

بخش دوم: مبانی برنامه‌نویی شیء‌گرا **(Object Orientation)**  در جاوا

# برنامه نویسی شی گرا (OOP)

در این فصل در مورد برنامه نویسی شی گرا و مفاهیمی مانند کلاس، شی، ویژگی ها و متدها صحبت خواهیم کرد. مفهوم واسط (Interface) و نقش آن در برنامه نویسی شی گرا را خواهیم دید. با مفهوم محصورسازی (Encapsulation) آشنا خواهیم شد و نحوه ی نوشتن و استفاده از کلاس ها را خواهیم آموخت.

* پیشینه زبان های برنامه نویسی

همان طور که می دانید برنامه نویسی با زبان ماشین ممکن است اما برای انسان ها کار بسیار سختی است به همین دلیل زبان اسمبلی به وجود آمده است برای برنامه نویسی ساده تر در سطح ماشین، در زبان اسمبلی نام گذاری برای دستورات عددی زبان ماشین انچام شده تا حفظ کردن دستورات و خواندن و نوشتن برنامه ها را ساده تر کند، اما همچنان برنامه نویسی به زبان اسمبلی به علت سطح پایین بودن (دور بودن از نحوه ی تفکر انسان) بسیار دشوار است. به همین دلیل زبان های سطح بالا به وجود آمده است مانند C، پاسکال، بیسیک و فورترن. این زبان های سطح بالا ساختار نزدیک تری به زبان انسان دارند مانند شرط ها، حلقه ها و مواردی از این دست که در زبان طبیعی انسان وجود دارد. اما حتی با وجود زبان های سطح بالا باز هم برنامه نویسی باید به ساختارهای کامپیوتری فکر کند و همچنان نیز بخشی از تفکر برنامه نویس معطوف به مواردی مانند حافظه، پردازنده، متغیرها در حافظه، آدرس در حافظه و سایر ویژگی های کامپیوتر خواهد بود.

خیلی بهتر است که ما بتوانیم امکانی را به وجود بیاوریم که در آن برنامه نویس بتواند به جای ساختار کامپیوتر، تنها به ساختار مساله ای که قرار است حل بکند فکر بکند و مولفه هایی که در نظر می گیرد بیش از کامپیوتر و ویژگی های آن، معطوف به مساله و ویژگی هایش باشد.

# فضای مساله و فضای راه حل

در دنیای نرم افزار دو فضا وجود دارد، فضای مساله و فضای راه حل، برنامه نویسی همان طور که می دانید ایجاد یک راه حل نرم افزاری برای یک موضوع واقعی است، مثلا موضوعی واقعی مانند مدیریت یک کتابخانه وجود دارد و برنامه نویسی برای این موضوع یعنی نوشتن نرم افزاری برای مدیریت کردن کتاب ها، اعضا، امانت دادن کتاب ها و مواردی از این دست. در این موضوع هم دو فضا وجود دارد، فضای مساله(Problem Space)، یعنی چیزی که می خواهیم آن را حل کنیم و در این جا منظور کتابخانه است، مولفه هایی که در این فضا وجود دارند چه چیزهایی هستند؟ چیزهایی مانند کتاب، عضو، قفسه، امانت داده شدن کتاب و ... . و یک فضای راه حل(Solution Space) وجود دارد، ما می خواهیم برنامه ای برای مدیریت کردن یک کتابخانه بنویسیم، مثلا برنامه ای به زبان جاوا، مولفه های فضای راه حل کتابخانه چه چیزهایی هست؟ در واقع هنگام ایجاد این نرم افزار باید به چه مولفه های نرم افزاری فکر کنیم؟ مولفه هایی مانند پروژه، برنامه، متغیر، تابع و ... .

برنامه نویسی به عبارت دیگر تلاشی است برای انجام یک نگاشت بین فضای مساله و فضای راه حل، یعنی برنامه نویس باید فضای مساله را بفهمد و در فضای راه حل برنامه ای برای این مساله ارائه بکند.

# رویکرد شیءگرا

به مساله ی کتابخانه باز هم فکر کنید، عناصر برنامه شما چه چیزهایی خواهند بود؟ احتمالا درباره توابع و متغیرها فکر خواهید کرد و چیزهایی مانند اینها، رویکر شی گرا نوع دیگری از برنامه نویسی را ترویج می کند، در رویکرد شی گرا(Object Oriented) برنامه نویس اجازه دارد عناصر فضای مساله را در فضای راه حل باز نمایی کند. یعنی دقیقا همان مفاهیم و اصطلاحاتی که در فضای مساله وجود دارد، در برنامه برنامه نویس هم خودشان را نشان می دهند.

*این مفاهیم و موجودیت ها شی نام دارند.*

* شیء (Object)

شی ها موجودیت هایی هستند که در فضای مساله وجود دارند و در رویکرد شی گرا همان موجودیت ها در برنامه یعنی فضای راه حل هم دیده می شوند. مثلا همان طور که ما در فضای مساله از کتاب صحبت می کنیم، در برنامه مان هم از کتاب صحبت می کنیم، در برنامه مان شی کتاب و عضو داریم. بنابر این اشیایی مانند کتاب و عضو و قفسه همان طور که در مساله حضور دارند، در راه حل نیز با همان موجودیت ها سر و کار داریم و برنامه نویسی شی گرا از همین اشیا و روابط آنها تشکیل شده است.

* برنامه نویسی شی گرا (Object Oriented Programming)

این رویکر به ما کمک می کند برنامه هایی بنویسیم که زبان فضای مساله را نشان می دهند. در این رویکرد ما می توانیم انواع داده ی جدیدی ایجاد کنیم، انواع داده ای که در خود زبان وجود ندارند و از این انواع برای نشان دادن اشیای موجود در آن مساله استفاده کنیم. مثلا می توانیم نوع داده ی کتاب و یا قفسه را ایجاد کنیم، اینها مانند سایر انواع داده برای ما قابل استفاده خواهند بود و می توانیم متغیرهایی از جنس آنها در برنامه تعریف و استفاده کنیم. یعنی برنامه ها همان طور که data type هایی مانند Integer و character دارد می تواند از این به بعد Book هم داشته باشد. و وقتی کسی برنامه شی گرا را می خواند، کلمات و عباراتی را خواهد دید که با فضای مساله همخوانی دارند و به این ترتیب زبان برنامه نویسی ما بسیار انعطاف پذیرتر، خواناتر و قوی تر خواهد بود.

* زبان های شی گرا

زبان های شی گرای مختلفی در طول سالیان اخیر معرفی شده اند مانند Smalltalk، C++، C# و البته جاوا. اسمالتاک یکی از اولین زبان های شی گرای موفق و به نحوی الهام بخش زبان جاوا بوده است. توجه داشته باشید که زبان های سطح بالای دیگری هم وجود دارند که شی گرا نیستند، مثلا زبان C زبانی شی گرا نیست و C++ است که زبان C را گسترش داده و امکانات شی گرایی را به آن اضافه کرده است.

# شی گرایی در برابر رویکرد رویه ای

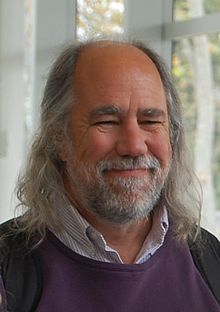
رویکردهای مختلفی برای برنامه نویسی و تولید نرم افزار وجود دارد که شی گرایی یکی از آنهاست. رویکرد رایج دیگری که قبل از شی گرایی در برنامه نویسی سطح بالا پرکاربرد بود، رویکرد رویه ای است. در برنامه نویسی شی گرا اساس برنامه ها و عناصر برنامه ها، اشیاء و تبادل پیام بین آنهاست. در برنامه نویسی رویه ای، مولفه های اصلی برنامه ها توابع، متغیرها و فراخوانی توابع روی متغیرها هستند. همچنین نحوه ی تفکر برنامه نویس در این دو رویکرد هم متفاوت است، برنامه نویسی که برنامه های شی گرا می نویسد، درباره ی اشیا مساله فکر می کند، سعی می کند آنها را شناسایی کند و به ازای هر کدام از آنها یک کلاس در برنامه ایجاد کند، اما در رویکرد رویه ای (Procedural) نوع تفکر به نحوی سطح پایین تر است و برنامه نویس درگیر ساختارهای کامپیوتری، حافظه و نحوه ی آدرس دهی خواهد بود که کمتر در برنامه نویسی شی گرا دغدغه و درگیری ذهنی برنامه نویس را تشکیل می دهند.

تا قبل از این فصل ما عملا با رویکرد رویه ای برنامه نویسی می کردیم چون هنوز اشیاء را یاد نگرفته بودیم اما از این پس کم کم با رویکرد شی گرا آشنا خواهیم شد.

# مشخصات برنامه نویسی شی گرا

افراد مختلف، توصیف های مختلفی درباره اشیاء و یا برنامه نویسی شی گرا کرده اند. ما با دو نقل قول این موضوع را واکاوی می کنیم. الن کی که آدم مهمی در حوزه ی برنامه نویسی شی گرا بوده، این حوزه را در پنج ویژگی خلاصه می کند: (در واقع ایشان برنامه نویسی با کمک اسمالتاک را با این 5 ویژگی توصیف کرده است ولی این توصیف برای خیلی از برنامه های شی گرا مثلا خیلی از برنامه هایی که به زبان جاوا نوشته می شود صادق است)

* در برنامه های شی گرا هر چیز یک شی است.
* یک برنامه مجموعه ای از اشیاء است، که این اشیاء با ارسال پیام با همدیگر ارتباط برقرار می کنند و از این طریق هر شی به سایر اشیاء می گوید که چه کار از آنها می خواهد.
* هر شی حافظه ی مخصوص به خودش را دارد و در واقع حافظه ی یک شیء می تواند از اشیاء دیگری تشکیل شده باشد، مثلا حافظه ی شی از جنس کتابخانه از اشیاء دیگری مانند نام و کد عضویت و ... تشکیل شده است. مجموعه ی این اشیاء تشکیل دهنده و مقدار آنها حالت شی (state) را مشخص می کند.
* هر شیء دارای یک نوع است، انواع مختلفی از اشیاء در برنامه نویسی شی گرا قابل تعریف است، مثلا علی علوی یک شی در حوزه ی مدیریت کتابخانه هست که نوع عضو دارد. خلاصه اشیاء مختلف هر کدام نوعی دارند.
* همه ی اشیاءی که هم نوع هستند رفتار مشابهی دارند، یعنی می توانند پیام های مشابهی دریافت کنند.

از یک جنبه ی دیگر، آقای بوچ که او نیز فرد مهمی در حوزه ی شی گرایی هست، توصیف دیگری از اشیاء دارد و آنها را به گونه ی دیگری تعریف می کند:

* بوچ می گوید که یک شی مجموعه ای از وضعیت، رفتار و هویت است. (state, behavior, identity)
* آقای بوچ در تعریف خودش "هویت" را به تعریف شی اضافه و روی آن تاکید کرده است.
* هر شی ویژگی هایی دارد، مثلا عضو کتابخانه ویژگی هایی مانند اسم و سن دارد. یا کتاب ویژگی هایی مانند تعداد صفحات و یا عنوان دارد و ویژگی هایی که هر شی دارد حالت و وضعیت آن شی را مشخص می کند. پس هر شی وضعیتی دارد که از مجموعه ویژگی های آن شی ناشی می شود.
* هر شی رفتاری (متدهایی) دارد. در واقع می تواند با کمک متدها رفتاری از خودش نشان بدهد. مثلا رفتار امانت گرفتن کتاب برای شی عضو کتابخانه وجود دارد، پس امانت گرفتن کتاب می تواند متدی باشد برای عضو کتابخانه.
* و هر شی منحصر به فرد است، یک شی با یک شی دیگر فرق می کند و هویتی متمایز دارد. حتی اگر دو شی وضعیت یکسان داشته باشند، یعنی دقیقا تک تک ویژگی های آنها با هم مساوی باشد، هویت متفاوتی دارند، مثلا اگر دو عضو در کتابخانه داشته باشیم با نام علی علوی، حتی اگر سن یکسان داشته باشند، هویت یکسانی ندارند، و هر کدام از این اشیاء در حافظه در مکان مشخصی قرار می گیرند و شاید بتوانیم بگوییم که مکان اشیاء در حافظه نشان دهنده ی هویت آنهاست.

# مفاهیم برنامه نویسی شی گرا

در این قسمت می خواهیم در مورد مفاهیم حوزه برنامه نویسی شی گرا صحبت کنیم. اولین و یا مهم ترین مفهوم شی است:

* شی (object)

شی ها موجودیت هایی هستند که در فضای مساله دیده می شوند. مثلا در فضای مساله ی کتابخانه، شی کتاب شاهنامه به عنوان یک کتاب و یا آقای علی علوی به عنوان یک عضو اشیاء مساله ی ما هستند.

یا برای مثال در نوشتن بازی فوتبال، علی کریمی یک شی است، فرهاد مجیدی یک شی است، ورزشگاه آزادی و یا خود توپ نیز اشیاء دیگر مساله ی ما هستند.

یا در نوشتن یک برنامه بانکی، مثلا خانم مریم علوی به عنوان مشتری بانک یک شی است، یا شعبه ی آزادی به عنوان یکی از شعبه های بانک یک شی است.

بنابراین ما اشیاءی در مساله می بینیم که در برنامه های ما نیز حضور دارند و به آنها Object گفته می شود.

* کلاس (class)

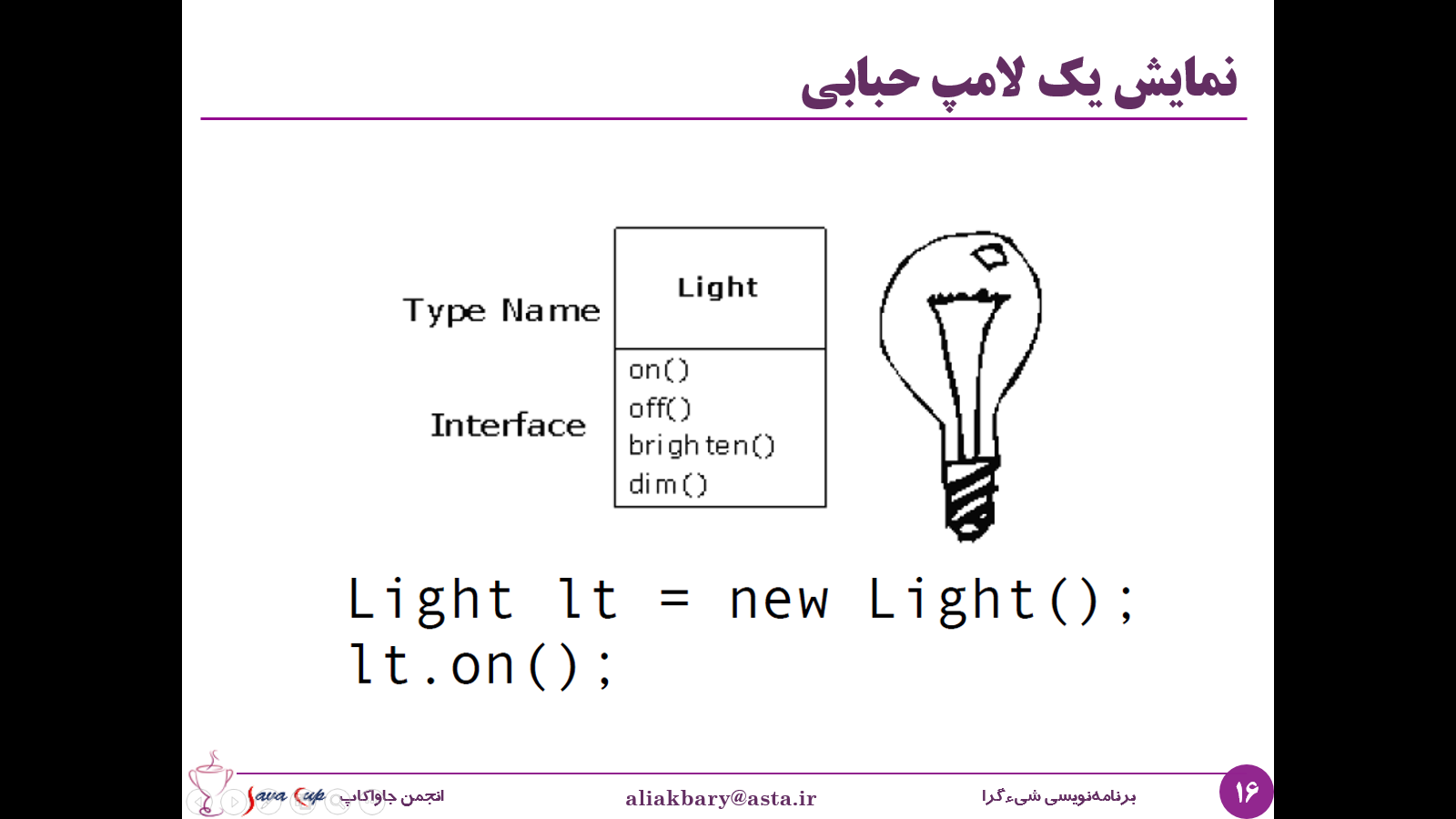
مفهوم بعدی کلاس هست. کلاس نوع یا رده یا دسته ای از اشیاء را نشان می دهد که رفتار و ویژگی های مشابه دارند. هر کلاس می تواند نمونه های مختلفی از اشیاء را داشته باشد، مثلا در فضای کتابخانه، کتاب یک کلاس و کتاب شاهنامه یک نمونه یا یک شیء از این کلاس است، به همین ترتیب عضو کتابخانه و یا قفسه ی کتاب کلاس هستند و می توانند اشیاء و نمونه هایی داشته باشند.

یا در بازی فوتبال کلاس ها کدام هستند؟ کلاس بازیکن، کلاس تیم فوتبال، کلاس ورزشگاه، که باز هم هر کدام از اینها می توانند نمونه هایی داشته باشند.

و در برنامه بانکی، کلاس مشتری، شعبه، حساب و ...

هر کلاس رفتار و ویژگی هایی را تعریف می کند به این معنی که مشخص می کند که اشایی که از این کلاس ایجاد می شوند چه رفتار(Behavior) و متدهایی می توانند داشته باشند و همچنین چه ویژگی هایی(Property) دارند، و هر شی از آن کلاس آن رفتارها و ویژگی ها را خواهد داشت، مثلا در مساله ی کتابخانه، عضو کتابخانه که یک کلاس هست مشخص می کند که ویژگی های این کلاس چه هستند، مثلا نام، سن و شماره عضویت. یعنی هر شی از نوع عضو کتابخانه که ایجاد بشود باید نام، سن و شغل داشته باشد، مثلا "علی علوی" با سن 20 سال و شماره عضویت 956432 . همچنین هر کلاس مشخص می کند که اشیاء آن کلاس چه رفتارهایی دارند مثلا عضو کتابخانه که یک کلاس هست رفتار به امانت گرفتن کتاب را دارد پس "علی علوی" نیز این رفتار را خواهد داشت.

حال بگذارید این مفاهیم را یک بار هم به صورت شماتیک به همراه کد جاوا مرور کنیم، فرض کنید می خواهیم برنامه ای برای یک لامپ بنویسیم، لامپی که می تواند روشن بشود، خاموش بشود، و یا نور آن کم و زیاد شود. نوع داده در اینجا نوع لامپ است که ما در اینجا نام آن را Light گذاشتیم، این کلاس رفتارهایی که گفتیم را پیاده سازی کرده است و متدهایی برای هر کدام از آنها دارد، ما می توانیم از روی این کلاس نمونه هایی بسازیم، مثلا یک نمونه به نام lt بسازیم که یک شیء جدید از این کلاس است:



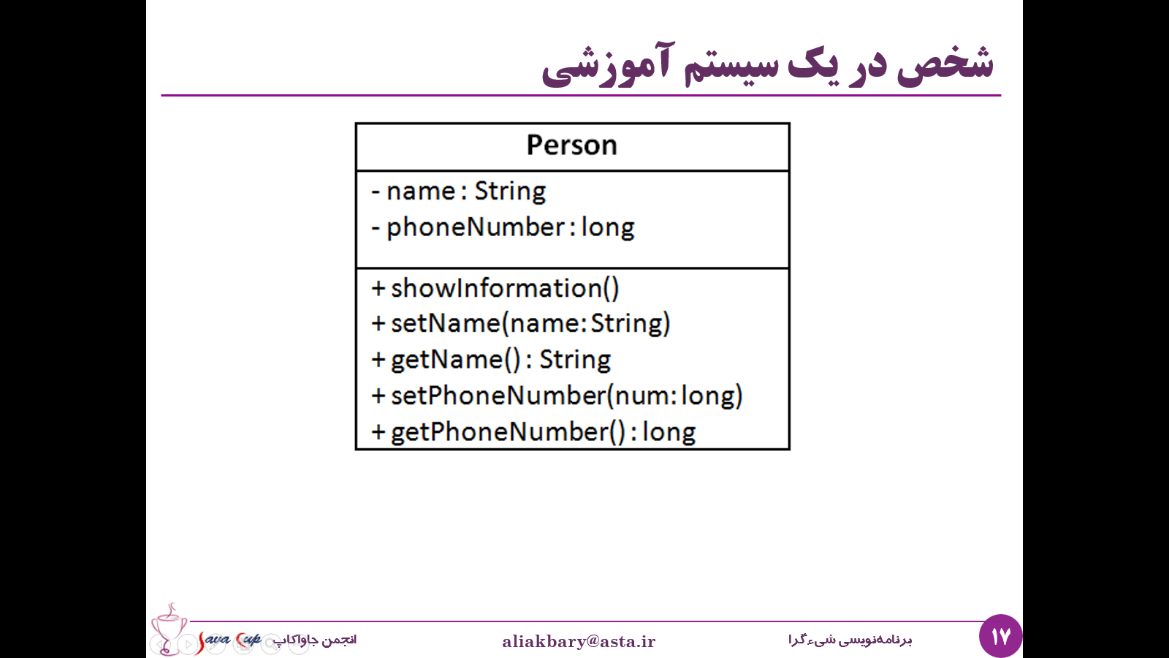
در این کد که جزئیات آن را بعدا بیشتر خواهیم دید یک متغیر از جنس کلاس Light ایجاد شده و یک نمونه از این کلاس در آن new شده است. حال این متغیر می تواند متدهای این کلاس را فراخوانی کند، مثلا می توانیم بگوییم:

lt.on();

و با این کار رفتار روشن شدن را فراخوانی و اجرا کنیم. به این نکته دقت کنید که کلاس لامپ عنوانی دارد که به آن نام نوع داده (type name) و یک واسط (interface) دارد که مشخص می کند چه امکانات، رفتارها و متدهایی برای این کلاس وجود دارد.

همچنین دیدیم که با کمک عملگر new توانستیم یک شی از کلاس مورد نظرمان بسازیم و با استفاده از نقطه بعد از نام شی می توانیم متدهای آن را فراخوانی کنیم.

مستطیلی که در بالا دیدیم و شامل نام کلاس و ویژگی ها و امضای متدهای کلاس هست را دیاگرام کلاس یا UML Class Diagram می نامند که در مورد آن نیز بعدها بیشتر صحبت خواهیم کرد.

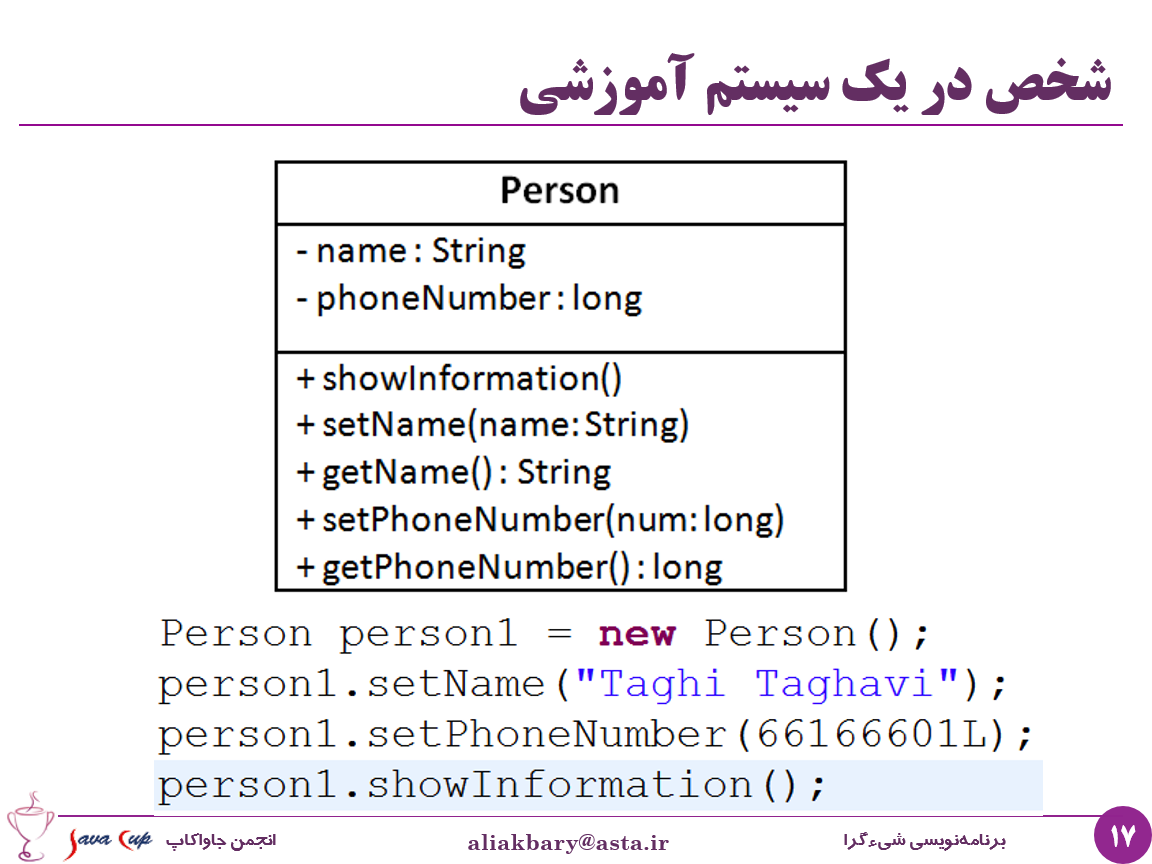
یا مثالی دیگر، فرض کنید که می خواهیم یک سیستم آموزشی بنویسیم، هر شخص در این سیستم یک نمونه از کلاس Person خواهد بود که می تواند ویژگی ها و متدهایی مانند زیر داشته باشد:

در اینجا نیز ما دیاگرام کلاس این موجودیت را رسم کرده ایم، این کلاس ویژگی هایی مانند اسم و شماره تلفن و رفتارهایی مثل نمایش دادن اطلاعات شخص، مشخص کردن اسم شخص، دریافت کردن اسم شخص و متدهای مشابهی برای دریافت و خروجی دادن شماره تلفن دارد.

چطور می شود از کلاس Person استفاده کرد؟ با فرض اینکه این کلاس به نحوی تعریف و پیاده سازی شده باشد ما می توانیم با استفاده از عملگر new از روی آن شی بسازیم:

Person person1 = new Person();

و به همین ترتیب می توانیم رفتارهای این کلاس را روی شی ایجاد شده خودمان فراخوانی کنیم:



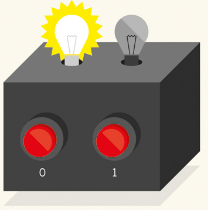
با این فراخوانی ها ابتدا نام شی ما مشخص می شود، سپس شماره تلفن آن را مقدار دهی می کنیم و در نهایت اطلاعات آن را نمایش می دهیم. و تمام این متدها رفتارهایی هستند که در کلاس Person تعریف و پیاده سازی شده اند.

# کنترل دسترسی

ما در هنگام طراحی کردن کلاس ها می توانیم دسترسی به بخش هایی از کلاس را محدود کنیم. هنوز وارد نحوه ی تعریف کردن کلاس ها نشده ایم اما این مفاهیم را فعلا می توانیم به صورت کلی مرور کنیم. در واقع در طراحی هر کلاس ما می توانیم بخش هایی عمومی داشته باشیم (Public) و بخش های خصوصی (Private)، کسی که می خواهد از کلاس ما استفاده بکند، فقط می تواند از بخش های عمومی آن استفاده کند. بخش هایی که به صورت عمومی تعریف کرده ایم اصطلاحا واسط کلاس (Interface) را تشکیل می دهند. پس واسط کلاس بخش هایی از کلاس است که توسط استفاده کننده قابل بهره برداری باشد. بخش های خصوصی داخل همان کلاس قابل استفاده هستند و لزوما توسط سایر کلاس ها قابل استفاده نخواهند بود، بخش های خصوصی عملا از دید استفاده کنندگان پنهان شده است، به این ویژگی Implementation Hiding یا پیاده سازی پنهان و یا پنهان سازی پیاده سازی گفته می شود که باعث می شود استفاده کننده و کاربر کلاس همه ی بخش های آن را نبیند و از جزئیات پیاده سازی آگاه نباشد.

# محصورسازی

محصورسازی (Encapsulation) یا لفافه بندی به معنای تعریف ویژگی ها و رفتارهای کلیدی در عین پنهان کردن بخشی از پیاده سازی کلاس است. لفاف بندی در برنامه نویسی شی گرا در قالب تعریف کلاس برای اشیاء نمود پیدا می کند. وقتی که ما یک کلاس می نویسیم و مشخص می کنیم که کدام بخش های آن public و کدام بخش ها private هستند، عملا محصورسازی انجام داده ایم. به زبان دیگر با تعریف کلاس هایی که برای آنها واسط هایی عمومی تعریف شده اما بخش هایی از پیاده سازی آنها از دید کاربران و استفاده کنندگان کلاس پنهان است محصورسازی کرده ایم.

یا به صورت مختصرتر محصورسازی یعنی تعریف نوع و واسط کلاس (نحوه ی استفاده از آن)، مثلا وقتی ما می گوییم نوع کتاب وجود دارد که بخش هایی از آن عمومی (public) و بخش هایی خصوصی (private) هستند در حال محصورسازی هستیم.

محصورسازی باعث می شود که محصول ما یعنی کلاسی که داریم ایجاد می کنیم، مانند یک جعبه ی سیاه (Black Box) باشد یعنی بخش هایی از جزئیات داخل آن برای استفاده کننده نامشخص باشد و ما با تعریف واسط (Interface) مشخص می کنیم که این جعبه چه دکمه های داشته باشد و در ازای استفاده از آنها چه عملکردی از خود نشان دهد.

این نوع طراحی در خیلی از محصولات تجاری هست و محدود به محصولات نرم افزاری نیست، مثلا وقتی یک تلوزیون طراحی می شود، جزئیات پیاده سازی و سیم کشی ها و آی سی های داخل آن از دید استفاده کننده پنهان می شه و استفاده کننده فقط دکمه هایی را می بیند و می تواند از آنها استفاده کند. یا تلفن همراه شما جزئیات مفصل سخت افزاری دارد که از دید شما پنهان است و شما فقط از طریق واسط کاربری می توانید از آن استفاده کنید.

* چرا محصورسازی؟

چرا محصورسازی (Encapsulation) در زمینه تولید نرم افزار کار خوبی است؟ به هر حال ما می توانیم جزئیات را به کاربر و استفاده کننده هم نشان دهیم در حالی که ما آنها را پنهان می کنیم و در عوض یک واسط ساده در اختیار کاربر قرار می دهیم. محصورسازی باعث می شود که بهره برداری و استفاده از کلاس آسان تر شود، برای مثال استفاده از ماوسی که جزئیاتی ندارد خیلی آسان تر از ابزاری خواهد بود که نیاز است با تمام جزئیات آن آشنا شویم.



در واقع وقتی جزئیات پیاده سازی را از دید کاربر پنهان می کنیم، او می تواند راحت تر و سریع تر محصول ما (کلاس ما) را درک و از آن استفاده کند. از دید دیگر پیاده سازی باز و غیر پنهان ممکن است باعث شود که کاربر از محصول ما به اشتباه استفاده کند پس محصورسازی احتمال بروز خطا را نیز کاهش می دهد. و در نهایت محصورسازی باعث می شود که کلاس شما زیباتر به نظر برسد و کاربر با آن راحت تر باشد.

# واسط

و در آخر مفهوم واسط را با هم مرور می کنیم. واسط (Interface) مشخص می کند که هر شی چه رفتارهایی می تواند از خودش نشان بدهد. در واقع واسط یک شی درخواست ها و پیام هایی هست که آن شی می تواند به آنها پاسخ بدهد. مثلا برای کلاس کتاب، ما می توانیم به عنوان واسط امکان امانت گرفته شدن را قرار دهیم، یعنی این امکان را فراهم کنیم که کاربر بتواند رفتار امانت گرفته شدن را روی کلاس کتاب فرخوانی کند. تا هر شی از نوع کتاب بتواند این رفتار را از خود نشان دهد، مثلا با این طراحی کتاب شاهنامه نیز از این رفتار پشتیبانی خواهد کرد.

واسط هر شی در کلاس آن شی مشخص می شود، پس در مثال قبل باید رفتار امانت گرفته شدن در کلاس کتاب نوشته و تعریف شود.

نکته: واسط (Interface) در زبان جاوا معنایی خاص نیز دارد که ما در آینده با آن آشنا خواهیم شد.

در واقع ما تا این بخش از کتاب به صورت خیلی کلی با مفاهیم اصلی شی گرایی آشنا شدیم و در ادامه این جلسه و نیز جلسات بعد کم کم دقیق تر به آنها خواهیم پرداخت و می آموزیم که چگونه آنها را در نوشتن برنامه به زبان جاوا به کار ببریم.

# تعریف اولین کلاس

* نحوه ی تعریف اولین کلاس

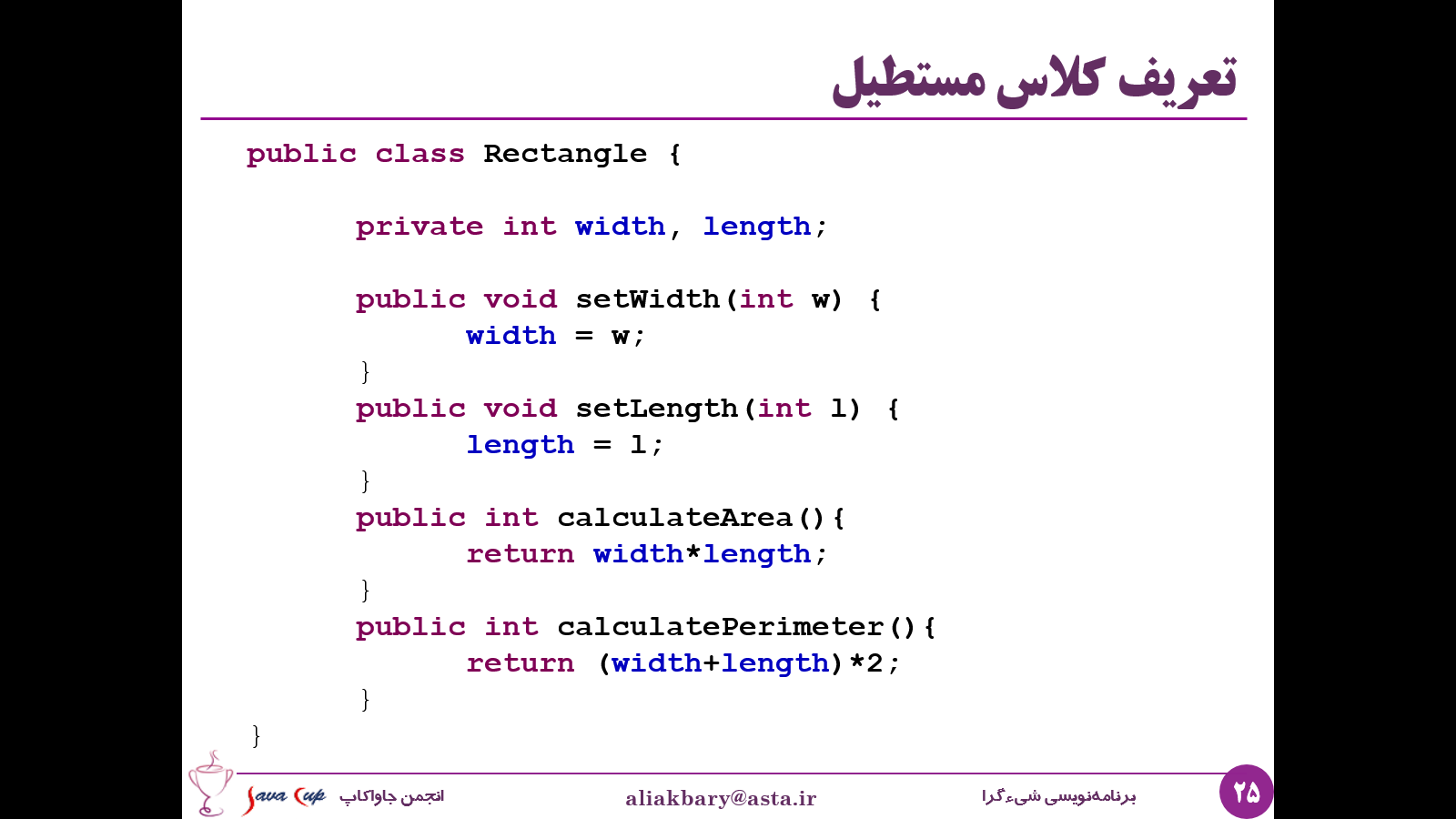
بعد از اینکه برخی از مفاهیم حوزه ی شی گرایی را با هم مرور کردیم، برگردیم به زبان جاوا و ببینیم در این زبان چگونه می توانیم این مفاهیم را پیاده سازی کنیم. همه ی مفاهیمی که دیدیم در همه ی زبان های شی گرا معنا دارند و از این جا به بعد می خواهیم ببینیم که چطور در جاوا از آنها استفاده کنیم.

اولین قدم ما برای برنامه نویسی شی گرا تعریف کلاس است، برای این کار ما باید کلاس های اشیاء را تشخیص بدهیم و برای هر کدام از آنها یک کلاس در برنامه جاوای خودمان ایجاد کنیم. سپس باید برای هر کلاس یک نام یا عنوان مشخص کنیم، سپس باید مشخص کنیم که این کلاس و در واقع هر شی ای که از این کلاس ایجاد می شود چه ویژگی هایی (Property) دارد. مثلا اگر می خواهیم کلاسی برای کتاب ایجاد کنیم، باید فکر کنیم که هر نمونه کتاب، چه ویژگی هایی دارد. هر ویژگی به شکل یک متغیر در کلاس تعریف خواهد شد. در ادامه باید مشخص کنیم که این کلاس چه کارکردهایی (Method) دارد. در این قسمت باید به این فکر کنیم که هر شی که از این کلاس ایجاد می شود چه کارهایی باید بتواند انجام بدهد، چه پیام هایی را بتواند بپذیرد، چه متدهایی قرار است روی آن فراخوانی شوند، هر کدام از این کارکردها به شکل یک متد داخل کلاس تعریف می شود.

البته برای اینکه محصورسازی کامل شود باید مشخص کنیم که هر کدام از این ویژگی ها و کارکردها عمومی هستند یا خصوصی، آیا استفاده کنندگان از این کلاس می توانند از این ویژگی ها و کارکردها استفاده کنند یا اینکه این ویژگی ها تنها داخل خود کلاس معنی دارند.

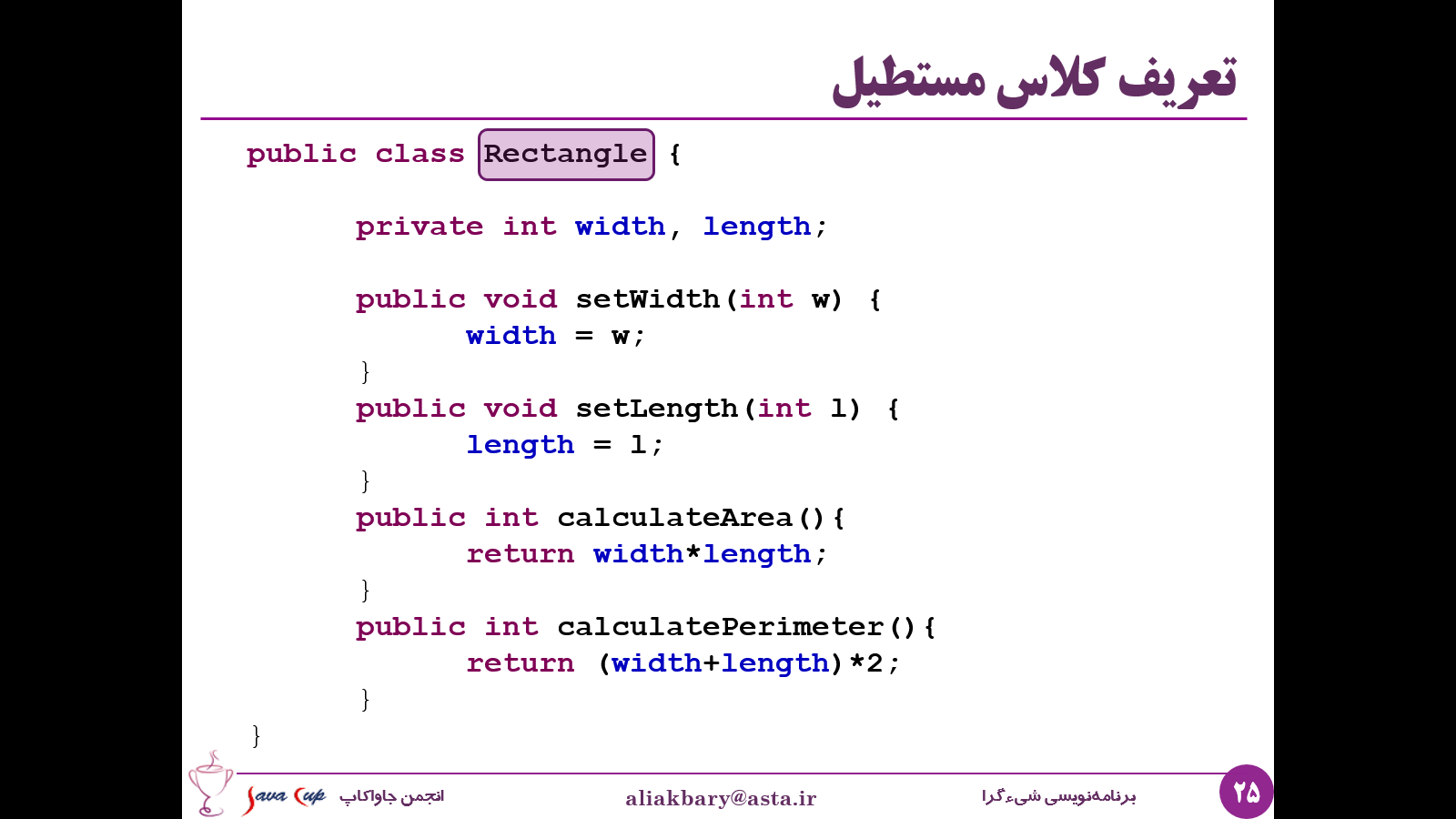
# مثال اول: مستطیل

به عنوان مثال فرض کنیم می خواهیم یک نوع موجودیت برای مستطیل بسازیم، مستطیل چیست؟ هر مستطیل یک شی است دارای طول و عرض (ویژگی ها) و نیز می توان برای هر مستطیل محیط و مساحت را محاسبه کرد (رفتار) و یا طول و عرض آن را تعیین کرد و تغییر داد (رفتار). حالا این مستطیل را در جاوا چطور تعریف کنیم؟

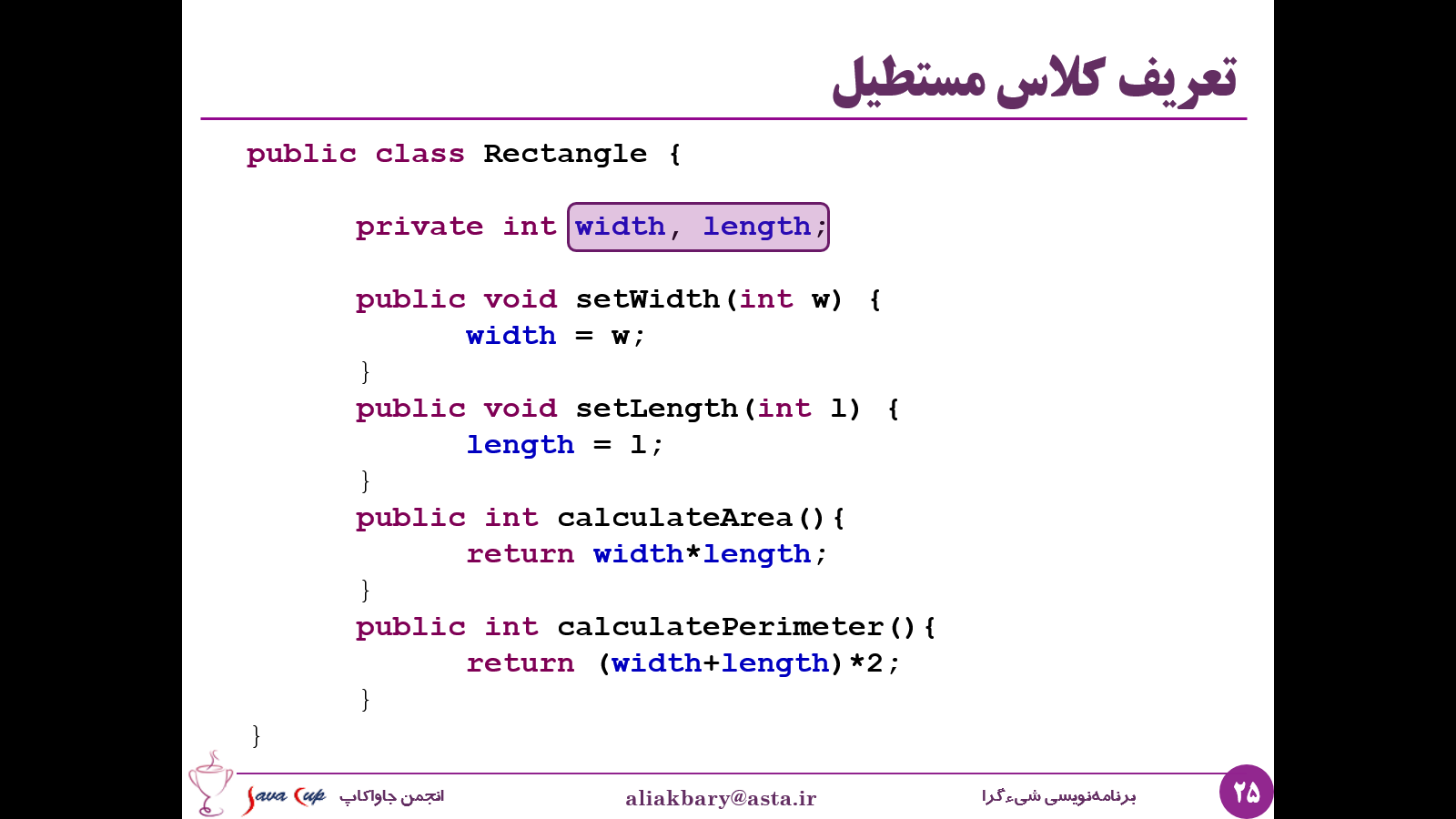


در خط اول همان طور که می بینید نام کلاس را نوشته ایم، سپس ویژگی های کلاس مانند طول و عرض را داخل آکولادی که محدوده ی کلاس ما را مشخص می کنند قرار داده ایم و در نهایت متدهایی که رفتارهای کلاس ما را تعریف می کنند را نوشته ایم. برخی از این رفتارها void هستند یعنی مقدار را برنمی گردانند و برخی از آنها عددی را به عنوان خروجی و نتیجه ی اجرای متد برمی گردانند. ما با تعریف این کلاس، نوع داده ی مستطیل را تعریف کرده ایم و با کمک آن می توانیم نمونه های مختلف مستطیل را ایجاد کنیم. یک بار دوره کنیم که این کلاس چه بخش هایی دارد:

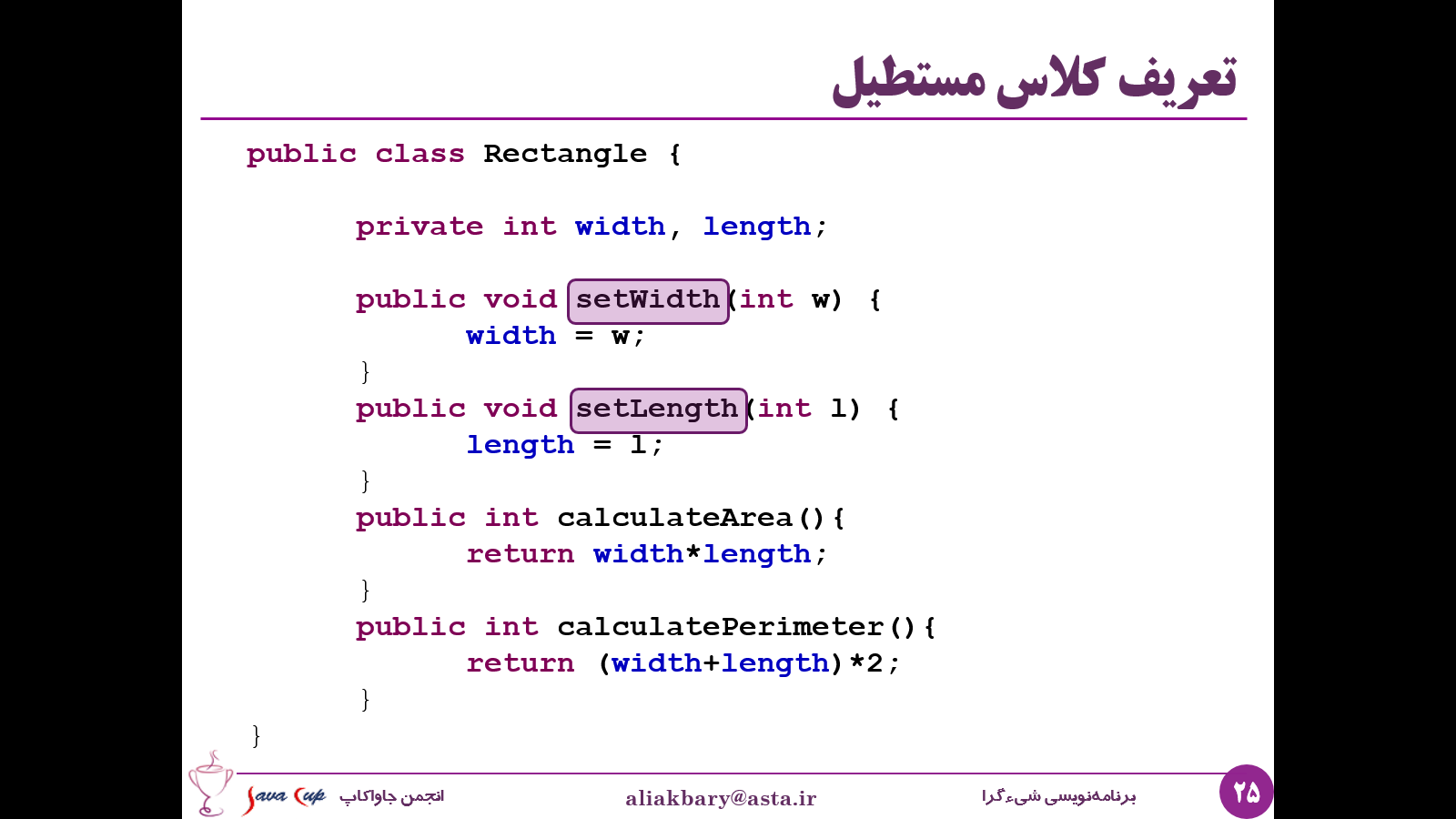
* این کلاس نامی دارد:



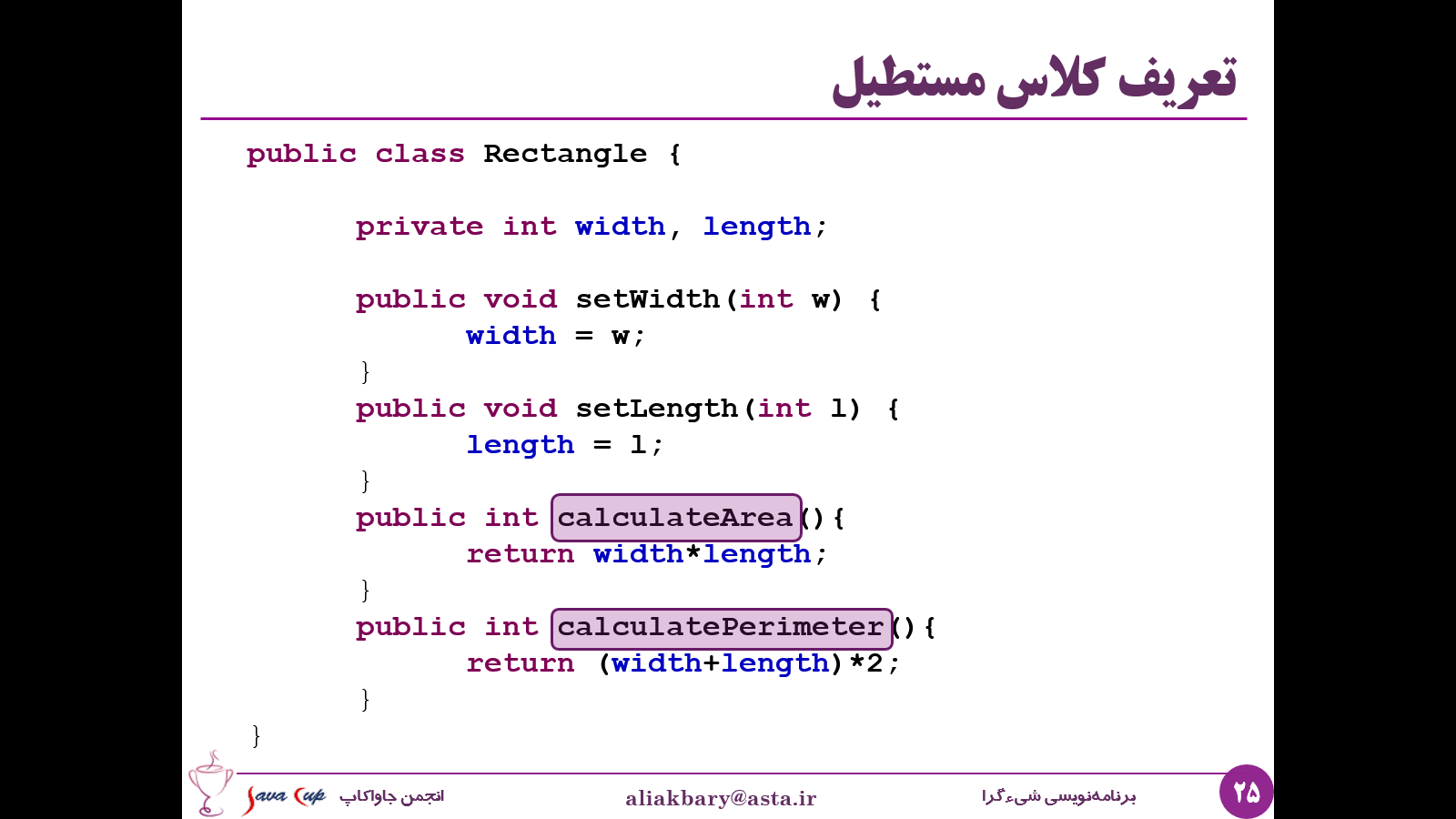
* ویژگی هایی دارد:



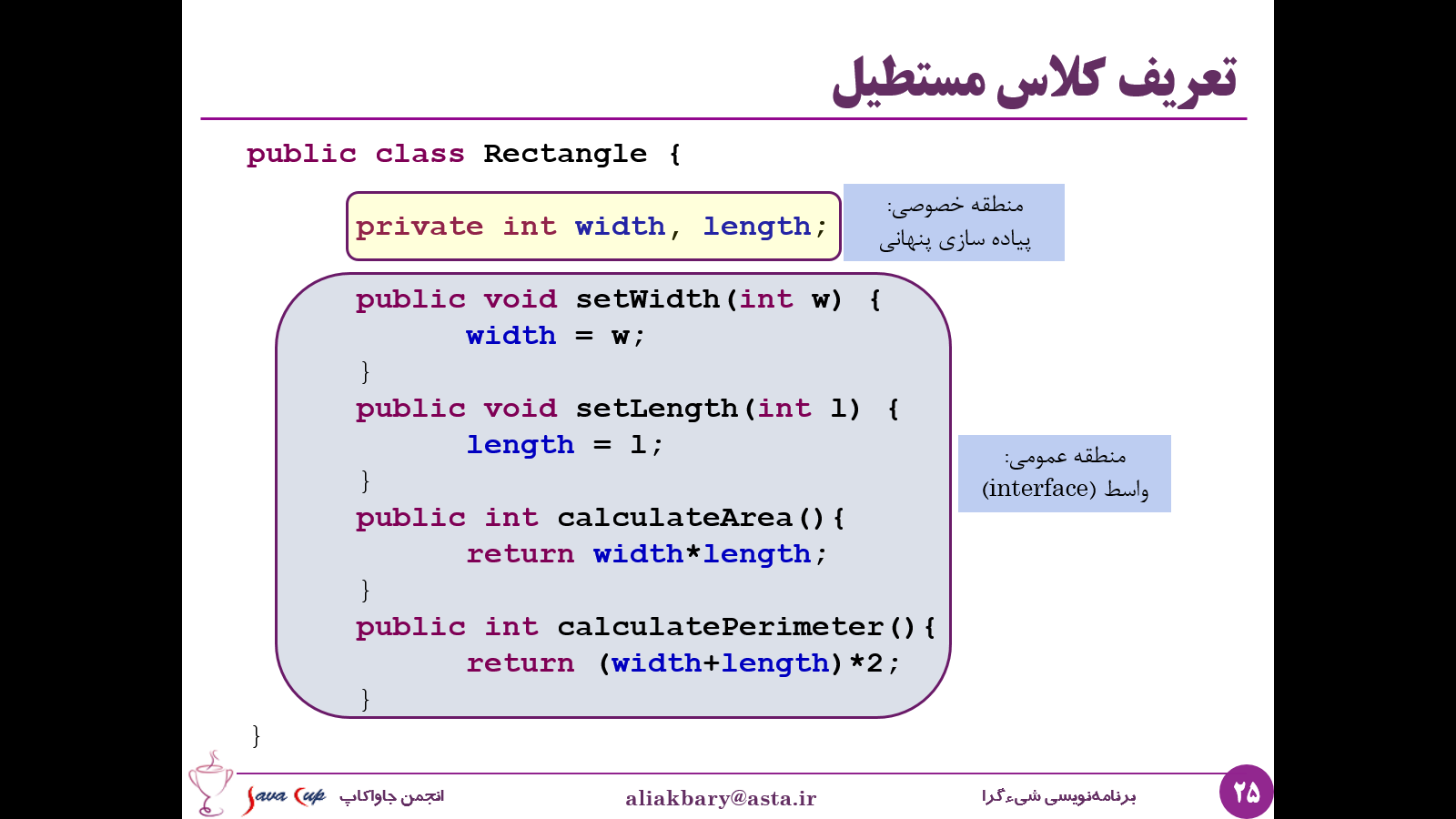
* متدهایی برای مقدار دهی ویژگی های آن تعریف شده است:



* و متدهای دیگری که دیگر رفتارهای این کلاس را مدل می کنند:



* همچنین این کلاس بخش هایی دارد که عمومی هستند و بخش هایی که خصوصی تعریف شده اند:

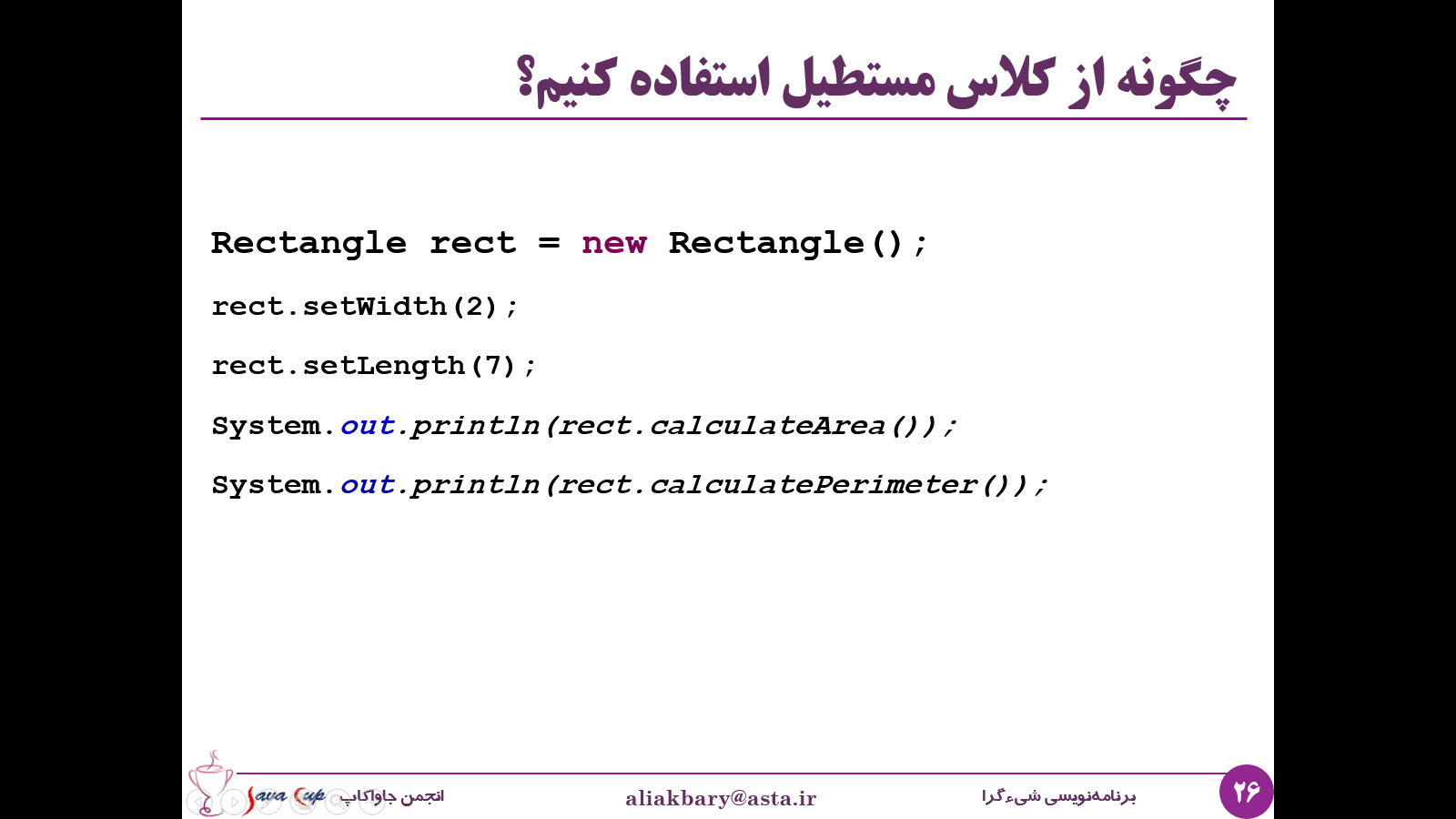


هر ویژگی و متد می تواند public و یا private تعریف شود، بخش های خصوصی از دید کاربر کلاس پنهان خواهد بود و با کمک آنها ما Implementation hiding انجام می دهیم و بخش هایی که public هستند منطقه ی عمومی ما خواهند بود که واسط استفاده از کلاس را تشکیل می دهند و مشخص می کنند که چطور می توان از این کلاس استفاده کرد.

# چگونه از کلاس مستطیل استفاده کنیم؟

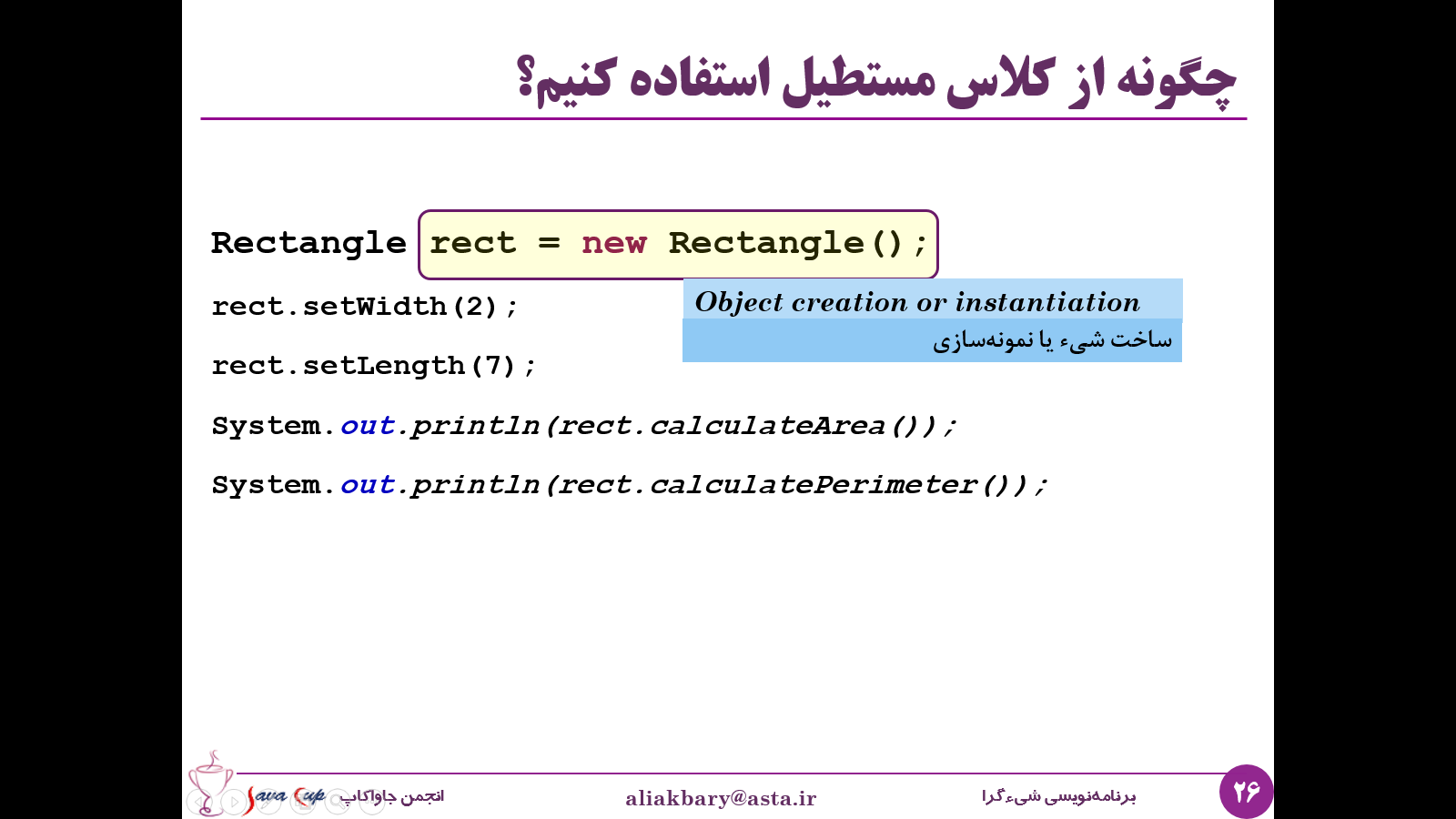
در قسمت قبل کلاس مستطیل را پیاده سازی کردیم، حال می خواهیم ببینیم که چگونه می توانیم از این کلاس استفاده کنیم؟ یا به صورت دقیق تر چطور اشیایی از این کلاس ایجاد کنیم؟

همان طور که قبلا هم دیده ایم می توانیم با کمک دستور new می توانیم نمونه ای از یک کلاس را ایجاد کنیم:

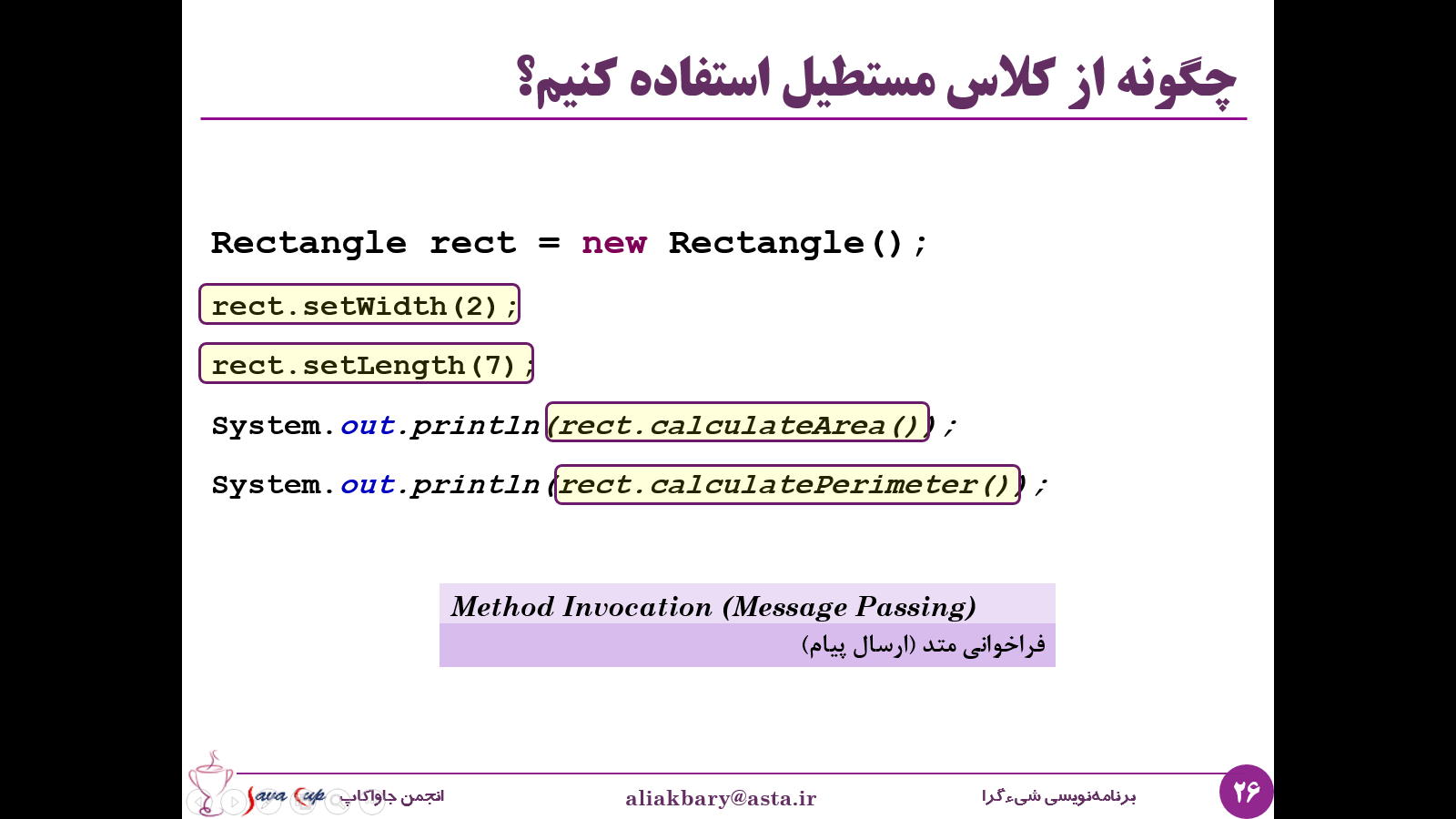


پس در خط اول این کد نمونه ای (شی ای) از کلاس مستطیل ایجاد کرده ایم. این شی حالا می تواند تمام رفتارهایی که در کلاس مستطیل تعریف کرده بودیم را از خود نشان دهد، مثلا rect.setWidth(2) عرض مستطیل ما را برابر 2 قرار می دهد و یا rect.calculateArea() مساحت مستطیل را برای ما محاسبه می کند و برمی گرداند. پس ما نکات زیر را در این مثلا یاد گرفتیم:

* ایجاد کردن شی از کلاسی که پیاده سازی کرده ایم که به آن Object Creation و یا Instantiation می گویند:



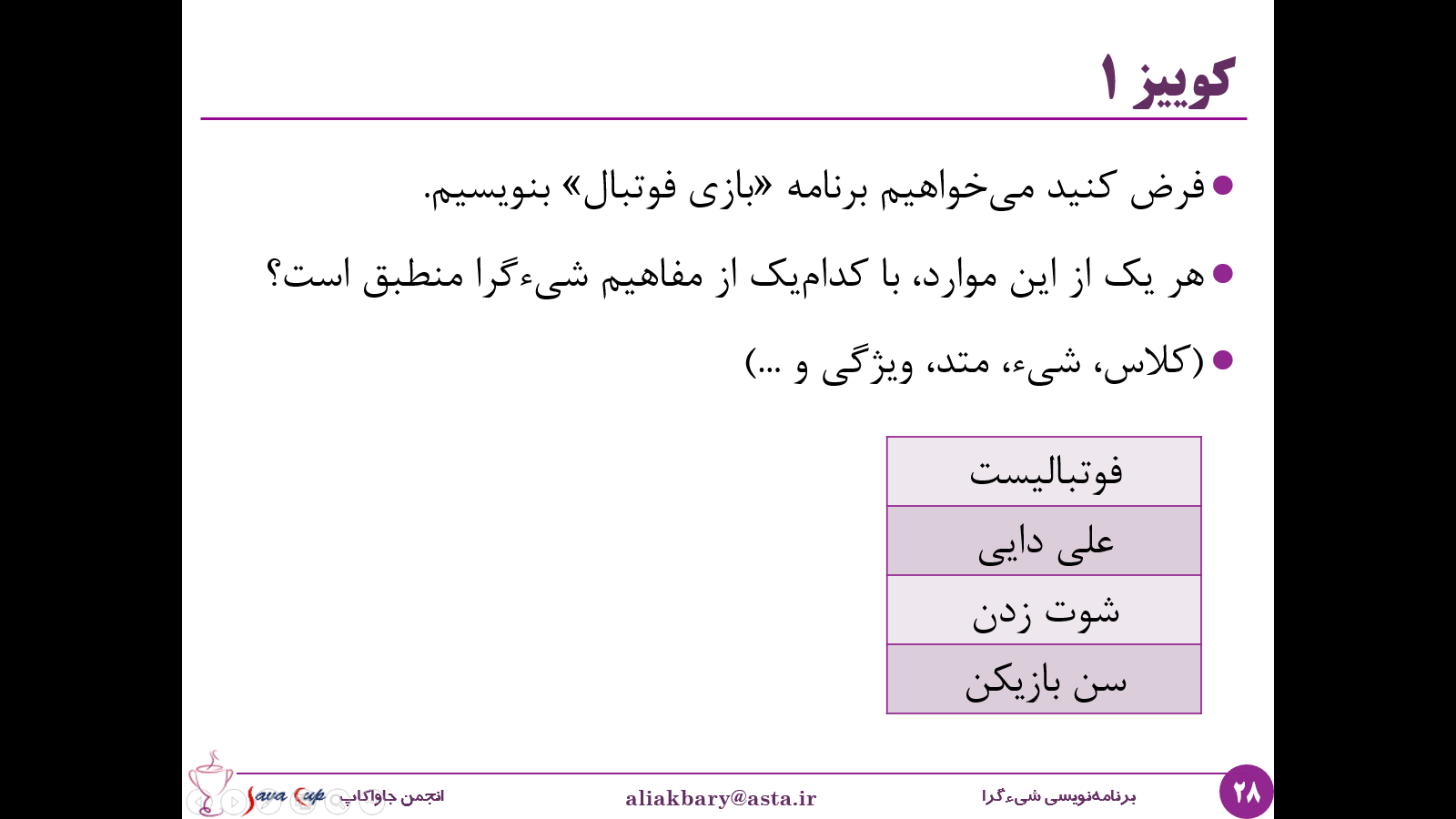
* ارسال دستور و پیام به شی و فراخوانی متدهای آن که به این کار Message Passing و یا Method Invocation گفته می شود:



# کویز

برای مرور مفاهیم شی گرا که با همدیگر دیدیم، یک کویز ساده را با هم مرور می کنیم:

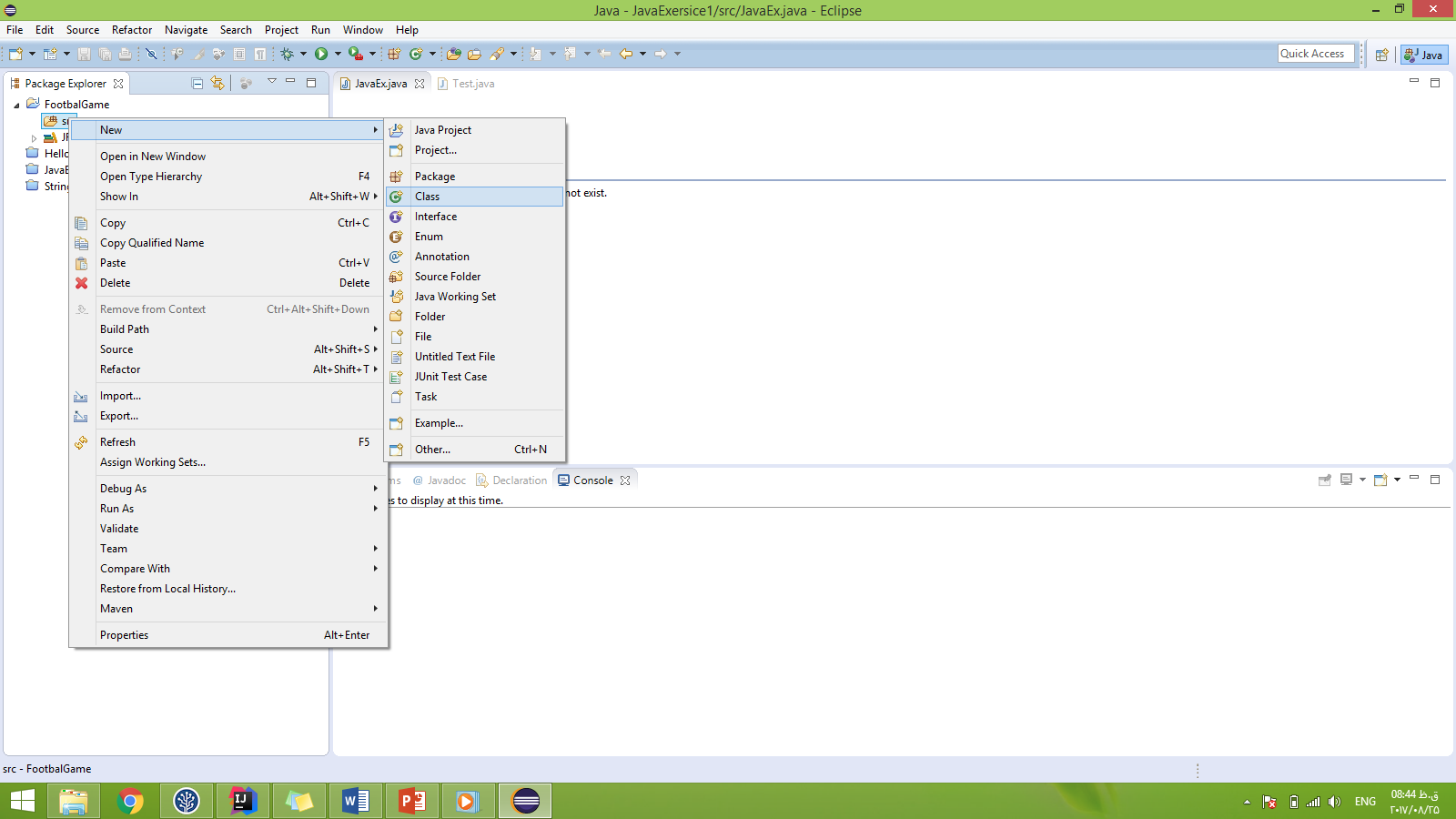
فرض کنید می خواهیم برنامه ای برای بازی فوتبال بنویسیم، هر یک مفاهیمی که در جدول زیر ذکر شده است، چه مفهومی از مفاهیم شی گرا را بازنمایی می کنند؟ مثلا فوتبالیست و یا شوت زدن، هر کدام از اینها چه هستند؟ کلاس؟ شی؟ متد؟ و یا ویژگی؟



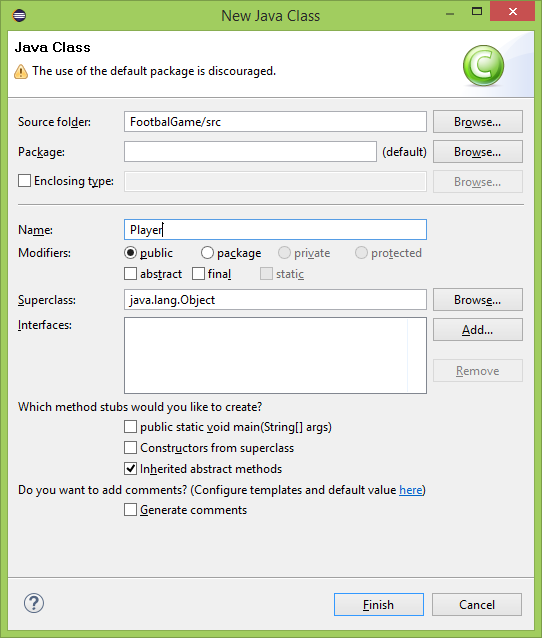
خب، فوتبالیست یک کلاس هست که می تواند نمونه های مختلفی داشته باشد، یکی از این نمونه ها علی دایی هست، پس علی دایی یک شی است، شی ای که از روی کلاس فوتبالیست ایجاد شده است. شوت زدن یک متد و رفتار است که هر شی از نوع فوتبالیست می تواند از خودش نشان دهد، پس رفتار شوت زدن در کلاس فوتبالیست تعریف شده و هر شی از کلاس فوتبالیست دارای این رفتار است و می تواند آن را فراخوانی کند، مثلا علی دایی می تواند رفتار شوت زدن را فراخوانی کند. رفتارهای دیگری هم می توانند در کلاس فوتبالیست تعریف شده باشند، مانند دویدن و ایستادن. و سن بازیکن یک ویژگی است، ویژگی که در کلاس فوتبالیست تعریف شده و هر Object از این کلاس این ویژگی را دارد و آن را مقدار دهی می کند، مثلا علی دایی چون یک شی از کلاس فوتبالیست است پس ویژگی سن را داراست و علی کریمی یک ویژگی سن دیگر مخصوص به خود دارد.

# تمرین عملی

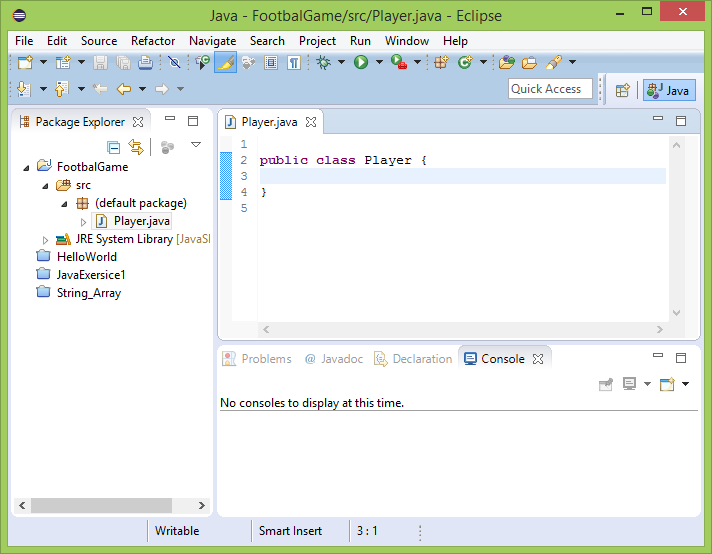
در این بخش مفاهیم را مجددا به صورت عملی تمرین می کنیم، در پروژه ای که در محیط اکلیپس داریم از طرق کلیک راست روی src پروژه و منوی new -> class یک کلاس جدید ایجاد می کنیم:



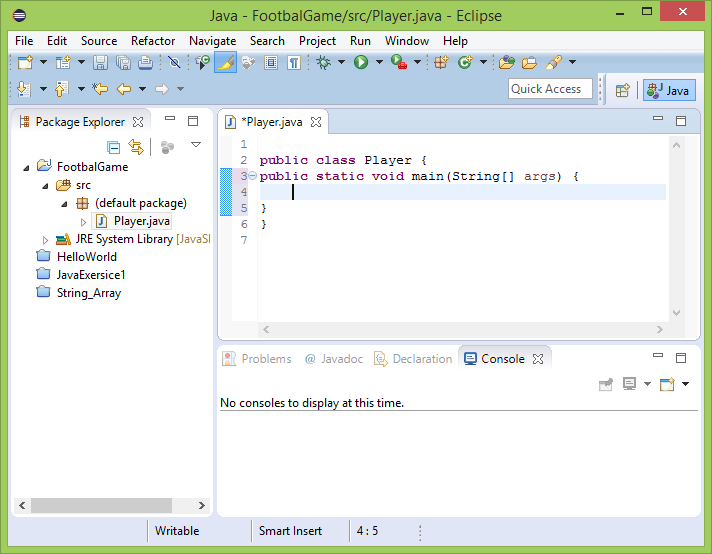
و نام این کلاس را بازیکن می گذاریم:



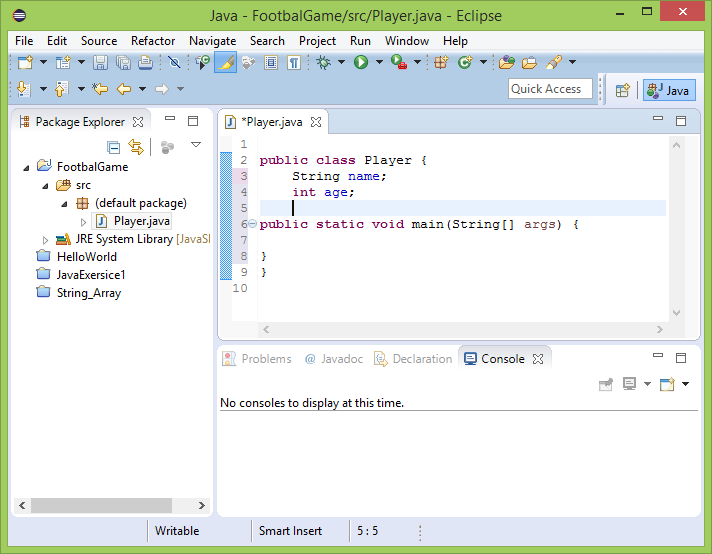
به این ترتیب کلاس جدیدی ساخته می شود:



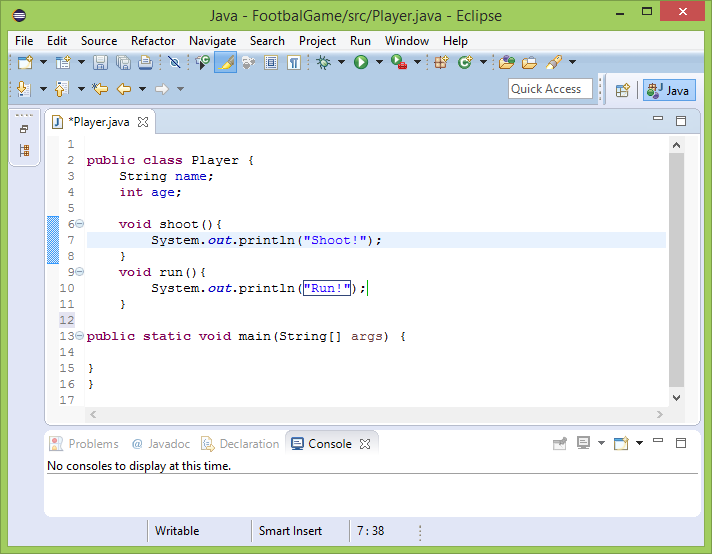
و مطابق روال گذشته می توانیم برای آن متد main تعریف کنیم:



حال باید ویژگی ها و رفتارهای بازیکن را به این کلاس اضافه کنیم:

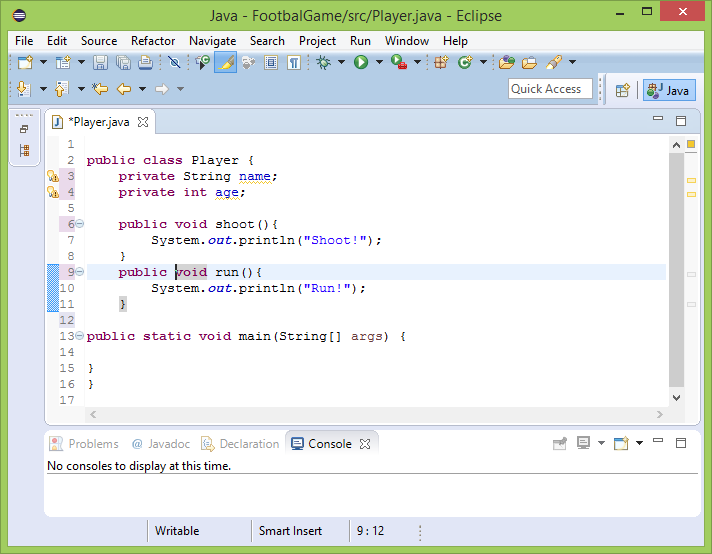


مثلا در اینجا ما در نظر گرفته ایم که هر بازیکن نام و سن دارد.



و به عنوان رفتار برای آن متدهای شوت زدن و دویدن را در نظر گرفته ایم، البته دیدیم که هر متد می تواند void باشد و یا خروجی داشته باشد و ما فعلا متدهایمان را void تعریف کرده ایم و بدنه ی خیلی ساده برای آنها مشخص کرده ایم که نیازی به بازگردادن خروجی ندارند.

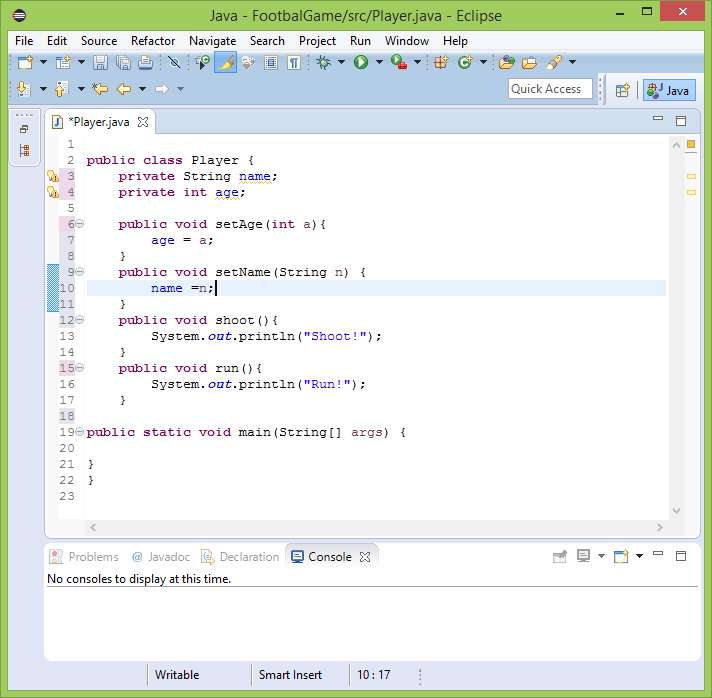
هر کدام از این بخش ها می توانند به نوبه ی خود public و یا private باشند، مثلا ما می توانیم ویژگی های بازیکن را خصوصی قرار دهیم و رفتارها را عمومی:



پس قدم بعدی پس از تعریف نام کلاس و ویژگی ها و رفتارهای آن مشخص کردن سطوح دسترسی ویژگی ها و متدهاست.

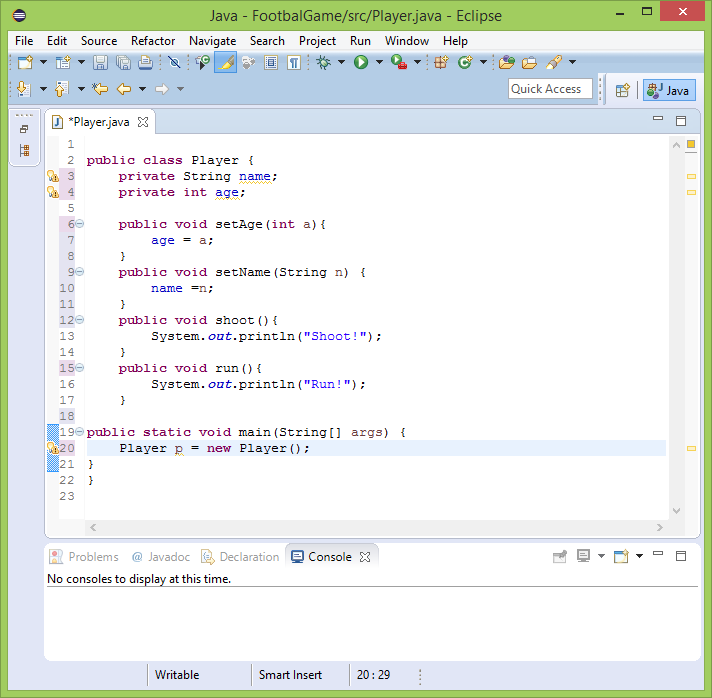
همچنین می توانیم متدهایی ایجاد کنیم که اصطلاحا به آنها Setter و Getter گفته می شود، در برنامه نویسی شی گرا این معمول است که ویژگی های یک کلاس را خصوصی تعریف می کنند و برای مقدار دهی آنها و نیز خواندن مقدار آنها متدهایی قرار می دهند که با عبارت set و سپس نام ویژگی و یا get به همراه نام ویژگی نام گذاری می شوند.

Setter ها متدهای عمومی هستند که هیچ مقداری برنمی گردانند و یکی از ویژگی های کلاس مورد نظر را مقدار دهی می کنند:

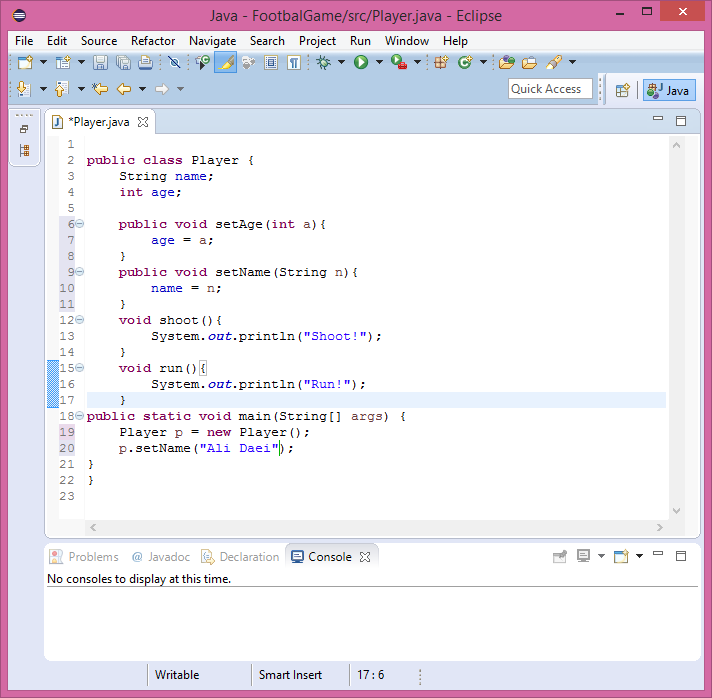


حال چطور می توانیم از کلاسی که ساخته ایم استفاده کنیم؟ ما یک کلاس بازیکن ایجاد کرده ایم که ویژگی هایی و متدهایی دارد و می خواهیم برنامه ای بنویسیم که از این کلاس شی ایجاد کند. مثلا داخل متد main چطور می توانیم از این کلاس استفاده کنیم؟

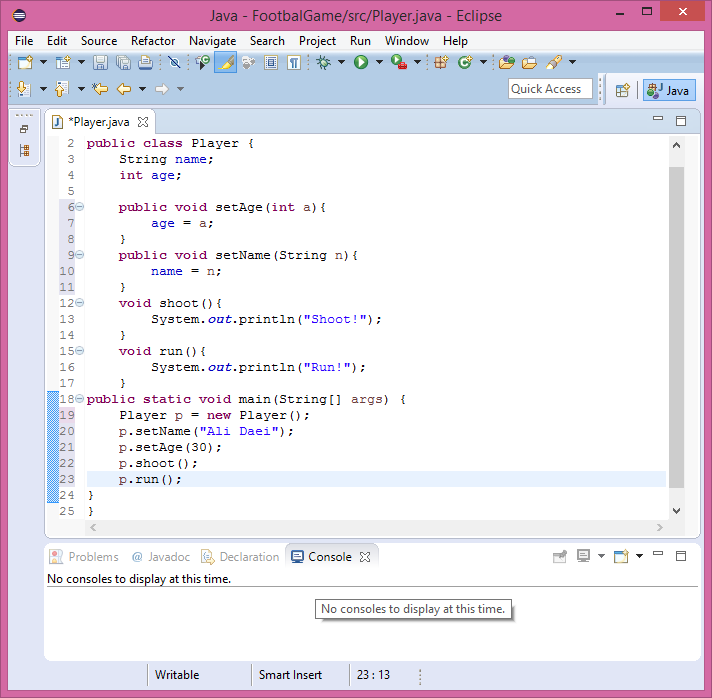
همان طور که گفتیم می توانیم از این کلاس با کمک دستور new شی بسازیم:



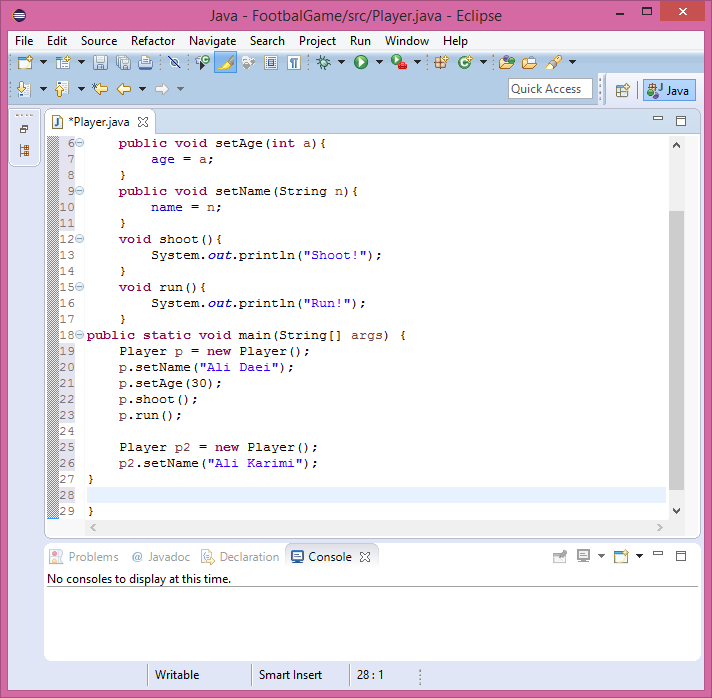
به این کار Object Creation گفته می شود، حال چطور می توانیم از این شی جدید استفاده کنیم؟ با کمک رفتارهایی که به صورت public برای آن تعریف شده است، مثلا می توانیم با کمک setter نوشته شده برای ویژگی name آن را مقدار دهی کنیم:



و استفاده از دیگر رفتارهای این شی هم به همین ترتیب مقدور است:

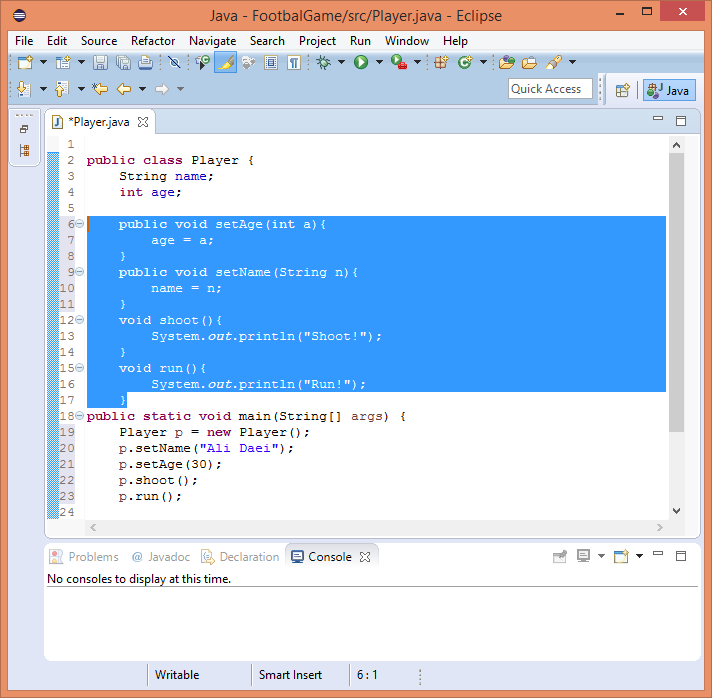


به همین ترتیب می توانیم شی دیگری نیز ایجاد کنیم و ویژگی های آن را مقدار دهی نماییم، یا رفتارهای آن را فراخوانی کنیم:



در این بخش ما داریم از کلاسی که ساخته ایم استفاده می کنیم. اما در بخش اول کلاس را تعریف کردیم و در واقع Encapsulation در بخش قبل انجام شده است، جایی که در آن ما ویژگی ها و رفتارهای کلاس را مشخص می کنیم و تصمیم می گیریم کدامیک از آنها عمومی و کدامیک خصوصی باشند.

مثلا اگر کسی بخواهد از کلاس Player که در این تمرین نوشتیم استفاده کند، مستقیما نمی تواند ویژگی name را فراخوانی کند، بلکه می تواند از بخش های استفاده کند که به صورت public برای این کار قرار داده شده اند، بخش های public در واقع واسط استفاده از این کلاس را تعریف می کنند:



در واقع ما در این بخش یک برنامه شی گرا نوشتیم، و اگر به برنامه ما نگاه کنید، مفاهیمی در آن وجود دارد که در فضای واقعی وجود دارند، مفاهیمی مانند بازیکن و نمونه هایی از آن مانند علی دایی و علی کریمی، و این جایی است که در آن گفته می شود با کمک برنامه نویسی شی گرا فضای مساله به فضای راه حل نزدیک شده است. و زبان برنامه نویسی ما به زبان مساله ما نزدیک شده است و دیگر محدود به انواع داده ی اولیه ای مانند int و یا داده های پیچیده تر مانند String نیستم. و همان طور که می توانستیم متغیری از جنس String داشته باشیم، از این به بعد می توانیم متغیری از جنس Player داشته باشیم. ما انواع داده را با این کار گسترش داده ایم.

مجموع این ویژگی ها موجب می شود که تفکر کردن و حل مساله ساده تر شود، در واقع برنامه نویسی شی گرا مدلی منعطف و پویا برای تولید برنامه هاست.

# جمع بندی

در پایان به جمع بندی و مرور مفاهیمی که در این جلسه دیدیم بپردازیم، ما در این جلسه با مفاهیم مهمی در حوزه ی شی گرایی آشنا شدیم، مثلا مفهوم کلاس که یک رده و گروه از اشیاء را نشان می دهد، همچنین با نحوه ی تعریف کردن اشیاء در زبان جاوا آشنا شدیم. مفهوم شی یا Object که یک نمونه و موجودیت در دنیای واقعی را نشان می دهد را دیدیم و با Instantiation یا نمونه سازی از کلاس آشنا شدیم. رفتارهای اشیاء را دیدیم که در قالب متدها پیاده سازی می شوند، و ویژگی های اشیاء که در قالب Property ها یا متغیرهایی که داخل کلاس ها دیده می شوند تعریف می شوند. با مفهوم محصورسازی یا Encapsulation آشنا شدیم که عبارت بود از تعریف کردن کلاس و مشخص کردن محدوده ی استفاده از آن به معنای محدوده ی عمومی و خصوصی آن. و در آخر مفهوم واسط را دیدیم، دیدیم که بخشی از کلاس که باز و public است در واقع نحوه ی استفاده از کلاس برای کاربر و استفاده کننده را نشان می دهد که به این بخش واسط یا Interface کلاس گفته می شود.

برای مطالعه ی بیشتر می توانید به فصل 1 و 3 از کتاب دایتل مراجعه کنید و نیز تمرینات همین فصل ها را نیز حل نمایید.

# تمرین

به عنوان تمرین با کمک زبان جاوا برای هر یک از موارد زیر کلاسی بسازید همراه ویژگی ها و رفتارها. سپس از هر کلاس در برنامه خود نمونه هایی ایجاد کنید و رفتارهای آنها را فراخوانی کنید و نیز ویژگی های آنها را مقدار دهی نمایید.

* مساله ی کتابخانه:

در این مساله کلاس هایی برای کتاب و عضو کتاب خانه در نظر بگیرید.

* مساله ی بانک:

برای این مساله کلاس های زیر را در نظر گرفته و همان طور که در بالا گفته شد ایجاد و استفاده کنید:



# جستجو کنید و بخوانید

پیشنهاد می کنم در مورد موارد زیر در اینترنت جستجو کنید و بیشتر مطالعه نمایید:

* Object Oriented Programing
* Procedural Programming
* Interface
* Encapsulation
* UML Class Diagram