VIA University College

Software Development with UML and Java 2

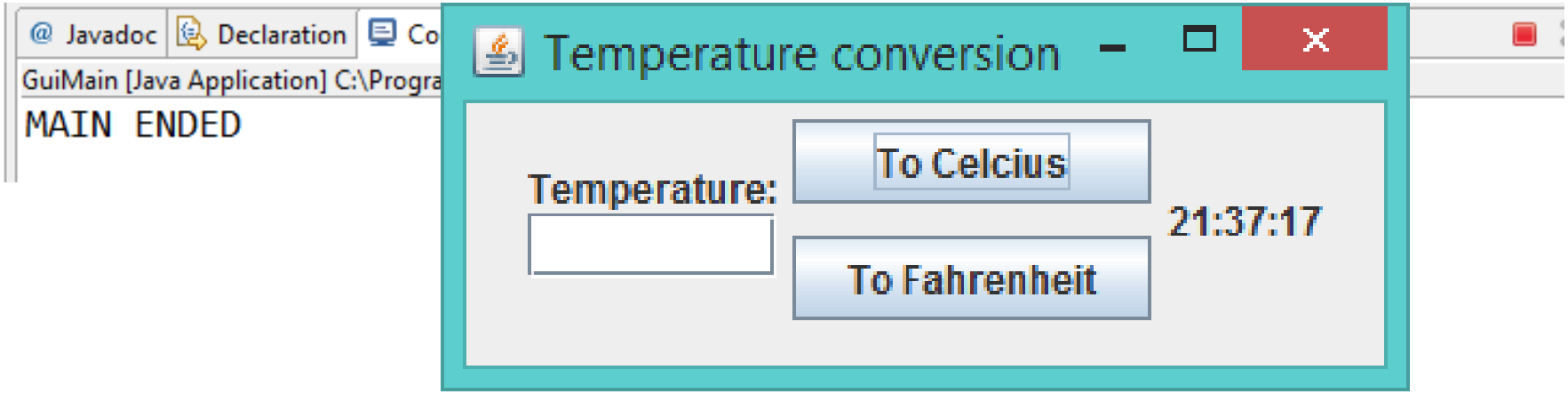
Joseph Okika (jook@via.dk)

Learning Objectives

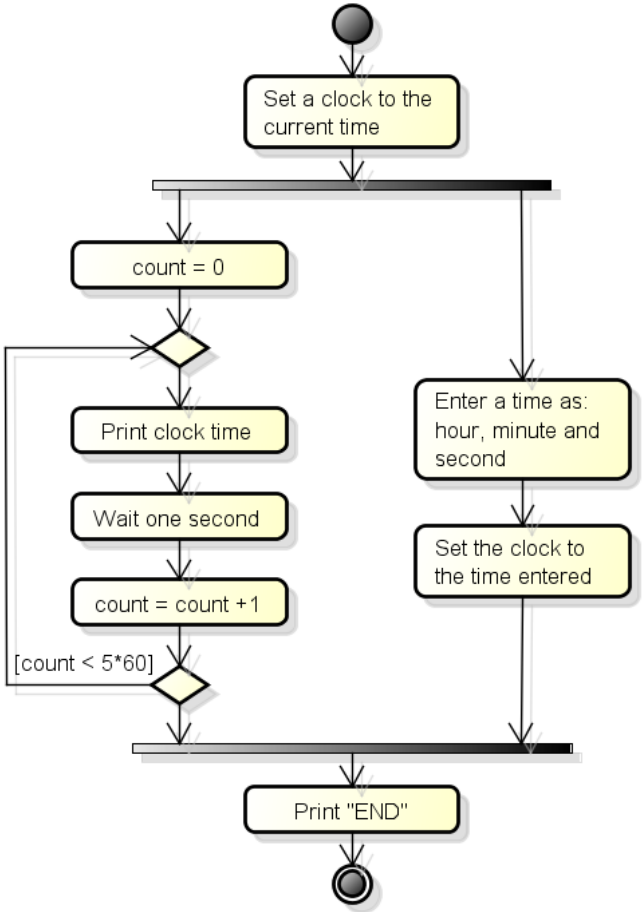
- Understand Java Thread and the different Thread states

A simple GUI

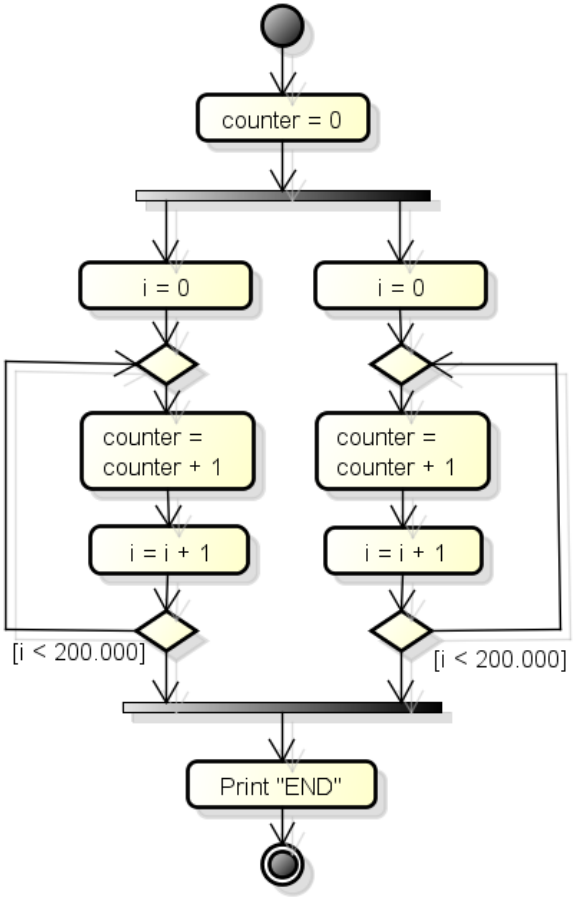
```
public class Main
{
   public static void main(String args[])
   {
      Temperature model = new Temperature();
      Clock clock = new Clock();
      TemperatureView view = new TemperatureView();
      view.startView(model, clock);
      System.out.println("MAIN ENDED");
```



Activities in parallel



Similar activities in parallel



Creating a thread (implements)

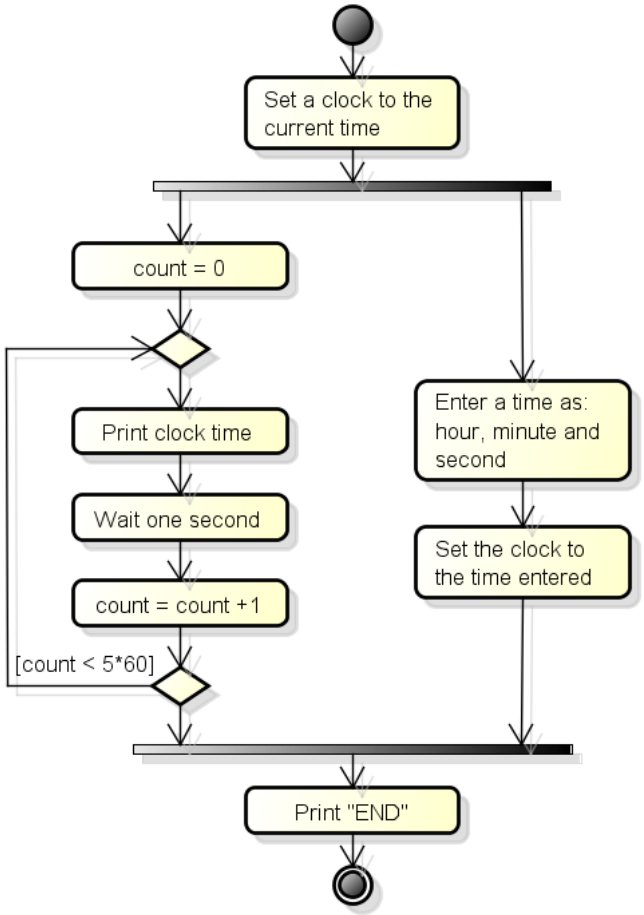
```
public class MyRunnable implements Runnable
{
   //... Instance variables
   MyRunnable(/* parameters */)
   {
      //...
   }
   @Override
   public void run()
   {
      // operations to perform
   }
}
```

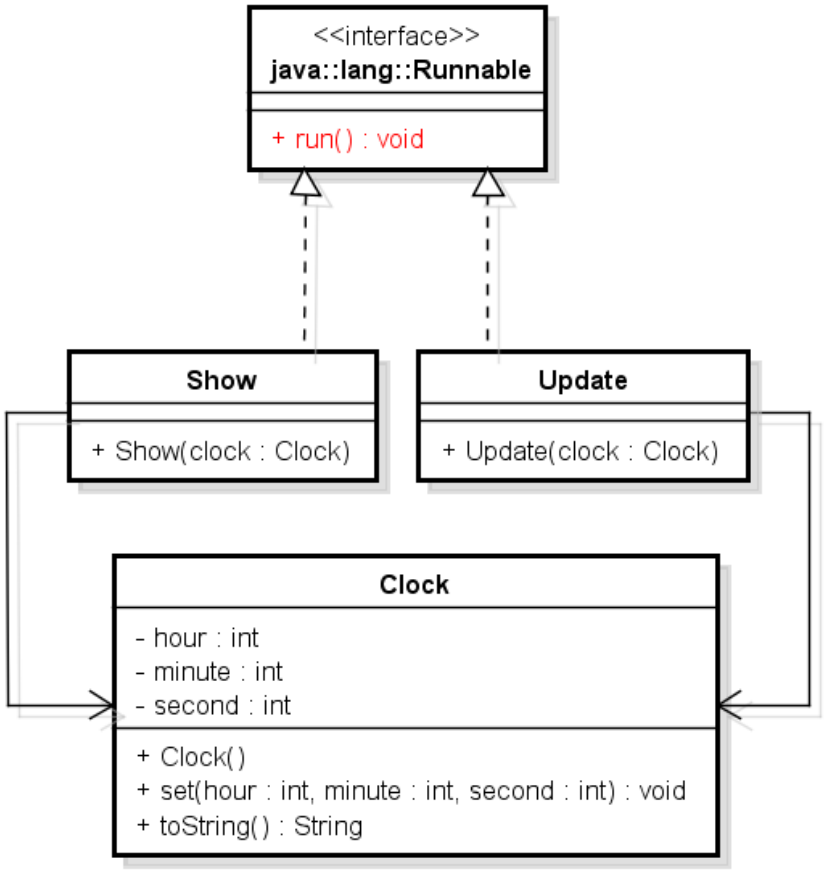
```
// Method starting the thread:
    Runnable myRunnable = new MyRunnable();
    MyThread myThread = new MyThread(myRunnable);
    myThread.start();
```

Creating a thread (extends)

```
public class MyThread extends Thread
{
   //... Instance variables
   MyThread(/* parameters */)
   {
      //...
   }
   @Override
   public void run()
   {
      // operations to perform
   }
}
// Method starting the thread:
    MyThread myThread = new MyThread();
    myThread.start();
```

Clock example





Clock example (Show)

```
public class Show implements Runnable
{
   private Clock clock;

   public Show(Clock clock)
   {
      this.clock = clock;
   }

   public void run()
   {
      for (int i=0; i<5*60; i++)
      {
         System.out.println(clock);
         // and some code to pause for one second
      }
   }
}
```

Clock example (Update)

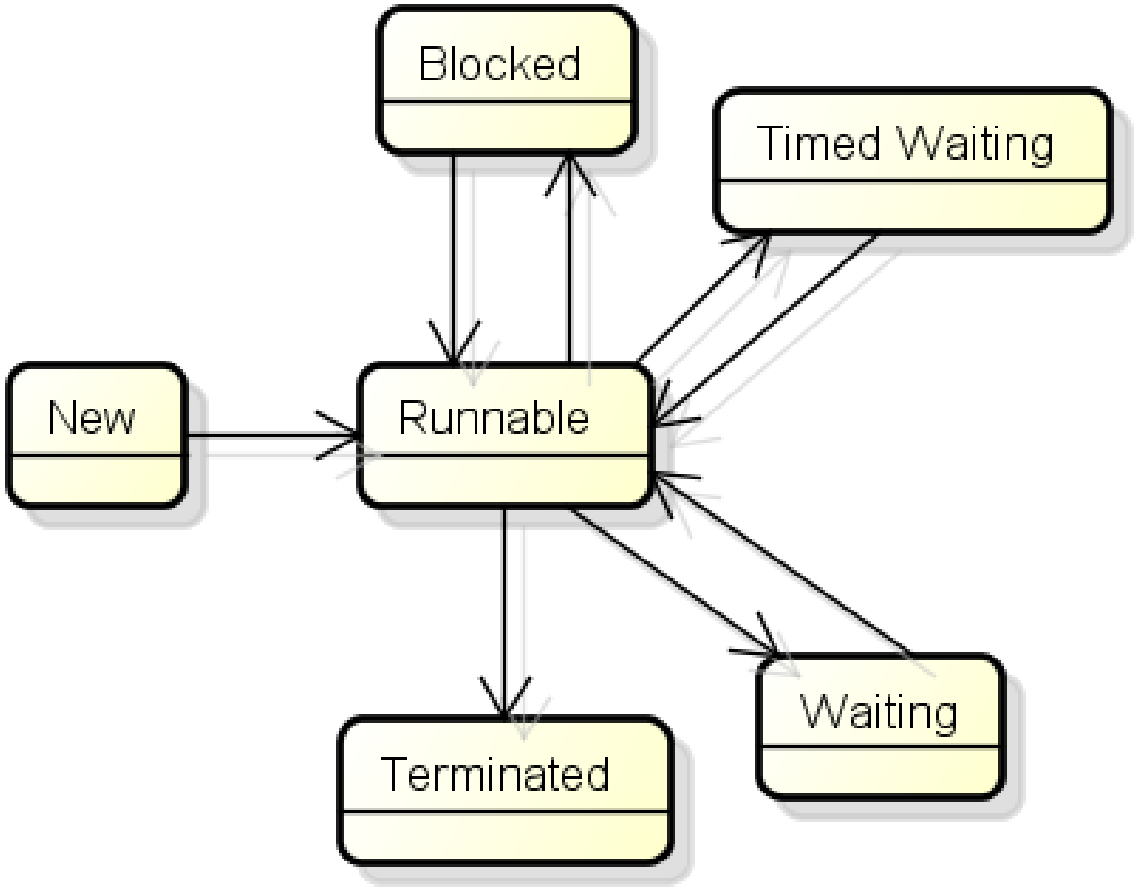
```
public class Update implements Runnable
{
   private Clock clock;

   public Update(Clock clock)
   {
      this.clock = clock;
   }

   public void run()
   {
      Scanner keyboard = new Scanner(System.in);
      int hour = keyboard.nextInt();
      int minute = keyboard.nextInt();
      int second = keyboard.nextInt();
      clock.set(hour, minute, second);
   }
}
```

Clock example (Main method)

```
public class TestClock
{
   public static void main(String[] args)
   {
      Clock clock = new Clock();
      Show showClock = new Show(clock);
      Update updateClock = new Update(clock);

      Thread showClockThread = new Thread(showClock);
      Thread updateClockThread = new Thread(updateClock);

      showClockThread.start();
      updateClockThread.start();

      System.out.println("MAIN ENDED");
   }
}
```

Thread states

- New
- Runnable
- Blocked
- Waiting
- Timed waiting
- Terminated

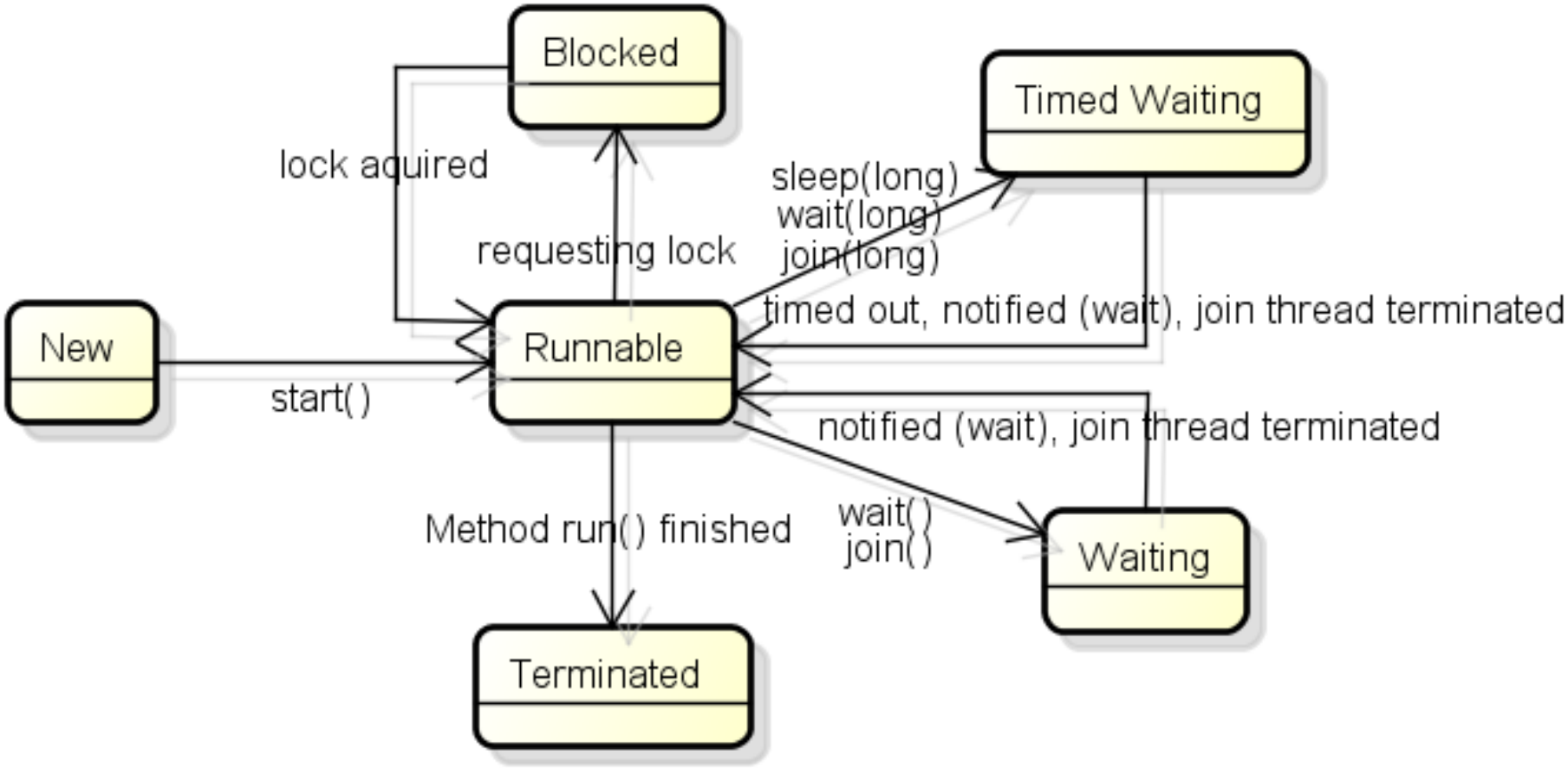


How to pause one second

```
try
{
   Thread.sleep(1000); // sleep for 1000 milliseconds
}
catch (InterruptedException e)
{
   // do nothing
}
```

Going from Runnable State to Timed Waiting State

Thread states (more detailed)



Terminated state (synchronization)

- Pause until a thread is terminated (Wait state)
  - join(), join(long)
- Terminate when another thread terminates
  - setDaemon(true)

Main thread in Wait state (join)

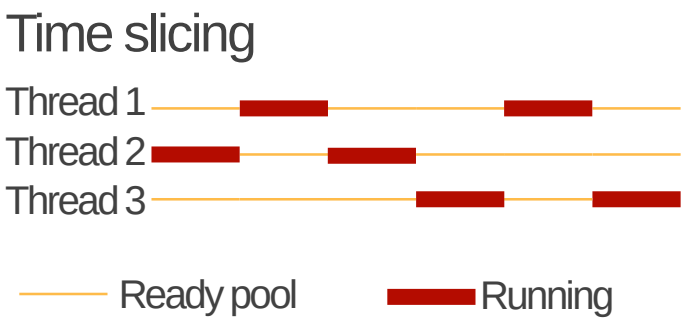
```
public class TestClock // the two clock threads controls when program ends
{
   public static void main(String[] args)
   {
      Clock clock = new Clock();
      Thread showClockThread = new Thread(new Show(clock));
      Thread updateClockThread = new Thread(new Update(clock));

      showClockThread.start();
      updateClockThread.start();

      try
      {
        showClockThread.join();
        updateClockThread.join();
      }
      catch (InterruptedException e) { /* nothing */ }
   }
}
```

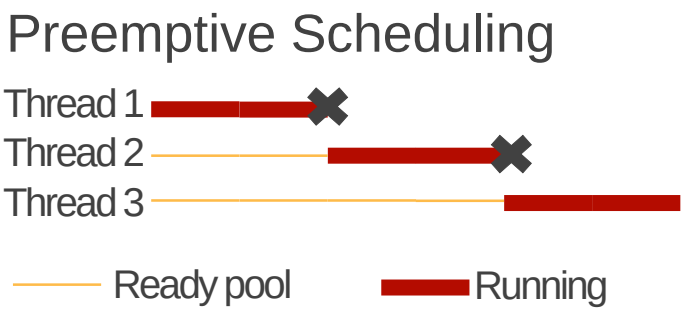
Daemon threads terminated when the Main thread terminates

```
public class TestClock // the Main thread controls when program ends
{
   public static void main(String[] args)
   {
      Clock clock = new Clock();
      Thread showClockThread = new Thread(new Show(clock));
      Thread updateClockThread = new Thread(new Update(clock));

      showClockThread.setDaemon(true);
      updateClockThread.setDaemon(true);

      showClockThread.start();
      updateClockThread.start();

      try
      {
        sleep(5000);
      }
      catch (InterruptedException e) { /* nothing */ }
   }
}
```

Runnable state

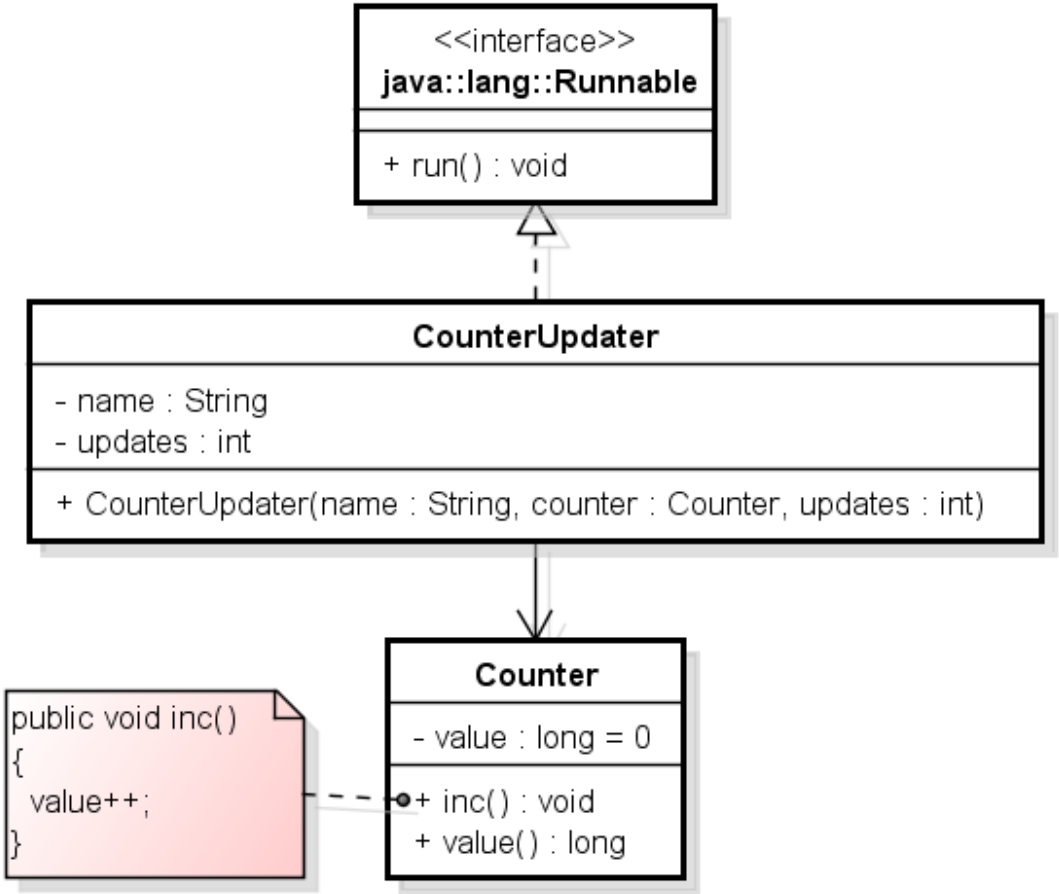
1. Running (scheduled CPU time)
   - Depends on thread priority, OS scheduling algorithm
2. Ready pool (ready for CPU time but not running)





- 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

```
Main Thread Ended
Updater2 finished: Counter.value = 37374
Updater1 finished: Counter.value = 37374

Main Thread Ended
Updater2 finished: Counter.value = 20957
Updater1 finished: Counter.value = 20957

Main Thread Ended
Updater1 finished: Counter.value = 20433
Updater2 finished: Counter.value = 20433

Main Thread Ended
Updater2 finished: Counter.value = 39105
Updater1 finished: Counter.value = 39262
```

Disassembled class file

C:\Windows\system32\cmd.exe

```
C:\_SVA\Work\Workspace\Eclipse_Workspace\AJP-testarea\Thread-Counter\bin>javap -c Counter
Compiled from "Counter.java"
public class Counter {
  public Counter();
    Code:
       0: aload_0
       1: invokespecial #10                 d java/lang/Object."<init>":()V
       4: aload_0
       5: lconst_0
       6: putfield      #12                 l value:J
       9: return

  public void inc();
    Code:
       0: aload_0
       1: dup
       2: getfield      #12                 // Field value:J
       5: lconst_1
       6: ladd
       7: putfield      #12                 // Field value:J
      10: return

  public long value();
    Code:
       0: aload_0
       1: getfield      #12                 // 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: Example

```
C:\_SVA\Work\Workspace\Eclipse
Compiled from "Counter.java"
public class Counter {
  public Counter();
    Code:
       0: aload_0
       1: invokespecial #10
       4: aload_0
       5: lconst_0
       6: putfield      #12
       9: return

  public void inc();
    Code:
       0: aload_0
       1: dup
       2: getfield      #12
       5: lconst_1
       6: ladd
       7: putfield      #12
      10: return

  public long value();
    Code:
       0: aload_0
       1: getfield      #12
       4: lreturn
}
```

Example: value = 10

value=10 → {10}

1 → {10, 1}

add → {11}

value=11 ← { }