

بخش ششم:

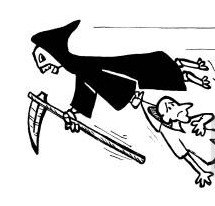
تولد و مرگ اشیاء (Object Initialization & Clean up)

# تولد و مرگ اشیاء (Object Initialization & Clean up)

در این فصل می خواهیم درباره ی فرآیند ایجاد و از بین رفتن اشیاء در جاوا صحبت کنیم. در واقع می خواهیم فرآیند تولد (Initialization) و مرگ (Cleanup) را برای اشیاء در زبان جاوا مرور کنیم. ابتدا در این باره صحبت خواهیم کرد که یک شی چگونه ایجاد می شود؟



وضعیت اولیه اشیاء چگونه تنظیم می شود؟ درباره ی نقش سازنده یا Constructor صحبت می کنیم و در انتها درباره ی فرآیند مرگ اشیاء یا Cleanup صحبت می کنیم و می بینیم که اشیاء چگونه از بین می روند و این از بین رفتن چه فرآیندی را طی می کند.



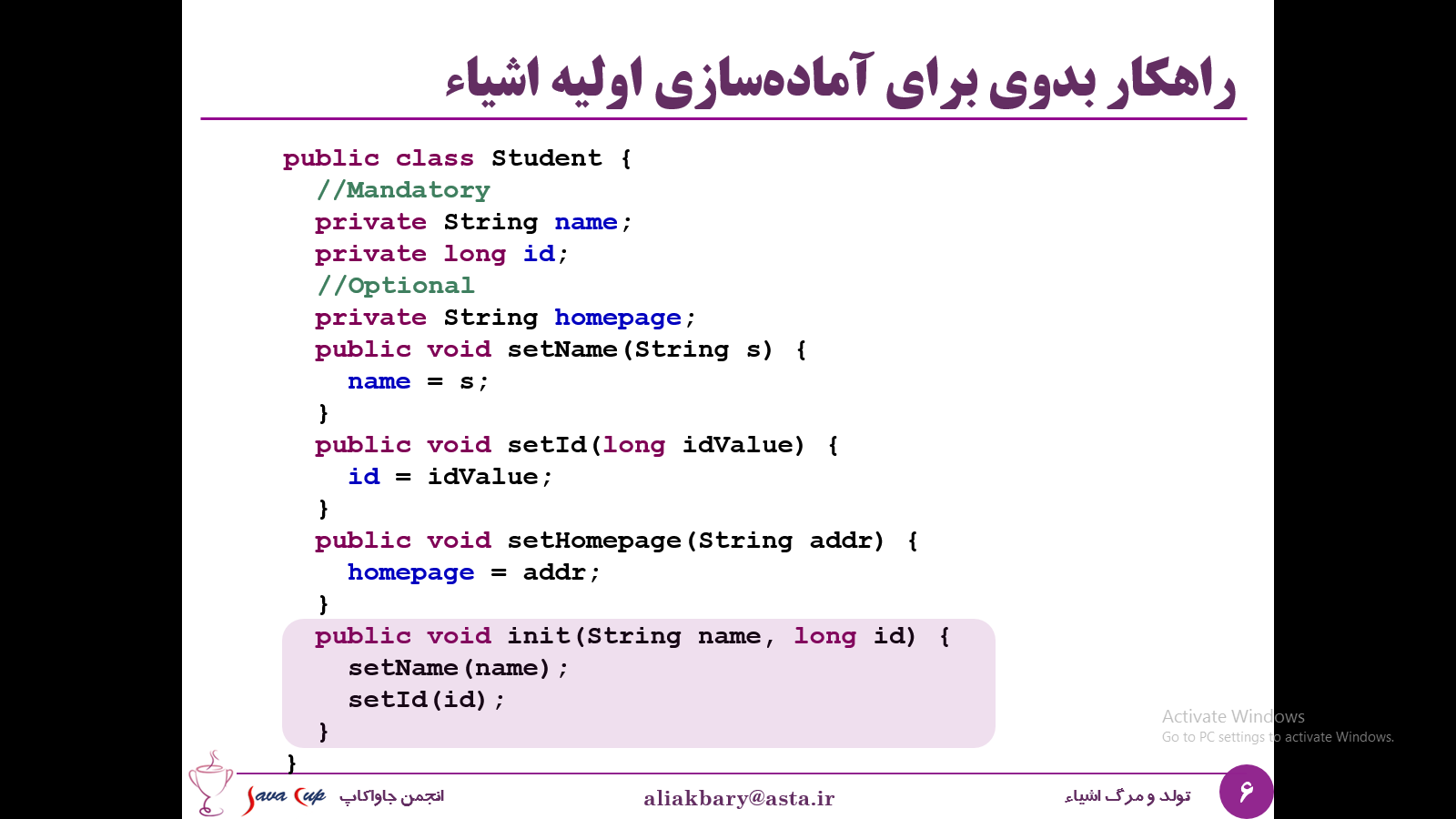
# آماده سازی اولیه اشیاء (Initialization)

واقعیت این است که یک شی بلافاصله بعد از ایجاد شدن لزوما آماده ی استفاده نیست و ممکن است که این شی هنوز در وضعیت نامعتبر باشد. به عنوان مثال فرض کنید که یک شی از جنس شخص ایجاد کنیم:

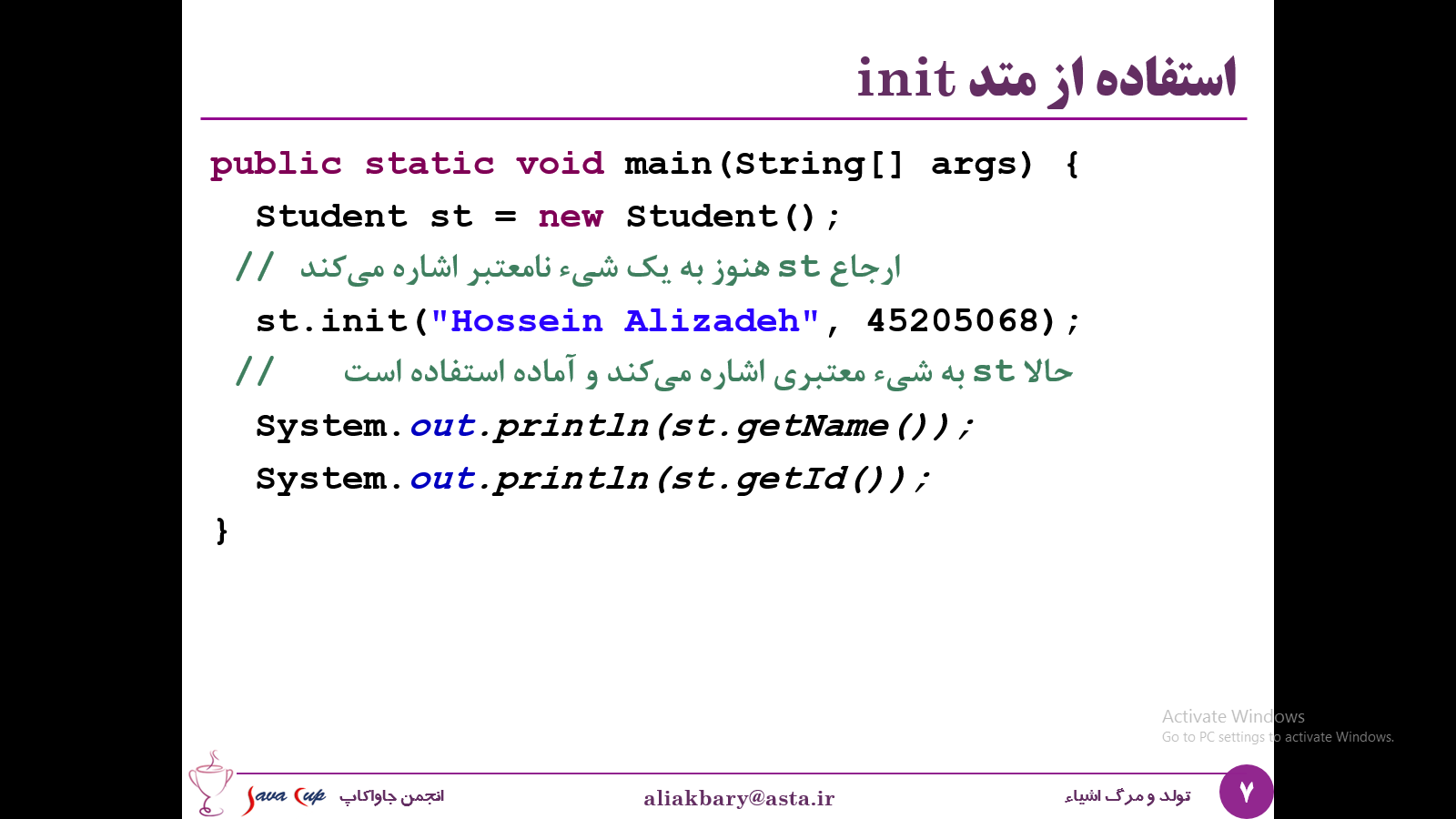
Person p = new Person();

در این حالت، شی ای که p به آن شاره می کند احتمالا معتبر نیست چرا که هیچ یک از ویژگی های لازمی که در این کلاس تعریف شده و وجود دارد را ندارد، به عنوان مثال این شخص هنوز نام یا شماره ملی ندارد. در نتیجه باید اصطلاحا آماده سازی اولیه بشود (Initialization) و مقدار دهی اولیه برای ویژگی های مهم آن صورت بگیرد تا به وضعیت معتبری برود و بتوانیم از آن استفاده کنیم.

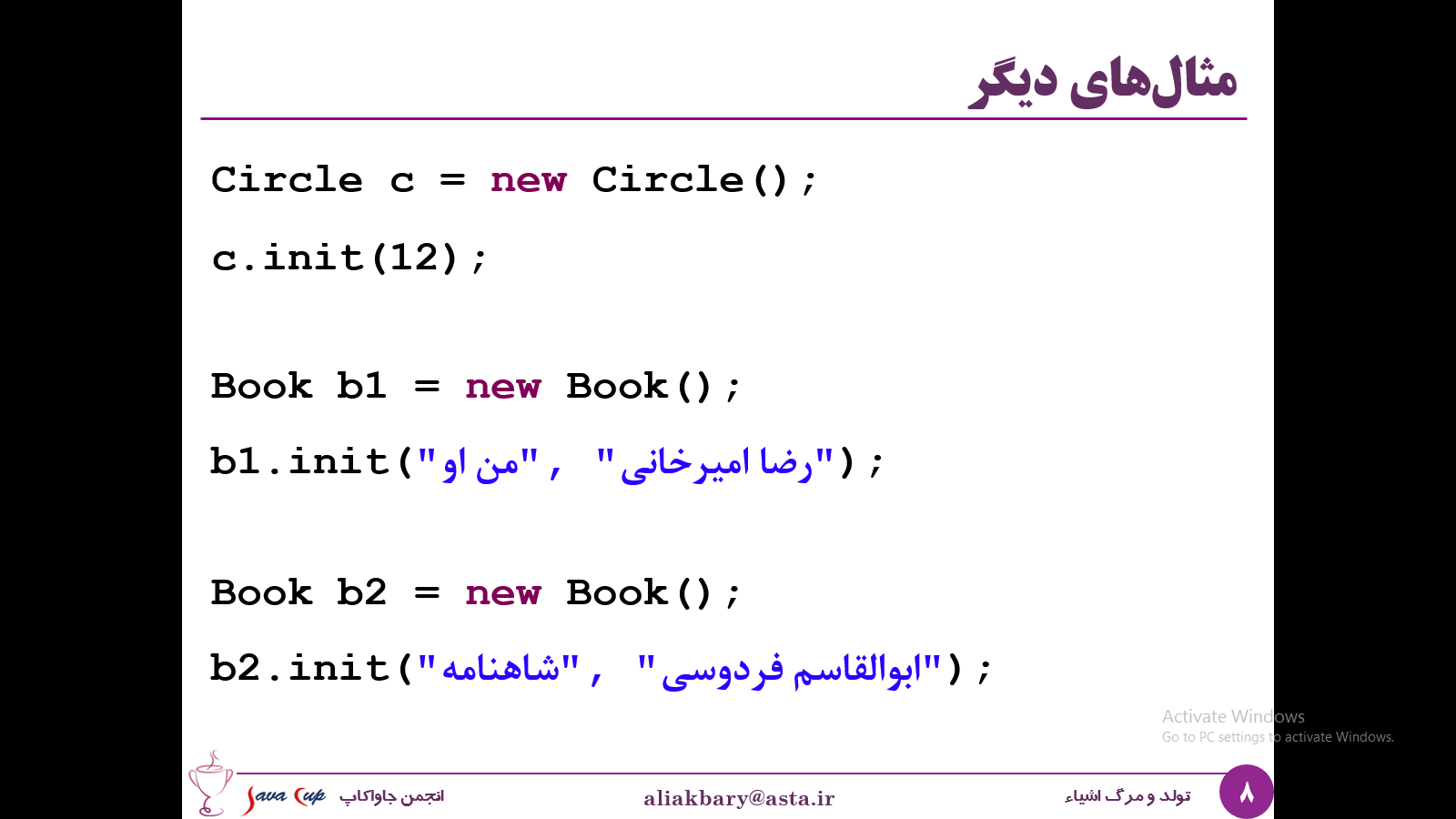
به مثال زیر نگاه کنید، فرض کنید کلاسی داریم به نام Student که ویژگی هایی دارد که برای استفاده از اشیایی از جنس این کلاس لازم است و همچنین ویژگی های دیگری که ضروری نیستند. مثلا هر شی باید نام و شماره دانشجویی داشته باشد اما اگر آدرس اینترنتی نداشت هم اشکالی ندارد. فرض کنید رفتارهایی مانند setter ها و یک متد به نام init برای آن تعریف کرده ایم که این متد init وظیفه ی آماده سازی کلاس را بر عهده دارد. به عنوان مثال این متد ویژگی نام و شماره دانشجویی را دریافت می کند و آن را برای شی تنظیم می نماید.



این یک راه اولیه برای آماده سازی اشیاء است، در واقع ما می توانیم برای هر کلاس متدی با این نام ایجاد کنیم و هر وقت که یک شی ایجاد کردیم، بلافاصله این متد را فراخوانی کنیم تا شی ما آماده سازی و مقدار دهی اولیه بشود. به عنوان مثال کلاسی که ایجاد کردیم را می توانیم به صورت زیر استفاده کنیم.

 واقع هر زمان یک شی ایجاد کردیم، بلافاصله بعد از آن متد init را فراخوانی کنیم و نام و شماره دانشجویی این شی جدید را مشخص کرده و به آن ارسال کنیم تا شی ما وارد وضعیت معتبر بشود. در واقع قبل از اینکه این متد فراخوانی شود شی ساخته شده است، اما هنوز قابل استفاده نیست. اما بعد از مقدار دهی اولیه آماده است تا از آن استفاده کنیم و مثلا متدهایی را روی آن فراخوانی نماییم.

برای کلاس های مختلف می توان این کار را انجام داد، مثلا فرض کنید کلاسی مانند Circle و یا Book داریم و هر زمان شی از جنس دایره می سازیم، بلافاصله متد init را فراخوانی کنیم و شعاع دایره را مشخص نماییم و یا هر وقت که یک کتاب را new می کنیم، از این طریق نام کتاب و نویسنده را مشخص نماییم. توجه کنید که برای هر شی جدیدی که بسازیم دوباره باید متد init را فراخوانی کنیم تا آن شی معتبر و قابل استفاده شود.



# اشکال راهکار استفاده از متد init چیست؟

تعریف و استفاده از متد init یک راهکار اولیه برای مقدار دهی و معتبرسازی اشیاء است و اشکالاتی دارد. به نظر شما اشکال این رویکرد چیست؟

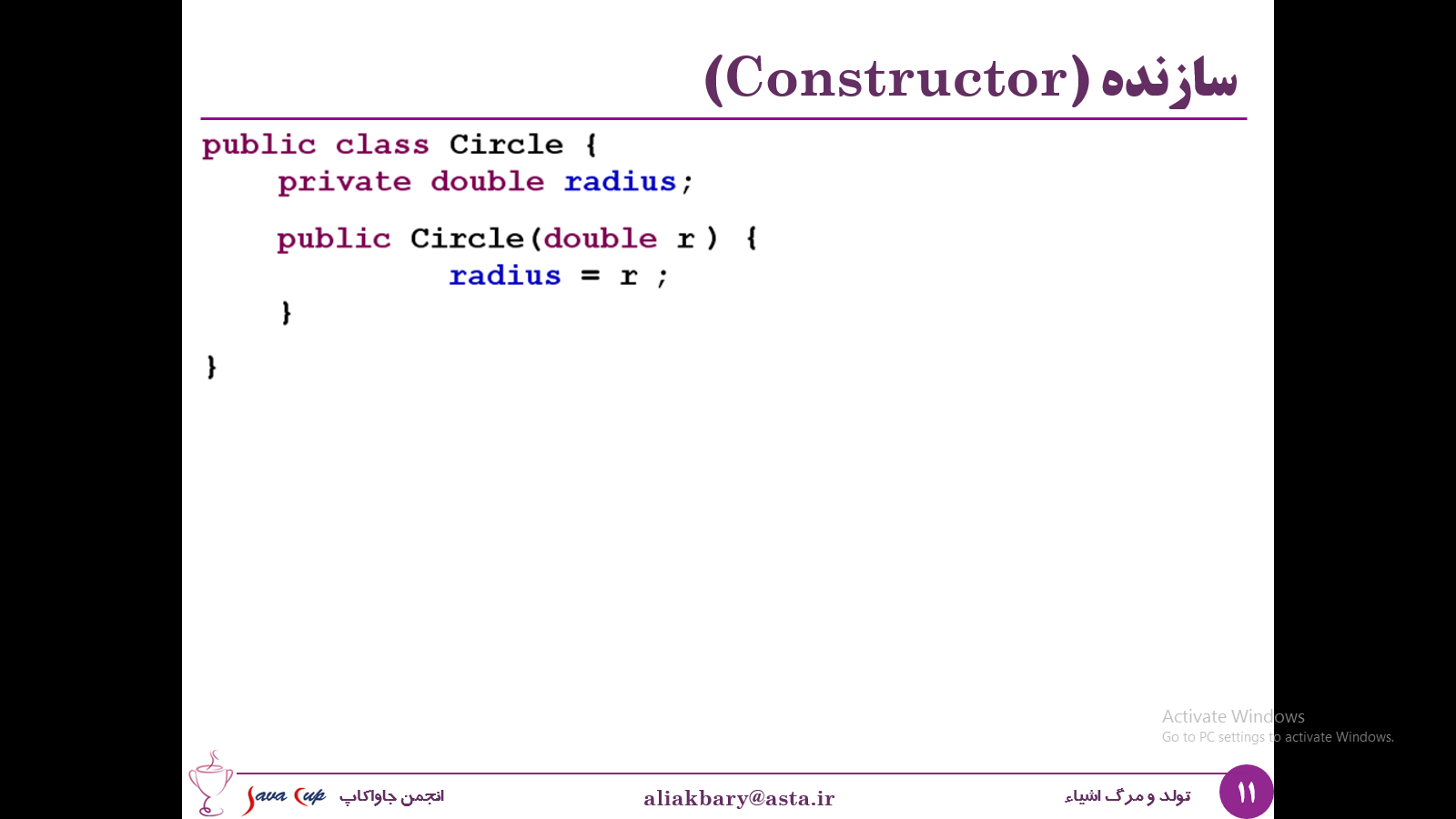
یک اشکال این است که این متد باید به صورت دستی (Manual) فراخوانی شود. یعنی برنامه نویس باید هر گاه شی ای می سازد خودش به صورت دستی این متد را فراخوانی کند در نتیجه ممکن است فراموش کند و تضمینی برای فراخوانی شدن این متد وجود ندارد.

نکته ی منفی دیگر این است که با این رویکرد قبل از فراخوانی متد init شی ما در حالت نامعتبر قرار دارد، در واقع برهه ای از زمان وجود دارد که شی ایجاد شده است اما نامعتبر است.

در نتیجه راهکار مطلوب برای این وضعیت این است که ما روشی را ابداع کنیم و کاری کنیم که فراخوانی این متد، یعنی مقدار دهی اولیه به صورت خودکار در همان لحظه ی ایجاد شدن شی اتفاق بیافتد. با این اتفاق دیگر هیچ گاه شی ما در حالت نامعتبر نخواهد بود چرا به به صورت اتوماتیک در لحظه ی ایجاد initialize شده است. این راهکار مطلوب با عنوان سازنده (Constructor) در زبان های برنامه نویسی شی گرا مانند جاوا وجود دارد و قابل استفاده است. در ادامه نحوه ی استفاده از آن را خواهیم دید.

# سازنده (constructor)

می خواهیم در مورد مفهوم سازنده صحبت کنیم. فرض کنید کلاسی با نام Circle داریم که یک property به نام شعاع دارد، برای این کلاس ما می توانیم یک متد خاص تعریف کنیم که هم نام کلاس ما هست و یک پارامتر می گیرد و ویژگی شعاع را تنظیم یا initialize می کند.



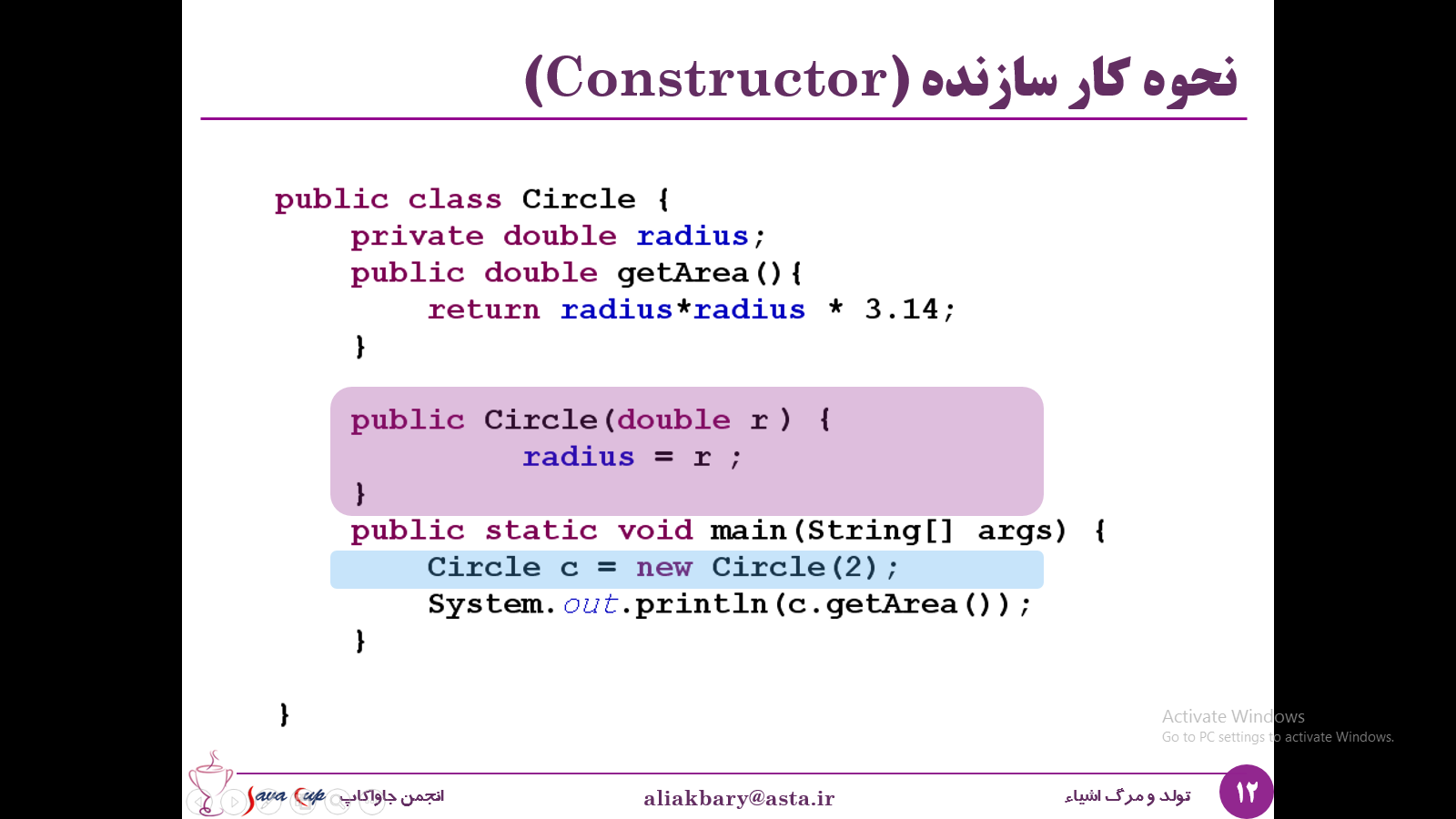
در این حالت ما یک متد خاص داریم که به آن Constructor (سازنده) گفته می شود. دقت کنید که این متد نامی مساوی نام کلاس دارد و هیچ نوع برگشتی هم ندارد، مثلا ما نگفتیم public void Circle و هیچ نوع برگشتی برای این متد مشخص نشده است. بقیه ی تعریف سازنده مشابه متد است و می تواند پارمترهایی داشته باشد و بدنه ای دارد که نحوه ی مقدار دهی اولیه شی را مشخص می کند.

این متدی هست که به جای راهکاری مانند متد init می خواهیم از آن استفاده کنیم. هر زمان که یک شی از کلاس Circle ایجاد شود، این constructor به صورت خودکار فراخوانی خواهد شد. مثلا وقتی که یک شی با کمک اپراتور new ساخته می شود، بلافاصله به صورت خودکار این متد فراخوانی و بدنه ی آن اجرا می شود. به این ترتیب شی ما از همان لحظه ی تولد حالت معتبر خواهد داشت.

پس برخلاف رویکر قبلی، در اینجا نیازی به فراخوانی Constructor توسط برنامه نویس نیست.

# نحوه ی کار سازنده (Constructor)

نحوه ی کار constructor چگونه است؟ فرض کنید که متد Circle به صورتی که در مثال قبل دیدیم نوشته شده است، حال به صورت زیر می توانیم از آن استفاده کنیم:



مثلا زمانی که شما یک شی از این کلاس new می کنید باید داخل پرانتز پارامترهای مورد نیاز constructor را ارسال کنید تا سازنده ی متناظر فراخوانی شود. به این ترتیب یک object جدید از جنس Circle ساخته خواهد شد که بلافاصله پس از ساخته شدن توسط روال سازنده مقدار دهی شده است. مثلا در مثال بالا ما شی ای ساختیم که مقدار شعاع برای آن برابر 2 خواهد بود. در نتیجه بلافاصله بعد از استفاده این شی قابل استفاده است و در هیچ زمانی در حالت نامعتبر قرار ندارد.

# سازنده هایی با پارامترهای مختلف

سازنده هایی که ما تعریف می‌کنیم می‌توانند صفر یا بیشتر پارامتر داشته باشد. همچنین یک کلاس می‌تواند سازنده های مختلفی داشته باشد که با کمک آنها نشان می‌دهیم به چه طرقی می‌توان از روی آن کلاس شی ایجاد کرد. شرط تعریف چند constructor برای یک کلاس این است که آنها در تعداد و یا نوع پارامترها تفاوت داشته باشند و اصطلاحا به این سازنده‌ها، Overloading Constructor گفته می‌شود.

اگر برنامه‌نویس، هیچ سازنده‌ای برای یک کلاس تعریف نکند، کامپایلر جاوا به صورت خودکار یک Default Constructor برای آن کلاس در نظر می‌گیرد، و می‌توان فرض کرد که بدنه‌ی این سازنده‌ی پیش فرض خالی است و کار خاصی انجام نمی‌دهد، البته بعدها خواهیم دید که کارهایی انجام می‌دهد. اما هنگامی که برنامه نویس اولین constructor را برای یک کلاس تعریف می‌کند، دیگر کامپایلر جاوا سازنده‌ی پیش فرض را ایجاد نمی‌کند. البته در برخی از زبان‌های برنامه‌نویسی Default constructor به سازنده‌ای که هیچ پارامتری ندارد گفته می‌شود اما معمولا در جاوا به سازنده‌ای گفته می‌شود به صورت پیش فرض توسط کامپایلر ایجاد می‌شود.

**public** **class** Circle {

**private** **double** radius;

**public** **double** getArea(){

**return** radius\*radius\*3.14;

}

**public** Circle(**double** r) {

radius = r;

}

**public** Circle() {

}

}

مثال اول:

برای مثال برنامه‌ی رو به رو را در نظر بگیرید، کلاسی به نام Circle که دو سازنده دارد، یک Constructor بدون پارامتر و یکی هم با یک پارامتر ورودی.

سوال اینجاست که چگونه می‌توانیم از این کلاس شی بسازیم و از آن استفاده کنیم؟

ما می‌توانیم ارجاعی مانند c را به دو صورت new یکی با پارامتر که شعاع دایره را از ما دریافت می‌کند و دیگری بدون پارامتر، در واقع اختیار اینکه از کدامیک از این سازنده‌ها استفاده کنیم دست خود ماست و اصلا لزومی ندارد که از هر دوی آنها استفاده کنیم.

Circle c;

c = **new** Circle();

c = **new** Circle(12);

و همانطور که اشاره کردیم اصطلاحا به این کار که سازنده‌هایی با پارامترهای مختلف ایجاد کنیم، Constructor overloading گفته می‌شود.

مثال دوم:

در این مثال نیز ما برای کلاس Constructor نوشته ایم در نتیجه کامپایلر سازنده ی پیش فرضی برای آن قرار نخواهد داد، همچنین برخلاف مثال قبل Circle سازنده ی بدون آرگومان ورودی ندارد، بنابراین دو خط اول متد main به درستی قابل اجراست، اما خط آخر خطاست و ما نمی توانیم برای این کلاس سازنده را به این شکل فراخوانی کنیم، چون سازنده ی بدون پارامتر برای این کلاس وجود ندارد، و به محض اینکه ما اولین سازنده را برای کلاسی ایجاد می کنیم، Default Constructor از بین می رود. در نتیجه در صورتی که هیچ سازنده ای برای Circle تعریف نمی کردیم شرایط برعکس می شد و این بار خط آخر main خطایی نداشت اما خط دوم غیر قابل کامپایل و اجرا بود چرا که در آن صورت کلاس ما سازنده ای با ورودی Double نداشت.

**public** **class** Circle {

**private** **double** radius;

**public** **double** getArea() {

**return** radius \* radius \* 3.14;

}

**public** Circle(**double** r) {

radius = r;

}

**public** **static** **void** main(String[] args) {

Circle c;

c = **new** Circle(12);

c = **new** Circle();

}

}

در چنین مواقعی، برنامه ی ما کامپایل نمی شود و یک Syntax Error با پیغام زیر دریافت خواهیم کرد:

The Constructor Circle() is undefined

# وظیفه‌ی سازنده

وظیفه‌ی یک سازنده چیست؟ و ما هنگامی که یک Constructor ایجاد می‌کنیم، قصد انجام چه کاری را داریم؟

هر زمان که یک شی ساخته می شود، Constructor ما به صورت خودکار فراخوانی خواهد شد، و وظیفه دارد که ویژگی های آن شی را مقداردهی اولیه(initialize) نماید. و اگر لازم باشد، برخی از این ویژگی ها خودشان باید ایجاد (new) شوند، و در واقع اگر ویژگی هایی از شی ما reference باشند، لازم است instantiate یا نمونه سازی بشوند.

مثال:

کلاس Car دارای ویژگی engine و آرایه ای از Tyre است. بنابراین در سازنده ی این کلاس، ما باید ویژگی engine را ایجاد(instantiate) کنیم و برای چرخ ها علاوه بر ایجاد آرایه، باید هر کدام از آنها را نیز new کنیم تا هر کدام از خانه های این آرایه یک شی معتبر شوند. بنابراین سازنده علاوه بر مقدار دادن به ویژگی های ساده، مانند ویژگی هایی که از جنس اعداد و یا رشته ها هستند، بایست ویژگی هایی که اشیاء پیچیده تری هستند را هم ایجاد کند و مقدار دهی اولیه نماید.

**public** **class** Car {

**private** Engine engine;

**private** Tyre[] tyres;

**public** Car() {

engine = **new** Engine();

tyres = **new** Tyre[4];

**for** (**int** i = 0; i < tyres.length; i++) {

tyres[i] = **new** Tyre();

}

}

}

# مقادیر پیش فرض

اما اگر سازنده برخی ویژگی های کلاس ما را مقدار دهی نکند، چه می شود؟ در واقع ممکن است کلاس ما ویژگی (Property) هایی داشته باشد که تنظیم و مقدار دهی آن در سازنده اجباری نباشد، در چنین شرایطی، مقدار آن ویژگی چه خواهد بود؟

اگر سازنده مقدار یک ویژگی را مشخص نکند، آن ویژگی مقدار پیش فرض نوع داده ی خودش را خواهد گرفت، مثلا اگر آن ویژگی از نوع int باشد، مقدار صفر می گیرد چون مقدار پیش فرض int صفر است، در ادامه با مقادیر پیش فرض انواع مختلف داده آشنا می شویم:

* مقدار پیش فرض Boolean : false
* برای سایر primitive data type ها(int, char, long, …) مقدار پیش فرض صفر است.
* برای سایر انواع داده که در واقع متغیرهای ارجاعی هستند، و قرار است به یک شی اشاره کنند، مقدار پیش فرض مقدار null است. (که پیش از این گفتیم به معنای اشاره به هیچ چیز است)

# کویز

* کلاس ConstructQuize با ویژگی های زیر که برخی از انواع داده ی اولیه و برخی ارجاع هستند را در نظر بگیرید. برای این کلاس سازنده ای نوشته ایم که دو ویژگی آن را مقداردهی می کند، سپس در متد main یک شی جدید ساخته و مقادیر مناسب را برای سازنده ی آن ارسال کرده ایم، و بلافاصله ویژگی های مختلف این شی را چاپ کرده ایم، شما بگویید خروجی این برنامه چه خواهد بود؟

**public** **class** ConstructorQuiz {

**private** **int** number;

**private** **double** real;

**private** **boolean** condition;

**private** String name;

**private** Circle circle;

**public** ConstructorQuiz(**int** num, String title) {

number = num;

name = title;

}

**public** **static** **void** main(String[] args) {

ConstructorQuiz q = **new** ConstructorQuiz(5, "Ali");

System.***out***.println(q.number);

System.***out***.println(q.real);

System.***out***.println(q.condition);

System.***out***.println(q.name);

System.***out***.println(q.circle);

}

}

پاسخ:

با توجه به اینکه سازنده ای که ما تعریف کرده ایم تنها دو ویژگی کلاس ما را مقدار دهی می کنند، سایر ویژگی ها مقادیر پیش فرض خواهند گرفت پس خروجی به صورت زیر خواهد بود:

5

0.0

false

Ali

Null

* خروجی برنامه تغییر یافته ی زیر چیست؟ در برنامه ی زیر تنها سازنده ی فراخوانی شده هنگام ساخته شدن شی تغییر کرده است و سازنده ی بدون آرگومان ورودی را فراخوانی کرده ایم، حال بگویید خروجی چه خواهد بود؟

پاسخ:

**public** **class** ConstructorQuiz {

**private** **int** number;

**private** **double** real;

**private** **boolean** condition;

**private** String name;

**private** Circle circle;

**public** ConstructorQuiz(**int** num, String title) {

number = num;

name = title;

}

**public** **static** **void** main(String[] args) {

ConstructorQuiz q = **new** ConstructorQuiz();

System.***out***.println(q.number);

System.***out***.println(q.real);

System.***out***.println(q.condition);

System.***out***.println(q.name);

System.***out***.println(q.circle);

}

}

پاسخ صحیح این سوال خطای کامپایل (Syntax Error) است، چون چنین سازنده ای در کلاس ما وجود ندارد، چون ما چنین سازنده ای تعریف نکرده ایم، و چون یک سازنده با دو آرگومان ورودی ایجاد کرده ایم، کامپایلر سازنده ی پیش فرضی هم برای کلاس ما قرار نداده است.

# سایر روش های مقدار دهی اولیه

دو روش مهم دیگر (به جز استفاده از Constructor) برای مقدار دهی اولیه اشیاء وجود دارد:

* مقدار دهی اولیه در خط (Inline Initialization)

می توان مقدار یک ویژگی از کلاس را در زمان تعریف آن ویژگی در همان محل انجام داد، مثلا برای کلاس Car می توانیم ویژگی engine را برخط new کنیم و یا ویژگی مانند numberOftyres را مقدار دهی نماییم، و به همین ترتیب می توانیم آرایه ی چرخ ها را مقدار دهی نماییم، پس هر ویژگی می تواند هنگام تعریف مقدار اولیه بگیرد، اما باید به این دقت کنیم که مانند Constructor زمانی این مقدار دهی اولیه انجام می شود که ما یک شی از این کلاس بسازیم، یعنی این دستورات مقدار دهی فراخوانی نمی شوند، تا زمانی که یک شی از کلاس Car ایجاد کنیم. و هر بار که یک شی جدید از این کلاس ایجاد کنیم، این دستورات برای ویژگی های همان شی فراخوانی خواهند شد.

**public** **class** Car {

**private** Engine engine = **new** Engine();

**private** **int** numberOfTyres = 4;

**private** Tyre[] tyres = **new** Tyre[numberOfTyres];

**public** Car() {

**for** (**int** i = 0; i < tyres.length; i++) {

tyres[i] = **new** Tyre();

}

}

}

البته در این مثال هم مشخص است که برخی از ویژگی ها مانند عناصر آرایه ی tyres را نمی توان به سادگی به صورت inline مقدار دهی کرد، مثلا اگر مقدار دهی ما نیاز به شرط و یا حلقه دارد و یا نیاز است چند دستور برای آن اجرا شوند، نمی توانیم آن را به صورت inline انجام دهیم و باید یا از سازنده استفاده کنیم، یا از روش دیگری که در ادامه با آن آشنا خواهیم شد به نام بلوک مقدار دهی اولیه استفاده نماییم.

* بلوک مقدار دهی اولیه (Initialization Block)

این روش هم مانند سازنده و یا مقدار دهی برخط برای مقدار دهی اولیه ی ویژگی های اشیا به کار می رود. همان طور که می دانید به بخشی از کد که بین آکولاد باز و آکولاد بسته قرار می گیرد، یک بلوک گفته می شود. مثلا دستورات در حلقه ها و یا شرط ها در صورتی که بیش از یک دستور داشته باشند، در یک بلوک قرار داده می شود.

**public** **class** Car {

**private** **int** numberOfTyres = 4;

**private** Tyre[] tyres;

{

tyres = **new** Tyre[numberOfTyres];

**for** (**int** i = 0; i < tyres.length; i++) {

tyres[i] = **new** Tyre();

}

}

}

بلوک مقدار دهی اولیه یک بلوک در متن و بدنه ی کلاس ماست و هیچ نام و نشانی ندارد، پس یک اسم ندارد و یا یک متد نیست، داخل این بلوک دستورات و برنامه هایی نوشته می شود که می خواهیم هر زمان یک شی از این کلاس ساخته شد اجرا شوند. برای مثال در کد بالا برای کلاس Car یک Initialization Block تعریف کرده ایم که برخی از ویژگی های این کلاس را مقدار دهی می کند.

# ترتیب مقدار دهی اولیه

ترتیب مقدار دهی اولیه در یک کلاس هنگامی که شی جدیدی از آن کلاس ایجاد می شود به صورت زیر است:

* اولین قدم اجرای تمام مقدار دهی های در خط (inline) است.
* قدم دوم اجرای تمامی بلوک های مقدار دهی اولیه است. اگر یک یا چند initialization block وجود داشته باشند در این مرحله اجرا خواهند شد. (البته معمول نیست که یک کلاس یک بلوک مقدار دهی اولیه داشته باشد اما امکان آن وجود دارد)
* و در مرحله ی سوم یکی از Constructor ها فراخوانی می شود، اما اگر یک کلاس چند سازنده داشته باشد کدامیک فراخوانی خواهد شد؟ سازنده ای که آرگومان های ورودی اش مطابق با فراخوانی انجام شده باشد. در واقع هنگامی که ما یک شی جدید می سازیم مشخص می کنیم که کدامیک از سازنده های آن فراخوانی شود.

نکته: در یک کلاس تعدد Constructor ها لزوما چیز بدی نیست، و اتفاق رایجی است که ما برای یک کلاس چندین سازنده تعریف کنیم، با توجه به اینکه می خواهیم اجازه دهیم به چه شکل هایی از کلاس ما شی ایجاد شود.

# کاربرد this برای سازنده ها

فرض کنید که بخواهیم داخل یک Constructor، یک Constructor دیگر را فراخوانی کنیم، ما می دانیم که داخل یک متد می توان متدی دیگری را با استفاده از نام آن فراخوانی کرد، اما سازنده که نام مشخصی ندارد را چگونه می توان فراخوانی کرد؟ کاربرد این فراخوانی برای زمانی است که می خواهیم از کدی که قبلا نوشته شده است استفاده ی مجدد (Code Reuse) کنیم، در واقع به جای آنکه آن کد را مجددا تکرار (Copy) کنیم که در آینده ممکن است مشکلات فراوانی برای ما ایجاد کند، می خواهیم با فراخوانی آن سازنده از امکانات داخلی آن استفاده کنیم.

برای این کار می توان از کلید واژه ی this استفاده کرد، همانطور که می دانیم سازنده ها اسم ندارند و عملا یک تابع مستقل نیستند که بتوان آنها را با کمک نامشان فراخوانی کرد، اما می توان با استفاده از this آنها را فراخوانی نمود، نکته ای که باید به آن توجه کنیم این است که این فراخوانی باید حتما اولین دستور سازنده ی ما باشد، پس اگر داخل یک سازنده می خواهیم سازنده ی دیگری از همان کلاس را فراخوانی کنیم، این فراخوانی باید حتما در اولین خط دستورات ما نوشته شود و با نوع و تعداد پارامترها دقیقا مشخص می کنیم که می خواهیم کدام سازنده فراخوانی شود، البته کلید واژه ی this کاربردهای دیگری هم دارد که بعدها با آن آشنا خواهیم شد.

مثال: به کد زیر دقت کنید، یک کلاس به نام Country تعریف کرده ایم که ویژگی های نام و جمعیت دارد، و3 سازنده برای آن تعریف کرده ایم، و داخل Constructor دوم، سازنده ی اول را فراخوانی کرده ایم، در واقع با نوشتن this() مشخص کرده ایم که باید سازنده ای بدون پارامتر در اینجا فراخوانی شود، به همین ترتیب در سازنده ای که دو پارامتر می گیرد سازنده ی تک پارامتری را فراخوانی و مقدار جمعیت را به آن ارسال کرده ایم، و پس از آن ویژگی نام را مقدار دهی کرده ایم.

پس دقت کنید که ما برای جلوگیری از تکرار کدهایی که در سازنده های معمولا کم پارامتر تر نوشته شده از کلمه ی کلیدی this استفاده کردیم. و این فراخوانی باید اولین دستور سازنده باشد.

**public** **class** Country {

**private** String name;

**private** **int** population;

**public** Country() {

name = "Iran";

}

**public** Country(**int** number) {

**this**();

population = number;

}

**public** Country(String n, **int** number) {

**this**(number);

name = n;

}

}

# خلاصه مقدار دهی اولیه

اگر بخواهیم این بخش را جمع بندی کنیم، دیدیم که امکانات مختلفی برای مقدار دهی اولیه اشیاء در زبان جاوا وجود دارد:

* In line initialization
* Initialization Block
* Constructors

و همواره باید انتخاب کنیم که کدام یک از این روش ها و یا ترکیب آنها برای هر کلاس مناسب است، اگر مقدار دهی اولیه یک ویژگی خیلی ساده و در حد یک مقدار مشخص، فراخوانی متد و یا instantiation ساده است، به سادگی می توان از inline initialization استفاده نمود و در همانجا که ویژگی تعریف شده است، مقدار آن را هم مشخص کرد.

اگر یک مجموعه کد برای مقداردهی اولیه نیاز دارید، و این مجموعه کد قرار است در تمامی سازنده ها تکرار شوند، و این تکه کد احتیاجی به پارامتر خاصی ندارد می توانید آن را در initialization block قرار دهید تا دیگر نیازی نباشد در سازنده های مختلف آن را تکرار کنید.

و اگر مقدار دهی اولیه ی شما نیاز به پارامترهایی دارد که هنگام ساخته شدن شی به آن ارسال می شوند، و توسط کاربر تعیین می گردند از دو روش قبل نمی توانید استفاده کنید، و باید constructor هایی تعریف کنید که پارامترهایی ورود بگیرند و ویژگی های اولیه ی شی را تعیین کنند.

# کویز

حال می خواهیم موضوع مقداردهی اولیه اشیاء را تمرین کنیم.

* فرض کنید کلاسی به نام Quiz به صورتی که در زیر آمده است تعریف کرده ایم، در این کلاس از شکل های مختلف مقدار دهی اولیه استفاده کرده ایم.

**public** **class** Quiz {

**public** **int** number = f();

**private** **int** f() {

System.***out***.println("Inline Initialization"); **return** 1;

}

{System.***out***.println("Initialization Block"); number = 2;}

**public** Quiz() {

System.***out***.println("NO-arg constructor"); number = 3;

}

**public** Quiz(**int** num) {

System.***out***.println("ONE-arg constructor"); number = num;

}

}

حال اگر در متد main به صورت زیر از این کلاس شی ایجاد کنیم، خروجی برنامه (مقادیری که چاپ خواهد شد) چه خواهد بود؟

Quiz q = **new** Quiz();

System.out.println(q.number);

q = **new** Quiz();

System.out.println(q.number);

q = **new** Quiz(7);

System.out.println(q.number);

پاسخ:

Inline Initialization

Initialization Block

NO-arg constructor

3

Inline Initialization

Initialization Block

NO-arg constructor

3

Inline Initialization

Initialization Block

ONE-arg constructor

7

هنگامی که اولین شی از کلاس ما ایجاد می شود، باید مراحل مقداردهی اولیه به ترتیب برای آن اجرا شوند، پس ابتدا مقداردهی درخط صورت می گیرد و تابع f فراخوانی شده و علاوه بر مقدار دهی number با مقدار 1 عبارت داخل تابع چاپ می شود، بعد از آن نوبت به بلوک مقداردهی اولیه می رسد که ویژگی number را برابر 2 قرار می دهد و عبارت Initialization Block را چاپ می نماید، و در نهایت یکی از سازنده ها فراخوانی می شود، کدام یک؟ سازنده ای که پارامتر ورودی ندارد، پس عبارت No-arg constructor چاپ و مقدار number برابر 3 خواهد شد و به همین ترتیب ...

نکته ای که برای ساخته شدن دومین و سومین شی مهم است این است که مقدار دهی درخط و بلوک مقدار دهی اولیه به ازای هر بار ساخته شدن شی برای ویژگی های آن شی فراخوانی می شود.

* حال همان کلاس را با تغییرات جزئی زیر در نظر بگیرید، این بار در سازنده ی دوم سازنده ی اول را با کمک this فراخوانی کرده ایم:

**public** Quiz(**int** num) {

**this**();

System.out.println("ONE-arg constructor"); number = num;

}

حال خروجی تکه کد زیر چه خواهد بود؟

Quiz q = **new** Quiz();

System.out.println(q.number);

q = **new** Quiz(7);

System.out.println(q.number);

پاسخ:

خروجی برنامه به صورت رو به رو خواهد بود. که بخش اول آن تفاوتی با سوال قبل ندارد، اما در بخش دوم هنگامی که نوبت به فراخوانی سازنده می رسد (بعد از اجرای مقدار دهی برخط و بلوک مقدار دهی) در ابتدای اجرای سازنده ی دارای یک پارامتر سازنده ی بدون پارامتر فراخوانی و اجرا می شود، پس ابتدا عبارت NO-arg constructor و سپس عبارت ONE-arg constructor چاپ خواهد شد.

Inline Initialization

Initialization Block

NO-arg constructor

3

Inline Initialization

Initialization Block

NO-arg constructor

ONE-arg constructor

7

# فرآیند مرگ اشیاء

* نابودگر (Destructor)