Synchronized

Synchronized

Java

Synchronized

Synchronized JVM

Synchronized

Synchronized

monitorenter monitorexit

monitorenter

+1 monitorexit -1

0

Java Synchronize

“ ”

“ ” monitorenter monitorexit

Reference

Synchronized

1. Synchronized Synchronized
   Synchronized(this)

2.
   Synchronized

   Synchronized

Synchronized

Synchronized

Synchronized

Synchronized method2

method2 method1

Synchronized

monitorenter

+1

JVM Java

Java 6 Monitor

/

Java

JDK

JDK Monitor

- Biased Locking
-
-

JDK Synchronized JVM

-

JVM CAS Mark Word

ID

- JVM
- CAS Mark Word

Synchronized

-
-

Synchronized

CAS

Synchronized

CAS Compareand Swap

CAS CPU

JNI Native C++ JDK

Unsafe

1.

2. CAS

CPU

3. ABA CAS

A

B A CAS

ReentrantLock

Synchronized ReentrantLock

Synchronized JVM

ReentrantLock Lock

volitile int

int AQS

AQS

AQS AbstractQueuedSynchronizer

Lock ReentrantLock

ReadWriteLock Semaphore CountDownLatch

FutureTask AQS

1. AQS volatile int state

   lock state=0

   state=1 state=1

2. AQS Node

   - Node

     waitStatus

     Node prev next

     FIFO

   - Node SHARED EXCLUSIVE

     Semaphore AQS

     ReentranLock

3. AQS ConditionObject

   Condition wait()

Condition signal()

4. AQS Condition Lock

Condition

Synchronized ReentrantLock

ReentrantLock Lock

ReentrantLock Synchronized

Synchronized

-
-
-
- Synchronized
-

Synchronized JVM

Synchronized

JVM Lock Lock

unLock()

finally{}

Synchronized

ReentrantLock

Java 6

Synchronized ReetrantLock

Synchronized ReetrantLock

ReentrantLock

ReentrantLock Sync Sync AQS

AOS AOS

CAS

ID ID

ReetrantLock JUC

JUC java.util.concurrent

Java

- CountDownLatch CyclicBarrier Semaphore

Synchronized

- ConcurrentHashMap ConcunrrentSkipListMap

  CopyOnWriteArrayList

- ArrayBlockingQueue SynchorousQueue

  PriorityBlockingQueue

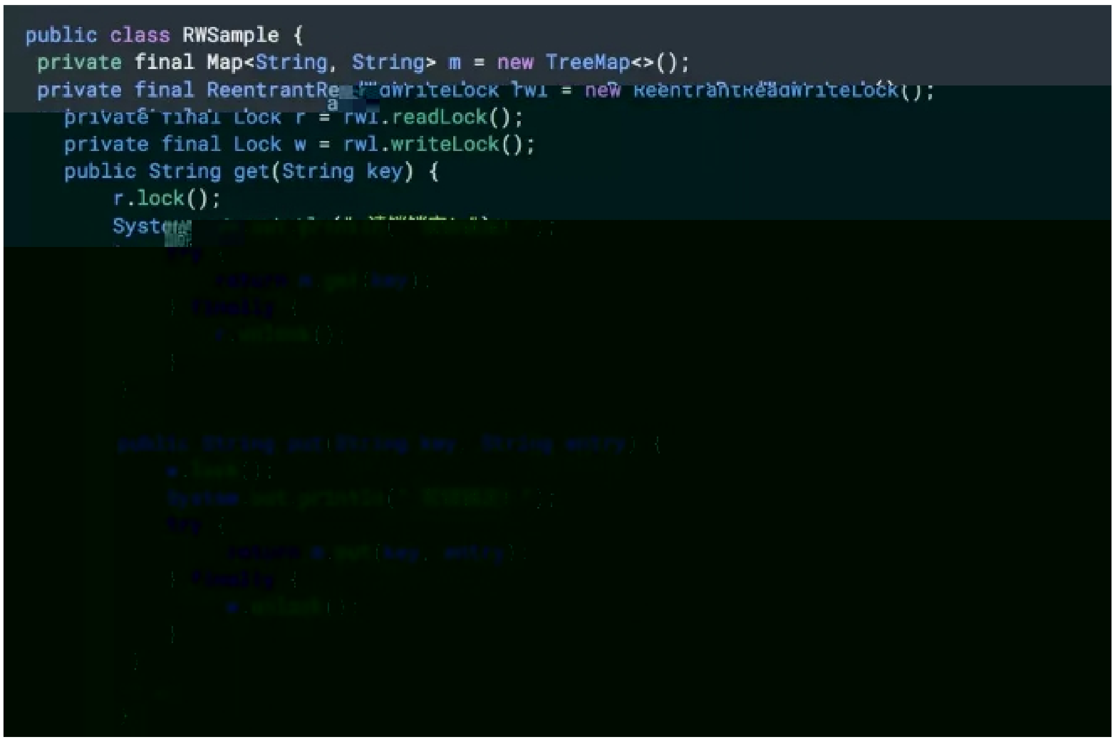
- Executor

ReadWriteLock StampedLock

ReentrantLock Synchronized

Java

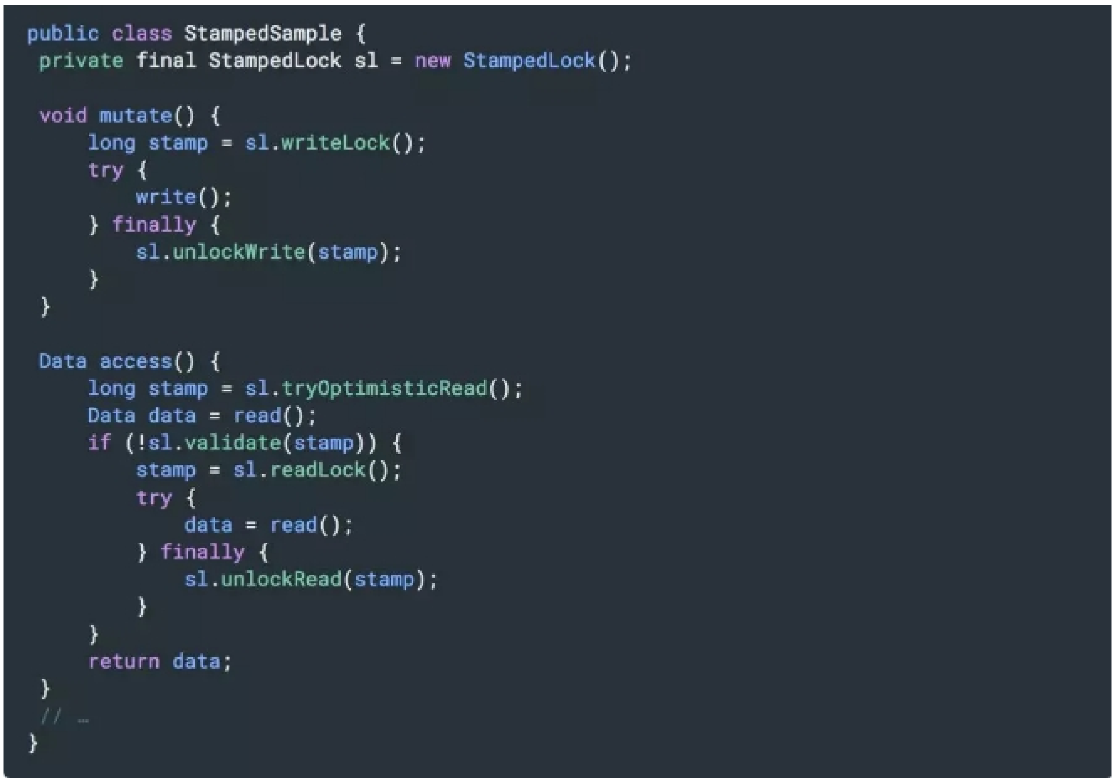
ReadWriteLock


```java
public class RWSample {
 private final Map<String, String> m = new TreeMap<>();
 private final ReentrantReadWriteLock rwl = new ReentrantReadWriteLock();
   private final Lock r = rwl.readLock();
   private final Lock w = rwl.writeLock();
   public String get(String key) {
       r.lock();
       System.out.println("[illegible]");
           try {
               return m.get(key);
           } finally {
               r.unlock();
           }
       }

       public String put(String key, String entry) {
           w.lock();
           System.out.println("[illegible]");
           try {
               return m.put(key, entry);
           } finally {
               w.unlock();
           }
       }
   }
```

Synchronized

JDK StampedLock

validate



```java
public class StampedSample {
 private final StampedLock sl = new StampedLock();

 void mutate() {
     long stamp = sl.writeLock();
     try {
         write();
     } finally {
         sl.unlockWrite(stamp);
     }
 }

 Data access() {
     long stamp = sl.tryOptimisticRead();
     Data data = read();
     if (!sl.validate(stamp)) {
         stamp = sl.readLock();
         try {
             data = read();
         } finally {
             sl.unlockRead(stamp);
         }
     }
     return data;
 }
 // …
}
```

Java

JUC CountDownLatch CyclicBarrier

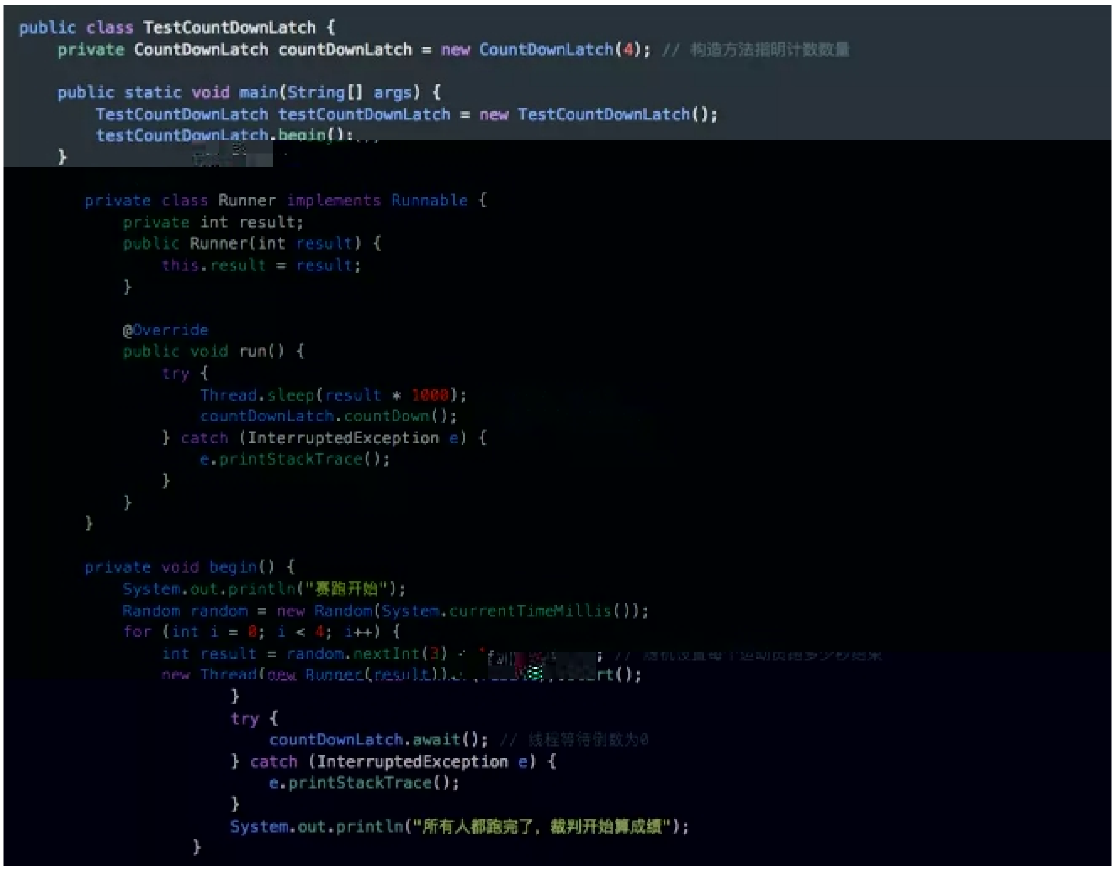
Semaphore

CountDownLatch

- “ ”
- 100

CountDownLatch

countDown 1 await



```java
public class TestCountDownLatch {
    private CountDownLatch countDownLatch = new CountDownLatch(4); // 构造方法指明计数数量

    public static void main(String[] args) {
        TestCountDownLatch testCountDownLatch = new TestCountDownLatch();
        testCountDownLatch.begin();
    }

    private class Runner implements Runnable {
        private int result;
        public Runner(int result) {
            this.result = result;
        }

        @Override
        public void run() {
            try {
                Thread.sleep(result * 1000);
                countDownLatch.countDown();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private void begin() {
        System.out.println("赛跑开始");
        Random random = new Random(System.currentTimeMillis());
        for (int i = 0; i < 4; i++) {
            int result = random.nextInt(3) [illegible]; // [illegible]
            new Thread(new Runner(result))[illegible]rt();
                }
                try {
                    countDownLatch.await(); // 线程等待倒数为0
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("所有人都跑完了，裁判开始算成绩");
            }
```

CyclicBarrier

CyclicBarrier

CyclicBarrier

CyclicBarrier await() await()

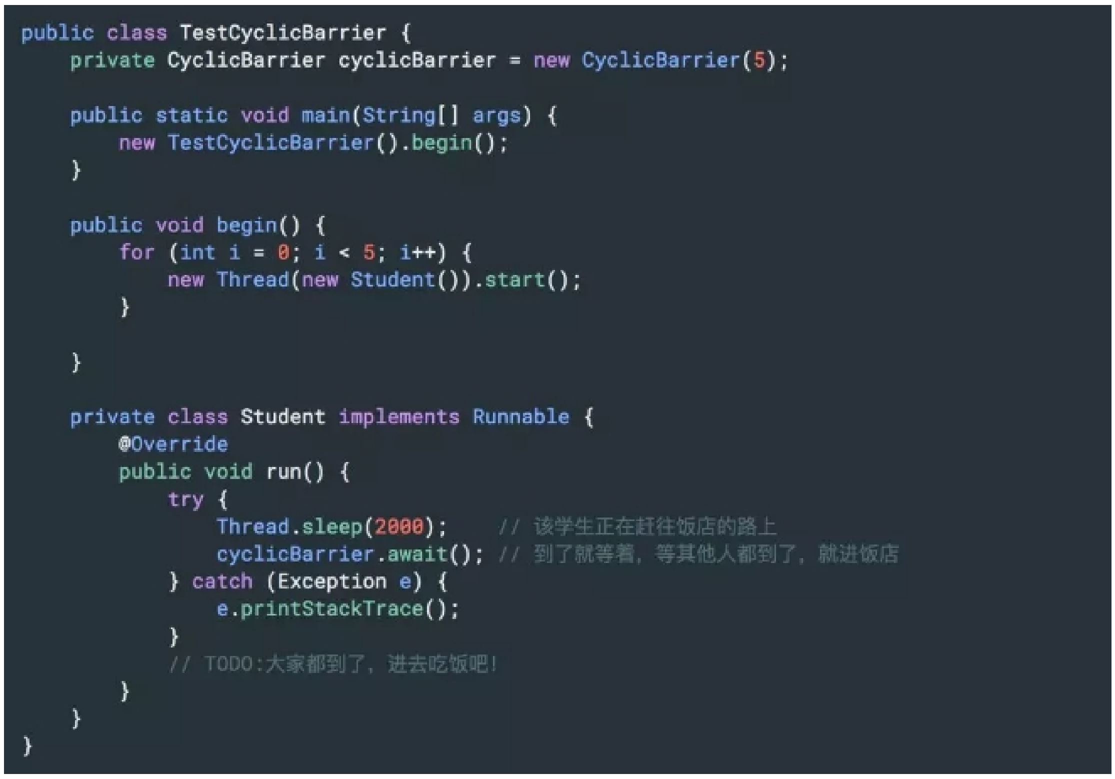
1 0

CyclicBarrier

await() N-1

Cyclic CyclicBarrier.await()

Barrier

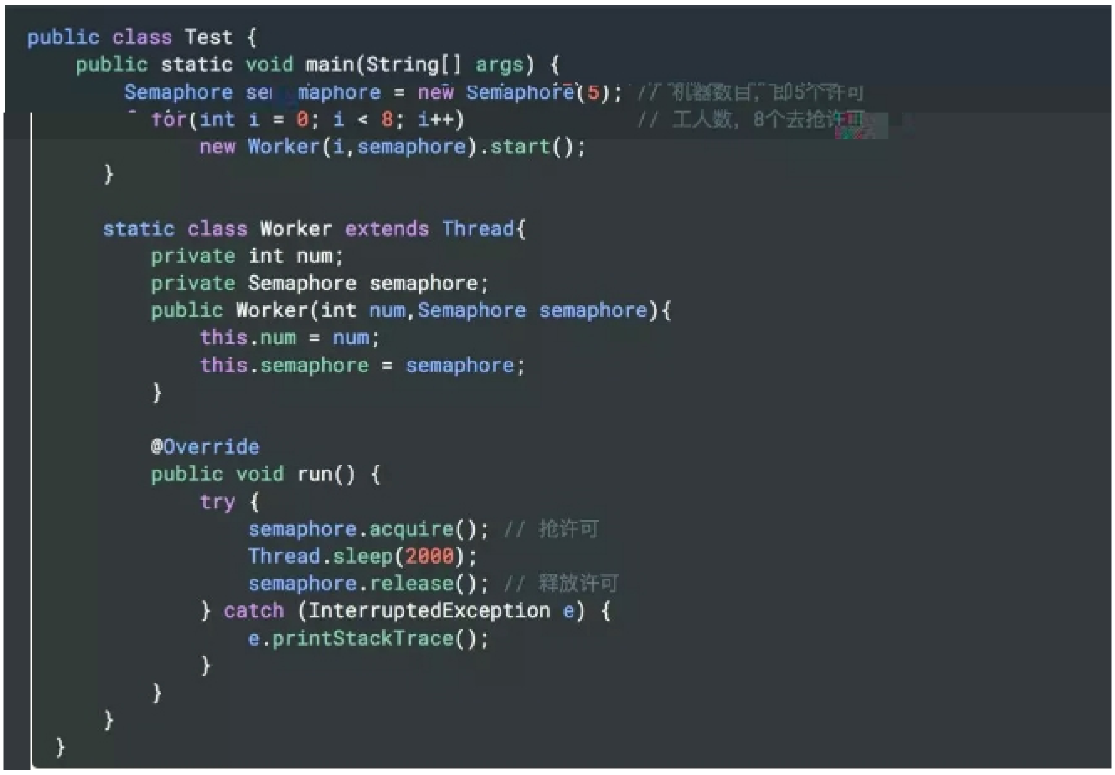


```java
public class TestCyclicBarrier {
    private CyclicBarrier cyclicBarrier = new CyclicBarrier(5);

    public static void main(String[] args) {
        new TestCyclicBarrier().begin();
    }

    public void begin() {
        for (int i = 0; i < 5; i++) {
            new Thread(new Student()).start();
        }

    }

    private class Student implements Runnable {
        @Override
        public void run() {
            try {
                Thread.sleep(2000);    // 该学生正在赶往饭店的路上
                cyclicBarrier.await(); // 到了就等着，等其他人都到了，就进饭店
            } catch (Exception e) {
                e.printStackTrace();
            }
            // TODO:大家都到了，进去吃饭吧！
        }
    }
}
```

Semaphore Java

acquire()

release()



```java
public class Test {
    public static void main(String[] args) {
        Semaphore semaphore = new Semaphore(5); // 机器数目，即5个许可
        for(int i = 0; i < 8; i++)             // 工人数，8个去抢许可
            new Worker(i,semaphore).start();
    }

    static class Worker extends Thread{
        private int num;
        private Semaphore semaphore;
        public Worker(int num,Semaphore semaphore){
            this.num = num;
            this.semaphore = semaphore;
        }

        @Override
        public void run() {
            try {
                semaphore.acquire(); // 抢许可
                Thread.sleep(2000);
                semaphore.release(); // 释放许可
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Semaphore 1

acquire

Semaphore

CyclicBarrier CountDownLatch

- CountDownLatch CyclicBarrier
- CountDownLatch countDown/await

  await countDown

countDown CyclicBarrier

await await

CountDownLatch N

CyclicBarrier

public CyclicBarrier(int parties, Runnable barrierAction)

CyclicBarrier N

N

CountDownLatch

Java

Java

- Java “ ”

  Worker AQS

  HashSet<Worker> workers

- workQueue

  BlockingQueue<Runnable> workQueue

  workQueue

  Workers

Java

- corePoolSize
- maximumPoolSize
- keepAliveTime
- workQueue execute

  Runnable

Worker

execute()

- corePoolSize
- corePoolSize
-

  maximumPoolSize
-

  maximumPoolSize

  RejectExecutionException

keepAliveTime

corePoolSize

corePoolSize

Java

1. SingleThreadExecutor

- corePoolSize 1
- maximumPoolSize 1
- keepAliveTime 0L
- workQueue new LinkedBlockingQueue<Runnable>()

2. FixedThreadPool

FixedThreadPool

FixedThreadPool

- corePoolSize nThreads
- maximumPoolSize nThreads
- keepAliveTime 0L
- workQueue new LinkedBlockingQueue<Runnable>()

3. CachedThreadPool

CachedThreadPool

60

JVM

SynchronousQueue 1

daemon SERVER

Executor

- corePoolSize 0
- maximumPoolSize Integer.MAX_VALUE
- keepAliveTime 60L
- workQueue new SynchronousQueue<Runnable>()

1

4. ScheduledThreadPool

ScheduledThreadPool

DEFAULT_KEEPALIVEMILLIS

- corePoolSize corePoolSize
- maximumPoolSize Integer.MAX_VALUE
- keepAliveTime DEFAULT_KEEPALIVE_MILLIS
- workQueue new DelayedWorkQueue()

Java

1. execute() ExecutorService.execute Runable

```
ExecutorService.execute(Runnable runable)
```

2. submit() ExecutorService.submit() Future

isDone() Future

get() isDone()

get() get()

```
FutureTask task = ExecutorService.submit(Runnable runnable);
FutureTask<T> task = ExecutorService.submit(Runnable runnable,T Result);
FutureTask<T> task = ExecutorService.submit(Callable<T> callable);
```

Java

Java Java

Java

Java Java

volatile

volatile Java

volatile

1.

2.

Java 8

lock unlock

-
-

read write

- load
- store

load store

- read
- write

use assgin

-
-

volatile 8

volatile

volatile

volatile

volatile

volatile

volatile

Java volatile

volatile Synchronized

Synchronized volatile

ThreadLocal Synchonized

ThreadLocal Synchronized

Synchronized

“ ”

ThreadLocal

“ ”

ThreadLocal

ThreadLocal Java

ID Cookie

ThreadLocal

ThreadLocal

Map

ThreadLocal

ThreadLocal

ThreadLocal remove

ThreadLocal ThreadLocalMap

ThreadLocalMap key

ThreadLocal

ThreadLocalMap OOM

remove

worker