آموزش زبان برنامه نویسی جاوا

Polymorphism (چند ریختی)

جلسه بیست و سوم

نویسنده : رحمان زارعی

جاوا را ساده،آسان و شیرین بنوشید!!!!

در این جلسه یکی دیگر از مفاهیم شی گرایی جاوا به نام چندریختی (Polymorphism)را قصد داریم بررسی کنیم.گاهی برای تعریف یک مفهوم ،تعریفی پیدا نمی شود!!! یعنی نمی توانیم تعریف مناسبی را ارائه کنیم!!! به همین خاطر با زدن مثال سعی میکنیم که مفهوم را جا بیندازیم! مثلا به نظر شما کسی هست تعریفی از آب داشته باشه؟!!! شاید یکی بگه اب از H2O تشکیل شده یا مایعی ای که شفافه و... اما همه اینها مثال هست و تعریفی از خود آب نیست!!!! منم در این جلسه بیشتر قصدم اینه که با مثال مفهوم چندریختی را بررسی کنیم چون اگه بخوایم فقط به تعریف و تمجید ☺ بسنده کنیم چیزی یاد نمیگیریم! بریم سراغ اصل مطلب:

نکته مهم در این آموزش:در اینجا برای یادگیری و ساده کردن مفهوم چندریختی تمام تلاشم را کردم و گام به گام که جلو میرید اگر مطلب غیر قابل فهم بود نگران نباشید چون همین جوری که جلوتر میرید مفهوم براتون روشن تر و قابل درک تر میشود.

فرآیند نمایش یک **شکل** در **شکل های متعدد** را چندریختی می گوییم.

چندریختی یا (Polymorphism) پلی مورفیسم از دو کلمه یونانی مشتق شده است،کلمه "Poly" به معنای بسیار و کلمه "morphs" به معنای اشکال هستش. و در کل پلی مورفیسم یا چندریختی به معنی اشکال مختلف می باشد.

از خود کلمه چندریختی هم میشه دریافت که چند ریخت!!!! مثل یک مرد هزار چهره که در هر مکان یک ریختی از چهره های خودش رو نشان میده ☺ صرفا جهت درک بهتر بود ☺

دلیل توضیح تئوری زیاد من این هست که قبل از رفتن به سراغ مثال یک امادگی ذهنی داشته باشیم.

چندریختی یعنی یک شی می تواند خود را به شکل های گوناگون در اورد.وقتی یک شی علاوه بر نقش خودش ،نقش های گوناگون را می تواند بازی کند!

مثال از دنیای واقعی.....

وقتی شما سر کلاس درس برنامه نویسی پیشرفته هستید شما در شکل **دانشجو** ظاهر شده اید.

وقتی به سوپرمارکت برای خرید شکلات تلخ مراجعه میکنید شما در شکل یک **مشتری** ظاهر شده اید.

وقتی در خانه در کنار پدر و مادر خود هستید شما به شکل یک **فرزند** ظاهر شده اید.

- در فروشگاه شکل و رفتار یک **مشتری** را دارد.
- در دانشگاه شکل و رفتار یک **دانشجو** را دارد.
- در اتوبوس شکل و رفتار یک **مسافر** را دارد.
- در خانه رفتار یک **فرزند** را دارد.

در اینجا یک **شخص** در **اشکال مختلف** ظاهر شده است.چندریختی هم این مفهوم را دنبال میکند وقتی یک **شی** در **اشکال مختلف** ظاهر میشود به آن چندریختی یا پلی مورفیسم میگوییم.

پس در چندریختی یک شی می تواند نقش و رفتار چندین شی دیگری را بازی کند.

مفاهیم جلسه قبل Overriding در چندریختی کاربرد فراوان دارد.در ادامه متوجه خواهیم شد.

متداول ترین استفاده از چندریختی در برنامه نویسی شی گرایی جاوا هنگامی است که شی کلاس پدر به شی کلاس فرزند اشاره دارد!!!! یعنی ما هنگام ساختن شی از کلاس پدر کاری میکنیم که هنگامی که شی پدر ایجاد می شود ، شی پدر به فرزند خود اشاره داشته باشد به معنی ساده تر شی پدر همان شی فرزند است یعنی شی پدر در شکل فرزند ظاهر شده است.میدونم مفهوم هنوز واضع نیست نگران نباشید حلش میکنیم ☺

در چندریختی شی کلاس پدر در نقش و شکل فرزندان خود ظاهر می شود.یعنی شی کلاس پدر می تواند به شکل های گوناگون فرزندان خود ظاهر شود.مثال زیر را ببینید:

فرض کنید یک کلاس به نام شخص (Person) داریم و کلاس هایی به نام کارمند (Employee)،مشتری (Consumer)،زن (woman) کلاس شخص (Person) را به ارث برده اند.یعنی کلاس شخص (person) پدر کلاس های کارمند (Employee)،مشتری (Consumer)،زن (woman) و کلاس های کارمند (Employee)،مشتری (Consumer)،زن (woman) فرزند کلاس شخص (Person) می باشند.

خب در چندریختی باید به گونه ای از کلاس پدر یعنی شخص (Person) شی ایجاد کنیم که مثلا یک شی بتواند به شکل کارمند (Employee)، شی دیگر به شکل مشتری (Consumer) و شی دیگری به شکل زن (woman) ظاهر شود.یعنی شی از نوع پدر باشد اما نقش و شکل فرزند خود را داشته و بازی کند.خب چطور چنین شی ایجاد کنیم که نوعش از نوع پدر باشد اما نقش یکی از فرزندان را بازی کند؟!

نحوه ایجاد یک شی از کلاس پدر که در شکل فرزند خود ظاهر می شود

در جلسات گذشته یادگرفتیم در حالت عادی برای شی ساختن از هر کلاسی بصورت زیر عمل میکنیم:

```
نام کلاس + نام شی + = + new + سازنده کلاس
```

مثلا برای ایجاد یک شی در حالت عادی از کلاس Person :

```
Person p=new Person();
```

خب برای این که یک شی از کلاس پدر ایجاد کنیم که در نقش و شکل فرزندان خود ظاهر شود بصورت زیر عمل میکنیم:

```
نام کلاس + نام شی + = + new + سازنده کلاس فرزند
```

- تنها تفاوت ساختن شی به روش چندریختی و این که شی کلاس پدر به شکل و نقش شی کلاس فرزند ظاهر شود با شی ساختن حالت عادی این است که **نوع شی از نوع کلاس پدر** و **سازنده** آن را از **سازنده کلاس فرزند** انتخاب میکنیم.

خب قصد داریم یک شی از کلاس Person (کلاس پدر) بسازیم که به شکل و نقش کلاس فرزند (زیر کلاس)خود Employee ظاهر شود:

```
Person p=new Employee();
```

- در اینجا نوع شی از نوع کلاس پدر یعنی person هستش و سازنده آن را سازنده کلاس فرزند قرار داده ایم.

تا اینجا با مفهوم چندریختی و روش ایجاد آن تا حدودی آشنا شدیم پس :

۱. چندریختی یعنی یک شی بتواند در اشکال گوناگون شی های دیگر ظاهر شود.
۲. در چندریختی درصورتی یک شی می تواند به شکل و نقش شی دیگر ظاهر شود که کلاس شی اول پدر کلاس شی دوم باشد.فرض کنید کلاس A کلاس B را به ارث برده است.یعنی کلاس B پدر کلاس A و کلاس A فرزند کلاس B می باشد.در این صورت می توانیم شی ای از نوع کلاس B (پدر) ایجاد کنیم که سازنده آن سازنده کلاس A (فرزند) هستش و در ان صورت آن شی که از نوع کلاس B (پدر) بود می تواند در شکل شی کلاس A (کلاس فرزند) ظاهر شود.

```
B b=new A();
```

شرط چندریختی این هست که ابتدا رابطه IS-A بین کلاس ها وجود داشته باشد یعنی رابطه پدر و فرزندی بین کلاس ها وجود داشته باشد:

```
package javalike;

public class Deer  extends Animal {

}

class Animal {

}
```

در اینجا کلاس Deer کلاس Animal را به ارث برده است.یعنی Deer فرزند Animal و Animal پدر Deer می باشد.

رابطه IS-A را در مثال بالا بصورت زیر بررسی میکنیم:

A Deer IS-A Animal

یک آهو یک حیوان است

مثلا در اینجا گفتیم آهو یک حیوان است و این یک رابطه درستی می باشد چون آهو جز خانواده و زیر مجموعه حیوانات می باشد.

اما اگر بگیم هواپیما یک حیوان است ،این یک رابطه IS-A اشتباه می باشد زیرا هواپیما زیر مجموعه حیوانات نمی باشد.

در کل نکته ۲ تنها میخواستیم این مفهوم را بگیم که درصورتی یک شی از کلاس می تواند در نقش و شکل شی کلاس دیگر ظاهر شود که رابطه IS-A رابطه پدر و فرزندی بین انها برقرار باشد.

همه این ها در اینجا مقدمه چینی بود برای ورود به اصل ماجرای چندریختی:

در این بخش هیجان انگیز شعبده بازی چندریختی را با هم تماشا خواهیم کرد!!!!!

در این بخش به شما نشان خواهم داد که در هنگام طراحی کلاس خود چگونه از رفتار یک متد بازنویسی شده (Overridden) در چندریختی بهره ببریم.

ما در جلسات قبل در مورد متدهای Overriding صحبت کردیم. مفهوم Overriding کردن متد این بود که کلاس فرزند می توانست در بدنه خود متد کلاس پدر را بازنویسی و تغییر دهد.

متد override شده (بازنویسی شده) اساسا در کلاس پدر پنهان شده است.یعنی دیگر به اصل متد در کلاس پدر دسرسی نداریم مگر این که با کلمه کلیدی super در متد Override شده در کلاس فرزند آن را صدا بزنیم:

```
package javalike;

public class Deer extends Animals {

	void sleep() {

		System.out.println("Deer sleeping....oOoOooOoo");
	}

	public static void main(String[] args) {
		Deer d = new Deer();
		d.sleep();
	}
}

class Animals {
	void sleep() {
		System.out.println("animal sleeping....oOoOooOoo");

	}
}
```

خروجی:

```
Deer sleeping....oOoOooOoo
```

در این مثال متد sleep را در کلاس فرزند override (بازنویسی)کردیم.به همین خاطر شی ساخته از کلاس فرزند تنها می تواند به متد بازنویسی شده (override شده) در کلاس فرزند اشاره کند و متد sleep درون کلاس فرزند پنهان شده است و قابل دسرس نیست مگر این که از کلمه کلیدی super برای صدا زدن متد sleep کلاس پدر در متد بازنویسی شده (override شده) درون کلاس فرزند استفاده کنیم:

```
package javalike;

public class Deer extends Animals {

	void sleep() {
		super.sleep();

		System.out.println("Deer sleeping....oOoOooOoo");
	}

	public static void main(String[] args) {
		Deer d = new Deer();
		d.sleep();
	}
}

class Animals {
	void sleep() {
		System.out.println("animal sleeping....oOoOooOoo");

	}
}
```

خروجی:

```
animal sleeping....oOoOooOoo
Deer sleeping....oOoOooOoo
```

پس تنها راه صدا زدن متد بازنویسی شده(override شده) کلاس پدر در کلاس فرزند استفاده از کلمه کلیدی super درون متد بازنویسی شده (override شده) است.خط نارنجی رنگ

تا اینجا با مفهوم ابتدایی چند ریختی و overriding (آموزش کامل در جلسات قبل) آشنا شده ایم حالا میخوایم بدونیم اصلا این چندریختی در عمل چه میکنه؟

ماجرای هیجان انگیز چندریختی:

این بار بجای شروع کردن با تعریف و توضیح و بعد مثال ابتدا با مثال شروع کنم و بعدش تعریف و توضیح!!!!! ☺

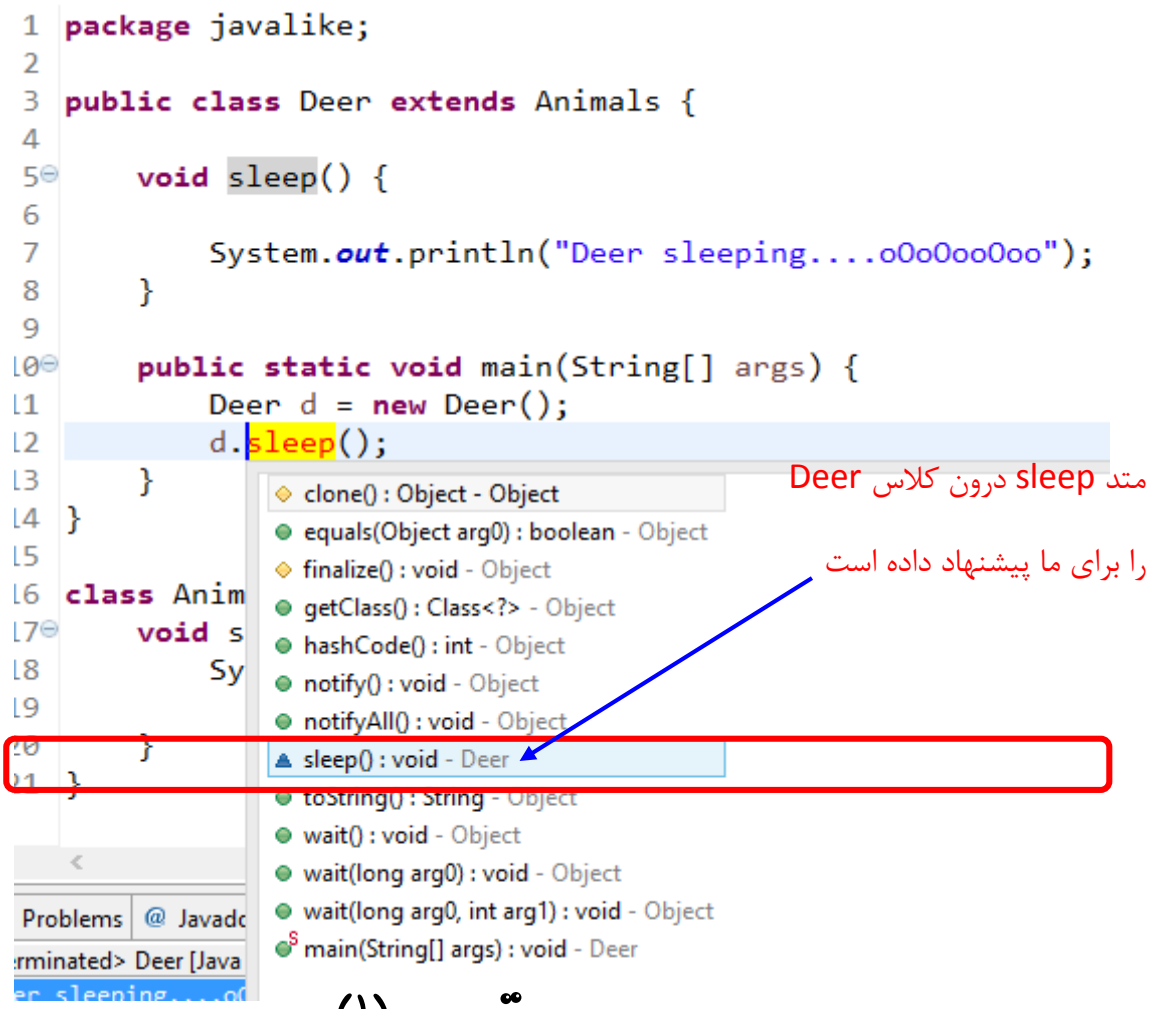
به مثال تکراری زیر که قبلا بررسی کردیم خوب نگاه کنید:

```
package javalike;

public class Deer extends Animals {

	void sleep() {

		System.out.println("Deer sleeping....oOoOooOoo");
	}

	public static void main(String[] args) {
		Deer d = new Deer();
		d.sleep();
	}
}

class Animals {
	void sleep() {
		System.out.println("animal sleeping....oOoOooOoo");

	}
}
```

خروجی:

```
Deer sleeping....oOoOooOoo
```

در این مثال شما مفاهیم ارث بری ، Overriding (بازنویسی) را مشاهده میکنید.کلاس Deer کلاس Animals را به ارث برده است.متد sleep کلاس Animals (پدر) درون کلاس Deer (فرزند) بازنویسی یا override شده است.وقتی که یک شی از کلاس Deer نظیر d میسازیم و با شی d قصد صدا زدن متدsleep را داریم برنامه به صورت خودکار متد sleep بازنویسی شده (override شده) درون کلاس فرزند یعنی Deer را به ما نمایش می دهد.تصویر(۱)



تصویر(۱)

پس وقتی ما در کلاس فرزند متد کلاس پدر را بازنویسی یا override میکنیم ،هنگام شی ساختن از کلاس فرزند بصورت خودکار برنامه متد بازنویسی شده (override شده) کلاس فرزند را شناسایی میکند.و ما تنها می توانیم این متد را صدا بزنیم.

حالا این مثال را به مثال زیر که یک مثال چندریختی است، تغییر می دهیم:

```
package javalike;

public class Deer extends Animals {

     void sleep() {

          System.out.println("Deer sleeping....oOoOooOoo");
     }

     public static void main(String[] args) {
          Animals a = new Deer();
```

```
            a.sleep();
      }
}

class Animals {
      void sleep() {
            System.out.println("animal sleeping....oOoOooOoo");

      }
}
```

خروجی:

```
Deer sleeping....oOoOooOoo
```

در اینجا چون رابطه IS-A بین کلاس Deer و Animals برقرار بود توانستیم در خط نارنجی از روش ساختن شی در چندریختی استفاده کنیم،یعنی توانستیم یک شی بسازیم از نوع کلاس پدر (Animals) و سازنده آن را سازنده کلاس فرزند (Deer) قرار بدیم.با این کار شی کلاس پدر(Animals) به شکل شی کلاس فرزند(Deer) نیز ظاهر می شود.در اینجا شی a که از نوع کلاس Animals (کلاس پدر) و سازنده آن سازنده کلاس Deer (فرزند) هست ایجاد کرده ایم. حالا شی ای که از نوع پدر و سازنده ان فرزند است و به نوعی می توان گفت شی ای است که علاوه بر نقش و شکل کلاس پدر می تواند باتوجه به نیاز در نقش و شکل کلاس فرزند نیز ظاهر شود مثل شی a ، دارای چه خصوصیات ، اختیارات و کارایی می باشد؟

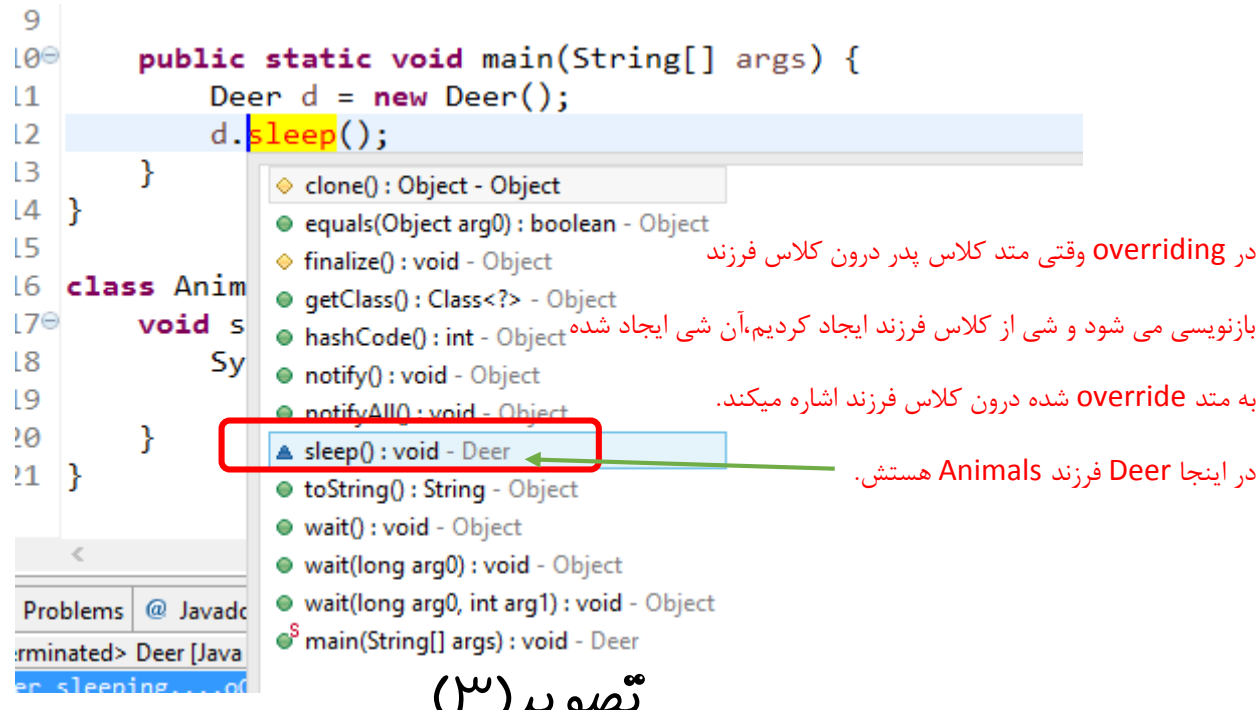
خصوصیات و اختیارات شی ای که به روش چندریختی تعریف می شود:

شی a (شی ای که به روش چند ریختی تعریف شده است)درون مثال بالا دارای خصوصیات زیر است:

شی a وقتی یک متد بازنویسی شده (override شده) مانند sleep را صدا می زند برنامه دستور متد sleep ای که درون کلاس فرزند قرار دارد را اجرا میکند: همان طور که در خط خاکستری می بینید شی a متد sleep را صدا زده است اما برنامه بین متد sleep کلاس پدر و کلاس فرزند متد sleep بازنویسی شده (override) شده کلاس فرزند را بصورت خودکار اجرا میکند. خب تا اینجا شبیه overriding متد کلاس پدر را در کلاس فرزند override کردیم و بعدش متد را صدا زدیم که متد override شده کلاس فرزند اجرا شد اما اینجا یک تفاوتی با overriding وجود دارد :

۱. در چندریختی نوع شی از نوع پدر و سازنده آن فرزندان می باشد.اما در overriding نوع و سازنده شی از کلاس فرزند انتخاب می شود.

۲. در چندریختی شی ایجاد شده وقتی به متد override شده اشاره میکند در اصل دارد به متد اصلی خود کلاس پدر اشاره میکند اما دستوری که اجرا می شود ،دستور درون متد بازنویسی شده (override شده) درون کلاس فرزند است!!!! تصویر (۲) و درحالی که در overriding شی ایجاد شده از کلاس فرزند تنها به متد override شده درون کلاس فرزند اشاره میکند.تصویر(۳) .....تصویر(۲) و تصویر(۳) را باهم مقایسه کنید.



۳. در کل در چندریختی به زبان جاوا ما یک شی که نوع آن از نوع کلاس پدر و سازنده آن از سازنده یکی از کلاسهای فرزندان انتخاب میکنیم.شی ایجاد شده به تمام متغیرها و متدهای کلاس پدر دسرسی دارد و علاوه بر آن متدهای کلاس پدر که درون کلاس فرزند override شده است را اجرا میکند با این تفاوت که به جای اجرای دستور درون متد کلاس پدر دستور درون متد بازنویسی شده (override شده) کلاس فرزند را اجرا می کند یعنی متد متعلق به کلاس پدر هست اما اجرا دستور متعلق به دستور متد override شده کلاس فرزند می باشد. و این نکته را در نظر بگیرید که در چندریختی شی ایجاد شده تنها به تمام ویژگی ها و رفتارهای کلاس پدر (کلاسی که از نوع آن می باشد) و به متد override شده کلاس فرزند دسرسی دارد.یعنی این شی ایجاد شده هیچ دسرسی به متغیرها و متدهای غیر override شده کلاس فرزند ندارد.مثال زیر را ببینید:

```
package javalike;

public class Deer extends Animals {
	int age;

	void sleep() {

		System.out.println("Deer sleeping....oOoOooOoo");
	}

	public static void main(String[] args) {
		Animals a = new Deer();
		a.sleep();
		a.eating();
		a.walk();
		a.color = "black";
		System.out.println(a.color);
	}
}

class Animals {
	String name;
	String color;
	int speed;

	void sleep() {
		System.out.println("animal sleeping....oOoOooOoo");

	}

	void eating() {
		System.out.println("animal is eating....");

	}
```

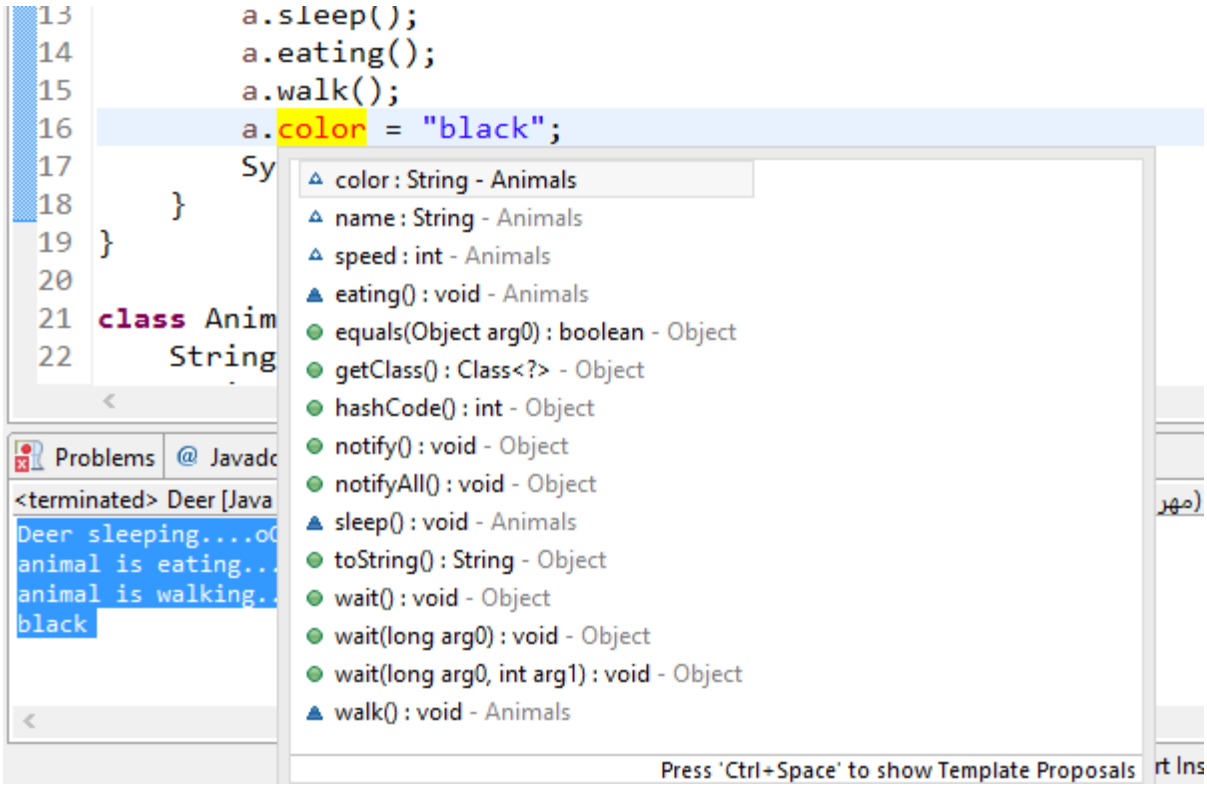
```
	void walk() {
		System.out.println("animal is walking....");

	}
}
```

خروجی:

```
Deer sleeping....o0o0oo0oo
animal is eating....
animal is walking....
black
```

- در این مثال در خط سبز رنگ یک چندریختی صورت گرفته است.شی a از نوع کلاس Animals (کلاس پدر) و سازنده ان از نوع سازنده کلاس Deer (فرزند) می باشد.با این اوصاف شی a چون نوعی از کلاس Animals هستش به تمام ویژگی ها و رفتار های کلاس Animals دسرسی دارد و همچنین چون سازنده آن از نوع کلاس فرزند(Deer) هستش و در کلاس فرزند (Deer) متد کلاس پدر (Animals) override شده است این شی (a) به متد override شده کلاس فرزند نیز دسرسی دارد تصویر(۴) با این تفاوت که متد را از درون کلاس پدر(animalse) صدا میزند ولی اجرای دستور مربوط میشه به دستورات درون متد override شده کلاس فرزند!

- خیلی به جزییات پرداختیم!!! کلا اینو نظر بگیرید که در چندریختی وقتی شی میسازیم این شی به تمام ویژگیها و رفتارهای کلاسی که از نوع آن می باشد و همچنین به متد override شده کلاس فرزند(کلاسی که از سازنده آن استفاده کرده ایم) نیز دسرسی دارد.



تصویر(۴)

- همان طور که از تصویر(۴) مشخص هست شی a به تمام ویژگی ها و رفتار های کلاس Animal و همچنین به متدی (sleep) از کلاس Animals که درون کلاس Deer بازنویسی (override شده) دسترسی دارد.

یک مثال خوب از چندریختی:هر مسئله آسونی یک زمانی سخت بوده،حل شده و راحت گردیده است.پس چندین بار این مثال زیر را بررسی کنید و ناامیدی هم ممنوع!!!!

```
package javalike_Polymorfism;

class Employee extends Person {

	public Employee(String name, int age) {
		super(name, age);

	}
```

```
	public void salamKardan() {

		System.out.println(this.name + " ba sen " + this.age + " salam kard");
	}

	public void walke() {

		System.out.println(this.name + " ba codemeli " + this.codemeli
				+ " dar hal rah raftan hast");
	}
}

class Woman extends Person {
	public Woman(String name, String codemeli) {
		super(name, codemeli);
	}

	public void khandidan() {
		System.out.println(this.name + " ba codemeli " + this.codemeli
				+ " is :-)");
	}
}

class Consumer extends Person {
	public Consumer(String name, int age) {
		super(name, age);
	}

	public void calculator(int a, int b) {
		int multiplication = a * b;
		System.out
				.println("jafar be andaze " + multiplication + "$ pool
darad");
	}
}

class Person {
	int age;
	String name;
	String codemeli;
	int weight;

	public Person(String name, int age) {
		this.name = name;
		this.age = age;
	}
```

```
	public Person(String name, String codemeli) {
		this.name = name;
		this.codemeli = codemeli;
	}

	public void salamKardan() {

		System.out.println(" ba sen " + " salam kard");
	}

	public void walke() {

		System.out.println(" ba vazn " + " dar hal rah raftan hast");
	}

	public void khandidan() {
		System.out.println("person is :-)");
	}

	public void calculator(int a, int b) {
		int sum = a + b;
		System.out.println(sum);
	}
}

public class MainTest {

	public static void main(String[] args) {
		Person ali = new Employee("ali", 20);
		ali.salamKardan();
		Person zahra = new Woman("zahra", "35102369587");
		zahra.khandidan();
		Person jafar = new Consumer("jafar", 25);
		jafar.calculator(1000, 900);

	}
}
```

خروجی:

```
ali ba sen 20 salam kard
zahra ba codemeli 35102369587 is :-)
jafar be andaze 900000$ pool darad
```

- یک مثال کاربردی از چندریختی و پلی مورفیسم را در بالا مشاهده میکنید.
- این مثال از پنج کلاس تشکیل شده است.

```
class Employee extends Person {

	public Employee(String name, int age) {
		super(name, age);

	}

	public void salamKardan() {

		System.out.println(this.name + " ba sen " + this.age + " salam kard");
	}

	public void walke() {

		System.out.println(this.name + " ba codemeli " + this.codemeli
				+ " dar hal rah raftan hast");
	}
}
```

- کلاس Employee کلاس person را به ارث برده است.
- چون پدر کلاس Employee دارای سازنده می باشد پس بالجبار باید در کلاسمون سازنده تعریف کنیم و و چون سازنده کلاس پدر از نوع پارامتر دار هستش از کلمه کلیدی **super** استفاده میکنیم.
- در این کلاس دو متد به نام های walke وsalamKardan بازنویسی (override) شده است.همان طور که میدانید وقتی یک کلاس کلاسی دیگر را به ارث میبرد به تمام ویژگی ها و رفتارهای پدرش دسرسی دارد و دیگه نیاز نیست اون ویژگی ها و رفتارهای که نیاز دارد و درون کلاس پدرش هست را دوباره پیاده سازی کند.در اینجا هم متغیرهای مورد نظرش ست (set) شده اند.این متغیرها حکم متغیرهای نمونه را دارند.

```
class Woman extends Person {
	public Woman(String name, String codemeli) {
		super(name, codemeli);
	}

	public void khandidan() {
		System.out.println(this.name + " ba codemeli " + this.codemeli
				+ " is :-)");
	}
}
```

- کلاس Woman کلاس person را به ارث برده است.
- در دلم مونده میخوام این بخش متغیرها رو توضیح بدم دوست ندارم کسی داخلش به مشکل برخورد کنه ☺

شما وقتی یک کلاس را به ارث میبرید در واقع تمام متغیرها و متدهای آن را به ارث برده اید.به عبارتی حساب کنید که تمام ویژگی ها و رفتارهای کلاس پدر در کلاس شما پیاده سازی و وجود دارند فقط کافیه که بهشون اشاره کنی و صداشون بزنی تا مثل شیر حاضر بشند ☺

```
public Woman(String name, String codemeli) {
		super(name, codemeli);
	}
```

در این کد ما دو متغیر محلی (پارامتر) از نوع رشته در سازنده کلاس تعریف کرده ایم حالا این دو متغیر را با استفاده از کلمه کلیدی super میفرستیم به سازنده کلاس پدر یعنی person

```
public Person(String name, String codemeli) {
		this.name = name;
		this.codemeli = codemeli;
	}
```

این دو متغیر ریخته میشن درون دو متغیر محلی (پارامتر) سازنده کلاس پدر.حالا با کلمه کلیدی this دو متغیر نمونه کلاس پدر صدا زده شده و درونشون پارامترهای سازنده ریخته شده است.

با این کار دو متغیر نمونه name و codemeli برای کلاس فرزند در واحد حافظه با مقادیری که بهشون دادیم ذخیره می شود.

دقیقا شبیه مقدار دهی به متغیر نمونه کلاس در حالی عادی هست فقط چون درون کلاس پدر قرار داره با دستور خاص خودش این متغیرها رو مقداردهی میکنیم.

- در همه کلاس های آبی،صورتی،نارنجی متدهای کلاس پدر override شده اند.
- در کلاس Person سری ویژگی ها و رفتارهای کلی میان همه اشخاص پیاده سازی شده است. و دارای دو سازنده می باشد.

```
public class MainTest {

	public static void main(String[] args) {
		Person ali = new Employee("ali", 20);
		ali.salamKardan();
		Person zahra = new Woman("zahra", "35102369587");
		zahra.khandidan();
		Person jafar = new Consumer("jafar", 25);
		jafar.calculator(1000, 900);
	}
}
```

- در کلاس MainTest عمل چند ریختی را انجام داده ایم.نوع شی ها از نوع کلاس پدر و سازنده ها از نوع سازنده فرزندان می باشد.
- ما در چندریختی متد کلاس پدر که override شده را اجرا میکنیم اما دستورات این متد که در کلاس فرزند وجود دارد اجرا می شود.یعنی شی کلاس پدر به شکل کلاس های فرزندش ظاهر شده است.
- همان طور که در این مثال مشاهده میکنید کلاس person (شخص) گاهی در شکل یک Woman (زن)،گاهی در شکل یک Consumer (مشتری) و گاهی در شکل یک Employee (کارمند) ظاهر شده است.پس وقتی بخوایم که یک شی از کلاس به شکل اشیای دیگر ظاهر شود،باید از چندریختی استفاده کنیم.

✓ در کل در چند ریختی وقتی یک شی از کلاس سازنده اش را از میان فرزندان خود انتخاب میکند و با override کردن متدهای خود درون کلاس های فرزندان ،به شکل فرزندان خود ظاهر می شود.

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.