آموزش زبان برنامه نویسی جاوا

چندنخی-Multithreading

جلسه سی و ششم

نویسنده : رحمان زارعی

جاوا را ساده،آسان و شیرین بنوشید!!!!

همین الان که پشت کامپیوتر خود نشستید و دارید این جلسه از آموزش های جاوا را مطالعه می کنید،سیستم شما دارد چند عمل را همزمان انجام میدهد!! مثلا برنامه PDFReader شما در حال اجرای این کتاب الکترونیکی هستش و احتمالا دارید با یک پخش کننده، موسیقی مورد علاقه خود را گوش می کنید و همزمان مرورگر فایرفاکس رو باز کردید و نکته ای از این کتاب را در گوگل سرچ می زنید و ....کامپیوتر شما همه این کارها رو براتون همزمان انجام می دهد بدون این که شما متوجه شوید!!! یعنی چندین عملیات بصورت همزمان!! چندنخی (Multithreading) در جاوا نیز به همین مفهوم اشاره دارد یعنی اومده که به ما کمک کنه بتوانیم برنامه ای بنویسیم که توانایی این رو داشته باشه که چندین عملیات رو همزمان برامون انجام دهد.

جاوا یک زبان برنامه نویسی چندنخی (Multithreading) است ، یعنی ما می توانیم برنامه های چندنخی (Multithreading) ،برنامه هایی که چندین عملیات را همزمان با هم برامون انجام میدن را بنویسیم و توسعه دهیم.

یک برنامه چندنخی شامل دو یا چند بخش است که می توانند بصورت همزمان اجرا شوند.و هر بخش در همان زمان اجرا می توانند کار متفاوتی را انجام دهند.

من نمی خوام زیاد وارد جزییات کار thread ها شوم!!! بیشتر هدف اینه که کاربردشون چیه و چطور ازشون در برنامه مون استفاده کنیم!!جهت آگاهی از جزییات thread ها در گوگل سرچ کنید!!!منابع در این مورد زیاد!! ما میخوایم سریع بریم شراغ کدنویسی ☺

همان طور که گفتیم ،چندنخی (Multithreading) به اجرای همزمان چند بخش از یک برنامه اشاره دارد، این یعنی ما برای هر بخش از برنامه خود یک thread تعریف کرده و این thread ها هستند که همزمان در حال اجرا هستند!!! به همین دلیل به آن Multithreading می گویند یعنی اجرای همزمان چند thread در برنامه مون. اگر تا اینجا متوجه نشدید اصلا نگران نباشید سعی می کنیم این مبحث چغر!!! رو به هر ترتیبی شده یاد بگیریم ☺

از مزایا و کاربرد های thread ها در جاوا :

۱. چندنخی (Multithreading) کاربر را محدود نمی کند،چرا که تردها (thread ها) هنگام اجرا مستقل از هم هستند و می توانند عملیات های متعدد را همزمان انجام بدهند.
۲. با thread ها شما می توانید چندین عملیات را باهم انجام بدید و این باعث صرفه جویی در زمان می شود.
۳. همان طور که گفتیم thread ها مستقل از یکدیگر هستند و روی هم تاثیری ندارند مگر استثنایی رخ دهد.

مثال از دنیای واقعی

احتمالا پیش خودتون بگید چقدر داره توضیح میده چرا نمیره سراغ کدنویسی اینم شد آموزش؟! ☺

منم مثل خودتون بی تابم برای رفتن سراغ کدنویسی اما چه کنیم که باید ابتدا مفهوم و کاربرد thread برامون روشن بشه

خب تا اینجا گفتیم هر بخش از برنامه که بطور همزمان اجرا می شود یک thread می باشد و به اجرای همزمان چند بخش از برنامه یا چند thread ، چندنخی (Multithreading) می گویند.

برای درک بهتر مفهوم و کاربرد thread در برنامه نویسی یک مثال از دنیای واقعی برای شما می زنیم!!

مثال از یک انسان می زنیم!! یک انسان را در نظر بگیرید! یک انسان از چندین بخش تشکیل شده است!!! مثلا بخش شنوایی یا گوش انسان ،بخش بینایی یا چشم انسان،بخش دهان انسان،بخش قلب ،بخش مغز،بخش دست و....

وقتی شما با دوست خود در مورد یک موضوع در حال صحبت کردن هستید ، همزمان دارید دوستتون رو می بینید این یعنی بخش چشم داره کار میکنه، بعد همزمان دارید در مورد موضوع تک چرخ زدن جعفر در خیابان صحبت می کنی! این یعنی بخش دهان همزمان در حال کار و اجرا می باشد و همچنین با حرکات دستتون دارید اشاره هایی می کنید این یعنی بخش دست در حال اجرا می باشد و یهو هیجان زده میشید و تپش قلبتون میره بالا این یعنی بخش تپش قلب همزمان در حال اجرا در بدنتون می باشد!!! در این مثال چند بخش از بدن شما همزمان در حال اجرا و کار کردن بود و هر بخش مستقل از یکدیگر بدون این که تاثیری رو هم بگذارند همزمان در حال اجرا بودند یعنی شما همزمان با چشمتون می دیدید و با دهانتون حرف می زدید و با دستتون اشاره می کردید و تپش قلبتون بالا می رفت بدون این که هر بخش اختلالی در سایر بخش های بدنتون ایجاد کند.اگه بدن انسان را به یک برنامه کامپیوتری تشبیه کنیم! به هر بخش از بدن شما که به طور همزمان یک عملیاتی را انجام می دهند یک thread می گوییم.

در برنامه نویسی هم کاربرد thread همین گونه می باشد،یعنی ما قصد داریم برنامه مون رو از تک بعدی و تک کاره ای خارج کنیم و برنامه ای با استفاده از مفهوم چندنخی بنویسیم که توانایی اجرای همزمان چندین کار را داشته باشد.

مثلا در بدن انسان از مفهوم چندنخی استفاده شده است و ما می توانیم همزمان که قلبمون در حال تپش و پمپاژ خون و معده مان در حال هضم غذا و کلیه هامون در حال تصفیه خون و چشمون در حال دیدن و گوششمون در حال شنیدن هست خم شویم و گوشی هوشمند خود را برداریم و پست جعفر که در اینستاگرام گذاشته رو با انگشت شست لایک کنیم ☺ اگر در بدن انسان از مفهوم چندنخی استفاده نشده بود یک فاجعه رخ می داد!!! یعنی در ساده ترین حالت ممکن که داشتیم با دوستمون

صحبت می کردیم ابتدا بخش چشم دوستمون رو میدید بعد که کار دیدن تمام می شد یهویی چشمون بسته می شد و بخش گوش کار می کرد!! بعد که قصد جواب دادن به دوستمون رو داشتیم یهویی بخش گوش غیر فعال میشد و بخش دهان کار میکرد البته اگر شانس اینو داشته باشیم که قلبمون کار و کنه و متوقف نشده باشه ☺

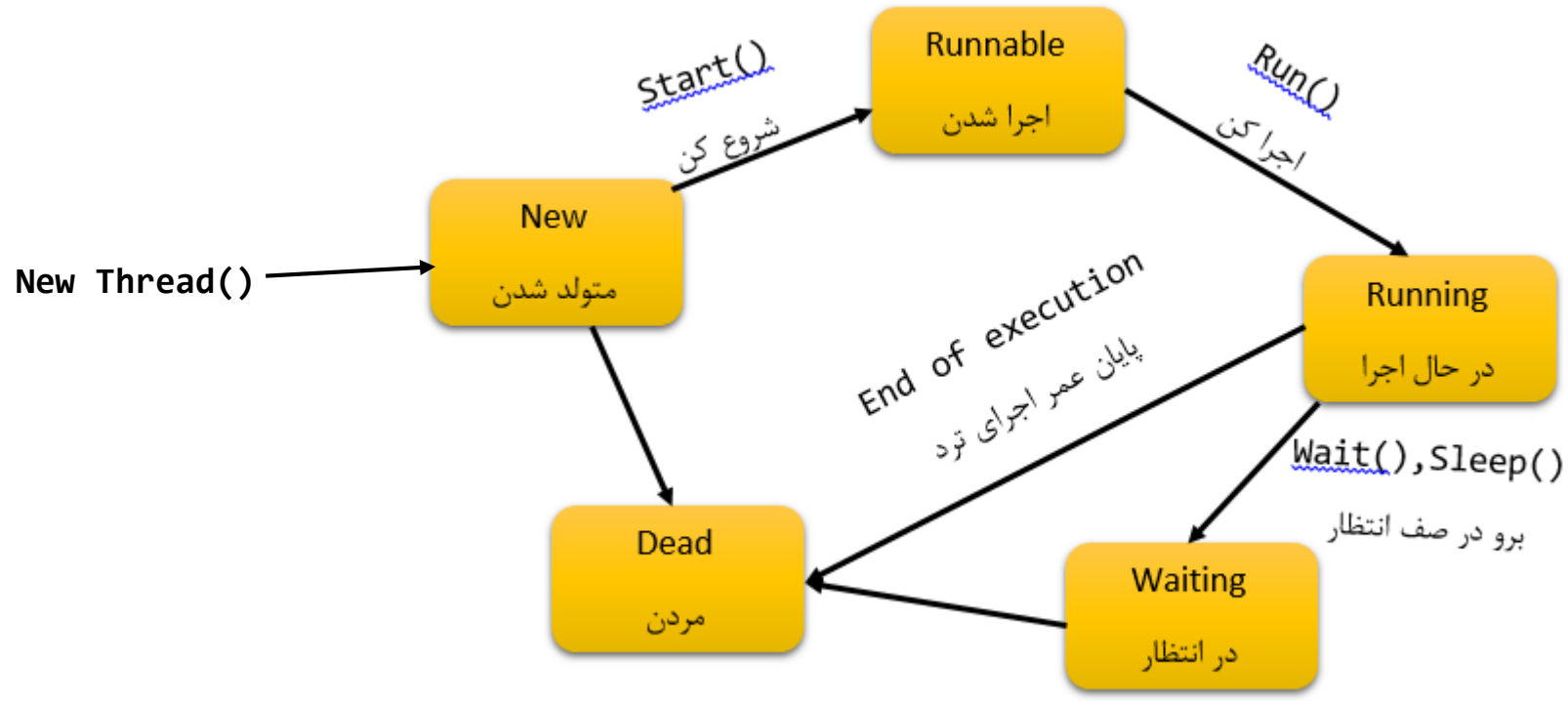
در برنامه نویسی هم مفهوم چندنخی (Multithreading) بسیار مهم و حیاتی برای برنامه ما می باشد. مثلا اگر قصد ساخت یک بازی را داشتیم باید از مفهوم چندنخی استفاده کنیم! چرا که یک بازی باید پاسخ گوی همزمان زدن دکمه های کیبورد توسط کاربر،حرکت اشیای درون بازی، گرافیک بازی و... باشد!!

امیدوارم توانسته باشم با مثال مفهوم چندنخی (Multithreading) را برای شما روشن کنم.

حالا می رویم سراغ مفهوم چندنخی (Multithreading) در زبان جاوا و چگونکی استفاده از آن هنگام کد نویسی ☺

چرخه حیات یک thread (Life Cycle of a Thread) :

یک thread در چرخه حیات خود مراحل مختلفی را طی می کند. برای مثال یک thread متولد می شود،اجرا می شود و سپس می میرد.نمودار زیر چرخه کامل حیات یک thread را نشان می دهد:



در زیر مراحل چرخه حیات thread را بررسی می کنیم:

New (متولد شدن):

در این مرحله یک thread جدید ایجاد کرده و چرخه حیات thread ایجاد شده شروع می شود.در این مرحله thread فقط متولد یا ایجاد شده است و هیچ کار دیگه ای نمی کند و در همین حالت باقی می ماند.

Runnable(اجرا کردن):

در این مرحله thread تازه متولد شده به حالت آماده به اجرا در می آید. thread در این حالت برای اجرای وظیفه ای در نظر گرفته می شود.

Waiting (در حالت انتظار):

گاهی اوقات یک thread به حالت انتظار می رود.خب چه موقع؟! وقتی که یک thread دیگر در حال اجرای وظیفه ای باشد ،thread اول تا اتمام کار thread دوم به حالت انتظار یا wait می رود و بعد از این که thread دوم انجام وظیفه اش تمام شد thread اول دوباره Runnable یا به وضعیت اجرایی شدن می رود.

مثلا وقتی که دوست شما دارد باهاتون صحبت می کند ،اگر گوش شما را یک thread فرض کنیم که Runnable شده و در وضعیت running قرار گرفته و حرف های دوستتون را می شنود.همزمان که thread گوش شما در حال شنیدن هست ،thread دهان شما در حال wainting یا انتظار قرار گرفته که بعد از اتمام گوش دادن ، thread دهان به وضعیت Runnable در آمده و شروع به حرف زدن می کند و thread گوش به حالت waiting یا انتظار می رود ☺

Dead (مردن یا خاتمه):

یک thread ، Runnable شده هنگامی به وضعیت Dead می رود یا خاتمه پیدا می کند که وظیفه ای که بر دوش داشته تمام شده باشد.

پیاده سازی Thread در جاوا

پیاده سازی thread در جاوا به این گونه است که ما می توانیم کلاس های خود را به یک thread تبدیل سازیم.یعنی کاری کنیم که کلاس های ما یک thread در نظر گرفته شوند.

ایجاد یک Thread با پیاده سازی اینترفیس Runnable :

برای این که کلاس شما یک thread در نظر گرفته شود کافیست در کلاس خود اینترفیس Runnable را پیاده سازی کنید.یعنی کلاس خود را به اینترفیس Runnable ، **implements** کنید.

برای تبدیل کلاس خود به یک thread کافیست گام های زیر را بردارید:

گام اول:

کلاس خود را به اینترفیس Runnable ، implements کنید:

فرض کنید کلاسی بصورت زیر داشته باشیم:

```
package javalike;

public class Cat {

}
```

حالا قصد داریم کاری کنیم که کلاس Cat یک Thread درنظر گرفته شود.برای این کار کلاس Cat را به اینترفیس Runabble، implements می کنیم:

```
package javalike;

public class Cat implements Runnable{

}
```

گام دوم:

اینترفیس Runnable در بدنه خود یک متد با نام run() دارد که شکل نوشتن آن بصورت زیر است:

```
public void run();
```

حال طبق مبحث اینترفیس که در جلسات گذشته مطالعه کردیم ،هر کلاسی که یک اینترفیس را implements کرد بلاجبار باید متدهای درون اینترفیس را در بدنه خود پیاده سازی کند. خب ما در گام اول کلاس Cat را به اینترفیس Runnable، implements کردیم پس لازم است متد run() اینترفیسRunnable را در کلاس Cat پیاده سازی کنیم:

```
package javalike;

public class Cat implements Runnable{


	public void run() {


	}

}
```

همان طور که گفتیم یک thread هنگام ایجاد و در حالت اجرایی شدن وظیفه و عمل مشخصی را انجام می دهد. این وظیفه و عملی که برای thread مشخص می کنیم باید درون متد ()run قرار گیرد. نتیجه می گیریم که هر دستور لازم الاجرایی که برای thread مشخص می کنیم باید درون متد()run قرار گیرد.

حالا ما قصد داریم دستور چاپ پیام "Hello Javalike" را در متد ()run پیاده سازی کنیم:

```
package javalike;

public class Cat implements Runnable{


	public void run() {

		System.out.println("Hello Javalike");
	}

}
```

خب اگر ما thread خود را متولد کنیم ، بعدش به مرحله اجرایی شدن برسانیم آن وظیفه ای که thread ما انجام می دهد اجرای دستورات درون متد ()run می باشد و از آنجایی که دستور ما در درون متد ()run در مثال بالا چاپ پیام "Hello Javalike" می باشد ، در محیط کنسول پیام مذکور چاپ می شود.

خب تا این جا چطور بود؟ امیدوارم که واضع و روشن توضیح داده باشم البته هنوز تمام نشده!! بریم سراغ گام بعدی ☺

گام سوم:

در گام سوم نیاز داریم که Thread خود را متولد یا ایجاد کنیم!!! یا به عبارت دیگر در مرحله new قرارش دهیم.ما در اینجا از واژه تولد یا ایجاد Thread استفاده کردیم! در برنامه نویسی جاوا برای تولد یا ایجاد یک Thread باید از کلاس Thread شی ایجاد کنیم شبیه شی سازی در حالت عادی از یک کلاس! خب Thread هم یک کلاس آماده در پکیج java.lang می باشد.
پس برای برای تولد یک Thread کافیست از کلاس Thread یک نمونه یا شی ایجاد کنیم.
هنگام شی سازی از کلاس Thread نیاز به صدا زدن سازنده آن داریم.این کلاس سازنده های گوناگونی دارد.یکی از متداول ترین سازنده های آن که ازش استفاده می کنیم بصورت زیر است:

```
Thread(Runnable arg0, String arg1)
```

این سازنده دو پارامتر میگیرد، پارامتر اول یعنی arg0 نمونه ای (شی) از کلاسی است که اینترفیس Runnable را implements کرده باشد. مثلا از آنجایی که کلاس Cat اینترفیس Runnable را implements کرده ،شی از نوع کلاس Cat می تواند جایگزین پارامتر arg0 شود.
بجای arg1 نیز می توانید نامی به بچه Thread تازه متولد شده خود بدهید ☺
خب تغییراتی که به کد مثال کلاس Cat می دهیم بصورت زیر است:
یک متد main برای کلاس Cat تعریف می کنیم و یک شی از نوع کلاس Cat و یک شی از نوع کلاس Thread ایجاد می کنیم:

```
package javalike;

public class Cat implements Runnable{


	public void run() {

		System.out.println("Hello Javalike");
	}

	public static void main(String args[]) {
		Cat cat=new Cat();
	Thread thCat=new  Thread(cat, "jesi");


	}

}
```

- در اینجا یک شی با نام cat از کلاس Cat ایجاد کرده و درون پارامتر اول سازنده کلاس Thread قرار داده ایم.

پارامتر دوم سازنده Thread را یک مقدار از نوع String که نام Thread ما را مشخص می کند قرار داده ایم.

- شی thCat که از نوع کلاس Thread می باشد ، Thread تازه متولد شده ما را تشکیل می دهد.

نکته: گام سوم را در تصویر(۱) با رنگ سبز مشخص کرده ایم:

گام چهارم:

همان طور که گفتیم Thread یک کلاس می باشد که دارای ویژگی ها و متدهایی می باشد.یکی از متدهای کلاس Thread بصورت زیر است:

```
Void start();
```

بعد از ایجاد یک شی از نوع کلاس Thread و متولد شدن Thread جدید ، نیاز داریم که Thread مان شروع به اجرا شدن کند. برای این کار از طریق شی ساخته شده از نوع Thread متد start را صدا می زنیم با این کار Thread ما شروع به اجرای وظایف خود که در متد run() برایش مشخص کرده ایم می کند.

پس متد start باعث می شود دستورات درون متد run ، Thread ما اجرا شود.

تغییرات مثال بالا را در زیر مشاهده می کنید:

```
package javalike;

public class Cat implements Runnable {

	public void run() {

		System.out.println("Hello Javalike");
	}

	public static void main(String args[]) {
		Cat cat = new Cat();
		Thread thCat = new Thread(cat, "jesi");
		thCat.start();

	}

}
```

اگر برنامه بالا را کامپایل و اجرا کنیم خروجی بصورت زیر خواهد بود:

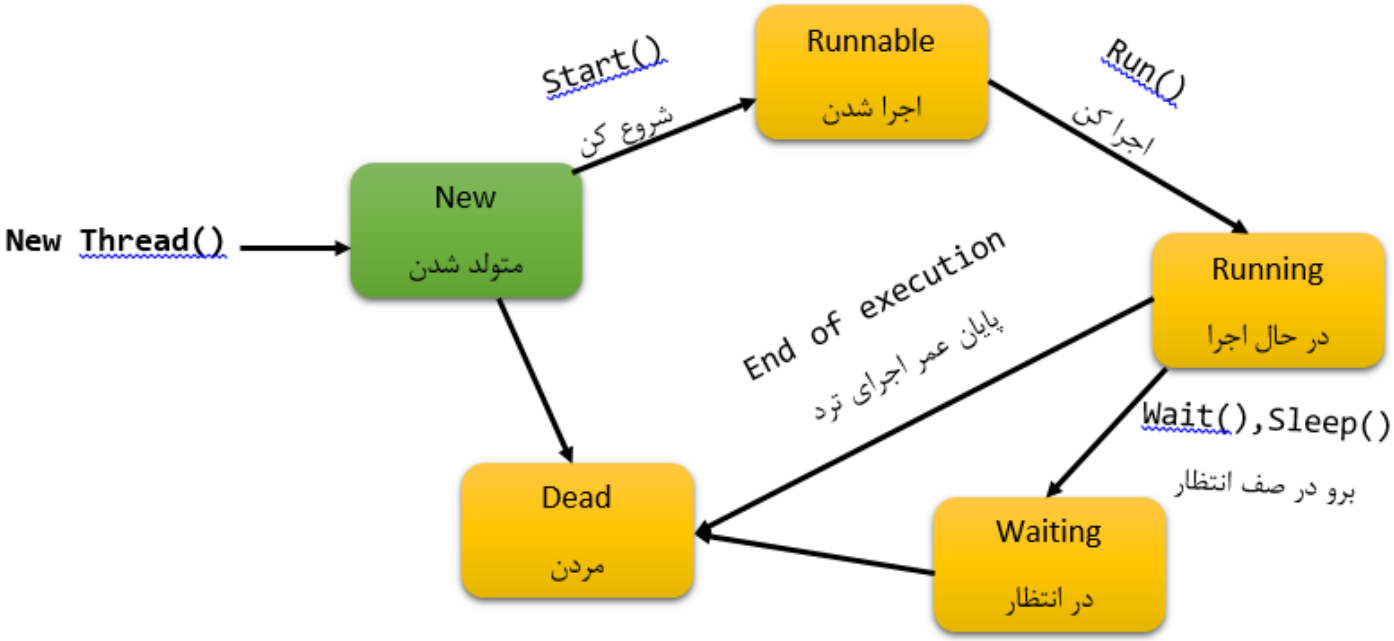
```
Hello Javalike
```

خلاصه ای از کارایی که کردیم :

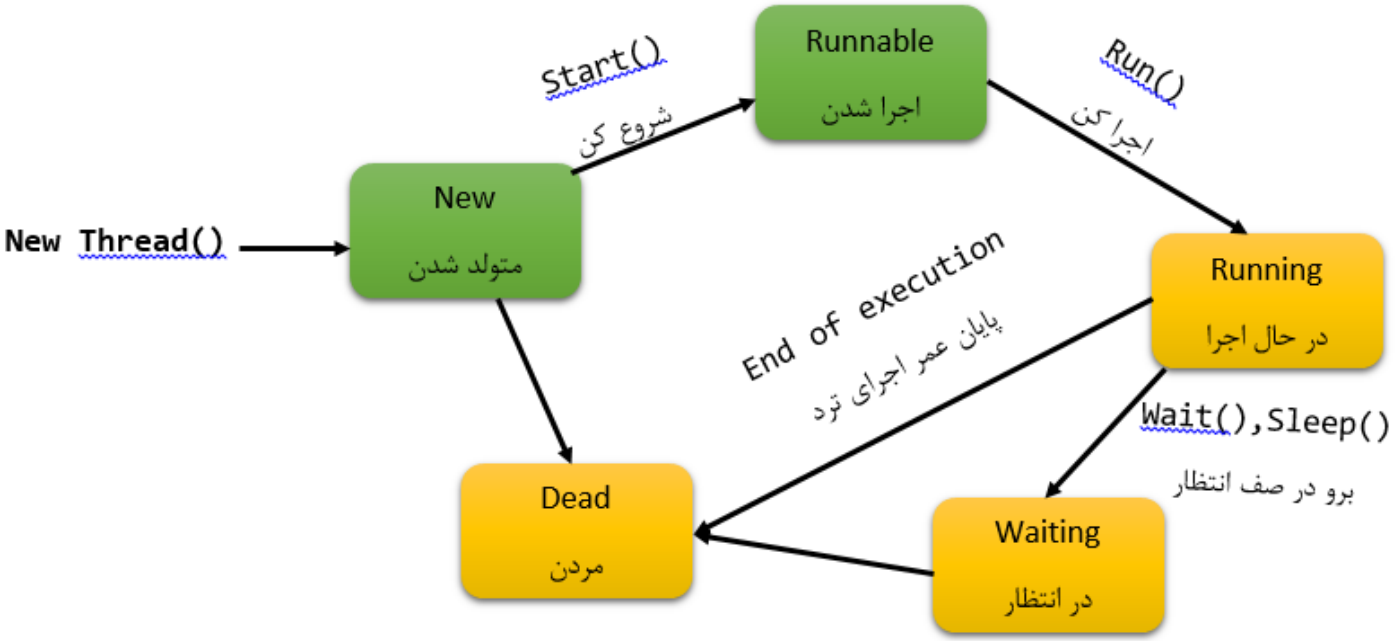
ما قصد داشتیم کاری کنیم که کلاس Cat یک Thread درنظر گرفته شود.برای این کار کلاس Cat را به اینترفیس Runnable، implements کردیم. با این کار باید متد run() موجود در اینترفیس Runnable را در کلاس خود پیاده سازی یا override می کردیم.بعد از پیاده سازی متد run در کلاس خود رفتیم سراغ دستوری که نیاز داشتیم هنگام اجرای Thread برامون اجرا شود! خب دستور مورد نظر خود را در متد run موجود در کلاس Cat پیاده سازی کردیم. تا اینجا کلاس Cat را تبدیل به یک Thread کردیم اما نیاز داشتیم که این Thread یعنی کلاس Cat را متولد کنیم! برای این کار از کلاس Cat یک شی ایجاد کردیم و بعدش رفتیم سراغ کلاس Thread و از آن هم یک شی ساختیم بعدش شی کلاس Cat را به عنوان پارامتر اول سازنده کلاس Thread قرار دادیم و پارامتر دوم سازنده کلاس Thread را یک مقدار از نوع String که نام Thread را تشکیل می داد قراردادیم.

تا اینجا مرحله تولد و ایجاد Thread را گذراندیم و نیاز داشتیم که Thread ما به مرحله اجرایی شدن برود برای این کار با استفاده از متد start ، Thread خود را استارت زده و به مرحله آغاز و اجرایی شدن وارد کردیم ، در این مرحله دیگه Thread ما شروع به اجرا کردن وظایف خود که از قبل درون متد run موجود در کلاس Cat پیاده سازی کرده بودیم را می کند.

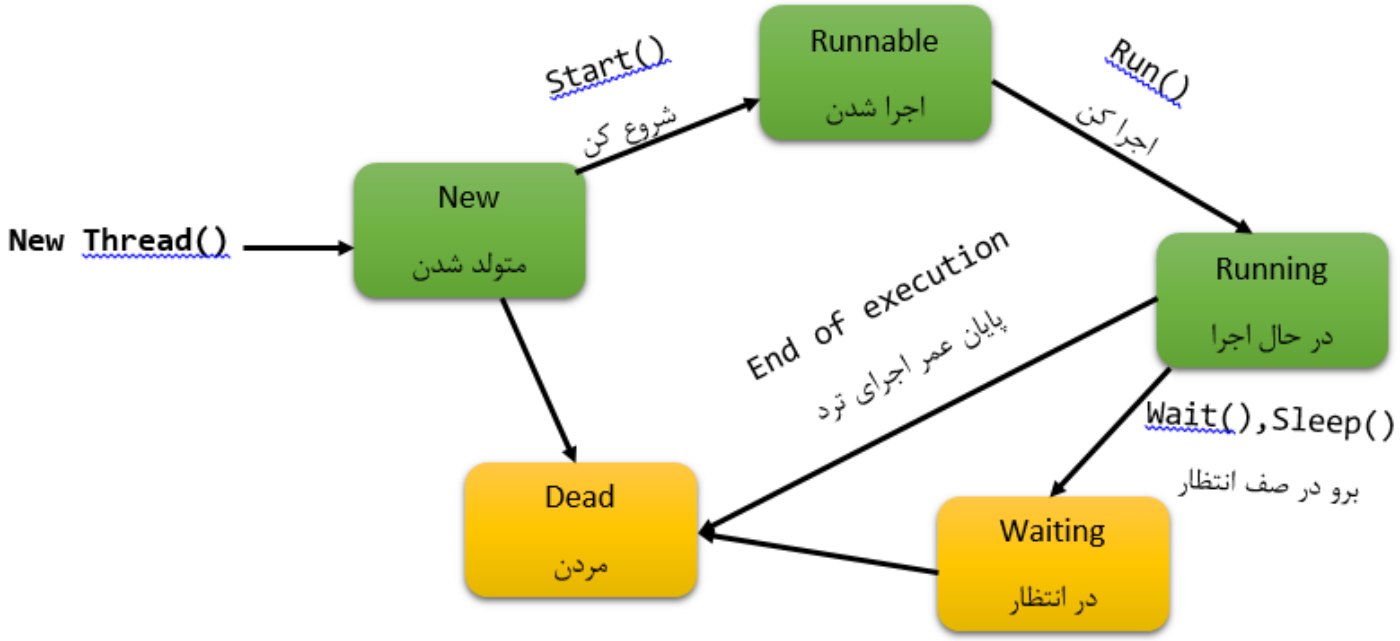
گام سوم و چهارم را در تصویر(۱) ، تصویر(۲) و تصویر(۳) مشاهده می کنید:



تصویر(۱)-مرحله ایجاد و تولد Thread جدید



تصویر(۲)- در این مرحله `Thread` ما با متد `start` به مرحله آماده به اجرا می رسد



تصویر(۴)- در این مرحله دستورات درون متد `run`، `Thread` ما در حال اجرا شدن می باشد.

پس بصورت خلاصه برای تبدیل کلاس خود به Threadبصورت زیر عمل می کنیم:

۱. کلاس خود را به اینترفیس Runnable ، implements می کنیم.
۲. متد run() را در کلاس خود پیاده سازی می کنیم.
۳. از کلاس خود شی می سازیم.
۴. از کلاس Thread شی ساخته و شی ایجاد شده از کلاس خود را به عنوان پارامتر به سازنده کلاس Thread می دهیم.
۵. با استفاده از متد start() ، Thread خود را به مرحله شروع و اجرا شدن می رسانیم و با اولیتی که سیستم عامل برای Thread شما برنامه ریزی کرده ، دستورات درون متد run() ، Thread شما شروع به اجرا شدن می کنند.

نکته: در بالا مورد ۵ به اولیت سیستم عامل برای اجرای Thread اشاره کردیم! این را بدانید که حتی بعد از استفاده از متد start اجرا شدن یا با تاخیر اجرا شدن و... Thread ما بستگی به اولیتی است که سیستم عامل برای Thread های ما درنظر و برنامه ریزی کرده است می باشد.

ما در جاوا خواندیم که هنگام اجرای برنامه ، خط به خط کدهای برنامه خوانده و اجرا می شود! اما در Thread ها اینگونه نیست! اجرا شدن Thread ها هنگام اجرای برنامه بستگی به اولیت بندی داره که سیستم عامل برای Thread های ما انجام داده است.مثلا اگر سه Thread داشته باشیم به نام های t2،t1 و t3 و ترتیب start خوردن این سه Thread در برنامه بصورت زیر باشد:

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

در اولین نگاه طبق قانون خط به خط اجرا شدن دستورات درون برنامه ، ابتدا باید دستورات Thread شماره یک و بعد دستورات Thread شماره دو و در نهایت دستورات Thread شماره سه اجرا شود، اما اینگونه نیست!!! ترتیب اجرای دستورات Thread ها را سیستم عامل مشخص می کند و یعنی امکان دارد با وجود start خوردن Thread شماره یک در ابتدا ، اول دستورات Thread شماره سه اجرا شود!!!! و بعدش دستورات Threadشماره یک و بعد Thread شماره دو.پس ترتیب اجرای Thread ها در دست ما نیست و سیستم عامل بر اساس اولیت Thread ها ، آنها را اجرا می کند.

برای درک بیشتر این موضوع به مثال زیر توجه کنید:

مثال:

```
package javalike;

class A implements Runnable {

	public void run() {

		System.out.println("Number1");
	}
}

class B implements Runnable {

	public void run() {

		System.out.println("Number2");
	}
}

class C implements Runnable {

	public void run() {

		System.out.println("Number3");
	}
}

public class Test {

	public static void main(String[] args) {
		A a = new A();
		B b = new B();
		C c = new C();
		Thread t1 = new Thread(a, "number1");
		Thread t2 = new Thread(b, "number2");
		Thread t3 = new Thread(c, "number3");
		t1.start();
		t2.start();
		t3.start();
	}

}
```

خروجی: بعد از کامپایل و اجرای برنامه خروجی بصورت زیر می باشد:

```
Number3
Number1
Number2
```

- در این مثال ما سه کلاس با نام های A,B,C داریم که اینترفیس Runnable را implements کرده اند. در واقع میشه گفت سه Thread با نام های A,B,C داریم.
- در کلاس Test از سه کلاس مذکور شی ساخته و همچنین سه بار از کلاس Thread شی ایجاد کرده و شی کلاس های A,B,C را به سازنده کلاس Thread های ساخته شده داده ایم.
- حال به ترتیب Thread های t1، t2 و t3 را با متد start ، استارت زده ایم! اما همان طور که در مورد تعیین اولیت اجرای Thread ها توسط سیستم عامل صحبت کردیم ، ابتدا به جای اجرای دستورات t1 ، Thread دستورات t3 ، Thread اجرا می شود بعدش t1 و در پایان دستورات t2 ، Thread اجرا می شود. پس نتیجه می گیریم ترتیب و حتی زمان اجرای Thread ها را سیستم عامل مشخص میکند. سیستم عامل دیگه دوست داره زورش میرسه ☺ بدن یک انسان رو در نظر بگیر مثلا فرض کن فردی دوست داشته باشه ترتیب اجرای بخش های بدنش به این صورت باشه که اول صحبت کنه بعدش قلبش کار کنه ! مگه میشه؟! نمیشه! ☺ جناب سیستم عامل هم حتما ی چیزایی میدونه که اولیت ها رو جابه جا میکنه ☺

- حال اگر دوباره برنامه مثال بالا را اجرا کنیم با خروجی متفاوتی روبرو می شویم:

```
Number1
Number2
Number3
```

و دوباره برنامه رو اجرا می کنیم:

```
Number1
Number3
Number2
```

- همان طور که مشاهده می کنید با هر بار اجرا کردن برنامه فوق خروجی متفاوتی داریم، دلیلش اینه که از Thread استفاده کردیم ، و برای اولیت و زمان اجرا Thread ها سیستم عامل تصمیم می گیرد.

راه دوم تبدیل کلاس خود به یک Thread....

ایجاد Thread با استفاده از به ارث بردن کلاس Thread :

راه دوم تبدیل کلاس خود به یک Thread این است که کلاس ما ، کلاس Thread را به ارث ببرد.برای تبدیل کلاس خود به یک Thread از طریق به ارث بردن کلاس Thread قدم های زیر را برمی داریم:

قدم اول:

کلاس ما ، کلاس Thread را به ارث ببرد. فرض کنید کلاسی بصورت زیر داریم:

```
package multithreading_Javalike;

public class A {

}
```

حالا قصد داریم کلاس فوق را به یک Thread تبدیل سازیم.برای این کار بصورت زیر عمل می کنیم:

```
package multithreading_Javalike;

public class A extends Thread{

}
```

در اینجا با استفاده از به ارث بردن کلاس Thread ، کلاس ما یعنی کلاس A فرزند کلاس Thread شده و به ویژگی ها و متدهای کلاس Thread دسرسی پیدا می کند.تا اینجا اولین قدم برای تبدیل کلاس خود به Thread را برداشته ایم.

قدم دوم:

کلاس Thread دارای متدی با نام run() هستش که شکل نوشتن آن بصورت زیر است:

```
public void run( )
```

همان طور که پیش تر گفتیم وظایف و دستوراتی را که به یک Thread می خواهیم نسبت دهیم در متد run() پیاده سازی می کنیم.

خب از آنجایی که کلاس ما ،کلاس Thread را به ارث برده است ، می توانیم متد run() را در کلاس خود override کنیم:

```
package multithreading_Javalike;

public class A extends Thread {
     public void run() {
}
      }
```

حالا دستور مورد نظر خود را درون متد run() موجود در کلاس خود پیاده سازی می کنیم:

```
package multithreading_Javalike;

public class A extends Thread {
	public void run() {

		for(int i=1;i<=5;i++){
			System.out.println("counter= "+i);
		}
	}
}
```

قدم سوم:

در این مرحله از کلاس خود شی می سازیم. همچنین از کلاس Thread شی ایجاد می کنیم. حالا شی ایجاد شده از کلاس خود را جایگزین پارامتر اول سازنده کلاس Thread می کنیم:

```
A a=new A();
Thread ta=new Thread(a, "A");
```

برای پارامتر دوم یک مقدار از نوع String درنظر می گیریم:

```
package multithreading_Javalike;

public class A extends Thread {
	public void run() {

		for(int i=1;i<=5;i++){
			System.out.println("counter= "+i);
		}
	}

	public static void main(String[] args) {
		A a=new A();
		Thread ta=new Thread(a, "A");
			}
}
```

تا اینجا Thread خود را ایجاد و متولد کرده ایم!!! ta نام Thread ما می باشد.که با اجرا شدن آن دستورات درون متد run کلاس A اجرا می شود.

کلاس Thread در بدنه خود همان طور که قبل گفتیم دارای متدی با نام start() می باشد. برای اجرای دستورات درون متد run() موجود در Thread خود، کافیست از طریق شی ساخته شده از کلاس Thread متد start() را صدا بزنیم، با این کار تمامی دستورات درون متد run() اجرا می شود:

```
ta.start();
```

برنامه ما بصورت زیر خواهد بود:

```
package multithreading_Javalike;

public class A extends Thread {
	public void run() {

		for(int i=1;i<=5;i++){
			System.out.println("counter= "+i);
		}
	}

	public static void main(String[] args) {
		A a=new A();
		Thread ta=new Thread(a, "A");
		ta.start();
	}
}
```

خروجی: بعد از کامپایل و اجرای برنامه خروجی بصورت زیر می باشد:

```
counter= 1
counter= 2
counter= 3
counter= 4
counter= 5
```

خب تا اینجا با دو روش ایجاد Thread در برنامه نویسی جاوا آشنا شدیم.حالا متدهای موجود در کلاس Thread را بررسی می کنیم:

```
public void run()
```

- دستورات و وظایفی که می خواهیم یک Thread اجرا کند درون این متد قرار می دهیم.

```
public void start()
```

- برای اجرای دستورات درون متد run() موجود در Thread از این متد استفاده می کنیم.

public final void setName(String name)

وقتی از کلاس Thread شی ایجاد کردیم از طریق شی ایجاد شده می توانیم این متد را صدا زده و برای Thread خود نامی را برگزینیم. متد getName() نام Thread را برای ما برمیگرداند.

public final void join()

اگه یادتون باشه گفتیم که ترتیب و اولویت اجرای Thread ها در دست سیستم عامل می باشد و اگر چند Thread داشته باشیم مشخص نیست کدومش زودتر اجرا می شود. اما دست نگهداریم!!! سیستم عامل هر چقدر هم که گردن کلفت باشه! آخر راهی باید باشه که حداقل یک تاثیری در اجرای Thread ها داشته باشیم! اگر Thread ما متد join() را صدا بزنه ، تا زمانی که وظایف و دستوراتش تمام نشده است سیستم عامل یهویی سراغ اجرای Thread دیگر نمی رود. در کل متد join() باعث میشه تا اتمام دستورات یک Thread ، سیستم عامل برای اجرای سایر Thread ها دست نگهدارد.

چه چیزی بهتر از مثال!!

کد زیر را مشاهده کنید:

```
package javalike;

class A implements Runnable {

	public void run() {
		for (int i = 1; i <= 5; i++)
			System.out.println("Counter= " + i);
	}
}

public class Test {

	public static void main(String[] args) {
		A a1 = new A();
		A a2 = new A();
		A a3 = new A();
		A a4 = new A();

		Thread t1 = new Thread(a1, "number1");
		Thread t2 = new Thread(a2, "number2");
		Thread t3 = new Thread(a3, "number3");
		Thread t4 = new Thread(a4, "number3");

		t1.start();
```

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

        }

}
```

خروجی:

تذکر: چون در برنامه از Thread استفاده کردیم خروجی این برنامه با هربار اجرا متفاوت خواهد بود.

```
Counter= 1
Counter= 1
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 1
Counter= 2
Counter= 2
Counter= 3
Counter= 5
Counter= 2
Counter= 4
Counter= 5
Counter= 3
Counter= 4
Counter= 5
Counter= 3
Counter= 4
Counter= 5
```

همان طور که مشاهده می کنید ترتیب اجرای شمارنده که از ۱ تا ۵ بوده به هم خورده! یعنی اگر از Thread استفاده نکرده بودیم خروجی می بایست بصورت زیر باشد:

```
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
```

```
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
```

خب چرا ترتیب شمارنده بهم هم خورده؟! دلیلش اینه که دستورات ما درون یک Thread قرار گرفته و ما چهار Thread ساخته ایم :

```
A a1 = new A();
A a2 = new A();
A a3 = new A();
A a4 = new A();

Thread t1 = new Thread(a1, "number1");
Thread t2 = new Thread(a2, "number2");
Thread t3 = new Thread(a3, "number3");
Thread t4 = new Thread(a4, "number3");
```

و به ترتیب زیر آنها را start زده ایم:

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

خوندیم که اولیت اجرای Thread ها در دست سیستم عامل می باشد. خب در اینجا هم سیستم عامل میاد شروع به اجرای دستورات مثلا ترد (Thread) t1 میکنه ، یهویی در وسط اجرا ، ترد (Thread) t1 را میبره به حالت انتظار (wait) و شروع به اجرای دستورات ترد t3 میکنه و باز وسط اجرا t3 رو میبره به حالت انتظار و میره سراغ t2 و... همین جور تا این که کل دستورات ترد(Thread) ها را اجرا کند. پس تا اینجا ترتیب و اولیت اجرای تردها در دست ما اصلا نبود اما این که نشد برنامه ما دوست داریم حداقل اختیار یکی از تردها رو در دست بگیریم! برای این کار از متد join استفاده می کنیم:

```
package javalike;

class A implements Runnable {

	public void run() {
		for (int i = 1; i <= 5; i++)
			System.out.println("Counter= " + i);
	}
}

public class Test {

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

```
            A a1 = new A();
            A a2 = new A();
            A a3 = new A();
            A a4 = new A();

            Thread t1 = new Thread(a1, "number1");
            Thread t2 = new Thread(a2, "number2");
            Thread t3 = new Thread(a3, "number3");
            Thread t4 = new Thread(a4, "number3");

            t1.start();
            try {
                  t1.join();
            } catch (InterruptedException e) {

                  e.printStackTrace();
            }
            t2.start();
            t3.start();
            t4.start();

      }

}
```

خروجی:

```
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
Counter= 1
Counter= 2
Counter= 1
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
Counter= 2
Counter= 3
Counter= 4
Counter= 5
Counter= 3
Counter= 4
Counter= 5
```

این مقادیر خروجی که با رنگ آبی مشخص شده است مربوط به ترد t1 می باشد.

- در این برنامه ما ترد t1 را بعد از start زدن ، متد join() را نیز برایش صدا زدیم. با این کار به سیستم عامل می گوییم که تا زمانی که دستورات ترد t1 تمام نشده است حق نداری وسط کار بری سراغ اجرای دستورات تردهای دیگر.
- همان طور که مشاهده میکنید شمارنده برای تردهای t2 تا t3 نامرتب و به هم ریخته می باشد. البته ما می توانیم با استفاده از متد join کاملا خودمون اجرای تردها رو اولیت بندی کنیم! مثلا کاری کنیم ابتدا ترد t1 بعد t2 تا t4 به ترتیب اجرا شوند:

```
package javalike;

class A implements Runnable {

   public void run() {
         for (int i = 1; i <= 5; i++)
               System.out.println("Counter= " + i);
   }
}

public class Test {

   public static void main(String[] args) {
         A a1 = new A();
         A a2 = new A();
         A a3 = new A();
         A a4 = new A();

         Thread t1 = new Thread(a1, "number1");
         Thread t2 = new Thread(a2, "number2");
         Thread t3 = new Thread(a3, "number3");
         Thread t4 = new Thread(a4, "number3");

         t1.start();
         try {
               t1.join();
         } catch (InterruptedException e) {

               e.printStackTrace();
         }
         t2.start();
         try {
               t2.join();
         } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
            t3.start();
            try {
                t3.join();
            } catch (InterruptedException e) {

                e.printStackTrace();
            }

            t4.start();
            try {
                t4.join();
            } catch (InterruptedException e) {

                e.printStackTrace();
            }
    }

}
```

خروجی:

```
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
```

- همان طور که مشاهده میکنید به ترتیب تردها اجرا شده اند و شماره های شمارنده مرتب می باشد.
- این است دنیای تردها (Thread) ☺
- دلیل این که متد `join` را در بلوک `try-catch` قرار دادیم کنترل رخ دادن استثنای احتمالی می باشد.

```
public static void sleep(long millisec)
```

- این متد همان طور که از اسمش پیداست، برای به خواب بردن یک Thread برای مدت زمان مشخص می باشد.
- این متد زمان مشخص بر اساس میلی ثانیه می باشد. که طریق پارامتر متد به آن داده می شود.
- هر ۱۰۰۰ میلی ثانیه ، یک ثانیه می باشد.
- گاهی نیاز داریم در برنامه خود مکث هایی ایجاد کنیم برای این کار از این متد استفاده می شود.
- مثلا در ساخت بازی برای کنترل سرعت اجرای بازی از این متد استفاده می شود.
- متد sleep یک متد استاتیک هستش و مستقیم می توانیم از طریق نام کلاس آن یعنی Thread آن را صدا بزنیم:

```
Thread.sleep(600);
```

مثال:

```
package javalike;

class A implements Runnable {

	public void run() {
		for (int i = 1; i <= 5; i++) {
			try {
				Thread.sleep(600);
			} catch (InterruptedException e) {

				e.printStackTrace();
			}
			System.out.println("Counter= " + i);
		}
	}
}

public class Test {

	public static void main(String[] args) {
		A a1 = new A();

		Thread t1 = new Thread(a1, "number1");

		t1.start();

	}

}
```

خروجی: بعد از اجرای برنامه متوجه خواهید شد با وقفه هر ۶۰۰ میلی ثانیه ای شمارنده ها در خروجی چاپ می شوند:

```
Counter= 1
Counter= 2
Counter= 3
Counter= 4
Counter= 5
```

- متد sleep را برای کنترل استثنای احتمالی باید میان بلوک try-catch گذاشت.

نکته مهم : برای استفاده از متد sleep برای ایجاد وقفه در برنامه ، نیازی نیست حتما کلاسمون ، کلاس Thread را به ارث ببرد! ما می توانیم از این متد در کلاس معمولی خود نیز استفاده کنیم:

مثال:

```
package multithreading_Javalike;

public class B {

	public static void main(String[] args) {
		for (int i = 0; i < 4; i++) {

			try {
				Thread.sleep(500);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
			System.out.println(i);
		}

	}

}
```

خروجی: با وقفه ۵۰۰ میلی ثانیه ای هر شماره در خروجی چاپ می شود:

```
0
1
2
3
```

بازی حدس عدد.....

برنامه زیر از کاربر یک عدد در خواست می کند، و بعد از وارد کردن عدد توسط کاربر، برنامه تلاش می کند عدد کاربر را حدس بزند! اگر برنامه موفق شد عدد کاربر را حدس بزند یک پیام مبنی بر پیدا کردن عدد و همچنین تعداد حدس هایی که برای پیدا کردن عدد زده است را در خروجی چاپ می کند. این برنامه برای حدس اعداد یک تا سه رقمی جواب می دهد.

```
package multithreading_Javalike;

import java.util.Random;
import java.util.Scanner;

class GuessANumber extends Thread {
	private int number;
	private int digit;

	public GuessANumber(int number) {
		this.number = number;

	}

	Random rand = new Random();

	public void run() {
		this.setName(" GuessANumberThread");
		int counter = 0;
		int guess = 0;
		do {
			guess = rand.nextInt(1000) + 1;
			System.out.println(this.getName() + " guesses " + guess);
			counter++;
		} while (guess != number);
		System.out.println("** Correct! " + this.getName() + " in " + counter
				+ " guesses **");
	}
}

public class MainTest {

	public static void main(String[] args) {
		Scanner input = new Scanner(System.in);
		System.out.println("Enter a Number...");
		int number = input.nextInt();
		GuessANumber gu = new GuessANumber(number);
```

```
            Thread tgu = new Thread(gu);

            tgu.start();

        }
}
```

خروجی: فرض کنیم کاربر عدد ۷۶۱ را وارد کرده باشد

```
Enter a Number...
761
```

خروجی برنامه بصورت زیر است:

```
GuessANumberThread guesses 372
 GuessANumberThread guesses 602
 GuessANumberThread guesses 66
 GuessANumberThread guesses 294
 GuessANumberThread guesses 228
 GuessANumberThread guesses 587
 GuessANumberThread guesses 969
 GuessANumberThread guesses 239
 GuessANumberThread guesses 939
 GuessANumberThread guesses 987
 GuessANumberThread guesses 619
 GuessANumberThread guesses 650
 GuessANumberThread guesses 874
 GuessANumberThread guesses 624
 GuessANumberThread guesses 348
 GuessANumberThread guesses 676
 GuessANumberThread guesses 263
 GuessANumberThread guesses 790
 GuessANumberThread guesses 568
 GuessANumberThread guesses 80
 GuessANumberThread guesses 885
 GuessANumberThread guesses 724
 GuessANumberThread guesses 271
 GuessANumberThread guesses 247
 GuessANumberThread guesses 605
 GuessANumberThread guesses 585
 GuessANumberThread guesses 68
 GuessANumberThread guesses 619
 GuessANumberThread guesses 339
 GuessANumberThread guesses 404
 GuessANumberThread guesses 369
 GuessANumberThread guesses 405
 GuessANumberThread guesses 567
 GuessANumberThread guesses 881
 GuessANumberThread guesses 155
 GuessANumberThread guesses 663
 GuessANumberThread guesses 150
 GuessANumberThread guesses 903
 GuessANumberThread guesses 54
 GuessANumberThread guesses 923
```

```
GuessANumberThread guesses 444
GuessANumberThread guesses 486
GuessANumberThread guesses 709
GuessANumberThread guesses 682
GuessANumberThread guesses 312
GuessANumberThread guesses 550
GuessANumberThread guesses 576
GuessANumberThread guesses 304
GuessANumberThread guesses 69
GuessANumberThread guesses 146
GuessANumberThread guesses 490
GuessANumberThread guesses 53
GuessANumberThread guesses 537
GuessANumberThread guesses 132
GuessANumberThread guesses 813
GuessANumberThread guesses 967
GuessANumberThread guesses 100
GuessANumberThread guesses 450
GuessANumberThread guesses 684
GuessANumberThread guesses 402
GuessANumberThread guesses 270
GuessANumberThread guesses 902
GuessANumberThread guesses 661
GuessANumberThread guesses 586
GuessANumberThread guesses 596
GuessANumberThread guesses 963
GuessANumberThread guesses 602
GuessANumberThread guesses 696
GuessANumberThread guesses 449
GuessANumberThread guesses 291
GuessANumberThread guesses 403
GuessANumberThread guesses 319
GuessANumberThread guesses 598
GuessANumberThread guesses 557
GuessANumberThread guesses 273
GuessANumberThread guesses 690
GuessANumberThread guesses 374
GuessANumberThread guesses 796
GuessANumberThread guesses 953
GuessANumberThread guesses 351
GuessANumberThread guesses 176
GuessANumberThread guesses 915
GuessANumberThread guesses 155
GuessANumberThread guesses 683
GuessANumberThread guesses 169
GuessANumberThread guesses 837
GuessANumberThread guesses 526
GuessANumberThread guesses 769
GuessANumberThread guesses 941
GuessANumberThread guesses 929
GuessANumberThread guesses 314
GuessANumberThread guesses 910
GuessANumberThread guesses 250
GuessANumberThread guesses 986
GuessANumberThread guesses 617
```

```
 GuessANumberThread guesses 667
 GuessANumberThread guesses 761
** Correct!  GuessANumberThread in 97 guesses **
```

- این برنامه عدد کاربر را بعد از ۹۷ بار حدس زدن پیدا می کند.
- در این برنامه از Thread استفاده شده است.

```
class GuessANumber extends Thread {
	private int number;


	public GuessANumber(int number) {
		this.number = number;

	}

	Random rand = new Random();
```

- کلاس GuessANumber برای ترد شدن ، کلاس Thread را به ارث برده است.
- متغیر number عددی که قراره برنامه آن را حدس بزند را در خود جا می دهد.
- public GuessANumber سازنده کلاس مون هستش که یک پارامتر از نوع عدد صحیح میگیرد و در متغیر number می ریزد.
- چون برنامه میخواد عدد را حدس بزند نیاز به تولید اعداد تصادفی داریم برای این کار از کلاس Random شی ساخته ایم.

```
public void run() {
		this.setName(" GuessANumberThread");
		int counter = 0;
		int guess = 0;
		do {
			guess = rand.nextInt(1000) + 1;
			System.out.println(this.getName() + " guesses " + guess);
			counter++;
		} while (guess != number);
		System.out.println("** Correct! " + this.getName() + " in " + counter
				+ " guesses **");
	}
```

- تمامی دستوراتی که قراره ترد ما اجرا کنه درون متد run قرار گرفته است.
- با استفاده از دستور setName نامی برای ترد خود انتخاب کرده ایم.کلمه this در اینجا کلاس Thread ما می باشد.

- counter برای شماردن تعداد حدس استفاده شده است.
- guess عددی را که برنامه حدس میزند داخلش ریخته می شود.
- از طریق دستور rand.nextInt(1000) + 1 برنامه به صورت تصادفی عدد مورد نظر را حدس می زند.
- هر بار که برنامه یک عدد را حدس می زند یکی به مقدار شمارنده counter اضافه می شود.
- حلقه do-while را داریم که کارش اینه عدد مورد نظر را حدس بزن تا هنگامی که عددی که برنامه حدس میزنه برابر با عدد مورد نظر کاربر نباشد. به گونه دیگه میگه تا هنگامی که عدد مورد نظر کاربر را پیدا نکردی به حدس زدن ادامه بده.

```
System.out.println("** Correct! " + this.getName() + " in " + counter
                        + " guesses **");
```

- اگر عدد مورد نظر توسط برنامه حدس زده شده دستور بالا اجرا می شود که میگه پیدا کردم عدد رو همچنین تعداد حدس هایی که زده رو نمایش می دهد.

```
public class MainTest {

      public static void main(String[] args) {
            Scanner input = new Scanner(System.in);
            System.out.println("Enter a Number...");
            int number = input.nextInt();
            GuessANumber gu = new GuessANumber(number);
            Thread tgu = new Thread(gu);

            tgu.start();

      }
}
```

- خب برای تست و اجرای برنامه از کلاس MainTest استفاده شده است.

```
                  Scanner input = new Scanner(System.in);
                  System.out.println("Enter a Number...");
                        int number = input.nextInt();
```

- برای ورودی گرفتن از کاربر از کلاس Scanner شی ایجاد کرده ایم.
- و دستور سوال از کاربر برای وارد کردن عدد و ریختن عدد در متغیر number

```
GuessANumber gu = new GuessANumber(number);
            Thread tgu = new Thread(gu);

            tgu.start();
```

- در پایان هم از کلاسمون شی ساخته و یک شی از کلاس `Thread` ایجاد کرده و درون سازنده اش شی کلاس مون را قرار داده ایم.
- و ترد `tgu` را با متد `start` شروع به اجرا می کنیم.

خب ما تا اینجا مبحث `Multithreading` در جاوا را بررسی کردیم.چطور بود؟ متوجه شدید؟! برای بهتر شدن آموزش پیشنهادات و انتقادات خودتون رو ایمیل کنید. البته مبحث `Multithreading` تنها به اینجا ختم نمیشه و خود چند شاخه مبحث داره که نمیشد یکجا بهشون در یک جلسه پرداخت!! همین الانشم ۳۲ صفحه شده ، شاید منابع دیگه خلاصه تر مبحث ترد رو آموزش دهند اما من دوست نداشتم توضیحاتم ناقص باشه. به امید خدا تمامی مباحث چندنخی در جاوا در یک بسته بهش خواهیم پرداخت اما فعلا برا مقدمه یادگیری ترد این جلسه خوبه .در آینده در آموزش ساخت بازی در قالب پروژه به مبحث ترد بر خواهیم خورد و آنجا هم بهش می پردازیم.

پیروز و موفق باشید

هر روز مفاهیم و مثال های جدید به سایت اضافه می شود برای اطلاع از مطالب جدید روی سایت عضو کانال شوید.