Build COMPETENCY

across your TEAM

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Multi Threading and synchronization

Google Cloud Platform

Windows Azure

amazon web services

APACHE kafka A distributed streaming platform

Apache SOFTWARE FOUNDATION

APACHE Spark

Java

hadoop

Understanding threads

Thread life cycle and Scheduling threads- Priorities , Sleep(),join()

Synchronization

- Thread: single sequential flow of control within a program
- Single-threaded program can handle one task at any time.
- Multitasking allows single processor to run several concurrent threads.
- Most modern operating systems support multitasking

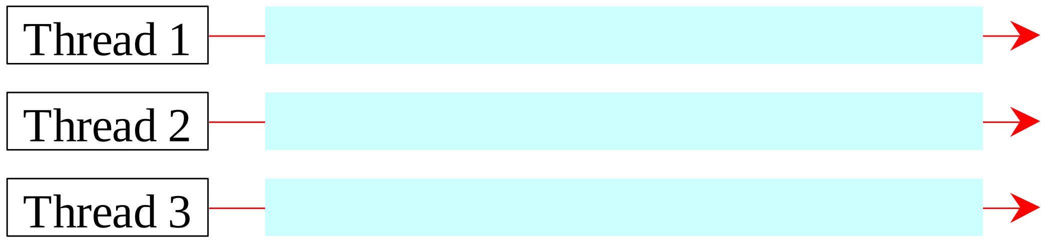
Advantages of Multithreading

- Reactive systems – constantly monitoring
- More responsive to user input – GUI application can interrupt a time-consuming task
- Server can handle multiple clients simultaneously
- Can take advantage of parallel processing

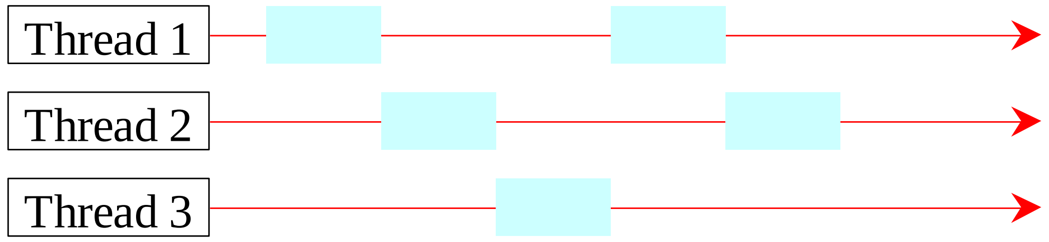
- Different processes do not share memory space.
- A thread can execute concurrently with other threads within a single process.
-
- All threads managed by the JVM share memory space and can communicate with each other

Threads Concept

Multiple threads on multiple CPUs



Multiple threads sharing a single CPU



Threads in Java

Creating threads in Java:

- Extend java.lang.Thread class

OR

- Implement java.lang.Runnable interface

Threads in Java

Creating threads in Java:

- Extend java.lang.Thread class
  - run() method must be overridden (similar to main method of sequential program)
  - run() is called when execution of the thread begins
  - A thread terminates when run() returns
  - start() method invokes run()
  - Calling run() does not create a new thread

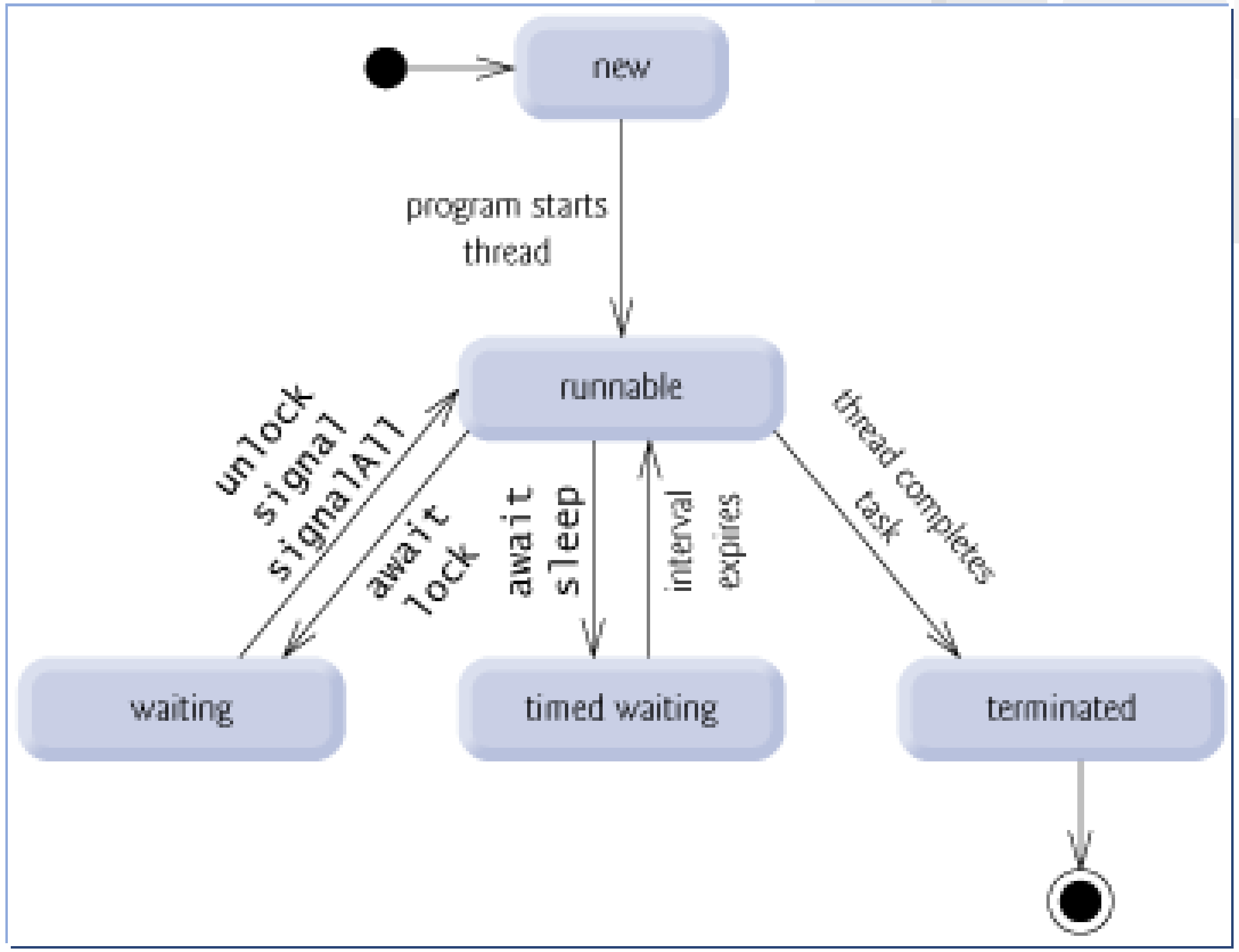
- Implement java.lang.Runnable interface

Threads in Java

Creating threads in Java:

- Extend java.lang.Thread class
- Implement java.lang.Runnable interface
  - If already inheriting another class (i.e., JApplet)
  - Single method: public void run()
  - Thread class implements Runnable

Thread States

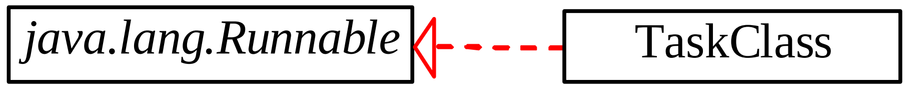


Thread termination

A thread becomes Not Runnable when one of these events occurs:

- Its sleep method is invoked.
- The thread calls the wait method to wait for a specific condition to be satisifed.
- The thread is blocking on I/O.

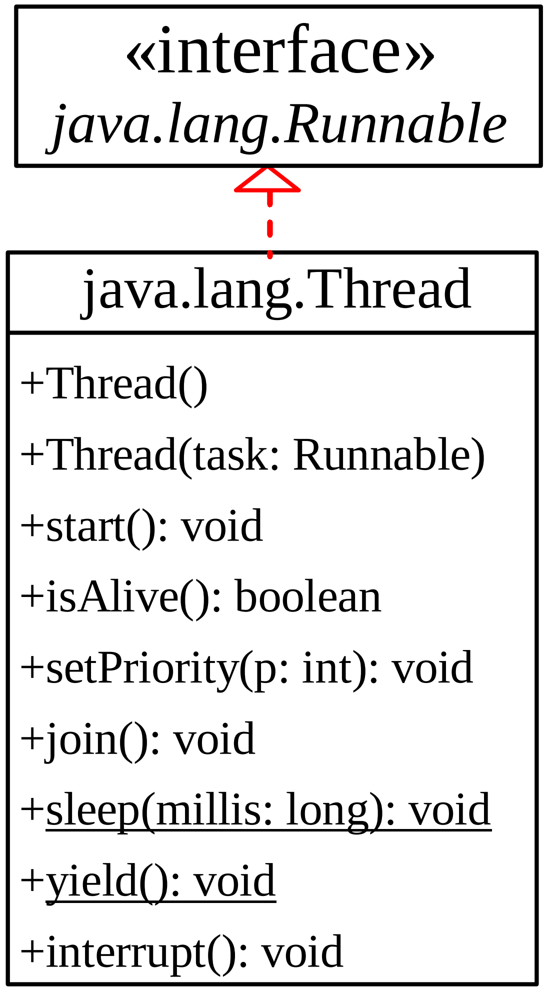
Creating Tasks and Threads



```
// Custom task class
public class TaskClass implements Runnable {
  ...
  public TaskClass(...) {
    ...
  }

  // Implement the run method in Runnable
  public void run() {
    // Tell system how to run custom thread
    ...
  }
  ...
}
```

```
// Client class
public class Client {
  ...
  public void someMethod() {
    ...
    // Create an instance of TaskClass
    TaskClass task = new TaskClass(...);

    // Create a thread
    Thread thread = new Thread(task);

    // Start a thread
    thread.start();
    ...
  }
  ...
}
```

The Thread Class



Creates a default thread.

Creates a thread for a specified task.

Starts the thread that causes the run() method to be invoked by the JVM.

Tests whether the thread is currently running.

Sets priority p (ranging from 1 to 10) for this thread.

Waits for this thread to finish.

Puts the runnable object to sleep for a specified time in milliseconds.

Causes this thread to temporarily pause and allow other threads to execute.

Interrupts this thread.

The Static yield() Method

You can use the yield() method to temporarily release time for other threads.

```
public void run() {
  for (int i = 1; i <= lastNum; i++) {
    System.out.print(" " + i);
    Thread.yield();
  }
}
```

Every time a number is printed, the print100 thread is yielded. So, the numbers are printed after the characters.

The Static sleep(milliseconds) Method

The sleep(long mills) method puts the thread to sleep for the specified time in milliseconds.

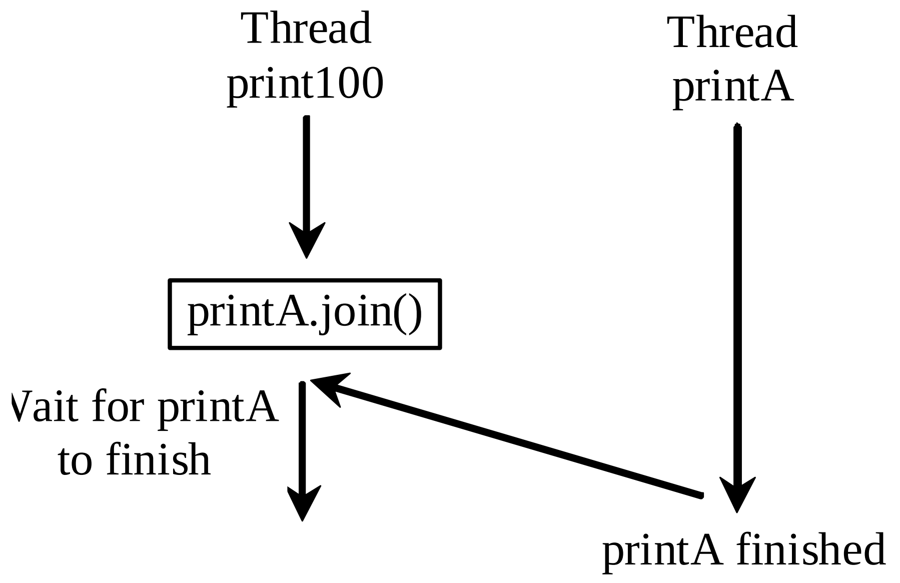
```
public void run() {
  for (int i = 1; i <= lastNum; i++) {
    System.out.print(" " + i);
    try {
      if (i >= 50) Thread.sleep(1);
    }
    catch (InterruptedException ex) {
    }
  }
}
```

Every time a number (>= 50) is printed, the print100 thread is put to sleep for 1 millisecond.

The join() Method

- You can use the join() method to force one thread to wait for another thread to finish.

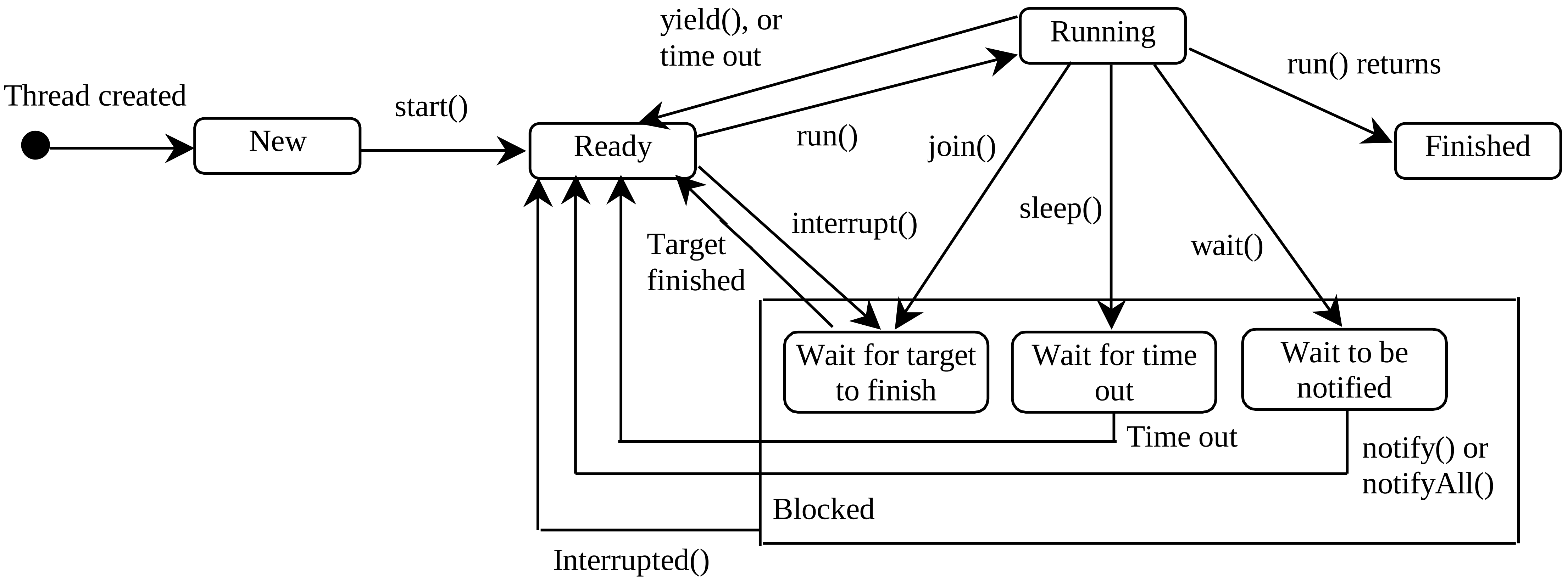
```
public void run() {
  Thread thread4 = new Thread(
    new PrintChar('c', 40));
  thread4.start();
  try {
    for (int i = 1; i <= lastNum; i++) {
      System.out.print(" " + i);
      if (i == 50) thread4.join();
    }
  }
  catch (InterruptedException ex) {
  }
}
```



- The numbers after 50 are printed after thread printA is finished.

Thread States

- A thread can be in one of five states: New, Ready, Running, Blocked, or Finished.



Thread Priority

- Each thread is assigned a default priority of `Thread.NORM_PRIORITY` (constant of 5). You can reset the priority using `setPriority(int priority)`.
- Some constants for priorities include `Thread.MIN_PRIORITY` `Thread.MAX_PRIORITY` `Thread.NORM_PRIORITY`
- By default, a thread has the priority level of the thread that created it.

Thread Scheduling

- An operating system's thread scheduler determines which thread runs next.
- Most operating systems use *timeslicing* for threads of equal priority.
- *Preemptive scheduling*: when a thread of higher priority enters the running state, it preempts the current thread.
- *Starvation*: Higher-priority threads can postpone (possible forever) the execution of lower-priority threads.

Thread Synchronization

- A shared resource may be corrupted if it is accessed simultaneously by multiple threads.
- Example: two unsynchronized threads accessing the same bank account may cause conflict.

Step	balance	thread[i]	thread[j]
1	0	newBalance = bank.getBalance() + 1;	
2	0		newBalance = bank.getBalance() + 1;
3	1	bank.setBalance(newBalance);	
4	1		bank.setBalance(newBalance);

Race Condition

- When two tasks are accessing a common resource in a way that causes conflict.
- Known as a *race condition* in multithreaded programs.
- A *thread-safe* class does not cause a race condition in the presence of multiple threads.

synchronized

- Problem: race conditions
- Solution: give exclusive access to one thread at a time to code that manipulates a shared object.
- Synchronization keeps other threads waiting until the object is available.
- The synchronized keyword synchronizes the method so that only one thread can access the method at a time.

Synchronizing Instance Methods and Static Methods

- A synchronized method acquires a lock before it executes.
- Instance method: the lock is on the object for which it was invoked.
- Static method: the lock is on the class.
- If one thread invokes a synchronized instance method (respectively, static method) on an object, the lock of that object (respectively, class) is acquired, then the method is executed, and finally the lock is released.
- Another thread invoking the same method of that object (respectively, class) is blocked until the lock is released.

Synchronizing Statements

•Invoking a synchronized instance method of an object acquires a lock on the object.
•Invoking a synchronized static method of a class acquires a lock on the class.
•A *synchronized block* can be used to acquire a lock on any object, not just *this* object, when executing a block of code.

```
synchronized (expr) {
  statements;
}
```

•expr must evaluate to an object reference.

•If the object is already locked by another thread, the thread is blocked until the lock is released.
•When a lock is obtained on the object, the statements in the synchronized block are executed, and then the lock is released.

Synchronizing Statements vs. Methods

Any synchronized instance method can be converted into a synchronized statement. Suppose that the following is a synchronized instance method:

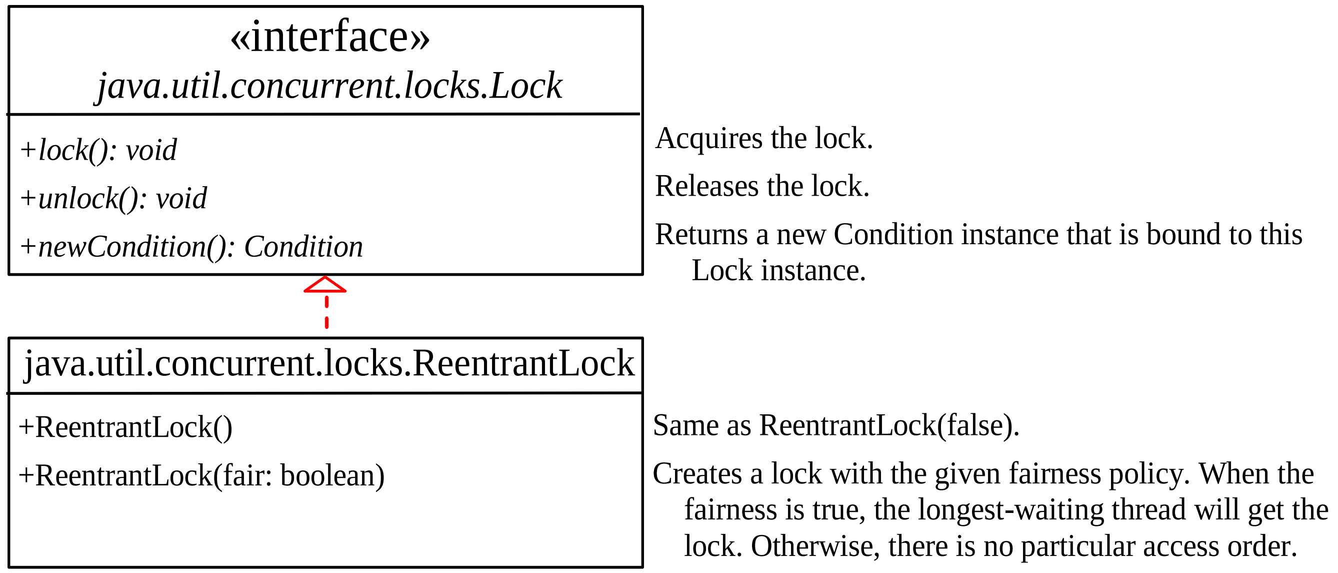
```
public synchronized void xMethod() {
  // method body
}
```

This method is equivalent to

```
public void xMethod() {
  synchronized (this) {
    // method body
  }
}
```

Synchronization Using Locks

- A synchronized instance method implicitly acquires a lock on the instance before it executes the method.
- You can use locks explicitly to obtain more control for coordinating threads.
- A lock is an instance of the Lock interface, which declares the methods for acquiring and releasing locks.
- newCondition() method creates Condition objects, which can be used for thread communication.



«interface»
java.util.concurrent.locks.Lock
+lock(): void
+unlock(): void
+newCondition(): Condition
java.util.concurrent.locks.ReentrantLock
+ReentrantLock()
+ReentrantLock(fair: boolean)

Acquires the lock.

Releases the lock.

Returns a new Condition instance that is bound to this Lock instance.

Same as ReentrantLock(false).

Creates a lock with the given fairness policy. When the fairness is true, the longest-waiting thread will get the lock. Otherwise, there is no particular access order.

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Q & A

Contact: amitmahadik@synergetics-india.com

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

Thank You