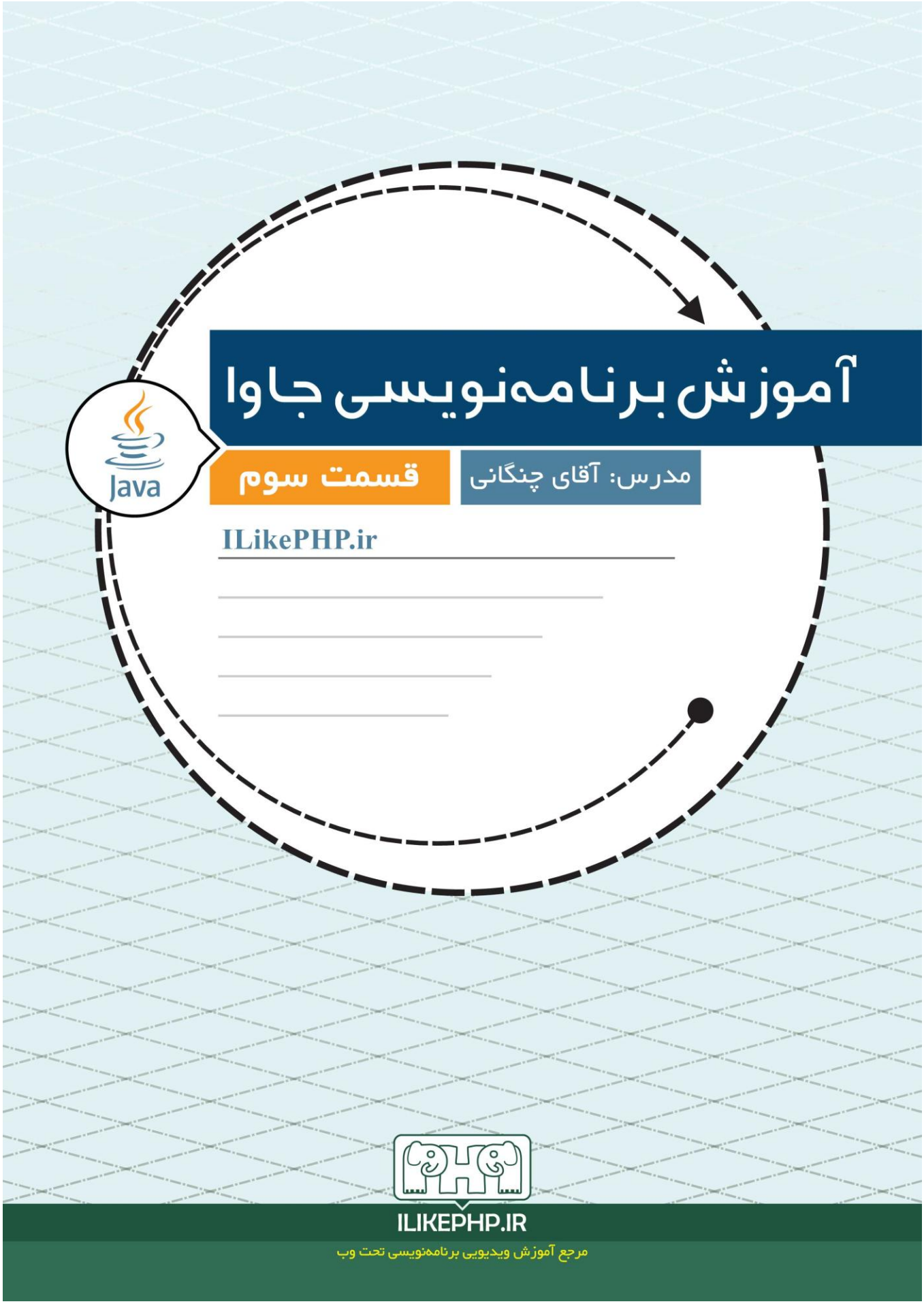
آموزش برنامه‌نویسی جاوا
مدرس: آقای چنگانی
قسمت سوم
ILikePHP.ir
Java
ILIKEPHP.IR
مرجع آموزش ویدیویی برنامه‌نویسی تحت وب

با سلام

چنگانی هستم .با قسمت سوم آموزش جاوا در خدمت شما هستم

اگه سوالی از جلسات قبل هست بنده در خدمتم.

جلسه قبلی دیدیم چطور میشه یک کلاس تعریف کنیم و برای ان یک سری ویژگی و توانایی تعریف کنیم و اینکه از این موجودیت نمونه بسازیم.

توی این جلسه در مورد کپسوله‌سازی موجودیت‌ها صحبت می‌کنیم.

جلسه قبل موجودیت Student رو به این صورت نوشتیم:

Main.java | Main2.java | Student.java | Main3.java

```
1   /**
2    * Created by Mohammad Changani on 9/10/15.
3    */
4   public class Student {
5       int id;
6       String name;
7       String family;
8       String username;
9       String password;
10
11      public void printName(){
12          System.out.println(name + " " + family);
13      }
14  }
15
```

در برنامه‌نویسی شی‌گرا قانوی وجود دارد با این مفهوم که همه چیز باید کپسوله و مخفی شود مگه ایمکه خلافش ثابت شود.

یعنی هر جزیی باید مخفی و سطح دسترسی نداشته باشد مگه اینکه واقعا لازم داشته باشیم.

خوب قبل از اینکه در مورد این قانون حرف بزنیم اول در مورد سطح دسترسی‌ها صحبت کنیم.

هر field یا method در داخل کلاس خود همیشه در دسترس هست. و اعضای یک کلاس به همه field یا method های کلاس خود بدون هیچ محدودیتی دسترسی دارند.

مثلا در عکس قبل داخل متد printName شما می‌تونید هرکدوم از fieldها را فراخوانی کنید و استفاده کنید.

ی public و private دو مورد از سطح دسترسی‌ها هستن که این جلسه در موردشون صحبت می‌کنیم.

به طور کلی اگر یک field یا method به صورت public تعریف شود به این معنی هست که دسترسی به ان از بیرون هم ممکن هست. یعنی اگر من داخل کلاس Student مقدار id را به این صورت تعریف کرده باشم:

```
public int id;
```

می‌تونیم به ویژگی id به این صورت دسترسی داشته باشم:

```
Student ali = new Student;()

ali.id = 1;
```

خوب ببینیم که اگه یک field یا method به صورت private تعریف شود چه اتفاقی می‌افتد. مثلا ما ویژگی name موجودیت Student رو به صورت private مینویسیم:

```
private String name;
```

وقتی یک field یا method به صورت private تعریف میشه در واقع دسترسی اون رو ما محدود کردیم به اجزای داخلی اون کلاس و از بیرون قابل دسترسی نیست یعنی نمیتونیم به این صورت بنویسیم:

```
Student ali = new Student;()

ali.name = “ali;“
```

سطح دسترسی‌های دیگه هم وجود دارد که در جلسات بعدی درموردشون صحبت میکنیم.

برگردیم سراغ قانون که اول جلسه گفتیم در مورد اینکه همه چیز باید مخفی شود پس طبق این قانون ما باید همه field یا method ها را به صورت private تعریف کنیم.

پس به این صورت میشه:

Main.java | Main2.java | Student.java | Main3.java

```java
/**
 * Created by Mohammad Changani on 9/10/15.
 */
public class Student {
    private int id;
    private String name;
    private String family;
    private String username;
    private String password;

    private void printName(){
        System.out.println(name + " " + family);
    }
}
```

خوب ولی همون جور که میدونید ما متد printName() رو برای این نوشتیم که بتونیم اسم و فامیل را برای یک نمونه چاپ گند و لازم هست از بیرون کلاس هم فراخوانی شود پس نمی‌تواند private باشد پس باید تبدیل به public شود.

در واقع نوع سطح دسترسی method ها بر میگرده به منطق برنامه که می‌نویسیم. شاید method وجود داشته باشد که لازم نباشد شما public تعریف کنید.

ولی در مورد field ها همه باید private باشند!! خوب الان دوتا سوال پیش میاد اینکه چرا اصلا باید private باشه؟! و دوم اینکه چطوری پس بهشون مقدار بدیم یا مقدارشون رو بخونیم.

سوال اول رو در موردش فکر کنید آخر جلسه در موردش حرف میزنیم ولی در مورد سوال بعدی اینکه چون field ها به صورت private هستند پس لازم هست برای اون ها یک متد نوشته شود که مقدارشون رو بخونیم و یک متد برای تغییر مقدار اون ها داشته باشیم به این صورت.

Main.java | Main2.java | Student.java | Main3.java

```
/**
 * Created by Mohammad Changani on 9/10/15.
 */
public class Student {
    private int id;
    private String name;
    private String family;
    private String username;
    private String password;

    public void printName(){
        System.out.println(name + " " + family);
    }

    public void a(int i){
        id = i;
    }

    public int b(){
        return id;
    }
}
```

خوب اینجوری میتونیم از بیرون کلاس با متد a مقداری را برای id مشخص کنیم (set کنیم) و با متد b مقدار id را بخونیم (get کنیم).

Student ali = new Student;()

ali.a(1;(

int i = ali.b;()

پس تا اینجا مشکل سطح دسترسی رو حل کردیم ولی همونجور که میبینید از اسم‌های خوبی برای این متدها استفاده نکردیم. و خیلی متد a و b گویا نیستند! خوب بهترین کار این هست که اسم این متدها رو هم استاندارد بنویسیم. پس ما از الان استاندارد میکنیم برای خودمون که متدهای که قرار یه مقداری رو برای یه field مشخص کنه رو با اسم set بنویسیم و برای متدهای که قرار یک field رو بخونه از کلمه get استفاده کنیم.

شما میتونید هر اسمی بگذارین ولی بهتره همیشه استانداردها رو رعایت کنید!

پس به این صورت شد:

Main.java | Main2.java | Student.java | Main3.java

```
1  /**
2   * Created by Mohammad Changani on 9/10/15.
3   */
4  public class Student {
5      private int id;
6      private String name;
7      private String family;
8      private String username;
9      private String password;
10
11     public void printName(){
12         System.out.println(name + " " + family);
13     }
14
15     public void setId(int i){
16         id = i;
17     }
18
19     public int getId(){
20         return id;
21     }
22 }
23
```

خوب بیایم یکم استانداردتر بکنیم! بهتر آرگومان ورودی متد setId رو هم از اسم معنا داری استفاده کنیم!

```
public void setId(int id}(

this.id = id;

{
```

خوب چون هم اسم آرگومان ورودی ما id هست و هم یک field به نام id داریم، برای اینکه بتونیم تمایزی بین اون‌ها ایجاد کنیم هر موقع به فیلد ها نیاز داشته باشیم از کلمه this استفاده میکنیم. در واقع this.id به متغییر id که به صورت private در بالا تعریف کردیم اشاره میکنه.

خوب الان برای همه field ها باید این get و set ها رو بنویسیم به همین صورت ولی خوب میتونید کارتون رو ساده تر کنید و رو صفحه کلیک راست کنید و از گزینه Generate گزینه getter and Setter رو انتخاب کنید و همه field ها رو انتخاب کنید و ok رو بزنید. خودش همه رو کامل میکنه!

Main.java | Main2.java | Student.java | Main3.java

```java
public class Student {
    private int id;
    private String name;
    private String family;
    private String username;
    private String password;

    public void printName(){
        System.out.println(name + " " + family);
    }

    public int getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public String getFamily() {
        return family;
    }

    public void setFamily(String family) {
        this.family = family;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

```

خوب اگه تا ایجا سوالی هست بنده در خدمتم اگه نه بریم بخش دوم آموزش

خوب برای اتمام این بحث لازم هست در مورد مفهوم به نام سازنده یا constructor صحبت کنیم. constructor کاربردهای زیادی دار ولی دوتا از کاربردهاش خیلی خیلی زیاد لازم میشه.

خوب اول بگیم که سازنده چی هست.

شما هر بار یک نمونه از یک کلاس میسازین (وقتی new می‌کنید) در واقع اولین اتفاقی که می‌افتد این هست که قسمتی از حافظه برای اون نمونه اختصاص داده می‌شود و سازنده اون کلاس فراخوانی می‌شود. پس هر نمونه به ازای هر بار new شدن سازنده ان در ابتدا صدا زده میشود.

سازنده هر کلاس در واقع یک متد هست که برخلاف همه متدهای موجود هیچ خروجی ندارد حتی void! و هم نام با اسم کلاس! مثلا برای کلاس Student به این صورت میشه:

```
public Student}()

{
```

یکی از کاربردهای سازنده این هست که اگه بخواهیم در زمان ساخته شدن یک نمونه یک فرایند همیشه اجرا شود ان را در سازنده می‌نویسیم. مثلا برای کلاس Student من می‌خوام هربار که نمونه‌ای ساخته میشه یک متن ابتدا چاپ شود و مقدار name برای همه مقدار "ali" مشخص شود.

```
public Student}()

 System.out.println(“create new student;(“
```

name = “ali;“

{

خوب یعنی شما هر نمونه‌ای بسازید به ازای هر نمونه یکبار عبارت ”create new student“ چاپ می‌شود و اسم همه نمونه ها ”ali“ می‌شود!!!!!

خوب کاربرد دوم این هست که شما می‌خواهین این محدودیت رو وضع کنید که هر نمونه‌ای که قرار از کلاس Student ساخته شود حتما id و name آن‌ها را در زمان ساخته شدن وارد شود! در واقع این محدوید این کمک رو میکنه که تضمین می‌کنه هیچ نمونه‌ای وجود نخواهد داشت که این دو ویژگی رو نداشته باشند!

خوب پس به این صورت میشه:

public Student(int id, String name){

System.out.println(“create new student;(“

this.id = id;

this.name = name;

{

خوب برای نمونه ساختن هم دیگه نمی‌تونیم به این صورت نوشت:

Student std1 = new Student;()

چون ما الان سازنده‌ای داریم که حتما باید مقدار id و name را بگیرد:

```
Student std1 = new Student(1, “ali;(“

System.out.println(std1.getId;(()

System.out.println(std1.getName;(()
```

خوب نکته بعدی این هست که شما می‌تونید مقدار id یا name رو هم عوض کنید:

```
Student std1 = new Student(1, “ali;(“

System.out.println(std1.getId;(()

System.out.println(std1.getName;(()

std1.setId(2;(

System.out.println(std1.getId;(()
```

نکته بعدی این که شما می‌تونید چند سازنده هم زمان داشته باشید:

```
public Student(int id, String name}(

 System.out.println(“create new student;(“

this.id = id;
```

```
 this.name = name;

{
```

```
public Student(int id, String name, String family}(

 System.out.println(“create new student;(“

this.id = id;

 this.name = name;

this.family = family;

{
```

و به این صورت هم ازشون استفاده کنید:

```
Student std1 = new Student(1, “ali;(“

Student std2 = new Student(2, “mohammad”, “ch;(“
```

آخرین نکته این جلسه اینکه شما میتونید برای این‌که حجم کدنویستون پایین بیاد و یک فراید را چندین بار ننویسید میتونید یک سازنده را بر اساس سازنده دیگه بسازید (روش استاندارد برای چند سازنده همزمان)

```
public Student(int id, String name}(

 System.out.println(“create new student;(“
```

```
this.id = id;
 this.name = name;
{
public Student(int id, String name, String family}(
 this(id, name;(
this.family = family;
{
```

این کد در واقع هیچ تفاوتی با کد بالا ندارد فقط استانداردتر هست و کدنویسی کمتری دارد و این به معنی این هست خطای کدنویسی کمتری دارید.

فقط چند نکته اینکه this() در واقع به یک سازنده از همون کلاس اشاره می‌کنه و در واقع داره اون سازنده رو مقدار دهی می‌کند پس ما اول داریم مقدار id و name را مشخص میکنیم و بعد هم مقدار family را.

و مقداردهی سازنده توسط this() حتما باید در اولین خط سازنده انجام شود!

قواعد نگارشی:

۱- نام field ها همیشه باید اسم باشند و نه فعل!

۲- نام method ها همیشه باید فعل باشند!

۳- تا میشه اسم‌هاتون رو با معنی و چند کلمه‌ای استفاده کنید حتی اگه لازم هست یک خط هم بشه اشکالی نداره!

۴- اگه از کلمه‌ای مخفف در نام‌گذاری استفاده می‌کنید هم هرو با حروف بزرگ بنویسید:

```
exportHtmlSource;()

// NOT: exportHTMLSource;()
```