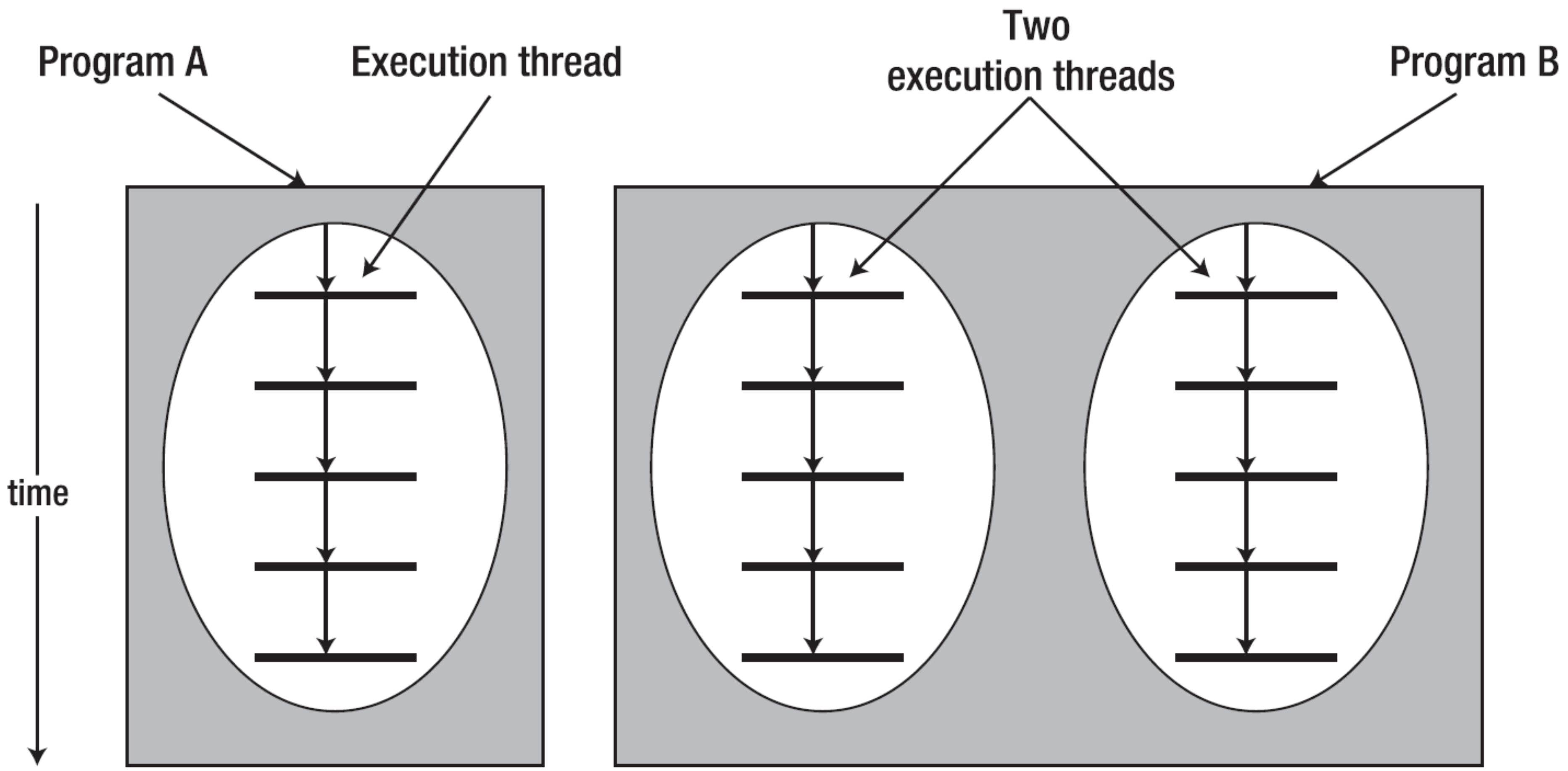
Java Programming

Chapter 11 – Threads in Java

Understanding Threads



Understanding Threads

- The support for threads is provided by:
  - The `java.lang.Thread` class
  - The `java.lang.Object` class
  - The `java.lang.Runnable` interface
- A non-multithreaded program has one thread of execution, called the *main thread*
- You can spawn other threads in addition to the main thread.
- You can write a thread class in one of two ways:
  - Extend the `java.lang.Thread` class
  - Implement the `Runnable` interface

Creating a Thread Using the Thread Class

1. Define the thread by writing your class that extends the `Thread` class, and by overriding its `run()` method in your class.
2. Instantiate the thread by instantiating your class, for example, inside a method of another class.
3. Start the thread by executing the `start()` method that your class inherited from the `Thread` class.

Listing 11-1. *ThreadTest.java*

```
public class ThreadTest {
 public static void main(String[] args) {
      Counter ct = new Counter();
      ct.start();
      System.out.println("The thread has been started");
   }
}
class Counter extends Thread {
  public void run() {
    for ( int i=1; i<=5; i++) {
       System.out.println("Count: " + i);
    }
  }
 }
```

Creating a Thread Using the Runnable Interface

- If your thread class already extends another class, it cannot extend the `Thread` class because Java supports only single inheritance.
- Using the `Runnable` interface:

1. Write your class that implements the `run()` method of the `Runnable` interface.
2. Instantiate your class.
3. Make an object of the `Thread` class by passing your class instance in the argument of the `Thread` constructor.
4. Start the thread by invoking the `start()` method on your `Thread` object.

Listing 11-2. *RunnableTest.java*

```
public class RunnableTest {
 public static void main(String[] args) {
      RunCounter rct = new RunCounter();
      Thread th = new Thread(rct);
      th.start();
      System.out.println("The thread has been started");
   }
}
class RunCounter extends Nothing implements Runnable {
  public void run() {
    for ( int i=1; i<=5; i++) {
     System.out.println("Count: " + i);
    }
 }
}
 class Nothing {
 }
```

Constructors of the Thread Class

- Thread()
- Thread(Runnable target)
- Thread(String name)
- Thread(Runnable target, String name)
- Thread(ThreadGroup group, String name)
- Thread(ThreadGroup group, Runnable target)
- Thread(ThreadGroup group, Runnable target, String name)
- Thread(ThreadGroup group, Runnable target, String name, long stackSize)

Spawning Multiple Threads

Listing 11-3. *MultipleThreads.java*

```java
public class MultipleThreads {
 public static void main(String[] args) {
      System.out.println("The main thread of execution started");
      RunCounter rct1 = new RunCounter("First Thread");
      RunCounter rct2 = new RunCounter("Second Thread");
      RunCounter rct3 = new RunCounter("Third Thread");
 }
}
class RunCounter implements Runnable {
     Thread myThread;
  RunCounter(String name) {
     myThread = new Thread(this, name);
     myThread.start();
  }
  public void run() {
    for ( int i=1; i<=5; i++) {
     System.out.println("Thread: " + myThread.getName() + " Count: " + i);
     }
   }
 }
```

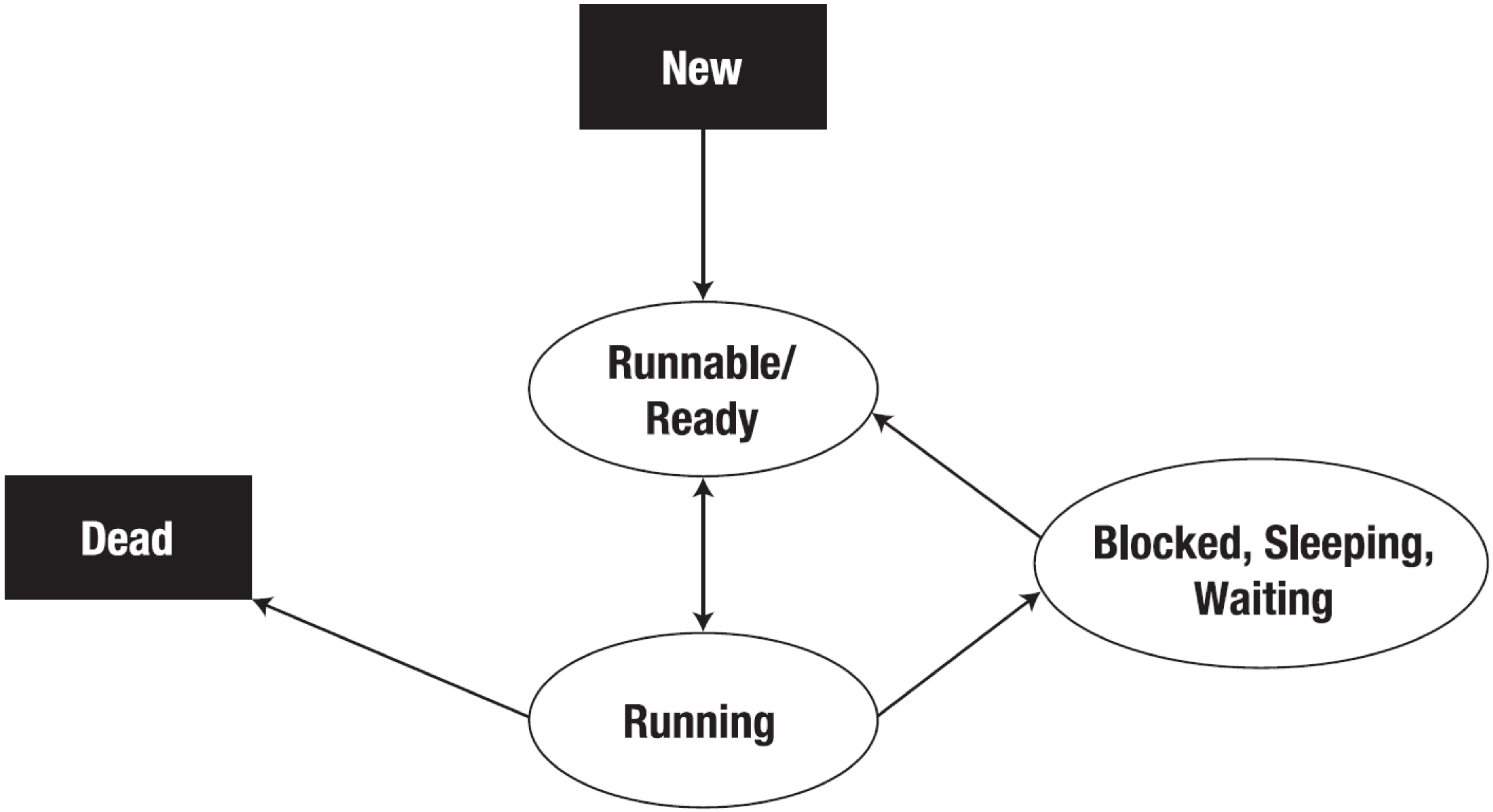
Lifecycle of a Thread: An Overview

- A thread's life starts when the method `start()` is invoked on it.
- It goes through various states before it finishes its task and is considered dead.
- Upon calling the `start()` method, the thread does not start running immediately.
- The `start()` method puts it into the runnable state (ready state).
- It stays in the runnable state until the scheduler puts it into the running state (`run()` is called).

States of a Thread

- *New:* has been instantiated but not yet started.
- *Ready/runnable:* when the `start()` method is invoked. Later, the thread can come back to this state from one of the nonrunnable states.
- *Running*: the thread is executing.
- *Nonrunnable*:
  - *Blocked:* waiting for a resource
  - *Sleeping*: by calling the `sleep()` method
  - *Waiting*: invokes the `wait()` method
- *Dead*: after the execution of its run() method is complete

Relationship between Different States of a Thread



Transition Between Running and Runnable States

- When the `start()` method is called, it puts the thread into the *runnable* state.
- The scheduler eventually transitions the thread from the *runnable* state into the *running* state.
- A call to the `yield()` method in the thread code puts the thread back into the runnable state:

```
Thread.yield();
```

Transition Between Runnable and Nonrunnable States

- Sleeping State:
  - A thread is put into the sleeping state by a call to the `sleep()` method in the thread code
  - After the sleep time expires, the thread goes into the runnable state, and eventually is put back into the running state by the scheduler.
- Blocked State:
  - Sometimes a method has to wait for some event to happen before it can finish.

```
while ((len = in.read(buf)) != -1)
{
  outFile.write(buf, 0, len);
}
```

Transition Between Runnable and Nonrunnable States (cont.)

- Waiting State:
  - Several threads executing concurrently may share a piece of code.
  - To synchronize a few things, you may put a thread into the waiting state by making a call to the `wait()` method in the shared code.
  - A thread in the wait state is brought out of this state by a call to the `notify()` method or the `notifyAll()` method.

```
Thread myThread = new Thread();
myThread.start();
try{
  myThread.join();
}catch (Exception ex){}
```

Understanding the Concurrent Access Problem

- Two threads may execute the same piece of code concurrently

```
public class Tracker {
        private int counter=0;
        public int nextCounter() {
                return counter++;
        }
}
```

- Two threads, thread1 and thread2, try to execute this method concurrently:
  - 1. thread1 reads counter = 0.
  - 2. thread2 reads counter = 0.
  - 3. thread1 adds one to the counter and sets counter = 1.
  - 4. thread2 adds one to the counter and sets counter = 1.

Object Locks

```
public class Tracker {
private int counter=0;
public synchronized int nextCounter() {
    return counter++;
}
}
```

```
public class Tracker {
private int counter=0;
public int nextCounter() {
   synchronized(this) {
    return counter++;
  }
}
}
```