- CountDownLatch
- CyclicBarrier
- **Phaser**
- Semaphore

CountDownLatch

- Pastga qarab hisoblash usuli

CountDownLatch 1

- The **CountDownLatch** class is another important class for concurrent execution.
- It is a synchronization aid that allows one or more than one thread to wait until a set of operations being performed in another thread is completed.
- In other words: CountDownLatch in Java is a kind of synchronizer which allows one Thread to wait for one or more Threads before starts processing.
- CountDownLatch classi bir vaqtda bajariladigan ishlar uchun muhim class dir.
- Bu synchronization/sinxronlash maqsadida yaratilgan bo'lib u bir yoki birnechta Thread lar, bosha Thread larni o'z ishini tugatib bo'lishini kutib turishini taminlaydi.
- Dehqoncha aytadigan bo'lsak CountDownLatch classi bitta Thread boshqa bir Thread larni o'z ishini tugashini kutush imkonini beradi.

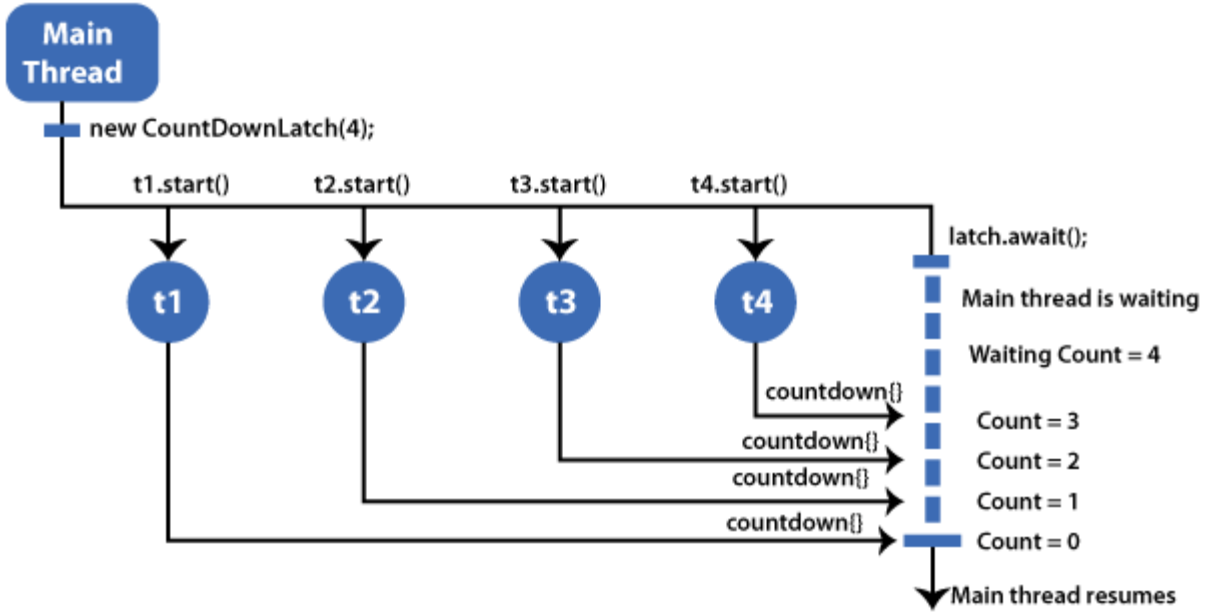
CountDownLatch 2

- CountDownLatch was **introduced with JDK 1.5 along with other concurrent utilities like CyclicBarrier, Semaphore,** ConcurrentHashMap **and** BlockingQueue in java.util.concurrent package. This class **enables a java thread to wait until other set of threads completes** their tasks. e.g. Application's main thread want to wait, till other service threads which are responsible for starting framework services have completed started all services.

CountDownLatch 3

- CountDownLatch class initializes with the count, which we pass to the constructor. This count is a number of thread.
- Count decremented each time when a thread complete its execution.
- When count reaches to zero, it means all threads have completed their execution, and thread waiting on latch resume the execution.
- CountDownLatch classidan instance olayotgandan constructorga count berib yuboramiz. B count Thread lar sonini anglatadi.
- Hargal Thread o’z ishini tugatganida Count birga kamayadi.
- Count 0 ga tushsa, bu degani barcha Threadlar o’z ishini tugatdi degani, va Latch/qulufni kutayotgan Thread o’z ishini davom ettirishi mumkin.

CountDownLatch UML



CountDownLatch Methods

- 1. await() method cause the current thread to wait until one of the following is not done: (bu method hozirgi thread ni quyidagi 2ta holat bo'lguncha kutushini taminlaydi.)
  - The latch has counted down to zero. Latch 0 ga teng bo'lganucha.
  - Interruption of the thread is done. Thread interruption qilinsa.

- 2. countdown() - It documents the count of the latch and releases all the waiting threads when the count reaches zero. It has done the following things: (U latch lar sonini bildiradi. Agar count 0 bo'lsa lutib turgan thread larni qo'yib yuboradi. U quyidagi ishlarni bajaradi.):
  - The count is decremented when the current count is greater than zero. (count sonini kamaytiradi.)
  - All waiting threads will be re-enabled for thread scheduling purposes when the new count is zero. (count 0 ga teng bo'lsa kutayotgan thread lar ishlashni davom ettiradi.)
-

CountDownLatch Example 1

- Thread class

```
public class MyThread extends Thread {
   private CountDownLatch cdLatch;

   public MyThread(CountDownLatch cdLatch) {
      this.cdLatch = cdLatch;
   }

   @Override
   public void run() {
         System.out.println(getName() + " doing some work.");
         Thread.sleep(1000);
         cdLatch.countDown();
    }
}
```

CountDownLatch Example 2

```
public static void main(String[] args) throws
InterruptedException {
        CountDownLatch cd = new CountDownLatch(3);

        MyThread t1 = new MyThread(cd);
        MyThread t2 = new MyThread(cd);
        MyThread t3 = new MyThread(cd);

        t1.start();
        t2.start();
        t3.start();

        cd.await();

        System.out.println("Main finished it is work.");
    }
```

- Main class

CountDownLatch Problem

- Solve File Scanner problem with CountDownLatch.

CountDownLatch Links

- https://www.javatpoint.com/countdownlatch-in-java
- https://howtodoinjava.com/java/multi-threading/when-to-use-countdownlatch-java-concurrency-example-tutorial/
- https://www.baeldung.com/java-countdown-latch
- https://javarevisited.blogspot.com/2012/07/countdownlatch-example-in-java.html#axzz7SOWzcfjL

CyclicBarrier

- Tsiklik to'siq

CyclicBarrier 1

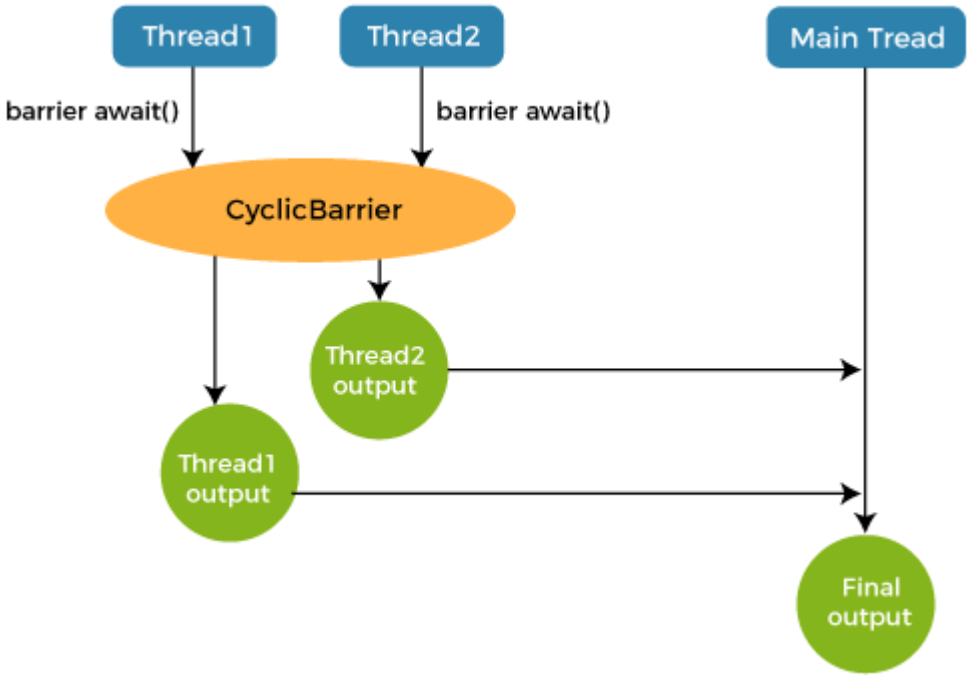
- The **cyclic barrier** is a concurrent utility mechanism of synchronization that allows a set of threads to all wait for each other to reach a common barrier point.
- It is useful in Java programs that involve a fixed-sized party of threads that must wait for each other.
- The word cyclic is used for its reused purpose. It means that the barrier can be re-used when all waiting threads are released
- Cyclic barrier synchronization mexanizni bo’lib u qaysidir nuqtagacha birnechta Thread larni bir birini kutub turushini taminlaydi.
- Javada aniq bit songa ega bo’lgan Thread lar bir birini kutub turishi kerak bo’lgan dasturlarda juda foydalidir.
- Cyclic – Tsils so’zini ishlatilishi manosi bu qayta ishlatilishi mumkin. Amosi shundaki barcha kutayotgan thread lar qo’yib yuborilgandan keyin nuqta qayta ishlatilishi mumkin.

CyclicBarrier 2

- The point is the barrier that ensures all the threads complete their execution before reaching here.
- Therefore, the mechanism is known as the cyclic barrier. In short, cyclic barrier preserves a **count of threads**.
- Note that we can use CyclicBarrier instead of CountDownLatch but vice-versa is not possible because the latch cannot be reused when the count reaches zero.

- Mano shundaki nuqtaga kelguncha barcha Thread lar o’z ishini tugatgan bo’ladi.
- Shu sababdan bu mexanizm Tsikl(aylana) nuqta deb nomlangan. Siqqacha aytsak …
- Eslatma biz CountDownLatch ni o’rniga CyclicBarrier ni ishlatsak bo’ladi ammo teskarisiga emas. Sababi latch dagi count 0 ga ten bo’lganida uni qayta ishlatib bo’lmaydi.

CyclicBarrier Uml



CyclicBarrier How to use

- In order to use the cyclic barrier, first, we should create an object of the CyclicBarrier class by using any of the two constructors. In the constructor, parse the required number of threads that will wait for each other. When the specified number of threads reaches at a common point, invoke the await() method on the CyclicBarrier object. The await() method suspends the threads until all the threads invoke the await() method on the same CyclicBarrier object. When all the threads invoked the await() method, then the barrier is tripped and all the threads continue their operation after tripping.
- If the current thread is the last thread to arrive, and a non-null barrier action was supplied in the constructor, then the current thread runs the action before allowing the other threads to continue.

CyclicBarrier Example 1

```
public class MyThread extends Thread {
    private CyclicBarrier cb;

    public MyThread(CyclicBarrier cb) {
        this.cb = cb;
    }

    @Override
    public void run() {
        System.out.println(getName() + " doing some
work.");
        Thread.sleep(1000);
        cb.await();
    }
}
```

- Thread Class

CyclicBarrier Example 2

```
public static void main(String[] args) {
        CyclicBarrier cb = new CyclicBarrier(5);

        MyThread t1 = new MyThread(cb);
        MyThread t2 = new MyThread(cb);
        MyThread t3 = new MyThread(cb);
        MyThread t4 = new MyThread(cb);
        MyThread t5 = new MyThread(cb);

        t1.start();
        t2.start();
        t3.start();
        t4.start();
        t5.start();

        System.out.println("Main Thread finished.");
    }
```

CyclicBarrier 3

- Optionally, we can pass the second argument to the constructor, which is a *Runnable* instance. This has logic that would be run by the last thread that trips the barrier:
- **public CyclicBarrier**(**int** parties, Runnable barrierAction)
- CyclicBarrier classining ikkinchi constructori Runnable qabul qiladi. Bu barrierAction ni CyclicBarrier dagi nuqtaga yetib kelgan ohirgi Thread ishlatadi. Yaniy barcha Threadlar nuqtaga yetib kelganidan keyin bu barrierAction ishga tushiriladi.

CyclicBarrier Links

- https://www.javatpoint.com/java-cyclicbarrier
- https://www.baeldung.com/java-cyclic-barrier
- https://www.java67.com/2015/06/how-to-use-cyclicbarrier-in-java.html

Phaser

- Fezer

Phaser 1

- **The Phaser allows us to build logic in which threads need to wait on the barrier before going to the next step of execution.**
- **Phaser in Java** is also one of the synchronization aid provided in concurrency util.
- Phaser is similar to other synchronization barrier utils like CountDownLatch and CyclicBarrier.

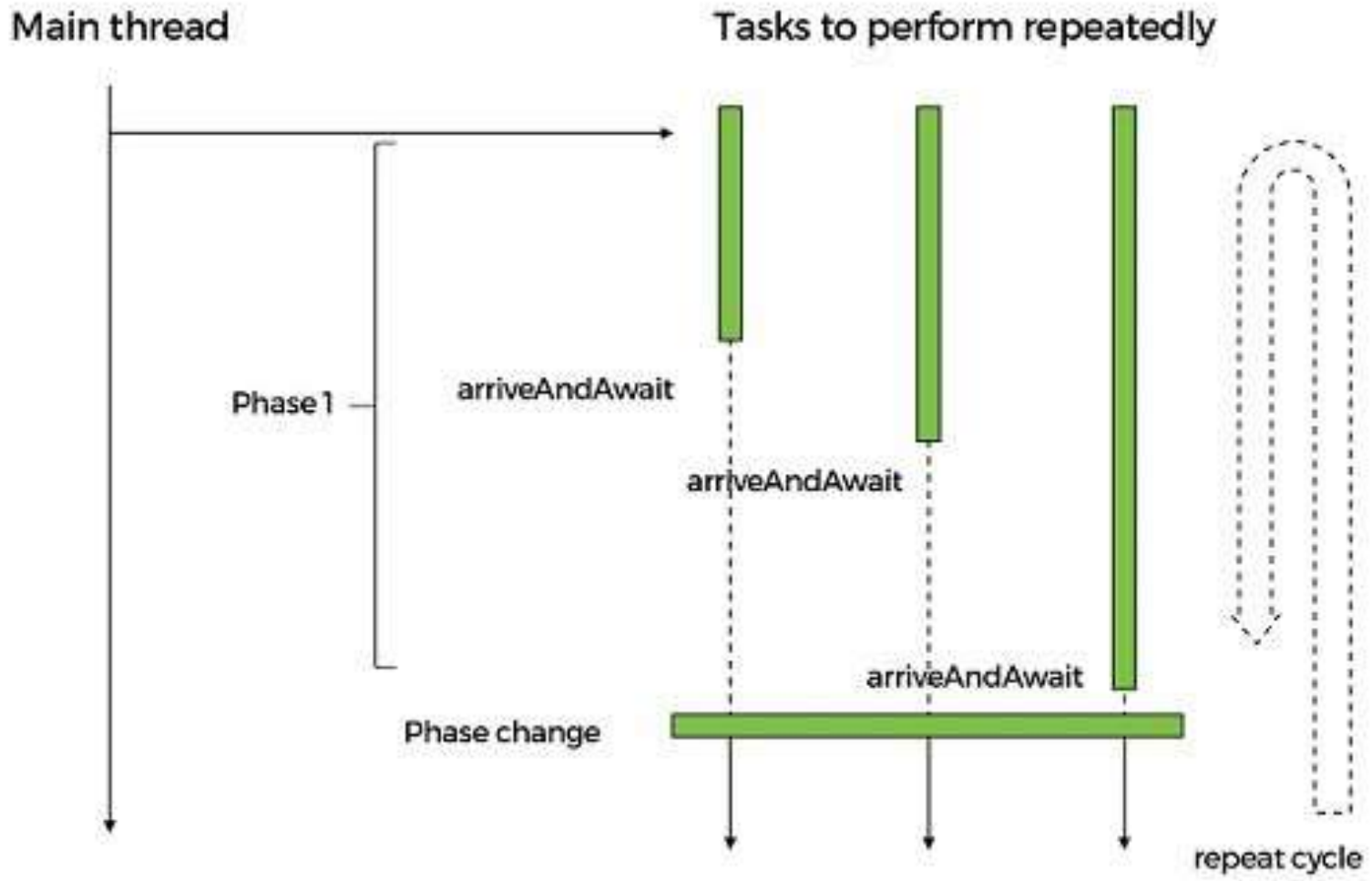
- Phaser bizga Thread lar bitta nuqtaga kelib kutib turushligi kerak bo'ladigan logikani qurish uchun kerak bo'ladi.
- Phaser synchronization maqsadida yaratilgan concurrency util classi hisoblanadi.
- Phaser classi synchronization barrier util dagi CountDownLatch and CyclicBarrier classlariga juga o'xshashdir.

Phaser 2

- What sets Phaser apart is it is **reusable** (like CyclicBarrier) and **more flexible** in usage.
- In both CountDownLatch and CyclicBarrier number of parties (thread) that are registered for waiting can't change where as in Phaser that **number can vary**.
- Also note that Phaser has been introduced in **Java 7**.

- Phaser classi CyclicBarrier ga o’xshab qayta ishlatish mumkin va u judayam moslashuvchan.
- CountDownLatch va CyclicBarrier classlarida Kutib turadigan Thread lar sonini o’zgarmas hisoblanadi. Phaser da bu son o’zgarishi mumkin.
- Phaser Java 7 da taqdim qilingan.

Phaser UML



Phaser constructors

- Phaser class in Java has 4 constructors:
- **Phaser()**- Creates a new phaser with no initially registered parties, no parent, and initial phase number 0.
- **Phaser(int parties)**- Creates a new phaser with the given number of registered unarrived parties, no parent, and initial phase number 0.
- **Phaser(Phaser parent)**- Creates a new phaser with the given parent with no initially registered parties.
- **Phaser(Phaser parent, int parties)**- Creates a new phaser with the given parent and number of registered unarrived parties.

Phaser Methods 1

- Some of the methods in Phaser class are as given below-
- **resgister()**- Adds a new unarrived party to this phaser. It returns the arrival phase number to which this registration applied. (Phaser ga yangi Thread ni registratsiya qiladi)

- **arriveAndAwaitAdvance()**- This method awaits other threads to arrives at this phaser. Returns the arrival phase number, or the (negative) current phase if terminated. If you want to wait for all the other registered parties to complete a given phase then use this method. (Bu method hozirgi Thread ni boshqa Threadlarni ham shu Phaser ga kelishini kutub turushga undaydi. Agar siz boshqa Phaser dan registratsiya qilgan Thread larni kutub turishni hohlasangiz shu methodni ishlating.)

Phaser Methods 2

- **arriveAndDeregister()** - Arrives at this phaser and deregisters from it without waiting for others to arrive. Returns the arrival phase number, or a negative value if terminated. (Phaser ga yetib kelganida bu Thread ni boshqa Thread larni kutub turmasligi uchun Phaser/Navbat dan olib tashlaydi.)

Phaser Example 1

```
public class MyThread extends Thread {
    private Phaser phaser;

    public MyThread(Phaser phaser) {
        this.phaser = phaser;
        System.out.println(phaser.register());
    }

    @Override
    public void run() {
        System.out.println(getName() + " doing some work.");
        Thread.sleep(1000);
        // Using await and advance so that all thread wait here
        phaser.arriveAndAwaitAdvance();
        phaser.arriveAndDeregister();
    }
}
```

- Thread Class

Phaser Example 2

```
public class PhaserDemo {
    public static void main(String[] args) {

        Phaser phaser = new Phaser();
        MyThread t1 = new MyThread(phaser);
        MyThread t2 = new MyThread(phaser);
        MyThread t3 = new MyThread(phaser);
        MyThread t4 = new MyThread(phaser);

        t1.start();
        t2.start();
        t3.start();
        t4.start();

        //For main thread
        phaser.arriveAndAwaitAdvance();
        System.out.println("Main finish work.");
    }
}
```

- Main

Phaser Problems

- Solve file Scanner problem with Phaser

Phaser Links

- https://www.netjstech.com/2016/01/phaser-in-java-concurrency.html
- https://www.baeldung.com/java-phaser
- https://www.geeksforgeeks.org/java-util-concurrent-phaser-class-in-java-with-examples/
- https://medium.com/double-pointer/guide-to-phaser-in-java-efd810e4fc1b

Semaphore

- Ручная сигнализация, сигнализировать

Semaphore

- Semaphore class used to limit the number of concurrent threads accessing a specific resource.
- In other words **Semaphore class** that contains constructors and various methods to control access over the shared resource

- Semaphore classi bir vaqtni o’zida umumiy manbani ishlatadigan Thread lar sonini cheklash uchun ishlatiladi.
- Boshqacha so’zlar bilan aytgandan Semaphore classida Umumiy manbadan foydalanishni cheklash uchun constructor va methodlar mavjut.

What is a semaphore?

- A Semaphore is used to limit the number of threads that want to access a shared resource. In other words, it is a non-negative variable that is shared among the threads known as a **counter**. It sets the limit of the threads.
  - If **counter > 0**, access to shared resources is provided.
  - If **counter = 0**, access to shared resources is denied.
- In short, the counter keeps tracking the number of permissions it has given to a shared resource. Therefore, semaphore grants permission to threads to share a resource.
- Semaphore Umumiy manbaga cheklov/limit qo’yish uchun ishlatiladi. Boshq so’z bilan aytsak, u counter deb nomlangan va Thread lar o'rtasida umumiy bo’lgan, manfiy bo'lmagan o'zgaruvchidir. U Thread lar limitini o’rnatadi.
  - If counter > 0 bo’lsa umumiy manba taqdim etiladi.
  - If counter = 0 bo’lsa umumiy manba taqdim etilmaydi.
- Qissa counter umumiy manbani olgan thread lar sonini sanaydi.

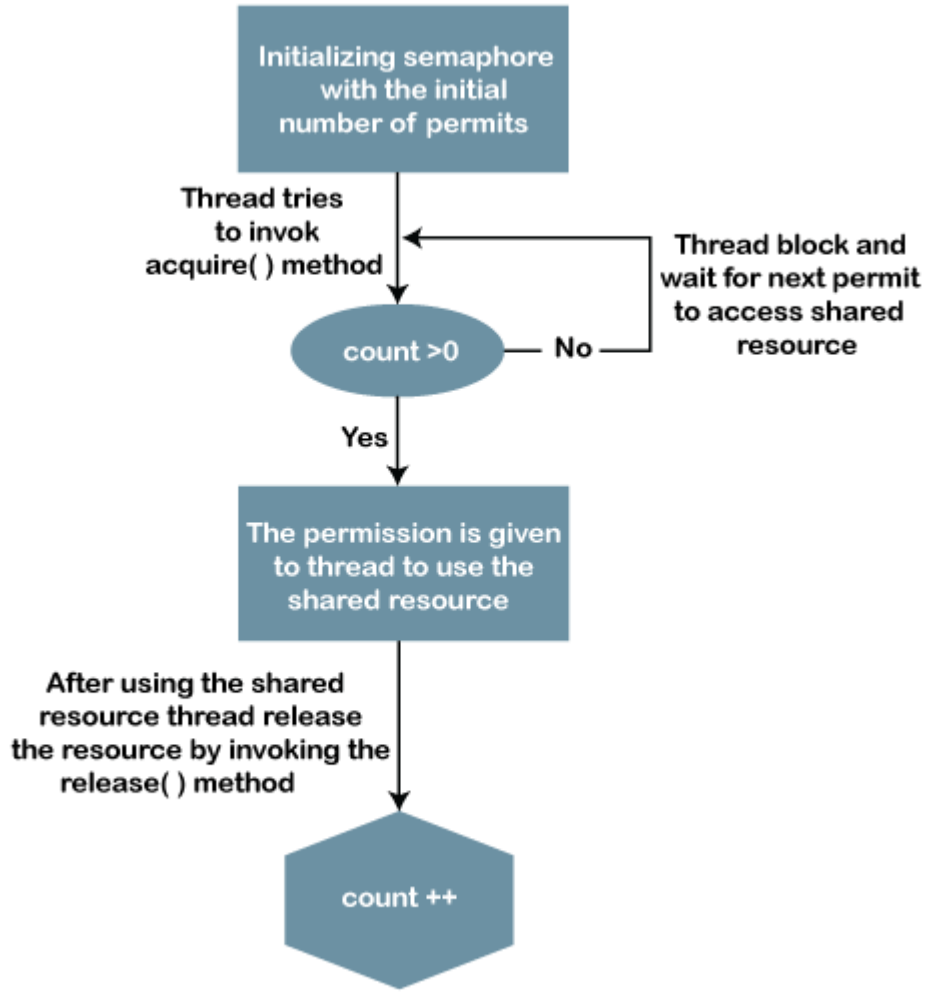
Characteristics of Semaphore

- There are the following characteristics of a semaphore:
- It provides synchronization among the threads.
- It decreases the level of synchronization. Hence, provides a low-level synchronization mechanism.
- The semaphore does not contain a negative value. It holds a value that may either greater than zero or equal to zero.
- We can implement semaphore using the test operation and interrupts, and we use the file descriptors for executing it.
-

Working of Semaphore

- Semaphore controls over the shared resource through a counter variable. The counter is a non-negative value. It contains a value either greater than 0 or equal to 0.
- If **counter > 0**, the thread gets permission to access the shared resource and the counter value is **decremented** by 1.
- Else, the thread will be blocked until a permit can be acquired.
- When the execution of the thread is completed then there is no need for the resource and the thread releases it. After releasing the resource, the counter value **incremented** by 1.
- If another thread is waiting for acquiring a resource, the thread will acquire a permit at that time.
- If **counter = 0**, the thread does not get permission to access the shared resource.

Semaphore flow chart.



Working of Semaphore in Java

dasturlash.uz

Semaphore Example1

- Thread Class

```
public class MyThread extends Thread {
    private String name;
    private Semaphore semaphore;
    public MyThread(Semaphore semaphore) { this.semaphore = semaphore;
                                           name = getName(); }
    public void run() {
        System.out.println("Thread " + name + " : acquiring lock...");
        //thread A acquire lock and the permit count decremented by 1
        semaphore.acquire();
        System.out.println("Thread " + name + " : got the permit!");
        try {
            System.out.println("Thread " + name + " : is performing action");
            Thread.sleep(3000);
        } finally {
            System.out.println("Thread " + name + " : releasing lock...");
            //invoking release() method after successful execution
            semaphore.release();
        }
    }
}
```

Semaphore Example2

- Main Class

```
public class SemaphoreDemoMain {
    public static void main(String[] args) {
        Semaphore semaphore = new Semaphore(3);
        MyThread t1 = new MyThread(semaphore);
        MyThread t2 = new MyThread(semaphore);
        MyThread t3 = new MyThread(semaphore);
        MyThread t4 = new MyThread(semaphore);

        t1.start();
        t2.start();
        t3.start();
        t4.start();
        System.out.println("Main finished.");
    }
}
```

Semaphore important methods

- *acquire()* - method acquire the permits from the semaphore, blocking until one is available. It reduces the number of available permits by 1.
(Bu method Semaphore dan ruxsatni oladi va count ni birga kamaytiradi. Agar ruxsat bo'lmasa, Thread ruxsat bo'lmagunicha in active statusda turadi. Thread ruxsatni olsa yoki interrupt qilinsa in active statusdan chiqadi.)

- *tryAcquire()* – return true if a permit is available immediately and acquire it otherwise return false. (Method agar ruxsat bo'sla true return qiladi va Thread ruxsatni oladi. Agar ruxsat bo'lmasa false return qiladi.)
- *release() – release a permit. (Ruxsatni qo'yib yuboradi.)*
- *availablePermits()* - return number of current permits available. (Mavjut bo'lgan ruxsatlar sonini return qiladi)

Types of Semaphores

- There are four types of semaphores, which are as follows:
  - Counting Semaphores
  - Bounded Semaphores
  - Timed Semaphores
  - Binary Semaphores

Timed *Semaphore*

- Next, we will discuss Apache Commons *TimedSemaphore*. *TimedSemaphore* allows a number of permits as a simple Semaphore but in a given period of time, after this period the time reset and all permits are released.

Using Semaphore as Lock

- Java allows us to use a semaphore as a lock. It means, it locks the access to the resource. Any thread that wants to access the locked resource, must call the **acquire()** method before accessing the resource to acquire the lock. The thread must release the lock by calling the **release()** method, after the completion of the task. Remember that set the upper bound to 1. For example
- Java Semaphore larni lock sifatida ishlatish imkonini beradi. Bu degani manmani quluflash degani. Harqanday Thread manbani olmoqchi bo'lganida acquire() methodni chaqirishi kerak va shunda Thread manbani quluflagan bo'ladi. Thread o'z ishini tugatganidan keyin release() methodini chaqirishi kerak. Esta tuting limitni 1ga tenglab qo'yishimiz kerak. Namuna:

```
Semaphore sem= new Semaphore(1);
...
sem.acquire();
try  {  //critical section  }
finally  {  sem.release();  }
```

Semaphore Links

- https://www.javatpoint.com/java-semaphore
- https://www.baeldung.com/java-semaphore