VIA University College

Software Development with UML and Java 2

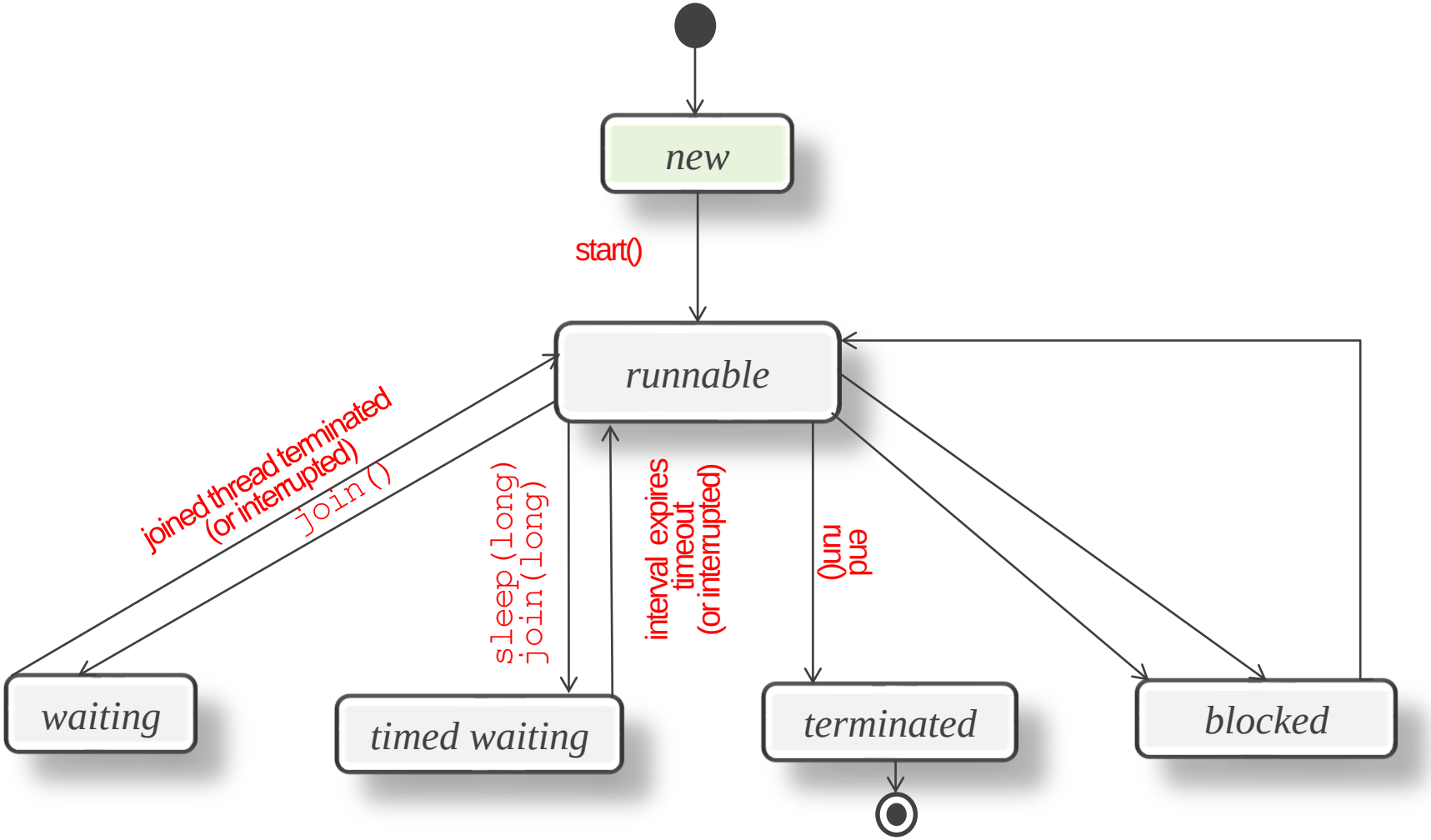
Autumn 2021

Joseph Okika (jook@via.dk)

Learning Objectives

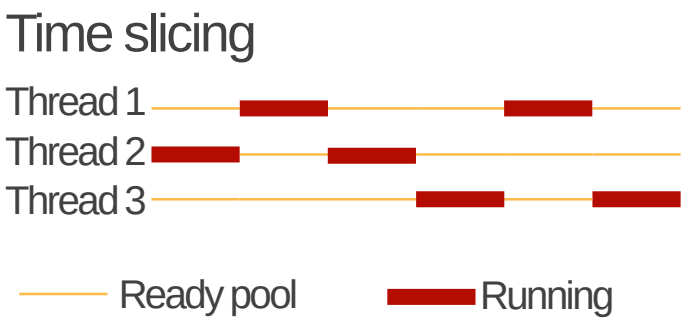
- By the end of class today, you will be able to:
  - ✓explain Thread synchronization
  - ✓ explain the concept - Monitor
  - ✓implement small programs using thread synchronization

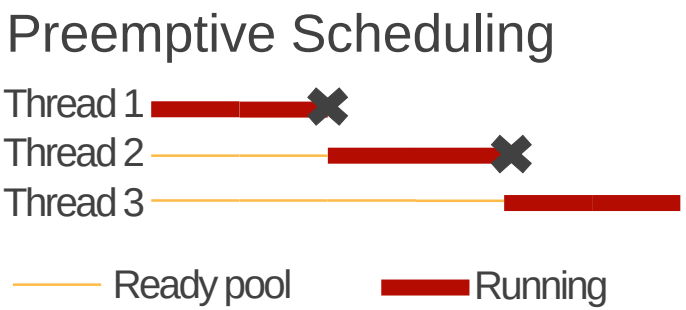
Thread States



Runnable state

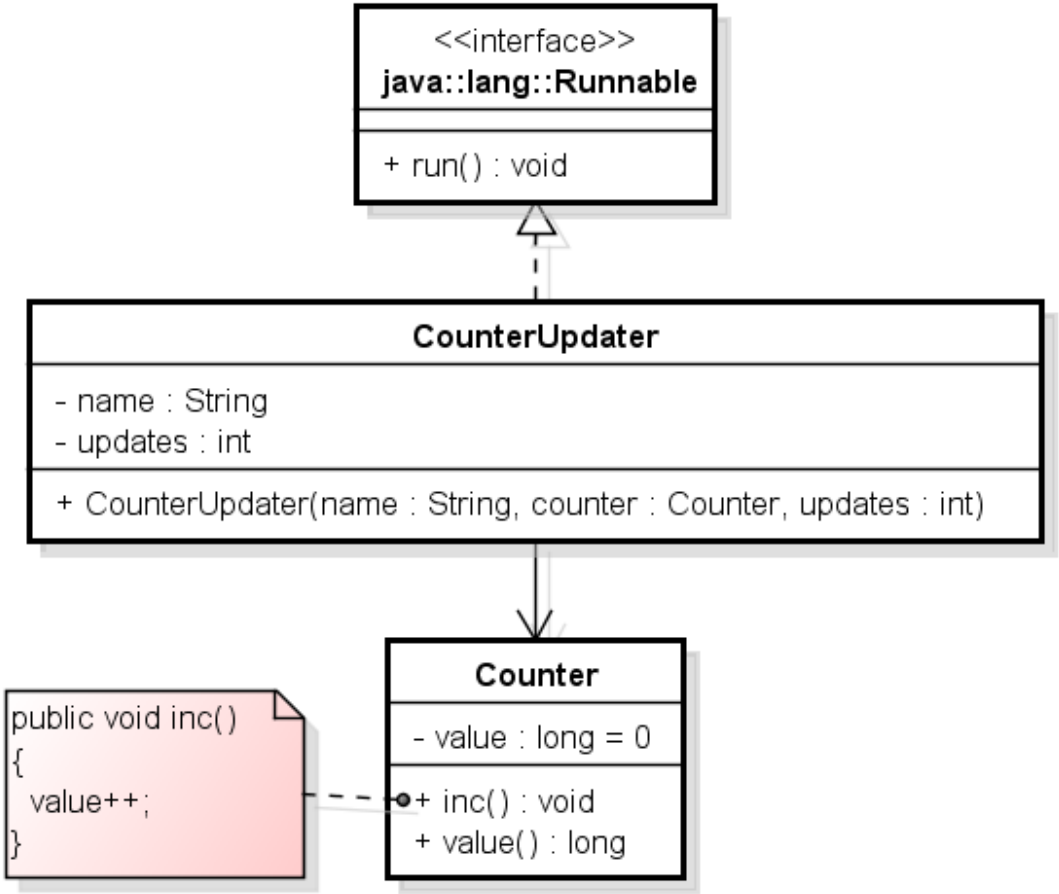
1. Running (scheduled CPU time)
   - Depends on thread priority, OS scheduling algorithm
2. Ready pool (ready for CPU time but not running)





- Give away CPU time voluntarily
  - yield()

Updating shared variables



Updating shared variables

```
public class TestCounter
{
   public static void main(String[] args)
   {
      Counter counter = new Counter();
      CounterUpdater c1
            = new CounterUpdater("Updater1", counter, 20000);
      CounterUpdater c2
            = new CounterUpdater("Updater2", counter, 20000);

      Thread t1 = new Thread(c1);
      Thread t2 = new Thread(c2);

      t1.start();
      t2.start();

      System.out.println("Main Thread Ended");
   }
}
```

What is the output?

Updating shared variables

```
Run: TestCounter
Main Thread Ended
Updater2 finished: Counter.value = 22985
Updater1 finished: Counter.value = 22985
```

```
Run: TestCounter
Main Thread Ended
Updater1 finished: Counter.value = 20421
Updater2 finished: Counter.value = 25173
```

```
Run: TestCounter
Main Thread Ended
Updater1 finished: Counter.value = 20107
Updater2 finished: Counter.value = 25608
```

```
Run: TestCounter
Main Thread Ended
Updater1 finished: Counter.value = 22351
Updater2 finished: Counter.value = 20843
```

Disassembled class file

Command Prompt

```
C:\_JOOK>javap -c C:\_JOOK\workspace\intellij\a21\sdj2\a21-sdj2-modules\out\producti
.session2\via\sdj2\sharedvar\Count
Compiled from "Counter.java"
public class via.sdj2.sharedvar.Co
  public via.sdj2.sharedvar.Counte
    Code:
       0: aload_0
       1: invokespecial #1                  thod java/lang/Object."<init>":()V
       4: return

  public void inc();
    Code:
       0: aload_0
       1: dup
       2: getfield      #7                  // Field value:J
       5: lconst_1
       6: ladd
       7: putfield      #7                  // Field value:J
      10: return

  public long value();
    Code:
       0: aload_0
       1: getfield      #7                  // Field value:J
       4: lreturn
}
```

```
public void inc()
{
  value++;
}
```

load a reference onto the stack from local variable 0

duplicate the value on top of the stack

get a field value of an object

push the long 1 onto the stack

add two longs

set field to value in an object

Disassembled class file

Command Prompt

```
C:\_JOOK>javap -c C:\_JOOK\workspace
.session2\via\sdj2\sharedvar\Counter
Compiled from "Counter.java"
public class via.sdj2.sharedvar.Coun
  public via.sdj2.sharedvar.Counter(
    Code:
       0: aload_0
       1: invokespecial #1
       4: return

  public void inc();
    Code:
       0: aload_0
       1: dup
       2: getfield      #7
       5: lconst_1
       6: ladd
       7: putfield      #7
      10: return

  public long value();
    Code:
       0: aload_0
       1: getfield      #7
       4: lreturn
}
```

Example: value = 10

value=10 → {10}

1 → {10, 1}

add → {11}

value=11 ← { }

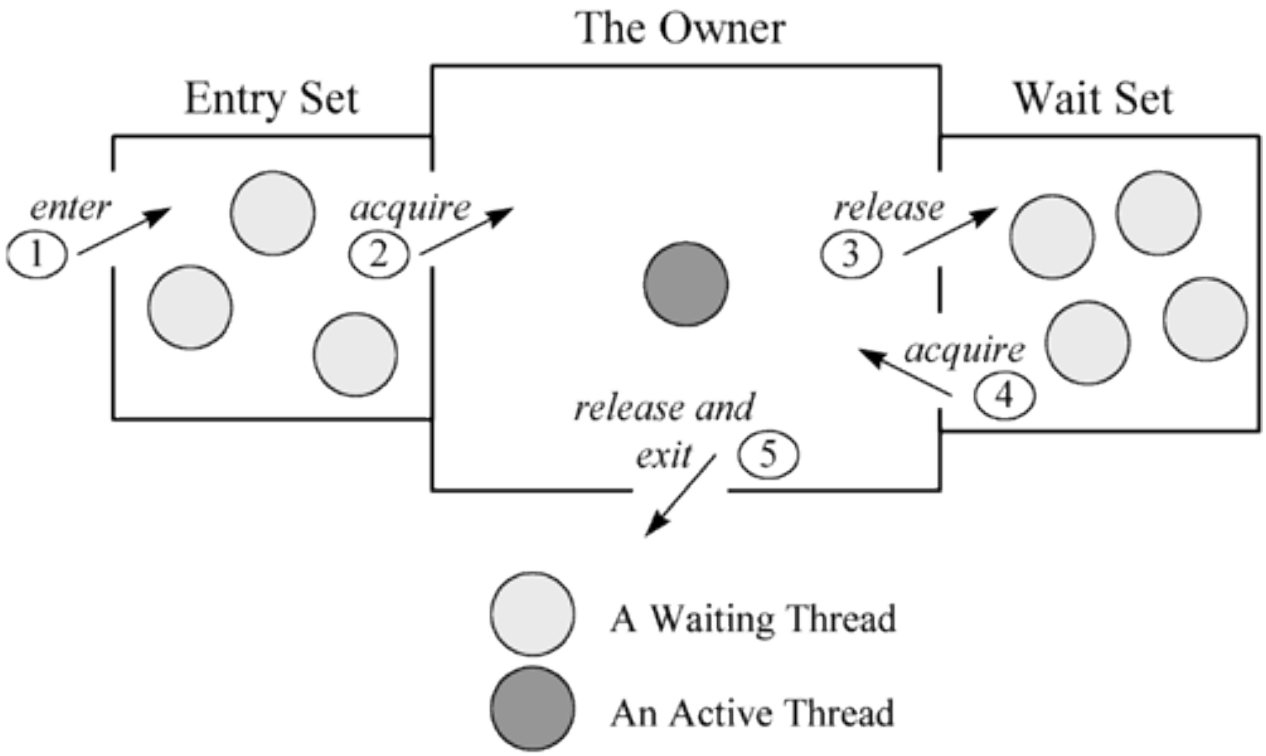
Thread Synchronization

- Thread synchronization is the process controlling and coordinating the access to a critical section by multiple threads
- Thread synchronization is always a challenging task when writing a multi-threaded program.
- Kinds:
  - Mutual exclusion synchronization
    - only one thread is allowed to have access to a section of code at a point in time
    - achieved through a lock via a monitor

Java Monitor

- A Monitor is a mechanism that ensures that at most one thread at a time can execute a given critical section or method.
- Every object in Java is potentially a Monitor
  - Keyword `synchronized` is used to define a critical section
  - Methods `wait()` and `wait(long)` are used to temporarily leave the Monitor and go to Wait State
  - Methods `notify()` and `notifyAll(long)` are used to wake up one or more threads from Wait State (making the waked-up thread go to Runnable and then directly to Blocked State until the Monitor is available)

Java Monitor ("The Owner")

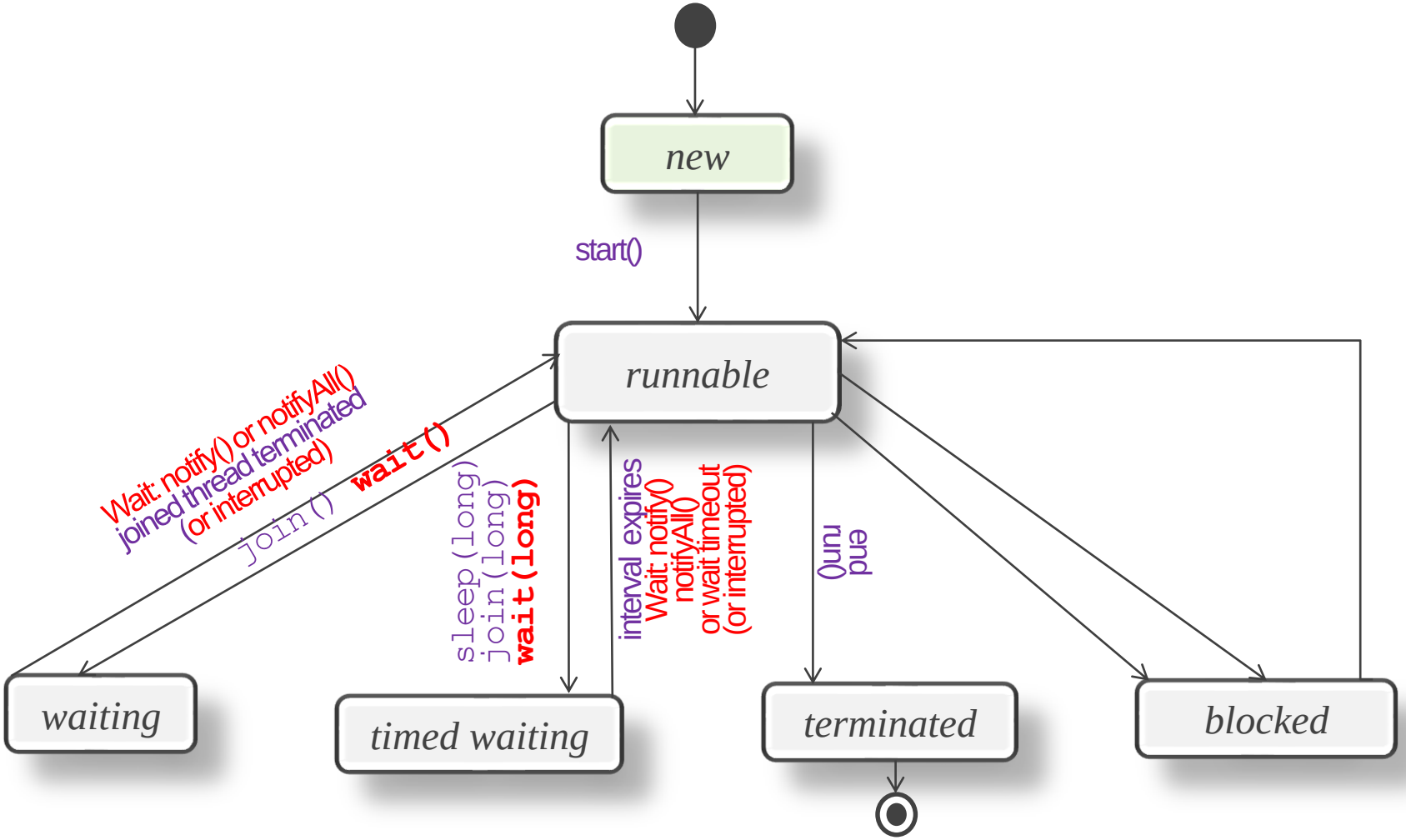


Bill Venners, "Thread Synchronization, Chapter 20 of Inside the Java Virtual Machine", http://www.artima.com/insidejvm/ed2/threadsynch.html

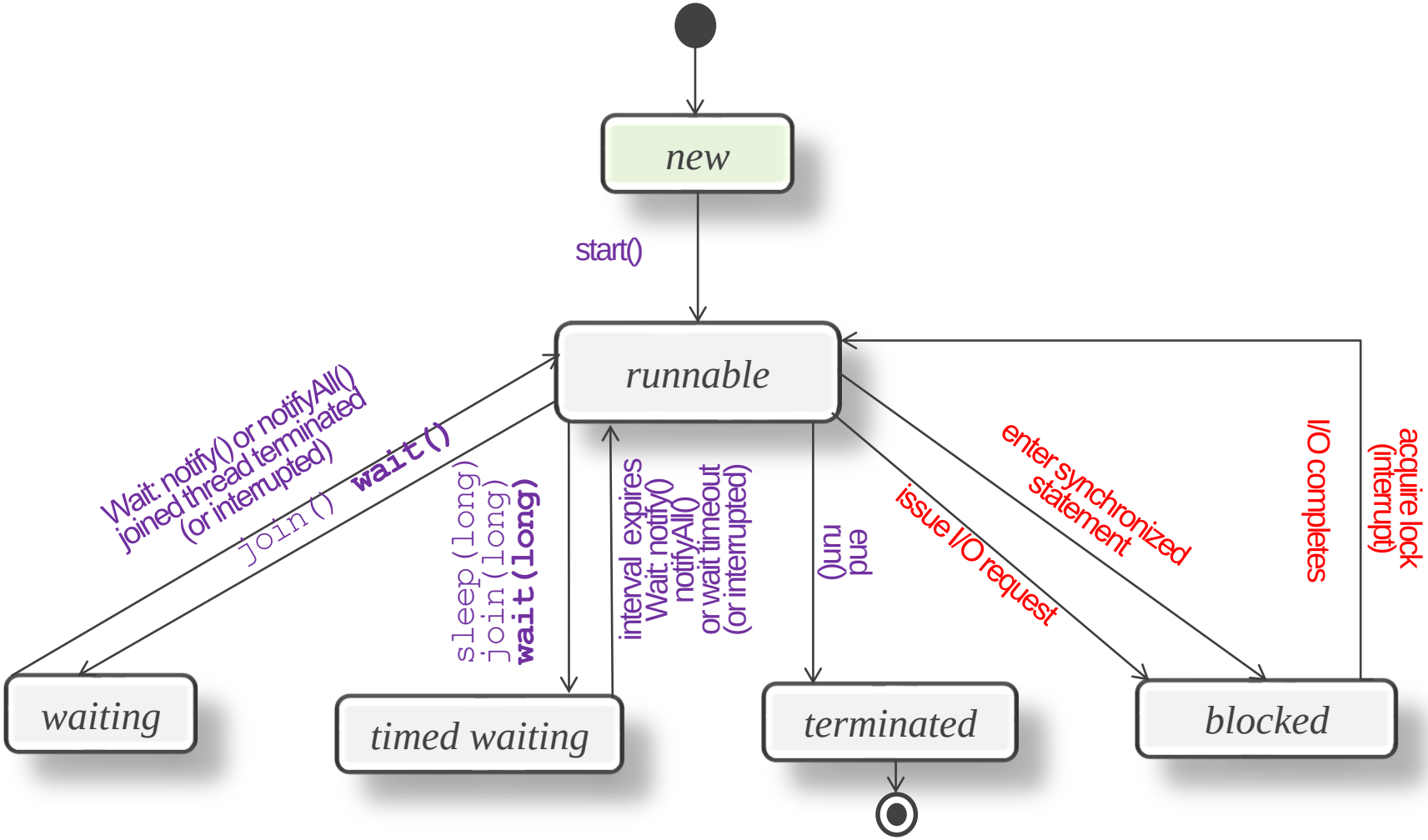
Synchronized methods

- A thread can call the methods (if it is in the monitor of that object – in a block or method synchronized on this object)
  - wait(), wait(long) // Going from Runnable to Wait state
  - // releasing the monitor's lock
  - notify(), notifyAll() // Wake up one or all threads waiting to
    // acquire the monitor's lock

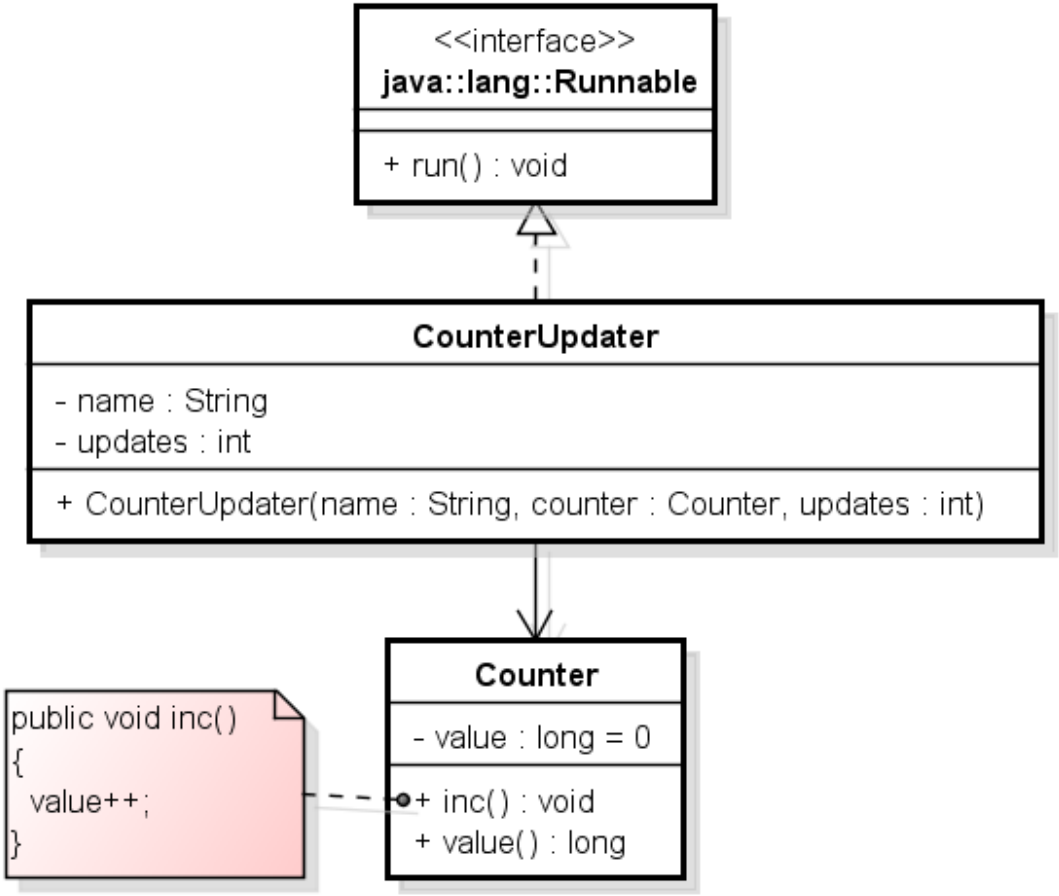
Thread States – wait/notify



Thread States – synchronized



Updating shared variables



Thread safe Counter (Monitor)

```
class Counter
{
  private long value;
  public void inc()
  {
    synchronized(this) // syncronized on the Counter object
    {
      value++;
    }
  }
  public long value()
  {
    synchronized(this)
    {
      return value;
    }
  }
}
```

Thread safe Counter (Monitor)

```
class Counter
{
  private long value;

  public synchronized void inc()
  {
    value++;
  }

  public synchronized long value()
  {
    return value;
  }
}
```

Monitor:
1) All instance variables are private
2) All methods are synchronized

Updating shared variables

```
Run: TestCounter
Main Thread Ended
Updater1 finished: Counter.value = 39699
Updater2 finished: Counter.value = 40000
```

```
Run: TestCounter
Main Thread Ended
Updater2 finished: Counter.value = 26187
Updater1 finished: Counter.value = 40000
```

```
Run: TestCounter
Main Thread Ended
Updater1 finished: Counter.value = 40000
Updater2 finished: Counter.value = 39229
```

```
Run: TestCounter
Main Thread Ended
Updater2 finished: Counter.value = 38382
Updater1 finished: Counter.value = 40000
```

Waiting for a shared object

```
public synchronized void method() throws InterrupterException
{
  if (! conditionToEnterIsOK)
    wait();

  // modify monitor data attributes
  notifyAll();
}
```

```
public synchronized void method() throws InterrupterException
{
  while (! conditionToEnterIsOK)
    wait();

  // modify monitor data attributes
  notifyAll();
}
```

Thread safe Counter (waiting)

```
class Counter
{
  private long value;
  public synchronized void inc()
  {
     while (value > 10)
     {
       try
       {
          wait();
       }
       catch (InterruptedException e)
       {
          //...
       }
     }
     value++;
     notifyAll();
   }
  //...
}
```

A simple thread start-up

Using inner anonymous class

```
new Thread(new Runnable()
{
  @Override public void run()
  {
    System.out.println("Thread started");
    // statements to execute in run method
  }
}).start();
```

Using Lambda expression

```
new Thread(() -> {
   System.out.println("Thread started");
   // statements to execute in run method
}).start();
```

Threads in a JavaFX application

One statement run method (e.g. calling a method)

```
Platform.runLater(() -> executeMethod1());
```

More statements in the run method

```
Platform.runLater(() -> {
 executeMethod1();
 executeMethod2();
});
```

…or as anonymous inner class

```
Platform.runLater(new Runnable()
{
  @Override public void run()
  {
    executeMethod1();
    executeMethod2();
  }
});
```