آموزش زبان برنامه نویسی جاوا

Encapsulation(کپسوله سازی)

جلسه بیست و ششم

نویسنده : رحمان زارعی

جاوا را ساده،آسان و شیرین بنوشید!!!!

Encapsulation (کپسوله سازی) یکی از چهار مفهوم اساسی برنامه نویسی شی گرایی است.سه مفهوم دیگر شامل وراثت،چندریختی و انتزاع می شود.

Encapsulation (کپسوله سازی) در جاوا فرآیند بسته بندی متغیر ها و متدها می باشد.

در Encapsulation (کپسوله سازی) متغیرهای نمونه یک کلاس از دید کلاس های دیگر پنهان می مانند.و این متغیرها تنها از طریق متدهای کلاس مون قابل دسرس هستند.بنابراین این متغیرها به متغیرهای مخفی شناخته شده هستند.

خب توضیحات خشک و خالی مون تمام شد مثلا میخواستیم بصورت رسمی ی توضیحاتی در مورد این مفهوم داده باشیم ☺

جناب Encapsulation (کپسوله سازی) کارش اینه که اطلاعات یک کلاس یعنی متغیرها و متدهای یک کلاس را طوری بسته بندی و باندپیچی میکنه و گره کوری می زنه بهش ☺ که هیچ کلاس دیگه نتونه اطلاعات کلاس مون رو دید بزنه ☺

خب کمی واضع تر توضیح میدی؟ وقتی شما یک کلاس تعریف کرده و تعدادی متغیر نمونه در بدنه ان تعریف میکنید بعد از شی سازی از کلاسون در کلاسی دیگر ،ما از طریق این شی ایجاد شده می توانیم به راحتی به متغیرهای کلاسمون دسرسی پیدا میکنیم.خب این حالت عادی و معمولی هست که با شی ایجاد شده از یک کلاس در کلاسی دیگر می توانستیم به راحتی به متغیرهای آن کلاس دسرسی پیدا کنیم.خب وقتی از مفهوم Encapsulation (کپسوله سازی) در کلاسمون استفاده کنیم فرآیندی اتفاق می افتد که تمام متغیرها و متدهای کلاسمون بسته بندی شده و دیگر کلاسی بیگانه نمی تواند به متغیرها یا متدهای کلاسمون حتی با ایجاد شی از نوع کلاسمون دسرسی پیدا کند.

مثال از دنیای واقعی:

متداول ترین مثالی که همه در آموزش Encapsulation (کپسوله سازی) در جاوا میزنند مثال داروی کپسول در پزشکی هست.

کپسول های دارویی همه ما با آن آشنا هستیم و حداقل یکبار هم که شده آنها را مصرف کرده ایم!

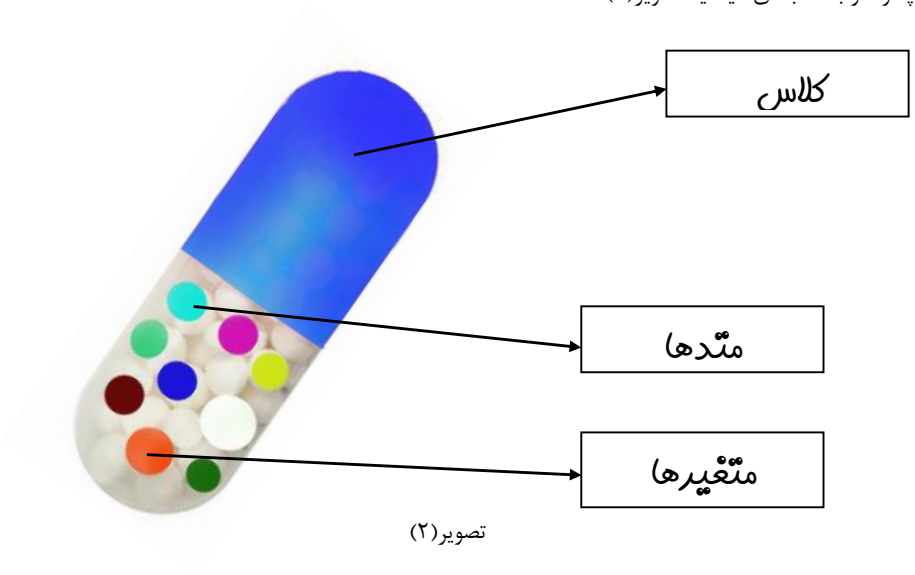
این کپسول هایی که ما برای درمان بیماری نظیر سرماخوردگی و... مصرف میکنیم ، ترکیب تعدادی داروی دیگر در یک بسته **واحد** به نام **کپسول** می باشد. مثال تصویر (۱)

تصویر(۱)

همان طور که در تصویر(۱) مشخص هست تعدادی داروی کوچک و ریز در یک بسته واحد به نام کپسول بسته بندی شده است.

این بسته بندی کپسولی مزیت هایی نظیر سهولت استفاده یعنی بجای مصرف تک به تک دانه های ریز دارو مستقیم یک دانه کپسول را می توانیم مصرف کنیم بدون این که محتوای کپسول را ببینیم. از دیگر مزیت های بسته بندی کپسولی امنیت ان هست شما مستقیم به محتوای درون کپسول دسرسی ندارید و فقط یک کپسول واحد را می بینید و مصرف میکنید اگر شما مستقیم به محتوای ریز درون کپسول دسرسی پیدا کنید و آن همه محتوای ریز را دست بگیرید و تک تک مصرف کنید امکان افتادن دانه ها روی زمین یا آلودگی دانه های دارو وجود دارد پس بهتر است همون یک دانه واحد کپسول را با یک آب بزنیم به رگ ☺

Encapsulation (کپسوله سازی) در جاوا نیز همین مفهوم بالا را دنبال میکند یعنی شما **متغیرها و متدهای** کلاس خودتون را در یک **بسته واحد** کپسوله و بسته بندی میکنید.تصویر(۲)



تصویر(۲)

تصویر(۲) شکل کپسوله سازی متغیرها و متدها ها را در یک کلاس مشاهده میکنید.متغیرها و متدها به دانه های ریز درون کپسول و کلاس به عنوان یک کپسول واحد تشبیه شده است. با این کار دیگر مستقیم نمی توانیم به متغیرها و متدهای کلاسمون در کلاسی دیگر دسرسی داشته باشیم.

پس تا اینجا مفهوم Encapsulation (کپسوله سازی) را یادگرفتیم حالا سراغ نکات پیاده سازی این مفهوم در جاوا میریم.

نحوه Encapsulation (کپسوله سازی) یک کلاس در جاوا:

- همه متغیرهای نمونه کلاس را از نوع `private` تعریف می کنیم.
- خب وقتی متغیرها را در کلاسمون از نوع `private` تعریف کردیم دیگه فرآیند بسته بندی انجام شده و کلاس های دیگر بصورت مستقیم نمی توانند به متغیرهای کلاسمون دسرسی پیدا کنند،خب حالا چطور کلاس های دیگه می توانند به متغیرهای کلاس ما دسرسی داشته باشند؟!!!

پاسخ: از طریق متدهای `public` (عمومی) `setter` (ست کننده ،مقدار دهی کننده) و `getter` (گیرنده،مقدار دهنده) می توان به متغیرها دسرسی پیدا کرد و مقدار آن ها را تغییر داد و مقدار های متغیرها را مشاهده کرد.خب ما تاحالا به متدهای `setter` و `getter` برخورد نداشته ایم حالا میخوایم این دو متد را بررسی کنیم:

متد setter :

همان طور که از اسم این نوع متدها مشخص هست برای مقداردهی ، ست کردن و نسبت دادن مقداری به یک متغیر استفاده می شود.

ما در گذشته برای مقداردهی کردن متغیری خاص نام ان را صدا میزدیم و مستقیم به آن مقداری را نسبت می دادیم.حالا برای مقداردهی کردن متغیری بصورت غیر مستقیم از متدهای setter استفاده می کنیم.

مثال زیر مقدار دهی به متغیرها در حالت عادی که در گذشته انجام میدادیم را نمایش میدهد:

```
package javalike_Encapsulation1;

public class A {

	public static void main(String[] args) {
		B b = new B();
		b.var1 = 5;
		b.var2 = "@javalike";
		b.var3 = 'U';
	}

}

class B {
	int var1;
	String var2;
	char var3;
}
```

[M] Commented [۱]: کلاس A و B درون این پکیج قرار دارند.

[M] Commented [۲]: تعریف کلاسی با نام A

[M] Commented [۳]: متد main که وظیفه اجرای برنامه را در کلاس A برعهده دارد

[M] Commented [۴]: ایجاد شی از کلاس B درون کلاس A

[M] Commented [۵]: از طریق شی ایجاد شده از نوع کلاس B ،متغیرهای کلاس B را مقداردهی کرده ایم.این نوع مقداردهی که در حالت عادی و درگذشته انجام میدادیم.

[M] Commented [۶]: تعریف کلاسی با نام B

[M] Commented [۷]: تعریف متغیر در بدنه کلاس B

حال در مثال زیر مقداردهی به متغیرها از طریق متد setter را بررسی میکنیم:

برای خواندن توضیحات از پایین یعنی کلاس B شروع کنید و بیایید بالا سراغ کلاس A

لازم به ذکر است این مثال ها تا اینجا ربطی به مفهوم کپسوله سازی نداره تنها داریم متد setter و getter را بررسی میکنیم.

```
package javalike_Encapsulation1;

public class A {

	public static void main(String[] args) {
		B b = new B();
```

```
            b.setVar1(5);
            b.setVar2("@javalike");
            b.setVar3('U');
      }

}

class B {
      int var1;
      String var2;
      char var3;

      public void setVar1(int var1) {
            this.var1 = var1;
      }

      public void setVar2(String var2) {
            this.var2 = var2;
      }

      public void setVar3(char var3) {
            this.var3 = var3;
      }
}
```

- همه متدهای setter از نوع void هستند.
- بهتر و زیباتر است که کنترل سطح دسرسی متدهای setter را از نوع public تعریف کنیم.
- نام انتخابی برای متد setter دلخواه است اما بهتر و زیباتر است که نام انتخابی بصورت ترکیب زیر باشد:

set + variableName

نام بعد از کلمه set باید با حرف بزرگ شروع شود البته زیباتر است ☺

set + نام متغیر

- پارامتر متد را همنام و از نوع متغیری که قرار است براش متد setter پیاده سازی کنیم قرار میدهیم.مثلا اگر نام متغیر نمونه کلاس age بود و نوعش از نوع int متد setter آن تا اینجا بصورت زیر است:

```
public void setAge(int age)
```

[Commented [M٨]: متدهای setter روش نوشتن و تعریف کردنشون شبیه متدهای عادی هستش با این تفاوت که تنها کاربرد آنها برای ست کردن و مقداری دهی به متغیر ها استفاده می شود.
در نظر داشته باشید که برای هر متغیر یک متد setter باید تعریف شود یعنی اگر سه متغیر داشتیم سه متد setter باید تعریف کنیم.
اگر متغیر ما از نوع عدد صحیح بود باید متد setter آن نیز یک پارمتر از نوع عدد صحیح داشته باشد که این پارامتر باید درون متغیر نمونه ای که قصد داشتیم براش متد setter تعریف کنیم ریخته شود.(متد setVar1)
اگر متغیر ما از نوع رشته بود باید متد setter آن نیز یک پارمتر از نوع رشته داشته باشد که این پارامتر باید درون متغیر نمونه ای که قصد داشتیم براش متد setter تعریف کنیم ریخته شود.(متدsetVar2)
پس اگر متغیر ما که خواستیم براش متد setter تعریف کنیم از هر نوعی بود باید متد setter ان یک پارامتر از نوع ان متغیر داشته و پارامتر را درون آن متغیر که قرار بود براش متد setter تعریف کنیم ریخته شود.

- حالا میریم سراغ بدنه متد setter ،بدنه متد setter هر متغیر یک دستور یک خطی ثابت دارد!! و اون هم ریختن پارامتر متد setter درون متغیر نمونه ای که قرار هست براش متد setter تعریف کنیم:فرض کنیم همان متغیر age را میخواهیم در بدنه متد setter مقداردهی کنیم بصورت زیر است:

```
public void setAge(int age){
this.age = age;
}
```

دلیل استفاده از کلمه کلیدی this این هست که متغیر نمونه و متغیر محلی(پارامتر) متد همنام هستند.با کلمه کلیدی this بین متغیر نمونه و محلی تمایز قائل می شویم.

- خب تا این جا با روش ایجاد یک متد setter بصورت کلی اشنا شدیم.
- حالا روش ایجاد متد setter یکی از متغیر های مثال بالا را بررسی میکنیم:
- روش ایجاد متد setter برای متغیر var1 :

1. public

```
public
```

2. void

```
public void
```

3. set

```
public void set
```

4. var1(نام متغیر)

```
public void setVar1
```

5. (int var1)(پارامتری همنام با متغیر مورد نظر)

```
public void setVar1(int var1)
```

6. {}

```
public void setVar1(int var1){
}
```

7. this.var2 = var2;(نسبت دادن پارامتر متد به متغیر نمونه مورد نظر)

```
public void setVar2(String var2) {
      this.var2 = var2;
}
```

دلیل این که نام متد setter را ترکیبی از کلمه **set** و **نام متغیر مورد نظر** قرار میدهیم:

۱) دلیل آوردن کلمه **set** در ابتدای نام متد مشخص کنیم که این متد یک متد setter هستش و کارش نسبت دادن و مقداردهی کردن متغیر هست.

۲) دلیل استفاده از نام متغیر بعد از کلمه setter این هست که مشخص کنیم این متد setter متعلق به متغیر مربوطه می باشد و برای مقداردهی به این متغیر استفاده می شود.

دوستان ما درحال حاضر فقط داریم مفهوم متد setter و getter را بررسی میکنیم،بعد از پایان مفاهیم این دو متد ، سراغ کاربرد متد setter و getter در Encapsulation (کپسوله سازی) می رویم.

متد getter:

همان طور که از این متد مشخص هست ، این متد مقدار متغیر مورد نظر را به ما برمی گرداند یا می دهد.

خب در جلسات گذشته برای گرفتن و دریافت مقدار متغیر نام آن متغیر را صدا می زدیم و از مقدارش استفاده میکردیم.در مثال زیر به روش عادی مقدار ی متغیرها را می گیریم و مقدار آن را در خروجی چاپ می کنیم:

```
package javalike_Encapsulation1;

public class A {

	public static void main(String[] args) {
		B b = new B();
		System.out.println(b.var);
		System.out.println(b.var2);
		System.out.println(b.var3);

	}
}
class B {
	int var = 10;
	String var2 = "@javalike";
	char var3 = 'R';

}
```

[M Commented [۹]: روش معمول برای صدا زدن و گرفتن مقدار متغیر
با صدا زدن به این روش مقدار متغیر به ما برگردانده می شود و حالا در این جا در خروجی این مقادیر چاپ کرده ایم.

[M Commented [۱۰]: تعریف متغیرها و مقداردهی اولیه به آنها

خروجی:

```
10
@javalike
R
```

حالا گرفتن و دریافت مقدار متغیر را میخواهیم به روش متد getter انجام دهیم:

ابتدا توضیحات کلاس B و سپس توضیحات کلاس A را بخوانید.

```
package javalike_Encapsulation1;

public class A {

	public static void main(String[] args) {
		B b = new B();
		System.out.println(b.getVar());
		System.out.println(b.getVar2());
		System.out.println(b.getVar3());

	}

}

class B {
	int var = 10;
	String var2 = "@javalike";
	char var3 = 'R';

	public int getVar() {
		return var;
	}

	public String getVar2() {
		return var2;
	}

	public char getVar3() {
		return var3;
	}

}
```

خروجی:

```
10
@javalike
R
```

Commented [M1]: در اینجا با شی ایجاد شده از کلاس B متدهای های getter هر یک از متغیرها را صدا زده و مقادیر برگردانده شده را در خروجی چاپ کرده ایم.

Commented [M2]: تعریف متد getter برای متغیر var

روش تعریف متد getter برای این متغیر یا هر متغیر دیگه (روش ایجاد متد getter برای همه متغیرها یکی و ثابت است):

ابتدا کنترل سطح دسرسی متد را public انتخاب کرده بعدش نوع متد ،همان طور که از قبل گفتیم متدهای getter مقداری را به ما برمیگردانند یا پس میدهند،خب حالا با توجه به نوع متغیر مربوطه نوع متد را برای برگردادن مقدار متغیر تعیین میکنیم که در اینجا چون متغیر var از نوع عدد صحیح بود نوع int را برای این متد انتخاب میکنیم.

بعد از تعیین نوع میریم سراغ نام متغیر:

نام متغیر ترکیب کلمه get و نام متغیر مربوطه که با حرف بزرگ شروع می شود(البته انتخاب نام دلخواه است اما زیباتر و بهتر است برای خوانا بودن کد برنامه و مشخص بودن این که این متد مربوط به کدام متغیر هستش از اینگونه نام را انتخاب کنیم).

متدهای getter پارامتری به عنوان ورودی ندارند چون وظیفه آنها تنها برگردادن مقدار متغیر است نه مقدار دهی به متغیر.

بعدش دو بلوک باز و بسته میکنیم{}

در پایان در بدنه متد مقدار متغیر (که همنام با متغیر نمونه مربوطه هستش) را با کلمه return برمیگردانیم.

این توضیح آموزشی که برای این متد دادم برای سایر متدها برقرار است.تنها نوع متد با توجه به نوع متغیر مورد نظر تغییر میکند.

روش ایجاد متد getter برای یک متغیر بصورت کلی:

۱) کلمه کلیدی public را ابتدای متد تعریف می کنیم.

۲) بعد از کلمه کلیدی public نوع متد که برابر نوع متغیر مربوطه می باشد.

۳) نام متد که ترکیبی از کلمات زیر هستش:

نام متغیر مربوطه که با حرف بزرگ شروع می شود + get

۴) استفاده از پرانتز بدون پارامتر جلوی متد

۵) دو بلوک باز و بسته می کنیم {}

۶) در بدنه متد مقدار متغیر را بصورت زیر برمی گردانیم:

Return نام متغیر نمونه مربوطه ;

فرض کنید متغیر نمونه کلاس ما از نوع عدد صحیح (int) و با نام age هستش.حال قصد داریم متد getter را برای این متغیر تعریف کنیم:

1) Public

```
public
```

2) int

```
public int
```

3) getAge

```
public int getAge
```

4) ()

```
public int getAge()
```

5) {}

```
public int getAge(){
        }
```

6) return age;

```
public int getAge(){
return age
}
```

با روش ایجاد متد getter و setter اشنا شدیم که این دو متد به ترتیب برای برگرداندن و مقداردهی به متغیرهامون ازش استفاده می کردیم.

کاربرد متد getter و setter در Encapsulation (کپسوله سازی):

ما همان طور که از قبل گفتیم بعد از کپسوله سازی و private کردن متغیر های نمونه کلاسمون ،کلاس های دیگر نمی توانند به متغیرهای کلاس ما به روش معمول دسرسی داشته باشند.خب راه دسرسی به این متغیرهای private شده کلاسمون در کلاس دیگر چگونه است؟تنها راه دسرسی به متغیرهای private یک کلاس در کلاس دیگر استفاده از متدهای getter وsetter هستش.یعنی با متدهای setter متغیر private را می توانیم صدا زده و مقداردهی کنیم و با متد getter می توانیم متغیر را صدا زدن و مقدارش را دریافت کنیم.در مثال قبل در کلاس A با شی سازی از کلاس B توانستیم به روش عادی به متغیر های کلاس B دسرسی پیدا کنیم چون متغیرهای کلاس B ،private نبودند فرقی نداشت که بصورت عادی یا با متدهای getter و setter متغیرهای کلاس B را صدا بزنیم.

خب گام به گام با مفهوم Encapsulation (کپسوله سازی):

به مثال زیر توجه کنید:

```
package javalike_Encapsulation1;

public class A {

	public static void main(String[] args) {
		B b = new B();
		System.out.println(b.var);
		System.out.println(b.var2);
		System.out.println(b.var3);

	}
}

class B {
	int var = 10;
	String var2 = "@javalike";
	char var3 = 'R';

}
```

در این مثال متغیرهای کلاس B ، private نیستند.و از نوع پیشفرض یا default هستند.پس به روش عادی با شی ایجاد شده از کلاس B توانستیم این متغیرها را صدا زده و مقادیرانها راچاپ کنیم.

در تصویر(۳) محدوده دسرسی شی b که از نوع کلاس B هستش را مشاهده میکنید که هیچ مانعی برای دسرسی به متغیرهای کلاس B وجود ندارد.

```
        B b = new B();
        System.out.println(b.var);
        System.out.println(b.
        System.out.println(b.

    }

}

class B {
    int var = 10;
    String var2 = "@javalike"
    char var3 = 'R';

}
```

var : int - B
var2 : String - B
var3 : char - B
equals(Object arg0) : boolean - Object
hashCode() : int - Object
toString() : String - Object
getClass() : Class<?> - Object
notify() : void - Object
notifyAll() : void - Object
wait() : void - Object
wait(long arg0) : void - Object
wait(long arg0, int arg1) : void - Object

تصویر(۳)

همان طور که در تصویر(۳) مشاهده میکنید شی b به تمام متغیرهای کلاس B دسرسی دارند.

حالا تمام متغیرهای کلاس B را private تعریف می کنیم، مثال زیر:

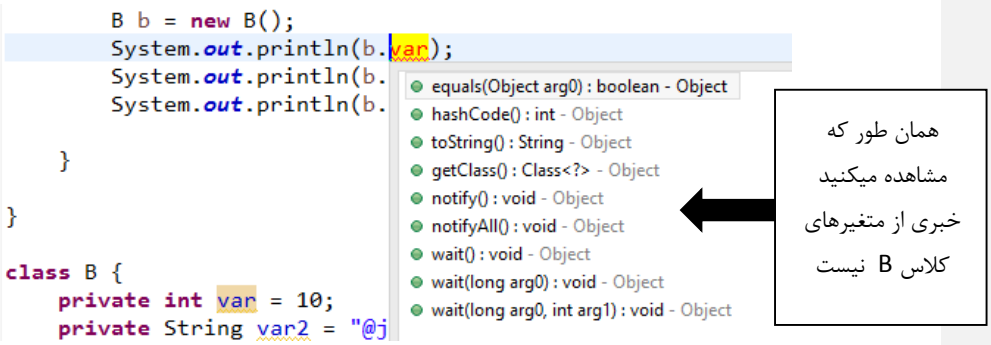
```
package javalike_Encapsulation1;

public class A {

      public static void main(String[] args) {
            B b = new B();
            System.out.println(b.var);
            System.out.println(b.var2);
            System.out.println(b.var3);

      }

}
```

```
class B {
	private int var = 10;
	private String var2 = "@javalike";
	private char var3 = 'R';

}
```

خروجی:

خطای کامپایل!!!

دلیل خطای کامپایل در مثال بالا این هست که متغیرهای کلاس B همگی private هستند و در کلاس دیگر نمی توانیم در حالت عادی به این متغیرها دسترسی پیدا کنیم.در تصویر(۴) عدم توانایی دسرسی شی b به متغیرهای کلاس B را مشاهده میکنید:



تصویر(۴)

لازم به ذکر است متغیرهای private یک کلاس تنها برای کلاس های دیگر در حالت عادی در دسرس نیستند و در داخل خود کلاس قابل دسرس هستند و مشکلی پیش نمی اید، مثال زیر :

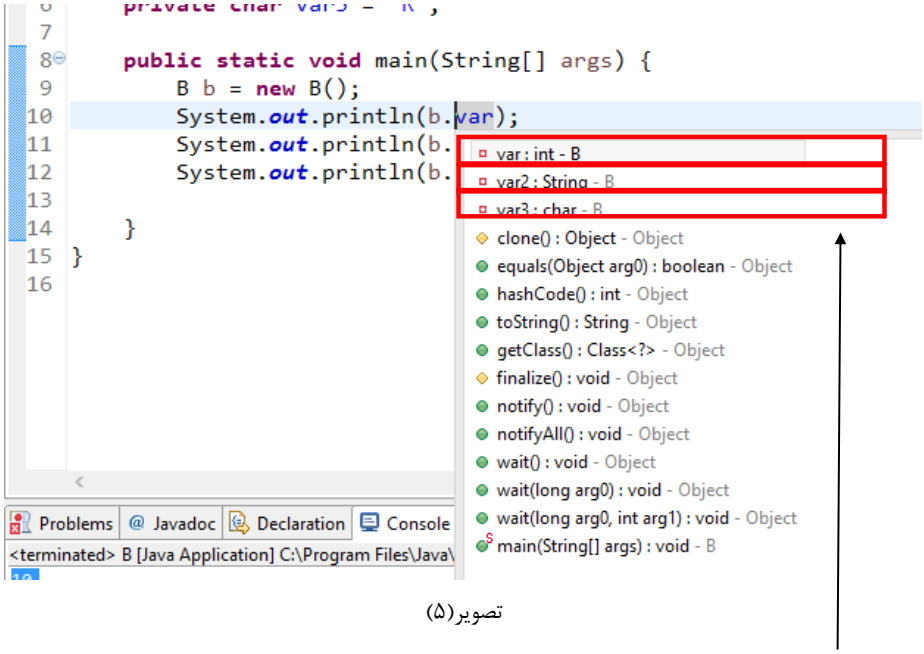
```
package javalike_Encapsulation1;

public class B {
	private int var = 10;
	private String var2 = "@javalike";
	private char var3 = 'R';

	public static void main(String[] args) {
		B b = new B();
```

```
            System.out.println(b.var);
            System.out.println(b.var2);
            System.out.println(b.var3);

        }
}
```

خروجی:

```
10
@javalike
R
```

- همان طور که گفتیم متغیرهای `private` یک کلاس در داخل خود کلاس در دسرس هستند و تنها در کلاس های دیگر غیرقابل دسرس می باشند.دسرسی شی `b` که از نوع کلاس `B` هستش را در تصویر(۵) ببینید:



تصویر(۵)

همان طور که در تصویر(۵) مشاهده میکنید شی `b` که در داخل کلاس `B` ایجاد شده به متغیرهای `private` کلاس `B` دسرسی دارد.مربع قرمز ریز کنار متغیرها علامت `private` بودن متغیر است.

حال در مثال زیر Encapsulation (کپسوله سازی) را در کلاس B رعایت کرده و تمام متغیرهای کلاس B را private و با تعریف متدهای public ، getter و setter برای هر متغیر راه دسرسی به متغیرهای private کلاس B را برای کلاس های دیگر فراهم می اوریم:

در مثال زیر به دلیل private بودن متغیرهای کلاس B از طریق متدهای setter و getter در کلاس A متغیرها را مقداردهی کرده و مقدار آنها را دریافت کرده و در خروجی چاپ کرده ایم.

ابتدا توضیحات کلاس B و سپس توضیحات کلاس A را ببینید.

```
package javalike_Encapsulation1;

public class A {

	public static void main(String[] args) {
		B b = new B();

		//Initialized instance variable
		b.setVar(50);
		b.setVar2("@javalike");
		b.setVar3('z');
		//Return the value of instance variable
		System.out.println(b.getVar());
		System.out.println(b.getVar2());
		System.out.println(b.getVar3());

	}

}

class B {
//============================================================================
	//create methods getter and setter for Instance variable class B
	public int getVar() {
		return var;
	}

	public void setVar(int var) {
		this.var = var;
	}

	public String getVar2() {
		return var2;
	}
```

[M] Commented [۱]۳: مقداردهی و ست کردن متغیرهای کلاس B با متدهای setter هر متغیر

[M] Commented [۱]٤: گرفتن و دریافت کردن مقدار متغیرهای کلاس B با متدهای getter هر متغیر و در پایان چاپ مقادیر انها

```
    public void setVar2(String var2) {
        this.var2 = var2;
    }

    public char getVar3() {
        return var3;
    }

    public void setVar3(char var3) {
        this.var3 = var3;
    }
//================================================
    //The definition of class instance variables
    private int var ;
    private String var2 ;
    private char var3;

}
```

[M]Commented [l٥]: پیاده سازی متدهای getter و setter متغیرهای private کلاس B

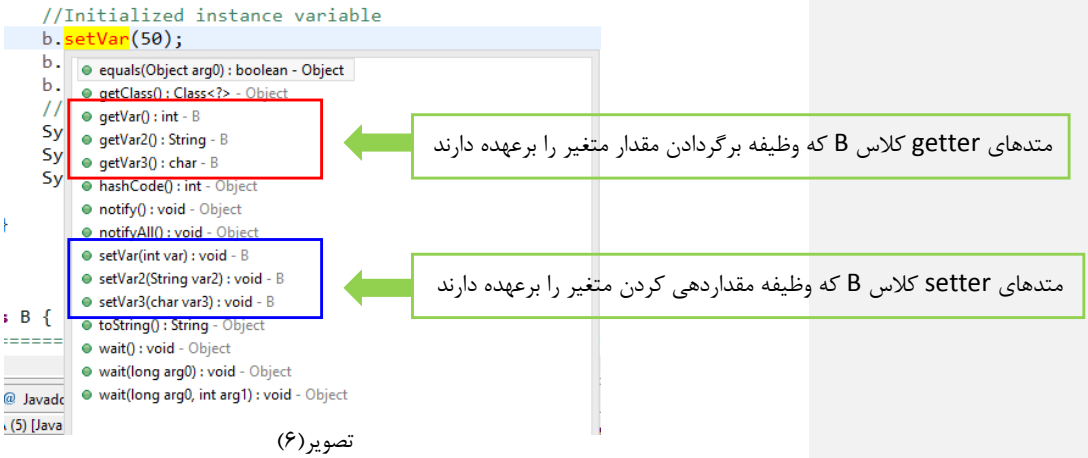
در اینجا برای هر متغیر یک متد getter و یک متد setter تعریف کرده ایم.

[M]Commented [l٦]: تعریف متغیرهای از نوع private در کلاس B

خروجی:

```
50
@javalike
z
```

همان طور که در مثال بالا مشاهده کردید با وجود **private** بودن متغیرهای کلاس **B** با متدهای **setter** و **getter** آن توانستیم به متغیرهای **private** دسترسی پیدا کنیم.در تصویر(۶) دسترسی شی **b** باز طریق متدهای **setter**و **getter** به متغیرهای **private** کلاس **B** را نشان میدهد:



تصویر(۶)

خب مفاهیم متدهای setter و getter و Encapsulation (کپسوله سازی) که یکی از مفاهیم مهم شی گرایی است را یادگرفتیم.

خلاصه:

Encapsulation (کپسوله سازی) در جاوا یعنی:

۱) تمامی متغیرهای نمونه کلاس باید **private** اعلام شوند.
۲) برای هر متغیر نمونه متدهای **getter** و **setter** پیاده سازی شود.
۳) تنها راه دسرسی به متغیرهای نمونه یک کلاس در کلاس دیگر از طریق متدهای **getter** و **setter** انجام می شود.

مزیت های Encapsulation (کپسوله سازی):

۱) متغیرها و ویژگی های یک کلاس می توانند فقط خواندنی (**read-only**) یا فقط قابل نوشتن (**write-only**) باشند.خب این مفهوم یعنی چه؟!! یعنی این که ما می توانیم سطح دسرسی به متغیرهای کلاسمون را برای استفاده سایر کلاس ها تعیین کنیم یعنی بگیم ای کلاس فلان تنها می توانی مقدار متغیرهای کلاس مرا دریافت کنی ، بخوانی، ببینی و بگیری و استفاده کنی اما نمی توانی مقدار متغیرها را دستکاری ،مقداردهی و ست کنی(**read-only**).و بلعکس تنها می توانیم متغیرها رو مقداردهی و ست کنیم و اما نمی توانی مقدارآنها را دریافت کنی،بگیری و استفاده کنی(**write-only**.)

در مثال زیر برنامه طوری طراحی شده که متغیرهای کلاس B فقط خواندنی هست و هرگز نمی توانیم مقدار متغیرهای نمونه کلاس B را در کلاسی دیگر تغییر دهیم.

```
package javalike_Encapsulation1;

public class A {

  public static void main(String[] args) {
        B b = new B();

        // Return the value of instance variable
        System.out.println(b.getVar());
        System.out.println(b.getVar2());
        System.out.println(b.getVar3());

  }
```

```
}

class B {
  //
================================================================================
  // create methods getter and setter for Instance variable class B
  public int getVar() {
        return var;
  }

  public String getVar2() {
        return var2;
  }

  public char getVar3() {
        return var3;
  }

  // ================================================
  // The definition of class instance variables
  private int var = 256;
  private String var2 = "java tutorial in @javalike";
  private char var3 = 'N';

}
```

۲) **کنترل بر متغیرهای کلاس**، یعنی شما می توانید بر مقدار متغیرهای کلاس خود توسط کلاس دیگر نظارت داشته باشید.مثلا شما یک کلاس اتومبیل دارید و اتومبیل یک ویژگی به نام سرعت دارد خب برای کنترل متغیر سرعت اتومبیل باید از متد `setter` استفاده کنید با این تفاوت که بگید اگر سرعت بیشتر ۲۰۰ شد دیگر حق افزایش سرعت را نداری!!!

```
package javalike_Encapsulation1;

class Car {
  private int speed;

  public int getSpeed() {
        return speed;
  }

  public void setSpeed(int speed) {
        if (speed <= 90)
              this.speed = speed;
```

```
        else
              System.out.println("woooo  Risk of dying :-(");
  }
}

public class Test {
  public static void main(String[] args) {
        Car pride =new Car();
        pride.setSpeed(140);
  }
}
```

خروجی:

```
woooo  Risk of dying :-(
```

نکته: ما اجازه داریم با توجه به نیاز بدنه متد `setter` و `getter` متغیرهامون رو تغییردهیم.

در کل `Encapsulation` (کپسوله سازی) کاربردهای زیادی مخصوص در زمینه امنیت برنامه دارد.

مثال از دنیای واقعی

- شما وقتی در اتومبیل شخصی خود نشسته و پدال گاز را فشار می دهید تنها در یک محدوده ی خاصی سرعت زیاد می شود و اجازه افزایش سرعت دلخواه خارج محدوده به شما داده نمی شود.
- یا این که شما هنگام روشن و خاموش کردن لامپ اتاق بجای اینکه مستقیم سیم برق را به سیم لامپ متصل کنید تنها با زدن دکمه پریز لامپ خاموش یا روشن می شود.
- یک نمونه دیگه هم دستگاه های خودپرداز بانک ها اگر در دستگاه های خودپرداز کپسوله سازی را رعایت نکنند یکی که میخواد بره پول بگیره و حسابش خالی هست و اعصاب نداره میزنه دستگاه رو خاموش میکنه یا تعداد موجودی دستگاه را کم و زیاد میکنه و.....☺

حرف آخر ☺

شاید پیش خودتون بگید شاید ما ۵۰ متغیر یا بیشتر داشتیم حالا باید بیایم برای هر متغیر یک متد `setter` و `getter` پیاده سازی کنیم این که خیلی خسته کننده است ☹ راه کار چیست؟!!!!!!

پاسخ،درسته اما فکر اینجاشم کردند! در ایکلیپس شما به راحتی می توانید بدون نوشتن متدهای `setter` و `getter` بصورت دستی برای متغیرهای کلاستون تنها با چندکلیک ساده متد `setter` و `getter` آنها را پیاده سازی کنید.

راهنمای تصویری ایجاد متد getter و setter برای متغیرهای کلاس:

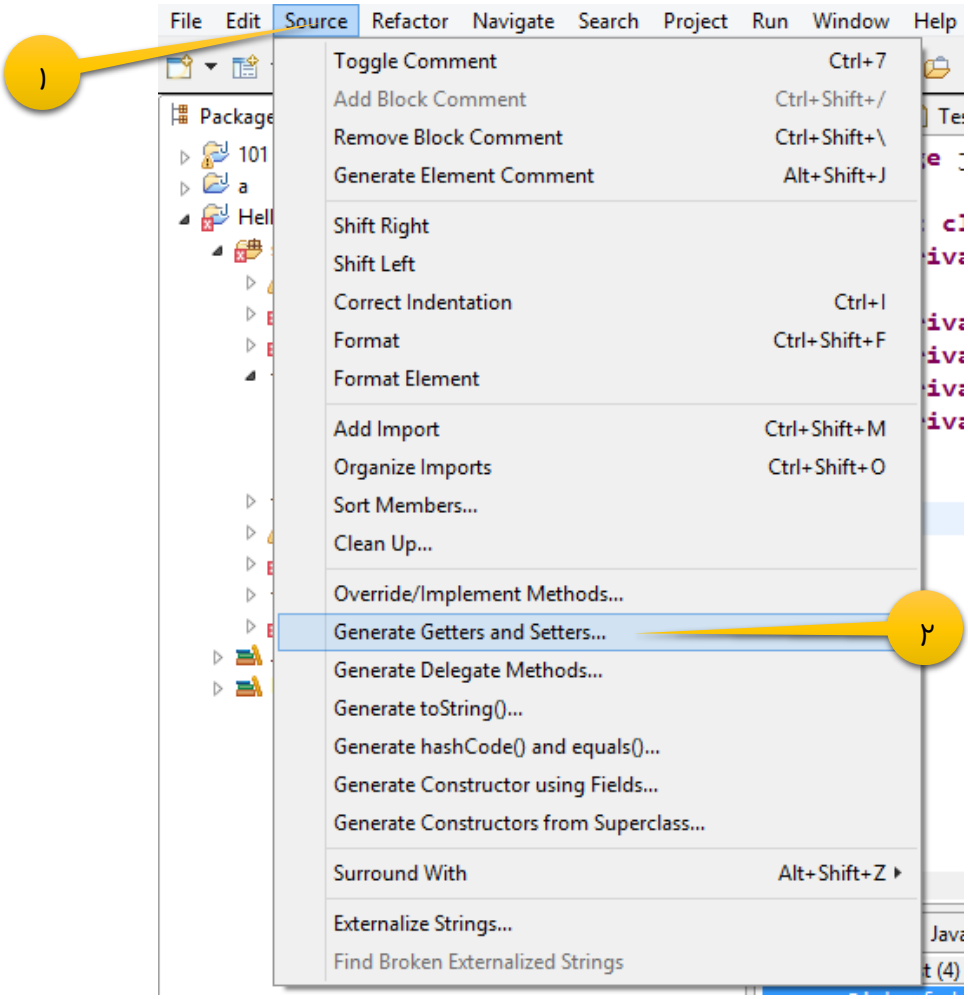
[M]Commented [V]: ما متد setter را برای متغیر private ،speed پیاده سازی کرده ایم.
سری محدودیت برای متغیر speed در متد setter تعیین کرده ایم
گفته ایم تنها حق داری با سرعت کمتر از ۹۰ برانی
در غیر این صورت خطر مرگ وجود دارد و اجازه افزایش سرعت بالای ۹۰ را نداری
پس ما میتوانیم بدنه متد getter یا setter خود را با توجه به نیاز تغییر دهیم.

فرض کنید کلاس Animal با ویژگی های زیر را در محیط Eclipse ایجاد کرده ایم:

```
package javalike_Encapsulation1;

public class Animal {
	private String name;

	private int age;
	private String color;
	private char gender;
	private String id;

}
```

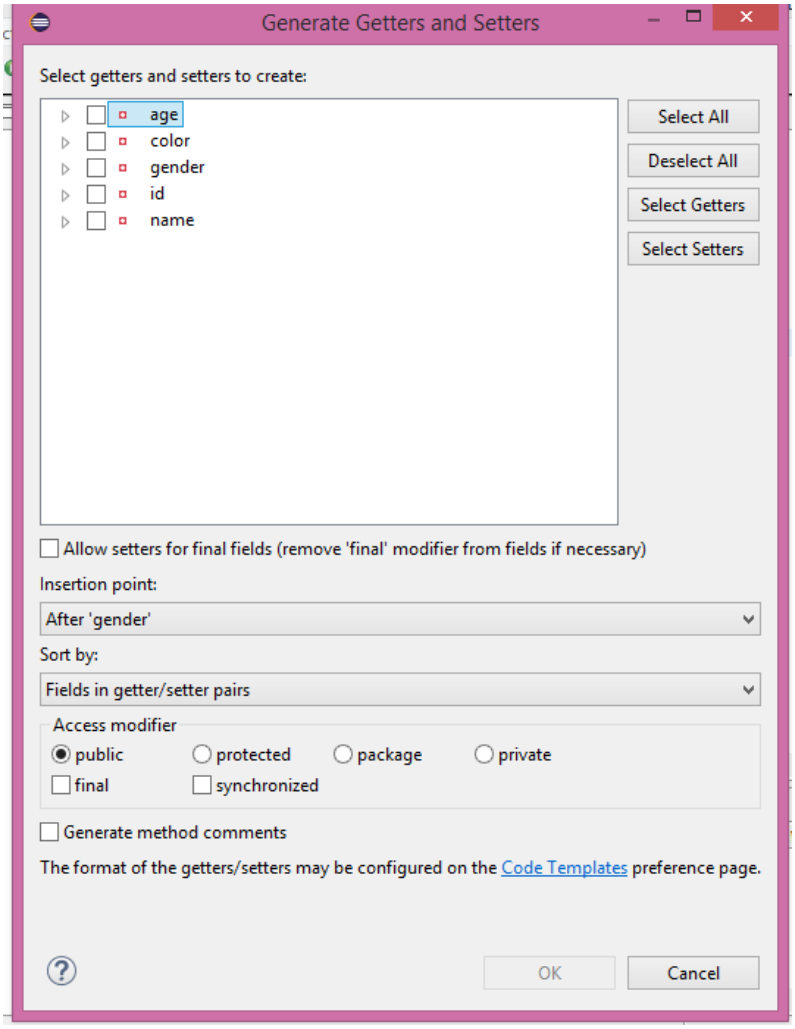
حالا قصد داریم متدهای getter و setter این کلاس را پیاده سازی کنیم،برای این کار بصورت زیر عمل میکنیم:

() در محیط Eclipse از منوبار ، منو سوم یعنی Sourceرا انتخاب میکنیم،بعدش از پین زیر منو باز شده منو Source گزینه ...Genarate Getter and Setters را انتخاب میکنیم.تصویر(۷)



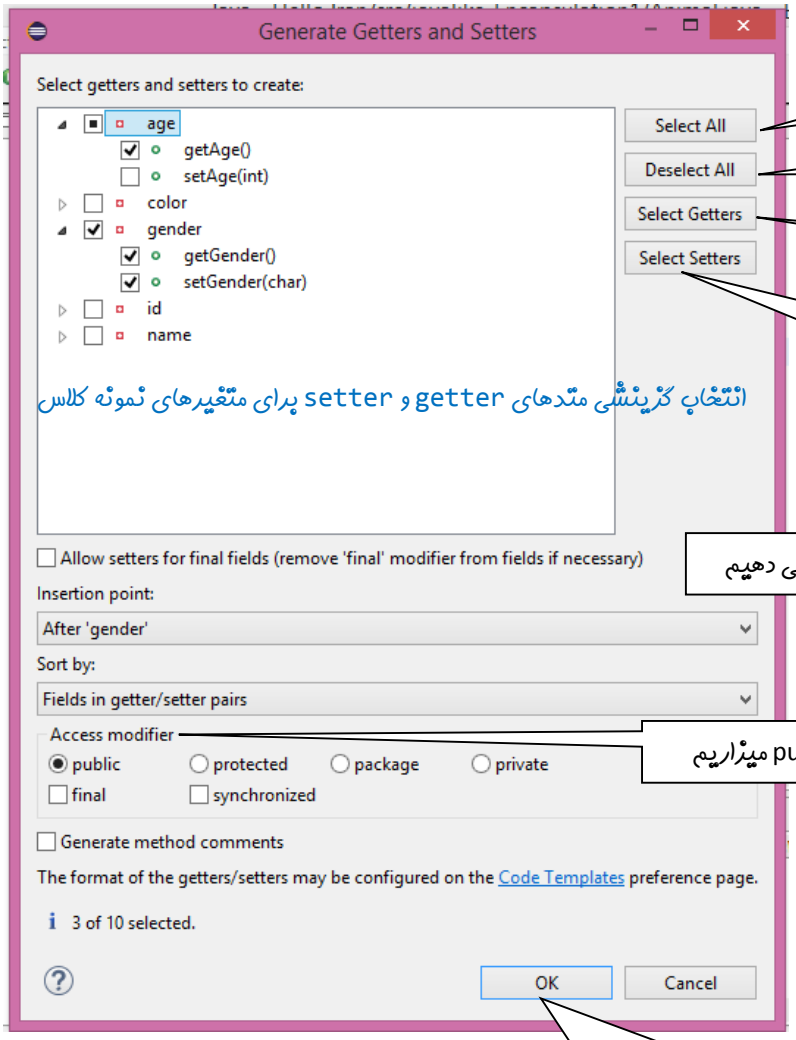
تصویر(۷)

۲) بعد از باز شدن Genarate Getter and Setters با تصویر(۸) برخورد میکنیم



تصویر(۸)

۳)در بخش Select getters and setters to create می توانید باتوجه به نیاز متدهای getter و setter را برای متغیر خاصی تیک بزنید تصویر(۹) یا این که با زدن دکمه Select All برای همه متغیرها این دو متد را پیاده سازی کنید.تصویر(۱۰)



با زدن این دکمه همه گزینه ها انتخاب می شود(تصویر(۱۰)

با زدن این دکمه همه گزینه ها از حالت انتخاب خارج می شوند

با زدن این دکمه فقط متدهای getter متغیر ها انتخاب می شوند

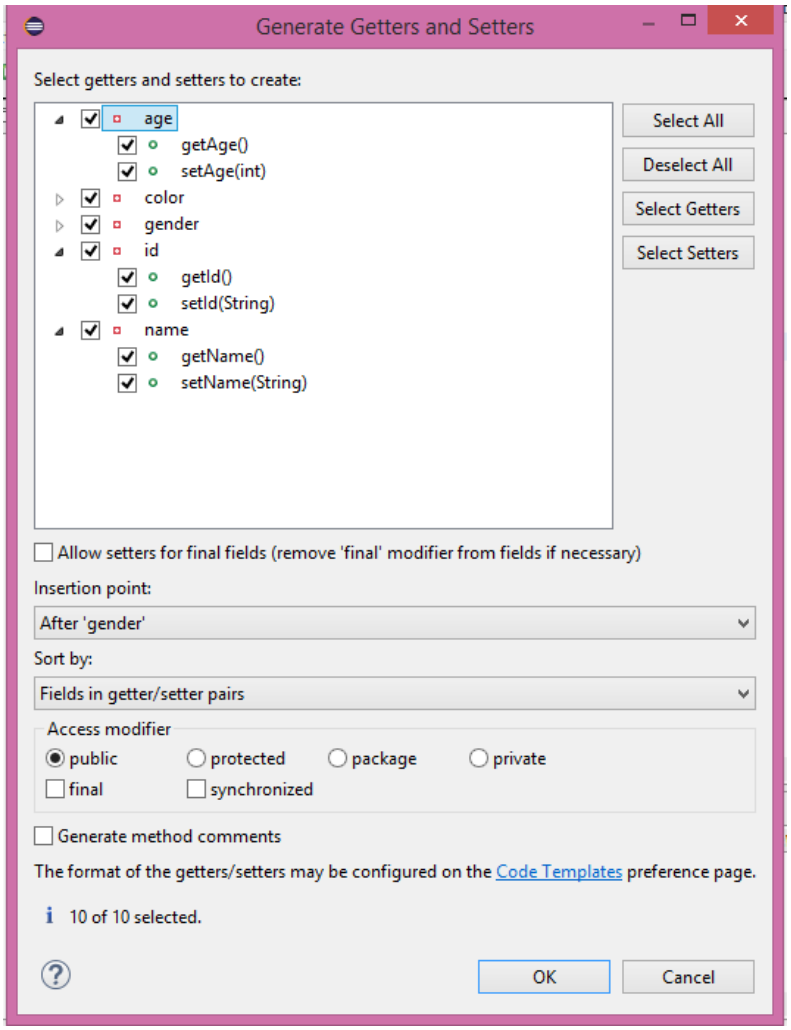
با زدن این دکمه فقط متدهای setter متغیر ها انتخاب می شوند

سایر گزینه ها را به حالت پیشفرض خود قرار می دهیم

کنترل سطح دسرسی متد نیز همون public میزاریم

در پایان با زدن دکمه ok تمامی متدهایی که تیک زده ایم پیاده سازی می شوید

تصویر(۹)



تصویر(۱۰)

فرض کنید تمام گزینه های متدهای `setter` و `getter` را برای متغیرهای نمونه کلاسمون را انتخاب کرده و دکمه ok را زده ایم.تصویر(۱۱):

```
package javalike_Encapsulation1;

public class Animal {
	private String name;

	private int age;
	private String color;
	private char gender;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getColor() {
		return color;
	}

	public void setColor(String color) {
		this.color = color;
	}

	public char getGender() {
		return gender;
	}

	public void setGender(char gender) {
		this.gender = gender;
	}

	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}

	private String id;

}
```

```java
package javalike_Encapsulation1;

public class Animal {
    private String name;

    private int age;
    private String color;
    private char gender;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getColor() {
        return color;
    }
    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public char getGender() {
        return gender;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    private String id;
```

تصویر(1)

در این جلسه سعی شد با مثال از دنیای واقعی و مثال های جاوا مفهوم Encapsulation (کپسوله سازی) را به سادگی برای شما روشن کنیم.

پیروز و موفق باشید
