```

In [ ]:

```
```

## ExecutorService



In [10]:

```
import java.util.concurrent.*;
import static java.lang.System.out;

System.out.println(Thread.currentThread());
ExecutorService executor =  Executors.newSingleThreadExecutor();
executor.submit(() -> out.println("Egzekucja w " + Thread.currentThread().getNam
executor.submit(() -> out.println("Egzekucja w " + Thread.currentThread().getNam
executor.submit(() -> out.println("Egzekucja w " + Thread.currentThread().getNam
executor.shutdown();
executor.awaitTermination(1, TimeUnit.SECONDS);
```

```
Thread[IJava-executor-4,5,main]
Egzekucja w pool-2-thread-1
Egzekucja w pool-2-thread-1
Egzekucja w pool-2-thread-1
```

Out[10]:

```
true
```

## Future

In [ ]:
```
interface Future<V> {
    boolean cancel(boolean mayInterruptIfRunning);
    V get();
    V get(long timeout, TimeUnit unit);
    boolean isCancelled();
    boolean isDone();
}
```

In [12]:
```
import java.util.concurrent.*;
final int sleepTime = 6;

Callable<Integer> task = () -> {
    try {
        TimeUnit.SECONDS.sleep(sleepTime);
        return 123;
    } catch (InterruptedException e) {
        throw new IllegalStateException("wątek został przerwany", e);
    }
};

ExecutorService executor = Executors.newFixedThreadPool(1);
Future<Integer> future = executor.submit(task);

out.println("obliczenie zakończone? " + future.isDone());

Integer result = future.get();

out.println("obliczenie zakończone? " + future.isDone());
out.print("wynik: " + result);
```

```
obliczenie zakończone? false
obliczenie zakończone? true
wynik: 123
```

## Rodzaje wykonawców (`ExecutorService`)

- `newCachedThreadPool` - tworzy wątki w zależności od potrzeb i usuwa je jeśli nie są używane przez 60 sekund
- `newFixedThreadPool` - cały czas przechowuje niezakończone wątki
- `newScheduledThreadPool` - posiada możliwość odroczonego i periodycznego wykonania wątków
- `newSingleThreadExecutor` - wykonanie jednowątkowe
- `newSingleThreadScheduledExecutor` - jw. ale z możliwością odroczonego i periodycznego wykonania

## `ScheduledExecutor`

In [13]:
```java
import java.util.concurrent.*;
import static java.lang.System.out;
ScheduledExecutorService executor = Executors.newScheduledThreadPool(2);

Runnable task = () -> out.println("Wykonanie odroczonego zadania w " + Thread.cu
ScheduledFuture<?> future = executor.schedule(task, 3, TimeUnit.SECONDS);

out.println("Przed oczekiwaniem");

TimeUnit.MILLISECONDS.sleep(1000);

out.println("Czas pozostały do wykonania " + future.getDelay(TimeUnit.MILLISECON
TimeUnit.SECONDS.sleep(3);
executor.shutdown();
executor.awaitTermination(1, TimeUnit.SECONDS);
```

```
Przed oczekiwaniem
Czas pozostały do wykonania 1919
Wykonanie odroczonego zadania w pool-5-thread-1
```

Out[13]:
```
true
```

## Hazard (Race condition)

In [14]:
```java
class RaceCondition {
    private int counter = 0;

    public void increment() {
        this.counter = this.counter + 1;
    }

    public int getCounter(){
        return this.counter;
    }
}
```

In [15]:
```java
import java.util.concurrent.*;
import java.util.stream.*;

RaceCondition object = new RaceCondition();

IntStream.range(0, 1000000).forEach(i -> object.increment());

System.out.println(object.getCounter());
```

```
1000000
```

In [16]:

```
import java.util.concurrent.*;
import java.util.stream.*;

ExecutorService executor = Executors.newFixedThreadPool(2);

RaceCondition object = new RaceCondition();

IntStream.range(0, 1000000).forEach(i -> executor.submit(object::increment));

executor.shutdown();
executor.awaitTermination(1, TimeUnit.SECONDS);
```

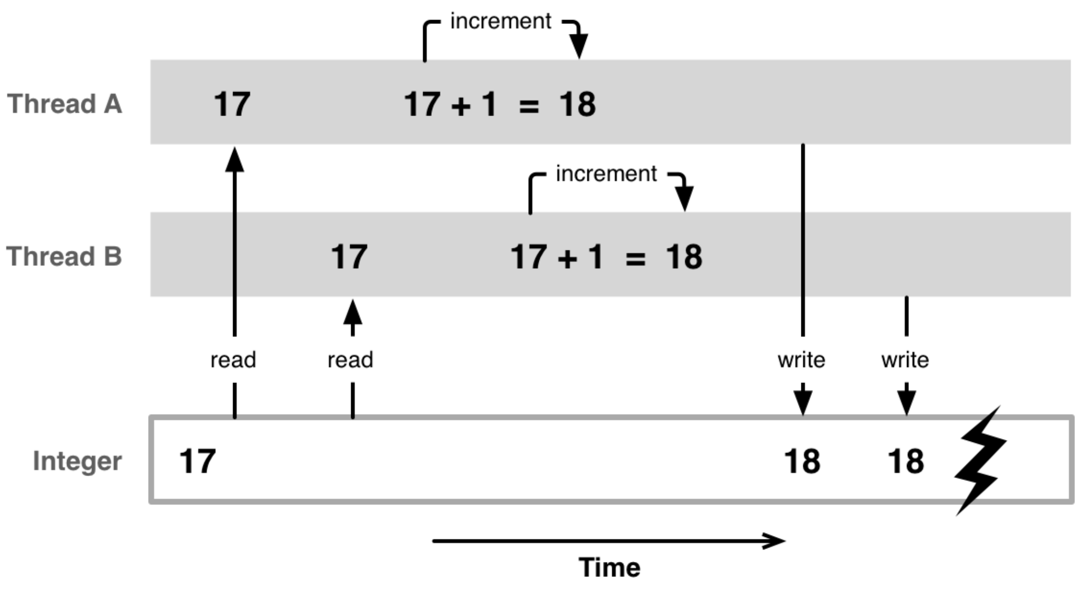
Out[16]:

```
true
```

In [17]:

```
System.out.println(object.getCounter());
```

```
996915
```



## Sekcja krytyczna (`synchronized`)

In [18]:
```java
class SynchronizedAccessors {
    private int counter = 0;

    public synchronized void increment() {
        this.counter = this.counter + 1;
    }

    public synchronized int getCounter(){
        return this.counter;
    }
}
```

In [19]:
```java
import java.util.concurrent.*;
import java.util.stream.*;

ExecutorService executor = Executors.newFixedThreadPool(2);

SynchronizedAccessors object = new SynchronizedAccessors();

IntStream.range(0, 1000000).forEach(i -> executor.submit(object::increment));

executor.shutdown();
executor.awaitTermination(1, TimeUnit.SECONDS);

System.out.println(object.getCounter());
```

```
1000000
```

In [ ]:
```java
class Monitor {
    private int counter = 0;

    public void increment() {
        synchronized(this) {
            this.counter += 1;
        }
    }
    public int getCounter(){
        synchronized(this) {
            return this.counter;
        }
    }
}
```

## `wait` i `notify`

In [20]:
```
import java.util.*;

class StringStack {
    private List<String> stack = new LinkedList<>();

    public void push(String value){
        synchronized(this) {
            stack.add(value);
            notify();
        }
    }

    public String pop(){
        synchronized(this) {
            while(stack.isEmpty()){
                try {
                    wait();
                } catch (InterruptedException ex) {
                    out.println("Wątek został przerwany");
                }
            }
            return stack.remove(stack.size() - 1);
        }
    }
}
```

In [21]:
```
StringStack stack = new StringStack();

Thread ideaProducer = new Thread(() -> {
    try {
        out.println("Filozof: Myślę");
        Thread.sleep(3000);
        out.println("Filozof: Produkuję myśl");
        stack.push("Myślę więc jestem");
        Thread.sleep(3000);
        out.println("Filozof: Produkuję myśl");
        stack.push("Różowe idee wściekle śpią");
    } catch (InterruptedException ex) {
        out.println("Wątek dodający został przerwany");
    }
});

Thread ideaConsumer = new Thread(() -> {
    out.println("Konsument: Czekam na jakąś mądrą myśl...");
    out.println("Konsument: Konsumuję myśl: " + stack.pop());
    out.println("Konsument: Konsumuję myśl: " + stack.pop());
});
```

```
ExecutorService executor = Executors.newFixedThreadPool(2);
executor.submit(ideaConsumer);
executor.submit(ideaProducer);
executor.shutdown();
executor.awaitTermination(7, TimeUnit.SECONDS);
```

```
Konsument: Czekam na jakąś mądrą myśl...
Filozof: Myślę
Filozof: Produkuję myśl
Konsument: Konsumuję myśl: Myślę więc jestem
Filozof: Produkuję myśl
Konsument: Konsumuję myśl: Różowe idee wściekle śpią
```

Out[21]:

```
true
```

## Klasa `ReentrantLock`

In [ ]:

```
import java.util.concurrent.locks.*;

class ReentrantLockAccessors {
    private int counter = 0;
    private Lock lock = new ReentrantLock();

    public void increment() {
        lock.lock();
        try {
            this.counter += 1;
        } finally {
            lock.unlock();
        }
    }

    public int getCounter(){
        lock.lock();
        try {
            return this.counter;
        } finally {
            lock.unlock();
        }
    }
}
```

## `AtomicInteger`

In [22]:
```
import java.util.stream.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.*;

AtomicInteger atomicInt = new AtomicInteger(0);

ExecutorService executor = Executors.newFixedThreadPool(2);

IntStream.range(0, 1000000).forEach(i -> executor.submit(atomicInt::incrementAnc

executor.shutdown();
executor.awaitTermination(1, TimeUnit.SECONDS);
System.out.println(atomicInt.get());
```

```
1000000
```

In [25]:
```
AtomicInteger atomicInt = new AtomicInteger(0);

ExecutorService executor = Executors.newFixedThreadPool(10);

IntStream.range(0, 10000).forEach(i -> {
    executor.submit(() -> atomicInt.updateAndGet(n -> n + i));
});

executor.shutdown();
executor.awaitTermination(1, TimeUnit.SECONDS);

System.out.println(atomicInt.get());
```

```
49995000
```

## `AtomicInteger`

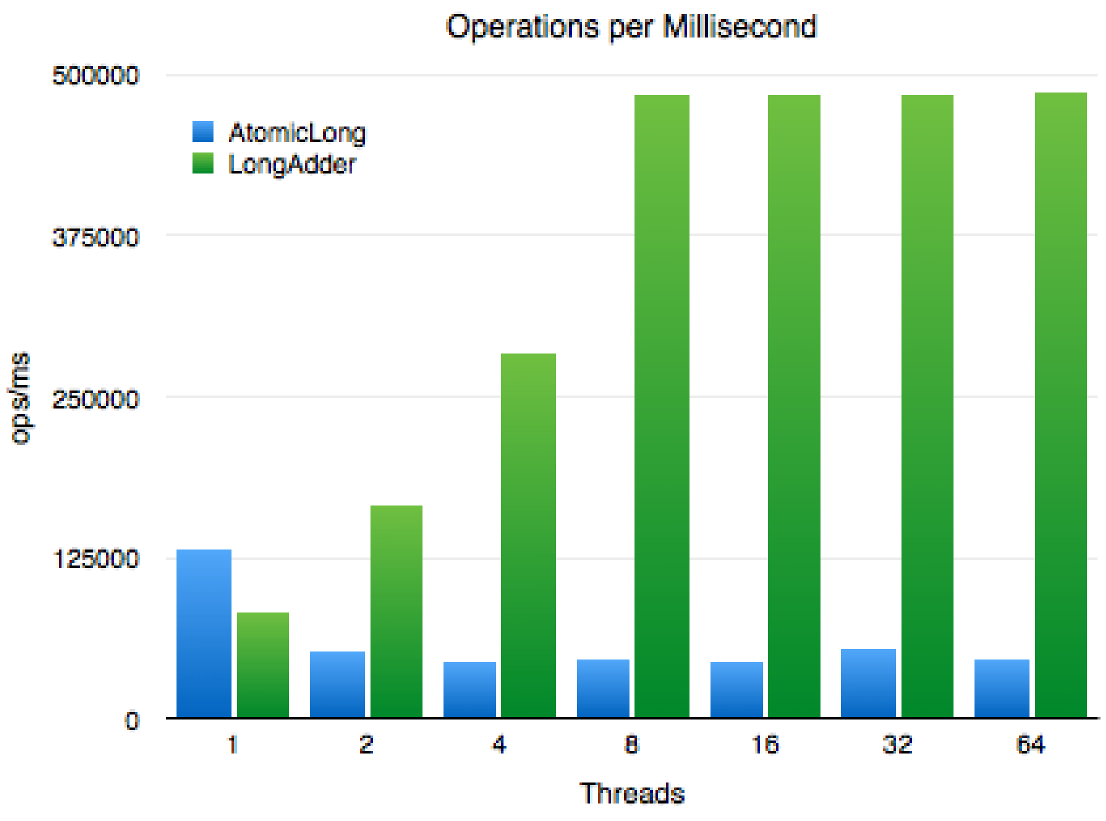
- `addAndGet`
- `compareAndSet`

- `decrementAndGet`
- `get`
- `getAndAdd`
- `getAndDecrement`
- `getAndIncrement`
- `getAndSet`
- ...

## `LongAdder`

In [26]:
```
import java.util.stream.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.*;

LongAdder adder = new LongAdder();
ExecutorService executor = Executors.newFixedThreadPool(4);

IntStream.range(0, 1000000).forEach(i -> executor.submit(adder::increment));

executor.shutdown();
executor.awaitTermination(1, TimeUnit.SECONDS);

System.out.println(adder.sumThenReset());
```

```
1000000
```



## `LongAccumulator`

In [27]:
```
import java.util.stream.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.*;
import java.util.function.*;

LongBinaryOperator operation = (x, y) -> x + y;
LongAccumulator accumulator = new LongAccumulator(operation, 0);

ExecutorService executor = Executors.newFixedThreadPool(4);

IntStream.range(0, 1000000).forEach(i -> executor.submit(() -> accumulator.accum

executor.shutdown();
executor.awaitTermination(1, TimeUnit.SECONDS);

System.out.println(accumulator.getThenReset());
```

```
499999500000
```

Operacja musi być przemienna, w przeciwnym razie wynik będzie niepoprawny.

## Przetwarzanie współbieżne a kolekcje

- `CopyOnWriteArrayList`
- `ConcurrentHashMap`
- `parallelStream`

## ConcurrentModificationException

In [28]:
```
class ParallelLinkedList {
    public void run() throws InterruptedException {
        List<Integer> list = new LinkedList<>();  // <------------------------
        ExecutorService executor = Executors.newFixedThreadPool(200);
        IntStream.range(1,50).forEach((i) -> executor.submit(() -> {
              System.out.println("" + i + " writing to LinkedList " + Thread.cur
              list.add(i);
        }));
        executor.submit(() -> {
              Iterator<Integer> iterator = list.iterator();
              try {
                  while(iterator.hasNext()){
                          Thread.sleep(20);
                          System.out.println("" + iterator.next() + " reading fr
                              Thread.currentThread().getName());
                  }
              } catch(Exception ex) {
                  ex.printStackTrace();
              }
        });
        IntStream.range(1,50).forEach((i) -> executor.submit(() -> {
              System.out.println("" + i + " writing to LinkedList " + Thread.cur
              list.add(i);
        }));
        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.SECONDS);
    }
}
```

In [29]:

```
new ParallelLinkedList().run();
```

```
3 writing to LinkedList pool-15-thread-3
9 writing to LinkedList pool-15-thread-9
8 writing to LinkedList pool-15-thread-8
6 writing to LinkedList pool-15-thread-6
7 writing to LinkedList pool-15-thread-7
5 writing to LinkedList pool-15-thread-5
32 writing to LinkedList pool-15-thread-32
4 writing to LinkedList pool-15-thread-4
1 writing to LinkedList pool-15-thread-1
2 writing to LinkedList pool-15-thread-2
45 writing to LinkedList pool-15-thread-45
40 writing to LinkedList pool-15-thread-40
42 writing to LinkedList pool-15-thread-42
41 writing to LinkedList pool-15-thread-41
13 writing to LinkedList pool-15-thread-63
39 writing to LinkedList pool-15-thread-39
38 writing to LinkedList pool-15-thread-38
36 writing to LinkedList pool-15-thread-36
37 writing to LinkedList pool-15-thread-37
35 writing to LinkedList pool-15-thread-35
33 writing to LinkedList pool-15-thread-33
34 writing to LinkedList pool-15-thread-34
31 writing to LinkedList pool-15-thread-31
30 writing to LinkedList pool-15-thread-30
29 writing to LinkedList pool-15-thread-29
28 writing to LinkedList pool-15-thread-28
27 writing to LinkedList pool-15-thread-27
26 writing to LinkedList pool-15-thread-26
25 writing to LinkedList pool-15-thread-25
24 writing to LinkedList pool-15-thread-24
23 writing to LinkedList pool-15-thread-23
40 writing to LinkedList pool-15-thread-90
45 writing to LinkedList pool-15-thread-95
22 writing to LinkedList pool-15-thread-22
20 writing to LinkedList pool-15-thread-20
26 writing to LinkedList pool-15-thread-76
21 writing to LinkedList pool-15-thread-21
19 writing to LinkedList pool-15-thread-19
14 writing to LinkedList pool-15-thread-14
17 writing to LinkedList pool-15-thread-17
18 writing to LinkedList pool-15-thread-18
15 writing to LinkedList pool-15-thread-15
16 writing to LinkedList pool-15-thread-16
13 writing to LinkedList pool-15-thread-13
java.util.ConcurrentModificationException
12 writing to LinkedList pool-15-thread-12
11 writing to LinkedList pool-15-thread-11
        at java.base/java.util.LinkedList$ListItr.checkForComodification(Linked
List.java:970)
10 writing to LinkedList pool-15-thread-10
44 writing to LinkedList pool-15-thread-94
        at java.base/java.util.LinkedList$ListItr.next(LinkedList.java:892)
46 writing to LinkedList pool-15-thread-96
```

```
        at REPL.$JShell$128$ParallelLinkedList.lambda$run$2($JShell$128.java:3
3)
21 writing to LinkedList pool-15-thread-71
        at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Execut
ors.java:539)
20 writing to LinkedList pool-15-thread-70
        at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPo
olExecutor.java:1136)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadP
oolExecutor.java:635)
        at java.base/java.lang.Thread.run(Thread.java:833)
48 writing to LinkedList pool-15-thread-98
49 writing to LinkedList pool-15-thread-99
47 writing to LinkedList pool-15-thread-97
39 writing to LinkedList pool-15-thread-89
43 writing to LinkedList pool-15-thread-93
17 writing to LinkedList pool-15-thread-67
41 writing to LinkedList pool-15-thread-91
34 writing to LinkedList pool-15-thread-84
38 writing to LinkedList pool-15-thread-88
32 writing to LinkedList pool-15-thread-82
37 writing to LinkedList pool-15-thread-87
36 writing to LinkedList pool-15-thread-86
31 writing to LinkedList pool-15-thread-81
30 writing to LinkedList pool-15-thread-80
35 writing to LinkedList pool-15-thread-85
29 writing to LinkedList pool-15-thread-79
33 writing to LinkedList pool-15-thread-83
28 writing to LinkedList pool-15-thread-78
27 writing to LinkedList pool-15-thread-77
25 writing to LinkedList pool-15-thread-75
24 writing to LinkedList pool-15-thread-74
23 writing to LinkedList pool-15-thread-73
22 writing to LinkedList pool-15-thread-72
18 writing to LinkedList pool-15-thread-68
42 writing to LinkedList pool-15-thread-92
19 writing to LinkedList pool-15-thread-69
16 writing to LinkedList pool-15-thread-66
15 writing to LinkedList pool-15-thread-65
12 writing to LinkedList pool-15-thread-62
14 writing to LinkedList pool-15-thread-64
9 writing to LinkedList pool-15-thread-59
10 writing to LinkedList pool-15-thread-60
11 writing to LinkedList pool-15-thread-61
8 writing to LinkedList pool-15-thread-58
7 writing to LinkedList pool-15-thread-57
4 writing to LinkedList pool-15-thread-54
3 writing to LinkedList pool-15-thread-53
6 writing to LinkedList pool-15-thread-56
5 writing to LinkedList pool-15-thread-55
2 writing to LinkedList pool-15-thread-52
1 writing to LinkedList pool-15-thread-51
46 writing to LinkedList pool-15-thread-46
49 writing to LinkedList pool-15-thread-49
43 writing to LinkedList pool-15-thread-43
48 writing to LinkedList pool-15-thread-48
47 writing to LinkedList pool-15-thread-47
44 writing to LinkedList pool-15-thread-44
```

## `CopyOnWriteArrayList`

In [30]:
```
import java.util.*;
import java.util.stream.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.*;

class ParallelCopyOnWriteArrayList {
    public void run() throws InterruptedException {
        List<Integer> list = new CopyOnWriteArrayList<>();  // <----------------
        ExecutorService executor = Executors.newFixedThreadPool(200);
        IntStream.range(1,50).forEach((i) -> executor.submit(() -> {
              System.out.println("" + i + " writing to CopyOnWriteArrayList " +
              list.add(i);
        }));
        executor.submit(() -> {
              Iterator<Integer> iterator1 = list.iterator();
              while(iterator1.hasNext()){
                    try {
                        Thread.sleep(20);
                        System.out.println("" + iterator1.next() + " reading fro
                        Thread.currentThread().getName());
                    } catch(Exception ex) {
                        ex.printStackTrace();
                    }
              }
        });
        IntStream.range(1,50).forEach((i) -> executor.submit(() -> {
              System.out.println("" + i + " writing to CopyOnWriteArrayList " +
              list.add(i);
        }));
        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.SECONDS);
    }
}
```

In [31]:
```
new ParallelCopyOnWriteArrayList().run();
```

```
2 writing to CopyOnWriteArrayList pool-16-thread-2
12 writing to CopyOnWriteArrayList pool-16-thread-12
11 writing to CopyOnWriteArrayList pool-16-thread-11
22 writing to CopyOnWriteArrayList pool-16-thread-22
10 writing to CopyOnWriteArrayList pool-16-thread-10
8 writing to CopyOnWriteArrayList pool-16-thread-8
9 writing to CopyOnWriteArrayList pool-16-thread-9
7 writing to CopyOnWriteArrayList pool-16-thread-7
4 writing to CopyOnWriteArrayList pool-16-thread-4
6 writing to CopyOnWriteArrayList pool-16-thread-6
1 writing to CopyOnWriteArrayList pool-16-thread-1
5 writing to CopyOnWriteArrayList pool-16-thread-5
33 writing to CopyOnWriteArrayList pool-16-thread-33
3 writing to CopyOnWriteArrayList pool-16-thread-3
43 writing to CopyOnWriteArrayList pool-16-thread-43
44 writing to CopyOnWriteArrayList pool-16-thread-44
42 writing to CopyOnWriteArrayList pool-16-thread-42
41 writing to CopyOnWriteArrayList pool-16-thread-41
2 writing to CopyOnWriteArrayList pool-16-thread-52
5 writing to CopyOnWriteArrayList pool-16-thread-55
16 writing to CopyOnWriteArrayList pool-16-thread-16
40 writing to CopyOnWriteArrayList pool-16-thread-40
38 writing to CopyOnWriteArrayList pool-16-thread-38
26 writing to CopyOnWriteArrayList pool-16-thread-26
37 writing to CopyOnWriteArrayList pool-16-thread-37
25 writing to CopyOnWriteArrayList pool-16-thread-25
23 writing to CopyOnWriteArrayList pool-16-thread-23
35 writing to CopyOnWriteArrayList pool-16-thread-35
47 writing to CopyOnWriteArrayList pool-16-thread-47
20 writing to CopyOnWriteArrayList pool-16-thread-20
32 writing to CopyOnWriteArrayList pool-16-thread-32
28 writing to CopyOnWriteArrayList pool-16-thread-28
31 writing to CopyOnWriteArrayList pool-16-thread-31
27 writing to CopyOnWriteArrayList pool-16-thread-27
17 writing to CopyOnWriteArrayList pool-16-thread-17
39 writing to CopyOnWriteArrayList pool-16-thread-39
36 writing to CopyOnWriteArrayList pool-16-thread-36
29 writing to CopyOnWriteArrayList pool-16-thread-29
21 writing to CopyOnWriteArrayList pool-16-thread-21
19 writing to CopyOnWriteArrayList pool-16-thread-19
18 writing to CopyOnWriteArrayList pool-16-thread-18
24 writing to CopyOnWriteArrayList pool-16-thread-24
15 writing to CopyOnWriteArrayList pool-16-thread-15
14 writing to CopyOnWriteArrayList pool-16-thread-14
13 writing to CopyOnWriteArrayList pool-16-thread-13
32 writing to CopyOnWriteArrayList pool-16-thread-82
29 writing to CopyOnWriteArrayList pool-16-thread-79
28 writing to CopyOnWriteArrayList pool-16-thread-78
27 writing to CopyOnWriteArrayList pool-16-thread-77
25 writing to CopyOnWriteArrayList pool-16-thread-75
24 writing to CopyOnWriteArrayList pool-16-thread-74
23 writing to CopyOnWriteArrayList pool-16-thread-73
16 writing to CopyOnWriteArrayList pool-16-thread-66
22 writing to CopyOnWriteArrayList pool-16-thread-72
48 writing to CopyOnWriteArrayList pool-16-thread-98
40 writing to CopyOnWriteArrayList pool-16-thread-90
47 writing to CopyOnWriteArrayList pool-16-thread-97
20 writing to CopyOnWriteArrayList pool-16-thread-70
49 writing to CopyOnWriteArrayList pool-16-thread-99
39 writing to CopyOnWriteArrayList pool-16-thread-89
```

```
45 writing to CopyOnWriteArrayList pool-16-thread-95
46 writing to CopyOnWriteArrayList pool-16-thread-96
44 writing to CopyOnWriteArrayList pool-16-thread-94
42 writing to CopyOnWriteArrayList pool-16-thread-92
41 writing to CopyOnWriteArrayList pool-16-thread-91
43 writing to CopyOnWriteArrayList pool-16-thread-93
38 writing to CopyOnWriteArrayList pool-16-thread-88
12 writing to CopyOnWriteArrayList pool-16-thread-62
36 writing to CopyOnWriteArrayList pool-16-thread-86
2 reading from CopyOnWriteArrayList pool-16-thread-50
35 writing to CopyOnWriteArrayList pool-16-thread-85
34 writing to CopyOnWriteArrayList pool-16-thread-84
31 writing to CopyOnWriteArrayList pool-16-thread-81
33 writing to CopyOnWriteArrayList pool-16-thread-83
30 writing to CopyOnWriteArrayList pool-16-thread-80
26 writing to CopyOnWriteArrayList pool-16-thread-76
21 writing to CopyOnWriteArrayList pool-16-thread-71
15 writing to CopyOnWriteArrayList pool-16-thread-65
19 writing to CopyOnWriteArrayList pool-16-thread-69
18 writing to CopyOnWriteArrayList pool-16-thread-68
17 writing to CopyOnWriteArrayList pool-16-thread-67
14 writing to CopyOnWriteArrayList pool-16-thread-64
13 writing to CopyOnWriteArrayList pool-16-thread-63
37 writing to CopyOnWriteArrayList pool-16-thread-87
10 writing to CopyOnWriteArrayList pool-16-thread-60
9 writing to CopyOnWriteArrayList pool-16-thread-59
11 writing to CopyOnWriteArrayList pool-16-thread-61
8 writing to CopyOnWriteArrayList pool-16-thread-58
7 writing to CopyOnWriteArrayList pool-16-thread-57
6 writing to CopyOnWriteArrayList pool-16-thread-56
4 writing to CopyOnWriteArrayList pool-16-thread-54
3 writing to CopyOnWriteArrayList pool-16-thread-53
34 writing to CopyOnWriteArrayList pool-16-thread-34
1 writing to CopyOnWriteArrayList pool-16-thread-51
48 writing to CopyOnWriteArrayList pool-16-thread-48
49 writing to CopyOnWriteArrayList pool-16-thread-49
30 writing to CopyOnWriteArrayList pool-16-thread-30
46 writing to CopyOnWriteArrayList pool-16-thread-46
45 writing to CopyOnWriteArrayList pool-16-thread-45
12 reading from CopyOnWriteArrayList pool-16-thread-50
11 reading from CopyOnWriteArrayList pool-16-thread-50
22 reading from CopyOnWriteArrayList pool-16-thread-50
10 reading from CopyOnWriteArrayList pool-16-thread-50
8 reading from CopyOnWriteArrayList pool-16-thread-50
9 reading from CopyOnWriteArrayList pool-16-thread-50
7 reading from CopyOnWriteArrayList pool-16-thread-50
4 reading from CopyOnWriteArrayList pool-16-thread-50
6 reading from CopyOnWriteArrayList pool-16-thread-50
1 reading from CopyOnWriteArrayList pool-16-thread-50
5 reading from CopyOnWriteArrayList pool-16-thread-50
33 reading from CopyOnWriteArrayList pool-16-thread-50
3 reading from CopyOnWriteArrayList pool-16-thread-50
43 reading from CopyOnWriteArrayList pool-16-thread-50
44 reading from CopyOnWriteArrayList pool-16-thread-50
42 reading from CopyOnWriteArrayList pool-16-thread-50
```

## `CommonPoolParallelism`

In [ ]:
```
import java.util.concurrent.*;

System.out.println(ForkJoinPool.getCommonPoolParallelism());

//-Djava.util.concurrent.ForkJoinPool.common.parallelism=5
```

## ConcurrentHashMap

In [32]:
```
import java.util.concurrent.*;

ConcurrentHashMap<String, String> map = new ConcurrentHashMap<>();
map.put("1", "jeden");
map.put("2", "dwa");
map.put("3", "trzy");
map.put("4", "cztery");
map.put("5", "pięć");
map.put("6", "sześć");
map.put("7", "siedem");

map.forEach(1, (key, value) -> System.out.printf("klucz: %s; wartość: %s; wątek:
        key, value, Thread.currentThread().getName()));
```

```
klucz: 1; wartość: jeden; wątek: IJava-executor-15
klucz: 2; wartość: dwa; wątek: IJava-executor-15
klucz: 4; wartość: cztery; wątek: ForkJoinPool.commonPool-worker-2
klucz: 5; wartość: pięć; wątek: ForkJoinPool.commonPool-worker-2
klucz: 6; wartość: sześć; wątek: ForkJoinPool.commonPool-worker-3
klucz: 3; wartość: trzy; wątek: IJava-executor-15
klucz: 7; wartość: siedem; wątek: ForkJoinPool.commonPool-worker-3
```

## search

In [34]:
```
String result = map.search(1, (key, value) -> {
    System.out.println(Thread.currentThread().getName());
    if ("dwa".equals(value)) {
        return key;
    }
    return null;
});
System.out.println("Wynik: " + result);
```

```
IJava-executor-16
IJava-executor-16
ForkJoinPool.commonPool-worker-6
Wynik: 2
```

## reduce

In [37]:
```
String result = map.reduce(1,
    (key, value) -> {
        // System.out.println("Przekształcenie: " + Thread.currentThread().getNa
        return key + "=" + value;
    },
    (s1, s2) -> {
         System.out.println("Redukcja: " + Thread.currentThread().getName());
        return s1 + ", " + s2;
    });

System.out.println("Wynik: " + result);
```

```
Redukcja: ForkJoinPool.commonPool-worker-8
Redukcja: ForkJoinPool.commonPool-worker-2
Redukcja: ForkJoinPool.commonPool-worker-4
Redukcja: ForkJoinPool.commonPool-worker-4
Redukcja: ForkJoinPool.commonPool-worker-4
Redukcja: ForkJoinPool.commonPool-worker-4
Wynik: 1=jeden, 2=dwa, 3=trzy, 4=cztery, 5=pięć, 6=sześć, 7=siedem
```

## `stream`

In [38]:
```
import java.util.stream.*;


long start = System.nanoTime();
long multiplier = 1000;

double result = LongStream.range(0, 1000000 * multiplier)
                          .filter(i -> i * i % 7 != 0)
                          .average()
                          .getAsDouble();

long end = System.nanoTime();

System.out.println(result);
System.out.println((end - start) * 1.0 / 1000000);
```

```
4.999999999166667E8
4652.017
```

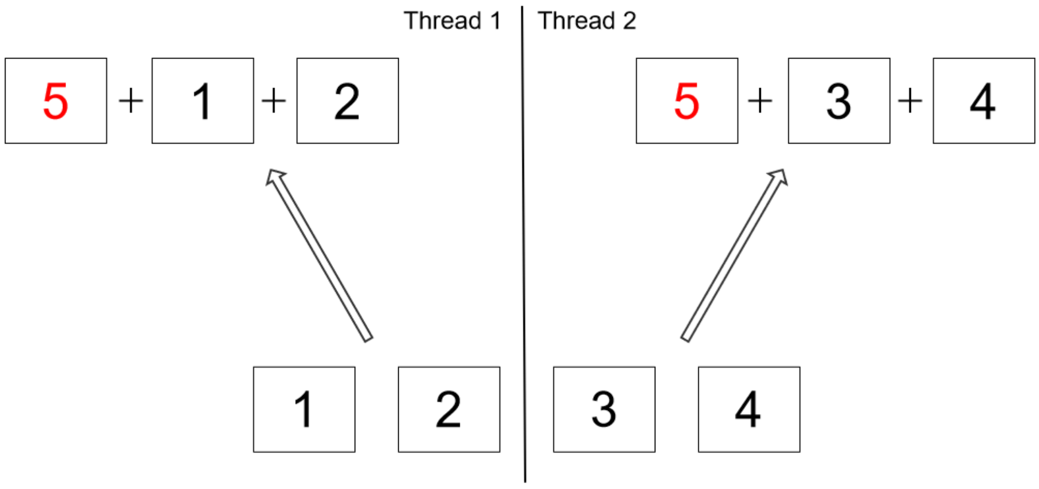
## `parallel` oraz `parallelStream`

In [39]:
```
import java.util.stream.*;

long start = System.nanoTime();
long multiplier = 1000;

double result = LongStream.range(0, 1000000 * multiplier)
                          .parallel()
                          .filter(i -> i * i % 7 != 0)
                          .average()
                          .getAsDouble();

long end = System.nanoTime();

System.out.println(result);
System.out.println((end - start) * 1.0 / 1000000);
```

```
4.999999999166667E8
438.3139
```

In [41]:
```
List<Integer> listOfNumbers = List.of(1, 2, 3, 4, 5);
int sum = listOfNumbers.parallelStream()
                       .reduce(5, Integer::sum);

System.out.println(sum);
```

```
40
```



**Wniosek:** trzeba uważać co zrównoleglamy i w jakich sytuacjach używamy `parallel()` .

Uwaga: podobnie jak w przypadku metod na `ConcurrentHashMap` , metoda `parallel` tworzy strumień korzystający z CommonPool, co może mieć negatywne konsekwencje, jeśli jakiś wątek w tej puli będzie przetwarzał się bardzo długo.
https://www.baeldung.com/java-8-parallel-streams-custom-threadpool

# Współbieżność w JavaFX

- `Platform.runLater(Runnable r)`
- `javafx.concurrent.Task`
- `javafx.concurrent.Service`

## runLater

In [ ]:
```
public class MyApplication extends Application {

  @Override
  public void start(Stage primaryStage) {
    ProgressBar progressBar = new ProgressBar(0);

    VBox vBox = new VBox(progressBar);
    Scene scene = new Scene(vBox, 960, 600);
    primaryStage.setScene(scene);
    primaryStage.show();

    Thread taskThread = new Thread(new Runnable() {
      @Override
      public void run() {
        double progress = 0;
        for(int i=0; i<10; i++){
          try {
            Thread.sleep(1000);
          } catch (InterruptedException e) {
            e.printStackTrace();
          }

          progress += 0.1;
          final double reportedProgress = progress;

          Platform.runLater(new Runnable() {              // <-----
            @Override
            public void run() {
              progressBar.setProgress(reportedProgress);
            }
          });
        }
      }
    });

    taskThread.start();
  }
}
// źródło: http://tutorials.jenkov.com/javafx/concurrency.html
```

## Task

In [ ]:
```
// źródło: https://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm

import javafx.concurrent.Task;
```

```
Task task = new Task<Void>() {
    @Override public Void call() {
        static final int max = 1000000;
        for (int i=1; i<=max; i++) {
            if (isCancelled()) {
               break;
            }
            updateProgress(i, max);
        }
        return null;
    }
};
ProgressBar bar = new ProgressBar();
bar.progressProperty().bind(task.progressProperty());
new Thread(task).start();
```

## Service

In [ ]:
```
// źródło: https://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm

public class FirstLineServiceApp extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        FirstLineService service = new FirstLineService();
        service.setUrl("http://google.com");
        service.setOnSucceeded(new EventHandler<WorkerStateEvent>() {
            @Override
            public void handle(WorkerStateEvent t) {
                System.out.println("done:" + t.getSource().getValue());
            }
        });
        service.start();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

In [ ]:
```
class FirstLineService extends Service<String> {
    private StringProperty url = new SimpleStringProperty();

    public final void setUrl(String value) {
        url.set(value);
    }

    public final String getUrl() {
        return url.get();
    }

    public final StringProperty urlProperty() {
       return url;
    }

    @Override
    protected Task<String> createTask() {
        final String _url = getUrl();
        return new Task<String>() {
            protected String call()
                throws IOException, MalformedURLException {
                    String result = null;
                    BufferedReader input = null;
                    try {
                        URL currentUrl = new URL(_url);
                        input = new BufferedReader(new InputStreamReader(current
                        result = input.readLine();
                    } finally {
                        if (input != null) {
                            input.close();
                        }
                    }
                    return result;
            }
        };
    }
}
```