Core Java

آموزش ساده و آسان جاوا

به نام خدا

تقدیم به هموطنان عزیزم

جاوا را با لذت یاد بگیرید!

RAHMAN.ZARIE92@GMAIL.COM رحمان زارعی

Core Java

آموزش ساده و آسان جاوا

آموزش زبان برنامه نویسی جاوا

Non Access Modifier

جلسه سی و نهم

نویسنده : رحمان زارعی

جاوا را ساده،آسان و شیرین بنوشید!!!

Core Java

آموزش ساده و آسان جاوا

سلام. قصد داریم یکی از مباحث جاوا را رو به روش ساده برای شما آموزش دهیم.

Modifiers ها به دو نوع زیر تقسیم می شوند:

1. Access control modifier
2. Non Access Modifier

ما در **جلسه ۱۶ آموزش مفاهیم پایه جاوا** در مورد **Modifiers** ها و نوع **Access control modifier** صحبت کردیم، حالا در این جلسه قصد داریم به نوع دوم یعنی Non-access Modifier بپردازیم.

Non Access Modifier

Non Access Modifier ها برخلاف Access control modifier **دسرسی** به متدها و متغیرها را تغییر نمی دهند.اما ویژگیهای خاصی رو به آنها می دهند.ما پنج نوع Non Access Modifier در جاوا داریم:

1. Final
2. Static
3. Transient
4. Synchronized
5. Volatile

Final

اصلاح کننده (modifier) `Final` برای تعریف یک متغیر از نوع `final` استفاده می شود!!! اگر یک متغیر از نوع `final` تعریف شود دیگر نمی توانیم محتوای آن متغیر را تغییر دهیم به عبارتی اصلاح کننده (modifier) `final` مانع از تغییر مقدار **متغیر** می شود.

Core Java

آموزش ساده و آسان جاوا

الزاما هنگام تعریف یک متغیر از نوع final باید آن متغیر را مقداردهی اولیه کنیم.کلمه کلیدی final برای متغیرها،متدها و کلاس ها می توانیم استفاده کنیم.

۱. متغیر final (Final Variable)

هنگامی که یک متغیر از نوع final تعریف می شود ، مقدار آن را نمی توانیم تغییر دهیم و به عبارتی مقدار متغیر همیشه ثابت(constant) باقی می ماند.

نحوه نوشتن (Syntax):

```
final int a = 5;
```

Example:

```
package javalike;

public class Example_modifier {

	final int a = 5;

	public static void main(String[] args) {
		Example_modifier em = new Example_modifier();
		em.a = 10;

	}

}
```

خروجی(output):

```
خطای کامپایل (compiler time error)
```

Core Java

آموزش ساده و آسان جاوا

ما متغیر a را در بدنه کلاس از نوع final تعریف کرده ایم ، و متغیری که از نوع final تعریف شود مقدارش ثابت است و نمی توانیم آن را تغییر دهیم.

۲. متد final (Final Method):

وقتی یک متد از نوع final تعریف می شود،آن متد را نمی توانیم override کنیم. منظور از override کردن این است که یک زیرکلاس یا کلاس فرزند می تواند با توجه **به نیاز خود** متدهای کلاس پدر را در بدنه خود پیاده سازی کند.

Example:

```
package javalike;

class StudyTonight {
	final void learn() {
		System.out.println("learning something new");
	}

}

class Student extends StudyTonight {
	void learn() {
		System.out.println("learning something interesting");
	}

	public static void main(String args[]) {
		Student object = new Student();
		object.learn();
	}
}
```

خروجی(output):

خطای کامپایل (compiler time error)

Core Java

آموزش ساده و آسان جاوا

- متد learn در کلاس StudyTonight از نوع final تعریف شده، و متدی که از نوع final تعریف شود را نمی توان در کلاس فرزند override کنیم.

- یک متد final را می توان در کلاس فرزند به ارث برد، اما نمی توان آن را override کنیم.

۳. کلاس final :

یک کلاس را می توانیم از نوع final تعریف کنیم. اگر کلاسی را از نوع final تعریف کنیم ، آن را نمی توانیم به ارث ببریم.

Example:

```
package javalike;

public final class Animal {


      class Dog extends Animal{

      }


}
```

خروجی(output):

```
خطای کامپایل (compiler time error)
```

- کلاس Animal از نوع final است به همین دلیل کلاس Dog نمی تواند آن را به ارث ببرد.

Core Java

آموزش ساده و آسان جاوا

- کلاس **String** در پکیج **java.lang** نمونه ای از کلاس **final** می باشد.

در زیر مثالی را می بینید که متغیرها و متدهای آن از نوع **final** تعریف شده اند.

Example:

```
package javalike;

class Cloth {
	final int MAX_PRICE = 999; // final variable
	final int MIN_PRICE = 699;

	final void display() // final method
	{
		System.out.println("Maxprice is" + MAX_PRICE);
		System.out.println("Minprice is" + MIN_PRICE);
	}
}
```

Static Modifier

اصلاح کننده (**Modifier**) **static** برای ایجاد متغیرها و متدهای کلاس استفاده می شود به گونه که بدون نمونه سازی (شی سازی) از کلاس قابل دسرس هستند.

Core Java

آموزش ساده و آسان جاوا

۱. متغیرهای استاتیک(Static with Variables)

متغیرهای استاتیک به عنوان عضوی از کلاس تعریف می شوند که بدون شی سازی از کلاس قابل دسرس می باشند.متغیر استاتیک تنها یک مقدار واحد می تواند بگیرد به عبارت دیگر مقدار یک متغیر استاتیک برای تمامی اشیای کلاس، یک مقدار می باشد و تمام اشیای کلاسی که دارای متغیر استاتیک می باشد تنها به همان یک مقدار متغیر استاتیک دسرسی دارند.

متغیر استاتیک برای نشان دادن ویژگی های مشترک یک کلاس استفاده می شود.و این موجب صرفه جویی در حافظه می شود. فرض کنید ۱۰۰ کارمند در یک شرکت وجود دارد. همه کارمندان دارای نام و شناسه کارمندی مختص خود می باشند، اما نام شرکت برای همه کارمندان یکسان خواهد بود. در اینجا ویژگی مشترک ما نام شرکت می باشد. بنابراین اگر یک کلاس برای کارمندان با نام `Employee` ایجاد کنیم متغیر `name_company` که دربرگیرنده نام شرکت می باشد باید از نوع `static` تعریف شود.

Example:

```
package javalike;

class Employee {
	int e_id;
	String name;
	static String company_name = "StudyTonight";
}
```

مثال برای متغیر استاتیک

Example:

```
package javalike;

class ST_Employee {
	int eid;
```

Core Java

آموزش ساده و آسان جاوا

```
	String name;
	static String company_name = "StudyTonight";

	public void show() {
		System.out.println(eid + " " + name + " " + company_name);
	}

	public static void main(String[] args) {
		ST_Employee se1 = new ST_Employee();
		se1.eid = 104;
		se1.name = "Abhijit";
		se1.show();
		ST_Employee se2 = new ST_Employee();
		se2.eid = 108;
		se2.name = "ankit";
		se2.show();
	}
}
```

خروجی(output):

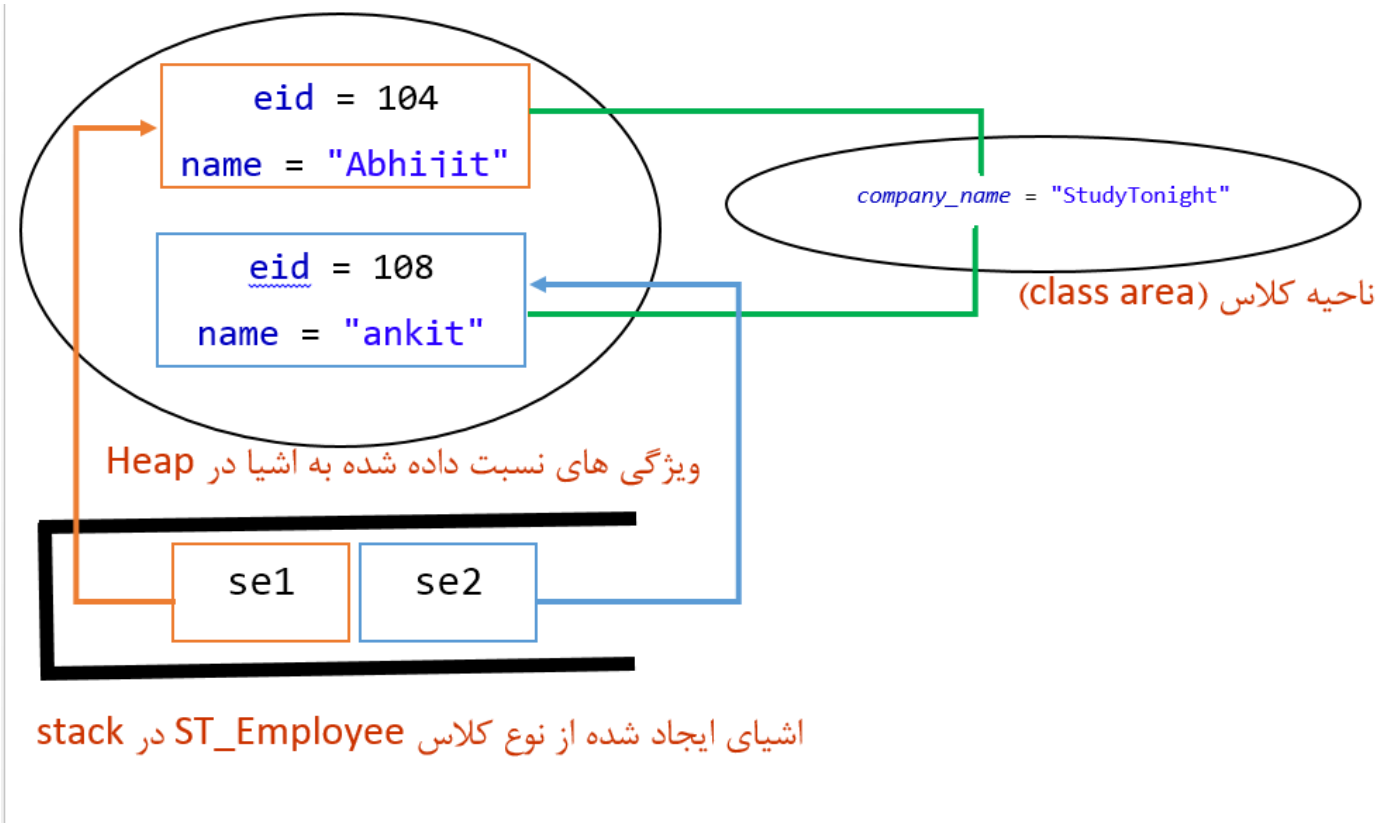
```
104 Abhijit StudyTonight

108 ankit StudyTonight
```

برای درک بهتر این مثال و مفهوم static به شکل(۱) توجه کنید:

Core Java

آموزش ساده و آسان جاوا



شکل(۱)

- همان طور که در شکل(۱) مشاهده می کنید،متغیر استاتیک `company_name` برای هر دو اشیای `se1` و `se2` که از نوع کلاس `ST_Employee` هستند یکسان می باشد.

Static variable vs Instance Variable

مقایسه متغیر `static` و متغیر نمونه:

Core Java

آموزش ساده و آسان جاوا

نکته

به متغیر غیر استاتیکی که در بدنه کلاس تعریف می شود متغیر نمونه می گوییم

متغیر نمونه	متغیر استاتیک
برای ویژگی های منحصربه فرد هر اشیا کاربرد دارد مثال:ویژگی شناسه کارمندی برای یک شرکت	برای ویژگی های مشترک میان اشیا کاربرد دارد مثال:ویژگی نام شرکت
تنها با شی از نوع کلاس قابل دسرس است	با نام کلاس قابل دسرس است
به ازای ایجاد هر شی جدید حافظه می گیرد	فقط یکبار حافظه رو دریافت می کند

Example:

```
package javalike;
public class Test {
	static int x = 100;
	int y = 100;

	public void increment() {
		x++;
		y++;
	}
	public static void main(String[] args) {
		Test t1 = new Test();
		Test t2 = new Test();
		t1.increment();
		t2.increment();
		System.out.println(t2.y);
		System.out.println(Test.x); // accessed without any instance of class.
	}
}
```

Core Java

آموزش ساده و آسان جاوا

خروجی(output):

```
101
102
```

- تفاوت ارزش دو متغیر را ببینید،متغیر x از نوع static است، و همان طور که می دانید یک متغیر static برای همه اشیای یک کلاس یکسان می باشد. به همین خاطر با صدا زدن متد increment توسط دو شی t1 و t2 ابتدا مقدار متغیر x از ۱۰۰ به ۱۰۱ و سپس به ۱۰۲ تغییر مقدار پیدا می کند.
- متغیر y چون از نوع متغیر نمونه هستش، دو متغیر y منحصربه فرد برای دو شی t1 و t2 ایجاد می شود و با صدا زدن متد increment تنها مقدار y از ۱۰۰ به ۱۰۱ تغییر مقدار پیدا می کند.

Static Method

یک متد را می توانیم static تعریف کنیم. برای صدا زدن و دسرسی به متدهای static یک کلاس نیازی به شی ساختن از کلاس نیست، همانند متغیرهای استاتیک تنها با نام کلاسی که دارای متدهای استاتیک می باشد می توانیم متدهای static را صدا بزنیم. معروف ترین متد استاتیکی که می توانیم برای شما مثال بزنیم متد استاتیک main() می باشد.

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

Example:

```
package javalike;

class Test {

     public static void square(int x) {
          System.out.println(x * x);
     }

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

```
            square(8); // static method square () is called without any instance
//of class.


      }
}
```

خروجی(output):

```
64
```

Static block

برای مقداردهی متغیرهای استاتیک یک کلاس می توانیم از بلوک استاتیک استفاده کنیم.بلوک استاتیک قبل از متد main() اجرا میشود.

Example:

```
package javalike;

class ST_Employee {
      int eid;
      String name;
      static String company_name;

      static {
            company_name = "StudyTonight"; // static block invoked before main()
                                                      // method
      }

      public void show() {
            System.out.println(eid + " " + name + " " + company_name);
      }
```

Core Java

آموزش ساده و آسان جاوا

```
	public static void main(String[] args) {
		ST_Employee se1 = new ST_Employee();
		se1.eid = 104;
		se1.name = "Abhijit";
		se1.show();

	}
}
```

خروجی(output):

```
104 Abhijit StudyTonight
```

Q چرا در یک **static context** ای مثل متد استاتیک نمی توانیم به یک متغیر غیر استاتیک اشاره کنیم؟

هنگامی که شما در یک **static context** ای نظیر متد **main()** سعی می کنید یک متغیر غیراستاتیک را صدا بزنید، خطای زمان کامپایل شبیه پیام زیر رخ می دهد:

"a non-static variable cannot be referenced from a static context"

دلیل رخ دادن این خطا این است که متغیرهای غیراستاتیک با نمونه ای از کلاس(شی) ارتباط دارد، و تنها زمانی که از کلاس خود نمونه ایجاد می کنیم ، متغیرهای غیراستاتیک ایجاد می شوند.بنابراین اگر بدون شی سازی از کلاس سعی کنید به متغیر غیراستاتیک کلاس دسرسی پیدا کنید کامپایلر به شما گیر خواهد داد! زیرا متغیر غیراستاتیک بدون شی ساختن از کلاس ایجاد نمی شود.

Core Java

آموزش ساده و آسان جاوا

مثالی از صدا زدن یک متغیر غیراستاتیک در یک static context ای نظیر متد main :

Example:

```
package javalike;

class Test {
	int x;

	public static void main(String[] args) {
		x = 10;
	}
}
```

خروجی(output):

به دلیل صدا زدن متغیر غیراستاتیک x در یک static context ای نظیر متد استاتیک main برنامه دچار خطای زمانی کامپایل می شود و اگر برنامه را run (اجرا) کنید با خروجی زیر برخورد می کنید:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
	Cannot make a static reference to the non-static field x

	at javalike.Test.main(Test.java:7)
```

مثال بالا را به صورت زیر تغییر می دهیم: صدا زدن متغیر غیراستاتیک x با نمونه ایجاد شده از کلاس (شی)

```
package javalike;

class Test {
	int x;

	public static void main(String[] args) {
		Test tt = new Test();
		tt.x = 10; // works fine with instance of class
	}
}
```

Core Java

آموزش ساده و آسان جاوا

خروجی(output):

برنامه کامل صحیح و بدون خطا می باشد، چرا که یک متغیر غیراستاتیک را در یک static context ای نظیر متد استاتیک main() با شی ایجاد شده از کلاس صدا زده ایم.

Q

چرا متد main() در جاوا static می باشد؟

زیرا متدهای استاتیک را می توان بدون نمونه سازی(شی ساختن) از یک کلاس صدا زد. و متد main() قبل از نمونه (شی) ایجاد شده از یک کلاس صدا زده می شود.

Transient modifier

اگر به مقدار یک متغیر هنگام ذخیره سازی شی از کلاس نیازی نداریم آن متغیر را از نوع transient تعریف می کنیم!

از اصلاح کننده(modifier) transient در serialization جاوا استفاده می کنیم. مثلا جایی که قصد داریم شی از نوع یک کلاس را در یک فایل ذخیره کنیم، و قصد داریم مثلا یک متغیر خاص از این کلاس در فایل ذخیره نشود ، برای این کار متغیر مورد نظر را از نوع transient تعریف می کنیم.

از توضیح اضافی صرف نظر می کنیم و با مثال مفهوم Transient modifier را بررسی می کنیم:

Core Java

آموزش ساده و آسان جاوا

Example: **TestClass.java**

```
package javapro.ir;

import java.io.Serializable;

public class TestClass implements Serializable {

	private static final long serialVersionUID = 8191670218412460916L;

	// will be serialized
	private String propertyOne;

	// will not be serialized
	private transient String propertyTwo;

	public TestClass(String propertyOne, String propertyTwo) {
		this.propertyOne = propertyOne;
		this.propertyTwo = propertyTwo;
	}

	public String getPropertyOne() {
		return propertyOne;
	}

	public String getPropertyTwo() {
		return propertyTwo;
	}

}
```

ما یک کلاس ساده که دارای یک ویژگی transient هست تعریف کرده ایم،حال در کلاس زیر قصد داریم برنامه را تست کنیم:

- برای درک این مثال باید مبحث serialization در جاوا رو مطالعه کنید.

Core Java

آموزش ساده و آسان جاوا

Main.java

```
package javapro.ir;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Main {

  public static void main(String[] args)
    throws Exception {

    TestClass testWrite = new TestClass("valueOne", "valueTwo");
    FileOutputStream fos = new FileOutputStream("testfile");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(testWrite);

    oos.close();

    TestClass testRead;
    FileInputStream fis = new FileInputStream("testfile");
    ObjectInputStream ois = new ObjectInputStream(fis);
    testRead = (TestClass)ois.readObject();
    ois.close();

    System.out.println("--Serialized object--");
    System.out.println("propertyOne: " + testWrite.getPropertyOne());
    System.out.println("propertyTwo: " + testWrite.getPropertyTwo());
    System.out.println("");
    System.out.println("--Read object--");
    System.out.println("propertyOne: " + testRead.getPropertyOne());
    System.out.println("propertyTwo: " + testRead.getPropertyTwo());

  }

}
```

Core Java

آموزش ساده و آسان جاوا

خروجی(output):

```
--Serialized object--
propertyOne: valueOne
propertyTwo: valueTwo

--Read object--
propertyOne: valueOne
propertyTwo: null
```

- برنامه بالا یک شی از نوع کلاس TestClass را در یک فایل ذخیره می کند و دوباره فایل را خوانده و شی ذخیره شده را می خواند.
- همان طور که می دانید در جاوا شی را از روش serialization در فایل ذخیره می کنیم. و شی که در یک فایل ذخیره شود به همراه آن ویژگی های آن نیز در فایل ذخیره می شود.
- مشاهده می کنید که هنگام خوانده شدن شی از فایل مقدار متغیر propertyTwo برابر null می باشد! چرا؟ چون متغیر propertyTwo را از نوع transient تعریف کرده ایم، و متغیری که در کلاس از نوع transient تعریف شود هنگام ذخیره سازی شی کلاس در فایل مقدار آن نادیده گرفته می شود.

```
import java.io.Serializable;
```

- چون قصد داریم کلاس خود را implements به اینترفیس Serializable کنیم باید پکیج بالا را در برنامه خود import کنید.

```
public class TestClass implements Serializable {
```

- برای ذخیره سازی شی از کلاس در یک فایل باید کلاس خود را implements به اینترفیس Serializable کنید.

```
private static final long serialVersionUID = 8191670218412460916L;

    // will be serialized
    private String propertyOne;

    // will not be serialized
    private transient String propertyTwo;
```

Core Java

آموزش ساده و آسان جاوا

- کلاس ما دارای سه ویژگی می باشد که ویژگی propertyTwo که از نوع کلاس String است، از نوع (modifier) transient تعریف شده است. همان طور که از قبل گفتیم متغیری که از نوع transient تعریف شود در هنگام ذخیره سازی شی از نوع کلاس مقدارش نادیده گرفته می شود.

```
public TestClass(String propertyOne, String propertyTwo) {
          this.propertyOne = propertyOne;
     this.propertyTwo = propertyTwo;
     }
```

- به عنوان پارامتر به سازنده کلاس دو ویژگی از نوع String داده ایم که مقدار آنها درون دو متغیر نمونه propertyOne و propertyTwo ریخته می شود. در کل از طریق سازنده کلاس دو متغیر نمونه مذکور را مقداردهی می کنیم.

```
public String getPropertyOne() {
     return propertyOne;
 }

 public String getPropertyTwo() {
     return propertyTwo;
 }
```

- متد getter دو متغیر propertyOne و propertyTwo هستند که مقدار آنها رو برای ما برمیگردانند.

بررسی کلاس Main.java

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
```

- پکیج های مربوط به کار با فایل در جاوا رو import می کنیم.

```
public class Main {
```

- کلاس Main برای ایجاد شی از کلاس TestClass و ذخیره سازی و خواندن شی ساخته شده از کلاس مذکور و اجرای برنامه را برعهده دارد.

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

Core Java

آموزش ساده و آسان جاوا

- متد main برای اجرای برنامه کاربرد دارد و برای کنترل استثناهای احتمالی ، این متد را به استثنای Exception ، throws کرده ایم. راه حل دیگر کنترل استثناهای احتمالی استفاده از بلوک try-catch می باشد.

```
TestClass testWrite = new TestClass("valueOne", "valueTwo");
```

- از کلاس TestClass شی به نام testWrite ایجاد کرده ایم.

```
FileOutputStream fos = new FileOutputStream("testfile");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(testWrite);
    oos.close();
```

- دستورات بالا جهت ذخیره ساختن شی testWrite در یک فایل با نام testfile می باشد.

```
TestClass testRead;
```

- قصد داریم شی ذخیره شده در فایل را بخوانیم، چون شی ذخیره شده از نوع کلاس TestClass است به همین خاطر یک شی از نوع این کلاس تعریف می کنیم.

```
FileInputStream fis = new FileInputStream("testfile");
    ObjectInputStream ois = new ObjectInputStream(fis);
    testRead = (TestClass)ois.readObject();
    ois.close();
```

- دستورات بالا جهت خواندن شی ذخیره شده از فایل و ریختن شی درون شی testRead است.

```
testRead = (TestClass)ois.readObject();
```

- با عمل casting شی که از فایل خوانده می شود را به شی ای از نوع کلاس TestClass تبدیل می کنیم و درون شی testRead می ریزیم.

```
1. System.out.println("--Serialized object--");
2. System.out.println("propertyOne: " + testWrite.getPropertyOne());
3. System.out.println("propertyTwo: " + testWrite.getPropertyTwo());
4. System.out.println("");
5. System.out.println("--Read object--");
6. System.out.println("propertyOne: " + testRead.getPropertyOne());
7. System.out.println("propertyTwo: " + testRead.getPropertyTwo());
```

۱. خط اول یک پیام برای کاربر در محیط کنسول چاپ می کند.

Core Java

آموزش ساده و آسان جاوا

۲. خط دوم مقدار متغیر `propertyOne` مربوط به شی `testWrite` که از نوع کلاس `TestClass` است را چاپ می کند.
۳. خط سوم مقدار متغیر `propertyTwo` مربوط به شی `testWrite` که از نوع کلاس `TestClass` است را چاپ می کند. در اینجا مقدار متغیر `propertyTwo` با وجودی که از نوع `transient` بود نمایش داده شد! چرا؟ چون همان طور که گفتیم تنها در سریال سازی و ذخیره شی در فایل ، مقدار متغیر از نوع `transient` نادیده گرفته می شود. و در اینجا `testWrite` هنوز در فایل ذخیره و خوانده نشده است.
۴. خط چهارم باعث می شود به سطر جدید بریم.
۵. خط پنجم یک پیام مبنی بر خوانده شدن شی از فایل نمایش داده می شود.
۶. خط ششم مقدار متغیر `propertyOne` شی که از فایل خواندیم را نمایش می دهد.
۷. خط هفتم مقدار متغیر `propertyTwo` شی که از فایل خواندیم را نمایش می دهد. و همان طور که در خروجی کنسول مشاهده می کنید این مقدار متغیر برابر `null` می باشد.چرا ؟ چون متغیر `propertyTwo` از نوع `transient` است و متغیری که از نوع `transient` باشد هنگام ذخیره سازی شی در فایل مقدارش نادیده گرفته می شود.

Synchronized modifier

وقتی که یک متد از نوع `synchronized` تعریف می شود ، تنها با یک `Thread` و در یک زمان قابل دسرسی می باشد. این خود یک مبحث مفصل به نام `Synchronization in java` است که در جلسه آموزشی مجزا به آن می پردازیم.

Volatile modifier

متغیرهایی که از نوع `volatile` تعریف می شوند ، در برنامه های چندنخی (multithreading program) در جاوا استفاده می شوند. کلمه کلیدی `volatile` تنها مختص متغیرها می باشد و برای متدها یا کلاس ها استفاده نمی شود.

کاربرد `Volatile modifier` در `multithreading program`:

Core Java

آموزش ساده و آسان جاوا

همان طور که از اسم multithreading (چندنخی) پیداست ، volatile modifier تنها برای برنامه هایی که بیش از یک thread دارند کاربر دارد.

اگر یک متغیر از نوع volatile تعریف شود ، تغییر اعمال شده روی متغیر توسط یک thread برای سایر thread ها نیز لحاظ می شود پس می توان گفت یک متغیر از نوع volatile برای همه thread های موجود در یک برنامه یکسان می باشد و اگر روی متغیر تغییراتی توسط یک thread اعمال شود این تغییرات متغیر برای همه thread ها لحاظ می شود. ما یک حافظه اصلی(Main Memory) و یک حافظه cathe داریم. زمانی که متغیر از نوع volatile تعریف شود ، این متغیر بجای حافظه cathe در حافظه اصلی قرار می گیرد و همه thread ها مستقیم به این متغیر در حافظه اصلی دسرسی دارند به همین علت اگر یک thread مقدار این متغیر را در حافظه اصلی تغییر داد ، برای همه thread ها تغییر مقدار متغیر اعمال می شود.

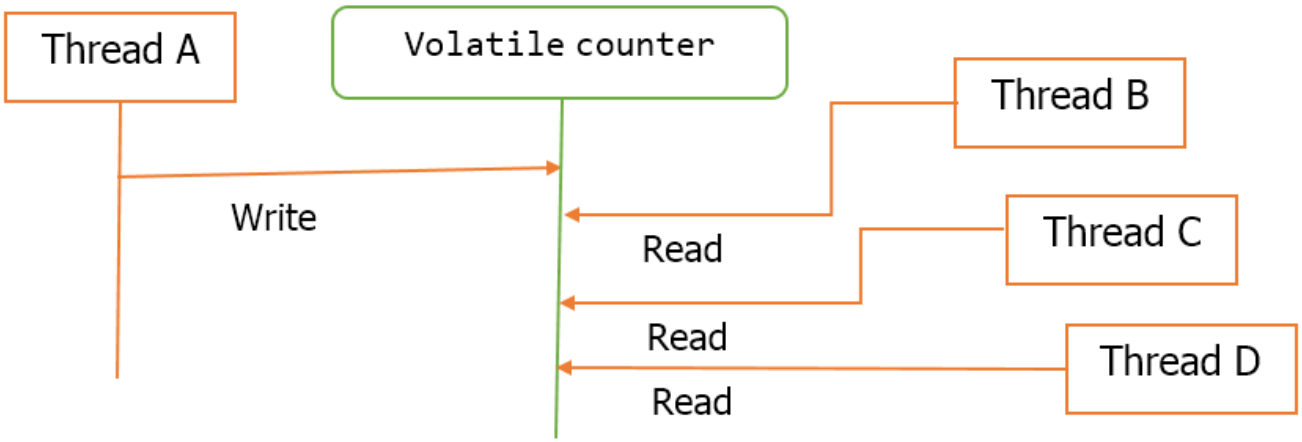
شاید براتون سوال پیش بیاد که یک متغیر بدونی که از نوع volatile تعریف شود ، احتمال دارد با تغییر توسط یک thread ، برای سایر thread ها این تغییر لحاظ شود! تا حدودی حق با شماست! اما در اینجا از کلمه احتمال استفاده کردیم! یعنی امکان داره این تغییر برای همه thread ها اعمال شود ممکن است اعمال نشود!! اینجاست که جناب volatile میگه من به شما تضمین می کنم و اطمینان می دهم اگر متغیر را از نوع من استفاده کردید به صورت تضمینی تغییر یک متغیر توسط یک thread برای سایر thread ها نیز لحاظ می شود.

برای درک بهتر دسرسی یکسان thread ها به متغیر از نوع volatile به شکل (۲) توجه کنید:

Core Java

آموزش ساده و آسان جاوا

فرض کنید چهار Thread با نام های A و B و C و D داشته باشیم ، و یک متغیر از نوع int با نام counter که از نوع volatile می باشد. دسترسی یکسان به متغیر counter برای این thread ها به صورت زیر است:



شکل(۲)

همان طور که در شکل(۲) مشاهده می کنید اگر توسط thread A روی متغیر counter که از نوع volatile تغییراتی اعمال شود، این تغییر مقدار متغیر counter برای سایر thread ها نیز این اعمال می شود، چون همه thread ها با متغیر counter که از نوع volatile است مستقیم در ارتباط هستند.

برای درک بهتر این مفاهیم به مثال زیر توجه کنید:

Example:

فرض کنید یک کلاس با نام Clock به صورت زیر داریم:

```
package javalike;

public class Clock {

	volatile int counter = 0;
	public void increaseCounter() {
		++counter;
	}
}
```

Core Java

آموزش ساده و آسان جاوا

- متغیر counter از نوع volatile تعریف شده است.
- متد increaseCounter کارش اضاف کردن مقدار عدد یک به مقدار متغیر counter است.

حال یک کلاس با نام A ایجاد می کنیم:

```
package javalike;

public class A  {

	public A() {

	}

}
```

قصد داریم کلاس A را تبدیل به یک Thread کنیم، به همین خاطر کلاس A را extends به کلاس Thread می کنیم و متد run کلاس Thread را در بدنه کلاس A پیاده سازی و Override می کنیم:

```
package javalike;

public class A extends Thread  {

	public A() {

	}

	public void run(){

	}
}
```

حال کلاس A را به صورت زیر تغییر می دهیم:

```
package javalike;

public class A extends Thread {

	Clock clock;
```

Core Java

آموزش ساده و آسان جاوا

```
	public A(Clock clock) {
		this.clock = clock;
	}

	public void run() {
		int oldValue = clock.counter;
		System.out.println("Thread [" + Thread.currentThread().getId()
				+ "] oldValue: " + oldValue);
		clock.increaseCounter();

		int newValue = clock.counter;
		System.out.println("Thread [" + Thread.currentThread().getId()
				+ "] NewValue: " + newValue);
	}
}
```

```
Clock clock;
```

- یک شی از نوع کلاس Clock در بدنه کلاس A تعریف کرده ایم.

```
public A(Clock clock) {
		this.clock = clock;
	}
```

- از طریق سازنده کلاس A یک پارامتر از نوع کلاس Clock دریافت و درون شی از نوع کلاس Clock که در بدنه کلاس تعریف کرده ایم می ریزیم.کلا میشه ست کردن و مقداردهی کردن شی clock که در بدنه کلاس A تعریف کردیم.

```
int oldValue = clock.counter;
```

- در متد run یک متغیر از نوع int تعریف کرده و از طریق شی clock که از نوع کلاس Clock است متغیر counter را صدا زده و مقدارش را درون متغیر oldValue ریخته ایم.

```
System.out.println("Thread [" + Thread.currentThread().getId()
				+ "] oldValue: " + oldValue);
```

Core Java

آموزش ساده و آسان جاوا

در دستور بالا ID تردی که start خورده و در جریان است و همچنین مقدار متغیر oldValue مربوط به آن ترد نمایش داده می شود.

```
clock.increaseCounter();
```

متد increaseCounter از طریق شی clock که از نوع کلاس Clock است صدا زده شده است.و باعث میشه یکی به مقدار متغیر counter اضاف شود.

```
int newValue = clock.counter;
          System.out.println("Thread [" + Thread.currentThread().getId()
                    + "] NewValue: " + newValue);
```

- دوباره مقدار متغیر counter مثل قبل را صدا زده و درون متغیر newValue می ریزیم.و در خط بعد id تردی که در جریان است به همراه مقدار newValue چاپ می شود.

حال یک کلاس دیگر با نام Main به صورت زیر پیاده سازی می کنیم:

```
package javalike;

public class Main {

     public static void main(String[] args) throws InterruptedException {

          Clock clock = new Clock();

          Thread thread_B = new A(clock);
          Thread thread_C = new A(clock);
          Thread thread_D = new A(clock);

          thread_B.start();
          thread_B.join();

          thread_C.start();
          thread_C.join();

          thread_D.start();
          thread_D.join();
```

```
    }

}
```

- پس برنامه ما از سه کلاس Clock ، A و Main تشکیل شده است و به این نکته توجه کنید که برای اجرای برنامه این سه کلاس باید در یک پکیج قرار داشته باشند.

ابتدا برنامه را اجرا می کنیم بعد دستورات درون کلاس Main را توضیح می دهیم.

خروجی(output): با اجرا(run) کردن برنامه خروجی به صورت زیر خواهد بود:

```
Thread [11] oldValue: 0
Thread [11] NewValue: 1
Thread [12] oldValue: 1
Thread [12] NewValue: 2
Thread [13] oldValue: 2
Thread [13] NewValue: 3
```

توضیحات کلاس Main :

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

چون از متد join در بدنه متد main استفاده کردیم ، برای کنترل استثنای مربوط به متد join ، متد main را throws به استثنای InterruptedException کرده ایم.

```
Clock clock = new Clock();
```

- در بدنه متد main از کلاس Clock شی ایجاد کرده ایم.

```
Thread thread_B = new A(clock);
        Thread thread_C = new A(clock);
        Thread thread_D = new A(clock);
```

- سه شی از نوع کلاس Thread ایجاد کرده ایم.
- شی از کلاس ترد A را درون شی های کلاس Thread های thread_B ، thread_C و thread_D ریخته ایم.

Core Java

آموزش ساده و آسان جاوا

- به عنوان پارامتر شی از نوع کلاس Clock را به سازنده کلاس ترد A داده ایم.

```
thread_B.start();
thread_B.join();
```

- ترد thread_B را start زده ایم و با صدا زدن متد join گفتیم تا اتمام تمامی دستورات مربوط به ترد thread_B هیچ thread دیگری نباید start بخورد.

```
thread_C.start();
thread_C.join();

thread_D.start();
thread_D.join();
```

- بعد از اتمام دستورات ترد thread_B به ترتیب دستورات ترد thread_C و سپس دستورات ترد thread_D اجرا می شود.

پس کلا ما دارای سه کلاس به صورت زیر هستیم:

Clock.java

```
package javalike;

public class Clock {

	volatile int counter = 0;

	public void increaseCounter() {
		++counter;
	}

}
```

A.java

```
package javalike;

public class A extends Thread {
```

```
	Clock clock;

	public A(Clock clock) {
		this.clock = clock;
	}



	  public void run() {
		int oldValue = clock.counter;
		System.out.println("Thread [" + Thread.currentThread().getId()
				+ "] oldValue: " + oldValue);
		clock.increaseCounter();

		int newValue = clock.counter;
		System.out.println("Thread [" + Thread.currentThread().getId()
				+ "] NewValue: " + newValue);
	}
}
```

Main.java

```
package javalike;

public class Main {

	public static void main(String[] args) throws InterruptedException {
		Clock clock = new Clock();
		Thread thread_B = new A(clock);
		Thread thread_C = new A(clock);
		Thread thread_D = new A(clock);

		thread_B.start();
		thread_B.join();

		thread_C.start();
		thread_C.join();
```

```
            thread_D.start();
            thread_D.join();

        }
}
```

خروجی(output): حتما سه کلاس بالا را برای اجرا درون یک پکیج قرار دهید، خب دوباره برنامه بالا را اجرا می کنیم:

```
Thread [11] oldValue: 0
Thread [11] NewValue: 1
Thread [12] oldValue: 1
Thread [12] NewValue: 2
Thread [13] oldValue: 2
Thread [13] NewValue: 3
```

- خب توضیحات خط به خط کدهای برنامه بالا رو دادیم ، حالا میریم سراغ اصل ماجرا!!!

منطق برنامه بالا:

در متد main کلاس Main یک Thread با نام thread_B ایجاد کرده و start زدیم، با استارت خوردن ترد thread_B دستورات درون متد run کلاس A اجرا می شود! چرا دستورات متد run کلاس A اجرا می شود؟! زیر ما شی از کلاس A را به شی thread_B داده ایم.

```
Thread thread_B = new A(clock);
```

خب حالا با اجرای دستورات درون بدنه متد run چه اتفاقی می افتد؟

```
public void run() {
int oldValue = clock.counter;
System.out.println("Thread [" + Thread.currentThread().getId()
+ "] oldValue: " + oldValue);
clock.increaseCounter();
```

```
        int newValue = clock.counter;
        System.out.println("Thread [" + Thread.currentThread().getId()
                + "] NewValue: " + newValue);
    }
```

- متغیر counter کلاس Clock صدا زده می شود و مقدارش درون متغیر oldValue ریخته می شود. و مقدار متغیر oldValue همراه با id ترد thread_B که در جریان است در خروجی چاپ می شود، که در اینجا چون ابتدا مقدار متغیر counter برابر صفر است، مقدار 0 چاپ می شود.
- حال با صدا زده شدن متد increaseCounter یکی به مقدار متغیر counter اضاف می شود،پس متغیر counter مقدارش از ۰ به ۱ تغییر مقدار پیدا می کند.
- حالا مقدار متغیر counter درون متغیر newValue ریخته شده و id ترد thread_B که در جریان است به همراه مقدار متغیر newValue چاپ می شود که اینبار مقدار ۱ برای ترد thread_B که در جریان است چاپ می شود.

پس تا اینجا برای ترد thread_B خروجی به صورت زیر است:

```
Thread [11] oldValue: 0
Thread [11] NewValue: 1
```

حال دستورات ترد thread_B تمام شده و ترد thread_C استارت میخورد :

```
thread_C.start();
thread_C.join();
```

و همان اتفاق هایی که برای ترد thread_B رخ دادن برای thread_C نیز روی می دهد.

نکته مهمی که اینجا وجود دارد این است که متغیر counter از نوع volatile است ، و همان طور که می دانید یک متغیر از نوع volatile برای همه تردهای یک برنامه یکسان می باشد و اگر تغییراتی روی این متغیر رخ داد برای همه تردها اعمال می شود چرا که همه تردها مستقیما با متغیر volatile در ارتباط هستند.

- خب ما با ترد thread_B مقدار متغیر counter را دستکاری و از مقدار ۰ به ۱ تغییرش دادیم، حال که ترد thread_C در متد run خود متغیر counter را صدا می زند ، مقدار ۱ را بهش میده و با صدا زدن متد increaseCounter یکی به مقدار متغیر counter اضاف می شود،پس متغیر counter مقدارش از ۱ به ۲

Core Java

آموزش ساده و آسان جاوا

- تغییر مقدار پیدا می کند. این خاصیت متغیر volatile است که اگر یک ترد مقدارش را تغییر داد ، این مقدار برای سایر تردها هم تغییر می کند.
- پس تا اینجا برای ترد thread_C خروجی به صورت زیر است:

```
Thread [12] oldValue: 1
Thread [12] NewValue: 2
```

حال دستورات ترد thread_C تمام شده و ترد thread_D استارت میخورد .

```
thread_D.start();
thread_D.join();
```

همان اتفاق هایی که برای تردهای قبل رخ داد برای ترد thread_D هم رخ می دهد.

از آنجایی که ترد thread_C مقدار متغیر counter که از نوع volatile را دستکاری کرده و از ۱ به ۲ تغییر داده ، بنابر خاصیت یک متغیر volatile این تغییر برای ترد thread_D نیز حساب می شود.

حال همان دستوراتی که در متد run برای تردهای قبلی رخ داد برای ترد thread_D نیز رخ می دهد.و باعث میشه ابتدا متغیر counter صدا زده شود و مقدار ۲ چاپ شود ، سپس متد increaseCounter صدا زده شود و مقدار متغیر counter از ۲ به ۳ تغییر کند.

- پس تا اینجا برای ترد thread_D خروجی به صورت زیر است:

```
Thread [13] oldValue: 2
Thread [13] NewValue: 3
```

و کل خروجی برای سه ترد thread_B ، thread_C و thread_D به صورت زیر خواهد بود:

```
Thread [11] oldValue: 0
Thread [11] NewValue: 1
Thread [12] oldValue: 1
Thread [12] NewValue: 2
```

```
Thread [13] oldValue: 2
Thread [13] NewValue: 3
```

در کل متغیری که از نوع volatile تعریف می شود ، اگر توسط یک ترد دسکاری شود ، مقدار دستکاری شده برای سایر تردها نیز اعمال می شود چرا که همه تردها مستقیم با متغیر volatile در ارتباط هستند یا به عبارتی یک متغیر volatile برای همه تردهای یک برنامه یکسان می باشد.

امیدوارم واضع این مفهوم چغر!!! و بد ترکیب!!! رو بیان کرده باشم ☺

پیروز و موفق باشید

Core Java

آموزش ساده و آسان جاوا

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

دخل و تصرف ، ویرایش و کپی زدن تمامی آموزش های جاوالایک به دور از اخلاق حرفه ای ست و حرام می باشد.