**Java**

***Thread***

Multitasking

* Multitasking allows several activities to occur concurrently on the computer
* Levels of multitasking:

– **Process‐based multitasking**

* + - Allows programs (processes) to run concurrently

– **Thread‐base multitasking (multithreading)**

* + - Allows parts of the same process (threads) to run concurrently

Prepared By - Rifat Shahriyar 2

Multithreading

* Advantages of multithreading over process-based multitasking

– Threads share the same address space

– Context switching between threads is usually inexpensive

– Communication between thread is usually inexpensive

* Java supports ***thread‐based multitasking*** and provides high-level facilities for ***multithreaded*** ***programming***

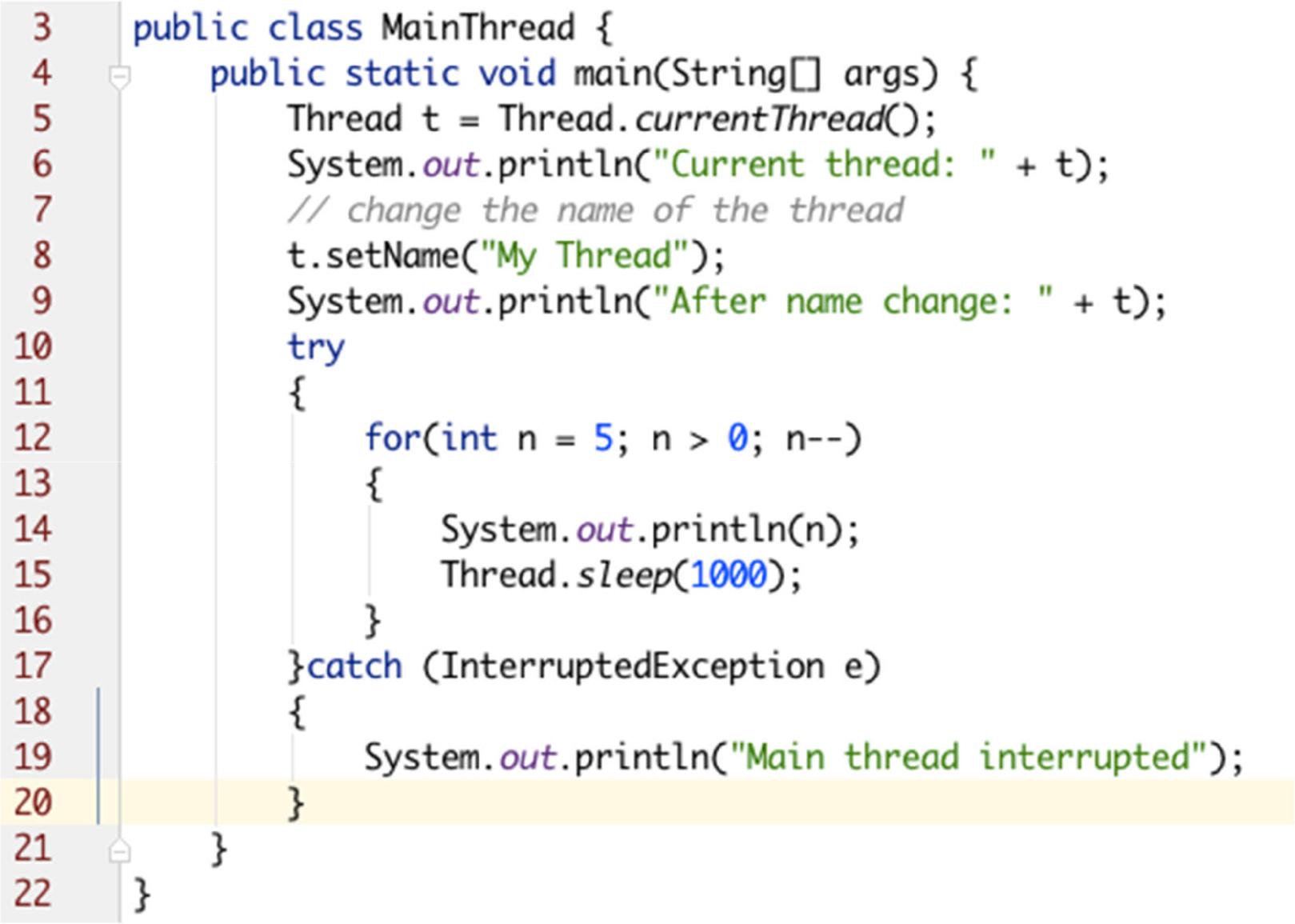
Prepared By - Rifat Shahriyar 3

Main Thread

* When a Java program starts up, one thread begins running immediately
* This is called the ***main thread*** of the program
* It is the thread from which the child threads will be spawned
* Often, it must be the last thread to finish execution

Prepared By - Rifat Shahriyar 4

Main Thread



Prepared By - Rifat Shahriyar 5

How to create Thread

1. By extending the ***Thread*** class
2. By implementing ***Runnable*** Interface

* *Extending Thread*

– Need to override the public void run() method

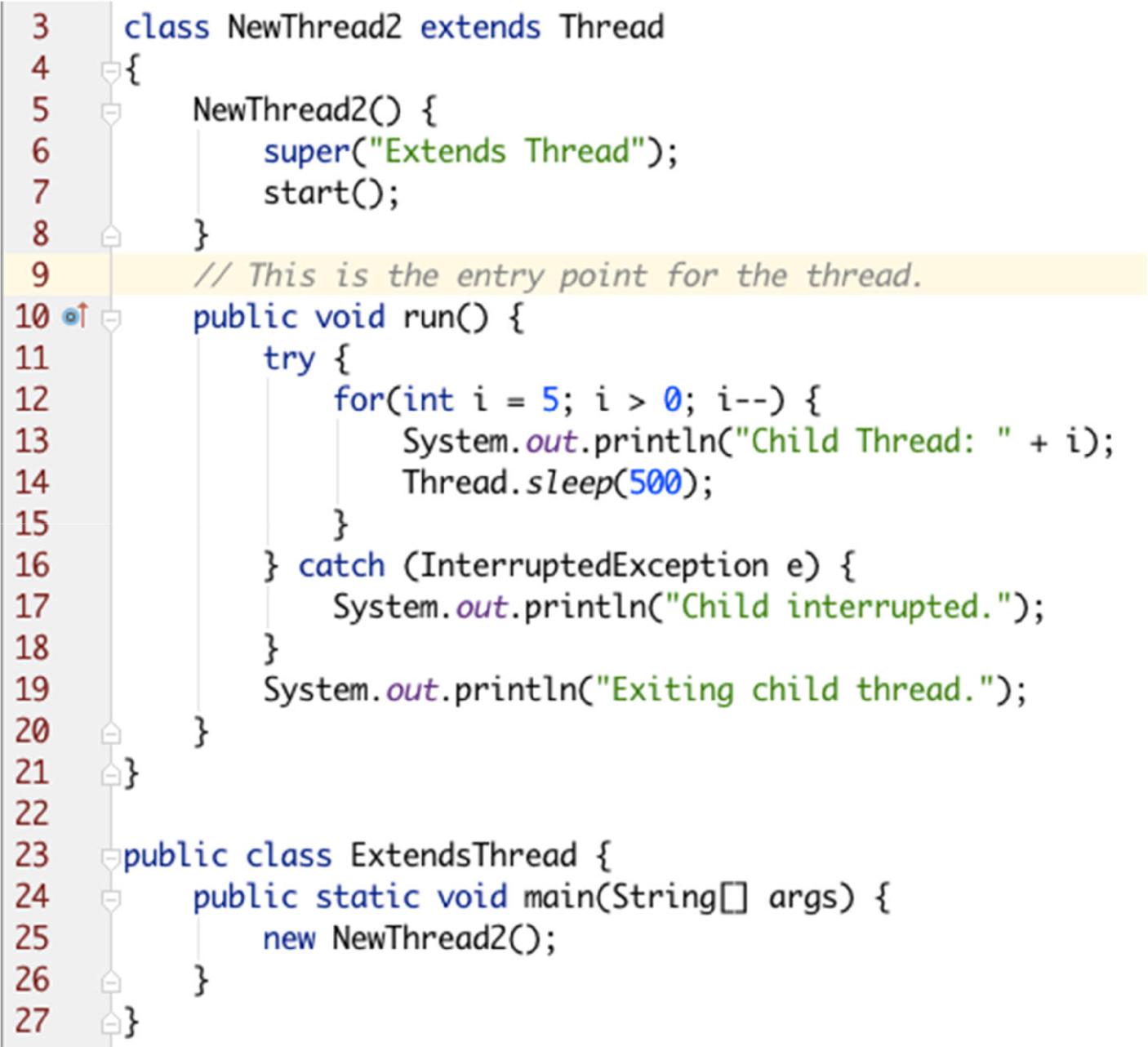
* *Implementing Runnable*

– Need to implement the public void run() method

* Which one is better ?

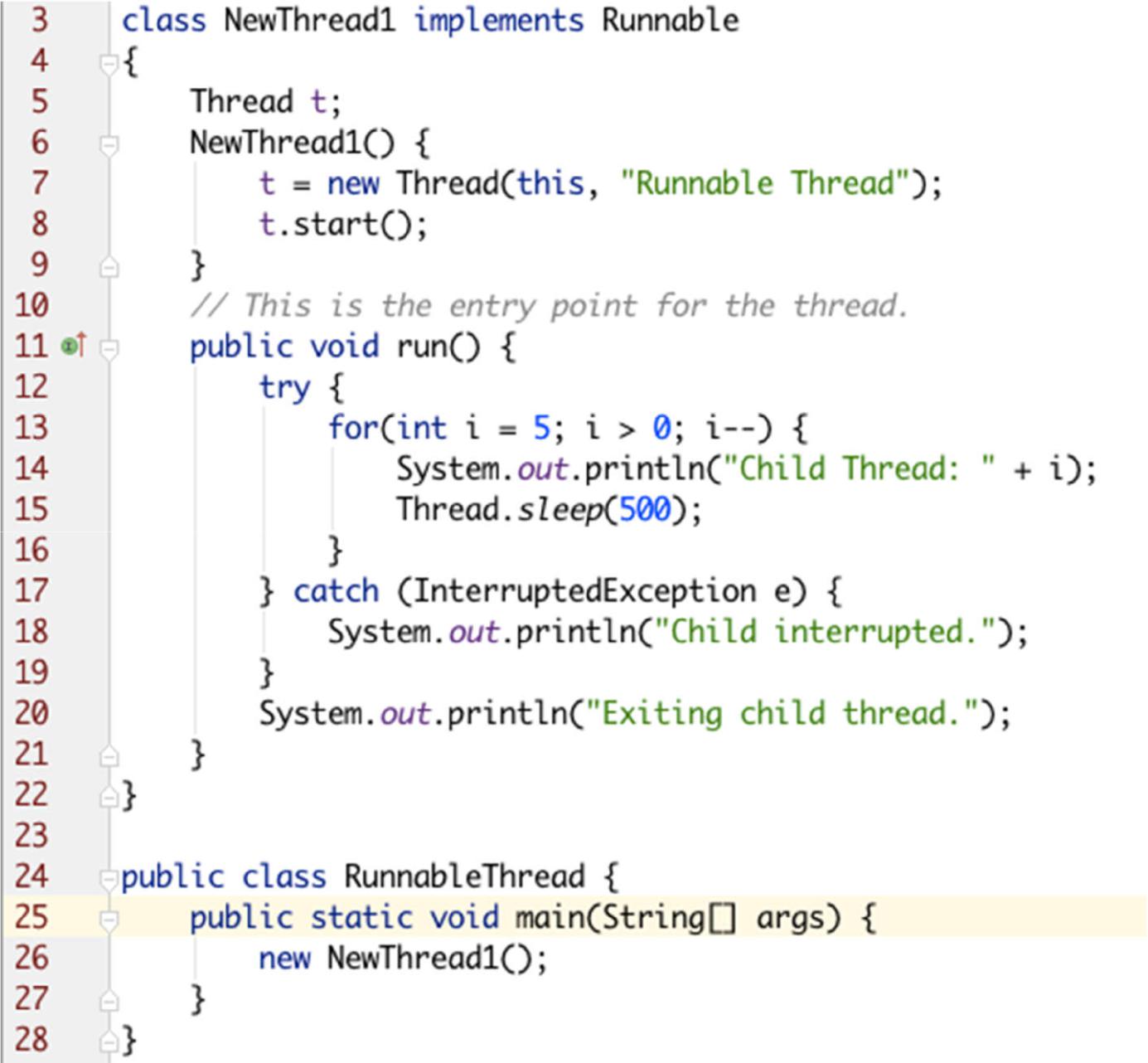
Prepared By - Rifat Shahriyar 6

Extending Thread



Prepared By - Rifat Shahriyar 7

Implementing Runnable



Prepared By - Rifat Shahriyar 8

Multiple Threads

* It is possible to create more than one thread inside the main
* In multiple threads, often you will want the main thread to finish last. This is accomplished by

– using a large delay in the main thread

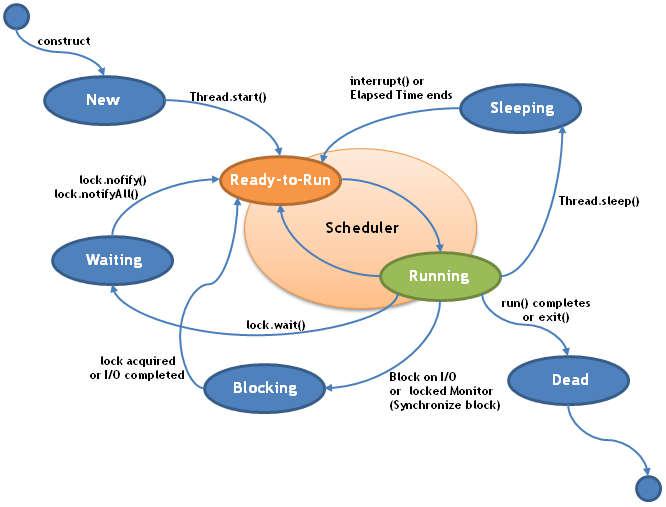
– using the **join()** method

* Whether a thread has finished or not can be known using **isAlive()** method
* ***Example****: MultipleThreads.java, JoinAliveThreads.java*

Prepared By - Rifat Shahriyar 9

Thread States

Source: https://avaldes.com/java-thread-states-life-cycle-of-java-threads/



Prepared By - Rifat Shahriyar 10

Thread Pool

* Thread Pools are useful when you need to limit the number of threads running in your application

– Performance overhead starting a new thread

– Each thread is also allocated some memory for its stack

* Instead of starting a new thread for every task to execute concurrently, the task can be passed to a thread pool

– As soon as the pool has any idle threads the task is assigned to one of them and executed

Prepared By - Rifat Shahriyar 11

Thread Pool

* Thread pools are often used in multi threaded servers

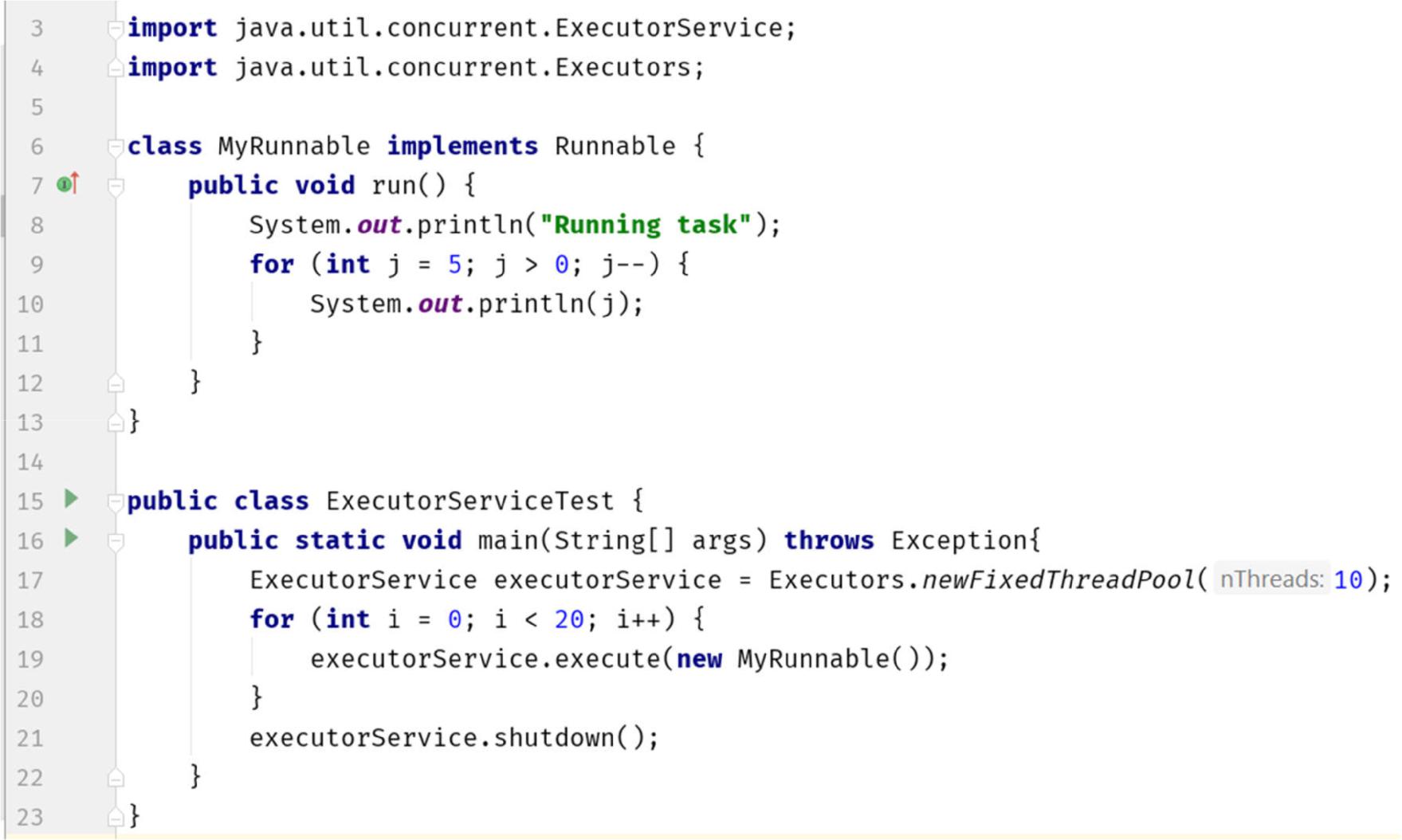
– Each connection arriving at the server via the network is wrapped as a task and passed on to a thread pool

– The threads in the thread pool will process the requests on the connections concurrently

* Java provides Thread Pool implementation with ***java.util.concurrent.ExecutorService***

Prepared By - Rifat Shahriyar 12

ExecutorService



Prepared By - Rifat Shahriyar 13

Callable and Future

* Runnable cannot return a result to the caller
* ***java.util.concurrent.Callable*** object allows to returnvalues after completion
* Callable task returns a Future object to return result
* The result can be obtained using get() that remains blocked until the result is computed
* Check completion by isDone(), cancel by cancel()
* ***Example****: CallableFutures.java*

Prepared By - Rifat Shahriyar 14

Synchronization

* When two or more threads need access to a **shared** **resource**, they need some way to ensure that theresource will be used by only one thread at a time
* The process by which this is achieved is called **synchronization**
* Key to synchronization is the concept of the **monitor**
* A monitor is an object that is used as a mutually exclusive lock

– Only one thread can own a monitor at a given time

Prepared By - Rifat Shahriyar 15

Synchronization

* When a thread acquires a lock, it is said to have entered the monitor
* All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor
* These other threads are said to be waiting for the monitor

Prepared By - Rifat Shahriyar 16

Synchronization

* Three ways to achieve synchronization.
* Synchronized method

***synchronized void call(String msg) { }***

* Synchronized block ***public void run() {***

***synchronized(target) { target.call(msg); } }***

* Lock (java.util.concurrent package)
* ***Example****: SynchronizedBlock.java, SynchronizedMethod.java,*

*SynchronizationLock.java*

Prepared By - Rifat Shahriyar 17

Inter Thread Communication

* One way is to use polling

– a loop that is used to check some condition repeatedly

– Once the condition is true, appropriate action is taken

* Java includes an elegant inter thread communication mechanism via the **wait()**, **notify()** and **notifyAll()** methods
* These methods are implemented as final methods in Object, so all classes have them
* All three methods can be called only from within a synchronized method

Prepared By - Rifat Shahriyar 18

Inter Thread Communication

* ***wait()***

– tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify()

* ***notify()***

– wakes up the first thread that called wait() on same object

* ***notifyAll()***

– wakes up all the threads that called wait() on same object. The highest priority thread will run first

* ***Example****: IncorrectPC.java, CorrectPC.java,**PCBlockingQueue.java*

Prepared By - Rifat Shahriyar 19

Suspend, Resume and Stop

* Suspend

– ***Thread t; t.suspend();***

* Resume

– ***Thread t; t.resume();***

* Stop

– ***Thread t; t.stop();***

– Cannot be resumed later

* suspend and stop can sometimes cause serious system failures
* ***Example****: SuspendResume.java*

Prepared By - Rifat Shahriyar 20