

به نام خدا



# برنامه‌سازی پیشرفته

دانشگاه شهید بهشتی . دانشکده مهندسی و علوم کامپیوتر

دکتر مجتبی وحیدی اصل

---

برنامه نویسی چند نخ (Multi Thread Programming) - بخش دوم

آریا زریاب

# فهرست مطالب

- در جاوا Daemon Thread .1
- Thread Group .2
- Multi-tasking .3
- در جاوا Synchronization .4
- Thread Control .5

# در جاوا Daemon Thread

- یک ترد در جاوا می باشد که برای ارائه خدمات به ترد کاربر (**User Thread**) استفاده می شود.
- در واقع Daemon Thread دارای خدماتی (**Service Thread**) است که کارهای پشتیبانی در پس زمینه (Background Supporting Tasks) را برای User Thread ها انجام می دهد.
- عمر Daemon Thread وابسته به ترد های کاربر است.
- یعنی وقتی همهی User Thread ها تمام شوند، **JVM** به صورت خودکار Daemon Thread ها را متوقف می کند.
- به صورت خودکار **Daemon Thread** هایی وجود دارند که در حال اجرا هستند. مانند Garbage collector، که برای جمع آوری زباله ها و مدیریت حافظه استفاده می شود، و finalizer که برای آزاد کردن منابع قبل از حذف شئ استفاده می شود.
- معمولاً یک ترد با اولویت پایین (Low Priority Thread) است.

# چرا JVM در صورت عدم وجود نخ کاربر، نخ daemon را خاتمه می‌دهد؟

- هدف اصلی Daemon Thread ارائه‌ی خدمات به User Thread ها برای انجام کارهای پشتیبان در پس زمینه است.
- اگر هیچ User Thread ای وجود نداشته باشد، نگه داشتن Daemon Thread بی‌معنی است، چون دیگر چیزی برای خدمت‌رسانی وجود ندارد.
- به همین دلیل JVM به صورت خودکار Daemon Thread ها را خاتمه می‌دهد زمانی که همه‌ی User Thread ها تمام شده باشند.

# ۲ متد مهم از Thread در کلاس Daemon

کلاس [java.lang.Thread](#) این ۲ متد زیر را برای ترد daemon فراهم میکند :

Method	Description
<b>public void setDaemon(boolean status)</b>	is used to mark the current thread as daemon thread or user thread
<b>public boolean isDaemon()</b>	is used to check that current thread is daemon

```
public class TestDaemonThread1 extends Thread {  
    public void run() {  
        if (Thread.currentThread().isDaemon()) { // checking for daemon thread  
            System.out.println("daemon thread work");  
        } else {  
            System.out.println("user thread work");  
        }  
    }  
  
    public static void main(String[] args) {  
        TestDaemonThread1 t1 = new TestDaemonThread1(); // creating thread  
        TestDaemonThread1 t2 = new TestDaemonThread1();  
        TestDaemonThread1 t3 = new TestDaemonThread1();  
  
        t1.setDaemon(true); // now t1 is daemon thread  
  
        // starting threads  
        t1.start();  
        t2.start();  
        t3.start();  
    }  
}
```

user thread work  
user thread work  
daemon thread work

# نکته و مثال

اگر بخواهیم نخی را به daemon تغییر دهیم، نباید از قبل در حال اجرا یا start شده باشد، در غیر اینصورت با exception مواجه می‌شویم.

```
class TestDaemonThread2 extends Thread {  
    public void run() {  
        System.out.println("Name: " + Thread.currentThread().getName());  
        System.out.println("Daemon: " + Thread.currentThread().isDaemon());  
    }  
  
    public static void main(String[] args) {  
        TestDaemonThread2 t1 = new TestDaemonThread2();  
        TestDaemonThread2 t2 = new TestDaemonThread2();  
  
        t1.start();  
        t1.setDaemon(true); // will throw IllegalThreadStateException  
        t2.start();  
    }  
}
```

Name: Thread-4  
Daemon: false

---

```
java.lang.IllegalThreadStateException: null  
    at java.base/java.lang.Thread.setDaemon(Thread.java:2239)  
    at TestDaemonThread2.main(#15:12)  
    at .(#16:1)
```

# جاوا در ThreadGroup

- جاوا راهی ساده برای گروه بندی چند **Thread** در یک شیء (**Object**) فراهم کرده است.
- با استفاده از **ThreadGroup** می‌توانیم چندین **Thread** را با یک فراخوانی متدا  
، **(suspend)**  
از سر بگیریم **(resume)**،  
یا قطع **(interrupt)** کنیم.
- در جاوا، گروه ترد ها با کلاس **java.lang.ThreadGroup** پیاده سازی شده است.
- هر **ThreadGroup** مجموعه‌ای از **Thread** ها را نمایش می‌دهد.
- یک **ThreadGroup** می‌تواند گروههای دیگر از **ThreadGroup** ها را هم شامل شود.  
در نتیجه ساختار آن به شکل درختی (**Tree Structure**) است:  
به جز گروه اولیه (**parent**)، هر گروه والد (**initial thread group**) دارد.
- هر **Thread** فقط می‌تواند به اطلاعات گروه خودش (**ThreadGroup** خود) دسترسی داشته باشد.
- هیچ **Thread**ی اجازه ندارد اطلاعات مربوط به گروه والد یا گروههای دیگر را بخواند یا تغییر دهد.

- تنها ۲ سازنده برای این کلاس وجود دارند:

Constructor	Description
<b>ThreadGroup(String name)</b>	<b>creates a thread group with given name</b>
<b>ThreadGroup(ThreadGroup parent, String name)</b>	<b>creates a thread group with given parent group and name</b>

```
ThreadGroup tg1 = new ThreadGroup("Group A");
Thread1 = new Thread(tg1, new MyRunnable(), "one");
Thread2 = new Thread(tg1, new MyRunnable(), "two");
Thread3 = new Thread(tg1, new MyRunnable(), "three");
```

- در این مثال، سه تا Thread داریم که همگی به یک گروه تعلق دارند.

نام گروه: Group A

کلاس: MyRunnable (کلاسی که رابط Runnable را پیاده سازی کرده است)

نام ترددها: one, two, three

```
public class ThreadGroupDemo implements Runnable {
    public void run() {
        System.out.println(Thread.currentThread().getName());
    }

    public static void main(String[] args) {
        ThreadGroupDemo runnable = new ThreadGroupDemo();
        ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup");

        Thread t1 = new Thread(tg1, runnable, "one");
        t1.start();
        Thread t2 = new Thread(tg1, runnable, "two");
        t2.start();
        Thread t3 = new Thread(tg1, runnable, "three");
        t3.start();

        System.out.println("Thread Group Name: " + tg1.getName());
        tg1.list();
    }
}
```

one

two

three

Thread Group Name: Parent ThreadGroup

java.lang.ThreadGroup[name=Parent ThreadGroup,maxpri=10]

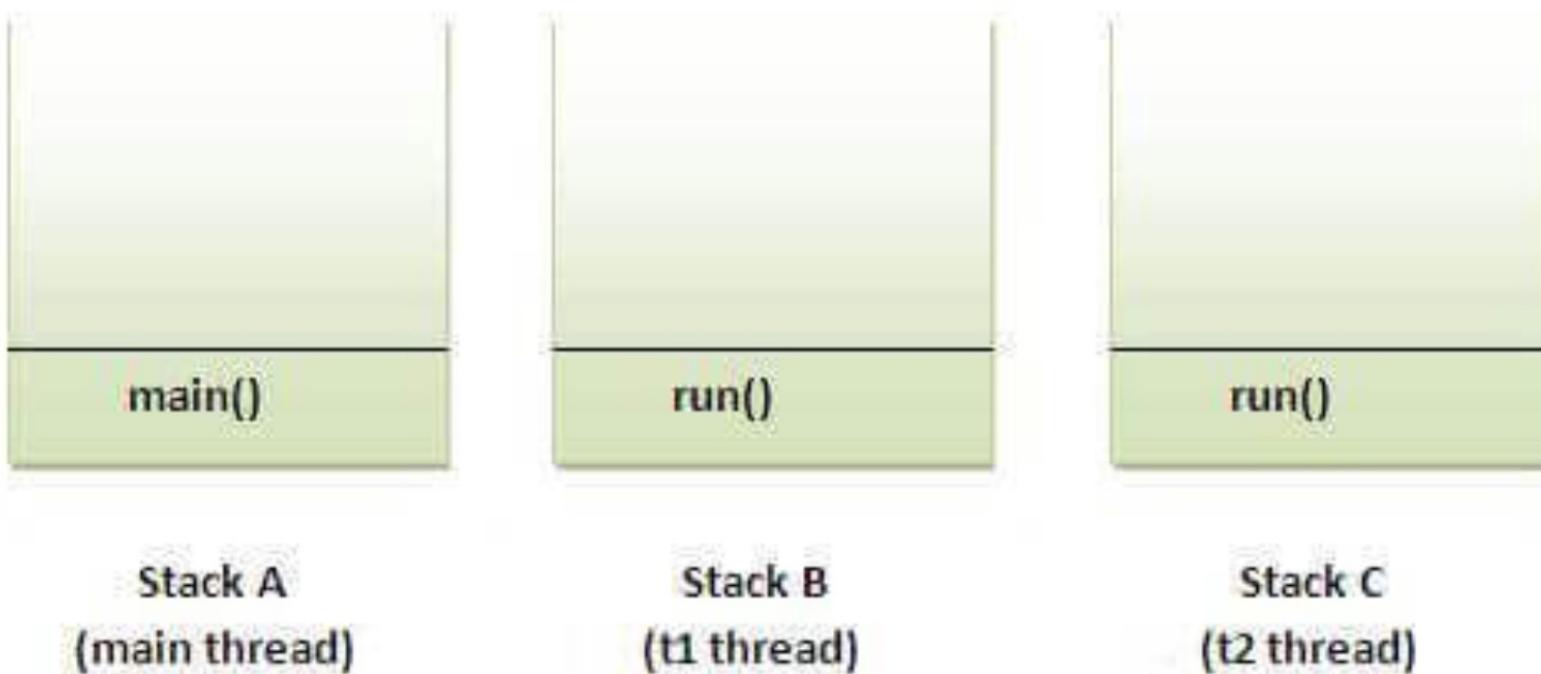
# چگونه یک کار (Task) را توسط چند نخ انجام دهیم؟

اگر بخواهید یک تاسک را توسط چندین ترد انجام دهید، کافیست تنها یک متدهای `run()` داشته باشید:

```
class TestMultitasking1 extends Thread {  
    public void run() {  
        System.out.println("task one");  
    }  
  
    public static void main(String args[]) {  
        TestMultitasking1 t1 = new TestMultitasking1();  
        TestMultitasking1 t2 = new TestMultitasking1();  
        TestMultitasking1 t3 = new TestMultitasking1();  
  
        t1.start();  
        t2.start();  
        t3.start();  
    }  
}
```

task one  
task one  
task one

Note: Each thread run in a separate callstack.



# multithreading ↗ multtasking

```
class Simple1 extends Thread {  
    public void run() {  
        System.out.println("task one");  
    }  
}  
  
class Simple2 extends Thread {  
    public void run() {  
        System.out.println("task two");  
    }  
}  
  
class TestMultitasking3 {  
    public static void main(String args[]) {  
        Simple1 t1 = new Simple1();  
        Simple2 t2 = new Simple2();  
  
        t1.start();  
        t2.start();  
    }  
}
```

task two  
task one

# در جاوا synchronization

- همگام سازی یا **Synchronization** در جاوا قابلیتی است که به ما این امکان را می‌دهد که دسترسی چند نخ (Thread) را به یک منبع مشترک **(Shared Resource)** کنترل کنیم.
- در واقع، وقتی چند نخ به طور هم زمان به یک شیء یا متغیر مشترک دسترسی دارند، احتمال بروز تداخل (**Interference**) و ناهماهنگی داده‌ها (**Inconsistency**) وجود دارد.
- با استفاده از همگام سازی، می‌توانیم این مشکل را برطرف کنیم تا در هر لحظه فقط یک نخ به منبع مورد نظر دسترسی داشته باشد.
- دلایل اصلی استفاده از همگام سازی در جاوا عبارت‌اند از:
  - جلوگیری از تداخل نخ‌ها (**Thread Interference**) **(1)**
  - زمانی که چند نخ به طور هم‌زمان مقدار یک متغیر مشترک را تغییر می‌دهند، ممکن است مقدار نهایی اشتباه شود.
  - همگام سازی تضمین می‌کند که تغییرات به صورت کنترل شده و پشت‌سرهم انجام شوند.
- حفظ سازگاری داده‌ها (**Data Consistency**) **(2)**
- همگام سازی مانع می‌شود داده‌ها در اثر اجرای هم‌زمان نخ‌ها دچار بی‌نظمی یا ناهماهنگی شوند.

- همگام‌سازی در جاوا به دو دسته‌ی اصلی تقسیم می‌شود:

(1) انحصار متقابل (Mutual Exclusion)

این نوع همگام‌سازی باعث می‌شود در یک زمان فقط یک نخ بتواند به بخش بحرانی (Critical Section) از کد دسترسی داشته باشد.

روش‌های پیاده‌سازی آن:

**Synchronized Method .1**

**Synchronized Block .2**

**Static Synchronization .3**

(2) همکاری بین نخ‌ها (Inter-thread Communication)

این بخش از همگام‌سازی به ارتباط و هماهنگی بین نخ‌ها مربوط است، به‌طوری که نخ‌ها بتوانند با یکدیگر تبادل داده یا سیگنال انجام دهند.

# مفهوم Lock یا قفل در جاوا

- همگامسازی (Synchronization) در جاوا بر پایه‌ی مفهومی به نام **قفل (Lock)** یا **مانیتور (Monitor)** ساخته شده است.
- در واقع، **هر شیء** در جاوا دارای یک قفل داخلی است که از آن برای کنترل دسترسی چند رشته (Thread) به منابع مشترک استفاده می‌شود.
- زمانی که یک Thread بخواهد به داده‌های یک شیء به صورت همزمان و ایمن دسترسی پیدا کند، باید ابتدا قفل آن شیء را به دست آورد (lock را بگیرد) و پس از پایان کار، قفل را آزاد کند تا نخ‌های دیگر بتوانند از آن استفاده کنند.

به عبارت دیگر:

- تا زمانی که یک Thread قفل یک شیء را در اختیار دارد، هیچ Thread دیگری نمی‌تواند به بخش‌های همزمان‌شده‌ی آن شیء دسترسی پیدا کند.

# مثال (بدون Synchronization)

در مثال زیر، به دلیل نبود همگام‌سازی، چند Thread به‌طور همزمان به منبع مشترک دسترسی دارند، در نتیجه خروجی برنامه ناهماهنگ و غیرقابل‌پیش‌بینی خواهد بود:

```
class MyThread1 extends Thread {  
    Table t;  
    MyThread1(Table t) {  
        this.t = t;  
    }  
    public void run() {  
        t.printTable(5);  
    }  
}  
  
class MyThread2 extends Thread {  
    Table t;  
    MyThread2(Table t) {  
        this.t = t;  
    }  
    public void run() {  
        t.printTable(100);  
    }  
}
```

```
class Table {  
    void printTable(int n) { // method not synchronized  
        for (int i = 1; i <= 5; i++) {  
            System.out.println(n * i);  
            try {  
                Thread.sleep(400);  
            } catch (Exception e) {  
                System.out.println(e);  
            }  
        }  
    }  
}  
  
public class TestSynchronization1 {  
    public static void main(String args[]) {  
        Table obj = new Table(); // shared resource (only one object)  
  
        MyThread1 t1 = new MyThread1(obj);  
        MyThread2 t2 = new MyThread2(obj);  
  
        t1.start();  
        t2.start();  
    }  
}
```

# خروجي

5

100

10

200

15

300

20

400

25

500

# متد synchronized در جاوا

- اگر متدى را در جاوا با کلیدواژه‌ی **synchronized** تعریف کنیم، به آن متد همگام‌سازی‌شده گفته می‌شود.  
    • به عبارت دیگر:
- زمانی که یک متد synchronized باشد،  **فقط یک Thread** در هر لحظه می‌تواند به آن متد از طریق همان شیء دسترسی داشته باشد.
- متدهای همگام‌سازی‌شده برای  **قفل کردن** شیء (Object Lock) هنگام دسترسی به منابع مشترک استفاده می‌شوند.
- وقتی یک Thread متد synchronized را فراخوانی می‌کند:
  1. ابتدا قفل آن شیء را در اختیار می‌گیرد (Lock acquisition)
  2. تا زمانی که متد تمام نشده، هیچ Thread دیگری نمی‌تواند وارد همان متد (یا متد هم‌زمان‌شده‌ی دیگری از همان شیء) شود
  3. پس از اتمام متد، قفل آزاد می‌شود (Lock release)

```
class MyThread1 extends Thread {
    Table t;
    MyThread1(Table t) {
        this.t = t;
    }
    public void run() {
        t.printTable(5);
    }
}

class MyThread2 extends Thread {
    Table t;
    MyThread2(Table t) {
        this.t = t;
    }
    public void run() {
        t.printTable(100);
    }
}
```

```
class Table {
    synchronized void printTable(int n) { // synchronized method
        for (int i = 1; i <= 5; i++) {
            System.out.println(n * i);
            try {
                Thread.sleep(400);
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }
}
```

```
public class TestSynchronization2 {
    public static void main(String args[]) {
        Table obj = new Table(); // shared resource (only one object)

        MyThread1 t1 = new MyThread1(obj);
        MyThread2 t2 = new MyThread2(obj);

        t1.start();
        t2.start();
    }
}
```

# خروجی

5  
10  
15  
20  
25  
100  
200  
300  
400  
500

# در جاوا synchronized block

- بلوک همگام‌سازی‌شده (**Synchronized Block**) در جاوا برای کنترل دسترسی چندین ترد به بخش خاصی از کد استفاده می‌شود.
- برخلاف متدهمگام‌سازی‌شده که کل متدهمگام‌سازی‌شده می‌توان فقط **بخش مشخصی از کد** را قفل کرد.

فرض کنید متدهما شامل ۵۰ خط کد است، اما فقط ۵ خط از آن نیاز به همگام‌سازی دارد.

در این حالت، با استفاده از Synchronized Block می‌توانید فقط همان بخش خاص را قفل کنید تا سرعت اجرای برنامه کاهش پیدا نکند.

## نکات مهم برای :Synchronized Block

1. بلوک همگام‌سازی‌شده برای قفل کردن یک شئ خاص در زمان دسترسی چند ترد به منبع مشترک استفاده می‌شود.
2. محدوده‌ی (Scope) بلوک همگام‌سازی‌شده از متدهمگام‌سازی‌شده کوچکتر است.
3. استفاده از آن باعث می‌شود فقط بخش بحرانی (CriticalSection) از کد قفل شود و بقیه قسمت‌ها آزاد بمانند.

Syntax to use synchronized block :

```
synchronized (object reference expression) {  
    // code block  
}
```

```
class MyThread1 extends Thread {
    Table t;
    MyThread1(Table t) {
        this.t = t;
    }
    public void run() {
        t.printTable(5);
    }
}

class MyThread2 extends Thread {
    Table t;
    MyThread2(Table t) {
        this.t = t;
    }
    public void run() {
        t.printTable(100);
    }
}
```

```
class Table {
    void printTable(int n) {
        synchronized(this) { // synchronized block
            for (int i = 1; i <= 5; i++) {
                System.out.println(n * i);
                try {
                    Thread.sleep(400);
                } catch (Exception e) {
                    System.out.println(e);
                }
            }
        } // end of the method
    }
}
```

```
public class TestSynchronization2 {
    public static void main(String args[]) {
        Table obj = new Table(); // shared resource (only one object)

        MyThread1 t1 = new MyThread1(obj);
        MyThread2 t2 = new MyThread2(obj);

        t1.start();
        t2.start();
    }
}
```

# خروجی

5  
10  
15  
20  
25  
100  
200  
300  
400  
500

# دستور های کنترلی نخ ها در جاوا

- زبان Java کنترل کامل روی برنامه های چندنخی (Multithreaded) را در اختیار برنامه نویس قرار می دهد.
- به کمک این قابلیت، می توان برنامه ای نوشت که در آن تردها را بتوان متوقف (suspend)، از سر گرفته (resume) یا حتی به طور کامل متوقف کرد .(stop).  
این کار به شما امکان می دهد جریان اجرای تردها را دقیقاً بر اساس نیاز خود مدیریت کنید.
- جاوا چند متد استاتیک (static methods) فراهم کرده است که می توانید برای کنترل رفتار تردها از آنها استفاده کنید.

No.	Method & Description
1	<b>public void suspend()</b> This method puts a thread in the suspended state and can be resumed using resume() method.
2	<b>public void stop()</b> This method stops a thread completely.
3	<b>public void resume()</b> This method resumes a thread, which was suspended using suspend() method.
4	<b>public void wait()</b> Causes the current thread to wait until another thread invokes the notify() method.
5	<b>public void notify()</b> Wakes up a single thread that is waiting on this object's monitor.

# پایان دادن ترد ها در جاوا

- در جاوا، پایان دادن به نخها (Killing Threads) یکی از بخش‌های مهم در کنترل اجرای چندنخی (Multithreading) است.
- به صورت پیش فرض، وقتی متدهای run() در یک نخ تمام می‌شود، آن نخ به صورت خودکار از بین می‌رود.
- اما گاهی نیاز داریم نخ را قبل از اتمام اجرای کاملش متوقف یا حذف کنیم.
- در نسخه‌های اولیه‌ی جاوا، برای کنترل اجرای نخها از متدهای زیر استفاده می‌شد:
  - suspend() .1
  - resume() .2
  - stop() .3

اما این متدها از نسخه‌ی 2 Java به بعد منسوخ (deprecated) شدند، چون می‌توانستند منجر به خرابی سیستم یا بی‌ثباتی داده‌ها شوند. به عنوان مثال، اگر نخی در حال به روزرسانی داده‌ای مشترک بود و ناگهان متوقف می‌شد، داده در وضعیت ناپایدار باقی می‌ماند.

- روش‌های مدرن برای توقف نخها :

## 1. استفاده از پرچم (Boolean Flag)

در این روش، متغیری از نوع boolean (مثلاً با نام exit) تعریف می‌شود که نشان می‌دهد نخ باید ادامه دهد یا متوقف شود.

## 2. استفاده از Thread.interrupt()

روش دیگر برای توقف یک نخ، استفاده از متدهای interrupt() است.

این متدهد باعث نمی‌شود نخ بلا فاصله متوقف شود، بلکه سیگنالی برای قطع اجرا ارسال می‌کند.

```
// Example of killing threads in Java
class MyThread implements Runnable {

    // to stop the thread
    private boolean exit;
    private String name;
    Thread t;

    MyThread(String threadname) {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
        exit = false;
        t.start(); // Starting the thread
    }

    // execution of thread starts from run() method
    public void run() {
        int i = 0;
        while (!exit) {
            System.out.println(name + ": " + i);
            i++;
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                System.out.println("Caught: " + e);
            }
        }
        System.out.println(name + " Stopped.");
    }

    // for stopping the thread
    public void stop() {
        exit = true;
    }
}
```

```
// Main class
public class Main {
    public static void main(String args[]) {
        // creating two objects t1 & t2 of MyThread
        MyThread t1 = new MyThread("First thread");
        MyThread t2 = new MyThread("Second thread");

        try {
            Thread.sleep(500);
            t1.stop(); // stopping thread t1
            t2.stop(); // stopping thread t2
            Thread.sleep(500);
        } catch (InterruptedException e) {
            System.out.println("Caught: " + e);
        }

        System.out.println("Exiting the main Thread");
    }
}
```

# خروجی

New thread: Thread[First thread,5,main]

New thread: Thread[Second thread,5,main]

First thread: 0

Second thread: 0

First thread: 1

Second thread: 1

First thread: 2

Second thread: 2

First thread: 3

Second thread: 3

First thread: 4

Second thread: 4

First thread Stopped.

Second thread Stopped.

Exiting the main Thread

# خودمون رو بسنجیم

این بخش برای این طراحی شده که در پایان مطالعه این اسلاید، بتونی خودت رو محک بزنی و ببینی آیا مفاهیم رو به خوبی یاد گرفتی یا نه. سوالات زیر رو مرور کن و سعی کن بدون نگاه کردن به متن درس، به اون ها پاسخ بدی.

- چه اتفاقی میفتند اگر daemon thread سعی کند یک عملیات I/O را انجام دهد و ترد main خاتمه یابد ؟
- چگونه میتوانیم بدون استفاده از متدهای منسوخ شده در جاوا کاری کنیم یک نخ متوقف شود و سپس تحت شرایط خاصی دوباره از سر گرفته شود ؟
- به نظر شما چرا متدهای stop() از جاوا منسوخ شده است ؟
- مثالی از اجرای چند ترد ارائه دهید که در آن بدون استفاده از synchronization به مشکل برسیم.
- حالا مثال نوشته شده تان را یکبار با متدهای synchronized و یکبار هم با block آن همگام سازی کنید.

## پایان

در صورت هرگونه سوال یا پیشنهاد میتوانید با من  
در ارتباط باشید (:)

gmail: [ariazaryab@gmail.com](mailto:ariazaryab@gmail.com)  
telegram: @Arriaw