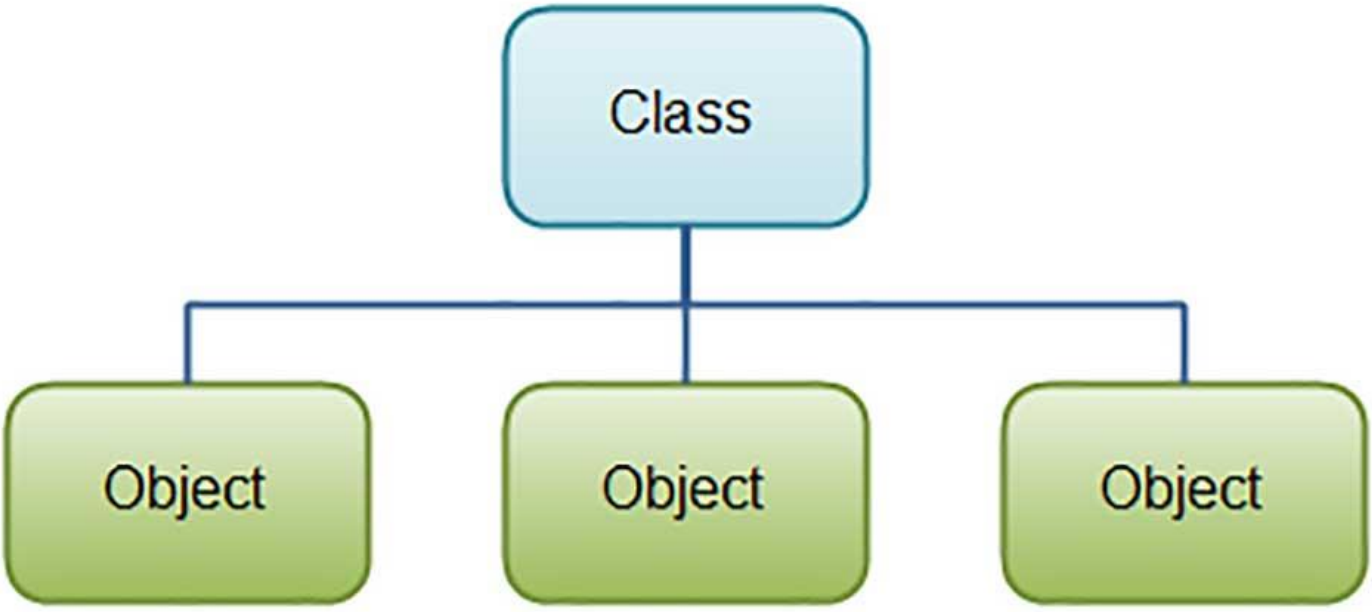
جلسه چهارم شی و کلاس جاوا



جاوا یک زبان شی گرا میباشد. برای اینکه بتوانید به زبان جاوا برنامه نویسی بکنید باید حتما با مفاهیم شی گرایی آشنا باشید. کلاس ها و اشیاء دو قسمت پایه ای و بسیار مهم در این نوع از برنامه نویسی میباشند. در این جلسه از **دوره آموزش جاوا** میخواهیم با شی و کلاس جاوا آشنا بشویم. با **برنامه چی** برای آشنایی با کلاس ها و اشیاء در جاوا همراه باشید.

دیدن این مقاله توی سایت

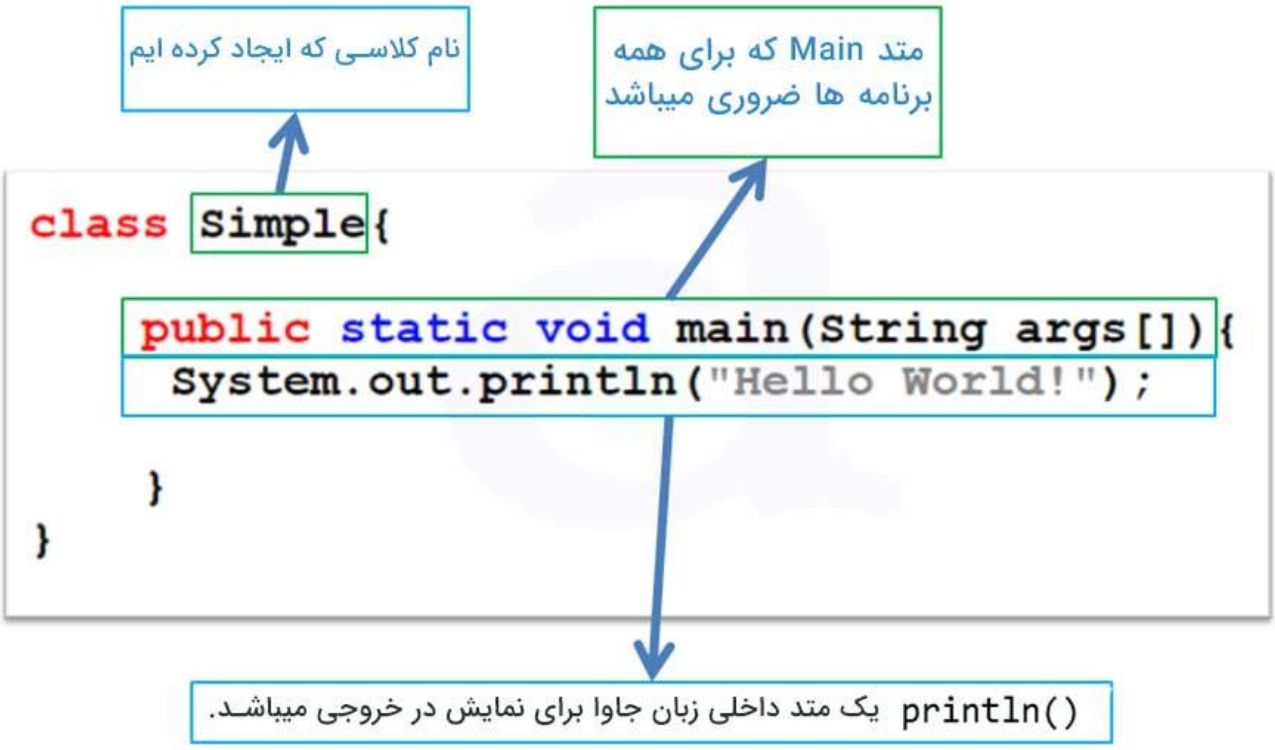
جاوا یک زبان شی گرا یا به اصطلاح Object Oriented است. به عنوان زبانی که ویژگی های شی گرایی را درون خود دارد، جاوا میتواند از خصوصیت های زیر پشتیبانی بکند:

- چندریختی (Polymorphism)
- ارث بری (Inheritance)
- کپسوله سازی (Encapsulation)
- انتزاع (Abstraction)
- کلاس ها (Classes)
- اشیاء (Objects)
- نمونه (Instance)

- متد (Method)
- تجزیه و تحلیل پیام (Message Parsing)

در این جلسه، ما میخواهیم در باره مفاهیم شی و کلاس جاوا، با زبان ساده صحبت کنیم.

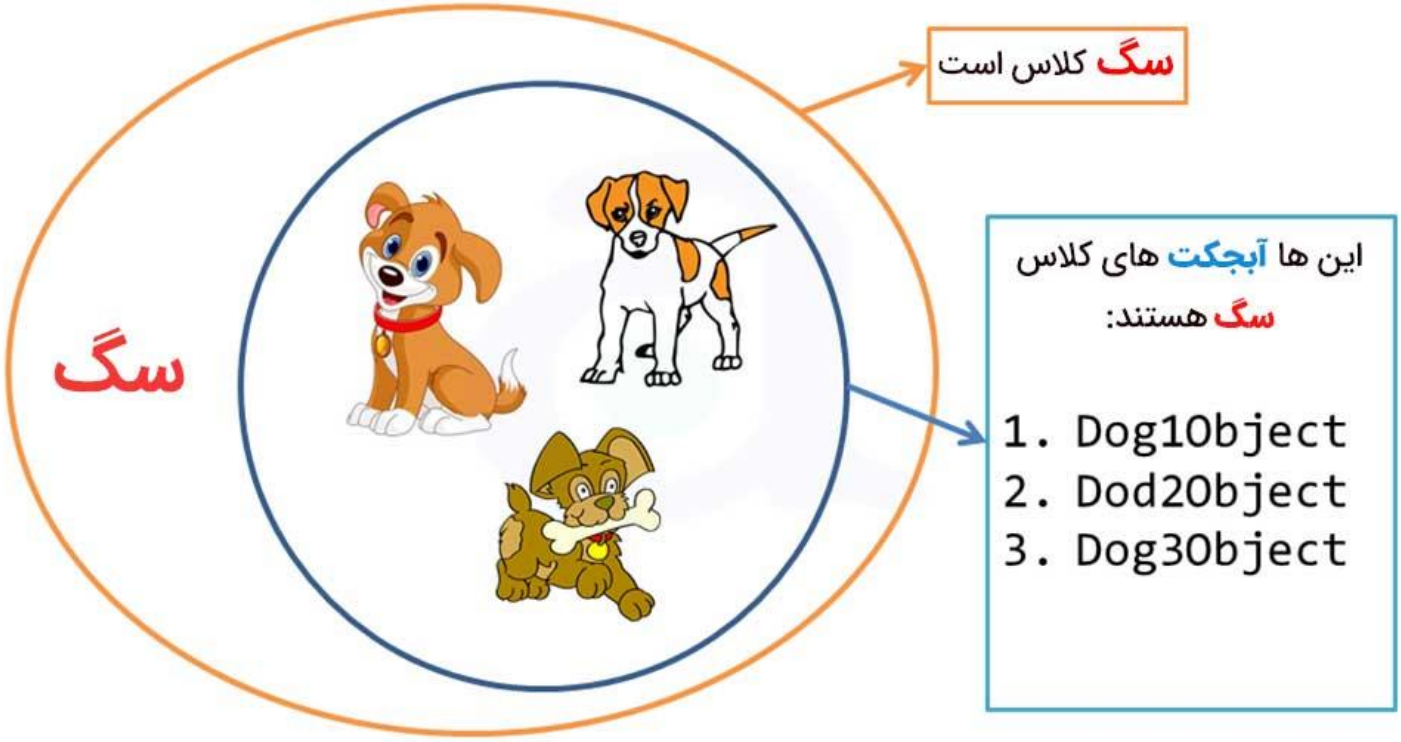
شی (Object)



اشیاء دارای دو قسمت به نام ویژگی و رفتار هستند. مثال: یک حیوان مانند سگ را در نظر بگیرید. سگ دارای یک سری ویژگی ها مانند رنگ مو، قد، وزن، نوع صدا و غیره ا ست. همین سگ علاوه بر ویژگی های خود، دارای رفتار هایی نیز میباشد. مانند تکان دادن دم، بازی کردن، دویدن، پریدن و غیره. ویژگی های و رفتار های Object ها هم دقیقا مانند همین است. Object ها از روی کلاس ساخته میشوند.

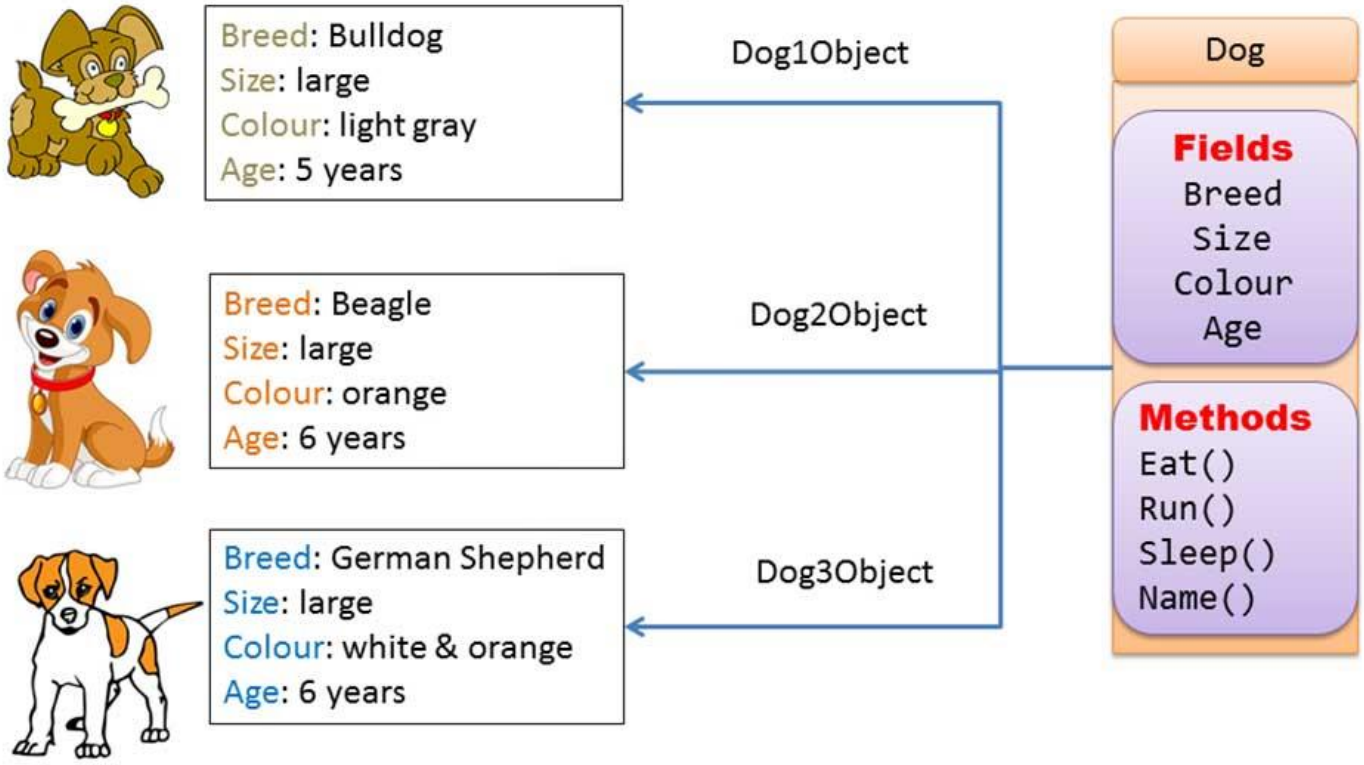
کلاس جاوا (Java Class)

کلاس جاوا را میتوانید به عنوان یک الگو یا شابلون برای تعیین کردن ویژگی ها و رفتار اشیائی که از روی آنها ساخته میشود، در نظر بگیرید.



کلاس یک طرح است که Object های منحصر بفرد از روی آن ساخته میشوند. کلاس مجموعه ای از کد ها میبا شد که درون فایل هایی نگه داری میشوند که نام این فایل ها دقیقا با نام کلاس درون آن فایل باید یکسان با شد. درون این کد ها متد هایی تعریف میشوند که میتوانند در بقیه قسمت های کد ما در دسترس با شند و فراخوانی شوند. در ابتدا باید بدانید که هر Object میتواند دو قسمت "ویژگی" و "رفتار" داشته باشد.

همانطور که توضیح داده شد کلاس جاوا مانند یک الگو است که اشیاء از روی آنها ساخته میشوند. در این رابطه به کلاسی که شی از روی آن ساخته میشود، کلاس پدر یا کلاس والد گفته میشود. ویژگی های اشیاء همین متغیرهایی هستند که درون کلاس پدر قرار دارند. رفتار شی را هم متد هایی که درون کلاس والد قرار دارند تشکیل میدهد.



پس یعنی کلاس هایی داریم که متد ها و متغیر ها درون آن قرار دارند. اگر یک شـــی از این کلاس جاوا بـ سازیم، متغیر های کلاس ویژگی های Object و متد های کلاس، رفتار Object ما را مشخص میکنند.

در زیر یک نمونه از کلاس آورده شده است:

```
public class Dog {
String breed;
int age;
String color;
void barking() {
}
void hungry() {
}
void sleeping() {
}
}
```

یک کلاس جاوا میتواند هرکدام از انواع متغیر های زیر را داشته باشد:

- **متغیر محلی (Local Variables):** متغیر هایی که داخل متد ها تعریف میشـوند را متغیر محلی یا Local Variables میگویند. این متغیر ها باید درون متد ها تعریف شـــده و مقدار دهی اولیه بشـــوند. هنگامی هم که متد کار خودش را انجام بدهد، این متغیر ها از بین میروند و دیگر در حافظه باقی نمی مانند. اگر دوباره در برنامه ما این متد فراخوانی شـــده باشـــد، متغیر ها دوباره تعریف شده و مقدار دهی اولیه میشوند.

> باید این را در نظر داشـــته باشـــید که متغیر های محلی فقط در همان متدی که تعریف میشوند، قابل دسترسی هستند. یعنی خارج از آن متد، نمیتوانید از این متغیر ها استفاده کنید. چون بیرون از آن متد، اصلا انگار که این متغیر ها را اصلا تعریف نکرده اید!

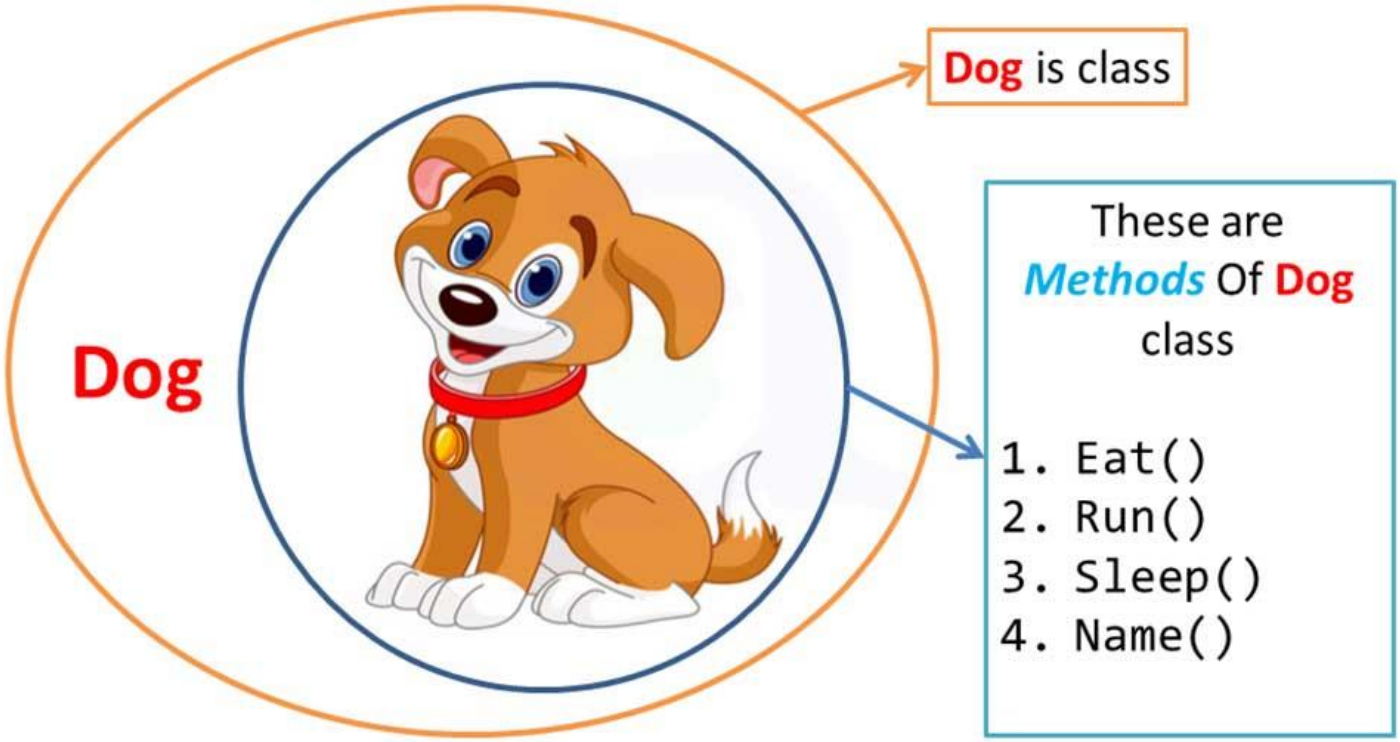
- **متغیر های نمونه (Instance Variables):** متغیر های نمونه یا Instance Variable ها متغیر هایی هستند که درون کلاس قرار دارند، اما داخل هیچ متدی قرار نگرفته اند. این متغیر ها وقتی مقدار دهی اولیه میشوند که از روی کلاس یک نمونه (Instance) یا شی ساخته میشود. متغیر های نمونه داخل همه متد ها، متد ســـازنده (Constructor) آن کلاس و بلوک های آن قابل دسترسی هستند.
- **متغیر های کلاس (Class Variables):** متغیر های کلاس، متغیر هایی هستند که درون کلاس و خارج از همه متد ها تعریف میشـوند. فرق متغیر های نمونه و متغیر های کلاس این اســت که متغیر های کلاس دارای کلیدواژه Static (استاتیک) میباشند.

کلاس ها میتوانند هر تعداد از متدها را درون خود داشته باشند. یعنی میتوانند انواع مختلف متد ها را درون خود جا بدهند. در مثال بالا ()barking و ()hungry و ()sleeping متد هســـتند. در ادامه این مقاله مو ضوعات و مواردی که برای یادگرفتن مبحث کلاس ها در زبان جاوا ضروری هستند را آموزش میدهیم. با برنامه چی همراه باشید.

نکته: یکی از تفاوت های متد با کلاس جاوا این اســـت که اولین کلمه در نام متد ها، با حروف کوچک شـــروع میشـــود. اما اولین کلمه در نام کلاس ها با حروف بزرگ شـــروع میشـــود. مثلا myFirstProject اسم متد و MyFirstProject نام یک کلاس میباشد.

متد سازنده (Constructor Method)

هرجا در مورد کلاس جاوا بحث میشود، متد سازنده (Constructor) یکی از سرفصل های مهم است. هر کلاس جاوا حتما باید یک متد Constructor یا همان متد سازنده داشته باشد. اگر ما صریحا یک متد سازنده را برای کلاس نسازیم، کامپایلر جاوا بصورت پیشفرض یک متد سازنده از پیش تعیین شده را برای کلاس ما میسازد.



متد سازنده یا متد Constructor یک متد است که داخل کلاس قرار میگیرد و نام آن با نام کلاس یکسان است (فقط اولین کلمه آن با حرف کوچک شروع میشود). یعنی اگر یک کلاس به نام MyApplication ساخته شید، نام متد سازنده باید myApplication() شد. اگر این متد را خودتان بنویسید، دیگر متد پیشفرض کامپایلر برای شما اجرا نمیشود. استفاده های متد سازنده را در ادامه کار کردن با جاوا یاد خواهید گرفت.

هر زمان که یک شی جدید از روی یک کلاس جاوا ساخته میشود، حداقل یک متد Constructor ساخته میشود. قانون اصلی متدهای سازنده این است که باید همنام با کلاس خود شان باشند. یک کلاس میتواند بیشتر از یک متد سازنده داشته باشد.

در زیر یک نمونه از متد Constructor را مشاهده میکنید.

```
public class Puppy {
public Puppy() {
}

public Puppy(String name) {
// This constructor has one parameter, name.
}
}
```

جاوا همچنین از الگوی طراحی Singleton هم پشـــتیبانی میکند که در آن میتوانید فقط یک نمونه از کلاس داشته باشید. (در مورد الگوهای طراحی به زودی به صورت مفصل در وبسایت برنامه چی مقاله هایی قرار میدهیم).

نکته: دو نوع از متد های سـازنده در جاوا وجود دارند که هر دوی این مدل ها را در قسـمت های بعدی این دوره آموزشی با جزئیات کامل بررسی خواهیم کرد.

ساختن یک شی (Object)

همانطور که ذکر شد، یک کلاس جاوا الگویی برای ساختن اشیاء میباشد. پس به زبان ساده تر، یک شی از روی یک کلاس جاوا سـاخته میشـود. در زبان جاوا، از کلمه کلیدی new برای سـاختن Object های جدید استفاده میکنیم.

برای ساختن یک Object باید سه مرحله زیر را طی کنید:

1. **معرفی کردن:** یک متغیر را به همراه نام متغیر معرفی میکنیم.
2. **نمونه سازی:** از کلیدواژه new برای ساختن یک Object جدید استفاده میکنیم.
3. **مقداردهی اولیه (Initialize):** بعد از کلید واژه new، متد Constructor کلاس مورد نظر را صـدا میزنیم. این صـدا زدن باعث میشـود که Object ما مقدار دهی شـود و خالی یا اصـطلاحا null نباشد.

در ادامه مثالی از ساختن یک Object برای شما قرار داده شده است.

```
public class Puppy {
public Puppy(String name) {
// This constructor has one parameter, name.
System.out.println("Passed Name is :" + name );
}
public static void main(String []args) {
// Following statement would create an object myPuppy
Puppy myPuppy = new Puppy( "tommy" );
}
}
```

اگر این کد ها را بنویسیم سپس کامپایل و اجرا کنیم، خروجی زیر نمایش داده میشود.

خروجی

```
Passed Name is :tommy
```

دسترسی به متغیر های نمونه و متد ها

متغیر ها نمونه و متد ها از طریق Object ها قابل دسترسی هستند. برای دسترسی به متغیر های نمونه (Instance Variable) ها، روش زیر را دنبال کنید:

```
/* First create an object */
ObjectReference = new Constructor();
/* Now call a variable as follows */
ObjectReference.variableName;
/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

مثال

این مثال به خوبی نشان میدهد که چگونه به متد ها و متغیر های نمونه دسترسی داشته باشید.

```
public class Puppy {
int puppyAge;

public Puppy(String name) {
// This constructor has one parameter, name.
System.out.println("Name chosen is :" + name );
}

public void setAge( int age ) {
puppyAge = age;
}

public int getAge( ) {
System.out.println("Puppy's age is :" + puppyAge );
return puppyAge;
}

public static void main(String []args) {
/* Object creation */
Puppy myPuppy = new Puppy( "tommy" );

/* Call class method to set puppy's age */
myPuppy.setAge( 2 );

/* Call another class method to get puppy's age */
myPuppy.getAge( );

/* You can access instance variable as follows as well */
System.out.println("Variable Value :" + myPuppy.puppyAge );
}
}
```

اگر این کد ها را کامپایل و اجرا کنیم، نتیجه زیر نمایش داده خواهد شد.

خروجی

```
Name chosen is :tommy

Puppy's age is :2

Variable Value :2
```

فایل منبع چیست؟

برای آسان تر فهمیدن ادامه این مقاله باید با فایل منبع یا سورس فایل آشنایی داشته باشید. سورس فایل به زبان ساده به همان فایل هایی گفته میشود که درون آنها کد وجود داشته باشد. یعنی اگر همین الان یک فایل نوت پد را باز کنید و یک سری کد درون آن بنویسید و آن را در جایی از کامپیوتر ذخیره کنید، یک سورس فایل ایجاد کرده اید. برنامه هایی که ما مینویسیم از تعداد زیادی فایل منبع یا سورس درست شده اند که هر کدام از آنها وظیفه بخصوصی در اپلیکیشن ر انجام میدهند.

درون محیط های برنامه نویسی مانند Eclipse و اندروید استودیو که قرار است با آنها نوشتن کد های جاوا را انجام بدهیم، همه سورس فایل هایی که مربوط به پروژه ما میبا شند و دارای کد های متفاوتی هستند را درون خودشان نمایش میدهند. از طریق همین IDE ها هم میتوانید این فایل های منبع را ایجاد، حذف و یا ویرایش کنید. پس درست کردن ریسورس فایل و کار هایی که در ادامه درباره آنها مطالبی را مطالعه خواهید کرد، کار عجیب و گیج کننده ای نیست.

قوانین مشخص کردن فایل منبع (Source File)

به عنوان آخرین قسمت این درس، میخواهم قوانین ثبت منابع را بررسی کنیم. این قوانین برای تعریف کردن کلاس، import کردن ها و درست کردن package ضروری میباشند. (درباره ایمپورت کردن و پکیج ها در ادامه همین جلسه توضیحات کافی داده شده است).

- در هر فایل فقط میتواند یک کلاس Public وجود داشته باشد.
- یک فایل منبع میتواند چندین کلاس که Public نباشند را درون خود داشته باشد.

- نام کلاس پابلیک باید با نام فایل منبع یکی با شد و پ سوند همه این فایل ها باید java. با شد. مثلا اگر نام کلاس ما {}Employee باشد، نام فایل منبع آن هم باید Employee.java باشد.
- اگر کلاس در داخل یک Package تعریف شده با شد، معرفی Package باید اولین چیزی با شد که در فایل منبع تعریف میشود.
- اگر قرار اســـت چیزی را در فا یل منبع وارد (import) کنیم، این وارد کردن با ید بین تعریف Package و تعریف اولین کلاس قرار بگیرد. اگر هم Package وجود نداشــته باشــد، import ها باید خط اول در فایل منبع ما با شند. (نگران این دو مورد آخر نبا شید، زیرا محیط برنامه نوی سی که با آن کار میکنیم، همه این import ها و تعریف کردن Package را بصـــورت اتوماتیک هرجا که نیاز باشد انجام میدهد.)
- Package و مواردی که import میکنیم، برای همه کلاس های درون فایل منبع اعمال میشـــوند. نمیتوانیم import ها و Package های مختلفی را برای کلاس های متفاوت در یک فا یل منبع تعریف کنیم. (برای این کار باید فایل های منبع متفاوت بسازیم و کلاس های را بصورت جداگانه درون آنها قرار بدهیم).
- کلاس ها سطوح د ستر سی مختلف دارند و انواع مختلفی از کلاس ها وجود دارند. مانند کلاس های انتزاعی (Abstract Classes)، کلاس های نهایی (final Classes) و غیره. درباره همه این اصطلاحات، در درس مربوط به Access Modifier ها توضیح داده خواهد شد.
- جدا از همه انواع کلاس هایی که به شما معرفی کردیم، جاوا دو مدل کلاس دیگر دارد که به آنها کلاس های داخلی (Inner Class) و کلاس های بدون نام (Anonymous Class) گفته میشود.

پکیج جاوا (Java Package) چیست؟

به زبان ســاده، پکیج یک راه برای دســته بندی فایل های کلاس و اینترفیس میباشــد. وقتی که در حال نو شتن یک برنامه به زبان جاوا ه ستید، ممکن ا ست صدها کلاس جاوا و اینترفیس نو شته ب شوند. یعنی صدها فایل منبع برای پروژه شما ایجاد خواهد شد. پس د سته بندی و مرتب کردن این فایل ها میتواند برنامه نوی سی (و حتی زندگی!) را برای شما ب سیار آ سان تر بکند. این فایل ها را با ا ستفاده از Package ها میتوانیم دسته بندی کنیم.

Package در زبان فارسی به معنی بسته بندی میباشد. همین معنی را هم در برنامه نویسی میدهد. ما پوشــه هایی تحت عنوان Package میســازیم و فایل هایی که مربوط به کار خاصــی در اپلیکیشــن ما میبا شند را درون پو شه های مربوطه میریزیم. (مثل همون کاری که تو ویندوز انجام میدیم و فایل های خودمون رو دســته بندی میکنیم). تفاوتی که در اینجا وجود دارد این اســت که باید در همه فایل هایی

که داخل یک پکیج قرار دارند، اسم آن پکیج در خط اول فایل (دقت کنید داخل فایل، نه اسم آن)، باید تعریف شود.

ایمپورت کردن (import) در جاوا چگونه است؟

کلمه import به زبان فارسی معنای وارد کردن میدهد. در زبان جاوا ممکن است که در فایل منبعی که میخواهید یک کلاس جاوا را در آن بسازید، نیاز داشته باشید که از کلاس های دیگری هم استفاده کنید. (با این بحثی که الان گفتم در درس مخصوص به ارث بری آشنا خواهید شد). برای اینکه یک کلاس که درون یک فایل دیگر قرار دارد را کاملا به فایل دیگری انتقال بدهید اما کدهای آن را ننویسید، باید از import استفاده کنید. این کار با یک دستور import و سپس دادن آدرس فایل مورد نظر و کلاس خاصی که میخواهیم انجام میشود. البته در اندروید استودیو که برای برنامه نویسی اندروید از آن استفاده میکنیم، این کار بصورت اتوماتیک برای شما انجام خواهد شد.

در زبان جاوا اگر یک نام کامل، که شامل نام پکیج و نام کلاس مورد نظر میباشد داده شود، کامپایلر به راحتی میتواند فایل منبع و یا کلاس مورد نظر را پیدا کند. عبارت import یک راه برای دادن مکان مناسب به کامپایلر است تا بتواند یک کلاس خاص را پیدا کند.

به عنوان مثال تکه کد زیر از کامپایلر جاوا میخواهد همه کلاس هایی که در آدرس نصب جاوا، در پوشه java و سپس پوشه io قرار دارند را به فایلی که درون ان هستیم import کند. (وقتی * میگذاریم، یعنی همه چیزهایی که وجود دارد را انتخاب کن).

```
import java.io.*;
```

یک مثال برای تمرین

برای یادگیری بهتر و تمرین، ما دو عدد کلاس جاوا درست میکنیم که نام های آنها Employee و EmployeeTest میباشد.

1. ابتدا محیط برنامه نویسی که برای زبان جاوا انتخاب کرده اید را باز کنید. (میتواند Eclipse یا Android Studio باشد). سپس کدهایی که در زیر نوشته شده را درون آن کپی کنید. یادتان باشد که نام این کلاس Employee میباشد و از نوع Public میباشد. این فایل را با نام Employee.java ذخیره کنید.

2. کلاس Employee دارای چهار متغیر نمونه (Instance Variable) میبا شد. نام های آنها name, age, designation و salary میباشـــد. در این کلاس یک متد ســـازنده یا همان Constructor وجود دارد که صـــریحا آن را تعریف کرده ایم. این متد به عنوان ورودی یک پارامتر را دریافت میکند.

```
import java.io.*;
public class Employee {
String name;
int age;
String designation;
double salary;
// This is the constructor of the class Employee
public Employee(String name) {
this.name = name;
}
// Assign the age of the Employee to the variable age.
public void empAge(int empAge) {
age = empAge;
}
/* Assign the designation to the variable designation.*/
public void empDesignation(String empDesig) {
designation = empDesig;
}
 /* Assign the salary to the variable salary.*/
public void empSalary(double empSalary) {
salary = empSalary;
}

/* Print the Employee details */
public void printEmployee() {
System.out.println("Name:"+ name );
System.out.println("Age:" + age );
System.out.println("Designation:" + designation );
System.out.println("Salary:" + salary);
}
}
```

همانطور که قبلا هم در این دوره آموزشی گفته شد، یک برنامه جاوا از متد main شروع به اجرا میکند. پس برای اینکه بتوانیم این کلاس Employee را اجرا کنیم، باید یک متد main در آن وجود داشـــته باشد و Object ها نیز ساخته بشوند. ما یک کلاس جداگانه برای این دستورات ایجاد میکنیم.

کدهای زیر مربوط به کلاس EmployeeTest میباشند. در این کلاس، دو نمونه از روی کلاس Employee ساخته میشود و متدهای آن فراخوانی میگردند. از این طریق مقادیری را برای متغیر ها تعیین میکنیم.

کدهای زیر را در فایل EmployeeTest.java ذخیره کنید.

```
import java.io.*;
public class EmployeeTest {

public static void main(String args[]) {
/* Create two objects using constructor */
Employee empOne = new Employee("James Smith");
Employee empTwo = new Employee("Mary Anne");

// Invoking methods for each object created
empOne.empAge(26);
empOne.empDesignation("Senior Software Engineer");
empOne.empSalary(1000);
empOne.printEmployee();

empTwo.empAge(21);
empTwo.empDesignation("Software Engineer");
empTwo.empSalary(500);
empTwo.printEmployee();
}
}
```

اکنون میتوانید هر دو کلاس جاوا را کامپایل کنید و سپس کلاس EmployeeTest را اجرا کنید تا نتیجه را که مانند شکل زیر است ببینید:

```
C:\> javac Employee.java
C:\> javac EmployeeTest.java
C:\> java EmployeeTest
Name:James Smith
Age:26
Designation:Senior Software Engineer
Salary:1000.0
Name:Mary Anne
Age:21
Designation:Software Engineer
Salary:500.0
```

در جلسه بعد چه چیزی یاد میگیریم؟

در جلسه بعدی، درباره مفاهیم شی گرایی در جاوا صحبت خواهیم کرد و در درس بعدی با این مفهوم در برنامه نویسی جاوا آشنا خواهید شد.

جلسه قبل: سینتکس جاوا

جلسه بعد: شی گرایی