آموزش زبان برنامه نویسی جاوا

Overriding در جاوا

جلسه بیست و دوم

نویسنده : رحمان زارعی

جاوا را ساده،آسان و شیرین بنوشید!!!!

در جلسات گذشته در مورد کلاس های پدر (superclasses) و فرزند (subclasses) صحبت کردیم.هنگامی که کلاس فرزند متد کلاس پدر را در بدنه خود پیاده سازی کند و دستورات درون بدنه آن را تغییر دهد ، به این عمل Override کردن متد کلاس پدر در کلاس فرزند میگوییم. ای تغییرات تنها شامل دستورات درون بدنه متد می شود و حق تغییر نوع،پارامترها و نام متد را نداریم میدونم هنوز مفهوم واضع نیست پس در ادامه با ما همراه شوید.

Override کردن متد کلاس پدر درون کلاس فرزند را با یک مثال ساده بررسی میکنیم:

فرض کنید کلاس A کلاس B را به ارث برده است یعنی کلاس A فرزند کلاس B هستش.

کلاس پدر (B) یک متد به نام calculator داره که دو پارامتر از ورودی میگیره و آن ها باهم جمع میکنه و در خروجی چاپ میکند بصورت زیر:

```
class B{

    void calculator(int x,int y){
        int sum=x+y;
  System.out.println("sum="+sum);

    }
}
```

همان طور که در مفاهیم ارث بری در جلسات گذشته بررسی کردیم وقتی یک کلاس ،کلاسی دیگر را به ارث می برد می تواند مستقیم از متد ها و ویژگی ها کلاس پدرش استفاده کند:

```
package javalike;

class B {

	void calculator(int x, int y) {
		int sum = x + y;
		System.out.println("sum=" + sum);
	}
}

public class A extends B {

	public static void main(String[] args) {

		A a = new A();
		a.calculator(10, 5);
	}
}
```

خروجی:

```
sum=15
```

- در اینجا کلاس A کلاس B را به ارث برده است،پس می تواند مستقیم از متغیرها و متدهای کلاس پدرش یعنی B استفاده کند.همان طور که مشاهده میکنید در متد main ،مستقیم با شی ساخته شده از کلاس A (فرزند) ،متد کلاس پدر را صدا زده ایم.
- اینجا ما با به ارث بردن از کلاس پدر ،مستقیم از متد کلاس پدر بدون هیچ تغییری استفاده کرده ایم.

خب جناب Override چه کاربردی داره؟!! حالا ماجرا را یکم تغییر میدیم!! کلاس A (فرزند) میخواد از متد `calculator` کلاس B (پدر) استفاده کند،اما یک مشکلی وجود دارد!!!!!!! کلاس A نیاز به متد `calculator` ای داره که دو پارامتر از ورودی بگیره و حاصل تفریق آنها را حساب کنه و در خروجی چاپ کند. اما متد `calculator` کلاس پدر تنها دو پارامتر از ورودی میگیرد و حاصل جمع آنها را حساب میکند! خب تکلیف چیست؟!! اینجا جناب Override خان به دادمون میرسه! Override میگه بیا متد `calculator` کلاس پدرت رو در کلاس خودت پیاده سازی کن با این تفاوت که بجای دستور جمع زدن دو پارامتر در بدنه متد `calculator` ،دستور تفریق دو پارامتر از هم را پیاده سازی کن! و در خروجی نمایش بده! بصورت زیر:

```
package javalike;

class B {

	void calculator (int x, int y) {
		int sum = x + y;
		System.out.println("sum=" + sum);
	}
}

public class A extends B {
	void calculator(int x, int y) {
		int subtract = x - y ;
		System.out.println("subtract=" + subtract);
	}

	public static void main(String[] args) {

		A a = new A();
		a.calculator(10,5);
	}
}
```

خروجی:

```
subtract=5
```

خب در اینجا در کلاس فرزند (A) متد calculator همنام با متد calculator کلاس پدر (B) می باشد.با توجه به نیاز متد calculator کلاس پدر (B) را در کلاس فرزند (A) Override کرده ایم.

پس Override کردن یعنی متد پدر را در کلاس فرزند پیاده سازی کنیم و باتوجه به نیاز در دستورات درون بدنه آن تغییر ایجاد کنیم.این تغییرات تنها می تواند شامل **تغییر دستورات درون** متد باشد.و ما نمی توانیم **نام،پارامتر و نوع** متد را تغییر بدیم.

با Override کردن متد کلاس پدر درون کلاس فرزند ،هنگام شی ساختن از کلاس فرزند و صدا زدن متد ،برنامه بصورت خودکار متد Override شده درون کلاس فرزند را تشخیص و برای شما معرفی میکند.یعنی دیگه کاری به متد همنام موجود در کلاس پدر ندارد.

به چه هنگام Override کردن به کارمون میاد؟ هنگامی که شما کلاس پدر را به ارث می برید و قصد دارید از متد کلاس پدر استفاده کنید و دستورات درون آن را با توجه نیاز تغییر بدید.مثل مثال قبل که متد محاسبه (calculator) در کلاس پدر دو

پارامتر از ورودی میگرفت و آنها را با هم جمع میکرد، اما در کلاس فرزند نیاز داشتیم این متد وقتی دو پارامتر از ورودی میگرد آن دو را از هم کم کند که برای این کار دست به Override کردن متد کلاس پدر در کلاس فرزند زدیم.

یک مثال از دنیای واقعی.....!!!

جعفر یک موتورسیکلت ۱۲۵ از پدر خود مشتی غلام به ارث برده است!! جعفر با موتورسیکلت پدر خود با دوستش گرگعلی مسابقه میزارد!!! گرگعلی در این مسابقه برنده می شود!!! جعفر امید خودش رو از موتورسیکلت از دست نمیده و دست به چاره ای می زند!! جعفر اقدام به دست کاری انجین موتور از ۱۲۵ به ۲۰۰ میکند و بار دیگر با گرگعلی مسابقه میدهد و پیروز می شود. جعفر یک موتورسیکلت ۱۲۵ را از پدر خود به ارث برد اما با توجه به نیاز آن را Override کرد و موتور را از ۱۲۵ به ۲۰۰ تغییر داد. ☺ ☺

نکته خیلی مهم : اگر متدی در کلاس پدر از نوع **final** تعریف شود ،دیگر نمی توانیم آن را در کلاس فرزند Override کنیم.

نکته خیلی مهم : کلاس های پدر چون کلاس های بزرگ تر،کلی تر و عمومی تری هستند،هنگام که متغیرها و متدهایی که در ان تعریف میکنیم باید این متغیرها و متدها بیشتر به سمت مفاهیم کلی و مشترک میان تمام فرزندان باشد. مثلا کلاس حیوانات ، کلاس حیوانات یک کلاس کلی هستش و به حیوان خاصی اشاره نکرده پس باید در ان مفاهیمی از متغیرها و متدهایی را درونش پیاده سازی کنیم که بین تمام خانواده حیوانات مشترک باشد!!! مثلا متد راه رفتن،آب خوردن،خوابیدن و متغیرهای نام،سن،وزن،نژاد بین همه حیوانات مشترک هستند پس میتوانیم این مفاهیم را در کلاس حیوانات پیاده سازی کنیم اما نمی توانیم متدی نظیر تخم گذاشتن را در حیوانات پیاده سازی کنیم!!! چون همه حیوانات تخم گذار نیستند.پس به این نکته دقت کنید هر چقدر به سمت پدر و پدر بزرگ و...یک کلاس می رویم و کلاس بزرگ تر می شود ،مفاهیمی باید پیاده سازی کنیم که کلی تر و مشترک میان تمام زیر کلاس ها یا کلاس های فرزند باشد.

نکته خیلی مهم : شما در یک کلاس که نقش پدر را دارد متدهای کلی را پیاده سازی می کنید،بعد فرزندان باتوجه به شرایط و نیازهای خود متدهای کلی که از پدر به ارث برده اند را Override میکنند. برای مثال کلاس حیوانات متدی به نام حرکت کردن را دارد ،حال کلاسی مانند سگ کلاس حیوانات را به ارث می برد.خب متد حرکت کردن که در کلاس پدر سگ یعنی حیوانات هستش کلی و برای همه حیوانات تعریف شده ،کلاس سگ میاد متد حرکت کردن پدر یعنی حیوانات را در بدنه خود پیاده سازی میکند و با توجه به نیاز خود این متد را Override میکند.مثال زیر گویای این حرف است:

```
package javalike;

class Animal {
	public void move() {
		System.out.println("Animals can move");
	}
}

class Dog extends Animal {
	public void move() {
		System.out.println("Dogs can walk and run");
	}

	public void bark() {
		System.out.println("Dogs ahoooo!!! ahoooo!!!");
	}
}

public class TestDog {

	public static void main(String args[]) {

		Dog b = new Dog(); // Dog object

		b.move(); // runs the method in Dog class
		b.bark();
	}
}
```

خروجی:

```
Dogs can walk and run
Dogs ahoooo!!! ahoooo!!!
```

- در این برنامه کلاس Dog کلاس Animal را به ارث برده است.
- کلاس Dog متد move کلاس Animal را در بدنه خود پیاده سازی کرده و باتوجه به نیاز آن را override کرده است.
- همان طور که در رنگ سبز می بینید ،متد move در کلاس Animal رفتار کلی حرکت کردن برای همه حیوان که مشترک هست راپیاده سازی کرده است. در خط زرد رنگ متد move کلاس Animal درون کلاس Dog، override و دستورات درون بدنه آن تغییر داده شده است. نوعی بومی سازی انجام شده یعنی متد move که حرکت کردن مشترک بین حیوانات هستش ("Animals can move") را به راه رفتن و پریدن ("Dogs can walk and run") که مخصوص حیوان سگ می باشد تغییر داده شده است.

- از کلاس Dog در کلاسی مجزا به نام TestDog شی ساخته و متدهای move و bark را صدا زده ایم.

قوانین مهم برای متدی که میخواد Overrid شود:

- **پارامتر** متد Overrid شده پدر در کلاس فرزند باید دقیقا با **پارامتر** متد پدر یکسان باشد.

```
package javalike;

class B {

	int calculator(int x, int y) {
		int sum = x + y;
		return sum;
	}
}

public class A extends B {
	int calculator(int x, int y) {
		int subtract = x - y;
		return subtract;
	}

	public static void main(String[] args) {

		A a = new A();

		System.out.println(a.calculator(10, 5));
	}
}
```

خروجی:

```
5
```

❖ همان طور که می بینید پارامتر متد override شده درون کلاس فرزند با پارامتر متد درون کلاس پدر یکسان است.

- **نوع** متد Overrid شده پدر در کلاس فرزند باید دقیقا با **نوع** متد پدر یکسان باشد.

```
package javalike;

class B {

	double calculator(int x, int y) {
		double sum = x + y;
		return sum;
	}
}

public class A extends B {
	int calculator(int x, int y) {
		int subtract = x - y;
		return subtract;
	}

	public static void main(String[] args) {

		A a = new A();

		System.out.println(a.calculator(10, 5));
	}
}
```

خروجی:

خطای کامپایل!!!

❖ همان طور که در برنامه بالا مشاهده میکنید نوع متد override شده درون کلاس فرزند با متد درون کلاس پدر یکسان نیست.

```
package javalike;

class B {

	int calculator(int x, int y) {
		int sum = x + y;
		return sum;
	}
}
```

```
public class A extends B {
	int calculator(int x, int y) {
		int subtract = x - y;
		return subtract;
	}

	public static void main(String[] args) {

		A a = new A();

		System.out.println(a.calculator(10, 5));
	}
}
```

خروجی:

```
5
```

❖ اینجا دیگه نوع متد override شده درون کلاس فرزند با متد درون کلاس پدر یکسان می باشد.

- **سطح دسرسی یا Modifier** متد override شده در کلاس فرزند را نمی توانیم محدود تر از سطح دسرسی متد در کلاس پدر کنیم.
- **یعنی فرزند نمی تونه محدود تر از پدر باشه اما می تونه آزاد تر از پدر باشه ☺**

برای مثال اگر متد کلاس پدر ما از نوع public بود نمی توانیم در کلاس فرزند آن متد را از نوع private تعریف کنیم:

```
package javalike;

class B {

	public int calculator(int x, int y) {
		int sum = x + y;
		return sum;
	}
}

public class A extends B {
	private int calculator(int x, int y) {
		int subtract = x - y;
		return subtract;
	}

	public static void main(String[] args) {

		A a = new A();
```

```
		System.out.println(a.calculator(10, 5));
	}
}
```

خروجی:

خطای کامپایل!!!

❖ به دلیل این که سطح دسرسی متد درون کلاس پدر public بوده اما در کلاس فرزند سطح دسرسی را محدود کرده و private تعریف کرده ایم که خطای کامپایل داده.

نکته: برعکس مفهوم قبل یعنی اگر سطح دسرسی متد پدر محدودتر بود نسبت به متد override شده درون کلاس فرزند مشکلی ایجاد نمی شود:

```
package javalike;

class B {

	private int calculator(int x, int y) {
		int sum = x + y;
		return sum;
	}
}

public class A extends B {
	public int calculator(int x, int y) {
		int subtract = x - y;
		return subtract;
	}

	public static void main(String[] args) {

		A a = new A();

		System.out.println(a.calculator(10, 5));
	}
}
```

خروجی:

```
5
```

- ❖ چون متد override شده درون کلاس فرزند (A) نسبت به متد درون کلاس پدر (B) محدودتر نیست مشکلی ایجاد نمی شود.
- ❖ پس نتیجه میگیریم متد override شده درون کلاس فرزند نباید محدود تر از متد درون کلاس پدر باشد.
- ❖ اگه سطح دسرسی متد override شده درون کلاس فرزند با متد کلاس پدر یکسان باشد که دیگه خبری نیست و مشکلی به وجود نمی اید.
- ❖ در اینجا سطح دسرسی متد کلاس فرزند آزاد تر از سطح دسرسی متد پدر خودش می باشد.پس خبری نیست ☺

- تنها کلاسی که فرزند یا زیرکلاس ،کلاس پدر باشد و کلاس پدر خود را به ارث برده باشد می تواند متد کلاس پدر را override کند.
- اگر متد کلاس پدر از نوع final تعریف شده باشد نمی توانیم آن را override کنیم.
- اگر متد کلاس پدر از نوع static تعریف شده باشد،هنگام override کردن این متد در کلاس فرزند حتما باید آن را نیز از نوع static تعریف کنیم:

```
package javalike;

class B {

   static int calculator(int x, int y) {
         int sum = x + y;
         return sum;
   }
}

public class A extends B {
   static int calculator(int x, int y) {
         int subtract = x - y;
         return subtract;
   }

   public static void main(String[] args) {

         System.out.println(calculator(10, 5));
   }
}
```

خروجی:

```
5
```

❖ چون نوع متد override شده درون کلاس فرزند static تعریف شده با نوع متد درون کلاس پدر یکسان است برنامه صحیح است.

- اگر یک کلاس ،کلاس دیگر را به ارث نبرد خب دیگه نمی تونه متد اون کلاس رو override کند.
- سازنده ها (Constructors) نمی تواند override شوند.

پیروز و موفق باشید
