آموزش زبان برنامه نویسی جاوا

متدها

جلسه هفدهم

نویسنده : رحمان زارعی

جاوا را ساده،آسان و شیرین بنوشید!!!!

داریم به ایستگاه های آخر آموزش های پایه ای جاوا می رسیم! ایستگاه شماره ۱۷! ایستگاه متدها!ما در جلسات قبل بصورت سطحی و گذرا نگاهی به متدها داشتیم حالا در این جلسه قصد داریم مفصل بهش بپردازیم چون خیلی خیلی با متدها کار داریم.

یک متد در جاوا مجموعه ای از دستوراتی است که عملیاتی را برای ما انجام می دهند.

برای مثال، شما وقتی متد `System.out.println()` را صدا میزنید،در واقع این متد چندین دستور را در خودش اجرا میکند تا دستور چاپ پیام در خروجی را نمایش دهد.

یکی از کاربرد های متدها نظم بخشیدن به دستورات برنامه نویسی مان هستش،برای مثال نیاز به دستوری داریم که دو عدد از ورودی بگیرد و حاصل جمع آنها را در خروجی چاپ کند. راه اول بصورت زیر است:

```
package javalike;

import java.util.Scanner;

public class Seventeenth_Session {

	public static void main(String[] args) {

		Scanner input = new Scanner(System.in);
		int a = input.nextInt();
		int b = input.nextInt();
		int sum = a + b;
		System.out.println(sum);

	}

}
```

خروجی:این برنامه را بصورت زیر تست کرده ایم.

```
5
10
15
```

- دو عدد ۵ و ۱۰ را به عنوان ورودی به برنامه دادیم و حاصل جمع این دو عدد ۱۵ شد.
- در این مثال تمام کد برنامه خود را در متد main کلاس خود نوشتیم. خب در نگاه اول این که مشکلی نداره! نظمشم که خیلی خوبه! تازه در متد main کلاسمون قرار داره! پس ایرادش چیه؟!!!!! درسته ظاهرا نظم خوبه داره، اما اگه همه کدهای برنامه خود را در متد main بزنیم تصور کنید ۱۰۰۰ دستور خاص که عملیاتی را برامون انجام می دهند را مستقیم و پشت سر هم در متد main کلاس پیاده سازی کنیم مثلا دستور اول حاصل جمع دو عدد دستور دوم قدر مطلق دو عدد دستور سوم میانگین نمرات دانشجویان دستور چهارم به توان ۲ رساندن یک عدد و.... ،وقتی همه این دستورات پشت سر هم قرار میگیره حتی اگه خودمون هم کدش رو زده باشیم بعد از مدتی قاطی میکنیم که این دستورات چکارهایی برنامون انجام میدهند یا این دستور مربوط به کدام بخش از عملیاتی که ذکر کردیم هست؟و....شاید بگید کامنت گذاری(قرار دادن توضیح)می کنیم بازم جالب نیست تمام کدامون در متد main پشت سر هم بزنیم! در کل حرفه ای نیست! قشنگ نیست! ☺ بر فرض خواستیم دستور مثلا عملیات فاکتوریل را بین ۱۰۰۰ دستور عملیاتیپیدا کنیم تصورش واقعا سخته و به زحمت میندازمون 😐
- راه کار چیه؟ متدها ما میایم برای هر دستور عملیاتی یک متد تعریف میکنیم و ان دستور را در بدنه متد پیاده سازی میکنیم و بعد با توجه به نیاز تنها نام متد را صدا میزنیم که برامون دستور را اجرا کند مثلا برای هزار دستور عملیاتی ۱۰۰۰ متد تعریف می کنیم که هر وقت نیاز به دستور خاص داشتیم دیگه با متدش کار کنیم نه دستورات درونش!!! مثلا همین متد `System.out.println()` ما با دستورات درونش دیگه کاری نداریم تنها صداش میزنیم و برامون کار انجام میده!فرض کنید هربار که میخواستیم یک پیام چاپ کنیم مستقیم دستورات درون این متد را در برناممون می نوشتیم تصور کنید چه کار طاقت فرسایی میشد و چقدر تعداد خط کد برنامون افزایش می یافت.پس اینجاست که درمی یابیم که متد ها چقدر مفید و با ارزش میباشند ☺

حالا همین مثال قبل را با راه دوم یعنی پیاده سازی دستورات درون متد بررسی میکنیم:

```java
package javalike;

import java.util.Scanner;

public class Seventeenth_Session {

	public static void total() {
		Scanner input = new Scanner(System.in);
		int a = input.nextInt();
		int b = input.nextInt();
		int sum = a + b;
		System.out.println(sum);

	}

	public static void main(String[] args) {

		total();

	}

}
```

خروجی: این برنامه را بصورت زیر تست کرده ایم.

```
99
2
101
```

- دو عدد ۹۹ و ۲ از ورودی گرفته و حاصل جمع آنها ۱۰۱ را در خروجی چاپ کرده است.
- در این مثال یک متد در بدنه کلاس خود به نام total ایجاد کرده و دستورات مورد نظر را درون آن پیاده سازی کرده ایم. و در متد main کلاس ،متد total را صدا زده ایم.
- وقتی متدtotal در متد main صدا زده شود،دستورات درون متد total خط به خط اجرا می شود.
- همون طور که در متد main مشاهده میکنید خبری از شلوغی نیست تنها نام یک متد را صدا زده ایم که برامون عملیاتی را اجرا میکند.
- شاید بگید کلاس مون شلوغ شده!!! بدنه کلاس محل تعریف متدها و متغیر های برنامه مورد نظر ما می باشد و برای این کار ساخته شده! چون هر کلاس تنها یک متد main داره و لازمه اجرای برنامه دارا بودن متد main می باشد باید در حد امکان بصورت منظم و مرتب کد های خود را درونش بزنیم که با زیاد شدن کدبرنامه دچار سردرگمی نشویم.
- قصد ما تا اینجا فقط درک تفاوت دستورات برنامه درون متد و خارج از متد بود , پیاده سازی ، ایجاد و جزییات متد ها رو در ادامه آموزش بررسی میکنیم.

حالا ما در ادامه اموزش یاد میگیریم که چطور متدی ایجاد کنیم که :

- یک مقدار برامون برگرداند.
- بدون برگردادن مقدار باشد.
- به عنوان ورودی پارامتر بگیرد.
- بدون ورودی و پارامتر باشد.

ایجاد متد

مثال زیر و توضیح پیرامون ترکیب و جز به جز متد را در نظر بگیرید:

```
public static int methodName(int a, int b) {
// body
}
```

جز به جز این متد بصورت زیر است:

- **public static** : اصلاح کننده (modifier)، سطح دسرسی به این متد public و نوع استفاده static می باشد.
- **int** نوع مقداری که بر میگرداند، از نوع عدد صحیح می باشد.
- methodName نام متد
- **int a, int b** پارامترهای ورودی متد (پارامترها نوعی متغیر محلی می باشند که تنها در بدنه متد قابل استفاده هستند)

شکل کلی پیاده سازی یک متد بصورت زیر است:

```
modifier returnType nameOfMethod (Parameter List) {
 // method body
}
```

Modifier (اصلاح کننده سطح دسرسی): به معنای نوع دسرسی به متد و گزینه های استفاده از متد می باشد.

returnType (نوع مقداری که متد بر میگرداند): متد ممکن است یک مقدار برگرداند یا هیچ مقداری برنگرداند.

nameOfMethod (نام متد): تعیین نام برای متد،نام متد و لیست پارامترهای متد امضا متد می باشند.

Parameter List (پارامتر های متد): لیست پارامترهای متد شامل نوع پارامترها،تعداد پارامترها (پارامتر نوعی متغیر محلی می باشد که تنها در بدنه متد قابل استفاده است) داشتن پارامتر برای یک متد اختیاری است یعنی با توجه به نیاز ، یک متد می تواند چندین پارامتر براش تعیین کنیم یا این که بدون پارامتر متد خود را تعریف کنیم.

method body (بدنه متد): دستورات مورد نظر خود را درون بدنه متد تعریف میکنیم.

مثال:

در زیر سورس کد یک متد به نام `minFunction` را مشاهده میکنید.این متد دو پارامتر `n1,n2` را میگیرد و مینیمم بین آن دو را محاسبه و برمیگرداند.

```
public static int minFunction(int n1, int n2) {
int min;
if (n1 > n2)
min = n2;
else
min = n1;

return min;
}
```

- همان طور که گفتیم یک متد می تواند مقداری را بعد از صدازدن و استفاده از ان به ما برگرداند یا نه هیچ مقداری را برنگرداند! برای برگردادن مقدار در یک متد باید ابتدا نوع ان را مشخص کرد، نوعی که متد هیچ مقداری را برنمیگرداند از نوع void می باشد و نوعی که مقدار به ما برمیگرداند می تواند با توجه به نیاز از نوع عدد صحیح،اعشاری،رشته،کاراکتر،شی و...باشد.
- بعد از تعیین نوع حالا بعد از عملیاتی که در بدنه متد انجام دادیم و قصد ما برگردادن نتیجه محاسباتی که در متد صورت گرفته،برای برگردادن نتیجه یا هر مقداری که مدنظر باشد از کلمه **return** استفاده میکنیم:

```
return value;
```

نکته مهم : مقداری که در متد خود جلوی کلمه **return** استفاده میکنیم باید نوعش از نوع متدی باشه که در آن قرار داره:

مثلا در مثال بالا تابع `minFunction` از نوع `int` می باشد پس مقداری که در این متد برگردانده می شود باید از نوع `int` باشد که در مثال بالا متغیر محلی `min` از نوع `int` هستش چون متدی که در ان قرار داره از نوع `int` می باشد.

صدا زدن متد

برای استفاده از متد باید آن را صدا زد! وقتی یک متد رو صدا میزنیم ، از اجرا دستورات درون بدنه خود یک مقدار را به ما برمیگرداند یا این که بعد از اجرای دستورات درون بدنه خود هیچ مقداری را برنمیگرداند!

مثال:

در اینجا متد sum هیچ مقداری را برنمیگرداند و نوعش void می باشد:

```
package javalike;

public class D {

	public void sum(int a, int b) {

		int sum = a + b;
		System.out.println(sum);
	}

	public static void main(String[] args) {
		D d = new D();
		d.sum(10, 15);

	}

}
```

خروجی:

```
25
```

- در بدنه کلاس یک متد به نام sum که سطح دسرسی ان از نوع public و نوع متد void می باشد.دارای دو پارامتر که از نوع عدد صحیح و عملیات حاصل جمع دو عدد را انجام داده و در خروجی چاپ می کند.و در متد main کلاس مون یک شی از کلاس خود ساخته و با استفاده از ان شی متد sum را صدا زده ایم.
- همان طور که مشاهده میکنید هیچ خبری از برگردادن مقدار در این متد مشاهده نمی شود. و این متد بعد از اجرای دستورات درون خود بسته می شود .

یکی از متدهای پرکاربرد جاوا که از نوع void هستش و مقداری را برنمیگرداند متد زیر می باشد:

```
System.out.println("this is javalike1.ir ");
```

حال مثال بالا را بصورت زیر تغییر می دهیم بصورتی که متد sum نتیجه اجرای دستورات را برمیگرداند:

نکته : بخش مهم یادگیری جاوا از روی مثال و تمرین کردن هست من خودم که این شیوه رو دنبال میکنم ، پس اگه توضیحات بنده گنگ یا غیرقابل فهم بود به مثالها توجه کنید بهتر درک میکنید.

```
package javalike;

public class D {

	public int sum(int a, int b) {

		int sum = a + b;
		return sum;
	}

	public static void main(String[] args) {
		D d = new D();
		int total = d.sum(10, 15);
		System.out.println(total);
	}

}
```

خروجی:

```
25
```

- در این مثال متد sum از نوع void به نوع int تغییر کرده، و درون متد بعد از اجرای دستورات مقداری برگردانده شده است.

```
int total = d.sum(10, 15);
```

- حالا در متد main متد sum صدا زده شده است.حالا این صدا زده شدن و استفاده کردن از ان کمی متفاوت از مثال قبل هست،چون این متد بعد از اجرای دستورات مقداری را به ما پس میدهد ما باید این مقدار را در یک متغیر که از نوع همون مقداری هست که قراره برگشت داده بشه ذخیره کنیم.در اینجا نتیجه برگشت داده شده متد sum درون متغیر total ریخته می شود.

```
System.out.println(total);
```

- در پایان مقدار متغیر total که همان مقدار برگردانده شده متد sum هستش را چاپ میکنیم.

❖ پس فرق یک متد از نوع **void** و غیر **void** این هست که متد از نوع **void** بعد از صدا زدن هیچ مقداری را برای استفاده به ما پس نمی دهد، و متد غیر **void** یک مقدار را از نوع متدش که می تواند **int,String,char,short,object** و... باشد به ما پس بدهد که میتوانیم در درون متغیری از نوع مقدار برگشت داده شده ذخیره کنیم.

❖ **نکته** (ربطی به نوع برگردادن یا عدم برگردادن مقدار متد نداره ☺) : صرفا جهت یادآوری!

در مثال قبل ما یک متد تعریف کردیم و برای صدا زدن و استفاده از ان یک شی از کلاسی که متد درونش قرار داشت ساختیم و با استفاده از شی به متدهای کلاس دسرسی پیدا کرده و متد مورد نظر را صدا زدیم.

همان طور که میدونید می توانیم مستقیم و بدون شی ساختن از یک کلاس از ویژگی ها و متدهای ان استفاده کنیم!خب چطور؟ متغیر یا متد مورد نظر را **static** تعریف کنیم.

مثال:

همان طور که در سورس کد زیر مشاهده میکنید با تعریف متد **sum** از نوع **staic** دیگر نیاز نیست که برای صدا زدن این متد در متد **main** از کلاس شی بسازیم و با شی این متد را صدا بزنیم:

```
package javalike;

public class D {

	public static int sum(int a, int b) {

		int sum = a + b;
		return sum;
	}

	public static void main(String[] args) {

		int total = sum(10, 15);
		System.out.println(total);
	}

}
```

خروجی:

25

- با استاتیک کردن یک متد یا متغیر نیازی به شی ساختن از کلاس برای دسرسی به متد یا متغیر استاتیک نیست.

مثال:

در مثال زیر قصد داریم مقدار دو متغیر a و b را با هم عوض کنیم! در اینجا یک متد به نام swapFunction داریم که این کار را برامون انجام میده:

```
package javalike;

public class swappingExample {

	public static void main(String[] args) {
		int a = 30;
		int b = 45;

		System.out.println("Before swapping, a = " + a + " and b = " + b);

		// Invoke the swap method
		swapFunction(a, b);
		System.out
				.println("\n**Now, Before and After swapping values will be
same here**:");
		System.out.println("After swapping, a = " + a + " and b is " + b);
	}

	public static void swapFunction(int a, int b) {

		System.out.println("Before swapping(Inside), a = " + a + " b = " + b);
		// Swap a with b
		int temp = a;
		a = b;
		b = temp;

		System.out.println("After swapping(Inside), a = " + a + " b = " + b);
	}
}
```

خروجی:

```
Before swapping, a = 30 and b = 45
Before swapping(Inside), a = 30 b = 45
After swapping(Inside), a = 45 b = 30

**Now, Before and After swapping values will be same here**:
After swapping, a = 30 and b is 45
```

- ابتدا متد **main** درون کلاس بررسی میکنیم:

```
int a = 30;
int b = 45;
```

۱. تعریف دو متغیر از نوع عدد صحیح و مقداردهی مستقیم به انها.

```
System.out.println("Before swapping, a = " + a + " and b = " + b);
```

۲. چاپ مقدارهای دو متغیر **a**و**b** در خروجی، این پیام برای مشخص کردن مقدار دو متغیر قبل از عوض کردن مقادیر آنها با هم می باشد.

```
swapFunction(a, b);
```

۳. صدا زدن متد برای عملیات جابه جایی مقدارهای تو متغیر **a**و**b**

- حالا با صدا زدن متد `swapFunction(a, b);` کنترل برنامه میره سراغ اجرای دستورات درون این متد:

بررسی متد `swapFunction(a, b);`:

۴. این متد دو پارامتر از نوع عدد صحیح از ورودی میگیرد.

```
System.out.println("Before swapping(Inside), a = " + a + " b = " + b);
```

۵. مقدار پارامترهایی که از ورودی متد گرفته شده را چاپ میکند،میخواد بگه قبل از عوض کردن آنها،مقدار متغیرها چقدر بوده!

```
int temp = a;
a = b;
b = temp;
```

۶. فرایند مبادله مقدار به متغیرها می باشد.ما برای ریختن مقدار متغیر **a** درون **b** و مقدار **b** درون **a** نیاز به یک متغیر کمکی به نام **temp** داریم یعنی ابتدا مقدار **a** ریخته درون **temp** و سپس مقدار **b** رو میریزیم درون **a** و در نهایت مقدار متغیر کمکی **temp** که حاوی مقدار **a** می باشد را درون **b** میریزیم.

```
System.out.println("After swapping(Inside), a = " + a + " b = " + b);
```

۷. در پایان مقدار تعویض شده متغیر ها با هم را چاپ می کنیم.

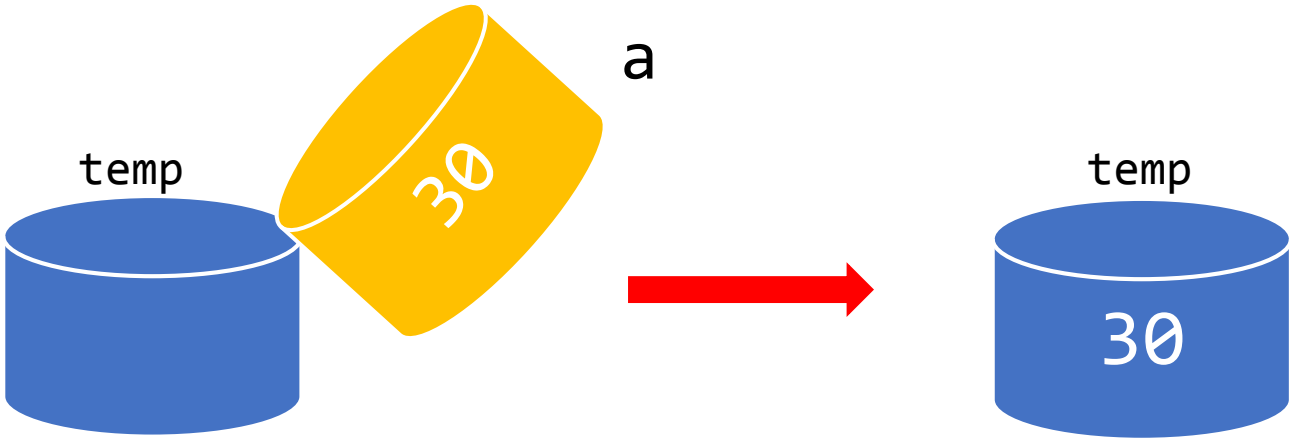
بعد از رسیدن به پایان اجرای دستورات متد *swapFunction*(a, b) کامپایلر خط بعد از این متد را در متد main اجرا میکند.

```
System.out
                    .println("\n**Now, Before and After swapping values will be
same here**:");
          System.out.println("After swapping, a = " + a + " and b is " + b);
```

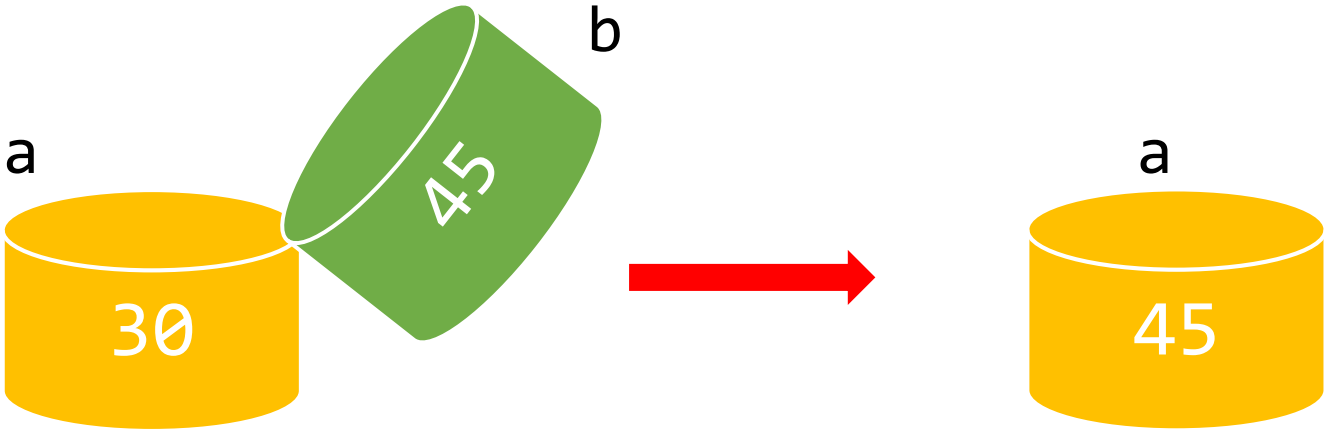
۸. در پایان مقدار دو متغیر a و b را چاپ میکند و میگه دو متغیر a,b که ابتدا در متد main تعریف کردیم مقدارش تغییر نکرد! زیرا متغیر a,b درون متد main و متد *swapFunction*(a, b) ربطی به هم ندارند و دو متغیر یا پارامتر درون متد *swapFunction*(a, b) تنها در بدنه این متد اعتبا دارد و قابل استفاده هست.

روش مبادله یا *swap* کردن دو متغیر بصورت شکل زیر برای درک بهتر طراحی شده است:

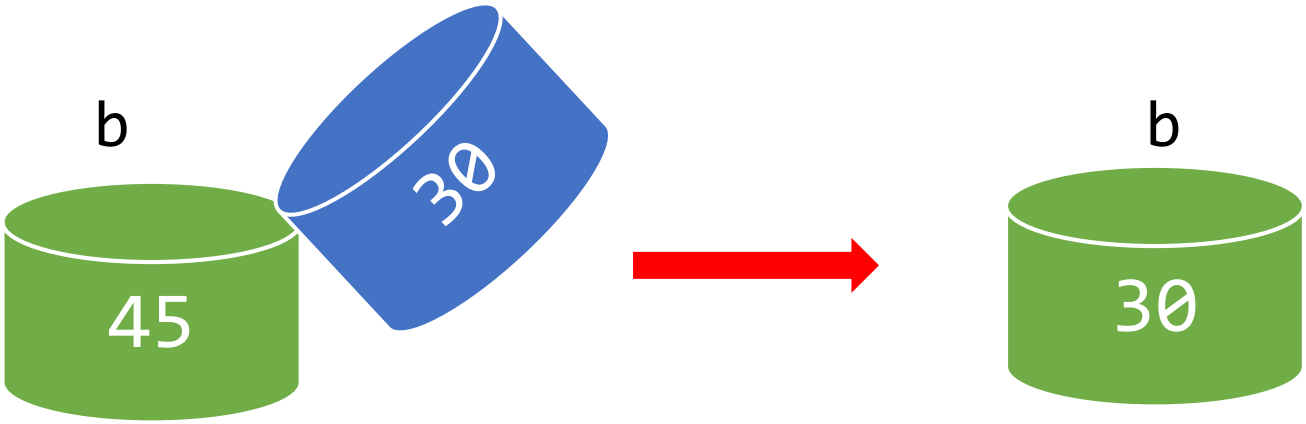

```
int a = 30;
int b = 45;
```




```
int temp = a;
```




```
a = b;
```




```
b = temp;
```

و در نهایت بعد از مبادله مقدار دو متغیر a,b شکل و مقدار این دو متغیر بصورت زیر است:

:Method Overloading

- عنوان معادل فارسی آن چیزی پیدا نکردم یا اگه بصورت فارسی میگفتم موجب گمراهی میشد ☺
- Overloading کردن متد یعنی چه؟! وقتی در یک کلاس دو یا چندین متد همنام با پارامترهای متفاوت داشتیم میگیم این متدهای همنام با پارامترهای متفاوت متدهای Overloading هستند.
- فرض کنید متدی به نام sum داریم در برنامه خود گاهی نیاز داریم متد sum دو عدد از ورودی بگیرد و مجموع آنها را محاسبه کند گاهی سه پارامتر از ورودی بگیرید و گاهی ۵ پارامتر در نگاه اول نیاز به سه متد با نام های sum1,sum2,sum3 داریم! و مشخص کردن پارامترها و دستورات درون برنامه اما اگه بخواهیم همین طوری برای متدهامون نام انتخاب کنیم، نام کم میاریم!!!! یا حداقل نام قشنگ کم میاریم! راه کار چیست؟جاوا این امکان را به شما داده که از متدهای همنام با پارامترهای متفاوت استفاده کنید اینجوری دیگه اسم کم نمیاریم ☺
- پس مفهوم Overloading ایجاد دو یا چند متد با نام های یکسان و پارامترهای متفاوت هستش.

مثال:

این مثال صرفا جهت یادگیری مفهوم overloading هستش وگرنه ما تو زندگیمون با همچین مثالی برنمیخوریم ☺

```
package javalike;
public class Mini {

	public static int sum(int a, int b) {

		return a + b;
	}

	public static int sum(int a, int b, int c) {

		return a + b + c;
	}

	public static int sum(int a, int b, int c, int d, int e) {

		return a + b + c + d + e;
	}

	public static void main(String[] args) {
		System.out.println(sum(1, 2));
		System.out.println(sum(1, 2, 3));
		System.out.println(sum(1, 2, 3, 4, 5));
	}

}
```

خروجی:

```
3
6
15
```

- در بدنه کلاس سه متد هم نام sum تعریف کرده که پارامتر آنها با هم فرق میکند و هر کدوم بعد از محاسبه حاصل جمع پارامترها نتیجه رو بر میگردونند و در متد main با دستور System.out.println(); مقداری که پس میدهند چاپ میشود.
- دلیل شی نساختن از کلاس و با شی صدا نزدن متدها ،استاتیک بودن متدهاست.

مثال:

یکی از مثال های زیبای موجود در این زمینه که دو متد داریم که مینیمم دو عدد را محاسبه میکنند و همنام هستند اما پارامترهای آنها متفاوت هست اینبار نه در تعداد بلکه در نوع پارامتر تفاوت دارند:

```
package javalike;
public class ExampleOverloading{
   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      double c = 7.3;
      double d = 9.4;
      int result1 = minFunction(a, b);
      // same function name with different parameters
      double result2 = minFunction(c, d);
      System.out.println("Minimum Value = " + result1);
      System.out.println("Minimum Value = " + result2);
   }
  // for integer
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;
      return min;
   }
   // for double
   public static double minFunction(double n1, double n2) {
     double min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;
      return min;
   }
}
```

خروجی:

```
Minimum Value = 6
Minimum Value = 7.3
```

- واقعا جاوا و برنامه نویسی حال میده وقتی ادم با این همه زیبایی برخورد میکنه ☺ ☺ ☺

```
int a = 11;
int b = 6;
double c = 7.3;
double d = 9.4;
```

- در اینجا ۴ متغیر دو تای اول از نوع صحیح و دوتای اخر از نوع اعشاری مشاهده میکنید.

```
int result1 = minFunction(a, b);
double result2 = minFunction(c, d);
```

- دو متد با نام یکسان و پارامتر متفاوت رو مشاهده میکنید.یکی از زیبایی های overloading اینجاس شما متد رو صدا میزنید حالا مقدار پارامتر ها رو عدد صحیح بده یا اعشاری فرق نمیکنه چون برای عدد صحیح متد با پارامتر صحیح و برای عدد اعشاری متد با پارامتر اعشاری را صدا میزنه!!

```
// for integer
public static int minFunction(int n1, int n2) {
int min;
if (n1 > n2)
min = n2;
else
min = n1;
return min;
}
// for double
public static double minFunction(double n1, double n2) {
double min;
if (n1 > n2)
min = n2;
else
min = n1;
return min;
}
}
```

- همان طور که مشاهده میکنید دستورات درون بدنه دو متد و نام متدها کاملا شبیه به هم هستند و این پارامتر های دو متد هست که تفاوت داره پس در موضوع overloading همین که پارامترهای متدها متفاوت باشه کافی ست.

```
System.out.println(“Minimum Value = “ + result1);
      System.out.println(“Minimum Value = “ + result2);
```

- در پایان مقداری که این متدها محاسبه میکنند و برمیگردانند چاپ می کنیم.

سازنده ها (The Constructors) :

در مورد سازنده ها در جلسات قبل صحبت کردیم،اینجا هم یادی ازش میکنیم چون جناب سازنده به نحوی با متد شباهت داره!

- سازنده وقتی یک شی از یک کلاس ساخته می شود ان را مقداردهی میکند.
- سازنده در یک کلاس باید هم نام با نام کلاس مربوطه می باشد.
- سازنده ها شبیه به متدها هستند اما هیچ مقداری را برنمیگردانند.
- به طور معمول استفاده ما از سازنده ها برای مقداردهی کردن به متغیرهای نمونه ای که در بدنه کلاس تعریف کرده ایم یا اجرای هر روش مورد نیاز برای راه اندازی کامل شی ای که از کلاس ایجاد میکنیم.
- همه کلاسها سازنده یا Constructors دارند،چه شما یک سازنده تعریف کنید یا خیر! زیرا جاوا بصورت اتوماتیک یک سازنده پیش فرض (default) برای مقداردهی اولیه به متغیرهایی که در بدنه کلاس هستند دارد و مقدار همه متغیرهای عضو کلاس را صفر می دهد.

مثال:

یک کلاس با سازنده پیشفرض!

```
package javalike;

class MyClass {
	int x;

	public static void main(String[] args) {
		MyClass m = new MyClass();
		System.out.println("x="+m.x);

	}
}
```

خروجی:

```
x=0
```

- برای این کلاس سازنده ای پیاده سازی نکردیم ، اما کلاس ما زرنگ تر از این حرفاست و از سازنده پیشفرض خود استفاده کرد و مقدار متغیر x را صفر داد و ما در متد main یک شی از کلاس MyClass بنام m ساختیم و از طریق شی m متغیر x را صدا زده و مقدار آن را در خروجی چاپ کردیم.

```
MyClass m = new MyClass();
```

- بخشی که با ابی مشخص کرده ایم سازنده کلاس ما را تشکیل میدهد که با استفاده از ان یک شی از کلاس ساختیم.

مثال:

کلاس با یک سازنده ساده، دست پرورده خودمان!

```
package javalike;

class MyClass {
	int x;

	MyClass() {
		x = 10;
	}

	public static void main(String[] args) {
		MyClass m = new MyClass();
		System.out.println(m.x);

	}
}
```

خروجی:

```
x=10
```

- اینجا خودمون یک سازنده برای کلاسمون مشخص کردیم و اجازه ندادیم سازنده پیشفرض مبیاد تو میدون و مقدار متغیرx مون رو صفر کنه
- هر بار که شی از کلاس ساخته میشه سازنده آن صدا زده میشه و دستورات درون آن اجرا می شود.

سازنده های پارامتر دار (parametarized constructor):

ما که به همین سادگی از سازنده ها نمی گذریم تا میتونیم کار ازشون میکشیم ☺

اکثر اوقات نیاز هست برای سازنده کلاس مان یک یا چند پارامتر تعریف کنیم.

تعین پارامتر برای سازنده های کلاس شبیه تعریف پارامتر برای متدهاست کافیست بعد از نام سازنده پارامترها یا متغیر محلی خود رو درون پرانتز تعریف کنیم.

مثال:

در اینجا یک مثال ساده از استفاده سازنده از پارامتر را بررسی میکنیم.

```
package javalike;

class MyClass {
	int x;

	MyClass(int i) {
		x = i;
	}

	public static void main(String[] args) {
		MyClass m = new MyClass(102);
		System.out.println("x="+m.x);

	}
}
```

خروجی:

```
x=102
```

- چون سازنده ما پارامتر ورودی دارد،برای ساختن شی از کلاس بعد از صدا زدن سازنده نیاز به مقداردهی به سازنده داریم.

و مثالی دیگر.....

```
package javalike;
```

```
class MyClass {
	int x;

	MyClass(int i) {
		x = i;
	}

	public static void main(String[] args) {
		MyClass m1 = new MyClass(102);
		MyClass m2 = new MyClass(800);
		System.out.println("x=" + m1.x);
		System.out.println("x=" + m2.x);

	}
}
```

خروجی:

```
x=102
x=800
```

- در اینجا دو شی به نام های m1,m2 از کلاسمان ساخته ایم.چون سازنده ما پارامتر دار هستش ،هر بار که سازنده برای ساختن شی صدا زده میشه باید پارامتر ورودی ان را مقدار دهی کرد که در اینجا برای شی m1 مقدار ۱۰۲ و برای شی m2 مقدار ۸۰۰ درنظر گرفته شده است.
- چرا مقدار x متفاوت است؟زیرا هر بار که یک شی از یک کلاس میسازیم یک نمونه از متغیرهای کلاس برای شی درنظر گرفته می شود که مقدار این متغیرها برای هر شی کاملا مجزا می باشد.

پیروز و موفق باشید

بازدید ازکانال

بازدید از سایت

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