Threads – Introduction

ISTE-121

Computational Problem Solving In The Information Domain II

Day 4a

Class Objectives

- Introduction to Threads
- What are Threads
- Java's implementation of threads
  - Thread class
  - Runnable class

Why Use Threads

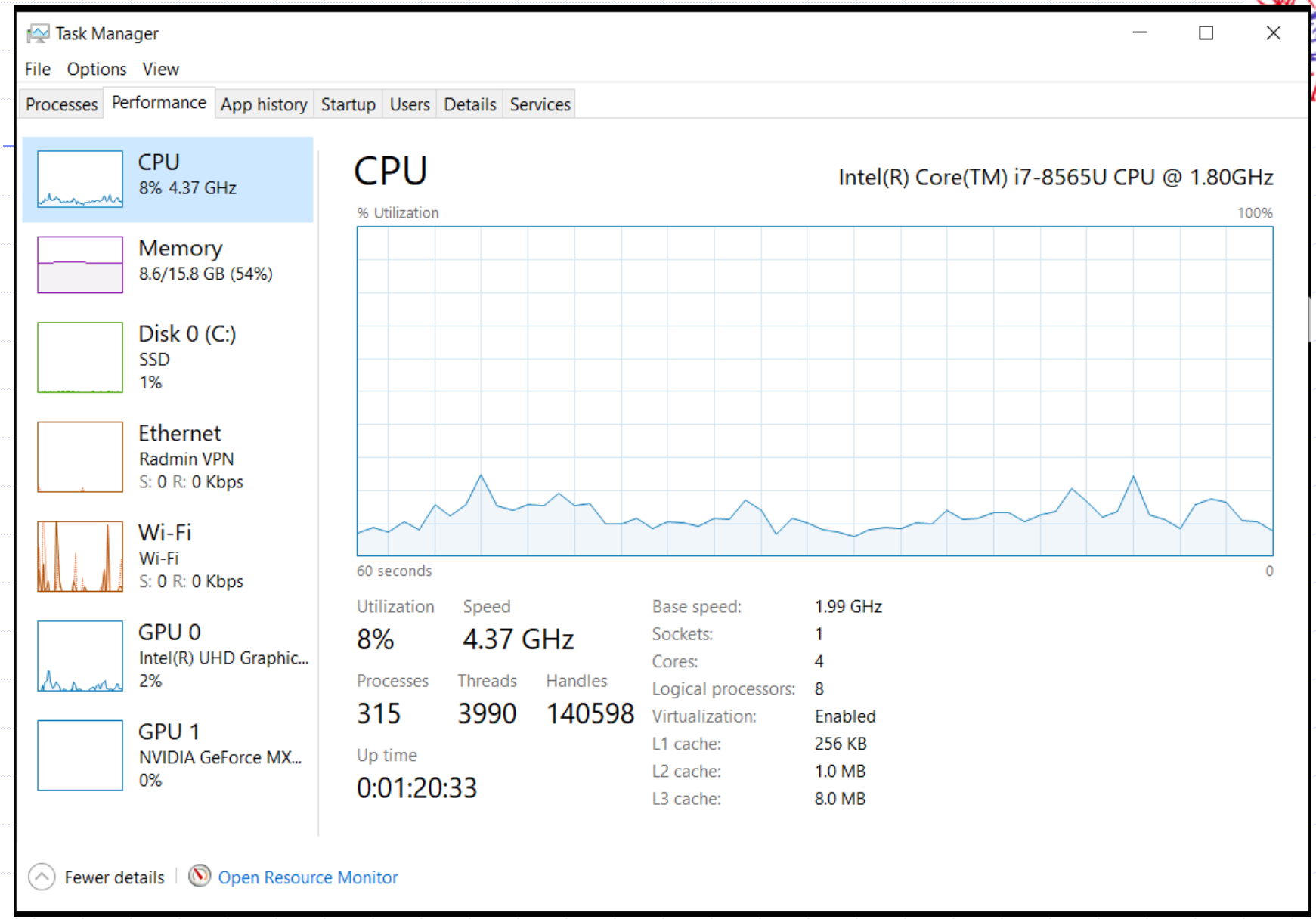
- Provides:
  - Parallelism for multiprocessor machines
  - Better throughput
  - Able to respond to events
  - Allows for multiple communications
  - Better system resource usage
  - Same code can be used for single and multiprocessor machines

Types of Programs to Thread

- Servers
- Independent tasks
- Many repetitive tasks
- Numeric programs
- Items that can run asynchronously

Threads ≠ Processes

- Threads exist **within** a Java process
- Also called "lightweight processes"
- Threads: a program state that gets scheduled onto a processor


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
CPU
8% 4.37 GHz
Memory
8.6/15.8 GB (54%)
Disk 0 (C:)
SSD
1%
Ethernet
Radmin VPN
S: 0 R: 0 Kbps
Wi-Fi
Wi-Fi
S: 0 R: 0 Kbps
GPU 0
Intel(R) UHD Graphic...
2%
GPU 1
NVIDIA GeForce MX...
0%
CPU
Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
% Utilization
100%
60 seconds
0
Utilization
8%
Speed
4.37 GHz
Processes
315
Threads
3990
Handles
140598
Up time
0:01:20:33
Base speed: 1.99 GHz
Sockets: 1
Cores: 4
Logical processors: 8
Virtualization: Enabled
L1 cache: 256 KB
L2 cache: 1.0 MB
L3 cache: 8.0 MB
Fewer details
Open Resource Monitor

Current Java execution begins with a main() method

- main() method

```
public static void
   main(String [] args)
   {
   // A - code start
   // B - more code
   // C - code ends
}
```

- main() runs as:

```
// A

// B

// C
```

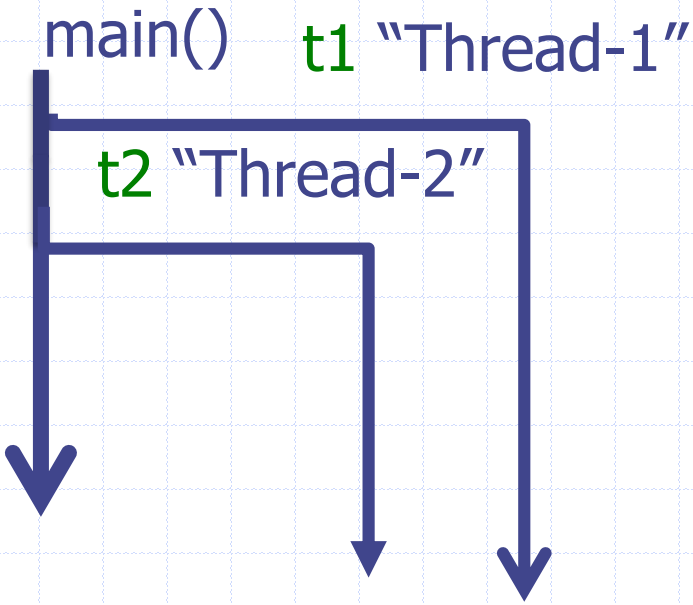
Code runs to end

Threads execution uses run()

```
import java.util.*;
/** QuickThread - demo of Thread class running
threads */
public class QuickThread{
   public static void main(String [] args) {
      MyTh t1 = new MyTh();
        t1.setName(„myThread1");
      MyTh t2 = new MyTh();
        t2.setName(„myThread2");
      t1.start();
      t2.start();
      System.out.println("Main is Done");
   }
}

class MyTh extends Thread {
   public void run() {
      for(int i=0; i<5; i++) {
         System.out.println(getName() + " " + i );
      }
   }
}
```

- Threads runs as:



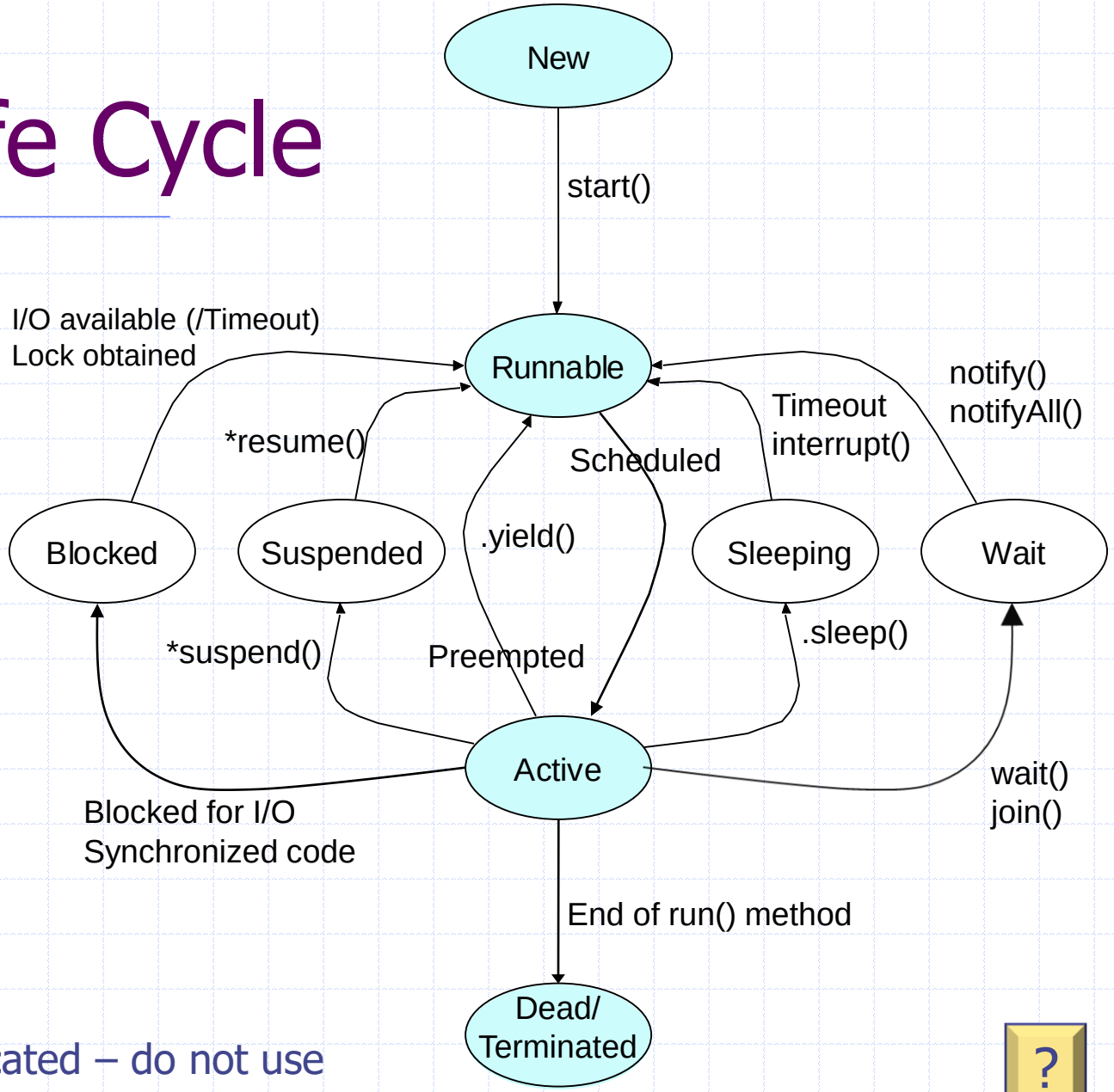
All code runs in parallel

Output of QuickThread

```
Main is Done
    myThread1 0
    myThread1 1
    myThread1 2
    myThread2 0
    myThread2 1
    myThread2 2
    myThread1 3
    myThread1 4
    myThread2 3
    myThread2 4
```

Random print output!!!

Life Cycle



* deprecated – do not use

New Thread

- Thread object is created
- Thread begins with the start() method
- Runs independent of other threads or code
- Does not start executing immediately
- Enters *Runnable* state first

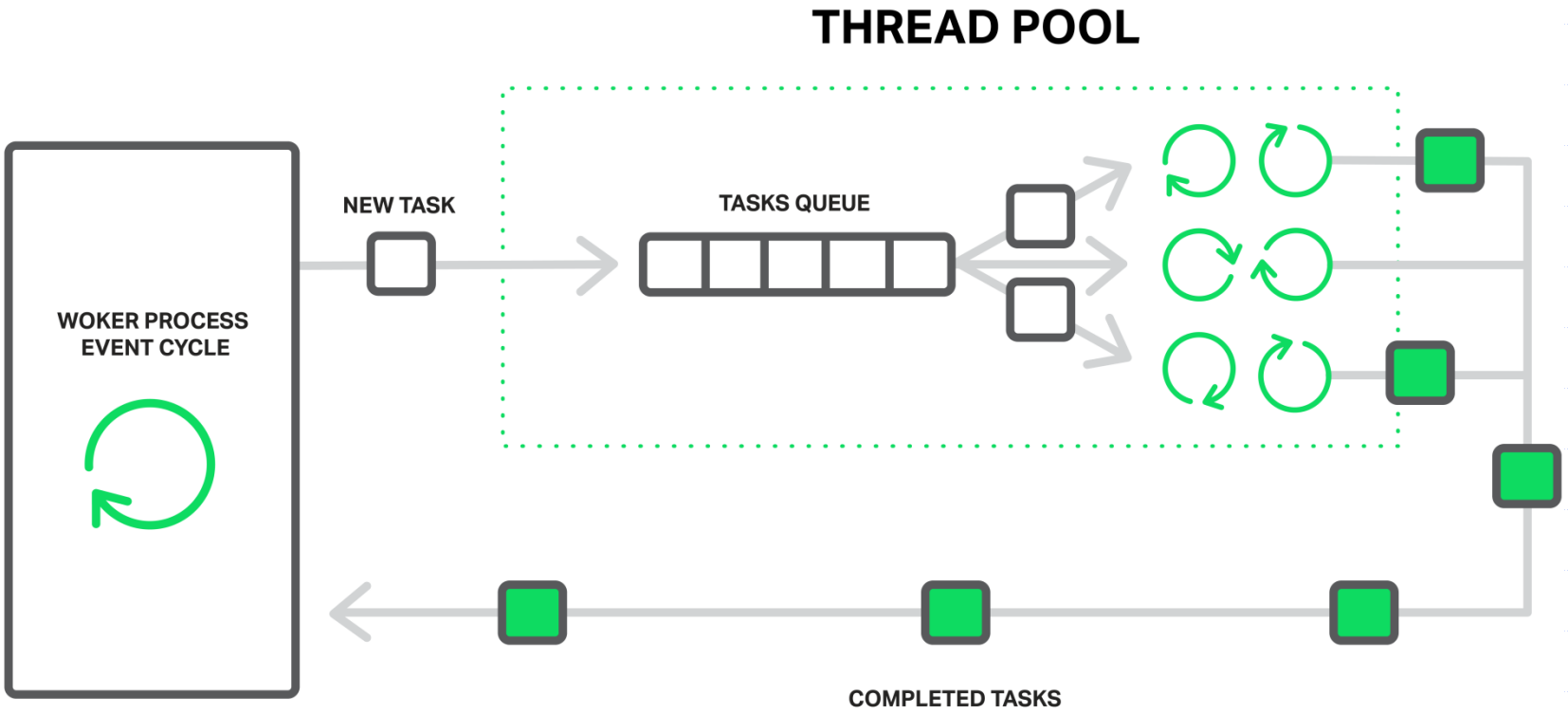
Runnable state

- A pool of threads ready to run
- Not actively running, but ready to run
- Scheduler will select a thread from this set of threads
  - Based on priority or other factors
- The only state a thread can go to from Runnable is to the *Active* state

Active state

- An Active thread has control of a CPU
- Only time a thread can execute code
- Can enter any of these states from here:
  - Blocked
  - Suspended
  - Sleeping
  - Wait
  - Runnable
  - Dead

Thread pool



https://dzone.com/articles/thread-pools-nginx-boost

Thread – summary how to…

1. Write a class that **extends Thread**
2. That class has a **run()** method
3. To begin the thread, execute:
   threadObj**.start()**;
   - Thread's start() calls our run()
4. When run() method ends, thread stops
   - Stops, like end of main() stops the main

Why never call the run() method directly?

class MyClass extends Thread

class MyClass extends Application & Thread ?

- Question: If we need to extend Application how do we extend Thread?

Runnable – how to…

Since we can only extend one class…

When class to be threaded must extend another
class X extends Application **implements Runnable**

- Create a Runnable object

  Runnable myRn = new MyRnble();

- Use object in Thread constructor

  Thread myTh = new Thread(myRn);

- Start new Thread object

  myTh.start();

Runnable - similar example


```
import java.util.*;
public class QuickRunnable{
   public static void main(String [] args) {
      Thread t1 = new Thread( new MyRun() ); // Create Runnable object
      Thread t2 = new Thread( new MyRun() ); // use it in Thread( )
      t1.start();                            // constructor, then
      t2.start();                            // normal Thread usage
      System.out.println("Main is Done");
}}
class MyRun implements Runnable {
   public void run() {
      for(int i=0; i<5; i++) {
         System.out.println( (new Date())+" "+i );
         Thread.yield();              // Need Thread. as not extending Thread
}  }}
```

Output of QuickRunnable

```
----jGRASP exec: java QuickRunnable
Main is Done
Thread2: 0
Thread1: 0
Thread2: 1
Thread1: 1
Thread2: 2
Thread1: 2
Thread2: 3
Thread1: 3
Thread2: 4
Thread1: 4
```

Thread vs. Runnable


```
public MyClassT extends Thread {
  public static void main(String [] args){

    Thread th = new MyClassT ();
    th.start();
  }

  // a separate method
  public void run()
  {
     // code to run as thread
  }
}
```

```
public MyClassR implements Runnable {
  public static void main(String [] args){
    Runnable runbl = new MyClassR ();
    Thread th = new Thread( runbl );
    th.start();
  }

  // a separate method
  public void run()
  {
     // code to run as thread
  }
}
```

Thread & Runnable classes Same as a regular class, plus+

- All functionality of regular class, plus thread execution capability
- Most threads use a constructor
- Save attributes like a regular class
- Contain run() and other methods
- Runnable classes also have all regular class capabilities, plus what they extend (i.e.: Application) plus the Thread capabilities

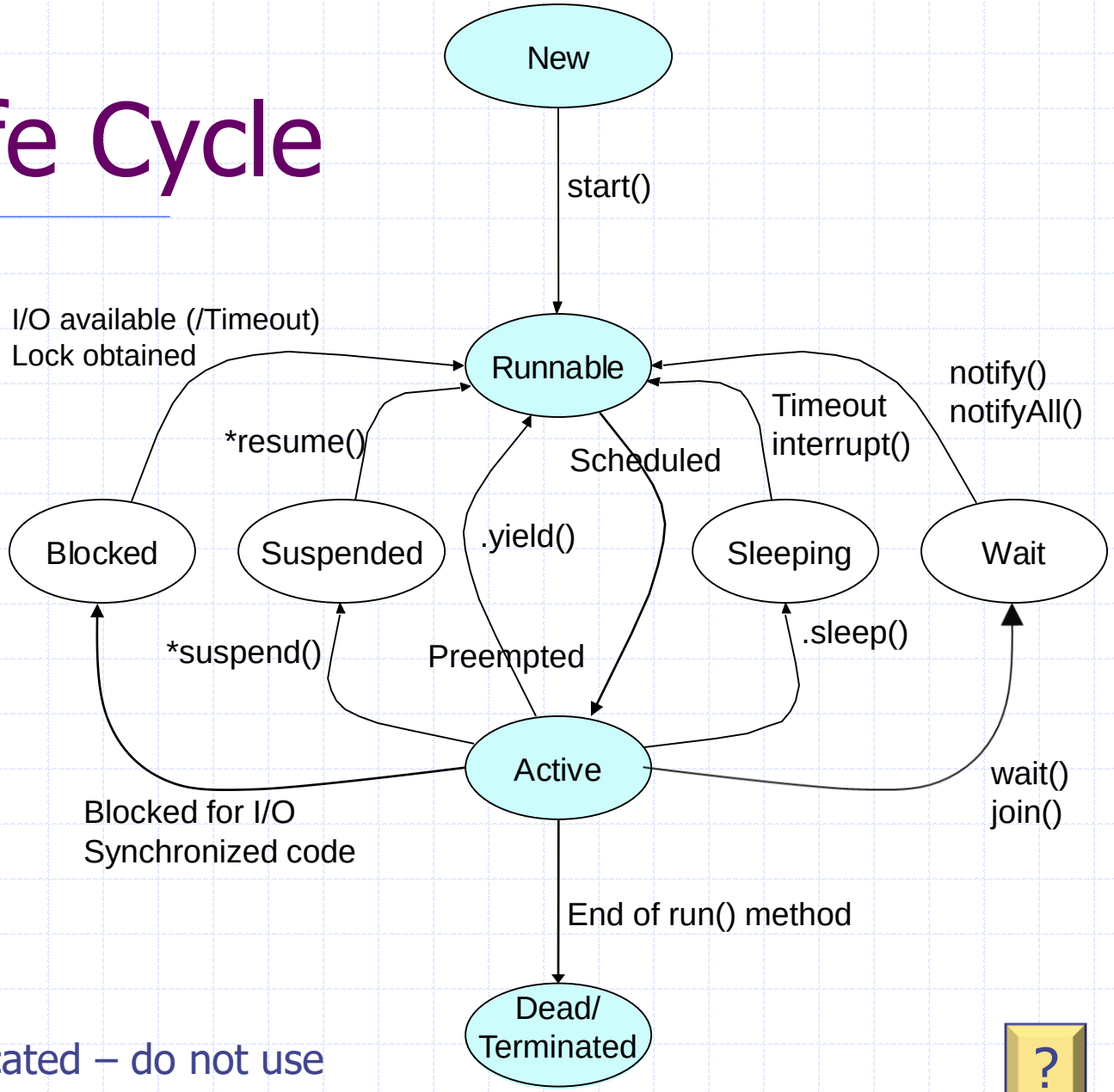
Thread/Runnable-Questions?

- Next topic – Controlling threads

Java code to control threads

- **yield()** – to move current thread out of the CPU and into the Runnable state
- **sleep()** – call to sleep() to stop execution for (at least) a set time.
- **join()** – causes a thread to wait for the completion of other threads
- Demo: Day04Demo.java

Life Cycle



* deprecated – do not use

yield()

threadObject.yield();

- Causes the current thread to ? the use of the CPU
- Thread returns directly to the ? state
- If other threads are waiting, this command gives them a chance at the ?

sleep()

Thread.sleep(numMilliseconds);

- A static thread method
  - Can be called directly **Thread.sleep(ms)**;
- ? it won't be interrupted – need to catch **InterruptedException**
- ? it will return to CPU in time specified - could be out of CPU longer
- Gives up CPU control while sleeping regardless of priority

join() – Wait for thread to end

threadObject.join();

- Another thread waits for this thread to die
- Usually used in main to wait for threads to complete
- No guarantee it won't get interrupted – need to catch **InterruptedException**
- Also with timeout: join(timeoutMs);