Concurrency in Java

How to create Thread

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

- *Implementing Runnable*
  - Need to implement the public void run() method
- *Extending Thread*
  - Need to override the public void run() method
- Which one is better ?

Runnable Interface

- Better than extending Thread
- Multiple threads can share a single runnable implementation
- ***Example**: Workers.java*

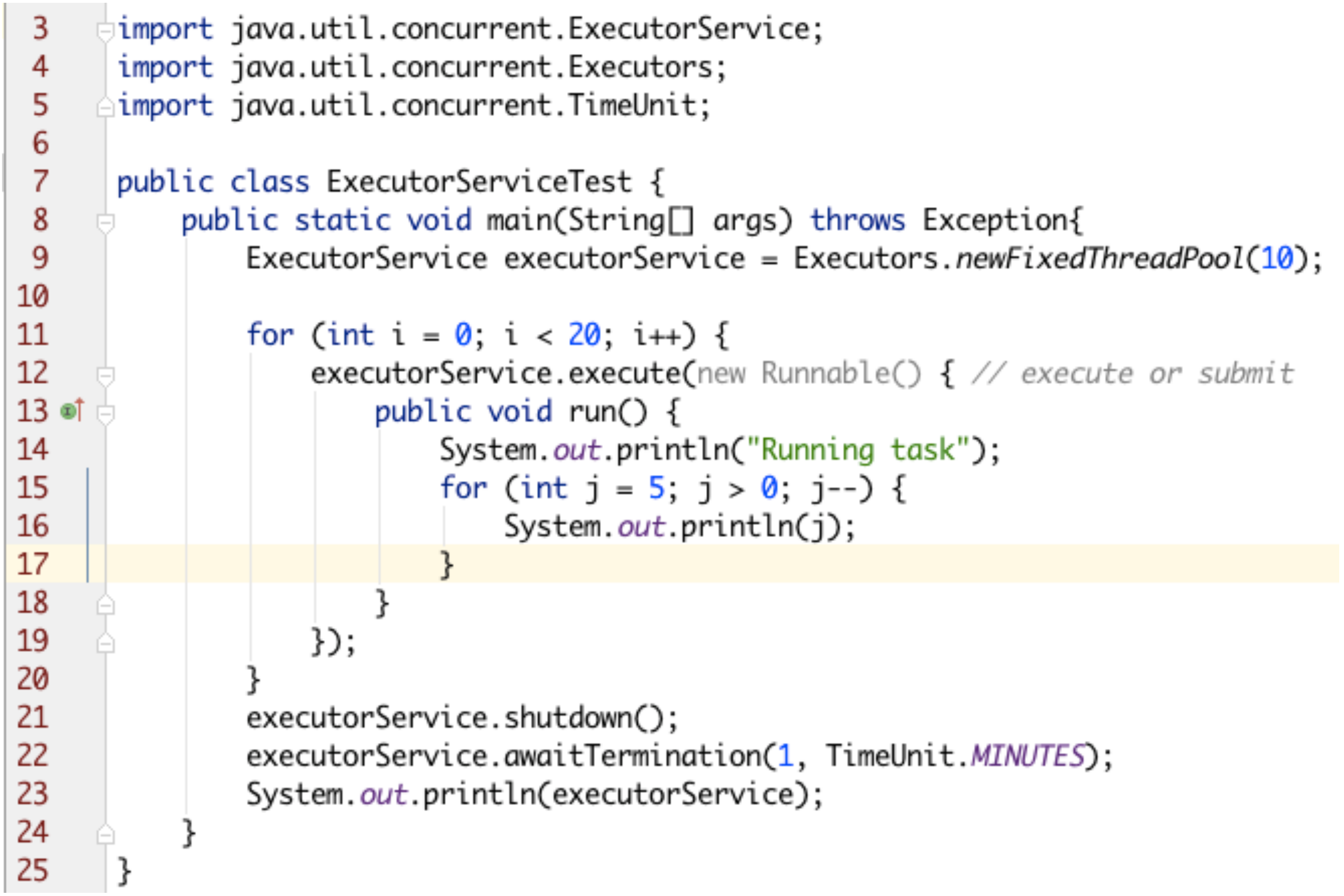
Thread Pool

- 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

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***

ExecutorService



```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ExecutorServiceTest {
    public static void main(String[] args) throws Exception{
        ExecutorService executorService = Executors.newFixedThreadPool(10);

        for (int i = 0; i < 20; i++) {
            executorService.execute(new Runnable() { // execute or submit
                public void run() {
                    System.out.println("Running task");
                    for (int j = 5; j > 0; j--) {
                        System.out.println(j);
                    }
                }
            });
        }
        executorService.shutdown();
        executorService.awaitTermination(1, TimeUnit.MINUTES);
        System.out.println(executorService);
    }
}
```

Synchronization

- When two or more threads need access to a **shared resource**, they need some way to ensure that the resource 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

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

Synchronization

- Two way to achieve synchronization.
- Synchronized method

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

- Synchronized block

```
public void run() {
        synchronized(target) { target.call(msg); }
}
```

Lock

- Introduced in java.util.concurrent
- Better and flexible locking support
- ***Example****: SynchronizationLock.java*

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

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

BlockingQueue

- Introduced in java.util.concurrent
- Easy solution for Producer Consumer problem
- ***Example****: PCBlockingQueue.java*

AtomicLong

- Introduced in java.util.concurrent
- Can be used for atomic counter instead of
  - Synchronized method
  - Synchronized block
  - Lock
- ***Example****: AtomicCounterTest.java, AtomicCounter.java*