

به نام خدا

آموزش پنجاه و نهم

اهداف آموزشی این قسمت عبارتند از:

۱. معرفی کلاس `ExecutorService`
۲. معرفی متدهای مرتبط با کلاس `ExecutorService`
۳. معرفی متد `submit`

در آموزش های پنجاه و یکم، پنجاه و دوم و پنجاه سوم به مفهوم `Thread` ها در زبان برنامه نویسی جاوا آشنا شدیم. در این آموزش قصد داریم تا ببینیم که به روشی می توانیم ترتیب اجرای `Thread` های ایجاد شده را کنترل نماییم (در واقع علت آنکه این آموزش با کمی فاصله از آموزش های مرتبط با `Thread` آمده است این می باشد که خواسته ایم تا با کمی تمرین مفهوم `Thread` در ذهن کاربران نهادینه شده سپس به مباحث تکمیلی بپردازیم).

در ابتدا پروژه ای در محیط برنامه نویسی اکیپس تحت عنوان `Session 59th` ایجاد کرده و کلاسی در آن به نام `HowToControlThreads` به معنی "چگونه `Thread` ها را کنترل کنیم" می سازیم:

```
public class HowToControlThreads extends Thread {  
  
    @Override  
    public void run() {  
  
        System.out.println("This is text from first thread 1");  
        System.out.println("This is text from first thread 2");  
        System.out.println("This is text from first thread 3");  
        System.out.println("This is text from first thread 4");  
        System.out.println("This is text from first thread 5");  
        System.out.println("This is text from first thread 6");  
        super.run();  
    }  
}
```

دوره آموزش جاوا

کلیه حقوق متعلق به وب سایت نردبان است.

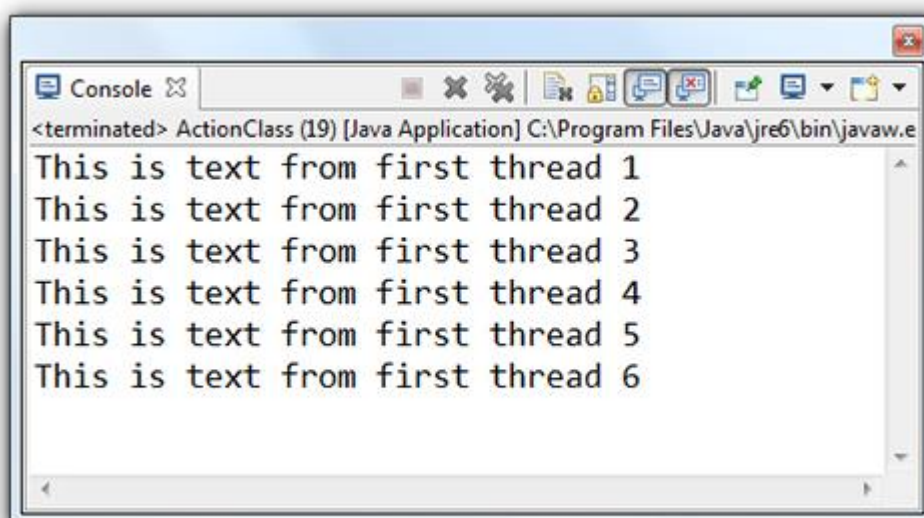
مدرس: بهزاد مرادی

همانطور که قبلاً توضیح داده شد دو روش متفاوت برای ایجاد یک Thread در زبان برنامه نویسی جاوا وجود دارد که یکی ارث بری از کلاس Thread جاوا است و دیگری استفاده از Runnable است. در مثال بالا از روش اول استفاده کرده ایم به این صورت که با نوشتن کلید واژه extends سپس نوشتن نام کلاس Thread کلاس خود را مجبور به ارث بری از کلاس Thread کرده ایم. قبلاً توضیح دادیم که هرآنچه بخواهیم در این کلاس اجرا شود می بایست داخل متد run قرار گیرد. خروجی Thread یی که در بالا ایجاد کرده ایم نمایش شش عبارت در پنجره Console است.

اکنون برای آنکه بتوانیم این Thread را اجرا کنیم می بایست کلاس دیگری تحت عنوان ActionClass ایجاد کرده و این بار گزینه public static void main را تیک بزیم چرا که این کلاس به منزله نقطه شروع برنامه جاوای ما خواهد بود:

```
public class ActionClass {  
  
    public static void main(String[] args) {  
        HowToControlThreads myObject1 = new HowToControlThreads();  
        myObject1.start();  
    }  
}
```

داخل متد main یک شیء تحت عنوان object از روی کلاس HowToControlThread ایجاد کرده سپس متد start را به آن ضمیمه می کنیم تا Thread ما اجرا شود. حال می توانیم برنامه خود را اجرا کنیم:



دوره آموزش جاوا

کلیه حقوق متعلق به وب سایت نردبان است.

مدرس: بهزاد مرادی

می بینیم دستوراتی که داخل Thread نوشته بودیم به ترتیب اجرا شدند.

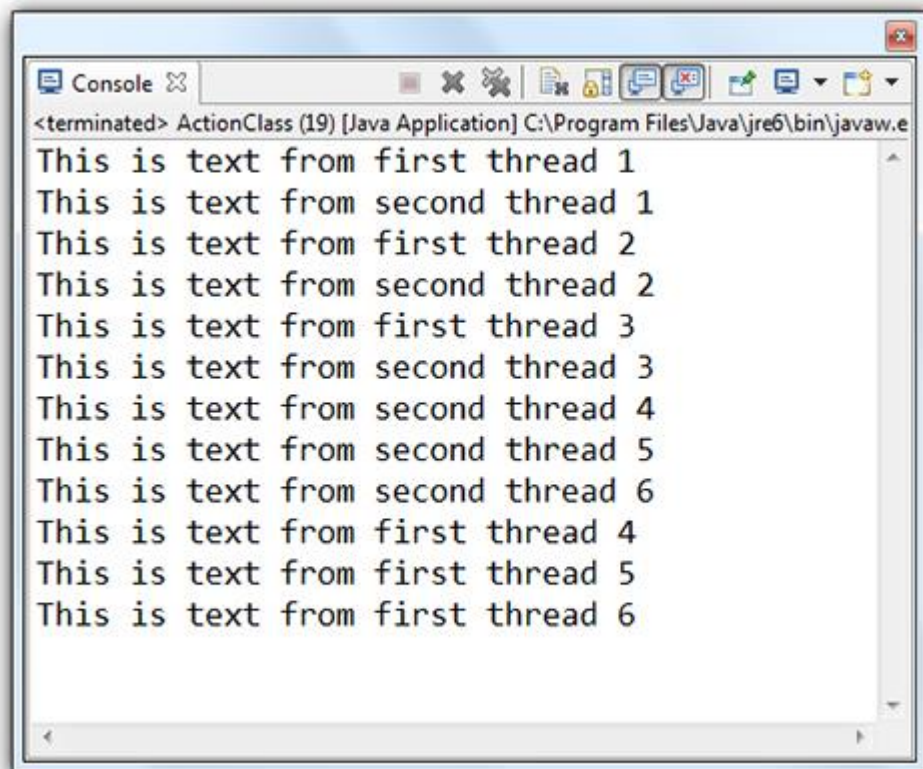
اکنون یک Thread دیگر به صورت زیر ایجاد می کنیم:

```
public class MySecondThread extends Thread {  
  
    @Override  
    public void run() {  
  
        System.out.println("This is text from second thread 1");  
        System.out.println("This is text from second thread 2");  
        System.out.println("This is text from second thread 3");  
        System.out.println("This is text from second thread 4");  
        System.out.println("This is text from second thread 5");  
        System.out.println("This is text from second thread 6");  
        super.run();  
    }  
}
```

همانطور که می بینیم کلاسی تحت عنوان MySecondThread ایجاد کرده و آن را از کلاس Thread جاوا extends کرده ایم. سپس داخل متد run این Thread شش دستور نوشته ایم تا داخل پنجره Console به نمایش درآیند (لازم به ذکر است که این بار از واژه second به معنی دوم استفاده کرده ایم تا متوجه شویم که از Thread دوم است). حال می بایست تا چیزی از روی این کلاس داخل کلاس ActionClass ایجاد کنیم:

```
public class ActionClass {  
  
    public static void main(String[] args) {  
        HowToControlThreads myObject1 = new HowToControlThreads();  
        myObject1.start();  
  
        MySecondThread myObject2 = new MySecondThread();  
        myObject2.start();  
    }  
}
```

همانطور که می بینیم چیزی تحت عنوان myObject2 از روی کلاس MySecondThread ایجاد کرده سپس برای آنکه بتوانیم Thread داخل آن را اجرا کنیم، متد start را به نام شیء ساخته شده از روی آن کلاس ضمیمه کرده ایم. حال برنامه خود را مجدد اجرا می کنیم:



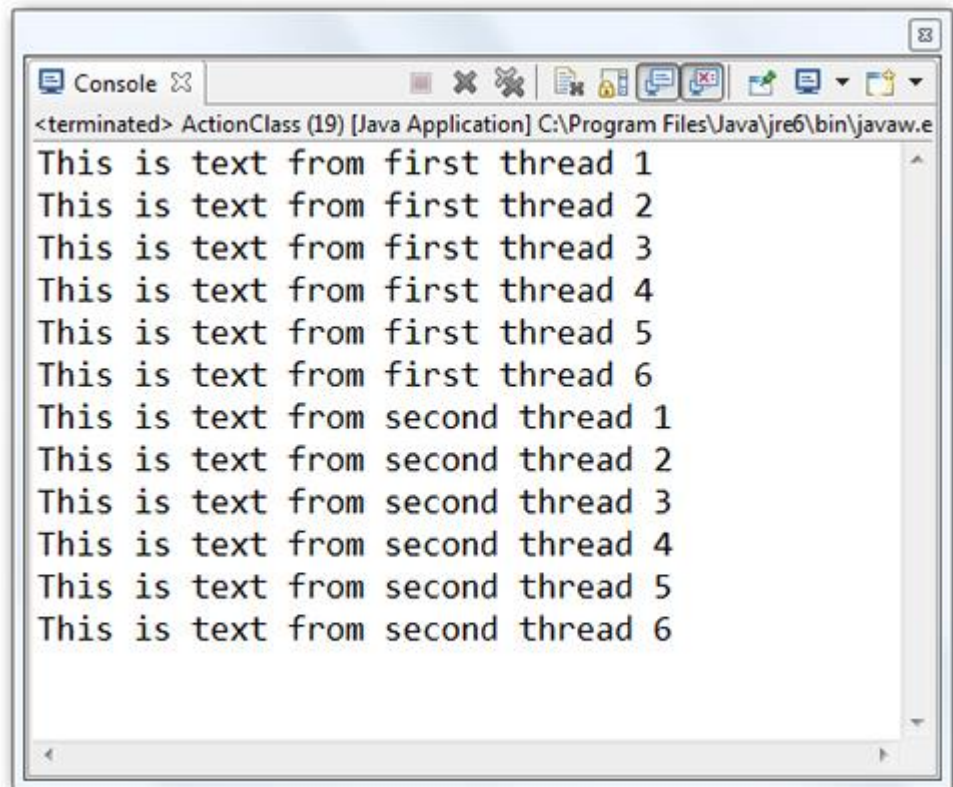
```
<terminated> ActionClass (19) [Java Application] C:\Program Files\Java\jre6\bin\javaw.e
This is text from first thread 1
This is text from second thread 1
This is text from first thread 2
This is text from second thread 2
This is text from first thread 3
This is text from second thread 3
This is text from second thread 4
This is text from second thread 5
This is text from second thread 6
This is text from first thread 4
This is text from first thread 5
This is text from first thread 6
```

می بینیم که هر دو Thread اجرا شده و اجرای آنها به این شکل است که هر کدام شانس بیشتری برای اجرا پیدا کنند اول اجرا خواهد شد. یک بار دیگر برنامه را اجرا می کنیم:

دوره آموزش جاوا

کلیه حقوق متعلق به وب سایت نردبان است.

مدرس: بهزاد مرادی



```
<terminated> ActionClass (19) [Java Application] C:\Program Files\Java\jre6\bin\javaw.e
This is text from first thread 1
This is text from first thread 2
This is text from first thread 3
This is text from first thread 4
This is text from first thread 5
This is text from first thread 6
This is text from second thread 1
This is text from second thread 2
This is text from second thread 3
This is text from second thread 4
This is text from second thread 5
This is text from second thread 6
```

می بینیم که در اجرا دوم Thread ها به ترتیب اجرا شده اند و این مسئله کاملاً تصادفی است. نکته جدیدی که در این آموزش می خواهیم مد نظر قرار دهیم این است که بینیم به چه صورت می توانیم کنترل اجرای Thread ها را در دست بگیریم. برای این منظور نیاز است تا از Interface `ExecutorService` استفاده کنیم. برای این منظور کد فوق را به صورت زیر تکمیل می کنیم:

```

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

public class ActionClass {

    public static void main(String[] args) {

        HowToControlThreads myObject1 = new HowToControlThreads();
        // myObject1.start();

        MySecondThread myObject2 = new MySecondThread();
        // myObject2.start();

        ExecutorService myController = Executors.newSingleThreadExecutor();
        myController.submit(myObject1);
        myController.submit(myObject2);

    }
}

```

اولین کاری که انجام می دهیم این است که اجرای Thread ها را بایستی متوقف کنیم. برای این منظور متدهای start مرتبط با نام شیء های ساخته شده از روی هر دو Thread را کامنت می کنیم.

سپس همانطور که می بینیم ابتدا نام Interface یی تحت عنوان ExecutorService را نوشته و نامی همچون myController برای آن در نظر می گیریم. سپس یک علامت مساوی قرار داده و نام کلاسی تحت عنوان Executors را می نویسیم. کاری که این کلاس انجام می دهد این است که حاوی تعدادی متد از پیش تعریف شده است که این امکان را به ما می دهند تا بتوانیم رفتار Thread های خود را کنترل کنیم. در ادامه پس از نوشتن نام کلاس Executors یک نقطه قرار داده و متدی همچون newSingleThreadExecutor را انتخاب می کنیم. همانطور که در تصویر زیر مشخص است به محض قرار دادن یک نقطه پس از کلاس Executors به کلیه متدهای این کلاس دسترسی خواهیم داشت:

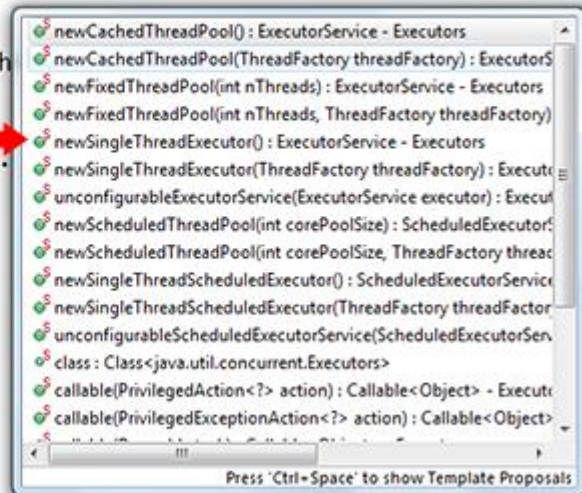
```

main(String[] args) {
    reads myObject1 = new HowToControlThreads();
    t();

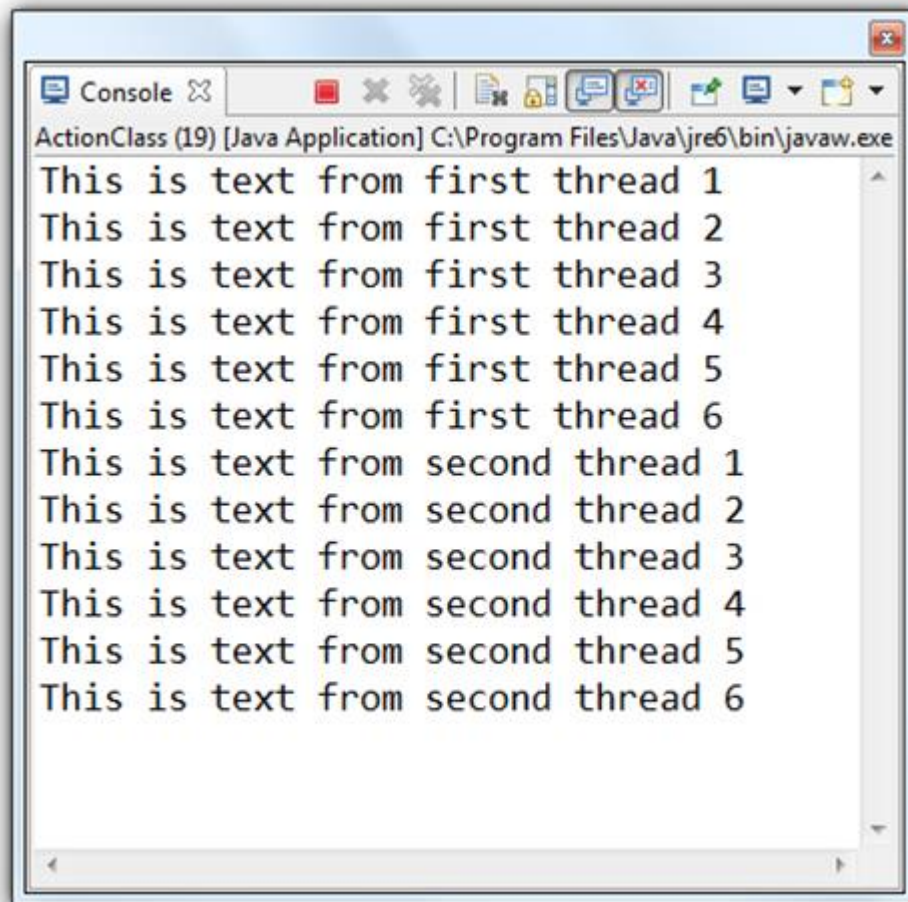
    myObject2 = new MySecondTh
    t();

    e myController = Executors.
    submit(myObject1);
    submit(myObject2);
}

```



در این مرحله برای آنکه بتوانیم Thread های خود را به ترتیب خاصی اجرا کنیم ابتدا بایستی نام شیئی ساخته شده از روی Interface یی تحت عنوان `ExecutorService` را نوشته سپس متدی تحت عنوان `submit` را به آن ضمیمه کنیم. سپس پارامتر این متد را بایستی نام شیئی های ساخته شده از روی Thread ها قرار دهیم. حال مجدد برنامه خود را اجرا می کنیم:



```
Console
ActionClass (19) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe
This is text from first thread 1
This is text from first thread 2
This is text from first thread 3
This is text from first thread 4
This is text from first thread 5
This is text from first thread 6
This is text from second thread 1
This is text from second thread 2
This is text from second thread 3
This is text from second thread 4
This is text from second thread 5
This is text from second thread 6
```

می بینیم که هر یک از Thread ها به ترتیبی که در متد submit آمده است اجرا شده اند و دیگر این احتمال وجود ندارد که هر Thread یی که شانس اول را برای اجرا شدن پیدا کند اول اجرا شود. اگر بخواهیم این مسئله را با حالت پیش مقایسه کنیم می توانیم کد خود را به صورت زیر ویرایش کرده سپس برنامه را اجرا کنیم:

دوره آموزش جاوا

کلیه حقوق متعلق به وب سایت نردبان است.

مدرس: بهزاد مرادی


```

public class ActionClass {

    public static void main(String[] args) {

        HowToControlThreads myObject1 = new HowToControlThreads();
        myObject1.start();

        MySecondThread myObject2 = new MySecondThread();
        myObject2.start();

        // ExecutorService myController = Executors.newSingleThreadExecutor();
        // myController.submit(myObject1);
        // myController.submit(myObject2);

    }
}

```

همانطور که می بینیم ابتدا متدهای start را از حال کامنت در آورده سپس ExecutorService را کامنت می کنیم و مجدد برنامه خود را اجرا می کنیم:

```

<terminated> ActionClass (19) [Java Application] C:\Program Files\Java\jre6\bin\
This is text from second thread 1
This is text from second thread 2
This is text from second thread 3
This is text from second thread 4
This is text from first thread 1
This is text from first thread 2
This is text from second thread 5
This is text from first thread 3
This is text from second thread 6
This is text from first thread 4
This is text from first thread 5
This is text from first thread 6

```

می بینیم که Thread ها به صورت تصادفی اجرا شده اند (لازم به ذکر است که پس از اجرای برنامه با این کد ممکن است مجدد هر یک از Thread ها به ترتیب اجرا شوند و نتایجی مشابه نتایج پیش را مشاهده کنیم اما این در حالی است که چنین نتیجه ای کاملاً تصادفی بوده و در صورتیکه برنامه خود را چند بار پشت سر هم اجرا کنیم هر دفعه می توانیم انتظار نتیجه متفاوتی را داشته باشیم).

در خاتمه بایستی گفت که متدهای مرتبط با کلاس Executors زیاد هستند و بسته به نیازی که در آینده خواهیم داشت آنها را مورد بررسی قرار خواهیم داد.

پس از مطالعه این آموزش انتظار می رود بتوانیم به سؤالات زیر پاسخ بدهیم:

۱. مزیت استفاده از ExecutorService در یک برنامه جاوا چیست؟
۲. چرا هر زمانیکه Thread های خود را اجرا می کنیم نتیجه متفاوتی را می بینیم؟
۳. وظیفه متد submit چیست؟