

بخش سوم:

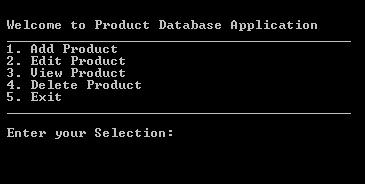
رشته، آرایه و چند داستان دیگر

# رشته، آرایه و چند داستان دیگر

در این فصل ما در مورد رشته ها ، آرایه ها و چند امکان دیگر در زبان جاوا صحبت خواهیم کرد. در پایان این فصل درس ما به جایی می رسد که مفاهیم اصلی برنامه نویسی رویه ای (procedural) را دوره کرده ایم. بعد از این جلسه به بحث شی گرایی خواهیم رسید و از فصل بعد مفاهیم برنامه نویسی شی گرا آغاز خواهد شد. البته تا پایان این فصل هم ما به موضوعاتی برمی خوریم که به اشیا بی ربط نیستند، به خصوص رشته ها و آرایه ها در واقع نمونه هایی از اشیا در زبان جاوا هستند اما تا پایان این فصل بدون باز کردن بحث شی گرایی، همان رویکرد رویه ای را در پیش می گیریم تا مفاهیم اولیه و پایه ای زبان رو به انتها برسانیم و از فصل بعد مسائل پیشرفته تری مانند بحث شی گرایی را شروع خواهیم کرد.

* در این جلسه ما از بحث دریافت اطلاعات از کاربران کار را شروع خواهیم کرد.
* بعد از آن خواهیم دید که کنترل انواع داده در جاوا چگونه است و خواهیم دید که جاوا در این زمینه زبان سخت گیری است
* تکمیل ساختارهای کنترل جریان برنامه مطلب بعدی است که به آن خواهیم پرداخت و در مورد switch و نیز Break و Continue صحبت خواهیم کرد.
* و مفاهیم مهمی مانند رشته ها و آرایه ها بخش مهمی از این فصل خواهند بود که در مورد آنها نیز صحبت خواهیم کرد.

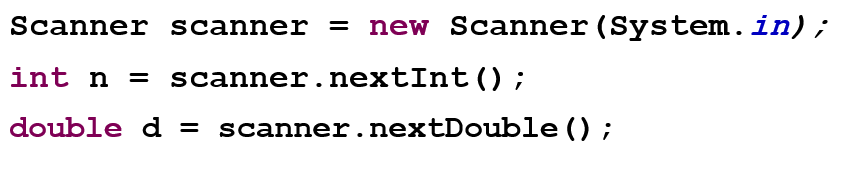
# برنامه های کنسول

در این بخش در مورد برنامه نویسی کنسول و یا Console applications صحبت می کنیم. برنامه های کنسول برنامه هایی هستند که مبتنی بر ورودی و خروجی متنی هستند و امکان ورود اطلاعات نیز مشاهده ی آنها را از طریق محیط متنی برای کاربر فراهم می کنند. این کنسول می تواند command line در ویندوز و یا یک shell در لینوکس باشد. اصطلاحا به ورودی و خروجی این برنامه ها standard input و standard output گفته می شود که به کمک آنها برنامه با کاربر تعامل می کند. ما فعلا در این دوره با برنامه ها کنسول سر و کار داریم و انواع دیگر برنامه مانند Desktop applications, web applications یا mobile applications را نمی بینیم. در واقع می توان گفت که آنها امکانات پیچیده و پیشرفته تری هستند که بعد از یاد گرفتن مفاهیم اولیه می توانید سراغشان بروید. ولی در پرانتز هم می توان گفت که در زبان جاوا بر خلاف برخی از زبان های ویژوال مثل زبان های حوزه دات نت، برنامه های دستکتاپ بخش عمده ای از بازار را به خودشان اختصاص نمی دهند و خیلی از برنامه نویسان جاوا شاید هیچ وقت desktop application تولید نکنند و شاید قدم بعدی برای شما برای ایجاد کردن واسط های کاربری پیشرفته تر بحث برنامه نویسی وب و یا برنامه نویسی موبایل باشد. به هر حال ما در این دوره برنامه های کنسولی ساده را تنها به عنوان تمرین مفاهیم مختلف برنامه نویسی دوره می کنیم تا مفاهیم را خوب تمرین کنیم.

# دریافت مقدار از ورودی

ما می دانیم که می توانیم با کمک دستور println مقداری را به کاربر نشان دهیم و اطلاعاتی را روی صفحه ی کنسول چاپ کنیم. اما چطور می توانیم اطلاعات را از کابر بگیریم؟

یک راه ساده برای این کار استفاده از System.in و کلاس کمکی Scanner است. در واقع از System.out برای چاپ اطلاعات و از System.in برای ورودی گرفتن اطلاعات می توانیم استفاده کنیم. مثال زیر را ببینید:

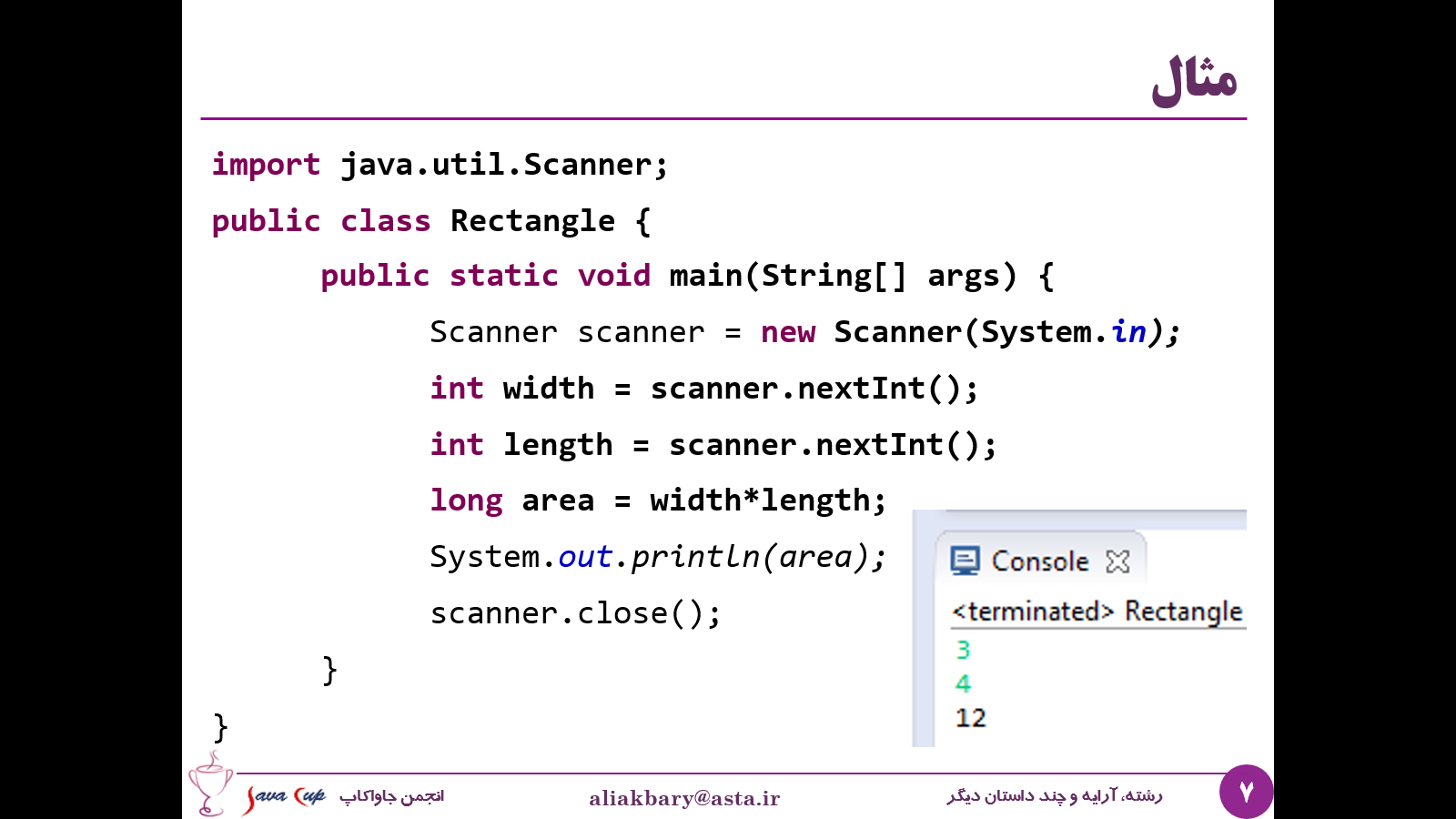


در این مثال از کلاس Scanner استفاده شده و با کمک new یک متغیر از آن ساخته ایم و System.in به آن پاس شده است. و این متغیر scanner در ادامه می تواند برای خواندن اطلاعات مورد استفاده قرار بگیرد. مثلا از scanner.nxtInt() می توانیم استفاده کنیم تا یک مقدار صحیح از کاربر بگیریم و آن را داخل یک متغیر بریزیم، یا از scanner.nxtDouble() می توانیم استفاده کنیم برای خواندن یک متغیر اعشاری. و متدهای مختلف دیگری که برای scanner وجود دارند. در قطعه کد بالا از امکاناتی استفاده شده است که هنوز در این دوره به آنها نپرداخته ایم، مثلا اپراتور new برای ساختن اشیای جدید یا کلا بحث کلاس ها و ساختن متغیر از روی آنها و اینکه متدها چطور بر روی اشیا قابل استفاده هستند. اما برای اینکه بتوانیم مفاهیم اولیه برنامه نویسی را تمرین کنیم می توانید این قطعه کد را حفظ کنید و جزئیات آن را فراموش کنید و اینکه در پشت صحنه ی این کد در واقع چه اتفاقی می افتد را در نظر نگیرید، و تنها با کمک این قطعه برنامه مقادیر را از کاربر بگیرید، برای برنامه هایی که خیلی از اوقات نیاز داریم بنویسیم و اطلاعاتی را از کاربر دریافت کنیم.

نکته: برای اینکه از Scanner در برنامه استفاده کنیم باید خط زیر را به ابتدای برنامه اضافه کنیم، اینکه این خط چه معنایی دارد نکته ای هست که بعدا به آن خواهیم پرداخت:



مثال: مثال زیر را در نظر بگیرید:



این یک کلاس کامل هست که درش اطلاعاتی را از کاربر گرفته ایم. همان طور که می بینید Scanner ابتدای برنامه import شده است، یک کلاس به نام مستطیل ایجاد شده است، همان طور که قبلا دیدیم برای ایجاد هر برنامه باید یک کلاس بنویسیم، متد main هم در آن نوشته شده است. همان طور که می بینید Scanner در آن ایجاد شده است و متغیرهای عرض و طول رو با کمک متد nxtInt() از کاربر گرفته ایم، در واقع این برنامه قرار است طول و عرض یک مستطیل را از کاربر بگیرده و مساحت آن را چاپ نماید. پس در ادامه مساحت را محاسبه و چاپ کرده ایم و در نهایت هم خوب است که Scanner را ببیندیم. جزئیات استفاده از اسکنر را در فصل های بعد خواهیم دید.

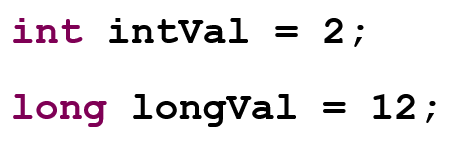
هنگامی که این برنامه اجرا شود کاربر می تواند به ازای دو خطی که nxtInt() را فراخوانی کرده است مقادیری در کنسول وارد کند، همانطور که در مثال بالا می بینید برای مثال کاربر می تواند اعداد 3و 4 را وارد کند و این اعداد به متغیرهای width و length نسبت داده می شوند و نهایت مساحت محاسبه و همان طور که می بینید چاپ می شود.

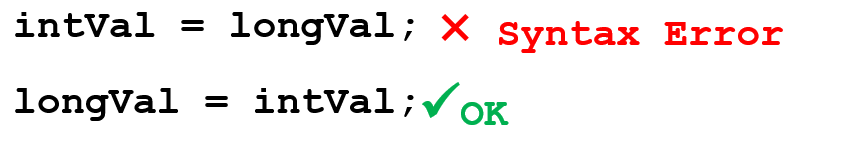
# تبدیل و کنترل نوع

* کنترل نوع

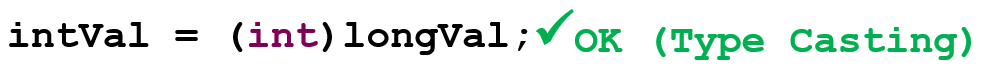
در این بخش درباره کنترل انواع داده در زبان جاوا صحبت می کنیم. واقعیت این است که جاوا زبان سخت گیری است درباره مقدار دهی کردن به متغیرهای مختلف و اجازه نمی دهد هر نوع مقداری را در هر نوع متغیری ذخیره کنیم. از این نظر جاوا با خیلی از زبان های دیگر متفاوت است مثلا زبان C و C++ به اندازه جاوا سخت گیر نیستند. یا خیلی از زبان های اسکریپتی مانند جاوااسکریپت نیز مانند جاوا سخت گیری نمی کنند و مثلا اجازه می دهند که متغیری از نوع رشته در متغیر عددی ذخیره شود یا برعکس یا اجازه می دهند که متغیر Boolean داخل یک عدد ذخیره شود. اما جاوا در این زمینه سخت گیری بیشتری می کند و این سخت گیری مزایایی را برای جاوا به ارمغان آورده است.

مثلا کد زیر را در نظر بگیرید، فرض کنید دو متغیر از جنس int و long داریم که مقداری را در خود ذخیره کرده اند:



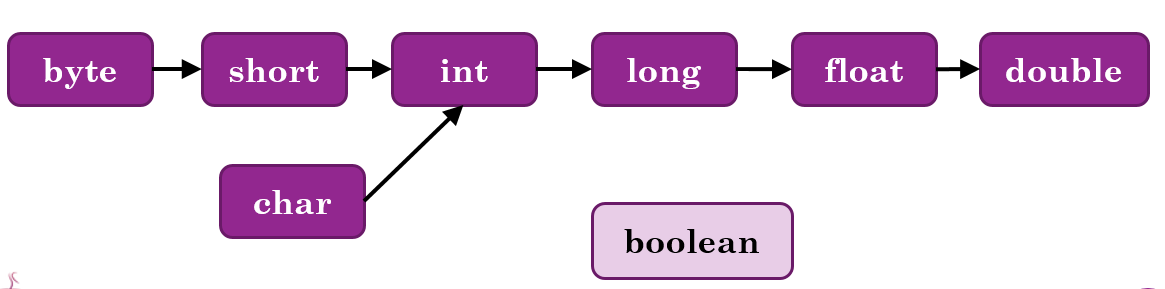
جاوا اجازه نمی دهد که متغیری از نوع long به متغیری از نوع int نسبت داده شود و این دستور منجر به Syntax Error می شود و جاوا جلوی اجرای چنین دستوری را می گیرد، اما بر عکس آن مجاز است یعنی می توان یک int را داخل یک نوع long ریخت و این هم منطقی هست چون integer قرار است در 4 بایت ذخیره شود و long در 8 بایت پس هر عدد integer در یک عدد long جا می شود، پس این انتساب مجاز است اما برعکس آن مجاز نیست.

اگر برنامه نویس به هر دلیلی بخواهد یک Long را در یک Integer قرار دهد، باید این متغیر را تغییر نوع بدهد، و آن را اصطلاحا Cast کند. این پرانتز حاوی نوع که در مثال زیر می بینید موجب می شود که جاوا با متغیر ما به صورت نوعی که جلوی آن ذکر کرده ایم برخورد کند و در اینجا فرض کند که این مقدار یک مقدار Integer است. به این کار type casting گفته می شود. با این کار مقدار longVal به صورت integer در نظر گفته شده و در intVal ریخته می شود و هیچ تغییری روی خود متغیر lognVal اتفاق نمی افتد.



# تبدیل مستقیم انواع داده

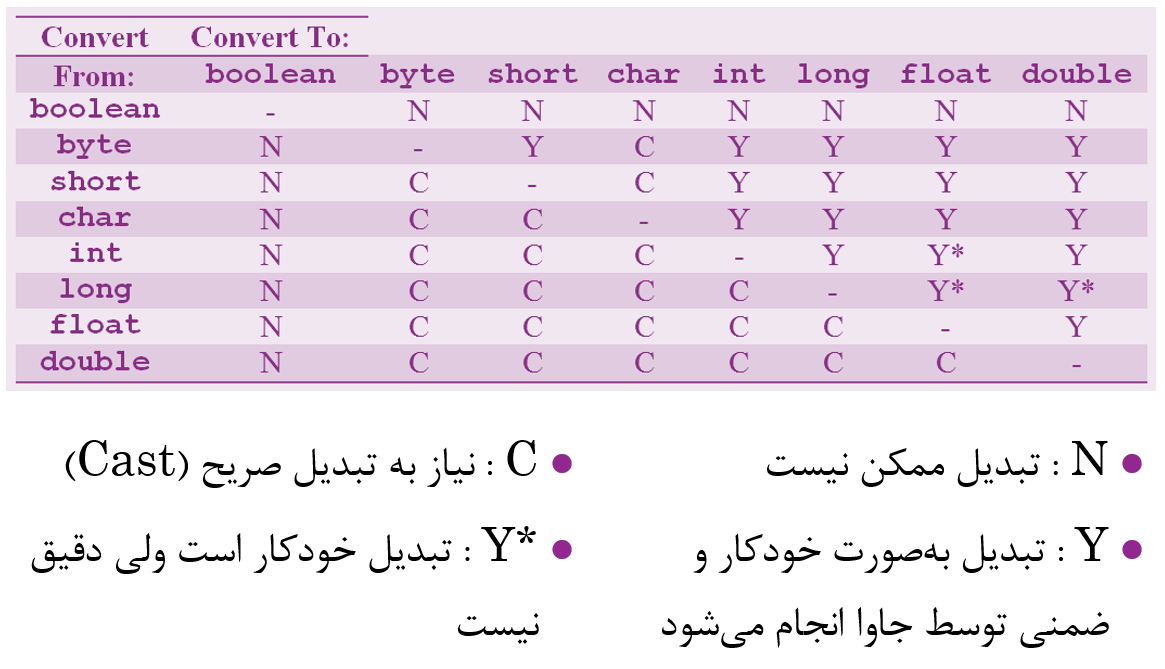
حالا ببینیم چه تبدیل هایی از نظر جاوا مجاز هستند و چه تبدیل هایی غیر مجاز. در شکل زیر انواع داده های اولیه در زبان جاوا نشان داده شده است. جهت فلش نشان می دهد که چه تبدیل هایی مجاز و چه تبدیل هایی غیر مجاز هستند. مثلا تبدیل کردن یک مقدار از نوع integer به یک مقدار از نوع long مجاز است ولی برعکس آن مجاز نیست. در این میان character یک نوع داده ی خاص است که کاراکتر را نگهداری می کند اما از آنجایی که کد یونیکد آن کاراکتر در این متغیر ذخیره می شود قابل تبدیل به int است و می توان از آن در عملیات های مختلف نیز استفاده کرد،



همان طور که می بینید نوع داده ی Boolean یک نوع داده ی خاص است که به هیچ نوع داده ای قابل تبدیل نیست و هیچ نوع داده ای هم به این نوع قابل تبدیل نیستند، بنابراین اگر در یک طرف از یک تساوی بولین قرار داده شود باید در طرف دیگر هم حتما همین نوع داده قرار داده شود. و این فلش ها اصطلاحا transitive یا تراگذر هستند و مثلا یک int را هم می توان داخل یک long و هم یک float و هم یک double قرار داد و الی آخر.

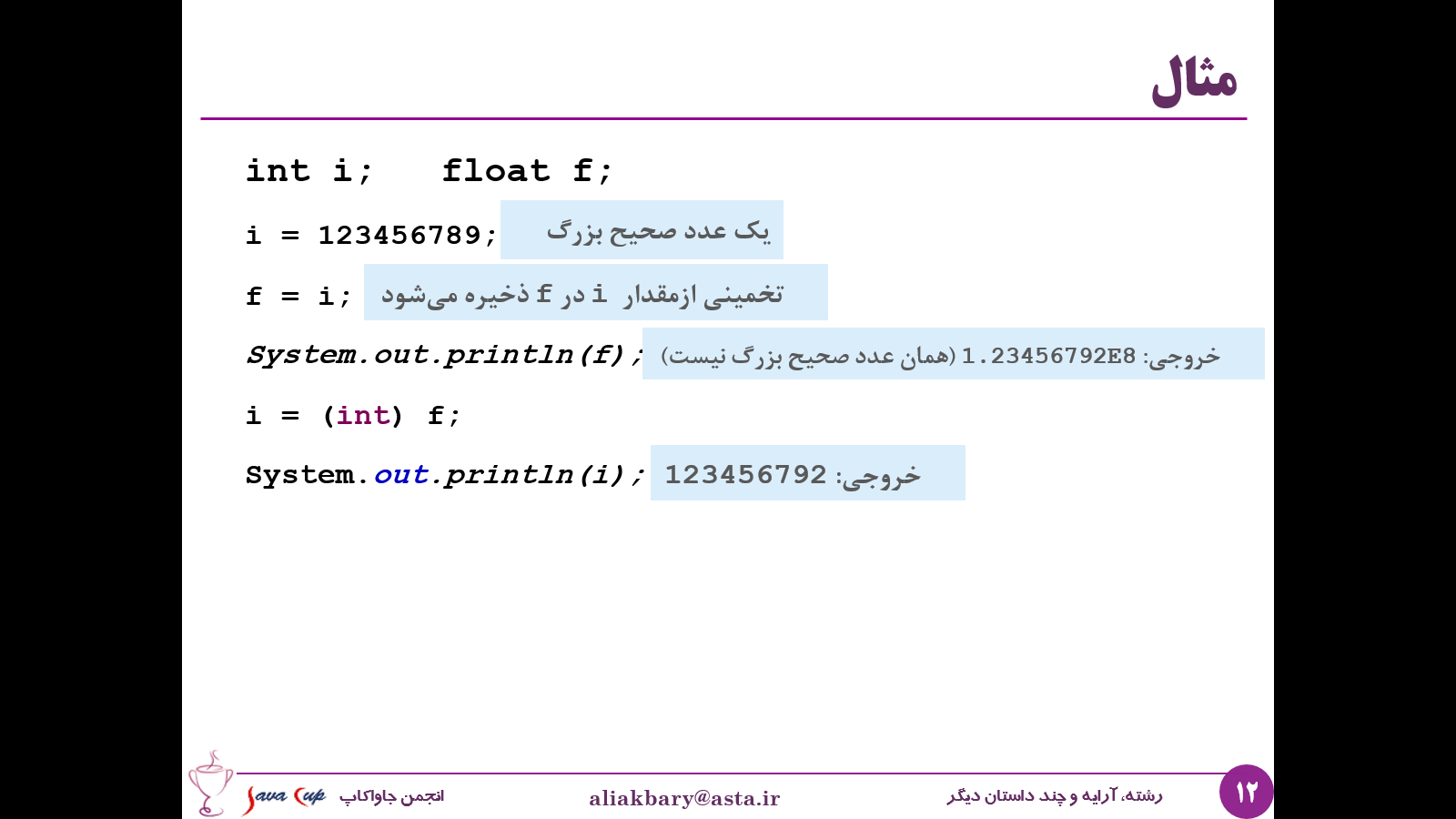
# جدول تبدیل انواع داده

این جدول نمای کامل و دقیق تری از امکان تبدیل انواع داده ای در جاوا را نشان می دهد. سطرها و ستون های این جدول انواع داده در زبان جاوا هستند و هر کدوم از خونه ها امکان تبدیل نوع این داده ها به یک دیگر را نشان می دهند.



حرف N به معنای عدم امکان تبدیل است، برای مثال نوع داده ی بولین را به هیچ داده ای نمی توان تبدیل کرد، حرف Y به معنای امکان تبدیل است مثلا یک متغیر از نوع بایت را می توان در متغیری از نوع int و یا long قرار داد، حرف C به معنای لزوم type casting است یعنی باید برای تبدیل این نوع داده ها، به صورت صریح تبدیل ذکر شود، مثلا اگر بخواهیم نوع داده ی کاراکتر را در داده ای از نوع بایت بریزیم حتما باید از type casting استفاده کنیم وگرنه syntax error خواهیم داشت. و یک مقدار Y\* هم در جدول دیده می شود و به این معناست که تبدیل نوع داده به صورت خودکار اتفاق می افتد اما این تبدیل به صورت دقیق نیست و در واقع تقریبی از مقدار عددی در متغیری که خواسته ایم ذخیره می شود. دلیلش هم این است که نحوه ی ذخیره ی داده در انواع اعشاری مانند float و double با نحوه ی ذخیره داده ها در انواع صحیح تفاوت دارد و به همین دلیل مقدار دقیق داده های صحیح را نگه نمی دارند.

این موضوع را با یک مثال ببینیم:



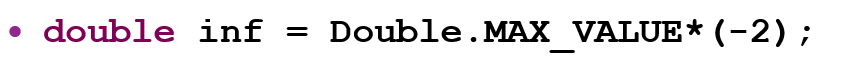
پس انواع اعشاری به این علت که تقریبی از عدد را در خود نگه می دارند هنگامی که متغیری از نوع صحیح به آنها انتساب داده شود، مقدار واقعی ذخیره نمی شود بلکه تقریبی از عدد در این متغیرها نگهداری می شود.

# چند نکته درباره اعداد اعشاری

* اعداد اعشاری می توانند مقدار مبهم (Not a number) را نگهداری کنند، برای مثال نتیجه ی تقسیم صفر بر صفر مبهم است. این مقدار به صورت Double.NaN هم قابل نمایش است.



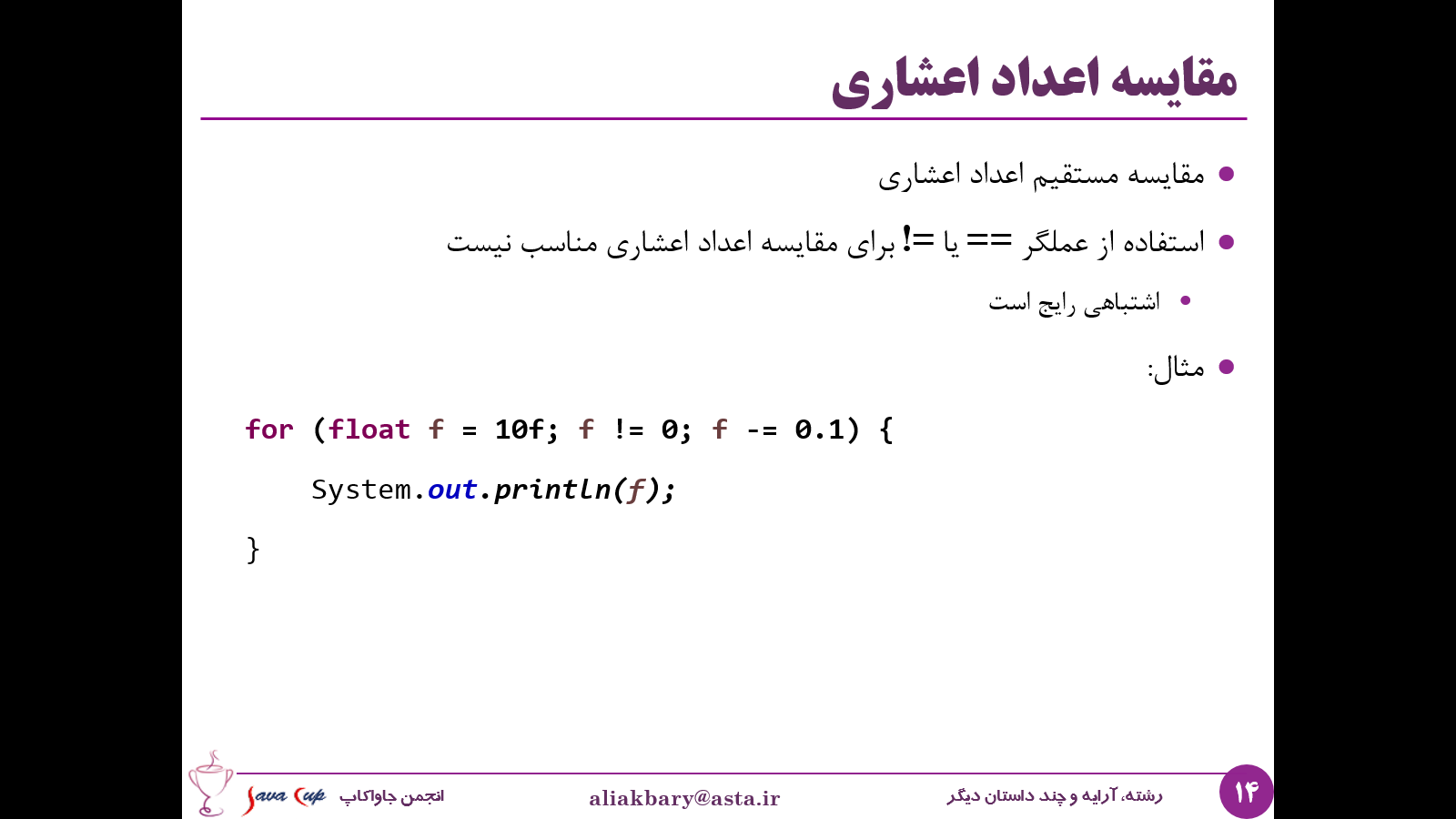
* همچنین مقدار بی نهایت نیز برای اعداد اعشاری تعریف شده است و Double.POSITIVE\_INFINITY و Double.NEGATIVE\_INFINITY نشان دهنده ی این مقادیر هستند. برای مثال در کد زیر حاصل عبارت منفی بینهایت خواهد شد:



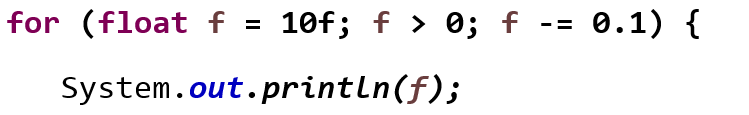
* نکته ی دیگر اینکه اگر می خواهید اعداد اعشاری را با فرمتی که خودتان می خواهید نمایش دهید می توانید به جای تابع println از تابع System.out.format استفاده نمایید. این متد شبیه متد printf در زبان های C و C++ است که یک رشته و یک مقدار می گیرد و داخل آن رشته ما می توانیم فرمت مقدار داده شده را مشخص کنیم. برای مثال %5.2f یعنی می خواهیم عدد اعشاری ما در فضایی 5 کاراکتری و با دو رقم اعشار نمایش داده شود.



* مقایسه ی اعداد اعشاری: یک نکته ی مهم درباره اعداد اعشاری این هست که بهتر است این اعداد را با اپراتورهای مساوی و یا نامساوی با هم مقایسه نکنیم، این اشتباه رایجی هست که خیلی از افراد این اعداد را با این اپراتورها با هم مقایسه می کنند. در حالی که اعداد اعشاری تقریبی از اعداد را نگهداری می کنند، در نتیجه ممکن است دو عدد که ظاهرا مساوی هستند تغییرات اندکی داشته باشند و این کار ما را خراب کند. به این مثال توجه کنید:



در ظاهر به نظر می رسد که این حلقه از 10 شروع شده و کم می شود تا به 0 برسد، اما در واقعیت این حلقه یک حلقه ی بینهایت است.. علت این است که متغیر اعشاری ما با اپراتور نامساوی با 0 مقایسه شده است در حالی که ممکن است این متغیر هرگز مقدار دقیق صفر را نگه ندارد. روش بهتر این است که در این مثال به جای اپراتور نامساوی از اپراتور بزرگتر استفاده نماییم:



و یا در سایر مثال ها می توانید به جای استفاده از مساوی فاصله ی متغیر اعشاری از مقدار مورد نظر را محاسبه کنید و برای مثال بگویید تا زمانی که فاصله کمتر از 0.001 بشود این دو عدد مساوی هستند، به هر حال مقایسه دو عدد اعشاری با اپراتورهای گفته شده یک اشتباه رایج است.

# انتساب های عدد

اگر یک متغیر بزرگ را داخل یک متغیر کوچک تر بریزیم چه اتفاقی خواهد افتاد؟ مثلا اگر یک متغیر از جنس long با مقدار خیلی بزرگ را داخل متغیری از نوع int بریزیم چه اتفاقی خواهد افتاد؟

اگر این سرریز برای اعداد صحیح اتفاق بیافتد در زمان اجرا هیچ خطایی رخ نخواهد داد بلکه تنها بیت های کم اهمیت تر از متغیر بزرگ مورد استفاده قرار می گیرند. مثلا اگر long را داخل int بریزیم 4 بایت کم ارزش long مورد استفاده قرار می گیرد و 4 بایت دیگر دور ریخته می شود.

مثال:

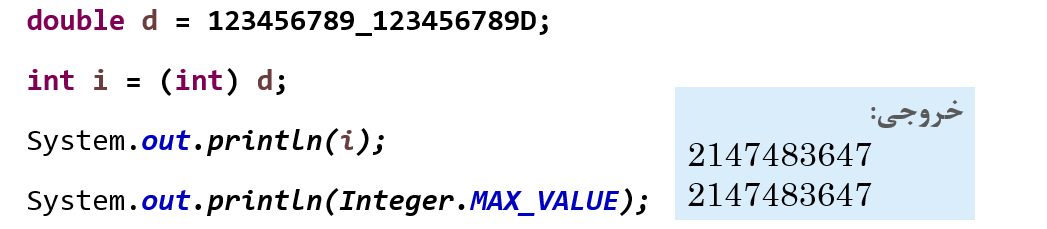


همان طور که در این مثال هم مشاهده می شود در این حالت ما خطایی دریافت نمی کنیم چرا که casting را انجام داده ایم اما نتیجه از نظر منطقی درست نیست (خطای منطقی داریم) و مقدار چاپ شده مقدار نامشخصی حاصل از تفسیر 4 بایت کم ارزش عدد اولیه است.

نکته: برای مشخص کردن یک عدد long می توان از حرف L در انتهای آن استفاده کرد و نیز هنگام نوشتن اعداد هرجا که بخواهید می توانید از کاراکتر underline استفاده کنید برای جداسازی ارقام و این کاراکتر هنگام کامپایل نادیده گرفته می شود.

اما سرریز انتساب برای اعداد اعشاری متفاوت است، اگر یک عدد اعشاری را در یک عدد صحیح بریزیم، آن عدد صحیح حداکثر مقدار ممکن را نگه خواهد داشت چراکه نوع ذخیره سازی اعداد اعشاری و اعداد صحیح متفاوت است و در واقع بیت های کم اهمیت و با اهمیت مانند حالت قبل وجود ندارند و رفتار جاوا در این زمینه متفاوت است.

مثال:



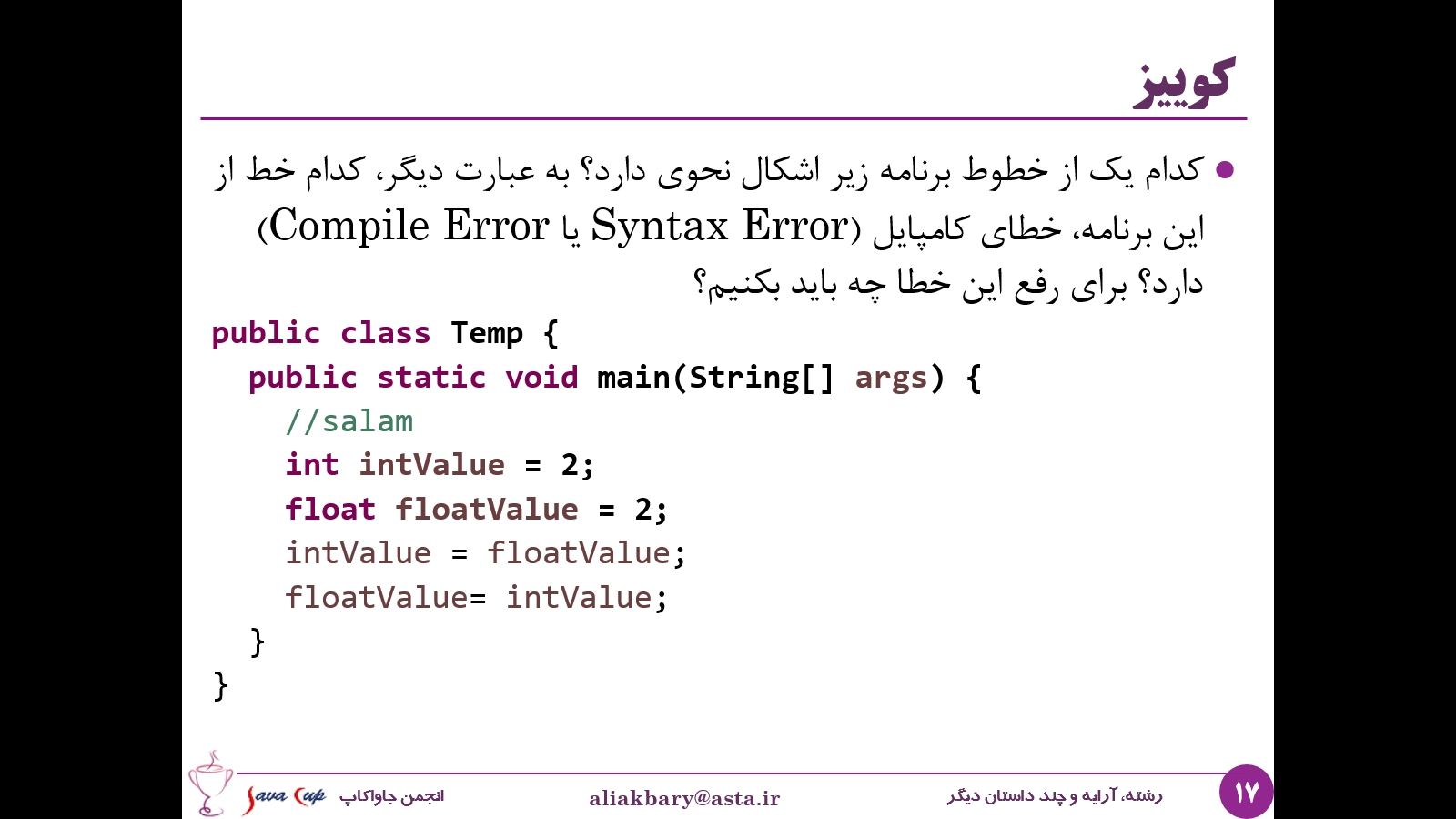
در مثال بالا یک عدد اعشاری بزرگ را در یک متغیر از نوع int ریخته ایم، نتیجه دقیقا برابر با بزرگترین عدد قابل نگهداری در int است. (Integer.MAX\_VALUE حداکثر عددی است که جاوا می تواند در یک متغیر از نوع int نگهداری کند)

نکته: به همین ترتیب Integer.MIN\_VALUE و یا Long.MAX\_VALUE و سایر مقادیر مانند اینها نیز در جاوا وجود دارند و قابل استفاده هستند.

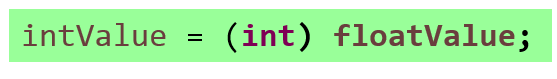
# کویز

در این بخش برای مرور مطالب یک کویز کوتاه خواهیم داشت:

* کدام یک از خطوط برنامه زير اشکال نحوی دارد؟ به عبارت دیگر، کدام خط از این برنامه، خطای کامپایل (Syntax Error يا Compile Error) دارد؟ برای رفع این خطا چه باید بکنیم؟

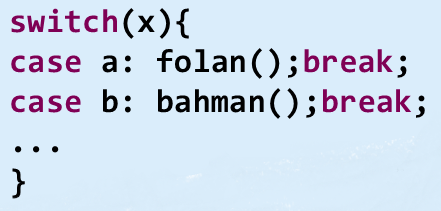


پاسخ: مشکل کد در جایی است که یک متغیر از نوع float داخل متغیری از نوع int ریخته شده است و این بخش برنامه را دچار خطای کامپایل می کند و برای حل این مشکل باید این انتساب با کمک casting انجام شود.



# سایر ساختارهای کنترل جریان برنامه

* ساختار Switch

ما قبلا با ساختار if-else آشنا شدیم و دیدیم که چطور با کمک این ساختار شرطی می توانیم اجرای برنامه ها را مدیریت کنیم و بخشی از برنامه ها را با یک شرط اجرا بکنیم و یا اجرا نکنیم. گاهی این ساختارهای شرطی با کمک switch راحت تر و بهتر نوشته می شوند و برنامه ما با کمک این ساختار خوانایی بهتری پیدا می کند. اگر if-else هایی که می نویسیم قرار است مقدار یک متغیر را با مقادیر مختلف چک بکنند و به ازای هر مقدار یک کار انجام بدهیم، در چنین شرایطی switch جایگزین خوبی برای if-else خواهد بود. مثلا اگر می خواهیم در صورتی که مقدار x برابر a باید یک کار انجام بدهیم، اگر برابر b باشد یک کار دیگر و ... استفاده از این ساختار مناسب است.

این کد در زبان جاوا نوشته شده است و معنای آن این است که اگر متغیر x مقدار a را داشت متد folan و اگر مقدار b داشت متد bahman اجرا شود.

این ساختار یک پارامتر می گیرد که یک متغیر است و case های مختلفی دارد که با همین کلمه کلیدی case مشخص می شوند و جلوی case مقادیر مختلف نوشته می شود و بعد از هر case باید بگوییم چه کاری باید انجام شود و در انتها هم از break باید استفاده کنیم تا انتهای case مشخص شود.

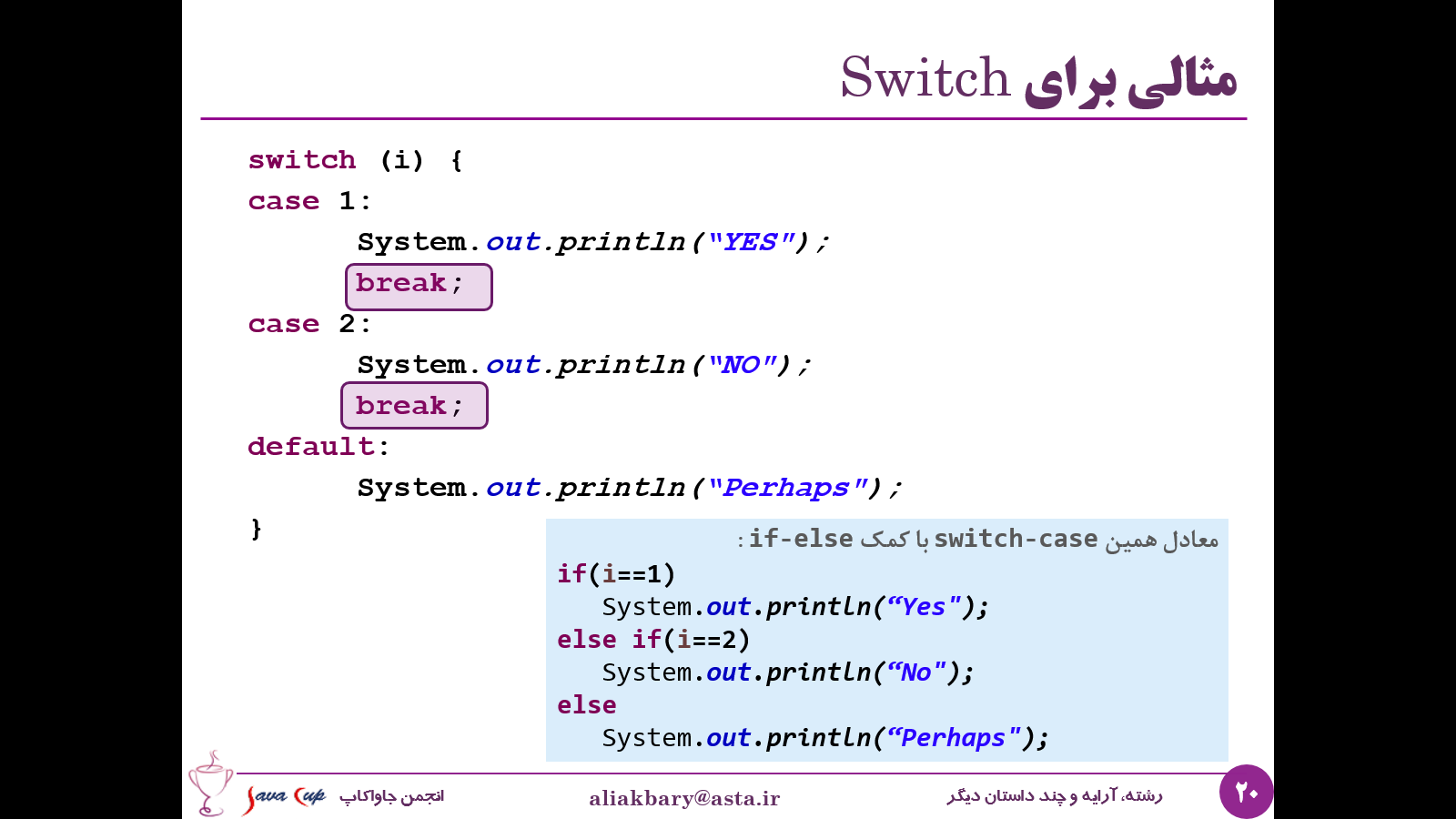
* انواع داده ی مختلفی برای switch قابل استفاده هستند، برای مثال انواع داده ای که به int قابل تبدیل هستند همه می توانند متغیر switch قرار بگیرند.
* در مثال فوق x نمی تواند از این انواع باشد: Boolean, float, double, long
* همچنین انواع دیگری مانند رشته و enum نیز به عنوان پارامتر switch قابل استفاده است که در فصل های بعد با آنها آشنا خواهیم شد.

نکته: رشته از نسخه ی جاوای 1.7 به بعد قابلیت استفاده به عنوان پارامتر switch را پیدا کرد و قبل از آن این امکان وجود نداشت.

مثال:

در مثال زیر مقدار i با 1 و 2 مقایسه می شود، اگر مقدار برابر 1 باشد عبارت YES چاپ خواهد شد، اگر نه و مقدار برابر 2 باشد عبارت NO و در غیر این صورت عبارت Perhaps چاپ خواهد شد. پس کلید واژه ی default مشخص می کند که اگر مقدار متغیر ما با هیچ کدام از case ها مطابقت نداشت، چه دستوری باید اجرا شود.

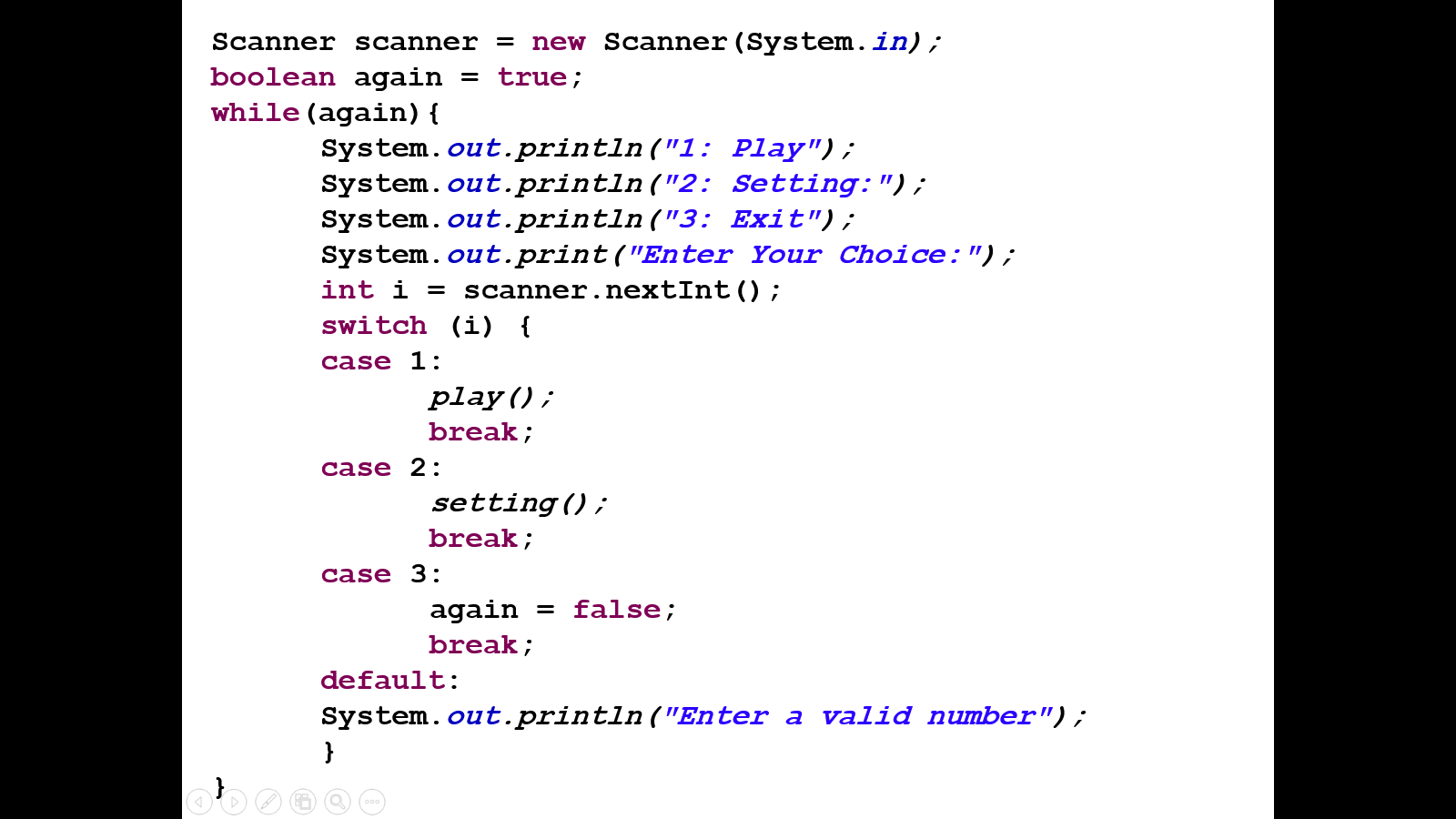
همچنین در ادامه معادل همین کد با if-else را مشاهده می کنید، هر switch را می توان با if-else هم پیاده سازی کرد اما استفاده از این ساختار (switch) می تواند برنامه ما را ساده تر و شفاف تر کند.



نکته: به این دقت کنید که در انتهای هر case حتما باید از عبارت break استفاده کنید وگرنه ممکن است دستورات case بعدی هم اجرا شوند، مثلا در کد بالا اگر break اول را نگذارید، در صورتی که مقدار i برابر با 1 باشد، هم عبارت Yes و هم عبارت No چاپ خواهد شد و بعد break دوم اجرا را به خارج از switch منتقل می کند.

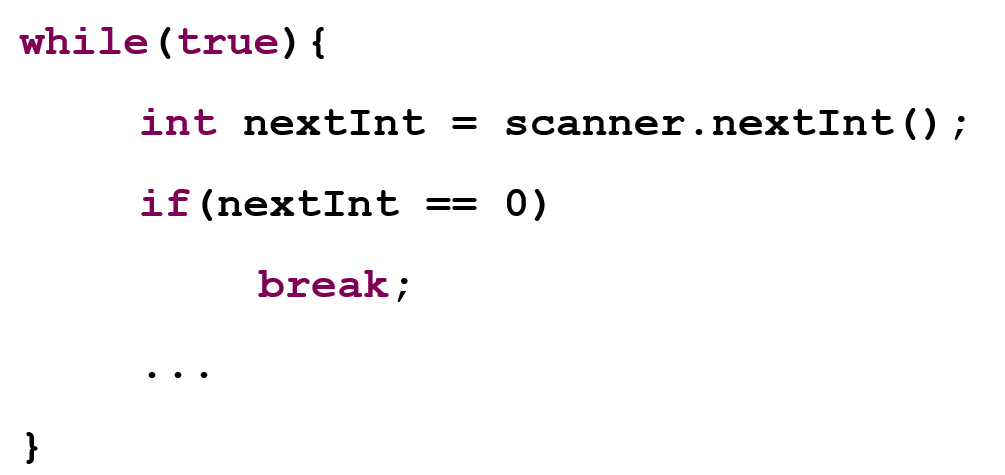
مثال دیگر: تکه کد زیر مثال دیگری برای حلقه ی switch است، در این تکه کد از کلاس Scanner که قبلا با آن آشنا شدیم استفاده شده است و در یک حلقه ی While تا زمانی که متغیر again برابر true هست حلقه اجرا می شود، هر بار متنی برای کاربر چاپ می شود و در واقع یک منو در اختیار کاربر گذاشته و یک عدد از کاربر گرفته می شود، اگر این عدد 1 باشد، متد play صدا زده می شود و اگر 2 باشد متد setting فراخوانی می شود و اگر کاربر مقدار 3 را وارد کند متغیر again را برابر false قرار می دهیم تا این حلقه دیگر ادامه پیدا نکند.

ناگفته پیداست که ما فرض کرده ایم که متدهای play و setting در این مثال وجود داشته اند اگرچه بدنه ی آنها را نیاورده ایم.



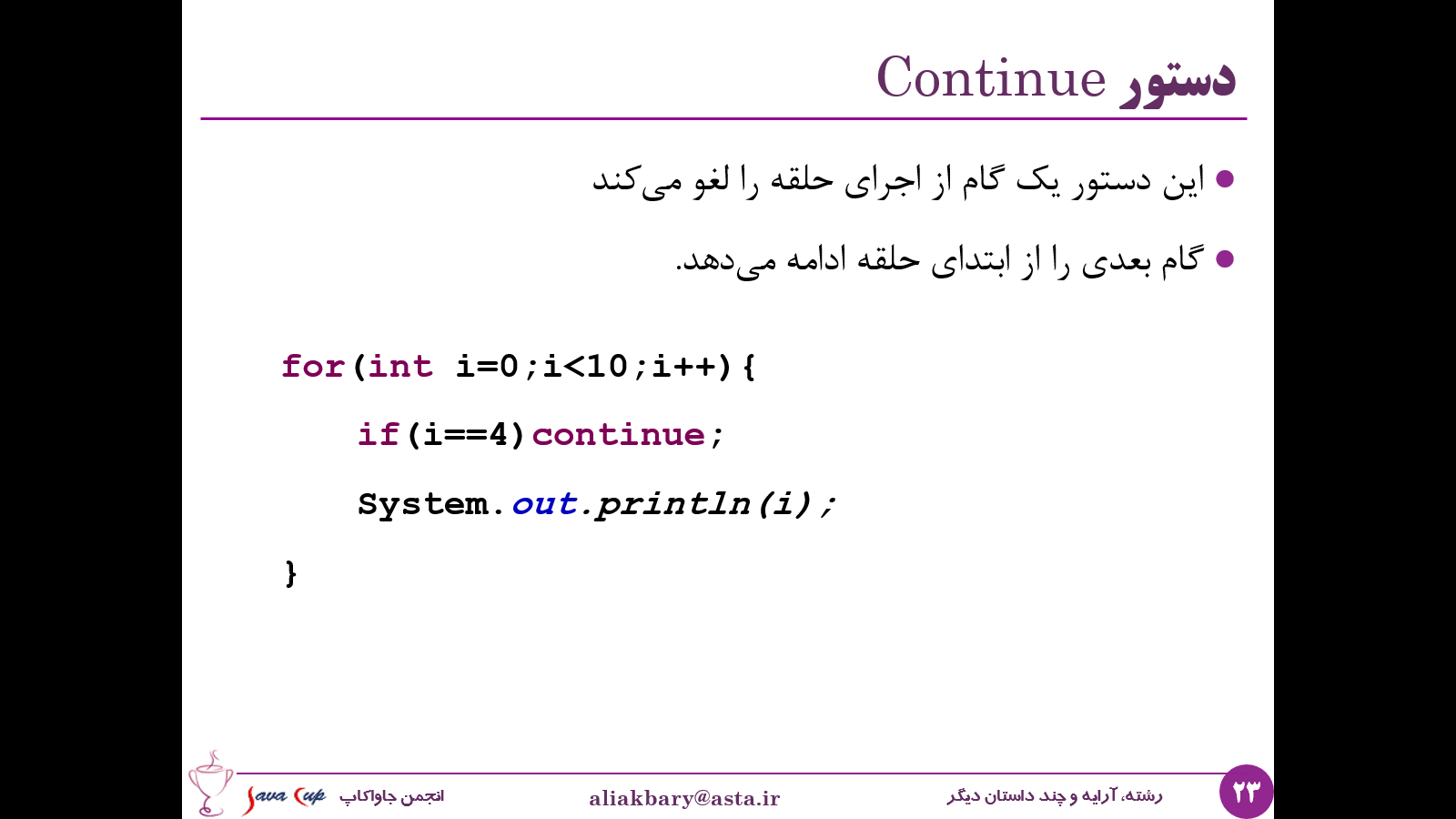
# دستور Break

ما با کاربرد دستور break در ساختار switch-case آشنا شدیم، و دیدیم که در انتهای هر case باید break را قرار دهیم. اما دستور break کاربرد دیگری در حلقه ها دارد، اگر داخل یک حلقه دستور break اجرا شود، اجرای آن حلقه متوقف شده و کنترل برنامه به خارج از حلقه منتقل و از آنجا ادامه پیدا می کند، برای مثال در قطعه کد زیر اگر مقدار nextInt برابر 0 بشود، کل حلقه متوقف خواهد شد.



# دستور Continue

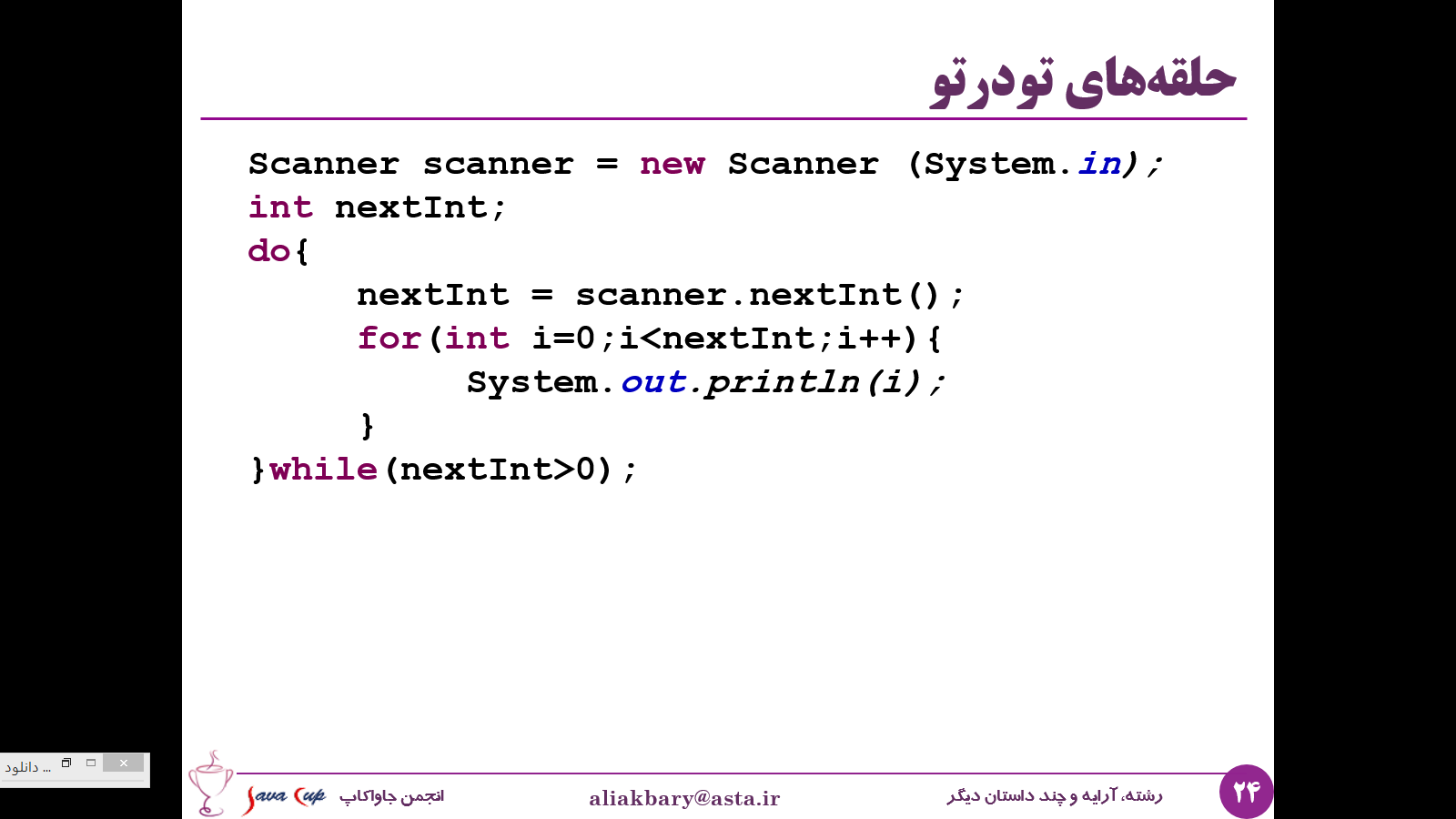
این دستور نیز در حلقه ها به کار می رود، این دستور برخلاف break کل حلقه را متوقف نمی کند بلکه تنها یک گام از اجرای حلقه را لغو می کند و به گام بعدی از اجرای همان حلقه می رود، تکه کد زیر را در نظر بگیرید. این حلقه قرار است از 0 تا 9 مقادیر مختلف i را چاپ کند، اما گفتیم اگر i==4 شد، ادامه ی این دور از حلقه اجرا نشود و کنترل به گام بعدی حلقه منتقل شود، پس مقدار 4 به ازای این حلقه چاپ نخواهد شد.



نکته: یک تفاوت ظریف وجود دارد بین حلقه ی for و سایر حلقه ها مانند while در زمینه ی دستور continue وقتی دستور Continue داخل حلقه ی for اجرا می شود، ما به ابتدای حلقه برمی گردیم و آخرین بخش از دستور for یک مرتبه اجرا می شود، مثلا در اینجا i++ اجرا می شود و بعد بقیه ی بخش ها، اما اگر continue را داخل یک while و یا do-while قرار بدهیم، این دستور باعث می شود تنها کنترل به ابتدای حلقه برگردد و دستور خاصی لزوما اجرا نمی شود و هر آنچه ابتدای حلقه ذکر شده باشد، همان اجرا خواهد شد.

# حلقه های تودرتو

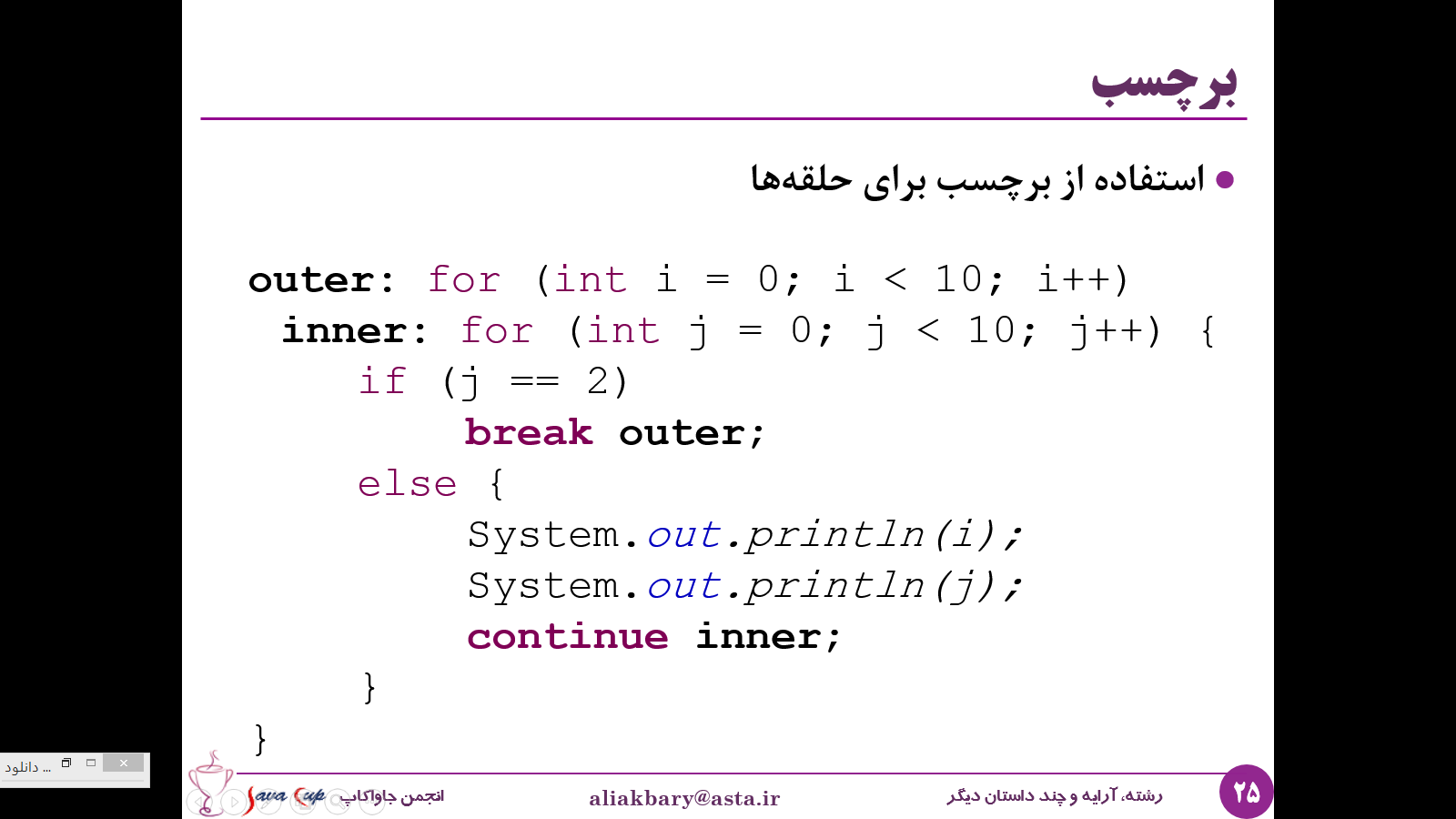
همان طور که می دانیم حلقه ها می توانند داخل هم قرار بگیرند که به این حالت حلقه های تودرتو یا اصطلاحا nested loops گفته می شود. مثلا در قطعه کد زیر یک do-while داریم که داخل آن یک حلقه ی for قرار گرفته است. این برنامه خیلی ساده است، قرار است هر بار یک عدد از کاربر بگیرد و اعداد 0 تا آن عدد را چاپ کند و این کار را تا زمانی که عدد وارد شده از 0 بزرگتر است انجام دهد.



اما اگر بخواهیم از break و continue در حلقه ی داخلی (در اینجا حلقه for) استفاده کنیم، این دستور تنها برای همان حلقه اعمال می شود نه برای حلقه ی خارجی. حال اگر ما در میان کدی که برای حلقه ی داخلی می نویسیم بخواهیم break و یا continue را برای حلقه ی خارجی اعمال کنیم چه باید بکنیم؟

برای این نیازمندی جاوا امکان خاصی را تعبیه کرده است. جاوا اجازه می دهد ما به حلقه ها اسم یا برچسب بدهیم و وقتی که از دستورات break و continue استفاده می کنیم، حلقه ی مورد نظر را ذکر کنیم. پس می توانیم به حلقه ها برچسب بدهیم، برچسب ها قبل از ذکر حلقه قرار می گیرند، مثلا در مثال زیر ما حلقه ی بیرونی را با برچسب outer و حلقه ی داخلی را با برچسب inner مشخص کرده ایم. در مثال زیر دستور break موجب می شود اجرای حلقه ی خارجی خاتمه پیدا کند و به ادامه ی برنامه بعد از حلقه پرداخته شود، و continue موجب می شود حلقه ی داخلی ما یک گام را نادیده بگیرد و به ابتدای حلقه ی داخلی برگردیم.

البته در این مثال خاص continue inner با continue تفاوت عملکردی ندارند اما break outer با break فرق می کند چرا که در اولی از حلقه ی دوم خارج می شویم و در دومی تنها حلقه ی داخلی پایان می یابد.



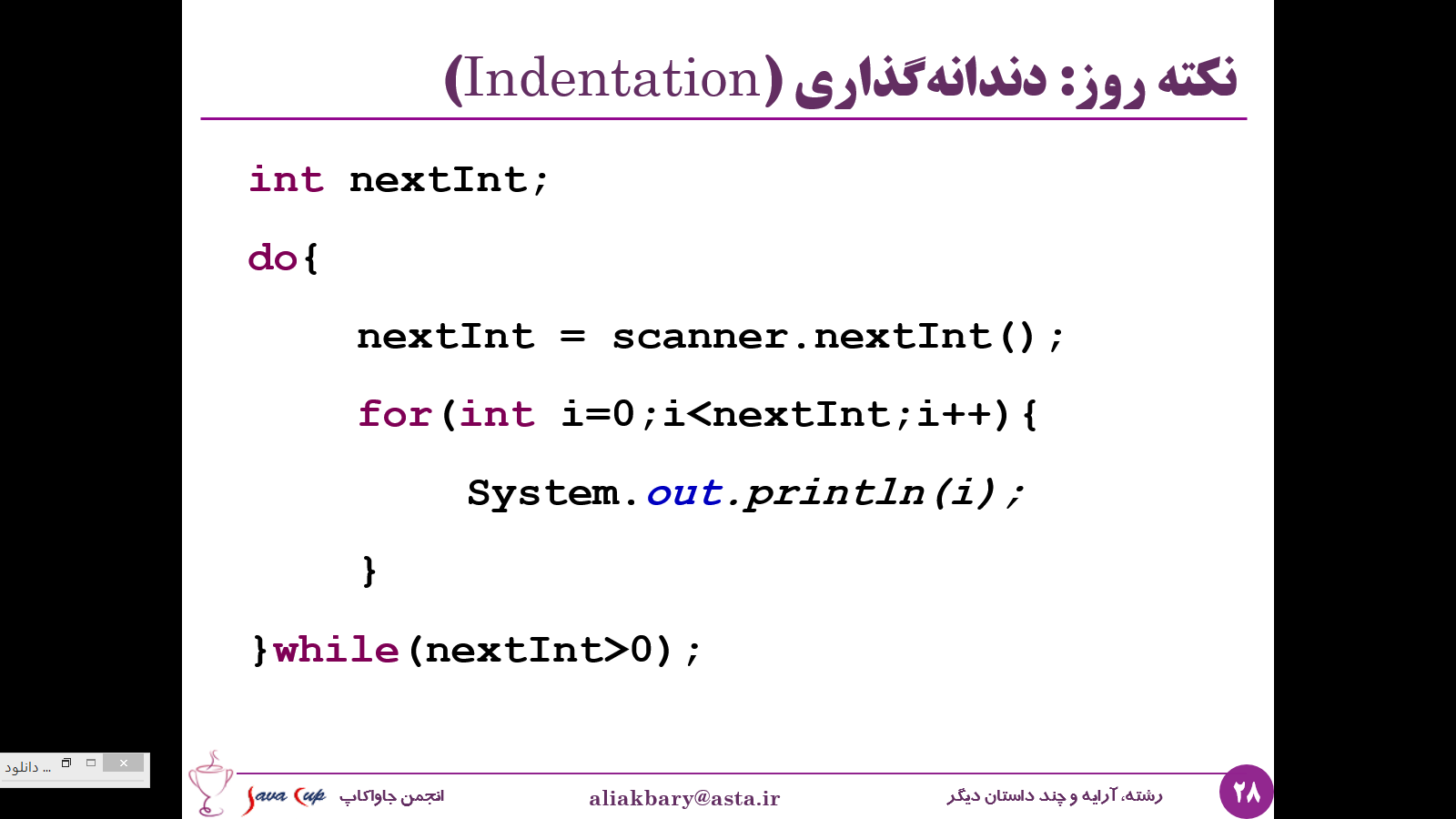
# نکته روز

* دندانه گذاری (indentation)

یک نکته ی جانبی در برنامه نویسی، که در جاوا و خیلی از زبان های دیگر مطرح هست موضوع دندانه گذاری یا indentation است که موجب می شود خوانایی برنامه شما بیشتر شود. این تکنیک برای برنامه نویسی اجباری نیست، یعنی از نظر زبان برنامه نویسی لزومی به رعایت آن وجود ندارد اما موجب می شود برنامه شما شفاف تر و خواناتر شود، قطعه برنامه زیر را ببینید:



تک تک خطوط این برنامه زیر هم نوشته شده است و ما ساختارهایی که درون هم قرار گرفته اند را نمی توانیم با یک نگاه از یکدیگر تمیز دهیم. اما با کمک دندانه گذاری ما می توانیم ساختارهای تودرتو را مشخص کنیم و خواندن برنامه را به شدت راحت تر کنیم. این تکنیک بسیار توصیه شده است پس همیشه هنگامی که حلقه و یا یک شرط را شروع می کنید، بخش های داخلی آن را یک tab یا مقداری جلوتر از خود آن حلقه یا شرط قرار دهید. همین طور برای متدها و کلاس ها هم دندانه گذاری توصیه می شود.



خیلی از محیط های توسعه مانند eclipse و intellij idea این امکانات را خودشان دارند و خوب است از آنها استفاده کنیم. حتی می توانیم کل کدی که قبلا نوشته ایم را فرمت کنیم و این دندانه گذاری را به آن اضافه کنیم.

# توضیحات (Comments)

نکته ی دیگر موضوع کامنت و توضیحات داخل برنامه است. توضیحات بخش هایی از برنامه هستند که توسط کامپایلر جاوا نادیده گرفته می شوند و کامپایلر آنها را نمی خواند و اجرا نمی کند. استفاده از آنها برای اضافه کردن توضیحات به برنامه مفید است، وقتی که می خواهید توضیحی در مورد بخش خاصی از برنامه تان بدهید، خوب است که برایش کامنتی ایجاد کنید و توضیح را در آن بنویسید تا بعدها اگر خودتان یا یکی از اعضای دیگر تیم قرار باشد آن برنامه را بخوانند، متوجه شوند چه طراحی و چه نیتی در پشت نوشتن آن برنامه بوده است و آن را خیلی راحت تر بفهمند.

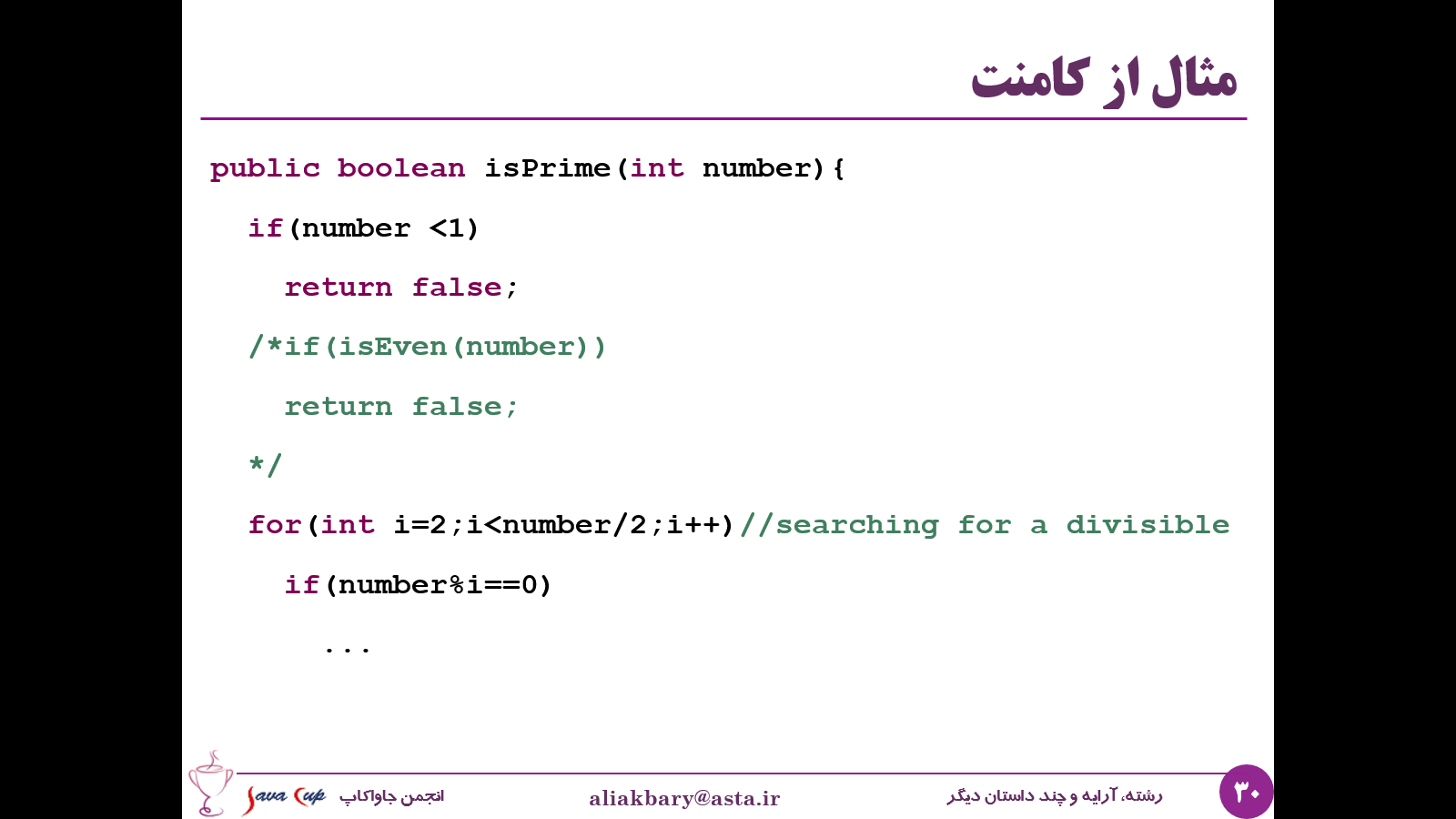
گاهی هم اگر بخواهید به صورت موقت بخشی از برنامه را حذف کنید اما نمی خواهید برای همیشه آن را حذف کنید، می توانید آن بخش را کامنت کنید.

کامنت ها دو شکل هستند، کامنت های یک خطی و چند خطی، کامنت های یک خطی با دو اسلش پشت سر هم شروع می شوند و کامنت های چند خطی با /\* شروع می شوند و با \*/ خاتمه پیدا می کنند.



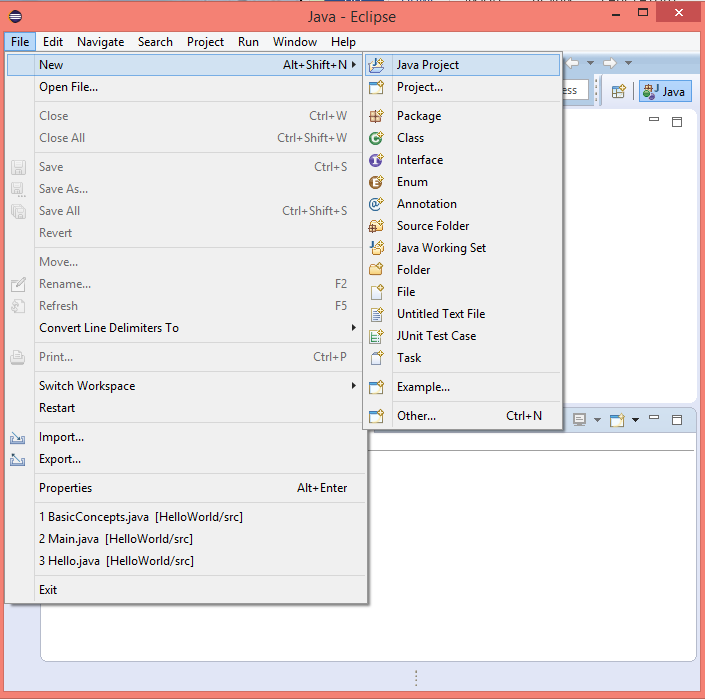
البته توصیه می شود که برنامه شما مملوء از کدهای کامنت شده نباشد و کم کم بخش هایی که کامنت کرده اید و دیگر نیازی به آنها ندارید را از برنامه پاک کنید و بیشتر از توضیحات برای توضیح برنامه تان استفاده کنید.

مثال: در مثال زیر چند خط از برنامه کامنت شده است و جلوی یکی از حلقه ها هم توضیحی داده شده است که نشان می دهد در این حلقه به دنبال چه چیزی بوده ایم. در واقع این توضیح برای خوانایی برنامه خیلی مفید است، بعدها خواهیم دید که امکانی در جاوا هست به نام java Doc برای ساختارمند کردن این توضیحات داخل برنامه ها که به صورت مشخص می توانیم برای متغیرها، متدها، کلاس ها و ساختارهای مختلف جاوا داک ایجاد کنیم. و Java Doc ها نوع خاصی از کامنت ها هستند که از روی آنها می توان مستندات مفیدی از روی متن برنامه های جاوا استخراج کرد.

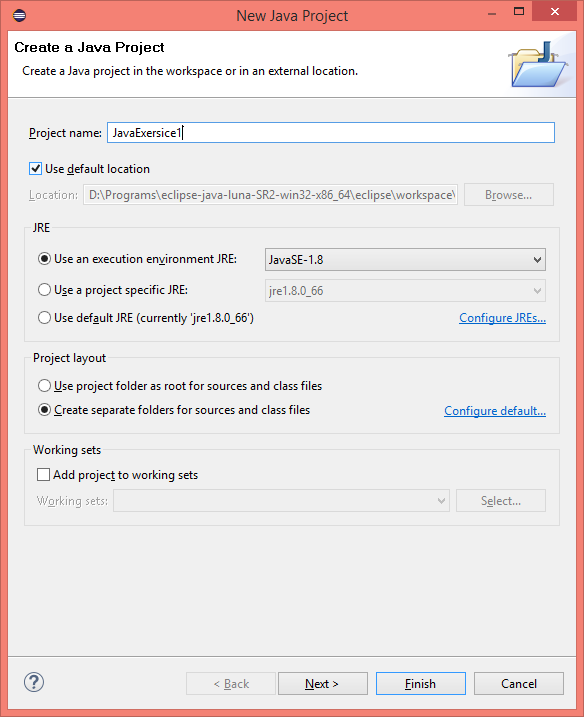


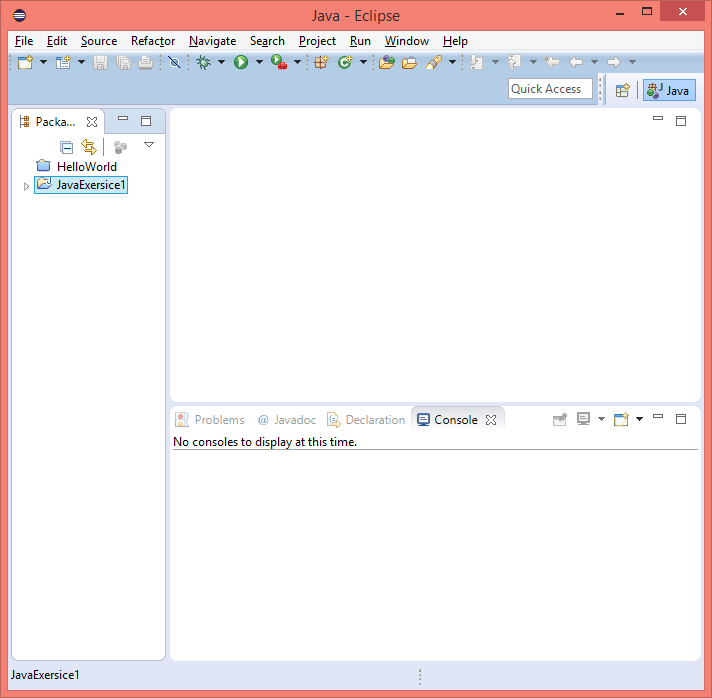
# تمرین عملی

در این بخش می خواهیم مفاهیمی که تا اینجا دیدیم را به صورت عملی مرور کنیم. در ابتدا یک پروژه ی جدید در اکلیپس می سازیم:

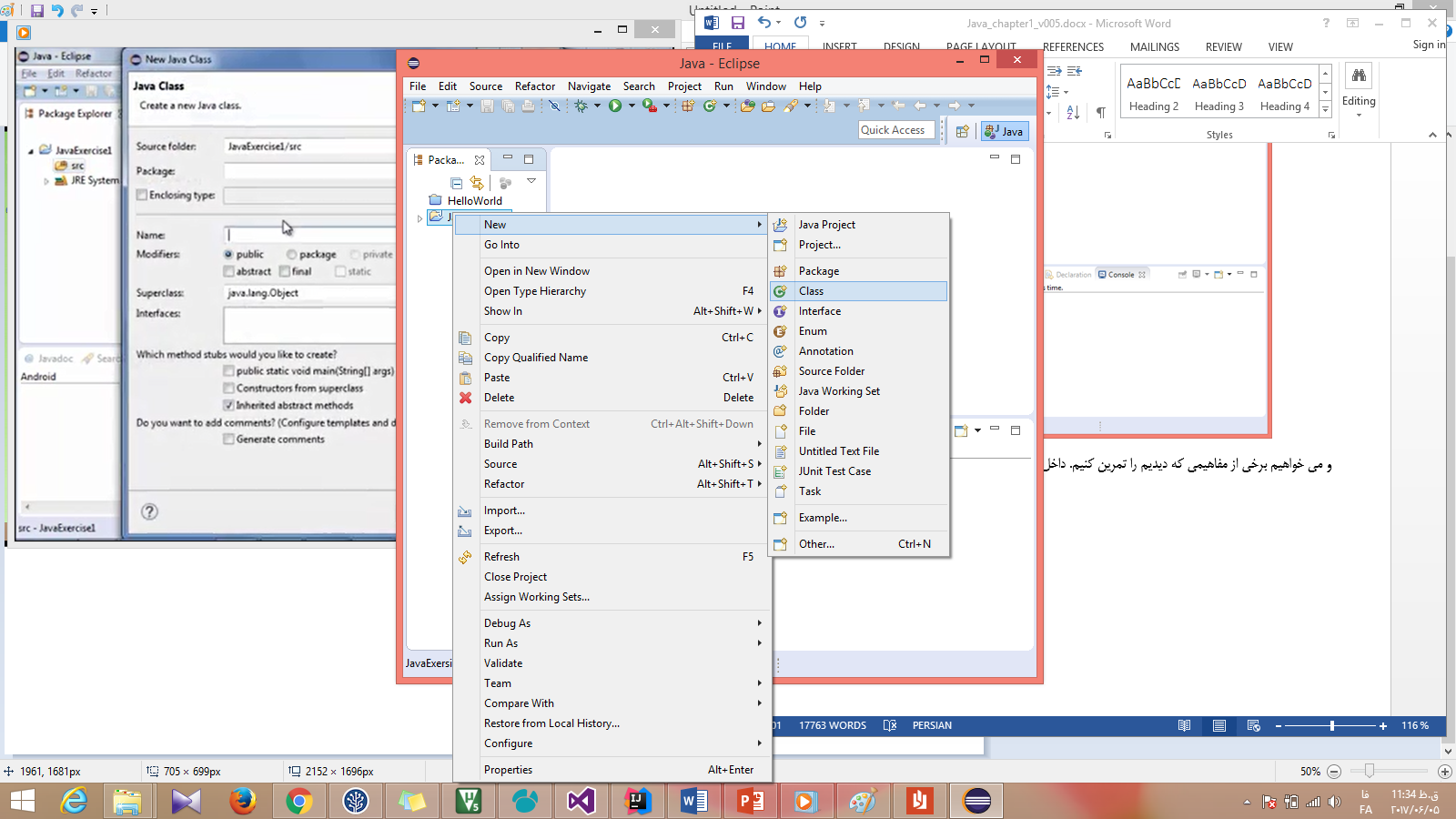


و برای آن پروژه اسمی انتخاب می کنیم:

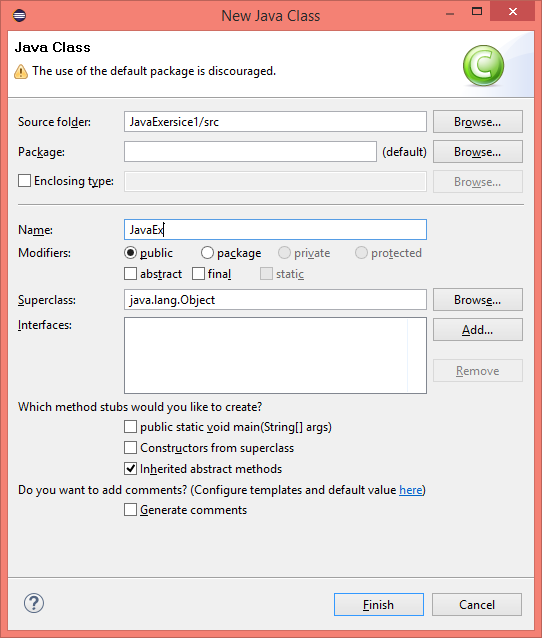




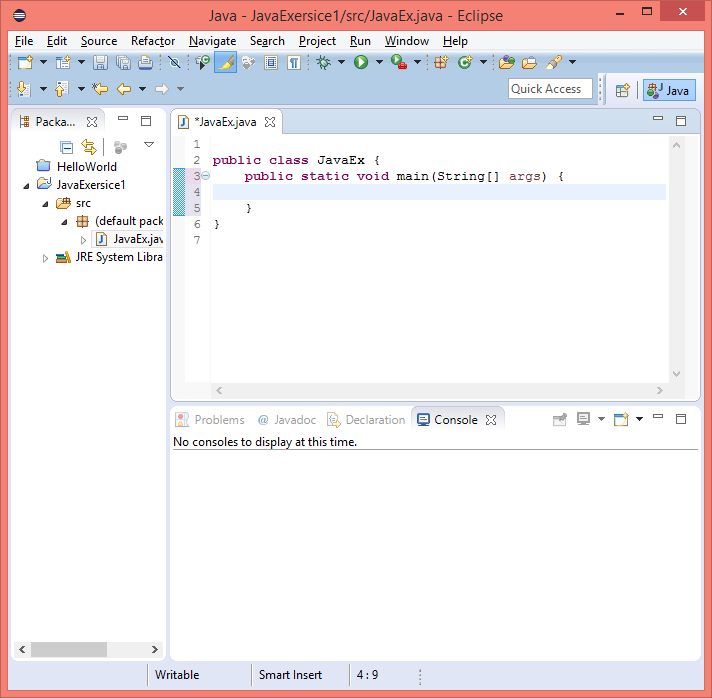
و می خواهیم برخی از مفاهیمی که دیدیم را تمرین کنیم. داخل پروژه مان به صورت زیر یک کلاس ایجاد می کنیم:



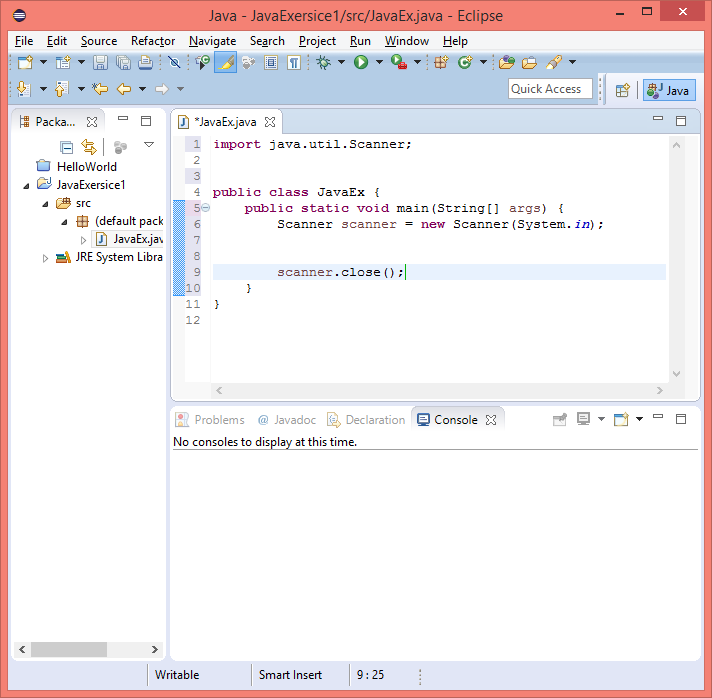
و نام آن را مشخص می کنیم:



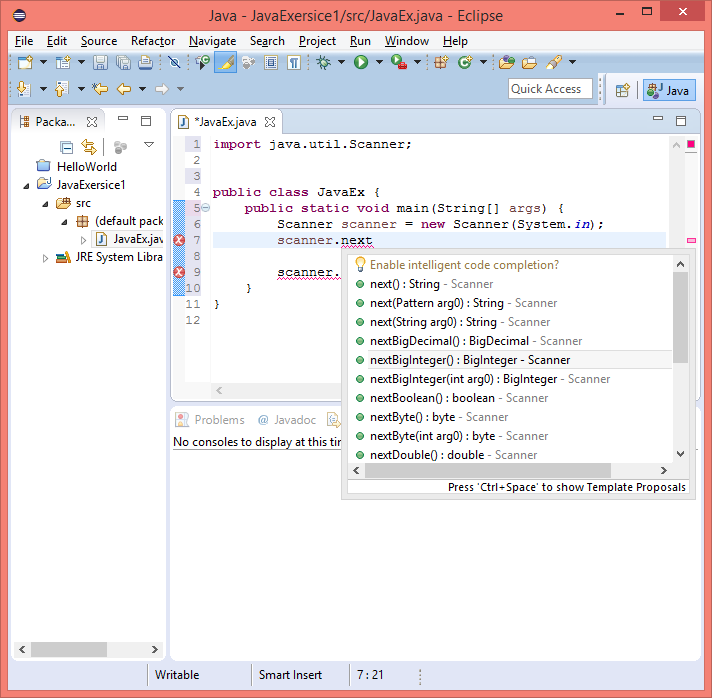
ابتدا برای کلاسمان متد main ایجاد می کنیم:



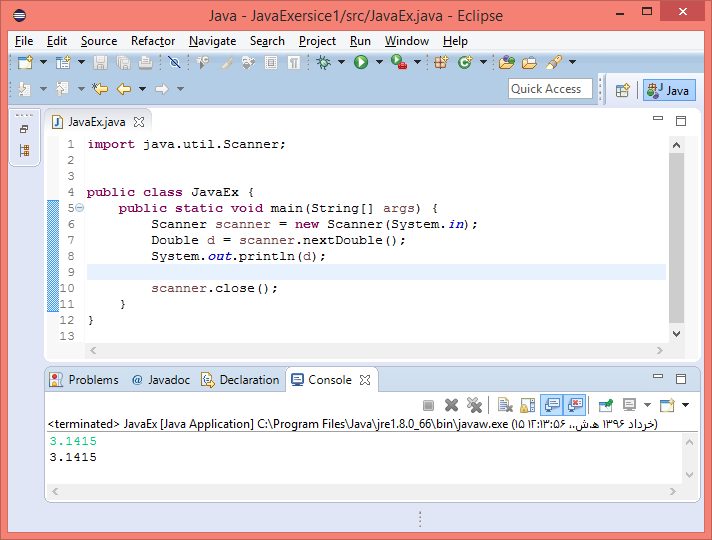
از تمرین دریافت و نشان دادن اطلاعات به کاربر شروع می کنیم، دیدیم که کلاسی داریم به نام Scanner و وقتی می خواهیم از آن استفاده کنیم باید آن را ابتدای برنامه import کنیم. پس با نوشتن کد زیر می توانیم یک اسکنر بسازیم و آن را در آخر کد ببندیم، اینکه بستن اسکنر چه می کند را در آینده خواهیم دید:



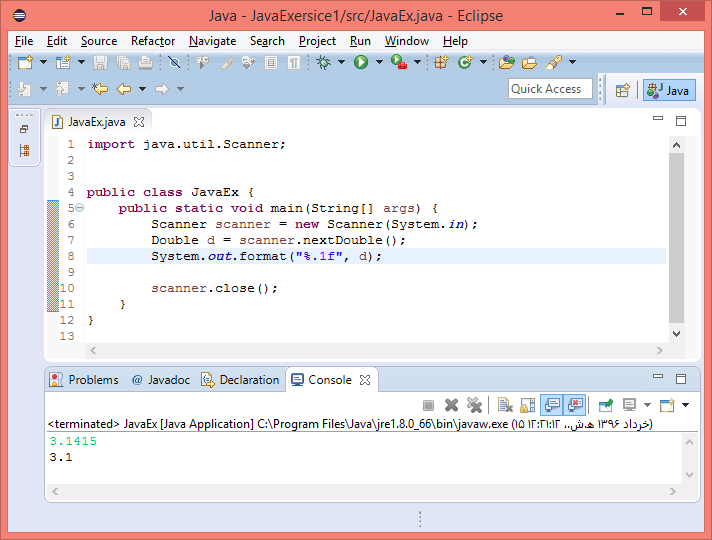
ما از اسکنر استفاده می کنیم تا مقادیری را از کاربر بگیریم، این کلاس متدهایی دارد مانند nextInt، nextBoolean، nextByte، nextFloat و ... که برای گرفتن مقادیر مختلف استفاده می شوند:



مثلا با نوشتن کد زیر می توانیم یک متغیر اعشاری را از کاربر بگیریم و همان مقدار را چاپ کنیم:

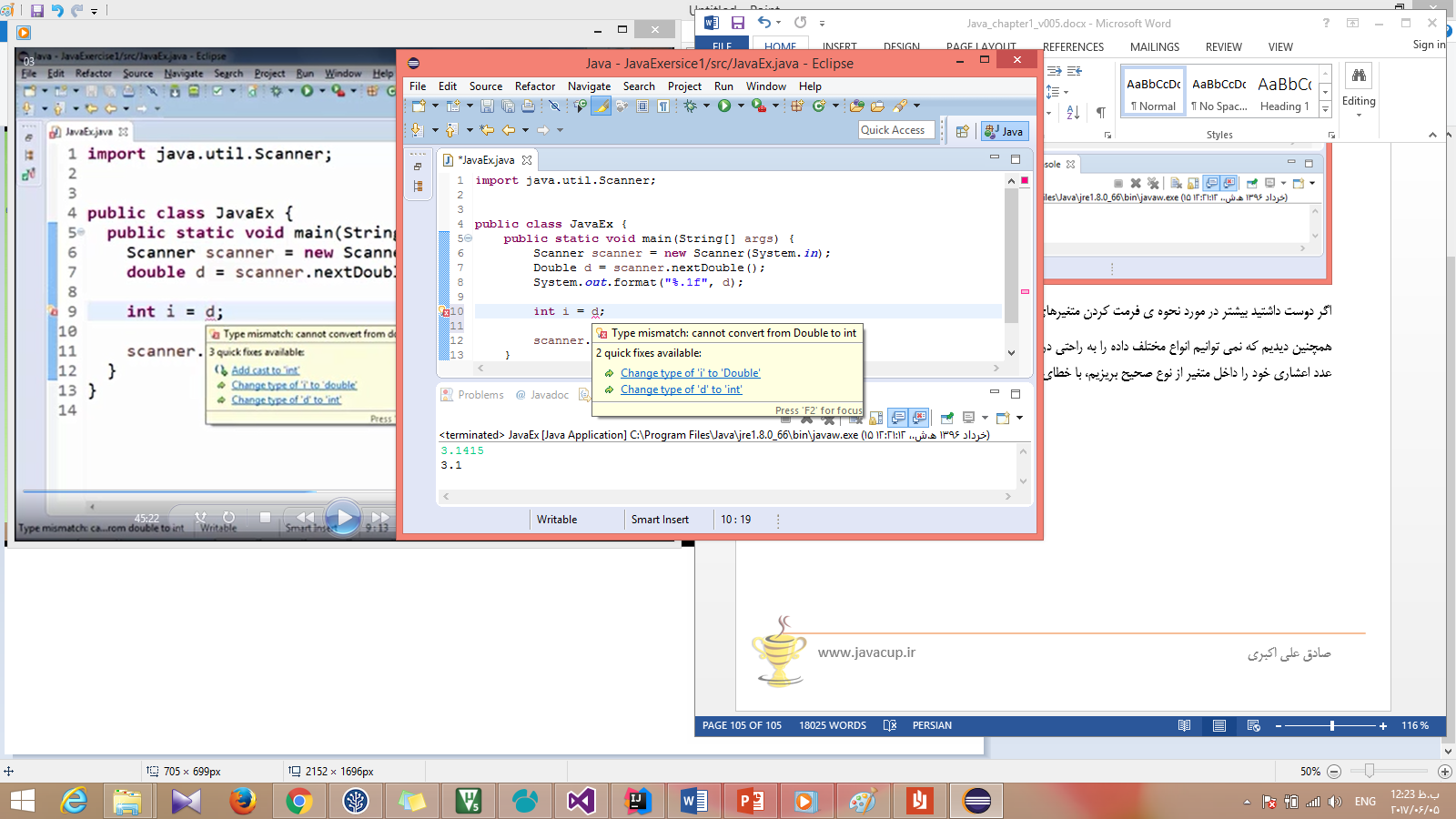


هنگامی که برنامه را اجرا می کنیم، برنامه منتظر مقدار ورودی است و این مقدار را می توانیم از طریق کنسول به برنامه بدهیم. برای چاپ کردن می توانیم از متد format هم استفاده کنیم و در این متد می توانیم جزئیات تعداد ارقام اعشاری و مانند اینها را نیز مشخص کنیم. مثلا اگر در این متد از %.1f استفاده کنیم، یعنی عدد اعشاری را با یک رقم اعشار نمایش دهد، و نتیجه ی اجرای کد زیر 3.1 خواهد بود:



اگر دوست داشتید بیشتر در مورد نحوه ی فرمت کردن متغیرهای اعشاری هنگام نمایش مطالعه کنید.

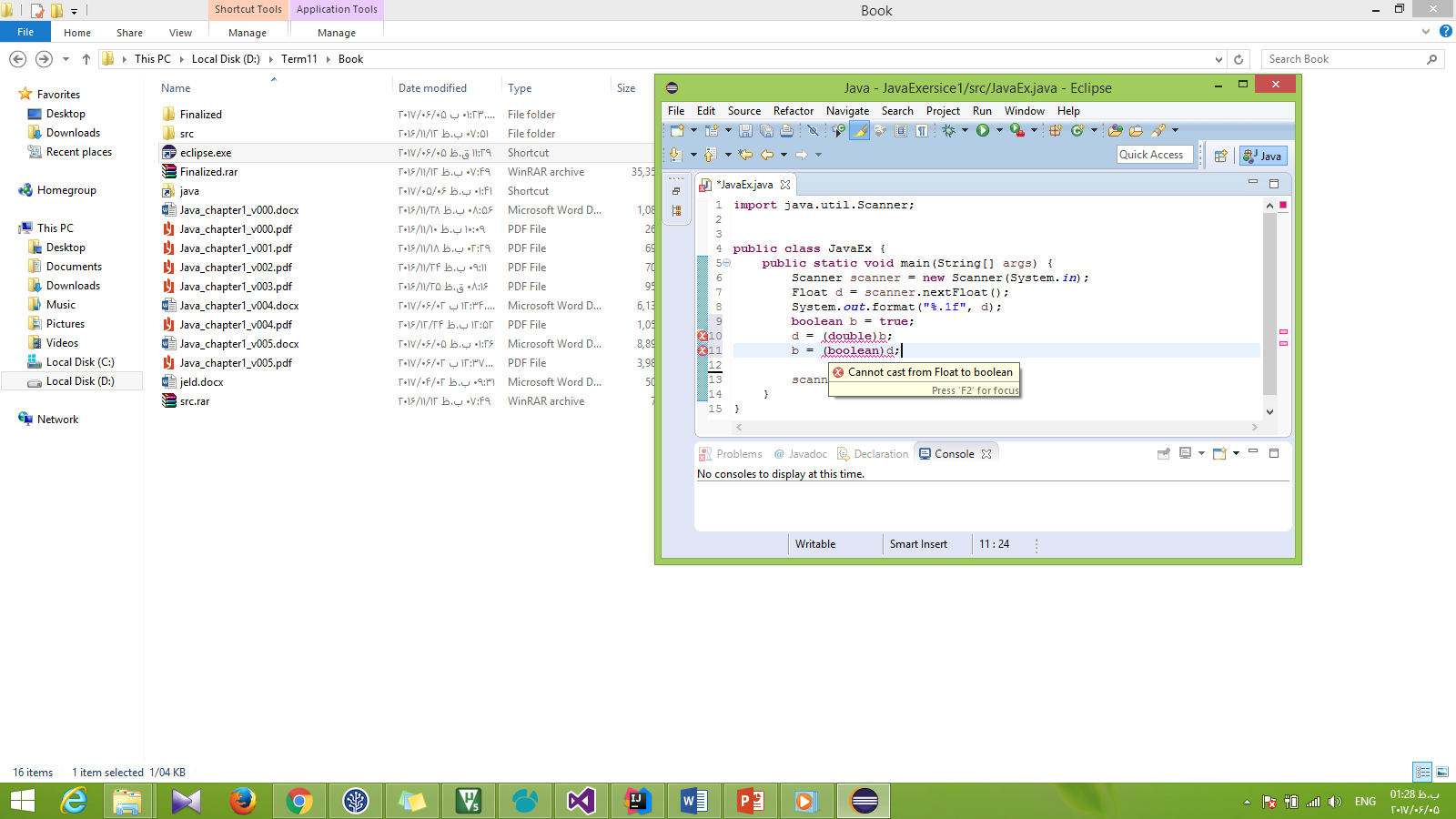
همچنین دیدیم که نمی توانیم انواع مختلف داده را به راحتی درون یکدیگر بریزیم، برای مثال اگر در این کد بخواهیم عدد اعشاری خود را داخل متغیر از نوع صحیح بریزیم، با خطای زمان کامپایل مواجه خواهیم شد:



همان طور که می بینید با خطای type mismatch مواجه شدیم که نشان دهنده ی عدم تطابق نوع هاست. در چنین شرایطی برای اینکه خطا نگیریم باید از cast استفاده کنیم:

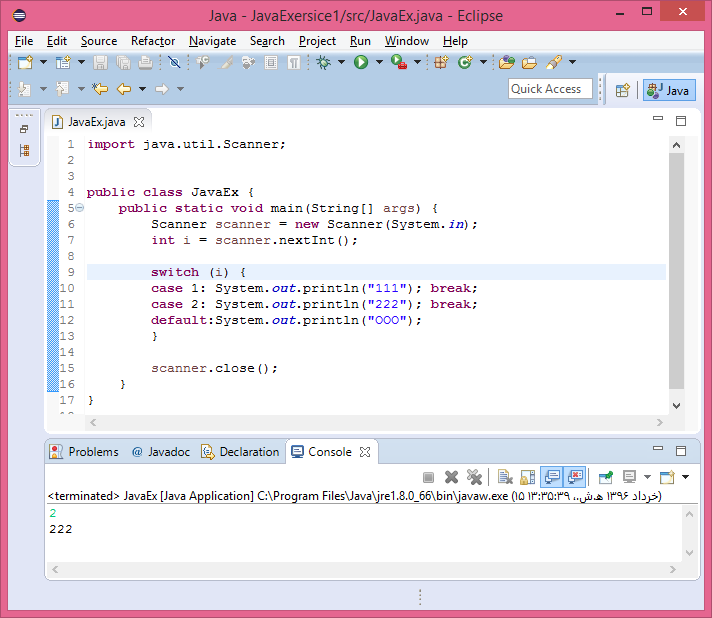
این قسمت باید با توجه به توضیحات استاد تکمیل شود

برخی تبدیل نوع ها هم اصلا ممکن نیست، برای مثال اگر ما یک متغیر از نوع Boolean ایجاد کنیم و مقداری به آن بدهیم، آنگاه این متغیر را در هیچ متغیر دیگری نمی توانیم بریزیم و حتی cast کردن هم در این مورد مجاز نیست. و از طرفی متغیرهای دیگر را نیز نمی توان به Boolean تغییر نوع داد:

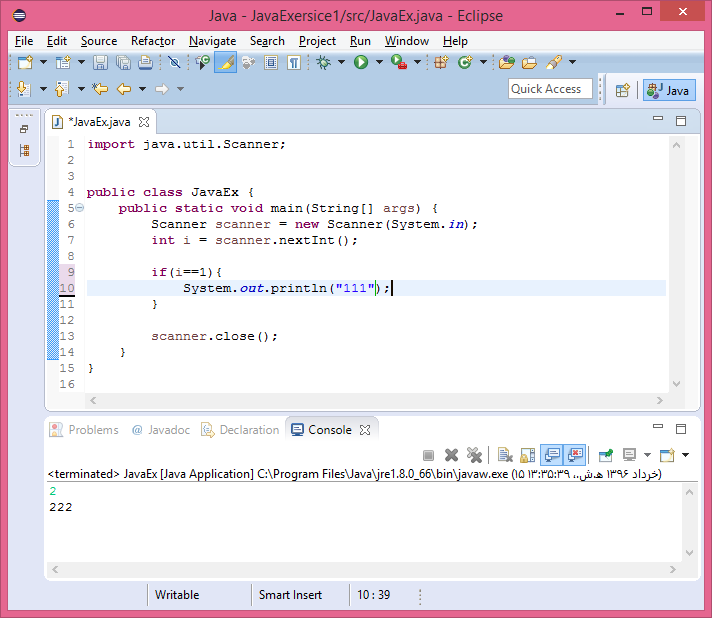


بعدها خواهیم دید که نوع های دیگری مانند رشته و آرایه نیز مانند Boolean امکان تبدیل نوع به داده هایی مانند integer و یا double را ندارند.

در این بخش با ساختار دیگری آشنا شدیم به نام سویچ، فرض کنید متغیری که از کاربر دریافت می کنیم یک integer باشد، در این صورت می توانیم روی آن یک switch تعریف کنیم و برای آن case های مختلفی بنویسیم، مثلا اگر 1 بود چند یک چاپ کنیم و در انتها حتما باید از Break استفاده کنیم. و دیدیم که کلمه کلیدی default هم وجود دارد که در صورتی که هیچ یک از case ها درست نباشند و اجرا نشوند بخش پیش فرض اجرا می شود. در پایان default هم اجباری به گذاشتن break نیست چون دستور دیگری بعد از آن وجود ندارد که بخواهد اجرا شود:

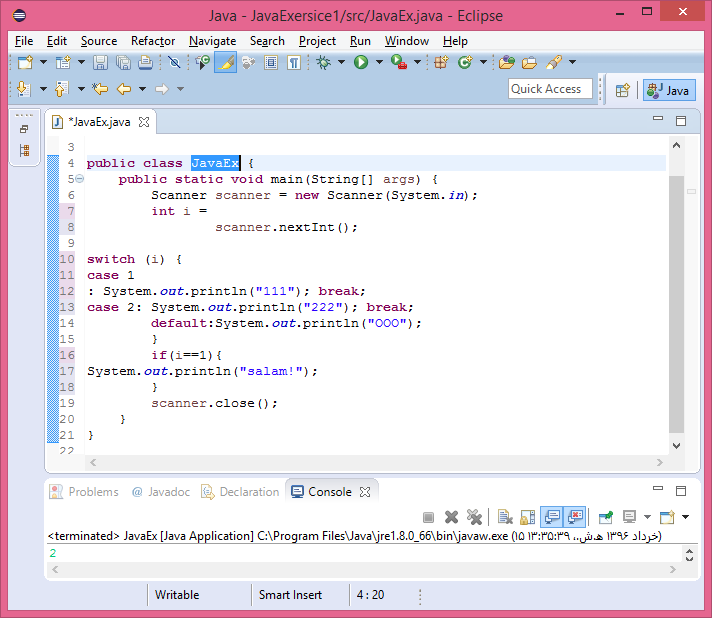


و چیز دیگری که در مورد آن صحبت کردیم بحث دندانه گذاری بود، اگر دقت بکنید کدهایی که در این بخش نوشتیم دارای indentation هستند و فرورفتگی هایی در آن دیده می شود، مثلا بدنه ی کلاس یک tab جلوتر از تعریف کلاس قرار گرفته است، همین طور وقتی که یک متد شروع شده است، بدنه ی متد مقداری جلوتر قرار گرفته است و یا اگر یک شرط بنویسیم یا یک حلقه، خوب است بدنه ی آن را یک tab جلوتر قرار دهیم:

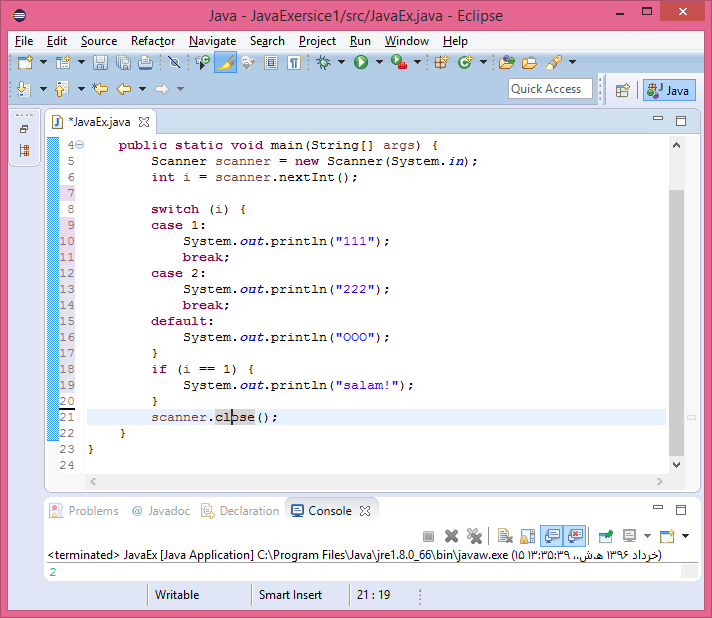


حال اگر برنامه ما دندانه گذاری درستی نداشته باشد و به هم ریخته باشد، خود محیط های توسعه امکانی برای مرتب کردن آن فراهم کرده اند. در اکلیپس این کار با فشردن ctrl+shift+f قابل انجام است. و البته لازم نیست این موارد را حفظ کنید و همواره با یک سرچ ساده می توانید آنها را بدست آورید.

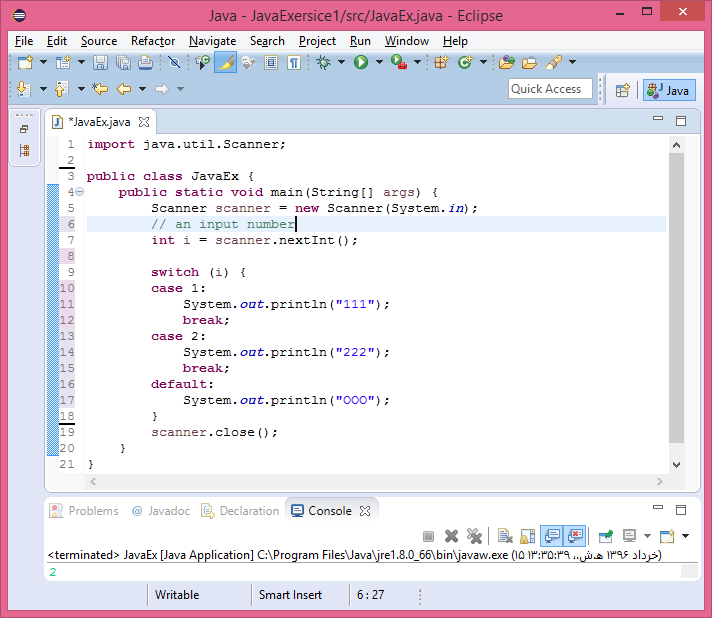
کد قبل از مرتب کردن و دندانه گذاری درست:



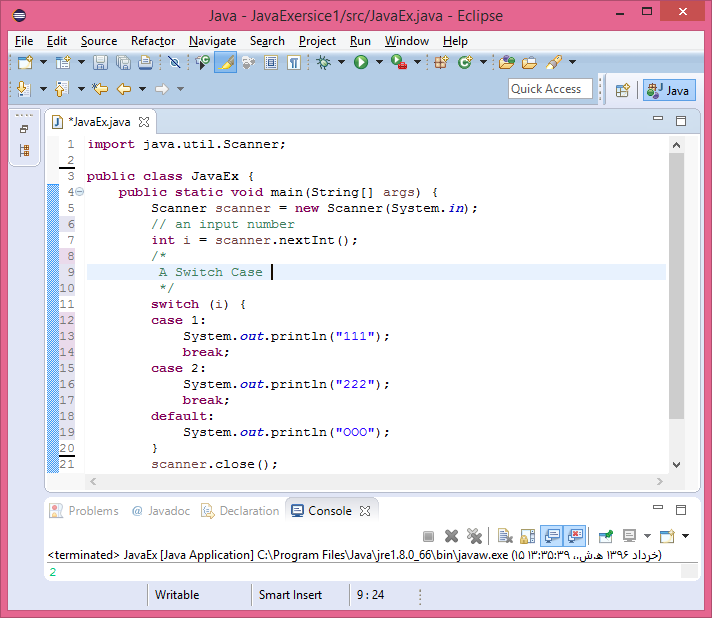
کد بعد از فشردن کلید ترکیبی ctrl + shift + f:



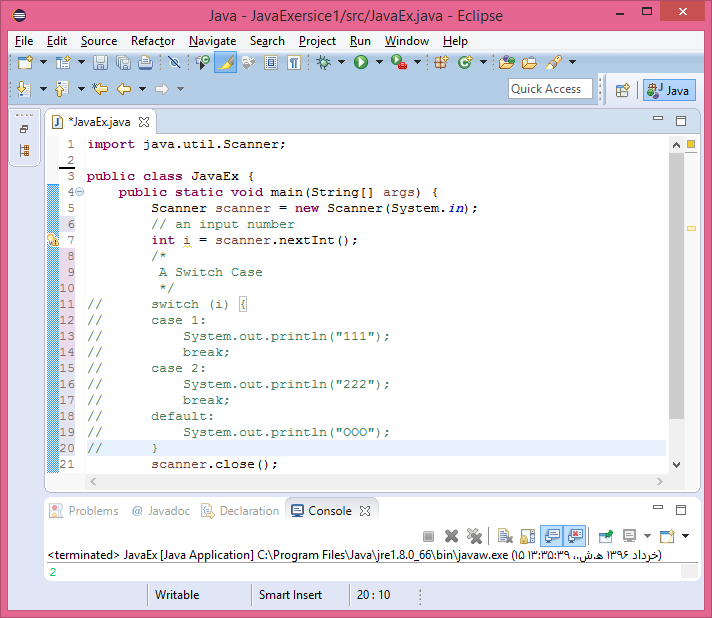
نکته ی دیگری که در موردش صحبت کردیم بحث کامنت ها بود، گفتیم که می توانیم با دو اسلش (//) کامنت را شروع کنیم و در کامنت هر چیزی که نوشته شود حتی اگر بی معنی و نا مشخص باشد موجب خطایی نخواهد شد چون این بخش از دید کامپایلر نادیده گرفته می شود. و دیدیم برای توضیح دادن در مورد کدهایی که نوشتیم می توانیم از کامنت ها استفاده کنیم.



و دیدیم که می توانیم کامنت های چند خطی نیز داشته باشیم و در میان /\*\*/ هر چیزی که نوشته شود کامپایل نخواهد شد و می توانیم از آن برای توصیف عملکرد کد خود استفاده کنیم:



و البته می توانیم کدهایی که قبلا نوشته ایم را نیز کامنت کنیم، مثلا:

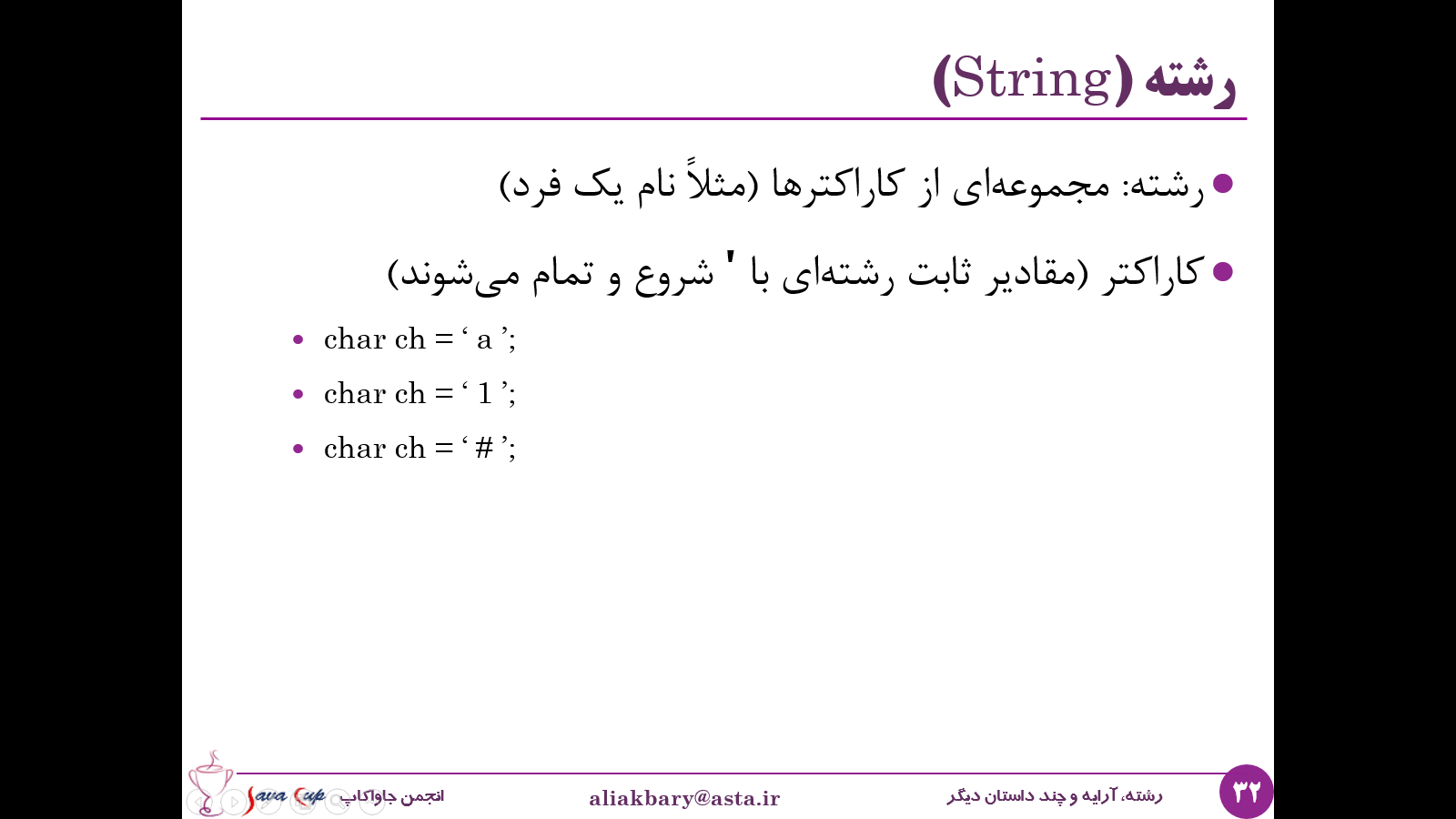


تا توسط جاوا اجرا نشود و مانند توضیحات با آن برخورد شود.

# رشته (String)

در این بخش درباره ی رشته ها صحبت می کنیم. رشته یک نوع داده ی جدید هست که با همه ی انواع داده ای که قبلا دیده ایم متفاوت است. نوع داده ای که می تواند مجموعه ای از کاراکترها را در خودش نگهدارد، مثلا متغیری از نوع String می تواند نام یک فرد، یک آدرس و یا چیزهایی مشابه را ذخیره کند.

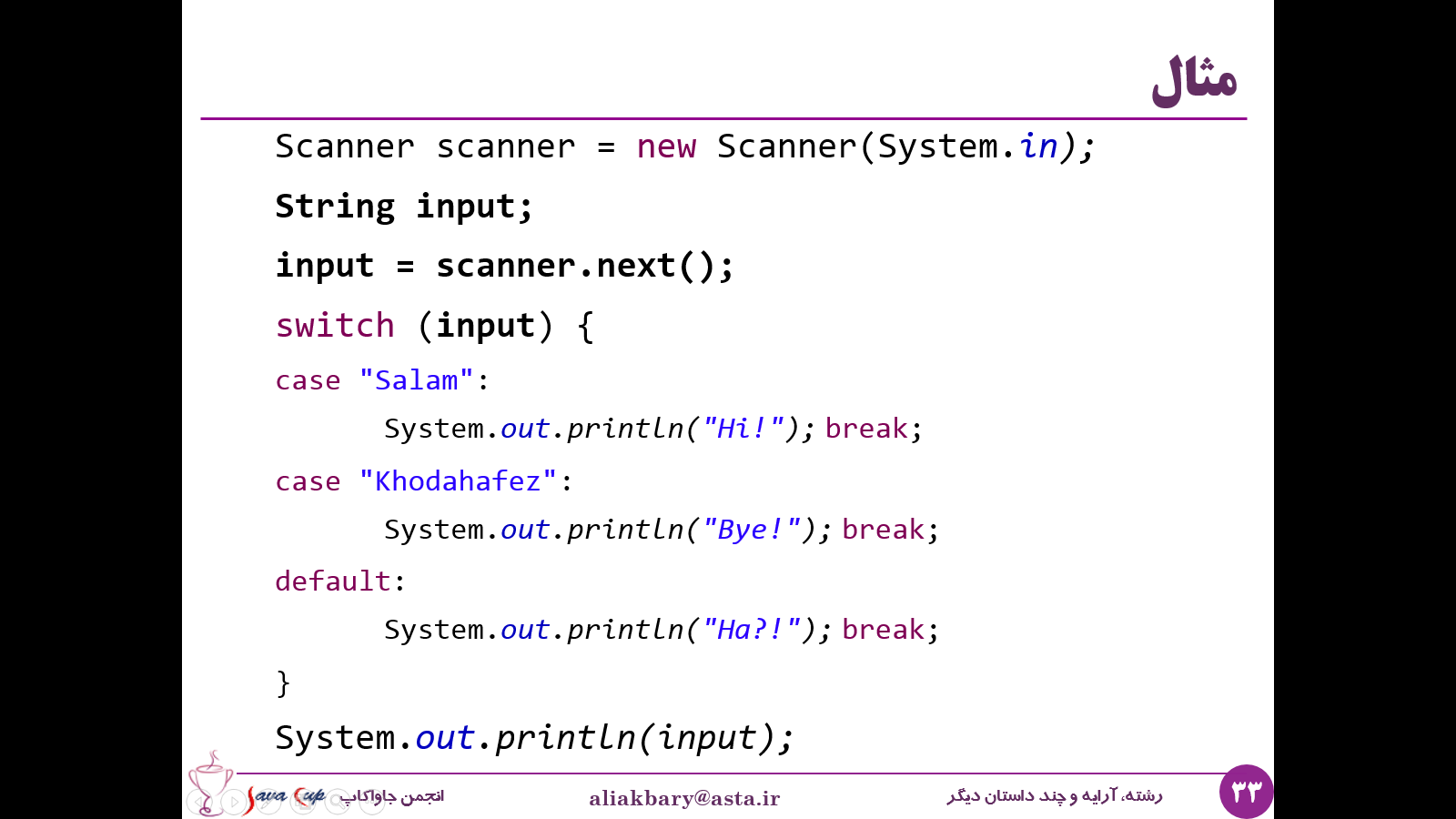
همان طور که قبلا دیدیم کاراکتر متغیری هست که مقادیر ثابت آن با یک کوتیشن (') شروع و با یک کوتیشن هم پایان پیدا می کنند و یک کاراکتر می تواند یک حرف، یک عدد و یا یک سمبل را شامل شود:



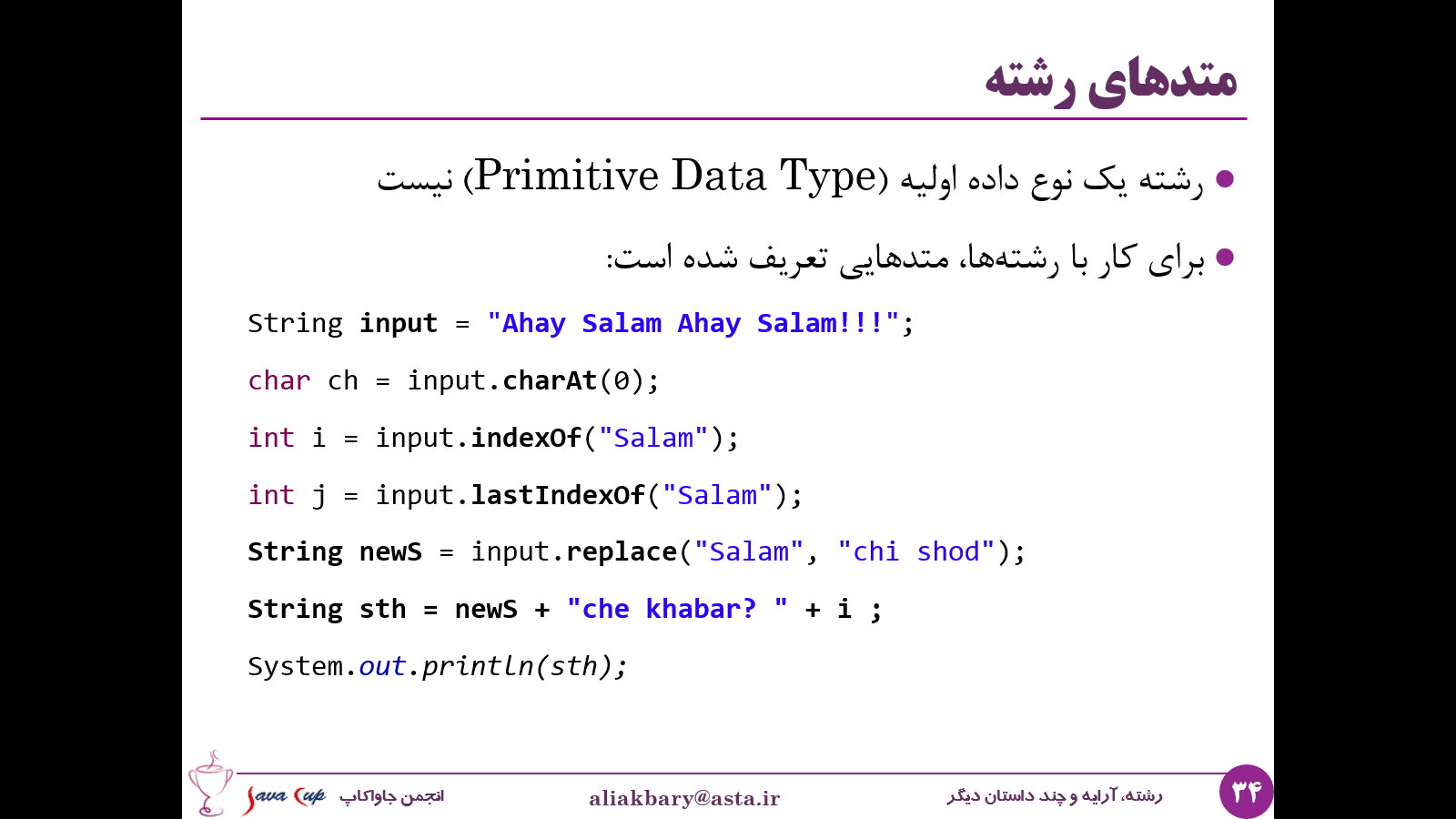
اما رشته ها با دابل کوتیشن شروع و پایان می یابند و می توانند مجموعه ای از کاراکترها را نشان دهند مثلا متغیری با نام St و از نوع String که مساوی مقدار "Ali" قرار گرفته است و یا:



بنابراین رشته مجموعه ای از صفر یا بیشتر کاراکتر هست که در کنار هم قرار می گیرند و با دابل کوتیشن شروع و پایان می پذیرند. به عنوان یک مثال تکه کد زیر را ببینید، در این کد به کمک Scanner از ورودی مقداری می خوانیم که این بار این مقدار از جنس String است و با کمک متد next خوانده می شود (خروجی متد next از جنس String است). و بعد مثلا روی آن یک switch اجرا می کنیم. همان طور که می دانید در Switch از رشته ها هم می توانیم استفاده کنیم و این امکانی است که از جاوای 7 به بعد فراهم شده است:



رشته یک نوع داده ی اولیه نیست (primitive data type نیست). در واقع رشته اولین نوع داده ای غیر اولیه ای هست که با آن آشنا می شویم و بعدها خواهیم دید که یک کلاس است. برای کار با رشته متدهایی تعریف شده است (کلاس رشته دارای متدهایی است) که می توانیم از آنها استفاده بکنیم. مثال زیر را در نظر بگیرید:



در مثال بالا متغیری از جنس رشته تعریف کرده ایم و مقداری را به آن انتساب داده ایم، حال می توانیم با استفاده از نام متغیر (input) و سپس دات و نام متد عملیات مختلفی را روی این رشته انجام دهیم. برخی از این متدها را در مثال بالا مشاهده می کنید. مثلا input.charAt که کاراکتر با ایندکس مشخص را برمی گرداند، در این مثال خروجی این متد (مقدار ch) کاراکتر صفرم رشته ی مورد نظر یعنی A خواهد بود (شماره گذاری کاراکترها از صفر شروع می شود پس کاراکتر صفرم همان اولین کاراکتر رشته ی ما خواهد بود). یا متد input.indexOf که یک رشته به عنوان پارامتر می گیرد و مشخص می کند این رشته در چندمین کاراکتر متغیر مربوطه دیده شده است، در اینجا رشته ی "Salam" در 5امین کاراکتر متغیر input برای بار اول مشاهده شده است پس مقدار i برابر 5 خواهد شد. و یا متد LastIndexOf که آخرین رخداد یک رشته را برمی گرداند، در واقع متد indexOf اولین رخداد یک رشته و این متد آخرین رخداد آن را بر می گرداند. پس در اینجا مقدار متغیر j برابر 16 خواهد شد. و یا متد replice که دو رشته را می گیرد و هر بار رشته ی اول را در متغیر مورد نظر ببیند آن را با رشته ی دوم جایگزین می کند، بنابر این خروجی این متد یا مقدار متغیر news بعد از اجرا برابر "Ahay chi shod Ahay chi shod!!!" خواهد بود. نکته ی دیگر استفاده از عملگر جمع برای رشته هاست، معنای این عملگر برای رشته ها متفاوت است، برای رشته ها استفاده از این عملگر به معنای به هم چسباندن رشته هاست، مثلا رشته ی news را می توانیم جمع کنیم با رشته ی "che khabar? " و حتی جمع کنیم با یک عدد مانند i . اصطلاحا بعد از انجام این دستور این رشته های به همدیگر concatenate می شوند و خروجی خط آخر تکه کد برابر Ahay chi shod Ahay chi shod!!!che khabar? 5 خواهد بود.

در مورد رشته ها جزئیات بیشتری می توان گفت که ما فعلا از آنها چشم پوشی می کنیم و در بحث برنامه نویسی شی گرا مفصل تر به آن خواهیم پرداخت.

کلاس رشته دارای متدهای دیگر نیز هست که در ادامه به برخی از آنها به صورت گذرا اشاره خواهیم کرد:

* Cancat

عملکرد این متد مشابه عملگر جمع در رشته هاست و رشته ها را به یکدیگر می چسباند.

* StartsWith
* endsWith

این دو متد به ترتیب با گرفتن رشته ای در رودی چک می کنند آیا رشته ی مورد نظر ما با رشته ی ورودی شروع شده و یا پایان یافته است با نه و مقدار خروجی آنها از جنس Boolean است.

* substring

از این متد می توانیم برای بیرون کشیدن بخشی از رشته استفاده کنیم، به این صورت که ایندکس شروع و طول زیر رشته ای که می خواهیم را به آن پاس می کنیم و این متد آن بخش از رشته را خروجی می دهد.

* Length

این متد طول رشته را مشخص می کند، در واقع مشخص می کند رشته ما از چند کاراکتر تشکیل شده است.

* Equals
* equalsIgnoreCase

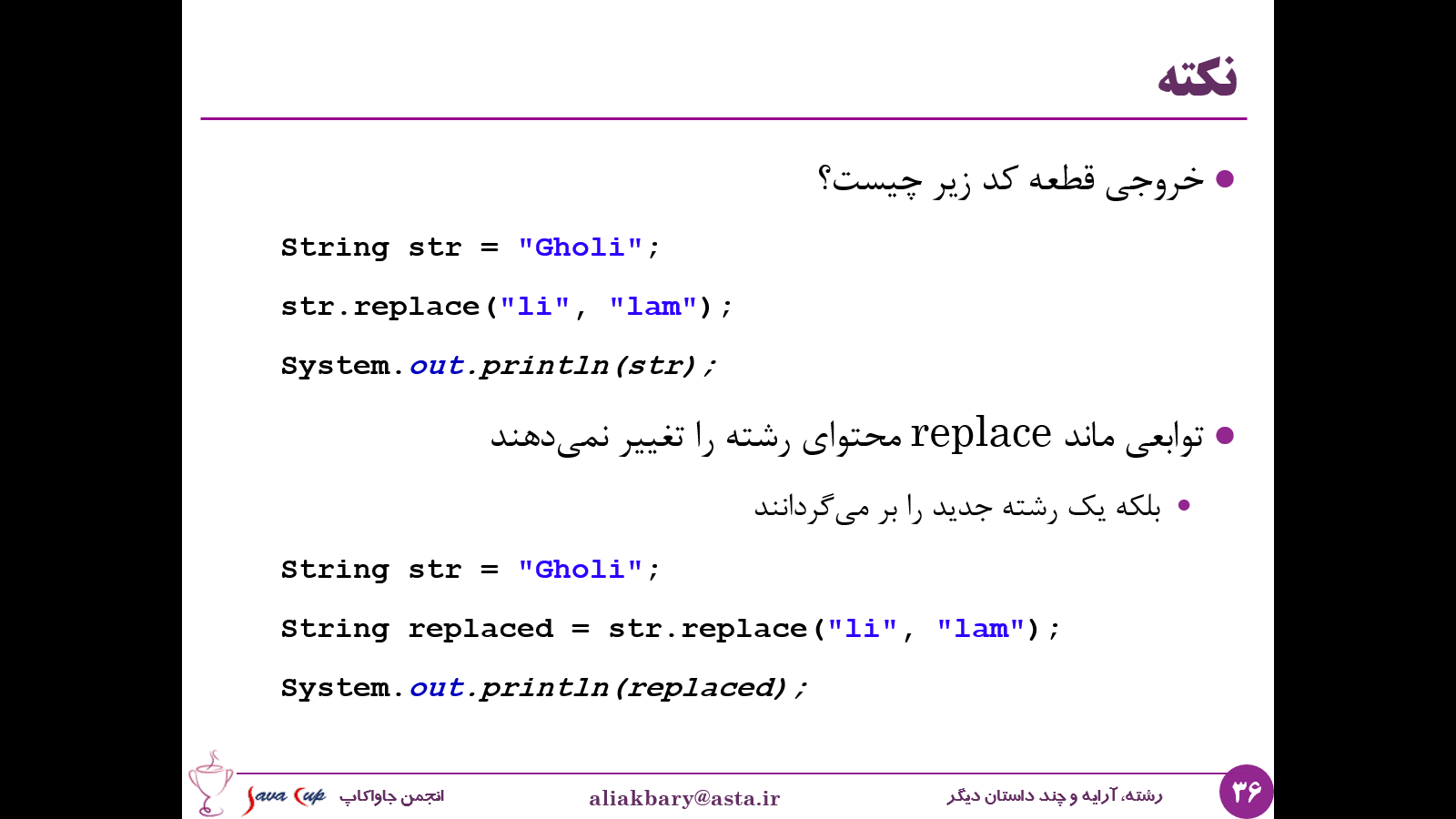
این دو متد برای مقایسه ی رشته ها استفاده می شوند. برای اینکه مقایسه کنیم که آیا دو متغیر از جنس رشته مقداری مساوی دارند یا نه از equals استفاده می کنیم و در صورتی که بزرگ و کوچک بودن حروف برای ما اهمیتی نداشته باشد از متد دوم استفاده خواهیم کرد.

بعدها این موضوع را خواهیم دید که برای مقایسه ی دو رشته استفاده از اپراتور == که برای مقایسه ی انواع داده ی اولیه استفاده می شود کار درستی نیست و نتیجه ی مطلوبی ندارد.

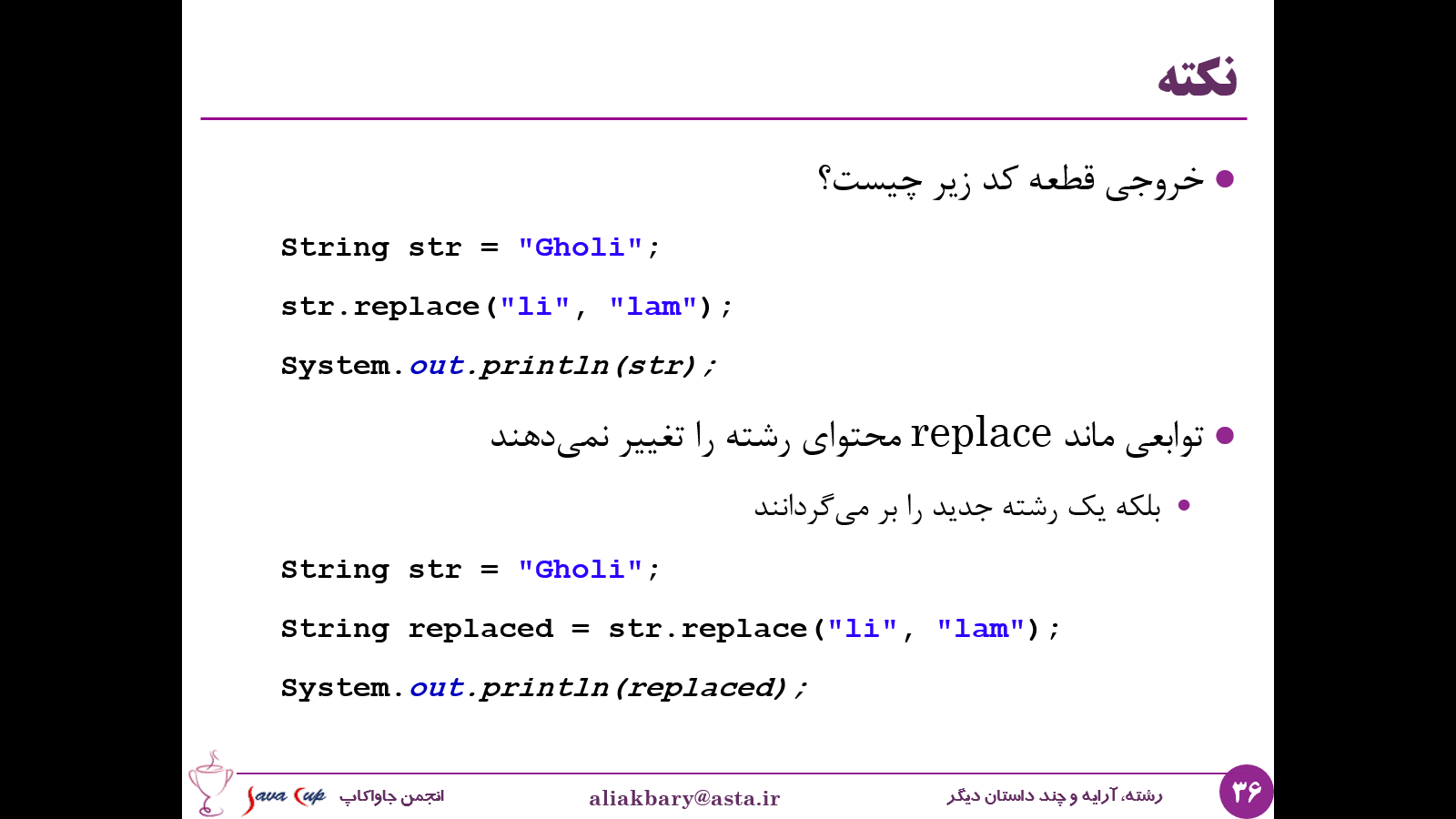
* Contains

این متد مشخص می کند که آیا رشته ای که به آن پاس کرده ایم در رشته ی اصلی وجود دارد یا نه و این متد هم یک مقدار true یا false برمی گرداند.

نکته: به این نکته دقت کنید که متد replace رشته ی اصلی ما را تغییر نمی دهد. به مثال زیر دقت کنید.



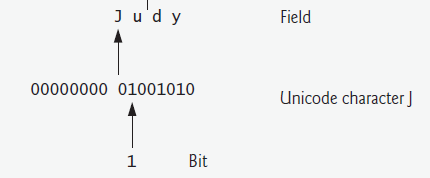
در این مثال رشته ی اصلی ما مقدار "Gholi" را دارد. اگر متد replace با ورودی های زیر را روی آن فراخوانی کنیم و در واقع اعلام کنیم که می خواهیم هر جا مقدار li دیده ی شد با مقدار lam جایگذاری شود و بعد از اجرای این دستور مقدار str را چاپ کنیم، مقدار این متغیر تغییر نخواهد کرد و همان "Gholi" باقی خواهد ماند، علت این موضوع این است که همانطور که گفتیم متد replice مقدار متغیر اصلی را تغییر نمی دهد بلکه رشته ی تغییر یافته ای ایجاد می کند و آن را به عنوان خروجی برمی گرداند. پس باید خروجی این متد را داخل متغیر دیگری و یا حداقل همان متغیر اصلی (در اینجا str) بریزیم. مثلا کد را به صورت زیر اصلاح کنیم:



خروجی این برنامه برابر "Gholam" خواهد بود.

# سلسله مراتب داده ها

می دانیم که متغیرهای ما در حافظه به صورت مجموعه ای از بیت ها ذخیره می شوند و این بیت ها در کنار هم بایت ها را می سازند و چند بایت در کنار هم یک کاراکتر را می سازد و کاراکترها در کنار هم کلمه ها و رشته ها را می سازند. پس سلسله مراتب داده ها از بیت ها شروع می شود تا به کلمه ها و رشته ها برسیم.



# جاوا و یونیکد

یک استاندارد برای بازنمایی کاراکترها وجود دارد به نام یونیکد که در واقع یک مجموعه از کارکترها را شامل می شود(character set) . در گذشته یک روش و استاندارد قدیمی تر به نام ASCII وجود داشت که مجموعه ی کاراکترهای بسیار محدودی را می توانست بازنمایی کند اما یونیکد سعی می کند تمام کاراکترهای موجود در تمام زبان ها را بازنمایی کند، مثلا کاراکترهای فارسی، انگلیسی، ژاپنی، چینی و دیگر زبان ها، همچنین اعداد و سمبل هایی که در کاربردهای مختلف استفاده می شوند.

روش های مختلفی برای کد گذاری یونیکد وجود دارد که به هر کدام از این روش ها یک Encoding گفته می شود که جاوا از آنها پشتیبانی می کند؛ روش هایی مانند UTF8، UTF16 و UTF32 . به این تفاوت ها دقت کنید و اگر دوست داشتید بیشتر دربارش بخوانید که character set و encoding چی هستند.

در گذشته روش های دیگری نیز برای کاربردهای فارسی وجود داشته است مانند Windows-1256 که در حال حاضر جای خود را به یونیکد داده است.

# کاراکترهای ویژه در جاوا

کاراکترهای ویژه ای در جاوا وجود دارند که آنها را با کمک یک back slash (\) نشان می دهیم، در واقع در این مواقع مجموع بک اسلش و کاراکتر بعدی با همدیگر نشان دهنده ی یک کاراکتر هستند. مثال:

* \n به معنای new line برای انتقال به خط جدید
* \t برای نمایش کاراکتر tab
* \" برای نمایش کاراکتر "
* \' برای نمایش کاراکتر '
* \\ برای نمایش کاراکتر \

برای مثال رشته ای مانند رشته ی رو به رو هنگام چاپ نتیجه ی زیر را خواهد داشت:

String s = "He said \"OK!\" and left";

He said "OK!" and left

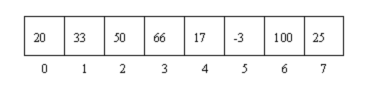
اگر back slash را در این مثال قبل از " قرار نمی دادیم، جاوا فکر می کرد که قصد بستن و اتمام رشته مان را داریم و دچار ابهام می شد.

و یا مثال دیگری:

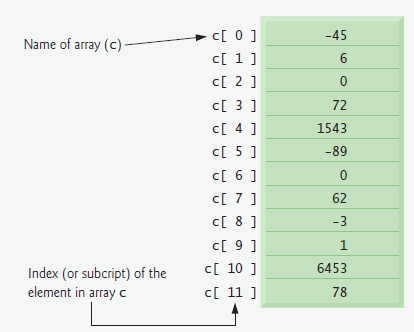


# آرایه

موضوع مهم بعدی آرایه (Array) است. آرایه ها نوع دیگری از داده ها هستند که می توانند مجموعه ای از مقادیر را در خودشان نگهداری کنند. در واقع آرایه مجموعه ای از داده های مرتبط به هم و یک نوع است. مثلا مجموعه ای از integer ها، مجموعه ای از Stringها و همین طور الی آخر. آرایه طول ثابتی دارد که هنگام ساخته شدن مشخص می شود. به داده های تشکیل دهنده ی یک آرایه عناصر و یا element ها گفته می شود. برای مثال همان طور که در شکل زیر مشاهده می کنید یک آرایه از integer مانند چند integer هست که در کنار هم قرار گرفته است.



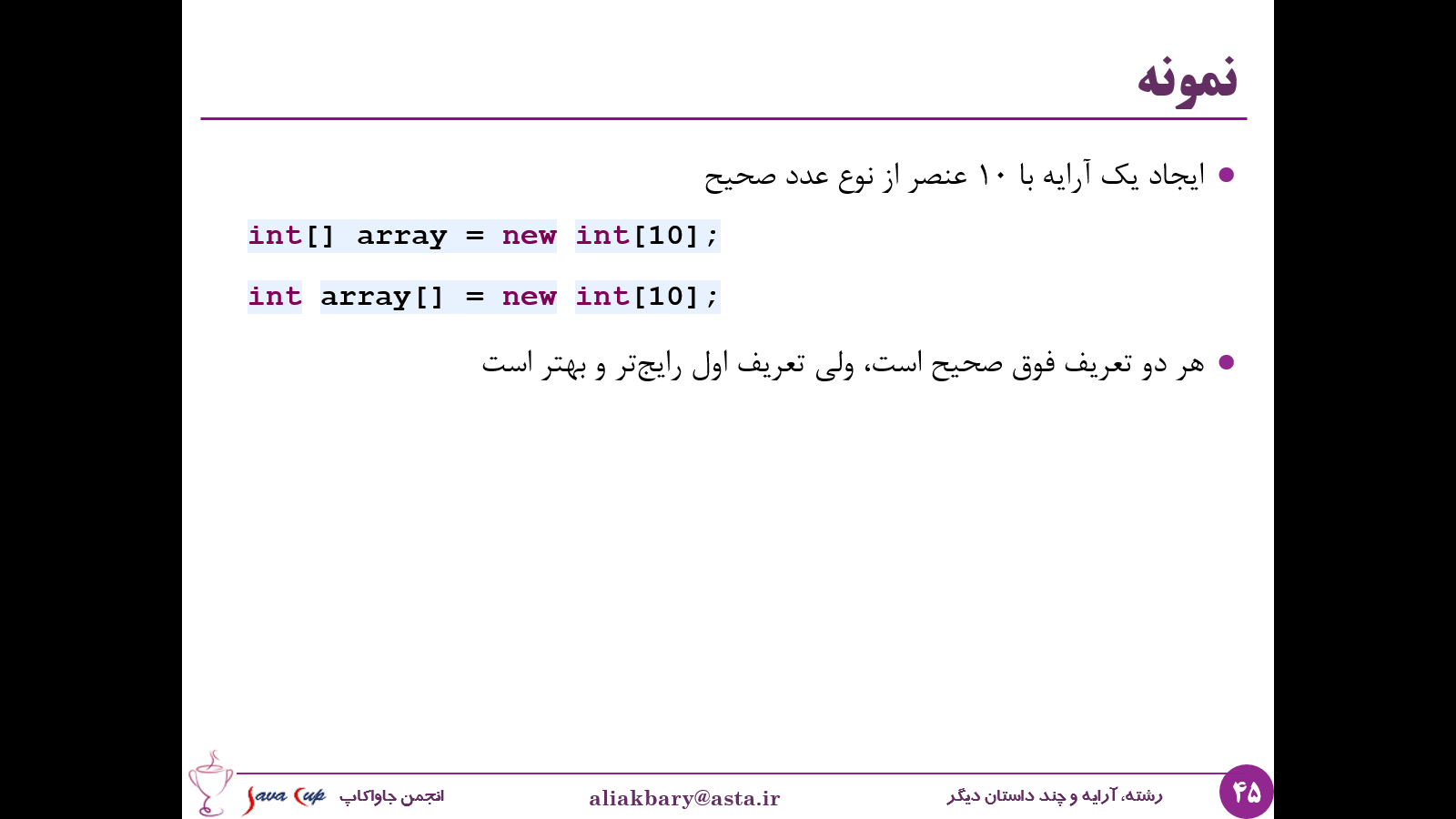
نکته ی مهم در مورد آرایه ها این است که همه ی مقادیر آنها یک نوع دارد و برای دست رسی به هر عنصر آرایه از اندیس محل قرار گیری آن عنصر در آرایه استفاده می شود. (اندیس یا ایندکس) مثلا اگر بخواهیم به خانه ی شماره 3 از آرایه دست رسی پیدا کنیم، در واقع باید به اندیس شماره 3 از آرایه مراجعه کنیم. عناصر آرایه معمولا به ترتیب اندیس ها در خانه های حافظه نگهداری می شوند، اما کل آرایه یک نام واحد دارد، پس با کمک نام آرایه و اندیس ها می توانیم به خانه های مختلف آن دست رسی پیدا کنیم.



برای مثال در شکل بالا یک آرایه از اعداد صحیح را می بینید که نام آن c است و اندیس هر خانه در کنار آن خانه نوشته شده است. نکته دیگر در مورد آرایه ها در زبان جاوا این است که اندیس عناصر آرایه در این زبان از صفر شروع می شود (مطابق شکل بالا) پس اولین خانه آرایه ی شکل بالا c[0] خواهد بود. مجموعه ی اسم آرایه و داخل کروشه اندیس آن نشان دهنده ی یکی از عناصر آرایه خواهد بود.

# نحوه ی ایجاد آرایه

به مثال زیر توجه کنید:



در هر دو خط ما آرایه ای با نوع integer و طول 10 تعریف و ایجاد کرده ایم. البته مدل اول بین برنامه نویسان جاوا رایج تر است و مدل دوم بیشتر برای تطابق با زبان های قدیمی تر مانند C قرار داده شده است. و بهتر است از مدل اول استفاده کنیم چرا که کل data type را در آن یک جا تعریف کرده ایم. همچنین اگر بخواهیم آرایه ای به طول n بسازیم (n یک متغیر باشد) می توانیم داخل کروشه به جای عدد از متغیر استفاده کنیم:



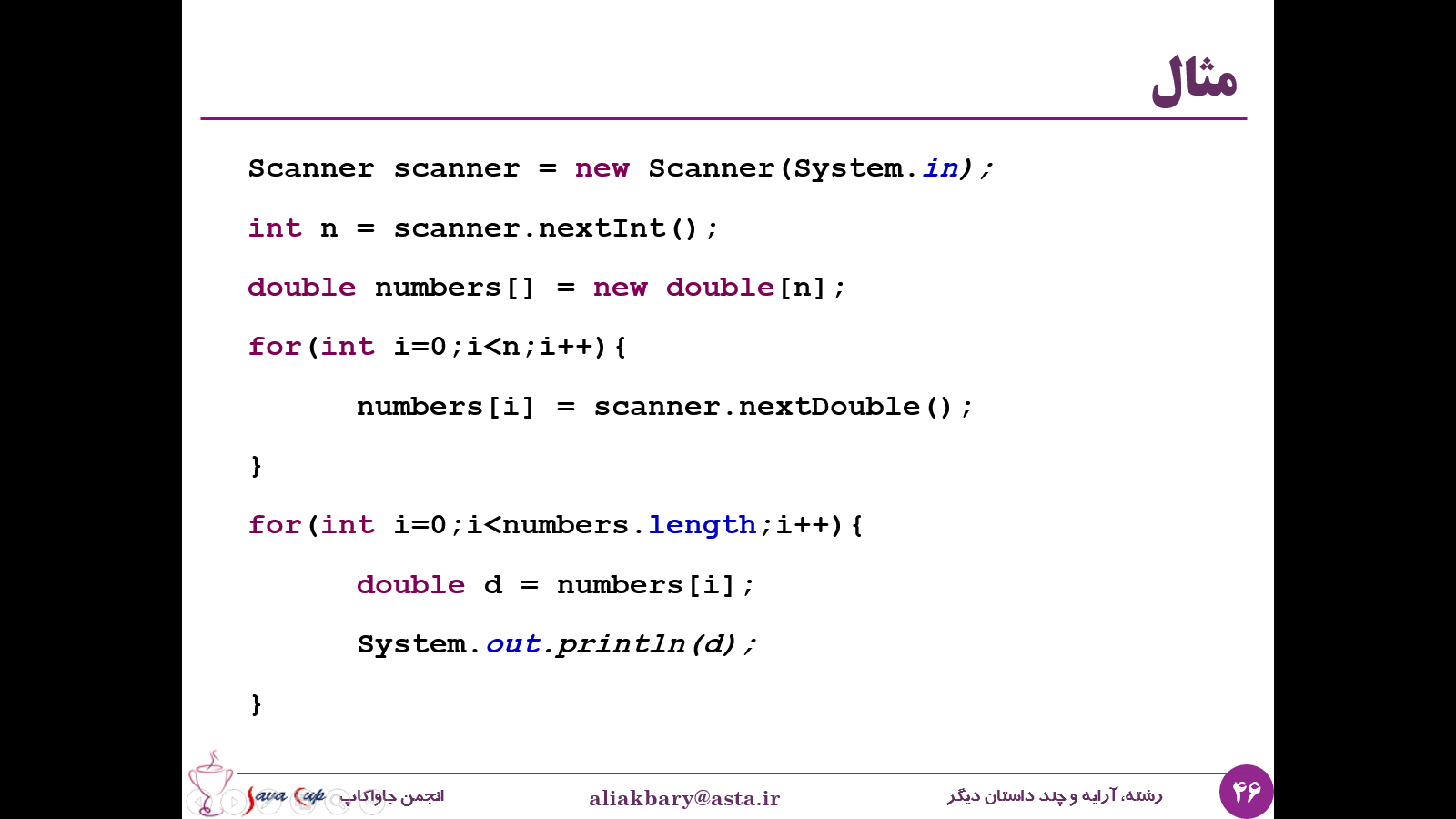
حال اگر بخواهیم یکی از عناصر آرایه مثلا عنصر پنجم را تغییر بدهیم کافیست نام آرایه و بعد کروشه و اندیس آن را برای دست رسی به آن عنصر (element) از آرایه استفاده کنیم. مثالا کد زیر مقدار 12 را در خانه با اندیس 5 از آرایه ای با نام array قرار می دهد. و همان طور که گفتیم خانه با اندیس 5 در واقع 6امین عنصر آرایه خواهد بود چرا که اندیس ها از صفر آغاز می شوند.



و نیز می توانیم با استفاده از متغیرها هم اندیس آرایه را مشخص کنیم، برای مثال می توانیم مقدار خانه ی m ام از آرایه را در متغیری ذخیره کنیم:



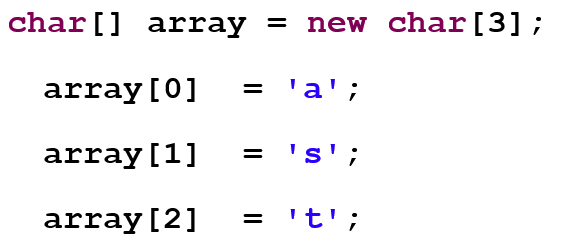
مثال: در تکه کد زیر با استفاده از scanner یک مقدار از کاربر گرفته شده و بعد آرایه ای از نوع double و با طول مقدار مشخص شده از سوی کاربر ایجاد شده، سپس در حلقه ای که از صفر تا n یعنی n مرتبه اجرا می شود تک تک مقادیر عناصر آرایه را از کاربر گرفته ایم. سپس در حلقه ی دیگری از صفر تا طول آرایه هر بار عنصر iام آرایه را در متغیر d ذخیره و آن را چاپ کرده ایم. این مثال ساده ای از نحوه ی استفاده از آرایه است.



نکته: هر متغیری از جنس آرایه، دارای فیلدی به نام length است که با کمک آن می توانیم طول آرایه را درخواست کنیم و ببینیم، در این مثال numbers.length طول آرایه ی length را برمی گرداند.

# راه کوتاه ایجاد آرایه

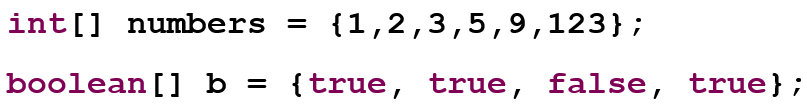
راه های کوتاه تری برای ایجاد آرایه هم وجود دارد، در واقع هنگامی که در زمان ایجاد آرایه مقادیر آن را هم می دانیم، می توانیم با راه های ساده تری هم آن را ایجاد کنیم. مثلا فرض کنید که می خواهیم آرایه ای با نوع کاراکتر و طول سه و مقادیر 'a', 's', 't' ایجاد کنیم. راه اولی که برای ایجاد آن دیدیم به صورت زیر است:



اما به جای این کار، می توانیم به صورت زیر هم آرایه را تعریف کنیم:

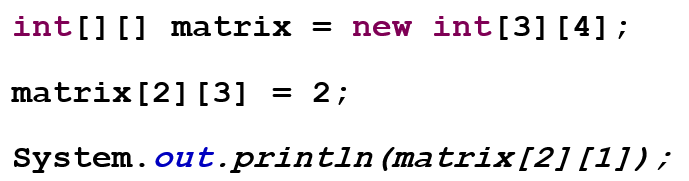


به این ترتیب آرایه ای با طول 3 ایجاد و خانه های آن مقدار دهی می شوند. مثال های دیگری از این شیوه ی ایجاد آرایه در زیر آمده است:

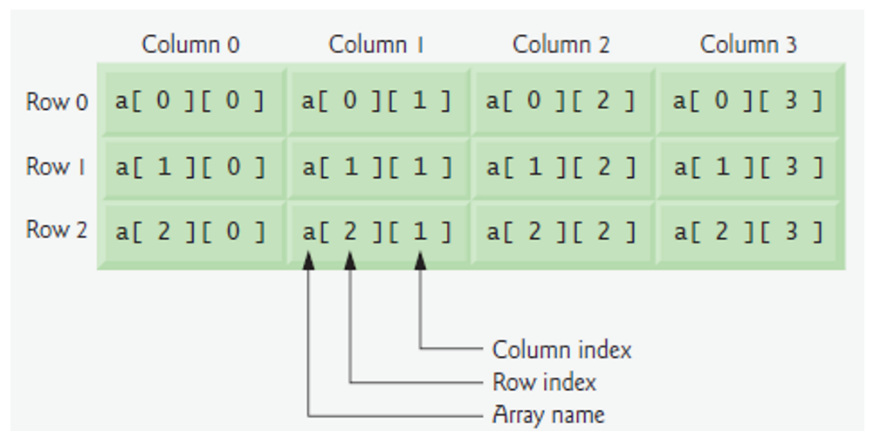


# آرایه های چند بعدی

آرایه ها می توانند چند بعدی هم باشند، آرایه هایی که تا اینجا دیدیم یک بعدی بودند، آرایه ی دو بعدی مانند مثال زیر با دو جفت کروشه نشان داده می شود، برای این آرایه می توانیم تعداد سطرها و ستون های آرایه را مشخص کنیم، مثال زیر آرایه ای با 3 سطر و 4 ستون است. و همان طور که در مثال زیر هم مشاهده می کنید برای دست رسی به عناصر این آرایه باید هم شماره سطر و هم شماره ستون را مشخص کنیم:

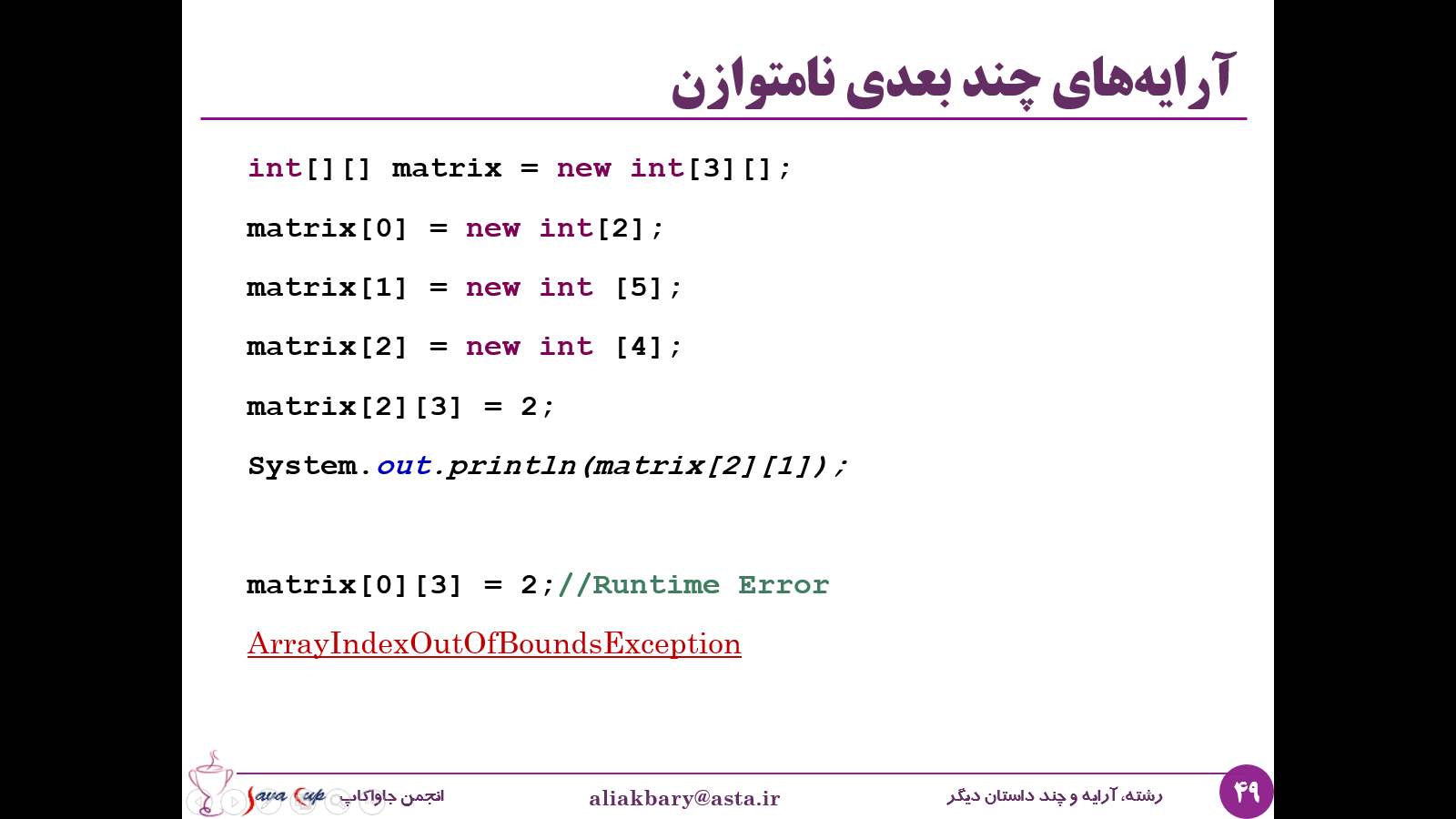


البته آرایه های چند بعدی هم هنگام ذخیره شدن در حافظه به صورت تک بعدی ذخیره می شوند و هنگام ذخیره چیزی به اسم سطر و ستون وجود ندارد. سطر و ستون برای توصیف و استفاده و مدیریت بهتر در زبان جاوا قرار داده شده است.

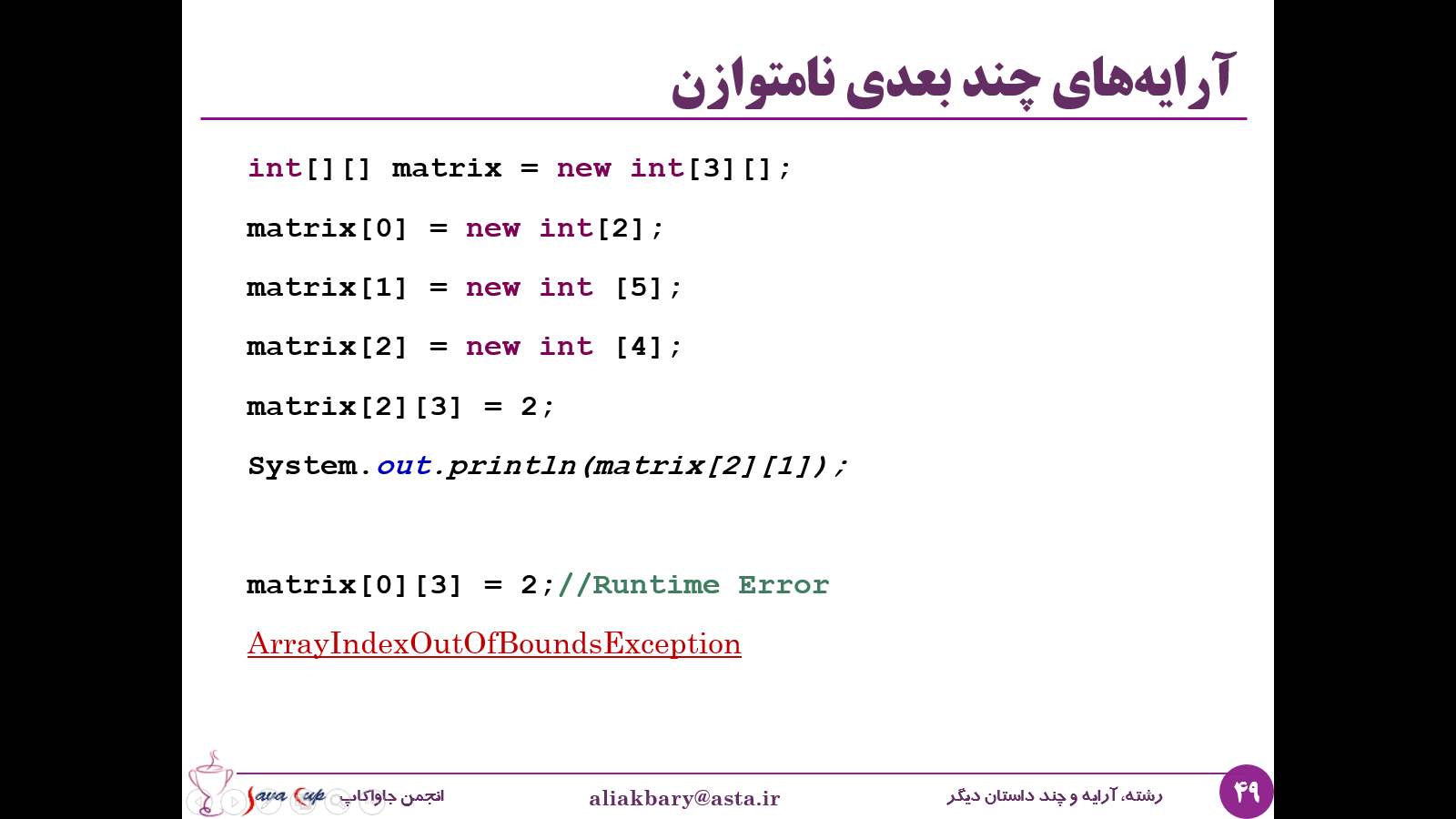


# آرایه های چند بعدی نامتوازن

آرایه های چند بعدی می توانند نامتوازن هم باشند. مثلا مانند تکه کد زیر می توان هنگام ساخت آرایه تنها تعداد سطرهای آن را مشخص کرد و تعداد ستون ها را مشخص نکرد. پس matrix در ابتدا آرایه ای است که 3 سطر دارد. سطر اول آن آرایه ای از 2 integer است، سطر یکم آرایه ای از 5 عنصر و در نهایت سطر دوم آرایه ای به طول 4 است. در واقع در این مثال ما به جای ساختن یک ماتریس تعدادی سطر با تعداد متفاوتی ستون ایجاد کرده ایم و ماتریس ما ماتریس متوازنی نیست. چنین امکانی در جاوا وجود دارد و بسته به اینکه برای هر سطر چه تعداد ستون ایجاد کرده باشیم می توانیم از عناصر آن استفاده کنیم.

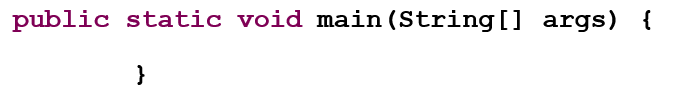


اما اگر مثلا در اینجا بخواهیم به خانه ی سوم از سطر صفرم دست رسی پیدا کنیم با خطای زمان اجرا مواجه می شویم چرا که سطر صفرم ماتریس ما دو عضو بیشتر ندارد و نمی توانیم به عنصر سوم آن دست رسی پیدا کنیم. در واقع وقتی دو عنصر بیشتر نداریم تنها اندیس 0 و 1 قابل استفاده است و در صورتی که از اعداد بالاتر از آن استفاده کنیم با خطای زمان اجرا مواجه می شویم. به خطاهای زمان اجرا اصطلاحا Exception گفته می شود و در این مثال با ArrayIndexOutOfBoundsException مواجه خواهیم شد که نشان می دهد اندیس استفاده شده فراتر از محدوده ی این آرایه است. درباره ی Exception و مفهوم آن بعدا صحبت خواهیم کرد.

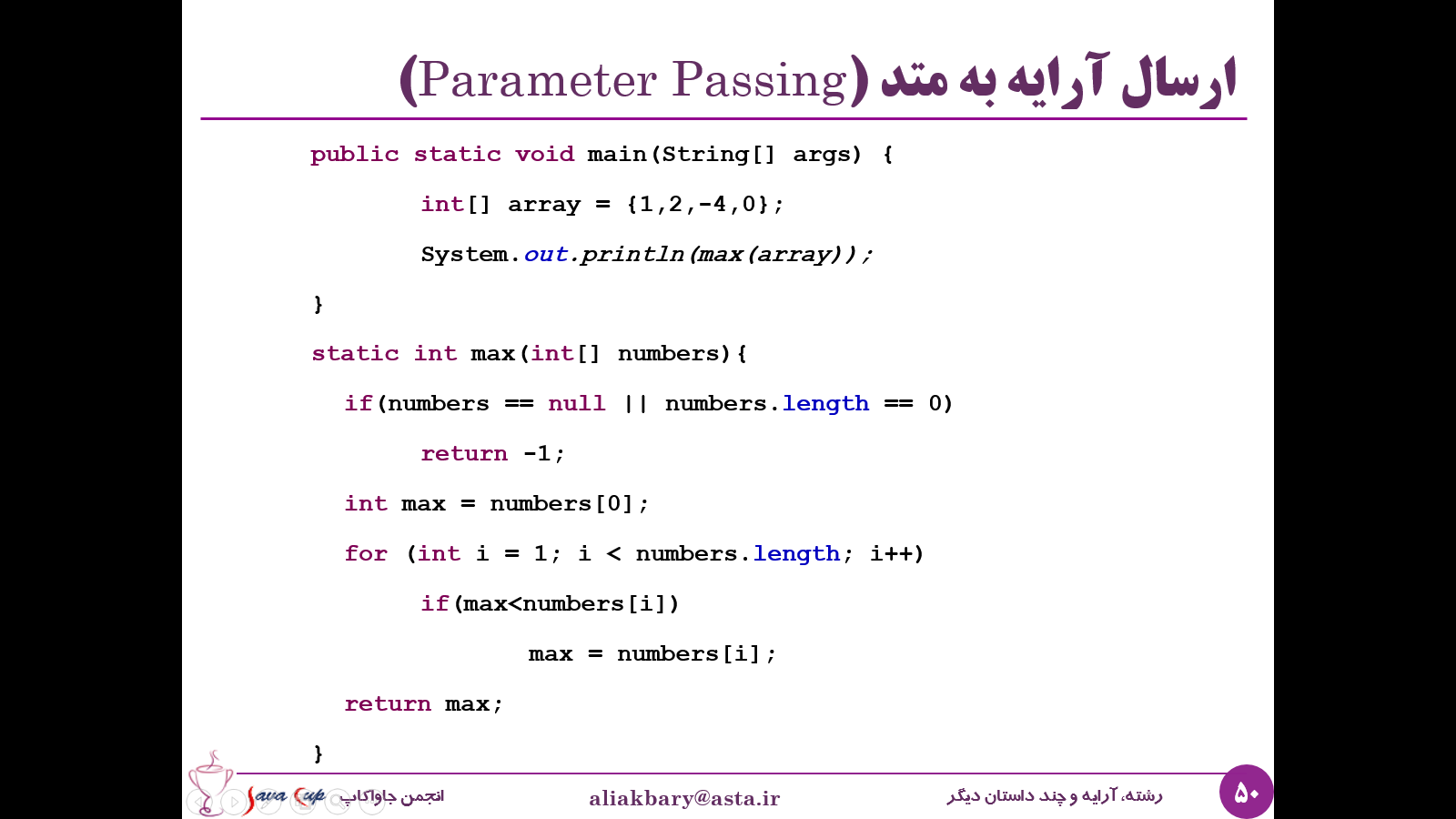


# ارسال آرایه به متد (Parameter Passing)

آرایه ها را می توانیم به متدها هم پاس بکنیم. همان طور که در گذشته هم دیده اید هنگامی که ما متد main را می نویسیم همواره آرایه ای از رشته ها هم به آن پاس می کنیم و برای هنگام اجرای برنامه می توانیم این پارامترها را مشخص کنیم.



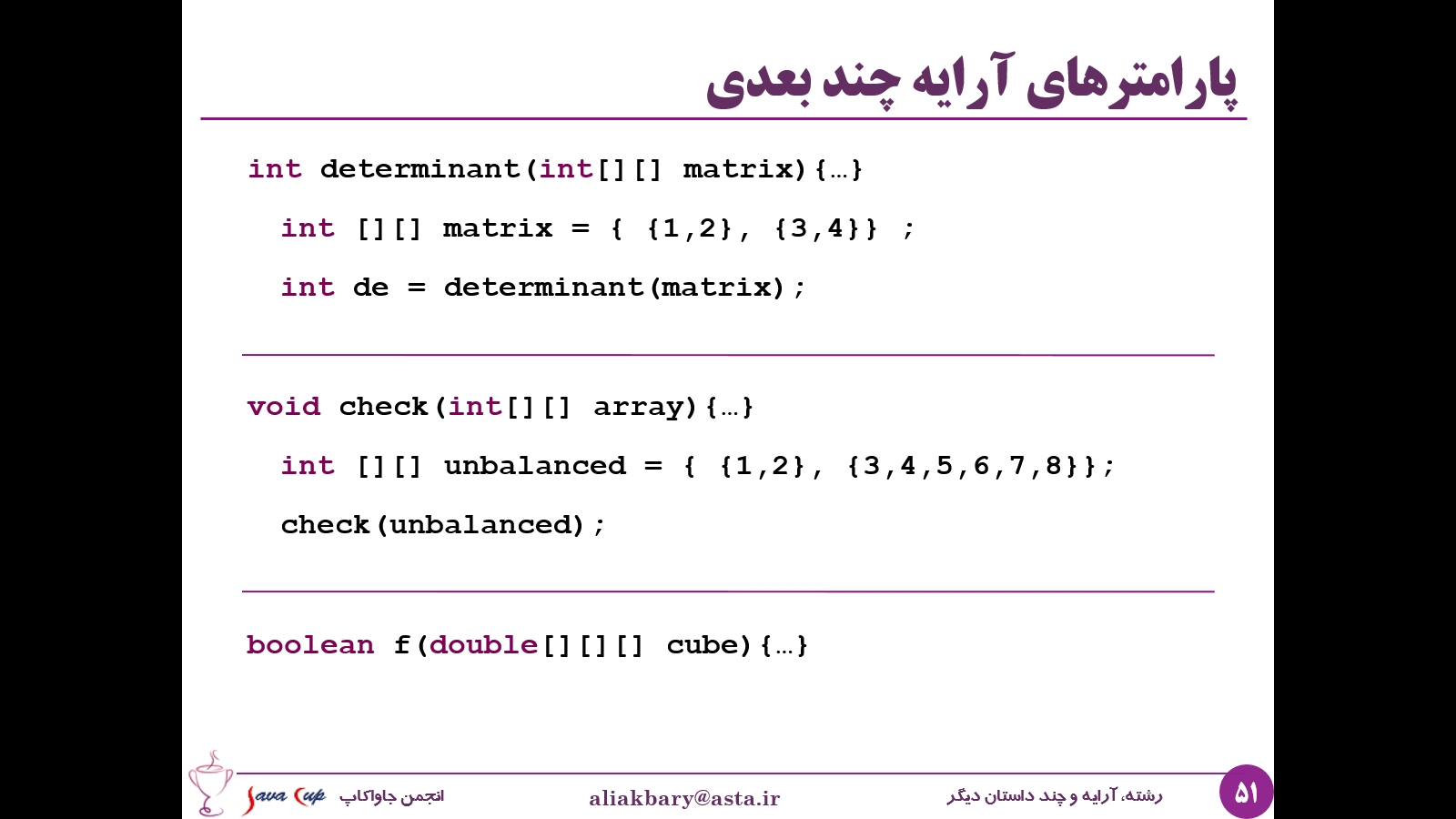
مثال: در این مثال ما آرایه ای از integer ها ساخته ایم و آن را به متدی مانند max ارسال کرده ایم که این متد را نیز می بینید، متد max آرایه ای از integer ها به عنوان پارمتر می گیرد و در صورتی که آرایه خالی باشد مقدار -1 و در غیر این صورت مقدار بیشینه آرایه را بر می گرداند. پس ما از آرایه ها مانند دیگر انواع داده می توانیم به عنوان پارامتر تابع استفاده کنیم. (در مورد مقدار null بعدا مفصل صحبت خواهیم کرد)



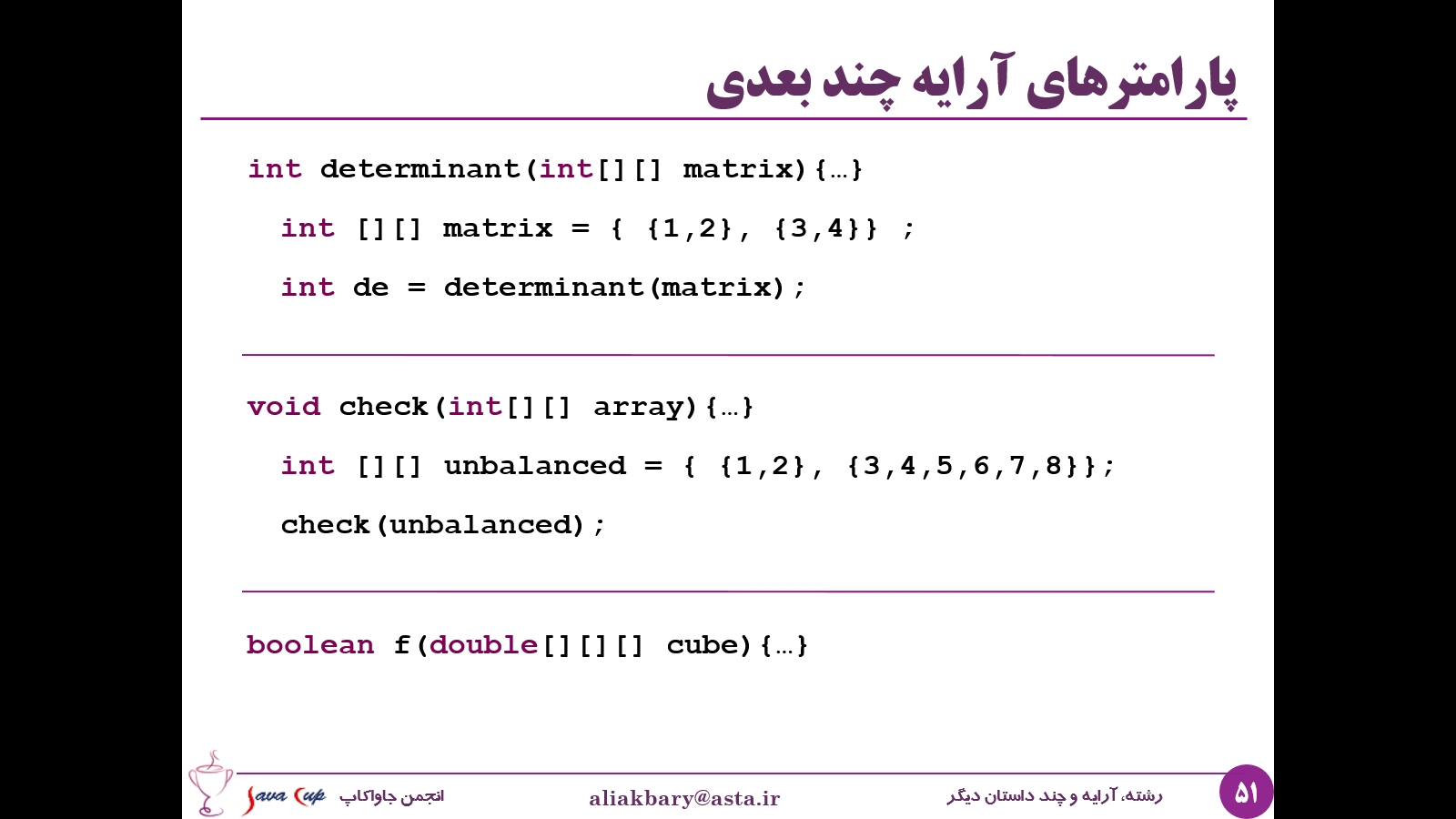
پس با نحوه ی تعریف پارامترهایی از جنس آرایه و نحوه ی ارسال آن هم آشنا شدیم.

# پارامترهای آرایه چند بعدی

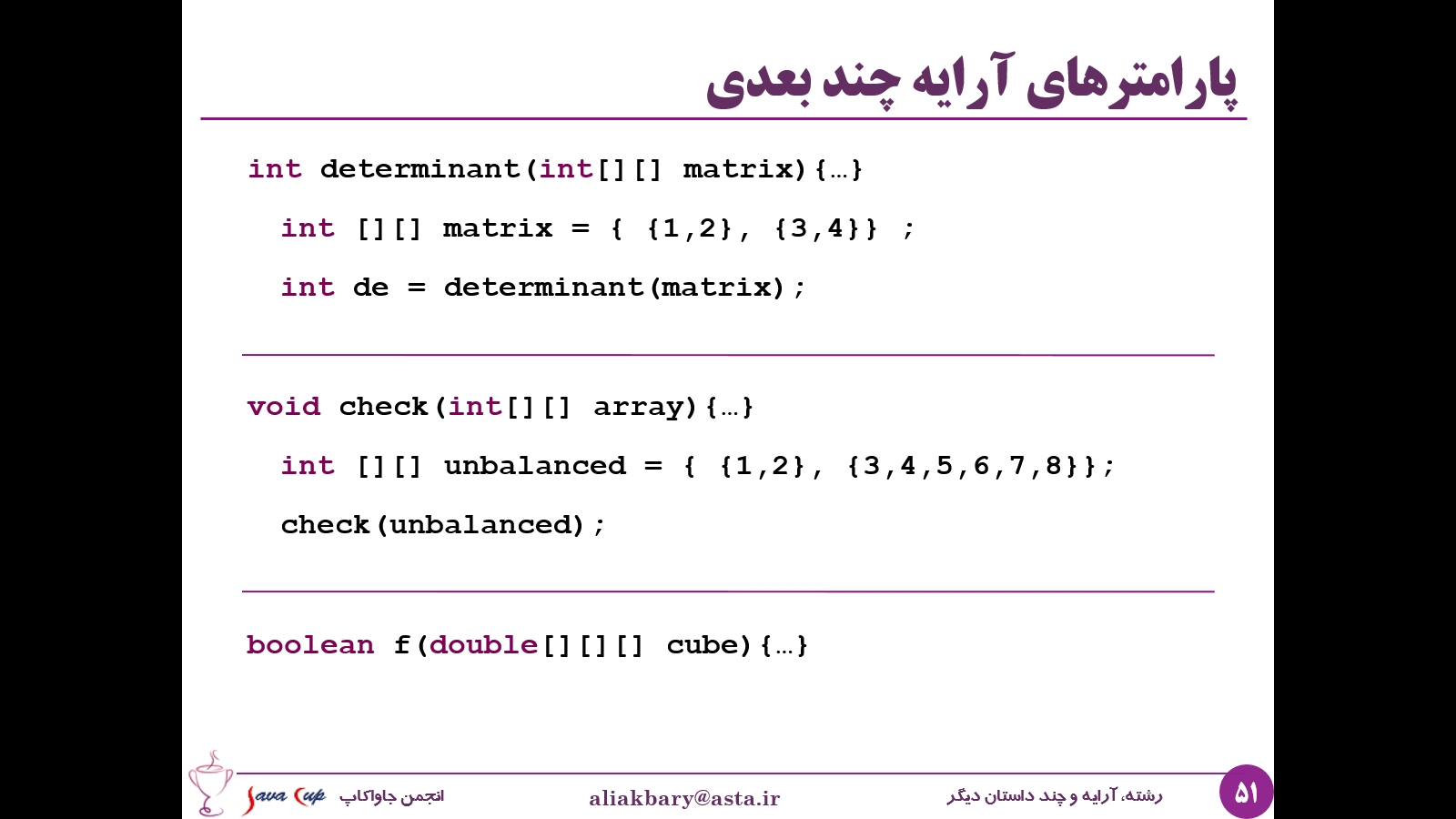
پارامترهای متدهای ما می توانند آرایه های چند بعدی هم باشند، در مثال زیر determinant متدی است که آرایه ای دو بعدی به عنوان پارامتر می گیرد و matrix یک آرایه ی دو بعدی است که به صورت خلاصه ایجاد و مقدار دهی شده است و آن را به متد گفته شده پاس کرده ایم:



یا آرایه ای از integer ها که آرایه ای دو بعدی است و می تواند همان طور که دیدیم حتی نامتوازن باشد و آن را به متد check پاس کرده ایم، فعلا به اینکه محتوای این متد چی کاری انجام می دهد کاری نداریم:



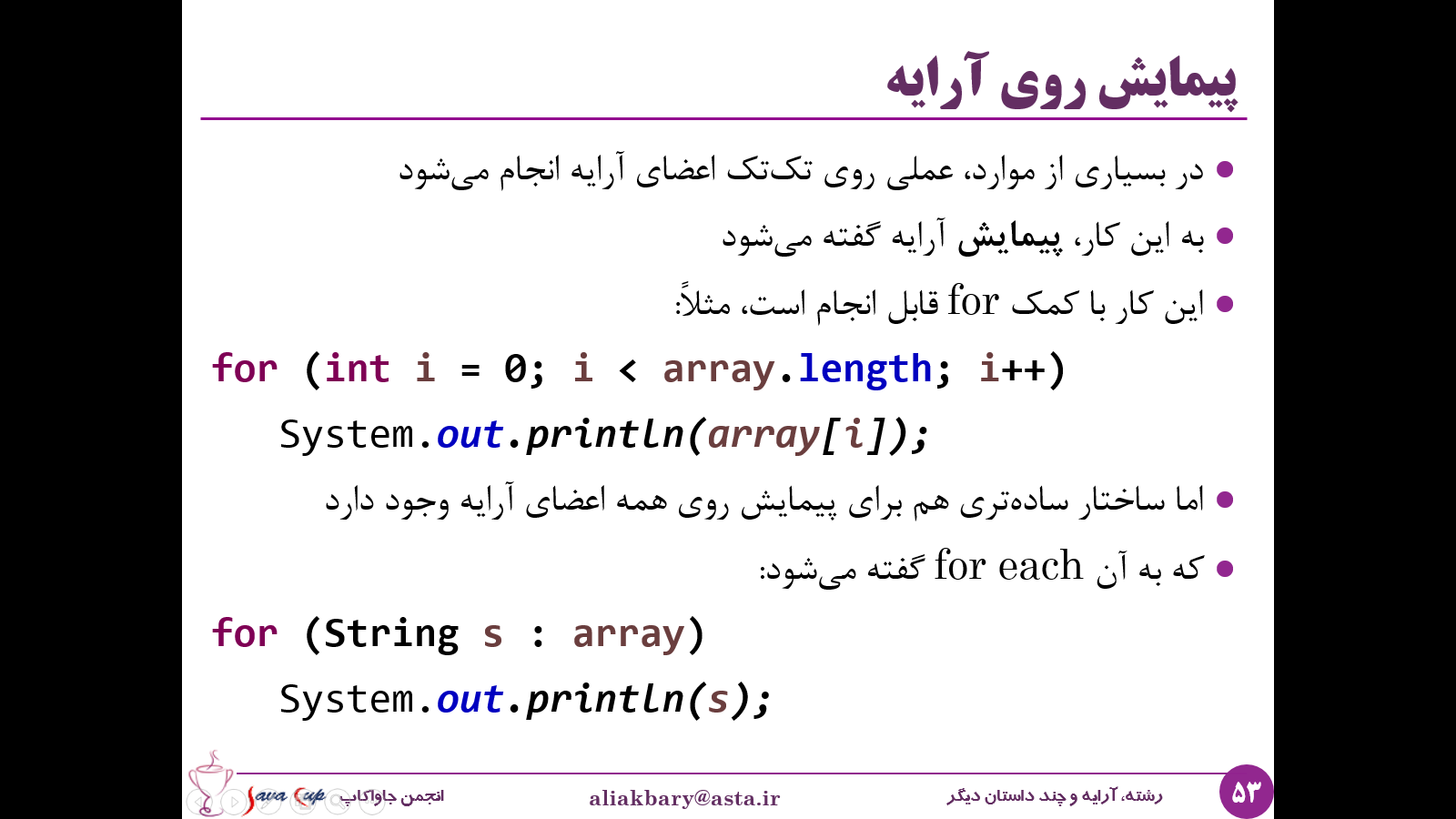
و یا آرایه هایی که چند بعدی با ابعاد بالاتر هستند، مثلا سه بعدی هستند و می توانیم آنها را ایجاد به عنوان پارامتر به متدها ارسال کنیم:



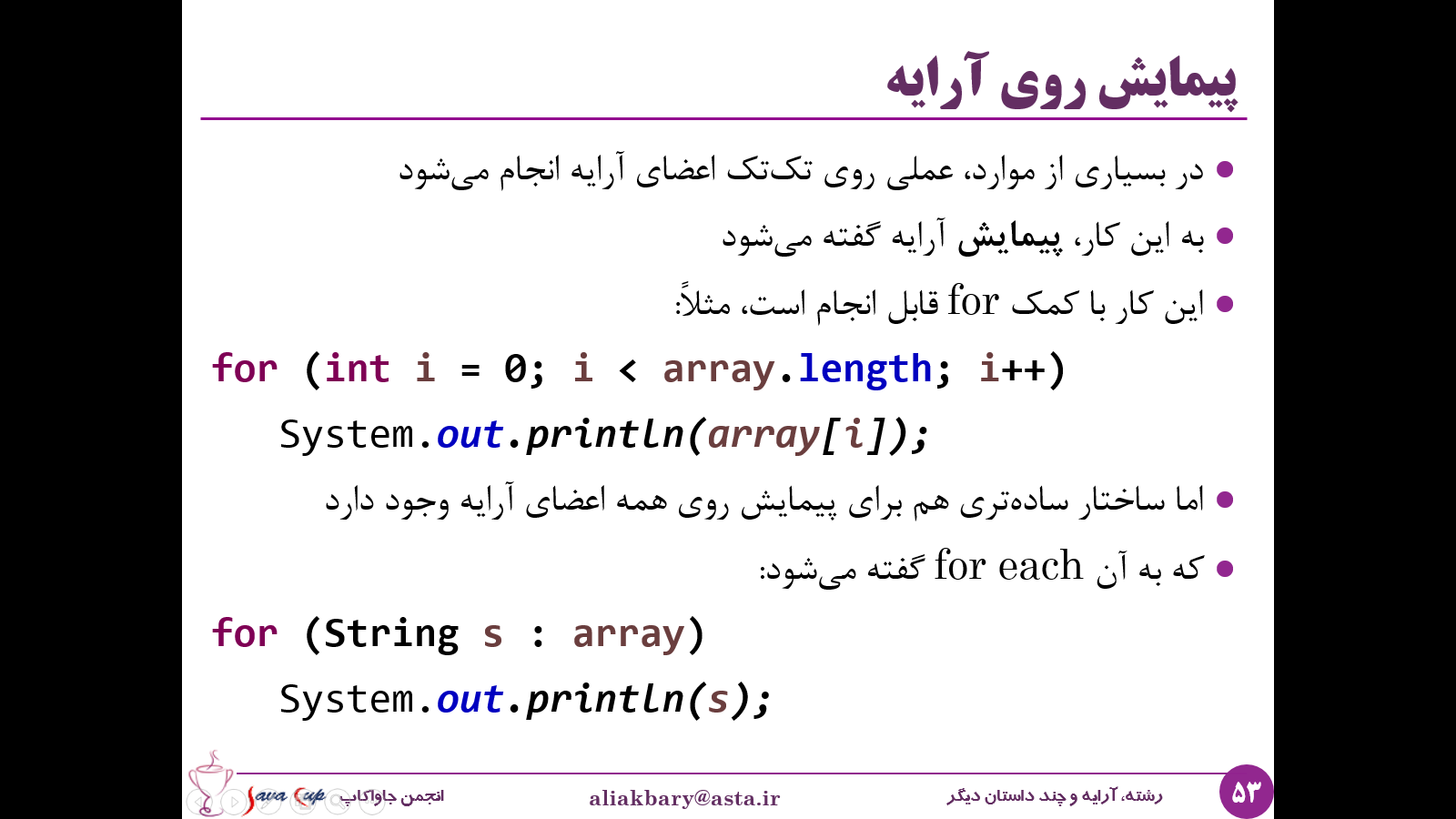
نکته: یک نکته ی مهم این است که وقتی آرایه ای به یک متد پاس می شود، عناصر این آرایه داخل متد کپی نمی شوند و یک کپی از آرایه به متد پاس نمی شود بلکه یک ارجاع یا به نوعی اشاره گری به آن آرایه به متد پاس می شود، در مورد این مفهوم و نیز نحوه ی ارسال پارامترها به متدها بعدا مفصل تر صحبت می کنیم، اما فعلا به این توجه داشته باشید که اگر آرایه ای را به متدی پاس کنیم و داخل آن متد عناصر آرایه را تغییر دهیم، آرایه ی اصلی (آرایه ی پاس شده به متد) تغییر می کند.

# پیمایش روی آرایه

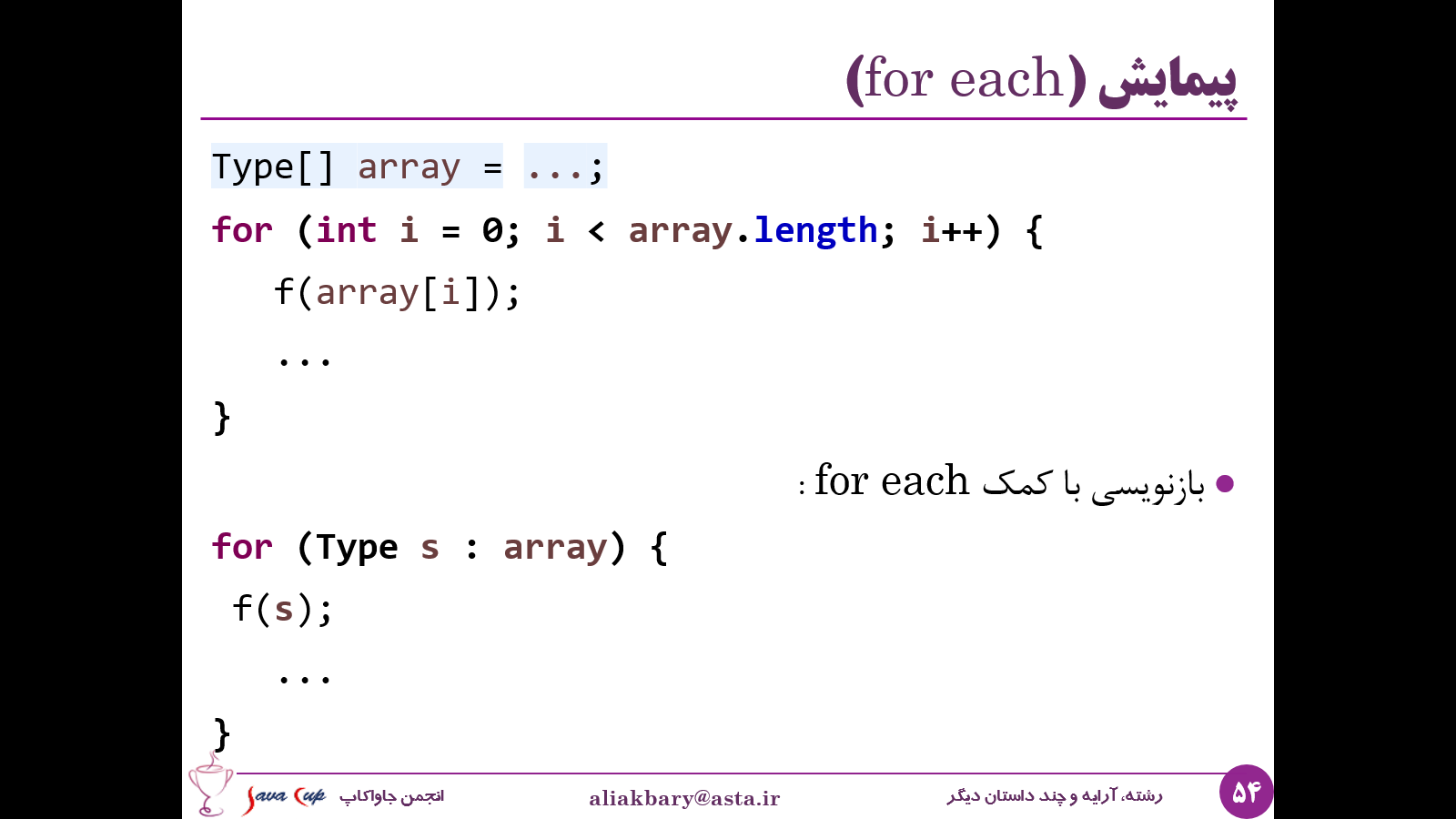
ما خیلی از اوقات نیاز داریم روی تمامی عناصر یک آرایه پیمایش کنیم (اگر بخواهیم به تک تک عناصر آرایه دست رسی پیدا کرده و عملی را روی آنها انجام دهیم به این کار اصطلاحا پیمایش آرایه گفته می شود). یه راه برای این کار که قبلا هم با آن آشنا شدیم استفاده از حلقه ی for است، مثلا شمارنده ای مانند i از صفر (اندیس اولین عنصر آرایه) تا کوچکتر از طول آرایه (اندیس آخرین عنصر آرایه) هر بار یکی یکی افزایش پیدا کند و از این طریق ما به عنصر i ام آرایه دست رسی پیدا کنیم:

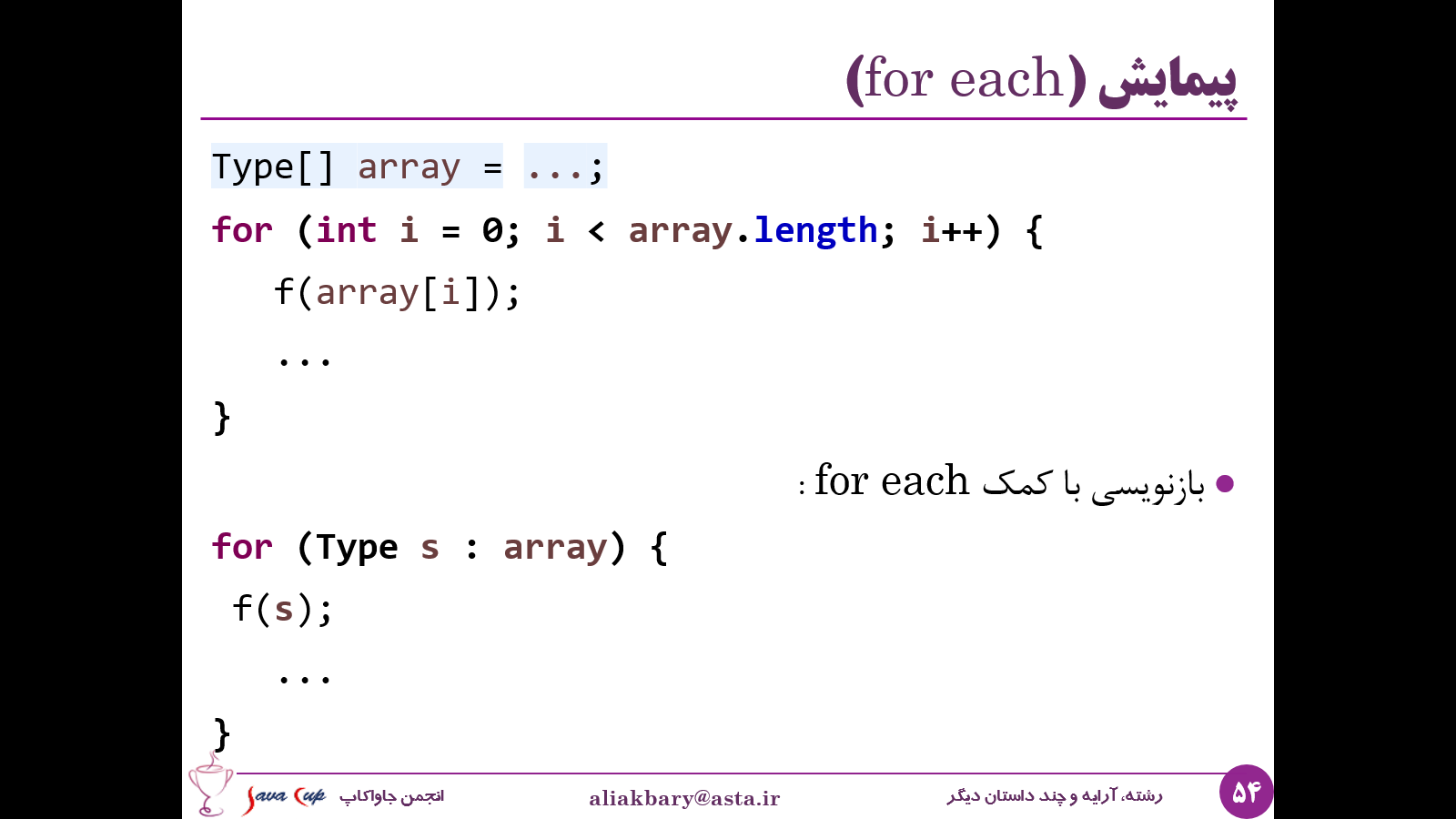


اما اگر نیاز ما این است که روی همه ی عناصر آرایه پیمایش انجام بدهیم، راه ساده تری هم وجود دارد که به آن ساختار for each گفته می شود، این ساختار نیز با حلقه ی for نمایش داده می شود اما به جای تعریف متغیر شمارنده و درگیری های مرتبط با آن، می توانیم به شکل زیر این کار را انجام بدهیم، عبارت زیر یعنی به ازای هر عنصری از جنس String که در آرایه ی array هست، نام آن عنصر را s بگذار و بدنه ی حلقه را برای آن اجرا کن:



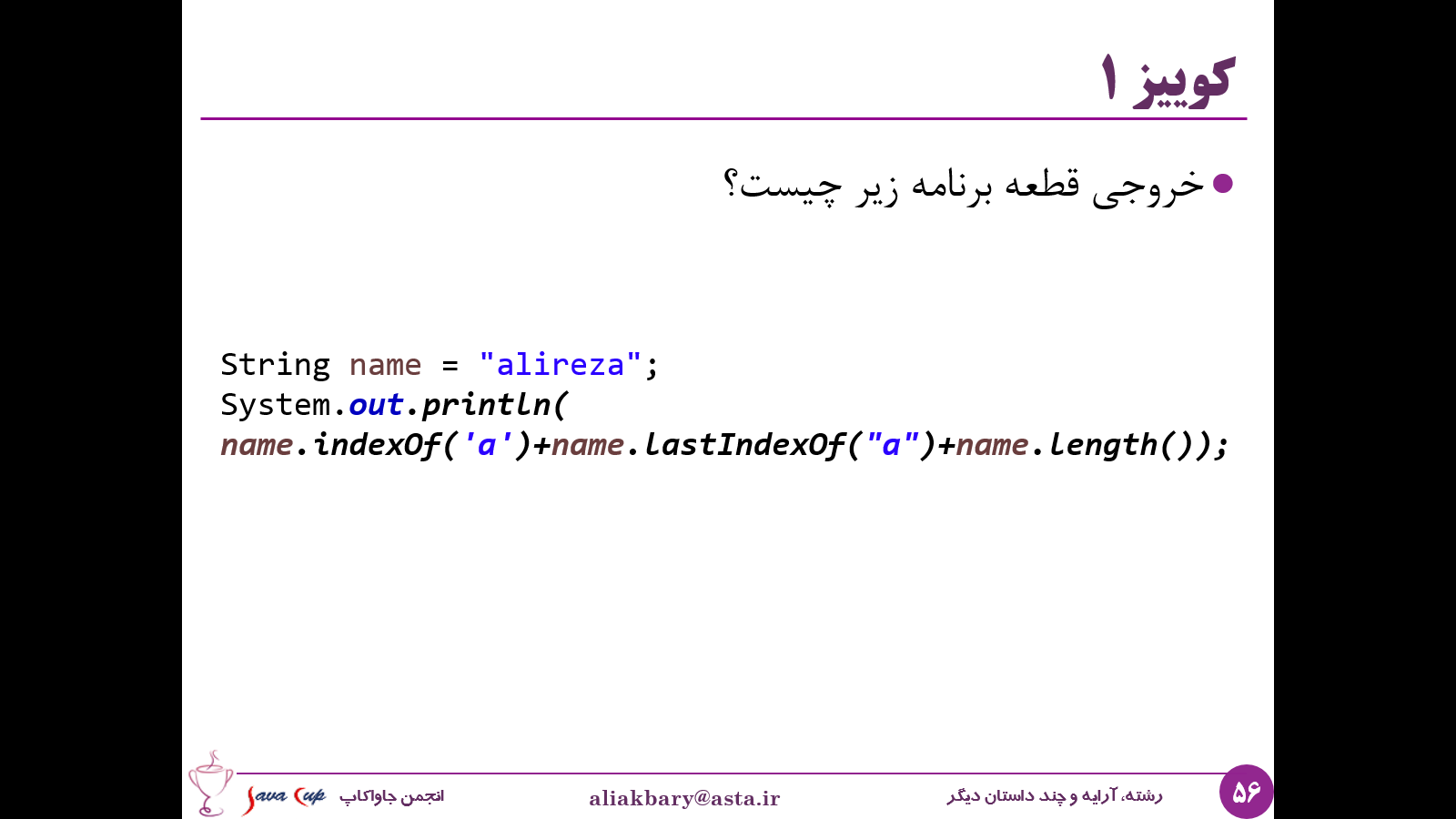
این امکان از جاوای 5 (جاوای 1.5) به جاوا اضافه شده است. بنابر این اگر ما آرایه ای از هر نوع دلخواه داشته باشیم، مثلا آرایه ای از نوع Type که این Type هر چیزی می تواند باشد، integer، character و یا هر نوع دیگری، به جای نوشتن کد شماره یک، می توانیم به سادگی با استفاده از حلقه ی for each مانند کد شماره 2 برنامه عمل کنیم:





# کویز

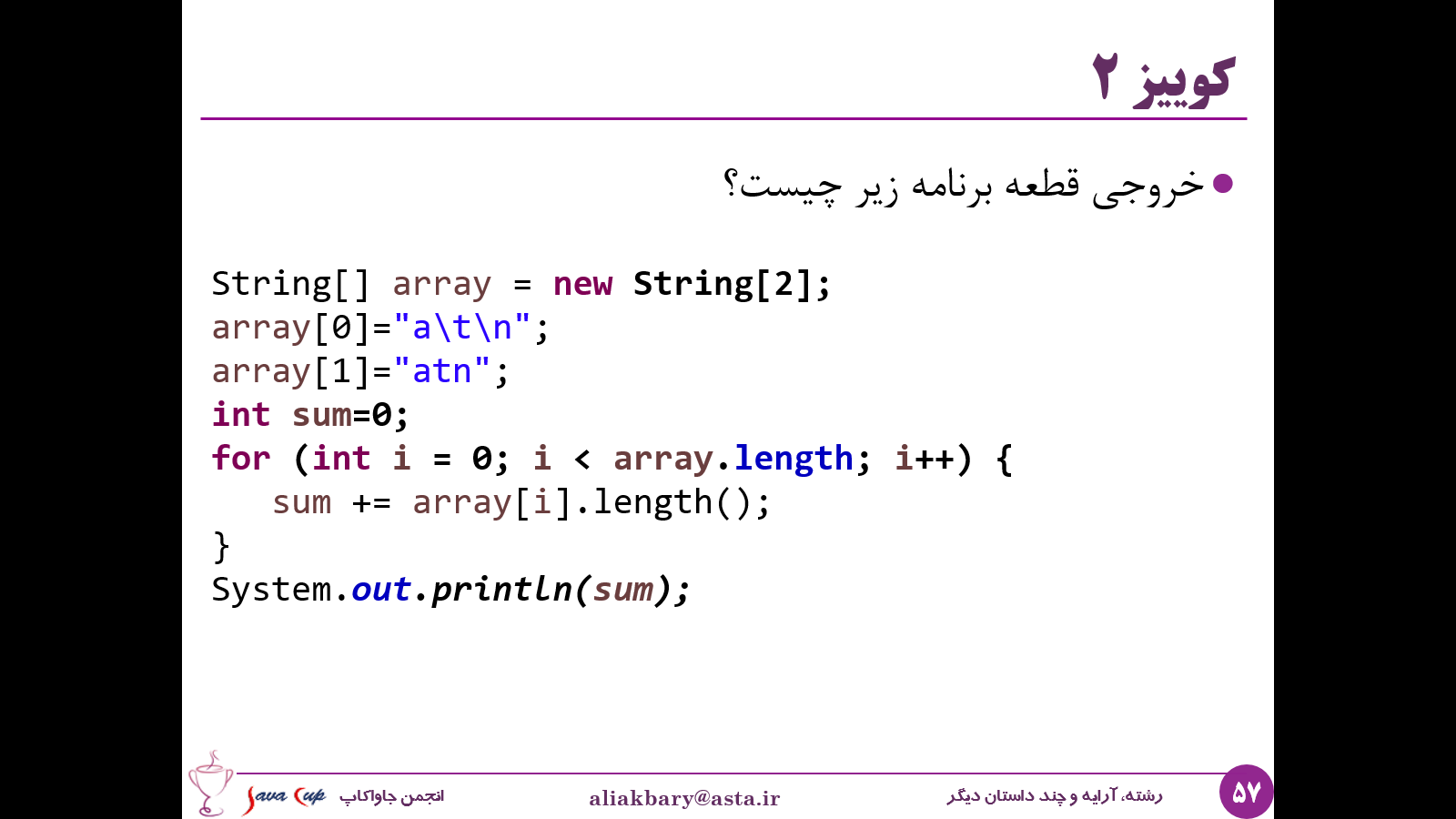
* خروجی قطعه برنامه زیر چیست؟



* پاسخ:

عدد 13. چرا که اندیس اولین a در رشته داده شده 0 و اندیس آخرین a برابر با 6 است و طول رشته هم 7 هست پس نتیجه ی چاپ شده برابر جمع این اعداد یعنی 13 خواهد بود.

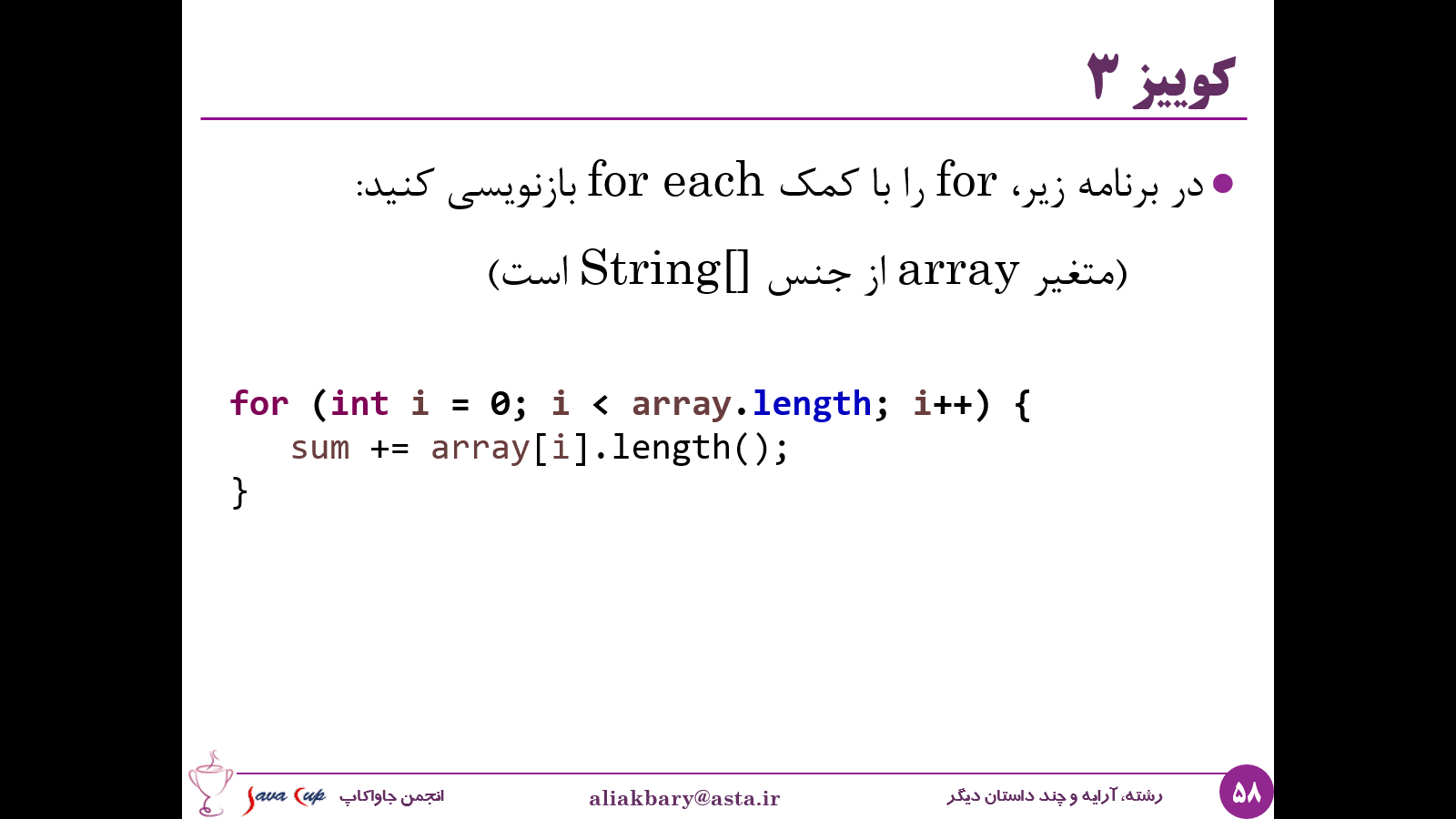
* خروجی قطعه برنامه زیر چیست؟



* پاسخ:

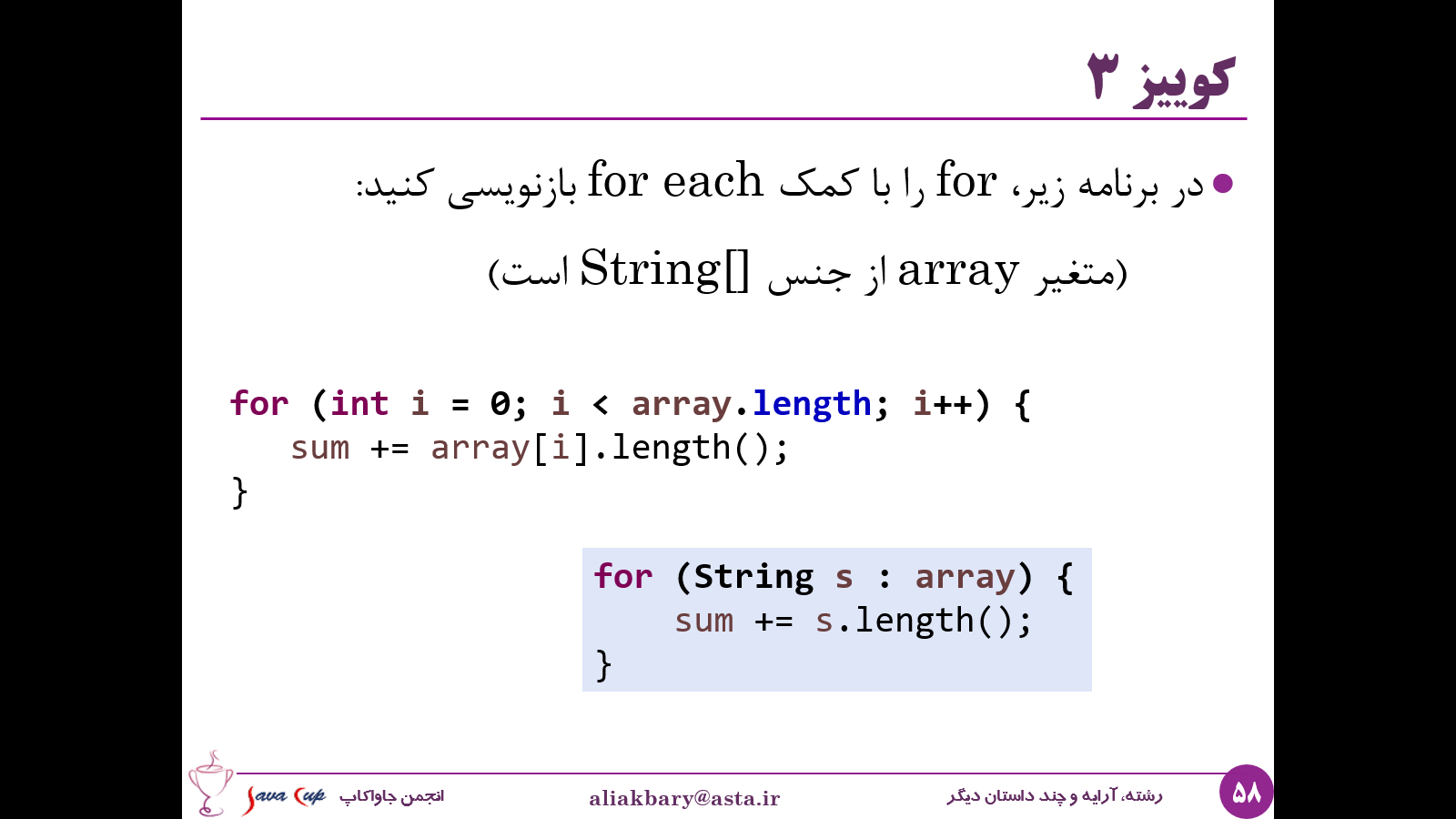
خروجی این برنامه 6 است، در اینجا یک آرایه ی دو عنصری از رشته ها داریم که عنصر اول 3 کاراکتر دارد، کارکتر a، کاراکتر tab و کاراکتر new line، و عنصر دوم نیز 3 کاراکتر دارد، در ادامه یک حلقه ایجاد کرده ایم که روی این آرایه پیمایش و هر بار طول عنصر iام آرایه را به متغیر sum اضافه می کند پس خروجی برنامه جمع این طول ها و برابر با 6 خواهد بود.

* قطعه کد زیر یک حلقه ی for ساده است که عملیاتی را روی یک آرایه انجام می دهد، این حلقه را با کمک for each پیاده سازی کنید، فرض کنید که متغیر array آرایه ای از جنس رشته ها است.



* پاسخ

همان طور که گفته شد for each یک حلقه ی ساده است که در آن با مشخص کردن نوع عناصر و نام متغیر پیمایش کننده ی آرایه می توان عملیاتی را برای انجام شدن روی همه ی عناصر آرایه مشخص کرد.

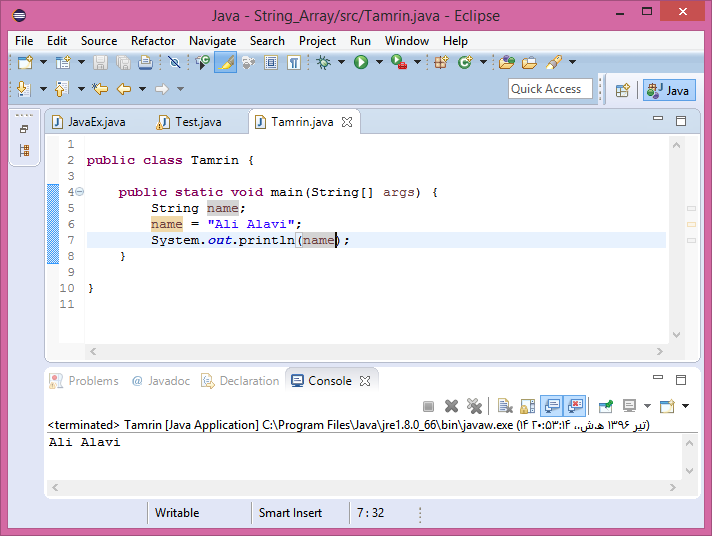


# تمرین عملی

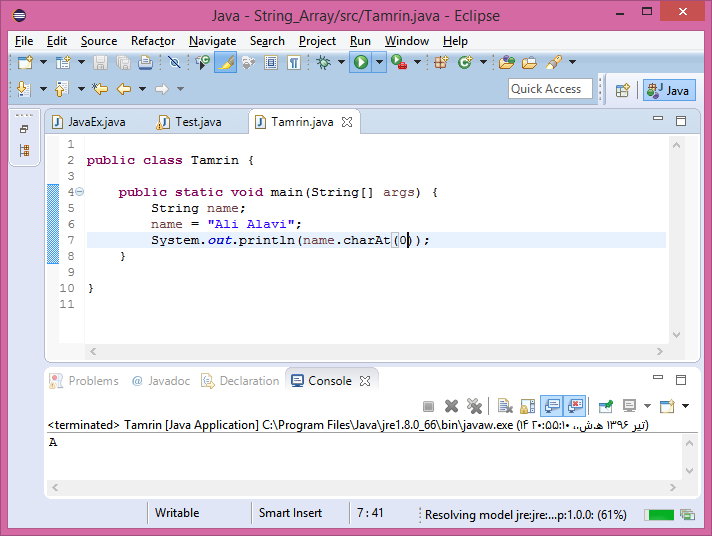
کلاسی به نام Tamrin ایجاد و متد main را برای آن تعریف کنید. سپس متغیری از جنس String مانند name در آن تعریف نمایید و آن را مقدار دهی کنید، همان طور که می بینید مقادیر ثابت یا اصطلاحا literal برای رشته ها درون "" (دابل کوتیشن) قرار می گیرد.



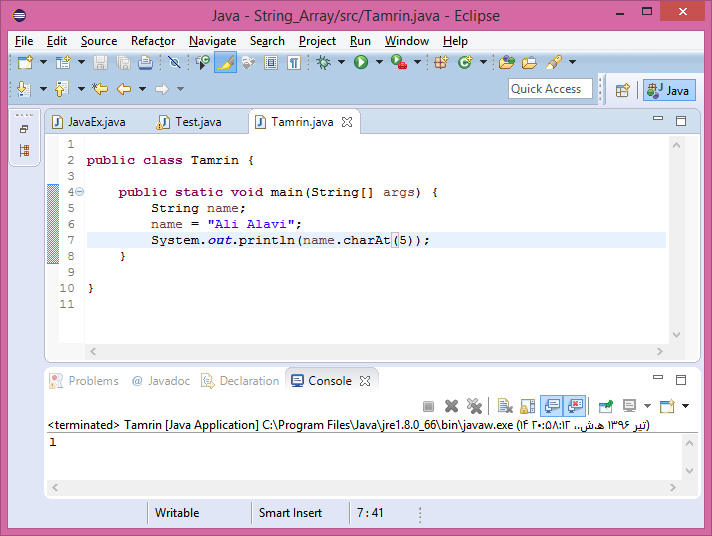
سپس می توانیم این متغیر را در خروجی چاپ نماییم.



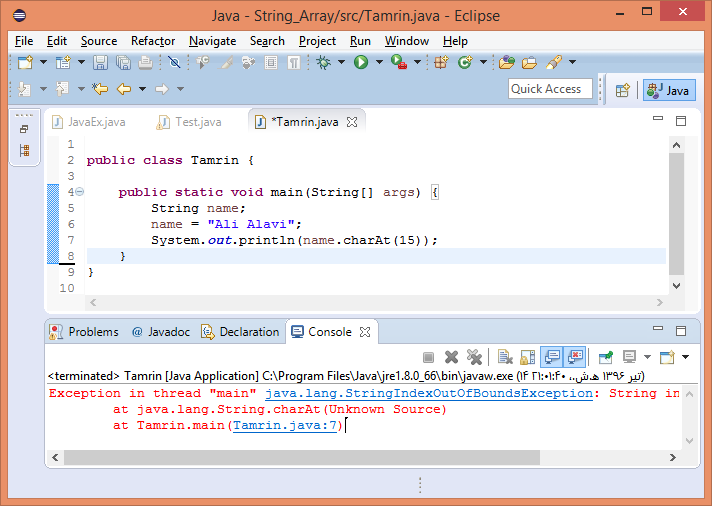
دیدیم که رشته ها متدهای مختلفی دارند و با کمک آنها می توانیم بر روی رشته ها کار انجام بدهیم. مثلا name.charAt(0) را چاپ نماییم. و خروجی این برنامه صفرمین کاراکتر این رشته یعنی A خواهد بود.



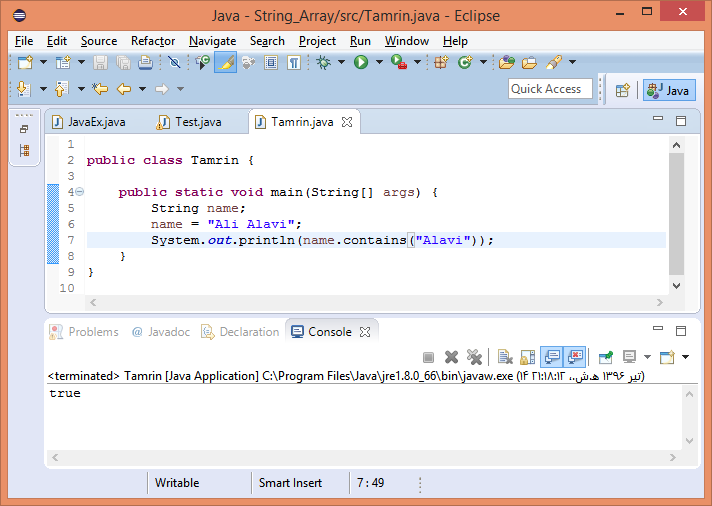
همین طور می توانیم اندیس های بزرگتری را هم به این متد بدهیم برای مثال name.charAt(5) که کاراکتر پنجم رشته یعنی حرف ششم آن (حرف l) چاپ خواهد شد.



اما اگر یک عدد بزرگ مانند 15 به آن بدهیم چه اتفاقی خواهد افتاد؟ در این صورت برنامه ما با خطای زمان اجرا مواجه خواهد شد. اصطلاحا یک exception اتفاق می افتد چون این رشته خانه ی شماره 15 ندارد (باید رشته ی ما 16 حرفی باشد تا خانه ی شماره 15 داشته باشد)



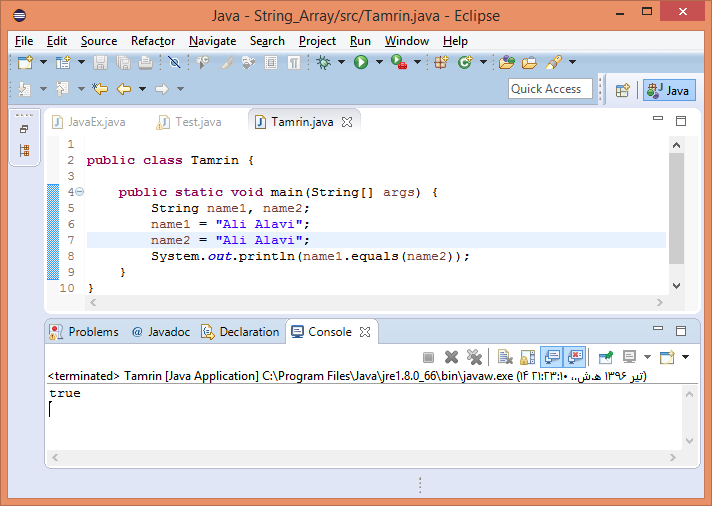
متدهای دیگری هم برای کار با رشته ها وجود دارد مثلا برای اینکه ببینیم آیا این رشته حاوی رشته ی مورد نظر ما هست یا نه می توانیم از متد contains استفاده کنیم، مثلا می خواهیم ببینیم که آیا رشته ی ما شامل رشته ی "Alavi" هست یا خیر. خروجی این متد یک true یا false خواهد بود.



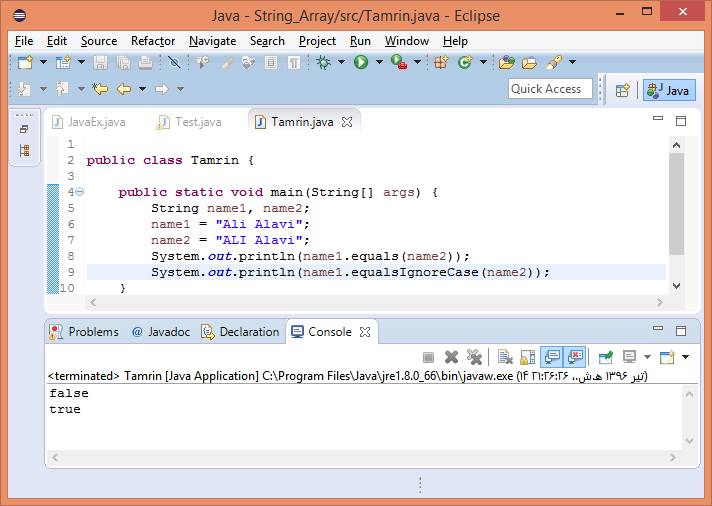
یا می توانیم دو رشته را با همدیگر مقایسه کنیم، فرض کنید متغیر name1 و name2 هر دو از نوع رشته داشته باشیم که یکی مقدار "Ali Alavi" و دیگری مقدار "Taghi Taghavi" داشته باشد، حال می توانیم با استفاده از متد equals بررسی کنیم آیا این دو رشته برابر هستند یا نه. و همانطور که می بینید نتیجه false است.



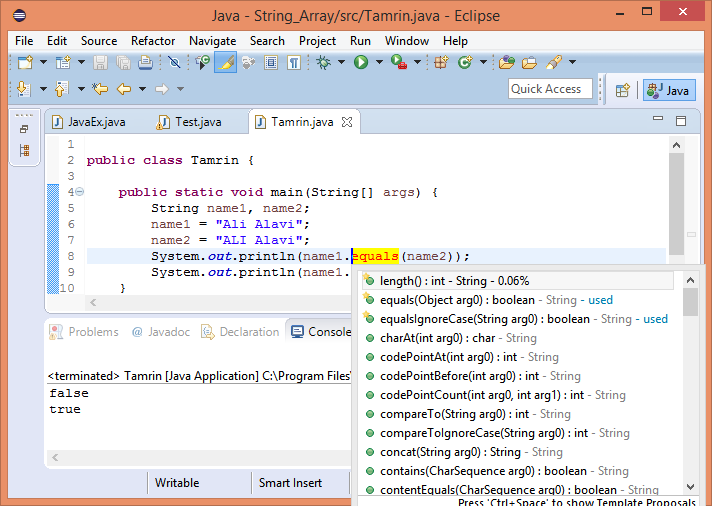
اما اگر دو مقدار با هم مساوی بودند، در آن صورت مقدار بازگشتی این متد true خواهد بود.



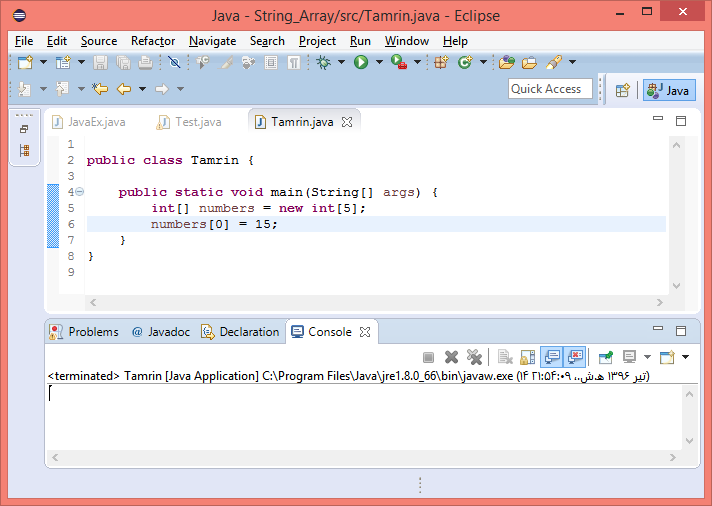
حالا بخشی از یکی از رشته ها را بزرگ بنویسید، در این صورت متد equals مقدار false برمی گرداند اما اگر از متد equalsIgnoreCase استفاده کنیم، مقداری که برمی گردد true خواهد بود چرا که اگر بزرگ و کوچک بودن حروف را در نظر نگیریم این دو رشته با هم برابر هستند.



و متدهای مختلف دیگری نیز برای کار با رشته ها وجود دارند که اگر از یک محیط توسعه مانند اکلیپس استفاده می کنید، بلافاصله بعد از نوشتن نقطه بعد از متغیر از جنس رشته لیست متدهای قابل استفاده برای کار با رشته ها را می توانید ببینید.



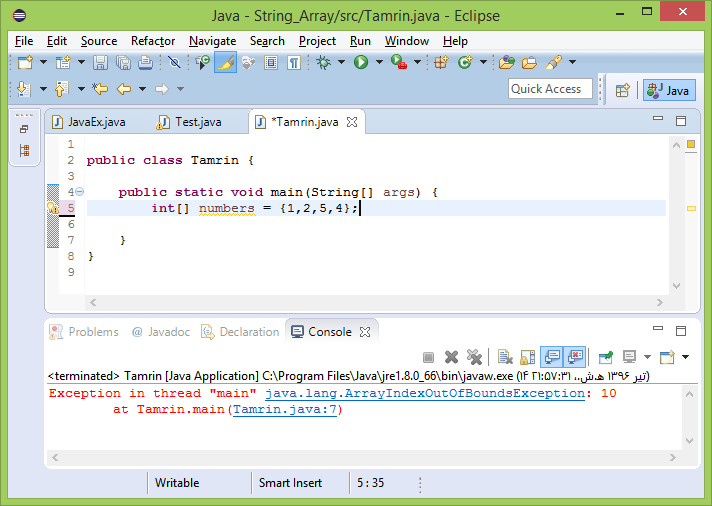
کمی هم می خواهیم در مورد آرایه ها تمرین کنیم، همان طور که دیدیم آرایه ها مجموعه ای از مقادیر هم نوع هستند، مثلا می خواهیم آرایه ای از integer ها ایجاد کنیم، لازم است این آرایه را new کنیم و بسازیم. مثلا آرایه ای به طول 5 بسازیم، بعد از آن می توانیم عناصر مختلف آرایه را مقدار دهی کنیم.



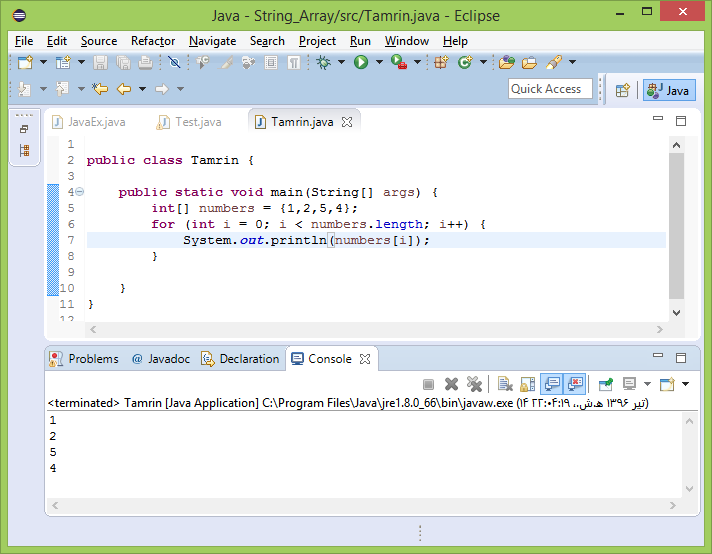
و این کد بدون هیچ مشکلی اجرا می شود، اما اگر به عنوان اندیس از عددی استفاده کنیم که در آرایه ما جا نشود، مثلا در اینجا اندیس 10 را مقدار دهی کنیم که وجود ندارد با خطای زمان اجرا مواجه می شویم که نشان می دهد چنین اندیسی برای آرایه با 5 خانه معنا ندارد.



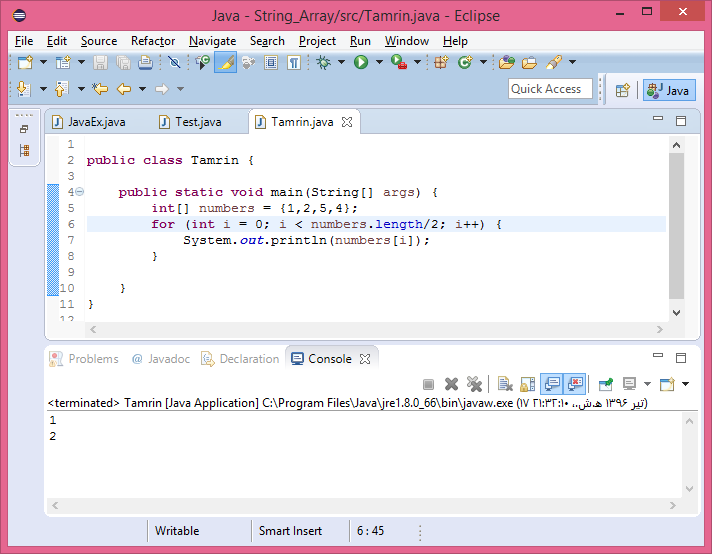
دیدیم که می توانیم آرایه را به صورت کوتاه هم مقداردهی کنیم.



ما می توانیم با کمک حلقه ها روی اعضای این آرایه پیمایش کنیم و مثلا مقدار عناصر را چاپ کنیم.

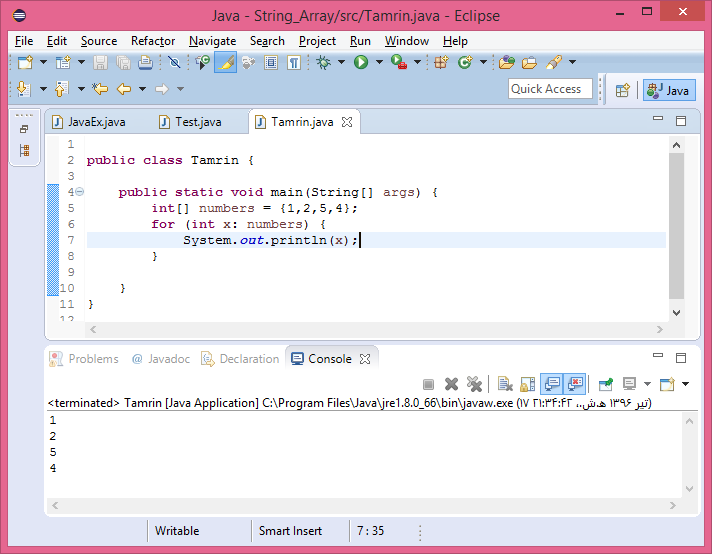


ی for ساختاری است که ما می توانیم انتخاب کنیم کدام بخش از آرایه را به چه صورت در آن پیمایش کنیم، مثلا می توانیم تنها نیمی از آرایه را پیمایش کنیم، و یا می توانیم عناصر را یکی در میان ملاقات کنیم. در واقع دست ما برای مشخص کردن نحوه ی پیمایش باز است.





و دیدیم که برای پیمایش کردن این آرایه می توانیم از حلقه ی foreach هم استفاده کنیم. حلقه اما با کمکfor each تنها می توانیم تک تک اعضای آرایه را پیمایش کنیم، البته با ساختاری مختصرتر، گویا تر و ساده تر اما انعطاف حلقه ی معمولی را ندارد و پیمایش را روی تک تک عناصر آرایه انجام می دهد.



# تمرین (آرایه)

دقت داشته باشید که پس از مطالعه و مرور مفاهیم بایست حتما تمریناتی را خودتان کد بزنید و بنویسید تا این مفاهیم برای شما جا بیافتد، در واقع تا دست به کد نشوید و برنامه ننویسید، این مفاهیم را کامل و دقیق فرا نخواهید گرفت. این ها تمریناتی هستند که از شما می خواهم تک تک آنها را خودتان کاملا پیاده سازی کنید. در هر تمرین از شما خواسته شده است که متدی را پیاده سازی کنید، پس از پیاده سازی برنامه لازم است تکه کدی برای استفاده از آن و نیز آزمایش درستی عملکرد آن بنویسید و متد خود را تست کنید.

* متدی بنویسید که عناصر یک آرایه از نوع عدد صحیح را مرتب کند.
* متدی بنویسید که دو آرایه از اعداد را با یکدیگر مقایسه کند.
  + در صورت یکسان بودن عناصر آرایه‌ها مقدار true و در غیر این صورت مقدار false را برگرداند.
* متدی بنویسید که دترمینان یک ماتریس را برگرداند.
  + ماتریس آرایه‌ای دوبعدی است که به عنوان پارامتر به متد فرستاده می‌شود.

# تمرین (رشته)

* متدی بنویسید که نام افراد را (به صورت آرایه‌ای از رشته‌ها) به عنوان پارامتر بگیرد و تعداد «سیّد»ها را برگرداند
  + (تعداد کسانی که اسمشان با seyyed شروع می‌شود)
* متدی بنويسيد که مشخص کند پارامترش يک رشته «از دو سر مساوی» است يا خير
  + يعنی رشته‌ای که اگر از انتها بخوانيمش، با خودش مساوی شود
  + مثلاً radar یا Hannah

# جمع بندی

در این فصل ما در مورد موضوعات مختلفی صحبت کردیم، ابتدا با نحوه ی خواندن اطلاعات از کنسول (گرفتن داده از کاربر) و کلاس scanner کمی آشنا شدیم، ساختارها و دستوراتی مانند switch, break و continue را مرور کردیم که در واقع ساختارهایی برای کنترل جریان اجرای برنامه هستند، در مورد مفهوم توضیحات یا comment در میان کد صحبت کردیم و دیدیم که می توانیم با کمک آنها به توضیح کد و برنامه خود بپردازیم و هنگام کامپایل این بخش ها نادیده گرفته می شوند، در مورد تکنیک دندانه گذاری یا indentation صحبت کردیم و دیدیم که این تکنیک اگرچه از دیدگاه تعریف زبان جاوا ضروری نیست اما به خوانایی برنامه کمک فراوانی می کند، همچنین دیدیم که محیط های توسعه مانند اکلیپس و یا اینتلیجی امکاناتی را برای تسهیل و تسریع در دندانه گذاری ارائه می کنند و خوب است از آنها استفاده کنیم. در مورد رشته ها صحبت کردیم و دیدیم که نوع مهمی از داده هاست که مجموعه ای از کاراکترها را نگه داری می کند و متدهای کار با رشته را نیز قدری مرور کردیم. کاراکترهای خاص که با بک اسلش شروع می شوند را دیدیم مانند کاراکتر خط جدید و یا تب و در نهایت در مورد آرایه ها صحبت کردیم، دیدیم چگونه می توان مجموعه ای از عناصر را تحت عنوان یک آرایه تعریف و مقدار دهی کرد، به توابع ارسال کرد و نیز پیمایش نمود.

# مطالعه تکمیلی

خوب است کتاب هایی که در این زمینه وجود دارند را نیز به عنوان مطالعه تکمیلی نگاهی بیاندازید، برای مثال بخش هایی از فصل 5 و 7 از کتاب Java How to Program (Deitel & Deitel) مربوط به مباحث این فصل ما هستند و تمرینات فصل 5 از کتاب دایتل را هم می توانید انجام بدهید. اگر کتاب دیگری نیز در دست دارید می توانید فصل های مربوط به این مباحث را در آن پیدا و مطالعه کنید. این کار به تعمیق دانش شما کمک می کند.

* جستجو کنید و بخوانید

پیشنهاد می کنم از اینترنت استفاده کنید و درباره بخش های مختلف مطالبی که در این فصل آموختیم جست و جو کنید، مثلا صفحات ویکی پیدا را جست و جو کنید و بخوانید، و اگر بتوانید انگلیسی مطالعه کنید هم خیلی خوب خواهد بود. کلید واژه های پیشنهادی ما برای جست و جو در ادامه آمده است:

* Unicode
* Character set
* Encoding
* Parameter Passing in Java
  + Call by value
  + Call by reference
* java.io.Console
* Java String Methods