Multithreading

Adam Krechowicz

13 maja 2020

Thread class

```
1  public class Threads extends Thread {
2
3    public void run(){
4      System.out.println("Hello from other thread");
5    }
6
7    public static void main(String[] args) {
8      Threads threads = new Threads();
9      threads.start();
10     //thread.run();
11   }
12 }
```

Thread class

```
1  public class Threads extends Thread {
2
3    public void run(){
4      while (1==1)
5        System.out.println("Hello from other thread");
6    }
7
8    public static void main(String[] args) {
9      Threads threads = new Threads();
10     threads.start();
11   }
12 }
```

Runnable interface

```
1
2  public class Interface implements Runnable {
3    public Interface() {
4      super();
5    }
6
7    public void run() {
8    }
9
10   public static void main(String[] args){
11     Thread t = new Thread(new Interface());
12     t.start();
13   }
14 }
```

Sleeping

```
1  public void run(){
2    while (1==1){
3      System.out.println("Different thread");
4      try {
5        Thread.sleep(1000);
6      } catch (InterruptedException e) {
7        return;
8      }
9    }
```

Thread interruption

```
1  public class Interruption extends Thread {
2    public void run(){
3      while(1==1){
4        try {
5          Thread.sleep(1000);
6        } catch (InterruptedException e) {
7          return;
8        }
9        if (Thread.interrupted())
10         return;
11     }
12   }
13   public static void main(String[] args) {
14     Interruption interruption = new Interruption();
15     interruption.interrupt();
16   }
17 }
```

Thread priorities

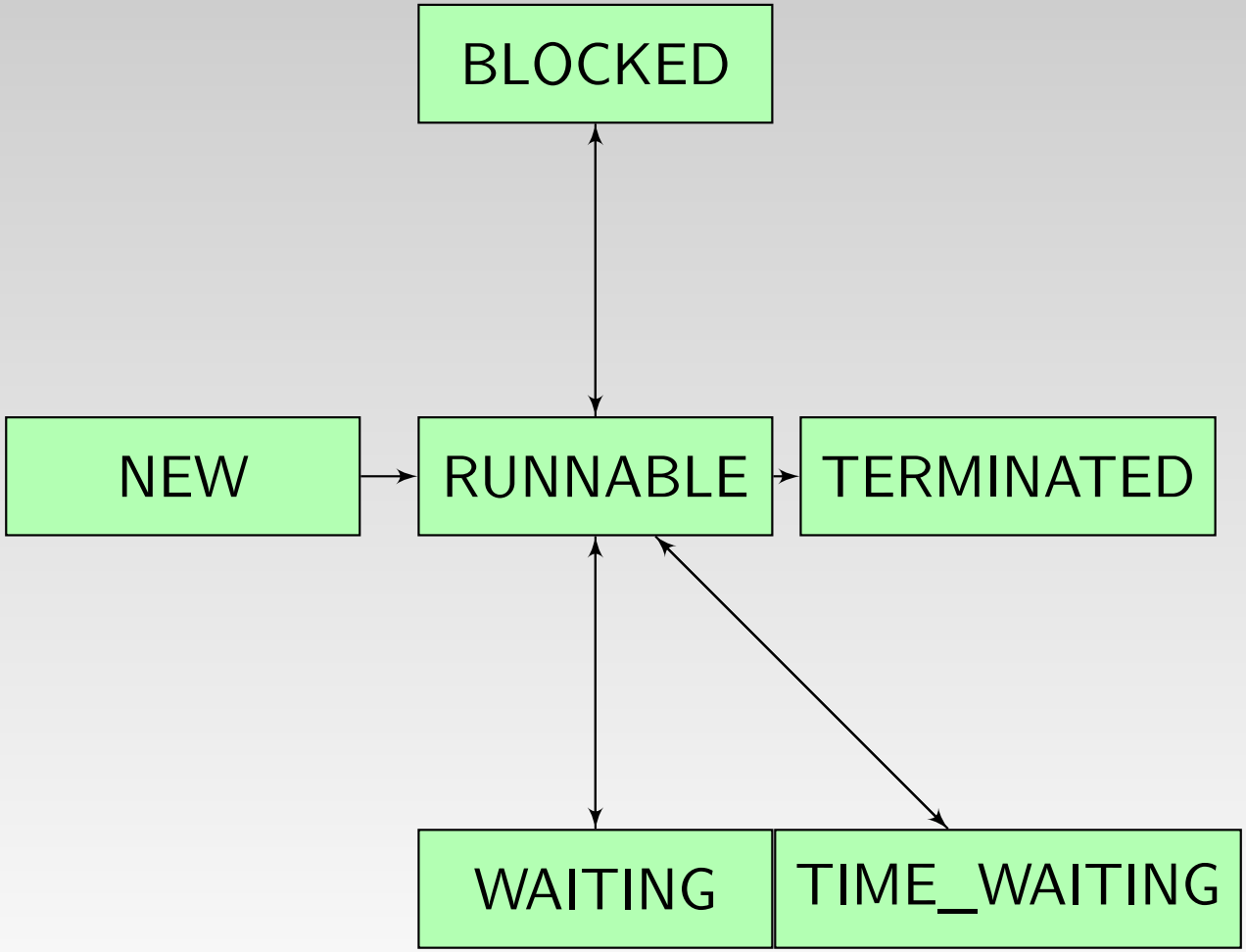
- Thread.MAX_PRIORITY
- Thread.MIN_PRIORITY
- Thread.NORM_PRIORITY

```
1    Threads threads = new Threads();
2    System.out.println("Priorytet: "+threads.getPriority());
3    threads.setPriority(Thread.MAX_PRIORITY);
```

Thread states

```
1    Threads threads = new Threads();
2    System.out.println(threads.getState());
3    threads.start();
4    System.out.println(threads.getState());;
```

Thread states



Ending threads

```
1     try {
2       threads.join();
3     } catch (InterruptedException e) {
4       e.printStackTrace();
5     }
```

- join()
- join(millis)
- join(millis, nanos)

Other methods

- dumpStack()
- getId()
- getName(), setName()
- isAlive()
- isInterrupted()

Deamons

```
1  public class Daemons extends Thread {
2
3    public void run(){
4      for (;;)
5        System.out.println("aaa");
6    }
7
8    public static void main(String[] args) {
9      Daemons daemons = new Daemons();
10     daemons.setDaemon(true);
11     System.out.println(daemons.isDaemon());
12     daemons.start();
13   }
14 }
```

Problems

- Problems with running threads
- Problems with values of data

Problem

```
1  public class Risks {
2    static Integer i = 0;
3    public static void main(String[] args) {
4      Thread t1 = new Thread() {
5        public void run() {
6          while (1 == 1) {
7              if (i == 0) i++;
8              else i--;
9          }
10       }
11     };
```

Problem

```
1   Thread t2 = new Thread() {
2     public void run() {
3       while (1 == 1) {
4           if (i == 0) i++;
5           else i--;
6       }
7     }
8   };
9
10  Thread t3 = new Thread() {
11    public void run() {
12      while (1 == 1)
13        System.err.println(i);
14    }
15  };
16  t3.start();
17  t1.start();
18  t2.start();
19 }
```

Result

i value

$$i == 0$$

Works!

```
1  -> if (i == 0)
2       i++;
3     else
4       i--;
5
```

Stopped!

```
1  -> if (i == 0)
2       i++;
3     else
4       i--;
5
```

Result

i value

$$i == 0$$

Stopped!

```
1      if (i == 0)
2  ->    i++;
3      else
4        i--;
5
```

Works!

```
1  ->  if (i == 0)
2        i++;
3      else
4        i--;
5
```

Result

i value

$$i == 1$$

Stopped!

```
1     if (i == 0)
2  ->   i++;
3     else
4       i--;
5
```

Works!

```
1     if (i == 0)
2  ->   i++;
3     else
4       i--;
5
```

Result

i value

i == 2

Works!

```
1     if (i == 0)
2  ->   i++;
3     else
4       i--;
5
```

Stopped!

```
1     if (i == 0)
2  ->   i++;
3     else
4       i--;
5
```

Other problem

```
1  class Watek1 extends Thread{
2    public void run(){
3      for (;;) {
4        System.err.println("1 - Start");
5        System.err.println("1 - Doing");
6        System.err.println("1 - End");
7      }
8    }
9  }
10
11 class Watek2 extends Thread{
12   public void run(){
13     for (;;) {
14       System.err.println("2 - Start");
15       System.err.println("2 - Doing");
16       System.err.println("2 - End");
17     }
18   }
19 }
```

Atomicity

Incrementation

```
1   i++;
```

1. Getting value of i
2. Adding 1
3. Writing result to i

Atomic operations

- Assigning primitive types (except for long and double)
- Assigning reference
- Reading primitive value
- Reading reference

Synchronized block

```
1  public class Risks {
2    static Integer i = 0;
3    public static void main(String[] args) {
4      Thread t1 = new Thread() {
5        public void run() {
6          while (1 == 1) {
7            synchronized (i) {
8              if (i == 0)
9                i++;
10             else
11               i--;
12           }
13         }
14       }
15     };
```

Synchronized block

```
1   Thread t2 = new Thread() {
2     public void run() {
3       while (1 == 1) {
4         synchronized (i) {
5           if (i == 0)
6             i++;
7           else
8             i--;
9         }
10      }
11    }
12  };
13  Thread t3 = new Thread() {
14    public void run() {
15      while (1 == 1)
16        synchronized (i) {
17          System.err.println(i);
18        }
19    }
20  };
```

Synchronized methods

```
1  public class SynchMethods {
2    private int i;
3    public int getI(){
4      return i;
5    }
6    public synchronized void metoda(){
7      i++;
8    }
9  }
```

Synchronized methods

```
1  class Watek extends Thread{
2    SynchMethods sm = null;
3    public Watek(SynchMethods sm){
4      this.sm = sm;
5    }
6    public void run(){
7      while (1==1){
8        sm.metoda();
9        System.out.println(sm.getI());
10     }
11   }
12 }
```

Synchronized methods

```
1  public static void main(String[] args) {
2    SynchMethods sm = new SynchMethods();
3    Watek w1 = new Watek(sm);
4    Watek w2 = new Watek(sm);
5    w1.run();
6    w2.run();
7  }
```

Synchronized methods

- Synchronization is always performed on some object
- Synchronized method performs synchronization on object that was called
- Synchronized static method?
- Synchronized constructor?

Volatile field

```
1  public class Volatiles extends Thread {
2
3    volatile int i = 0;
4
5    public void run(){
6      while(1==1){
7        i = i+1;
8        System.out.println(i);
9      }
10   }
11
12   public static void main(String[] args) {
13     (new Volatiles()).start();
14     (new Volatiles()).start();
15   }
16 }
```

Atomic variables

- java.util.concurrent.atomic.AtomicIteger
- java.util.concurrent.atomic.AtomicBoolean
- java.util.concurrent.atomic.AtomicLong
- java.util.concurrent.atomic.AtomicIntegerArray
- java.util.concurrent.atomic.AtomicLongArray
- java.util.concurrent.atomic.AtomicReference

Atomic variables

```
1   AtomicInteger i = new AtomicInteger();
2   i.set(5);
3   int j = i.get();
4   System.out.println(i);
5   System.out.println(i.addAndGet(3));
6   System.out.println(i.decrementAndGet());
7   System.out.println(i.incrementAndGet());
8   System.out.println(i.getAndIncrement());
9   System.out.println(i);
10  if (i.compareAndSet(9, 1))
11    System.out.println("Bylo 9")
```

Synchronized collections

- BlockingQueue: ArrayBlockingQueue, PriorityBlockingQueue, SynchronousQueue
- ConcurrentMap
- ConcurrentHashMap

Object class

- wait()
- notify()
- notifyAll()

Object class

```
1  public class ObjectMth {
2    Object o;
3    public ObjectMth() {
4      super();
5      o = new Object();
6    }
7    class Watek1 extends Thread{
8      public void run(){
9        System.err.println("Czekam na drugi watek");
10       synchronized(o){
11         try {
12           o.wait();
13         } catch (InterruptedException e) {
14           e.printStackTrace();
15         }
16       }
17       System.err.println("Drugi watek skonczyl");
18     }
19   }
```

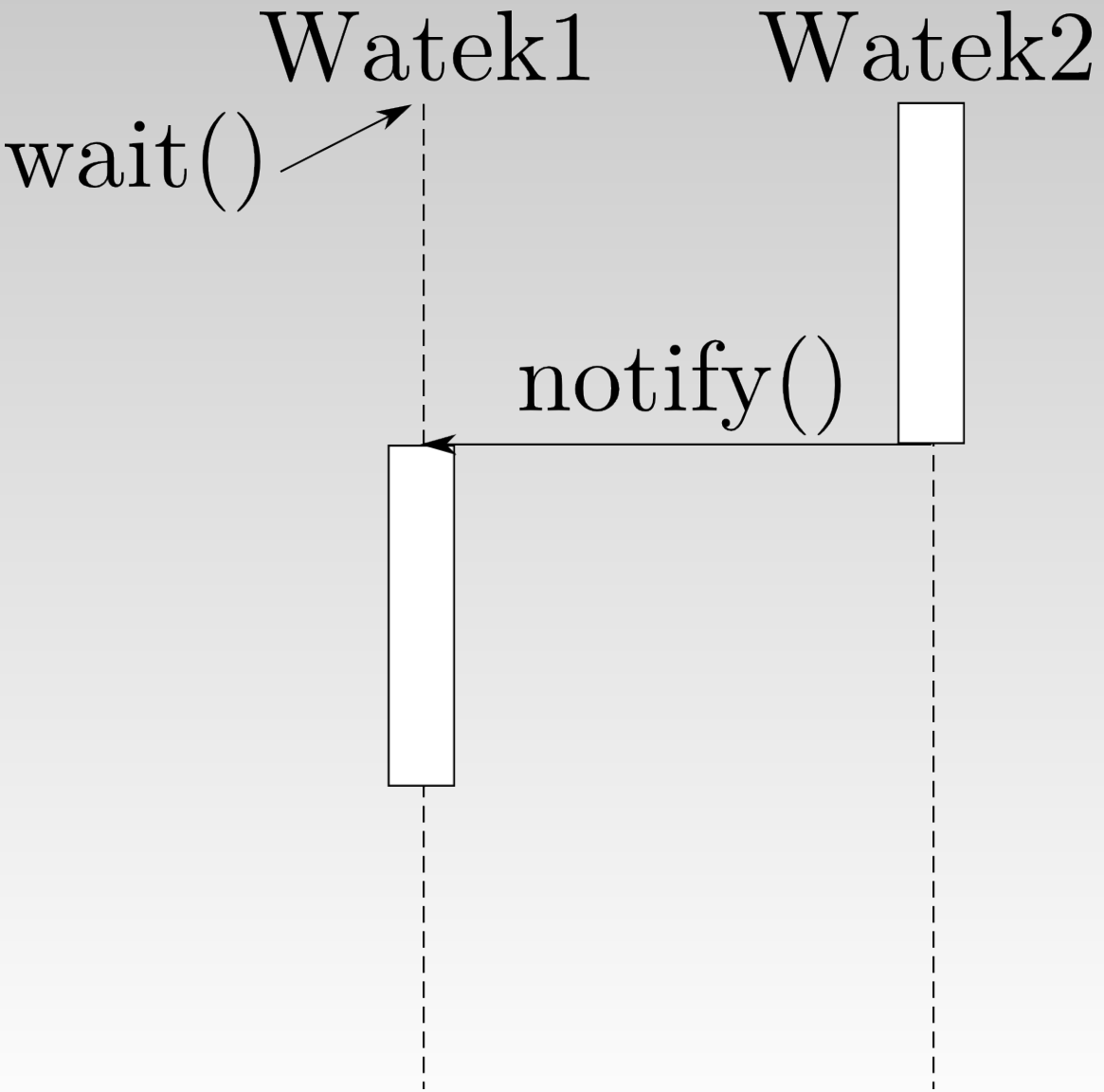
Object class

```
1   class Watek2 extends Thread{
2     public void run(){
3       System.err.println("Drugi watek zaczyna dzialac");
4       try {
5         Thread.sleep(2000);
6       } catch (InterruptedException e) {
7         e.printStackTrace();
8       }
9       System.err.println("Drugi watek konczy dzialac");
10      synchronized(o){
11        o.notify();
12      }
13    }
14
15  }
```

Object class

```
1    public static void main(String[] args) {
2      ObjectMth objectMth = new ObjectMth();
3      (objectMth.new Watek1()).start();
4      (objectMth.new Watek2()).start();
5    }
6  }
```

Object class



Running single thread

```
1  public class ExtExecute implements Runnable {
2    public ExtExecute() {
3      super();
4    }
5
6    public static void main(String[] args) {
7      ExecutorService executor = Executors.newSingleThreadExecutor();
8      executor.execute(new ExtExecute());
9    }
10
11   public void run() {
12     System.out.println("Hello");
13   }
14 }
```

Pool of threads

```
1  public class Pools implements Runnable {
2    public Pools() {
3      super();
4    }
5
6    public static void main(String[] args) {
7      Pools pools = new Pools();
8      ExecutorService es = Executors.newFixedThreadPool(3);
9      es.execute(pools);
10     es.execute(pools);
11     es.execute(pools);
12     es.execute(pools);
13   }
14
15   public void run() {
16     System.out.println("Hello");
17     while (1==1){
18     }
19   }
20
21 }
```

Scheduling

```
1    ScheduledExecutorService e = Executors.newScheduledThreadPool(10);
2    e.schedule(new ExtExecute(), 1000, TimeUnit.MILLISECONDS);
```

- schedule(command, delay, timeUnit);
- scheduleAtFixedRate(command, delay, period, timeUnit)
- shutdown()
- shutdownNow()

THE END!

Comments available via Skype (Krechowicz.PSk@outook.com) and via mail (a.krechowicz@tu.kielce.pl)
Additional reading:

- https://docs.oracle.com/javase/tutorial/essential/concurrency/
- https://docs.oracle.com/javase/8/docs/technotes/guides/concurrency/index.html
- Thinking in Java chapters by Bruce Eckel:
  - Concurrency