

جاوا به زبان ساده



سر شناسه: ابراهیمی، یونس - ۱۳۶۰

عنوان و نام پدید آورنده: جاوا به زبان ساده / مؤلف: یونس ابراهیمی

مشخصات نشر: نپن دانش، تهران - ۱۳۹۶

مشخصات ظاهری: ۶۱۶ صفحه مصور

شابک:

وضعیت فهرست نویسی: فیبا

موضوع: جاوا (زبان برنامه نویسی کامپیوتر)

رده بندی دیوی:

رده بندی کنگره:

شماره کتاب شناسی ملی:

این اثر مشمول قانون حمایت مؤلفان، مصنفان و هنرمندان مصوب ۱۳۴۸ است. هر کس تمام یا قسمتی از این اثر را بدون اجازه ناشر، نشر یا پخش کند مورد پیگرد قانی قرار خواهد گرفت.

عنوان: جاوا به زبان ساده

مؤلف: یونس ابراهیمی

ناشر: نپن دانش

سال چاپ: ۱۳۹۶

نوبت چاپ: دوم

تیراز: ۲۰۰

قیمت: ۵۹۰۰۰ تومان

تقدیم به

همسر و پسر عزیزم

مبانی زبان جاوا

۱۵.	جاوا چیست؟
۱۷.	JVM چیست؟
۱۸.	JDK و NetBeans
۱۹.	نصب JDK و NetBeans
۲۶.	پیکربندی JDK
۳۰.	ساخت یک برنامه ساده در JAVA
۴۵.	استفاده از IntelliSense در NetBeans
۴۸.	رفع خطاهای NetBeans
۵۱.	کاراکترهای کنترلی
۵۳.	توضیحات
۵۵.	متغیر
۵۶.	انواع ساده
۵۷.	استفاده از متغیرها
۶۷.	ثابت
۶۳.	تبدیل ضمنی
۶۴.	تبدیل صریح
۶۵.	عبارات و عملگرها
۶۶.	عملگرهای ریاضی
۷۰.	عملگرهای تخصیصی
۷۱.	عملگرهای مقایسه ای
۷۳.	عملگرهای منطقی
۷۵.	عملگرهای بیتی
۸۱.	تقدم عملگرها
۸۳.	گرفتن ورودی از کاربر
۸۵.	ساختارهای تصمیم

۸۵.....	دستور if
۸۸.....	دستور if else
۸۹.....	دستور if تو در تو
۹۱.....	عملگر شرطی
۹۲.....	دستور if چندگانه
۹۳.....	استفاده از عملگرهای منطقی
۹۶.....	دستور switch
۱۰۰.....	تکرار
۱۰۰.....	حلقه While
۱۰۲.....	حلقه do While
۱۰۳.....	حلقه for
۱۰۶.....	آرایه‌ها
۱۰۹.....	حلقه foreach
۱۱۰.....	آرایه‌های چند بعدی
۱۱۶.....	آرایه دندانه دار
۱۱۸.....	متند
۱۲۰.....	مقدار برگشتی از یک متند
۱۲۳.....	پارامتر و آرگومان
۱۲۶.....	ارسال آرگومان به روش مقدار
۱۲۷.....	ارسال آرایه به عنوان آرگومان
۱۲۹.....	محدوده متغیر
۱۳۰.....	آرگومان های متغیر (VarArgs)
۱۳۱.....	سریارگذاری متدها

۱۳۲ بازگشت (Recursion)
۱۳۴ شمارش (Enumeration)
۱۳۸ آرگومان های خط فرمان (Command Line Arguments)
۱۳۹ برنامه نویسی شیء گرا (OOP)
۱۳۹ کلاس
۱۴۲ سازنده ..
۱۴۷ سطح دسترسی
۱۵۰ کپسوله سازی (Encapsulation)
۱۵۱ خواص (Properties)
۱۵۵ Package
۱۶۲ وراثت
۱۶۶ سطح دسترسی Protect
۱۶۷ اعضای static
۱۶۹ Override
۱۷۱ کلاس آبجکت (java.lang.Object)
۱۷۴ Unboxing و Boxing
۱۷۶ aggregation
۱۷۸ instanceof
۱۸۰ رابط (Interface)
۱۸۴ کلاسهای انتزاعی (Abstract Class)
۱۸۵ کلاس final و متدهای final
۱۸۶ چند ریختی (Polymorphism)
۱۹۱ کلاس های تو در تو (nested classes)
۱۹۲ کلاس داخلی استاتیک و غیر استاتیک
۱۹۴ کلاس های محلی (Local Classes)
۱۹۵ کلاس داخلی بی نام (Anonymous Inner Class)

۱۹۷	ایجاد آرایه ای از کلاسها
۱۹۸	عبارات لامبدا
۲۰۲	مدیریت استثناءها و خطایابی
۲۰۳	استثناءهای اداره نشده
۲۰۴	دستور try و catch
۲۰۷	بلوک finally
۲۰۸	ایجاد استثناء
۲۱۰	معرفی یک استثناء توسط کاربر
۲۱۱	مقایسه اشیاء با استفاده از رابطهای Comparable و Comparator
۲۱۸	کلکسیون‌ها (Collections)
۲۱۸	کلاس ArrayList
۲۲۳	ListIterator و Iterator
۲۲۶	Vector
۲۲۸	List
۲۳۱	Map
۲۳۵	Set
۲۳۷	HashSet
۲۴۰	LinkedList
۲۴۶	Queue
۲۴۹	HashMap
۲۵۳	TreeMap
۲۵۷	TreeSet
۲۶۴	Stack

۲۶۷	PriorityQueue
۲۷۳	Hashtable
۲۷۵	BitSet
۲۸۰	ArrayDeque
۲۸۴	Properties
۲۹۰	جنریک ها (Generics)
۲۹۰	متدهای جنریک
۲۹۲	کلاس جنریک
۲۹۴	کلکسیون عمومی (Generic Collection)
۲۹۶	Object Initializer

SWING

۳۰۰	برنامه نویسی ویژوال
۳۰۱	AWT چیست ؟
۳۰۲	SWING چیست ؟
۳۰۶	ایجاد یک برنامه Swing ساده
۳۰۸	JOptionPane
۳۱۲	کنترل کننده رویداد
۳۱۹	کنترل ها
۳۳۳	نامگذاری کنترل ها
۳۳۵	JFrame
۳۴۰	مدیریت لایه ها و چیدمان کنترل ها
۳۴۱	BorderLayout
۳۴۵	CardLayout
۳۴۸	FlowLayout
۳۵۰	GridLayout

۳۵۲	BoxLayout
۳۵۴	ایجاد حاشیه برای کنترل‌ها
۳۵۶	TitleBorder
۳۵۹	MatteBorder
۳۶۱	JButton
۳۶۴	کنترل JLabel
۳۶۶	کنترل JPasswordField و JTextField
۳۷۱	JTextArea
۳۷۴	JRadioButton
۳۷۶	JCheckBox
۳۷۹	JPanel
۳۸۱	کنترل JComboBox
۳۸۴	کنترل JList
۳۸۹	کنترل JSpinner
۳۹۱	کنترل JSlider
۳۹۶	JTabbedPane
۴۰۳	JMenuBar
۴۱۰	کنترل JToolBar
۴۱۴	کنترل JTree
۴۱۸	کنترل JToggleButton
۴۲۲	کادرهای محاوره ای (Dialogs)
۴۲۴	کنترل JFileChooser
۴۳۲	کنترل JColorChooser

کار با تاریخ، فایل و رشته

۴۳۸	کلاس Date
۴۴۴	کلاس Math
۴۴۸	ایجاد عدد تصادفی
۴۵۲	رشته‌ها و عبارات با قاعده
۴۵۴	کلاس String
۴۵۳	مقایسه رشته‌ها
۴۵۵	الحق یا چسباندن رشته‌ها
۴۵۶	تکه تکه کردن رشته‌ها
۴۵۷	جستجوی رشته‌ها
۴۶۰	تغییر بزرگی و کوچکی حروف یک رشته
۴۶۱	استخراج و جایگزین کردن رشته‌ها
۴۶۲	جایگزین کردن رشته‌ها با استفاده از متند replace
۴۶۳	فرمت بندی رشته‌ها و اعداد
۴۶۸	کلاس StringBuilder
۴۷۱	File System
۴۷۱	پکیج Java IO
۴۷۴	کلاس‌های Reader و Writer
۴۷۵	کلاس‌های OutputStream و InputStream
۴۷۶	کلاس File
۴۸۴	کلاس InputStreamReader
۴۸۸	کلاس OutputStreamWriter
۴۹۰	کلاس RandomAccessFile
۴۹۵	کلاس ByteArrayInputStreamStream
۴۹۷	کلاس ByteArrayOutputStream

۴۹۹	کلاس‌های ObjectOutputStream و ObjectInputStream
۵۰۶	کلاس BufferedReader
۵۱۰	کلاس BufferedWriter
۵۱۲	کلاس StringReader
۵۱۵	کلاس StringWriter
۵۱۶	کلاس PrintWriter
۵۲۰	زبان نشانه گذاری توسعه پذیر (XML)
۵۲۴	مدیریت فایل‌های XML
۵۳۰	ساخت XML با روش مبتنی بر DOM
۵۳۲	ساخت XML با روش مبتنی بر Stream
۵۳۴	پرس و جوی محتوای XML با XPath
۵۳۶	استفاده از XPath

کار با بانک اطلاعاتی

۵۴۱	MySQL چیست؟
۵۴۱	MySQL مبانی
۵۴۵	دستورات MySQL
۵۴۷	نصب سرور MySQL
۵۵۸	نصب نرم افزار MySQL Administrator و آشنایی با محیط آن
۵۶۳	آشنایی با محیط MySQL Administrator
۵۶۵	ایجاد جدول و دیتابیس با استفاده از محیط کنسول MySQL
۵۷۱	ایجاد جدول و دیتابیس با استفاده از محیط MySQL Administrator
۵۸۰	JDBC چیست؟

۵۸۲	JDBC Driver چیست؟
۵۸۴	ارتباط با بانک
۵۸۶	اجرای دستورات بر روی بانک
۵۹۹	پاک کردن اشیاء بی استفاده و آزاد کردن حافظه
۶۰۱	ثبت، حذف، ویرایش و انتخاب اطلاعات با استفاده از NetBeans

مقدمه

همگام با پیشرفت فناوری‌های دیگر، زبان‌های برنامه نویسی نیز ارتقا پیدا کردند. وقتی زبان JAVA طراحی و پیاده سازی شد، تحول بزرگی در دنیای برنامه نویسی به وجود آمد. این زبان برنامه نویسی موفق که در سال ۱۹۹۵ به طور رسمی به بازار معرفی شد، توانست چنان محبوبیتی در جهان پیدا کند که در حال حاضر در بیش از ۳ میلیارد سیستم مورد استفاده قرار گرفته و تاکنون بیش از ۱۰۰۰ جلد کتاب پیرامون آن به رشته تحریر درآمده است. خوشبختانه JAVA این روزها به عنوان یکی از دروس رشته‌های کامپیوتر در دانشگاه‌های کشور تدریس می‌شود و این نشان از توانایی‌ها و اهمیت این زبان است. منابع متعددی برای معرفی و به کارگیری زبان JAVA عرضه شده است که جای تقدیر و تشکر دارد. اما کتاب حاضر دارای ویژگی‌های بارزی از جمله، بیان ساده مطالب، ارائه مثال‌های متنوع، بررسی دقیق و موشکافانه موضوعات و سلسله مراتب آموزشی است. مثال‌هایی که در کتاب ارائه شده‌اند همگی دارای هدف خاصی هستند به طوریکه هر کدام، یک یا چند نکته زبان جاوا را به خواننده آموزش می‌دهند. در این کتاب ما به شما نحوه برنامه نویسی به زبان جاوا را به صورت تصویری آموزش می‌دهیم. سعی کنید حتماً بعد از خواندن مباحث، آن‌ها را به صورت عملی تمرین کنید و اینکه قابلیت و مفهوم کدها را بفهمید نه آنها را حفظ کنید.

برای مطالعه مطالب جدید جاوا به سایت www.w3-farsi.com مراجعه فرمایید.

راه‌های ارتباط با نویسنده

وب سایت: www.w3-farsi.com

لينك تلگرام: https://telegram.me/ebrahimi_younes

ID تلگرام: @ebrahimi_younes

پست الکترونیکی: younes.ebrahimi.1391@gmail.com

فصل اول

مبانی زبان جاوا

جاوا چیست؟

جاوا (JAVA) یک زبان برنامه نویسی شیءگرای است که نخستین بار توسط James Gosling (جیمز گاسلینگ) در شرکت Sun Microsystems ایجاد گردید. در سال ۱۹۹۰ شرکت Sun Microsystems در حال توسعه نرمافزاری برای استفاده ابزارهای الکترونیکی بود که مسئولیت تیم، که آن را، تیم پروژه Green نامیدند، جیمز گاسلینگ بر عهده گرفت.

در سال ۱۹۹۱ تیم تصمیم گرفت که زبان جدید را OAK (بلوط) بنامند. علت این نام گذاری وجود درختان بلوط در محوطه اطراف ساختمان محل کار اعضای تیم Green بود. در سال ۱۹۹۲ تیم پروژه Green زبان جدیدی را معرفی کرد که با ابزارهای مختلف خانگی و لمسی کار می‌کرد. در سال ۱۹۹۳ وب جهانی توسعه یافت و زبان OAK با معرفی Applet که قابلیت‌های زیادی به کامپیوترهای متصل به وب می‌افزود، مشهور شد.

در سال ۱۹۹۵ زبان OAK به تغییر نام پیدا کرد و توسط Microsoft و Netscape پشتیبانی شد. از آنجا که مراسم تغییر نام در کافی شاپ برگزار شده بود و همچنین علاقه اعضا تیم Green به قهوه، یک فنجان قهوه داغ به عنوان نماد جاوا در نظر گرفته شد. معتبرترین داستان درباره دلیل این نامگذاری این است که، جیمز گاسلینگ به نوعی قهوه علاقه داشت، که در جزیره‌ای به نام جاوا، که در اندونزی در جنوب شرقی آسیا است، می‌روید. کلاً زبان برنامه نویسی جاوا به سه دسته کلی تقسیم می‌شود:

- JAVA Standard Edition یا JAVA SE برای نوشتن برنامه‌های کوچک دستکتابی کاربرد دارد.
- JAVA Micro Edition یا JAVA ME برای برنامه نویسی برای منابع سخت افزاری (MEMORY, CPU) محدود، مثل موبایل و لوازم خانگی کاربرد دارد.
- JAVA Enterprise Edition یا JAVA EE برای برنامه‌های بزرگ که معمولاً بر روی شبکه‌های بزرگ مخصوصاً اینترنت نصب و اجرا می‌شوند کاربرد دارد.

تاریخچه جاوا

از زمان انتشار اولین نسخه جاوا (java 1.0) تا به امروز، شرکت Sun تقریباً هر دو سال یکبار نسخه‌ای جدیدی از این زبان را منتشر می‌نماید. در این نسخه تازه، معمولاً قابلیت‌های جدیدی افزوده شده و ایرادهای نسخه قبل رفع می‌شوند. نکته قابل توجه در مورد شماره گذاری نسخه‌های مختلف جاوا آن است که تا چهارمین نسخه آن شماره گذاری بصورت $x.x$ بود که x همان شماره نسخه مورد

نظر می‌باشد. از نسخه پنجم به بعد شماره گذاری بصورت x Java تغییر یافت. یعنی بجای اینکه نسخه پنجم را بصورت ۱.۵ نامگذاری کنند، بصورت ۵.۰ java نامگذاری کردند. در ادامه به معرفی نسخه‌های مختلف جاوا بر اساس نسخه پایه‌ای آن یا همان نسخه استاندارد (Standard Edition(SE)) می‌پردازیم. این نسخه شامل همه ملزومات مورد نیاز جهت Desktop Programming می‌باشد.

در جدول زیر نسخه‌های مختلف جاوا و ویژگی‌های آنها ذکر شده است :

نام کد	نسخه	تاریخ پیدایش
Oak	java 1.0	January 1996
	java 1.1	February 1997
playground	J2SE 1.2	December 1998
Kestrel	J2SE 1.3	May 2000
Merlin	J2SE 1.4	February 2002
Tiger	J2SE 5.0	September 2004
Mustang	Java SE 6	December 2006
Dolphin	Java SE 7	July 2011
	Java SE 8	March 2014
	Java SE 9	September 2017
	Java SE 10	March 2018
	Java SE 11 (18.9 LTS)	September 2018
	Java SE 12 (19.3)	March 2019

برای آشنایی بیشتر با این زبان به لینک‌های زیر مراجعه کنید :

[https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))

https://en.wikipedia.org/wiki/Java_version_history

ضریب اطمینان عملکرد برنامه‌های نوشته شده به این زبان بالا است و وابسته به سیستم عامل خاصی نیست، به عبارت دیگر می‌توان آن را روی هر رایانه با هر نوع سیستم عاملی اجرا کرد و این، همان شعار جاوا است: "یک بار بنویس، همه جا اجرا کن".

آموزش ویدئویی جاوا به زبان ساده

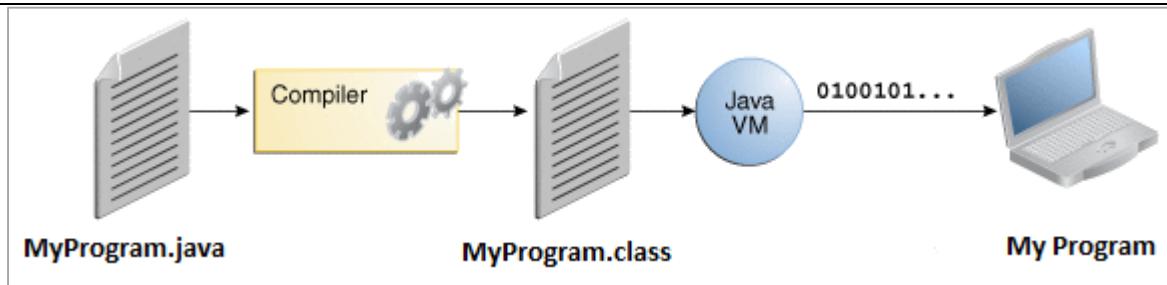
برای اطلاعات بیشتر به لینک زیر مراجعه فرمایید

www.w3-farsi.com/product

W3-farsi.com تخصصی‌ترین سایت آموزش جاوا در ایران

JVM چیست؟

برای اجرای برنامه‌های نوشته شده و کامپایل شده به زبان جاوا نیاز به سکویی یا برنامه‌ای است که به آن ماشین مجازی جاوا (Java Virtual Machine) یا به اختصار JVM گفته می‌شود. این ماشین کدهای کامپایل شده به زبان جاوا را گرفته و آنها را اجرا می‌کند. شاید این جمله را شنیده باشید که کدهای زبان جاوا بر روی هر ماشین قابل اجرا می‌باشند و اصطلاحاً Java Multi Platform است. شخصی که دستگاهی با سیستم عامل ویندوز دارد، از سایت سان میکروسیستمز، JVM مربوط به سیستم عامل ویندوز را نصب می‌کند. سپس برنامه‌ای را به زبان جاوا می‌نویسد و آن را کامپایل می‌نماید. پس از آن برنامه کامپایل شده را برای دوست خود که دستگاه دیگری با سیستم عامل لینوکس دارد، ارسال می‌کند. این شخص قبلاً JVM مخصوص سیستم عامل لینوکس را از سایت Sun Microsystems دانلود و بر روی دستگاه خود نصب نموده است. به همین دلیل هیچکدام از این دو نفر لازم نیست نگران باشند که سیستم عامل دستگاه‌هایشان با یکدیگر متفاوت است. می‌توان نحوه اجرای کدهای جاوا را به صورت زیر خلاصه کرد :



همانطور که در شکل بالا مشاهده می‌کنید :

- برنامه‌نویس کدهای خود را درون فایلی با پسوند `.java` می‌نویسید.
- وقتی برنامه نویس برنامه خود را اجرا می‌کند، کدهای برنامه توسط کمپایلر جاوا به `bytecode` تبدیل می‌شوند و درون فایلی با همان نام قبلی اما این بار با پسوند `.class` ذخیره می‌شوند.
- ماشین مجازی جاوا (`Java Virtual Machine`) فایل `.class` را اجرا می‌کند.

ماشین مجازی جاوا یا `JVM` بر روی تمام سیستم‌عامل‌های مطرح (ویندوز، مکینتاش و لینوکس) قابل نصب است. به همین دلیل فایل `.class` برنامه شما در تمام این سیستم‌عامل‌ها می‌تواند اجرا شود و به همین دلیل است که به جاوا زبان مستقل از سیستم‌عامل گفته می‌شود. شعار جاوا این است: یک بار بنویس، همه جا اجرا کن!

JDK و NetBeans

محیط توسعه یکپارچه‌ای است که دارای ابزارهایی برای کمک به شما برای توسعه برنامه‌های `JAVA` می‌باشد. توصیه می‌کنیم که از محیط `Netbeans` برای ساخت برنامه استفاده کنید، چون این محیط دارای ویژگی‌های زیادی برای کمک به شما جهت توسعه برنامه‌های `JAVA` می‌باشد. توسط `Netbeans` می‌توان در استانداردهای مختلف جاوا مانند `J2EE`, `J2SE` و `J2ME` برنامه نویسی کرد. همچنین از محیط زبان‌های `PHP`, `HTML`, `C` و نیز `Groovy` پشتیبانی می‌کند. قبل از نصب `Netbeans` می‌بایست `JDK` را نصب نمایید، در غیر این صورت برای نصب دچار مشکل خواهد شد. `JDK` که مخفف عبارت `Java Development Toolkit` می‌باشد ترکیبی از کمپایلر زبان جاوا، کلاس‌های کتابخانه‌ای (`Java Class Libraries`) و `JVM` و فایل‌های راهنمای آن‌ها می‌باشد. برای اینکه ما بتوانیم با استفاده از زبان برنامه نویسی جاوا، برنامه بنویسیم به این مجموعه نیاز داریم. تعداد زیادی از پردازش‌ها که وقت شما را هدر می‌دهند به صورت خودکار توسط `NetBeans` انجام می‌شوند. یکی از این ویژگی‌ها اینتلی سنس (`Intellisense`) است که شما را در تایپ سریع کدهایتان کمک می‌کند. `NetBeans` برنامه شما را خطایابی می‌کند و حتی خطاهای کوچک (مانند بزرگ یا کوچک نوشتتن حروف) را برطرف

می‌کند. با این برنامه‌های قدرتمند بازدهی شما افزایش می‌یابد و در وقت شما با وجود این ویژگی‌های شگفت‌انگیز صرفه جویی می‌شود آزاد است و می‌توان آن را دانلود و از آن استفاده کرد. این برنامه ویژگی‌های کافی را برای شروع برنامه نویسی JAVA در اختیار شما قرار می‌دهد. در آموزش‌ها از NetBeans نسخه ۸,۰/۲ استفاده شده است و استفاده از این نسخه برای انجام تمرینات این سایت کافی می‌باشد. برای دانلود نرم افزارهای مورد نیاز به سایت [w3-farsi.com](http://www.w3-farsi.com) و لینک زیر مراجعه کنید :

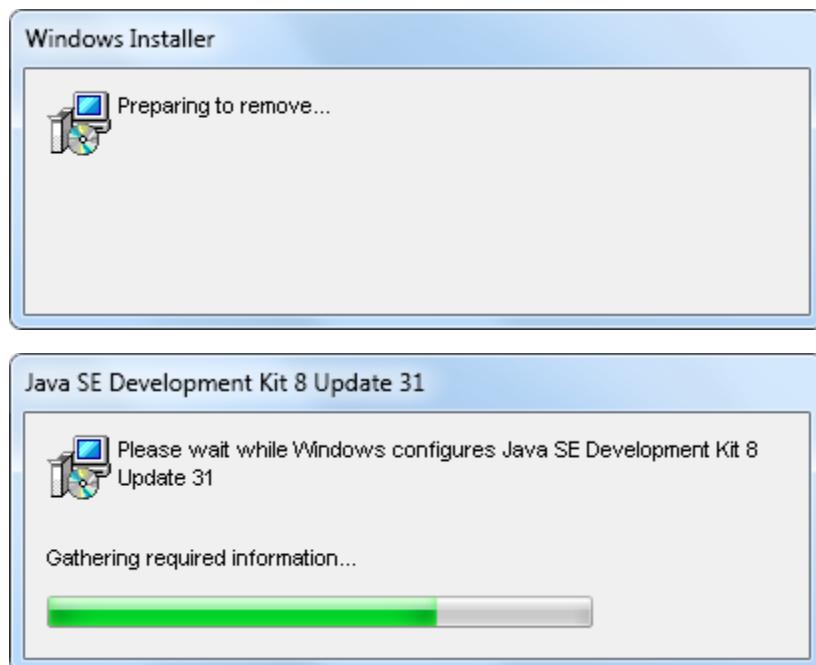
<http://www.w3-farsi.com/?p=7529>

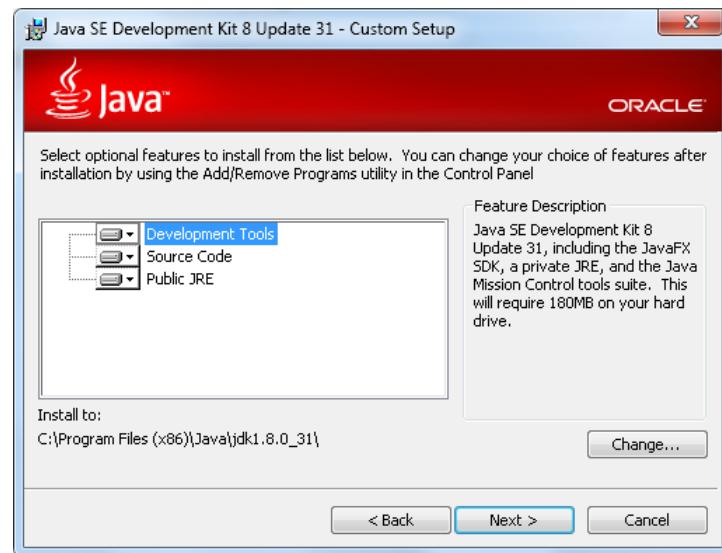
در درس آینده مراحل نصب و راه اندازی دو نرم افزار JDK و NetBeans را توضیح می‌دهیم.

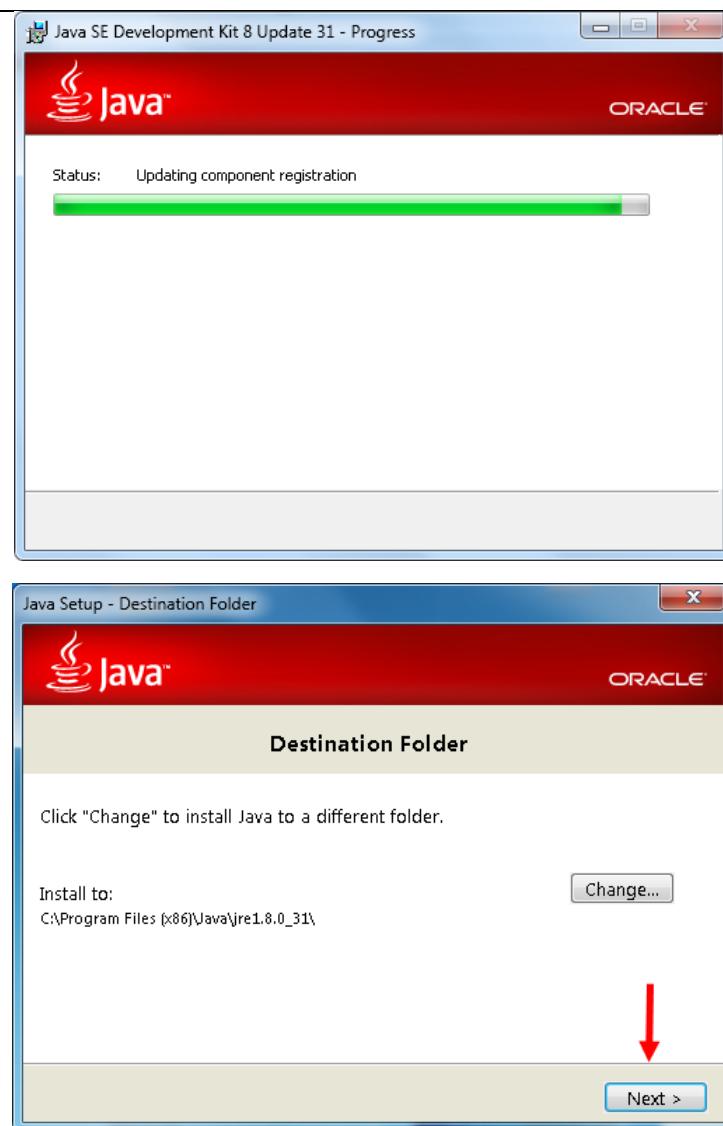
نصب JDK و NetBeans

در درس قبل در مورد نرم افزارهای NetBeans و JDK توضیحات مختصری ارائه دادیم. در این درس می‌خواهیم شما را با نحوه نصب این دو نرم افزار آشنا کنیم. نصب این نرم افزارها مانند اکثر نرم افزارهای دیگر بسیار آسان بود و بعد از زدن چند دکمه Next نصب می‌شوند. در زیر مراحل تصویری نصب این دو نرم افزار نشان داده شده است.

نصب JDK

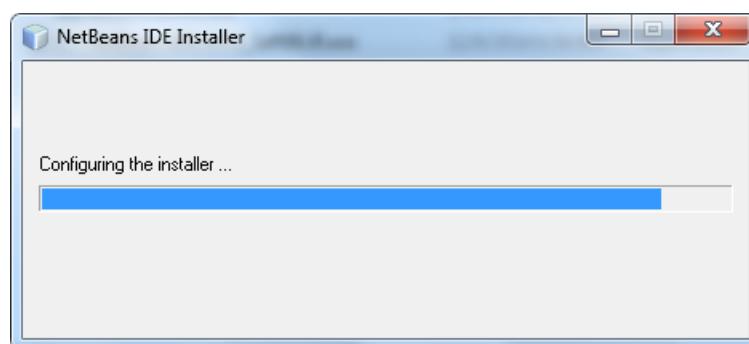


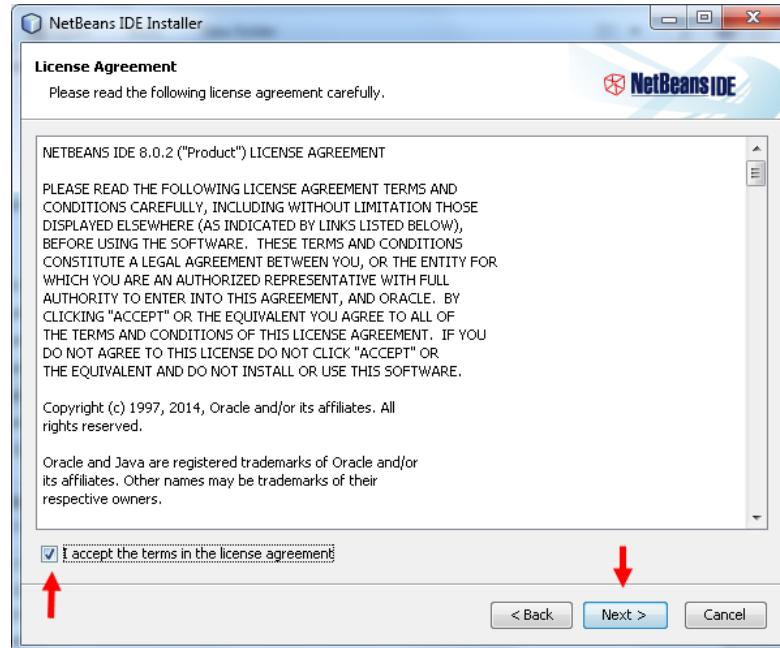
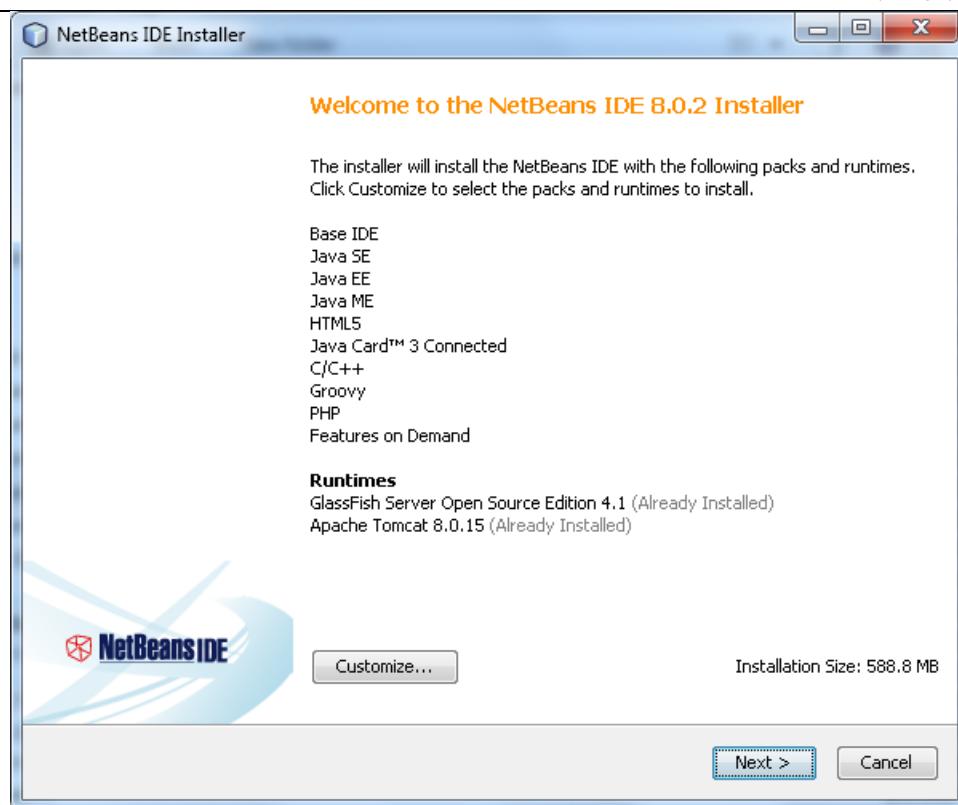


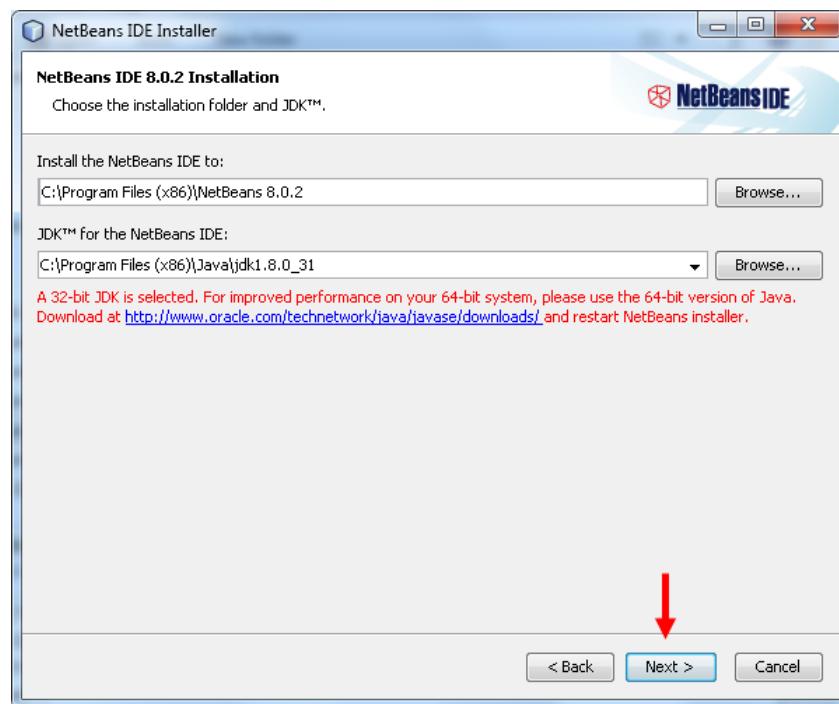
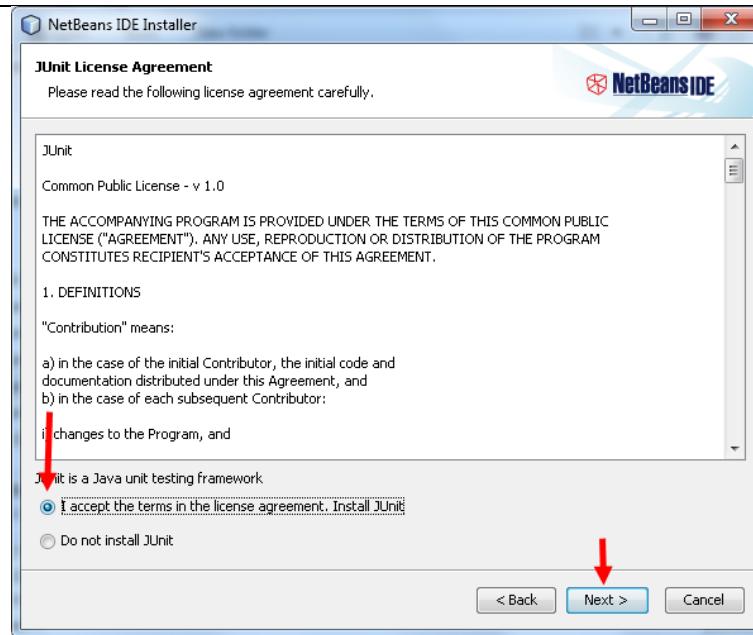


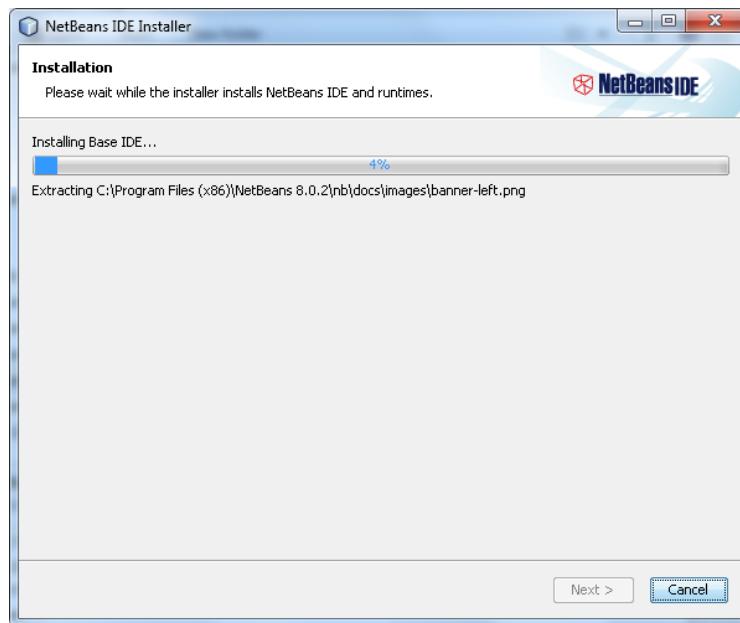
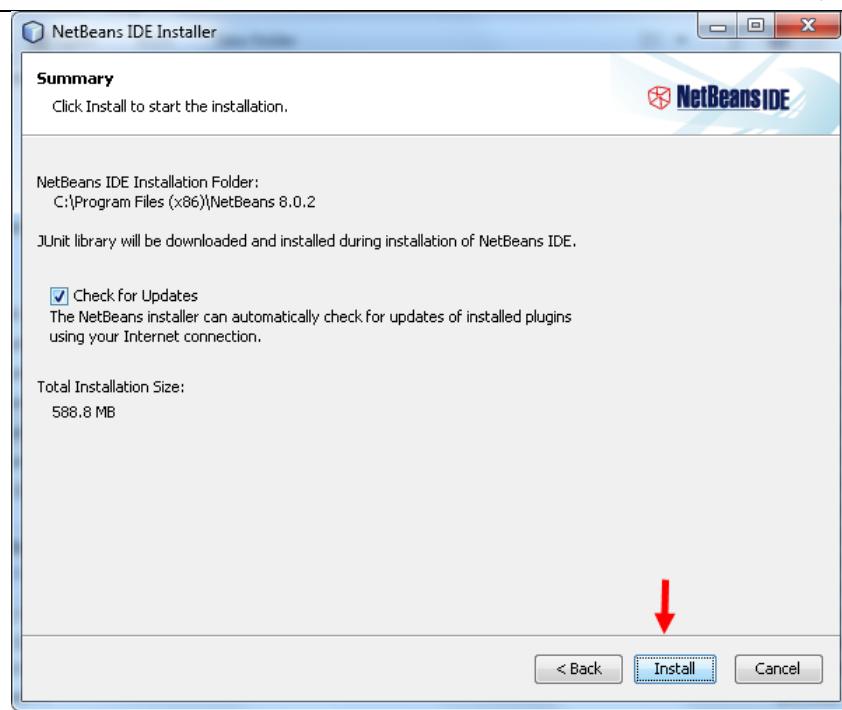


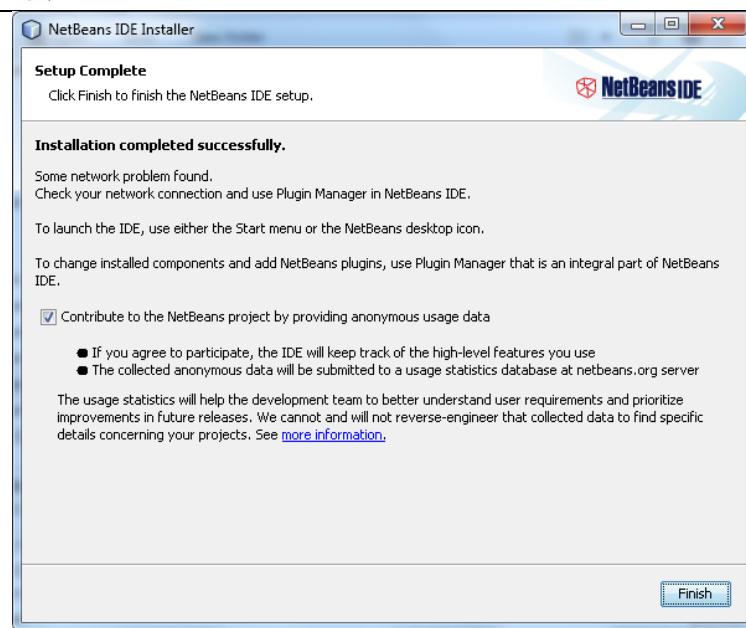
نصب NetBeans





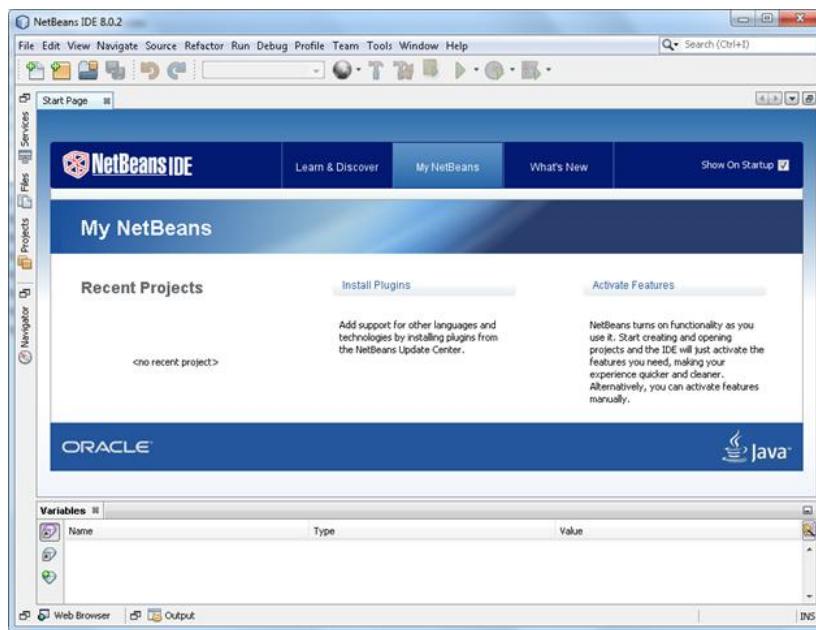






وقتی برای اولین بار بر روی آیکون NetBeans بر روی دسکتاپ کلیک کرده و آن را اجرا می‌کنید، صفحه اول برنامه به صورت زیر نمایش

داده می‌شود که نشان دهنده نصب کامل آن است :



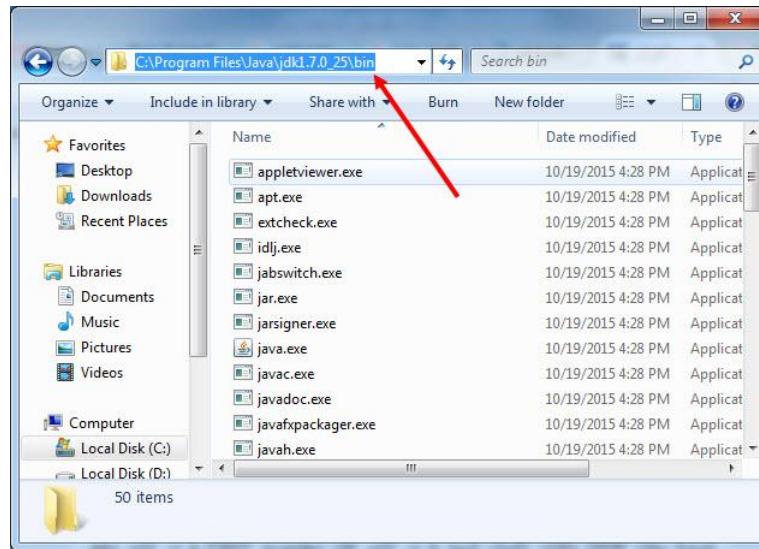
در درس آینده درباره ایجاد پروژه در NetBeans توضیح می‌دهیم.

پیکربندی JDK

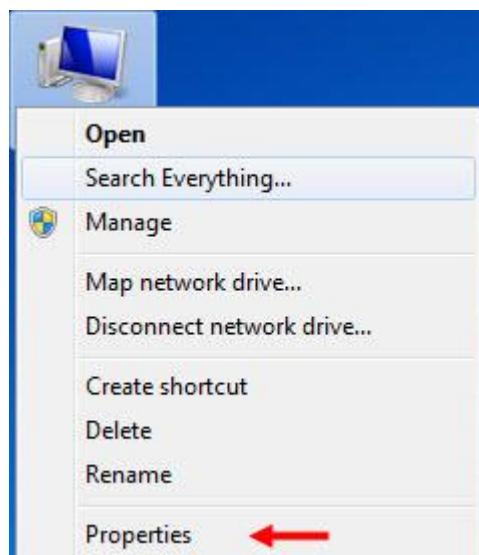
در درس قبل ما نسخه نهایی JDK یا Java Development Kit را بر روی سیستم عاملمان نصب کردیم. ولی این پایان کار نیست. برای

اجرای برنامه‌های جاوا لازم است که مسیر پوشه bin این نرم افزار را در متغیر path معرفی کنیم. برای این کار ابتدا مسیر پوشه bin را

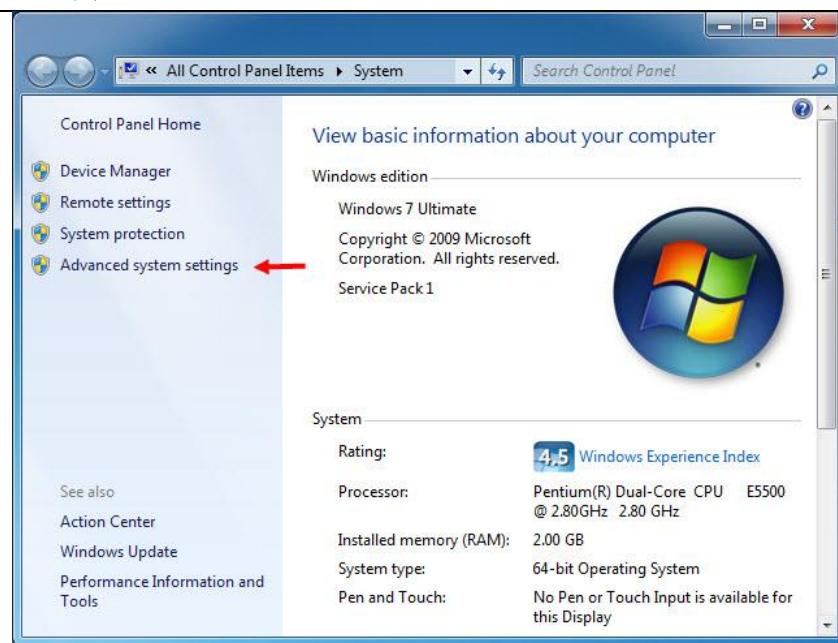
کپی کنید :



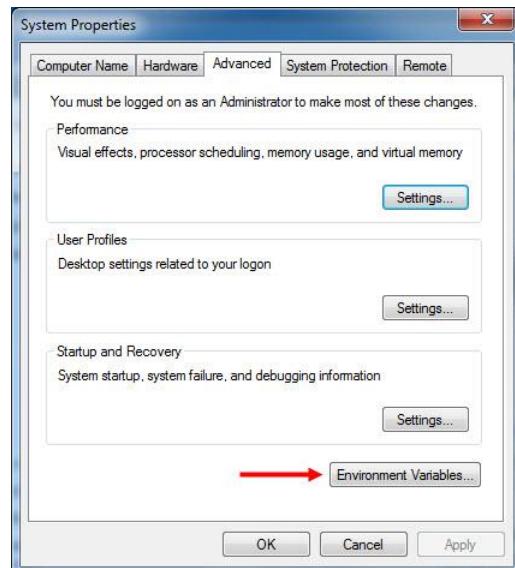
سپس بر روی MyComputer راست کلیک کرده و روی گزینه Properties کلیک کنید :



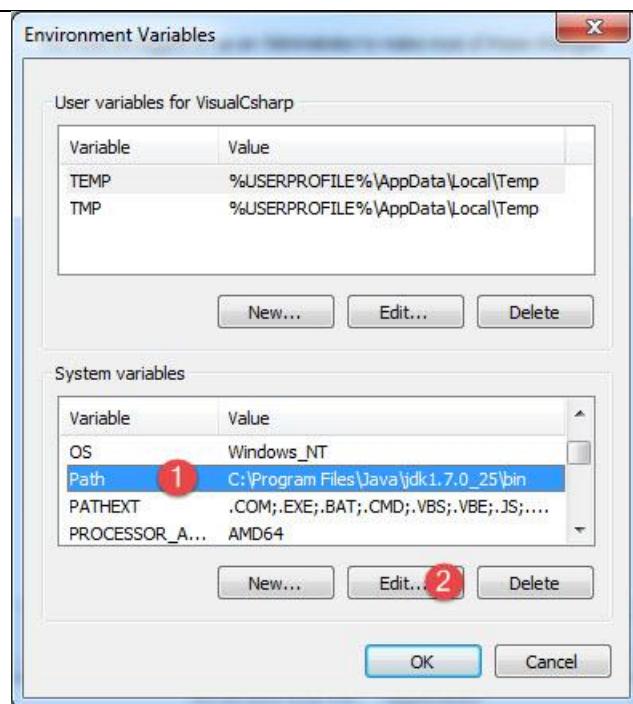
از پنل سمت چپ این صفحه Advanced system settings را باز کنید:



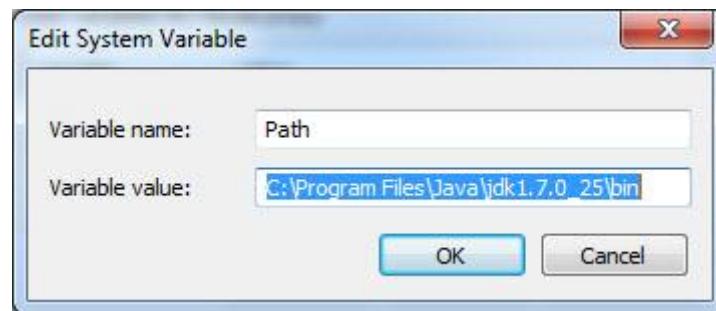
به روی tab Advanced کلیک کنید.



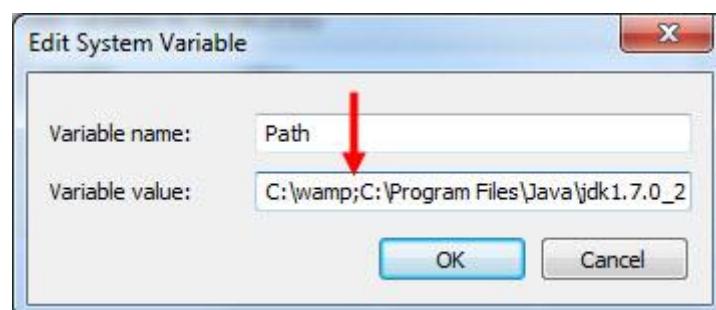
در قسمت پایین و بخش System Variables روی گزینه Path کلیک کرده و سپس گزینه Edit را بزنید:



در پنجره باز شده اگر قسمت Value Variable خالی بود مسیر پوشه bin را در آن کپی کنید :



و اگر از قبل مسیرهای دیگری وجود داشت ابتدا علامت سمیکالان (;) را در انتهای آنها گذاشته و سپس مسیر پوشه bin را کپی می کنید :



حال cmd را اجرا می‌کنیم :



و کد زیر را در داخل آن می‌نویسیم و دکمه Enter را می‌زنیم :

```
javac -version
```

مشاهده می‌کنید که خطای برطرف شده و نسخه JDK نمایش داده می‌شود که نشان دهنده این است که مراحل را درست انجام داده‌اید:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\JavaTutorials>javac -version
javac 1.7.0_25

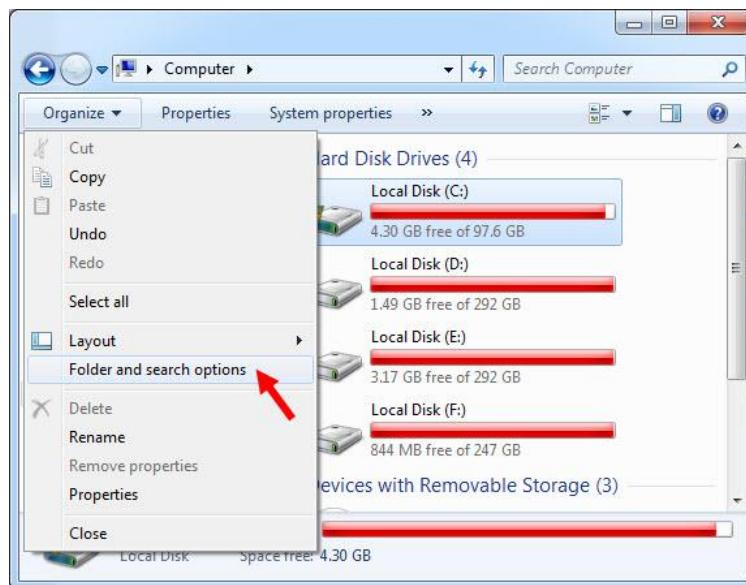
C:\Users\JavaTutorials>
```

ساخت یک برنامه ساده در JAVA

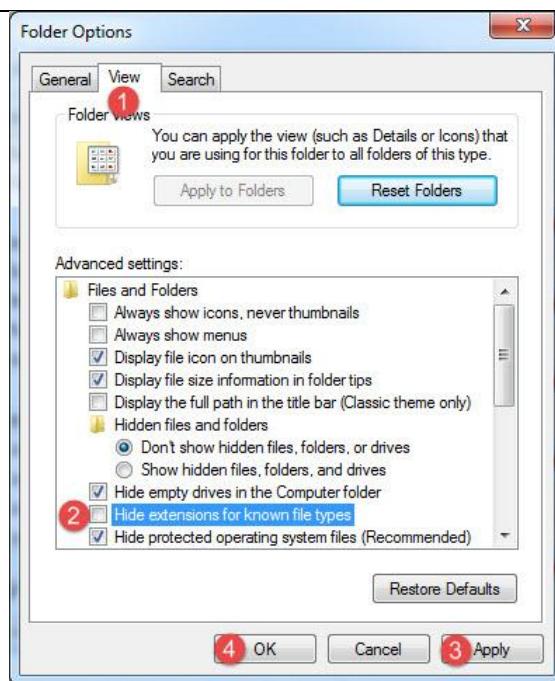
اجازه بدهید یک برنامه بسیار ساده به زبان جاوا بنویسیم. این برنامه یک پیغام را نمایش می‌دهد. در این درس می‌خواهم ساختار و دستور زبان یک برنامه ساده جاوا را توضیح دهم.

قبل از ایجاد برنامه به این نکته توجه کنید که کدهای جاوا را می‌توان در داخل یک ویرایشگر متن ساده مانند NotePad نوشت و اجرا کرد. فقط کافیست که JDK بر روی سیستم شما نصب باشد. استفاده از نرم افزارهایی مانند NetBeans فقط برای راحتی در کدنویسی و کاهش خطای می‌باشد.

یکی از تنظیماتی که قبل از شروع این درس توصیه می‌کنیم که اعمال کنید این است که پسوند فایل‌ها داخل ویندوز را قابل مشاهده کنید. برای این کار به My Computer رفته و به صورت زیر از منوی Organize گزینه Folder and search options را بزنید:



از پنجره باز شده به صورت زیر به سربرگ View رفته و تیک کنار گزینه Hide Extension for known file types را بردارید:



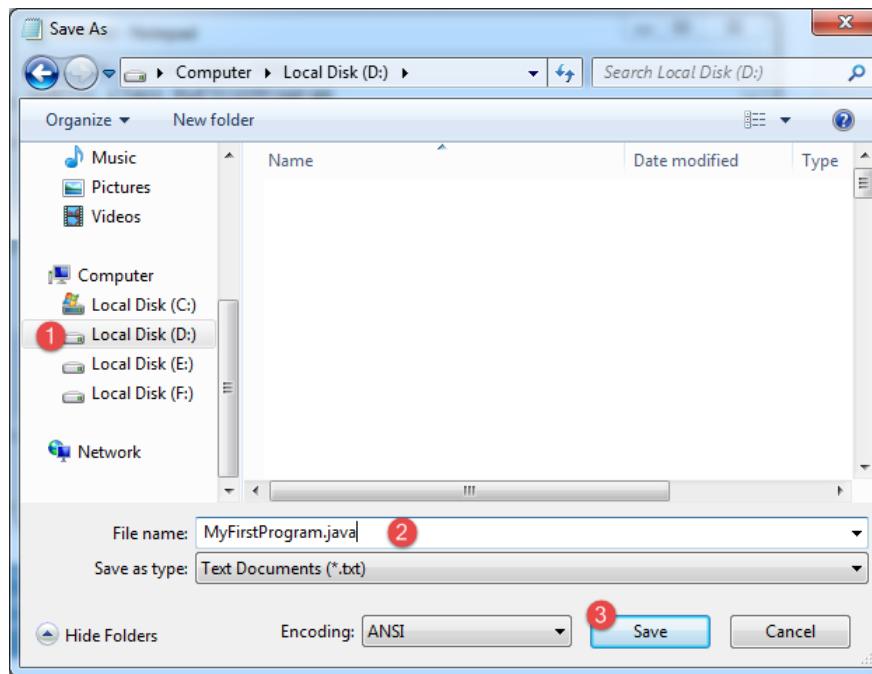
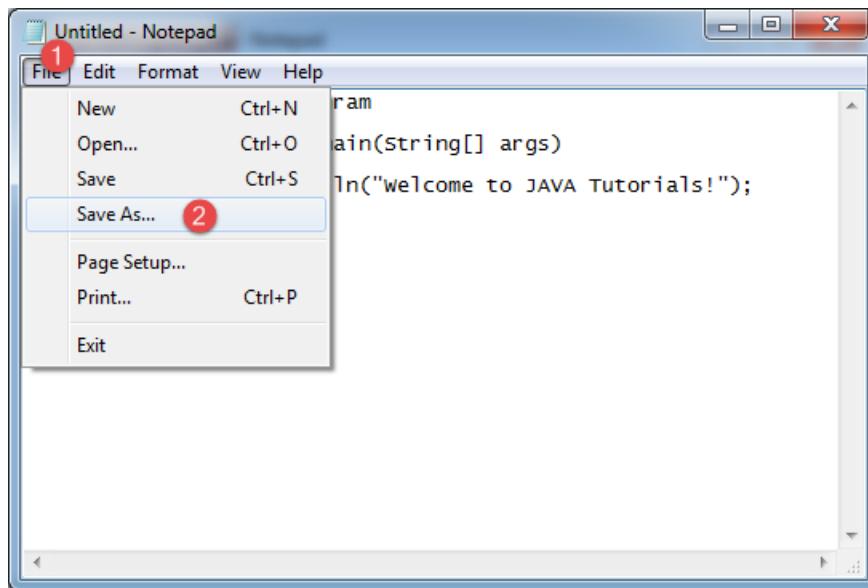
در ادامه شما را نحوه ایجاد اولین برنامه در جاوا با و بدون استفاده از NetBeans آشنا می کنیم.

بدون استفاده از NetBeans

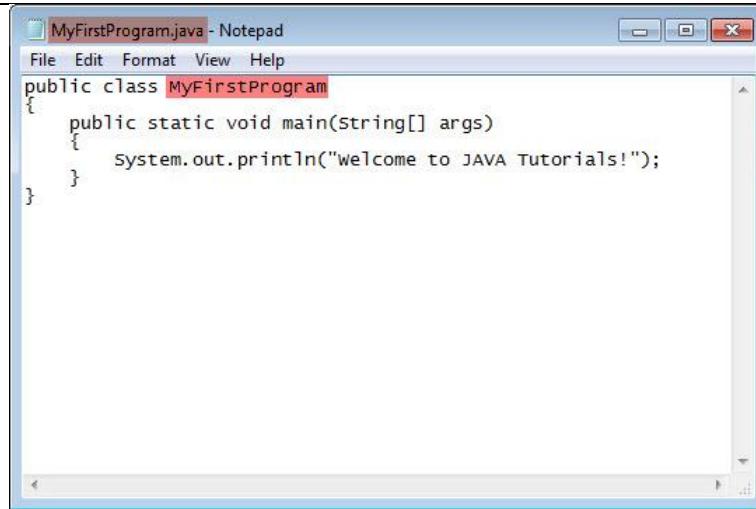
همانطور که گفته شد، شما برای کامپایل و اجرای برنامه‌های جاوا به ابزاری به نام JDK نیاز دارید. JDK مخفف عبارت Java Development Kit است و شامل ابزارهای مورد نیاز شما برای اجرای برنامه‌های جاوا می‌شود. این مجموعه شامل ابزاری به نام JVM یا Java Virtual Machine است که ماشین مجازی جاوا نام دارد و وظیفه‌ی کامپایل و اجرای کدهای شما را برعهده دارد. خود JVM هم شامل ابزارهای دیگری است. مثلاً java compiler یا javac کامپایل کردن برنامه‌ها را برعهده دارد. در درس قبل JDK را نصب کردیم و الان فرض می‌کنیم که شما هیچ IDE مانند netbeans و یا eclipse در اختیار ندارید و می‌خواهید یک برنامه جاوا بنویسید. در این برنامه می‌خواهیم پیغام Welcome to JAVA Tutorials را در داخل آن نوشه (حروف بزرگ و کوچک را رعایت کنید) و با پسوند .java ذخیره کنید:

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to JAVA Tutorials!");
    }
}
```

و در درایو D و با نام و پسوند MyFirstProgram.java ذخیره می‌کنیم :

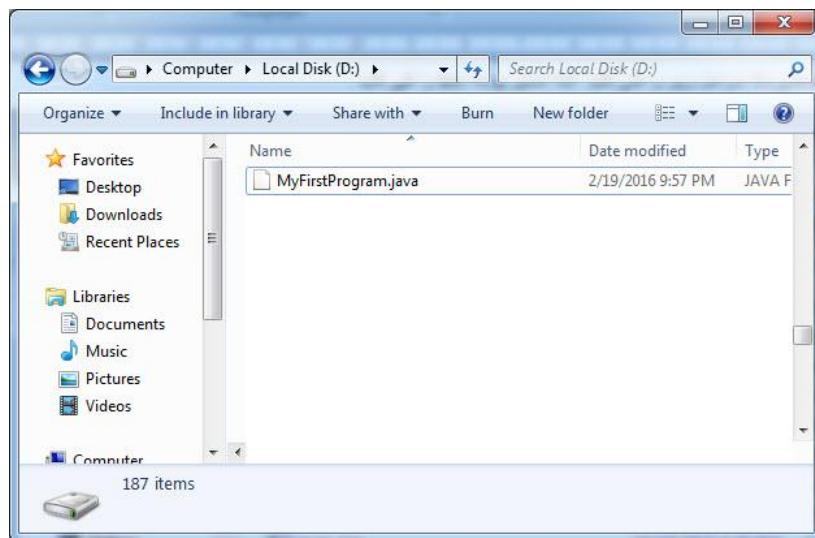


نگران توضیح کدهای بالا نباشید، در ادامه در مورد آنها توضیح می‌دهیم. پس شکل نهایی برنامه، باید به صورت زیر باشد :



```
MyFirstProgram.java - Notepad
File Edit Format View Help
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("welcome to JAVA Tutorials!");
    }
}
```

حال نوبت به اجرای برنامه می‌رسد :



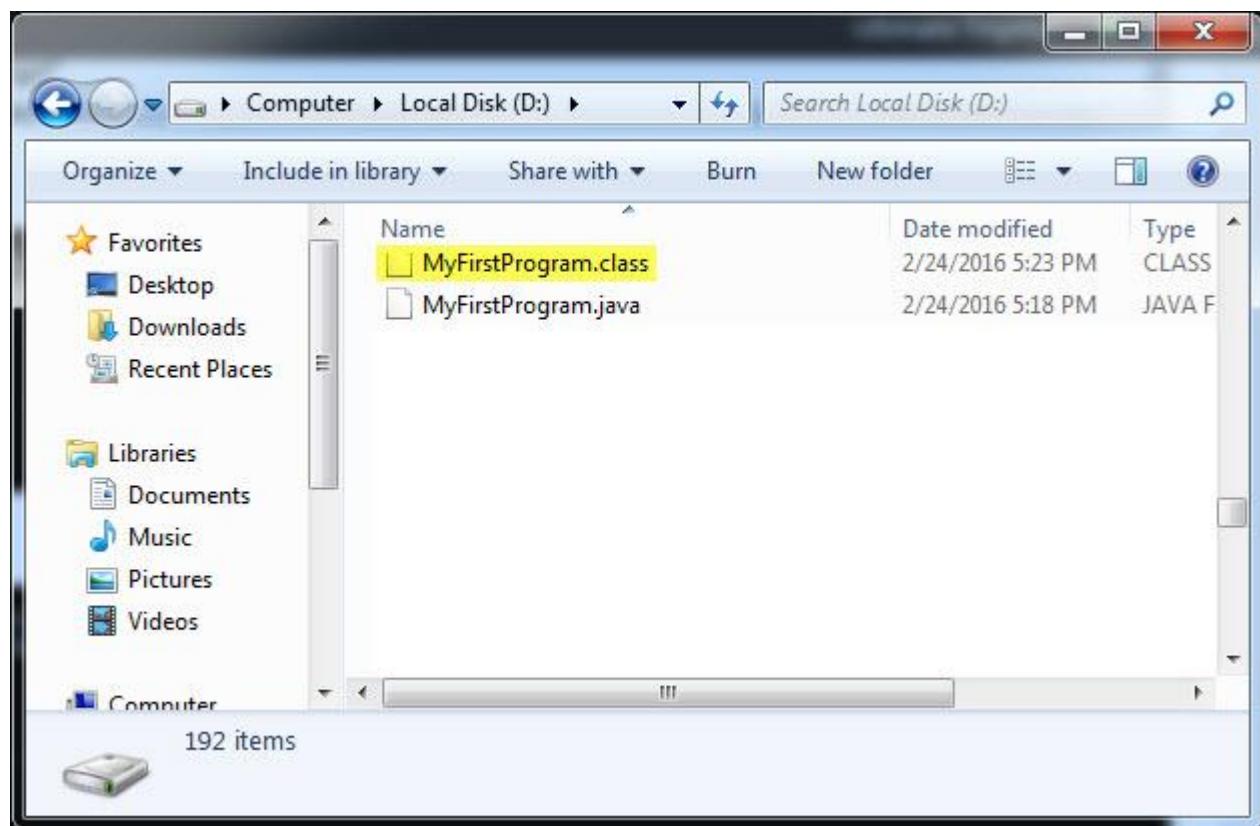
فایل ما در درایو D قرار دارد. ابتدا cmd را باز کرده و کد زیر را در داخل آن نوشته و دکمه Enter را می‌زنید :

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\JavaTutorials>d:
D:>javac MyFirstProgram.java
D:>
```

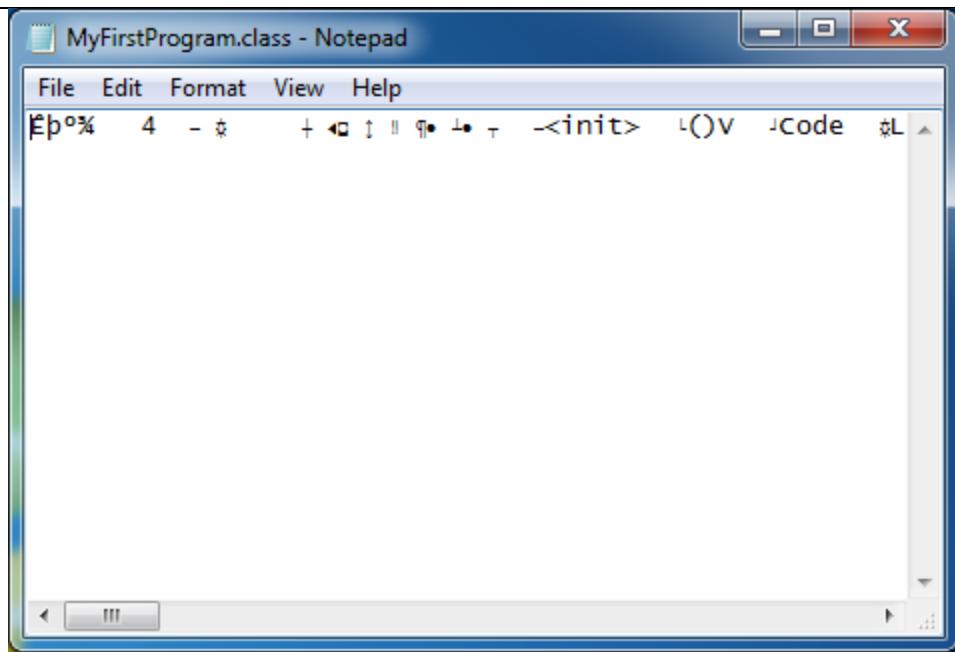
با اجرای کد بالا هیچ پیغامی چاپ نمی‌شود چون که دستور `javac` برنامه را کامپایل کرده و یک فایل همنام با کلاس `MyFirstProgram`

و با پسوند `.class` ایجاد می‌کند :



اگر فایل ایجاد شده یعنی `MyFirstProgram.class` را با برنامه NotePad باز کنید مشاهده می‌کنید که شامل کدهایی نا مفهوم می‌باشد

:



به این کدهای نامفهوم که برای ماشین مجازی جاوا (JVM) قابل فهم می‌باشد، گفته می‌شود. حال برای اجرای فایل MyFirstProgram.class باید دستور زیر را بنویسیم :

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\JavaTutorials>d:
D:>javac MyFirstProgram.java

D:>java MyFirstProgram
Welcome to JAVA Tutorials!

D:>
```

مشاهده می‌کنید که فایل جاوا اجرا و پیغام Welcome to JAVA Tutorials چاپ شد.

یک نکته را متذکر می‌شویم و آن این است که نام فایل دارای پسوند .java باید با نام کلاسی که در داخل آن است، یکی باشد، در غیر اینصورت با خطأ مواجه می‌شویم. فرض کنید ما نام فایل بالا را به Program.java تغییر داده و آن را کامپایل می‌کنیم. در اینصورت خطای زیر به ما نمایش داده می‌شود :

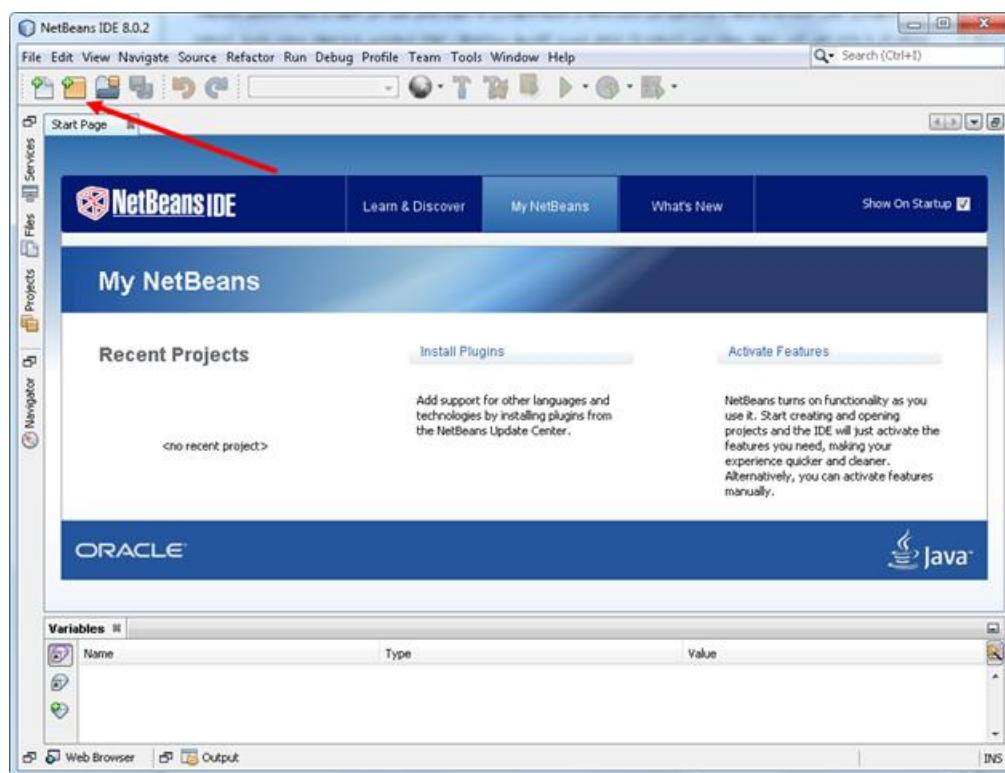
```
Program.java:3: error: class MyFirstProgram is public,
should be declared in a file named MyFirstProgram.java
```

```
public class MyFirstProgram
^
1 error
```

خطای بالا به این معنی است که کلاسی با نام MyFirstProgram باید در فایلی با نام MyFirstProgram.java تعریف شود.

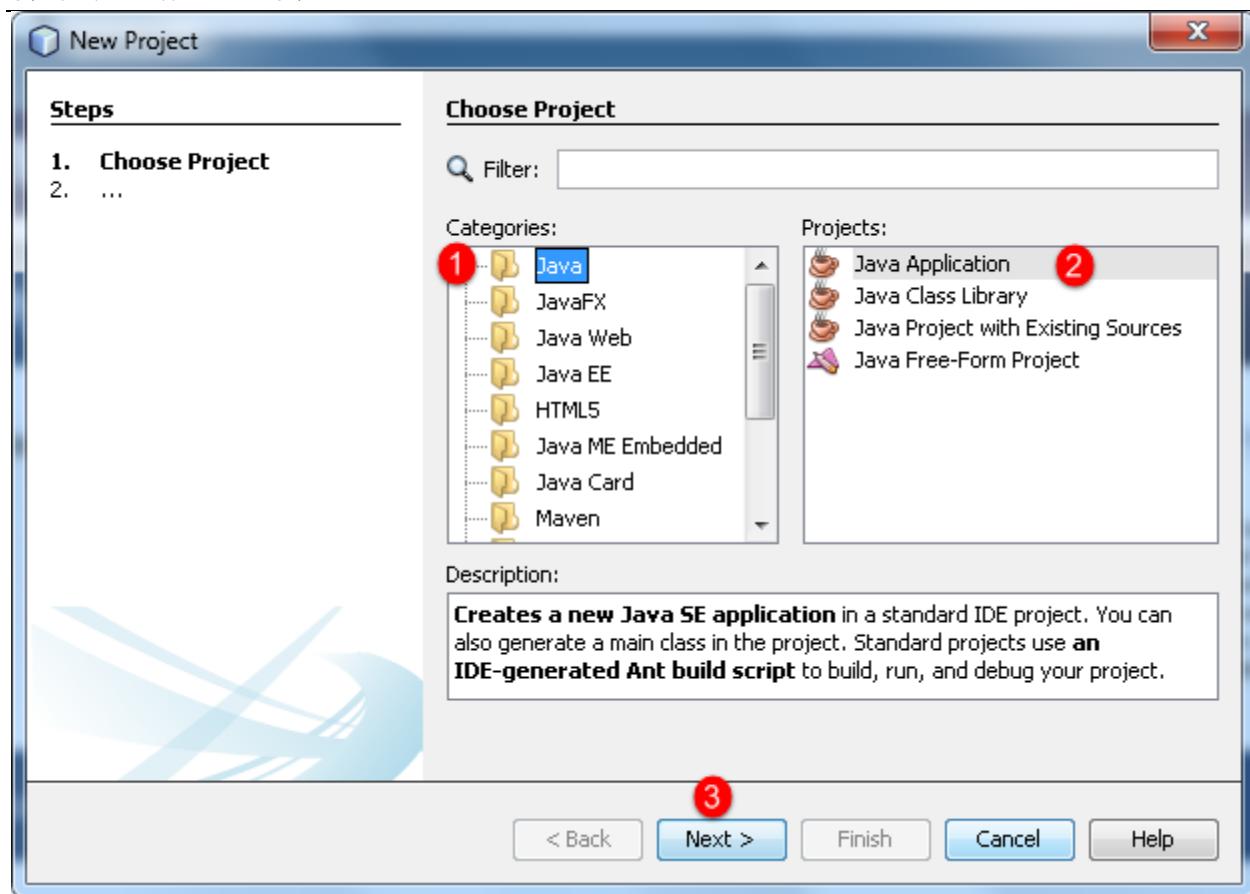
با استفاده از NetBeans

برنامه NetBeans را اجرا کنید. از مسیری که در شکل زیر نشان داده شده است یک پروژه جدید ایجاد کنید:

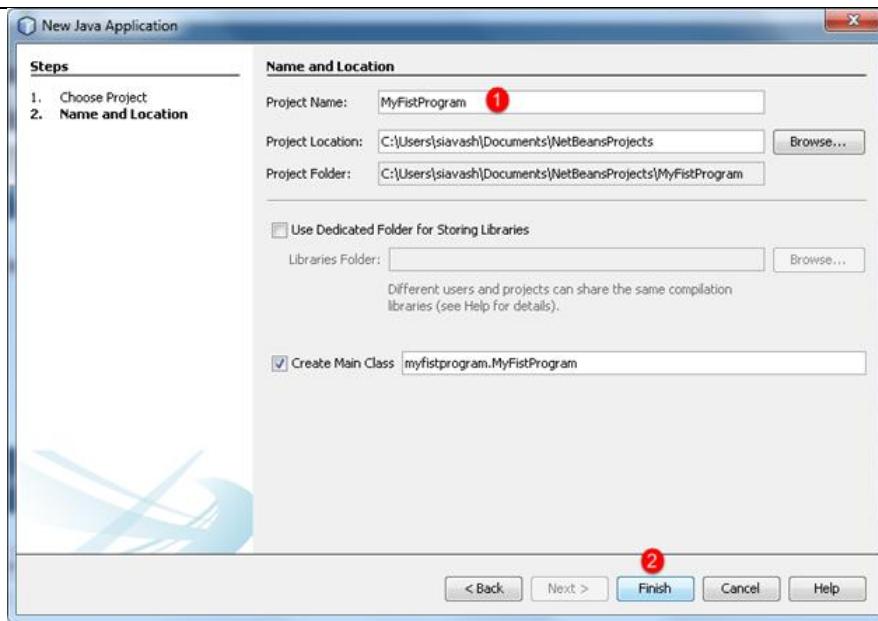


حال با یک صفحه مواجه می‌شویم. طبق شماره‌هایی که در شکل زیر نمایش داده شده‌اند گزینه‌ها را انتخاب کرده و به مرحله بعد بروید

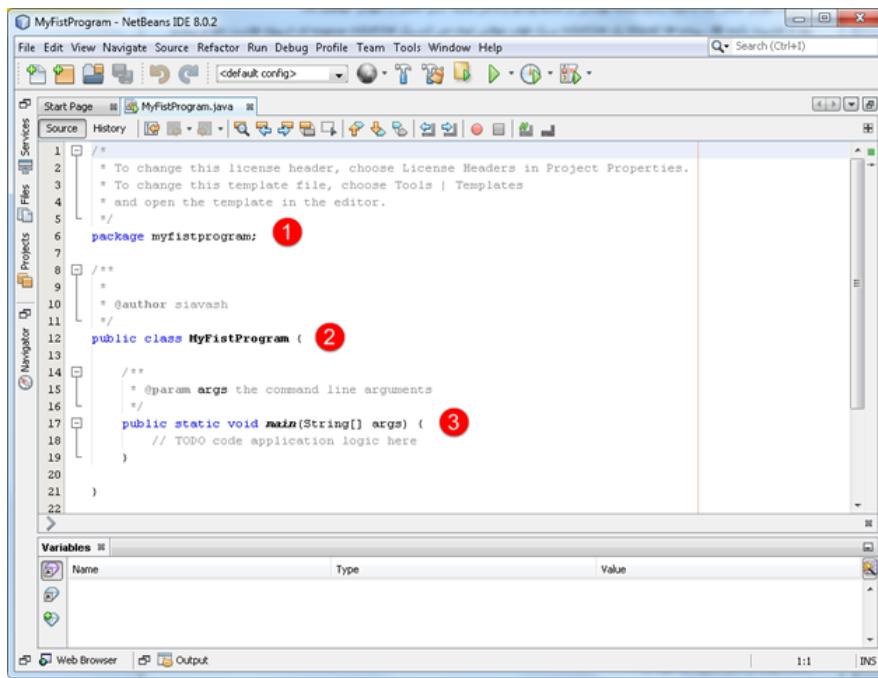
:



با زدن دکمه Next صفحه‌ای به صورت زیر نمایش داده می‌شود. در این پنجره نام پروژه‌تان (MyFirstProgram) را نوشته و سپس بر روی دکمه Finish کلیک کنید :



بعد از فشردن دکمه Finish، وارد محیط کدنویسی برنامه به صورت زیر می‌شویم:



محیط کدنویسی یا IDE جایی است که ما کدها را در آن تایپ می‌کنیم. کدها در محیط کدنویسی به صورت رنگی تایپ می‌شوند در نتیجه تشخیص بخش‌های مختلف کد را راحت می‌کند. همانطور که در شکل بالا مشاهده می‌کنید ما کدهای پیشفرض را به سه قسمت تقسیم

کرده‌ایم. قسمت اول Package، قسمت دوم کلاس و قسمت سوم متدهای main() نگران اصطلاحاتی که به کار بردیم نباشد آنها را در فصول بعد توضیح خواهیم داد. در محل کد نویسی کدهایی از قبل نوشته شده که برای شروع شما آنها را پاک کنید و کدهای زیر را در

محل کدنویسی بنویسید :

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("Welcome to JAVA Tutorials!");
8     }
9 }
```

ساختار یک برنامه در جاوا

مثال بالا ساده‌ترین برنامه‌ای است که شما می‌توانید در جاوا بنویسید. هدف در مثال بالا نمایش یک پیغام در صفحه نمایش است. هر زبان برنامه نویسی دارای قواعدی برای کدنویسی است. اجازه بدهید هر خط کد را در مثال بالا توضیح بدهیم. در خط اول Package تعريف شده است که شامل کدهای نوشته شده توسط شما است و از تداخل نامها جلوگیری می‌کند. درباره Package در درس‌های آینده توضیح خواهیم داد. در خط ۴، آکولاد () نوشته شده است. آکولاد برای تعریف یک بلوک کد به کار می‌رود. جاوا یک زبان ساخت یافته است که شامل کدهای زیاد و ساختارهای فراوانی می‌باشد. هر آکولاد باز () در جاوا باید دارای یک آکولاد بسته () نیز باشد. همه کدهای نوشته شده از خط ۴ تا خط ۹ یک بلوک کد است.

در خط ۴ یک کلاس تعریف شده است. درباره کلاس‌ها در فصل‌های آینده توضیح خواهیم داد. در مثال بالا کدهای شما باید در داخل یک کلاس نوشته شود. بدنه کلاس شامل کدهای نوشته شده از خط ۴ تا ۹ می‌باشد. خط ۵ متدهای main() یا متدهای نامیده می‌شود. هر متدهای شامل یک سری کد است که وقتی اجرا می‌شوند که متدهای صدا بزنیم. درباره متدهای صدا زدن آن در فصول بعدی توضیح خواهیم داد. متدهای main() نقطه آغاز اجرای برنامه است. این بدان معناست که ابتدا تمام کدهای داخل متدهای main() و سپس بقیه کدها اجرا می‌شود. درباره متدهای main() در فصول بعدی توضیح خواهیم داد. متدهای main() و سایر متدهای دارای آکولاد و کدهایی در داخل آنها می‌باشند و وقتی کدها اجرا می‌شوند که متدها را صدا بزنیم. هر خط کد در جاوا به یک سیمیکولن (;) ختم می‌شود. اگر سیمیکولن در آخر خط فراموش شود برنامه با خطأ مواجه می‌شود. مثالی از یک خط کد در جاوا به صورت زیر است :

```
System.out.println("Welcome to JAVA Tutorials!");
```

این خط کد پیغام Welcome to JAVA Tutorials! را در صفحه نمایش نشان می‌دهد. از متدها println() برای چاپ یک رشته

استفاده می‌شود. یک رشته گروهی از کاراکترها است که به وسیله دابل کوتبشون ("") محصور شده است، مانند:

.Tutorials!"

یک کاراکتر می‌تواند یک حرف، عدد، علامت یا باشد. در کل مثال بالا نحوه استفاده از متدها println() نشان داده شده است. این متدها از کلاس PrintStream بوده و از آن برای چاپ مقدیر استفاده می‌شود. out یک فیلد استاتیک کلاس System و کلاس System هم یک کلاس از پیش تعریف شده در جاوا می‌باشد. جاوا فضای خالی و خطوط جدید را نادیده می‌گیرد. بنابراین شما می‌توانید همه برنامه را در یک خط بنویسید. اما اینکار خواندن و اشکال زدایی برنامه را مشکل می‌کند. یکی از خطاهای معمول در برنامه نویسی فراموش کردن سیمیکولن در پایان هر خط کد است. به مثال زیر توجه کنید:

```
System.out.println("Welcome to JAVA Tutorials!");
```

جاوا فضای خالی بالا را نادیده می‌گیرد و از کد بالا اشکال نمی‌گیرد. اما از کد زیر ایراد می‌گیرد:

```
System.out.println();
    "Welcome to JAVA Tutorials!");
```

به سیمیکولن آخر خط اول توجه کنید. برنامه با خطای نحوی مواجه می‌شود چون دو خط کد مربوط به یک برنامه هستند و شما فقط باید یک سیمیکولن در آخر آن قرار دهید. همیشه به یاد داشته باشید که جاوا به بزرگی و کوچکی حروف حساس است. یعنی به طور مثال MAN و man در جاوا با هم فرق دارند. رشته‌ها و توضیحات از این قاعده مستثنی هستند که در درس‌های آینده توضیخ خواهیم داد. مثلاً کدهای زیر با خطای مواجه می‌شوند و اجرا نمی‌شوند:

```
system.out.println("Welcome to JAVA Tutorials!");
SYSTEM.OUT.PRINTLN("Welcome to JAVA Tutorials!");
sYsTem.oUt.pRinTLn("Welcome to JAVA Tutorials!");
```

تفصیل در بزرگی و کوچکی حروف از اجرای کدها جلوگیری می‌کند. اما کد زیر کاملاً بدون خطای است:

```
System.out.println("Welcome to JAVA Tutorials!");
```

همیشه کدهای خود را در داخل آکولاد بنویسید.

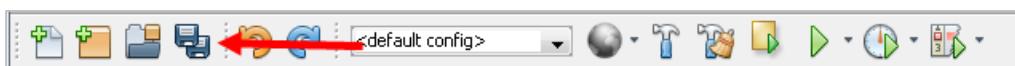
```
{
```

```
statement1;
}
```

این کار باعث می‌شود که کدنویسی شما بهتر به چشم بباید و تشخیص خطاهای راحت‌تر باشد. یکی از ویژگیهای مهم جاوا نشان دادن کدها به صورت تو رفتگی است بدین معنی که کدها را به صورت تو رفتگی از هم تفکیک می‌کند و این در خوانایی برنامه بسیار مؤثر است.

ذخیره پروژه و اجرای برنامه

برای ذخیره پروژه و برنامه می‌توانید به مسیر **File > Save All** Ctrl+Shift+S استفاده کنید. همچنین می‌توانید از قسمت **Toolbar** بر روی شکل زیر کلیک کنید:

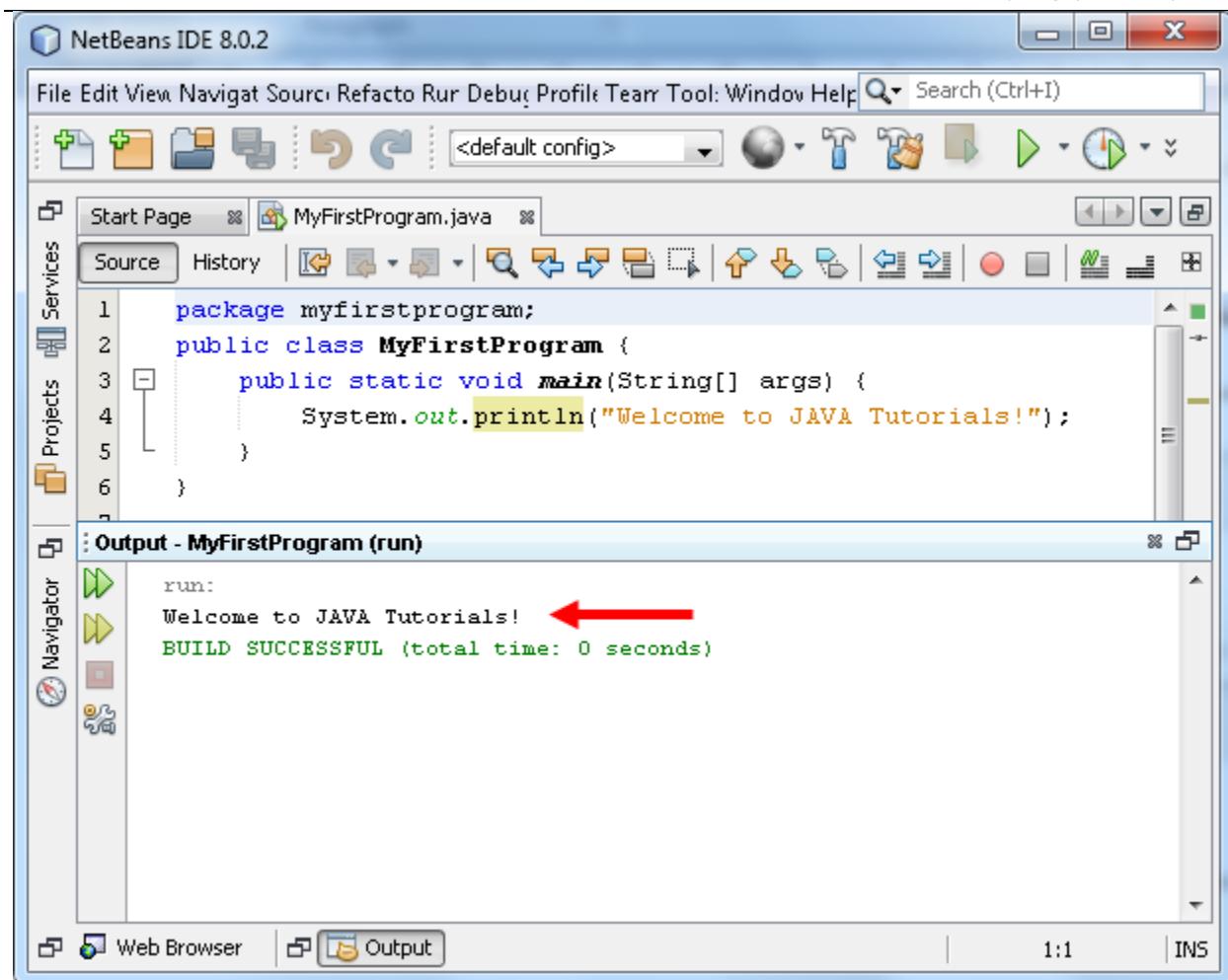


و برای اجرای برنامه هم از فلش سبزرنگ موجود در **Toolbar** و یا دکمه F6 استفاده کنید:



با اجرای برنامه بالا مشاهده می‌کنید که رشتة Welcome to JAVA Tutorials! در خروجی برنامه به صورت زیر نمایش داده می‌شود

:



وجود خط سبز در پایین فرش قرمز در شکل بالا نشان دهنده اجرای بدون نقص برنامه می‌باشد. حال که با خصوصیات و ساختار اولیه جاوا آشنا شدید در درس‌های آینده مطالب بیشتری از این زبان برنامه نویسی قدرتمند خواهید آموخت.

استفاده از Package

برای دسته بندی کلاس‌ها و قرار دادن کلاس‌های مرتقبه با هم در یک مکان، جاوا از مفهومی به نام بسته یا package استفاده می‌کند. پکیج معادل فضای نام در سی‌شارپ می‌باشد. یک دلیل برای گروه بندی کلاس‌ها در package این است که امکان دارد دو برنامه نویس از دو کلاس هم نام استفاده کنند. با این کار از چنین برخوردهایی جلوگیری به عمل می‌آید. یعنی اگر دو کلاس هم نام در دو Package غیر همانم باشند مشکلی به وجود نمی‌آید. همانطور که در مثال بالا دیدید به طور پیشفرض NetBeans هنگام ایجاد برنامه یک Package با اسمی که برای برنامه انتخاب کردہ‌ایم با حروف کوچک و در داخل این Package هم کلاسی به همین اسم ایجاد می‌کند :

```
package myfirstprogram;

public class MyFirstProgram
{
    ...
}
```

برای وارد کردن کلاس یک Package در داخل دیگر از کلمه کلیدی import به صورت زیر استفاده می‌شود :

```
import PackageName.ClassName
```

همانطور که در مثال بالا مشاهده می‌کنید برای استفاده از کلاسی که در یک Package قرار دارد در دیگر ابتدا کلمه

سپس نام Package، بعد علامت نقطه و در آخر نام کلاس را می‌نویسیم. مثلاً برای استفاده از کلاس MyFirstProgram مربوط به پکیج

به صورت زیر عمل می‌شود : myfirstprogram

```
import myfirstprogram.MyFirstProgram;
```

بسته‌ها را می‌توان به صورت تو در تو تعریف کرد. در این حالت در تعریف بسته یک کلاس، از بیرونی ترین بسته شروع کرده و هر بسته را

با نقطه (.) به بسته بعدی متصل می‌کنیم :

```
import firstPackage.secondPackage.ClassName
```

نکته‌ای که بهتر است در همینجا به آن اشاره کنم این است که اگر بخواهید کد زیر را بدون استفاده از NetBeans کامپایل و اجرا کنید،

کامپایل می‌شود ولی در زمان اجرا با خطای Error: Could not find or load main class MyFirstProgram مواجه می‌شود

:

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to JAVA Tutorials!");
    }
}
```

کد بالا را با نام و پسوند java و در درایو D ذخیره کنید. حال cmd را اجرا کرده و سعی کنید کد بالا را کامپایل و اجرا

نمایید :

```
Microsoft Windows [Version 6.1.7601]
```

```
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\YounesJava>d:

D:\>javac MyFirstProgram.java

D:\>java MyFirstProgram
Error: Could not find or load main class MyFirstProgram

D:\>java -cp . myfirstprogram.MyFirstProgram
Welcome to JAVA Tutorials!
```

همانطور که مشاهده می‌کنید برای کامپایل کد بالا به صورت زیر عمل کرده‌ایم :

```
javac MyFirstProgram.java
```

اما برای اجرای کد، اگر از دستور زیر استفاده کنیم با خطأ مواجه می‌شویم :

```
java MyFirstProgram
```

دلیل وجود خطأ، خط اول کد، یعنی package myfirstprogram می‌باشد. برای رفع این مشکل یا باید خط مذکور را از برنامه حذف و برنامه را مجدداً کامپایل و اجرا کرد یا اگر آن را پاک نکنید باید از کد زیر برای اجرای کد استفاده کنید :

```
java -cp . myfirstprogram.MyFirstProgram
```

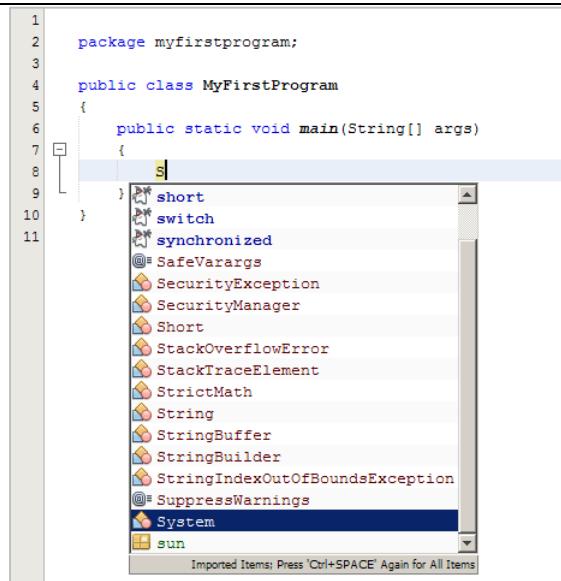
که در این صورت برنامه اجرا و پیغام Welcome to JAVA Tutorials! چاپ می‌شود. در درس‌های آینده توصیه می‌کنیم که اگر می‌خواهید از NotePad و cmd برای اجرای کدها استفاده کنید این خط را از اول هم فایل‌ها حذف و سپس برنامه را کامپایل و اجرا کنید.

استفاده از IntelliSense در NetBeans

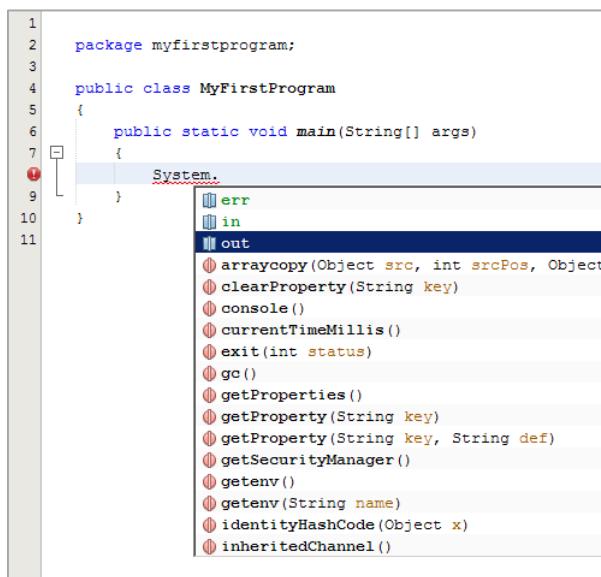
شاید یکی از ویژگیهای مهم NetBeans، اینتلی سنس باشد. ما را قادر می‌سازد که به سرعت به کلاس‌ها و متدها و... دسترسی پیدا کنیم. وقتی که شما در محیط کدنویسی حرفی را تایپ کنید IntelliSense فوراً فعال می‌شود. کد زیر را در داخل متند main() بنویسید.

```
System.out.println("Welcome to JAVA Tutorials!");
```

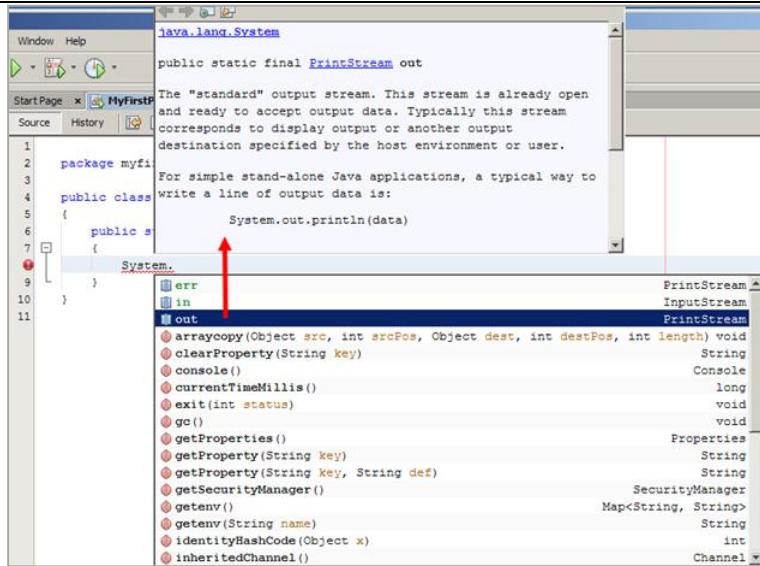
اولین حرف را تایپ کرده و سپس دکمه‌های ترکیبی Ctrl+Space را فشار دهید تا IntelliSense فعال شود:



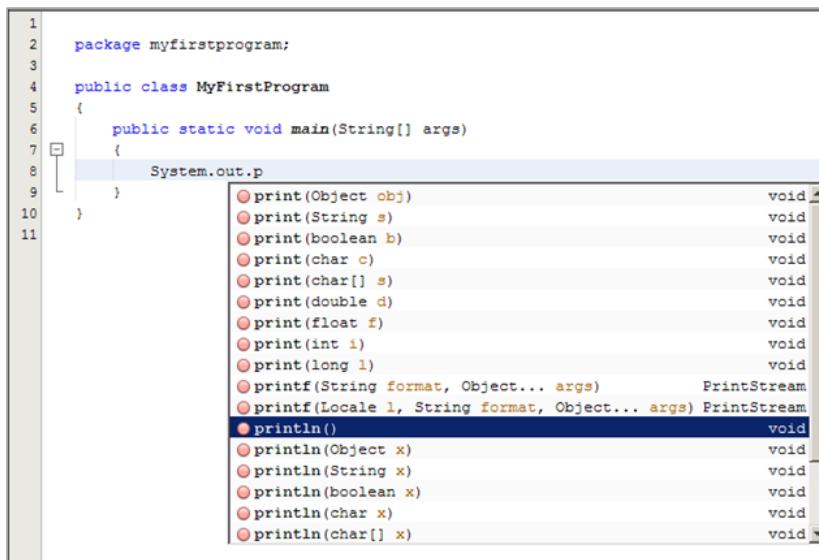
لیستی از کلمات به شما پیشنهاد می‌دهد که بیشترین تشابه را با نوشته شما دارند. شما می‌توانید با زدن دکمه tab گزینه مورد نظرتان را انتخاب کنید. با تایپ نقطه (.) شما با لیست پیشنهادی دیگری مواجه می‌شوید:



اگر بر روی گزینه‌ای که می‌خواهید انتخاب کنید لحظه‌ای مکث کنید توضیحی در رابطه با آن مشاهده خواهید کرد مانند شکل زیر:



هر چه که به پایان کد نزدیک می‌شوید لیست بیشنهادی محدودتر می‌شود. برای مثال با تایپ p، اینتل لایسنس فقط کلماتی را که دارای حرف p هستند را نمایش می‌دهد:



با تایپ حرف‌های بیشتر لیست محدودتر می‌شود. اگر IntelliSense نتواند چیزی را که شما تایپ کردید پیدا کند هیچ چیزی را نمایش نمی‌دهد. برای ظاهر کردن IntelliSense کافیست دکمه ترکیبی Ctrl+Space را فشار دهید. برای انتخاب یکی از متد‌هایی که دارای چند حالت هستند، می‌توان با استفاده از دکمه‌های مکان نما (بالا و پایین) یکی از حالت‌ها را انتخاب کرد. مثلاً متد () println() همانطور که در شکل زیر مشاهده می‌کنید دارای چندین حالت نمایش پیغام در صفحه است:

همانطور که در شکل زیر مشاهده می‌کنید دارای چندین حالت نمایش پیغام در صفحه است:

The screenshot shows a Java code editor with the following code:

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         System.out.p
8     }
9 }
10
11

```

A tooltip from the IDE's Intellisense feature is displayed, listing various methods available for the `System.out` object. The methods listed are:

- print(Object obj)
- print(String s)
- print(boolean b)
- print(char c)
- print(char[] s)
- print(double d)
- print(float f)
- print(int i)
- print(long l)
- printf(String format, Object... args)
- printf(Locale l, String format, Object... args)
- println()
- println(Object x)
- println(String x)
- println(boolean x)
- println(char x)
- println(char[] x)

The method `println()` is highlighted in the tooltip.

به طور هوشمند کدهایی را به شما پیشنهاد می‌دهد و در نتیجه زمان نوشتن کد را کاهش می‌دهد.

رفع خطاها

بیشتر اوقات هنگام برنامه نویسی با خطا مواجه می‌شویم. تقریباً همه برنامه‌هایی که امروزه می‌بینید حداقل از داشتن یک خطا رنج می‌برند. خطاها می‌توانند برنامه شما را با مشکل مواجه کنند. در جاوا سه نوع خطا وجود دارد :

خطای کمپایلری

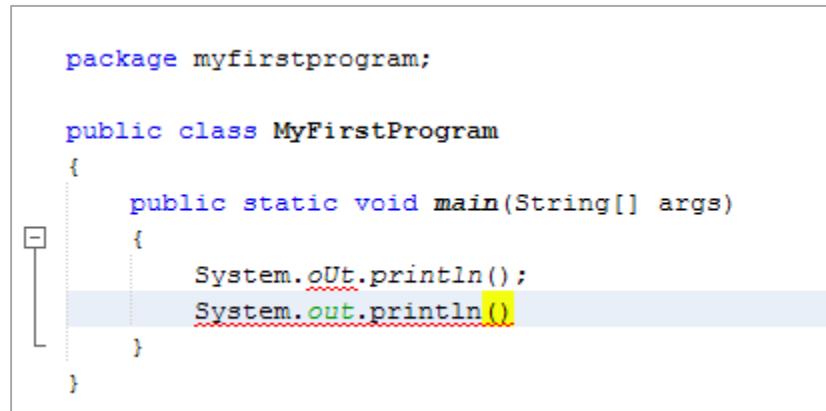
این نوع خطا از اجرای برنامه شما جلوگیری می‌کند. این خطاها شامل خطای دستور زبان می‌باشد. این بدین معنی است که شما قواعد کد نویسی را رعایت نکرده‌اید. یکی دیگر از موارد وقوع این خطا هنگامی است که شما از چیزی استفاده می‌کنید که نه وجود دارد و نه ساخته شده است. حذف فایل‌ها یا اطلاعات ناقص در مورد پروژه ممکن است باعث به وجود آمدن خطای کمپایلری شود. استفاده از برنامه بوسیله برنامه دیگر نیز ممکن است باعث جلوگیری از اجرای برنامه و ایجاد خطای کمپایلری شود.

خطاهای منطقی

این نوع خطا در اثر تغییر در یک منطق موجود در برنامه به وجود می‌آید. رفع این نوع خطاها بسیار سخت است چون شما برای یافتن آن‌ها باید کد را تست کنید. نمونه‌ای از یک خطای منطقی برنامه‌ای است که دو عدد را جمع می‌کند ولی حاصل تفریق دو عدد را نشان می‌دهد. در این حالت ممکن است برنامه نویس علامت ریاضی را اشتباه تایپ کرده باشد.

استثناء

این نوع خطاهای هنگامی رخ می‌دهند که برنامه در حال اجراست. این خطا هنگامی روی می‌دهد که کاربر یک ورودی نامعتبر به برنامه بدهد و برنامه نتواند آن را پردازش کند. NetBeans دارای ابزارهایی برای پیدا کردن و برطرف کردن خطاهای هستند. وقتی در محیط کدنویسی در حال تایپ کد هستیم یکی از ویژگیهای NetBeans تشخیص خطاهای ممکن قبل از اجرای برنامه است. زیر کدهایی که دارای خطای کمپایلری هستند خط قرمز کشیده می‌شود.



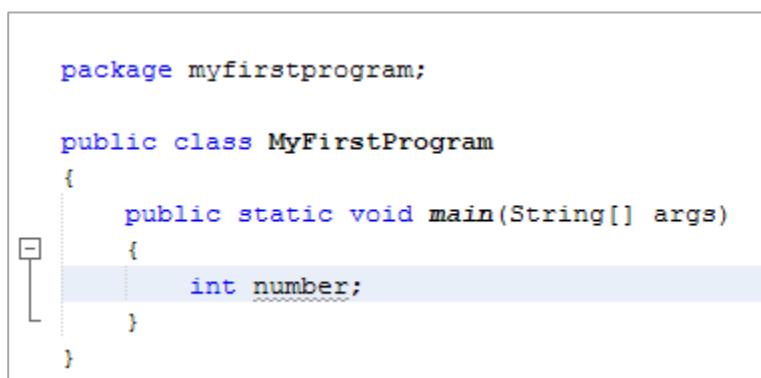
```

package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.oUt.println();
        System.out.println()
    }
}

```

هنگامی که شما با ماوس روی این خطوط توقف کنید توضیحات خطا را مشاهده می‌کنید. شما ممکن است با خط سبز هم مواجه شوید که نشان دهنده اخطار در کد است ولی به شما اجازه اجرای برنامه را می‌دهند. به عنوان مثال ممکن است شما یک متغیر را تعریف کنید ولی در طول برنامه از آن استفاده نکنید. (در درس‌های آینده توضیح خواهیم داد).



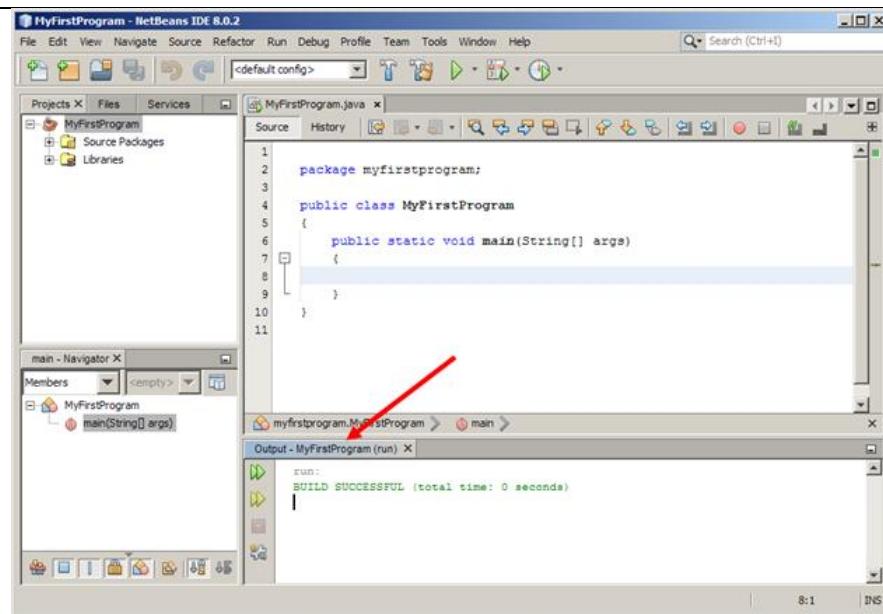
```

package myfirstprogram;

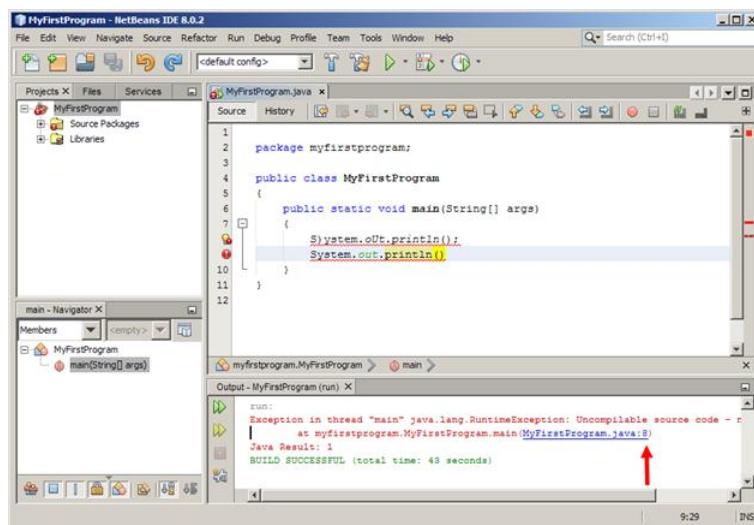
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int number;
    }
}

```

در باره رفع خطاهای در آینده توضیح بیشتری می‌دهیم. پنجه Output که در شکل زیر با فلش قرمز نشان داده شده است به شما امکان مشاهده خطاهای، هشدارها و رفع آن‌ها می‌دهد.

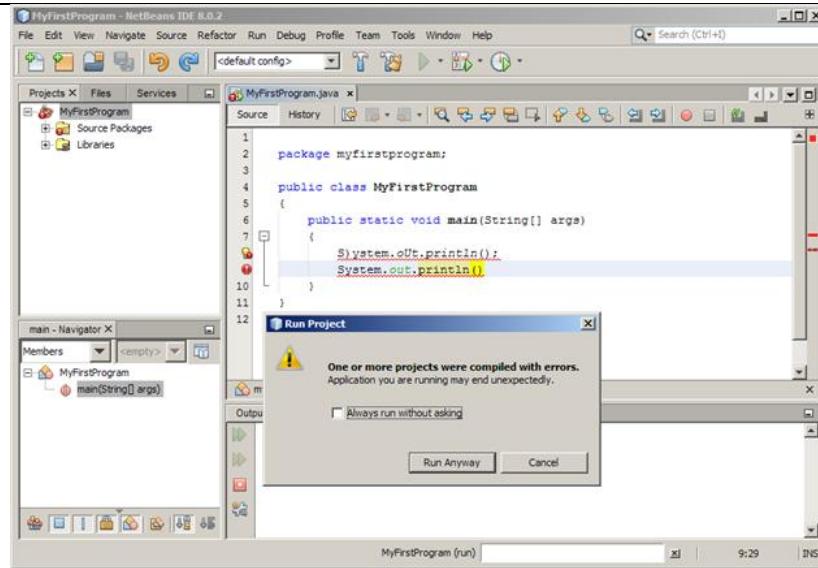


همانطور که در شکل زیر مشاهده می‌کنید هرگاه برنامه شما با خطأ مواجه شود لیست خطاها در پنجره Output نمایش داده می‌شود.



در شکل بالا علت به وجود آمدن خطأ و شماره خطی که خطأ در آن رخ داده است، نمایش داده شده است. اگر برنامه شما دارای خطأ باشد

و آن را اجرا کنید با پنجره زیر روبرو می‌شوید :



مربع کوچک داخل پنجره بالا را تیک زنید چون دفعات بعد که برنامه شما با خطأ مواجه شود دیگر این پنجره به عنوان هشدار نشان داده نخواهد شد. با کلیک بر روی دکمه Run Anyway برنامه با وجود خطأ نیز اجرا می‌شود. اما با کلیک بر روی دکمه Cancel اجرای برنامه متوقف می‌شود و شما باید خطاهای موجود در پنجره Output را بر طرف نمایید.

کاراکترهای کنترلی

کاراکترهای کنترلی کاراکترهای ترکیبی هستند که با یک بک اسلش (\) شروع می‌شوند و به دنبال آن‌ها یک حرف یا عدد می‌آید و یک رشته را با فرمت خاص نمایش می‌دهند. برای مثال برای ایجاد یک خط جدید و قرار دادن رشته در آن می‌توان از کاراکتر کنترلی \n استفاده کرد:

```
System.out.println("Hello\nWorld!");
```

```
Hello  
World
```

مشاهده کردید که کمپایلر بعد از مواجهه با کاراکتر کنترلی \n نشانگر ماوس را به خط بعد برد و بقیه رشته را در خط بعد نمایش می‌دهد. متدها Println() هم مانند کاراکتر کنترلی \n یک خط جدید ایجاد می‌کند، البته بدین صورت که در انتهای رشته یک کاراکتر کنترلی \n اضافه می‌کند:

```
System.out.println("Hello World!");
```

کد بالا و کد زیر هیچ فرقی با هم ندارند :

```
System.out.print("Hello World!\n");
```

متد `Print()` کارکردی شبیه به `Println()` دارد با این تفاوت که نشان گر ماوس را در همان خط نگه می‌دارد و خط جدید ایجاد

نمی‌کند. جدول زیر لیست کاراکترهای کنترلی و کارکرد آنها را نشان می‌دهد :

عملکرد	کاراکتر کنترلی	عملکرد	کاراکتر کنترلی
چاپ کوتیشن	\'	Form Feed	\f
چاپ دابل کوتیشن	\\"	خط جدید	\n
چاپ بک اسلش	\	سر سطر رفتن	\r
حرکت به عقب	\b	حرکت به صورت افقی	\t

ما برای استفاده از کاراکترهای کنترلی از بک اسلش (\) استفاده می‌کنیم. از آنجاییکه \ معنای خاصی به رشته‌ها می‌دهد برای چاپ بک اسلش () باید از \|() استفاده کنیم :

```
System.out.println("We can print a \\ by using the \\\\ escape sequence.");
```

```
We can print a \ by using the \\ escape sequence.
```

یکی از موارد استفاده از \|، نشان دادن مسیر یک فایل در ویندوز است :

```
System.out.println("C:\\\\Program Files\\\\Some Directory\\\\SomeFile.txt");
```

```
C:\Program Files\Some Directory\SomeFile.txt
```

از آنجاییکه از دابل کوتیشن ("") برای نشان دادن رشته‌ها استفاده می‌کنیم برای چاپ آن از " " استفاده می‌کنیم :

```
System.out.println("I said, \"Motivate yourself!\".");
```

```
I said, "Motivate yourself!".
```

همچنین برای چاپ کوتیشن (‘) از \ استفاده می‌کنیم :

```
System.out.println("The programmer\'s heaven.");
```

```
The programmer's heaven.
```

برای ایجاد فاصله بین حروف یا کلمات از \t استفاده می‌شود :

```
System.out.println("Left\tRight");
```

```
Left Right
```

هر تعداد کاراکتر که بعد از کاراکتر کنترلی \t بیایند به اول سطر منتقل و جایگزین کاراکترهای موجود می‌شوند :

```
System.out.println("Mitten\rK");
```

```
K
```

مثلاً در مثال بالا کاراکتر K بعد از کاراکتر کنترلی \r آمده است. کاراکتر کنترلی حرف K را به ابتدای سطر برد و جایگزین Mitten می‌کند.

برای مشاهده لیست مقادیر مبنای ۱۶ برای کاراکترهای یونیکد به لینک زیر مراجعه نمایید:

```
http://www.ascii.cl/htmlcodes.htm
```

اگر کمپایلر به یک کاراکتر کنترلی غیر مجاز بخورد کند، برنامه پیغام خطای زمانی اتفاق می‌افتد که برنامه نویس برای چاپ اسلش (\) از \\ استفاده می‌کند.

توضیحات

وقتی که کدی تایپ می‌کنید شاید بخواهید که متنی جهت یادآوری وظیفه آن کد به آن اضافه کنید. در جاوا (و بیشتر زبانهای برنامه نویسی) می‌توان این کار را با استفاده از توضیحات انجام داد. توضیحات متونی هستند که توسط کمپایلر نادیده گرفته می‌شوند و به عنوان بخشی از کد محسوب نمی‌شوند.

هدف اصلی از ایجاد توضیحات، بالا بردن خوانایی و تشخیص نقش کدهای نوشته شده توسط شما، برای دیگران است. فرض کنید که می‌خواهید در مورد یک کد خاص، توضیح بدهید، می‌توانید توضیحات را در بالای کد یا کنار آن بنویسید. از توضیحات برای مستند سازی برنامه هم استفاده می‌شود. در برنامه زیر نقش توضیحات نشان داده شده است :

```

1 package myfirstprogram;
2 {
3     public class MyFirstProgram
4     {
5         public static void main(String[] args)
6         {
7             // This line will print the message hello world
8             System.out.println("Hello World!");
9         }
10    }
11 }
```

در کد بالا، خط ۷ یک توضیح درباره خط ۸ است که به کاربر اعلام می‌کند که وظیفه خط ۸ چیست؟ با اجرای کد بالا فقط جمله Hello World چاپ شده و خط ۷ در خروجی نمایش داده نمی‌شود چون کمپایلر توضیحات را نادیده می‌گیرد. توضیحات بر سه نوعند :

توضیحات تک خطی

```
// single line comment
```

توضیحات چند خطی

```
/* multi
line
comment */
```

توضیحات مستند سازی

```
/** 
Documentation Comments
*/
```

توضیحات تک خطی همانگونه که از نامش پیداست، برای توضیحاتی در حد یک خط به کار می‌رond. این توضیحات با علامت // شروع می‌شوند و هر نوشهایی که در سمت راست آن قرار بگیرد جز توضیحات به حساب می‌آید. این نوع توضیحات معمولاً در بالا یا کنار کد قرار می‌گیرند. اگر توضیح درباره یک کد به بیش از یک خط نیاز باشد از توضیحات چند خطی استفاده می‌شود. توضیحات چند خطی با /* شروع و با */ پایان می‌یابند. هر نوشهایی که بین این دو علامت قرار بگیرد جز توضیحات محسوب می‌شود. نوع دیگری از توضیحات،

توضیحات مستند سازی نامیده می‌شوند. این نوع با `**/ شروع و به */ ختم می‌شوند. از این نوع برای مستند سازی برنامه استفاده می‌شود و در درس‌های آینده در مورد آنها توضیح خواهیم داد.`

متغیر

متغیر مکانی از حافظه است که شما می‌توانید مقادیری را در آن ذخیره کنید. می‌توان آن را به عنوان یک ظرف تصور کرد که داده‌های خود را در آن قرار داده‌اید. محتویات این ظرف می‌تواند پاک شود یا تغییر کند. هر متغیر دارای یک نام نیز هست. که از طریق آن می‌توان متغیر را از دیگر متغیرها تشخیص داد و به مقدار آن دسترسی پیدا کرد. همچنین دارای یک مقدار می‌باشد که می‌تواند توسط کاربر انتخاب شده باشد یا نتیجه یک محاسبه باشد. مقدار متغیر می‌تواند تهی نیز باشد. متغیر دارای نوع نیز هست بدین معنی که نوع آن با نوع داده‌ای که در آن ذخیره می‌شود یکی است. متغیر دارای عمر نیز هست که از روی آن می‌توان تشخیص داد که متغیر باید چقدر در طول برنامه مورد استفاده قرار گیرد. و در نهایت متغیر دارای محدوده استفاده نیز هست که به شما می‌گوید که متغیر در چه جای برنامه برای شما قابل دسترسی است. ما از متغیرها به عنوان یک انبار موقتی برای ذخیره داده استفاده می‌کنیم. هنگامی که یک برنامه ایجاد می‌کنیم احتیاج به یک مکان برای ذخیره داده، مقادیر یا داده‌هایی که توسط کاربر وارد می‌شوند داریم. این مکان همان متغیر است. برای این از کلمه متغیر استفاده می‌شود چون ما می‌توانیم بسته به نوع شرایط هر جا که لازم باشد مقدار آن را تغییر دهیم. متغیرها موقتی هستند و فقط موقعي مورد استفاده قرار می‌گیرند که برنامه در حال اجراست و وقتی شما برنامه را می‌بندید محتویات متغیرها نیز پاک می‌شود. قبلًا ذکر شد که به وسیله نام متغیر می‌توان به آن دسترسی پیدا کرد. برای نامگذاری متغیرها باید قوانین زیر را رعایت کرد :

- نام متغیر باید با یک از حروف الفبا (a-z or A-Z) شروع شود.
- نمی‌تواند شامل کاراکترهای غیرمجاز مانند . \$ ^ ? # باشد.
- نمی‌توان از کلمات رزرو شده در جاوا برای نام متغیر استفاده کرد.
- نام متغیر نباید دارای فضای خالی (spaces) باشد.
- اسامی متغیرها نسبت به بزرگی و کوچکی حروف حساس هستند. در جاوا دو حرف مانند a و A دو کاراکتر مختلف به حساب می‌آینند.

دو متغیر با نامهای `myNumber` و `MyNumber` دو متغیر مختلف محاسبه می‌شوند چون یکی از آن‌ها با حرف کوچک `m` و دیگری با حرف بزرگ `M` شروع می‌شود. شما نمی‌توانید دو متغیر را که دقیق شبهه هم هستند را در یک `scope` (محدوده) تعریف کنید. `Scope` به معنای

یک بلوک کد است که متغیر در آن قابل دسترسی و استفاده است. در مورد Scope در فصل‌های آینده بیشتر توضیح خواهیم داد. متغیر دارای نوع هست که نوع داده‌ای را که در خود ذخیره می‌کند را نشان می‌دهد. معمول‌ترین انواع داده int، long، short، byte، double، char، float می‌باشند. برای مثال شما برای قرار دادن یک عدد صحیح در متغیر باید از نوع int استفاده کنید.

أنواع ساده

أنواع ساده انواعی از داده‌ها هستند که شامل اعداد، کاراکترها و مقادیر بولی می‌باشند. به انواع ساده انواع اصلی نیز گفته می‌شود چون از آن‌ها برای ساخت انواع پیچیده‌تری مانند کلاس‌ها و ساختارها استفاده می‌شود. انواع ساده دارای مجموعه مشخصی از مقادیر هستند و محدوده خاصی از اعداد را در خود ذخیره می‌کنند. در جدول زیر انواع ساده و محدوده آن‌ها آمده است :

نوع	دامنه
byte	اعداد صحیح بین ۱۲۷ تا -۱۲۸
short	اعداد صحیح بین ۳۲۷۶۸ تا -۳۲۷۶۷
int	اعداد صحیح بین ۲۱۴۷۴۸۳۶۴۸ تا -۲۱۴۷۴۸۳۶۴۷
long	اعداد صحیح بین ۹۲۲۳۳۷۷۰۳۶۸۵۴۷۷۸۰۷ تا ۹۲۲۳۳۷۷۰۳۶۸۵۴۷۷۸۰۸

جدول زیر انواعی که مقادیر با ممیز اعشار را می‌توانند در خود ذخیره کنند را نشان می‌دهد :

نوع	دامنه تقریبی	دقت
float	۳/۴۰۲۸۲۳۳۸ تا -۳/۴۰۲۸۲۳۳۸	۷ رقم
double	۱۷۹۷۶۹۳۱۳۴۸۶۲۳۳۲۳۰۸ تا ۱۷۹۷۶۹۳۱۳۴۸۶۲۳۳۲۳۰۸	۱۵-۱۶ رقم

برای به خاطر سپردن آن‌ها باید از نماد علمی استفاده شود. نوع دیگری از انواع ساده برای ذخیره داده‌های غیر عددی به کار می‌روند و در جدول زیر نمایش داده شده‌اند :

نوع	مقادیر مجاز
char	کاراکترهای یونیکد
boolean	false یا true

نوع `char` برای ذخیره کاراکترهای یونیکد استفاده می‌شود. کاراکترها باید داخل یک کوتیشن ساده قرار بگیرند مانند (`'a'`). نوع `boolean` می‌تواند مقادیر درست (`true`) یا نادرست (`false`) را در خود ذخیره کند و بیشتر در برنامه‌هایی که دارای ساختار تصمیم گیری هستند مورد استفاده قرار می‌گیرد.

استفاده از رشته‌ها

از رشته برای ذخیره گروهی از کاراکترها مانند یک پیغام استفاده می‌شود. مقادیر ذخیره شده در یک رشته باید داخل دابل کوتیشن قرار گیرند تا توسط کمپایلر به عنوان یک رشته در نظر گرفته شوند، مانند (`"massage"`). جاوا دارای نوعی به نام رشته نیست، بلکه رشته‌ها اشیایی هستند که از روی کلاس `String` (حروف S به صورت بزرگ نوشته می‌شود) ساخته می‌شوند. با مفاهیم شیء و کلاس در درس‌های آینده آشنا می‌شوید. فقط در همین حد کافی است که بدانید که از رشته‌ها برای نمایش متن استفاده می‌شود. مثلاً برای نمایش متن `Hello World` می‌توان به صورت زیر عمل کرد :

```
String str ="Hello World";
```

دلیل اینکه در این قسمت درباره رشته‌ها مختصری توضیح دادیم این است که ممکن است در آموزش‌های بعدی با آن‌ها سرو کار داشته باشیم. در آینده به طور مفصل در مورد رشته‌ها توضیح می‌دهیم.

استفاده از متغیرها

در مثال زیر نحوه تعریف و مقدار دهنده متغیرها نمایش داده شده است :

```
1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {
```

```

9      //Declare variables
10     int      num1;
11     int      num2;
12     double   num3;
13     double   num4;
14     boolean  boolVal;
15     char     myChar;
16
17     //Assign values to variables
18     num1    = 1;
19     num2    = 2;
20     num3    = 3.54;
21     num4    = 4.12;
22     boolVal = true;
23     myChar  = 'R';
24
25     //Show the values of the variables
26     System.out.println(MessageFormat.format("num1 = {0}",      num1));
27     System.out.println(MessageFormat.format("num3 = {0}",      num3));
28     System.out.println(MessageFormat.format("num4 = {0}",      num4));
29     System.out.println(MessageFormat.format("boolVal = {0}",   boolVal));
30     System.out.println(MessageFormat.format("num2 = {0}",      num2));
31     System.out.println(MessageFormat.format("myChar = {0}",    myChar));
32   }
33 }
```

```

num1 = 1
num2 = 2
num3 = 3.54
num4 = 4.12
boolVal = true
myChar = R
```

تعريف متغير

در خطوط ۱۱-۱۶ متغیرهایی با نوع و نام متفاوت تعريف شده‌اند. ابتدا باید نوع داده‌هایی را که این متغیرها قرار است در خود ذخیره کنند را مشخص کنیم و سپس یک نام برای آن‌ها در نظر بگیریم و در آخر سیمیکولون بگذاریم. همیشه به یاد داشته باشید که قبل از مقدار دهی و استفاده از متغیر باید آن را تعريف کرد.

```

int      num1;
int      num2;
double   num3;
double   num4;
boolean  boolVal;
char     myChar;
```

نحوه تعريف متغیر به صورت زیر است :

```
data_type identifier;
```

همان نوع داده است مانند `int`, `double` و Identifier نیز نام متغیر است که به ما امکان استفاده و دسترسی به

مقدار متغیر را می‌دهد. برای تعریف چند متغیر از یک نوع می‌توان به صورت زیر عمل کرد :

```
data_type identifier1, identifier2, ... identifierN;
```

مثال

```
int num1, num2, num3, num4, num5;
```

در مثال بالا ۵ متغیر از نوع صحیح تعریف شده است. توجه داشته باشید که بین متغیرها باید علامت کاما (,) باشد.

نامگذاری متغیرها

- نام متغیر باید با یک حرف یا زیرخط و به دنبال آن حرف یا عدد شروع شود.
- نمی‌توان از کاراکترهای خاص مانند #, %, & یا عدد برای شروع نام متغیر استفاده کرد مانند numbers2.
- نام متغیر نباید دارای فاصله باشد. برای نام‌های چند حرفی می‌توان به جای فاصله از علامت زیرخط یا ... استفاده کرد.

نامهای مجاز :

```
num1 myNumber studentCount total      first_name    _minimum
num2 myChar   average   amountDue  last_name    _maximum
name  counter   sum       isLeapYear color_of_car _age
```

نامهای غیر مجاز :

```
123      #numbers#  #ofstudents  1abc2
123abc   $money     first name   ty.np
my number this&that  last name   1:00
```

اگر به نامهای مجاز در مثال بالا توجه کنید متوجه قراردادهای به کار رفته در نامگذاری آن‌ها خواهید شد. یکی از روش‌های نامگذاری، نامگذاری کوهان شتری است. در این روش که برای متغیرهای دو کلمه‌ای به کار می‌رود، اولین حرف اولین کلمه با حرف کوچک و اولین حرف دومین کلمه با حرف بزرگ نمایش داده می‌شود مانند : myNumber. توجه کنید که اولین حرف کلمه Number با حرف بزرگ شروع شده است. مثال دیگر کلمه numberOfStudents است. اگر توجه کنید بعد از اولین کلمه حرف اول سایر کلمات با حروف بزرگ نمایش داده شده است.

محدوده متغیر

در کد ابتدایی درس، متغیرها در داخل متند (`main`) تعریف شده‌اند. در نتیجه، این متغیرها فقط در داخل متند (`main`) قابل دسترسی و استفاده هستند. محدوده یک متغیر مشخص می‌کند که متغیر در کجا کد قابل دسترسی است. هنگامیکه برنامه به پایان متند (`main`) می‌رسد متغیرها از محدوده خارج و بدون استفاده می‌شوند تا زمانی که برنامه در حال اجراست. محدوده متغیرها انواعی دارد که در درس‌های بعدی با آن‌ها آشنا می‌شوید. تشخیص محدوده متغیر بسیار مهم است چون به وسیله آن می‌فهمید که در کجا کد می‌توان از متغیر استفاده کرد. باید یاد آور شد که دو متغیر در یک محدوده نمی‌توانند دارای نام یکسان باشند. مثلاً کد زیر در برنامه ایجاد خطای می‌کند :

```
int num1;
int num1;
```

از آنجاییکه جاوا به بزرگی و کوچک بودن حروف حساس است می‌توان از این خاصیت برای تعریف چند متغیر هم نام ولی با حروف متفاوت (از لحاظ بزرگی و کوچکی) برای تعریف چند متغیر از یک نوع استفاده کرد مانند :

```
int num1;
int Num1;
int NUM1;
```

مقداردهی متغیرها

می‌توان فوراً بعد از تعریف متغیرها مقادیری را به آن‌ها اختصاص داد. این عمل را مقداردهی می‌نامند. در زیر نحوه مقداردهی متغیرها نشان داده شده است :

```
data_type identifier = value;
```

به عنوان مثال :

```
int myNumber = 7;
```

همچنین می‌توان چندین متغیر را فقط با گذاشتن کاما بین آن‌ها به سادگی مقداردهی کرد :

```
data_type variable1 = value1, variable2 = value2, ... variableN, valueN;
int num1 = 1, num2 = 2, num3 = 3;
```

تعریف متغیر با مقدار دهی متغیرها متفاوت است. تعریف متغیر یعنی انتخاب نوع و نام برای متغیر ولی مقدار دهی یعنی اختصاص یک مقدار به متغیر.

اختصاص مقدار به متغیر

در زیر نحوه اختصاص مقادیر به متغیرها نشان داده شده است:

```
num1 = 1;
num2 = 2;
num3 = 3.54;
num4 = 4.12;
boolVal = true;
myChar = 'R';
```

به این نکته توجه کنید که شما به متغیری که هنوز تعریف نشده نمی‌توانید مقدار بدهید. شما فقط می‌توانید از متغیرهایی استفاده کنید که هم تعریف و هم مقدار دهی شده باشند. مثلاً متغیرهای بالا همه قابل استفاده هستند. در این مثال num1 و num2 هر دو تعریف شده‌اند و مقادیری از نوع صحیح به آن‌ها اختصاص داده شده است. اگر نوع داده با نوع متغیر یکی نباشد برنامه پیغام خطا می‌دهد.

جانگهدار (Placeholders)

به متدهای format() از کلاس MessageFormat در خطوط (۳۲-۲۷) توجه کنید. برای استفاده از متدهای format() و کلاس MessageFormat در خطوط (خط ۳) مربوط به آن‌ها را در برنامه وارد کنید:

```
import java.text.MessageFormat;
```

این متدهای آرگومان قبول می‌کنند. آرگومان‌ها اطلاعاتی هستند که متدهای استفاده از آن‌ها کاری انجام می‌دهند. آرگومان‌ها به وسیله کاما از هم جدا می‌شوند. آرگومان اول یک رشته قالب بندی شده است و آرگومان دوم مقداری است که توسط رشته قالب بندی شده مورد استفاده قرار می‌گیرد. اگر به وقت نگاه کنید رشته قالب بندی شده دارای عدد صفری است که در داخل دو آکولاد محصور شده است. البته عدد داخل دو آکولاد می‌تواند از صفر تا n باشد. به این اعداد جانگهدار می‌گویند. این اعداد بوسیله مقدار آرگومان بعد جایگزین می‌شوند. به عنوان مثال جانگهدار {0} به این معناست که اولین آرگومان (مقدار) بعد از رشته قالب بندی شده در آن قرار می‌گیرد. متدهای format() عملای می‌توانند هر تعداد آرگومان قبول کند اولین آرگومان همان رشته قالب بندی شده است که جانگهدار در آن قرار دارد و دومین آرگومان مقداری است که جایگزین جانگهدار می‌شود. در مثال زیر از ۴ جانگهدار استفاده شده است:

```
System.out.println(MessageFormat.format("The values are {0}, {1}, {2}, and {3}.", value1,
value2, value3, value4);
```

```
System.out.println(MessageFormat.format("The values are {0}, {1}, {2}, and {3}.", value1, value2, value3, value4));
```

جانگهدارها از صفر شروع می‌شوند. تعداد جانگهدارها باید با تعداد آرگومان‌های بعد از رشته قالب بندی شده برابر باشد. برای مثال اگر شما چهار جانگهدار مثل بالا داشته باشید باید چهار مقدار هم برای آن‌ها بعد از رشته قالب بندی شده در نظر بگیرید. اولین جا نگهدار با دومین آرگومان و دومین جا نگهدار با سومین آرگومان جایگزین می‌شود. در ابتدا فهمیدن این مفهوم برای کسانی که تازه برنامه نویسی را شروع کرده‌اند سخت است. اما در درسن‌های آینده مثال‌های زیادی در این مورد مشاهده خواهید کرد.

ثابت

ثابت‌ها انواعی از متغیرها هستند که مقدار آن‌ها در طول برنامه تغییر نمی‌کند. ثابت‌ها حتماً باید مقدار دهی اولیه شوند و اگر مقدار دهی آن‌ها فراموش شود در برنامه خطأ به وجود می‌آید. بعد از این که به ثابت‌ها مقدار اولیه اختصاص داده شد هرگز در زمان اجرای برنامه نمی‌توان آن را تغییر داد. برای تعریف ثابت‌ها باید از کلمه کلیدی `final` استفاده کرد. معمولاً نام ثابت‌ها را طبق قرارداد با حروف بزرگ می‌نویسند تا تشخیص آن‌ها در برنامه راحت باشد. نحوه تعریف ثابت در زیر آمده است :

```
final data_type identifier = initial_value;
```

: مثال

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        final int NUMBER = 1;

        NUMBER = 10; //ERROR, Cant modify a constant
    }
}
```

در این مثال می‌بینید که مقدار دادن به یک ثابت، که قبلاً مقدار دهی شده برنامه را با خطأ مواجه می‌کند. نکته‌ی دیگری که نباید فراموش شود این است که، نباید مقدار ثابت را با مقدار دیگر متغیرهای تعریف شده در برنامه برابر قرار داد. مثال :

```
int someVariable;
final int MY_CONST = someVariable;
```

ممکن است این سؤال برایتان پیش آمده باشد که دلیل استفاده از ثابت‌ها چیست؟ اگر مطمئن هستید که مقادیری در برنامه وجود دارند که هرگز در طول برنامه تغییر نمی‌کنند بهتر است که آن‌ها را به صورت ثابت تعریف کنید. این کار هر چند کم، کیفیت برنامه شما را بالا می‌برد.

تبديل ضمنی

تبديل ضمنی یا تبدیل بزرگ کننده یا widening conversion یک نوع تبدیل است که به طور خودکار انجام می‌شود. در این نوع تبدیل در صورتی یک متغیر از یک نوع داده می‌تواند به یک نوع دیگر تبدیل شود که مقدار آن از مقدار داده‌ای که می‌خواهد به آن تبدیل شود کمتر باشد. به عنوان مثال نوع داده‌ای byte می‌تواند مقادیر ۰ تا ۲۵۵ را در خود ذخیره کند و نوع داده‌ای int مقادیر ۰-۲۱۴۷۴۸۳۶۴۸-۲۱۴۷۴۸۳۶۴۷ را شامل می‌شود. پس می‌توانید یک متغیر از نوع byte را به یک نوع int تبدیل کنید :

```
byte number1 = 5;
int number2 = number1;
```

در مثال بالا مقدار number1 برابر ۵ است در نتیجه متغیر number2 که یک متغیر از نوع صحیح است می‌تواند مقدار number1 را در خود ذخیره کند چون نوع صحیح از نوع بايت بزرگ‌تر است. پس متغیر number1 که یک متغیر از نوع بايت است می‌تواند به طور ضمنی به number2 که یک متغیر از نوع صحیح است تبدیل شود. اما عکس مثال بالا صادق نیست.

```
int number1 = 5;
byte number2 = number1;
```

در این مورد ما با خطأ مواجه می‌شویم. اگر چه مقدار ۵ متغیر number1 در محدوده مقادیر byte یعنی اعداد بین ۰-۲۵۵ قرار دارد اما متغیری از نوع بايت حافظه کمتری نسبت به متغیری از نوع صحیح اشغال می‌کند. نوع byte شامل ۸ بیت یا ۸ رقم دو دویی است در حالی که نوع int شامل ۳۲ بیت یا رقم باینری است. یک عدد باینری عددی متشكل از ۰ و ۱ است. برای مثال عدد ۵ در کامپیوتر به عدد باینری ۱۰۱ ترجمه می‌شود. بنابراین وقتی ما عدد ۵ را در یک متغیر از نوع بايت ذخیره می‌کنیم، عددی به صورت زیر نمایش داده می‌شود

```
00000101
```

و وقتی آن را در یک متغیر از نوع صحیح ذخیره می‌کنیم، به صورت زیر نمایش داده می‌شود:

بنابراین قرار دادن یک مقدار `int` در یک متغیر `byte` درست مانند این است که ما سعی کنیم که یک توب فوتیوال را در یک سوراخ کوچک گلف جای دهیم. برای قرار دادن یک مقدار `int` در یک متغیر از نوع `byte` می‌توان از تبدیل صریح استفاده کرد که در درس‌های آینده توضیح داده می‌شود. نکته دیگری که نباید فراموش شود این است که شما نمی‌توانید اعداد با ممیز اعشار را به یک نوع `int` تبدیل کنید چون این کار باعث از بین رفتن بخش اعشاری این اعداد می‌شود.

```
double number1 = 5.25;  
  
int number2 = number1; //Error
```

تیدیلاتی که جاوا به صورت ضمنی می‌تواند انجام دهد در زیر آمده است :

`byte > short > int > long > float > double`

تبدیل صریح

تبديل صريح یا تبدیل کوچک کننده یا Casting نوعی تبدیل است که برنامه را مجبور می‌کند که یک نوع داده را به نوعی دیگر تبدیل کند اگر این نوع تبدیل از طریق تبدیل صمنی انجام نشود. در هنگام استفاده از این تبدیل باید دقیق باشد. چون در این نوع تبدیل ممکن است، مقادیر اصلاح یا حذف شوند. ما می‌توانیم این عملیات را با استفاده از Cast انجام دهیم. فقط نام دیگر تبدیل

صريح است و دستور آن به صورت زیر است :

```
datatypeA variableA = value;  
datatypeB variableB = (datatypeB)variableA;
```

همانطور که قللاً مشاهده کردید نوع `int` را توانستیم به نوع `byte` Cast کنیم اما اکنون با استفاده از عمل `Cast` این تبدیل انجام خواهد

شد:

```
int number1 = 5;  
  
byte number2 = (byte)number1;
```

حال اگر برنامه را اجرا کنید با خطأ مواجه نخواهید شد. همانطور که پیشتر اشاره شد ممکن است در هنگام تبدیلات مقادیر اصلی تغییر کنند. برای مثال وقتی که یک عدد با ممیز اعشار مثلاً از نوع int تبدیل می‌کنیم مقدار اعداد بعد از ممیز از بین می‌روند :

```
double number1 = 5.25;
int number2 = (int)number1;
System.out.println(number2);
```

5

خروجی کد بالا عدد ۵ است چون نوع داده‌ای int نمی‌تواند مقدار اعشار بگیرد. حالت دیگر را تصور کنید. اگر شما بخواهید یک متغیر را که دارای مقداری بیشتر از محدوده متغیر مقصود هست تبدیل کنید چه اتفاقی می‌افتد؟ مانند تبدیل زیر که می‌خواهیم متغیر number1 را که دارای مقدار ۳۰۰ است را به نوع بایت تبدیل کنیم که محدود اعداد بین -۲۵۵ و ۲۵۵ را پوشش می‌دهد.

```
int number1 = 300;
byte number2 = (byte)number1;
System.out.println(MessageFormat.format("Value of number2 is {0}.", number2));
```

Value of number2 is 44.

خروجی کد بالا عدد ۴۴ است. فقط می‌تواند شامل اعداد ۰ تا ۲۵۵ باشد و نمی‌تواند مقدار ۳۰۰ را در خود ذخیره کند. حال می‌خواهیم ببینیم که چرا به جای عدد ۳۰۰ ما عدد ۴۴ را در خروجی می‌گیریم. این کار به تعداد بیت‌ها بستگی دارد. یک byte دارای ۸ بیت است در حالی که int دارای ۳۲ بیت است. حال اگر به مقدار باینری ۲ عدد توجه کنید متوجه می‌شوید که چرا خروجی عدد ۴۴ است.

300 =	000000000000000000000000100101100
255 =	11111111
44 =	00101100

خروجی بالا نشان می‌دهد که بیشترین مقدار byte که عدد ۲۵۵ است می‌تواند فقط شامل ۸ بیت باشد (۱۱۱۱۱۱۱) بنابراین فقط ۸ بیت اول مقدار int به متغیر byte انتقال می‌یابد که شامل (۰۰۱۰۱۱۰۰) یا عدد ۴۴ در مبنای ۱۰ است.

عبارات و عملگرهای جاوا

ابتدا با دو کلمه آشنا شوید :

عملگر: نمادهایی هستند که اعمال خاص انجام می‌دهند.

- عملوند: مقادیری که عملگرها بر روی آن‌ها عملی انجام می‌دهند.

مثالاً $X+Y$ یک عبارت است که در آن X و Y عملوند و علامت $+$ عملگر به حساب می‌آیند. زبانهای برنامه نویسی جدید دارای عملگرهایی هستند که از اجزاء معمول زبان به حساب می‌آیند. جاوا دارای عملگرهای مختلفی از جمله عملگرهای ریاضی، تخصیصی، مقایسه‌ای، منطقی و بیتی می‌باشد. از عملگرهای ساده ریاضی می‌توان به عملگر جمع و تفریق اشاره کرد. سه نوع عملگر در جاوا وجود دارد :

- یگانی (Unary) - به یک عملوند نیاز دارد
- دودوبی (Binary) - به دو عملوند نیاز دارد
- سه تایی (Ternary) - به سه عملوند نیاز دارد

انواع مختلف عملگر که در ای بخش مورد بحث قرار می‌گیرند، عبارت‌اند از :

- عملگرهای ریاضی
- عملگرهای تخصیصی
- عملگرهای مقایسه‌ای
- عملگرهای منطقی
- عملگرهای بیتی

عملگرهای ریاضی

جاوا از عملگرهای ریاضی برای انجام محاسبات استفاده می‌کند. جدول زیر عملگرهای ریاضی جاوا را نشان می‌دهد :

عملگر	دسته	مثال	نتیجه
$+$	Binary	<code>var1 = var2 + var3;</code>	Var1 برابر است با حاصل جمع Var2 و Var3
$-$	Binary	<code>var1 = var2 - var3;</code>	Var1 برابر است با حاصل تفریق Var2 و Var3
$*$	Binary	<code>var1 = var2 * var3;</code>	Var1 برابر است با حاصل ضرب Var2 در Var3

Var1 برابر است با حاصل تقسیم var2 بر var3	<code>var1 = var2 / var3;</code>	Binary	/
Var1 برابر است با باقیمانده تقسیم var2 و var3	<code>var1 = var2 % var3;</code>	Binary	%
Var1 برابر است با مقدار var2	<code>var1 = +var2;</code>	Unary	+
Var1 برابر است با مقدار var2 ضربدر -1	<code>var1 = -var2;</code>	Unary	-

دیگر عملگرهای جاوا عملگرهای کاهش و افزایش هستند. این عملگرها مقدار 1 را از متغیرها کم یا به آنها اضافه می‌کنند. از این متغیرها

اغلب در حلقه‌ها استفاده می‌شود :

عملگر	دسته	مثال	نتیجه
++	Unary	<code>var1 = ++var2;</code>	مقدار var1 برابر است با var2 بعلاوه 1
--	Unary	<code>var1 = --var2;</code>	مقدار var1 برابر است با var2 منهای 1
++	Unary	<code>var1 = var2++;</code>	مقدار var1 برابر است با var2. به متغیر var2 یک واحد اضافه می‌شود.
--	Unary	<code>var1 = var2--;</code>	مقدار var1 برابر است با var2. از متغیر var2 یک واحد کم می‌شود.

به این نکته توجه داشته باشید که محل قرار گیری عملگر در نتیجه محاسبات تأثیر دارد. اگر عملگر قبل از متغیر var2 بباید افزایش یا کاهش var1 اتفاق می‌افتد. چنانچه عملگرها بعد از متغیر var2 قرار بگیرند ابتدا var1 برابر var2 می‌شود و سپس متغیر var2 افزایش یا کاهش می‌یابد. به مثال‌های زیر توجه کنید :

```
package myfirstprogram;
import java.text.MessageFormat;
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1;
```

```

        x = ++y;

        System.out.println(MessageFormat.format("x= {0}",x));
        System.out.println(MessageFormat.format("y= {0}", y));
    }
}

x=2
y=2

```

```

package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1;

        x = --y;

        System.out.println(MessageFormat.format("x= {0}",x));
        System.out.println(MessageFormat.format("y= {0}", y));
    }
}

x=0
y=0

```

همانطور که در دو مثال بالا مشاهده می‌کنید، درج عملگرهای `--` و `++` قبل از عملوند `y` باعث می‌شود که ابتدا یک واحد از `y` کم و یا یک واحد به `y` اضافه شود و سپس نتیجه در عملوند `x` قرار بگیرد. حال به دو مثال زیر توجه کنید :

```

package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1;

        x = y--;

        System.out.println(MessageFormat.format("x= {0}",x));
        System.out.println(MessageFormat.format("y= {0}", y));
    }
}

```

}

$$\begin{array}{l} x=1 \\ y=0 \end{array}$$

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 0;
        int y = 1;

        x = y++;

        System.out.println(MessageFormat.format("x= {0}",x));
        System.out.println(MessageFormat.format("y= {0}", y));
    }
}

x=1
y=2
```

همانطور که در دو متن، بالا مشاهده می‌کنید، در عملگرهای $-$ و $+$ بعد از عملوند y باعث می‌شود که ابتدا مقدار y در داخل متغیر x قرار

نگرد و سیس، یک واحد از ۱ کم و نا یک واحد به آن، اضافه شود. حال، م-تبانیم با اتحاد یک برنامه نجوه عملکرد عملگرهای ریاضی، در

جاوا را یاد بگیریم :

```

        System.out.println(MessageFormat.format("The quotient of {0} and {1} is {2}.",
                                              num1, num2, ((double)num1 / num2)));
        System.out.println(MessageFormat.format("The remainder of {0} and {1} is {2}.",
                                              num1, num2, (num1 % num2)));
    }
}

```

```

The sum of 5 and 3 is 8.
The difference of 5 and 3 is 2.
The product of 5 and 3 is 15.
The quotient of 5 and 3 is 1.67.
The remainder of 5 divided by 3 is 2

```

برنامه بالا نتیجه هر عبارت را نشان می‌دهد. در این برنامه از متدهای `println()` برای نشان دادن نتایج در سطرهای متفاوت استفاده شده است. در این مثال با یک نکته عجیب مواجه می‌شویم و آن حاصل تقسیم دو عدد صحیح است. وقتی که دو عدد صحیح را بر هم تقسیم کنیم حاصل باید یک عدد صحیح و فاقد بخش کسری باشد. اما همانطور که مشاهده می‌کنید اگر فقط یکی از اعداد را به نوع اعشاری تبدیل کنیم (در مثال می‌بینید) حاصل به صورت اعشار نشان داده می‌شود.

عملگرهای تخصیصی

نوع دیگر از عملگرهای جاوا عملگرهای جایگزینی نام دارند. این عملگرهای مقدار متغیر سمت راست خود را در متغیر سمت چپ قرار می‌دهند. جدول زیر انواع عملگرهای تخصیصی در جاوا را نشان می‌دهد:

عملگر	مثال	نتیجه
=	var1 = var2;	مقدار var1 برابر است با مقدار var2
+=	var1 += var2;	مقدار var1 برابر است با حاصل جمع var1 و var2
-=	var1 -= var2;	مقدار var1 برابر است با حاصل تفاضل var1 و var2
*=	var1 *= var2;	مقدار var1 برابر است با حاصل ضرب var1 در var2
/=	var1 /= var2;	مقدار var1 برابر است با حاصل تقسیم var1 بر var2
%=	var1 %= var2;	مقدار var1 برابر است با باقیمانده تقسیم var1 بر var2

از عملگر `=` برای اتصال دو رشته نیز می‌توان استفاده کرد. استفاده از این نوع عملگرها در واقع یک نوع خلاصه نویسی در کد است. مثلاً

شکل اصلی کد `var1 += var2` به صورت `var1 = var1 + var2` می‌باشد. این حالت کدنویسی زمانی کارایی خود را نشان می‌دهد

که نام متغیرها طولانی باشد. برنامه زیر چگونگی استفاده از عملگرهای تخصیصی و تأثیر آن‌ها را بر متغیرها نشان می‌دهد.

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int number;

        System.out.println("Assigning 10 to number...");
        number = 10;
        System.out.println(MessageFormat.format("Number = {0}", number));

        System.out.println("Adding 10 to number...");
        number += 10;
        System.out.println(MessageFormat.format("Number = {0}", number));

        System.out.println("Subtracting 10 from number...");
        number -= 10;
        System.out.println(MessageFormat.format("Number = {0}", number));
    }
}

Assigning 10 to number...
Number = 10
Adding 10 to number...
Number = 20
Subtracting 10 from number...
Number = 10
```

در برنامه از ۳ عملگر تخصیصی استفاده شده است. ابتدا یک متغیر و مقدار ۱۰ با استفاده از عملگر `=` به آن اختصاص داده شده است.

سپس به آن با استفاده از عملگر `+=` مقدار ۱۰ اضافه شده است. و در آخر به وسیله عملگر `-=` عدد ۱۰ از آن کم شده است.

عملگرهای مقایسه‌ای

از عملگرهای مقایسه‌ای برای مقایسه مقادیر استفاده می‌شود. نتیجه این مقادیر یک مقدار بولی (منطقی) است. این عملگرها اگر نتیجه مقایسه دو مقدار درست باشد مقدار `true` و اگر نتیجه مقایسه اشتباه باشد مقدار `false` را نشان می‌دهند. این عملگرها به طور معمول در دستورات شرطی به کار می‌روند به این ترتیب که باعث ادامه یا توقف دستور شرطی می‌شوند. جدول زیر عملگرهای مقایسه‌ای در جاوا را نشان می‌دهد:

عملگر	دسته	مثال	نتیجه
<code>==</code>	Binary	<code>var1 = var2 == var3</code>	در صورتی <code>true</code> است که مقدار <code>var2</code> با مقدار <code>var3</code> برابر باشد در غیر اینصورت <code>false</code> است
<code>!=</code>	Binary	<code>var1 = var2 != var3</code>	در صورتی <code>true</code> است که مقدار <code>var2</code> با مقدار <code>var3</code> برابر نباشد در غیر اینصورت <code>false</code> است
<code><</code>	Binary	<code>var1 = var2 < var3</code>	در صورتی <code>true</code> است که مقدار <code>var2</code> کوچکتر از <code>var3</code> باشد در غیر اینصورت <code>false</code> است
<code>></code>	Binary	<code>var1 = var2 > var3</code>	در صورتی <code>true</code> است که مقدار <code>var2</code> بزرگتر از <code>var3</code> باشد در غیر اینصورت <code>false</code> است
<code><=</code>	Binary	<code>var1 = var2 <= var3</code>	در صورتی <code>true</code> است که مقدار <code>var2</code> کوچکتر یا مساوی <code>var3</code> باشد در غیر اینصورت <code>false</code> است
<code>>=</code>	Binary	<code>var1 = var2 >= var3</code>	در صورتی <code>true</code> است که مقدار <code>var2</code> بزرگتر یا مساوی <code>var3</code> باشد در غیر اینصورت <code>false</code> است

برنامه زیر نحوه عملکرد این عملگرهای نشان می‌دهد :

```
package myfirstprogram;

import java.text.MessageFormat;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int num1 = 10;
        int num2 = 5;

        System.out.println(MessageFormat.format("{0} == {1} : {2}", num1, num2, num1 == num2));
        System.out.println(MessageFormat.format("{0} != {1} : {2}", num1, num2, num1 != num2));
        System.out.println(MessageFormat.format("{0} < {1} : {2}", num1, num2, num1 < num2));
        System.out.println(MessageFormat.format("{0} > {1} : {2}", num1, num2, num1 > num2));
        System.out.println(MessageFormat.format("{0} <= {1} : {2}", num1, num2, num1 <= num2));
        System.out.println(MessageFormat.format("{0} >= {1} : {2}", num1, num2, num1 >= num2));
    }
}
```

```

10 == 5 : False
10 != 5 : True
10 < 5 : False
10 > 5 : True
10 <= 5 : False
10 >= 5 : True

```

در مثال بالا ابتدا دو متغیر را که می‌خواهیم با هم مقایسه کنیم را ایجاد کرده و به آن‌ها مقادیری اختصاص می‌دهیم. سپس با استفاده از یک عملگر مقایسه‌ای آن‌ها را با هم مقایسه کرده و نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که هنگام مقایسه دو متغیر از عملگر == به جای عملگر = باید استفاده شود. عملگر = عملگر تخصیصی است و در عبارتی مانند $y = x$ مقدار y را در به x اختصاص می‌دهد. عملگر == عملگر مقایسه‌ای است که دو مقدار را با هم مقایسه می‌کند مانند $x == y$ و اینطور خوانده می‌شود x برابر است با y .

عملگرهای منطقی

عملگرهای منطقی بر روی عبارات منطقی عمل می‌کنند و نتیجه آن‌ها نیز یک مقدار بولی است. از این عملگرهای اغلب برای شرط‌های پیچیده استفاده می‌شود. همانطور که قبلاً یاد گرفتیم مقادیر بولی می‌توانند `true` یا `false` باشند. فرض کنید که `var2` و `var3` دو مقدار بولی هستند.

عملگر	نام	دسته	مثال
&&	منطقی AND	Binary	<code>var1 = var2 && var3;</code>
	منطقی OR	Binary	<code>var1 = var2 var3;</code>
!	منطقی NOT	Unary	<code>var1 = !var1;</code>

عملگر منطقی (AND(&&))

اگر مقادیر دو طرف عملگر `AND` باشند عملگر `true` مقدار `true` را بر می‌گرداند. در غیر اینصورت اگر یکی از مقادیر با هر دوی آن‌ها باشند مقدار `false` را بر می‌گرداند. در زیر جدول درستی عملگر `AND` نشان داده شده است :

X	Y	<code>X && Y</code>
<code>true</code>	<code>true</code>	<code>true</code>
<code>true</code>	<code>false</code>	<code>false</code>

false	true	false
false	false	false

برای درک بهتر تأثیر عملگر AND یادآوری می‌کنم که این عملگر فقط در صورتی مقدار true را نشان می‌دهد که هر دو عاملوند مقدارشان true باشد. در غیر اینصورت نتیجه تمام ترکیب‌های بعدی false خواهد شد. استفاده از عملگر AND مانند استفاده از عملگرهای مقایسه‌ای است. به عنوان مثال نتیجه عبارت زیر درست (true) است اگر سن (age) بزرگ‌تر از ۱۸ و salary کوچک‌تر از ۱۰۰۰ باشد.

```
result = (age > 18) && (salary < 1000);
```

عملگر AND زمانی کارآمد است که ما با محدود خاصی از اعداد سرو کار داریم. مثلاً عبارت $100 \geq x \geq 10$ بدین معنی است که x می‌تواند مقداری شامل اعداد ۱۰ تا ۱۰۰ را بگیرد. حال برای انتخاب اعداد خارج از این محدوده می‌توان از عملگر منطقی AND به صورت زیر استفاده کرد.

```
inRange = (number <= 10) && (number >= 100);
```

عملگر منطقی (OR(||)

اگر یکی یا هر دو مقدار دو طرف عملگر OR درست (true) باشد، عملگر OR مقدار true را بر می‌گرداند. جدول درستی عملگر OR در زیر نشان داده شده است:

X	Y	X Y
true	true	true
true	false	true
false	true	true
false	false	false

در جدول بالا مشاهده می‌کنید که عملگر OR در صورتی مقدار false را بر می‌گرداند که مقادیر دو طرف آن false باشند. کد زیر را در نظر بگیرید. نتیجه این کد در صورتی درست (true) است که رتبه نهایی دانش آموز (finalGrade) بزرگ‌تر از ۷۵ یا نمره نهایی امتحان آن ۱۰۰ باشد.

```
isPassed = (finalGrade >= 75) || (finalExam == 100);
```

عملگر منطقی (NOT(!))

برخلاف دو اپراتور OR و AND عملگر منطقی NOT یک عملگر یگانی است و فقط به یک عاملوند نیاز دارد. این عملگر یک مقدار یا اصطلاح بولی را نفی می‌کند. مثلاً اگر عبارت یا مقدار true باشد آنرا false و اگر false باشد، آنرا true می‌کند. جدول زیر عملکرد اپراتور NOT را نشان می‌دهد:

	X	!X
true		false
false		true

نتیجه کد زیر در صورتی درست است که age (سن) بزرگتر یا مساوی ۱۸ نباشد.

```
isMinor = !(age >= 18);
```

عملگرهاي بيتي

عملگرهای بیتی به شما اجازه می‌دهند که شکل بازتری انواع داده‌ها را دستکاری کنید. برای درک بهتر این درس توصیه می‌شود که شما سیستم پاینتری و نحوه تبدیل اعداد دهدی به پاینتری را از لینک زیر پاد بگیرید:

<http://www.w3-farsi.com/?p=5698>

در سیستم باینری (دودویی) که کامپیوتر از آن استفاده می‌کند، وضعیت هر چیز یا خاموش است یا روشن. برای نشان دادن حالت روشن از عدد ۱ و برای نشان دادن حالت خاموش از عدد ۰ استفاده می‌شود. بنابراین اعداد باینری فقط می‌توانند صفر یا یک باشند. اعداد باینری را اعداد در مبنای ۲ و اعداد اعشاری را اعداد در مبنای ۱۰ می‌گویند. یک بیت نشان دهنده یک رقم باینری است و هر بایت نشان دهنده ۸ بیت است. به عنوان مثال برای یک داده از نوع `int` به ۳۲ بیت یا ۸ بایت فضا برای ذخیره آن نیاز داریم، این بدین معناست که اعداد از ۳۲ رقم ۰ و ۱ برای ذخیره استفاده می‌کنند. برای مثال عدد ۱۰۰ و قتی به عنوان یک متغیر از نوع `int` ذخیره می‌شود در کامپیوتر به صورت زیر خواهد می‌شد :

عدد ۱۰۰ در مبنای ده معادل عدد ۱۱۰۰۱۰۰ در مبنای ۲ است. در اینجا ۷ رقم سمت راست نشان دهنده عدد ۱۰۰ در مبنای ۲ است و مابقی

صفرهای سمت راست برای پر کردن بیت‌هایی است که عدد از نوع int نیاز دارد. به این نکته توجه کنید که اعداد باینری از سمت راست

به چپ خوانده می‌شوند. عملگرهای بیتی جاوا در جدول زیر نشان داده شده‌اند :

عملگر	نام	دسته	مثال
&	بیتی AND	Binary	$x = y \& z;$
	بیتی OR	Binary	$x = y z;$
^	بیتی XOR	Binary	$x = y ^ z;$
~	بیتی NOT	Unary	$x = \sim y;$
&=	بیتی تخصیصی AND	Binary	$x \&= y;$
=	بیتی تخصیصی OR	Binary	$x = y;$
^=	بیتی تخصیصی XOR	Binary	$x ^= y;$

عملگر بیتی (AND(&)

عملگر بیتی AND کاری شبیه عملگر منطقی AND انجام می‌دهد با این تفاوت که این عملگر بر روی بیت‌ها کار می‌کند. اگر مقادیر دو طرف

آن ۱ باشد مقدار ۱ را بر می‌گرداند و اگر یکی یا هر دو طرف آن صفر باشد مقدار صفر را بر می‌گرداند. جدول درستی عملگر بیتی AND در زیر

آمده است:

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

در زیر نحوه استفاده از عملگر بیتی AND آمده است :

```
int result = 5 & 3;
System.out.println(result);
1
```

همانطور که در مثال بالا مشاهده می‌کنید نتیجه عملکرد عملگر AND بر روی دو مقدار ۵ و ۳ عدد یک می‌شود. اجازه بدھید بینیم که چطور

این نتیجه را به دست می‌آید:

```
5: 00000000000000000000000000000000101
3: 00000000000000000000000000000000011
-----
1: 00000000000000000000000000000000001
```

ابتدا دو عدد ۵ و ۳ به معادل باینری‌شان تبدیل می‌شوند. از آنجاییکه هر عدد صحیح ۳۲ بیت است از صفر برای پر کردن بیت‌های خالی استفاده می‌کنیم. با استفاده از جدول درستی عملگر بیتی AND می‌توان فهمید که چرا نتیجه عدد یک می‌شود.

عملگر بیتی (| OR)

اگر مقادیر دو طرف عملگر بیتی OR هر دو صفر باشند نتیجه صفر در غیر اینصورت ۱ خواهد شد. جدول درستی این عملگر در زیر آمده است

:

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

نتیجه عملگر بیتی OR در صورتی صفر است که عملوندهای دو طرف آن صفر باشند. اگر فقط یکی از دو عملوند یک باشد نتیجه یک خواهد شد. به مثال زیر توجه کنید :

```
int result = 7 | 9;
```

```
System.out.println(result);
```

15

وقتی که از عملگر بیتی OR دو مقدار در مثال بالا (۷ و ۹) استفاده می‌کنیم نتیجه ۱۵ می‌شود. حال بررسی می‌کنیم که چرا این نتیجه

به دست آمده است؟

با استفاده از جدول درستی عملگر بیتی OR می‌توان نتیجه استفاده از این عملگر را تشخیص داد. عدد ۱۱۱۱ باینری معادل عدد ۱۵ صحیح

عملگر سنتی XOR (^)

جدوا، درست، این عملگر در زیر آمده است :

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

تأثیر عملگر بت، XOR را بر روی دو مقدار مشاده می‌کند:

```
int result = 5 ^ 7;  
  
System.out.println(result);
```

2

در نزد معادا، باین‌ها، اعداد بالا (۵ و ۷) نشان، داده شده است.

با نگاه کردن به جدول درستی عملگر بیتی XOR، می‌توان فهمید که چرا نتیجه عدد ۲ می‌شود.

عملگر پیتی (NOT (~

اين عملگر يك عملگر يگان، است و فقط به يك عملوند نياز دارد. در زير جدوا، درست، اين عملگر آمده است:

	X	NOT X
	1	0
	0	1

عملگر بیتی NOT مقادیر بیت‌ها را معکوس می‌کند. در زیر چگونگی استفاده از این عملگر آمده است:

```
int result = ~7;  
  
System.out.println(result);
```

به نمایش، بانیز، مثا، بالا که در زیر نشان داده شده است توجه نمایید:

عملگر یتی، تغییر مکان (shift)

این نوع عملگرها به شما اجازه می‌دهند که بیت‌ها را به سمت چپ یا راست جا به جا کنید. دو نوع عملگر بیتی تغییر مکان وجود دارد که هر کدام دو عملوند قبول می‌کنند. عملوند سمت چپ این عملگرها حالت باینتری یک مقدار و عملوند سمت راست تعداد جابه جایی بیت‌ها را نشان می‌دهد.

مثال	دسته	نام	عملگر
<code>x = y << 2;</code>	Binary	تغییر مکان به سمت چپ	<code>>></code>

<code>x = y >> 2;</code>	Binary	تغییر مکان به سمت راست	<<
--------------------------------	--------	------------------------	----

عملگر تغییر مکان به سمت چپ

این عملگر بیت‌های عملوند سمت چپ را به تعداد ۷ مکان مشخص شده توسط علملوند سمت راست، به سمت چپ منتقل می‌کند. به

عنوان مثال :

```
int result = 10 << 2;  
  
System.out.println(result);
```

40

در مثال بالا ما بیت‌های مقدار ۱۰ را دو مکان به سمت چیز منتقل کرده‌ایم، حال باید تأثیر این انتقال را بررسی کنیم:

مشاهده می‌کنید که همه بیت‌ها به اندازه دو واحد به سمت چپ منتقل شده‌اند. در این انتقال دو صفر از صفرهای سمت چپ کم می‌شود و در عوض، دو صفر به سمت راست اضافه می‌شود.

عملگر تغییر مکان، به سمت راست

اب، عملگر شیه به عمگ تغییر مکان، به سمت حب است با این تفاوت که بیت‌ها را به سمت راست حا به حا می‌کند. به عنوان مثلاً:

```
int result = 100 >> 4;  
  
System.out.println(result);
```

6

با استفاده از عملگر تغییر مکان، به سمت راست بیت‌های، مقدار ۱۰۰ را به اندازه ۴ واحد به سمت حب حا به حا مکنیم. احازه دهدید تأثیر

ابن حا به حایه را مورد بررسی قرار دهیم :

هر بیت به اندازه ۴ واحد به سمت راست منتقل می‌شود، بنابراین ۴ بیت اول سمت راست حذف شده و چهار صفر به سمت چپ اضافه می‌شود.

تقدم عملگرها

تقدم عملگرها مشخص می‌کند که در محاسباتی که بیش از دو عملوند دارند ابتدا کدام عملگر اثرش را اعمال کند. عملگرها در جاوا در محاسبات دارای حق تقدم هستند. به عنوان مثال :

```
number = 1 + 2 * 3 / 1;
```

اگر ما حق تقدم عملگرها را رعایت نکنیم و عبارت بالا را از سمت چپ به راست انجام دهیم نتیجه ۹ خواهد شد ($1+2=3 \times 3=9$) و در آخر $9/1=9$. اما کمپایلر با توجه به تقدم عملگرها محاسبات را انجام می‌دهد. برای مثال عمل ضرب و تقسیم نسبت به جمع و تفریق تقدم دارند. بنابراین در مثال فوق ابتدا عدد ۲ ضربدر ۳ و سپس نتیجه آن‌ها تقسیم بر ۱ می‌شود که نتیجه ۶ به دست می‌آید. در آخر عدد ۶ با ۱ جمع می‌شود و عدد ۷ حاصل می‌شود. در جدول زیر تقدم برخی از عملگرهای جاوا آمده است :

تقدم	عملگر
بالاترین	<code>++, -, (used as prefixes); +, - (unary)</code>
	<code>*, /, %</code>
	<code>+, -</code>
	<code><<, >></code>
	<code><, >, <=, >=</code>
	<code>==, !=</code>
	<code>&</code>
	<code>^</code>
	<code> </code>
	<code>&&</code>

	=, *=, /=, %=, +=, -=
پایین ترین	++, - (used as suffixes)

ابتدا عملگرهای با بالاترین و سپس عملگرهای با پایین ترین حق تقدم در محاسبات تأثیر می‌گذارند. به این نکته توجه کنید که تقدم عملگرها ++ و -- به مکان قرارگیری آن‌ها بستگی دارد (در سمت چپ یا راست عملوند باشند). به عنوان مثال :

```
int number = 3;
number1 = 3 + ++number; //results to 7
number2 = 3 + number++; //results to 6
```

در عبارت اول ابتدا به مقدار `number` یک واحد اضافه شده و ۴ می‌شود و سپس مقدار جدید با عدد ۳ جمع می‌شود و در نهایت عدد ۷ به دست می‌آید. در عبارت دوم مقدار عددی ۳ به مقدار `number` اضافه می‌شود و عدد ۶ به دست می‌آید. سپس این مقدار در متغیر `number2` قرار می‌گیرد. و در نهایت مقدار `number` به ۴ افزایش می‌یابد. برای ایجاد خوانایی در تقدم عملگرها و انجام محاسباتی که در آن‌ها از عملگرهای زیادی استفاده می‌شود از پرانتز استفاده می‌کنیم :

```
number = ( 1 + 2 ) * ( 3 / 4 ) % ( 5 - ( 6 * 7 ) );
```

در مثال بالا ابتدا هر کدام از عباراتی که داخل پرانتز هستند مورد محاسبه قرار می‌گیرند. به نکته‌ای در مورد عبارتی که در داخل پرانتز سوم قرار دارد، توجه کنید. در این عبارت ابتدا مقدار داخلی ترین پرانتز مورد محاسبه قرار می‌گیرد. یعنی مقدار ۶ ضربدر ۷ شده و سپس از ۵ کم می‌شود. اگر دو یا چند عملگر با حق تقدم یکسان موجود باشد ابتدا باید هر کدام از عملگرها را که در ابتدای عبارت می‌آیند مورد ارزیابی قرار دهید. به عنوان مثال :

```
number = 3 * 2 + 8 / 4;
```

هر دو عملگر * و / دارای حق تقدم یکسانی هستند. بنابر این شما باید از چپ به راست آن‌ها را در محاسبات تأثیر دهید. یعنی ابتدا ۳ را ضربدر ۲ می‌کنید و سپس عدد ۸ را بر ۴ تقسیم می‌کنید. در نهایت نتیجه دو عبارت را جمع کرده و در متغیر `number` قرار می‌دهید.

گرفتن ورودی از کاربر

جاوا تعدادی متدهای برای گرفتن ورودی از کاربر در اختیار شما قرار می‌دهد. این متدها در کلاس `Scanner` قرار دارند. این کلاس در پکیج `java.util` قرار دارد و در نتیجه برای استفاده از آن باید آن را در برنامه به صورت زیر وارد کنید :

```
import java.util.Scanner;
```

از کلاس `MessageFormat` هم برای قالب بندي خروجي استفاده می‌کنیم. این دو کلاس را در خطوط ۳ و ۴ وارد کردہ‌ایم. متدهای کلاس `Scanner` که مقادیر وارد شده توسط کاربر را از صفحه کلید می‌خوانند عبارت‌اند از :

متدها	توضیح
<code>nextByte()</code>	برای دریافت یک نوع داده از نوع <code>byte</code> به کار می‌رود.
<code>nextShort()</code>	برای دریافت یک نوع داده از نوع <code>short</code> به کار می‌رود.
<code>nextInt()</code>	برای دریافت یک نوع داده از نوع <code>int</code> به کار می‌رود.
<code>nextLong()</code>	برای دریافت یک نوع داده از نوع <code>long</code> به کار می‌رود.
<code>next()</code>	برای دریافت یک کلمه ساده به کار می‌رود.
<code>nextLine()</code>	برای دریافت یک خط رشته به کار می‌رود.
<code>nextBoolean()</code>	برای دریافت یک نوع داده از نوع <code>boolean</code> به کار می‌رود.
<code>nextFloat()</code>	برای دریافت یک نوع داده از نوع <code>float</code> به کار می‌رود.
<code>nextDouble()</code>	برای دریافت یک نوع داده از نوع <code>double</code> به کار می‌رود.

به برنامه زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import java.text.MessageFormat;
```

```

4   import java.util.Scanner;
5
6   public class MyFirstProgram
7   {
8       public static void main(String[] args)
9       {
10          String name;
11          int age;
12          double height;
13
14          Scanner input = new Scanner(System.in);
15
16          System.out.print("Enter your Name: ");
17          name = input.next();
18
19          System.out.print("Enter your Age: ");
20          age = input.nextInt();
21
22          System.out.print("Enter your Height:");
23          height = input.nextDouble();
24
25          System.out.println();
26
27          System.out.println(MessageFormat.format("Name is {0}.", name));
28          System.out.println(MessageFormat.format("Age is {0}.", age));
29          System.out.println(MessageFormat.format("Height is {0}.", height));
30      }
31  }

```

```

Enter your Name: john
Enter your Age: 18
Enter your Height:160.5

```

```

Name is john.
Age is 18.
Height is 160.5.

```

اجازه دهید که برنامه را تشریح کنیم. ابتدا در خطوط ۳ و ۴ برنامه، کلاس Scanner و MessageFormat را با استفاده از کلمه import که بروزرسانی شده است تشریح کنیم. در خطوط ۱۰ و ۱۱ و ۱۲ یک شیء برای دریافت نام، یک متغیر از نوع صحیح به نام age برای دریافت سن و یک متغیر از نوع double برای دریافت قد شخص تعریف نموده ایم. درباره خط ۱۴ زیاد توضیح نمی دهم، فقط کافیست که این را بدانید که وجود این خط برای دریافت ورودی از کاربر اجباری است. در درس های آینده با مفاهیم متدها، شیء و کلاس آشنا خواهد شد. برنامه از کاربر می خواهد که نام خود را وارد کند (خط ۱۶). در خط ۱۷ شما به عنوان کاربر نام خود را وارد می کنید. مقدار متغیر name، برابر مقداری است که توسعه متدها می شود. از آنجاییکه name از نوع رشته است باید از متدهای next() برای دریافت استفاده کنیم. در خط ۱۹ برنامه از شما می خواهد که سن خود را وارد کند. در خط ۲۰ شما سن خود را وارد می کنید. مقدار متغیر age، برابر مقداری است که توسعه متدهای nextInt() می شود. از آنجاییکه age از نوع صحیح است باید از متدهای nextInt() برای دریافت استفاده کنیم. سپس برنامه

از ما قد را سؤال می‌کند (خط ۲۲). چون `height` از نوع `double` است پس برای خواندن آن از متدهای `nextDouble()` استفاده کردہ‌ایم.

حال شما می‌توانید با اجرای برنامه و وارد کردن مقادیر نتیجه را مشاهده کنید.

ساختارهای تصمیم

تقریباً همه زبانهای برنامه نویسی به شما اجازه اجرای کد را در شرایط مطمئن می‌دهند. حال تصور کنید که یک برنامه دارای ساختار تصمیم گیری نباشد و همه کدها را اجرا کند. این حالت شاید فقط برای چاپ یک پیغام در صفحه مناسب باشد ولی فرض کنید که شما بخواهید اگر مقدار یک متغیر با یک عدد برابر باشد سپس یک پیغام چاپ شود آن وقت با مشکل مواجه خواهید شد. جاوا راههای مختلفی برای رفع این نوع مشکلات ارائه می‌دهد. در این بخش با مطالعه زیر آشنا خواهید شد :

- دستور `if`

- دستور `if...else`

- عملگر سه تایی

- دستور `if` چندگانه

- دستور `if` تو در تو

- عملگرهای منطقی

- دستور `switch`

دستور `if`

می‌توان با استفاده از دستور `if` و یک شرط خاص که باعث ایجاد یک کد می‌شود یک منطق به برنامه خود اضافه کنید. دستور `if` ساده‌ترین دستور شرطی است که به برنامه می‌گوید، اگر شرطی برقرار است کد معینی را انجام بده. ساختار دستور `if` به صورت زیر است :

```
if (condition)
    code to execute;
```

قبل از اجرای دستور `if` ابتدا شرط بررسی می‌شود. اگر شرط برقرار باشد یعنی درست باشد سپس کد اجرا می‌شود. شرط یک عبارت مقایسه‌ای است. می‌توان از عملگرهای مقایسه‌ای برای تست درست یا اشتباه بودن شرط استفاده کرد. اجازه بدھید که نگاهی به نحوه

استفاده از دستور `if` در داخل برنامه بیندازیم. برنامه زیر پیغام Hello World را اگر مقدار `number` کمتر از ۱۰ و Goodbye World را اگر مقدار `number` از ۱۰ بزرگ‌تر باشد در صفحه نمایش می‌دهد.

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         //Declare a variable and set it a value less than 10
8         int number = 5;
9
10        //If the value of number is less than 10
11        if (number < 10)
12            System.out.println("Hello World.");
13
14        //Change the value of a number to a value which
15        // is greater than 10
16        number = 15;
17
18        //If the value of number is greater than 10
19        if (number > 10)
20            System.out.println("Goodbye World.");
21    }
22 }
```

```
Hello World.
Goodbye World.
```

در خط ۸ یک متغیر با نام `number` تعریف و مقدار ۵ به آن اختصاص داده شده است. وقتی به اولین دستور `if` در خط ۱۱ می‌رسیم برنامه تشخیص می‌دهد که مقدار `number` از ۱۰ کمتر است یعنی ۵ کوچک‌تر از ۱۰ است.

منطقی است که نتیجه مقایسه درست می‌باشد بنابراین دستور `if` دستور را اجرا می‌کند (خط ۱۲) و پیغام Hello World چاپ می‌شود. حال مقدار `number` را به ۱۵ تغییر می‌دهیم (خط ۱۶). وقتی به دومین دستور `if` در خط ۱۹ می‌رسیم برنامه مقدار `number` را با ۱۰ مقایسه می‌کند و چون مقدار `number` یعنی ۱۵ از ۱۰ بزرگ‌تر است برنامه پیغام Goodbye World را چاپ می‌کند (خط ۲۰). به این نکته توجه کنید که دستور `if` را می‌توان در یک خط نوشت :

```
if ( number > 10 ) System.out.println("Goodbye World.");
```

شما می‌توانید چندین دستور را در داخل دستور `if` بنویسید. کافیست که از یک آکولاد برای نشان دادن ابتدا و انتهای دستورات استفاده کنید. همه دستورات داخل بین آکولاد جز بدن دستور `if` هستند. نحوه تعریف چند دستور در داخل بدن `if` به صورت زیر است :

```
if (condition)
```

```
{
    statement1;
    statement2;
    .
    .
    .
    statementN;
}
```

این هم یک مثال ساده :

```
if (x > 10)
{
    System.out.println("x is greater than 10.");
    System.out.println("This is still part of the if statement.");
}
```

در مثال بالا اگر مقدار x از ۱۰ بزرگتر باشد دو پیغام چاپ می‌شود. حال اگر به عنوان مثال آکولاد را حذف کنیم و مقدار x از ۱۰ بزرگتر نباشد مانند کد زیر :

```
if (x > 10)
    System.out.println("x is greater than 10.");
    System.out.println("This is still part of the if statement. (Really?)");
```

کد بالا در صورتی بهتر خوانده می‌شود که بین دستورات فاصله بگذاریم :

```
if (x > 10)
    System.out.println("x is greater than 10.");
System.out.println("This is still part of the if statement. (Really?)");
```

می‌بیند که دستور دوم (خط ۳) در مثال بالا جز دستور `if` نیست. اینجاست که چون ما فرض را بر این گذاشته‌ایم که مقدار x از ۱۰ کوچکتر است پس خط `(Really?)` چاپ می‌شود. در نتیجه اهمیت وجود آکولاد مشخص می‌شود. به عنوان تمرین همیشه حتی اگر فقط یک دستور در بدنه `if` داشتید برای آن یک آکولاد بگذارید. فراموش نکنید که از قلم انداختن یک آکولاد باعث به وجود آمدن خطأ شده و یافتن آن را سخت می‌کند. یکی از خطاهای معمول کسانی که برنامه نویسی را تازه شروع کرده‌اند قرار دادن سیمیکولون در سمت راست پرانتز `if` است. به عنوان مثال :

```
if (x > 10);
    System.out.println("x is greater than 10");
```

به یاد داشته باشید که if یک مقایسه را انجام می‌دهد و دستور اجرایی نیست. بنابراین برنامه شما با یک خطای منطقی مواجه می‌شود.

همیشه به یاد داشته باشید که قرار گرفتن سیمیکولن در سمت راست پرانتز if به منزله این است که بلوک کد در اینجا به پایان رسیده است. به این نکته توجه داشته باشید که شرط‌ها مقادیر بولی هستند، بنابراین شما می‌توانید نتیجه یک عبارت را در داخل یک متغیر بولی ذخیره کنید و سپس از متغیر به عنوان شرط در دستور if استفاده کنید. اگر مقدار year برابر ۲۰۰۰ باشد سپس حاصل عبارت در متغیر `isNewMillenium` ذخیره می‌شود. می‌توان از متغیر برای تشخیص کد اجرایی بدنه دستور if استفاده کرد خواه مقدار متغیر درست باشد یا نادرست.

```
boolean isNewMillenium = year == 2000;

if (isNewMillenium)
{
    System.out.println("Happy New Millenium!");
}
```

دستور if...else

دستور if فقط برای اجرای یک حالت خاص به کار می‌رود. یعنی اگر حالتی برقرار بود کارخاصی انجام شود. اما زمانی که شما بخواهید اگر شرط خاصی برقرار شد یک دستور و اگر برقرار نبود، دستور دیگر اجرا شود، باید از دستور if...else استفاده کنید. ساختار دستور if...else در زیر آمده است :

```
if (condition)
{
    code to execute if condition is true;
}
else
{
    code to execute if condition is false;
}
```

از کلمه کلیدی else نمی‌توان به تنها یی استفاده کرد بلکه حتماً باید با if به کار برد شود. اگر فقط یک کد اجرایی در داخل بدنه if و بدنه else دارید استفاده از آکولاد اختیاری است. کد داخل بلوک else فقط در صورتی اجرا می‌شود که شرط داخل دستور if نادرست باشد. در زیر نحوه استفاده از دستور if...else آمده است.

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int number = 5;
```

```

8
9     //Test the condition
10    if (number < 10)
11    {
12        System.out.println("The number is less than 10.");
13    }
14    else
15    {
16        System.out.println("The number is either greater than or equal to 10.");
17    }
18
19    //Modify value of number
20    number = 15;
21
22    //Repeat the test to yield a different result
23    if (number < 10)
24    {
25        System.out.println("The number is less than 10.");
26    }
27    else
28    {
29        System.out.println("The number is either greater than or equal to 10.");
30    }
31}
32

```

در خط ۷ یک متغیر به نام `number` تعریف کرده‌ایم و در خط ۱۰ تست می‌کنیم که آیا مقدار متغیر `number` از ۱۰ کمتر است یا نه و چون کمتر است در نتیجه کد داخل بلوک `if` اجرا می‌شود (خط ۱۲) و اگر مقدار `number` را تغییر دهیم و به مقداری بزرگ‌تر از ۱۰ تغییر دهیم (خط ۲۰)، شرط نادرست می‌شود (خط ۲۳) و کد داخل بلوک `else` اجرا می‌شود (خط ۲۹). مانند بلوک `if` نباید به آخر کلمه کلیدی `else` سیمیکولن اضافه شود.

دستور `if` تو در تو

می‌توان از دستور `if` تو در تو در جاوا استفاده کرد. یک دستور ساده `if` در داخل دستور `if` دیگر.

```

if (condition)
{
    code to execute;

    if (condition)
    {
        code to execute;
    }
    else if (condition)
    {
        if (condition)
        {
            code to execute;
        }
    }
}

```

```

else
{
    if (condition)
    {
        code to execute;
    }
}

```

اجازه بدھید کہ نحوه استفادہ از دستور if تو در تو را نشان دھیم :

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int age = 21;
8
9         if (age > 12)
10        {
11            if (age < 20)
12            {
13                System.out.println("You are teenage");
14            }
15            else
16            {
17                System.out.println("You are already an adult.");
18            }
19        }
20        else
21        {
22            System.out.println("You are still too young.");
23        }
24    }
25 }

```

You are already an adult.

اجازه بدھید کہ برنامہ را کالبد شکافی کنیم. ابتدا در خط ۷ یک متغیر به نام age تعریف می‌کنیم و مقدار آن را برابر ۲۱ قرار می‌دهیم. سپس به اولین دستور if می‌رسیم (خط ۹). در این قسمت اگر سن شما بیشتر از ۱۲ سال باشد برنامه وارد بدنے دستور if می‌شود در غیر اینصورت وارد بلوک else (خط ۲۰) مربوط به همین دستور if می‌شود. حال فرض کنیم که سن شما بیشتر از ۱۲ سال است و شما وارد بدنے اولین if شده‌اید. در بدنے اولین if یک دستور if دیگر را مشاهده می‌کنید. اگر سن کمتر ۲۰ باشد دستور You are teenage چاپ می‌شود (خط ۱۷) و چون مقدار متغیر تعریف شده در خط ۷ بزرگ‌تر از ۱۳ در غیر اینصورت دستور You are already an adult (خط ۱۸) مربوط به بخش else می‌شود. حال فرض کنید که مقدار متغیر age کمتر از ۱۲ بود، در این صورت دستور است پس دستور مربوط به بخش else خط ۱۷ چاپ می‌شود.

بخش else خط ۲۰ یعنی You are still too young چاپ می‌شد. پیشنهاد می‌شود که از if تو در تو در برنامه کمتر استفاده کنید

چون خوانایی برنامه را پایین می‌آورد

عملگر شرطی

عملگر شرطی (?:) در جاوا مانند دستور شرطی if...else عمل می‌کند. در زیر نحوه استفاده از این عملگر آمده است:

```
<condition> ? <result if true> : <result if false>
```

عملگر شرطی تنها عملگر سه تایی جاوا است که نیاز به سه عملوند دارد، شرط، یک مقدار زمانی که شرط درست باشد و یک مقدار زمانی که شرط نادرست باشد. اجازه بدھید که نحوه استفاده این عملگر را در داخل برنامه مورد بررسی قرار دهیم.

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int number = -10;
8
9         int ABS = (number > 0) ? (number) : -(number);
10
11        System.out.println("ABS      = " + ABS);
12    }
13 }
```

10

برنامه بالا نحوه استفاده از این عملگر شرطی را نشان می‌دهد. در این برنامه قصد ما به دست آوردن قدر مطلق یک عدد است. ابتدا در خط ۷ یک متغیر از نوع int تعریف کرده و مقدار آن را ۱۰- می‌گذاریم. در خط ۹ یک متغیر از نوع صحیح تعریف کردہ‌ایم تا نتیجه را در آن قرار دهیم. خط ۹ به این صورت تعریف می‌شود: "اگر مقدار number از ۰ بزرگ‌تر باشد خود مقدار را در متغیر ABS قرار بده در غیر اینصورت آن را در منفی ضرب کرده و آن را در متغیر ABS قرار بده". حال برنامه بالا را با استفاده از دستور if else می‌نویسیم:

```
int number = -10;

if(number > 0)
{
    System.out.println(number);
}
else
{
    System.out.println(-(number));
}
```

هنگامی که چندین دستور در داخل یک بلوک if یا else دارید از عملگر شرطی استفاده نکنید چون خوانایی برنامه را پایین می‌آورد.

دستور if چندگانه

اگر بخواهید چند شرط را بررسی کنید چکار می‌کنید؟ می‌توانید از چندین دستور if استفاده کنید و بهتر است که این دستورات if را به صورت زیر بنویسید :

```
if (condition)
{
    code to execute;
}
else
{
    if (condition)
    {
        code to execute;
    }
    else
    {
        if (condition)
        {
            code to execute;
        }
        else
        {
            code to execute;
        }
    }
}
```

خواندن کد بالا سخت است. بهتر است دستورات را به صورت تو رفتگی در داخل بلوک else بنویسید. می‌توانید کد بالا را ساده‌تر کنید :

```
if (condition)
{
    code to execute;
}
else if (condition)
{
    code to execute;
}
else if (condition)
{
    code to execute;
}
else
{
    code to execute;
}
```

حال که نحوه استفاده از دستور `if else` را یاد گرفتید باید بدانید که مانند `else if` نیز به دستور `if` وابسته است. دستور `if` وقتی اجرا می‌شود که اولین دستور `if` اشتباه باشد حال اگر `else if` دستور `if` اشتباه باشد دستور `else` بعدی اجرا می‌شود. و اگر آن نیز اجرا نشود در نهایت دستور `else` اجرا می‌شود. برنامه زیر نحوه استفاده از دستور `if else` را نشان می‌دهد :

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {

        int choice = 2;

        if (choice == 1)
        {
            System.out.print("You might like my black t-shirt.");
        }
        else if (choice == 2)
        {
            System.out.print("You might be a clean and tidy person.");
        }
        else if (choice == 3)
        {
            System.out.print("You might be sad today.");
        }
        else
        {
            System.out.print("Sorry, your favorite color is not in the choices above.");
        }
    }
}
```

You might be a clean and tidy person

خروجی بنامه بالا به متغیر `choice` وابسته است. بسته به اینکه شما چه چیزی انتخاب می‌کنید پیغام‌های مختلفی چاپ می‌شود. اگر عددی که شما تایپ می‌کنید در داخل حالت‌های انتخاب نباشد کد مربوط به بلوک `else` اجرا می‌شود.

استفاده از عملگرهای منطقی

عملگرهای منطقی به شما اجازه می‌دهند که چندین شرط را با هم ترکیب کنید. این عملگرها حداقل دو شرط را درگیر می‌کنند و در آخر یک مقدار بولی را بر می‌گردانند. در جدول زیر برخی از عملگرهای منطقی آمده است :

عملگر	مثال	تلفظ	تأثیر
-------	------	------	-------

مقدار z در صورتی true است که هر دو شرط دو طرف عملگر false باشد. اگر فقط مقدار یکی از شروط مقدارشان باشد مقدار z، false خواهد شد.	$z = (x > 2) \&& (y < 10)$	And	&&
مقدار z در صورتی true است که یکی از دو شرط دو طرف عملگر مقدارشان true باشد. اگر هر دو شرط مقدارشان باشد مقدار z، false خواهد شد.	$z = (x > 2) (y < 10)$	Or	
مقدار z در صورتی true است که مقدار شرط false باشد و در صورتی false است که مقدار شرط true باشد.	$z = !(x > 2)$	Not	!

به عنوان مثال جمله $(x > 2) \&& (y < 10) = z$ را به این صورت بخوانید: "در صورتی مقدار z برابر true است که مقدار x بزرگ‌تر از ۲ و مقدار y کوچک‌تر از ۱۰ باشد در غیر اینصورت false است". این جمله بدین معناست که برای اینکه مقدار کل دستور true باشد باید مقدار همه شروط true باشد. عملگر منطقی (||) OR تأثیر متفاوتی نسبت به عملگر منطقی (&&) AND دارد. نتیجه عملگر منطقی OR برابر true است اگر فقط مقدار یکی از شروط true باشد. و اگر مقدار هیچ یک از شروط true نباشد نتیجه false خواهد شد. می‌توان عملگرهای منطقی AND و OR را با هم ترکیب کرده و در یک عبارت به کار برد مانند :

```
if ( (x == 1) && ( (y > 3) || z < 10) ) 
{
    //do something here
}
```

در اینجا استفاده از پرانتز مهم است چون از آن در گروه بندی شرط‌ها استفاده می‌کنیم. در اینجا ابتدا عبارت $(z < 10) || (y > 3)$ مورد بررسی قرار می‌گیرد (به علت تقدم عملگرها). سپس نتیجه آن بوسیله عملگر AND با نتیجه $(x == 1)$ مقایسه می‌شود. حال باید نحوه استفاده از عملگرهای منطقی در برنامه را مورد بررسی قرار دهیم :

```
1 package myfirstprogram;
2
3 import java.util.Scanner;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {
9         int age;
10        String gender;
```

```

11
12     Scanner input = new Scanner(System.in);
13
14     System.out.print("Enter your age: ");
15     age = input.nextInt();
16
17     System.out.print("Enter your gender (male/female):");
18     gender = input.next();
19
20     if (age > 12 && age < 20)
21     {
22         if (gender == "male")
23         {
24             System.out.println("You are a teenage boy.");
25         }
26         else
27         {
28             System.out.println("You are not a teenage girl.");
29         }
30     }
31     else
32     {
33         System.out.println("You are not a teenager.");
34     }
35 }
36 }
```

```

Enter your age: 18
Enter your gender (male/female): female
You are a teenage girl.
Enter you age: 10
Enter your gender (male/female): male
You are not a teenager.
```

برنامه بالا نحوه استفاده از عملگر منطقی AND را نشان می‌دهد (خط ۲۰). وقتی به دستور `if` می‌رسید (خط ۲۰) برنامه سن شما را چک می‌کند. اگر سن شما بزرگتر از ۱۲ و کوچکتر از ۲۰ باشد (سنتان بین ۱۲ و ۲۰ باشد) یعنی مقدار هر دو `true` باشد، سپس کدهای داخل بلوك `if` اجرا می‌شوند. اگر نتیجه یکی از شروط `false` باشد کدهای داخل بلوك `else` اجرا می‌شود. عملگر AND عاملوند سمت چپ را مرود بررسی قرار می‌دهد. اگر مقدار آن `false` باشد دیگر عاملوند سمت راست را بررسی نمی‌کند و مقدار `false` را بر می‌گرداند. بر عکس عملگر || عاملوند سمت چپ را مورد بررسی قرار می‌دهد و اگر مقدار آن `true` باشد سپس عاملوند سمت راست را نادیده می‌گیرد و مقدار `true` را بر می‌گرداند.

```

if (x == 2 & y == 3)
{
    //Some code here
}

if (x == 2 | y == 3)
```

```
{
    //Some code here
}
```

نکته مهم اینجاست که شما می‌توانید از عملگرهای `&` و `|` به عنوان عملگر بیتی استفاده کنید. تفاوت جزئی این عملگرها وقتی که به عنوان عملگر بیتی به کار می‌روند این است که دو عملوند را بدون در نظر گرفتن مقدار عملوند سمت چپ مورد بررسی قرار می‌دهند. به عنوان مثال حتی اگر مقدار عملوند سمت چپ `false` باشد عملوند سمت چپ به وسیله عملگر بیتی `(&)` AND ارزیابی می‌شود. اگر شرطها را در برنامه ترکیب کنید استفاده از عملگرهای منطقی `&&` و `||` به جای عملگرهای بیتی `(&)` AND و `(|)` OR بهتر خواهد بود.

یکی دیگر از عملگرهای منطقی عملگر `!` NOT است که نتیجه یک عبارت را خنتی یا منفی می‌کند. به مثال زیر توجه کنید:

```
if (!(x == 2))
{
    System.out.println("x is not equal to 2.");
}
```

اگر نتیجه عبارت `x == 2` باشد عملگر! آن را `True` می‌کند.

دستور switch

در جاوا ساختاری به نام `switch` وجود دارد که به شما اجازه می‌دهد که با توجه به مقدار ثابت یک متغیر چندین انتخاب داشته باشید. دستور `switch` معادل دستور `if` تو در تو است با این تفاوت که در دستور `switch` متغیر فقط مقادیر ثابتی از اعداد، رشته‌ها و یا کاراکترها را قبول می‌کند. مقادیر ثابت مقادیری هستند که قابل تغییر نیستند. در زیر نحوه استفاده از دستور `switch` آمده است :

```
switch (testVar)
{
    case compareVal1:
        code to execute if testVar == compareVal1;
        break;
    case compareVal2:
        code to execute if testVar == compareVal2;
        break;
    .
    .
    .
    case compareValN:
        code to execute if testVar == compareValN;
        break;
    default:
        code to execute if none of the values above match the testVar;
        break;
}
```

ابتدا یک مقدار در متغیر `switch` که در مثال بالا `testVar` است قرار می‌دهید. این مقدار با هر یک از عبارت‌های `case` داخل بلوک `switch` مقایسه می‌شود. اگر مقدار متغیر با هر یک از مقادیر موجود در دستورات `case` برابر بود که مربوط به آن `case` اجرا خواهد شد. به این نکته توجه کنید که حتی اگر تعداد خط کدهای داخل دستور `case` از یکی بیشتر باشد نباید از آکولاد استفاده کنیم. آخر هر دستور `case` با کلمه کلیدی `break` تشخیص داده می‌شود که باعث می‌شود برنامه از دستور `switch` خارج شده و دستورات بعد از آن اجرا شوند. اگر این کلمه کلیدی از قلم بیوپتد برنامه با خطأ مواجه می‌شود. دستور `switch` یک بخش `default` دارد. این دستور در صورتی اجرا می‌شود که مقدار متغیر با هیچ یک از مقادیر دستورات `case` برابر نباشد. دستور `default` اختیاری است و اگر از بدنه `switch` حذف شود هیچ اتفاقی نمی‌افتد. مکان این دستور هم مهم نیست اما بر طبق تعریف آن را در پایان دستورات می‌نویسند. به مثالی در مورد دستور `switch` توجه کنید :

```

1 package myfirstprogram;
2
3 import java.util.Scanner;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {
9         Scanner input = new Scanner(System.in);
10
11         int choice;
12
13         System.out.println("What's your favorite pet?");
14         System.out.println("[1] Dog");
15         System.out.println("[2] Cat");
16         System.out.println("[3] Rabbit");
17         System.out.println("[4] Turtle");
18         System.out.println("[5] Fish");
19         System.out.println("[6] Not in the choices");
20         System.out.print("Enter your choice: ");
21
22         choice = input.nextInt();
23
24         switch (choice)
25         {
26             case 1:
27                 System.out.println("Your favorite pet is Dog.");
28                 break;
29             case 2:
30                 System.out.println("Your favorite pet is Cat.");
31                 break;
32             case 3:
33                 System.out.println("Your favorite pet is Rabbit.");
34                 break;
35             case 4:
36                 System.out.println("Your favorite pet is Turtle.");
37                 break;

```

```

38     case 5:
39         System.out.println("Your favorite pet is Fish.");
40         break;
41     case 6:
42         System.out.println("Your favorite pet is not in the choices.");
43         break;
44     default:
45         System.out.println("You don't have a favorite pet.");
46         break;
47     }
48 }
49 }
```

```

What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 2
Your favorite pet is Cat.
What's your favorite pet?
[1] Dog
[2] Cat
[3] Rabbit
[4] Turtle
[5] Fish
[6] Not in the choices

Enter your choice: 99
You don't have a favorite pet.
```

برنامه بالا به شما اجازه انتخاب حیوان مورد علاقه‌تان را می‌دهد. به اسم هر حیوان یک عدد نسبت داده شده است. شما عدد را وارد می‌کنید و این عدد در دستور `switch` با مقادیر `case` مقایسه می‌شود و با هر کدام از آن مقادیر که برابر بود پیغام مناسب نمایش داده خواهد شد. اگر هم با هیچ کدام از مقادیر `case` ها برابر نبود دستور `default` اجرا می‌شود. یکی دیگر از ویژگیهای دستور `switch` این است که شما می‌توانید از دو یا چند `case` برای نشان داده یک مجموعه کد استفاده کنید. در مثال زیر اگر مقدار `number`, ۱، ۲ یا ۳ باشد یک کد اجرا می‌شود. توجه کنید که `case` ها باید پشت سر هم نوشته شوند.

```

switch(number)
{
    case 1:
    case 2:
    case 3:
        System.out.println("This code is shared by three values.");
        break;
}
```

همانطور که قبلًا ذکر شد دستور `switch` معادل دستور `if` تو در تو است. برنامه بالا را به صورت زیر نیز می‌توان نوشت:

```
if (choice == 1)
    System.out.println("Your favorite pet is Dog.");
else if (choice == 2)
    System.out.println("Your favorite pet is Cat.");
else if (choice == 3)
    System.out.println("Your favorite pet is Rabbit.");
else if (choice == 4)
    System.out.println("Your favorite pet is Turtle.");
else if (choice == 5)
    System.out.println("Your favorite pet is Fish.");
else if (choice == 6)
    System.out.println("Your favorite pet is not in the choices.");
else
    System.out.println("You don't have a favorite pet.");
```

کد بالا دقیقاً نتیجه‌ای مانند دستور `switch` دارد. دستور `default` معادل دستور `else` می‌باشد. حال از بین این دو دستور (`switch` و `else`) کدامیک را انتخاب کنیم. از دستور `switch` موقعی استفاده می‌کنیم که مقداری که می‌خواهیم با دیگر مقادیر مقایسه شود ثابت باشد. مثلًا در مثال زیر هیچگاه از `switch` استفاده نکنید.

```
int myNumber = 5;
int x = 5;

switch (myNumber)
{
    case x:
        System.out.println("Error, you can't use variables as a value" +
                           " to be compared in a case statement.");
        break;
}
```

مشاهده می‌کنید که با اینکه مقدار `x` عدد ۵ است و به طور واضح با متغیر `myNumber` مقایسه شده است برنامه خطای دهد چون `x` یک ثابت نیست بلکه یک متغیر است یا به زبان ساده‌تر، قابلیت تغییر را دارد. اگر بخواهیم از `x` استفاده کنیم و برنامه خطای ندهد باید از کلمه `final` به صورت زیر استفاده کنیم.

```
int myNumber = 5;
final int x = 5;

switch (myNumber)
{
    case x:
        System.out.println("Error has been fixed!");
        break;
}
```

از کلمه کلیدی final برای ایجاد ثابت‌ها استفاده می‌شود. توجه کنید که بعد از تعریف یک ثابت نمی‌توان مقدار آن را در طول برنامه

تغییر داد. به یاد داشته باشید که باید ثابت‌ها را حتماً مقداردهی کنید. دستور switch یک مقدار را با مقادیر case ها مقایسه می‌کند و

شما لازم نیست که به شکل زیر مقادیر را با هم مقایسه کنید :

```
switch (myNumber)
{
    case x > myNumber:
        System.out.println("switch statements can't test if a value is less than " +
                           "or greater than the other value.");
    break;
}
```

تکرار

ساختارهای تکرار به شما اجازه می‌دهند که یک یا چند دستور کد را تا زمانی که یک شرط برقرار است تکرار کنید. بدون ساختارهای تکرار

شما مجبورید همان تعداد کدها را بنویسید که بسیار خسته کننده است. مثلاً شما مجبورید ۱۰ بار جمله "Hello World." را تایپ کنید

مانند مثال زیر :

```
System.out.println("Hello World.");
```

البته شما می‌توانید با کپی کردن این تعداد کد را راحت بنویسید ولی این کار در کل کیفیت کدنویسی را پایین می‌آورد. راه بهتر برای

نوشتن کدهای بالا استفاده از حلقه‌ها است. ساختارهای تکرار در جاوا عبارت‌اند از :

while •

do while •

for •

حلقه While

ابتدا بیک شرط را مورد بررسی قرار می‌دهد و تا زمانیکه شرط برقرار باشد کدهای while ابتدا تکرار در جاوا حلقه است.

درون بلوک اجرا می‌شوند. ساختار حلقه while به صورت زیر است:

```
while(condition)
{
    code to loop;
}
```

بررس هیچکدام از کدها را اجرا نمی‌کند. برای متوقف شدن حلقة باید مقادیر داخل حلقة while اصلاح شوند.

به یک متغیر شمارنده در داخل بدن حلقه نیاز داریم. این شمارنده برای آزمایش شرط مورد استفاده قرار می‌گیرد و ادامه یا توقف حلقه به نوعی به آن وابسته است. این شمارنده را در داخل بدن پاید کاهش یا افزایش دهیم. در برنامه زیر نحوه استفاده از حلقه while آمده

است:

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int counter = 1;
8
9         while (counter <= 10)
10        {
11             System.out.println("Hello World!");
12             counter++;
13         }
14     }
15 }
```

برنامه بالا ۱۰ بار پیغام Hello World را چاپ می‌کند. اگر از حلقه در مثال بالا استفاده نمی‌کردیم مجبور بودیم تمام ۱۰ خط را تایپ کنیم.

اجازه دهید که نگاهی به کدهای برنامه فوق بیندازیم. ابتدا در خط ۷ یک متغیر تعریف و از آن به عنوان شمارنده حلقه استفاده شده است. سپس به آن مقدار ۱ را اختصاص می‌دهیم چون اگر مقدار نداشته باشد نمی‌توان در شرط از آن استفاده کرد.

در خط ۹ حلقه while را وارد می‌کنیم. در حلقه while ابتدا مقدار اولیه شمارنده با ۱۰ مقایسه می‌شود که آیا از ۱۰ کمتر است یا با آن برابر است. نتیجه هر بار مقایسه ورود به بدنه حلقه while و چاپ پیغام است. همانطور که مشاهده می‌کنید بعد از هر بار مقایسه مقدار شمارنده یک واحد اضافه می‌شود (خط ۱۲). حلقه تا زمانی تکرار می‌شود که مقدار شمارنده از ۱۰ کمتر باشد.

اگر مقدار شمارنده یک بماند و آن را افزایش ندهیم و یا مقدار شرط هرگز false نشود یک حلقه بین‌هایت به وجود می‌آید. به این نکته توجه کنید که در شرط بالا به جای علامت < از => استفاده شده است. اگر از علامت < استفاده می‌کردیم کد ما ۹ بار تکرار می‌شد چون مقدار اولیه ۱ است و هنگامی که شرط به ۱۰ بررسد false می‌شود چون ۱۰ < ۱۰ نیست. اگر می‌خواهیم یک حلقه بی‌نهایت ایجاد کنید که هیچگاه متوقف نشود باید یک شرط ایجاد کنید که همواره درست (true) باشد.

```
while(true)
{
    //code to loop
}
```

این تکنیک در برخی موارد کارایی دارد و آن زمانی است که شما بخواهید با استفاده از دستورات break و return که در آینده توضیح خواهیم داد از حلقه خارج شوید.

do While حلقه

حلقه do while یکی دیگر از ساختارهای تکرار است. این حلقه بسیار شبیه حلقه while است با این تفاوت که در این حلقه ابتدا کد اجرا می‌شود و سپس شرط مورد بررسی قرار می‌گیرد. ساختار حلقه while do به صورت زیر است :

```
do
{
    code to repeat;
} while (condition);
```

همانطور که مشاهده می‌کنید شرط در آخر ساختار قرار دارد. این بدین معنی است که کدهای داخل بدن حداقل یکبار اجرا می‌شوند.

برخلاف حلقه while که اگر شرط نادرست باشد دستورات داخل بدن اجرا نمی‌شوند. یکی از موارد برتری استفاده از حلقه do while

نسبت به حلقه while زمانی است که شما بخواهید اطلاعاتی از کاربر دریافت کنید. به مثال زیر توجه کنید :

استفاده از while

```
//while version

System.out.print("Enter a number greater than 10: ");
number = input.nextInt();

while(number < 10)
{
    System.out.println("Enter a number greater than 10: ");
    number = input.nextInt();
}
```

استفاده از do while

```
//do while version

do
{
    System.out.println("Enter a number greater than 10: ");
    number = input.nextInt();
}
while(number < 10)
```

مشاهده می‌کنید که از کدهای کمتری در بدن do while نسبت به while استفاده شده است.

حلقه for

یکی دیگر از ساختارهای تکرار حلقة for است. این حلقة عملی شبیه به حلقة while انجام می‌دهد و فقط دارای چند خصوصیت اضافی است. ساختار حلقة for به صورت زیر است :

```
for(initialization; condition; operation)
{
    code to repeat;
}
```

مقدار دهنده اولیه (initialization) اولین مقداری است که به شمارنده حلقة می‌دهیم. شمارنده فقط در داخل حلقة for قابل دسترسی است. شرط (condition) در اینجا مقدار شمارنده را با یک مقدار دیگر مقایسه می‌کند و تعیین می‌کند که حلقة ادامه یابد یا نه. عملگر (operation) که مقدار اولیه متغیر را کاهش یا افزایش می‌دهد. در زیر یک مثال از حلقة for آمده است:

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        for(int i = 1; i <= 10; i++)
        {
            System.out.println("Number " + i);
        }
    }
}
```

```
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
```

برنامه بالا اعداد ۱ تا ۱۰ را با استفاده از حلقه `for` می‌شمارد. ابتدا یک متغیر به عنوان شمارنده تعریف می‌کنیم و آن را با مقدار ۱ مقدار دهی اولیه می‌کنیم. سپس با استفاده از شرط آن را با مقدار ۱۰ مقایسه می‌کنیم که آیا کمتر است یا مساوی؟ توجه کنید که قسمت سوم حلقه (`i++`) فوراً اجرا نمی‌شود. کد اجرا می‌شود و ابتدا رشته `Number` و سپس مقدار جاری `i` یعنی ۱ را چاپ می‌کند. آنگاه یک واحد به مقدار `i` اضافه شده و مقدار `i` برابر ۲ می‌شود و بار دیگر `i` با عدد ۱۰ مقایسه می‌شود و این حلقه تا زمانی که مقدار شرط `true` شود ادامه می‌یابد. حال اگر بخواهید معکوس برنامه بالا را پیاده سازی کنید یعنی اعداد از بزرگ به کوچک چاپ شوند باید به صورت زیر عمل کنید :

```
for (int i = 10; i > 0; i--)
{
    //code omitted
}
```

کد بالا اعداد را از ۱۰ به ۱ چاپ می‌کند (از بزرگ به کوچک). مقدار اولیه شمارنده را ۱۰ می‌دهیم و با استفاده از عملگر کاهش (-) برنامه‌ای که شمارش معکوس را انجام می‌دهد ایجاد می‌کنیم. می‌توان قسمت شرط و عملگر را به صورت‌های دیگر نیز تغییر داد. به عنوان مثال می‌توان از عملگرهای منطقی در قسمت شرط و از عملگرهای تخصیصی در قسمت عملگر افزایش یا کاهش استفاده کرد. همچنین می‌توانید از چندین متغیر در ساختار حلقه `for` استفاده کنید.

```
for (int i = 1, y = 2; i < 10 && y > 20; i++, y -= 2)
{
    //some code here
}
```

```
{}
```

به این نکته توجه کنید که اگر از چندین متغیر شمارنده یا عملگر در حلقه for استفاده می‌کنید باید آن‌ها را با استفاده از کاما از هم جدا کنید. گاهی اوقات با وجود درست بودن شرط می‌خواهیم حلقه متوقف شود. سؤال اینجاست که چطور این کار را انجام دهید؟ با استفاده از کلمه کلیدی break حلقه را متوقف کرده و با استفاده از کلمه کلیدی continue می‌توان بخشی از حلقه را رد کرد و به مرحله بعد رفت.

برنامه زیر نحوه استفاده از continue و break را نشان می‌دهد :

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("Demonstrating the use of break\n");
8
9         for (int x = 1; x < 10; x++)
10        {
11            if (x == 5)
12                break;
13
14            System.out.println("Number " + x);
15        }
16
17        System.out.println("\nDemonstrating the use of continue\n");
18
19        for (int x = 1; x < 10; x++)
20        {
21            if (x == 5)
22                continue;
23
24            System.out.println("Number "+ x);
25        }
26    }
27 }
```

Demonstrating the use of break.

Number 1
Number 2
Number 3
Number 4

Demonstrating the use of continue.

Number 1
Number 2
Number 3
Number 4
Number 6
Number 7

```
Number 8
Number 9
```

در این برنامه از حلقه `for` برای نشان دادن کاربرد دو کلمه کلیدی فوق استفاده شده است اگر به جای `for` از حلقهای `while` و `do...while` استفاده می‌شود نتیجه یکسانی به دست می‌آمد. همانطور که در شرط برنامه (خط ۱۱) آمده است وقتی که مقدار `x` به عدد ۵ رسید سپس دستور `break` اجرا شود (خط ۱۲). حلقه بلافصله متوقف می‌شود حتی اگر شرط ۱۰ `< x` برقرار باشد. از طرف دیگر در خط ۲۲ حلقه `for` فقط برای یک تکرار خاص متوقف شده و سپس ادامه می‌یابد. وقتی مقدار `x` برابر ۵ شود حلقه از ۵ رد شده و مقدار ۵ را چاپ نمی‌کند و بقیه مقادیر چاپ می‌شوند.

آرایه‌ها

آرایه نوعی متغیر است که لیستی از آدرس‌های مجموعه‌ای از داده‌های هم نوع را در خود ذخیره می‌کند. تعریف چندین متغیر از یک نوع برای هدفی یکسان بسیار خسته کننده است. مثلاً اگر بخواهید صد متغیر از نوع اعداد صحیح تعریف کرده و از آن‌ها استفاده کنید. مطمئناً تعریف این همه متغیر بسیار کسالت آور و خسته کننده است. اما با استفاده از آرایه می‌توان همه آن‌ها را در یک خط تعریف کرد. در زیر راهی ساده برای تعریف یک آرایه نشان داده شده است :

```
datatype[] arrayName = new datatype[length];
```

نوع داده‌هایی را نشان می‌دهد که آرایه در خود ذخیره می‌کند. کروشه که بعد از نوع داده قرار می‌گیرد و نشان دهنده استفاده از آرایه است. که نام آرایه را نشان می‌دهد. هنگام نامگذاری آرایه بهتر است که نام آرایه نشان دهنده نوع آرایه باشد. به عنوان مثال برای نامگذاری آرایه‌ای که اعداد را در خود ذخیره می‌کند از کلمه `number` استفاده کنید. طول آرایه که به کمپایلر می‌گوید شما قصد دارید چه تعداد داده یا مقدار را در آرایه ذخیره کنید. از کلمه کلیدی `new` هم برای اختصاص فضای حافظه به اندازه طول آرایه استفاده می‌شود. برای تعریف یک آرایه که ۵ مقدار از نوع اعداد صحیح در خود ذخیره می‌کند باید به صورت زیر عمل کنیم :

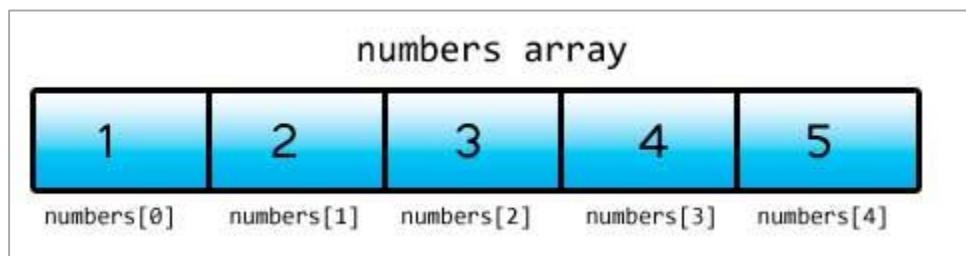
```
int[] numbers = new int[5];
```

در این مثال ۵ آدرس از فضای حافظه کامپیوتر شما برای ذخیره ۵ مقدار رزرو می‌شود. حال چطور مقادیرمان را در هر یک از این آدرس‌ها ذخیره کنیم؟ برای دسترسی و اصلاح مقادیر آرایه از اندیس یا مکان آن‌ها استفاده می‌شود.

```
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
```

```
numbers[3] = 4;
numbers[4] = 5;
```

اندیس یک آرایه از صفر شروع شده و به یک واحد کمتر از طول آرایه ختم می‌شود. به عنوان مثال شما یک آرایه ۵ عضوی دارید، اندیس آرایه از ۰ تا ۴ می‌باشد چون طول آرایه ۵ است پس ۰-۴ برابر است با ۰-۴. این بدان معناست که اندیس ۰ نشان دهنده اولین عضو آرایه است و اندیس ۱ نشان دهنده دومین عضو و الی آخر. برای درک بهتر مثال بالا به شکل زیر توجه کنید :



به هر یک از اجزاء آرایه و اندیس‌های داخل کروشه توجه کنید. کسانی که تازه شروع به برنامه نویسی کرده‌اند معمولاً در گذاشتن اندیس دچار اشتباه می‌شوند و مثلاً ممکن است در مثال بالا اندیس‌ها را از ۱ شروع کنند. اگر بخواهید به یکی از اجزای آرایه با استفاده از اندیسی دسترسی پیدا کنید که در محدوده اندیس‌های آرایه شما نباشد با پیغام خطای `ArrayIndexOutOfBoundsException` مواجه می‌شوید و بدین معنی است که شما آدرسی را می‌خواهید که وجود ندارد. یک راه بسیار ساده‌تر برای تعریف آرایه به صورت زیر است :

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

به سادگی و بدون احتیاج به کلمه کلیدی `new` می‌توان مقادیر را در داخل آکولاد قرار داد. کمپایلر به صورت اتوماتیک با شمارش مقادیر طول آرایه را تشخیص می‌دهد.

دستیابی به مقادیر آرایه با استفاده از حلقه for

در زیر مثالی در مورد استفاده از آرایه‌ها آمده است. در این برنامه ۵ مقدار از کاربر گرفته شده و میانگین آن‌ها حساب می‌شود:

```

1 package myfirstprogram;
2
3 import java.util.Scanner;
4 import java.text.MessageFormat;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
```

```

10     Scanner input = new Scanner(System.in);
11
12     int[] numbers = new int[5];
13     int total = 0;
14     double average;
15
16     for (int i = 0; i < numbers.length; i++)
17     {
18         System.out.print("Enter a number: ");
19         numbers[i] = input.nextInt();
20     }
21     for (int i = 0; i < numbers.length; i++)
22     {
23         total += numbers[i];
24     }
25
26     average = total / (double)numbers.length;
27
28     System.out.println(MessageFormat.format("Average = {0}", average));
29
30 }
```

```

Enter a number: 90
Enter a number: 85
Enter a number: 80
Enter a number: 87
Enter a number: 92
Average = 86
```

در خط ۱۲ یک آرایه تعریف شده است که می‌تواند ۵ عدد صحیح را در خود ذخیره کند. خطوط ۱۳ و ۱۴ متغیرهایی تعریف شده‌اند که از آن‌ها برای محاسبه میانگین استفاده می‌شود. توجه کنید که مقدار اولیه total صفر است تا از بروز خطا هنگام اضافه شدن مقدار به آن جلوگیری شود. در خطوط ۱۶ تا ۲۰ حلقه for برای تکرار و گرفتن ورودی از کاربر تعریف شده است. از خاصیت طول (length) آرایه برای تشخیص تعداد اجزای آرایه استفاده می‌شود. اگر چه می‌توانستیم به سادگی در حلقه for مقدار ۵ را برای شرط قرار دهیم ولی استفاده از خاصیت طول آرایه کار راحت‌تری است و می‌توانیم طول آرایه را تغییر دهیم و شرط حلقه for با تغییر جدید هماهنگ می‌شود. در خط ۱۹ ورودی دریافت شده از کاربر با استفاده از متد () nextInt دریافت و در آرایه ذخیره می‌شود. اندیس استفاده شده در number (خط ۱۹) مقدار نegative جاری در حلقه است. برای مثال در ابتدای حلقه مقدار -۱ صفر است بنابراین وقتی در خط ۱۹ اولین داده از کاربر گرفته می‌شود، اندیس آن برابر صفر می‌شود. در تکرار بعدی -۱ یک واحد اضافه می‌شود و در نتیجه در خط ۱۹ و بعد از ورود دومین داده توسط کاربر اندیس آن برابر یک می‌شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می‌باید. در خطوط ۲۱-۲۴ از حلقه for دیگر برای اندیس آن برابر یک می‌شود. این حالت تا زمانی که شرط در حلقه for برقرار است ادامه می‌باید. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به عنوان اندیس دسترسی به مقدار هر یک از داده‌های آرایه استفاده شده است. در این حلقه نیز مانند حلقه قبل از مقدار متغیر شمارنده به استفاده می‌کنیم.

هر یک از اجزای عددی آرایه به متغیر `total` اضافه می‌شوند. بعد از پایان حلقه می‌توانیم میانگین اعداد را حساب کنیم (خط ۲۶). مقدار `total` را بر تعداد اجزای آرایه (تعداد عدها) تقسیم می‌کنیم. برای دسترسی به تعداد اجزای آرایه می‌توان از خاصیت `length` آرایه `total` استفاده کرد. توجه کنید که در اینجا ما مقدار خاصیت `length` را به نوع `double` تبدیل کرده‌ایم بنابراین نتیجه عبارت یک مقدار از نوع `double` خواهد شد و دارای بخش کسری می‌باشد. حال اگر عملوندهای تقسیم را به نوع `double` تبدیل نکنیم نتیجه تقسیم یک عدد از نوع صحیح خواهد شد و دارای بخش کسری نیست. خط ۲۸ مقدار میانگین را در صفحه نمایش چاپ می‌کند. طول آرایه بعد از مقدار دهی نمی‌تواند تغییر کند. به عنوان مثال اگر یک آرایه را که شامل ۵ جز است مقدار دهی کنید دیگر نمی‌توانید آن را مثلاً به ۱۰ جز تغییر اندازه دهید. البته تعداد خاصی از کلاس‌ها مانند آرایه‌ها عمل می‌کنند و توانایی تغییر تعداد اجزای تشکیل دهنده خود را دارند. آرایه‌ها در برخی شرایط بسیار پر کاربرد هستند و تسلط شما بر این مفهوم و اینکه چطور از آن‌ها استفاده کنید بسیار مهم است.

حلقه foreach

حلقه `foreach` یکی دیگر از ساختارهای تکرار در جاوا می‌باشد که مخصوصاً برای آرایه‌ها، لیست‌ها و مجموعه‌ها طراحی شده است. حلقه `foreach` با هر بار گردش در بین اجزاء، مقادیر هر یک از آن‌ها را در داخل یک متغیر موقتی قرار می‌دهد و شما می‌توانید بواسطه این متغیر به مقادیر دسترسی پیدا کنید. در زیر نحوه استفاده از حلقه `foreach` آمده است :

```
for (datatype temporaryVar : array)
{
    code to execute;
}
```

متغیری است که مقادیر اجزای آرایه را در خود نگهداری می‌کند. `temporaryVar` باید دارای نوع باشد تا بتواند مقادیر آرایه را در خود ذخیره کند. به عنوان مثال آرایه شما دارای اعدادی از نوع صحیح باشد باید نوع متغیر موقتی از نوع اعداد صحیح باشد یا هر نوع دیگری که بتواند اعداد صحیح را در خود ذخیره کند مانند `double` یا `long`. سپس علامت دو نقطه (`:`) و بعد از آن نام آرایه را می‌نویسیم. در زیر نحوه استفاده از حلقه `foreach` آمده است :

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int[] numbers = { 1, 2, 3, 4, 5 };
8
9         for (int n : numbers)
```

```

10      {
11          System.out.println(n);
12      }
13  }
14 }

1
2
3
4
5

```

در برنامه آرایه‌ای با ۵ جزء تعریف شده و مقادیر ۱ تا ۵ در آن‌ها قرار داده شده است (خط ۷). در خط ۹ حلقه `foreach` شروع می‌شود. ما یک متغیر موقتی تعریف کرده‌ایم که اعداد آرایه را در خود ذخیره می‌کند. در هر بار تکرار از حلقه `foreach` متغیر موقتی `n`، مقادیر عددی را از آرایه استخراج می‌کند. حلقه `foreach` مقادیر اولین تا آخرین جزء آرایه را در اختیار ما قرار می‌دهد.

حلقه `foreach` برای دریافت هر یک از مقادیر آرایه کاربرد دارد. بعد از گرفتن مقدار یکی از اجزای آرایه، مقدار متغیر موقتی را چاپ می‌کنیم. حلقه `foreach` ما را قادر می‌سازد که به داده‌ها دسترسی یابیم و یا آن‌ها را بخوانیم و اصلاح کنیم. برای درک این مطلب در مثال زیر مقدار هر یک از اجزا آرایه افزایش یافته است :

```

int[] numbers = { 1, 2, 3 };

for(int n : numbers)
{
    n++;
    System.out.println(n);
}

2
3
4
5
6

```

آرایه‌های چند بعدی

آرایه‌های چند بعدی، آرایه‌هایی هستند که برای دسترسی به هر یک از عناصر آن‌ها باید از چندین اندیس استفاده کنیم. یک آرایه چند بعدی را می‌توان مانند یک جدول با تعدادی ستون و ردیف تصور کنید. با افزایش اندیس‌ها اندازه ابعاد آرایه نیز افزایش می‌باید و آرایه‌های چند بعدی با بیش از دو اندیس به وجود می‌آیند. نحوه ایجاد یک آرایه با دو بعد به صورت زیر است :

```
datatype[][][] arrayName = new datatype[lengthX][lengthY];
```

و یک آرایه سه بعدی به صورت زیر ایجاد می شود :

```
datatype[][][] arrayName = new datatype[lengthX][lengthY][lengthZ];
```

می توان یک آرایه با تعداد زیادی بعد ایجاد کرد به شرطی که هر بعد دارای طول مشخصی باشد. به دلیل اینکه آرایه های سه بعدی یا آرایه های با بیشتر از دو بعد بسیار کمتر مورد استفاده قرار می گیرند. اجازه بدھید که در این درس بر روی آرایه های دو بعدی تمرکز کنیم. در تعریف این نوع آرایه ابتدا نوع آرایه یعنی اینکه آرایه چه نوعی از انواع داده را در خود ذخیره می کند را، مشخص می کنیم. سپس دو جفت کروشه قرار می دهیم. به تعداد کروشه ها توجه کنید. اگر آرایه ما دو بعدی است باید ۲ جفت کروشه و اگر سه بعدی است باید ۳ جفت کروشه قرار دهیم. سپس یک نام برای آرایه انتخاب کرده و بعد تعریف آنرا با گذاشتن کلمه new، نوع داده و طول هر بعد آن کامل می کنیم. در یک آرایه دو بعدی برای دسترسی به هر یک از عناصر به دو مقدار نیاز داریم یکی مقدار X و دیگری مقدار Y که مقدار X نشان می کنیم. در یک آرایه دو بعدی برای دسترسی اگر ما آرایه دو بعدی را به صورت جدول در نظر بگیریم. یک آرایه سه بعدی را دهنده ردیف و مقدار Y نشان دهنده ستون آرایه است البته اگر ما آرایه دو بعدی را به صورت جدول در نظر بگیریم. یک آرایه سه بعدی را می توان به صورت یک مکعب تصور کرد که دارای سه بعد است و X طول، Y عرض و Z ارتفاع آن است. یک مثال از آرایه دو بعدی در زیر آمده است :

```
int[][] numbers = new int[3][5];
```

کد بالا به کمپایلر می گوید که فضای کافی به عناصر آرایه اختصاص بده (در این مثال ۱۵ خانه). در شکل زیر مکان هر عنصر در یک آرایه دو بعدی نشان داده شده است.



مقدار ۳ را به `x` اختصاص می‌دهیم چون ۳ سطر و مقدار ۵ را به ۷ چون ۵ ستون داریم اختصاص می‌دهیم. چطور یک آرایه چند بعدی را مقدار دهی کنیم؟ چند راه برای مقدار دهی به آرایه‌ها وجود دارد. یک راه این است که مقادیر عناصر آرایه را در همان زمان تعریف آرایه،

مشخص کنیم :

```
datatype[][] arrayName = { { r0c0, r0c1, ... r0cX },
                           { r1c0, r1c1, ... r1cX },
                           .
                           .
                           { rYc0, rYc1, ... rYcX } };
```

به عنوان مثال :

```
int[][] numbers = {
    { 1, 2, 3, 4, 5 },
    { 6, 7, 8, 9, 10 },
    { 11, 12, 13, 14, 15 }
};
```

و یا می‌توان مقدار دهی به عناصر را به صورت دستی انجام داد مانند :

```
array[0][0] = value;
array[0][1] = value;
array[0][2] = value;
array[1][0] = value;
array[1][1] = value;
array[1][2] = value;
array[2][0] = value;
array[2][1] = value;
array[2][2] = value;
```

همانطور که مشاهده می‌کنید برای دسترسی به هر یک از عناصر در یک آرایه دو بعدی به سادگی می‌توان از اندیس‌های `X` و `Y` و یک جفت کروشه مانند مثال استفاده کرد.

گردش در میان عناصر آرایه‌های چند بعدی

گردش در میان عناصر آرایه‌های چند بعدی نیاز به کمی دقت دارد. یکی از راههای آسان استفاده از حلقه `foreach` و یا حلقه `for` تو در تو است. اجازه دهید ابتدا از حلقه `foreach` استفاده کنیم.

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
```

```

6   {
7     int[][] numbers = { { 1, 2, 3, 4, 5 },
8                           { 6, 7, 8, 9, 10 },
9                           { 11, 12, 13, 14, 15 } ,
10                         };
11
12   for (int[] number : numbers)
13   {
14     for (int num : number)
15     {
16       System.out.print(num + " ");
17     }
18   }
19 }
20 }
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

مشاهده کردید که گرددش در میان مقادیر عناصر یک آرایه چند بعدی چقدر راحت است. به وسیله حلقه `foreach` نمی‌توانیم انتهای ردیف‌ها را مشخص کنیم. برنامه زیر نشان می‌دهد که چطور از حلقه `for` برای خواندن همه مقادیر آرایه و تعیین انتهای ردیف‌ها استفاده کنید.

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5   public static void main(String[] args)
6   {
7     int[][] numbers = { { 1, 2, 3, 4, 5 },
8                           { 6, 7, 8, 9, 10 },
9                           { 11, 12, 13, 14, 15 } ,
10                         };
11
12   for (int row = 0; row < numbers.length; row++)
13   {
14     for (int col = 0; col < numbers[row].length; col++)
15     {
16       System.out.print(numbers[row][col] + " ");
17     }
18
19     //Go to the next line
20     System.out.println();
21   }
22 }
23 }
```

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

همانطور که درمثال بالا نشان داده شده است با استفاده از یک حلقه ساده `for` نمیتوان به مقادیر دسترسی یافت بلکه به یک حلقه تو در تو نیاز داریم. در اولین حلقه `for` (خط ۱۲) یک متغیر تعریف شده است که در میان ردیفهای آرایه (`row`) گردش میکند. این حلقه تا زمانی ادامه مییابد که مقدار ردیف کمتر از طول اولین بعد باشد. در این مثال از خاصیت `length` کلاس `Array` استفاده کردهایم. این خاصیت طول آرایه را در یک بعد خاص نشان میدهد. به عنوان مثال برای به دست آوردن طول اولین بعد آرایه که همان تعداد ردیفها میباشد از دستور `numbers.length` استفاده کردهایم.

در داخل اولین حلقه `for` دیگری تعریف شده است (خط ۱۴). در این حلقه یک شمارنده برای شمارش تعداد ستون‌های (`columns`) هر ردیف تعریف شده است و در شرط داخل آن بار دیگر از خاصیت `length` استفاده شده است، ولی این بار به نوعی دیگر از آن استفاده میکنیم. همانطور که مشاهده میکنید ابتدا نام آرایه را نوشتهایم و سپس یک اندیس به آن اختصاص دادهایم و بعد از خاصیت `length` استفاده نمودهایم :

```
numbers[row].length
```

استفاده از `row` به عنوان اندیس باعث میشود که به عنوان مثال وقتی که مقدار ردیف (`row`) صفر باشد، حلقه دوم از [۰][۰] تا [۳][۰] اجرا شود. سپس مقدار هر عنصر از آرایه را با استفاده از حلقه نشان میدهیم، اگر مقدار ردیف (`row`) برابر ۰ و مقدار ستون (`col`) برابر ۰ باشد مقدار عنصری که در ستون ۱ و ردیف ۱ (`numbers[0][0]`) قرار دارد، نشان داده خواهد شد که در مثال بالا عدد ۱ است.

بعد از اینکه دومین حلقه تکرار به پایان رسید، فوراً دستورات بعد از آن اجرا خواهند شد، که در اینجا دستور (`System.out.println()`) که به برنامه اطلاع میدهد که به خط بعد برود. سپس حلقه با اضافه کردن یک واحد به مقدار `row` این فرایند را دوباره تکرار میکند. سپس دومین حلقه `for` اجرا شده و مقادیر دومین ردیف نمایش داده میشود. این فرایند تا زمانی اجرا میشود که مقدار `row` کمتر از طول اولین بعد باشد. حال بباید آنچه را از قبل یاد گرفتهایم در یک برنامه به کار ببریم. این برنامه نمره چهار درس مربوط به سه دانش آموز را از ما میگیرد و معدل سه دانش آموز را حساب میکند.

```

1 package myfirstprogram;
2
3 import java.util.Scanner;
4 import java.text.MessageFormat;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         Scanner input = new Scanner(System.in);
11

```

```

12 double[][] studentGrades = new double[3][4];
13 double total;
14
15 for (int student = 0; student < studentGrades.length; student++)
16 {
17     total = 0;
18
19     System.out.println(MessageFormat.format("Enter grades for Student {0}", student + 1));
20
21     for (int grade = 0; grade < studentGrades[student].length; grade++)
22     {
23         System.out.print(MessageFormat.format("Enter Grade #{0}: ", grade + 1));
24         studentGrades[student][grade] = input.nextDouble();
25         total += studentGrades[student][grade];
26     }
27
28     System.out.print(
29         MessageFormat.format("Average is {0}", (total / studentGrades[student].length)));
30     System.out.println();
31 }
32 }
33 }
```

Enter grades for Student 1

Enter Grade #1: 92

Enter Grade #2: 87

Enter Grade #3: 89

Enter Grade #4: 95

Average is 90.75

Enter grades for Student 2

Enter Grade #1: 85

Enter Grade #2: 85

Enter Grade #3: 86

Enter Grade #4: 87

Average is 85.75

Enter grades for Student 3

Enter Grade #1: 90

Enter Grade #2: 90

Enter Grade #3: 90

Enter Grade #4: 90

Average is 90.00

در برنامه بالا یک آرایه چند بعدی از نوع `double` تعریف شده است (خط ۱۲). همچنین یک متغیر به نام `total` تعریف می‌کنیم که مقدار محاسبه شده معدل هر دانش آموز را در آن قرار دهیم. حال وارد حلقه `for` تو در تو می‌شویم (خط ۱۵). در اولین حلقه `for` یک متغیر به نام `sudent` برای تشخیص پایه درسی هر دانش آموز تعریف کردہ‌ایم. از خاصیت `length` هم برای تشخیص تعداد دانش آموزان استفاده شده است. وارد بدن حلقه `for` می‌شویم. در خط ۱۷ مقدار متغیر `total` را برابر صفر قرار می‌دهیم. بعداً مشاهده می‌کنید که چرا این کار را انجام دادیم. سپس برنامه یک پیغام را نشان می‌دهد و از شما می‌خواهد که شماره دانش آموز را وارد کنید (`student + 1`). عدد ۱ را به `student` اضافه کردہ‌ایم تا به جای نمایش ۰، `Student 1` شروع شود، تا طبیعی‌تر به نظر برسد. سپس به

دومین حلقه `for` در خط ۲۱ می‌رسیم. در این حلقه یک متغیر شمارنده به نام `grade` تعریف می‌کنیم که طول دومین بعد آرایه را با استفاده از `studentGrades[student].length` به دست می‌آورد. این طول تعداد نمراتی را که برنامه از سؤال می‌کند را نشان می‌دهد. برنامه چهار نمره مربوط به دانش آموز را می‌گیرد. هر وقت که برنامه یک نمره را از کاربر دریافت می‌کند، نمره به متغیر `total` اضافه می‌شود. وقتی همه نمره‌ها وارد شدند، متغیر `total` هم جمع همه نمرات را نشان می‌دهد. در خط ۲۸ معدل دانش آموز نشان داده می‌شود. معدل از تقسیم کردن `total` (جمع) بر تعداد نمرات به دست می‌آید. از `studentGrades[student].length` هم برای به دست آوردن تعداد نمرات استفاده می‌شود.

آرایه دندانه دار

آرایه دندانه دار یا `jagged array` آرایه‌ای چند بعدی است که دارای سطرهای با طول متغیر می‌باشد. نمونه ساده‌ای از آرایه‌های چند بعدی، آرایه‌های مستطیلی است که تعداد ستون‌ها و سطرهای آن‌ها برابر است. اما آرایه‌های دندانه دار دارای سطرهای (آرایه‌های) با طول متفاوت می‌باشند. بنابر این آرایه‌های دندانه دار را می‌توان آرایه‌ای از آرایه‌ها فرض کرد. دستور نوشتن این نوع آرایه‌ها به صورت زیر است :

```
datatype[][] arrayName;
```

ابتدا `datatype` که نوع آرایه است و سپس چهار کروشه باز و بسته و بعد از آن نام آرایه را می‌نویسیم. مقداردهی به این آرایه‌ها کمی گیج کننده است. به مثال زیر توجه کنید :

```
int[][] myArrays = new int[3][];
myArrays[0] = new int[3];
myArrays[1] = new int[5];
myArrays[2] = new int[2];
```

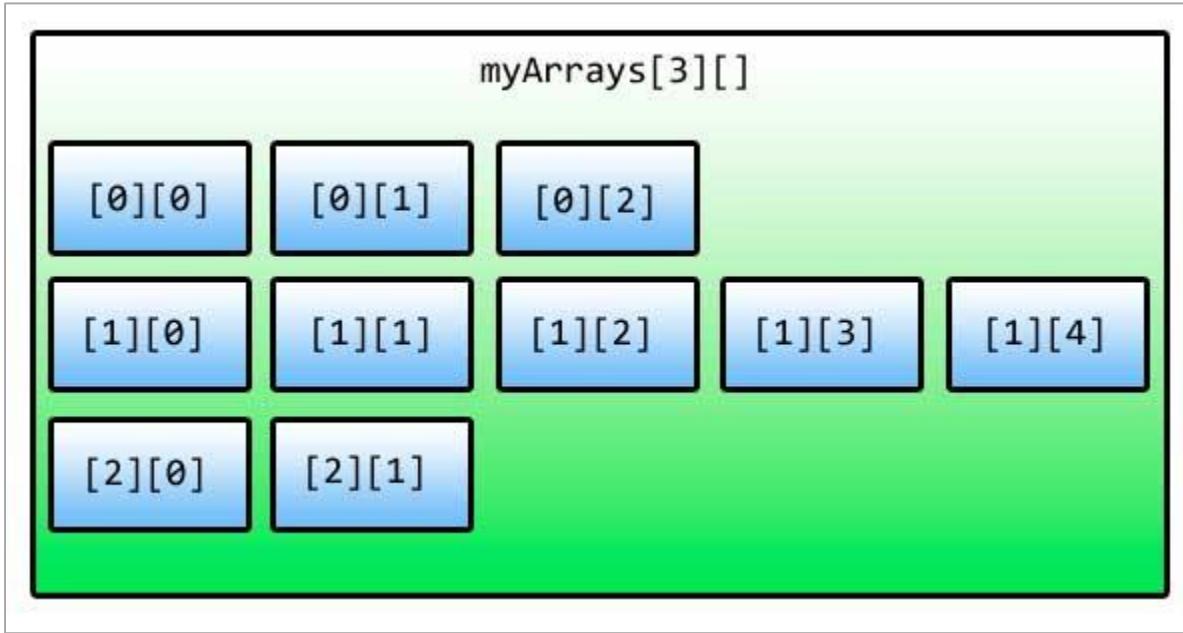
در کد بالا سه آرایه تعریف شده است که اندیس آن‌ها از صفر شروع می‌شود. اعداد ۳ و ۵ و ۲ هم به تعداد عناصری که هر کدام از آن‌ها در خود می‌توانند جای دهنده اشاره دارند. برای مقداردهی هر آرایه هم باید ابتدا اندیس آرایه و سپس اندیس عناصر آن را بنویسیم. مثلاً مقداردهی اولین عنصر اولین آرایه مثال بالا به صورت زیر عمل می‌کنیم :

```
myArrays[0][0] = 1;
```

و برای مثلاً دومین عنصر دومین آرایه هم به صورت زیر :

```
myArrays[1][1] = 4;
```

شکل زیر هم اندیس عناصر آرایه‌ای که در بالا تعریف کردہ‌ایم را نشان می‌دهد:



با توجه به توضیحاتی که داده شد می‌توان عناصر آرایه‌ای که در ابتدای درس ایجاد کردیم را به صورت زیر مقداردهی کرد:

```
myArrays[0][0] = 1;
myArrays[0][1] = 2;
myArrays[0][2] = 3;

myArrays[1][0] = 5;
myArrays[1][1] = 4;
myArrays[1][2] = 3;
myArrays[1][3] = 2;
myArrays[1][4] = 1;

myArrays[2][0] = 11;
myArrays[2][1] = 22;
```

یک روش بهتر برای مقداردهی آرایه‌های دندانه دار به صورت زیر است که در آن می‌توان طول سطرها را هم مشخص نکرد:

```
int[][] myArrays = {{1,2,3}, {5,4,3,2,1}, {11,22}};
```

برای دسترسی به مقدار عناصر یک آرایه دندانه دار باید اندیس سطر و ستون آن را در اختیار داشته باشیم (array[row][column]):

```
System.out.println(myArrays[1][2]);
```

نمی‌توان از حلقه `foreach` برای دسترسی به عناصر آرایه دندانه دار استفاده کرد :

```
for(int array : myArrays)
{
    System.out.println(array);
}
```

اگر از حلقه `foreach` استفاده کنیم با خطأ مواجه می‌شویم چون عناصر این نوع آرایه‌ها، آرایه هستند نه عدد یا رشته یا ... برای حل این مشکل باید نوع متغیر موقتی (`array`) را تغییر داده و از حلقه `foreach` دیگری برای دسترسی به مقادیر استفاده کرد.

```
for(int[] array : myArrays)
{
    for(int number : array)
    {
        System.out.println(number);
    }
}
```

همچنین می‌توان از یک حلقه `for` تو در تو به صورت زیر استفاده کرد :

```
for (int row = 0; row < myArrays.length; row++)
{
    for (int col = 0; col < myArrays[row].length; col++)
    {
        System.out.println(myArrays[row][col]);
    }
}
```

در اولین حلقه از `length` برای به دست آوردن تعداد سطرها (که همان آرایه‌های یک بعدی هستند) و در دومین حلقه از `length` برای به دست آوردن عناصر سطر جاری استفاده می‌شود.

متدها

متدها به شما اجازه می‌دهند که یک رفتار یا وظیفه را تعریف کنید و مجموعه‌ای از کدها هستند که در هر جای برنامه می‌توان از آن‌ها استفاده کرد. متدها دارای آرگومان‌هایی هستند که وظیفه متدها را مشخص می‌کنند. متدها در داخل کلاس تعریف می‌شود. نمی‌توان یک متدها را در داخل متدها دیگر تعریف کرد. وقتی که شما در برنامه یک متدها را صدا می‌زنید برنامه به قسمت تعریف متدها و کدهای آن را اجرا می‌کند. در جاوا متدها وجود دارد که نقطه آغاز هر برنامه است و بدون آن برنامه‌ها نمی‌دانند با ید از کجا شروع شوند، این متدها `main()` نام دارد. پارامترها همان چیزهایی هستند که متدها منتظر دریافت آنها هستند. آرگومان‌ها مقادیری هستند که به پارامترها ارسال می‌شوند.

گاهی اوقات دو کلمه پارامتر و آرگومان به یک منظور به کار می‌روند. ساده‌ترین ساختار یک متدها است :

```
returnType MethodName()
{
    code to execute;
}
```

به برنامه ساده زیر توجه کنید. در این برنامه از یک متده از یک متده برای چاپ یک پیغام در صفحه نمایش استفاده شده است :

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     static void PrintMessage()
6     {
7         System.out.println("Hello World!");
8     }
9
10    public static void main(String[] args)
11    {
12        PrintMessage();
13    }
14 }
```

Hello World!

در خطوط ۵-۸ یک متده تعریف کرده ایم. مکان تعریف آن در داخل کلاس مهم نیست. به عنوان مثال می توانید آن را زیر متده تعریف کنید. می توان این متده را در داخل متده دیگر صدا زد (فراخوانی کرد). متده دیگر ما در اینجا متده main() است که می توانیم در داخل آن نام متده که برای چاپ یک پیغام تعریف کرده ایم (یعنی متده PrintMessage()) را صدا بزنیم. متده main() به صورت static تعریف شده است. برای اینکه بتوان از متده PrintMessage() در داخل متده main() استفاده کنیم باید آن را به صورت static تعریف کنیم. کلمه static به طور ساده به این معناست که می توان از متده استفاده کرد بدون اینکه از کلاس نمونه های ساخته شود. متده main() همواره باید به صورت static تعریف شود چون برنامه فوراً و بدون نمونه سازی از کلاس از آن استفاده می کند. وقتی به مبحث برنامه نویسی شیء گرا رسیدید به طور دقیق کلمه static مورد بحث قرار می گیرد. برنامه class (مثال بالا) زمانی اجرا می شود که برنامه دو متده را که تعریف کرده ایم را اجرا کند و متده main() به صورت static تعریف شود. در باره این کلمه کلیدی در درس های آینده مطالب بیشتری می آموزیم.

در تعریف متده بالا بعد از کلمه static کلمه کلیدی void آمده است که نشان دهنده آن است که متده مقدار برگشتی ندارد. در درس آینده در مورد مقدار برگشتی از یک متده و استفاده از آن برای اهداف مختلف توضیح داده خواهد شد. نام متده main() است.

به این نکته توجه کنید که در نامگذاری متدهای از روش پاسکال (حروف اول هر کلمه بزرگ نوشته می‌شود) استفاده کردہ‌ایم. این روش نامگذاری قراردادی است و می‌توان از این روش استفاده نکرد، اما پیشنهاد می‌شود که از این روش برای تشخیص متدها استفاده کنید.

بهتر است در نامگذاری متدها از کلماتی استفاده شود که کاران متدهای را مشخص می‌کند مثلاً نام‌هایی مانند GoToBed یا OpenDoor همچنین به عنوان مثال اگر مقدار برگشتی متدهای می‌توانید اسم متدهای خود را به صورت یک کلمه سوالی انتخاب کنید، مانند IsTeenager یا IsLeapyear، ولی از گذاشتن علامت سوال در آخر اسم متدهای خودداری کنید. دو پرانتزی که بعد از نام می‌آید نشان دهنده آن است که نام متعلق به یک متدهای است. در این مثال در داخل پرانتزها هیچ چیزی نوشته نشده چون پارامتری ندارد. در درس‌های آینده در مورد متدهای بیشتر توضیح می‌دهیم.

بعد از پرانتزها دو آکولاد قرار می‌دهیم که بدنه متدهای را تشکیل می‌دهد و کدهایی را که می‌خواهیم اجرا شوند را در داخل این آکولادها می‌نویسیم. در داخل متدهای main() که در خط ۱۲ ایجاد کرده‌ایم را صدا می‌زنیم. برای صدا زدن یک متدهای کافیست نام آن را نوشته و بعد از نام پرانتزها را قرار دهیم. اگر متدهای پارامتر باشد باید شما آراگومانها را به ترتیب در داخل پرانتزها قرار دهید. در این مورد نیز در درس‌های آینده توضیح بیشتری می‌دهیم. با صدا زدن یک متدهای داخل بدنه آن اجرا می‌شوند. برای اجرای متدهای PrintMessage() برنامه از متدهای main() به محل تعریف متدهای PrintMessage() می‌روند. مثلاً وقتی ما متدهای main() را در خط ۱۳ صدا می‌زنیم برنامه از خط ۱۲ به خط ۷، یعنی جایی که متدهای PrintMessage() تعریف شده می‌روند. اکنون ما یک متدهای Class داریم و همه متدهای این برنامه می‌توانند آن را صدا بزنند.

مقدار برگشتی از یک متدهای

متدهای می‌توانند مقدار برگشتی از هر نوع داده‌ای داشته باشند. این مقادیر می‌توانند در محاسبات یا به دست آوردن یک داده مورد استفاده قرار بگیرند. در زندگی روزمره فرض کنید که کارمند شما یک متدهای داشته باشد و شما او را صدا می‌زنید و از او می‌خواهید که کار یک سند را به پایان برساند. سپس از او می‌خواهید که بعد از اتمام کارش سند را به شما تحویل دهد. سند همان مقدار برگشتی متدهای است. نکته مهم در مورد یک متدهای مقدار برگشتی و نحوه استفاده شما از آن است. برگشت یک مقدار از یک متدهای آسان است. کافیست در تعریف متدهای روش زیر عمل کنید :

```
returnType MethodName()
{
    return value;
}
```

در اینجا نوع داده‌ای مقدار برگشتی را مشخص می‌کند (`int`, `bool`....). در داخل بدنه متده کلمه کلیدی `returnType` و بعد از آن یک مقدار یا عبارتی که نتیجه آن یک مقدار است را می‌نویسیم. نوع این مقدار برگشتی باید از انواع ساده بوده و در هنگام نامگذاری متده و قبل از نام متده ذکر شود. اگر متده ما مقدار برگشتی نداشته باشد باید از کلمه `void` قبل از نام متده استفاده کنیم. مثال زیر یک متده دارای مقدار برگشتی است را نشان می‌دهد.

```

1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 public class MyFirstProgram
6 {
7     static int CalculateSum()
8     {
9         int firstNumber = 10;
10        int secondNumber = 5;
11
12        int sum = firstNumber + secondNumber;
13
14        return sum;
15    }
16
17    public static void main(String[] args)
18    {
19        int result = CalculateSum();
20
21        System.out.println(MessageFormat.format("Sum is {0}.", result));
22    }
23 }
```

Sum is 15.

همانطور که در خط ۷ مثال فوق مشاهده می‌کنید هنگام تعریف متده از کلمه `void` به جای `int` استفاده کردہ‌ایم که نشان دهنده آن است که متده دارای مقدار برگشتی از نوع اعداد صحیح است. در خطوط ۹ و ۱۰ دو متغیر تعریف و مقدار دهی شده‌اند.

توجه کنید که این متغیرها، متغیرهای محلی هستند. و این بدان معنی است که این متغیرها در سایر متدها مانند متده `main()` قابل دسترسی نیستند و فقط در متده که در آن تعریف شده‌اند قابل استفاده هستند. در خط ۱۲ جمع دو متغیر در متغیر `sum` قرار می‌گیرد. در خط ۱۴ مقدار برگشتی `sum` توسط دستور `return` فراخوانی می‌شود. در داخل متده `main()` یک متغیر به نام `result` در خط ۱۹ تعریف می‌کنیم و متده `CalculateSum()` را فراخوانی می‌کنیم.

متند () CalculateSum() مقدار ۱۵ را برابر میگرداند که در داخل متغیر result ذخیره میشود. در خط ۲۱ مقدار ذخیره شده در متغیر result چاپ میشود. متند که در این مثال ذکر شد متند کاربردی و مفیدی نیست. با وجودیکه کدهای زیادی در متند بالا نوشته شده ولی همیشه مقدار برگشتی ۱۵ است، در حالیکه میتوانستیم به راحتی یک متغیر تعریف کرده و مقدار ۱۵ را به آن اختصاص دهیم. این متند در صورتی کارآمد است که پارامترهایی به آن اضافه شود که در درس‌های آینده توضیح خواهیم داد. هنگامی که میخواهیم در داخل یک متند از دستور if یا switch استفاده کنیم باید تمام کدها دارای مقدار برگشتی باشند. برای درک بهتر این مطلب به مثال زیر توجه

: کنید

```

1 package myfirstprogram;
2
3 import java.util.Scanner;
4 import java.text.MessageFormat;
5
6 public class MyFirstProgram
7 {
8     static int GetNumber()
9     {
10         Scanner input = new Scanner(System.in);
11
12         int number;
13
14         System.out.print("Enter a number greater than 10: ");
15         number = input.nextInt();
16
17         if (number > 10)
18         {
19             return number;
20         }
21         else
22         {
23             return 0;
24         }
25     }
26
27     public static void main(String[] args)
28     {
29         int result = GetNumber();
30
31         System.out.println(MessageFormat.format("Result = {0}.", result));
32     }
33 }
```

```

Enter a number greater than 10: 11
Result = 11
Enter a number greater than 10: 9
Result = 0
```

در خطوط ۸-۲۵ یک متده با نام `() GetNumber` تعریف شده است که از کاربر یک عدد بزرگتر از ۱۰ را می‌خواهد. اگر عدد وارد شده توسط کاربر درست نباشد متده مقدار صفر را بر می‌گرداند. و اگر قسمت `if` و یا دستور `return` را از آن حذف کنیم در هنگام اجرای برنامه با پیغام خطای مواجه می‌شویم.

چون اگر شرط دستور `if` نادرست باشد (کاربر مقداری کمتر از ۱۰ را وارد کند) برنامه به قسمت `else` می‌رود تا مقدار صفر را برگرداند و چون قسمت `else` حذف شده است برنامه با خطای مواجه می‌شود و همچنین اگر دستور `return` حذف شود چون برنامه نیاز به مقدار برگشتی دارد پیغام خطای می‌دهد.

پارامتر و آرگومان

پارامترها داده‌های خامی هستند که متده آن‌ها را پردازش می‌کند و سپس اطلاعاتی را که به دنبال آن هستید در اختیار شما قرار می‌دهد. فرض کنید پارامترها مانند اطلاعاتی هستند که شما به یک کارمند می‌دهید که بر طبق آن‌ها کارش را به پایان برساند. یک متده می‌تواند هر تعداد پارامتر داشته باشد. هر پارامتر می‌تواند از انواع مختلف داده باشد. در زیر یک متده با `N` پارامتر نشان داده شده است :

```
returnType MethodName(datatype param1, datatype param2, ... datatype paramN)
{
    code to execute;
}
```

پارامترها بعد از نام متده و بین پرانتزها قرار می‌گیرند. بر اساس کاری که متده انجام می‌دهد می‌توان تعداد پارامترهای زیادی به متده اضافه کرد. بعد از فراخوانی یک متده باید آرگومان‌های آن را نیز تأمین کنید. آرگومان‌ها مقادیری هستند که به پارامترها اختصاص داده می‌شوند. ترتیب ارسال آرگومان‌ها به پارامترها مهم است. عدم رعایت ترتیب در ارسال آرگومان‌ها باعث به وجود آمدن خطای منطقی و خطای زمان اجرا می‌شود. اجازه بدھید که یک مثال بزنیم :

```
1 package myfirstprogram;
2
3 import java.util.Scanner;
4 import java.text.MessageFormat;
5
6 public class MyFirstProgram
7 {
8     static int CalculateSum(int number1, int number2)
9     {
10         return number1 + number2;
11     }
12
13     public static void main(String[] args)
14     {
15         Scanner input = new Scanner(System.in);
```

```

16     int num1, num2;
17
18     System.out.print("Enter the first number: ");
19     num1 = input.nextInt();
20     System.out.print("Enter the second number: ");
21     num2 = input.nextInt();
22
23     System.out.println(MessageFormat.format("Sum = {0}", CalculateSum(num1, num2)));
24 }

```

```

Enter the first number: 10
Enter the second number: 5
Sum = 15

```

در برنامه بالا یک متد به نام (`CalculateSum()` خطوط ۸-۱۱) تعریف شده است که وظیفه آن جمع مقدار دو عدد است. چون این متد مقدار دو عدد صحیح را با هم جمع می‌کند پس نوع برگشتی ما نیز باید `int` باشد. متد دارای دو پارامتر است که اعداد را به آن‌ها ارسال می‌کنیم. به نوع داده‌ای پارامترها توجه کنید. هر دو پارامتر یعنی `number1` و `number2` مقادیری از نوع اعداد صحیح (`int`) دریافت می‌کنند. در بدنه متد دستور `return` نتیجه جمع دو عدد را بر می‌گرداند. در داخل متد (`main()` برنامه از کاربر دو مقدار را درخواست می‌کند و آن‌ها را داخل متغیرها قرار می‌دهد. حال متد را که آرگومان‌های آن را آماده کرده‌ایم فراخوانی می‌کنیم. مقدار `num1` به پارامتر اول و مقدار `num2` به پارامتر دوم ارسال می‌شود. حال اگر مکان دو مقدار را هنگام ارسال به متد تغییر دهیم (یعنی مقدار `num2` به پارامتر اول و مقدار `num1` به پارامتر دوم ارسال شود) هیچ تغییری در نتیجه متد ندارد چون جمع خاصیت جایه جایی دارد.

فقط به یاد داشته باشید که باید ترتیب ارسال آرگومان‌ها هنگام فراخوانی متد دقیقاً با ترتیب قرار گیری پارامترها تعریف شده در متد مطابقت داشته باشد. بعد از ارسال مقادیر `۱۰` و `۵` به پارامترها، پارامترها آن‌ها را دریافت می‌کنند. به این نکته نیز توجه کنید که نام پارامترها طبق قرارداد به شیوه کوهان شتری یا `camelCasing` (حرف اول دومین کلمه بزرگ نوشته می‌شود) نوشته می‌شود. در داخل بدنه متد (خط `۱۰`) دو مقدار با هم جمع می‌شوند و نتیجه به متد فراخوان (متدهای `CalculateSum()` را فراخوانی می‌کند) ارسال می‌شود. در درس آینده از یک متغیر برای ذخیره نتیجه محاسبات استفاده می‌کنیم ولی در اینجا مشاهده می‌کنید که می‌توان به سادگی نتیجه جمع را نشان داد (خط `۱۰`). در داخل متد (`main()`) از ما دو عدد که قرار است با هم جمع شوند درخواست می‌شود.

در خط `۲۳` متد (`CalculateSum()`) را فراخوانی می‌کنیم و دو مقدار صحیح به آن ارسال می‌کنیم. دو عدد صحیح در داخل متد با هم جمع شده و نتیجه آن‌ها برگردانده می‌شود. مقدار برگشت داده شده از متد به وسیله متد (`format()` از کلاس `MessageFormat` نمایش داده می‌شود (خط `۲۳`). در برنامه زیر یک متد تعریف شده است که دارای دو پارامتر از دو نوع داده‌ای مختلف است:

```

1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 public class MyFirstProgram
6 {
7     static void ShowMessageAndNumber(String message, int number)
8     {
9         System.out.println(message);
10        System.out.println(MessageFormat.format("Number = {0}", number));
11    }
12
13    public static void main(String[] args)
14    {
15        ShowMessageAndNumber("Hello World!", 100);
16    }
17}

```

```
Hello World!
Number = 100
```

در مثال بالا یک متده تعریف شده است که اولین پارامتر آن مقداری از نوع رشته و دومین پارامتر آن مقداری از نوع `int` دریافت می‌کند. متده سادگی دو مقداری که به آن ارسال شده است را نشان می‌دهد. در خط ۱۵ متده را اول با یک رشته و سپس یک عدد خاص فراخوانی می‌کنیم. حال اگر متده به صورت زیر فراخوانی می‌شد :

```
ShowMessageAndNumber(100, "Welcome to Gimme JAVA!");
```

در برنامه خطابه وجود می‌آمد چون عدد ۱۰۰ به پارامتری از نوع رشته و رشته `Hello World!` به پارامتری از نوع اعداد صحیح ارسال می‌شد. این نشان می‌دهد که ترتیب ارسال آرگومان‌ها به پارامترها هنگام فراخوانی متده مهم است. به مثال ۱ توجه کنید در آن مثال دو عدد از نوع `int` به پارامترها ارسال کردیم که ترتیب ارسال آن‌ها چون هردو پارامتر از یک نوع بودند مهم نبود. ولی اگر پارامترهای متده دارای اهداف خاصی باشند ترتیب ارسال آرگومان‌ها مهم است.

```

void ShowPersonStats(int age, int height)
{
    System.out.println(MessageFormat.format("Age = {0}", age));
    System.out.println(MessageFormat.format("Height = {0}", height));
}

//Using the proper order of arguments
ShowPersonStats(20, 160);

//Acceptable, but produces odd results
ShowPersonStats(160, 20);

```

در مثال بالا نشان داده شده است که حتی اگر متدهای دو آرگومان با یک نوع داده‌ای قبول کند باز هم بهتر است ترتیب بر اساس تعریف پارامترها رعایت شود. به عنوان مثال در اولین فراخوانی متدهای بالا اشکالی به چشم نمی‌آید چون سن شخص ۲۰ و قد او ۱۶ سانتی متر است. اگر آرگومان‌ها را به ترتیب ارسال نکنیم سن شخص ۱۶ و قد او ۲۰ سانتی متر می‌شود که به واقعیت نزدیک نیست. دانستن مبانی مقادیر برگشتی و ارسال آرگومان‌ها باعث می‌شود که شما متدهای کارآمدتری تعریف کنید. تکه کد زیر نشان می‌دهد که شما حتی می‌توانید مقدار برگشتی از یک متدهای اولیه را به عنوان آرگومان به متدهای دیگر ارسال کنید.

```
int MyMethod()
{
    return 5;
}

void AnotherMethod(int number)
{
    System.out.println(number);
}
// Codes skipped for demonstration

AnotherMethod(MyMethod());
```

چون مقدار برگشتی متدهای MyMethod عدد ۵ است و به عنوان آرگومان به متدهای AnotherMethod ارسال می‌شود خروجی کد بالا هم عدد ۵ است.

ارسال آرگومان به روش مقدار

ارسال آرگومان‌ها به روش مقدار بدان معناست که شما یک کپی از مقدار متغیر را ارسال می‌کنید نه اصل متغیر یا ارجاع به آن را. در این حالت وقتی که آرگومان ارسال شده را در داخل متدهای اصلاح می‌کنیم مقدار اصلی آرگومان در خارج از متدهای تغییر نمی‌کند. اجازه دهید که ارسال با مقدار آرگومان را با یک مثال توضیح دهیم:

```
1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 public class MyFirstProgram
6 {
7     static void ModifyNumberVal(int number)
8     {
9         number += 10;
10        System.out.println(
11            MessageFormat.format("Value of number inside method is {0}.", number));
12    }
13
14    public static void main(String[] args)
15    {
16        int num = 5;
17        System.out.println(MessageFormat.format("num = {0}\n", num));
18    }
19}
```

```

18     System.out.println("Passing num by value to method ModifyNumberVal() ...");
19     ModifyNumberVal(num);
20     System.out.println(
21         MessageFormat.format("Value of num after exiting the method is {0}", num));
22 }
23
24 }

num = 5

Passing num by value to method ModifyNumberVal() ...
Value of number inside method is 15.
Value of num after exiting the method is 5.

```

در برنامه بالا متدى تعریف شده است که کار آن اضافه کردن عدد ۱۰ به مقداری است که به آنها ارسال می‌شود (خطوط ۱۱-۱۷). این متددارای یک پارامتر است که نیاز به یک مقدار آرگومان (از نوع int) دارد. وقتی که متدد را صدای زنیم و آرگومانی به آن اختصاص میدهیم (خط ۱۹)، کپی آرگومان به پارامتر متدد ارسال می‌شود. بنابراین مقدار اصلی متغیر خارج از متدد هیچ ارتباطی به پارامتر متدد ندارد. سپس مقدار ۱۰ را به متغیر پارامتر (number) اضافه کرده و نتیجه را چاپ می‌کنیم. برای اثبات اینکه متغیر num هیچ تغییری نکرده است مقدار آن را یکبار قبل از ارسال به متدد (خط ۱۶) و بار دیگر بعد از ارسال به متدد (خط ۲۰) چاپ کرده و مشاهده می‌کنیم که تغییری نکرده است.

ارسال آرایه به عنوان آرگومان

می‌توان آرایه‌ها را به عنوان آرگومان به متدد ارسال کرد. ابتدا شما باید پارامترهای متدد را طوری تعریف کنید که آرایه دریافت کنند. به مثال زیر توجه کنید.

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     static void TestArray(int[] numbers)
6     {
7         for(int number : numbers)
8         {
9             System.out.println(number);
10        }
11    }
12
13    public static void main(String[] args)
14    {
15        int[] array = { 1, 2, 3, 4, 5 };
16        TestArray(array);
17    }
18 }

```

```

1
2

```

```
3
4
5
```

مشاهده کردید که به سادگی می‌توان با گذاشتن کروشه بعد از نوع داده‌ای پارامتر یک متد ایجاد کرد که پارامتر آن، آرایه دریافت می‌کند.

وقتی متد در خط ۱۶ فراخوانی می‌شود، آرایه را فقط با استفاده از نام آن و بدون استفاده از اندیس ارسال می‌کنیم. پس آرایه‌ها به روش ارجاع به متدها ارسال می‌شوند. در خطوط ۷-۱۰ از حلقه `foreach` برای دسترسی به اجزای اصلی آرایه که به عنوان آرگومان به متد ارسال کرده‌ایم استفاده می‌کنیم. در زیر نحوه ارسال یک آرایه به روش ارجاع نشان داده شده است.

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     static void IncrementElements(int[] numbers)
6     {
7         for (int i = 0; i < numbers.length; i++)
8         {
9             numbers[i]++;
10        }
11    }
12
13    public static void main(String[] args)
14    {
15        int[] array = { 1, 2, 3, 4, 5 };
16
17        IncrementElements(array);
18
19        for (int num : array)
20        {
21            System.out.println(num);
22        }
23    }
24 }
```

```
2
3
4
5
6
```

برنامه بالا یک متد را نشان می‌دهد که یک آرایه را دریافت می‌کند و به هر یک از عناصر آن یک واحد اضافه می‌کند. به این نکته توجه کنید که از حلقه `foreach` نمی‌توان برای افزایش مقادیر آرایه استفاده کنیم چون این حلقه برای خواندن مقادیر آرایه مناسب است نه اصلاح آن‌ها. در داخل متد ما مقادیر هر یک از اجزای آرایه را افزایش داده‌ایم. سپس از متد خارج شده و نتیجه را نشان می‌دهیم. مشاهده می‌کنید که هر یک از مقادیر اصلی متد هم اصلاح شده‌اند. راه دیگر برای ارسال آرایه به متد، مقدار دهی مستقیم به متد فراخوانی شده است. به عنوان مثال :

```
IncrementElements( new int[] { 1, 2, 3, 4, 5 } );
```

در این روش ما آرایه‌ای تعریف نمی‌کنیم بلکه مجموعه‌ای از مقادیر را به پارامتر ارسال می‌کنیم که آن‌ها را مانند آرایه قبول کند. از آنجاییکه در این روش آرایه‌ای تعریف نکرده‌ایم نمی‌توانیم در متدهای Main() نتیجه را چاپ کنیم. اگر از چندین پارامتر در متدهای استفاده می‌کنید، همیشه برای هر یک از پارامترهایی که آرایه قبول می‌کنند از یک جفت کروشه استفاده کنید. به عنوان مثال :

```
void MyMethod(int[] param1, int param2)
{
    //code here
}
```

به پارامترهای متدهای بالا توجه کنید، پارامتر اول (param1) آرگومانی از جنس آرایه قبول می‌کند ولی پارامتر دوم (param2) یک عدد صحیح. حال اگر پارامتر دوم (param2) هم آرایه قبول می‌کرد باید برای آن هم از کروشه استفاده می‌کردیم :

```
void MyMethod(int[] param1, int[] param2)
{
    //code here
}
```

محدوده متغیر

متدها در جاوا دارای محدوده هستند. محدوده یک متغیر به شما می‌گوید که در کجا برنامه می‌توان از متغیر استفاده کرد و یا متغیر قابل دسترسی است. به عنوان مثال متغیری که در داخل یک متده تعریف می‌شود فقط در داخل بدنه متده قابل دسترسی است. می‌توان دو متغیر با نام یکسان در دو متده مختلف تعریف کرد. برنامه زیر این ادعا را اثبات می‌کند :

```
1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 public class MyFirstProgram
6 {
7     static void DemonstrateScope()
8     {
9         int number = 5;
10
11         System.out.println(MessageFormat.format(
12             "number inside method DemonstrateScope() = {0}", number));
13     }
14
15     public static void main(String[] args)
16     {
17         int number = 10;
```

```
19     DemonstrateScope();  
20  
21     System.out.println(MessageFormat.format(  
22         "number inside the Main method() = {0}", number));  
23     }  
24 }
```

مشاهده می‌کنید که حتی اگر ما دو متغیر با نام یکسان تعریف کنیم که دارای محدوده‌های متفاوتی هستند، می‌توان به هر کدام از آن‌ها مقادیر مختلفی اختصاص داد. متغیر تعریف شده در داخل متدهای Main در خط ۹ هیچ ارتباطی به متغیر داخل متدهای DemonstrateScope در خط ۱۶ ندارد. وقتی به مبحث کلاس‌ها رسیدیم در این باره بیشتر توضیح خواهیم داد.

آرگومان‌های متغیر (VarArgs)

آرگومان‌های متغیر (VarArgs)، در ۵ جا اضافه شده است و به شما اجازه می‌دهد تعداد آرگومان‌های دلخواه (متغیر) و البته همنوع را به متدهای ارسال کنید. برای ایجاد متدهای که به تعداد دلخواه یارامتر دریافت کند از علامت سه نقطه (...) به صورت زیر استفاده می‌شود:

```
public static void MethodName(int ... argument)
{
    // method body
}
```

له مثا، زد توهه کند :

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     static int CalculateSum(int ... numbers)
6     {
7         int total = 0;
8
9         for (int number : numbers)
10        {
11            total += number;
12        }
13
14        return total;
15    }
16
17    public static void main(String[] args)
18    {
19        System.out.println("1 + 2 + 3 = " + CalculateSum(1, 2, 3));
20        System.out.println("1 + 2 + 3 + 4 = " + CalculateSum(1, 2, 3, 4));
21        System.out.println("1 + 2 + 3 + 4 + 5 = " + CalculateSum(1, 2, 3, 4, 5));
22    }
23 }
```

```

22     }
23 }

1 + 2 + 3 = 6
1 + 2 + 3 + 4 = 10
1 + 2 + 3 + 4 + 5 = 15

```

همانطور که در مثال بالا مشاهده می‌کنید یک متده با نام `CalculateSum()` در خط ۳ تعریف شده است. برای اینکه این متده تعداد دلخواه پارامتر دریافت کند، از علامت سه نقطه (...) بعد از نوع داده‌ای پارامتر آن استفاده شده است. در اصل کلمه `numbers` یک آرایه است، که وقتی ما آرگومان‌ها را به متده ارسال می‌کنیم در این آرایه ذخیره می‌شوند. حال متده را سه بار با تعداد مختلف آرگومانها فراخوانی می‌کنیم و سپس با استفاده از حلقه `foreach` این آرگومانها را جمع و به متده فراخوان برگشت می‌دهیم. وقتی از چندین پارامتر در یک متده استفاده می‌کنید فقط یکی از آنها باید دارای علامت سه نقطه (...) بوده و همچنین از لحاظ مکانی باید آخرين پارامتر باشد. اگر این پارامتر (پارامتری که دارای سه نقطه است) در آخر پارامترهای دیگر قرار نگیرد و یا از چندین پارامتر سه نقطه دار استفاده کنید با خطا مواجه می‌شوید. به مثالهای اشتباه و درست زیر توجه کنید :

```

void SomeFunction(int ... x, int ... y) //ERROR
void SomeFunction(int ... x, int y, int z) //ERROR
void SomeFunction(int x, int y, int ... z) //Correct

```

سربارگذاری متدها

سربارگذاری متدها به شما اجازه می‌دهد که چندین متده با نام یکسان تعریف کنید که دارای امضا و تعداد پارامترهای مختلف هستند. برنامه از روی آرگومان‌هایی که شما به متده ارسال می‌کنید به صورت خودکار تشخیص می‌دهد که کدام متده را فراخوانی کرده‌اید یا کدام متده نظر شماست. امضا یک متده نشان دهنده ترتیب و نوع پارامترهای آن است. به مثال زیر توجه کنید :

```
void MyMethod(int x, double y, string z)
```

که امضا یک متده بالا (`MyMethod(int, double, string)`) می‌باشد. به این نکته توجه کنید که نوع برگشتی و نام پارامترها شامل امضا یک متده نمی‌شوند. در مثال زیر نمونه‌ای از سربارگذاری متده آمده است.

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     static void ShowMessage(double number)

```

```

6   {
7     System.out.println("Double version of the method was called.");
8   }
9
10  static void ShowMessage(int number)
11  {
12    System.out.println("Integer version of the method was called.");
13  }
14
15  public static void main(String[] args)
16  {
17    ShowMessage(9.99);
18    ShowMessage(9);
19  }
20

```

```

Double version of the method was called.
Integer version of the method was called.

```

در برنامه بالا دو متدهای مشابه تعریف شده‌اند. اگر سربارگذاری متدهای مشابه را در صورتی که فراخوانی می‌شوند لازم باشد. رازی در نوع پارامترهای متدهای مشابه است. کمپایلر بین دو یا چند متدهای مشابه در صورتی که پارامترهای متفاوتی داشته باشند. وقتی یک متدهای مشابه را فراخوانی می‌کنیم، متدهای مشابه را تشخیص می‌دهد. در فرق می‌گذارد که پارامترهای متفاوتی داشته باشند. وقتی یک متدهای مشابه را فراخوانی می‌کنیم، متدهای مشابه را در نتیجه ارسال کرده‌ایم در اینجا (خطوط ۵-۸) ShowMessage() از نوع double را به متدهای مشابه int (خط ۱۸) ارسال کردیم. در بار دوم که متدهای مشابه int را به متدهای مشابه double ارجاع می‌نماییم. در اینجا (خط ۱۷) ShowMessage() از نوع int را به متدهای مشابه double ارجاع می‌نماییم. معنای اصلی این است که سربارگذاری متدهای مشابه را در صورتی که پارامترهای متفاوتی داشته باشند. وقتی یک متدهای مشابه را فراخوانی می‌کنیم، متدهای مشابه را در نتیجه ارسال کرده‌ایم.

هدف اصلی از سربارگذاری متدهای مشابه است که بتوان چندین متدهای مشابه را تعریف کرد تعداد زیادی از متدهای مشابه را فراخوانی می‌شوند مانند متدهای `System.out.println()` از کلاس `out`. قبل از مشاهده کردید که این متدهای مشابه را آرگومان از نوع رشته دریافت کند و آن را نمایش دهد، و در حالت دیگر می‌توانند دو یا چند آرگومان قبول کند.

بازگشت (Recursion)

بازگشت فرایندی است که در آن متدهای مشابه خود را فراخوانی می‌کنند تا زمانی که به یک مقدار مورد نظر برسد. بازگشت یک مبحث پیچیده در برنامه نویسی است و تسلط به آن کار راحتی نیست. به این نکته هم توجه کنید که بازگشت باید در یک نقطه متوقف شود و گرنه برای بی‌نهایت بار، متدهای مشابه را فراخوانی می‌کند. در این درس یک مثال ساده از بازگشت را برای شما توضیح می‌دهیم. فاکتوریل یک عدد صحیح مثبت ($n!$) شامل حاصل ضرب همه اعداد مثبت صحیح کوچکتر یا مساوی آن می‌باشد. به فاکتوریل عدد ۵ توجه کنید.

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

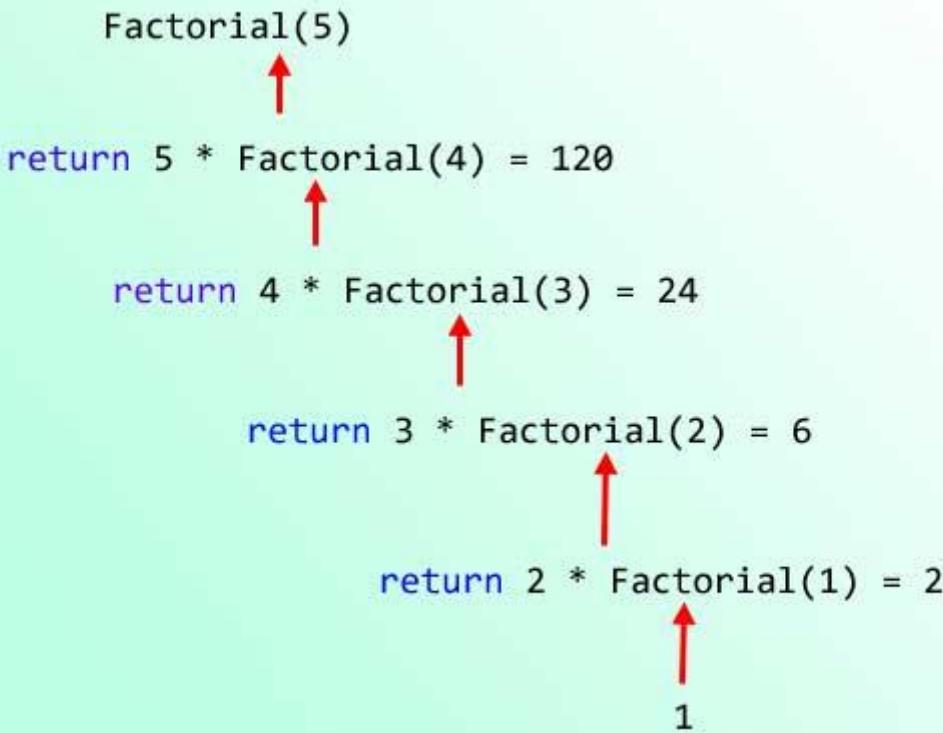
بنابراین برای ساخت یک متدهای بازگشتی باید به فکر توقف آن هم باشیم. بر اساس توضیح بالاگشت، فاکتوریل فقط برای اعداد مثبت صحیح است. کوچکترین عدد صحیح مثبت ۱ است. در نتیجه از این مقدار برای متوقف کردن بازگشت استفاده می‌کنیم.

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     static long Factorial(int number)
6     {
7         if (number == 1)
8             return 1;
9
10        return number * Factorial(number - 1);
11    }
12
13    public static void main(String[] args)
14    {
15        System.out.println(Factorial(5));
16    }
17 }
```

120

متدهای بازگشتی را بر می‌گرداند چون محاسبه فاکتوریل می‌تواند خیلی بزرگ باشد. متدهای آرگومان که یک عدد است و می‌تواند در محاسبه مورد استفاده قرار گیرد را می‌پذیرد. در داخل متدهای دستور `if` می‌نویسیم و در خط ۷ می‌گوییم که اگر آرگومان ارسال شده برابر ۱ باشد سپس مقدار ۱ را برگردان در غیر اینصورت به خط بعد برو. این شرط باعث توقف تکرارها نیز می‌شود. در خط ۱۰ مقدار جاری متغیر `number` در عددی یک واحد کمتر از خودش (`1 - number`) ضرب می‌شود. در این خط متدهای `Factorial` خود را فراخوانی می‌کند و آرگومان آن در این خط همان `1 - number` است. مثلاً اگر مقدار جاری `number` برابر ۱۰ باشد، یعنی اگر ما بخواهیم فاکتوریل عدد ۱۰ را به دست بیاوریم آرگومان متدهای `Factorial` در اولین ضرب ۹ خواهد بود. فرایند ضرب تا زمانی ادامه می‌یابد که آرگومان ارسال شده با عدد ۱ برابر نشود. شکل زیر فاکتوریل عدد ۵ را نشان می‌دهد.



کد بالا را به وسیله یک حلقه for نیز می‌توان نوشت.

```

factorial = 1;

for ( int counter = number; counter >= 1; counter-- )
    factorial *= counter

```

این کد از کد معادل بازگشته آن آسان‌تر است. از بازگشت در زمینه‌های خاصی در علوم کامپیوتر استفاده می‌شود. استفاده از بازگشت حافظه زیادی اشغال می‌کند پس اگر سرعت برای شما مهم است از آن استفاده نکنید.

شمارش (Enumeration)

شمارش راهی برای تعریف داده‌هایی است که می‌توانند مقادیر محدودی که شما از قبل تعریف کرده‌اید را بپذیرند. به عنوان مثال شما می‌خواهید یک متغیر تعریف کنید که فقط مقادیر جهت (جغرافیایی) مانند north, east, west, south را در خود ذخیره کند. ابتدا یک enumeration تعریف می‌کنید و برای آن یک اسم انتخاب کرده و بعد از آن تمام مقادیر ممکن که می‌توانند در داخل بدنه آن قرار بگیرند تعریف می‌کنید. به نحوه تعریف یک enumeration توجه کنید:

```
enum enumName
{
    value1,
    value2,
    value3,
    .
    .
    .
    valueN
}
```

ابتدا کلمه کلیدی `enum` و سپس نام آن را به کار می‌بریم. در جاوا برای نامگذاری `enumeration` از روش پاسکال استفاده کنید. در بدنه `enum` مقادیری وجود دارد که برای هر کدام یک نام در نظر گرفته شده است و به وسیله کاما از هم جدا شده‌اند. به یک مثال توجه کنید :

```
enum Direction
{
    North,
    East,
    South,
    West
}
```

به نحوه استفاده از `enumeration` در یک برنامه جاوا توجه کنید.

```
1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 enum Direction
6 {
7     North,
8     East,
9     South,
10    West
11 }
12
13 public class MyFirstProgram
14 {
15     public static void main(String[] args)
16     {
17         Direction myDirection;
18
19         myDirection = Direction.North;
20
21         System.out.println(MessageFormat.format("Direction: {0}", myDirection));
22     }
23 }
```

Direction: North

ابتدا enumeration را در خطوط ۵-۱۱ تعریف می‌کنیم. توجه کنید که enumeration را خارج از کلاس قرار داده‌ایم. این کار باعث می‌شود که enumeration در سراسر برنامه در دسترس باشد. می‌توان enumeration را در داخل کلاس هم تعریف کرد ولی در این صورت فقط در داخل کلاس قابل دسترس است.

```
public class MyFirstProgram
{
    enum Direction
    {
        //Code omitted
    }

    public static void main(String[] args)
    {
        //Code omitted
    }
}
```

برنامه را ادامه می‌دهیم. در داخل بدن enumeration نام چهار جهت جغرافیایی وجود دارد (خطوط ۷-۱۰). در خط ۱۷ یک متغیر تعریف شده است که مقدار یک جهت را در خود ذخیره می‌کند. نحوه تعریف آن به صورت زیر است :

```
enumType variableName;
```

در اینجا نوع داده شمارشی (مثلاً `Direction` یا مسیر) می‌باشد و `variableName` نیز نامی است که برای آن انتخاب کرده‌ایم که در مثال قبل `myDirection` است. سپس یک مقدار به متغیر `myDirection` اختصاص می‌دهیم (خط ۱۹). برای اختصاص یک مقدار به صورت زیر عمل می‌کنیم :

```
variable = enumType.value;
```

ابدأ نوع Enumeration سپس علامت نقطه و بعد مقدار آن (مثلاً `North`) را می‌نویسیم. می‌توان یک متغیر را فوراً، به روش زیر مقدار دهی کرد :

```
Direction myDirection = Direction.North;
```

حال در خط ۲۱ با استفاده از `println()` مقدار `myDirection` را چاپ می‌کنیم. `enum` ها مانند کلاس‌ها و اینترفیس‌ها از انواع ارجاعی به حساب می‌آیند و بنابر این می‌توانند دارای سازنده و فیلد و متده باشند. به هر عضو از یک `enum` می‌توان یک عدد اختصاص داد. به مثال زیر توجه کنید :

```

1 package myfirstprogram;
2
3 enum Direction
4 {
5     North(3),
6     East(5),
7     South(7),
8     West(4);
9
10    private final int directionindex;
11
12    Direction(int index)
13    {
14        this.directionindex = index;
15    }
16
17    public int GetDirectionIndex()
18    {
19        return this.directionindex;
20    }
21
22}
23
24 public class MyFirstProgram
25 {
26     public static void main(String[] args)
27     {
28         Direction myDirection = Direction.East;
29         System.out.println(myDirection.GetDirectionIndex());
30     }
31
32}

```

5

همانطور که در کد بالا مشاهده می‌کنید یک نوع شمارشی با نام `Direction` در خطوط ۲۱-۳۱ تعریف کرده‌ایم. به مقادیر این نوع شمارشی در خطوط ۵-۸ مقادیری را اختصاص داده‌ایم. در خط ۱۰ یک فیلد با نام `directionindex` تعریف کرده‌ایم تا مقدار هر یک از عناصر شمارشی را که می‌خواهیم به وسیله سازنده‌ای که در خطوط ۱۲-۱۵ تعریف شده است در آن قرار دهیم. برای به دست آوردن این مقادیر هم از متدهای `GetDirectionIndex()` (خطوط ۲۰-۲۷) استفاده می‌کنیم. حال فرض کنید می‌خواهیم مقدار عددی که به `East` اختصاص داده شده است را به دست آوریم. برای این کار همانطور که در خط ۲۷ مشاهده می‌کنید ابتدا یک نمونه از `Direction` ایجاد کرده و را به آن اختصاص می‌دهیم، سپس در خط ۲۸ با فراخوانی متدهای `GetDirectionIndex()` و `East` را به دست می‌آوریم. از همین عدد ۵ به دست آمده در مثال بالا می‌توان در ساختارهایی مانند `if` و `switch` استفاده کرد :

```

Direction myDirection = Direction.East;

switch(myDirection.GetDirectionIndex())
{

```

```

    case 1 : System.out.println("incorrect!");
        break;
    case 5 : System.out.println("That is Correct.");
        break;
}

```

آرگومان‌های خط فرمان (Command Line Arguments)

برای اجرای موفق یک برنامه جاوا باید یک متدهم به نام متده (main) وجود داشته باشد که نقطه آغاز برنامه است. این متده بده صورت public static void main(String[] args) تعریف شود. همه ما می‌دانیم که برای متدها می‌توان آرگومان ارسال کرد، اما برای متده [] چطور؟ جواب مثبت است. شما می‌توانید از طریق دستور خط فرمان ویندوز یا همان CMD آرگومان‌هایی را برای این متده ارسال کنید. برای روشن شدن مطلب یک برنامه کنسول به نام Sample ایجاد کنید، سپس کدهای برنامه را به صورت زیر بنویسید :

```

public class Sample
{
    public static void main(String[] args)
    {
        System.out.println("First Name is " + args[0]);
        System.out.println("Last Name is " + args[1]);
    }
}

```

به پارامتر args توجه کنید. در حقیقت این پارامتر یک آرایه رشته‌ای است که می‌تواند چندین آرگومان از نوع رشته قبول کند. فایل Sample.java را به یک درایو یا پوشه مشخص که مسیر گیج کننده‌ای نداشته باشد انتقال دهید. در این مثال ما فایل آن را مستقیماً در درایو C قرار می‌دهیم. حال CMD ویندوز را اجرا کنید، سپس کدهای زیر (خطوط قرمز) را در داخل CMD نوشته و دکمه Enter را بزنید :

```

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\VisualCsharp>cd/

C:\>javac Sample.java

C:\>java Sample Steven Clark
First Name is Steven
Last Name is Clark

C:\>

```

همانطور که در کد بالا مشاهده می‌کنید، بعد از نوشتن نام فایل، عبارت Steven Clark را نوشته‌ایم. دو کلمه این عبارت جایگزین دو متغیر به نام‌های args[0] و args[1] که تعریف کرده‌ایم، می‌شوند. این دو متغیر، به ترتیب خانه‌های اول و دوم آرایه هستند. کلمه دومین عنصر آرایه است ذخیره در متغیر رشته‌ای args[0] که اولین عنصر آرایه و کلمه Clark را در متغیر رشته‌ای args[1] که دومین عنصر آرایه است ذخیره

و سپس با استفاده از متدها `println()` آنها را چاپ می‌کنیم. در حقیقت بسیاری از برنامه‌ها از این تکنیک استفاده می‌کنند. شما می‌توانید با ارسال آرگومان‌هایی به متدهای `main()` نحوه اجرای برنامه را تغییر دهید.

برنامه نویسی شیء گرا (OOP)

برنامه نویسی شیء گرا یا Object Oriented Programming شامل تعریف کلاس‌ها و ساخت اشیاء مانند ساخت اشیاء در دنیای واقعی است. برای مثال یک ماشین را در نظر بگیرید. این ماشین دارای خواصی مانند رنگ، سرعت، مدل، سازنده و برخی خواص دیگر است. همچنین دارای رفتارها و حرکاتی مانند شتاب و پیچش به چپ و راست و ترمز است. اشیاء در جاوا تقليدی از یک شیء مانند ماشین در دنیای واقعی هستند. برنامه نویسی شیء گرا با استفاده از کدهای دسته بندی شده کلاس‌ها و اشیاء را بیشتر قابل کنترل می‌کند.

در ابتدا ما نیاز به تعریف یک کلاس برای ایجاد اشیاء مان داریم. شیء در برنامه نویسی شیء گراء از روی کلاسی که شما تعریف کردید ایجاد می‌شود. برای مثال نقشه ساختمان شما یک کلاس است که ساختمان از روی آن ساخته شده است. کلاس شامل خواص یک ساختمان مانند مساحت، بلندی و مواد مورد استفاده در ساخت خانه می‌باشد. در دنیای واقعی ساختمان‌ها نیز بر اساس یک نقشه (کلاس) پایه گذاری (تعریف) شده‌اند. برنامه نویسی شیء گرا یک روش جدید در برنامه نویسی است که بوسیله برنامه نویسان مورد استفاده قرار می‌گیرد و به آن‌ها کمک می‌کند که برنامه‌هایی با قابلیت استفاده مجدد، خوانا و راحت طراحی کنند. جاوا نیز یک برنامه شیء گراست. در درس زیر به شما نحوه تعریف کلاس و استفاده از اشیاء آموخته داده خواهد شد. همچنین شما با دو مفهوم وراثت و چند ریختی که از مباحث مهم در برنامه نویسی شیء گرا هستند در آینده آشنا می‌شوید.

کلاس

کلاس به شما اجازه می‌دهد یک نوع داده‌ای که توسط کاربر تعریف می‌شود و شامل فیلدها و خواص (properties) و متدها است را ایجاد کنید. کلاس در حکم یک نقشه برای یک شیء می‌باشد. شیء یک چیز واقعی است که از ساختار، خواص و یا رفتارهای کلاس پیروی می‌کند.

وقتی یک شیء می‌سازید یعنی اینکه یک نمونه از کلاس ساخته‌اید (در درس ممکن است از کلمات شیء و نمونه به جای هم استفاده شود). ابتدا ممکن است فکر کنید که کلاس‌ها و ساختارها شبیه هم هستند. تفاوت مهم بین این دو این است که کلاس‌ها از نوع مرجع و ساختارها از نوع داده‌ای هستند. در درس‌های آینده این موضوع شرح داده خواهد شد. اگر یادتان باشد در بخش‌های اولیه این آموزش

کلاسی به نام MyFirstProgram تعریف کردیم که شامل متدهای main() بود و ذکر شد که این متدهای نقطه آغاز هر برنامه است. برای تعریف

یک کلاس از کلمه کلیدی class به صورت زیر استفاده می‌شود :

```
class ClassName
{
    field1;
    field2;
    ...
    fieldN;

    method1;
    method2;
    ...
    methodN;
}
```

این کلمه کلیدی را قبل از نامی که برای کلاسمان انتخاب می‌کنیم می‌نویسیم. در نامگذاری کلاس‌ها هم از روش نامگذاری Pascal استفاده می‌کنیم. در بدنه کلاس فیلدها و متدهای آن قرار داده می‌شوند. فیلدها اعضای داده‌ای خصوصی هستند که کلاس از آن‌ها برای رفتارها و ذخیره مقادیر خاصیت‌هایش (property) استفاده می‌کند. متدها رفتارها یا کارهایی هستند که یک کلاس می‌تواند انجام دهد. در زیر نحوه تعریف و استفاده از یک کلاس ساده به نام person نشان داده شده است.

```
1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 class Person
6 {
7     public String name;
8     public int age;
9     public double height;
10
11     public void TellInformation()
12     {
13         System.out.println(MessageFormat.format("Name: {0}", name));
14         System.out.println(MessageFormat.format("Age: {0} years old", age));
15         System.out.println(MessageFormat.format("Height: {0}cm", height));
16     }
17 }
18
19 public class MyFirstProgram
20 {
21     public static void main(String[] args)
22     {
23         Person firstPerson = new Person();
24         Person secondPerson = new Person();
25
26         firstPerson.name    = "Jack";
27         firstPerson.age    = 21;
28         firstPerson.height = 160;
```

```

29     firstPerson.TellInformation();

30     System.out.println(); //Separator

31     secondPerson.name    = "Mike";
32     secondPerson.age     = 23;
33     secondPerson.height  = 158;
34     secondPerson.TellInformation();
35   }
36 }
37 }
38 }
```

Name: Jack
 Age: 21 years old
 Height: 160cm

Name: Mike
 Age: 23 years old
 Height: 158cm

برنامه بالا شامل دو کلاس Person (خطوط ۵-۱۷) و MyFirstProgram (خطوط ۱۹-۳۸) می‌باشد. می‌دانیم که کلاس

شامل متده است که برنامه برای اجرا به آن احتیاج دارد ولی اجازه دهید که بر روی کلاس Person تمرکز کنیم. در خطوط ۵-۱۷ کلاس Person تعریف شده است. در خط ۵ یک نام به کلاس اختصاص داده‌ایم تا به وسیله آن قابل دسترسی باشد. در داخل بدن کلاس فیلدهای آن تعریف شده‌اند (خطوط ۹-۷).

این سه فیلد تعریف شده خصوصیات واقعی یک فرد در دنیای واقعی را در خود ذخیره می‌کنند. یک فرد در دنیای واقعی دارای نام، سن، و قد می‌باشد. در خطوط ۱۱-۱۶ یک متده هم در داخل کلاس به نام TellInformation() تعریف شده است که رفتار کلاسمان است و مثلًاً اگر از فرد سوالی بپرسیم در مورد خودش چیزهایی می‌گوید. در داخل متده کدهایی برای نشان دادن مقادیر موجود در فیلدها نوشته شده است. نکته‌ای درباره فیلدها وجود دارد و این است که چون فیلدها در داخل کلاس تعریف و به عنوان اعضای کلاس در نظر گرفته شده‌اند محدوده آن‌ها یک کلاس است.

این بدین معناست که فیلدها فقط می‌توانند در داخل کلاس یعنی جایی که به آن تعلق دارند و یا به وسیله نمونه ایجاد شده از کلاس مورد استفاده قرار بگیرند. در داخل متده main() و در خطوط ۲۳ و ۲۴ دو نمونه یا دو شیء از کلاس Person ایجاد می‌کنیم. برای ایجاد یک نمونه از یک کلاس باید از کلمه کلیدی new و به دنبال آن نام کلاس و یک جفت پرانتز قرار دهیم. وقتی نمونه کلاس ایجاد شد، سازنده را صدا می‌زنیم. یک سازنده متده خاصی است که برای مقداردهی اولیه به فیلدهای یک شیء به کار می‌رود. وقتی هیچ آرگومانی در داخل پرانتزها قرار ندهید، کلاس یک سازنده پیشفرض بدون پارامتر را فراخوانی می‌کند. درباره سازنده‌ها در درس‌های آینده توضیح خواهیم

داد. در خطوط ۲۶-۲۹ مقادیری به فیلدهای اولین شیء ایجاد شده از کلاس (first Person) اختصاص داده شده است.

برای دسترسی به فیلدها یا متدهای یک شیء از علامت نقطه (دات) استفاده می‌شود. به عنوان مثال کد firstPerson.name نشان دهنده فیلد name از شیء firstPerson می‌باشد. برای چاپ مقادیر فیلدها باید متده را firstPerson.TellInformation() از شیء TellInformation فراخوانی می‌کنیم.

در خطوط ۳۴-۳۶ نیز مقادیری به شیء دومی که قبلاً از کلاس ایجاد شده تخصیص می‌دهیم و سپس متده را TellInformation() فراخوانی می‌کنیم. به این نکته توجه کنید که secondPerson و firstPerson نسخه‌های متفاوتی از هر فیلد دارند بنابراین تعیین یک نام برای secondPerson هیچ تاثیری بر نام firstPerson ندارد. در مورد اعضای کلاس در درس‌های آینده توضیح خواهیم داد.

سازنده

سازنده‌ها متدهای خاصی هستند که وجود آن‌ها برای ساخت اشیاء لازم است. آن‌ها به شما اجازه می‌دهند که مقادیری را به هر یک از اعضای داده‌ای یک آرایه اختصاص دهید و کدهایی که را که می‌خواهید هنگام ایجاد یک شیء اجرا شوند را به برنامه اضافه کنید. اگر از هیچ سازنده‌ای در کلاستان استفاده نکنید، کمپایلر از سازنده پیشفرض که یک سازنده بدون پارامتر است استفاده می‌کند. می‌توانید در برنامه‌تان از تعداد زیادی سازنده استفاده کنید که دارای پارامترهای متفاوتی باشند. در مثال زیر یک کلاس که شامل سازنده است را مشاهده می‌کنید:

```

1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 class Person
6 {
7     public String name;
8     public int age;
9     public double height;
10
11    //Explicitly declare a default constructor
12    public Person()
13    {
14    }
15
16    //Constructor that has 3 parameters
17    public Person(String n, int a, double h)
18    {
19        name = n;
20        age = a;
21        height = h;
22    }
23 }
```

```

24 public void ShowInformation()
25 {
26     System.out.println(MessageFormat.format("Name: {0}", name));
27     System.out.println(MessageFormat.format("Age: {0} years old", age));
28     System.out.println(MessageFormat.format("Height: {0}cm", height));
29 }
30
31
32 public class MyFirstProgram
33 {
34     public static void main(String[] args)
35     {
36         Person firstPerson = new Person();
37         Person secondPerson = new Person("Mike", 23, 158);
38
39         firstPerson.name = "Jack";
40         firstPerson.age = 21;
41         firstPerson.height = 160;
42         firstPerson.ShowInformation();
43
44         System.out.println(); //Separator
45
46         secondPerson.ShowInformation();
47     }
48 }
```

Name: Jack
 Age: 21 years old
 Height: 160cm

Name: Mike
 Age: 23 years old
 Height: 158cm

همانطور که مشاهده می‌کنید در مثال بالا دو سازنده را به کلاس Person اضافه کرده‌ایم. یکی از آن‌ها سازنده پیشفرض (خطوط ۱۲-۱۴) و دیگری سازنده‌ای است که سه آرگومان قبول می‌کند (خطوط ۱۹-۲۱). به این نکته توجه کنید که سازنده درست شبیه به یک متده است با

این تفاوت که

- نه مقدار برگشتی دارد و نه از نوع void است.

- نام سازنده باید دقیقاً شبیه نام کلاس باشد.

سازنده پیشفرض در داخل بدنهاش هیچ چیزی ندارد و وقتی فراخوانی می‌شود که ما از هیچ سازنده‌ای در کلاسمان استفاده نکنیم. در آینده متوجه می‌شویم که چطور می‌توان مقادیر پیشفرضی به اعضای داده‌ای اختصاص داد، وقتی که از یک سازنده پیشفرض استفاده می‌کنیم. به دومین سازنده توجه کنید. اولاً که نام آن شبیه نام سازنده اول است. سازنده‌ها نیز مانند متدها می‌توانند سربارگذاری شوند. حال اجازه دهید که چطور می‌توانیم یک سازنده خاص را هنگام تعریف یک نمونه از کلاس فراخوانی کنیم.

```
Person firstPerson = new Person();
Person secondPerson = new Person("Mike", 23, 158);
```

در اولین نمونه ایجاد شده از کلاس Person از سازنده پیشفرض استفاده کرده‌ایم چون پارامتری برای دریافت آرگومان ندارد. در دومین نمونه ایجاد شده، از سازنده‌ای استفاده می‌کنیم که دارای سه پارامتر است. کد زیر تأثیر استفاده از دو سازنده مختلف را نشان می‌دهد:

```
firstPerson.name = "Jack";
firstPerson.age = 21;
firstPerson.height = 160;
firstPerson.ShowInformation();

System.out.println(); //Separator

secondPerson.ShowInformation();
```

همانطور که مشاهده می‌کنید لازم است که به فیلدهای شیء ای که از سازنده پیشفرض استفاده می‌کند مقادیری اختصاص داده شود تا آنها ShowInformation() را نمایش دهد. حال به شیء دوم که از سازنده دارای پارامتر استفاده می‌کند توجه کنید، مشاهده می‌کنید که با فراخوانی متدهای ShowInformation() همه چیز همانطور که انتظار می‌رود اجرا می‌شود. این بدین دلیل است که شما هنگام تعریف نمونه و از قبل مقادیری به هر یک از فیلدها اختصاص داده‌اید بنابراین آنها نیاز به مقدار دهی مجدد ندارند، مگر اینکه شما بخواهید این مقادیر را اصلاح کنید.

اختصاص مقادیر پیشفرض به سازنده پیشفرض

در مثال‌های قبلی یک سازنده پیشفرض با بدنه خالی نشان داده شد. شما می‌توانید به بدنه این سازنده پیشفرض کدهایی اضافه کنید. همچنین می‌توانید مقادیر پیشفرضی به فیلدهای آن اختصاص دهید.

```
public Person()
{
    this.name = "No Name";
    this.age = 0;
    this.height = 0;
}
```

همانطور که در مثال بالا می‌بینید سازنده پیشفرض ما چیزی برای اجرا دارد. اگر نمونه‌ای ایجاد کنیم که از این سازنده پیشفرض استفاده کند، نمونه ایجاد شده مقادیر پیشفرض سازنده پیشفرض را نشان می‌دهد.

```
Person person1 = new Person();
person1.ShowInformation();
```

```
Name: No Name
Age: 0 years old
Height: 0cm
```

استفاده از کلمه کلیدی this

راهی دیگر برای ایجاد مقادیر پیشفرض استفاده از کلمه کلیدی this است. مثال زیر اصلاح شده مثال قبل است و نحوه استفاده از `this` سازنده با تعداد پارامترهای مختلف را نشان می‌دهد.

```

1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 class Person
6 {
7     public String name;
8     public int age;
9     public double height;
10
11    public Person()
12    {
13        this.name = "No Name";
14        this.age = 0;
15        this.height = 0;
16    }
17
18    public Person(String n)
19    {
20        this.name = n;
21    }
22
23    public Person(String n, int a)
24    {
25        this.name = n;
26        this.age = a;
27    }
28
29    public Person(String n, int a, double h)
30    {
31        this.name = n;
32        this.age = a;
33        this.height = h;
34    }
35
36    public void ShowInformation()
37    {
38        System.out.println(MessageFormat.format("Name: {0}", name));
39        System.out.println(MessageFormat.format("Age: {0} years old", age));
40        System.out.println(MessageFormat.format("Height: {0}cm\n", height));
41    }
42
43
44    public class MyFirstProgram
45    {
```

```

46 public static void main(String[] args)
47 {
48     Person firstPerson = new Person();
49     Person secondPerson = new Person("Jack");
50     Person thirdPerson = new Person("Mike", 23);
51     Person fourthPerson = new Person("Chris", 18, 152);
52
53     firstPerson.ShowInformation();
54     secondPerson.ShowInformation();
55     thirdPerson.ShowInformation();
56     fourthPerson.ShowInformation();
57 }
58 }
```

Name: No Name
 Age: 0 years old
 Height: 0cm

Name: Jack
 Age: 0 years old
 Height: 0cm

Name: Mike
 Age: 23 years old
 Height: 0cm

Name: Chris
 Age: 18 years old
 Height: 152cm

ما چهار سازنده بری اصلاح کلاسمن تعريف کردہ‌ایم (خطوط ۱۱، ۱۸، ۲۵، ۳۲). شما می‌توانید تعداد زیادی سازنده برای موقع لزوم در کلاس داشته باشید. اولین سازنده یک سازنده پیشفرض است. دومین سازنده یک پارامتر از نوع رشته دریافت می‌کند. سومین سازنده دو پارامتر و چهارمین سازنده سه پارامتر می‌گیرد. به چارمین سازنده در خطوط ۳۲-۳۷ توجه کنید. سه سازنده دیگر به این سازنده وابسته هستند. در خطوط ۱۱-۱۶ یک سازنده پیشفرض بدون پارامتر تعريف شده است. به کلمه کلیدی `this` توجه کنید. این کلمه کلیدی به شما اجازه می‌دهد که یک سازنده دیگر موجود در داخل کلاس را فراخوانی کنید. مقادیر پیشفرضی به فیلدها از طریق سازنده پیشفرض اختصاص می‌دهیم. چون ما سه مقدار پیشفرض برای فیلدها بعد از کلمه کلیدی `this` سازنده پیشفرض (خطوط ۱۳-۱۵) در نظر گرفته‌ایم، در نتیجه سازنده‌ای که دارای سه پارامتر است (چهارمین سازنده) فراخوانی شده و سه آرگومان به پارامترهای آن ارسال می‌شود. کدهای داخل بدن سازنده‌ای اجراء می‌شوند و مقادیر پارامترهای آن به هر یک از اعضای داده‌ای به همان فیلدهای تعريف شده در خطوط ۷-۹ اختصاص داده می‌شود. اگر کدی در داخل بدن سازنده پیشفرض بنویسیم قبل از بقیه کدها اجرا می‌شود. دومین سازنده (خطوط ۲۳-۲۵) به یک آرگومان نیاز دارد که همان فیلد `name` کلاس `Person` است. وقتی این پارامتر با یک مقدار رشته‌ای پر شد، سپس به پارامترهای سازنده چهارم ارسال شده و در کنار دو مقدار پیشفرض دیگر (۰ برای `age` و ۰ برای `height`) قرار می‌گیرد. در خط ۲۵ سومین سازنده تعريف

شده است که بسیار شبیه دومین سازنده است با این تفاوت که دو پارامتر دارد. مقدار دو پارامتر سومین سازنده به اضافه‌ی یک مقدار بیشفرض صفر برای سومین آرگومان، به چهارمین سازنده با استفاده از کلمه کلیدی `this` ارسال می‌شود.

```
Person firstPerson = new Person();
Person secondPerson = new Person("Jack");
Person thirdPerson = new Person("Mike", 23);
Person fourthPerson = new Person("Chris", 18, 152);
```

همانطور که مشاهده می‌کنید با ایجاد چندین سازنده برای یک کلاس، چندین راه برای ایجاد یک شیء بر اساس داده‌هایی که نیاز داریم به وجود می‌آید. در مثال بالا ۴ نمونه از کلاس `Person` ایجاد کرده‌ایم و چهار تغییر در سازنده آن به وجود آورده‌ایم. سپس مقادیر مربوط به فیلدهای هر نمونه را نمایش می‌دهیم. یکی از موارد استفاده از کلمه کلیدی `this` به صورت زیر است. فرض کنید نام پارامترهای متدهای کلاس شما یا سازنده، شبیه نام یکی از فیلدها باشد.

```
public Person(String name, int age, double height)
{
    name = name;
    age = age;
    height = height;
}
```

این نوع کدنویسی ابهام بر انگیز است و کمپایلر نمی‌تواند متغیر را تشخیص داده و مقداری به آن اختصاص دهد. اینجاست که از کلمه کلیدی `this` استفاده می‌کنیم.

```
public Person(String name, int age, double height)
{
    this.name = name;
    this.age = age;
    this.height = height;
}
```

قبل از هر فیلدی کلمه کلیدی `this` را می‌نویسیم و نشان می‌دهیم که این همان چیزی است که می‌خواهیم به آن مقداری اختصاص دهیم. کلمه کلیدی `this` ارجاع یک شیء به خودش را نشان می‌دهد.

سطح دسترسی

سطح دسترسی مشخص می‌کند که متدها یک کلاس یا اعضای داده‌ای در چه جای برنامه قابل دسترسی هستند. در این درس می‌خواهیم به سطح دسترسی `public` و `private` نگاهی بیندازیم. سطح دسترسی `public` زمانی مورد استفاده قرار می‌گیرد که شما بخواهید به یک متod یا فیلد در خارج از کلاس و حتی پروژه دسترسی یابید. به عنوان مثال به کد زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 class Test
6 {
7     public int number;
8 }
9
10 public class MyFirstProgram
11 {
12     public static void main(String[] args)
13     {
14         Test x = new Test();
15
16         x.number = 10;
17     }
18 }
```

در این مثال یک کلاس به نام `Test` تعریف کرده‌ایم (خطوط ۵-۸). سپس یک فیلد یا عضو داده‌ای به صورت `public` در داخل کلاس `main()` تعریف می‌کنیم (خط ۷). با تعریف این عضو به صورت `public` می‌توانیم آن را در خارج از کلاس `Test` و در داخل متدهای `Test` مقدار دهی کنیم. حال سطح دسترسی `public` را به `private` تغییر می‌دهیم :

```

1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 class Test
6 {
7     private int number;
8 }
9
10 public class MyFirstProgram
11 {
12     public static void main(String[] args)
13     {
14         Test x = new Test();
15
16         x.number = 10;
17     }
18 }
```

همانطور که در مثال بالا مشاهده می‌کنید این بار از کلمه `private` در تعریف فیلد `number` استفاده کرده‌ایم (خط ۷). وقتی که برنامه را کامپایل می‌کنیم با خطأ مواجه می‌شویم چون `number` در داخل کلاس `MyFirstProgram` و یا هر کلاس دیگر قابل دسترسی نیست.

نکته دیگر اینکه اگر شما برای یک کلاس سطح دسترسی تعریف نکنید آن کلاس دارای سطح دسترسی داخلی (`default modifier`) می‌شود به این معنی که فقط کلاس‌های داخل پروژه‌ای که با آن کار می‌کنند و می‌توانند به آن کلاس دسترسی یابند. اگر یک کلاس را به

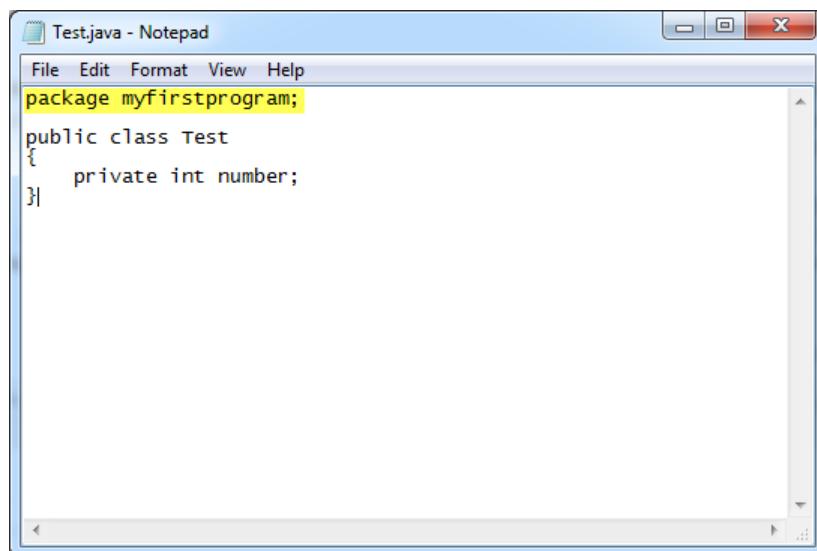
صورت `public` و اعضای آن را به صورت `private` تعریف کنیم، آنگاه می‌توان یک نمونه از کلاس را در داخل کلاس‌های دیگر ایجاد کرد.

ولی اعضای آن قابل دسترسی نیستند. اعضای داده‌ای `private` فقط به وسیله متدهای `Test` قابل دسترسی هستند.

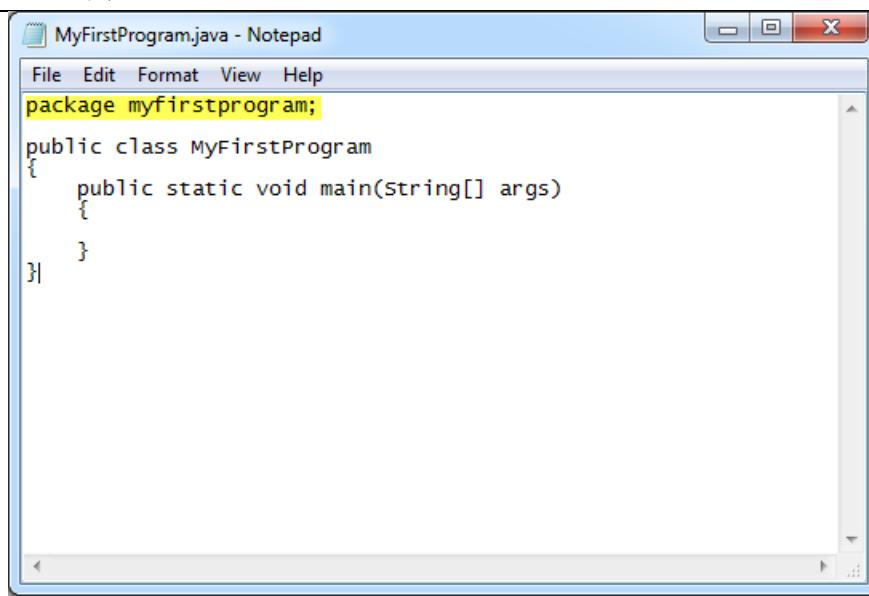
```

1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 public class Test
6 {
7     private int number;
8 }
9
10 public class MyFirstProgram
11 {
12     public static void main(String[] args)
13     {
14         Test x = new Test();
15
16         x.number = 10;
17     }
18 }
```

کد بالا کامپایل نمی‌شود، چون جاوا به شما اجازه استفاده از دو کلاس عمومی در یک فایل را نمی‌دهد. برای حل این مشکل یا باید کلمه `public` را مثلًا از ابتدای کلاس `Test` حذف کنیم و یا اینکه چون ما می‌خواهیم حتماً کلاس به صورت `public` تعریف شود و اعضای `public` آن به صورت `private` آن را باید در یک فایل جدا به صورت زیر تعریف و با پسوند `.java` تعریف کنیم :



همانطور که در شکل بالا و پایین مشاهده می‌کنید باید دو کلاس در داخل یک Package باشند :



حال اگر از کلاس Test در کلاس MyFirstProgram نمونه‌ای ایجاد کرده و بخواهیم از فیلد number استفاده کنیم با خطاب مواجه می‌شویم،

چون که اعضای با سطح دسترسی private در خارج از کلاسی که به آن تعلق دارند، غیرقابل دسترس هستند :

```
package myfirstprogram;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Test x = new Test();
        x.number = 10;
    }
}
```

سطوح دسترسی دیگری هم در جاوا وجود دارد که بعد از مبحث وراثت در درس‌های آینده در مورد آنها توضیح خواهیم داد.

کپسوله سازی (Encapsulation)

کپسوله سازی (تلفیق داده‌ها با یکدیگر) یا مخفی کردن اطلاعات فرایندی است که طی آن اطلاعات حساس یک موضوع از دید کاربر مخفی می‌شود و فقط اطلاعاتی که لازم باشد برای او نشان داده می‌شود.

وقتی که یک کلاس تعریف می‌کنیم معمولاً تعدادی اعضای داده‌ای (فیلد) برای ذخیره مقادیر مربوط به شیء نیز تعریف می‌کنیم. برخی از این اعضای داده‌ای توسط خود کلاس برای عملکرد متدها و برخی دیگر از آن‌ها به عنوان یک متغیر موقت به کار می‌روند. به این اعضای داده‌ای، اعضای مفید نیز می‌گویند چون فقط در عملکرد متدها تأثیر دارند و مانند یک داده قابل رویت کلاس نیستند. لازم نیست که کاربر

به تمام اعضای داده‌ای کلاس دسترسی داشته باشد. اینکه فیلدها را طوری تعریف کنیم که در خارج از کلاس قابل دسترسی

باشند بسیار خطرناک است چون ممکن است کاربر رفتار و نتیجه یک متدها را تغییر دهد. به برنامه ساده زیر توجه کنید :

```

1 package myfirstprogram;
2
3 class Test
4 {
5     public int five = 5;
6
7     public int AddFive(int number)
8     {
9         number += five;
10        return number;
11    }
12 }
13
14 public class MyFirstProgram
15 {
16     public static void main(String[] args)
17     {
18         Test x = new Test();
19
20         x.five = 10;
21         System.out.println(x.AddFive(100));
22     }
23 }
```

110

متدهای کلاس Test به نام AddFive() دارای هدف ساده‌ای است و آن اضافه کردن مقدار ۵ به هر عدد می‌باشد. در داخل متدهای کلاس Test ایجاد کردہ‌ایم و مقدار فیلد آن را از ۵ به ۱۰ تغییر می‌دهیم (در اصل نباید تغییر کند چون ما از برنامه خواسته‌ایم هر عدد را با ۵ جمع کند ولی کاربر به راحتی آن را به ۱۰ تغییر می‌دهد). همچنین متدهای AddFive() را فراخوانی و مقدار ۱۰۰ را به آن ارسال می‌کنیم. مشاهده می‌کنید که قابلیت متدهای AddFive() به خوبی تغییر می‌کند و شما نتیجه متفاوتی مشاهده می‌کنید. اینجاست که اهمیت کپسوله سازی مشخص می‌شود. اینکه ما در درس‌های قبلی فیلدها را به صورت public تعریف کردیم و به کاربر اجازه دادیم که در خارج از کلاس به آنها دسترسی داشته باشد کار اشتباهی بود. فیلدها باید همیشه به صورت private تعریف شوند.

خواص (Properties)

Property (خصوصیت) استانداردی در جاوا برای دسترسی به اعضای داده‌ای با سطح دسترسی private در داخل یک کلاس می‌باشد. هر property دارای دو بخش می‌باشد، یک بخش جهت مقدار دهنده (بلوک set) و یک بخش برای دسترسی به مقدار (بلوک get) یک

داده `private` می‌باشد. `property` ها باید به صورت `public` تعریف شوند تا در کلاس‌های دیگر نیز قابل دسترسی می‌باشند. در مثال

زیر نحوه تعریف و استفاده از `property` آمده است :

```

1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 class Person
6 {
7     private String name;
8     private int age;
9     private double height;
10
11     public void setName(String name)
12     {
13         this.name = name;
14     }
15
16     public String getName()
17     {
18         return name;
19     }
20
21     public void setAge(int age)
22     {
23         this.age = age;
24     }
25
26     public int getAge()
27     {
28         return age;
29     }
30
31     public void setHeight(double height)
32     {
33         this.height = height;
34     }
35
36     public double getHeight()
37     {
38         return height;
39     }
40
41
42     public Person(String name, int age, double height)
43     {
44         this.name = name;
45         this.age = age;
46         this.height = height;
47     }
48
49 }
50
51 public class MyFirstProgram
52 {
53     public static void main(String[] args)
54     {
55         Person person1 = new Person("Jack", 21, 160);
56         Person person2 = new Person("Mike", 23, 158);
57
58         System.out.println(MessageFormat.format("Name: {0}", person1.getName()));
59         System.out.println(MessageFormat.format("Age: {0} years old", person1.getAge()));
60         System.out.println(MessageFormat.format("Height: {0}cm", person1.getHeight()));
61     }
62 }
```

```

61     System.out.println(); //Separator
62
63
64     System.out.println(MessageFormat.format("Name: {0}", person2.getName()));
65     System.out.println(MessageFormat.format("Age: {0} years old", person2.getAge()));
66     System.out.println(MessageFormat.format("Height: {0}cm", person2.getHeight()));
67
68
69     person1.setName("Frank");
70     person1.setAge(19);
71     person1.setHeight(162);
72
73     person2.setName("Ronald");
74     person2.setAge(25);
75     person2.setHeight(174);
76
77     System.out.println(); //Separator
78
79     System.out.println(MessageFormat.format("Name: {0}", person1.getName()));
80     System.out.println(MessageFormat.format("Age: {0} years old", person1.getAge()));
81     System.out.println(MessageFormat.format("Height: {0}cm", person1.getHeight()));
82
83     System.out.println(); //Separator
84
85     System.out.println(MessageFormat.format("Name: {0}", person2.getName()));
86     System.out.println(MessageFormat.format("Age: {0} years old", person2.getAge()));
87     System.out.println(MessageFormat.format("Height: {0}cm", person2.getHeight()));
88 }
89 }
```

Name: Jack
 Age: 21 years old
 Height: 160cm

Name: Mike
 Age: 23 years old
 Height: 158cm

Name: Frank
 Age: 19 years old
 Height: 162cm

Name: Ronald
 Age: 25 years old
 Height: 174cm

در برنامه بالا نحوه استفاده از `property` آمده است. همانطور که مشاهده می‌کنید در این برنامه ما سه خصوصیت که هر کدام مربوط به اعضای داده‌ای هستند تعریف کردہ‌ایم (سه فیلد با سطح دسترسی `private`).

```

private String name;
private int age;
private double height;
```

دسترسی به مقادیر این فیلدها فقط از طریق `property` های ارائه شده امکان پذیر است.

```

11 public void setName(String name)
12 {
13     this.name = name;
14 }
15
16 public String getName()
17 {
18     return name;
19 }
20
21 public void setAge(int age)
22 {
23     this.age = age;
24 }
25
26 public int getAge()
27 {
28     return age;
29 }
30
31 public void setHeight(double height)
32 {
33     this.height = height;
34 }
35
36 public double getHeight()
37 {
38     return height;
39 }

```

وقتی یک خاصیت ایجاد می‌کنیم، باید سطح دسترسی آن را `public` تعریف کرده و نوع داده‌ای را که بر می‌گرداند یا قبول می‌کند را مشخص کنیم. به این نکته توجه کنید که نام `property` ها همانند نام فیلدهای مربوطه می‌باشد با این تفاوت که حرف اول آن‌ها بزرگ نوشته می‌شود. البته قبل از نام آن‌ها کلمات `set` و `get` هم نوشته می‌شود. مثلاً برای فیلد `name` (خط ۷) دو متدهای `setName` و `getName` (خطوط ۱۱-۱۹) ایجاد شده‌اند. البته یادآور می‌شویم که شباهت نام `property` ها و فیلدها اجبار نیست و یک قرارداد می‌باشد.

در داخل بدن دو بخش می‌بینید، یکی بخش `set` و دیگری بخش `get`:

- بخش `get`، که با کلمه `get` نشان داده شده است به شما اجازه می‌دهد که یک مقدار را از فیلدها (اعضای داده‌ای) استخراج کنید.
 - بخش `set`، که با کلمه `set` نشان داده شده است برای مقدار دهی به فیلدها (اعضای داده‌ای) به کار می‌رود.
- به عنوان مثال به عبارت زیر توجه کنید :

```
this.name = name;
```

این عبارت که در خط ۱۳ کد بالا آمده است برای مقداردهی به فیلد name به کار رفته است. this به شی جاری ایجاد شده از کلاس اشاره دارد، کلمه name بعد آن همان فیلد تعریف شده در خط ۷ کد ابتدای آموزش و کلمه name سمت راست علامت مساوی هم پارامتر تعریف شده در خط ۱۱ است. برای دسترسی به یک خاصیت می‌توانید از علامت دات (.) استفاده کنید.

```
System.out.println(MessageFormat.format("Name: {0}", person1.getName()));
System.out.println(MessageFormat.format("Age: {0} years old", person1.getAge()));
System.out.println(MessageFormat.format("Height: {0}cm", person1.getHeight()));
```

فراخوانی یک خاصیت باعث اجرای کد داخل بدنه بلوک get آن می‌شود. سپس این بلوک مانند یک متدهای مقداری به فراخوان برگشت می‌دهد. مقداردهی به یک property بسیار آسان است.

```
person1.setName("Frank");
person1.setAge(19);
person1.setHeight(162);
```

دستورات بالا بخش set مربوط به هر property را فراخوانی کرده و مقادیری به هر یک از فیلدها اختصاص می‌دهد. استفاده از property‌ها کد نویسی را انعطاف‌پذیر می‌کند، مخصوصاً اگر بخواهید یک اعتبارسنجی برای اختصاص یک مقدار به فیلدها یا استخراج یک مقدار از آن‌ها ایجاد کنید. مثلًا شما می‌توانید یک محدودیت ایجاد کنید که فقط اعداد مثبت به فیلد age (سن) اختصاص داده شود. می‌توانید با تغییر بخش set خاصیت Age این کار را انجام دهید:

```
public void setAge(int age)
{
    if (age > 0)
    {
        this.age = age;
    }
    else
    {
        this.age = 0;
    }
}
```

حال اگر کاربر بخواهد یک مقدار منفی به فیلد age اختصاص دهد مقدار age صفر خواهد شد.

Package

(پکیج) راهی برای دسته بندی کدهای برنامه می‌باشد. هر چیز در جاوا حداقل در یک Package قرار دارد. وقتی برای یک کلاس اسمی انتخاب می‌کنید ممکن است برنامه نویسان دیگر به صورت اتفاقی اسمی شبیه به آن برای کلاسشن انتخاب کنند. وقتی شما از آن کلاس‌ها در برنامه‌تان استفاده کنید از آنجاییکه از کلاس‌های همنام استفاده می‌کنید در برنامه ممکن است خطأ به وجود آید.

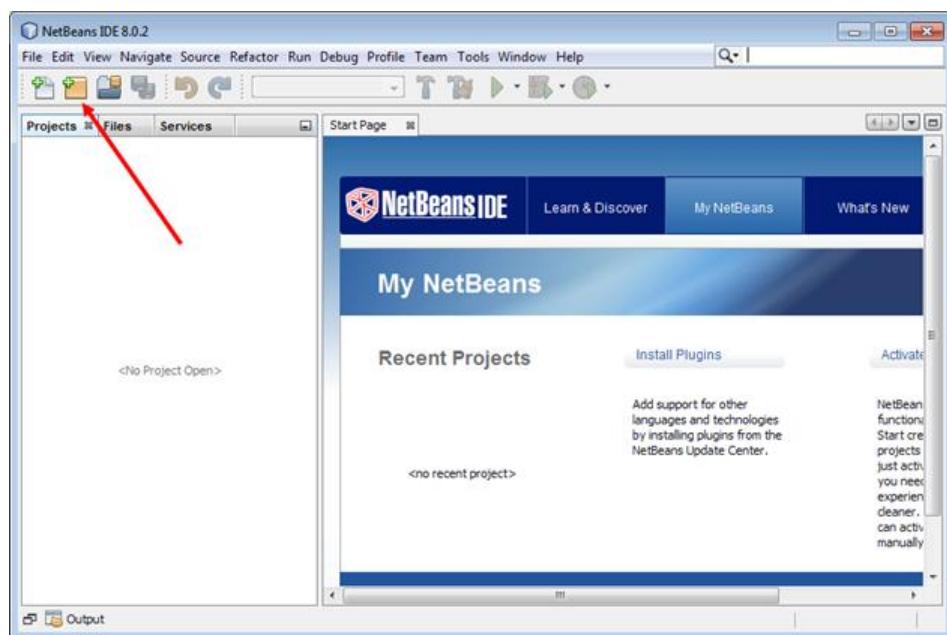
از وقوع این خطاهای جلوگیری کرده یا آنها را کاهش می‌دهند. تاکنون و در درس‌های قبلی ما فقط با یک پکیج آشنا شدایم و

آن پکیجی به نام `myfirstprogram` بود که کلاسی به همین نام (`MyFirstProgram`) و متد (`main()`) را در خود داشت. هنگامی که

یک پروژه جدید ایجاد کنید به صورت پیشفرض یک فضای نام برای شما ایجاد خواهد شد که نام آن شبیه به نام پروژه‌تان می‌باشد. در

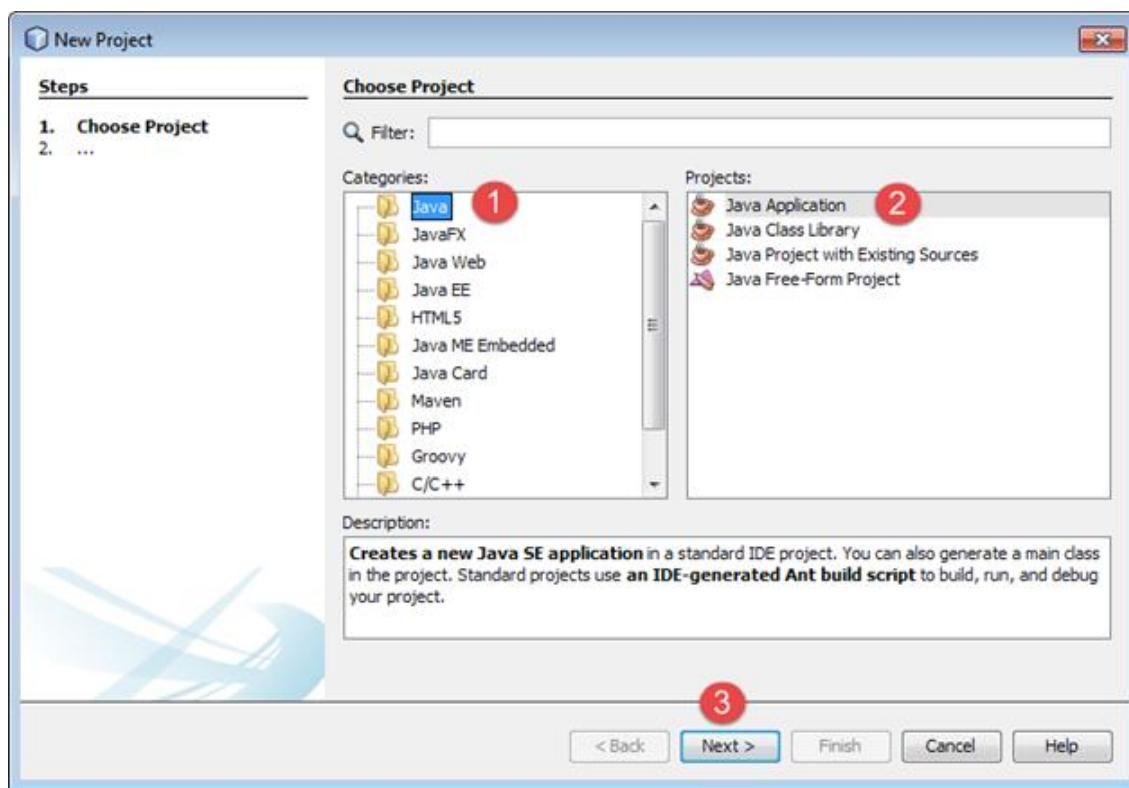
این درس به شما نشان می‌دهیم که چگونه کلاس‌هایتان در در کدهای جداگانه بنویسید و سپس از آنها در فایل‌های جدا استفاده کنید.

برنامه NetBeans را اجرا و یک پروژه جدید ایجاد کنید:



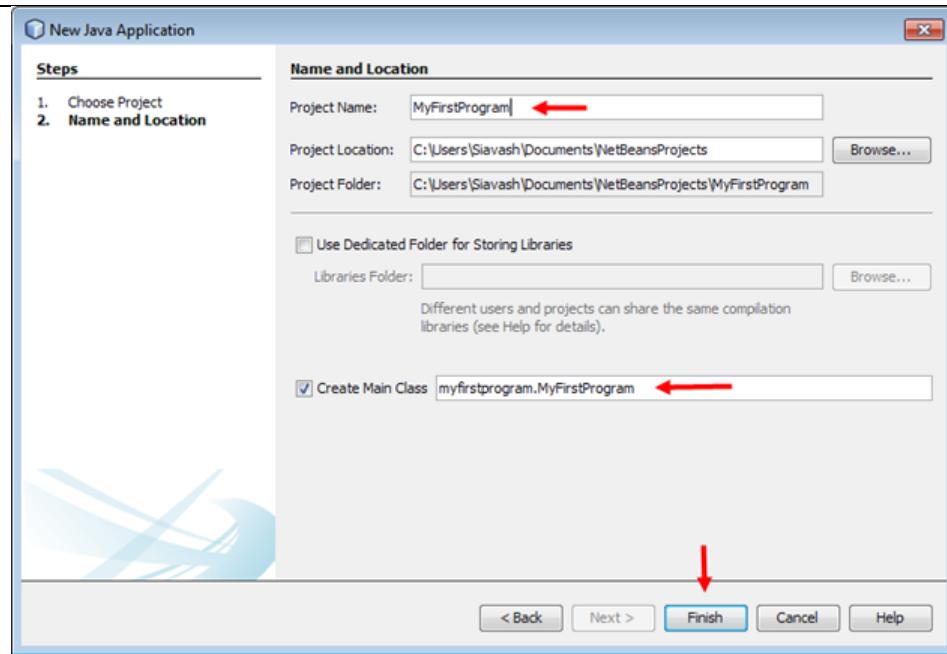
بعد از کلیک بر روی گزینه `New Project` و یا زدن دکمه‌های ترکیبی `Ctrl+Shift+N` پنجره‌ای به صورت زیر به نمایش در می‌آید که بر

طبق شکل گزینه‌ها را تنتخاب کنید :

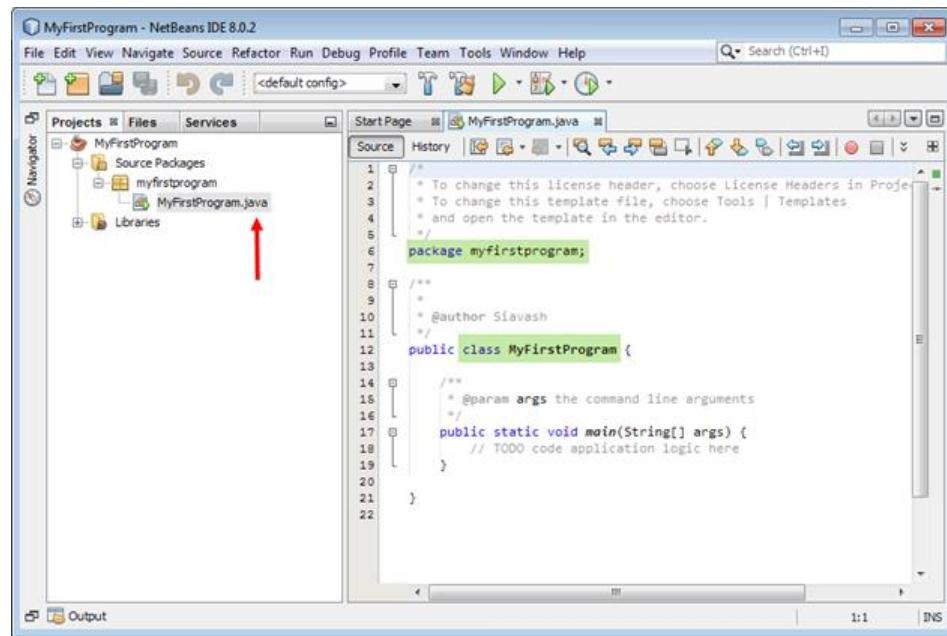


بعد از طی مراحل شکل بالا صفحه زیر به نمایش در می‌آید. در این صفحه و در قسمت Project Name نام پروژه‌تان را انتخاب کنید.

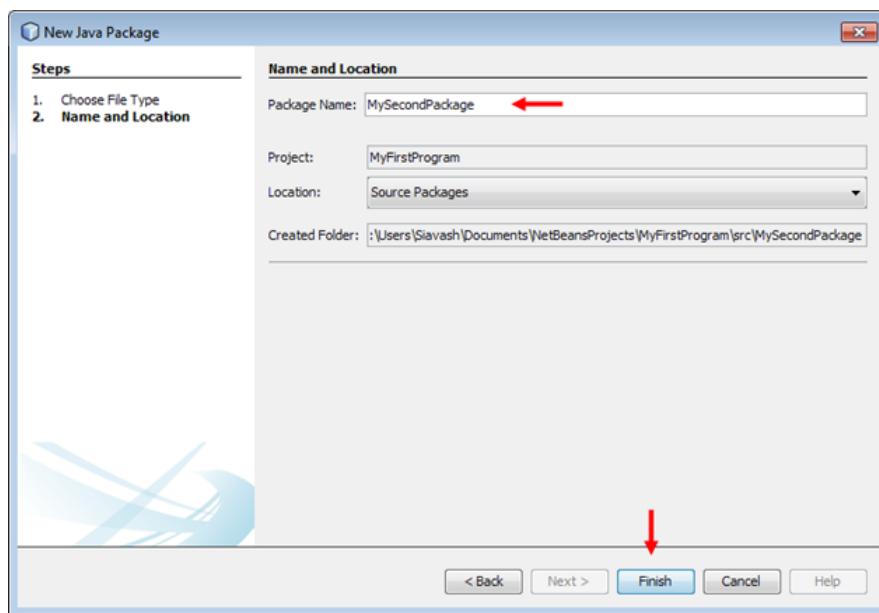
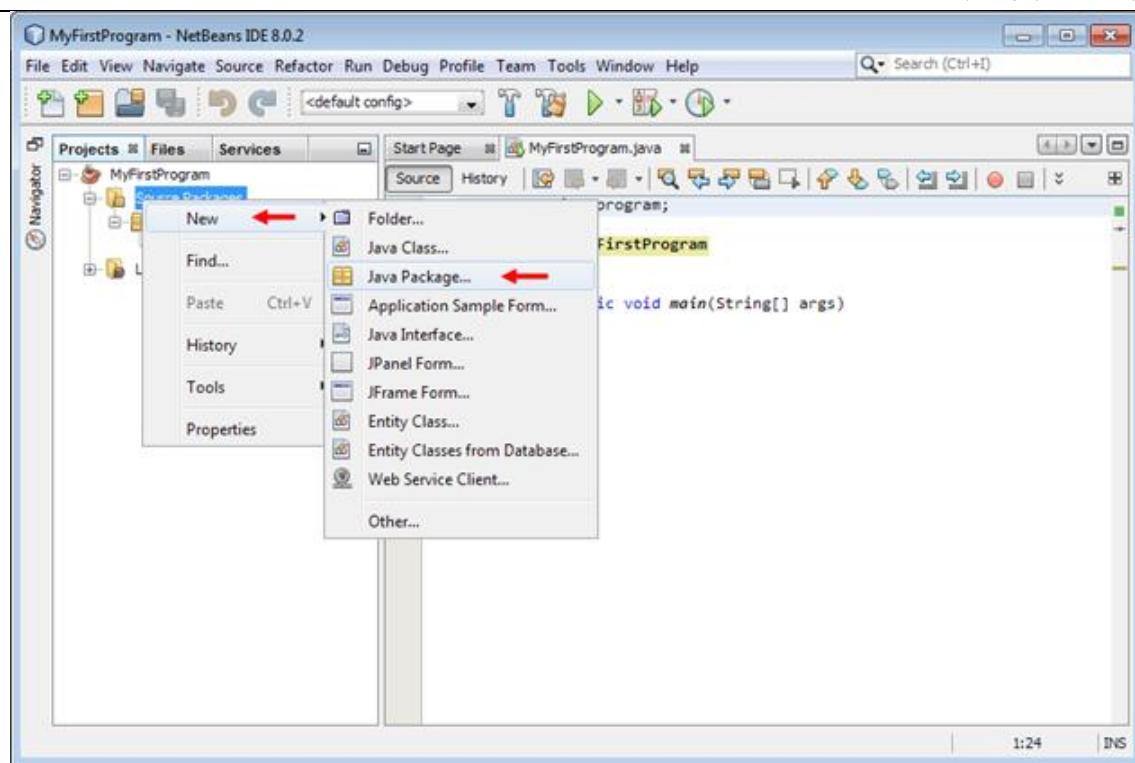
مشاهده می‌کنید که در کادری پایین‌تر از آن بسته به نام پروژه یک Package به همراه نام کلاس ایجاد می‌کند :



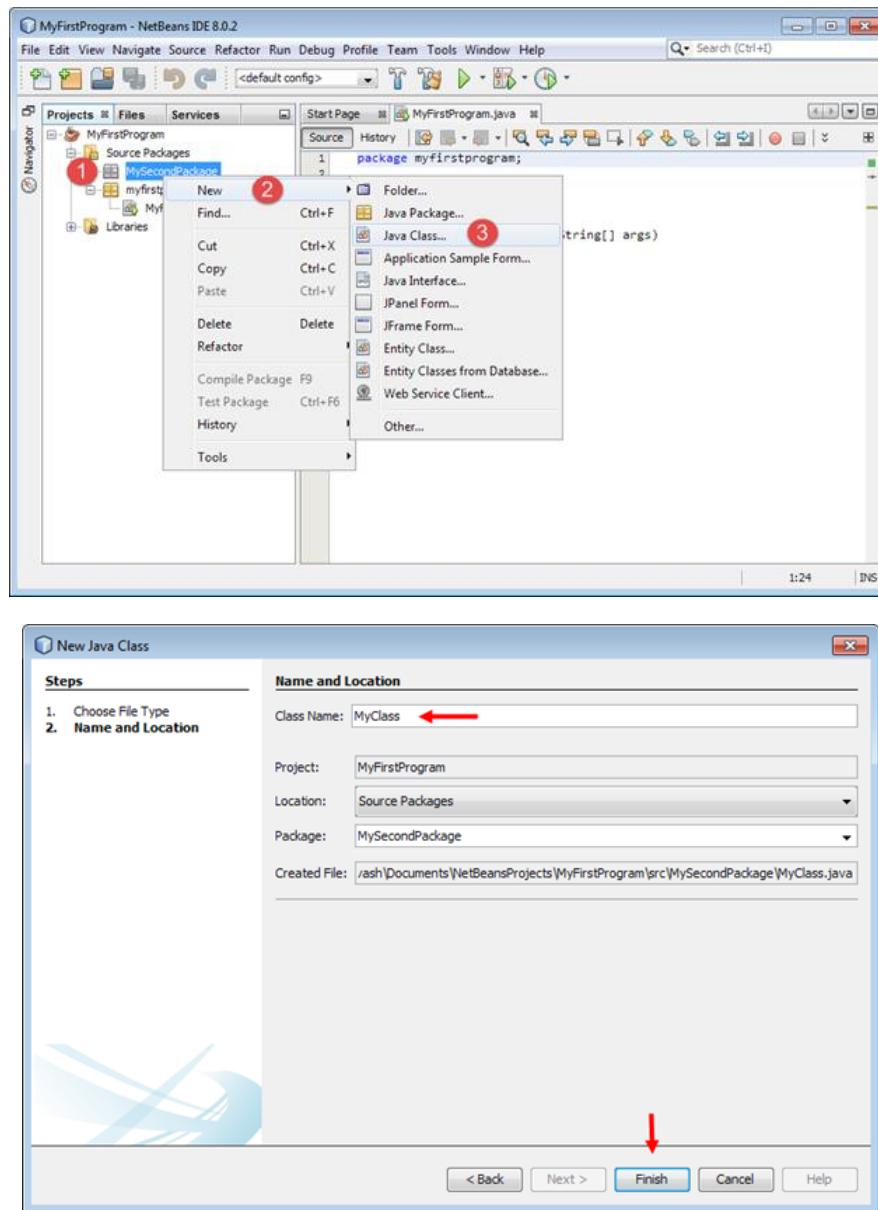
بعد از زدن دکمه finish در شکل بالا، یک class به صورت زیر ایجاد می‌شود :



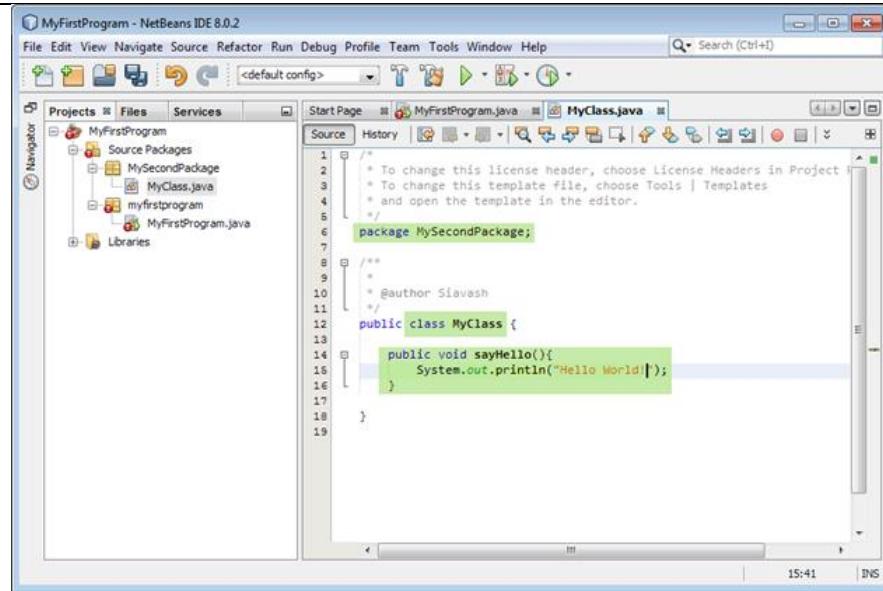
توجه کنید که پسوند کلاس در جاوا به صورت `.java` می‌باشد. پس تا اینجا ما یک Package و یک کلاس و یک متاداریم. حال می‌خواهیم یک پکیج دیگر ایجاد و از کلاس‌ها و متدهای آن در داخل این پکیج استفاده کنیم. برای این کار بر روی گزینه Source مانند شکل زیر راست کلیک کرده و یک پکیج جدید به نام `MySecondPackage` ایجاد می‌کنیم :



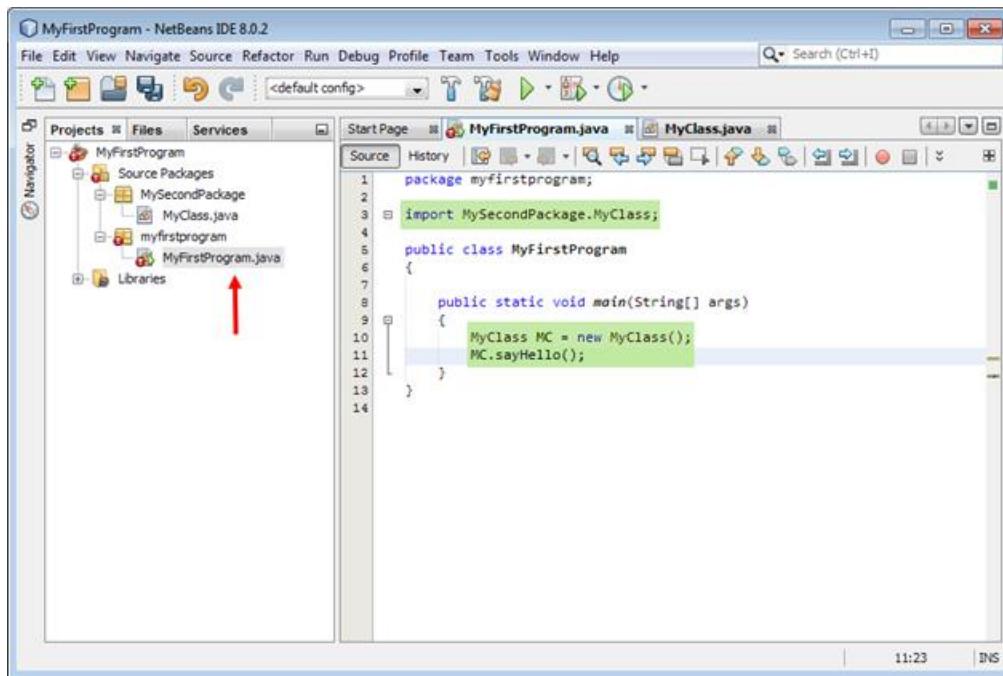
بعد از ایجاد این Package یک کلاس به نام MyClass به آن، به روش زیر اضافه می‌کنیم :



بعد از اضافه کردن کلاس یک متد به نام `sayHello()` به شکل زیر به کلاس اضافه نمایید:



حال فرض کنید که میخواهید از این کلاس و متده استفاده کنید. برای این کار بر روی کلاس مذکور دو بار کلیک کرده و سپس مانند شکل زیر و با استفاده از کلمه `import` اولی وارد نمایید و سپس با ایجاد یک شیء از کلاس متد مربوط به آن را فراخوانی کنید :



پس در کل میتوان نتیجه گرفت که با استفاده از کلمه کلیدی `import` میتوان همه محتویات یک پکیج را در داخل پکیج دیگر وارد کرد.

اگر قصد وارد کردن فقط یک کلاس از یک پکیج را در داخل پکیج دیگر داشته باشیم به صورت زیر عمل میکنیم :

```
import Package.Class;
```

و اگر بخواهیم تمامی کلاس‌های یک Package را وارد Package دیگر کنیم به صورت زیر عمل می‌نماییم :

```
import Package.*;
```

اگر از کلمه import استفاده نکنیم مجبوریم که در ابتدای نام هر کلاس Package مربوط به آن را به صورت زیر ذکر کنیم :

```
MySecondPackage.MyClass MC = new MySecondPackage.MyClass();
```

شما محدود به دسته بندی کدهای کلاستان در داخل یک Package نیستید. می‌توانید یک Package تو در تو ایجاد کنید و کدهایتان را در درون آن بنویسید. برای دسترسی به کلاس Sample، مجبورید اول نام تمام Package هایی را که کلاس Sample در آن‌ها قرار دارد بنویسید.

```
Package1.Package2.Sample
```

یا می‌توان از کلمه کلیدی import استفاده کرد :

```
import Package1.Package2.Sample
```

وراثت

وراثت به یک کلاس اجازه می‌دهد که خصوصیات یا متدهایی را از کلاس دیگر به ارث برد. وراثت مانند رابطه پدر و پسری می‌ماند به طوریکه فرزند خصوصیاتی از قبیل قیافه و رفتار را از پدر خود به ارث بردہ باشد.

- کلاس پایه یا کلاس والد کلاسی است که بقیه کلاس‌ها از آن ارث می‌برند.

- کلاس مشتق یا کلاس فرزند کلاسی است که از کلاس پایه ارث بری می‌کند.

همه متدهای خصوصیات کلاس پایه می‌توانند در کلاس مشتق مورد استفاده قرار بگیرند به استثنای اعضا و متدهای با سطح دسترسی private. همه کلاس‌ها در جاوا از کلاس Object ارث بری می‌کنند. مفهوم اصلی وراثت در مثال زیر نشان داده شده است :

```
1 package myfirstprogram;
2
3 class Parent
4 {
5     private String message;
6
7     public void setMessage(String message)
```

```

8
9     {
10        this.message = message;
11    }
12
13    public String getMessage()
14    {
15        return message;
16    }
17
18    public void ShowMessage()
19    {
20        System.out.println(message);
21    }
22
23    public Parent(String message)
24    {
25        this.message = message;
26    }
27
28    class Child extends Parent
29    {
30        public Child(String message)
31        {
32            super(message);
33        }
34    }

```

در این مثال دو کلاس با نامهای `Child` و `Parent` تعریف شده است. در این مثال یک فیلد را با سطح دسترسی `private` (خط ۵) و خاصیت مربوط به آن را با سطح دسترسی `public` (خط ۱۵-۲۷) تعریف کرده‌ایم. سپس یک متدهای `getMessage()` و `ShowMessage()` را برای نمایش پیام تعریف کرده‌ایم. یک سازنده در کلاس `Parent` تعریف شده است که یک آرگومان از نوع رشته قبول می‌کند و یک پیغام نمایش می‌دهد (خطوط ۲۲-۲۵). حال به کلاس `Child` توجه کنید (خط ۲۸-۳۴). این کلاس تمام متدها و خاصیت‌های کلاس `Parent` را به ارث برده است. نحوه ارث بری یک کلاس به صورت زیر است :

```
class DerivedClass extends BaseClass
```

براحتی می‌توان با قرار دادن کلمه کلیدی `extends` بعد از نام کلاس و سپس نوشتن نام کلاسی که از آن ارث بری می‌شود (کلاس پایه) این کار را انجام داد. در داخل کلاس `Child` هم یک سازنده ساده وجود دارد که یک آرگومان رشته‌ای قبول می‌کند. وقتی از وراثت در کلاس‌ها استفاده می‌کنیم، هم سازنده کلاس مشتق و هم سازنده پیشفرض کلاس پایه هر دو اجرا می‌شوند. سازنده پیشفرض یک سازنده بدون پارامتر است. اگر برای یک کلاس سازنده‌ای تعریف نکنیم کمپایلر به صورت خودکار یک سازنده برای آن ایجاد می‌کند. اگر هنگام

صدا زدن سازنده کلاس مشتق بخواهیم سازنده کلاس پایه را صدا بزنیم باید از کلمه کلیدی `super` استفاده کنیم. کلمه کلیدی `super` یک سازنده از کلاس پایه را صدا می‌زند.

در مثال بالا به وسیله تأمین مقدار پارامتر `message` سازنده کلاس مشتق و ارسال آن به داخل پرانتز کلمه کلیدی `super`، سازنده معادل آن در کلاس پایه فراخوانی شده و مقدار `message` را به آن ارسال می‌کند. سازنده کلاس `Parent` هم این مقدار (مقدار `message`) را در `Parent` یک عضو داده‌ای (`private`) قرار می‌دهد. می‌توانید کدهایی را به داخل بدنه سازنده `Child` اضافه کنید تا بعد از سازنده `Parent` و `Child` بسازیم تا نشان دهیم که چگونه کلاس `Child` متدها و خواص کلاس `Parent` را به ارث می‌برد.

```

1  public class MyFirstProgram
2  {
3
4      public static void main(String[] args)
5      {
6          Parent myParent = new Parent("Message from parent.");
7          Child myChild = new Child("Message from child.");
8
9          myParent.ShowMessage();
10
11         myChild.ShowMessage();
12
13         myParent.setMessage("Modified message of the parent.");
14         myParent.ShowMessage();
15
16         myChild.setMessage("Modified message of the child.");
17         myChild.ShowMessage();
18
19         //myChild.message; ERROR: can't access private members of base class
20     }
21 }
```

```

Message from parent.
Message from child.
Modified message of the parent.
Modified message of the child.
```

هر دو شیء را با استفاده از سازنده‌های مربوط به خودشان مقدار دهی می‌کنیم (خطوط ۶-۷). سپس با استفاده از ارث بری و از طریق شیء `Child` به اعضا و متدهای کلاس `Parent` دسترسی می‌یابیم. حتی اگر کلاس `Child` از کلاس `Parent` ارث برد باز هم اعضای با سطح دسترسی `private` در کلاس `Child` قابل دسترسی نیستند (خط ۱۸). سطح دسترسی `Protect` که در درس آینده توضیح داده خواهد شد به شما اجازه دسترسی به اعضا و متدهای کلاس پایه را می‌دهد. به نکته دیگر توجه کنید. اگر کلاس دیگری بخواهد از کلاس `Child` ارث بری کند، باز هم تمام متدها و خواص کلاس `Child` که از کلاس `Parent` به ارث برده است را به ارث می‌برد.

```
class GrandChild extends Child
{
    //Empty Body
}
```

این کلاس هیچ چیزی در داخل بدن ندارد. وقتی کلاس GrandChild را ایجاد می‌کنید و یک خاصیت از کلاس Parent را فراخوانی می‌کنید با خطأ مواجه می‌شوید. چون هیچ سازنده‌ای قبول کند در داخل بدن GrandChild تعریف نشده است بنابراین شما می‌توانید فقط از سازنده پیشفرض یا بدون پارامتر استفاده کنید.

```
GrandChild myGrandChild = new GrandChild();
myGrandChild.setMessage("Hello my grandchild!");
myGrandChild.ShowMessage();
```

وقتی یک کلاس ایجاد می‌کنیم و سازنده GrandChild می‌کنیم ابتدا سازنده کلاس Parent فراخوانی می‌شود و سپس سازنده GrandChild اجرا می‌شود. برنامه زیر ترتیب اجرای سازنده‌ها را نشان می‌دهد. دوباره کلاس‌ها را برای خوانایی بیشتر در داخل کدهای جدا قرار می‌دهیم.

```
1 package myfirstprogram;
2
3 class Parent
4 {
5     public Parent()
6     {
7         System.out.println("Parent constructor was called!");
8     }
9 }
10
11 class Child extends Parent
12 {
13     public Child()
14     {
15         System.out.println("Child constructor was called!");
16     }
17 }
18
19 class GrandChild extends Child
20 {
21     public GrandChild()
22     {
23         System.out.println("GrandChild constructor was called!");
24     }
25 }
26
27 public class MyFirstProgram
28 {
29 }
```

```

30   public static void main(String[] args)
31   {
32     GrandChild myGrandChild = new GrandChild();
33   }
34 }
```

```

Parent constructor was called!
Child constructor was called!
GrandChild constructor was called!
```

سطح دسترسی Protect

سطح دسترسی `protect` اجازه می‌دهد که اعضای کلاس، فقط در کلاسهای مشتق شده از کلاس پایه قابل دسترسی باشند. بدیهی است که خود کلاس پایه هم می‌تواند به این اعضا دسترسی داشته باشد. کلاس‌هایی که از کلاس پایه ارث بری نکرده‌اند نمی‌توانند به اعضای با سطح دسترسی `protect` یابند. در مورد سطوح دسترسی `public` و `private` قبلًا توضیح دادیم. در جدول زیر نحوه دسترسی به سه سطح ذکر شده نشان داده شده است :

قابل دسترسی در	public	private	protected
داخل کلاس	true	true	true
خارج از کلاس	true	false	false
کلاس مشتق	true	false	true

مشاهده می‌کنید که `public` بیشترین سطح دسترسی را دارد. صرف نظر از مکان، اعضای `public` در هر جا فراخوانی می‌شوند و قابل دسترسی هستند. اعضای `private` فقط در داخل کلاسی که به آن تعلق دارند قابل دسترسی هستند. کد زیر رفتار اعضای دارای این سطح دسترسی را نشان می‌دهد :

```

1 package myfirstprogram;
2
3 class Parent
4 {
5   protected int protectedMember = 10;
6   private  int privateMember   = 10;
7   public   int publicMember    = 10;
8 }
9 }
```

```

10 class Child extends Parent
11 {
12     public Child()
13     {
14         protectedMember = 100;
15         privateMember = 100;
16         publicMember = 100;
17     }
18 }
19
20 public class MyFirstProgram
21 {
22     public static void main(String[] args)
23     {
24         Parent myParent = new Parent();
25
26         myParent.protectedMember = 100;
27         myParent.privateMember = 100;
28         myParent.publicMember = 100;
29     }
30 }
```

کدهایی که با خط قرمز نشان داده شده‌اند نشان دهنده وجود خطأ هستند چون آن‌ها اجازه دسترسی به فیلدهای `protect` کلاس `Parent` را ندارند. همانطور که در خط ۱۵ مشاهده می‌کنید کلاس `Child` سعی می‌کند که به عضو `private` کلاس `Parent` دسترسی یابد. از آنجاییکه اعضای `private` در خارج از کلاس قابل دسترسی نیستند، حتی کلاس مشتق در خط ۱۵ نیز ایجاد خطأ می‌کند. اگر شما به خط ۱۴ توجه کنید کلاس `Child` می‌تواند به عضو `protect` کلاس `Parent` دسترسی یابد چون کلاس `Child` از کلاس `Parent` مشتق شده است.

حال به خط ۲۶ جاییکه می‌خواهیم در کلاس `MyFirstProgram` به فیلد `protect` کلاس `Parent` دسترسی یابیم نگاهی بیندازید. می‌بینید که برنامه پیغام خطأ می‌دهد چون کلاس `MyFirstProgram` از کلاس `Parent` مشتق نشده است. همچنین کلاس `MyFirstProgram` به اعضای `private` کلاس `Parent` نیز نمی‌تواند دسترسی یابد.

اعضای `static`

اگر بخواهیم عضو داده‌ای (فیلد) یا خاصیتی ایجاد کنیم که در همه نمونه‌های کلاس قابل دسترسی باشد از کلمه کلیدی `static` استفاده می‌کنیم. کلمه کلیدی `static` برای اعضای داده‌ای و خاصیت‌هایی به کار می‌رود که می‌خواهند در همه نمونه‌های کلاس تقسیم شوند. وقتی که یک متدها یا خاصیت به صورت `static` تعریف شود، می‌توانید آن‌ها را بدون ساختن نمونه‌ای از شی، فراخوانی کنید. به چند مثال توجه کنید :

```

1 package myfirstprogram;
2
3 class SampleClass
4 {
5     public static String StaticMessage = "This is the static message!";
6 }
7
8 public class MyFirstProgram
9 {
10    public static void main(String[] args)
11    {
12        System.out.println(SampleClass.StaticMessage );
13    }
14 }

```

This is the static message!

در مثال بالا یک شیء استاتیک به نام `StaticMessage` (خط ۵) تعریف کردہ‌ایم. مقدار شیء `StaticMessage` در همه نمونه‌های کلاس `SampleClass` قابل دسترسی است. برای فراخوانی یک متدهای خاصیت و یا یک متغیر استاتیک، به سادگی می‌توان نام کلاس و بعد از آن علامت دات (.) و در آخر نام متدهای خاصیت را نوشت. این موضوع را می‌توان در خط (۱۲) مشاهده می‌کنید که لازم نیست هیچ نمونه‌ای از کلاس ایجاد شود. یکی دیگر از کاربردهای این کلمه کلیدی در شمارش اشیاء است. به مثال زیر توجه کنید :

```

1 package myfirstprogram;
2
3 class SampleClass
4 {
5     public static int number = 1;
6
7     public SampleClass()
8     {
9         System.out.println("Number is : " + number++);
10    }
11 }
12
13 public class MyFirstProgram
14 {
15     public static void main(String[] args)
16     {
17         SampleClass Sample1 = new SampleClass ();
18         SampleClass Sample2 = new SampleClass ();
19         SampleClass Sample3 = new SampleClass ();
20     }
21 }

```

Number is : 1
Number is : 2
Number is : 3

همانطور که در خط ۵ کد بالا مشاهده می‌کنید یک متغیر استاتیک با مقدار اولیه ۱ ایجاد کرده‌ایم و در داخل سازنده کلاس در خط ۹ ابتدا مقدار آن را چاپ و سپس یک واحد به آن اضافه کرده‌ایم. حال در خطوط ۱۷-۱۹ سه شیء از روی کلاس SampleClass ایجاد می‌کنیم. همانطور که در خروجی مشاهده می‌کنید با هر بار ایجاد شیء یک بار سازنده کلاس فراخوانی و در نتیجه مقدار متغیر چاپ می‌شود. یکی از خواص متغیرهای استاتیک این است که مقدار قبلی خود را حفظ می‌کنند. و از این خاصیت در مثال بالا برای شمارش اشیاء ساخته شده از کلاس استفاده کرده‌ایم.

Override

فرض کنید شما متند A را در کلاس A دارید و کلاس B از کلاس A ارث بری می‌کند، در این صورت متند A در کلاس B در دسترس خواهد بود. اما متند A دقیقاً همان متندی است که از کلاس A به ارث برده شده است. حال اگر بخواهید که این متند رفتار متفاوتی از خود نشان دهد چکار می‌کنید؟ برای حل این مشکل باید متند کلاس پایه را **Override** کنید. به تکه کد زیر توجه کنید:

```

1 package myfirstprogram;
2
3 class Parent
4 {
5     public void ShowMessage()
6     {
7         System.out.println("Message from Parent.");
8     }
9 }
10
11 class Child extends Parent
12 {
13     public void ShowMessage()
14     {
15         System.out.println("Message from Child.");
16     }
17 }
18
19 public class MyFirstProgram
20 {
21     public static void main(String[] args)
22     {
23         Parent myParent = new Parent();
24         Child myChild = new Child();
25
26         myParent.ShowMessage();
27         myChild.ShowMessage();
28     }
29 }
```

```
Message from Parent.
Message from Child.
```

همانطور که در کد بالا مشاهده می‌کنید دو کلاس یه نام Parent (خطوط ۹-۱۱) و Child (خطوط ۱۲-۱۳) تعریف کرده‌ایم. کلاس

که از کلاس Parent ارث می‌برد شامل متدی است که متد () ShowMessage() از کلاس پایه را override یا به صورت دیگری پیاده سازی می‌کند. همانطور که مشاهده می‌کنید این دو متد دقیقاً شبیه به هم هستند و تنها اختلاف آن‌ها در پیامی است که نشان می‌دهند. برای کردن یک متد قواعدی وجود دارد که در زیر به آن‌ها اشاره شده است :

- تابع override شده یکی باشد.
- نوع مقدار بازگشتی تابع باید همانند یا فرزندی از نوع مقدار بازگشتی تابع override شده کلاس پدر باشد.
- سطح دسترسی تابع نمی‌تواند محدودتر از سطح دسترسی تابع override شده باشد. برای مثال: اگر تابع کلاس پدر به صورت public تعریف شده باشد، در این صورت تابع کلاس فرزند نمی‌تواند protected یا private باشد.
- تنها توابعی از کلاس پدر که توسط کلاس فرزند ارث بری شده‌اند می‌توانند override شوند.
- توابع static نمی‌توانند override شوند، ولی می‌توانند دوباره در کلاس فرزند تعریف شوند.
- اگر تابعی نمی‌تواند ارث برد شود، همانطور هم نمی‌تواند override شود.
- کلاس فرزند موجود در پکیج یکسان با کلاس پدر، می‌تواند تمامی توابعی از کلاس پدر را که به صورت private یا final تعریف نشده باشند را override کند.
- کلاس فرزند در یک پکیج دیگر از کلاس پدر تنها می‌تواند توابع public یا protected کلاس پدر را که final نیستند را override کند.
- سازنده‌ها نمی‌توانند override شوند.

با استفاده از کلمه کلیدی super (خط ۱۵) می‌توانید متد کلاس پایه را در داخل متد override شده فراخوانی کنید:

```

1 package myfirstprogram;
2
3 class Parent
4 {
5     public void ShowMessage()
6     {
7         System.out.println("Message from Parent.");
8     }
9 }
10
11 class Child extends Parent
12 {
13     public void ShowMessage()
14     {
15         super.ShowMessage();
16     }
17 }
```

```

15     super.ShowMessage();
16     System.out.println("Message from Child.");
17 }
18 }
19
20 public class MyFirstProgram
21 {
22     public static void main(String[] args)
23     {
24         Parent myParent = new Parent();
25         Child myChild = new Child();
26
27         myParent.ShowMessage();
28         myChild.ShowMessage();
29     }
30 }
```

```

Message from Parent.
Message from Parent.
Message from Child.
```

م توان یک کلاس دیگر که از کلاس Child ارث بری می‌کند ایجاد کرده و دوباره متده را به override از ShowMessage() کرده و آنرا به

صورت دیگر پیاده سازی کنیم. اگر بخواهید متده را که ایجاد کردہاید به وسیله سایر کلاس‌ها override نشود کافیست که از کلمه کلیدی

به صورت زیر استفاده کنید :

```
public final void ShowMessage()
```

حال اگر کلاس دیگری از کلاس Child ارث ببرد نمی‌تواند متده را override از ShowMessage() کند.

کلاس آبجکت (java.lang.Object)

همه کلاس‌های جاوا به طور مستقیم یا غیر مستقیم از کلاس java.lang.Object قرار دارد، ارث می‌برند. کلاس آبجکت در جاوا با کلمه کلیدی Object نشان داده می‌شود. برای راحتی در این درس از کلمه آبجکت به جای java.lang.Object استفاده می‌کنیم.

در زیر لیست برخی از متدهای معمول در کلاس آبجکت آمده است :

متده	کاربرد
یک شیء از کلاس Class را بر می‌گرداند که نشان دهنده کلاس زمان اجرای شیء است.	getClass()
کد هش شیء انتخاب شده بر می‌گرداند.	hashCode()
مشخص می‌کند که آیا یک شیء با شیء دیگر برابر است یا نه؟	equals()

یک نسخه از شیء جاری را بر می‌گرداند.	<code>clone()</code>
یک نمایش رشته‌ای از شیء جاری را بر می‌گرداند.	<code>toString()</code>
با برنامه‌های چند نخی به کار می‌رود و نخ منتظر شیء جاری را فعال می‌کند.	<code>notify()</code>
با برنامه‌های چند نخی به کار می‌رود و تمام نخ‌های منتظر شیء جاری را فعال می‌کند.	<code>notifyAll()</code>
باعث می‌شود نخ شیء منتظر شود تا نخ دیگر <code>notifyAll</code> یا <code>notify</code> را فراخوانی کند.	<code>wait()</code>
با ابزار زباله روب (Garbage Collector) در زمان از بین بردن شبیه فراخوانی می‌شود.	<code>finalize()</code>

همه متدهای این کلاس معمولاً مورد استفاده قرار نمی‌گیرند. از آنجاییکه همه کلاسهای جاوا از این کلاس ارث می‌برند، آن‌ها نیز دارای این متدها به جز متدهای `Static` می‌باشند. وقتی یک کلاس ایجاد می‌کنید، این کلاس به صورت ضمنی از کلاس آبجکت ارث می‌برد. بنابراین وقتی یک کلاس تعریف می‌کنید کدها در حقیقت به صورت زیر به وسیله کمپایلر خوانده می‌شوند :

```
class MyClass extends java.lang.Object
{}
```

اینکه چرا همه کلاسهای در جاوا از `Object` ارث بری می‌کنند به دلیل امکان استفاده از چندریختی است که در درس آینده درباره آن توضیح می‌دهیم. به عنوان مثال یکی از سربارگذاری‌های متدهای `System.out.println()` قبول نوع آبجکت به عنوان آرگومان است. به همین دلیل است که شما می‌توانید تقریباً هر چیز را به عنوان آرگومان به متدهای `System.out.println()` ارسال کنید. نکته‌ای که باید در اینجا به آن اشاره کنیم این است که انواع داده‌های اصلی شیء نیستند و از کلاس `Object` ارث بری نمی‌کنند و در نتیجه نمی‌توانند از متدهای این کلاس هم استفاده کنند. مثلاً اجرای کد زیر باعث بروز خطای می‌شود :

```
int number = 100;
System.out.println(number.toString());
```

اگر بخواهید یک نوع `int` را به نوع رشته‌ای به صورت بالا تبدیل کنید و متدهای `toString()` را فراخوانی کنید باید از کلاس پوششی به صورت زیر استفاده نمایید :

```
int number = 100;
System.out.println(new Integer(number).toString());
```

کد بالا را به صورت زیر هم می‌توانید بنویسید :

```
int number = 100;
System.out.println(Integer.toString(number));
```

همانطور که در بالا هم اشاره شد یکی از متدهای کلاس `Object` متدهای `toString()` است. این متدهای `toString()` هش شده شیء جاری را بر

می‌گرداند. به مثال زیر توجه کنید :

```
package myfirstprogram;

class Person
{
    private String name, family;

    public Person(String n, String f)
    {
        this.name = n;
        this.family = f;
    }

    public class MyFirstProgram
    {

        public static void main(String[] args)
        {
            Person person1 = new Person("John", "SKith");

            System.out.println(person1.toString());
        }
    }
}
```

object.Person@5cd622e5

همانطور که مشاهده می‌کنید با فراخوانی متدهای `toString()` یک کد هش شده تولید می‌شود. البته این کد ممکن است در سیستم شما

متفاوت باشد. رفتار پیشفرض متدهای `toString()` به صورت زیر است :

```
public String toString()
{
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

خروجی این متدهای `toString()` یک کد هش شده است ولی شاید شما بخواهید این رفتار پیشفرض را تغییر داده و متدهای `toString()` را اصطلاحاً `Override` کنید.

فرض کنید در مثال بالا شما می‌خواهید با فراخوانی این متدهای `toString()` مقدار `hashCode()` این کلاس چاپ شوند :

```

1 package myfirstprogram;
2
3 class Person
4 {
5     private String name, family;
6
7     public Person(String n, String f)
8     {
9         this.name = n;
10        this.family = f;
11    }
12
13    @Override
14    public String toString()
15    {
16        return name + " " + family;
17    }
18}
19
20 public class MyFirstProgram
21 {
22     public static void main(String[] args)
23     {
24         Person person1 = new Person("John", "SKith");
25
26         System.out.println(person1.toString());
27     }
28 }
```

John SKith

همانطور که مشاهده می‌کنید با `toString` متد () کردن متده `Override` در خطوط ۱۳-۱۶ و تغییر رفتار پیشفرض آن (خط ۱۵) باعث شدیم که با فراخوانی آن در خط ۲۵ خروجی متفاوتی را دریافت کیم. خط ۲۵ را به صورت زیر هم می‌توان نوشت :

```
System.out.println(person1);
```

این بدان خاطر است که متده `toString` متد () `println` شی ارسال شده به آن را به طور خودکار فراخوانی می‌کند.

Unboxing و Boxing

در زبان برنامه نویسی جاوا، ۸ نوع داده اصلی وجود دارد که هر کدام از آنها دارای یک کتابخانه کلاس (`library class`) از نوع مرجع، مربوط به خود هستند. به عنوان مثال کلاس `java.lang.Integer` کتابخانه مربوط به نوع اصلی `int` می‌باشد. این نوع کلاس‌ها را `wrapper class` یا کلاس‌های پوششی می‌نامند. این نوع کلاس‌ها غیر قابل تغییر بوده، و وقتی ایجاد می‌شوند نمی‌توان مقدار آنها را عوض کرد. همچنین چون از نوع `final` هستند، در نتیجه نمی‌توانند دارای زیر کلاس هم باشند. در جدول زیر انواع اصلی و کلاس‌های

مربوط به آنها آمده است :

نوع اصلی	نوع مرجع
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

قبل از 5 SE Java برای مقدار دهی به یک متغیر از نوع Integer باید حتماً یک شی از کلاس Integer ایجاد می‌کردیم و همچنین برای مقدار دهی به یک متغیر از نوع int با استفاده از یک شی Integer باید حتماً از متدها intValue() و `new Integer(intValue)` استفاده می‌کردیم، اما بعد از 5 SE و با افزوده شدن قابلیت Boxing و Unboxing این مشکل برطرف و به طور خودکار این تبدیل‌ها انجام شد.

فرایندی است که طی آن یک نوع اصلی مانند int به یک نوع مرجع مانند Integer تبدیل می‌شود. عکس، عمل تبدیل یک نوع مرجع به یک نوع مقداری می‌باشد. کد زیر فرایند boxing را نشان می‌دهد:

```
int i = 100;
Integer number = i;
```

در پس زمینه، کمپایلر کد بالا را به صورت زیر به نوع مرجع تبدیل می‌کند:

```
Integer number = Integer.valueOf(i);
```

در فرایند boxing نوع مقداری به سادگی با یک متغیر از نوع مرجع برابر قرار داده می‌شود. در زیر نحوه تبدیل یک نوع مرجع به نوع مقداری به وسیله unboxing نشان داده شده است.

```
Integer number = 100;
int i = number;
```

در پس زمینه، کمپایلر کد بالا را به صورت زیر به نوع اصلی تبدیل می‌کند:

```
Integer number= Integer.valueOf(100);
int i = number.intValue();
```

aggregation

فرایندی است که طی آن یک کلاس به عنوان یک عضو به کلاس دیگر اضافه می‌شود. به عنوان مثال کلاس Person

می‌تواند یک فیلد از نوع کلاس Name داشته باشد. به کد زیر توجه کنید:

```
1 package myfirstprogram;
2
3 class Name
4 {
5     public String FirstName;
6     public String LastName;
7
8     public Name(String f, String l)
9     {
10         this.FirstName = f;
11         this.LastName = l;
12     }
13 }
14
15 class Person
16 {
17     private Name myName;
18
19     public Person(Name name)
20     {
21         myName = new Name(name.FirstName, name.LastName);
22     }
23
24     @Override
25     public String toString()
26     {
27         return myName.FirstName + " " + myName.LastName;
28     }
29 }
30
31 public class MyFirstProgram
32 {
33     public static void main(String[] args)
34     {
35         Person person1 = new Person(new Name("John", "Smith"));
36
37         System.out.println(person1.toString());
38     }
}
```

39 }

John Smith

حال برنامه را به صورت بخش بخش توضیح می‌دهیم :

```
class Name
{
    public String FirstName;
    public String LastName;

    public Name(String f, String l)
    {
        this.FirstName = f;
        this.LastName = l;
    }
}
```

یک کلاس که قرار است به عنوان یک فیلد در کلاس دیگر به کار رود را تعریف می‌کنیم (خطوط ۱۳-۱۴). این کلاس دارای یک سازنده است که نام (f) و نام خانوادگی (l) را از شخص دریافت می‌کند. سپس این مقادیر را در خصوصیت‌های متناظر با آنها قرار می‌دهیم (خطوط ۱۵-۱۶).

```
class Person
{
    private Name myName;

    public Person(Name name)
    {
        myName = new Name(name.FirstName, name.LastName);
    }

    @Override
    public String toString()
    {
        return myName.FirstName + " " + myName.LastName;
    }
}
```

این کلاس شامل یک فیلد از نوع Name و خاصیت متناظر با آن است. این خصوصیت مقدار نام هر شیء Person را در خود نگهداری می‌کند. به این نکته توجه کنید که سازنده یک شیء Name را می‌پذیرد و سپس با استفاده از این شیء فیلد myName را مقداردهی می‌کند. به این فرایند ترکیب (aggregation) می‌گویند. همچنین در کلاس Person متد () از کلاس Object را بازنویسی override می‌کنیم به طوری که در هنگام فراخوانی نام کامل شخص را نمایش دهد.

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Person person1 = new Person(new Name("John", "Smith"));

        System.out.println(person1.toString());
    }
}
```

در بالا یک شیء Person را ایجاد و از سازندهای که یک شیء از نوع کلاس Name به عنوان آرگومان قبول می‌کند استفاده کرده‌ایم. این شیء را مستقیماً در داخل پرانتزها تعریف و همچنین مقادیر خصوصیات lastname و firstname را به آن ارسال کرده‌ایم. در نهایت، با استفاده از متده سفارشی `toString()` نتیجه را چاپ می‌کنیم. به این نکته توجه کنید که کلاس‌ها می‌توانند شیء‌هایی از نوع خود کلاس داشته باشند. به عنوان نمونه کلاس Person می‌تواند یک عضو با نوع Person داشته باشد. به کد زیر توجه نمایید :

```
class Person
{
    public Person Sibling;
    public String Name;
}
```

مشاهده می‌کنید که چگونه خصوصیتی از نوع Person در داخل کلاس Person تعریف شده است. پس هنگامی که یک شیء Person تعریف می‌کنید، شیء ایجاد شده یک شیء از نوع Person در داخل خود دارد.

```
Person person1      = new Person();
person1.Sibling     = new Person();
person1.Name        = "John Smith";
person1.Sibling.Name = "Mike Smith";
```

کد بالا چگونگی دسترسی و مقدار دهی به عضو Person(Sibling) را نشان داده است. از آنجایی که خصوصیت sibling از نوع Person است پس می‌تواند یک شیء در داخل خود داشته باشد. بنابراین می‌توان هر تعداد sibling که می‌خواهید در داخل person1 داشته باشید :

```
person1.Sibling.Sibling = new Person();
person1.Sibling.Sibling.Name = "Franc Smith";
person1.Sibling.Sibling.Sibling = new Person();
person1.Sibling.Sibling.Sibling.Name = "Bob Smith";
//And so on...
```

عملگر instanceof

عملگر instanceof در جاوا به شما اجازه می‌دهد که تست کنید که آیا یک شیء یک نمونه از یک نوع خاص (کلاس، زیر کلاس، اینترفیس) است یا نه. عملگر instanceof به دو عملوند نیاز دارد و یک مقدار بولی را بر می‌گرداند. به عنوان مثال، فرض کنید یک کلاس به نام Animal داریم، سپس یک نمونه از آن ایجاد می‌کنیم :

```

1 package myfirstprogram;
2
3 class Animal
4 {
5 }
6
7
8 public class MyFirstProgram
9 {
10     public static void main(String[] args)
11     {
12         Animal myAnimal = new Animal();
13
14         if (myAnimal instanceof Animal)
15         {
16             System.out.println("myAnimal is an Animal!");
17         }
18     }
19 }
```

myAnimal is an Animal

رفتار عملگر instanceof را در این مثال مشاهده کردید. همانطور که می‌بینید از آن به عنوان شرط در عبارت if استفاده شده است. کاربرد آن در مثال بالا این است که چک می‌کند که آیا شیء myAnimal یک نمونه از Animal است و چون نتیجه درست است کدهای داخل دستور if اجرا می‌شود. این عملگر همچنین می‌تواند چک کند که آیا یک شیء خاص در سلسله مراتب وراثت یک نوع خاص است.

به این مثال توجه کنید :

```

1 package myfirstprogram;
2
3 class Animal
4 {
5 }
6
7
8 class Dog extends Animal
9 {
10 }
11
12
13 public class MyFirstProgram
14 {
15     public static void main(String[] args)
```

```

16 {
17     Dog myDog = new Dog();
18
19     if (myDog instanceof Animal)
20     {
21         System.out.println("myDog is an Animal!");
22     }
23 }
24

```

```
myDog is an Animal!
```

همانطور که در مثال بالا می‌بینید ما یک کلاس به نام Dog ایجاد کرده‌ایم که از کلاس Animal ارث می‌برد. سپس یک نمونه از این کلاس (Dog) ایجاد می‌کنیم و سپس با استفاده از عملگر instanceof تست می‌کنیم که آیا نمونه ایجاد شده جز کلاس Animal است یا یک کلاس مشتق شده از کلاس Animal می‌باشد. از آنجاییکه کلاس Dog از کلاس Animal ارث می‌برد (سگ من یک حیوان است)، نتیجه عبارت درست (true) است. حال جمله بالا را تغییر دهیم: "حیوان من یک سگ است". وقتی جمله برعکس می‌شود چه اتفاقی می‌افتد؟

```

Animal myAnimal = new Animal();
if (myAnimal instanceof Dog)
{
    System.out.println("myAnimal is a Dog!");
}

```

این باعث خطا نمی‌شود و عبارت فقط نتیجه false را بر می‌گرداند. می‌توان از کد بالا درک کرد که همه حیوانات سگ نیستند ولی همه سگ‌ها حیوان هستند.

رابط (Interface)

اینترفیس‌ها شبیه به کلاس‌ها هستند اما فقط شامل تعاریفی برای متدها و خواص (Property) می‌باشند. اینترفیس‌ها را می‌توان به عنوان پل‌گین‌های کلاس‌ها در نظر گرفت. کلاسی که یک اینترفیس خاص را پیاده سازی می‌کند لازم است که کدهایی برای اجرا توسط اعضاء و متدهای آن فراهم کند چون اعضاء و متدهای اینترفیس هیچ کد اجرایی در بدنه خود ندارند. اجازه دهید که نحوه تعریف و استفاده از یک اینترفیس در کلاس را توضیح دهیم :

```

1 package myfirstprogram;
2
3 interface ISample
4 {
5     public void ShowMessage(String message);
6 }
7

```

```

8  class Sample implements ISample
9  {
10     public void ShowMessage(String message)
11     {
12         System.out.println(message);
13     }
14 }
15
16 public class MyFirstProgram
17 {
18     public static void main(String[] args)
19     {
20         Sample sample = new Sample();
21
22         sample.ShowMessage("Implemented the ISample Interface!");
23     }
24 }
```

Implemented the ISample Interface!

در خطوط ۳-۶ یک اینترفیس به نام `ISample` تعریف کردہ‌ایم. بر طبق قراردادهای نامگذاری، اینترفیس‌ها به شیوه پاسکال نامگذاری می‌شوند و همه آن‌ها باید با حرف `I` شروع شوند. همچنین در تعریف آن‌ها باید از کلمه کلیدی `interface` استفاده شود. یک متدهای داخل بدن اینترفیس تعریف می‌کنیم (خط ۵). به این نکته توجه کنید که متدهای تعریف شده فاقد بدن است و در آخران باید از سیمیکولون استفاده شود. وقتی که متدهای داخل اینترفیس تعریف می‌کنیم فقط لازم است که عنوان متدهای (نوع، نام و پارامترهای آن) را بنویسید. به این نکته نیز توجه کنید که متدهای خواص تعریف شده در داخل اینترفیس سطح دسترسی ندارند چون باید همیشه هنگام اجرای کلاس‌ها در دسترس باشند. برای پیاده‌سازی یک کلاس از کلمه کلیدی `implements` استفاده می‌شود. کلاسی که اینترفیس را اجرا می‌کند کدهای واقعی را برای اعضای آن فراهم می‌کند. همانطور که در مثال بالا می‌بینید کلاس `Sample`، متدهای `ISample` را اجرا و تغذیه می‌کند. می‌توان چند اینترفیس را در کلاس اجرا کرد.

```

class Sample implements ISample1, ISample2, ISample3
{
    //Implement all interfaces
}
```

به مثال زیر توجه کنید :

```

1 package myfirstprogram;
2
3 interface IFirstInterface
4 {
5     public void FirstMessage(String message);
6 }
```

```

7
8 interface ISecondinterface
9 {
10     public void SecondMessage(String message);
11 }
12
13
14 class Sample implements IFirstinterface , ISecondinterface
15 {
16     @Override
17     public void FirstMessage(String message)
18     {
19         System.out.println(message);
20     }
21
22     @Override
23     public void SecondMessage(String message)
24     {
25         System.out.println(message);
26     }
27 }
28
29 public class MyFirstProgram
30 {
31     public static void main(String[] args)
32     {
33         Sample sample = new Sample();
34
35         sample.FirstMessage("Implemented the IFirstinterface Interface!");
36         sample.SecondMessage("Implemented the ISecondinterface Interface!");
37     }
38 }
```

Implemented the IFirstinterface Interface!
Implemented the ISecondinterface Interface!

درست است که می‌توان از چند اینترفیس در کلاس استفاده کرد ولی باید مطمئن شد که کلاس می‌تواند همه اعضای اینترفیسها را تغذیه کند. همانطور که در کد بالا مشاهده می‌کنید دو اینترفیس به نام‌های `IFirstinterface` و `ISecondinterface` در خطوط ۳-۱۱ در خوطو ۳-۱۱ تعریف شده‌اند که به ترتیب دارای دو متده به نام‌های `FirstMessage()` و `SecondMessage()` می‌باشند. در خط ۱۴ کلاس `Sample` این دو اینترفیس را پیاده سازی کرده است و در نتیجه همانطور که اشاره شد لازم است که کدهای بدنے دو متده موجود در این دو رابط را تغذیه کند. که این کار در خطوط ۱۷-۲۶ انجام شده است. عبارت `@Override` موجود در خطوط ۱۶ و ۲۲ به این دلیل قرار داده شده است که به کمپایلر اعلام کند که این دو متده `Override` شده‌اند و به نوعی یک اخطار به کمپایلر هست که می‌گوید این تابع مربوط به کلاس یا اینترفیسی است که کلاس جاری از آن مشتق شده و یا پیاده سازی کرده است. وجود `@` هم به خاطر قرار داد خود زبان می‌باشد. اگر یک کلاس از کلاس پایه ارث ببرد و در عین حال از اینترفیس‌ها هم استفاده کند، در این صورت باید نام کلاس پایه قبل از نام اینترفیس

ها ذکر شود. به شکل زیر :

```
class Sample extends BaseClass implements ISample1, ISample2
{
}
```

نکته دیگر اینکه نمی‌توان از یک اینترفیس نمونه‌ای ایجاد کرد چون اینترفیس‌ها دارای سازنده نیستند، مثلاً کد زیر اشتباه است :

```
ISample sample = new ISample();
```

اینترفیسها حتی می‌توانند از اینترفیس‌های دیگر با استفاده از کلمه کلیدی extends از ارث بری کنند. به مثال زیر توجه کنید :

```
1 package myfirstprogram;
2
3 interface IBase
4 {
5     void BaseMethod();
6 }
7
8 interface ISample extends IBase
9 {
10    void ShowMessage(String message);
11 }
12
13 class Sample implements ISample
14 {
15     @Override
16     public void ShowMessage(String message)
17     {
18         System.out.println(message);
19     }
20
21     @Override
22     public void BaseMethod()
23     {
24         System.out.println("Method from base interface!");
25     }
26 }
27
28 public class MyFirstProgram
29 {
30     public static void main(String[] args)
31     {
32         Sample sample = new Sample();
33
34         sample.ShowMessage("Implemented the ISample Interface!");
35         sample.BaseMethod();
36     }
37 }
```

```
Implemented the ISample Interface!
Method from base interface!
```

همانطور که در خط ۸ کد بالا مشاهده می‌کنید رابط IBase از رابط ISample ارث بری کرده است پس حتی اگر کلاس Sample فقط

اینترفیس ISample را پیاده سازی کند، لازم است که همه اعضای Ibase را هم پیاده سازی کند چون ISample از آن ارث بری می‌کند.

کلاسهای انتزاعی (Abstract Class)

کلاسهای مجرد (abstract) کلاس‌هایی هستند که کلاس پایه سایر کلاس‌ها هستند. این نوع کلاس‌ها می‌توانند مانند کلاسهای عادی دارای سازنده باشند. شما نمی‌توانید از کلاسهای انتزاعی نمونه ایجاد کنید چون که هدف اصلی از به کار بردن کلاسهای انتزاعی استفاده از آن‌ها به عنوان کلاس پایه برای کلاسهای مشتق است. برای تعریف یک کلاس انتزاعی از کلمه کلیدی abstract استفاده می‌شود. به مثال

زیر در مورد استفاده از کلاسهای انتزاعی توجه کنید :

```

1 abstract class Base
2 {
3     protected int    number;
4     protected String name;
5
6     public abstract void ShowMessage();
7
8     public Base(int number, String name)
9     {
10         this.number = number;
11         this.name   = name;
12     }
13 }
14
15 class Derived extends Base
16 {
17     @Override
18     public void ShowMessage()
19     {
20         System.out.println("Hello World!");
21     }
22
23     public Derived(int number, String name)
24     {
25         super(number, name);
26     }
27 }
```

در داخل کلاس انتزاعی دو فیلد محافظت شده (protected) تعریف کرده‌ایم (خطوط ۳-۴) که می‌خواهیم آن‌ها را توسط سازنده کلاس مقدار دهی کنیم (خطوط ۸-۱۲). یک متدهم را به صورت انتزاعی (abstract) تعریف کرده‌ایم (خط ۶). به این نکته توجه کنید که برای تعریف این متدهم کلمه کلیدی abstract را به کار بردہ‌ایم.

این متدهای باید به وسیله کلاس‌هایی که از این کلاس ارث می‌برند `override` یا به صورت دیگر پیاده سازی شود، ولی از آن جاییکه به صورت `abstract` تعریف شده است فاقد بدنی می‌باشد. می‌بینید که کلاس‌های `abstract` می‌توانند شامل `property`‌های معمولی مانند `Name`، `property` مثال بالا باشند. کلاس‌های `abstract` حداقل باید یک عضو `abstract` داشته باشند.

یک کلاس دیگر تعریف می‌کنید که از کلاس `Base` ارث بری کند. سپس در خطوط ۱۷-۲۱ متدهای `abstract` را به صورت دیگر پیاده سازی می‌کنیم (کنیم). همچنین یک سازنده تعریف می‌کنیم (خطوط ۲۳-۲۶) و با استفاده از کلمه کلیدی `super` مقادیر پارامترها را به سازنده پایه ارسال می‌کنیم. نمی‌توان از یک کلاس `abstract` نمونه ایجاد کرد ولی از کلاس‌هایی که از این نوع کلاس‌ها مشتق می‌شوند، می‌توان نمونه ایجاد کرد.

کلاس `final` و متدهای `final`

کلاس `final` (کلاس نهایی)، کلاسی است که دیگر کلاس‌ها نمی‌توانند از آن ارث بری کنند و چون قابلیت ارث بری ندارد نمی‌تواند مجرد هم باشد. مثال زیر یک کلاس `final` را نشان می‌دهد :

```
final class Base
{
    private int someField;

    public void SomeMethod()
    {
        //Do something here
    }

    //Constructor
    public Base()
    {
        //Do something here
    }
}

class Derived extends Base
{
    //This class cannot inherit the Base class
}
```

برای تعریف این کلاس‌ها از کلمه کلیدی `final` استفاده می‌شود. مشاهده می‌کنید که کلاس نهایی مانند کلاس‌های عادی، دارای فیلد، خواص، و متدهایی باشند. کلاس مشتق (`Derived`) در مثال بالا با خط قرمز نشان داده شده است چون نمی‌تواند از کلاس نهایی (`Base`) ارث بری کند. وقتی یک کلاس را نهایی می‌کنیم، تمام متدهای آن نیز نهایی می‌شوند. استفاده از این کلاس‌ها همانطور که ذکر شد زمانی مفید است که بخواهید کلاسی ایجاد کنید که دیگر کلاس‌ها نتوانند از آن ارث بری کنند.

متدهای Final

متدهای `final` به متدهای گفته می‌شود که هیچ زیر کلاسی نتواند آن را بازنویسی یا `Override` کند. به مثال زیر توجه کنید :

```
package myfirstprogram;

class Parent
{
    final void ShowMessage()
    {
        System.out.println("This is a final method!");
    }
}

class Child extends Parent
{
    @Override
    void ShowMessage()
    {
        System.out.println("This is a final method that Overridden!");
    }
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
    }
}
```

اگر به کدهای بالا توجه کنید و آن را در محیط NetBeans بنویسید مشاهده می‌کنید که در خط ۱۴ کد بالا خطای وجود می‌آید. چون کلاس `Child` از کلاس `Parent` ارث بری کرده است و زیر کلاس محسوب می‌شود و طبق تعریف هیچ زیر کلاسی نمی‌تواند متدهای `final` را بازنویسی کند.

چند ریختی (Polymorphism)

چند ریختی به کلاس‌هایی که در یک سلسله مراتب وراثتی مشابه هستند اجازه تغییر شکل و سازگاری مناسب می‌دهد و همچنین به برنامه نویس این امکان را می‌دهد که به جای ایجاد برنامه‌های خاص، برنامه‌های کلی و عمومی‌تری ایجاد کند. به عنوان مثال در دنیای واقعی همه حیوانات غذا می‌خورند، اما روش‌های غذا خوردن آنها متفاوت است. در یک برنامه برای مثال، یک کلاس به نام `Animal` ایجاد می‌کنید. بعد از ایجاد این کلاس می‌توانید آن را چند ریخت (تبديل) به کلاس `Bird` کنید و متدهای `Fly` را فراخوانی کنید. به مثالی درباره چند ریختی توجه کنید :

```

1 package myfirstprogram;
2
3 class Animal
4 {
5     public void Eat()
6     {
7         System.out.println("The animal ate!");
8     }
9 }
10
11 class Dog extends Animal
12 {
13     @Override
14     public void Eat()
15     {
16         System.out.println("The dog ate!");
17     }
18 }
19
20 class Bird extends Animal
21 {
22     @Override
23     public void Eat()
24     {
25         System.out.println("The bird ate!");
26     }
27 }
28
29 class Fish extends Animal
30 {
31     @Override
32     public void Eat()
33     {
34         System.out.println("The fish ate!");
35     }
36 }
37
38 public class MyFirstProgram
39 {
40     public static void main(String[] args)
41     {
42         Dog    myDog    = new Dog();
43         Bird   myBird   = new Bird();
44         Fish   myFish   = new Fish();
45         Animal myAnimal = new Animal();
46
47         myAnimal.Eat();
48
49         myAnimal = myDog;
50         myAnimal.Eat();
51         myAnimal = myBird;
52         myAnimal.Eat();
53         myAnimal = myFish;
54         myAnimal.Eat();
55     }
56 }
```

```
The animal ate!
The dog ate!
The bird ate!
The fish ate!
```

همانطور که مشاهده می‌کنید ۴ کلاس مختلف تعریف کرده‌ایم. Animal کلاس پایه است و سه کلاس دیگر از آن مشتق می‌شوند. هر کلاس متدهای Eat() مربوط به خود را دارد. نمونه‌ای از هر کلاس ایجاد کرده‌ایم (۴۲-۴۵). حال متدهای Eat() را به وسیله نمونه ایجاد شده از کلاس Animal به صورت زیر فراخوانی می‌کنیم :

```
Animal myAnimal = new Animal();
myAnimal.Eat();
```

در مرحله بعد چندريختی روی می‌دهد. همانطور که در مثال بالا مشاهده می‌کنید شیء Dog را برابر نمونه ایجاد شده از کلاس Animal قرار می‌دهیم (خط ۴۹) و متدهای Eat() را باز دیگر فراخوانی می‌کنیم (خط ۵۰). حال با وجود اینکه ما از نمونه کلاس Animal استفاده کرده‌ایم ولی متدهای Eat() کلاس Dog فراخوانی می‌شود. این به دلیل تأثیر چند ریختی است.

سپس دو شیء دیگر (Bird و Fish) را برابر نمونه ایجاد شده از کلاس Animal قرار می‌دهیم و متدهای Eat() مربوط به هر یک را فراخوانی می‌کنیم (خطوط ۵۱-۵۴). به این نکته توجه کنید که وقتی در مثال بالا اشیاء را برابر نمونه کلاس Animal قرار می‌دهیم از عمل Cast استفاده نکرده‌ایم چون این کار (cast) وقتی که بخواهیم یک شیء از کلاس مشتق (مثلاً Dog) را در شبی از کلاس پایه (Animal) با سازنده هر کلاس مشتق دیگر مقدار دهی اولیه کرد :

```
Animal myDog = new Dog();
Animal myBird = new Bird();
Animal myFish = new Fish();

myDog.Eat();
myBird.Eat();
myFish.Eat();
```

اجازه دهید که برنامه بالا را اصلاح کنیم تا مفهوم چند ریختی را بهتر متوجه شوید :

```
1 package myfirstprogram;
2
3 class Animal
4 {
5     public void Eat()
6     {
7         System.out.println("The animal ate!");
8     }
}
```

```
9 }
10
11 class Dog extends Animal
12 {
13     @Override
14     public void Eat()
15     {
16         System.out.println("The dog ate!");
17     }
18
19     public void Run()
20     {
21         System.out.println("The dog ran!");
22     }
23 }
24
25 class Bird extends Animal
26 {
27     @Override
28     public void Eat()
29     {
30         System.out.println("The bird ate!");
31     }
32
33     public void Fly()
34     {
35         System.out.println("The bird flew!");
36     }
37 }
38
39 class Fish extends Animal
40 {
41     @Override
42     public void Eat()
43     {
44         System.out.println("The fish ate!");
45     }
46
47     public void Swim()
48     {
49         System.out.println("The fish swam!");
50     }
51 }
52
53 public class MyFirstProgram
54 {
55     public static void main(String[] args)
56     {
57         Animal animal1 = new Dog();
58         Animal animal2 = new Bird();
59         Animal animal3 = new Fish();
60
61         Dog myDog = (Dog)animal1;
62         Bird myBird = (Bird)animal2;
63         Fish myFish = (Fish)animal3;
64
65         myDog.Run();
```

```

66     myBird.Fly();
67     myFish.Swim();
68 }
69 }
```

```

The dog ran!
The bird flew!
The fish swam!
```

در بالا سه شیء از کلاس Animal ایجادو آن‌ها را بوسیله سه سازنده از کلاس‌های مشتق مقدار دهی اولیه کردہ‌ایم (خطوط ۵۷-۵۹). سپس با استفاده از عمل cast اشیا ایجاد شده از کلاس Animal را در نمونه‌هایی از کلاس‌های مشتق ذخیره می‌کنیم (خطوط ۶۱-۶۳). وقتی این کار را انجام دادیم می‌توانیم متدهای مخصوص به هر یک از کلاس‌های مشتق را فراخوانی کنیم (خطوط ۶۵-۶۷). یک را می‌انباریم و سپس کد زیر مشخص شده است ولی در این روش شما نمی‌توانید اشیا ایجاد شده از کلاس Animal را در نمونه‌هایی از کلاس‌های مشتق ذخیره می‌کنید.

```
((Dog)animal1).Run();
((Bird)animal2).Fly();
((Fish)animal3).Swim();
```

از چند ریختی می‌توان در رابطه‌ها هم استفاده کرد. به کد زیر توجه کنید :

```

1 package myfirstprogram;
2
3 interface IAnimal
4 {
5     public void Eat();
6 }
7
8 class Dog implements IAnimal
9 {
10    @Override
11    public void Eat()
12    {
13        System.out.println("The dog ate!");
14    }
15 }
16
17 class Bird implements IAnimal
18 {
19    @Override
20    public void Eat()
21    {
22        System.out.println("The bird ate!");
23    }
24 }
25
26 class Fish implements IAnimal
```

```

28
29     @Override
30     public void Eat()
31     {
32         System.out.println("The fish ate!");
33     }
34 }
35
36 public class MyFirstProgram
37 {
38     public static void main(String[] args)
39     {
40         IAnimal myDog = new Dog();
41         IAnimal myBird = new Bird();
42         IAnimal myFish = new Fish();
43
44         myDog.Eat();
45         myBird.Eat();
46         myFish.Eat();
47     }
48 }
```

```

The dog ate!
The bird ate!
The fish ate!
```

تسلط کامل بر چند ریختی و وراثت برای درک بهتر شیء گرایی ضروری است.

کلاس‌های تو در تو (nested classes)

به کلاسی که در داخل کلاس دیگر تعریف شود کلاس تو در تو گفته می‌شود. از کلاس‌های تو در تو برای گروه بندی منطقی کلاس‌ها در یک مکان استفاده می‌شود، با این کار خوانایی کدها بیشتر و دستکاری آن‌ها راحت‌تر می‌شود. کلاس تو در تو عضوی از کلاسی است که در داخل آن قرار دارد، بنابراین می‌تواند به صورت `public`, `private` و `protect` تعریف شود. این کلاس‌ها می‌توانند توسط زیر کلاس (`subclass`) هم به ارت برده شوند. نحوه ایجاد یک کلاس تو در تو به صورت زیر می‌باشد.

```

class OuterClass
{
    class InnerClass
    {
    }
}
```

در جاوا چند نوع کلاس تو در تو وجود دارد که در زیر به آن‌ها اشاره شده است :

- کلاس‌های داخلی استاتیک

- کلاس‌های داخلی غیر استاتیک

- کلاس‌های محلی

- کلاس‌های بی نام

در درس‌های آینده در باره این کلاس‌ها توضیح می‌دهیم.

کلاس داخلی استاتیک و غیر استاتیک

همانطور که در درس قبل اشاره شده در جاوا ۴ نوع کلاس تو در تو وجود دارد که در این درس به دو نوع از آن‌ها می‌پردازیم.

کلاس‌های داخلی استاتیک

این نوع کلاس‌ها به صورت زیر تعریف می‌شوند :

```
public class Outer
{
    public static class Nested
    {
    }
}
```

برای ایجاد شیء از این نوع کلاس‌ها ابتدا باید نام کلاس بیرونی و سپس علامت نقطه و بعد نام کلاس داخلی استاتیک را بنویسید. به کد

زیر توجه کنید :

```
Outer.Nested instance = new Outer.Nested();
```

یک کلاس داخلی استاتیک یک کلاس عادی است که در داخل یک کلاس دیگر قرار دارد. کلاس‌های استاتیک داخلی فقط به اعضای

استاتیک کلاسی که در آن قرار دارند، دسترسی دارند. به مثال زیر توجه کنید :

```
package myfirstprogram;

class Outer
{
    static String message = "Hello World!";

    public static class Nested
    {
        void ShowMessage()
        {
            System.out.println(message);
        }
    }
}
```

```

}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Outer.Nested instance = new Outer.Nested();

        instance.ShowMessage();
    }
}

Hello World!

```

همانطور ک در مثال بالا متغیر `message` یک عضو استاتیک از کلاس `Outer` است و برای دسترسی به آن باید به صورتی که در خطوط مشاهده می‌کنید، عمل نمایید.

کلاس‌های داخلی غیر استاتیک یا Inner Classes

کلاس‌های داخلی غیر استاتیک در جاوا به نام `inner classes` معروف‌اند. برای ایجاد یک نمونه از این کلاس‌ها ابتدا باید یک نمونه از کلاس خارجی ایجاد کنید. نحوه تعریف این نوع کلاس‌ها به صورت زیر است :

```

public class Outer
{
    public class Inner
    {
    }
}

```

در زیر نحوه ایجاد نمونه از یک کلاس داخلی آمده است :

```

Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();

```

به نحوه قرار داده کلمه `new` بعد از شیء ایجاد شده از کلاس خارجی در کد بالا توجه کنید. کلاس داخلی غیر استاتیک به فیلدهای کلاس خارجی دسترسی دارد، حتی اگر این فیلدها به صورت `private` تعریف شده باشند. به کد زیر توجه کنید :

```

package myfirstprogram;

class Outer
{
    private String text = "I am private!";
}

```

```

public class Inner
{
    public void ShowMessage()
    {
        System.out.println(text);
    }
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Outer outerClass = new Outer();
        Outer.Inner innerClass = outerClass.new Inner();

        innerClass.ShowMessage();
    }
}

```

I am private!

کلاس‌های محلی (Local Classes)

کلاس‌های محلی در جاوا شبیه به کلاس‌های داخلی غیر استاتیک (inner class) هستند. این نوع کلاس‌ها در داخل یک متده است.

محدوده ({...}) در داخل متده تعریف می‌شوند. به مثال زیر توجه کنید :

```

class Outer
{
    public void printText()
    {
        class Local
        {

        }

        Local local = new Local();
    }
}

```

کلاس‌های محلی تنها در داخل متده و یا بلوکی که در آن تعریف شده‌اند قابل دسترسی هستند. این کلاس‌ها همانند کلاس‌های غیر استاتیک

می‌توانند به فیلدها و متدهای کلاسی که در داخل آن قرار دارند دسترسی داشته باشند. از نسخه ۸ جاوا این کلاس‌ها می‌توانند به

متغیرهای محلی و پارامترهای متده که در آن تعریف شده‌اند دسترسی داشته باشند. پارامترها باید به صورت final تعریف شده باشند.

کلاس‌های محلی می‌توانند در داخل متدهای استاتیک تعریف شوند که در این صورت فقط به قسمت‌های استاتیک کلاسی که در آن

تعریف شده‌اند دسترسی خواهند داشت. این کلاس‌ها نمی‌توانند دارای انواع استاتیک باشند (می‌توانند شامل ثابت‌ها باشند و متغیرهای

آن‌ها به صورت `final static` تعریف می‌شوند، چون کلاس‌های محلی در حالت عادی غیر استاتیک هستند حتی اگر در داخل متدهای استاتیک تعریف شوند.

کلاس داخلی بی‌نام (Anonymous Inner Class)

به کلاس داخلی که نام نداشته باشد، کلاس داخلی بی‌نام یا `Anonymous Inner Class` گفته می‌شود. از این کلاس‌ها برای `Override` استفاده می‌شود. نحوه تعریف کلاس‌های داخلی بی‌نام به صورت زیر است :

```
AnonymousInner ObjectName = new AnonymousInner()
{
    Modifier ReturnType MethodName()
    {
        //Some Code
    }
};
```

همانطور که احتمالا از کد بالا متوجه شده‌اید، تعریف و ایجاد شیء از این نوع کلاس‌ها به صورت همزمان می‌باشد. به این نکته هم توجه کنید که بعد از علامت آکولاد بسته از علامت سمیکالن (`;`) استفاده می‌شود. همانطور که در درس زیر ذکر شد، برای پیاده‌سازی یا `Override` یک متدهای از کلاس پایه توسط کلاس فرزند باید دو کلاس ایجاد شود که یکی از دیگری ارث بری کند :

<http://www.w3-farsi.com/?p=7974>

حال همین مثال اول لینک بالا را با استفاده از کلاس‌های بی‌نام پیاده‌سازی می‌کنیم :

```
1 package myfirstprogram;
2
3 class Parent
4 {
5     public void ShowMessage()
6     {
7         System.out.println("Message from Parent.");
8     }
9 }
10
11 public class MyFirstProgram
12 {
13     public static void main(String[] args)
14     {
15
16         Parent parent = new Parent()
17         {
18             public void ShowMessage()
19             {
20                 System.out.println("Message from Child.");
21             }
22         }
23     }
24 }
```

```

21         }
22     };
23
24     parent.ShowMessage();
25 }
26

```

در کد بالا، یک کلاس تعریف شده است (خطوط ۱-۷) که دارای یک متده میباشد (۳-۶). این متده را به وسیله کلاس بی نام (۱۴-۲۰) :

کردہایم (۱۶-۱۹). خطوط ۷-۱ کد بالا میتواند یک کلاس انتزاعی (abstract class) باشد :

```

abstract class Base
{
    abstract void ShowMessage();
}

```

و یا یک رابط : (interface)

```

interface IBase
{
    void ShowMessage();
}

```

میتوان کلاس بی نام را به عنوان آرگومان به یک متده هم ارسال کرد :

```

1 package myfirstprogram;
2
3 interface MyInterface
4 {
5     String Message();
6 }
7
8 class MyClass
9 {
10    public void ShowMessage(MyInterface m)
11    {
12        System.out.println(m.Message());
13    }
14 }
15
16 class MyFirstProgram
17 {
18     public static void main(String args[])
19     {
20         MyClass MC = new MyClass();
21
22         MC.ShowMessage(new MyInterface()
23         {
24             @Override
25             public String Message()
26             {
27                 return "Hello World!";
28             }
29         });
30     }
31 }
32

```

```

28         }
29     });
30 }
31 }
```

در خط ۳-۶ کد بالا یک رابط که دارای یک متده است را نوشته‌ایم. در خطوط ۸-۱۴ هم کلاسی داریم که دارای یک متده می‌باشد که یک آرگومان از نوع رابط MyInterface دریافت می‌کند. برای پیاده سازی یا Override متده مربوط به رابط همانطور که مشاهده می‌کنید در خط ۲۲ متده ShowMessage() مربوط به کلاس MyClass را فراخوانی کرده و یک کلاس بی نام به آن ارسال می‌کنیم. سپس در داخل همین کلاس بی نام متده Message() مربوط به رابط را Override می‌کنیم. ارسال کلاس بی نام به عنوان آرگومان همانطور که در آینده خواهید دید در مبحث Swing کاربرد دارد.

ایجاد آرایه‌ای از کلاسها

در این درس به شما نشان می‌دهیم که چگونه می‌توان آرایه‌ای از کلاس‌ها ایجاد کرد. ساخت آرایه‌ای از کلاس‌ها تقریباً شبیه به ایجاد آرایه‌ای از انواع داده‌ای مانند int است. به عنوان مثال می‌توان آرایه‌ای از کلاس Person ایجاد کرد:

```

1 package myfirstprogram;
2
3 class Person
4 {
5     public String Name;
6
7     public Person(String name)
8     {
9         this.Name = name;
10    }
11 }
12
13 public class MyFirstProgram
14 {
15     public static void main(String[] args)
16     {
17         Person[] people = new Person[3];
18
19         people[0] = new Person("Johnny");
20         people[1] = new Person("Mike");
21         people[2] = new Person("Sonny");
22
23         for (Person person : people)
24         {
25             System.out.println(person.Name);
26         }
27     }
28 }
```

ابتدا یک کلاس که دارای یک فیلد است تعریف می‌کنیم. سپس یک آرایه از کلاس ایجاد شده را تعریف و سپس عناصر آن را مانند بالا مقدار دهی می‌کنیم. سپس مقدار فیلد هر یک از نمونه‌ها را با استفاده از یک حلقه `foreach` نمایش می‌دهیم. می‌توان از تکنیک‌های دیگر که قبلاً در مورد ایجاد آرایه آموختید هم استفاده کنید. مثلاً خطوط ۲۱-۲۷ کد بالا را می‌توان به صورت زیر هم نوشت:

```
Person[] people = new Person[]
{
    new Person("Johnny"),
    new Person("Mike"),
    new Person("Sonny")
};
```

در اینجا، تعداد عناصر آرایه `people`، ۳ می‌باشد و کمپایلر هم با شمارش تعداد نمونه‌ها آن را تشخیص می‌دهد. از این تکنیک برای ساخت آرایه‌های چند بعدی و دندانه دار هم می‌توان استفاده کرد.

عبارات لامبدا

عبارة لامبدا (`Lambda expressions`) در اصل یک متند بی نام است. این متند ناشناس به تنهایی قابل اجرا نیست و برای پیاده‌سازی متندی که در یک رابط تابعی یا `functional interface` تعریف شده به کار می‌رود. رابط تابعی، اینترفیسی است که فقط و فقط یک متند در آن تعریف شده است. فرض کنید یک رابط تابعی به صورت زیر داریم:

```
interface MyMessage
{
    void ShowMessage(String message);
}
```

یک راه برای `Override` کردن متند این رابط استفاده از کلاس‌های داخلی بی نام به صورت زیر می‌باشد:

```
package myfirstprogram;

public class MyFirstProgram
{
    interface MyMessage
    {
        void ShowMessage(String message);
    }

    public static void main(String[] args)
    {
        MyMessage m = new MyMessage()
        {
            @Override
            public void ShowMessage(String message)
            {
                System.out.println(message);
            }
        };
    }
}
```

```

        }
    };

    m.ShowMessage("Hello World!");
}
}

Hello World!

```

ولی یک راه ساده‌تر هم وجود دارد و آن استفاده از عبارات لامبدا است. در کد زیر متده ShowMessage() با استفاده از عبارات لامبدا پیاده شده است:

```

package myfirstprogram;

public class MyFirstProgram
{
    interface MyMessage
    {
        void ShowMessage(String message);
    }

    public static void main(String[] args)
    {
        MyMessage m = (message) -> System.out.println(message);

        m.ShowMessage("Hello World!");
    }
}

Hello World

```

روش کلی استفاده از عبارات لامبدا به صورت زیر است :

parameter -> expression body

در عبارت لامبدا ابتدا پارامترها را بدون ذکر نوعشان می‌نویسیم و بعد از عملگر <- استفاده می‌کنیم. سپس دستوراتی را که قرار است اجرا شوند را می‌نویسیم. نوع پارامترها به صورت خودکار به وسیله کمپایلر تشخیص داده می‌شود. البته امضاء عبارات لامبدا باید شبیه به امضاء متده باشد. عبارات لامبدا دارای اشکال زیادی هستند. به عنوان مثال در مثال بالا از یک عبارت لامبدا یک استفاده کردۀایم که دارای یک دستور ساده اجرابی است. اگر متده شما برای عبارت لامبدا هیچ پارامتری نداشته باشد، همه کاری که لازم است نجام دهید این است که

هیچ پارامتری در داخل عبارت لامبدا قرار ندهیم :

```
package myfirstprogram;
```

```
public class MyFirstProgram
{
    interface MyMessage
    {
        void ShowMessage();
    }

    public static void main(String[] args)
    {
        MyMessage m = () -> System.out.println("Hello World!");

        m.ShowMessage();
    }
}
```

Hello World!

به این نکته توجه کنید که شما می‌توانید نوع پارامترهای عبارت لامبدا را نشان دهید.

```
package myfirstprogram;

public class MyFirstProgram
{
    interface MyMessage
    {
        void ShowMessage(String message);
    }

    public static void main(String[] args)
    {
        MyMessage m = (String message) -> System.out.println(message);

        m.ShowMessage("Hello World!");
    }
}
```

Hello World!

اگر عبارات لامبادای شما دارای چندین دستور اجرایی باشند می‌توانید آنها را داخل آکولاد قرار دهید. به عبارت لامبادایی که دارای آکولاد باشد بلک لامبدا می‌گویند.

```
package myfirstprogram;

public class MyFirstProgram
{
    interface MyMessage
    {
        void ShowMessage(String message);
    }

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

```

MyMessage m = (message) ->
{
    System.out.println(message);
    System.out.println("Some more message");
};

m.ShowMessage("Hello World!");
}
}

```

```
Hello World!
Some more message
```

در زیر مثالی از یک عبارت لامبدا که دارای مقدار برگشتی است نشان داده شده است :

```
SampleInterface GetSquare = (number) -> { return (number * number); };
```

به این نکته توجه کنید که هنگام استفاده از دستور return باید همیشه از دستورات لامبایی استفاده کنید که دارای آکولاد میباشد.

اگر یک عبارت لامبدا فقط دارای یک دستور return ساده باشد میتوانید به سادگی آن را به expression lambda تبدیل کنید.

```
SampleInterface GetSquare = (number) -> (number * number);
```

به این نکته توجه کنید که استفاده از پرانتز در کد بالا برای فهم بهتر آن است. اگر یک عبارت لامبدا دارای یک پارامتر ساده و یک دستور

return است، میتوانید برای سادگی بیشتر پرانتزها را حذف نمایید:

```
SampleInterface GetSquare = (number) -> number * number;
```

عبارت لامبایی بالا دارای دو پارامتر است و حاصل جمع آنها را بر میگرداند.

یک مثال از عبارت لامبدا

میتوان عبارات لامبدا را به عنوان آرگومان به دیگر متدها ارسال کرد. به مثال زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.util.Scanner;
4
5 public class MyFirstProgram
6 {
7     interface Arithmetic
8     {
9         int Operation(int x, int y);
10    }
11}
```

```

12 public static int GetResult(Arithmetic arithmetic, int num1, int num2)
13 {
14     return arithmetic.Operation(num1, num2);
15 }
16
17 public static void main(String[] args)
18 {
19     Scanner input = new Scanner(System.in);
20
21     System.out.print("Enter first number: ");
22     int num1 = input.nextInt();
23     System.out.print("Enter second number: ");
24     int num2 = input.nextInt();
25
26     System.out.println(); // Separator
27
28     System.out.println("Sum      = " + GetResult((x, y)->x + y, num1, num2));
29     System.out.println("Difference = " + GetResult((x, y)->x - y, num1, num2));
30     System.out.println("Product   = " + GetResult((x, y)->x * y, num1, num2));
31     System.out.println("Quotient  = " + GetResult((x, y)->x / y, num1, num2));
32 }
33 }
```

ابتدا در خطوط ۷-۱۰ یک رابط ایجاد می‌کنیم که دارای متده است که دو آرگومان از نوع اعداد صحیح را قبول کرده و یک مقدار صحیح را بر می‌گرداند. سپس از این رابط به عنوان یکی از پارامترهای متده `GetResult()` استفاده می‌کنیم (خطوط ۱۲-۱۵) دو پارامتر دیگر، دو عدد هستند که همانطور که خواهیم دید در عملیاتهای مختلف نقش دارند. از آنجاییکه اولین پارامتر یک رابط است، آرگومانی که به آن ارسال می‌شود باید پیاده سازی متده آن باشد. با استفاده از عبارت لامبда می‌توان کدهای خواناتری نوشت و همچنین کدنویسی را کاهش داد. در خطوط ۲۸-۳۱ متده `GetResult()` را فراخوانی می‌کنیم. توجه کنید که متده در هر فراخوانی یک عبارت لامبدا را به عنوان اولین آرگومان قبول می‌کند.

مدیریت استثناءها و خطایابی

بهترین برنامه نویسان در هنگام برنامه نویسی با خطاهای و باگ‌ها در برنامه‌شان مواجه می‌شوند. درصد زیادی از برنامه‌ها هنگام تست برنامه با خطای مواجه می‌شوند. بهتر است برای از بین بردن یا به حداقل رساندن این خطاهای، به کاربر در مورد دلایل به وجود آمدن آن‌ها اخطار داده شود. خوشبختانه جاوا برای این مشکل راه حلی ارائه داده است. جاوا دارای مجموعه بزرگی از کلاس‌هایی است که برای برطرف کردن خطاهای خاص از آن‌ها استفاده می‌کند. استثناءها در جاوا راهی برای نشان دادن دلیل وقوع خطای در هنگام اجرای برنامه است.

جاوا دارای مجموعه بزرگی از کلاس‌های استثناء است که شما می‌توانید با استفاده از آن‌ها خطایابی که در موقعیت‌های مختلف روی می‌دهند را برطرف کنید. حتی می‌توانید یک کلاس استثناء شخصی ایجاد کنید. استثناءها توسط برنامه به وجود می‌آیند و شما لازم است

که آن‌ها را اداره کنید. به عنوان مثال در دنیای کامپیوتر یک عدد صحیح هرگز نمی‌تواند بر صفر تقسیم شود. اگر بخواهید این کار را انجام دهید (یک عدد صحیح را بر صفر تقسیم کنید)، با خطأ مواجه می‌شوید. اگر یک برنامه در جاوا با چنین خطایی مواجه شود پیغام خطای "java.lang.ArithemticException: / by zero" نشان داده می‌شود که بدین معنا است که عدد را نمی‌توان بر صفر تقسیم کرد.

باگ (Bug) صطلaha خطا يا کدی است که رفتارهای ناخواسته‌ای در برنامه ایجاد می‌کند. خطایابی فرایند برطرف کردن باگ‌ها است، بدین معنی که خطاهرا را از برنامه پاک کنیم. Netbeans دارای ابزارهایی برای خطایابی هستند، که خطاهرا را یافته و به شما اجازه می‌دهند آن‌ها را برطرف کنید. در درس‌های آینده خواهید آموخت که چگونه از این ابزارهای کارآمد جهت برطرف کردن باگ‌ها استفاده کنید. قبل از اینکه برنامه را به پایان برسانید لازم است که برنامه‌تان را اشکال زدایی کنید.

استثناءهای اداره نشده

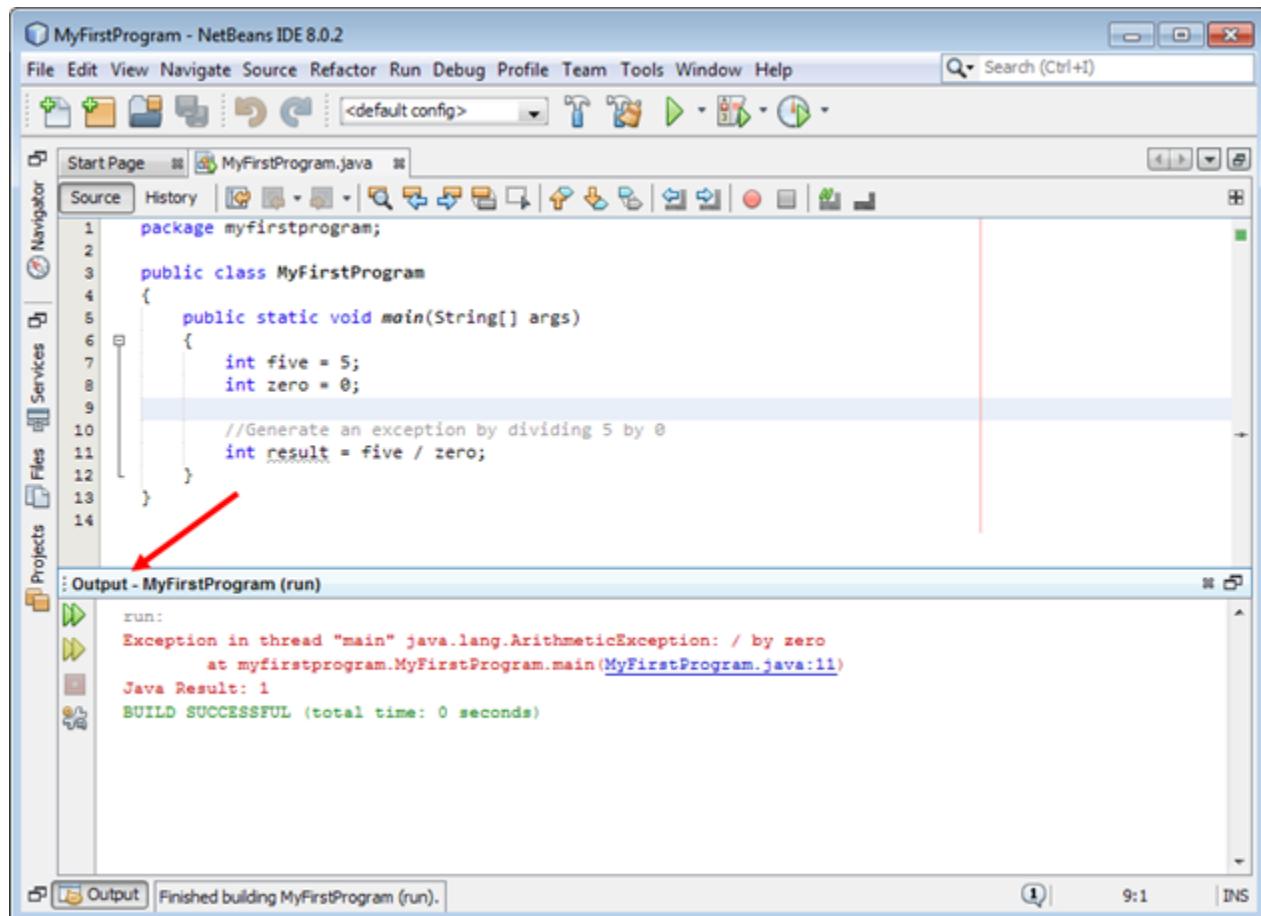
استثناءهای اداره نشده، استثناءهایی هستند که به درستی توسط برنامه اداره نشده‌اند و باعث می‌شوند که برنامه به پایان برسد. در اینجا می‌خواهیم به شما نشان دهیم که وقتی یک برنامه در زمان اجرا با یک استثناء مواجه می‌شود و آن را اداره نمی‌کند چه اتفاقی می‌افتد. در آینده خواهید دید که یک استثناء چگونه به صورت بالقوه باعث نابودی جریان و اجرای برنامه شما می‌شود. به برنامه زیر توجه کنید :

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int five = 5;
8         int zero = 0;
9
10        //Generate an exception by dividing 5 by 0
11        int result = five / zero;
12    }
13}
```

همانطور که در مثال بالا مشاهده می‌کنید تقسیم یک عدد صحیح بر صفر غیر مجاز است و باعث ایجاد خطای `java.lang.ArithmeticException: / by zero` می‌شود. برنامه را با زدن دکمه F6 اجرا می‌کنیم. برنامه با موفقیت اجرا شده ولی با بیگام خطای زیر مواجه می‌شوید:

```
Exception in thread "main" java.lang.ArithmetricException: / by zero  
        at myfirstprogram.MyFirstProgram.main(MyFirstProgram.java:11)
```

همانطور که مشاهده می‌کنید با اجرای برنامه اطلاعاتی درباره خروجی‌ها و یا خطاهای آن در پنجره Output نمایش داده می‌شود :



پنجره Output پنجره‌ای مفید است که در مورد استثناء اطلاعاتی در اختیار شما می‌گذارد و معمولاً محل دقیق خطا را به شما نشان می‌دهد که در شکل بالا خط ۱۱ (MyFirstProgram.java:11) نشان داده شده است.

دستور catch و try

می‌توان خطاهای را با استفاده از دستور try...catch اداره کرد. بدین صورت که کدی را که احتمال می‌دهید ایجاد خطا کند در داخل بلوک try قرار می‌دهید. بلوک catch هم شامل کدهایی است که وقتی اجرا می‌شوند که برنامه با خطا مواجه شود. تعریف ساده‌ی این دو بلوک به این صورت است که بلوک try سعی می‌کند که دستورات را اجرا کند و اگر در بین دستورات خطایی وجود داشته باشد برنامه دستورات مربوط به بخش catch را انجام می‌دهد. برنامه زیر نحوه استفاده از دستور try...catch را نمایش می‌دهد :

```

1 package myfirstprogram;
2

```

```

3  public class MyFirstProgram
4  {
5      public static void main(String[] args)
6      {
7          int result;
8          int x = 5;
9          int y = 0;
10
11         try
12         {
13             result = x / y; //ERROR
14         }
15         catch(ArithmeticException e)
16         {
17             System.out.println("An attempt to divide by 0 was detected.");
18         }
19     }
20 }
```

An attempt to divide by 0 was detected.

در داخل بلوک try، مقدار x را که ۵ است بر y که مقدار آن ۰ است تقسیم کرده‌ایم. نتیجه محاسبه به وجود آمدن خطای / by zero (عدد تقسیم بر صفر) است. از آنجاییکه در برنامه بالا خطایی به وجود آمده است کدهای داخل بلوک catch اجرا می‌شوند. بنابراین :

```

try
{
    result = x / y; //Error: Jump to catch block
    System.out.println("This line will not be executed.");
}
catch(ArithmeticException e)
{
    System.out.println("An attempt to divide by 0 was detected.");
}
```

همانطور که در مثال بالا مشاهده می‌کنید از یک نوع استثناء مخصوص به یک خطا در داخل بلوک catch که کلاس ArithmeticException است استفاده کرده‌ایم. همچنین می‌توانید مقدار استثنای در داخل یک متغیر قرار داده و سپس آن را نمایش دهید:

```

try
{
    result = x / y; //ERROR
}
catch(ArithmeticException error)
{
    System.out.println(error.getMessage());
}

/ by zero
```

متغیر دارای اطلاعات مفیدی در مورد استثناء به وجود آمده است. برای نمایش اطلاعاتی در مورد استثناء هم از متند `(getMessage())`

استفاده می‌کنیم. همه کلاسهای استثناء توضیحاتی در مورد خطاهای آینده در مورد خصوصیات استثناءها بیشتر توضیح می‌دهیم. اگر فکر می‌کنید که در بلوک `try` ممکن است با چندین خطای مواجه شوید می‌توانید از چندین بلوک `catch` استفاده نمایید ولی به یاد داشته باشید که برای هر کدام از آن خطاهای از کلاس استثناء مربوط به هر یک استفاده کنید:

```
int result;
int x = 5;
int y;

try
{
    y = input.nextInt();
    result = x / y; //ERROR
}
catch(ArithmetricException error)
{
    System.out.println(error.getMessage());
}
catch(InputMismatchException error)
{
    System.out.println(error.getMessage());
}
```

از انجاییکه مقدار `y` به وسیله ورودی که از کاربر گرفته می‌شود، تعیین می‌شود، مقدار آن باید با توجه به مثال بالا غیر صفر باشد (عدد تقسیم بر صفر تعریف نشده است). اما یک مشکل وجود دارد. چون ممکن است که کاربر یک مقدار غیر عددی وارد کند (مثلاً یک حرف) که در این صورت برنامه نمی‌تواند حرف را به عدد تبدیل کند و خطای نوع (`InputMismatchException`) اتفاق می‌افتد. وقتی استثناء اتفاق افتاد بلوک `catch` مربوط به این خطای اجرا می‌شود و محاسبه خارج قسمت تقسیم `x` بر `y` نادیده گرفته می‌شود. قسمت `catch` کد بالا را به صورت زیر هم می‌توان نوشت:

```
catch(ArithmetricException | InputMismatchException error)
{
    System.out.println(error.getMessage());
}
```

حال فرض کنید شما می‌خواهید تمام خطاهای احتمالی که ممکن است در داخل بلوک `try` اتفاق می‌افتد را فهمیده و اداره کنید این کار چگونه امکانپذیر است؟ به راحتی و با استفاده از کلاس عمومی `Exception` می‌توانید این کار را انجام داد. هر کلاس استثناء در جاوا از این کلاس ارث بری می‌کند بنابراین شما می‌توانید هر نوع استثنایی را در شیئی از کلاس `Exception` ذخیره نمایید.

```
try
{
```

```
//Put your codes to test here
}
catch (Exception error)
{
    System.out.println(error.getMessage());
}
```

با استفاده از این روش دیگر لازم نیست نگران اتفاق خطاهاي احتمالي باشيد چون بلوك catch برای هر گونه خطایی که در داخل بلوك try تشخیص داده شود پیغام مناسبی نشان می‌دهد. به این نکته توجه کنید که اگر بخواهید از کلاس پایه Exception همراه با سایر کلاس‌های استثناء دیگر که از آن مشتق می‌شوند در برنامه استفاده کنید باید کلاس پایه Exception در آخرین بلوك catch قرار گیرد.

```
try
{
    //Put your codes to test here
}
catch (ArithmaticException e)
{
    System.out.println("Division by zero is not allowed.");
}
catch (InputMismatchException e)
{
    System.out.println("Error on converting the data to proper type.");
}
catch (Exception e)
{
    System.out.println("An error occurred.");
}
```

اگر کلاس پایه Exception را در اولین بلوك catch قرار دهیم و خطایی در برنامه رخ دهد چون تمام کلاس‌های استثناء از این کلاس مشتق می‌شوند در نتیجه اولین بلوك catch اجرا شده و سایر بلوك‌ها حتی با وجود اینکه خطای مورد نظر به آن‌ها مربوط باشد اجرا نمی‌شوند.

بلوک finally

گاهی اوقات می‌خواهید برخی کدها همیشه اجرا شوند خواه استثنای رخ دهد، خواه رخ ندهد، در این صورت از بلوك finally استفاده می‌شود. قبل‌آیا در بلوك try یک استثنای رخ دهد همه کدهای موجود در این بلوك نادیده گرفته شده و برنامه به قسمت catch می‌رود. کدهای نادیده گرفته شده ممکن است در برنامه نقش حیاتی داشته باشند.

هدف بلوك finally هم حفظ نقش این کدها به صورت غیر مستقیم است. کدهایی را که فکر می‌کنید کدهای پایه‌ای هستند و برای اجرای برنامه لازم هستند را در داخل بلوك finally قرار دهید. برنامه زیر نحوه استفاده از این بلوك را نشان می‌دهد :

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int result;
8         int x = 5;
9         int y = 0;
10
11         try
12         {
13             result = x / y; //ERROR
14         }
15         catch(ArithmeticException error)
16         {
17             System.out.println(error.getMessage());
18         }
19         finally
20         {
21             System.out.println("finally blocked was reached.");
22         }
23     }
24 }
```

```
\ by zero.
finally blocked was reached.
```

بلوک finally بعد از بلوک catch نوشته می‌شود. اگر از چندین بلوک catch در برنامه استفاده می‌کنید بلوک finally باید بعد از همه آن‌ها قرار گیرد. می‌توان از بلوک try و finally در صورتی که بلوک catch نداشته باشیم به صورت زیر استفاده کرد.

```

try
{
    //some code
}
finally
{
    //some code
}
```

از این بلوک معمولاً برای بستن یک اتصال پایگاه داده یا بستن یک فایل استفاده می‌شود.

ایجاد استثناء

شما می‌توانید در هر جای برنامه یک خطای ساختگی ایجاد کنید. همچنین اگر پیغام پیشفرض استثناءها را دوست ندارید می‌توانید به دلخواه خودتان یک پیغام برای نمایش ایجاد کنید. به مثال زیر توجه کنید :

```

1 package myfirstprogram;
2
```

```

3 import java.util.Scanner;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {
9         Scanner input = new Scanner(System.in);
10
11         int firstNumber, secondNumber, result;
12
13         System.out.print("Enter the first number: ");
14         firstNumber = input.nextInt();
15
16         System.out.print("Enter the second number: ");
17         secondNumber = input.nextInt();
18
19         try
20         {
21             if (secondNumber == 0)
22             {
23                 throw new ArithmeticException();
24             }
25             else
26             {
27                 result = firstNumber / secondNumber;
28             }
29         }
30         catch (ArithmeticException error)
31         {
32             System.out.println(error.getMessage());
33         }
34     }
35 }
36 }
```

```

Enter the first number: 10
Enter the second number: 0
null
```

در خط ۲۳ و درست قبل از یک نمونه ایجاد شده از کلاس exception از کلمه کلیدی throw استفاده کردایم. می‌توان مستقیماً یک

نمونه از کلاس exception ایجاد و یک خطا را به دام انداخت. به مثال زیر توجه کنید :

```

ArithmeticException error = new ArithmeticException();

throw error;
```

همچنین می‌توان یک پیغام خطای سفارشی را به وسیله یکی دیگر از سربارگذاری‌های کلاس Exception که یک رشته را دریافت و آن را به عنوان پیغام خطای نمایش می‌دهد، نمایش داد.

```
throw new ArithmeticException("Cannot divide by zero!");
```

در این حالت پیغام خطای پیشفرض تغییر کرده و در متده استثناء بیشتر در موقعی به کار می‌رود getMessag ذخیره می‌شود.

که یک کد در حالت عادی خطأ ندارد ولی شما می‌خواهید در هر صورت به عنوان یک خطأ در نظر گرفته شود.

تعريف یک استثناء توسط کاربر

در جاوا می‌توان یک استثناء سفارشی ایجاد کرد. استثناء سفارشی، استثنایی است که توسط کاربر تعریف می‌شود و باید از کلاس پایه Exception ارث بری کند. برای این کار یک کلاس جداگانه که از کلاس پایه Exception می‌برد ایجاد می‌کنیم. به کد زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.util.Scanner;
4
5
6 class NegativeNumberException extends Exception
7 {
8     public NegativeNumberException()
9     {
10         super("The operation will result to a negative number.");
11     }
12
13     public NegativeNumberException(String message)
14     {
15         super(message);
16     }
17
18     public NegativeNumberException(String message, Exception inner)
19     {
20         super(message, inner);
21     }
22 }
23
24 public class MyFirstProgram
25 {
26
27     public static void main(String[] args)
28     {
29         Scanner input = new Scanner(System.in);
30
31         int firstNumber, secondNumber, difference;
32
33         System.out.print("Enter the first number: ");
34         firstNumber = input.nextInt();
35
36         System.out.print("Enter the second number: ");
37         secondNumber = input.nextInt();
38
39         difference = firstNumber - secondNumber;
40
41         try
42         {
43             if (difference < 0)

```

```

44         {
45             throw new NegativeNumberException();
46         }
47     }
48     catch (NegativeNumberException error)
49     {
50         System.out.println(error.getMessage());
51     }
52 }
53

```

```

Enter the first number: 10
Enter the second number: 11
The operation will result to a negative number.

```

در خط ۶ مشاهده می‌کنید کلاس ایجاد شده توسط ما از کلاس `Exception` ارث بری کرده است. به عنوان یک قرارداد باید به آخر نام کلاس‌های استثنایی که توسط کاربر تعریف می‌شوند کلمه `Exception` اضافه شده و ^۳ سازنده برای آن‌ها تعریف شود.

- اولین سازنده بدون پارامتر می‌باشد.
- دومین سازنده یک آرگومان از نوع رشته برای نمایش پیغام خطا قبول می‌کند.
- سومین سازنده که دو آرگومان قبول می‌کند، یکی پیغام خطا را نمایش داده و یکی بخش `inner` است آن برای نشان دادن علت وقوع استثناء می‌باشد.

حال می‌خواهیم یک کلاس استثناء خیلی سفارشی ایجاد کنیم. به خطوط ۴۹-۵۱ توجه کنید. چون که قرار است که از کاربر ورودی دریافت کنیم از کلاس `Scanner` در خط ۴۹ استفاده کردہ‌ایم. از آنجاییکه تولید یک عدد منفی در هیچ برنامه‌ای یک استثناء محسوب نمی‌شود، ما به صورت دستی و برای خودمان یک استثناء ایجاد کردہ‌ایم. ابتدا از کاربر می‌خواهیم که دو مقدار را وارد کند (خطوط ۴۳-۴۷). سپس تفاوت دو عدد را محاسبه می‌کنیم (خط ۴۹). در داخل بلوک `try` تست می‌کنیم که آیا حاصل تفریق دو عدد، یک عدد منفی است (خط ۴۰-۴۷). اگر یک عدد منفی بود سپس یک نمونه از کلاس `NegativeNumberException` ایجاد می‌کنیم (خط ۴۸). بعد از ایجاد نمونه به وسیله بلوک `catch` و برای نشان داده پیغام خطا آن را اداره می‌کنیم (خطوط ۴۸-۵۱).

مقایسه اشیاء با استفاده از رابطه‌های `Comparable` و `Comparator`

دو رابط مفید برای مقایسه اشیایی که توسط کاربر تعریف شده‌اند وجود دارد. این دو رابط `Comparable` و `Comparator` می‌باشند. رابط `Comparable` در یک کلاس پیاده سازی می‌شود و اجزه‌هی دهد کلاس یا شیء ایجاد شده از کلاس با اشیاء دیگر از همین کلاس مقایسه شوند. رابط `Comparator` در یک کلاس جداگانه پیاده سازی می‌شود. همانطور که از نام این رابط پیداست، پیاده سازی آن باعث ایجاد یک

کلاس مقایسه پذیر می‌شود. گاهی اوقات لازم است که بخواهید دو شیء را بر اساس یک فیلد یا خاصیت مقایسه کنید، این کار را می‌توانید

با استفاده از پیاده سازی رابط Comparable انجام دهید. حال به نحوه استفاده از رابط Comparable می‌پردازیم.

رابط Comparable

در مثال زیر یک کلاس نشان داده شده است که رابط Comparable را پیاده سازی می‌کند:

```

1 package myfirstprogram;
2
3 class Person implements Comparable<Person>
4 {
5     public String FirstName;
6     public String LastName;
7     public int Age;
8
9     public Person(String FirstName, String LastName, int Age)
10    {
11        this.FirstName = FirstName;
12        this.LastName = LastName;
13        this.Age = Age;
14    }
15
16    @Override
17    public int compareTo(Person other)
18    {
19        if (this.Age > other.Age)
20            return 1;
21        else if (this.Age < other.Age)
22            return -1;
23        else
24            return 0;
25    }
26
27
28 public class MyFirstProgram
29 {
30     public static void main(String[] args)
31     {
32         Person person1 = new Person("John", "Smith", 21);
33         Person person2 = new Person("Mark", "Logan", 19);
34         Person person3 = new Person("Luke", "Adams", 20);
35
36         Person youngest = GetYoungest(person1, person2, person3);
37         Person oldest = GetOldest(person1, person2, person3);
38
39         System.out.println(String.format("The youngest person is %s %s",
40             youngest.FirstName, youngest.LastName));
41         System.out.println(String.format("The oldest person is %s %s",
42             oldest.FirstName, oldest.LastName));
43     }
44
45     private static Person GetYoungest(Person person1, Person person2, Person person3)
46     {
47         Person youngest = person1;

```

```

48
49     if (person2.compareTo(youngest) == -1)
50         youngest = person2;
51
52     if (person3.compareTo(youngest) == -1)
53         youngest = person3;
54
55     return youngest;
56 }
57
58     private static Person GetOldest(Person person1, Person person2, Person person3)
59     {
60         Person oldest = person1;
61
62         if (person2.compareTo(oldest) == 1)
63             oldest = person2;
64
65         if (person3.compareTo(oldest) == 1)
66             oldest = person3;
67
68         return oldest;
69     }
70 }
```

The youngest person is Mark Logan
The oldest person is John Smith

وقتی که یک کلاس از رابط Comparable استفاده می‌کند (خط ۳)، لازم است که تنها متدهای آن یعنی متدهای compareTo() را نیز پیاده‌سازی کند (خطوط ۱۷-۲۵) متدهای compareTo() یک مقدار صحیح را بر می‌گرداند. این متدهای همچنین یک آرگومان قبول می‌کند که همان شیء است که قرار است با شیء جاری مقایسه شود. در داخل متدهای compareTo() در مثال بالا ما سن (age) شخص فعلی را با سن شخص دیگر مقایسه کرده‌ایم. طبق قرارداد اگر سن شخص مورد نظر ما از سن شخص دیگر بیشتر بود مقداری بزرگ‌تر از صفر، اگر کمتر بود مقداری کمتر از صفر و اگر مساوی بود مقدار صفر برگشت داده می‌شود. خطوط ۲۸-۶۹ پیاده‌سازی رابط Comparable توسط شیء Person را نشان می‌دهد. برنامه جوانترین و پیرترین شخص را تشخیص می‌دهد.

در خطوط ۳۲-۳۴ سه شیء Person با مقادیر کاملًا اختیاری ایجاد شده است. در خطوط ۳۶-۳۷ متغیرهایی برای نگهداری جوانترین و پیرترین شخص تعریف شده‌اند. در خط ۳۶ متدهای GetYoungest() را فراخوانی کرده‌ایم. این متدها در خطوط ۴۵-۵۶ تعریف شده است و سه شخص را که قرار است از لحاظ سنی با هم مقایسه شوند را قبول می‌کند.

در خط ۴۷ فرض را بر این گذاشته‌ایم که اولین شخص (person1) جوانترین شخص است. سپس با استفاده از پیاده‌سازی متدهای compareTo() تست می‌کنیم که آیا شخص دوم (person2) از شخص اول جوانتر است یا نه. در داخل متدهای مذکور سن شخص دوم و

اول را با هم مقایسه می‌کنیم. اگر سن شخص دوم کمتر بود، باید مقدار ۱- برگشت داده شده و در خط ۵۰ person2 به عنوان جوانترین شخص معرفی شود. در خطوط ۵۲-۵۳ از تکنیک مشابه برای شخص سوم استفاده کردہ‌ایم. بعد از مقایسه جوانترین شخص در خط ۵۵ به عنوان نتیجه برگشت داده می‌شود. در خط ۳۷ متدهای GetOldest() که در خطوط ۵۸-۶۹ تعریف شده است فراخوانی می‌شود. کدهای داخل این متدهای شبیه به متدهای GetYoungest() است با این تفاوت که تست می‌شود که آیا سن شخص دیگر بزرگتر از سن شخص مورد نظر ماست یا نه؟ بنابراین باید انتظار داشته باشیم که مقدار ۱ به جای ۱- توسط متدهای برگشت داده شود. در خطوط ۳۹ و ۴۱ نام جوانترین و پیرترین شخص چاپ می‌شود.

نکته: پیاده سازی رابط Comparable یک مشکل دارد و آن این است که اشیاء فقط به یک طریق مرتب می‌شوند مثلاً بر حسب نام. با این روش نمی‌توان یک بار بر حسب نام، یک بار دیگر بر حسب مثلاً فامیلی و ... اشیاء را مرتب کرد.

رابط Comparator

در یک کلاس جداگانه پیاده سازی می‌شود. همانطور که از نام این رابط پیداست، پیاده سازی آن باعث ایجاد یک کلاس مقایسه‌پذیر می‌شود. به وسیله این رابط می‌توان چندین مقایسه برای کلاس Person ایجاد کرد. یعنی می‌توان مقایسه‌های مختلفی بر روی اشیاء یک کلاس انجام داد. روش کار به صورت زیر است :

- ابتدا کلاس اصلی یعنی که کلاسی که می‌خواهیم مقایسه بر روی آن انجام شود، باید رابط Comparable را پیاده سازی کند.
- در چند کلاس جدا که رابط Comparator را پیاده سازی کده‌اند، و در داخل متدهای compare() این رابط، مقایسه‌های مختلف را انجام می‌دهیم.
- یک لیست می‌سازیم و اشیاء مختلف کلاس اصلی را در آن قرار می‌دهیم. با ایجاد لیست می‌توانیم از متدهای sort() کلاس Collection استفاده کنیم. چون این متدهای دو پارامتر می‌گیرد اولین پرامتر یک لیست است که همان اشیاء کلاس اصلی می‌باشند و دومین پارامتر هم یک شیء از کلاسی که رابط Comparator را پیاده سازی کرده است و کار مرتب سازی را انجام می‌دهد.

در مثال زیر، اشیاء ایجاد شده از کلاس Person بر اساس سن (age)، نام و یا نام خانوادگی مورد مقایسه قرار می‌گیرند.

```

1 package myfirstprogram;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.Comparator;
6 import java.util.List;
7 import java.util.Scanner;
8

```

```

9  class Person implements Comparable<Person>
10 {
11     public String FirstName;
12     public String LastName;
13     public int Age;
14
15     public Person(String FirstName, String LastName, int Age)
16     {
17         this.FirstName = FirstName;
18         this.LastName = LastName;
19         this.Age = Age;
20     }
21
22     @Override
23     public int compareTo(Person other)
24     {
25         if (this.Age > other.Age)
26             return 1;
27         else if (this.Age < other.Age)
28             return -1;
29         else
30             return 0;
31     }
32 }
33
34 class FirstNameComparer implements Comparator<Person>
35 {
36     @Override
37     public int compare(Person person1, Person person2)
38     {
39         return person1.FirstName.compareTo(person2.FirstName);
40     }
41 }
42
43 class LastNameComparer implements Comparator<Person>
44 {
45     @Override
46     public int compare(Person person1, Person person2)
47     {
48         return person1.LastName.compareTo(person2.LastName);
49     }
50 }
51
52 class AgeComparer implements Comparator<Person>
53 {
54     @Override
55     public int compare(Person person1, Person person2)
56     {
57         return person1.compareTo(person2);
58     }
59 }
60
61 public class MyFirstProgram
62 {
63     public static void main(String[] args)
64     {
65         ArrayList<Person> persons = new ArrayList<Person>();
66         persons.add(new Person("John", "Smith", 21));
67         persons.add(new Person("Mark", "Logan", 19));
68         persons.add(new Person("Luke", "Adams", 20));
69
70         System.out.println("Original Order");
71         for(Person p : persons)
72             System.out.println(String.format("%s %s, Age: %s", p.FirstName, p.LastName, p.Age));

```

```

73     System.out.println("\nSort persons based on their:");
74     System.out.println("[1] FirstName[2] LastName[3]Age");
75
76     System.out.print("Enter your choice: ");
77     Scanner input = new Scanner(System.in);
78     int choice = input.nextInt();
79
80     ReorderPersons(choice, persons);
81
82     System.out.println("\nNew Order");
83     for(Person p : persons)
84         System.out.println(String.format("%s %s, Age: %s", p.FirstName, p.LastName, p.Age));
85 }
86
87 private static void ReorderPersons(int choice, List persons)
88 {
89     Comparator comparer;
90
91     if (choice == 1)
92         comparer = new FirstNameComparer();
93     else if (choice == 2)
94         comparer = new LastNameComparer();
95     else
96         comparer = new AgeComparer();
97
98     Collections.sort(persons, comparer);
99 }
100 }
```

Original Order

John Smith, Age: 21
 Mark Logan, Age: 19
 Luke Adams, Age: 20

Sort persons based on their:

[1]
 FirstName
 [2] LastName
 [3]Age
 Enter your choice: 1

New Order

John Smith, Age: 21
 Luke Adams, Age: 20
 Mark Logan, Age: 19

Original Order

John Smith, Age: 21
 Mark Logan, Age: 19
 Luke Adams, Age: 20

Sort persons based on their:

[1]
 FirstName
 [2] LastName
 [3]Age
 Enter your choice: 2

New Order

Luke Adams, Age: 20
 Mark Logan, Age: 19

```

John Smith, Age: 21
Original Order
John Smith, Age: 21
Mark Logan, Age: 19
Luke Adams, Age: 20

Sort persons based on their:
[1]
FirstName
[2] LastName
[3]Age
Enter your choice: 3

New Order
Mark Logan, Age: 19
Luke Adams, Age: 20
John Smith, Age: 21

```

هنگام استفاده از این رابط لازم است یک متده است که دو شیء قبول می‌کند و یک عدد صحیح را به عنوان نتیجه بر می‌گرداند را پیاده سازی کند. از آنجاییکه از رابط Comparator استفاده کرده‌ایم متده compare() به طور خودکار دو شیء Person قبول می‌کند. همانطور که اشاره شد، این رابط به وسیله کلاس جداگانه پیاده سازی می‌شود و ما این کار روا در خطوط ۳۴-۵۹ انجام داده‌ایم. در این خطوط سه کلاس تعریف کرده‌ایم که این رابط را پیاده سازی کرده (خطوط ۴۳، ۵۴ و ۵۶) و در نتیجه وظیفه دارند که متده compare() این رابط را هم Override کنند. در داخل بدنه این متده هم مقایسه‌های مختلف را می‌نویسیم.

در مثال بالا با استفاده از کلاس FirstNameComparer که رابط Comparator را پیاده سازی می‌کند، دو شیء Person بر اساس نام مقایسه می‌شوند. در متده Compare() به سادگی و با استفاده از متده Comparable compareTo() از رابط استفاده کرده‌ایم (چون خاصیت FirstName یک رشته است) و یک مقدار را به عنوان نتیجه بر می‌گردانیم.

به روشه مشابه از کلاس‌های LastNameComparer و AgeComparer برای مقایسه اشیاء بر اساس نام خانوادگی و سن استفاده می‌کنیم. متده compare() در صورتی که دو پارامتر با هم برابر باشند، مقدار ۰، اگر پارامتر اول از پارامتر دوم بزرگتر باشد، مقداری بزرگتر از ۰ و اگر پارامتر اول از پارامتر دوم کوچکتر باشد مقداری کوچکتر از ۰ را بر می‌گرداند. در مثال بالا از کاربر سؤال می‌شود که لیستی از اشیاء را قرار است بر اساس کدام خاصیت مرتب کند. در خطوط ۶۴-۶۶ اشیا با مقادیر از پیش تعریف شدهای برای هر یک از خاصیت‌های ایشان ایجاد شده است. در خطوط ۶۸-۷۰ ترتیب عادی و اصلی این اشیاء نمایش داده شده است. در خطوط ۷۴-۷۵ لیستی از انتخاب‌هایی که کاربر بر اساس آنها می‌تواند عملیات مرتب سازی را انجام دهد آورده شده است. در خطوط ۷۷-۷۹ از کاربر در مورد انتخاب‌ش سؤال می‌شود. در خط ۸۱ متده ReorderPersons از پیش تعریف شده خطوط ۸۸-۱۰۰ را فراخوانی می‌کنیم. این متده انتخاب کاربر و لیستی از اشیاء

که قرار است بر اساس خاصیتی که کاربر انتخاب کرده است مرتب شوند را قبول می‌کند. در داخل متدهای معرفی کردہ‌ایم که از نوع Comparator است و در نتیجه می‌تواند هر نوع کلاسی که رابط مذکور را پیاده سازی می‌کند را شامل شود. در خطوط ۹۷-۹۲ چک می‌کنیم که اگر کاربر یک مقدار عددی خاص را انتخاب کرد، چه کارهایی انجام شود. در خط ۴۰ از متدهای sort() کلاس Collections استفاده کرده‌ایم. این متدهای دارای یک نسخه سربارگذاری شده است که یک شیء Comparator را قبول می‌کند. ما در خط ۹۹ کلاس مقایسه کننده بر اساس نوع انتخاب کاربر را به این متدهای دهیم و سپس متدهای sort() شیء Person را بر اساس این کلاس مرتب می‌کنند.

به طور خلاصه کلاسی که رابط Comparable را پیاده سازی کند، به کلاسی مقایسه پذیر تبدیل می‌شود و کلاسی که رابط Comparator را پیاده سازی کند، می‌تواند به وسیله متدهای Arrays.sort() یا Collections.sort() ارتقای مرتب شود، کاری که ما در خط ۹۹ کد بالا انجام داده‌ایم.

کلکسیون‌ها (Collections)

قبل‌آید گرفتیم که آرایه‌ها به ما اجازه ذخیره چندین مقدار از یک نوع را می‌دهند. آرایه‌ها از کلاس Java.util.Arrays ارث بری می‌کنند که این کلاس دارای خواص و متدهایی برای کار با داده‌های ساده‌ای مانند طول آرایه می‌باشد. آرایه‌های ساده در جاوا دارای طول ثابتی هستند که یک بار تعریف و مقدار دهی می‌شوند و شما نمی‌توانید طول یک آرایه خاص را افزایش یا کاهش دهید. جاوا گزینه بهتری برای جایگزین کردن با آرایه‌ها پیشنهاد می‌دهد و بیشتر آن‌ها کلاس‌ها و رابطه‌ایی هستند که در پکیج و کلاس java.util.ArrayList قرار دارند. به عنوان مثال کلاس ArrayList رفتاری شبیه به یک آرایه معمولی دارد با این تفاوت که به شما اجازه می‌دهد که طول آن را به صورت پویا تغییر داده یا یک عنصر را در طول اجرای برنامه به آن اضافه کرده و یا از آن حذف نمایید. در درس بعد پی می‌برید که چگونه یک کلاس که شامل مجموعه‌ای از اشیاء است را به وسیله اجرا کردن و یا ارث بری از رابطه‌ها و متدها ایجاد کنیم.

کلاس ArrayList

کلاس ArrayList به شما اجازه ذخیره مقادیر انواع داده‌ای مختلف، و توانایی حذف و اضافه عناصر آرایه در هر لحظه را می‌دهد. در مثال زیر به سادگی کاربرد کلاس ArrayList آمده است.

```
package myfirstprogram;

import java.util.ArrayList;

public class MyFirstProgram
{
    public static void main(String[] args)
```

```
{
    ArrayList myArray = new ArrayList();

    myArray.add("John");
    myArray.add(5);
    myArray.add(true);
    myArray.add(3.65);
    myArray.add('R');

    for (Object element : myArray)
    {
        System.out.println(element);
    }
}
```

```
John
5
true
3.65
R
```

برای استفاده از این کلاس ابتدا باید در قسمت فضاهای نامی، فضای نام `java.util.ArrayList` را وارد کنیم (خط ۳). همانطور که در مثال مشاهده می‌کنید یک نمونه از کلاس `ArrayList` ایجاد می‌کنیم. برای اضافه کردن یک عنصر به آرایه باید از متد `(add())` استفاده کنیم. از آنجاییکه شیء ایجاد شده از کلاس `ArrayList` آرگومانی از نوع `Object` قبول می‌کند بنابراین می‌توان مقادیری از هر نوع داده‌ای به آن ارسال کرد چون هر چیز در جاوا از `Object` ارث بری می‌کند.

حال برای نمایش توانایی این کلاس در نگهداری انواع داده‌ای مختلف پنج مقدار از پنج نوع مختلف داده را به آن اضافه می‌کنیم. سپس همه مقادیر را با استفاده از دستور `foreach` می‌خوانیم. چون کلاس `ArrayList` دارای انواع داده‌ای مختلفی است نمی‌توانیم از یک نوع داده‌ای خاص برای خواندن مقادیر استفاده کنیم. لذا برای این کار باید از نوع `Object` که می‌تواند هر نوع داده‌ای در خود ذخیره کند استفاده نمود. به این نکته توجه کنید که برای دسترسی به هر عنصر می‌توانید از طریق اندیس آن اقدام نمایید. کد زیر نحوه استفاده از حلقه `for` برای دسترسی به هر یک از اعضاء را نشان می‌دهد.

```
for (int i = 0; i < myArray.size(); i++)
{
    System.out.println(myArray.get(i));
}
```

به متد `(size())` در کد بالا توجه کنید. این متد درست شبیه به خاصیت `length` آرایه معمولی است و کار آن شمارش تعداد عناصر شیء `ArrayList` می‌باشد. در کد بالا همانطور که نشان داده شده است می‌توان به هر یک از عناصر با استفاده از اندیسشان دست یافت

(get(i)). نکته دیگر این است که شما می‌توانید به کلاس `ArrayList` یک ظرفیت ابتدایی بدهید. به عنوان مثال شما می‌توانید با

استفاده از یک سازنده سربارگذاری شده نشان دهید که یک شیء `ArrayList` می‌تواند دارای ۵ عنصر باشد.

```
ArrayList myArray = new ArrayList(5);
```

کد بالا ۵ مکان خالی به وجود می‌آورد و شما می‌توانید با استفاده از متده `add()` یکی دیگر به آن‌ها اضافه کنید. می‌توان با استفاده از متده `remove()` کلاس `ArrayList` عناصر را پاک کرد. متده `remove()` یک شیء که مطابق مقدار یک عنصر در آرایه است را قبول می‌کند. این متده به محض رسیدن به مقدار مورد نظر آن را حذف می‌کند. اگر عنصری را که مکانی غیر از مکان آخر آرایه باشد حذف کنید بقیه عناصر بعد از آن عنصر مکان خود را تنظیم می‌کنند به این معنی که فرض کنید آرایه‌ای دارای ۵ عنصر است و شما عنصر ۳ را حذف می‌کنید، در این صورت جای خالی این عنصر توسط عنصر ۴ و جای عنصر ۴ توسط عنصر ۵ پر می‌شود. به تکه کد زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.util.ArrayList;
4 import java.text.MessageFormat;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         ArrayList myArray = new ArrayList();
11
12         myArray.add("John");
13         myArray.add(5);
14         myArray.add(true);
15         myArray.add(3.65);
16         myArray.add('R');
17
18         for (int i = 0; i < myArray.size(); i++)
19         {
20             System.out.println(MessageFormat.format("myArray[{0}] = {1}", i, myArray.get(i)));
21         }
22
23         myArray.remove(2);
24
25         System.out.println("\nAfter removing myArray[1] (The value true)...\\n");
26
27         for (int i = 0; i < myArray.size(); i++)
28         {
29             System.out.println(MessageFormat.format("myArray[{0}] = {1}", i, myArray.get(i)));
30         }
31     }
32 }
```

myArray[0] = John
myArray[1] = 5
myArray[2] = true
myArray[3] = 3.65
myArray[4] = R

After removing myArray[2] (The value 5)...

```
myArray[0] = John
myArray[1] = 5
myArray[2] = 3.65
myArray[3] = R
```

از آنجاییکه در مثال بالا مقدار عنصر `myArray[2]` را حذف کردهایم (خط ۲۳) همه عناصر متوالی در آرایه بالا مکان خود را تغییر می‌دهند.

بنابراین عنصر `myArray[3]` جای `myArray[2]`، عنصر `myArray[4]` جای `myArray[3]` و ... را می‌گیرد.

جستجو، جایگزینی و به دست آوردن اندیس مقادیر

با استفاده از متدهای `contains()` می‌توان چک کرد که آیا یک مقدار خاص در داخل آرایه وجود دارد یا خیر. این متدهای آرگومان از نوع

شیء را قبول کرده و اگر یک مقدار را در داخل لیست عناصر پیدا کند `true` را بر می‌گرداند، از متدهای `lastIndexOf()` و `indexOf()` برای تشخیص اندیس یک مقدار خاص استفاده می‌شود.

- متدهای `indexOf()` و `lastIndexOf()` اندیس اولین محل وقوع یک مقدار خاص را بر می‌گردانند.

- متدهای `indexOf()` و `lastIndexOf()` اندیس آخرین محل وقوع یک مقدار خاص را بر می‌گردانند.

- هر دو متدهای `indexOf()` و `lastIndexOf()` اندیس اولین محل وقوع یک مقدار خاص را بر می‌گردانند.

برای جایگزین کردن یک مقدار با یک مقدار موجود در `ArrayList` از متدهای `set()` استفاده می‌شود. فرض کنید که می‌خواهید عدد ۱۵ را

جایگزین عدد ۵ در مثال بالا کنید برای اینکار باید به صورت زیر عمل کنید :

```
myArray.set(2,15);
```

عدد ۲ در مثال بالا نشان دهنده اندیس مقداری از `ArrayList` است که ما می‌خواهیم مقداری دیگر را جایگزین آن کنیم. در این مثال

اندیس ۲ نشان دهنده مقدار ۵ است. برای به دست آوردن تعداد عناصر در یک `ArrayList` از متدهای `size()` استفاده می‌شود.

```
myArray.size();
```

و برای پاک کردن همه عناصر از متدهای `clear()` استفاده می‌شود :

```
myArray.clear();
```

مرتب سازی مقادیر ArrayList

مرتب سازی در ArrayList زمانی معنا دارد که همه آیتم‌ها از یک نوع مثلاً عدد صحیح باشند. برای ایجاد یک کلاس از

یک نوع خاص به صورت زیر عمل می‌شود :

```
ArrayList <type> CollectionName = new ArrayList();
```

که در آن type نوع داده‌ای مجموعه و CollectionName نامی است که برای مجموعه انتخاب کردہ‌ایم. فرض کنید که می‌خواهیم یک مجموعه از نوع اعداد صحیح ایجاد و مرتب کنیم. به مثال زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.text.MessageFormat;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        ArrayList <Integer> myArray = new ArrayList();
12
13        myArray.add(1);
14        myArray.add(5);
15        myArray.add(3);
16        myArray.add(2);
17
18        Collections.sort(myArray);
19
20        for (int i = 0; i < myArray.size(); i++)
21        {
22            System.out.println(MessageFormat.format("myArray[{0}] = {1}", i, myArray.get(i)));
23        }
24    }
25 }
```

```
myArray[0] = 1
myArray[1] = 2
myArray[2] = 3
myArray[3] = 5
```

همانطور که در مثال بال مشاهده می‌کنید در خط ۱۶-۱۹ پکیج Collections را وارد برنامه کردہ‌ایم. وجود این پکیج برای مرتب سازی الزامی است. در خط ۱۱ یک مجموعه از نوع اعداد صحیح ایجاد و در خطوط ۱۶-۱۹ به صورت نامرتب چند آیتم به آن اضافه کردہ‌ایم. سپس در خط ۱۸ با استفاده از متد sort() کلاس Collections مجموعه را مرتب نموده‌ایم.

با استفاده از متد sort() می‌توان مقادیر یک آرایه را مرتب نمود. اعداد از بزرگ به کوچک و رشته بر اساس حروف الفبا مرتب می‌شوند. اگر از این متد استفاده کنید همه اجزا با هم مقایسه می‌شوند. به عنوان مثال نمی‌توان یک رشته و یک عدد از نوع int را در داخل

قرار داد و آن‌ها را با متدهای Sort() مرتب نمود. در درس آینده یاد خواهید گرفت که چگونه از یک مقایسه گر سفارشی برای مرتب کردن عناصر استفاده نمود.

ListIterator و Iterator

برای دسترسی، ویرایش و حذف هر یک از عناصر یک کلکسیون ابتدا باید عنصر مورد نظر را پیدا کنیم. برای این کار لازم است که در میان عناصر بگردیم. سه راه برای گردش در میان عناصر یک کلکسیون یا مجموعه وجود دارد:

۱. با استفاده از رابط **Iterator**

۲. با استفاده از رابط **ListIterator**

۳. با استفاده از حلقه **foreach**

دسترسی به عناصر مجموعه با استفاده از رابط Iterator

رابط **Iterator** یک مجموعه یا کلکسیون را رو به جلو پیمایش کرده و شما را قادر می‌سازد که عناصر را حذف و ویرایش کنید. هر کلکسیون برای ایجاد یک پیمایشگر دارای متدهای `iterator()` و `next()` می‌باشد. وقتی یک شیء از **Iterator** می‌سازیم می‌توانیم به متدهای آن که در جدول زیر آمده‌اند نیز دست یابیم:

متدها	توضیح
<code>hasNext()</code>	چک می‌کند که آیا عنصری بعد از عنصر فعلی وجود دارد یا نه؟
<code>next()</code>	عنصر بعدی را بر می‌گرداند.

به مثال زیر توجه کنید:

```
package myfirstprogram;

import java.util.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ArrayList<String> arraylist = new ArrayList<>();

        arraylist.add("A");
        arraylist.add("B");
        arraylist.add("C");
    }
}
```

```

arraylist.add("D");

Iterator iterator = arraylist.iterator();      //Declaring Iterator

while (iterator.hasNext())
{
    System.out.println(iterator.next());
}
}

```

A
B
C
D

مهمترین خط کد بالا خط زیر است :

```
Iterator iterator = arraylist.iterator();
```

در کد بالا با فراخوانی متد () از کلکسیونمان که در اینجا `arraylist` است آن را به مجموعه‌ای قبل پیمایش توسط `Iterator` می‌کنیم و سپس با استفاده از متدهای این رابط آن را پیمایش می‌کنیم.

دسترسی به عناصر مجموعه با استفاده از رابط `ListIterator`

رابط `ListIterator` یک مجموعه را با استفاده از متدهایی که در جدول زیر آمده‌اند، هم رو به جلو و هم رو به عقب پیمایش می‌کند :

متدها	توضیح
<code>hasNext</code>	چک می‌کند که آیا عنصری بعد از عنصر فعلی وجود دارد یا نه؟
<code>next</code>	عنصر بعد از عنصر فعلی را بر می‌گرداند.
<code>hasPrevious</code>	چک می‌کند که آیا عنصری قبل از عنصر فعلی وجود دارد یا نه؟
<code>previous</code>	عنصر قبل از عنصر فعلی را بر می‌گرداند.

این رابط فقط در مجموعه‌های قابل دسترسی است که رابط `List` را پیاده سازی کرده باشند :

```
package myfirstprogram;
```

```

import java.util.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ArrayList<String> arraylist = new ArrayList<>();

        arraylist.add("A");
        arraylist.add("B");
        arraylist.add("C");
        arraylist.add("D");

        ListIterator listiterator = arraylist.listIterator();

        while (listiterator.hasNext()) //In forward direction
        {
            System.out.println(listiterator.next());
        }

        System.out.println("\n\n");

        while (listiterator.hasPrevious()) //In backward direction
        {
            System.out.println(listiterator.previous());
        }
    }
}

```

A
B
C
D

D
C
B
A

دسترسی به عناصر مجموعه با استفاده از حلقه foreach

Foreach نسخه‌ای از حلقه for است که می‌تواند در میان عناصر یک مجموعه گردش کند. از این حلقه نمی‌توان برای ویرایش عناصر یک مجموعه استفاده کرد. حلقه foreach می‌تواند در میان هر مجموعه‌ای از اشیاء که رابط Iterable را پیاده سازی کرده باشد گردش کند

:

```

package myfirstprogram;

import java.util.*;

```

```

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ArrayList<String> arraylist = new ArrayList<>();

        arraylist.add("A");
        arraylist.add("B");
        arraylist.add("C");
        arraylist.add("D");

        for (String str : arraylist)
        {
            System.out.println(str);
        }
    }
}

```

A
B
C
D

Vector

کلاس Vector هم، رابط List را پیاده سازی می کند. همانند ArrayList می تواند تعداد نامشخصی عنصر داشته باشد ولی سرعت آن در جستجو، اضافه و حذف عناصر نسبت به ArrayList کمتر است. سه روش برای ایجاد یک شیء از کلاس Vector وجود دارد. در روش اول تعداد عناصر را مشخص نمی کنیم. در این حالت ظرفیت پیشفرض vector ده عنصر می باشد:

```
Vector myVector = new Vector();
```

نکته ای که در این روش وجود دارد این است که وقتی شما بخواهید یا زده مین عنصر را وارد کنید، ظرفیت Vector دو برابر می شود. در روش دوم یک ظرفیت برای Vector تعریف می کنیم :

```
Vector myVector = new Vector(int initialCapacity)
```

در مثال بالا Vector سه عنصر را می تواند در خود جای دهد. و روش سوم هم دو آرگومان برای Vector در نظر می گیریم. آرگومان اول به معنای ظرفیت پیشفرض و آرگومان دوم هم به معنای ظرفیت نهایی می باشد :

```
Vector myVector = new vector(int initialcapacity, capacityIncrement)
```

یعنی در مثال فوق بعد از وارد کردن پنجمین عنصر سایز Vector به ۱۰ و بعد از وارد کردن یازدهمین عنصر ظرفیت Vector به ۱۶ افزایش می یابد. متدهای مهم کلاس Vector در زیر آمده است :

متدها	توضیح
addElement	یک عنصر را به آخر Vector اضافه می‌کند.
capacity	ظرفیت فعلی Vector را برمی‌گرداند.
size	اندازه فعلی (تعداد عناصر) Vector را برمی‌گرداند.
contains	چک می‌کند که آیا یک عنصر خاص در Vector وجود دارد یا نه؟
containsAll	چک می‌کند که آیا همه عناصر یک مجموعه در داخل Vector هستند یا خیر؟
elementAt	چک می‌کند که آیا یک عنصر خاص در موقعیتی خاص از Vector وجود دارد یا نه؟
firstElement	اولین عنصر Vector را برمی‌گرداند.
lastElement	آخرین عنصر Vector را برمی‌گرداند.
get	یک عنصر که در موقعیت یا اندیسی خاص از Vector است، را برمی‌گرداند.
isEmpty	اگر Vector دارای هیچ عنصری نباشد مقدار true را برمی‌گرداند.
removeElement	یک عنصر خاص از Vector را حذف می‌کند.
removeAll	همه عناصر Vector را که در یک مجموعه هستند حذف می‌کند.
setElementAt	یک عنصر را جایگزین یک عنصر با اندیس خاص می‌کند.

برای نحوه کار با کلاس Vector به مثال زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.util.*;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {

```

```

9      /* Vector of initial capacity(size) of 2 */
10     Vector<String> myVector = new Vector<String>(2);
11
12     /* Adding elements to a vector*/
13     myVector.addElement("Apple");
14     myVector.addElement("Orange");
15     myVector.addElement("Mango");
16     myVector.addElement("Fig");
17
18     /* check size */
19     System.out.println("Size is: " + myVector.size());
20     System.out.println("Default capacity increment is: " + myVector.capacity());
21
22     /* capacityIncrement */
23     myVector.addElement("fruit1");
24     myVector.addElement("fruit2");
25     myVector.addElement("fruit3");
26
27     /*size and capacityIncrement after two insertions*/
28     System.out.println("Size after addition: " + myVector.size());
29     System.out.println("Capacity after increment is: " + myVector.capacity());
30
31     /*Display Vector elements*/
32     System.out.println("\nElement are: ");
33     for(String vector : myVector)
34     {
35         System.out.println(vector);
36     }
37 }
38 }
```

```

Size is: 4
Default capacity increment is: 4
Size after addition: 7
Capacity after increment is: 8

Element are:
Apple
Orange
Mango
Fig
fruit1
fruit2
fruit3
```

همانطور که در کد بالا مشاهده می‌کنید، ظرفیت Vector برابر عدد ۲ است (خط ۱۰) ولی چون ما در خطوط ۱۳-۱۶ چهار عنصر به آن اضافه کردہ‌ایم، ظرفیت آن به ۴ تغییر می‌کند. به این نکته توجه کنید که متدهای size() و capacity() تعداد عناصر و حجم را نشان می‌دهد. به ادامه کد توجه کنید. در خطوط ۲۳-۲۵ سه عنصر دیگر به Vector اضافه کردہ‌ایم. در نتیجه خروجی تعداد عناصر را ۷ و حجم را ۸ نشان می‌دهد. در آخر هم با استفاده از یک حلقة foreach تمام عناصر Vector را چپ کردہ‌ایم.

List

یکی از interface های اصلی Collection Framework جاواست که برای نگهداری یک لیست از آن استفاده می‌شود، زیر کلاس‌های مختلفی همچون Vector، ArrayList، LinkedList در جاوا وجود دارند که هر کدام با استفاده از ساختمان داده‌های متفاوت به نحوی مفهوم لیست را پیاده سازی کرده‌اند. یک لیست باید حداقل شرایط زیر را برآورده کند :

- بتوان به آن عناصری را اضافه یا حذف کرد.
- بتواند اعضای تکراری را نگه داری کند.
- بتوان با استفاده از اندیس (با شروع از صفر) به عناصر داخل آن دسترسی داشت.

کاربردهای List به صورت کلی عبارت‌اند از :

- نگه داری یک لیست و مجاز بودن اعضای تکراری
- نگه داری یک مجموعه با اندازه پویا و قابل افزایش
- کاربرد ساده‌تر نسبت به آرایه

متدهای List در جدول زیر آمده‌اند، کلاس‌هایی که رابط List را پیاده سازی می‌کنند، موظف هستند این متدها را پیاده سازی کنند.

متدهای List	کاربرد
یک عنصر جدید به List اضافه می‌کند.	add(Object o)
یک عنصر جدید را در اندیس مورد نظر درج می‌کند.	add(index, Object o)
تمام اعضای یک کلکسیون دیگر را به لیست اضافه می‌کند.	addAll(Collection c)
تمام اعضای داخل لیست را حذف می‌کند.	clear()
وجود یک عنصر در لیست را بررسی می‌کند.	contains(Object o)
اگر تمام اعضای کلکسیون c در لیست فعلی وجود داشته باشند true بر می‌گرداند.	containsAll(Collection c)
لیست فعلی را با لیست دیگر مقایسه می‌کند.	equals(Object o)
عنصر موجود در اندیس index را بر می‌گرداند.	get(index)
کل لیسترا محاسبه می‌کند.	hashCode()

اولین اندیس یک عنصر را بر می‌گرداند.	indexOf(Object o)
بررسی می‌کند که لیست خالی است یا خیر؟	isEmpty()
یک iterator برای پیمایش لیست بر می‌گرداند.	Iterator iterator()
در صورت وجود عناصر تکراری آخرین اندیس یک عنصر را بر می‌گرداند.	lastIndexOf(Object o)
یک ListIterator در اختیار ما قرار می‌دهد.	ListIterator()
یک عنصر را از لیست حذف می‌کند.	remove(Object o)
تمام عناصر یک کلکسیون دیگر را از لیست فعلی حذف می‌کند.	removeAll(Collection c)
اشتراک لیست فعلی را با کلکسیون دیگر محاسبه می‌کند و سایر اعضاء را حذف می‌کند.	retainAll(Collection c)
عنصر داخل اندیس index را به مقدار ۰ تغییر می‌دهد.	set(index, Object o)
تعداد عناصر داخل لیست را بر می‌گرداند.	size()
یک زیر-لیست از اندیس fromIndex تا toIndex در اختیار ما قرار می‌دهد.	subList(fromIndex, toIndex)
عناصر را در داخل یک آرایه در اختیار ما قرار می‌دهد.	Object[] toArray()

در ادامه با یک مثال با List آشنا می‌شویم و مثال‌های بیشتر را به آموزش‌های بعدی موقول می‌کنیم.

```
package myfirstprogram;

import java.util.List;
import java.util.LinkedList;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        List list = new LinkedList();
        list.add("A");
        list.add("B");
        list.add("C");
        list.add("D");

        System.out.println(list);

        int indexOfC = list.indexOf("C");
        System.out.println(indexOfC);

        list.set(indexOfC, "X");
        System.out.println(list);
    }
}
```

```
[A, B, C, D]
2
[A, B, X, D]
```

در مثال بالا ابتدا یک `List` ایجاد می‌کنیم، چون `List` یک `interface` است برای ساخت یک شیء جدید از آن باید از یکی از زیر کلاس‌های پیاده سازی شده آن استفاده کنیم که در اینجا `LinkedList` را انتخاب کردیم. سپس با استفاده از متدهای `add()` و `set()` چندین عضو به لیست اضافه کردیم. سپس از چاپ لیست، با استفاده از متدهای `get()` و `size()` می‌توانیم مقدار عضوی را در منظور `indexOfC` بگیریم. بعد از چاپ `C` با استفاده از متدهای `get()` و `set()` مقدار عضوی را در منظور `indexOfC` تغییر می‌دهیم و در انتهای مجددًا لیست را چاپ می‌کنیم.

Map

یکی از `Collection` های اصلی `Map` یکی از `interface` جاواست که برای نگه داری زوج کلید-مقدار (`key-value`) از آن استفاده می‌شود و در پکیج `java.util.Map` قرار دارد. زیر کلاس‌های مختلفی همچون `HashMap`، `WeakHashMap`، `LinkedHashMap` و `TreeMap` در جاوا وجود دارند که هر کدام با استفاده از الگوریتم‌ها و ساختمان داده‌های متفاوت به نحوی مفهوم نگه داری زوج کلید-مقدار را پیاده سازی کرده‌اند. به هر کدام با استفاده از `Entry` گفته می‌شود که با استفاده از `Map`.`Entry` می‌توان به آن‌ها دسترسی داشت و کلید (`key`) یا مقدار (`value`) را استخراج کرد. کلید در `Map` یک عنصر یکتاست ولی مقدار متناظر با آن می‌تواند یک `Object` ساده یا پیچده (مثلًاً یک لیست) و یا حتی `null` باشد، به علاوه ممکن است برای چند کلید مختلف مقادیر تکراری داشته باشیم. `null` نیز می‌تواند یک کلید منحصر به فرد در نظر گرفته شود لذا هر `Map` می‌تواند حداقل یک کلید با مقدار `null` نیز داشته باشد. کاربردهای `Map` به صورت کلی عبارت‌اند از:

- قادر باشیم زوج کلید-مقدار را در `Map` ذخیره کنیم و هر زمان که لازم شد تنها با داشت کلید بتوانیم مقدار متناظر با آن را استخراج کنیم.
- راهکاری برای تشخیص وجود یا عدم وجود کلید در `Map` داشته باشیم.
- راهکاری برای تشخیص وجود یا عدم وجود یک مقدار در `Map` داشته باشیم.
- قادر باشیم مقدار متناظر با یک کلید را تغییر دهیم.
- قادر باشیم تمام اعضای `Map` را لیست کنیم.

متدهای Map در جدول زیر آمده‌اند، کلاس‌هایی که از کلاس Map ارث بری دارند موظف هستند این متدها را پیاده سازی کنند.

متدها	کاربرد
<code>clear()</code>	تمام عناصر داخل Map را پاک می‌کند.
<code>containsKey(Object key)</code>	بررسی می‌کند که آیا کلید (key) مورد نظر در Map وجود دارد یا خیر؟
<code>containsValue(Object val)</code>	بررسی می‌کند که آیا مقدار مورد نظر در Map وجود دارد یا خیره؟
<code>entrySet()</code>	تمام زوج‌های کلید-مقدار را در قالب یک set به ما ارائه می‌دهد.
<code>equals(Object obj)</code>	تساوی این Map با Map دیگری را بررسی می‌کند.
<code>get(Object key)</code>	مقدار متناظر با یک کلید (key) را برمی‌گرداند.
<code>hashCode()</code>	کل Map را محاسبه می‌کند.
<code>isEmpty()</code>	بررسی می‌کند که Map خالی است یا خیر؟
<code>keySet()</code>	تمام کلیدها را در داخل یک در اختیار ما قرار می‌دهد.
<code>put(Object key, Object val)</code>	یک زوج کلید-مقدار را در Map قرار می‌دهد.
<code>putAll(Map m)</code>	تمام عناصر یک Map دیگر را به Map فعلی اضافه می‌کند.
<code>remove(Object key)</code>	یک کلید (key) را حذف می‌کند که این عمل باعث حذف مقدار متناظر با آن نیز می‌شود.
<code>size()</code>	تعداد کلیدهای موجود را برمی‌گرداند.
<code>Collection values()</code>	یک Collection از روی مقادیر ایجاد کرده و برمی‌گرداند.

Map.Entry

یک `interface` برای دسترسی به زوج کلید-مقدار (`entry`) است که متدهایی را برای استخراج کلید یا مقدار در اختیار ما

قرار می‌دهد. متدهای `Map.Entry()` عبارت‌اند از :

متدها	کاربرد
<code>equals(Object o)</code>	تصاویر یک <code>entry</code> را با <code>entry</code> دیگری برسی می‌کند.
<code>getKey()</code>	از داخل یک زوج کلید-مقدار، کلید را استخراج می‌کند.
<code>getValue()</code>	از داخل یک زوج کلید-مقدار، مقدار را استخراج می‌کند.
<code>hashCode()</code>	. متناظر با <code>entry</code> را محاسبه می‌کند.
<code>setValue(Object o)</code>	به کلید فعلی یک مقدار جدید نسبت می‌دهد.

در ادامه با دو مثال با `Map` آشنا می‌شویم و مثال‌های بیشتر را به آموزش‌های بعدی موقول می‌کنیم. در مثال اول یک `Map` به صورت زوچ‌های نام-نمره ایجاد می‌کنیم که در آن کلید نام افراد و مقدار نمره آن‌هاست.

```
package myfirstprogram;

import java.util.Map;
import java.util.HashMap;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Map myMap = new HashMap<>();

        myMap.put("Jenny", 85);
        myMap.put("Jenny", 87);
        myMap.put("Peter", "No Score");
        myMap.put("Mary Jane", 64);
        myMap.put("Azhar", 87);

        System.out.println(myMap);
    }
}

{Azhar=87, Jenny=87, Mary Jane = 64, Peter = No Score}
```

در مثال بالا ابتدا یک شیء از Map ایجاد کردیم، چون Map تنها یک interface است برای استفاده از new و ساخت یک شیء واقعی باید از یکی از زیر کلاس‌های آن استفاده کنیم که در اینجا از HashMap که بعداً با آشنا می‌شویم استفاده کردیم. پس از ساخت Map با استفاده از متدهای put() زوج‌های کلید و مقدار را در Map قرار دادیم، چون کلید نمی‌تواند تکراری باشد پس از فراخوانی متدهای put()، مقدار قبلی متناظر با Jenny یعنی ۸۵ از Map پاک می‌شود. ولی دقت کنید که مقادیر می‌توانند تکراری باشند (برخلاف کلید)، به عنوان مثال Jenny و Azhar هر دو دارای مقدار ۸۷ هستند. در مثال دوم با Map.Entry آشنا می‌شویم و برای سادگی بیشتر از همان کد مثال قبلی استفاده می‌کنیم.

```
package myfirstprogram;

import java.util.Map;
import java.util.Set;
import java.util.HashMap;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Map myMap = new HashMap<>();

        myMap.put("Jenny", 87);
        myMap.put("Peter", "No Score");
        myMap.put("Mary Jane", 64);
        myMap.put("Azhar", 87);

        Set<Map.Entry> s = myMap.entrySet();

        for (Map.Entry me : s)
        {
            System.out.println("[ " + me.getKey() + " ]=( " + me.getValue() + " )");
        }
    }
}

[Azhar] = (79)
[Jenny] = (87)
[Mary Jane] = (64)
[Peter] = (No Score)
```

همانطور که مشاهده می‌کنید کلیدها را بین [] و مقادیر را بین () قرار می‌دهیم. پس از ساخت Map متدهای entry() را فراخوانی می‌کنیم و نتیجه را در یک Set می‌ریزیم، باید مشخص کنیم که عناصر این از نوع Map.Entry هستند. پس از آن در داخل یک حلقه for هر کدام از Map ها را استخراج می‌کنیم و سپس با فراخوانی متدهای getKey() و getValue() مقدار را به دست آورده و چاپ می‌کنیم.

Set

یک از interface های اصلی Collection Framework جاواست که برای نگه داری یک مجموعه بدون تکرار از آن استفاده می‌شود، زیر کلاس‌های مختلفی همچون HashSet و TreeSet در جاوا وجود دارند که هر کدام با استفاده از الگوریتم‌ها و ساختمان داده‌های متفاوت به نحوی مفهوم "مجموعه بدون تکرار" را پیاده سازی کرده‌اند. null نیز می‌تواند یک مقدار منحصر به فرد در نظر گرفته شود لذا هر Set می‌تواند حداکثر یک مقدار null نیز داشته باشد. کاربردهای Set به صورت کلی عبارت‌اند

از :

- نگه داری یک مجموعه بدون تکرار
- صرفه جویی حافظه در الگوریتم‌های جست و جوی پیشرفته (با تشخیص گره‌های تکراری)
- تشخیص عضویت یک شیء در مجموعه با بار محاسباتی کم.
- محاسبه سریع اشتراک دو مجموعه

متدهای Set در جدول زیر آمده‌اند، کلاس‌هایی که از کلاس Set ارث بری دارند موظف هستند این متدها را پیاده سازی کنند.

متدها	کاربرد
add(Object o)	یک عنصر جدید به Set اضافه می‌کند.
addAll(Collection c)	تمام اعضای یک کلکسیون دیگر را به Set اضافه می‌کند.
clear()	تمام اعضای داخل Set را حذف می‌کند.
contains(Object o)	وجود یک عنصر در Set را بررسی می‌کند.
containsAll(Collection c)	اگر تمام اعضای کلکسیون c در Set فعلی وجود داشته باشند true بر می‌گرداند.
equals(Object o)	فعلی را با Set دیگری مقایسه می‌کند.
hashCode()	کل Set را محاسبه می‌کند.

بررسی می‌کند که Set خالی است یا خیر؟	<code>isEmpty()</code>
یک iterator برای پیمایش Set برمی‌گرداند.	<code>iterator()</code>
یک عنصر را از Set حذف می‌کند.	<code>remove(Object o)</code>
تمام عناصر یک کلکسیون دیگر را از Set فعلی حذف می‌کند.	<code>removeAll(Collection c)</code>
اشتراک Set فعلی را با کلکسیون دیگر محاسبه می‌کند و سایر اعضاء را حذف می‌کند.	<code>retainAll(Collection c)</code>
تعداد عناصر داخل Set را برمی‌گرداند.	<code>size()</code>
عناصر را در داخل یک آرایه در اختیار ما قرار می‌دهد.	<code>Object[] toArray()</code>

در ادامه با دو مثال با Set آشنا می‌شویم و مثال‌های بیشتر را به آموزش‌های بعدی موقول می‌کنیم. در مثال اول یک Set ایجاد می‌کنیم و در انتها متوجه می‌شویم که عنصر تکراری در آن نگه داری نمی‌شود.

```
package myfirstprogram;

import java.util.Set;
import java.util.HashSet;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Set set = new HashSet();

        set.add(null);
        set.add("Jenny");
        set.add("Jenny");
        set.add("Jenny");
        set.add("Azhar");
        set.add(null);
        set.add("Azhar");

        System.out.println(set);
    }
}
```

[null, Jenny, Azhar]

همانطور که مشاهده می‌کنید با اینکه عنصر "Jenny" سه بار و عنصر null دوبار به مجموعه اضافه شدند در نتیجه نهایی از هر عنصر تنها یک نمونه وجود دارد. در مثال بالا ابتدا یک شیء از Set ایجاد کردیم، چون Set تنها یک interface است برای استفاده از new و

ساخت یک شیء واقعی باید از یکی از زیر کلاس‌های آن استفاده کنیم که در اینجا از HashSet که بعداً با آشنا می‌شویم استفاده کردیم.

پس از ساخت Set با استفاده از متد add() عناصری را به آن اضافه کردیم. در مثال دوم با retainAll آشنا می‌شویم.

```
package myfirstprogram;

import java.util.Set;
import java.util.HashSet;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Set A = new HashSet();
        Set B = new HashSet();
        A.add(2);
        A.add(4);
        A.add(6);

        B.add(2);
        B.add(4);
        B.add(5);

        System.out.println(A);
        System.out.println(B);

        A.retainAll(B);
        System.out.println(A);
    }
}
```

```
[2, 4, 6]
[2, 4, 5]
[2, 4]
```

در کد قبل دو مجموعه به نام‌های A و B ایجاد می‌کنیم، مجموعه اول شامل اعداد ۲، ۴، ۶ و مجموعه دوم شامل اعداد ۲، ۴، ۵ خواهد بود.

پس از استفاده از retainAll اعضایی از A که در مجموعه B قرار ندارند از A حذف می‌شوند و نتیجه به صورت اشتراک دو مجموعه خواهد

بود.

HashSet

کلاس HashSet زیر کلاس AbstractSet است و مفهوم Set یعنی یک مجموعه بدون تکرار را پیاده سازی می‌کند و در پکیج java.util.HashSet قرار دارد. قبلاً با مفهوم Set آشنا شدیم، کلاس HashSet برای پیاده سازی Set از الگوریتم درهم سازی (hashing) استفاده می‌کند. مهم‌ترین کاربرد HashSet این است که تشخیص دهیم شیء خاصی در مجموعه وجود دارد یا خیر؟ چون از جدول Hash و الگوریتم درهم سازی استفاده می‌کند عمل جست و جو در آن بسیار سریع است و از این ویژگی آن در جست

و جوهای پیشرفته مانند الگوریتم‌های جست و جوی هوش مصنوعی استفاده می‌شود. برای ساخت HashSet باید از یکی از سازنده‌های

زیر استفاده کنیم :

`HashSet()`

یک HashSet خالی با جدول نگه دارنده‌ای با اندازه پیش فرض ایجاد می‌کند.

`HashSet(Collection c)`

یک HashSet جدید ایجاد می‌کند و اعضای کلکسیون c را به آن اضافه می‌کند.

`HashSet(int Capacity)`

یک HashSet خالی با جدول نگه دارنده‌ای با اندازه Capacity ایجاد می‌کند، هر زمان که لازم باشد اندازه جدول نگه دارنده به صورت

خودکار افزایش می‌یابد.

`HashSet(int Capacity, float fillRatio)`

یک HashSet خالی با جدول نگه دارنده‌ای با اندازه Capacity ایجاد می‌کند، هر زمان که لازم باشد اندازه جدول نگه دارنده به صورت خودکار افزایش می‌یابد، پارامتر fillRatio عددی بین ۰٪ و ۱٪ است و مشخص می‌کند که جدول نگه دارنده چه زمان باید تغییر اندازه داده شود، به عنوان مثال ۰/۸ به معنی آن است که هنگامی که ظرفیت جدول به بیش از هشتاد درصد رسید اندازه جدول افزایش یابد.

متدهای مهم HashSet در جدول زیر آمده‌اند :

متدهای مهم HashSet	کاربرد	متدهای مهم HashSet
یک عنصر جدید به مجموعه اضافه می‌کند.		<code>add(Object o)</code>
کل مجموعه را پاک می‌کند.		<code>clear()</code>
وجود یا عدم وجود یک عنصر در مجموعه را بررسی می‌کند.		<code>contains(Object o)</code>
خالی بودن مجموعه را بررسی می‌کند.		<code>isEmpty()</code>
یک Iterator برای پیمایش اعضای مجموعه در اختیار ما قرار می‌دهد.		<code>iterator()</code>

یک عنصر را از مجموعه حذف می‌کند.	<code>remove(Object o)</code>
اندازه مجموعه را برمی‌گرداند.	<code>size()</code>
یک آرایه از روی اعضای مجموعه برمی‌گرداند.	<code>toArray()</code>

مهمترین ویژگی‌های HashSet عبارت‌اند از :

- متدهای `(contains()` و `remove()` و `add()`) در HashSet فوق العاده سریع هستند و این مهم‌ترین ویژگی HashSet است.
- در HashSet عضو تکراری وجود ندارد.
- اعضای داخل مجموعه HashSet ترتیب خاصی ندارند.

در مثال زیر با استفاده از متدهای `add()` و `remove()` اضافه می‌کنیم و سپس کل مجموعه را چاپ می‌کنیم، با متدهای `contains()` و `remove()` عنصری را حذف می‌کنیم و مجدداً کل مجموعه را چاپ می‌کنیم، در انتها نیز وجود یا عدم وجود دو مورد در مجموعه را با متدهای `contains()` بررسی می‌کنیم.

```
package myfirstprogram;

import java.util.HashSet;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        HashSet mySet = new HashSet();

        mySet.add("A");
        mySet.add("B");
        mySet.add("C");
        mySet.add("A");
        mySet.add("D");
        mySet.add("E");

        System.out.println(mySet);

        mySet.remove("D");
        System.out.println(mySet);

        System.out.println(mySet.contains("D"));
        System.out.println(mySet.contains("E"));
    }
}
```

```
[A, B, C, D, E]
[A, B, C, E]
false
true
```

در کد بالا مشاهده می‌کنید که HashSet عضو تکراری نگه داری نمی‌کند، به علاوه هنگامی که عضوی در مجموعه وجود نداشته باشد متندار مقدار `false` را بر می‌گرداند.

LinkedList

کلاس `LinkedList` پیاده سازی `interface List` و `Dequeue` است و برای همین صفات لیست و صف را با هم دارد، این کلاس به صورت ساختمان داده لیست دو پیوندی (Doubly-LinkedList) پیاده سازی شده است و برای همین می‌توان به سادگی و در زمان کم به ابتداء و انتهای عنصر جدید اضافه کرد، به علاوه به دلیل دو پیوندی بودن عمل حذف به صورت بهینه‌تری انجام می‌پذیرد. از این کلاس می‌توان به عنوان یک لیست ساده با دسترسی تصادفی، صف دو طرفه، صف FIFO یا حتی صف LIFO استفاده کرد. این کلاس از لحاظ عملکرد بسیار شبیه `ArrayList` است و تفاوت آن‌ها در ساختمان داده‌ای است که به کار می‌برند، شباهتها و تفاوت این دو کلاس عبارت‌اند از:

- از یک آرایه پویا و `LinkedList` از ساختمان داده لیست دو پیوندی استفاده می‌کند.
- عمل درج یا حذف عنصر در ابتداء در `LinkedList` بسیار سریع است ولی در `ArrayList` ممکن است سریع نباشد.
- عمل درج یا حذف عنصر در انتهای در هر دو کلاس به سرعت انجام می‌شود.
- عمل جست و جو در هر دو کلاس به صورت خطی است و چندان بهینه نیست.
- عمل حذف با داشتن گره، در `LinkedList` سریع‌تر است ولی اگر گره در دسترس نباشد و مجبور به عمل جست و جو قبل از حذف گره باشیم هر دو کلاس تقریباً یکسان هستند و چندان بهینه نیستند.
- دسترسی تصادفی در `ArrayList` بسیار سریع است ولی در `LinkedList` سریع نیست.

کلاس `LinkedList` از طریق مسیر `java.util.LinkedList` قابل دستیابی است. برای ساخت شیء از کلاس `LinkedList` باید از یکی از سازنده‌های زیر استفاده کنیم:

```
LinkedList()
```

این سازنده یک `LinkedList` خالی ایجاد می‌کند.

```
LinkedList(Collection c)
```

این سازنده یک `LinkedList` جدید ایجاد می‌کند و اعضای کلکسیون `c` را به عنوان آیتم‌های اولیه به `LinkedList` اضافه می‌کند.

متدهای مهم `LinkedList` در جدول زیر آمده‌اند :

متدها	کاربرد
<code>add(Object o)</code>	عنصر جدید را به انتهای لیست اضافه می‌کند.
<code>add(int index, Object o)</code>	عنصر جدید را در اندیس مورد نظر درج می‌کند.
<code>addAll(Collection c)</code>	اعضای یک کلکسیون دیگر را به انتهای لیست اضافه می‌کند.
<code>addFirst(Object o)</code>	عنصر جدید را به ابتدای لیست اضافه می‌کند.
<code>addLast(Object o)</code>	عنصر جدید را به انتهای لیست اضافه می‌کند.
<code>clear()</code>	تمام عناصر داخل لیست را پاک می‌کند.
<code>contains(Object o)</code>	بررسی می‌کند که عنصر مورد نظر در لیست وجود دارد یا خیر؟
<code>element()</code>	عنصر ابتدای لیست را بر می‌گرداند ولی آن را حذف نمی‌کند.
<code>get(int index)</code>	عنصر موجود در اندیس مورد نظر را بر می‌گرداند.
<code>getFirst()</code>	عنصر ابتدای لیست را بر می‌گرداند ولی آن را حذف نمی‌کند.
<code>getLast()</code>	عنصر انتهای لیست را بر می‌گرداند ولی آن را حذف نمی‌کند.
<code>indexOf(Object o)</code>	اولین اندیس مورد نظر یک عنصر را پیدا می‌کند.
<code>lastIndexOf(Object o)</code>	آخرین اندیس یک عنصر خاص را پیدا می‌کند.
<code>offer(Object o)</code>	یک عنصر به انتهای لیست اضافه می‌کند.

یک عنصر به ابتدای لیست اضافه می‌کند.	<code>offerFirst(Object o)</code>
یک عنصر به انتهای لیست اضافه می‌کند.	<code>offerLast(Object o)</code>
عنصر ابتدای لیست را برمی‌گرداند ولی آن را حذف نمی‌کند.	<code>peek()</code>
عنصر ابتدای لیست را برمی‌گرداند ولی آن را حذف نمی‌کند	<code>peekFirst()</code>
عنصر انتهای لیست را برمی‌گرداند ولی آن را حذف نمی‌کند	<code>peekLast()</code>
عنصر ابتدای لیست را برمی‌گرداند و آن را حذف می‌کند	<code>poll()</code>
عنصر ابتدای لیست را برمی‌گرداند و آن را حذف می‌کند	<code>pollFirst()</code>
عنصر انتهای لیست را برمی‌گرداند و آن را حذف می‌کند	<code>Object pollLast()</code>
عنصر ابتدای لیست را برمی‌گرداند و آن را حذف می‌کند	<code>pop()</code>
اگر از <code>LinkedList</code> به عنوان <code>Stack</code> استفاده کرده باشیم عنصری را به بالای آن اضافه می‌کند.	<code>push(Object o)</code>
عنصر ابتدای لیست را برمی‌گرداند و آن را حذف می‌کند	<code>remove()</code>
عنصر موجود در اندیس مورد نظر را برمی‌گرداند و حذف می‌کند.	<code>remove(int index)</code>
عنصر مورد نظر را پیدا کرده و حذف می‌کند.	<code>remove(Object o)</code>
عنصر ابتدای لیست را برمی‌گرداند و آن را حذف می‌کند	<code>removeFirst()</code>
عنصر انتهای لیست را برمی‌گرداند و آن را حذف می‌کند	<code>removeLast()</code>
عنصر موجود در اندیس مورد نظر را به شیء دیگری تغییر می‌دهد.	<code>set(int index, Object o)</code>
اندازه لیست را برمی‌گرداند.	<code>size()</code>

یک آرایه معادل از روی عناصر لیست بر می‌گرداند.	toArray()
--	-----------

بسیاری از متدهای فوق عمل یکسانی انجام می‌دهند و اینکه در عین کاربرد از کدام یک از آن‌ها استفاده کنیم بیشتر به سلایق برنامه نویس مربوط می‌شود. در ادامه با چند مثال با کاربرد `LinkedList` آشنا می‌شویم. در این مثال با `LinkedList` به عنوان یک لیست با دسترسی تصادفی آشنا می‌شویم:

```
package myfirstprogram;

import java.util.LinkedList;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        LinkedList myLL = new LinkedList();

        myLL.add("A");
        myLL.add("B");
        myLL.add("C");

        System.out.println(myLL);

        System.out.println(myLL.get(1));
        System.out.println(myLL);

        System.out.println(myLL.remove(1));
        System.out.println(myLL);

        System.out.println(myLL.indexOf("C"));
    }
}
```

```
[A, B, C]
B
[A, B, C]
B
[A, C]
1
```

در کد بالا ابتدا یک `LinkedList` ایجاد می‌کنیم و سپس با استفاده از متدهای `add()` و `get()` عناصری را به آن اضافه می‌کنیم، می‌دانیم که این متدهای `add()` و `get()` از ساخت لیست یک بار آن را چاپ می‌کنند. پس از ساخت لیست یک بار آن را چاپ می‌کنیم. با استفاده از متدهای `remove()` و `indexOf()` عنصر موجود در لیست را به انتهای لیست اضافه می‌کنیم. عنصر موجود در آن‌دیس یک را با متدهای `remove()` و `indexOf()` حذف کرده و آن را چاپ کرده و سپس کل لیست را چاپ می‌کنیم. عنصر موجود در آن‌دیس یک را با متدهای `remove()` و `indexOf()` حذف کرده و آن را چاپ می‌کنیم، می‌بینیم که عنصر `B` حذف شد زیرا در `LinkedList` آن‌دیس از صفر شروع می‌شود. در انتها با استفاده از

متدها indexOf() اندیس عنصر C را به دست آورده و چاپ می‌کنیم. در مثال دوم، به ابتداء و انتهای لیست عنصر اضافه کرده و سپس آنها

را حذف می‌کنیم:

```
package myfirstprogram;

import java.util.LinkedList;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        LinkedList myLL = new LinkedList();

        myLL.add("0");

        myLL.addFirst("A1");
        myLL.addLast("B1");
        System.out.println(myLL);

        myLL.addFirst("A2");
        myLL.addLast("B2");
        System.out.println(myLL);

        myLL.removeFirst();
        System.out.println(myLL);

        myLL.removeLast();
        System.out.println(myLL);
    }
}
```

```
[A1, 0, B1]
[A2, A1, 0, B1, B2]
[A1, 0, B1, B2]
[A1, 0, B1]
```

در کد بالا ابتداء با استفاده از متدها add() و addFirst() یک عنصر به لیست اضافه می‌کنیم، سپس عنصر A1 را با استفاده از متدها removeFirst() و removeLast() بحذف می‌کنیم. انتهای لیست اضافه می‌کنیم و بعد لیست را چاپ می‌کنیم. عنصر A2 را به ابتدای لیست و عنصر B1 را به انتهای لیست اضافه می‌کنیم. با استفاده از متدها addLast() و addFirst() عنصر ابتدای لیست را حذف می‌کنیم و عنصر B2 را به انتهای لیست اضافه می‌کنیم و لیست را چاپ می‌کنیم. با استفاده از متدها removeFirst() و removeLast() عنصر ابتدای لیست را حذف می‌کنیم. با استفاده از متدها addFirst() و addLast() عنصر انتهای لیست را حذف می‌کنیم. می‌توانیم به اشکال زیر از LinkedList به

عنوان پیاده سازی Queue و Deque و List استفاده کنیم:

```
Queue queue = new LinkedList();
Deque deque = new LinkedList();
```

```
List list = new LinkedList();
```

استفاده از `LinkedList` به عنوان صف (Stack) یا همان LIFO

می‌توانیم به عنوان صف LIFO نیز استفاده کنیم، کلمه LIFO مخفف عبارت Last in First Out است و بدین معنی است که عنصری که دیرتر اضافه می‌شود زودتر خارج می‌شود، به چنین صفی Stack نیز گفته می‌شود. در جاوا کلاس اختصاصی برای Stack نیز وجود دارد ولی می‌توانیم از `LinkedList` نیز به جای آن استفاده کنیم:

```
package myfirstprogram;

import java.util.LinkedList;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        LinkedList myLIFO = new LinkedList();

        myLIFO.push("A");
        myLIFO.push("B");
        myLIFO.push("C");

        System.out.println(myLIFO);

        System.out.println(myLIFO.pop());
        System.out.println(myLIFO);

        System.out.println(myLIFO.pop());
        System.out.println(myLIFO);

        System.out.println(myLIFO.pop());
        System.out.println(myLIFO);
    }
}
```

```
[C, B, A]
C
[B, A]
B
[A]
A
[]
```

در کد بالا با استفاده از متدهای `push()` و `pop()` عناصر را به صف LIFO اضافه می‌کنیم، در انتهای اضافه کردن عنصر C در بالای صف قرار خواهد داشت. در ادامه با استفاده از متدهای `push()` و `pop()` عناصر را از صف حذف می‌کنیم، می‌دانیم که در این شیوه عنصری که دیرتر اضافه شده زودتر حذف می‌شود لذا ابتدا C و سپس B و در انتهای A از صف حذف می‌شوند. هر بار پس از عمل حذف صف را نیز چاپ می‌کنیم.

Queue

یکی از `interface` های مهم در `Collection Framework` جاواست که برای تعریف مفهوم صف از آن استفاده می‌شود، صف ساختمان داده‌ای است که بتوان اشیایی را با ترتیبی خاص به آن وارد کرد و با الگوریتم مشخصی از آن استخراج یا حذف نمود، الگوریتم‌ها و پیاده سازی‌های مختلفی از انواع صف وجود دارند که در مباحث بسیار پیشرفته‌ای همچون تئوری صفها (`Queueing Theory`) مطرح می‌شوند، سه پیاده سازی ساده و مهم صف در جاوا عبارت‌اند از:

- `LinkedList`: صفی با ساختار FIFO را پیاده سازی می‌کند که از لیست پیوندی استفاده می‌کند.

- `ArrayList`: صفی با ساختار FIFO است که نحوه پیاده سازی آن با `LinkedList` متفاوت است.

- `PriorityQueue`: صف اولویت را پیاده سازی می‌کند.

کلمه FIFO مخفف عبارت First In First Out است، صف FIFO صفی است که در آن عنصری که ابتدا وارد شود اولین عنصری خواهد بود که خارج می‌شود، مشابه صف نانوایی و بسیاری از صف‌های دیگر که در زندگی روزمره تجربه می‌کنیم. در کنار ساختمان داده‌های ساده و معمولی فوق کلاس‌های پیشرفته‌تری برای کار در مباحثی همچون شبکه، چند نخی (multithreading)، مسائل همزمانی (concurrency) `ConcurrentLinkedDeque`, `LinkedTransferQueue`, `SynchronousQueue`, `DelayQueue` و ... مانند وجود دارند. به صورت کلی یک صف صرف نظر از ساختمان داده داخلی آن و الگوریتم ورود یا خروج اعضا باید دارای ویژگی‌های کلی زیر باشد:

- صف دارای بخشی به نام سر (`head`) باشد.

- بتوان مواردی را به آن اضافه یا حذف کرد.

- بتوان عنصر سر را بدون حذف استخراج کرد (برای تست)

از طریق مسیر `java.util.Queue` قابل دسترسی است و متدهای زیر را تعریف می‌کند. کلاس‌هایی که از این کلاس ارث بری داشته باشند موظف هستند این متدها را پیاده سازی کنند: متدهای مربوط به مفهوم صف:

متدهای مربوط به مفهوم صف:	کاربرد
---------------------------	--------

یک عنصر جدید به صف اضافه می‌کند. اگر قادر به این کار نباشد یک خطا پرتاب می‌کند.	<code>add(Object o)</code>
عنصر سر (<code>head</code>) را بر می‌گرداند ولی آن را از صف حذف نمی‌کند.	<code>element()</code>
یک عنصر جدید به صف اضافه می‌کند اگر قادر به این کار نباشد تنها <code>false</code> بر می‌گرداند.	<code>offer(Object o)</code>
عنصر سر را بر می‌گرداند، اگر صف خالی شده باشد <code>null</code> بر می‌گرداند.	<code>peek()</code>
عنصر سر را بر می‌گرداند و آن را از صف حذف می‌کند، اگر صف خالی باشد <code>null</code> بر می‌گرداند.	<code>poll()</code>
عنصر سر را حذف می‌کند.	<code>remove()</code>

متدهایی که از `Collection` به ارت رفته‌اند:

متدهایی که از <code>Collection</code> به ارت رفته‌اند:	کاربرد
تمام اعضای یک کلکسیون دیگر را به صف اضافه می‌کند.	<code>addAll(Collection c)</code>
تمام اعضای داخل صف را حذف می‌کند.	<code>clear()</code>
وجود یک عنصر در صف را بررسی می‌کند.	<code>contains(Object o)</code>
اگر تمام اعضای کلکسیون <code>c</code> در صف فعلی وجود داشته باشند <code>true</code> بر می‌گرداند.	<code>containsAll(Collection c)</code>
صف فعلی را با صف دیگر مقایسه می‌کند.	<code>equals(Object o)</code>
کل صف را <code>HashCode</code> محاسبه می‌کند.	<code>hashCode()</code>
بررسی می‌کند که صف خالی است یا خیر؟	<code>isEmpty()</code>
یک <code>iterator</code> برای پیمایش صف بر می‌گرداند.	<code>iterator()</code>
یک عنصر را از صف حذف می‌کند.	<code>remove(Object o)</code>

تمام عناصر یک کلکسیون دیگر را از صف فعلی حذف می‌کند.	removeAll(Collection c)
اشتراک صف فعلی را با کلکسیون دیگر محاسبه می‌کند و سایر اعضاء را حذف می‌کند.	retainAll(Collection c)
تعداد عناصر داخل صف را برمی‌گرداند.	size()
عناصر را در داخل یک آرایه در اختیار ما قرار می‌دهد.	toArray()

در مثال زیر یک صف ساده ایجاد می‌کنیم، عناصری به آن اضافه می‌شوند و سپس تک تک عناصر را از صف خارج می‌کنیم.

```
package myfirstprogram;

import java.util.Queue;
import java.util.LinkedList;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Queue q = new LinkedList();

        q.add(1);
        q.add(8);
        q.add(3);

        System.out.println(q);

        System.out.println(q.poll());
        System.out.println(q);

        System.out.println(q.poll());
        System.out.println(q);

        System.out.println(q.poll());
        System.out.println(q);
    }
}
```

```
[1, 8, 3]
1
[8, 3]
8
[3]
3
[]
```

در کد بالا یک Queue ایجاد کردیم، چون Queue تنها یک interface است برای ساخت شیء جدید باید از یکی از زیر کلاس‌های پیاده

سازی شده آن استفاده کنیم، در این مثال از LinkedList استفاده کردیم، با استفاده از متده add() عناصر ۱، ۸ و ۳ را به صف اضافه کرده

و سپس کل صف را چاپ نمودیم. در ادامه با استفاده متدهای `poll()` عنصر سرصف را حذف و چاپ کرده و پس از آن صف باقیمانده را نیز چاپ میکنیم، مشاهده میکنیم عناصر با همین ترتیبی که وارد صف شده‌اند از صف نیز خارج می‌شود زیرا `LinkedList` صفحی با ساختار FIFO است.

HashMap

قبلًا با مفهوم `Map` به عنوان وسیله‌ای برای نگه داری زوج‌های کلید-مقدار آشنا شدیم، کلاس `HashMap` زیرکلاس `AbstractMap` است و از الگوریتم درهم سازی برای نگه داری زوج‌ها نگه داری می‌کند. مهمترین مزیت `HashMap` این است که دسترسی به کلیدها در زمان بسیار کوتاهی صورت می‌گیرد و با داشتن کلید به سرعت می‌توانیم مقدار را به دست بیاوریم. از `HashMap` می‌توان برای ساخت جداولی همچون دفترچه تلفن، لیست نمرات و غیره استفاده کرد. به علاوه در بسیاری از کاربردهای هوش مصنوعی از `HashMap` برای شناسایی گره‌های تکراری در جست و جوهای پیشرفته استفاده می‌شود. در `HashMap` مقدار `null` هم به عنوان کلید و هم به عنوان مقدار مجاز است ولی حداقل بیک کلید با مقدار `null` می‌تواند در آن وجود داشته باشد. کلاس `HashMap` از طریق مسیبر `java.util.HashMap` قابل دستیابی است. برای ساخت شیء جدید از کلاس `HashMap` باید از یکی از سازنده‌های زیر استفاده کنیم :

```
HashMap()
```

یک `HashMap` جدید با جدول نگه دارنده خالی ایجاد می‌کند.

```
HashMap(int capacity)
```

یک `HashMap` جدید با جدول نگه دارنده‌ای با اندازه `capacity` ایجاد می‌کند، البته هر زمان که لازم باشد جدول نگه دارنده به صورت خودکار افزایش اندازه خواهد داشت.

```
HashMap(int capacity, float loadFactor)
```

مشابه سازنده قبلی است با این تفاوت که پارامتر `loadFactor` تعیین می‌کند که چه زمان اندازه جدول نگه دارنده باید افزایش یابد.

```
HashMap(Map mp)
```

یک `HashMap` جدید ایجاد می‌کند و اعضای `mp` را به آن اضافه می‌کند. متدهای مهم `HashMap` در جدول زیر آمدده‌اند :

توضیح

متدها

کل مجموعه را پاک می‌کند.	<code>clear()</code>
بررسی می‌کند که شیء مورد نظر در بخش کلیدها قرار دارد یا خیر؟	<code>containsKey(Object key)</code>
بررسی می‌کند که شیء مورد نظر رد بخش مقادیر قرار دارد یا خیر؟	<code>containsValue(Object value)</code>
یک مجموعه از کل مقادیر موجود در <code>HashMap</code> بر می‌گرداند.	<code>entrySet()</code>
مقدار متناظر با یک کلید را بر می‌گرداند.	<code>get(Object key)</code>
بررسی می‌کند که مجموعه خالی است یا خیر؟	<code>isEmpty()</code>
مجموعه کلیدها را بر می‌گرداند.	<code>keySet()</code>
مقدار <code>v</code> را به کلید <code>k</code> نسبت می‌دهد.	<code>put(Object k, Object v)</code>
تمام اعضای <code>mp</code> را به مجموعه اضافه می‌کند.	<code>putAll(Map mp)</code>
کلید مورد نظر و مقدار متناظر با آن را از مجموعه حذف می‌کند.	<code>remove(Object key)</code>
اندازه مجموعه (تعداد کلیدها) را بر می‌گرداند.	<code>size()</code>
مجموعه مقادیر را به صورت یک کلکسیون بر می‌گرداند.	<code>Collection values()</code>

در مثال زیر یک دفترچه تلفن ساده طراحی می‌کنیم که کلید نام افراد و مقدار شماره آن‌هاست، هدف آن است که با داشتن نام اشخاص

به سرعت بتوانیم شماره آن‌ها را بازیابی کنیم.

```
package myfirstprogram;

import java.util.HashMap;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        HashMap myHashMap = new HashMap();

        myHashMap.put("Jenny", "0916532365");
        myHashMap.put("Peter", "0912534689");
```

```

    myHashMap.put("Mary Jane", "09152136458");
    myHashMap.put("Azhar", "09132536489");

    System.out.println(myHashMap.get("Jenny"));
}
}

```

0916532365

در کد بالا ابتدا یک سری داده با استفاده از متد `put()` به `HashMap` اضافه می‌کنیم، کلید نام افراد و مقدار شماره آن‌هاست. در ادامه با استفاده از متد `get()` شماره شخصی با نام Jenny را از مجموعه استخراج کرده و چاپ می‌کنیم. می‌دانیم که در ساختار `HashMap` کلید منحصر به فرد است و کلید تکراری وجود ندارد ولی در یک دفترچه تلفن، ممکن است برای یک نفر چندین شماره ثبت کرده باشیم، برای اینکه بتوانیم به یک کلید چندین مقدار نسبت دهیم باید از راهکاری ابتکاری استفاده کنیم. برای این منظور می‌توانیم مقدار را به صورت یک کلکسیون در نظر بگیریم. در ادامه با این روش آشنا می‌شویم.

```

1 package myfirstprogram;
2
3 import java.util.HashMap;
4 import java.util.TreeSet;
5
6 public class MyFirstProgram
7 {
8     static HashMap myHashMap = new HashMap();
9
10    static void putNumber(String name, String phoneNumber)
11    {
12        TreeSet list;
13
14        if (myHashMap.get(name) == null)
15        {
16            list = new TreeSet();
17        }
18        else
19        {
20            list = (TreeSet) myHashMap.get(name);
21        }
22
23        list.add(phoneNumber);
24
25        myHashMap.put(name, list);
26    }
27
28    public static void main(String[] args)
29    {
30        putNumber("Jenny", "0916532365");
31        putNumber("Peter", "0912534689");
32        putNumber("Jenny", "09152136458");
33        putNumber("Azhar", "09132536489");
34    }
}

```

```

35     }
36 }
37 }
```

```
[09152136458, 0916532365]
```

در کد بالا از HashMap به شیوه‌ای استفاده کردیم که کلید در آن از نوع String و مقدار در آن TreeSet است لذا مقدار در آن می‌تواند شامل چندین عضو باشد مهمترین قسمت کد بالا متدهای phoneNumber است. در ابتدای این متدها با استفاده از یک if مشخص می‌کنیم که آیا قبلًا مقداری برای کلید name نسبت داده شد است؟ اگر خروجی متدهای get() برابر با null باشد یعنی قبلًا به این کلید (name) هیچ مقداری نسبت داده نشده است لذا یک لیست خالی از جنس TreeSet ایجاد می‌کنیم، در غیر این صورت (کد داخل بخش else)، لیست نمراتی که قبلًا به کلید (name) نسبت داده شده بود را استخراج می‌کنیم. بعد از بررسی بالا می‌توانیم شماره جدید را با استفاده از متدهای add() به لیست اضافه کنیم (می‌دانیم که TreeSet مقدار تکراری نگه داری نمی‌کند و به علاوه اعضا در آن به صورت مرتب شده، نگه داری می‌شوند)، پس از آن با استفاده از متدهای put() لیست جدید را به کلید (name) نسبت می‌دهیم. همانطور که در خروجی می‌کنید هنگامی که عبارت "Jenny" را به عنوان کلید myHashMap به کار می‌بریم خروجی به صورت یک لیست دو عنصری چاپ می‌شود (در حقیقت محتوای یک TreeSet چاپ می‌شود). در این مثال با بعضی از متدهای مهم HashMap آشنا می‌شویم.

```

package myfirstprogram;

import java.util.Map;
import java.util.Set;
import java.util.HashMap;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        HashMap myHashMap = new HashMap();

        myHashMap.put("A", "10");
        myHashMap.put("B", "20");
        myHashMap.put("C", "30");
        myHashMap.put("D", "40");
        myHashMap.put("E", "50");

        //containsKey
        System.out.println(myHashMap.containsKey("A"));
        System.out.println(myHashMap.containsKey("10"));

        //containsValue
        System.out.println(myHashMap.containsValue("A"));
        System.out.println(myHashMap.containsValue("10"));

        Set<Map.Entry<String, String>> allValue = myHashMap.entrySet();
    }
}
```

```

for (Map.Entry<String, String> x : allValue)
{
    System.out.println(x.getKey() + " : " + x.getValue());
}
}

true
false
false
true
D : 40
E : 50
A : 10
B : 20
C : 30

```

در کد بالا ابتدا یک `HashMap` جدید ایجاد کردیم و با استفاده از متدهای `put()` و `get()` زوج‌های کلید و مقداری را در آن درج کردیم. سپس متدهای `containsKey()` و `containsValue()` را مورد بررسی قرار دادیم، این متدهای `containsKey()` و `containsValue()` یعنی می‌بینیم که کلید A وجود دارد ولی کلید A وجود ندارد. سپس از آن متدهای `entrySet()` و `keySet()` را مورد بررسی قرار دادیم، این متدهای `entrySet()` و `keySet()` با اعضای از نوع `Map.Entry` و `Set` هستند. پس از آن با استفاده از متدهای `getKey()` و `getValue()` مقدار مجموعه را در یک `Set` با اعضای از نوع `String` ذخیره کردیم. در داخل یک حلقه `for` هر آیتم از `allValue` را در متغیر `x` استخراج کرده سپس با متدهای `getKey()` و `getValue()` کلید و مقدار آن را به دست آورده و چاپ کردیم.

TreeMap

کلاس `TreeMap` زیر کلاس `AbstractMap` است و مفهوم `Map` را پیاده سازی می‌کند، قبل از اینکه `Map` آشنا شدیم، کلاس `TreeMap` درخت موازن Red-Black برای نگهداری زوج کلید-مقدار استفاده می‌کند و درخت مربوطه از روی کلیدها ساخته می‌شود، برای خلاف `HashMap` که نظم خاصی را برای کلیدها حفظ نمی‌کند در `TreeMap` کلیدها به صورت پیش فرض بر اساس ترتیب الفبایی نگهداری می‌شوند. عملیات درج، حذف و جست و جو در `TreeMap` بسیار سریع است و این ویژگی آن باعث می‌شود به عنوان یک ابزار بهینه سازی از آن استفاده شود. کلاس `TreeMap` از طریق مسیر `java.util.TreeMap` قابل دستیابی است. برای ساخت یک شیء جدید از `TreeMap` می‌توانیم از یکی از سازنده‌های زیر استفاده کنیم :

`TreeMap()`

این سازنده یک TreeMap خالی ایجاد می‌کند.

TreeMap(Comparator cmp)

اگر بخواهیم کلیدها بر اساس یک ترتیب اختصاصی نگه داری شوند می‌توانیم برای اینکار یک Comparator ایجاد کنیم و آن را به سازنده ارسال کنیم.

TreeMap(Collection c)

این سازنده یک TreeMap جدید ایجاد می‌کند و اعضای کلکسیون c را به آن اضافه می‌کند.

TreeMap(SortedMap sm)

مشابه سازنده قبلی است با این تفاوت که در اینجا آیتم‌ها با همان ترتیب sm در TreeMap نگه داری می‌شوند. متدهای مهم در جدول زیر آمده‌اند:

متدها	کاربرد
ceilingEntry(Object key)	یک زوج کلید-مقدار که کلید آن بزرگ‌تر یا مساوی با key باشد بر می‌گرداند.
ceilingKey(Object key)	یک کلید که بزرگ‌تر یا مساوی با key باشد بر می‌گرداند.
clear()	کل مجموعه را پاک می‌کند.
comparator()	ای که برای مقایسه کلیدها به کار می‌رود را بر می‌گرداند اگر مجموعه از ترتیب طبیعی پیروی کند و Comparator نداشته باشد خروجی این متده null خواهد بود.
containsKey(Object key)	بررسی می‌کند که کلید مورد نظر وجود دارد یا خیره؟
containsValue(Object value)	بررسی می‌کند که مقدار مورد نظر وجود دارد یا خیر؟
descendingKeySet()	یک لیست معکوس از کلیدهای مجموعه را بر می‌گرداند.

معکوس مپ فعلی را بر می‌گرداند. (ترتیب کلیدها معکوس خواهد بود)	<code>NavigableMap descendingMap()</code>
تمامی زوج‌های کلید-مقدار را به صورت یک <code>Set</code> بر می‌گرداند.	<code>entrySet()</code>
اولین زوج کلید-مقدار با کوچک‌ترین کلید را بر می‌گرداند.	<code>Map.Entry firstEntry()</code>
کوچک‌ترین کلید را بر می‌گرداند.	<code>firstKey()</code>
یک زوج کلید-مقدار بر می‌گرداند که در آن کلید کوچک‌تر یا مساوی با <code>key</code> باشد.	<code>Map.Entry floorKey(Object key)</code>
کلیدی که کوچک‌تر یا مساوی با <code>key</code> باشد بر می‌گرداند.	<code>floorKey(Object key)</code>
مقدار متناظر با یک کلید را بر می‌گرداند.	<code>Object get(Object key)</code>
یک زیر مجموعه به صورت <code>SortedMap</code> بر می‌گرداند که در آن کلیدها از <code>toKey</code> کوچک‌تر هستند.	<code>headMap(Object toKey)</code>
یک زوج کلید-مقدار بر می‌گرداند که در آن کلید از <code>key</code> بزرگ‌تر باشد.	<code>higherEntry(Object key)</code>
کلیدی که از <code>key</code> بزرگ‌تر باشد را بر می‌گرداند.	<code>higherKey(Object key)</code>
کل کلیدها را بر می‌گرداند.	<code>keySet()</code>
آخرین زوج کلید-مقدار با بزرگ‌ترین کلید را بر می‌گرداند.	<code>Map.Entry lastEntry()</code>
بزرگ‌ترین کلید را بر می‌گرداند.	<code>Object lastKey()</code>
یک زوج کلید-مقدار بر می‌گرداند که در آن کلید از <code>key</code> کوچک‌تر باشد.	<code>lowerEntry(Object key)</code>
اولین زوج کلید-مقدار با کوچک‌ترین کلید را حذف می‌کند.	<code>pollFirstEntry()</code>
آخرین زوج کلید-مقدار با بزرگ‌ترین کلید را حذف می‌کند.	<code>pollLastEntry()</code>

مقدار value را به کلید key نسبت می‌دهد.	put(Object key, Object value)
کل اعضای mp را به مجموعه اضافه می‌کند.	void putAll(Map mp)
زوج کلید-مقدار منتظر با کلید key را حذف می‌کند.	remove(Object key)
تعداد کلیدها را بر می‌گرداند.	size()
یک زیر مجموعه بر می‌گرداند.	subMap(Object fromKey, Object toKey)
زیر مجموعه‌ای را بر می‌گرداند که در آن کلیدها بزرگ‌تر یا مساوی fromkey هستند.	SortedMap tailMap(Object fromKey)
مجموعه‌ای از مقادیر را بر می‌گرداند.	values()

به مثال زیر توجه کنید :

```
package myfirstprogram;

import java.util.Map;
import java.util.TreeMap;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        TreeMap treemap = new TreeMap();

        treemap.put(2, "A");
        treemap.put(9, "B");
        treemap.put(3, "A");
        treemap.put(7, "D");
        treemap.put(7, "X");
        treemap.put(12, "C");

        System.out.println(treemap);

        Map.Entry first = treemap.pollFirstEntry();
        System.out.println(first.getKey() + ", " + first.getValue());

        System.out.println(treemap);
        System.out.println(treemap.get(3));
        System.out.println(treemap.keySet());
    }
}
```

```

System.out.println("<=7 :" + treemap.floorKey(7));
System.out.println(">=7 :" + treemap.ceilingKey(7));
System.out.println("<7 :" + treemap.lowerKey(7));
System.out.println(">7 :" + treemap.higherKey(7));

System.out.println("<=5 :" + treemap.floorKey(5));
System.out.println(">=5 :" + treemap.ceilingKey(5));
System.out.println("<5 :" + treemap.lowerKey(5));
System.out.println(">5 :" + treemap.higherKey(5));
}

{2=A, 3=A, 7=X, 9=B, 12=C}
2 , A
{3=A, 7=X, 9=B, 12=C}
A
[3, 7, 9, 12]
<=7 :7
>=7 :7
<7 :3
>7 :9
<=5 :3
>=5 :7
<5 :3
>5 :7

```

در مثال بالا ابتدا یک TreeMap ایجاد کرده و به صورت نامنظم داده‌هایی را به آن اضافه کرده و سپس آن را چاپ می‌کنیم، مشاهده می‌کنیم که مجموعه به صورت مرتب (بر اساس کلیدها) چاپ می‌شود، به علاوه چون کلیدها منحصر به فرد است برای کلید 7 فقط مقدار 2 (آخرین مقداری که برای این کلید درج شد) نگه داری می‌شود. همچنین می‌بینیم که مقادیر می‌توانند تکراری باشند (مقادیر کلیدهای X و 3 برابر با A است). سپس اولین زوج کلید-مقدار را حذف و چاپ می‌کنیم و بعد از آن کل مجموعه را مجددًا چاپ می‌کنیم. مقدار متضطرر با کلید 3 را چاپ می‌کنیم. مجموعه کل کلیدها را با متدهای keySet() به دست آورده و چاپ می‌کنیم. در ادامه یک بار با کلید 7 که جزو کلیدهای مجموعه است و یک بار با کلید 5 که جزو کلیدهای مجموعه نیست متدهای floorKey() و ceilingKey() و lowerKey() را مورد بررسی قرار می‌دهیم.

TreeSet

کلاس TreeSet زیر کلاس AbstractSet است و مفهوم Set یعنی یک مجموعه بدون تکرار را پیاده سازی می‌کند، قبلًا با مفهوم آشنا شدیم، کلاس TreeSet از ساختمان داده درخت موازنۀ برای نگه داری اعضای مجموعه استفاده می‌کند، لذا اعضای این مجموعه به صورت مرتب نگه داری می‌شوند و عملیات درج، حذف، جست و جو در زمان کوتاهی (زمان لگاریتمی) صورت می‌پذیرد.

در حقیقت TreeSet از TreeMap استفاده می‌کند و کلاس TreeMap با استفاده از ساختمان داده Red-Black Tree پیاده سازی شده

است که بسیار سریع است.

اشیای داخل TreeSet به صورت پیش فرض بر اساس ترتیب الفبایی مرتب هستند مگر آنکه در سازنده آن صریحاً از یک Comparator

اختصاصی استفاده کنیم. مهمترین ویژگی‌های TreeSet عبارت‌اند از :

- همیشه مجموعه مورد نظر مرتب است و برای مرتب سازی نیاز به استفاده از الگوریتم‌های sort نداریم.
- عملیات درج، حذف و جست و جو همگی با سرعت بسیار بالایی اجرا می‌شوند.

به دلیل بهینه بودن TreeSet از آن در الگوریتم‌های پیشرفته‌ای همچون الگوریتم‌های مسیر یابی استفاده می‌شود. از طریق مسیر j می‌توانیم به این کلاس دسترسی داشته باشیم. برای ساخت شیء جدید از TreeSet می‌توانیم از یکی از سازنده‌های زیر استفاده کنیم :

`TreeSet()`

یک TreeSet خالی ایجاد می‌کند.

`TreeSet(Collection c)`

یک TreeSet ایجاد می‌کند و اعضای کلکسیون c را به آن اضافه می‌کند.

`TreeSet(Comparator cmp)`

از cmp برای مقایسه و مرتب سازی اعضا استفاده می‌کند.

`TreeSet(SortedSet ss)`

یک TreeSet جدید ایجاد کرده و اعضای ss را به آن اضافه می‌کند به علاوه ترتیب اعضا را نیز به همان شکل قبلی نگه می‌دارد. متدهای

مهم TreeSet در جدول زیر آمده‌اند :

کاربرد

متدها

یک عضو جدید به مجموعه اضافه می‌کند.	<code>boolean add(Object o)</code>
اعضای یک کلکسیون دیگر را به مجموعه اضافه می‌کند.	<code>boolean addAll(Collection c)</code>
عضوی از مجموعه که از شیء <code>Object</code> بزرگ‌تر یا مساوی باشد را بر می‌گرداند، اگر چنینشی ای وجود نداشته باشد خروجی <code>null</code> خواهد بود.	<code>Object ceiling(Object o)</code>
کل مجموعه را پاک می‌کند.	<code>void clear()</code>
مجموعه <code>Comparator</code> را بر می‌گرداند، اگر از ترتیب طبیعی استفاده شده باشد خروجی <code>null</code> خواهد بود.	<code>Comparator comparator()</code>
وجود یا عدم وجود یک شیء در مجموعه را بررسی می‌کند.	<code>boolean contains(Object o)</code>
یک <code>Iterator</code> معکوس در اختیار ما قرار می‌دهد که می‌توانیم مجموعه را از انتهای به ابتدا پیمایش کنیم.	<code>Iterator descendingIterator()</code>
معکوس مجموعه فعلی را در اختیار ما قرار می‌دهد.	<code>NavigableSet descendingSet()</code>

عضو ابتدایی (کوچکترین عضو) مجموعه را بر می‌گرداند.	<code>Object first()</code>
عضوی از مجموعه که از شیء <code>o</code> کوچکتر یا مساوی باشد را بر می‌گرداند، اگر چنین موردی وجود نداشته باشد خروجی <code>null</code> خواهد بود.	<code>Object floor(Object o)</code>
زیر مجموعه‌ای را بر می‌گرداند که اعضای آن از <code>to</code> کوچکتر باشند.	<code>SortedSet headSet(Object to)</code>
مانند متد قبلی است با این تفاوت که پارامتر دوم مشخص می‌کند که اشیا منحصراً از <code>to</code> کوچکتر باشند یا کوچکتر مساوی؟	<code>NavigableSet headSet(Object to,boolean include)</code>
شی ای از مجموعه را بر می‌گرداند که از <code>o</code> بزرگتر باشد.	<code>Object higher(Object o)</code>
خالی بودن مجموعه را بررسی می‌کند.	<code>boolean isEmpty()</code>
یک <code>Iterator</code> برای پیمایش ابتدا تا انتهای لیست در اختیار ما قرار می‌دهد.	<code>Iterator iterator()</code>
آخرین (بزرگترین) عضو مجموعه را بر می‌گرداند.	<code>Object last()</code>

عضوی از مجموعه که از شیء ۰ کوچک‌تر باشد را برمی‌گرداند.	<code>Object lower(Object o)</code>
عضو ابتدایی (کوچک‌ترین عضو) را از مجموعه حذف می‌کند.	<code>Object pollFirst()</code>
عضو انتهایی (بزرگ‌ترین عضو) را از مجموعه حذف می‌کند.	<code>Object pollLast()</code>
شی مورد نظر را از مجموعه حذف می‌کند.	<code>boolean remove(Object o)</code>
اندازه مجموعه را برمی‌گرداند.	<code>int size()</code>
یک زیر مجموعه که اعضای آن از <code>from</code> بزرگ‌تر و از <code>to</code> کوچک‌تر باشند بر می‌گرداند. پارامترهای <code>ti</code> و <code>fi</code> تعیین می‌کنند که اعضای مساوی با <code>to</code> و <code>from</code> نیز در زیر مجموعه قرار گیرند یا خیر؟	<code>NavigableSet subSet(Object from,boolean fi,Object to,boolean ti)</code>
یک زیر مجموعه که اعضای آن از <code>from</code> بزرگ‌تر-مساوی و از <code>to</code> کوچک‌تر باشند بر می‌گرداند.	<code>SortedSet subSet(Object from,Object to)</code>
یک زیر مجموعه بر می‌گرداند که اعضای آن از <code>from</code> بزرگ‌تر باشند.	<code>SortedSet tailSet(Object from)</code>
یک زیر مجموعه بر می‌گرداند که اعضای آن از <code>from</code> بزرگ‌تر باشند.	<code>NavigableSet tailSet(Object from,boolean include)</code>

پارامتر دوم مشخص می‌کند که
اشیا از `from` بزرگ‌تر باشند یا
بزرگ‌تر مساوی؟

در ادامه با چند مثال با `TreeSet` بیشتر آشنا می‌شویم. در مثال زیر یک `TreeSet` ساده ایجاد می‌کنیم و چند متده بسیار مهم آن را مورد بررسی قرار می‌دهیم.

```
package myfirstprogram;

import java.util.TreeSet;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        TreeSet treeset = new TreeSet();

        treeset.add(5);
        treeset.add(1);
        treeset.add(4);
        treeset.add(9);
        treeset.add(1);
        treeset.add(6);
        treeset.add(5);

        System.out.println(treeset);

        //first and last
        System.out.println(treeset.first());
        System.out.println(treeset.last());

        //higher
        System.out.println(">5 :" + treeset.higher(5));
        System.out.println(">8 :" + treeset.higher(8));

        //ceiling
        System.out.println(">=5 :" + treeset.ceiling(5));
        System.out.println(">=8 :" + treeset.ceiling(8));

        //lower
        System.out.println("<5 :" + treeset.lower(5));
        System.out.println("<8 :" + treeset.lower(8));

        //floor
        System.out.println("<=5 :" + treeset.floor(5));
        System.out.println("<=8 :" + treeset.floor(8));
    }
}

[1, 4, 5, 6, 9]
1
```

```

9
>5 :6
>8 :9
>=5 :5
>=8 :9
<5 :4
<8 :6
<=5 :5
<=8 :6

```

در کد بالا ابتدا یک TreeSet ایجاد می‌کنیم و سپس اعدادی را با استفاده از متدهای add به آن اضافه می‌کنیم و سپس مجموعه را چاپ می‌کنیم مشاهده می‌کنیم که مجموعه دارای عضو تکراری نیست و به صورت مرتب نگه داری می‌شود. پس از آن با استفاده از متدهای first و last اعضای ابتدایی (کوچکترین) و انتهایی (بزرگترین) مجموعه را چاپ می‌کنیم. سپس متدهای higher و ceiling را مورد بررسی قرار می‌دهیم، مشاهده می‌کنیم که این متدهای عضوی را بر می‌گرداند که حتماً از شیء مورد نظر بزرگ‌تر باشد. سپس متدهای lower و floor را مورد بررسی قرار می‌دهیم، مشاهده می‌کنیم که این متدهای عضوی را بر می‌گرداند که حتماً از شیء مورد نظر کوچک‌تر باشد. سپس متدهای pollFirst و pollLast را مورد بررسی قرار می‌دهیم، مشاهده می‌کنیم که این متدهای عضوی را بر می‌گرداند که کوچک‌تر یا مساوی با شیء مورد نظر باشد.

```

package myfirstprogram;

import java.util.TreeSet;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        TreeSet treeset = new TreeSet();

        treeset.add(5);
        treeset.add(1);
        treeset.add(4);
        treeset.add(9);
        treeset.add(8);
        treeset.add(6);
        treeset.add(5);

        System.out.println(treeset);

        System.out.println(treeset.subSet(4, 8));

        System.out.println(treeset.pollFirst());
        System.out.println(treeset.pollLast());
        System.out.println(treeset);
    }
}

```

```

        System.out.println(treeset.descendingSet());
    }
}

```

```

[1, 4, 5, 6, 8, 9]
[4, 5, 6]
1
9
[4, 5, 6, 8]
[8, 6, 5, 4]

```

در کد بالا مشابه کد قبلی یک TreeSet ایجاد می‌کنیم و سپس آن را چاپ می‌کنیم، سپس متدها subSet() را با پارامترهای ۴ و ۸ فراخوانی می‌کنیم مشاهده می‌کنیم که اعضای زیر مجموعه مورد نظر بزرگتر مساوی ۴ و کوچکتر از ۸ هستند. سپس عنصر ابتدایی را حذف و چاپ می‌کنیم. سپس عنصر انتهایی را حذف و چاپ می‌کنیم. مجدداً کل مجموعه را چاپ می‌کنیم. سپس با استفاده از متدها descendingSet() و subSet() عکس مجموعه را به دست آورده و آن را چاپ می‌کنیم.

Stack

زیر کلاس Vector است و ساختمان داده‌ای با همین نام را پیاده سازی می‌کند، در حقیقت Stack یک صف LIFO محسوب می‌شود، به ساختمان داده Stack پشته نیز گفته می‌شود. عبارت Last In First Out LIFO مخفف صفی است که عضوی که آخر وارد می‌شود اول خارج می‌شود. کاربردهای زیادی در برنامه نویسی دارد که از جمله آن‌ها می‌توان به موارد زیر اشاره کرد :

- تبدیل postfix به infix
- تبدیل الگوریتم‌های بازگشتی به غیر بازگشتی
- فرم غیر بازگشتی جست و جوی اول عمق (Depth First Search)
- بعضی از الگوریتم‌های زمان بندی

یکی از موارد روزمره‌ای که با آن رو به رو می‌شویم حافظه مرورگرها در مورد url هاست، هنگامی که دکمه بازگشت را می‌زنیم همیشه به آخرین مورد قبلی باز می‌گردیم. کلاس Stack از طریق مسیر `java.util.Stack` قابل دستیابی است. برای ساخت یک شیء جدید از کلاس Stack باید از سازنده زیر استفاده کنیم :

```
Stack()
```

متدهای مهم Stack در جدول زیر آمده‌اند ولی مهم‌ترین آن‌ها، peek و pop هستند.

متدها	کاربرد
add(Object o)	یک عنصر به انتهای مجموعه اضافه می‌کند.
add(int index, Object o)	یک عنصر در اندیس مورد نظر درج می‌کند.
addAll(Collection c)	اعضای یک کلکسیون دیگر را به انتهای مجموعه اضافه می‌کند.
AddElement(Object o)	یک عنصر به انتهای مجموعه اضافه می‌کند.
capacity()	ظرفیت مجموعه را بر می‌گرداند.
clear()	کل مجموعه را پاک می‌کند.
contains(Object o)	وجود یا عدم وجود یک عنصر را بررسی می‌کند.
elementAt(int index)	عنصر موجود در اندیس index را بر می‌گرداند.
elements()	یک Enumeration از اعضای مجموعه بر می‌گرداند.
boolean empty()	خالی بودن یا نبودن مجموعه را بررسی می‌کند.
firstElement()	اولین عنصر مجموعه را بر می‌گرداند.
get(int index)	عنصر موجود در اندیس مورد نظر را بر می‌گرداند.
int indexOf(Object o)	اندیس یک عنصر را پیدا می‌کند.
boolean isEmpty()	خالی بودن مجموعه را بررسی می‌کند.
iterator()	یک Iterator برای پیمایش مجموعه بر می‌گرداند.
Object lastElement()	آخرین عضو مجموعه را بر می‌گرداند.

عنصر بالای پشته (آخرین عنصر) را بر می‌گرداند ولی آن را از پشته حذف نمی‌کند.	peek()
عنصر بالای پشته را بر گردانده و آن را از پشته حذف می‌کند.	pop()
عنصر جدید را به بالای پشته اضافه می‌کند.	Object push(Object o)
عنصر موجود در اندیس index را حذف می‌کند.	remove(int index)
اندیس یک شیء را بر می‌گرداند (در این حالت شروع اندیس از ۱ تعبیر می‌شود)	search(Object o)
عنصر موجود در اندیس index را به شیء ۰ تغییر می‌دهد.	Object set(int index, Object o)
کل مجموعه را به صورت آرایه بر می‌گرداند.	toArray()

به مثال زیر توجه کنید :

```
package myfirstprogram;

import java.util.Stack;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Stack stack = new Stack();

        stack.push("URL_1");
        stack.push("URL_2");
        stack.push("URL_3");
        stack.push("URL_4");

        while (!stack.empty())
        {
            System.out.println(stack.pop());
        }
    }
}
```

```
URL_4
URL_3
URL_2
URL_1
```

مثال بالا عملکرد یک مرورگر را به صورت خیلی ساده شبیه سازی می‌کند، ابتدا URL_1 و سپس URL_2 و ... را مشاهده می‌کنیم. عمل مشاهده را با عمل push شبیه سازی می‌کنیم. سپس عمل فشردن دکمه بازگشت را شبیه سازی می‌کنیم، هر بار که دکمه بازگشت زده می‌شود مرورگر باید آخرین صفحه مشاهده شده قبلی را نمایش دهد، به عنوان مثال هنگامی که در حال مشاهده URL_4 هستیم اگر کاربر دکمه بازگشت را فشار دهد مرورگر باید URL_3 را نمایش دهد، اگر کاربر در حال مشاهده URL_3 باشد با زدن دکمه بازگشت مرورگر باید URL_2 را نمایش دهد. این عمل تا زمانی قابل انجام است که آدرسی برای نمایش وجود داشته باشد، به عبارتی پشته خالی نباشد (empty) در کد بالا هر بار که متده pop فراخوانی می‌شود عنصر بالای پشته حذف می‌شود. البته پیاده سازی عملکرد دکمه بازگشت در مرورگر کمی پیچیده‌تر است و مثال بالا یک نمونه بسیار بسیار ساده از این عمل می‌باشد.

PriorityQueue

کلاس PriorityQueue یک صف اولویت را با استفاده از ساختمان داده Min-Heap پیاده سازی می‌کند، لذا در کاربردهایی که نیاز داریم به صورت پیوسته به عنصر کمینه دستیابی داشته باشیم مفید واقع می‌شود، برای نمونه می‌توان به پیاده سازی الگوریتم مرتب سازی Sort اشاره کرد. به صورت پیش فرض این کلاس از ترتیب طبیعی (الفبایی) پیروی می‌کند مگر آنکه در سازنده آن از یک Comparator اختصاصی برای تعریف نوع خاصی از ترتیب استفاده کنیم. به دلیل استفاده از ساختار Min-Heap بیشتر عملیات این کلاس مانند poll و ... در زمانی لگاریتمی یا حتی ثابت انجام می‌شوند، به عبارت دیگر این عملیات بسیار سریع انجام می‌شوند. این کلاس از طریق مسیر `java.util.PriorityQueue` قابل دستیابی است.

- می‌تواند عضو تکراری PriorityQueue داشته باشد.

- نمی‌تواند عضو null داشته باشد.

برای ساخت شیء جدید از این کلاس می‌توانیم از یکی از سازنده‌های زیر استفاده کنیم:

```
PriorityQueue()
```

یک صف اولویت خالی ایجاد می‌کند.

```
PriorityQueue(Collection c)
```

یک صف اولویت جدید ایجاد می‌کند و اعضای کلکسیون c را به آن اضافه می‌کند.

```
PriorityQueue(int capacity)
```

یک صف اولویت جدید ایجاد می‌کند، چون کلاس `PriorityQueue` به صورت `Min Heap` پیاده سازی می‌شود برای ابتدای کار یک آرایه برای آن در نظر گرفته می‌شود که پارامتر `capacity` اندازه این آرایه را مشخص می‌کند البته در عین کار ممکن است اندازه آرایه افزایش یابد.

```
PriorityQueue(int capacity, Comparator cmp)
```

مشابه سازنده قبلی است با این تفاوت که در اینجا برای مقایسه اشیای داخل صف اولویت از یک `Comparator` اختصاصی استفاده می‌کنیم.

```
PriorityQueue(PriorityQueue other)
```

یک صف اولویت جدید ایجاد می‌کند و اعضای یک صف اولویت دیگر را به آن اضافه می‌کند.

```
PriorityQueue(SortedSet ss)
```

یک صف اولویت جدید ایجاد می‌کند و اعضای یک مجموعه مرتب را به آن اضافه می‌کند. متدهای مهم کلاس `PriorityQueue` در جدول زیر آمده‌اند :

متدها	کاربرد
<code>add(Object o)</code>	یک عنصر جدید به صف اولویت اضافه می‌کند، این متده بسیار سریع است و در زمانی لگاریتمی انجام می‌شود.
<code>clear()</code>	کل مجموعه را پاک می‌کند.
<code>comparator()</code>	اگر از یک <code>Comparator</code> اختصاصی استفاده کرده باشیم آن را بر می‌گرداند اما اگر از ترتیب طبیعی استفاده کرده باشیم خروجی این متده <code>null</code> خواهد بود.
<code>contains(Object o)</code>	بررسی می‌کند که یک شیء مورد نظر در این مجموعه قرار دارد یا خیر؟ این متده در زمانی خطی انجام می‌شود و چندان بهینه نیست.
<code>isEmpty()</code>	مشخص می‌کند که صف اولویت خالی است یا خیر.

یک Iterator از روی مجموعه بر می‌گرداند که دارای نظم خاصی نیست و فقط اعضای مجموعه را پیمایش می‌کند.	iterator()
مانند متده add عمل می‌کند.	offer(Object o)
عنصر سر (کوچکترین عضو) صف را بر می‌گرداند ولی آن را حذف نمی‌کند، اگر صف خالی باشد خروجی این متده null خواهد بود، این متده در زمان ثابت اجرا می‌شود و فوق العاده سریع است.	peek()
عنصر سر (کوچکترین عضو) صف را حذف می‌کند و آن را بر می‌گرداند، اگر صف خالی باشد خروجی این متده null خواهد بود، این متده در زمان لگاریتمی انجام می‌شود و سریع است.	poll()
یک شیء خاص را در زمان خطی از مجموعه حذف می‌کند، این متده چندان بهینه نیست.	remove(Object o)
مانند متده poll عمل می‌کند.	remove()
اندازه مجموعه را بر می‌گرداند.	size()
اعضای مجموعه را به صورت یک آرایه بر می‌گرداند.	toArray()

ابتدا با یک مثال ساده با PriorityQueue آشنا می‌شویم.

```
package myfirstprogram;

import java.util.PriorityQueue;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        PriorityQueue priorityqueue = new PriorityQueue();

        priorityqueue.add(1);
        priorityqueue.add(10);
        priorityqueue.add(2);
        priorityqueue.add(9);
        priorityqueue.add(5);

        System.out.println(priorityqueue);

        System.out.println(priorityqueue.peek());
    }
}
```

```

        System.out.println(priorityqueue);

        System.out.println(priorityqueue.poll());

        System.out.println(priorityqueue);
    }
}

```

```
[1, 5, 2, 10, 9]
```

```
1
```

```
[1, 5, 2, 10, 9]
```

```
1
```

```
[2, 5, 9, 10]
```

در کد بالا ابتدا یک PriorityQueue ایجاد می‌کنیم و سپس اعضا‌یابی را به آن اضافه می‌کنیم، سپس آن را چاپ می‌کنیم، مشاهده می‌کنیم که اعضا‌ی داخل آن دارای نظم قابل تشخیصی نیستند فقط عنصر ابتدای لیست کمینه است. سپس با استفاده از متده peek() کوچک‌ترین عنصر را به دست آورده و چاپ می‌کنیم. مشاهده می‌کنیم که استفاده از متده peek() تغییری در صفت اولویت ایجاد نمی‌کند. سپس با استفاده از متده poll() کوچک‌ترین عنصر را به دست آورده، آن را چاپ می‌کنیم و بعد از آن کل لیست را چاپ می‌کنیم. مشاهده می‌کنیم که متده poll() عنصر کمینه را از لیست حذف کرده است و در لیست جدید مجدداً کوچک‌ترین عنصر را ابتدای لیست قرار دارد. در این مثال PriorityQueue عنصر کمینه همیشه در ابتدای لیست قرار دارد.

مرتب سازی Heap Sort

در این مثال به سادگی الگوریتم مرتب سازی Heap Sort را با استفاده از PriorityQueue پیاده سازی می‌کنیم.

```

package myfirstprogram;

import java.util.LinkedList;
import java.util.PriorityQueue;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        PriorityQueue priorityqueue = new PriorityQueue();
        LinkedList result = new LinkedList();

        priorityqueue.add(1);
        priorityqueue.add(10);
        priorityqueue.add(2);
        priorityqueue.add(9);
        priorityqueue.add(5);
        priorityqueue.add(18);

        while (!priorityqueue.isEmpty())
        {

```

```

        result.add(priorityqueue.poll());
    }

    System.out.println(result);
}

[1, 2, 5, 9, 10, 18]

```

در کد بالا ابتدا یک `PriorityQueue` و یک `LinkedList` خالی ایجاد می‌کنیم، سپس اعضای نامرتبی را به `PriorityQueue` اضافه می‌کنیم. در ادامه در داخل یک حلقه `while` تا زمانی که `PriorityQueue` خالی نشده است عنصر کمینه آن را حذف می‌کنیم و به اضافه می‌کنیم لذا در هر مرحله کوچکترین عضو از صف اولویت خارج شده و به لیست پیوندی اضافه می‌شود و در نتیجه `LinkedList` لیست پیوندی در نهایت مرتب خواهد بود. الگوریتم مرتب سازی فوق بسیار سریع است.

ترتیب معکوس

اگر بخواهیم صف اولویت به صورت معکوس نگه داری شود یعنی عنصر ابتدای لیست بزرگترین آیتم آن باشد (به عبارتی به صورت `Max Heap` پیاده سازی شود) می‌توانیم از `Collections.reverseOrder()` به عنوان `Comparator` اختصاصی استفاده کنیم. در این صورت متدهای `peek()` و `poll()` عنصر بیشینه را بر می‌گردانند. مثال قبل را با این نکته بازنویسی می‌کنیم :

```

package myfirstprogram;

import java.util.Collections;
import java.util.LinkedList;
import java.util.PriorityQueue;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        PriorityQueue priorityqueue = new PriorityQueue(Collections.reverseOrder());
        LinkedList result = new LinkedList();

        priorityqueue.add(1);
        priorityqueue.add(10);
        priorityqueue.add(2);
        priorityqueue.add(9);
        priorityqueue.add(5);
        priorityqueue.add(18);

        while (!priorityqueue.isEmpty())
        {
            result.add(priorityqueue.poll());
        }
    }
}

```

```
        System.out.println(result);
    }
}
```

[18, 10, 9, 5, 2, 1]

Hashtable

کلاس `Hashtable` زیر کلاس `Dictionary` است و از `Serializable` و `Clonable` interface های `Map` و `Serializable` ارث بری می‌کند. همانند `HashMap` این کلاس نیز برای نگه داری زوج کلید-مقدار به کار می‌رود، همانطور که از نام آن پیداست این کلاس نیز همچون کلاس `HashMap` از الگوریتم درهم سازی برای نگه داری زوج‌های کلید-مقدار استفاده می‌کند. برخی از تفاوت‌های `Hashtable` و `HashMap` عبارت‌اند از :

- در Hashtable مجاز به استفاده از کلید یا مقدار null نیستیم ولی در HashMap چنین محدودیتی وجود ندارد.
 - Hashtable کلاسی سنکرون شده (synchronized) و در نتیجه Thread-safe است ولی HashMap نیست.
 - در برنامه‌ها معمولی، که جند رسماً نیستند HashMap سریع‌تر است.

در برنامه‌های پیشرفته چند ریسمانی بهتر است از ConcurrentHashMap به جای Hashtable استفاده شود. کلاس Hashtable از طریق مسیر `java.util.Hashtable` قابل دستیابی است. برای ساخت یک شیء جدید از این کلاس می‌توانیم از یکی از سازنده‌های زیر استفاده کنیم:

Hashtable()

یک **Hashtable** خالی، جدید با ظرفیت اولیه بیش از 11 درایه و $load\ factor=0.75$ ایجاد می‌کند.

Hashtable(int capacity)

مشابه سازنده قبل، است با این تفاوت که در بینجا خودمان، اندازه جدوا، اولیه را تعریف می‌کنیم.

```
Hashtable(int capacity, float loadFactor)
```

یک Hashtable خالی ایجاد می‌کند که اندازه جدول و loadFactor آن را خودمان تعیین می‌کنیم (پارامتر loadFactor مشخص می‌کند که چه زمانی اندازه Hashtable باید تغییر کند).

Hashtable(Map mp)

یک Hashtable جدید ایجاد می‌کند و اعضای mp را به آن اضافه می‌کند. متدهای مهم این کلاس در جدول زیر آمده‌اند :

کاربرد	متدها
کل مجموعه را پاک می‌کند.	<code>clear()</code>
وجود یک مقدار را در مجموعه بررسی می‌کند.	<code>contains(Object value)</code>
وجود یک کلید را در مجموعه بررسی می‌کند.	<code>containsKey(Object key)</code>
وجود یک مقدار را بررسی می‌کند.	<code>containsValue(Object value)</code>
یک Enumeration از مقدارها بر می‌گرداند.	<code>elements()</code>
یک Set از زوج‌های کلید-مقدار بر می‌گرداند.	<code>Set entrySet()</code>
تساوی این مجموعه را با مجموعه دیگری بررسی می‌کند.	<code>equals(Object o)</code>
مقدار متناظر با یک کلید را در اختیار ما قرار می‌دهد.	<code>get(Object key)</code>
خالی بودن مجموعه را بررسی می‌کند.	<code>boolean isEmpty()</code>
یک Enumeration از کلیدها بر می‌گرداند.	<code>keys()</code>
یک Set از کلیدها بر می‌گرداند.	<code>keySet()</code>
مقدار value را به کلید key نسبت می‌دهد.	<code>put(Object key, Object value)</code>
اعضای یک Map دیگر را به مجموعه اضافه می‌کند.	<code>putAll(Map mp)</code>
کلید مورد نظر (key) و مقدار متناظر با آن را از مجموعه پاک می‌کند.	<code>remove(Object key)</code>
تعداد کلیدها را بر می‌گرداند.	<code>size()</code>

کل مجموعه را به صورت یک رشته متنی بر می‌گرداند.	<code>toString()</code>
مقدارها را به صورت یک کلکسیون بر می‌گرداند.	<code>Collection values()</code>

به مثال زیر توجه کنید :

```
package myfirstprogram;

import java.util.Hashtable;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Hashtable hashtable = new Hashtable();

        hashtable.put(1, "one");
        hashtable.put(2, "two");
        hashtable.put(3, "three");

        System.out.println(hashtable);

        System.out.println(hashtable.get(2));

        System.out.println(hashtable.keySet());

        System.out.println(hashtable.values());
    }
}

{3 = three, 2 = two, 1 = one}
two
[3, 2, 1]
[three, two, one]
```

در کد بالا ابتدا یک `Hashtable` جدید ایجاد می‌کنیم و با استفاده از متدهای `put()` و `get()` مقداری اطلاعات به آن اضافه می‌کنیم. سپس کل مجموعه را چاپ می‌کنیم. در ادامه مقدار منتظر با کلید ۲ را چاپ کرده و سپس با استفاده از متدهای `keySet()` و `values()` کلیدها و با استفاده از متدهای `keySet()` و `values()` مجموعه مقادیر را چاپ می‌کنیم.

پیمایش با Enumeration

اگر برای پیمایش اعضاء `Enumeration` را ترجیح می‌دهید می‌توانید از متدهای `keys()` و `elements()` برای به دست آوردن مربوط به کلیدها و از متدهای `nextElement()` و `hasMoreElements()` برای به دست آوردن `Enumeration` مربوط به مقادیر استفاده کنید. کد زیر کلیدها و مقدارها را با استفاده از `Enumeration` پیمایش می‌کند.

```

package myfirstprogram;

import java.util.Hashtable;
import java.util.Enumeration;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Hashtable hashtable = new Hashtable();

        hashtable.put(1, "one");
        hashtable.put(2, "two");
        hashtable.put(3, "three");

        for (Enumeration key = hashtable.keys(); key.hasMoreElements();)
        {
            System.out.print(key.nextElement() + " ");
        }

        System.out.println();

        for (Enumeration value = hashtable.elements(); value.hasMoreElements();)
        {
            System.out.print(value.nextElement() + " ");
        }
    }
}

3 2 1
three two one

```

در داخل حلقه for ابتدا یک Enumeration به دست می‌آوریم، سپس با متدهای hasMoreElements() و nextElement() به دست آورده و آن را چاپ می‌کنیم.

BitSet

کلاس BitSet کلاسی است که آرایه‌ای از بیت‌ها را شبیه سازی می‌کند و می‌توان از آن در الگوریتم‌های مختلفی مانند غربال اراتستن، بزرگترین زیر مجموعه مشترک و ... استفاده کرد، به علاوه این کلاس عملگرهای بیتی همچون AND ، OR و غیره را نیز در اختیار ما قرار می‌دهد.

مهمترین ویژگی BitSet این است که مصرف حافظه آن بهینه سازی شده است و برای کاربردهایی که در آن به تعداد زیادی عنصر صفر و یک نیاز داریم مناسب است.

به صورت پیشفرض تمام خانه‌های BitSet صفر هستند. این کلاس از طریق مسیر `java.util.BitSet` قابل دستیابی است. برای

استفاده از این کلاس می‌توانیم از یکی از سازنده‌های زیر استفاده کنیم.

BitSet()

این کلاس یک BitSet خالی ایجاد می‌کند و اندازه آن در طی زمان بر حسب نیاز افزایش می‌باید.

BitSet(int nbits)

این کلاس یک BitSet با تعداد بیت‌های مورد نیاز ما ایجاد می‌کند که البته در طول زمان ممکن است اندازه آن نیز افزایش یابد. متدهای

مهم این کلاس در جدول زیر آمده‌اند :

متدها	کاربرد
<code>and(BitSet other)</code>	بیت‌های BitSet فعلی را با بیت‌های BitSet دیگری AND می‌کند.
<code>andNot(BitSet other)</code>	ابتدا بیت‌های BitSet دیگر را معکوس کرده و سپس با BitSet فعلی AND (NOT B) می‌کند. (یعنی $(A \cap \neg B)$)
<code>cardinality()</code>	تعداد بیت‌هایی که یک هستند را برمی‌گرداند.
<code>clear()</code>	کل بیت‌ها را صفر می‌کند.
<code>clear(int bitIndex)</code>	یک بیت خاص را صفر می‌کند.
<code>clear(int from,int to)</code>	تمام بیت‌های داخل یک باز خاص (از اندیس <code>from</code> تا اندیس <code>to</code>) را صفر می‌کند.
<code>equals(Object o)</code>	برابری BitSet با شیء دیگری را بررسی می‌کند.
<code>flip(int bitIndex)</code>	بیت مشخص شده در اندیس <code>bitIndex</code> را معکوس می‌کند.
<code>flip(int from,int to)</code>	بیت‌های یک بازه مشخص را معکوس می‌کند.

مشخص می‌کند که بیت یک اندیس خاص صفر (false) است یا یک (true)	get(int index)
یک زیر مجموعه از BitSet فعلی بر می‌گرداند.	BitSet(int from,int to)
اگر BitSet فعلی با دیگر حداقل یک بیت ۱ (true) مشترک داشته باشد true بر می‌گرداند.	intersects(BitSet other)
اگر BitSet هیچ بیت یکی نداشته باشد true بر می‌گرداند.	isEmpty()
اندیس سمت راست ترین یک موجود در BitSet را بر می‌گرداند.	length()
اندیس اولین بیت صفر بعد از اندیس from را بر می‌گرداند.	nextClearBit(int from)
اندیس اولین بیت یک بعد از اندیس from را بر می‌گرداند.	nextSetBit(int from)
BitSet فعلی را با دیگر OR می‌کند.	or(BitSet other)
اندیس اولین بیت صفر قبل از اندیس from را بر می‌گرداند.	previousClearBit(int from)
اندیس اولین بیت یک قبل از اندیس from را بر می‌گرداند.	previousSetBit(int from)
بیت مشخص در اندیس index را یک (true) می‌کند.	set(int index)
بیت‌های بازه from تا to را یک (true) می‌کند.	set(int from,int to)
بیت موجود در اندیس index را بر اساس مقدار value مقداری دهی می‌کند (true برابر یک و false معادل صفر).	set(int index,boolean value)
تمام بیت‌های یک بازه مشخص را به مقدار value تغییر می‌دهد.	set(int from,int to,boolean value)
تعداد کل بیت‌های BitSet را نمایش می‌دهد.	size()
معادل BitSet را به صورت آرایه‌ای از بایت‌ها بر می‌گرداند.	toByteArray()

معادل BitSet را به صورت آرایه‌ای از Long ها بر می‌گرداند.	<code>toLongArray()</code>
فعلی را با BitSet دیگری xor می‌کند.	<code>xor(BitSet other)</code>

BitSet به String تبدیل

متاسفانه کلاس BitSet ابزاری برای تبدیل یک رشته صفر و یک به BitSet را ندارد و خودمان باید کد لازم برای این کار را به صورت زیر بنویسیم:

```
static void fromString(BitSet bs, String str)
{
    for (int i = 0; i < str.length(); i++)
    {
        if (str.charAt(i) == '1')
            bs.set(i, true);
    }
}
```

در کد بالا در داخل یک حلقه for به ازای هر کاراکتری از رشته str که ۱ باشد اندیس متناظر با آن در BitSet را یک (true) می‌کنیم.

نمایش دودویی BitSet

متاسفانه کلاس BitSet ابزار مناسبی برای نمایش دودویی BitSet نیز در اختیار ما قرار نمی‌دهد و مجددًا خودمان باید کد لازم را بنویسیم، کد زیر با داشتن تعداد بیت‌ها (nbits) یک String از روی بیت‌های BitSet می‌سازد و بر می‌گرداند.

```
static String toString(BitSet bs, int nbites)
{
    String result = "";
    for (int i = 0; i < nbites; i++)
    {
        if (bs.get(i))
        {
            result += "1";
        }
        else
        {
            result += "0";
        }
    }
    return result;
}
```

منطق کد بالا بسیار ساده است، اگر بیت اندیس i در BitSet یک باشد یک ا به انتهای رشته result اضافه می‌کنیم در غیر این صورت

یک ه به انتهای رشته اضافه می‌کنیم. با کمک دو متدهایم تا مثالی از کلاس BitSet مشاهده کنیم.

```
package myfirstprogram;

import java.util.BitSet;

public class MyFirstProgram
{
    static void fromString(BitSet bs, String str)
    {
        for (int i = 0; i < str.length(); i++)
        {
            if (str.charAt(i) == '1')
                bs.set(i, true);
        }
    }
    static String toString(BitSet bs, int nbits)
    {
        String result = "";
        for (int i = 0; i < nbits; i++)
        {
            if (bs.get(i))
            {
                result += "1";
            }
            else
            {
                result += "0";
            }
        }
        return result;
    }
    public static void main(String[] args)
    {
        int nbits = 6;
        BitSet A = new BitSet(nbits);
        BitSet B = new BitSet(nbits);

        fromString(A, "110011");
        fromString(B, "100111");

        System.out.println("A :\t" + toString(A, nbits));
        System.out.println("B :\t" + toString(B, nbits));

        B.flip(0, 6);           //B=Not(B)
        System.out.println("B :\t" + toString(B, nbits));

        A.or(B);               //A=A OR B
        System.out.println("A :\t" + toString(A, nbits));

        A.and(B);              //A=A AND B
        System.out.println("A :\t" + toString(A, nbits));

        A.set(0);               //A[0]=1
    }
}
```

```

        System.out.println("A :\t" + toString(A, nbits));

        A.set(4, 6, true); //A[4]=1,A[5]=1
        System.out.println("A :\t" + toString(A, nbits));

    }
}

```

```

A : 110011
B : 100111
B : 011000
A : 111011
A : 011000
A : 111000
A : 111011

```

در کد بالا ابتدا دو BitSet خالی به نام A و B ایجاد کردیم و سپس با متدهای کمکی (fromString()) آن دو را مقدار دهی کرد و برای اطمینان آن‌ها را کمک متدهای اندیس ۰ تا ۶ B را با کمک متدهای flip() معکوس و بعد با کمک متدهای or بیتست A را با بیتست دوم OR کردیم. سایر خطوط نیز مشابه هستند و همراه آن‌ها در کد توضیحات کافی ارائه شده است.

ArrayDeque

کلاس ArrayDeque زیر کلاس AbstractCollection است و مفهوم صف دو طرفه را با استفاده از ساختمان داده آرایه حلقوی (circular array) پیاده سازی کرده است و کلاسی فوق العاده سریع و بهینه است این کلاس برای پیاده سازی صف دو طرفه به جای لیست پیوندی از آرایه استفاده می‌کند و دارای مزایای زیر است :

- اگر به عنوان صف FIFO از آن استفاده شود از LinkedList سریع‌تر خواهد بود.
- اگر به عنوان صف LIFO یا همان پشته از آن استفاده شود از کلاس Stack سریع‌تر خواهد بود.
- می‌توان به عنوان یک آرایه پویا (آرایه قابل رشد) به جای ArrayList هم از آن استفاده کرد.

در بعضی از کاربردها که نیاز به حذف گره میانی در عین پیمایش لیست داریم ممکن است LinkedList از ArrayDeque بهینه‌تر باشد.

این کلاس از طریق مسیر java.util.ArrayDeque قابل دستیابی است. برای ساخت شیء جدید از این کلاس می‌توانیم از یکی از سازنده‌های زیر استفاده کنیم :

```
ArrayDeque()
```

یک صف دو طرفه خالی ایجاد می‌کند.

<code>ArrayDeque(Collection c)</code>

یک صف دو طرفه جدید ایجاد می‌کند و اعضای کلکسیون `c` را به آن اضافه می‌کند.

<code>ArrayDeque(int numElements)</code>
--

به صورت پیش فرض `ArrayDeque` با یک آرایه با اندازه ۱۶ شروع به کار می‌کند و در صورت نیاز اندازه آن افزایش می‌یابد، با استفاده از سازنده فوق می‌توانیم آرایه اولیه‌ای با اندازه‌ای به غیر از ۱۶ به اندازه `numElements` در آن ایجاد کنیم که البته باز هم اندازه آن در صورت نیاز افزایش خواهد یافت. متدهای مهم این کلاس در جدول زیر آمدہ‌اند :

متدها	کاربرد
<code>add(Object o)</code>	یک شیء جدید به انتهای صف اضافه می‌کند.
<code>addFirst(Object o)</code>	یک شیء جدید به ابتدای صف اضافه می‌کند.
<code>addLast(Object o)</code>	یک شیء جدید به انتهای صف اضافه می‌کند.
<code>clear()</code>	کل مجموعه را پاک می‌کند.
<code>contains(Object o)</code>	وجود یا عدم وجود یک شیء را بررسی می‌کند.
<code>descendingIterator()</code>	یک <code>Iterator</code> معکوس برای پیمایش از انتها به ابتدای لیست بر می‌گرداند.
<code>element()</code>	عنصر ابتدایی صف را بر می‌گرداند ولی آن را حذف نمی‌کند.
<code>getFirst()</code>	مانند متدهای قبل عمل می‌کند.
<code>getLast()</code>	عنصر انتهایی صف را بر می‌گرداند ولی آن را حذف نمی‌کند.
<code>isEmpty()</code>	خالی بودن یا نبودن مجموعه را بررسی می‌کند.
<code>iterator()</code>	یک <code>iterator</code> برای پیمایش ابتدا تا انتهای لیست بر می‌گرداند.

یک عنصر به انتهای صف اضافه می‌کند.	<code>offer(Object o)</code>
یک عنصر به ابتدای صف اضافه می‌کند.	<code>offerFirst(Object o)</code>
یک عنصر به انتهای صف اضافه می‌کند.	<code>offerLast(Object o)</code>
عنصر ابتدای صف را بر می‌گرداند ولی آن را حذف نمی‌کند.	<code>peek()</code>
عنصر ابتدای صف را بر می‌گرداند ولی آن را حذف نمی‌کند.	<code>peekFirst()</code>
عنصر انتهای صف را بر می‌گرداند ولی آن را حذف نمی‌کند.	<code>peekLast()</code>
عنصر ابتدای صف را حذف کرده و بر می‌گرداند.	<code>poll()</code>
عنصر ابتدای صف را حذف کرده و بر می‌گرداند.	<code>pollFirst()</code>
عنصر انتهای صف را حذف کرده و بر می‌گرداند.	<code>pollLast()</code>
عنصر بالای پشته را حذف کرده و بر می‌گرداند.	<code>pop()</code>
یک عنصر به بالای پشته اضافه می‌کند.	<code>push(Object o)</code>
عنصر ابتدای صف را حذف می‌کند.	<code>remove()</code>
یک شیء را از صف حذف می‌کند.	<code>remove(Object o)</code>
عنصر ابتدای صف را حذف می‌کند	<code>removeFirst()</code>
عنصر انتهای صف را حذف می‌کند.	<code>removeLast()</code>
ابتدایی ترین نمونه از یک شیء را حذف می‌کند.	<code>removeFirstOccurrence(Object o)</code>
عنصر انتهای صف را حذف می‌کند.	<code>removeLast()</code>
انتهایی ترین نمونه از یک شیء را حذف می‌کند.	<code>removeLastOccurrence(Object o)</code>

تعداد اعضای مجموعه را بر می‌گرداند.	<code>size()</code>
اعضای مجموعه را به صورت یک آرایه بر می‌گرداند.	<code>toArray()</code>

بسیاری از متدهای فوق مشابه هم عمل می‌کنند و برنامه نویس باید طبق سلیقه خود یکی از آن‌ها را انتخاب کند.

استفاده از ArrayDeque به عنوان صف FIFO

```
package myfirstprogram;

import java.util.ArrayDeque;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ArrayDeque fifo = new ArrayDeque();

        fifo.add("A");
        fifo.add("B");
        fifo.add("C");
        fifo.add("D");

        while (!fifo.isEmpty())
        {
            System.out.println(fifo.poll());
        }
    }
}
```

A
B
C
D

درکد قبل از `ArrayDeque` به عنوان صف FIFO استفاده کردیم و مشاهده می‌کنید که شیء ای که اول وارد صف اول هم از صف خارج شده، برای خروج اشیا از صف از یک حلقه `while` استفاده کردیم و تا زمانی که صف خالی نشد (متدهای `isEmpty()` مقدار `true`) با متدهای `poll()` خارج کرده و چاپ نمودیم.

استفاده از ArrayDeque به عنوان صف LIFO

اگر قصد داشته باشیم از `ArrayDeque` به عنوان پشته (صف LIFO) استفاده کنیم بهتر است از متدهای `(push(), (pop(), (peek()`) استفاده کنیم، دقت کنید که در این حالت متدهای `peek()` عنصر بالای پشته را بر می‌گرداند ولی آن را حذف نمی‌کند و متدهای `pop()` عنصر ابتدای صف را با متدهای `poll()` خارج کرده و چاپ نمودیم.

```

package myfirstprogram;

import java.util.ArrayDeque;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ArrayDeque stack = new ArrayDeque();

        stack.push("A");
        stack.push("B");
        stack.push("C");
        stack.push("D");

        while (!stack.isEmpty())
        {
            System.out.println(stack.peek());
            System.out.println(stack.pop());
        }
    }
}

D
D
C
C
B
B
A
A

```

همانطور که مشاهده می‌کنید این بارشی‌ای که آخر وارد شده اول خارج شده است، همچنین مشاهده می‌کنید که خروجی متدهای `peek()` و `pop()` یکسان است. حلقه `while` تا زمانی خالی شدن صف (برقرار شد شرط `isEmpty()`) اجرا می‌شود.

Properties

کلاس `Properties` زیر کلاس `Hashtable` است و برای نگهداری جداولی به کار می‌رود که در آن کلید و مقدار هر دو از نوع متنی هستند، مهمترین ویژگی این کلاس این است که به ما اجازه خواندن و نوشتن فایل‌های تنظیمات (`configuration files`) و دسترسی به تنظیمات سیستمی را می‌دهد. مهم‌ترین متدهای این کلاس متدهای `load()` و `store()` برای خواندن و نوشتن تنظیمات در فایل و `loadFromXML()` و `storeToXML()` برای خواندن و نوشتن در فایل تنظیمات XML هستند. این کلاس توسط بسیاری از کلاس‌های دیگر برای ارائه اطلاعات یا تغییر تنظیمات استفاده می‌شود. این کلاس از مسیر `java.util.Properties` قابل دستیابی است. برای ساخت یک

شیء از این کلاس باید از یکی از سازنده‌های زیر استفاده کنیم :

Properties()

یک جدول خالی ایجاد می‌کند.

Properties()

از روی یک Properties دیگر یک لیست اولیه ایجاد می‌کند. متدهای مهم این کلاس در جدول زیر آمده‌اند:

متدها	کاربرد
<code>getProperty(String key)</code>	کلید را دریافت کرده و مقدار منتظر با آن را در صورت وجود بر می‌گرداند.
<code>getProperty(String key, String default)</code>	مانند متده قبلی است با این تفاوت که اگر مقدار مورد نظر پیدا نشد مقدار <code>default</code> را بر می‌گرداند.
<code>list(PrintStream out)</code>	لیست را بر روی یک استریم خروجی می‌نویسد.
<code>list(PrintWriter out)</code>	لیست را بر روی یک استریم خروجی می‌نویسد.
<code>load(InputStream in)</code>	جدول را از روی یک استریم ورودی بارگذاری می‌کند.
<code>load(Reader reader)</code>	جدول را از روی یک Reader می‌خواند.
<code>loadFromXML(InputStream in)</code>	جدول را از روی یک جریان ورودی XML می‌خواند.

لیست کلیدها را به صورت یک Enumeration بر می‌گرداند.	<code>propertyNames()</code>
یک مقدار را به یک کلید خاص نسبت می‌دهد.	<code>setProperty(String key, String value)</code>
عمل ذخیره سازی جدول را به همراه توضیحات انجام می‌دهد.	<code>store(OutputStream out, String comments)</code>
مانند متدهای قبلی عمل می‌کند با این تفاوت که بر روی یک Writer می‌نویسد.	<code>store(Writer writer, String comments)</code>
برای ذخیره جدول بر روی فایل XML همراه با توضیحات به کار می‌رود.	<code>storeToXML(OutputStream out, String comments)</code>
مانند متدهای قبلی عمل می‌کند با این تفاوت که در اینجا قادر به تعیین encoding نیز هستیم.	<code>storeToXML(OutputStream out, String comments, String encoding)</code>
کلیدها را به صورت یک Set بر می‌گرداند.	<code>stringPropertyNames()</code>

چون کلاس فوق زیر کلاس `Hashtable` است علاوه بر متدهای قبلی از متدهای کلاس `Hashtable` همچون `(get()` و `(put()` و ... نیز می‌توانیم استفاده کنیم.

چاپ اطلاعات سیستمی !

با استفاده از `(System.getProperties())` می‌توانیم اطلاعات مفیدی را در مورد سیستمی که از آن استفاده می‌کنیم به دست بیاوریم، کد زیر مسیر جاوا، ساختار سیستم عامل، مسیر `home` کاربر و نام سیستم عامل را چاپ می‌کند.

```
package myfirstprogram;
```

```

import java.util.Properties;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Properties myProperty = System.getProperties();

        System.out.println(myProperty.get("java.home"));
        System.out.println(myProperty.get("os.arch"));
        System.out.println(myProperty.get("user.home"));
        System.out.println(myProperty.get("os.name"));
    }
}

```

ذخیره در فایل

در این مثال یاد می‌گیریم که چگونه یه فایل (یا به عبارتی فایل تنظیمات) را در یک فایل ذخیره کنیم :

```

package myfirstprogram;

import java.io.*;
import java.util.Properties;

public class MyFirstProgram
{
    public static void main(String[] args) throws Exception
    {
        FileOutputStream fileoutputstream = new FileOutputStream("D:/Sample.config");
        Properties myProperty = new Properties();

        myProperty.setProperty("username", "root");
        myProperty.setProperty("password", "123456");
        myProperty.setProperty("database", "localhost");

        myProperty.store(fileoutputstream, "Database Config!");
    }
}

```

در کد بالا یک `Properties` خالی ایجاد می‌کنیم و سپس با استفاده از متدهای `setProperty()` داده‌هایی را در آن می‌نویسیم و در آخر به

سادگی با استفاده از متدهای `store()` آن را در فایل می‌نویسیم. اگر فایل مورد نظر را باز کنید به صورت زیر خواهد بود :

```

#Database Config!
#Mon May 01 08:04:02 IRDT 2017
password=123456
database=localhost
username=root

```

البته در فایل مورد نظر تاریخ نیز درج می‌شود که در این مثال آن را حذف کردیم.

خواندن از فایل

در مثال قبل یک `Properties` را در فایل نوشته‌یم، در این مثال قصد داریم همان فایل را بخوانیم و در خروجی چاپ کنیم.

```
package myfirstprogram;

import java.io.*;
import java.util.Properties;

public class MyFirstProgram
{
    public static void main(String[] args) throws Exception
    {
        FileInputStream fileInputStream = new FileInputStream("D:/Sample.config");
        Properties myProperty = new Properties();

        myProperty.load(fileInputStream);

        System.out.println("username :" + myProperty.getProperty("username"));
        System.out.println("password :" + myProperty.getProperty("password"));
        System.out.println("database :" + myProperty.getProperty("database"));
    }
}

username :root
password :123456
database :localhost
```

همانطور که مشاهده می‌کنید به سادگی با استفاده از متد `load()` محتوای فایل را در یک `Properties` خواندیم و مقادیر مورد نظر را به سادگی چاپ کردیم.

ذخیره در فایل XML

کد زیر همان `Properties` مثال قبل را در یک فایل XML ذخیره می‌کند :

```
package myfirstprogram;

import java.io.*;
import java.util.Properties;

public class MyFirstProgram
{
    public static void main(String[] args) throws Exception
    {
        FileOutputStream fileOutputStream = new FileOutputStream("D:/Sample.xml");
        Properties myProperty = new Properties();

        myProperty.setProperty("username", "root");
        myProperty.setProperty("password", "123456");
        myProperty.setProperty("database", "localhost");
```

```

        myProperty.storeToXML(fileoutputstream,"Database Config!!");
    }
}

```

مشاهده می‌کنید که با استفاده از متده است `storeToXML()` به سادگی قادر به نوشتن تنظیمات در فایل XML هستیم، اگر فایل XML مورد نظر را باز کنید مشاهده می‌کنید که محتوای آن به صورت زیر است :

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Database Config!!</comment>
<entry key="password">123456</entry>
<entry key="database">localhost</entry>
<entry key="username">root</entry>
</properties>

```

خواندن از فایل XML

در این مثال از فایل XML قبلی خوانده و نتایج را در خروجی چاپ می‌کنیم.

```

package myfirstprogram;

import java.io.*;
import java.util.Properties;

public class MyFirstProgram
{
    public static void main(String[] args) throws Exception
    {
        FileInputStream fileinputstream = new FileInputStream("D:/Sample.xml");
        Properties myProperty = new Properties();

        myProperty.loadFromXML(fileinputstream);

        System.out.println("username :" + myProperty.getProperty("username"));
        System.out.println("password :" + myProperty.getProperty("password"));
        System.out.println("database :" + myProperty.getProperty("database"));
    }
}

username :root
password :123456
database :localhost

```

همانطور که مشاهده می‌کنید با استفاده از متده است `loadFromXML()` به سادگی قادر به خواندن از فایل XML هستیم.

جنریک ها (Generics)

جنریک ها، کلاس ها، متدها یا رابطه ایی هستند که بسته به نوع داده ای که به آنها اختصاص داده می شود رفتارشان را سازگار می کنند. به عنوان مثال می توان یک متدهای جنریک تعریف کرد که هر نوع داده ای را قبول کند. همچنین می توان یک متدهای جنریک کرد که بسته به نوع دریافتی، مقادیری از انواع داده ای مانند `int`, `double` یا `String` را نشان دهد. اگر از جنریک ها استفاده نکنید باید چند متدهای مختلف را در چندین متدهای جنریک ایجاد کنید.

```
public void Show(int number)
{
    System.out.println(number);
}

public void Show(double number)
{
    System.out.println(number);
}

public void Show(String message)
{
    System.out.println(message);
}
```

با استفاده از جنریک ها می توان متدهای جنریکی ایجاد کرد که هر نوع داده ای را قبول کند.

```
public <E> void Show(E item)
{
    System.out.println(item);
}
```

متدهای جنریک را در درس های آینده توضیح خواهیم داد. حتماً این سوال را از خودتان می پرسید که چرا نباید از نوع آبجکت که هر نوع داده ای را قبول می کند استفاده کنیم؟ در آینده مشاهده می کنید که با استفاده از جنریک ها نیاز به عمل `cast` (تبديل صريح) ندارید. درباره جنریک ها در درس های بعد مطالب بیشتری توضیح می دهیم.

متدهای جنریک

اگر بخواهید چندین متدهای جنریک با عملکرد مشابه ایجاد کنید و فقط تفاوت آنها در نوع داده ای باشد که قبول می کنند (مثلاً یکی نوع `int` و دیگری نوع `double` را قبول کند) می توان از متدهای جنریک برای صرفه جویی در کدنویسی استفاده کرد. ساختار عمومی یک متدهای جنریک به شکل زیر است :

```
<type> returnType methodName(type argument1)
{
    type someVariable;
}
```

مشاهده می‌کنید که قبل از نوع برگشتی متدهای یک نوع در داخل دو علامت بزرگتر و کوچکتر آمده است (`<type>`) که همه انواع درجاوا

می‌توانند جایگزین آن شوند. برنامه زیر مثالی از نحوه استفاده از متدهای جنریک می‌باشد :

```
package myfirstprogram;

public class MyFirstProgram
{
    public static <X> void Show(X val)
    {
        System.out.println(val);
    }

    public static void main(String[] args)
    {
        int    intValue    = 5;
        double doubleValue = 10.54;
        String stringValue = "Hello";
        boolean boolValue  = true;

        Show(intValue);
        Show(doubleValue);
        Show(stringValue);
        Show(boolValue);
    }
}
```

```
5
10.54
Hello
true
```

یک متدهای جنریک ایجاد کردہ ایم که هر نوع داده‌ای را قبول کرده و مقادیر آنها را نمایش می‌دهد (خطوط ۵-۸). سپس داده‌های مختلفی

با وظایف یکسان به آن ارسال می‌کنیم. متدهای نیز نوع `X` را بسته به نوع داده‌ای که به عنوان آرگومان ارسال شده است تغییر می‌دهد. به

عنوان مثال وقتی یک داده از نوع `int` ارسال می‌کنیم، همه مکانهایی که `X` در آنها وجود دارد به `int` تبدیل می‌شوند و متدهای صورت

زیر در می‌آید :

```
public static void Show (int val)
{
    System.out.println(val);
}
```

به یک نکته در مورد استفاده از متدهای جنریک توجه کنید و آن این است که شما نمی‌توانید در داخل کدهای مربوط به متدهای محاسبات انجام دهید مثلاً دو عدد را با هم جمع کنید چون کمپایلر نمی‌تواند نوع واقعی عملوندها را تشخیص دهد، ولی به سادگی می‌توان مقادیر را در داخل متند نشان داد چون کمپایلر هر نوع داده‌ای را که توسط متند `System.out.println()` استفاده می‌شود را می‌توان تشخیص دهد.

```
public static <X> void Show(X val1, X val2)
{
    System.out.println(val1 + val2);
}
```

شما می‌توانید چندین نوع خاص را برای متند جنریک ارسال کنید، برای این کار هر نوع را به وسیله کاما از دیگری جدا کنید.

```
public static <X, Y> void Show(X val1, Y val2)
{
    System.out.println(val1);
    System.out.println(val2);
}
```

به مثال زیر که در آن دو مقدار مختلف به متند ارسال شده است توجه کنید :

```
Show(5, true);
```

مشاهده می‌کنید که `X` با نوع `int` و `Y` با نوع `boolean` جایگزین می‌شود. این نکته را نیز پادآور شویم که شما می‌توانید دو آرگومان هم نوع را هم به متند ارسال کنید :

```
Show(5, 10);
```

کلاس جنریک

تعریف یک کلاس جنریک بسیار شبیه به تعریف یک متند جنریک است. کلاس جنریک دارای یک علامت بزرگتر و کوچکتر و یک نوع پارامتر خاص می‌باشد. برنامه زیر مثالی از یک کلاس جنریک می‌باشد :

```
1 package myfirstprogram;
2
3 import java.text.MessageFormat;
4
5 class GenericClass
```

```

6
7     {
8         private T someField;
9
10        public GenericClass(T someVariable)
11        {
12            someField = someVariable;
13        }
14
15        public void SetSomeProperty(T value)
16        {
17            this.someField = value;
18        }
19
20        public T GetSomeProperty()
21        {
22            return someField;
23        }
24    }
25
26    public class MyFirstProgram
27    {
28        public static void main(String[] args)
29        {
30            GenericClass genericDouble = new GenericClass(30.50);
31            GenericClass genericString = new GenericClass("Hello World!");
32
33            System.out.println(MessageFormat.format("Double Value : {0}",
34                genericDouble.GetSomeProperty()));
35            System.out.println(MessageFormat.format("String Value : {0}",
36                genericString.GetSomeProperty()));
37        }
38    }

```

```

Double Value : 30.5
String Value : Hello World!

```

در مثال بالا یک کلاس جنریک (خطوط ۵-۲۴) که دارای یک فیلد (خط ۷)، یک خاصیت (خطوط ۱۶-۲۲) و یک سازنده (خطوط ۹-۱۲) است را ایجاد می‌کنیم. تمام مکان‌هایی که ورودی T در آن‌ها قرار دارد بعداً توسط انواعی که مد نظر شما است جایگزین می‌شوند. وقتی یک نمونه از کلاس جنریک تان ایجاد می‌کنید، یک نوع هم برای آن در نظر بگیرید (`<int>`). مانند متدهای جنریک می‌توانید چندین نوع پارامتر به کلاسهای جنریک اختصاص دهید.

```

public class GenericClass<T1, T2, T3>
{
    private T1 someField1;
    private T2 someField2;
    private T3 someField3;
}

```

چون نمی‌دانید T1، T2 و T3 از چه نوعی هستند نمی‌توانید مانند مثال زیر از آن‌ها نمونه جدید ایجاد کنید.

```
public GenericClass //Constructor
{
    someField1 = new T1();
    someField2 = new T2();
    someField3 = new T3();
}
```

کلاسهای غیر جنریک می‌توانند از کلاسهای جنریک ارث بری کنند، اما باید یک نوع برای پارامتر کلاس پایه جنریک تعریف کنید.

```
public class MyClass extends GenericClass<Integer>
{
}
```

یک کلاس جنریک هم می‌تواند از یک کلاس غیر جنریک ارث بری کند.

کلکسیون عمومی (Generic Collection)

می‌توان یک کلکسیون عمومی تعریف کرد که شامل هر نوع داده‌ای باشد. برای ایجاد یک کلکسیون عمومی از کلاس `List<T>` مربوط به فضای نامی `java.util.List` استفاده می‌شود. `List<T>` می‌تواند مجموعه‌ای از اشیاء نوع `T` باشد. در نتیجه `List<int>` مجموعه‌ای از مقادیر صحیح است. کلاس `List<T>` دارای متدهای `((), add(), remove(), removeAll()` و دیگر متدهایی است که می‌توانیم از آنها برای حذف و اضافه کردن عناصر استفاده کنیم.

```
package myfirstprogram;

import java.util.List;
import java.util.ArrayList;

class Animal
{
    public String Type;

    public Animal(String type)
    {
        this.Type = type;
    }
}
public class MyFirstProgram
{
    public static void main(String[] args)
    {
```

```
List<Animal> animals = new ArrayList<>();

animals.add(new Animal("Dog"));
animals.add(new Animal("Cat"));
animals.add(new Animal("Rat"));

for (Animal animal : animals)
{
    System.out.println(animal.Type);
}
}
```

Dog
Cat
Rat

می‌توانید از کلاس `<T>` قرار دارد استفاده کنید. `HashMap<TKey, TValue>` که در پکیج `java.util` استفاده می‌کند.

مقدار را مشخص می‌کند.

```
package myfirstprogram;

import java.util.HashMap;

class Animal
{
    public String Type;

    public Animal(String type)
    {
        this.Type = type;
    }
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        HashMap<String, Animal> animals = new HashMap<>();

        animals.put("Animal1", new Animal("Dog"));
        animals.put("Animal2", new Animal("Cat"));
        animals.put("Animal3", new Animal("Rat"));

        for (Animal animal : animals.values())
        {
            System.out.println(animal.Type);
        }
    }
}
```

Dog
Cat
Rat

یک `HashMap` تعریف کرده‌ایم که دارای کلیدهایی از نوع `String` و مقادیری از نوع کلاس `Animal` می‌باشد. برای به دست آوردن مقدار آیتم‌های `HashMap` می‌توان از متدهای `values()` که شامل همه آیتم‌های دیکشنری است استفاده کرد.

Object Initializer

به شما اجازه می‌دهند خاصیت‌ها را در داخل کلاس مقداردهی کنید. اگر به عنوان مثال چندین خاصیت داشته باشید و نخواهید که یک سازنده را جهت مقداردهی به آنها تعریف کنید، می‌توانید از `object initializer` استفاده نمایید. به عنوان

مثال به کد زیر توجه کنید :

```
package myfirstprogram;

class Sample
{
    public int    Property1;
    public String Property2;
    public boolean Property3;
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Sample sampleClass = new Sample();

        sampleClass.Property1 = 100;
        sampleClass.Property2 = "Sample";
        sampleClass.Property3 = true;
    }
}
```

همانطور که مشاهده می‌کنید، لازم است که مقادیر را تک به تک به خاصیت‌ها اختصاص دهیم. با استفاده از `object initializers` می‌توان کد را ساده‌تر کرد :

```
package myfirstprogram;

class Sample
{
    public int    Property1;
    public String Property2;
    public boolean Property3;
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Sample sampleClass = new Sample()
        {
```

```
        Property1 = 100;
        Property2 = "Hello";
        Property3 = true;
    });
}
```

مان یک خاصیت از نوع `Animal` که دارای دو خاصیت `Name` و `Age` هست را دارا می‌باشد :

```
Sample object Animal {
    String name;
    int age;
}
```

```
package myfirstprogram;

class Animal
{
    public String Name;
    public int Age;
}

class Sample
{
    public int Property1;
    public String Property2;
    public boolean Property3;
    public Animal Property4;
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Sample sampleClass = new Sample()
        {{
            Property1 = 100;
            Property2 = "Hello";
            Property3 = true;
            Property4 = new Animal()
            {{}}
```

```

        Name = "Kitty";
        Age = 3;
    }});
}
}
}
```

نوع دیگر از مقداردهنده ها collection initializers می باشند، که در کلکسیون های عمومی (generic) استفاده می شوند :

```

package myfirstprogram;

import java.util.List;
import java.util.ArrayList;
import java.text.MessageFormat;

class Person
{
    public String FirstName;

    public Person(String f)
    {
        this.FirstName = f;
    }
}

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        List<Person> people = new ArrayList<Person>()
        {
            add(new Person("John"));
            add(new Person("Jenny"));
            add(new Person("Joe"));
        };

        for (Person person : people)
        {
            System.out.println(MessageFormat.format("{0}", person.FirstName));
        }
    }
}
```

```

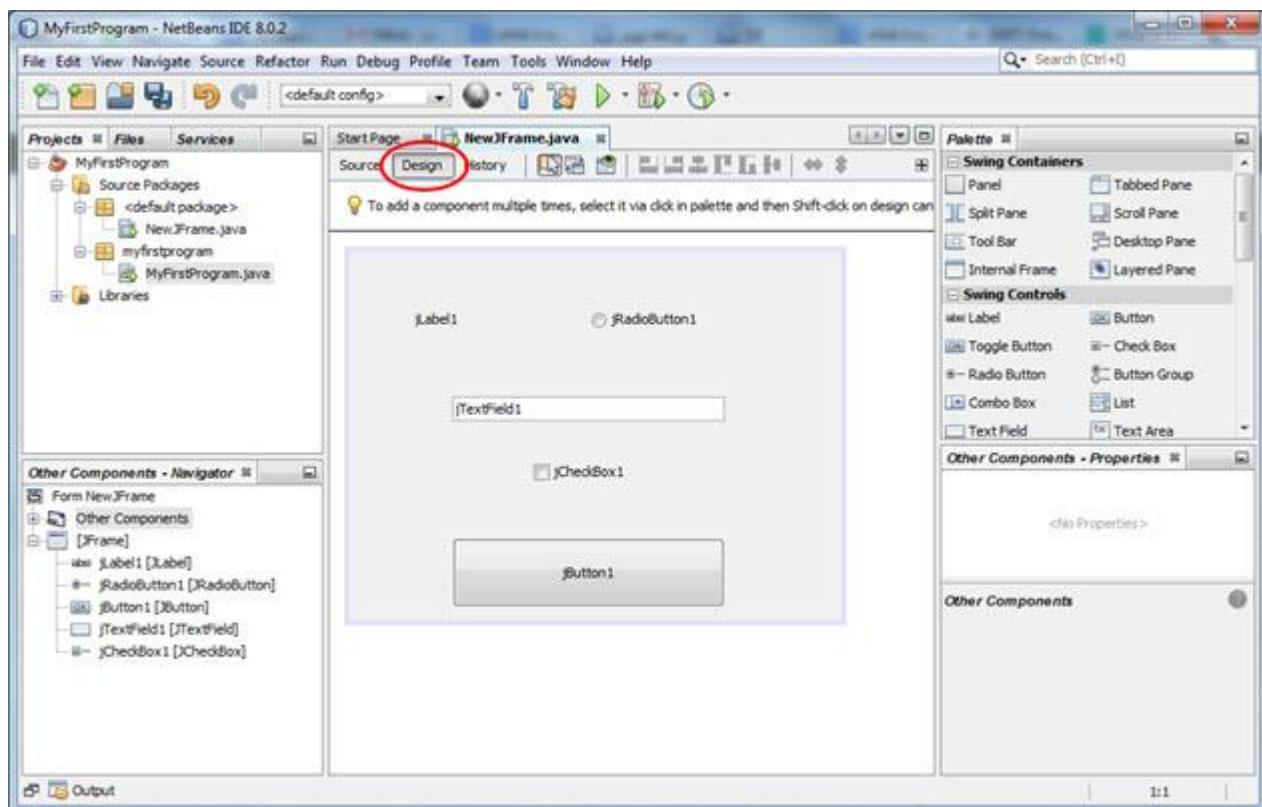
John
Jenny
Joe
```

SWING

برنامه نویسی ویژوال

رابط گرافیکی کاربر یا (GUI) به کاربر اجازه می‌دهد که با استفاده از اجزای بصری مختلف با برنامه ارتباط برقرار کند. در روزهای اولیه دنیای کامپیوتر برنامه‌ها مبتنی بر متن بودند، بدین معنی که شما باید دستورات متعددی برای ایجاد یک برنامه مفید تایپ می‌کردید و این کار مستلزم حفظ کردن یک لیست طولانی از دستورات بود. برنامه‌های نرم افزاری امروزی دارای رابط گرافیکی هستند. این رابط گرافیکی تقریباً در همه برنامه‌هایی که امروزه با آن‌ها سرو کار دارید به چشم می‌خورد. یک رابط گرافیکی حرفه‌ای باید جذاب و ساده باشد. ایجاد یک برنامه با رابط کاربری قبلًا یک کار سخت و کسل کننده بود. مثلاً برای ایجاد یک پنجره ساده که یک متن را نمایش دهد نیاز بود که تعداد زیادی کد تایپ شود. اما با ورود برنامه نویسی ویژوال این کار راحت شد. برنامه نویسی ویژوال ایجاد برنامه‌های گرافیکی را راحت کرد، به طوری که شما می‌توانید محیط برنامه خود را با کشیدن کنترل‌های لازم از جعبه ابزار به نوعی "نقاشی" کنید. کنترل‌ها اجزای بصری هستند که GUI یا رابط گرافیکی را تشکیل می‌دهند. نمونه‌ای از کنترل‌ها عبارت‌اند از buttons، NetBeans نیز محیطی را برای ایجاد و طراحی فرم‌ها به صورت زیر برای .radio buttons و check boxes، labels، text boxes

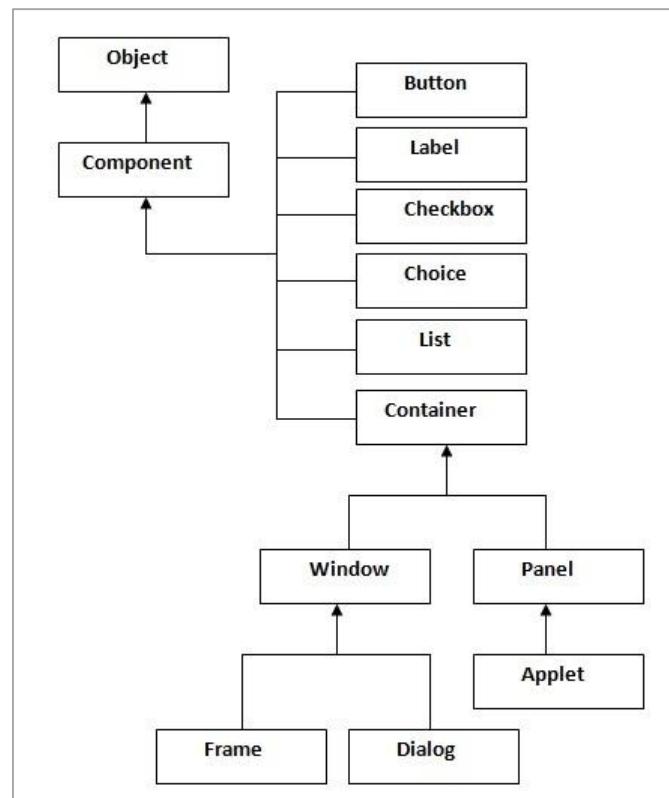
شما فراهم می‌آورد :



شکل بالا حالت طراحی (Design) در NetBeans را نشان می‌دهد. در این شکل فرم ویندوزی و کنترل‌هایی که بر روی آن کشیده شده‌اند نشان داده شده است. در حالت طراحی، شما می‌توانید چگونگی به نظر رسیدن فرم را در حین اجرای برنامه مشاهده کنید. کدهایی که باعث ایجاد و مقداردهی به کنترل‌ها می‌شوند از دید کاربر مخفی هستند، بنابراین شما می‌توانید بر روی کارکرد برنامه بیشتر تمرکز کنید. همچنین می‌توانید از ابزارهای NetBeans مانند چپ چین یا راست چین کردن، تغییر اندازه و برای طراحی کنترل‌ها استفاده کرد.

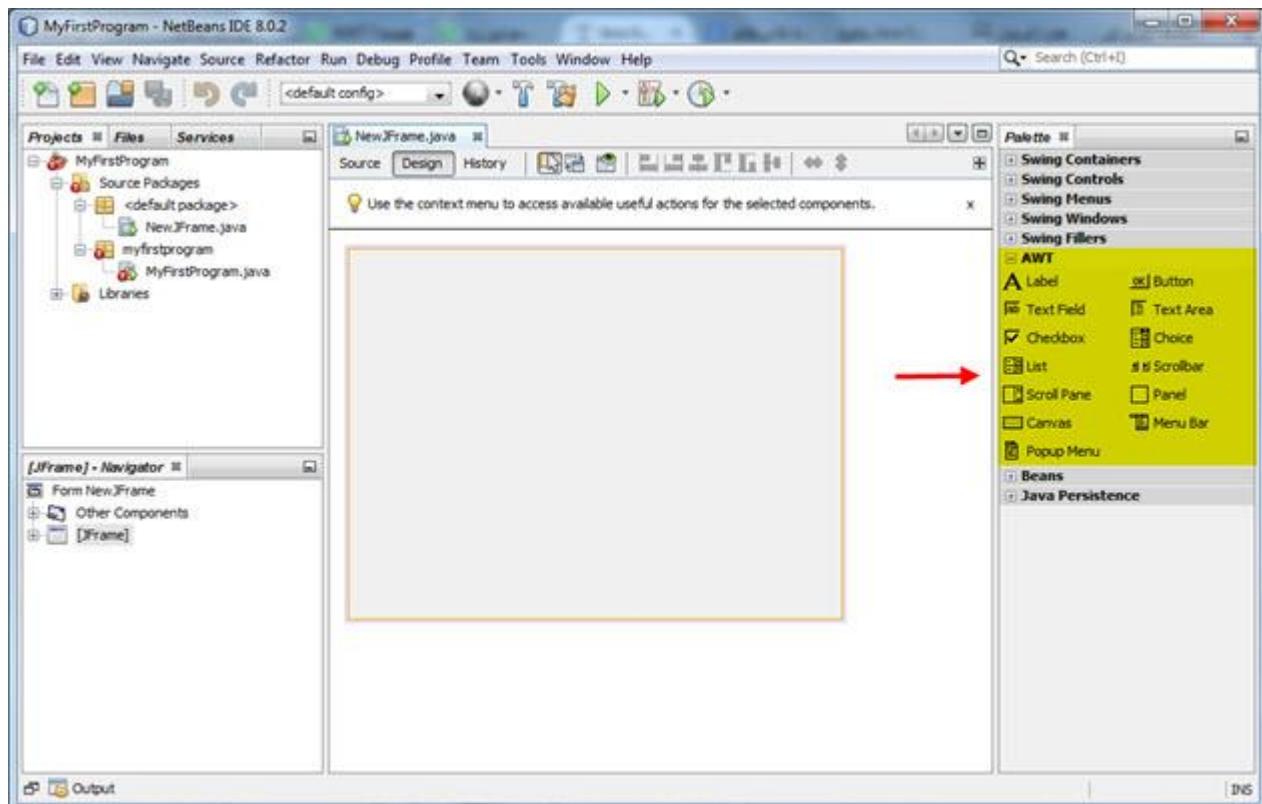
چیست؟ AWT

IAWT یا جعبه ابزار انتزاعی، مجموعه‌ای از کلاس‌های است، که توسط برنامه نویسان جاوا برای ایجاد اشیاء رابط گرافیکی کاربر (GUI) مانند دکمه‌های، اسکرول بارها و پنجره‌ها به کار می‌روند. AWT قسمتی از کلاس‌های پایه‌ای جاوا (JFC) است که توسط شرکت Sun Microsystems برنامه نویسی شده‌اند. JFC مجموعه‌ای جامع از کتابخانه‌های کلاس GUI است که با استفاده از آن‌ها به راحتی می‌توانید رابط کاربری برنامه را ایجاد کرده و توسعه دهید. پکیج `java.awt` دارای کلاس‌هایی برای کار با رابط گرافیکی است که در زیر نحوه ارث بری آن‌ها نشان داده شده است :



همانطور که در شکل بالا مشاهده می‌کنید در پکیج AWT، کلاس‌های Label، Button و ... از کلاس Component ارث بری می‌کنند و اشیاء

ساخته شده از این کلاس‌ها کنترل‌های بصری AWT را که در شکل زیر مشاهده می‌کنید به وجود می‌آورند :



کنترل‌های AWT یک مجموعه از اولین کنترل‌ها برای پلت فرم جاوا هستند که از آن‌ها برای ایجاد برنامه‌های جاوای قابل استفاده در محیط‌های Windows و Linux استفاده می‌شود. این کامپوننتهای خیلی کند و غیر قابل اطمینان بوده و برای تمام پلت فرم‌های جاوا نیز نمی‌توانند مورد استفاده قرار گیرند. با توجه به مشکلات کنترل‌های AWT در ایجاد برنامه‌های کاربردی، شرکت Sun با همکاری Netscape و سایر شرکت‌های دیگر مجموعه‌ای ای دیگر از کامپوننتهای کتابخانه‌ها را بنام Java Foundation Class با نام اختصاری JFC بوجود آورند که کنترل‌های Swing نیز بخشی از آن‌ها هستند. از آنجاییکه در این سری آموزشی می‌خواهیم کنترل‌ها و در کل کتابخانه Swing را آموزش دهیم به همین توضیحات در مورد AWT بسنده می‌کنیم و در درس آینده شما را با Swing آشنا می‌کنیم.

SWING چیست؟

کلمه Swing از یک موزیک محبوب و معروف در آمریکا که در طی سالهای ۱۹۴۵ – ۱۹۳۰ پخش می‌شد است، الهام گرفته شده است. در این زمان هنوز جاوا وجود نیامده بود. کامپوننتهای AWT یک مجموعه از اولین کامپوننتهای برای پلت فرم جاوا بودند که از آن‌ها برای ایجاد

برنامه‌های جاوا، قابل استفاده در محیط‌های Windows و Linux استفاده می‌شده است. این کامپوننتهای خیلی کند و غیر قابل اطمینان بودند و برای تمام پلت فرم‌های جاوا نیز نمی‌توانست مورد استفاده قرار گیرند. با توجه به مشکلات کامپوننتهای AWT در ایجاد برنامه‌های کاربردی، شرکت Sun با همکاری Netscape و سایر شرکت‌های دیگر از کامپوننتهای و کتابخانه‌ها را بنام Java Foundation Class با نام اختصاری JFC بوجود آورند که کنترل‌های Swing نیز بخشی از آن‌ها بودند. کنترل‌های Swing برخلاف کنترل‌های AWT، هماهنگی کاملی با سایر فریم ورکهای جاوا داشته و بر روی تمام پلت فرم‌ها قابل استفاده می‌باشد. این کنترل‌های علاوه بر خواص کنترل‌های AWT دارای خواص جدیدی‌تری نیز بوده و معایب کنترل‌های AWT را در آن‌ها بر طرف شده است. به عنوان مثال در کامپوننت Button از مجموعه کامپوننتهای AWT نمی‌توان از عکس استفاده کرد ولی این قابلیت در کنترل JButton از کتابخانه Swing گنجانده شده است.

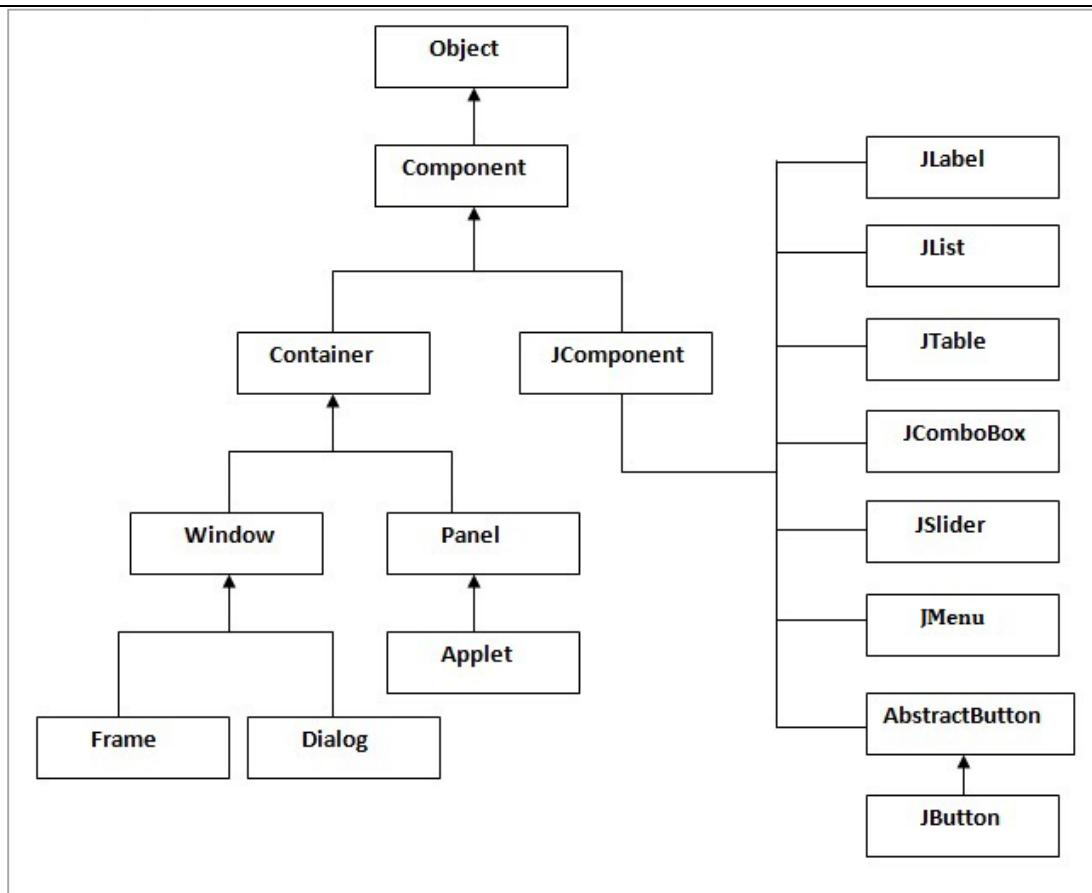
مهم‌ترین وجه تمایز بین کنترل‌های Swing و AWT این است که کامپوننتهای Swing همگی با حرف J شروع می‌شوند. به عنوان مثال کنترل JButton از کنترل‌های Swing معادل کنترل Button از AWT است، و یا کنترل JComboBox از کنترل Choice از مجموعه کنترل‌های AWT است. جدول زیر لیست کنترل‌های AWT و معادل آن‌ها را در کامپوننهای Swing نشان می‌دهد.

	SWING	AWT
JButton		Button
JPanel		Canvas
JCheckBox		CheckBox
JRadioButton in ButtonGroup		CheckBox in CheckBoxGroup
JComboBox		Choice
JComponent		Component
JPanel		Container
JLabel		Label
JList		List
JMenu		Menu

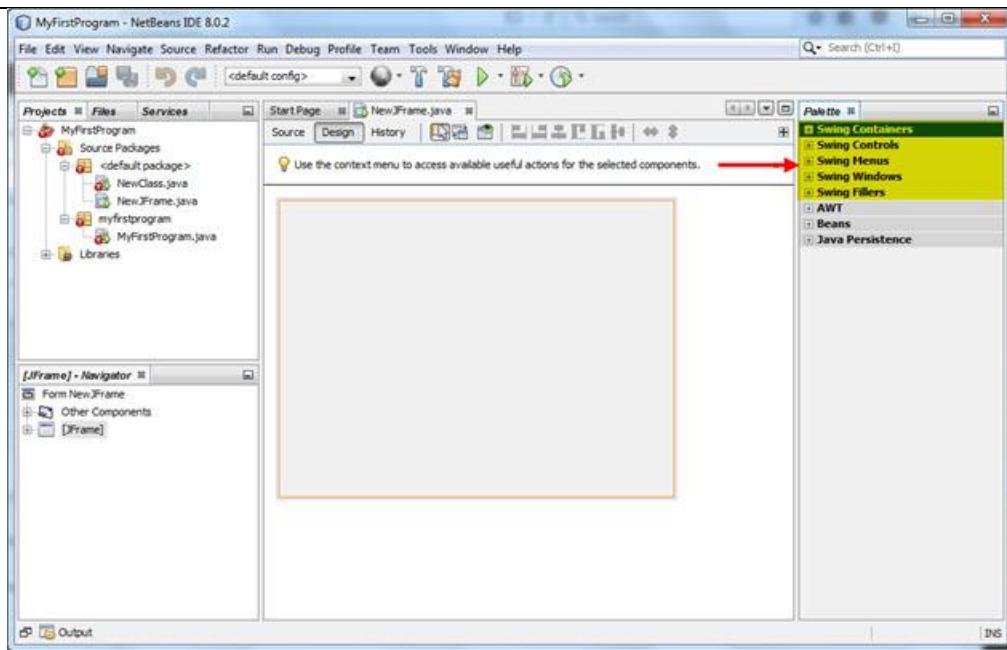
JMenuBar	MenuBar
JMenuItem	MenuItem
JPanel	Panel
JPopupMenu	PopupMenu
JScrollBar	Scrollbar
JScrollPane	ScrollPane
JTextArea	TextArea
JTextField	TextField
JApplet	Applet
JDialog	Dialog
JFileChooser	FileDialog
JFrame	Frame
JWindow	Window

علاوه بر کنترل‌های فوق، Swing دارای کنترل‌های جدید دیگری نیز هست که هیچ جایگزینی برای آن‌ها در کنترل‌های AWT وجود ندارد.

پکیج `java.swing` دارای کلاس‌هایی برای کار با رابط گرافیکی است که در زیر نحوه ارث بری آن‌ها نشان داده شده است:



همانطور که در شکل بالا مشاهده می‌کنید در پکیج swing، کلاس‌های **JLabel**, **JButton**، کلاس‌های **JComponent** و ... از کلاس **Component** و این کلاس هم از کلاس **Object** ارث بری می‌کند و اشیاء ساخته شده از این کلاس‌ها، کنترل‌های بصری Swing را که در شکل زیر مشاهده می‌کنید به وجود می‌آورند :



حال که مختصری با سوینگ آشنا شدید در درس‌های آینده با کنترل‌های آن و نحوه استفاده از هر یک آشنا می‌شوید.

ایجاد یک برنامه Swing ساده

وقت آن رسیده است که کار با کنترل‌های Swing را تجربه کنید. به این نکته توجه کنید که این درس به شما نحوه ایجاد یک برنامه ویندوزی که در آن از یک اداره کننده رویداد (event-handling) استفاده شده است، را نشان می‌دهد. من هر مرحله را به صورت گام به گام انجام داده و به طور مختصر در مورد هر کدام توضیح می‌دهم. مفاهیم فریم، کنترل، کنترل کننده رویداد (event-handling) و برخی قسمت‌های Swing در طراحی یک برنامه ویندوزی مورد استفاده قرار می‌گیرد و در مورد هر کدام از آن‌ها در درس‌های مربوطه‌شان بحث خواهد شد. قبل از پرداختن به ادامه درس لینک زیر را مشاهده کنید تا نحوه پیکربندی JDK برای اجرای فایل‌های java یاد بگیرید و دلیل استفاده از نرم افزار NetBeans در آموزش‌ها را بفهمید.

ساخت یک برنامه در جاوا

حال می‌خواهیم یک برنامه ایجاد کنیم که در این برنامه وقتی بر روی یک دکمه کلیک شد متن داخل آن تغییر کند. یک ویرایشگر متن مانند NotePad++ باز کرده و کدهای زیر را در داخل آن بنویسید :

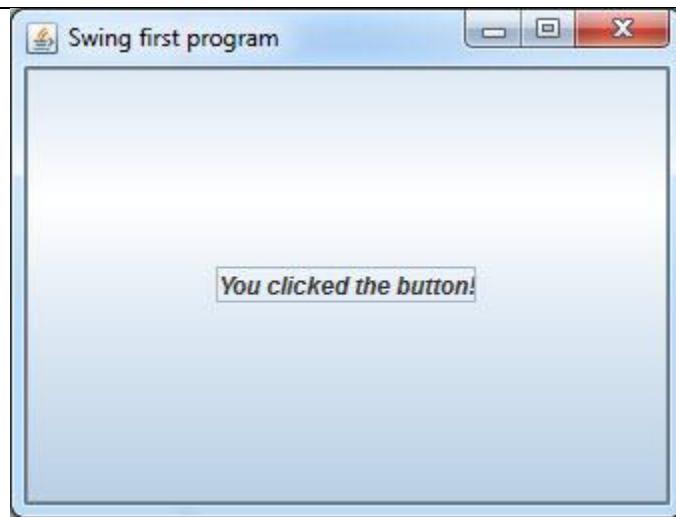
```

1 package myfirstprogram;
2
3 import javax.swing.*;
4 import java.awt.event.*;

```

```
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10        JFrame frame = new JFrame("Swing first program");
11
12        JButton button = new JButton("Click ME!");
13
14        button.addMouseListener(new MouseListener())
15        {
16            @Override
17            public void mouseClicked(MouseEvent e)
18            {
19                button.setText("You clicked the button!");
20            }
21
22            @Override
23            public void mousePressed(MouseEvent e)
24            {
25            }
26
27            @Override
28            public void mouseReleased(MouseEvent e)
29            {
30            }
31
32            @Override
33            public void mouseEntered(MouseEvent e)
34            {
35            }
36
37            @Override
38            public void mouseExited(MouseEvent e)
39            {
40            }
41        });
42
43        frame.add(button);
44        frame.setSize(337, 255);
45        frame.setVisible(true);
46    }
47
48
49
50
51 }
```

با اجرای کد بالا پنجره‌ای به صورت زیر نمایش داده می‌شود که با کلیک بر روی آن متن داخل دکمه تغییر می‌کند :



برنامه بالا یکی از ساده‌ترین برنامه‌هایی بود که جهت آشنایی شما به Swing آموزش دادیم. در این کتاب قصد داریم که Swing را از طریق کدنویسی نه از طریق برنامه‌هایی مانند NetBeans به شما آموزش دهیم تا شما بیشتر با مفاهیم کدها آشنا شوید.

کلاس JOptionPane

JOptionPane کلاسی از جاواست است که از آن برای نشان دادن یک پیغام فوری، اطلاعات و یا یک هشدار به کاربران استفاده می‌شود. برای نشان دادن یک پیغام به راحتی می‌توان از متد (showMessageDialog) کلاس JOptionPane استفاده نمایید. ساده‌ترین حالت متد (showMessageDialog) این است که یک رشته متنی را به عنوان آرگومان قبول می‌کند و آن را نمایش می‌دهد. یک فایل با پسوند java توسط Notepad++ ایجاد کرده و کدهای زیر را در داخل آن نوشته و با نام MyFirstProgram ذخیره کنید :

```
package myfirstprogram;

import javax.swing.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        JOptionPane.showMessageDialog(null, "Hello World !");
    }
}
```

بین دو پرانتز مقابل آن اول کلمه null را داریم که کلمه‌ای کلیدی در جاواست و اینجا فقط به این معناست که جعبه پیام به هیچ چیز دیگری در برنامه وابسته نیست. برنامه را اجرا کرده و نتیجه را مشاهده نمایید :



شما همچنین می‌توانید به راحتی و با استفاده از یکی دیگر از سربارگذاریهای متدها (`showMessageDialog`) یک عنوان و یک آیکون برای جعبه پیامتان بگذارید.

```
JOptionPane.showMessageDialog (null, "Hello World !", "A Message", JOptionPane.WARNING_MESSAGE);
```

نتیجه اجرای کد بالا به صورت زیر است.



همانطور که در بالا مشاهده کردید، می‌توان به جعبه پیام برای نشان دادن معنی و مفهوم آن یک آیکون اضافه کرد. در جدول زیر انواع آیکون‌ها و کاربرد آن‌ها در جعبه پیام نشان داده شده است :

استفاده	عضو	آیکون
برای نشان دادن اطلاعات به کاربر	INFORMATION_MESSAGE	
برای نشان دادن یک پیغام خطا	ERROR_MESSAGE	

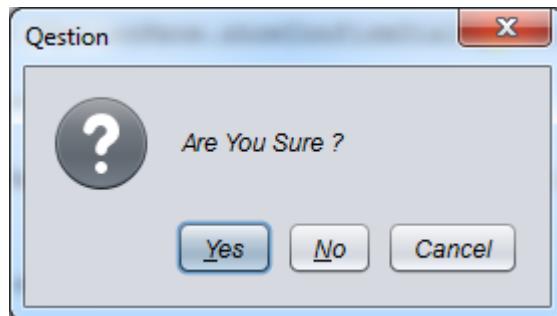
برای نشان دادن یک هشدار	WARNING_MESSAGE	
برای سؤال کردن از کاربر	QUESTION_MESSAGE	

اگر بخواهید که جعبه متن هیچ گونه آیکونی نداشته باشد می‌توانید از متده استفاده کنید. برای استفاده از این

متده می‌توانید به صورت زیر عمل نمایید :

```
JOptionPane.showConfirmDialog(null, "Hello World !", "A Message", JOptionPane.YES_NO_CANCEL_OPTION);
```

خروجی کد بالا به صورت زیر است :



جدول زیر دکمه‌هایی را که می‌توان با این متده نمایش داد را نشان می‌دهد :

دکمه‌هایی که نمایش می‌دهد	عضو
OK_CANCEL_OPTION	OK, Cancel
YES_NO_OPTION	Yes, No
YES_NO_CANCEL_OPTION	Yes, No, Cancel

متده showConfirmDialog() یک مقدار را از int یا عدد صحیح بر می‌گرداند. تشخیص اینکه چه دکمه‌ای توسط شما در جعبه متن

فشار داده می‌شود مفید است. به عنوان مثال با کلیک بر روی دکمه‌های NO، Yes و Cancel این متده به ترتیب مقادیر ۰، ۱ و ۲ را بر

می‌گرداند.

```

int result =
    JOptionPane.showConfirmDialog(null, "Are You Sure ?", "Question", JOptionPane.YES_NO_CANCEL_OPTION);

if (result == 0)
{
    //You pressed the Yes button
}

if (result == 1)
{
    //You pressed the No button
}

if (result == 2)
{
    //You pressed the Cancel button
}

```

برای تشخیص دکمه فشرده شده به جای اعداد ۱ و ۲ و ۳ می‌توان از مقادیری که در کد زیر آمده است هم استفاده کنید.

```

if (result == JOptionPane.YES_OPTION)
{
    //You pressed the Yes button
}

if (result == JOptionPane.NO_OPTION)
{
    //You pressed the No button
}

if (result == JOptionPane.CANCEL_OPTION)
{
    //You pressed the Cancel button
}

```

یکی دیگر از متدهای کلاس JOptionPane می‌باشد که از آن برای دریافت ورودی از کاربر استفاده می‌شود.

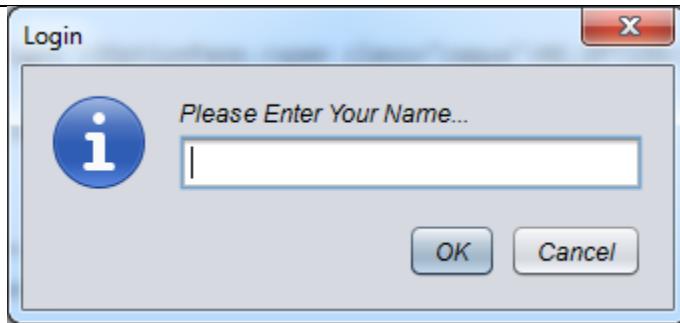
خروجی این متد از نوع رشته است.

```

String Name = JOptionPane.showInputDialog(null, "Please Enter Your Name...", "Login",
JOptionPane.INFORMATION_MESSAGE);

```

خروجی کد بالا به صورت زیر است.



کنترل کننده رویداد

از مکانیزم کنترل کننده رویداد (Event Handler) برای کنترل رویدادها که در هنگام اجرای برنامه به وقوع می‌پیوندند استفاده می‌کند. رویدادها رفتارهایی یا اتفاقاتی هستند که هنگام اجرای برنامه به وقوع می‌پیوندند. کنترل رویداد فرایند نظارت بر وقوع یک رویداد مشخص می‌باشد. فرم‌های ویندوزی از کنترل کننده رویداد برای اضافه کردن یک قابلیت و پاسخ به کاربر استفاده می‌کنند. بدون کنترل کننده رویداد، فرم‌ها و رابط گرافیکی تا حد زیادی بدون استفاده هستند. کنترل رویداد در Swing دارای سه جزء است :

- رویداد (Event): یک رویداد تغییر حالت یک شیء (مثلاً یک جعبه متن) است.
- منبع رویداد (Event Source): شیئی است که رویداد را تولید می‌کند (مثلاً دکمه‌ای که رویداد کلیک را تولید می‌کند).
- شنونده رویداد (Event Listener): اینترفیسی است که منتظر وقوع رویداد است.

مکانیسم کنترل رویداد در جاوا به این صورت است :

۱. یک کلاس تولید می‌کنیم که شنونده رویداد مربوطه را پیاده سازی می‌کند .
۲. چون این کلاس شنونده رویداد که یک رابط است را پیاده سازی کرده است، پس باید کدهای بدنه متدهای این رابط که همان کدهایی هستند که ما می‌خواهیم بعد از وقوع رویداد اجرا شوند را تأمین کند .
۳. کلاسی که شنونده را پیاده سازی می‌کند را به کنترل وصل می‌کنیم .

در جدول زیر کلاس‌های رویداد، اینترفیس‌ها و متدهای این اینترفیس‌ها که در کنترل رویدادها به کار برده می‌شوند. این کلاس‌ها و اینترفیس‌ها در پکیج‌های `java.awt.event`, `java.awt`, `java.util` قرار دارند:

متدهای اضافه کننده	کلاس رویداد	متدهای شنونده رویداد	شنونده رویداد
addActionListener	ActionEvent	actionPerformed	ActionListener
addAdjustmentListener	AdjustmentEvent	adjustmentValueChanged	AdjustmentListener
addComponentListener	ComponentEvent	componentHidden	ComponentListener
		componentMoved	
		componentResized	
		componentShown	
addContainerListener	ContainerEvent	componentAdded	ContainerListener
		componentRemoved	
addFocusListener	FocusEvent	focusGained	FocusListener
		focusLost	
addItemListener	ItemEvent	itemStateChanged	ItemListener
addKeyListener	KeyEvent	keyPressed	KeyListener
		keyReleased	
		keyTyped	
addMouseListener	MouseEvent	mouseClicked	MouseListener
		mouseEntered	
		mouseExited	
		mousePressed	
		mouseReleased	
addMouseMotionListener	MouseEvent	mouseDragged	MouseMotionListener

		mouseMoved	
addTextListener	TextEvent	textValueChanged	TextListener
addWindowListener	WindowEvent	windowActivated	WindowListener
		windowClosed	
		windowClosing	
		windowDeactivated	
		windowDeiconified	
		windowIconified	
		windowOpened	

به ستون آخر سمت چپ جدول بالا توجه کنید. همانطور که در ابتدای درس اشاره کردیم بعد از پیاده سازی شنونده توسعه یک کلاس باید آن کلاس را به کنترل مورد نظرمان وصل کنیم این کار را با متدهایی انجام می‌دهیم که اسم آن‌ها با کلمه add شروع می‌شود که در جدول بالا و در ستون (متدهای کنترل) نام این متدها ذکر شده است. به علت کمبود فضا در جدول بالا در جداول زیر کاربرد کلاس‌های رویداد و متدها (ستون دوم و سوم) آمده است :

متدها	توضیح
	وقتی یک رویداد اکشن اتفاق می‌افتد فراخوانی می‌شود
	actionPerformed
	وقتی که یک کنترل مخفی شود فراخوانی می‌شود
	adjustmentValueChanged
	وقتی که یک کنترل حرکت کند فراخوانی می‌شود
	componentHidden
	وقتی که اندازه یک کنترل تغییر کند فراخوانی می‌شود
	componentMoved
	وقتی که اندازه یک کنترل تغییر کند فراخوانی می‌شود
	componentResized
	وقتی که یک کنترل نمایش داده شود فراخوانی می‌شود
	componentShown

وقتی که یک کنترل اضافه شود فراخوانی می‌شود	componentAdded
وقتی که یک کنترل حذف شود فراخوانی می‌شود	componentRemoved
وقتی کنترل فوکوس می‌گیرد فراخوانی می‌شود	focusGained
وقتی کنترل فوکوس از دست می‌دهد فراخوانی می‌شود	focusLost
وقتی آیتم انتخاب شده تغییر می‌کند فراخوانی می‌شود	itemStateChanged
وقتی کاربر یک کلید را فشار می‌دهد فراخوانی می‌شود	keyPressed
وقتی کاربر کلید را آزاد می‌کند فراخوانی می‌شود	keyReleased
وقتی کاربر کلید را فشرده و سپس رها می‌کند فراخوانی می‌شود	keyTyped
وقتی کاربر با ماوس کلیک می‌کند فراخوانی می‌شود	mouseClicked
وقتی کاربر با ماوس روی یک کنترل می‌کند رود فراخوانی می‌شود	mouseEntered
وقتی ماوس از محدود یک کنترل خارج می‌شود فراخوانی می‌شود	mouseExited
وقتی کاربر دکمه ماوس را فشار می‌دهد فراخوانی می‌شود	mousePressed
وقتی کاربر دکمه ماوس را رها می‌کند فراخوانی می‌شود	mouseReleased
وقتی با ماوس یک کنترل را Drag می‌کنید فراخوانی می‌شود	mouseDragged
وقتی نشانگر ماوس جابجا شود فراخوانی می‌شود	mouseMoved
وقتی مقدار متن کنترل تغییر کند فراخوانی می‌شود	textValueChanged
هنگام فعال شدن پنجره فراخوانی می‌شود.	windowActivated

هنگام بسته شدن پنجره فراخوانی می‌شود	windowClosed
وقتی پنجره در حال بسته شدن است فراخوانی می‌شود	windowClosing
وقتی پنجره غیر فعال می‌شود فراخوانی می‌شود	windowDeactivated
وقتی پنجره از حال آیکون به حالت عادی تغییر می‌کند فراخوانی می‌شود	windowDeiconified
وقتی پنجره مینیمایز می‌شود فراخوانی می‌شود	windowIconified
وقتی پنجره باز می‌شود فراخوانی می‌شود	windowOpened

کلاس رویداد	توضیحات
ActionEvent	وقتی تولید می‌شود که بر روی یک دکمه کلیک، یک آیتم انتخاب و یا بر روی لیست آیتم کلیک شود.
MouseEvent	وقتی تولید می‌شود که ماوس Drag یا کلیک شود و یا از یک کنترل خارج و یا وارد یک کنترل شود.
KeyEvent	وقتی تولید می‌شود که یک دکمه از صفحه کلید توسط کاربر فشرده شود.
ItemEvent	وقتی تولید می‌شود که item check-box یا list کلیک شود.
TextEvent	وقتی تولید می‌شود که مقدار textarea یا textfield تغییر کند.
MouseWheelEvent	وقتی تولید می‌شود که دکمه وسط ماوس حرکت داده شود.
WindowEvent	وقتی تولید می‌شود که پنجره فعال، غیرفعال، بسته و ... شود.
ComponentEvent	وقتی تولید می‌شود که یک کنترل مخفی شود، حرکت کند، اندازه‌اش تغییر کند.
ContainerEvent	وقتی تولید می‌شود که یک کنترل به پنجره اضافه و یا از آن حذف شود.
AdjustmentEvent	وقتی تولید می‌شود که اسکرول تغییر کند.

وقتی تولید می‌شود که یک کنترل فوکوس گرفته و یا فوکوس خود را از دست بدهد.	FocusEvent
--	------------

برای درک بهتر کنترل رویداد (Event handling) به یک مثال توجه کنید. فرض کنید که می‌خواهیم وقتی بر روی یک دکمه کلیک شد یک پیام نمایش داده شود. برنامه NotePad را باز کرده و کدهای زیر را در داخل آن نوشته و سپس با نام MyFirstProgram ذخیره کنید:

```

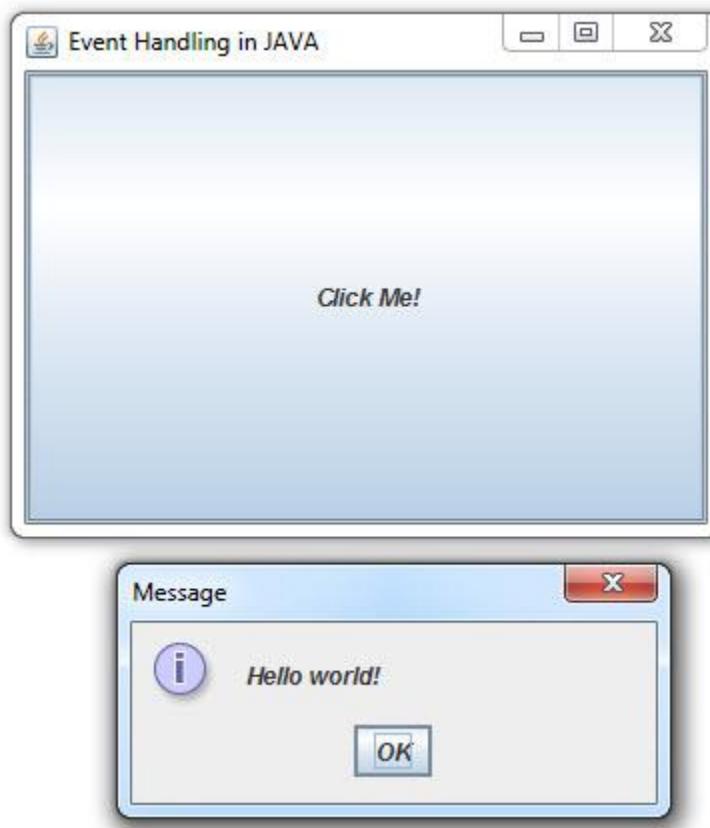
1 package myfirstprogram;
2
3 import javax.swing.*;
4 import java.awt.event.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10        JFrame frame1 = new JFrame("Event Handling in JAVA");
11        frame1.setSize(356 , 263);
12        frame1.setVisible(true);
13
14        JButton button1 = new JButton("Click Me!");
15        frame1.add(button1);
16
17        class ListenToClick implements ActionListener           //Step 1
18        {
19            @Override
20            public void actionPerformed(ActionEvent e)
21            {
22                JOptionPane.showMessageDialog(null, "Hello world!"); //Step 2
23            }
24        }
25
26        button1.addActionListener(new ListenToClick());          //Step 3
27    }
28 }
```

در مثال بالا چون ما قرار است از کنترل‌های Swing استفاده و همچنین رویدادها را کنترل کنیم ابتدا کلاس‌ها و پکیج‌های مربوطه را در خطوط ۱ و ۲ اضافه می‌کنیم. سپس در خطوط ۱۵-۱۰ یک فریم ایجاد و یک دکمه به آن اضافه می‌کنیم. در درس‌های بعد در مورد این دو کنترل و همچنین این کدها بیشتر توضیح می‌دهیم. حال نوبت به سه مرحله‌ای می‌رسد که در ابتدای درس توضیح دادیم:

۱. در خطوط ۱۷-۲۴ یک کلاس به نام ListenToClick ایجاد می‌کنیم و به وسیله آن شنونده ActionListener را پیاده سازی می‌کنیم.

۲. شنونده ActionListener دارای یک متده به نام actionPerformed() است. این متده باید توسط کلاس ListenToClick پیاده سازی (Override) شود. پس باید کدهایی که می خواهیم بعد از کلیک بر روی دکمه یعنی رویدادمان اجرا شوند را در داخل بدنه متده actionPerformed() مینویسیم (خط ۲۲).
۳. در خط ۲۶ با استفاده از متده addActionListener() کلاس ListenToClick را به دکمه وصل می کنیم.

حال برنامه را اجرا کرده و با کلیک بر روی دکمه نتیجه را مشاهده نمایید :



خطوهای ۲۶-۲۷ را به دو صورت زیر هم می توان نوشت :

```
ActionListener ListenToClick = new ActionListener() //Step 1
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        JOptionPane.showMessageDialog(null, "Hello world!"); //Step 2
    }
};
```

```
button1.addActionListener(ListenToClick); //Step 3
```

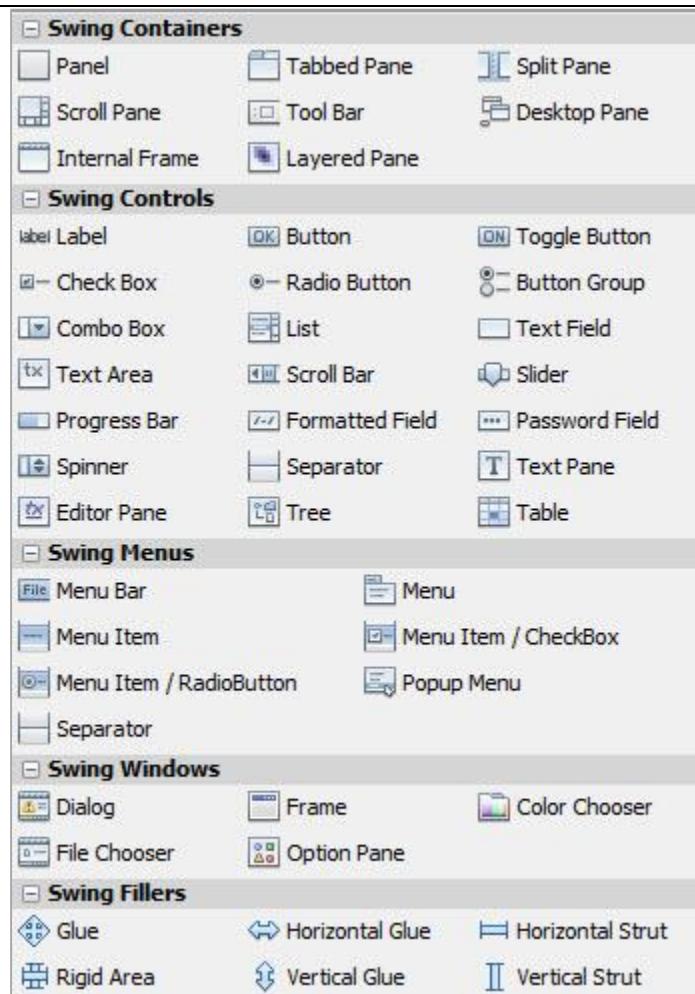
یا به صورت خلاصه‌تر زیر :

```
button1.addActionListener(new ActionListener() //Step 1 , Step3
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        JOptionPane.showMessageDialog(null, "Hello world!"); //Step 2
    }
});
```

در آخر این نکته را یادآور می‌شوم که ما در مثال بالا برخی از موارد مانند تعیین اندازه و محل قرار گیری دکمه و ... را توضیح نداده‌ایم تا شما اصل مطلب که همان کنترل رویداد است را یاد بگیرید. در درس‌های آینده در مورد کنترل‌های Swing بیشتر توضیح می‌دهیم.

کنترل‌ها

کنترل‌ها اجزای بصری هستند که به وسیله‌ی آن‌ها محیط‌های گرافیکی (GUI) ساخته می‌شود. هرچیزی که شما در یک محیط گرافیکی می‌بینید یک کنترل است، حتی Frame نیز یک کنترل است. کنترل‌ها در دسته‌های مختلفی در قسمت نوار ابزار قرار دارند که در محیط Netbeans و در قسمت Palette قائل مشاهده هستند :



بیشتر کنترل‌ها از کلاس `JComponent` ارث بری می‌کنند که دارای خواص، متدها و رویدادهای متفاوتی برای آن‌ها می‌باشد.

خواص کنترل‌ها

در قسمت زیر بعضی از خصیت‌های مفید کلاس `JComponent` را مشاهده می‌کنید.

خاصیت	توضیح
<code>accessible name</code>	نامی که کنترل به وسیله آن قابل دسترسی است.
<code>accessible parent</code>	نام والد کنترل را نشان می‌دهد.
<code>background</code>	رنگ قسمت زمینه‌ی کنترل را مشخص می‌کند.

حاشیه کنترل را مشخص می‌کند.	border
مشخص می‌کند که آیا کنترل رنگ پس زمینه داشته باشد یا transparent باشد.	contentareafilled
نوع نشانگر ماوس را مشخص می‌کند	cursor
آیکون کنترل را زمانی که کنترل غیر فعال است تعیین می‌کند.	disabledicon
این قسمت مشخص می‌کند که کنترل فعال است یا نه. اگر مقدار آن False باشد کنترل غیرفعال می‌شود.	enabled
رنگ پیش زمینه کنترل و همچنین رنگ قلم کنترل را مشخص می‌کند.	foreground
مشخص می‌کند که یک کنترل می‌تواند توسط دکمه‌ی Tab مورد دستیابی قرار گیرد یا خیر.	focusable
نوع و اندازه فونت کنترل را مشخص می‌کند.	font
حداکثر اندازه‌ای که کنترل می‌تواند داشته باشد را مشخص می‌کند.	maximumsize
فاصله‌ی بین لبه‌های کنترل مربوطه و لبه‌ی کنترل دیگر را مشخص می‌کند.	margin
کمترین سایز کنترل را مشخص می‌کند.	minimumsize
نام کنترل را مشخص می‌کند. این نام برای اشاره به کنترل در زمان کد نویسی استفاده می‌شود.	name
والد کنترل را مشخص می‌کند.	parent
متنی که در داخل کنترل قرار می‌گیرد.	text
عرض کنترل براساس پیکسل را مشخص می‌کند.	Horizontal Size
ارتفاع کنترل براساس پیکسل را مشخص می‌کند.	Vertical Size

مهمترین خاصیت در جدول بالا خاصیت name است. این خاصیت به شما اجازه می‌دهد که از آن نام برای ارجاع به کنترل در قسمت کد

نویسی استفاده کنید. در این قسمت به بحث در مورد بعضی از خاصیت‌هایی که در بیشتر کنترل‌ها وجود دارند می‌پردازیم.

رویدادهای کنترل

در جدول پایین شما برخی از رویدادهای مفید که بین بیشتر کنترل‌های رایج هستند را مشاهده می‌کنید.

رویداد	توضیح
actionPerformed	این رویداد وقتی رخ می‌دهد که بر روی کنترل با دکمه‌ی سمت چپ ماوس کلیک شود.
componentResized	این رویداد وقتی رخ می‌دهد که اندازه کنترل تغییر کند.
componentMoved	این رویداد وقتی رخ می‌دهد که کنترل حرکت داده شود.
mouseDragged	زمانی که عملیات Drag & Drop بر روی کنترل انجام می‌شود این رویداد رخ می‌دهد.
focusGained	زمانی رخ می‌دهد که Focus روی کنترل قرار گیرد.
keyPressed	این رویداد زمانی رخ می‌دهد که focus بر روی کنترل باشد و کلیدی از روی صفحه کلید فشرده و رها شود.
keyReleased	این رویداد زمانی رخ می‌دهد که focus بر روی کنترل باشد و کلیدی که از روی صفحه کلید فشرده شده، رها شود. این رویداد بعد از دو رویداد قبلی رخ می‌دهد.
focusLost	وقتی که کنترل Focus خود را از دست بدهد این رویداد رخ می‌دهد.
mouseClicked	یک حالت پیشرفته‌تر از رویداد actionPerformed است. کلیک کردن می‌تواند شامل فشردن کلیدهای صفحه‌ی کلید نیز باشد. ولی اگر شما نیاز به اطلاعاتی نظیر: تعداد کلیک بر روی کنترل، چرخیدن دکمه چرخنده‌ی ماوس و ... باشید باید از این رویداد استفاده کنید
mousePressed	این رویداد زمانی رخ می‌دهد که دکمه‌ای از ماوس در داخل کنترل پایین نگه داشته شود.
mouseEntered	زمانی که مکان نما به یک کنترل وارد می‌شود این رویداد رخ می‌دهد.

این رویداد زمانی رخ می‌دهد که مکان نما کنترل را ترک کند.	mouseExited
این رویداد زمانی رخ می‌دهد که مکان نما وقتی در داخل محدوده کنترل است حرکت کند.	mouseMoved
این رویداد زمانی رخ می‌دهد که دکمه‌ای از ماوس را که قبلاً فشرده‌ایم، در داخل کنترل رها کنیم.	mouseReleased
این رویداد زمانی رخ می‌دهد که بر روی کنترل Focus باشد و دکمه چرخدنده ماوس حرکت کند.	mouseWheelMoved

در مورد بیشتر رویدادهای بالا در بخش‌های آینده صحبت خواهیم کرد.

متدهای کنترل

به استثنای container های سطح بالا، همه کنترل‌های Swing ای که نام آن‌ها با حرف J شروع می‌شود از کلاس JComponent ارث بری می‌کنند. به عنوان مثال JPanel، JScrollPane، JButton و JTable از JFrame و JDialog از Container کلاس JComponent از کلاس Component مشتق می‌شود. از آنجاییکه کلاس JComponent پایه کنترل‌های Swing است در نتیجه کنترل‌های Swing از متدهای این کلاس استفاده می‌کنند. در جدول زیر متدهای پر کاربرد این کلاس ذکر شده است :

متدهای کنترل	کاربرد
setBorder() getBorder()	برای به دست آوردن و تعیین حاشیه (Border) کنترل به کار می‌رود.
setForeground() setBackground()	برای تعیین رنگ فونت و پس زمینه کنترل به کار می‌رود.
getForeground() getBackground()	برای به دست آوردن رنگ فونت و پس زمینه کنترل به کار می‌رود.
setFont() getFont()	برای به دست آوردن و تعیین فونت کنترل به کار می‌رود. اگر کنترل فونت نداشته باشد از فونت کنترل والد آن استفاده می‌شود.
setCursor() getCursor()	برای به دست آوردن و تعیین نشانگر ماوس زمانیکه ماوس بر روی کنترل می‌رود به کار می‌رود.

برای نمایش یک متن در tooltip کنترل به کار می‌رود.	<code>setToolTipText()</code>
برای به دست آوردن و تعیین نام کنترل به کار می‌رود.	<code>setName()</code> <code>getName()</code>
تشخیص می‌دهد که آیا کنترل در صفحه نمایش داده شده است یا نه؟ این بدهی معناست که خاصیت <code>visible</code> کنترل و والد آن باید <code>true</code> باشد.	<code>isShowing()</code>
برای تشخیص و تعیین فعال بودن کنترل به کار می‌رود.	<code>setEnabled()</code> <code>isEnabled()</code>
برای تشخیص و تعیین نمایش و عدم نمایش کنترل به کار می‌رود.	<code>setVisible()</code> <code>isVisible()</code>
برای اضافه کردن یک کنترل به <code>container</code> به کار می‌رود.	<code>add()</code>
برای حذف یکی از یا همه کنترل‌های یک <code>container</code> به کار می‌رود.	<code>remove()</code> <code>removeAll()</code>
برای به دست آوردن بالاترین یک کنترل به کار می‌رود.	<code>getTopLevelAncestor()</code>
برای به دست آوردن اولین یک کنترل به کار می‌رود.	<code>getParent()</code>
برای به دست آوردن تعداد کنترل‌های یک <code>container</code> به کار می‌رود.	<code>getComponentCount()</code>
برای به دست آوردن یکی از یا همه کنترل‌های یک به کار می‌رود.	<code>getComponent()</code> <code>getComponents()</code>
بعد سوم (z-order) یک کنترل در داخل یک را برمی‌گرداند. بالاترین کنترل دارای بعد سوم کوچکتر (عدد کمتر) و پایین‌ترین کنترل دارای بعد سوم بزرگتری است.	<code>getComponentZOrder()</code>
برای تعیین بهترین اندازه کنترل در مقیاس پیکسل به کار می‌رود. اندازه کنترل نباید بزرگ‌تر از <code>maximum size</code> و یا کوچک‌تر از <code>minimum size</code> باشد.	<code>setPreferredSize()</code> <code>setMaximumSize()</code> <code>setMinimumSize()</code>
برای به دست آوردن <code>minimum size</code> و <code>maximum size</code> کنترل در مقیاس پیکسل به کار می‌رود.	<code>getPreferredSize()</code> <code>getMaximumSize()</code> <code>getMinimumSize()</code>

برای تعیین تراز بندی کنترل نسبت به کنترل‌های دیگر به کار می‌رود. مقداری که ایم متدهای setAlignmentX() و setAlignmentY()	
برای به دست آوردن تراز بندی کنترل نسبت به کنترل‌های دیگر به کار می‌رود.	getAlignmentX() getAlignmentY()
برای به دست آوردن عرض و ارتفاع یک کنترل بر حسب پیکسل به کار می‌رود.	getWidth() getHeight()
برای به دست آوردن اندازه کنترل بر حسب پیکسل به کار می‌رود.	getSize() getSize()
برای به دست آوردن نقطه x و y کنترل نسبت به گوشه بالا و سمت چپ کنترل والد آن بر حسب پیکسل به کار می‌رود.	getX() getY()
برای به دست آوردن محدوده یک کنترل (عرض و ارتفاع) بر حسب پیکسل به کار می‌رود.	getBounds()
برای به دست آوردن مکان یک کنترل نسبت به گوشه بالا سمت چپ کنترل والد آن بر حسب پیکسل به کار می‌رود.	getLocation()
مکان نسبی را نسبت به گوشه بالا سمت چپ صفحه نمایش بر می‌گرداند.	getLocationOnScreen()
اندازه حاشیه کنترل را بر می‌گرداند.	getInsets()
مکان کنترل را نسبت به گوشه بالا سمت چپ کنترل والد تعیین می‌کند.	setLocation(Point)
اندازه کنترل را بر مبنای پیکسل تعیین می‌کند.	setSize()
اندازه و مکان نسبی کنترل را نسبت به گوشه بالا سمت چپ کنترل والد مشخص می‌کند.	setBounds()

به این نکته توجه کنید که این کلاس دارای متدهای بیشتری است که در این درس به این تعداد بسنده کردہ‌ایم. در درس‌های آینده با

کاربرد این متدها بیشتر آشنا می‌شوید.

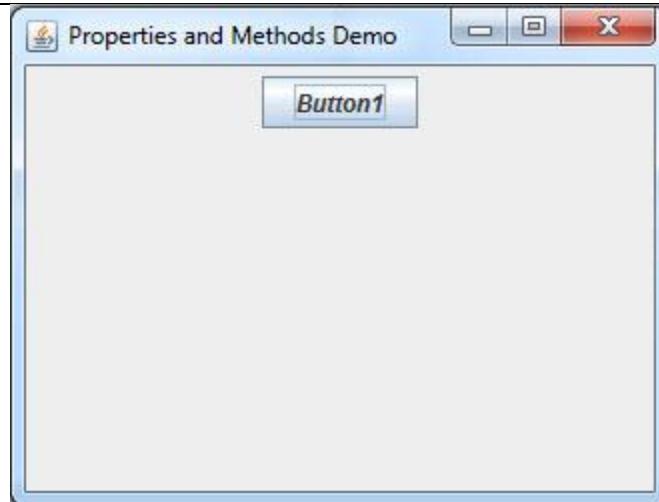
کار با برخی از متدها و خاصیت‌های کنترل‌ها

در این درس می‌خواهیم شما را با برخی از متدها و خاصیت‌هایی که در درس قبل نام بردیم آشنا کنیم. ولی ابتدا باید یک نکته‌ای را یادآور شویم که چون کنترل‌ها همه کلاس هستند باید برای استفاده از هر یک از این کلاس‌ها باید یک نمونه از آن‌ها ایجاد کرد. در این درس ما به سه کلاس برای توضیح بیشتر خاصیت‌ها و متدها نیاز داریم که عبارت‌اند از JButton، JFrame و JPanel. در مورد این سه کنترل در درس‌های آینده بیشتر توضیح می‌دهیم. فقط در این درس می‌خواهیم یک دکمه را به پنل و پنل را فریم اضافه کنیم و سپس خاصیت‌ها و متدهای پر کاربرد را تست کنیم. برای شروع برنامه NotePad را باز کرده و کدهای زیر را در داخل آن بنویسید :

```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         JFrame frame1 = new JFrame("Properties and Methods Demo");
11         JPanel panel1 = new JPanel();
12         JButton button1 = new JButton("Button1");
13
14         panel1.add(button1);
15         frame1.add(panel1);
16
17         frame1.setSize(330 , 250);
18         frame1.setVisible(true);
19     }
20 }
```

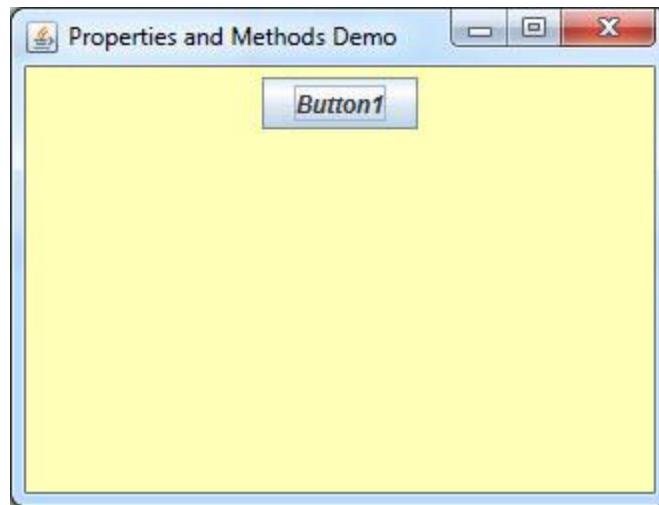
در همین کد بالا کاربرد چندین متدهای می‌توان فهمید. در خطوط ۱۰-۱۲ یک Frame، یک Panel و دو Button ایجاد کرده‌ایم. در خط ۱۴ دکمه را به پنل و در خط ۱۵ هم Panel را به Frame اضافه کرده‌ایم. این کار را با استفاده از متدهای add() و setVisible() انجام داده‌ایم. پس فعلًاً با کاربرد آشنایی شدید. در خط ۱۷ با استفاده از متدهای setFrame() و setSize() یک اندازه برای Frame تعیین کرده و در خط ۱۸ با استفاده از متدهای set() و setVisible() فریم را نمایش می‌دهیم. با اجرای برنامه بالا پنجره‌ای به صورت زیر ایجاد می‌شود :



به این نکته توجه کنید که تمام کدهای پایین را می‌توانید در خط ۱۳ کد بالا نوشته و بعد از تست و مشاهده نتیجه پاک کرده و کد بعدی را بنویسید. فرض کنید که می‌خواهیم رنگ پس زمینه Panel را تغییر دهیم. برای این کار از متدهای setBackground استفاده می‌کنیم. این متدهای تغییر رنگ پس زمینه، یک شیء از کلاس Color با عنوان پارامتر قبول می‌کند. در خط ۱۳ کد زیر را بنویسید :

```
panel1.setBackground(Color.getHSBColor(10, 20, 50));
```

اعدادی که در داخل متدهای getHSBColor نوشته شده‌اند می‌توانند اعدادی بین ۰-۲۵۵ باشند. برنامه را اجرا و نتیجه را مشاهده کنید :



البته به جای کد بالا می‌توانید از نام رنگ‌ها هم به صورت زیر استفاده کنید :

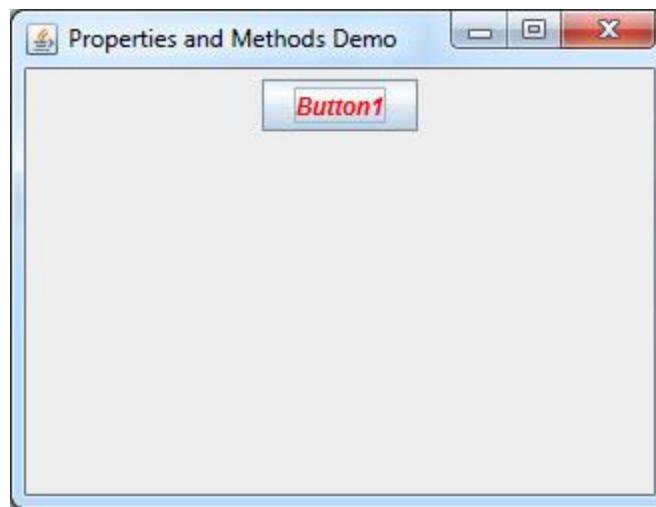
```
panel1.setBackground(Color.BLUE);
```

حال فرض کنید که می خواهیم رنگ نوشته Button را به قرمز تغییر دهیم. برای این کار از متده استفاده می کنیم.

برای این کار به صورت زیر عمل می کنیم :

```
button1.setForeground(Color.red);
```

کد را اجرا و نتیجه را مشاهده کنید :

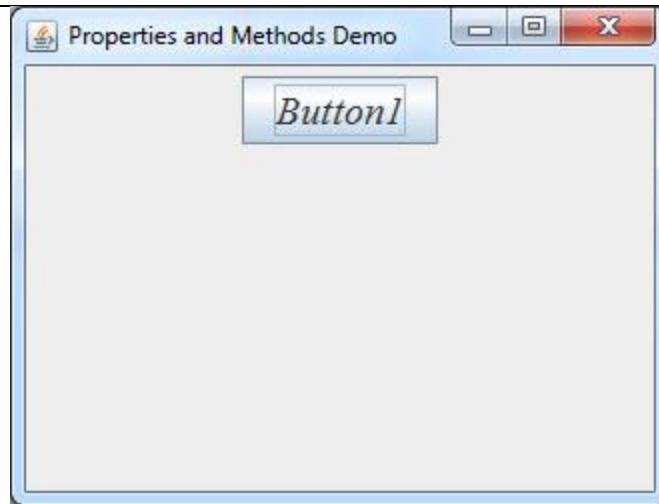


برای تغییر فونت دکمه هم می توان از متده `setFont()` و یک شیء از کلاس `Font` استفاده کرد. برای این کار به صورت زیر عمل می کنیم :

```
button1.setFont(new Font("Times New Roman", Font.ITALIC, 20));
```

همانطور که مشاهده می کنید شیء `Font` سه پارامتر می گیرد که اولی نام، دومی حالت و سومی اندازه فونت را مشخص می کند. برنامه را

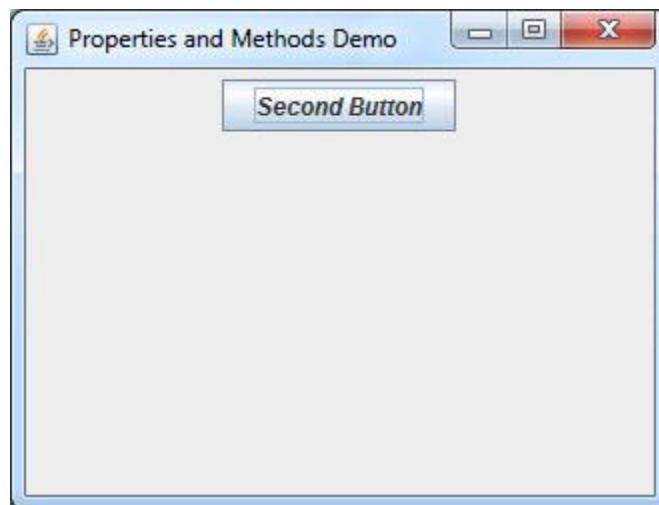
اجرا و نتیجه را مشاهده کنید :



برای تغییر نام یا اختصاص یک نام به یک کنترل هم می‌توان از متده استفاده کرد. فرض کنید که می‌خواهیم نام دکمه button2 را به Second Button تغییر بدم. برای این کار به صورت زیر عمل می‌کنیم :

```
button2.setText("Second Button");
```

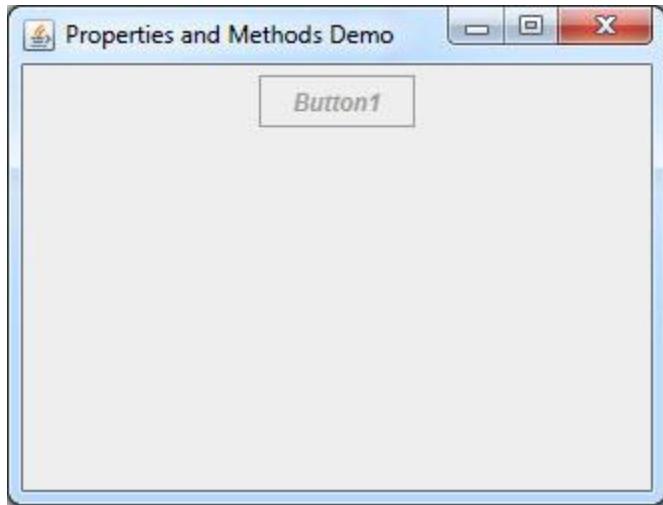
برنامه را اجرا و نتیجه را مشاهده کنید :



برای غیرفعال کردن یک کنترل هم از متده setEnabled() استفاده می‌شود. فرض کنید که می‌خواهیم نام دکمه button1 را غیرفعال کنیم. برای این کار کد زیر را بنویسید :

```
button1.setEnabled(false);
```

برنامه را اجرا و نتیجه را مشاهده کنید :



برای تغییر نشانگر ماوس هم می‌توان از متد (`setCursor()`) به همراه یک شیء از کلاس `Cursor` استفاده کرد. این کلاس دارای ثابت‌هایی

هست که در زیر یک مثال از آن‌ها را مشاهده می‌کنید :

```
button1.setCursor(new Cursor(Cursor.WAIT_CURSOR));
```

حال اگر برنامه را اجرا کرده و بر روی دکمه با ماوس توقف کنید، مشاهده می‌کنید که شکل نشانگر ماوس تغییر می‌کند. برای اختصاص آیکون به یک کنترل مثلاً دکمه هم از کلاس `ImageIcon` استفاده می‌شود. من از قبل یک آیکون را در داخل پوشه برنامه قرار داده‌ام. برای

اختصاص این آیکون به دکمه بعد از خط ۹ کد زیر را بنویسید :

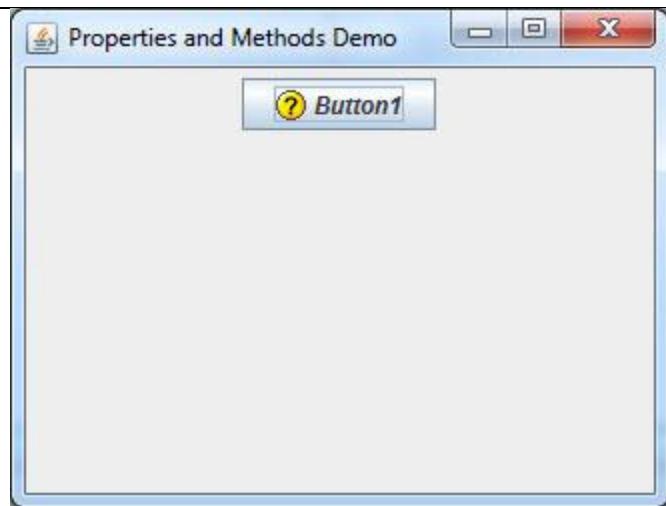
```
 ImageIcon icon = new ImageIcon("C:/FrameDemo/icon.gif", null);
```

من در کد بالا از آدرس مطلق آیکون استفاده کرده‌ام و ممکن است که این آدرس برای شما به نحو دیگری باشد. سپس سازنده کلاس

را به صورت زیر تغییر دهید :

```
JButton button1 = new JButton("Button1", icon);
```

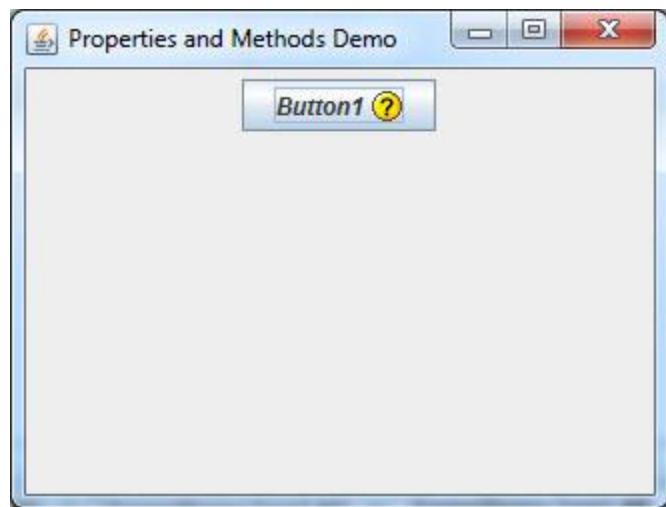
برنامه را اجرا و نتیجه را مشاهده کنید :



حال اگر بخواهید آیکون در سمت راست و متن در سپ چپ باشد می‌توانید از متده استفاده کنید. این رابط مقادیر مختلفی برای تراز بندی دارد که یکی از آنها LEFT است و متن را در سمت چپ آیکون قرار می‌دهد. کد زیر را بعد از ایجاد دکمه بنویسید :

```
button1.setHorizontalTextPosition(SwingConstants.LEFT);
```

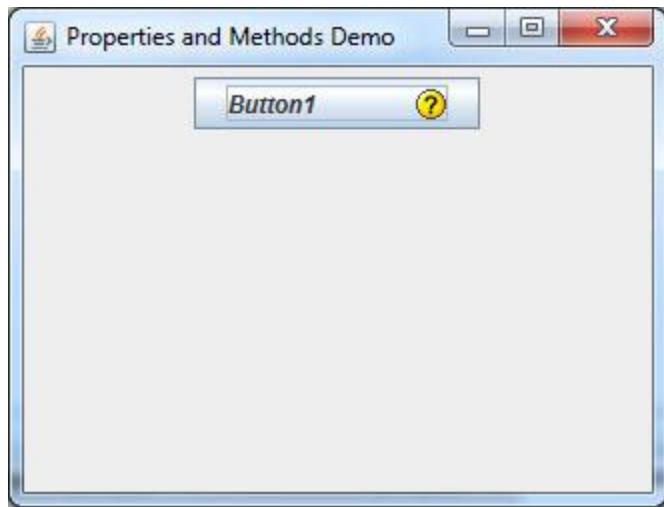
برنامه را اجرا کنید :



برای ایجاد فاصله بین متن و آیکون هم از متده استفاده می‌شود :

```
button1.setIconTextGap(50);
```

کد بالا بین متن و آیکون ۵۰ پیکسل فاصله می‌اندازد :

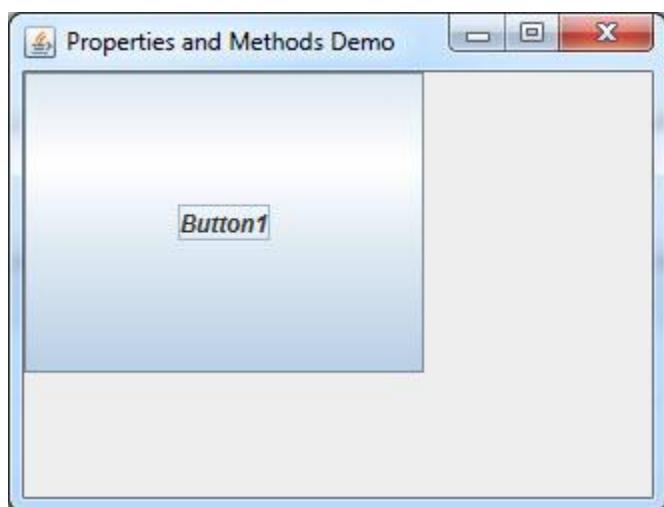


برای تغییر اندازه کنترل از متدهای `setSize()` استفاده می‌شود. با فرض اینکه همان کد ابتدایی درس را در اختیار دارید بعد از خط ۱۲ کدهای

زیر را بنویسید:

```
panel1.setLayout(null);
button1.setSize(200,150);
```

در مورد متدهای `setLayout()` لازم نیست چیزی بدانید چون در مورد آن در قسمت کار با لایه‌ها توضیح خواهیم داد. فقط در این حد بدانید که پارامتر این متدهای `null` باشد تا عملکرد متدهای `setSize()` قابل مشاهده باشد. حال برنامه را اجرا و نتیجه را مشاهده کنید :

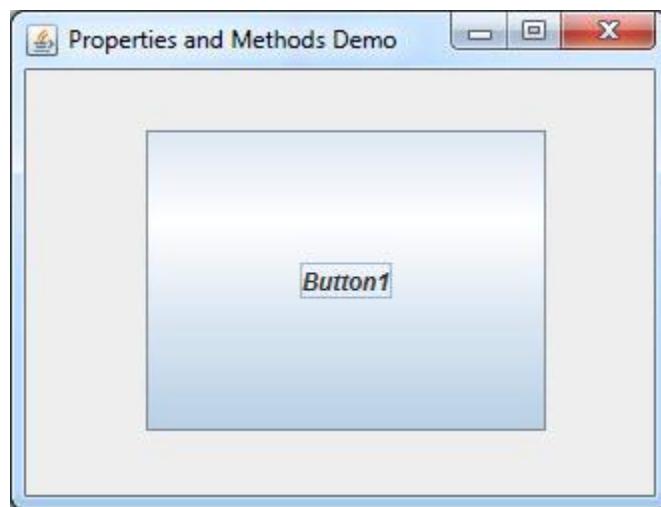


مشاهده می‌کنید که دکمه نسبت به گوشه بالا و سمت چپ کنترل تراز می‌شود. حال اگر بخواهیم دکمه را به طرف چپ یا راست بکشیم

باید از متده استفاده کنیم :

```
button1.setBounds(60,30,200,150);
```

دو عدد اول به ترتیب مختصات x و y نسبت به گوشه بالا و سمت چپ Panel و دو عدد بعدی هم عرض و ارتفاع کنترل را مشخص می‌کنند :



نامگذاری کنترل‌ها

همیشه کنترل‌ها را نامگذاری کنید. ما از خاصیت name برای نامگذاری کنترل‌ها استفاده می‌کنیم. نامگذاری کنترل‌ها از قواعد نامگذاری متغیرها استفاده می‌کند، برای مثال فاصله‌ها، کاراکترهای ویژه، و استفاده از کلمات کلیدی در نامگذاری کنترل‌ها نیز ممنوع است. یک سری قرارداد در نامگذاری کنترل‌ها وجود دارد که در ادامه به آن‌ها اشاره می‌کنیم.

شما می‌توانید بسته به نوع استفاده‌ای که از کنترل می‌کنید آنرا نامگذاری نمایید. برای مثال، وقتی که قرار است در داخل یک JTextField نام یک شخص یا کاربر وارد شود می‌توانید نام آن JTextField را برای خوانایی بیشتر firstname قرار دهید. اما در نامگذاری کنترل‌ها بهتر است که نام واقعی آن کنترل قبل از نامی که برای آن انتخاب کردہ‌ایم بیاید. برای مثال به جای firstname در نامگذاری Field، می‌توانید از TextFieldFirstName استفاده کنید. با توجه به این قرار دادها، ما می‌توانیم در داخل Intellisense که با آن کار می‌کنیم هر کنترلی را شناسایی کنیم.

تکنیک دیگری که برای نامگذاری کنترل‌ها به کار می‌رود، خلاصه کردن نام آن‌ها است. برای مثال، به جای استفاده از کلمه‌ی

تعداد زیادی کلمه‌ی کوتاه شده برای هر کنترل وجود دارد و شما حتی می‌توانید از کلمه‌ی کوتاه شده‌ای که خود برای آن کنترل انتخاب

کرده‌اید استفاده کنید. البته این باید گویا باشد. به طوری که برای کسانی که به کد شما نگاه می‌کنند واضح باشد.

تکنیک دیگری که برای نامگذاری کنترل‌ها بکار می‌رود جابجا کردن نام اصلی کنترل با نامی که ما برای آن انتخاب کردہ‌ایم و همچنین استفاده از روش کوهان شتری (Camel case) است. برای مثال یک `TextField` داریم که نام کاربر یا شخصی را از ورودی می‌گیرد، می‌توانید آنرا به صورت `firstNameTextField` نامگذاری کنید و یا یک دکمه دارید که عملیات محاسباتی را انجام می‌دهد، می‌توانید نام آنرا `calculateButton` بگذارید.

در درس‌های آینده از تکنیک سوم (Camel Case) استفاده می‌کنیم. نیازی به این نیست که شما تمامی کلمات اختصاری کنترل‌ها را حفظ کنید. وقتی که شما یک کنترل را ایجاد می‌کنید، تنها کافیست عدد پسوندی آنرا حذف کنید و به جای آن، نامی را با توجه به کاربرد کنترل به آن اضافه کنید. وقتی که شما در حال تایپ کردن هستید و قصد پیدا کردن کنترل خاصی را دارید کافیست براحتی نام کنترل را تایپ کرده تا پنجره‌ی `intellisence` نام آن کنترل و تمامی کنترل‌های مشابه آنرا را برای شما نمایش دهد. برای مثال، یک `TextField` و تمامی کنترل‌های `TextField` در `intellisence` برنامه نمایش داده می‌شوند، فقط قسمتی که مربوط به نامی که شما برای آن کنترل انتخاب می‌کنید ممکن است طولانی باشد، که شما با توجه به قواعدی که پیش‌تر ذکر شد می‌توانید نسبت به نامگذاری و کوتاه کردن نام کنترل‌ها اقدام کنید. در زیر مثال‌هایی از نامگذاری کنترل‌ها بسته به کاربرد آن‌ها ذکر شده است :

کاربرد	نام پیشنهادی
کلیدی که برای نشان دادن یک پیغام به کار می‌رود.	<code>buttonConfirm, confirmButton, btnConfirm</code>
یک <code>TextField</code> که یک آدرس را از کاربر دریافت می‌کند.	<code>textFieldAddress, addressTextBox, txtAddress</code>
یک فرم که اطلاعات شخصی را از کاربر دریافت می‌کند.	<code>formPersonalInformation, personalInformationForm, frmPersonalInformation</code>
یک <code>ComboBox</code> که لیستی از محصولات را نمایش می‌دهد.	<code>comboBoxProducts, productsComboBox, cmbProducts</code>

یک <code>RadioButton</code> که نشان دهنده مذکور بودن یک کاربر است.	<code>radioButtonMale, maleRadioButton, radMale</code>
یک <code>MenuItem</code> برای ذخیره کردن یک فایل.	<code>menuItemSave, saveMenuItem, mnuSave</code>
یک <code>CheckBox</code> برای فرستادن یک نامه.	<code>checkBoxSubscribe, subscribeCheckBox, chkSubscribe</code>

لازم نیست که تمامی کنترل‌های یک فرم را نام گذاری کنید. کنترل‌هایی که هیچوقت در کدنویسی مورد استفاده قرار نمی‌گیرند، می‌توانند از نام پیش فرض خود استفاده کنند. برای مثال، اکثر `Label` ها صرفاً برای نامگذاری بکار می‌روند و در کدنویسی مورد استفاده قرار نمی‌گیرند. لذا لازم نیست خاصیت `name` آن‌ها را تغییر دهیم. همیشه عادت کنید که کنترل‌های خود را نامگذاری کنید.

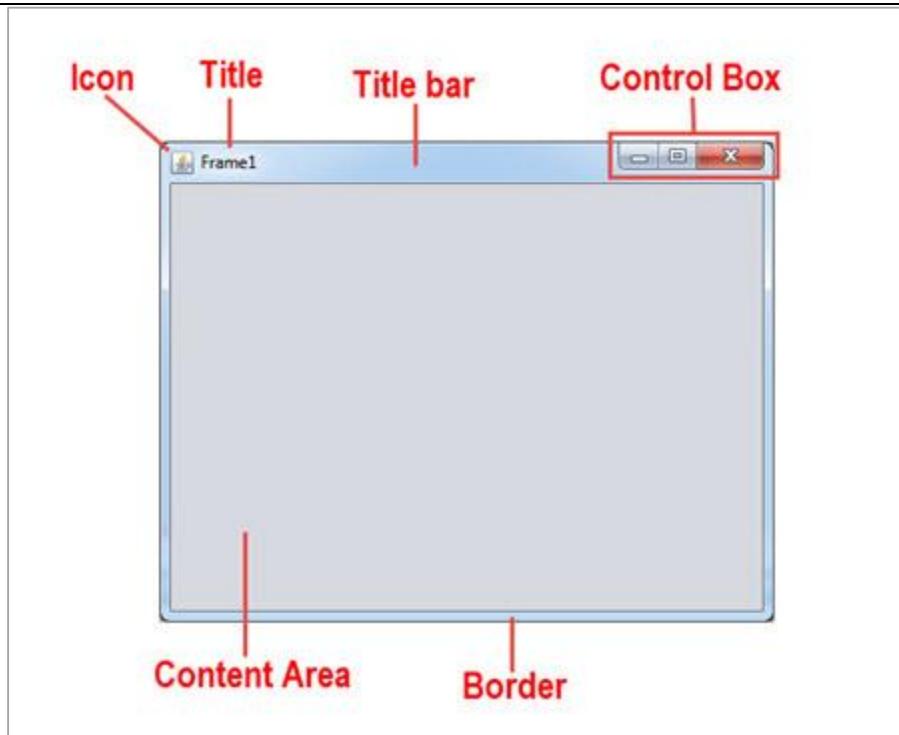
JFrame

پنجره‌ای است که شما در برنامه‌هایتان، کنترل‌های دیگر را بر روی آن قرار می‌دهید. شما می‌توانید در یک برنامه بیش از یک `Frame` داشته باشید. هر `Frame` از خواص و متدهای کلاس `Window` ارث می‌برد. از لحاظ ویژوالی یا بصری، در بالاترین سطح کلاس‌های `Swing` قرار داردن. `Frame` دارای دو بعد است:

- بعد ظاهري
- بعد لایه بندی

Frame ظاهر

در شکل زیر شما نام قسمت‌های مختلف `Frame` را مشاهده می‌کنید.



در قسمت بالا، نوار عنوان (Title Bar) قرار دارد. نوار عنوان از یک آیکن (Icon)، یک عنوان (Title)، و یک جعبه‌ی کنترل (Control Box) تشکیل شده است. جعبه‌ی کنترل (Control Box) دکمه‌های کوچکنمایی (Minimizing)، بزرگنمایی (Maximizing)، و بستن (Closing) را در بر دارد. قسمت داخلی یا همان Content Area مکانی است که ما کنترل‌ها را در آن قرار می‌دهیم. قسمت حاشیه‌یا Border که شامل قسمت Title Bar نیز می‌شود، به شما اجازه می‌دهد که سایز فرم را تغییر دهید. نمونه‌ای از کلاس JFrame می‌باشد و دارای دو سازنده اصلی می‌باشد. که یکی از آن‌ها برای ایجاد یک Frame بدون عنوان و دیگری برای ایجاد Frame همراه با عنوان به کار می‌رود :

```
JFrame frame = new JFrame();
JFrame frame = new JFrame("Title Bar");
```

حال اجازه دهید که یک Frame مانند شکل بالا و با استفاده از کدنویسی ایجاد کنیم. به کدهای زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import javax.swing.*;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
```

```

8
9     {
10    JFrame frame1 = new JFrame("FrameDemo");
11    frame1.setSize(430 ,315);
12    frame1.setVisible(true);
13 }

```

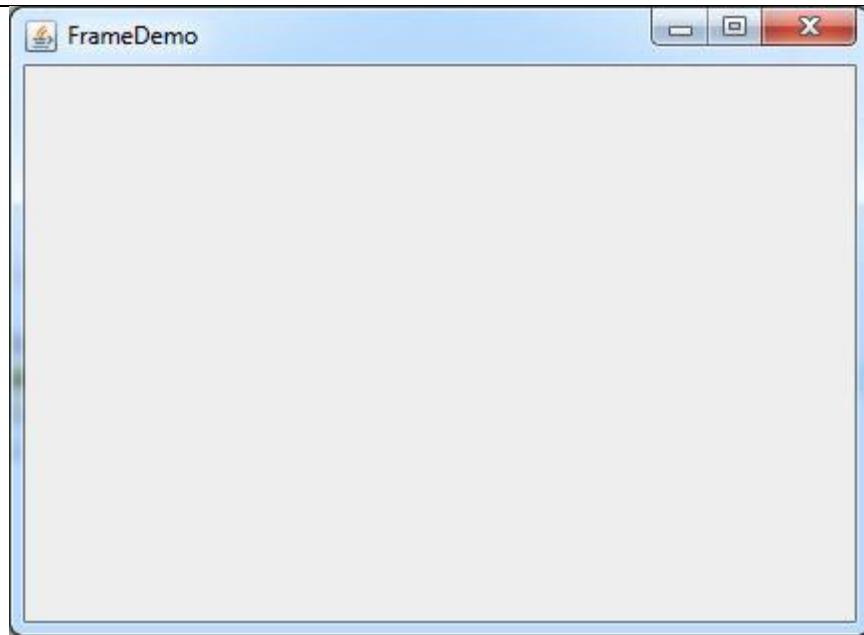
در خط ۱ کد بالا چون قرار است که یک Frame ایجاد کنیم در نتیجه کلاس‌های Swing را وارد برنامه کرده‌ایم. در خط ۵ یک کلاس به نام Frame ایجاد کرده‌ایم. در خط ۹ یک نمونه از کلاس JFrame ایجاد کرده و در داخل سازنده آن یک عنوان برای MyFirstProgram می‌نویسیم. در خط ۱۰ با استفاده از متدهای setSize و setVisible فریم تعیین می‌کنیم و در خط ۱۱ با استفاده از متد () هم آن را نمایش می‌دهیم. به این نکته توجه کنید که ما در کد بالا فقط اندازه و عنوان را تعیین کردیم و با اجرای کد قسمت ControlBox به صورت پیشفرض در Frame وجود دارد. در حالت پیشفرض وقتی که یک Frame را می‌بندید در اصل مخفی می‌شد. کلاس JFrame دارای ثابت‌هایی برای بستن یا مخفی کردن Frame می‌باشد که در زیر لیست آنها آمده است :

ثابت	توضیح
EXIT_ON_CLOSE	System.exit(0) فراخوانی می‌شود.
DISPOSE_ON_CLOSE	dispose() فراخوانی می‌شود.
DO NOTHING ON CLOSE	پنجره بسته نمی‌شود.
HIDE_ON_CLOSE	در حکم مخفی شدن (setVisible(false)) پنجره می‌بندد. این ثابت پیشفرض است.

برای استفاده از این ثابت‌ها بعد از ایجاد یک نمونه از کلاس JFrame متد setDefaultCloseOperation() را فراخوانی کرده و یکی از ثابت‌های بالا را به آن ارسال می‌کنیم. مثلاً بعد از خط ۷ کد بالا می‌توانیم خط زیر را بنویسیم :

```
frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

با اجرای کد بالا فریمی به صورت زیر ایجاد می‌شود :

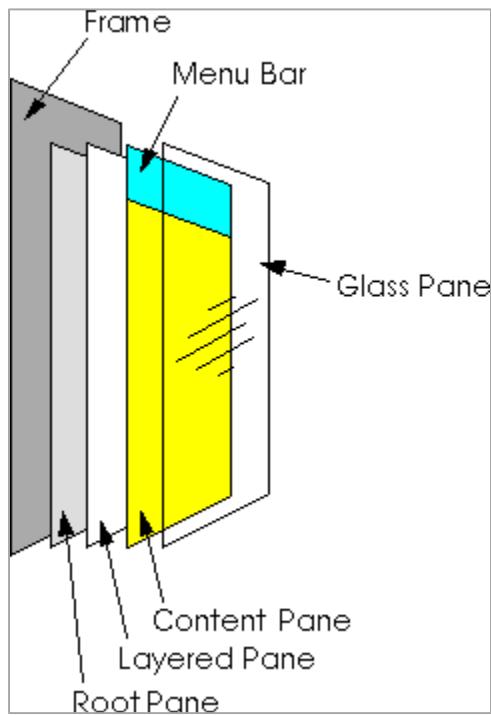


لایه بندی Frame

ظاهر ساده Frame دارای چهار لایه اصلی، قاب یا pan می‌باشد :

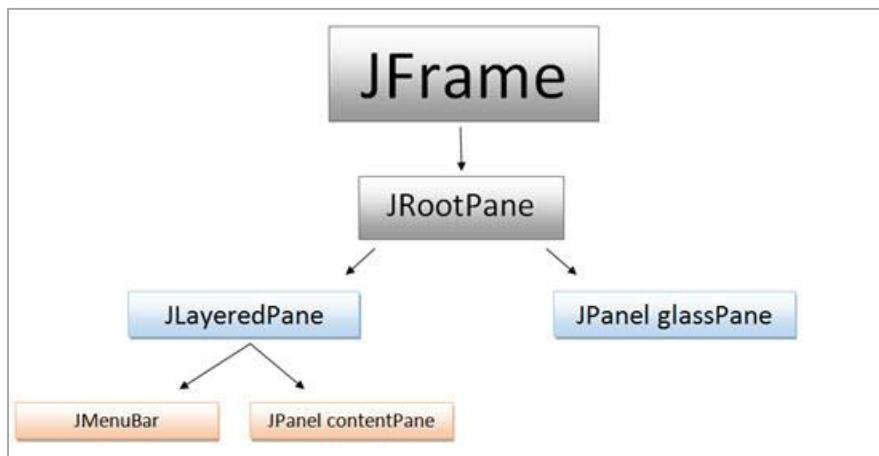
- Root Pane
- Layered Pane
- Content Pane
- Glass Pane

لایه‌های تشکیل دهنده یک Frame به صورت زیر می‌باشند :



یک JFrame دارای یک نمونه از کامپونت glassPane و JRootPane میباشد. خود JFrame شامل دو کامپونت JMenuBar و JPanel contentPane میباشد. نیز به نوبه خود از دو کامپونت JLayeredPane تشکیل شده است

:



معمولًاً کامپونت های Content Pane به swing اضافه نمیشوند و بطور مستقیم به خود JFrame اضافه نمیشوند و بهتر است که در

هنگام اضافه کردن کنترل‌ها به Frame مشخص کنیم که این کنترل در کدام قاب قرار بگیرد. مثلاً برای اضافه کردن یک کنترل به فریمی:

که در مثال بالا ایجاد کردیم از دو متند add() و getContentPane() به صورت زیر عمل می‌کنیم:

```
frame1.getContentPane().add(...);
```

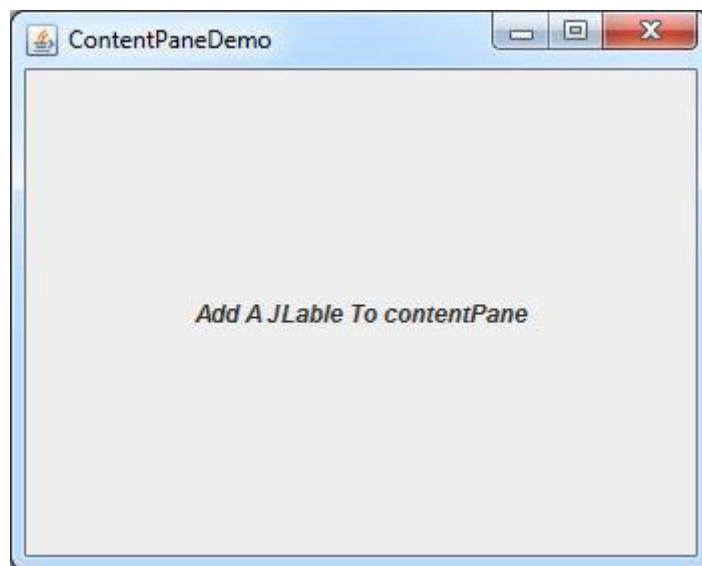
فرض کنید که می‌خواهیم یک کنترل Label به فریم اضافه کنیم در این صورت بعد از خط ۷ کد ابتدایی درس کدهای زیر را بنویسید :

```
JLabel label = new JLabel("Add A JLabel To contentPane", JLabel.CENTER);
frame1.getContentPane().add(label);
```

در کد بالا ما یک کنترل Label را به لایه ContentPane اضافه کردیم. همانطور که مشاهده می‌کنید ابتدا یک کنترل Label ایجاد کرده

و سپس در سازنده کلاس آن یک متن برای آن انتخاب کرده و آن را در وسط فریم قرار داده‌ایم. برای اضافه کردن این کنترل به لایه

ContentPane هم از متند getContentPane() استفاده نموده‌ایم. خروجی کد بالا به صورت زیر است :



مدیریت لایه‌ها و چیدمان کنترل‌ها

یکی از بحث‌های مهم در ساختن یک Graphical User Interface یا یک GUI یا به زبان ساده‌تر یکی از مهم‌ترین و پر کاربردترین مسائله ای که در هنگام ایجاد یک فریم با آن رو برو هستیم، مدیریت لایه‌ها و چیدمان کنترل‌های آن فریم (Layout) است. اینکه بتوانیم فریمی بسازیم که در آن کنترل‌های مختلف در جاهای متفاوت در فریم قرار گیرند، به زیبایی و کارایی برنامه ما می‌افزاید. توصیه می‌کنیم که درس قبل را که در آن در مورد لایه‌ها توضیح دادیم مطالعه بفرمایید.

مدیر لایه با `layout manager` شیء است که اندازه و مکان کنترل‌ها را در داخل یک جعبه (`container`) کنترل می‌کند. منظور از جعبه در جاوا کنترل‌های مانند `Panel` یا `Frame` است، چون که کنترل‌های دیگر در داخل این دو کنترل قرار می‌گیرند. در این بخش ما از کلمه `Container` به جای جعبه استفاده می‌کنیم. هر `Container` دارای یک شیء مدیر لایه پیشفرض است که آن لایه را کنترل می‌کند. در جاوا پنج مدیر لایه وجود دارد:

- `FlowLayout`
- `GridLayout`
- `BorderLayout`
- `CardLayout`
- `GridBagLayout`
- `BoxLayout`
- `GroupLayout`
- `SpringLayout`

برای استفاده از یک مدیر لایه کافیست که یک شیء از مدیر لایه ایجاد و آن را به متدهای `setLayout()` ارسال کنیم :

```
aContainer.setLayout(new LayoutManager());
```

در درس‌های بعدی به توضیح کار با مدیران لایه می‌پردازیم.

BorderLayout

مدیر لایه پیشفرض `BorderLayout` کلاس‌های `JApplet`, `JInternalFrame`, `JDialog`, `JWindow`, `JFrame` و `JContentPanel` بوده و دارای پنج منطقه چپ، راست، بالا، پایین و مرکز می‌باشد. یعنی کنترل‌ها را می‌توان در این پنج منطقه قرار داد. بسته به اندازه ویندوز این پنج منطقه تغییر اندازه می‌دهند و کل صفحه را پوشش می‌دهند. اگر یک کنترل به `Frame` اضافه کرده و منطقه آن را مشخص نکنید به طور پیشفرض در وسط فریم قرار می‌گیرد و اگر چندین کنترل را به یک منطقه اضافه کنید آخرین کنترل اضافه شده نمایش داده می‌شود.

کلاس `BorderLayout` دارای دو سازنده زیر می‌باشد :

```
public BorderLayout()
public BorderLayout(int hgap, int vgap)
```

مناطقی که می‌توان هنگام اضافه کردن کنترل به Container مشخص کرد، عبارت‌اند از :

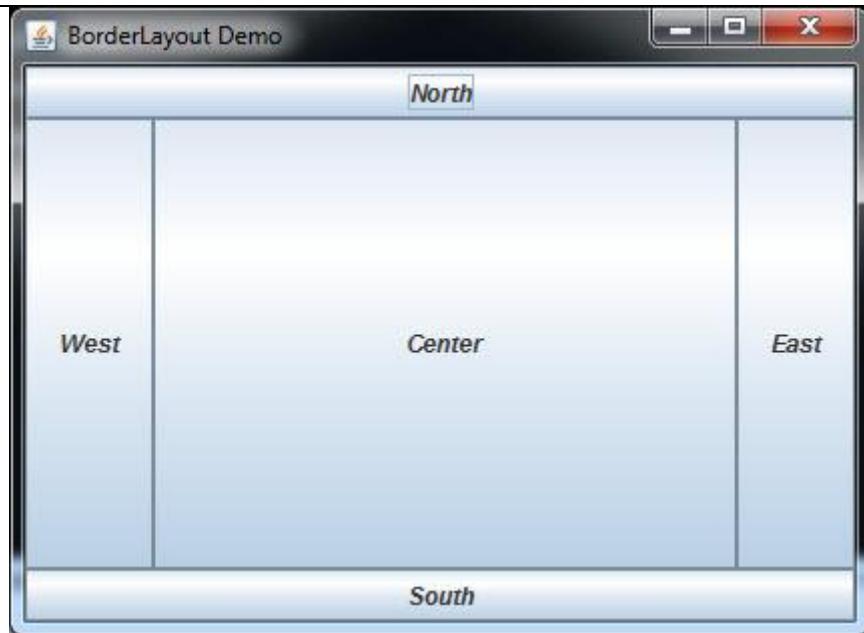
CENTER	•
EAST	•
NORTH	•
SOUTH	•
WEST	•

برای آشنایی بیشتر با عملکرد این کلاس به کد زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         JFrame frame1 = new JFrame("BorderLayout Demo");
11         frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13         frame1.setLayout(new BorderLayout());
14         frame1.setFont(new Font("Helvetica", Font.PLAIN, 14));
15         frame1.add(new JButton("North"), BorderLayout.NORTH);
16         frame1.add(new JButton("South"), BorderLayout.SOUTH);
17         frame1.add(new JButton("East"), BorderLayout.EAST);
18         frame1.add(new JButton("West"), BorderLayout.WEST);
19         frame1.add(new JButton("Center"), BorderLayout.CENTER);
20
21         frame1.setSize(430 ,315);
22         frame1.setVisible(true);
23     }
24 }
```

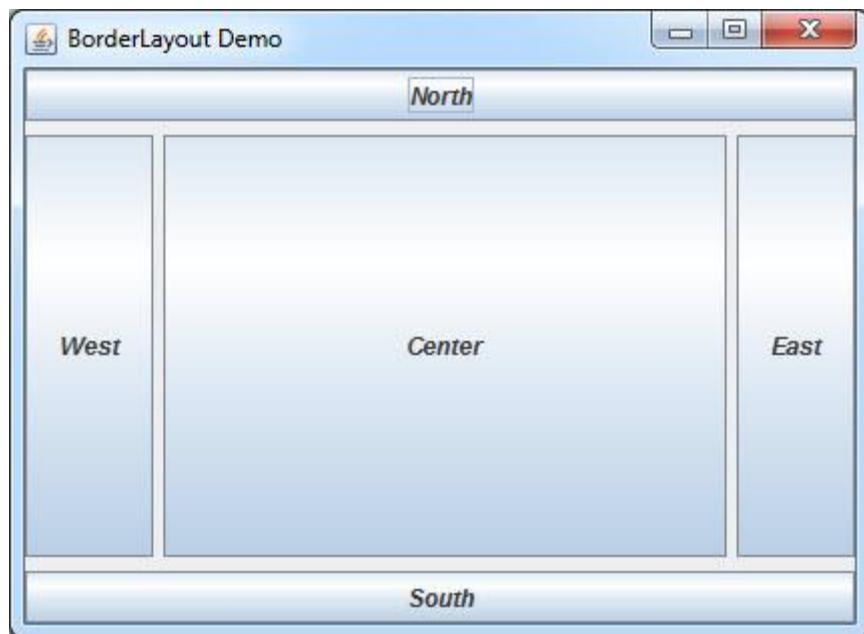
نتیجه اجرای کد بالا به صورت زیر است :



برای ایجاد فاصله بین این قسمت‌ها هم در قسمت سازنده مقادیری از نوع عدد صحیح می‌نویسید. vgap، فاصله عمودی و hgap فاصله افقی از دیگر کنترل‌ها می‌باشد. مثلاً خط ۱۳ کد بالا را به صورت زیر تغییر دهید :

```
setLayout(new BorderLayout(5,8));
```

بعد از اجرای دوباره برنامه فاصله‌هایی که تعیین کرده‌اید در بین قسمت‌ها قابل مشاهده است :



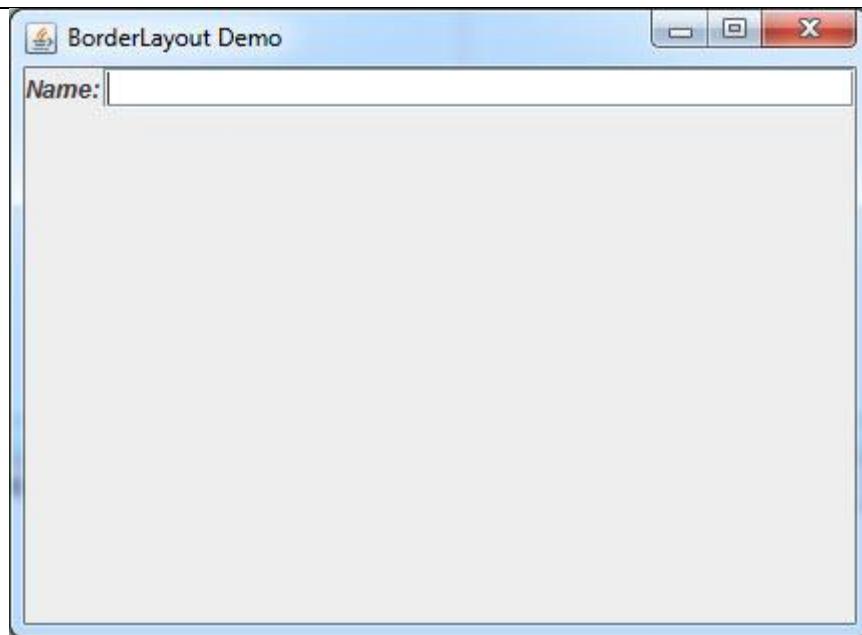
اگر بخواهید چند کنترل را در یک ناحیه قرار دهید، ابتدا باید آن‌ها را به یک Container اضافه کرده و ناحیه آن‌ها را در آن مشخص کنید و سپس Container را به مثلاً Frame اضافه کرده و ناحیه آن را مشخص کنید. فرض کنید که می‌خواهیم یک Label و یک textfield را در ناحیه بالای فریم اضافه کنیم. ابتدا باید آن‌ها را به یک Panel اضافه کرده و سپس Panel را به Frame اضافه کنیم. به کد زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10        JFrame frame1 = new JFrame("BorderLayout Demo");
11        frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        frame1.setLayout(new BorderLayout());
13        frame1.setFont(new Font("Helvetica", Font.PLAIN, 14));
14
15        JPanel topPanel = new JPanel(new BorderLayout());
16        JLabel label    = new JLabel("Name: ");
17        JTextField text = new JTextField();
18        topPanel.add(label, BorderLayout.WEST);
19        topPanel.add(text,  BorderLayout.CENTER);
20        frame1.add(topPanel, BorderLayout.NORTH);
21
22        frame1.setSize(430 ,315);
23        frame1.setVisible(true);
24    }
25 }
```

همانطور که در کد بالا مشاهده می‌کنید ابتدا در خط ۱۵ یک Panel ایجاد کرده و مدیر لایه آن را BorderLayout قرار می‌دهیم. سپس در خطوط ۱۶ و ۱۷ یک Label و یک TextField ایجاد کرده و آن‌ها را در خط ۱۸ و ۱۹ در قسمت چپ و وسط Panel قرار می‌دهیم.

سپس Panel را در خط ۲۰ به قسمت بالای Frame اضافه کرده‌ایم. خروجی کد بالا به صورت زیر است :



CardLayout

با استفاده از CardLayout شما می‌توانید چندین کنترل را در یک فضا نمایش دهید. برای درک بهتر برگه‌های پاسور را در نظر بگیرید که همه روی هم قراردارند و در لحظه فقط یکی از آن‌ها قابل مشاهده است یا تب‌های مرورگرatan را در نظر بگیرید. زمانی که شما یکی از تب‌ها کلیک می‌کنید محتوای پنجره تغییر می‌کند، در حقیقت تمامی محتویات یک tab قبلًا بارگذاری شده و در بالای یکدیگر قرار گرفته‌اند. زمانی که شما روی یکی از تب‌ها کلیک می‌کند فقط یکی از آن‌ها نمایش داده می‌شود و مابقی تب‌ها غیرقابل مشاهده می‌شوند. کتابخانه Swing شامل کنترل‌های آماده برای پنجره‌های همراه با تب‌ها بنام JTabbedPane می‌باشد. برای درک بهتر عملکرد این کلاس به کد زیر

توجه کنید :

```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        // main window
12        JFrame Frame1 = new JFrame("CardLayout Example");
13        Frame1.setSize(430,315);
14
15        final CardLayout cardLayout = new CardLayout();
16        final JPanel cardPanel = new JPanel(cardLayout);

```

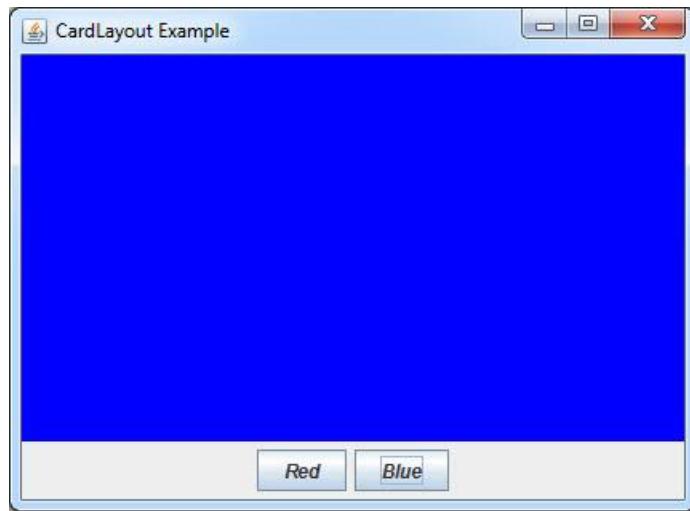
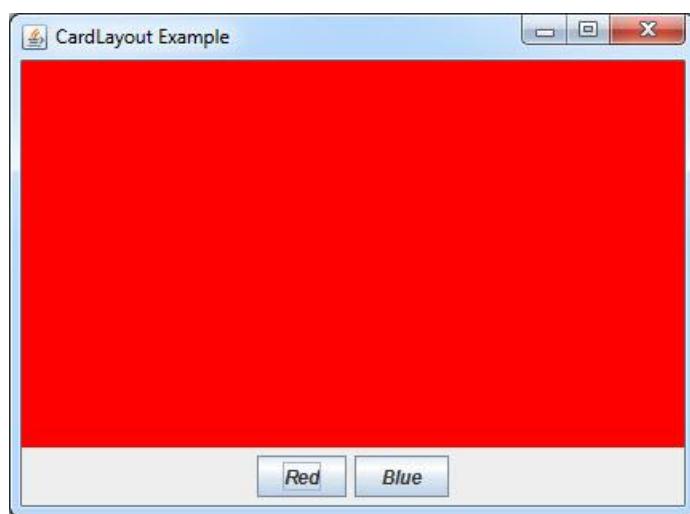
```

17
18     // create two dummy panels (the "cards") to show
19     JPanel card1 = new JPanel();
20     card1.setBackground(Color.red);
21
22     JPanel card2 = new JPanel();
23     card2.setBackground(Color.blue);
24
25     cardPanel.add(card1,"RedCard");
26     cardPanel.add(card2,"BlueCard");
27
28     // create two buttons
29     JPanel buttonPanel = new JPanel();
30     JButton b1 = new JButton("Red");
31     JButton b2 = new JButton("Blue");
32     buttonPanel.add(b1);
33     buttonPanel.add(b2);
34
35     Frame1.add(cardPanel, BorderLayout.CENTER);
36     Frame1.add(buttonPanel, BorderLayout.SOUTH);
37
38     // create action listeners for buttons
39     b1.addActionListener(new ActionListener()
40     {
41         public void actionPerformed(ActionEvent event)
42         {
43             cardLayout.show(cardPanel, "RedCard");
44         }
45     });
46
47     b2.addActionListener(new ActionListener()
48     {
49         public void actionPerformed(ActionEvent event)
50         {
51             cardLayout.show(cardPanel, "BlueCard");
52         }
53     });
54
55     // Show main Frame1
56     Frame1.setVisible(true);
57 }
}

```

هدف کلی کد بالا این است که با کلیک بر روی دو دکمه دو کارت با رنگ‌های قرمز و آبی که بر روی هم قرار دارند را نمایان کنیم. یا فرض کنید که این دو دکمه حکم tab دارند. ابتدا یک کلاس در در خطوط ۵-۶ تعریف کردہ‌ایم. سپس در داخل متدهای main() و در خطوط ۱۲ و ۱۳ یک Frame ایجاد می‌کنیم. در خط ۱۵ یک نمونه از کلاس CardLayout ایجاد و آن را در خط ۱۶ به یک Panel وصل می‌کنیم. این پنل همان پنلی است که کارت‌های قرمز و آبی در آن قرار می‌گیرند. برای ایجاد این دو کارت دو پنل ۱۹-۲۳ ایجاد کرده و آن‌ها را به پنل اصلی یعنی cardPanel در خط ۲۵ و ۲۶ اضافه می‌کنیم. به پارامتر دوم متدهای add() در این دو خط توجه کنید. این پارامترها نام‌هایی هستند که ما به این دو کارت اختصاص داده‌ایم. حال نوبت به اضافه کردن دو دکمه می‌رسد. در خط ۲۹ یک Panel دیگر ایجاد کرده و

دو دکمه به نامهای Red و Blue به این پنل اضافه می‌کنیم (خطوط ۳۰-۳۳). حال نوبت به اضافه کردن دو پنل که یکی از آن‌ها حاوی کارت‌ها و دیگری حاوی دکمه‌ها است به Frame می‌رسد. این دو پنل را در خطوط ۳۶ و ۳۷ به فرم اضافه می‌کنیم. چون layout manager فرم به صورت پیشفرض BorderLayout است می‌توانیم یکی از پنل‌ها را در وسط و دیگری را در پایین فرم قرار دهیم. در خطوط ۴۹-۵۳ هم دو کنترل کننده رویداد برای دو دکمه ایجاد می‌کنیم تا با کلیک بر روی دکمه‌ها کارت مربوط به آن نمایش داده شود. در خطوط ۴۳ و ۵۱ هم با استفاده از متدهای show() کلاس CardLayout دو کارت را نمایش می‌دهیم. حال برنامه را اجرا و با کلیک بر روی دو دکمه نتیجه را مشاهده نمایید :



FlowLayout

مدیر لایه پیشفرض JPanel است. با استفاده از FlowLayout کنترل‌ها را در یک سطر تراز می‌کنیم. اگر عرض container کمتر از عرض همه کنترل‌های باشد یعنی کنترل‌ها نتوانند در یک سطر قرار بگیرند به صورت خودکار در دو یا چند سطر قرار می‌گیرند. به کد زیر توجه کنید :

```
package myfirstprogram;

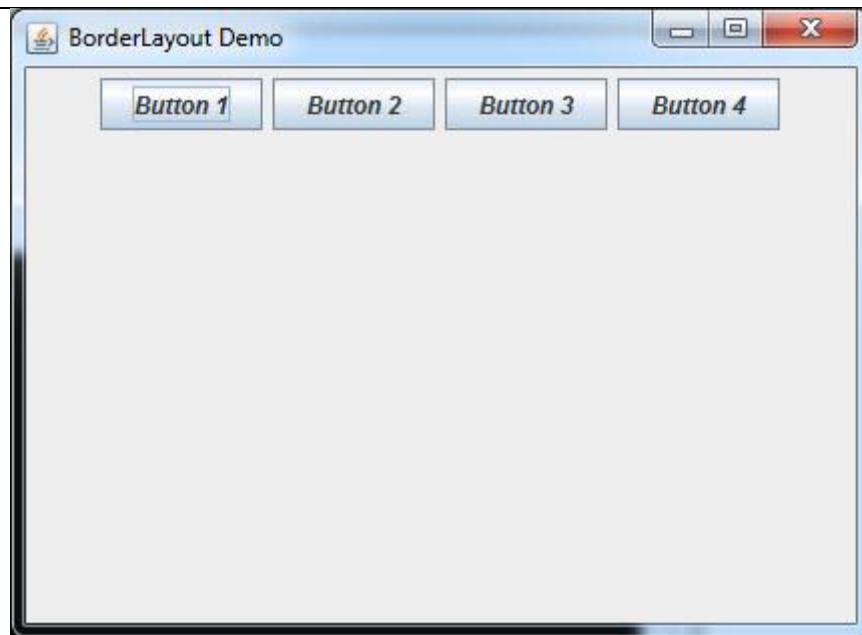
import java.awt.*;
import javax.swing.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        JFrame frame1 = new JFrame("FlowLayout Demo");
        frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame1.setLayout(new FlowLayout());
        frame1.setFont(new Font("Helvetica", Font.PLAIN, 14));

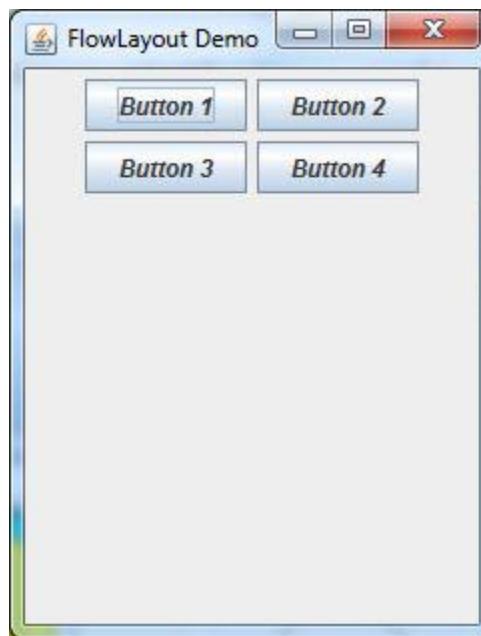
        frame1.add(new JButton("Button 1"));
        frame1.add(new JButton("Button 2"));
        frame1.add(new JButton("Button 3"));
        frame1.add(new JButton("Button 4"));

        frame1.setSize(430, 315);
        frame1.setVisible(true);
    }
}
```

خروجی کد بالا به صورت زیر است :



حال پنجره را کوچک و بزرگ کرده و نتیجه را مشاهده نمایید :



سازنده کلاس FlowLayout را در سه حالت زیر می‌توان مقداردهی کرد :

```
public FlowLayout()  
public FlowLayout(int alignment)
```

```
public FlowLayout(int alignment, int hgap, int vgap)
```

در حالت پیشفرض کنترل‌ها در وسط قرار می‌گیرند مگر اینکه نحوه تراز بندی آن‌ها را با استفاده از ثابت‌های زیر مشخص کنیم :

LEFT •

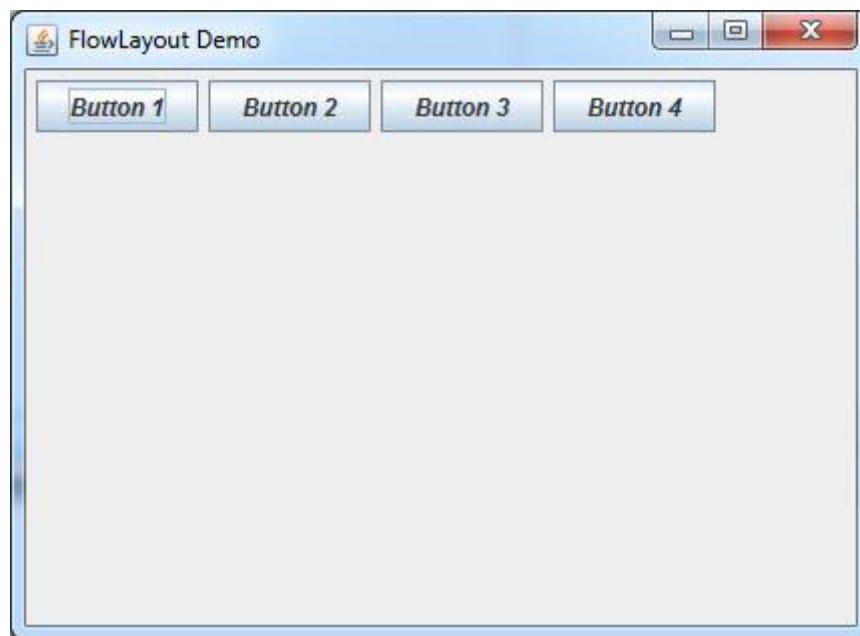
FCENTER •

RIGHT •

حال خط ۱۱ کد بالا را به صورت زیر تغییر دهید :

```
frame1.setLayout(new FlowLayout(FlowLayout.LEFT));
```

با اجرای کد مشاهده می‌کنید که کنترل‌ها نسبه به سمت چپ container تراز شده‌اند :



GridLayout

برای تنظیم کنترل‌ها را در ردیف و ستون‌های هم اندازه و منظم و از چپ به راست در کنار هم قرار می‌دهد. سازنده این کلاس را به سه صورت می‌توان مقداردهی کرد :

```
public GridLayout()
public GridLayout(int rows, int columns)
public GridLayout(int rows, int columns, int hgap, int vgap)
```

مثلاً در سازنده می‌توان تعداد سطر و ستون‌هایی که قرار است کنترل‌ها درون آن‌ها قرار بگیرند را مشخص کنید. به کد زیر توجه کنید:

```
package myfirstprogram;

import java.awt.*;
import javax.swing.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        JFrame frame1 = new JFrame("GridLayout Demo");
        frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame1.setLayout(new GridLayout(4, 3));
        frame1.setFont(new Font("Helvetica", Font.PLAIN, 14));

        frame1.add(new JButton("January"));
        frame1.add(new JButton("February"));
        frame1.add(new JButton("March"));
        frame1.add(new JButton("April"));
        frame1.add(new JButton("May"));
        frame1.add(new JButton("June"));
        frame1.add(new JButton("July"));
        frame1.add(new JButton("August"));
        frame1.add(new JButton("September"));
        frame1.add(new JButton("October"));
        frame1.add(new JButton("November"));
        frame1.add(new JButton("December"));

        frame1.setSize(430, 315);
        frame1.setVisible(true);
    }
}
```

اعداد صفر و دو در خط پررنگ شده کد بالا به ترتیب نشان دهنده سطر و ستون هستند. در کد بالا تعداد ستون رو ۳ گذاشته‌ایم و تعداد

سطر را ۴. کد را اجرا و نتیجه را مشاهده کنید:



اگر تعداد سطرهای و ستون‌ها را برابر ۰ قرار دهید در هنگام اجرای برنامه با خطای `IllegalArgumentException` مواجه می‌شوید.

BoxLayout

در مدل لایه بندی `BoxLayout` کنترل‌ها در یک ردیف یا یک ستون قرار می‌گیرند. کلاس `BoxLayout` دارای یک سازنده است و به صورت

زیر مقداردهی می‌شود :

```
public BoxLayout(Container target, int axis)
```

همان کنترلی است که دیگر کنترل‌ها در آن قرار می‌گیرند و `axis` هم نحوه قرارگیری کنترل در `Container` است که به دو

صورت زیر مقداردهی می‌شود :

`BoxLayout.X_AXIS` •

`BoxLayout.Y_AXIS` •

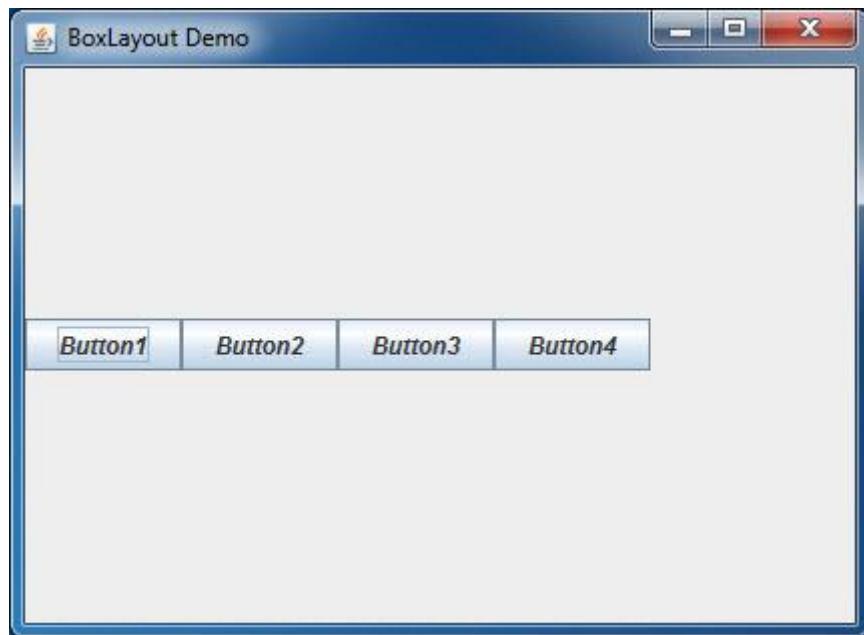
برای درک بهتر عملکرد این کلاس به کد زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class MyFirstProgram
7 {
```

```

8
9
10 JFrame frame1 = new JFrame("BoxLayout Demo");
11 frame1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12 frame1.setFont(new Font("Helvetica", Font.PLAIN, 14));
13
14 Container container = frame1.getContentPane();
15 container.add(new JButton("Button1"));
16 container.add(new JButton("Button2"));
17 container.add(new JButton("Button3"));
18 container.add(new JButton("Button4"));
19
20 frame1.setLayout(new BoxLayout(container, BoxLayout.X_AXIS));
21
22 frame1.setSize(430 ,315);
23 frame1.setVisible(true);
24
25 }
```

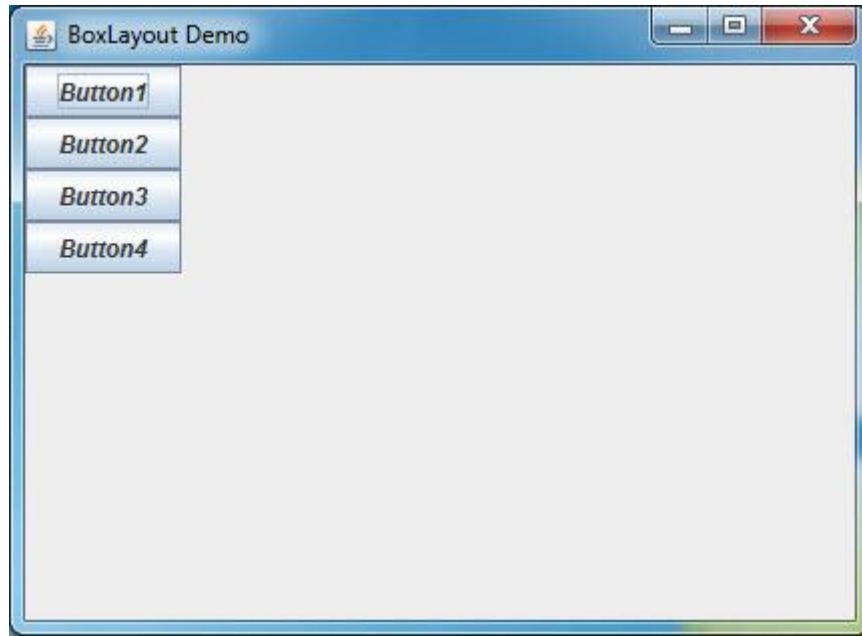
نکته‌ای که در کد بالا وجود دارد این است که چون کنترل‌ها به طور مستقیم به Frame اضافه نمی‌شوند در خط ۱۶ یک Container ایجاد کرده‌ایم که اشاره به لایه ContentPane دارد. سپس در خطوط ۱۵-۱۸ چهار دکمه را به آن اضافه می‌کنیم. سپس در خط ۲۰ نحوه قرارگیری کنترل‌ها در این Container را مشخص می‌کنیم. خروجی کد بالا به صورت زیر است :



حال خط ۲۰ کد بالا را به صورت زیر تغییر دهید :

```
frame1.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
```

یک بار دیگر برنامه را اجرا و نتیجه را مشاهده نمایید :



به این نکته توجه کنید که می‌توان خط ۱۶ کد بالا را کاملاً حذف کرد و برای اضافه کردن دکمه‌ها به Frame به صورت زیر عمل کرد :

```
frame1.getContentPane().add(new JButton("."));
```

و همچنین خط ۲۰ کد بالا را به صورت زیر تغییر داد :

```
frame1.setLayout(new BoxLayout(frame1.getContentPane(), BoxLayout.Y_AXIS));
```

ایجاد حاشیه برای کنترل‌ها

در java کلاس‌هایی برای کار با حاشیه کنترل‌ها وجود دارد. از این کلاس‌ها می‌توان برای کشیدن خط به دور کنترل‌ها، پر کردن فضای خالی اطراف آن‌ها و همچنین اضافه کردن یک عنوان به کنترل‌ها استفاده کرد. این کلاس‌ها در پکیج border قرار دارند و برای کار با آن‌ها باید این import را Package کرد :

```
import javax.swing.border.*
```

در زیر لیست برخی از این کلاس‌های پکیج border ذکر شده است :

- BevelBorder

- CompoundBorder

-
- EtchedBorder •
 - LineBorder •
 - MatteBorder •
 - SoftBevelBorder •
 - TitledBorder •

برای ایجاد حاشیه با استفاده از کلاس‌های بالا یک شیء از آن‌ها را به عنوان آرگومان به متده استفاده از `setBorder()` ارسال می‌کنیم. به کد زیر

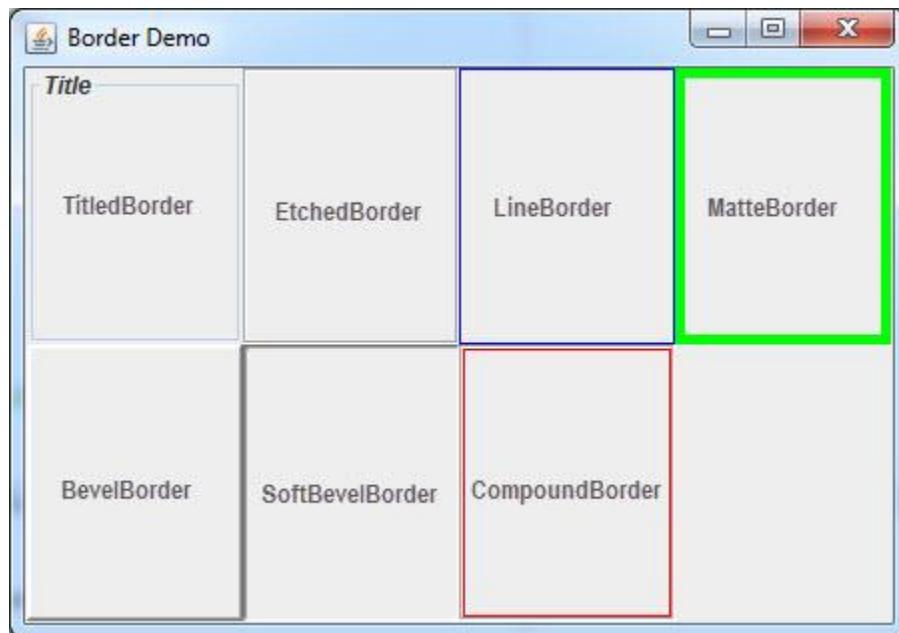
توجه کنید :

```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.border.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        JFrame frame1 = new JFrame("Border Demo");
12        frame1.setLayout(new GridLayout(2, 4));
13        frame1.setSize(450, 315);
14
15        JPanel Panel1 = new JPanel();
16        Panel1.setBorder(new TitledBorder("Title"));
17
18        JPanel Panel2 = new JPanel();
19        Panel2.setBorder(new EtchedBorder());
20
21        JPanel Panel3 = new JPanel();
22        Panel3.setBorder(new LineBorder(Color.BLUE));
23
24        JPanel Panel4 = new JPanel();
25        Panel4.setBorder(new MatteBorder(5, 5, 5, 5, Color.GREEN));
26
27        JPanel Panel5 = new JPanel();
28        Panel5.setBorder(new BevelBorder(BevelBorder.RAISED));
29
30        JPanel Panel6 = new JPanel();
31        Panel6.setBorder(new SoftBevelBorder(BevelBorder.LOWERED));
32
33        JPanel Panel7 = new JPanel();
34        Panel7.setBorder(new CompoundBorder(new EtchedBorder(),new LineBorder(Color.RED)));
35
36        frame1.add(Panel1);
37        frame1.add(Panel2);
38        frame1.add(Panel3);
39        frame1.add(Panel4);
40        frame1.add(Panel5);
41        frame1.add(Panel6);
42        frame1.add(Panel7);
43
44        frame1.setVisible(true);
45    }
46 }
```

در کد و برای آشنایی شما با نحوه استفاده از کلاس‌های مذکور ۷ پنل را حاشیه دار کردہ‌ایم و در ادامه در مورد آن‌ها بیشتر توضیح می‌دهیم.

خروجی کد به صورت زیر است :



کلاس TitleBorder

از کلاس `TitleBorder` برای کشیدن خط به دور کنترل به همراه یک عنوان استفاده می‌شود. سازنده این کلاس به صورت زیر مقداردهی می‌شود :

```
TitledBorder(Border border)
TitledBorder(String string)
TitledBorder(Border border, String string)
TitledBorder(Border border, String string, titleJustification, titlePosition)
TitledBorder(Border border, String string, titleJustification, titlePosition, Font font)
TitledBorder(Border border, String string, titleJustification, titlePosition, Font font,
Color.yellow)
```

نوع حاشیه کنترل را مشخص می‌کند. `titleJustification`, چپ چین، راست چین یا وسط چین بودن عنوان را مشخص می‌کند و مقادیر زیر را می‌پذیرد :

LEFT •

CENTER •

RIGHT •

: titlePosition بالا یا پایین قرار گرفتن عنوان را مشخص می‌کند و مقادیر زیر را می‌پذیرد :

TOP	•
ABOVE_TOP	•
BELOW_TOP	•
BOTTOM	•
ABOVE_BOTTOM	•
BELOW_BOTTOM	•

نوع و اندازه فونت عنوان و Color رنگ آن را مشخص می‌کند. برای درک بهتر استفاده از کلاس فوق و ثابت‌های آن به مثال زیر توجه کنید :

```

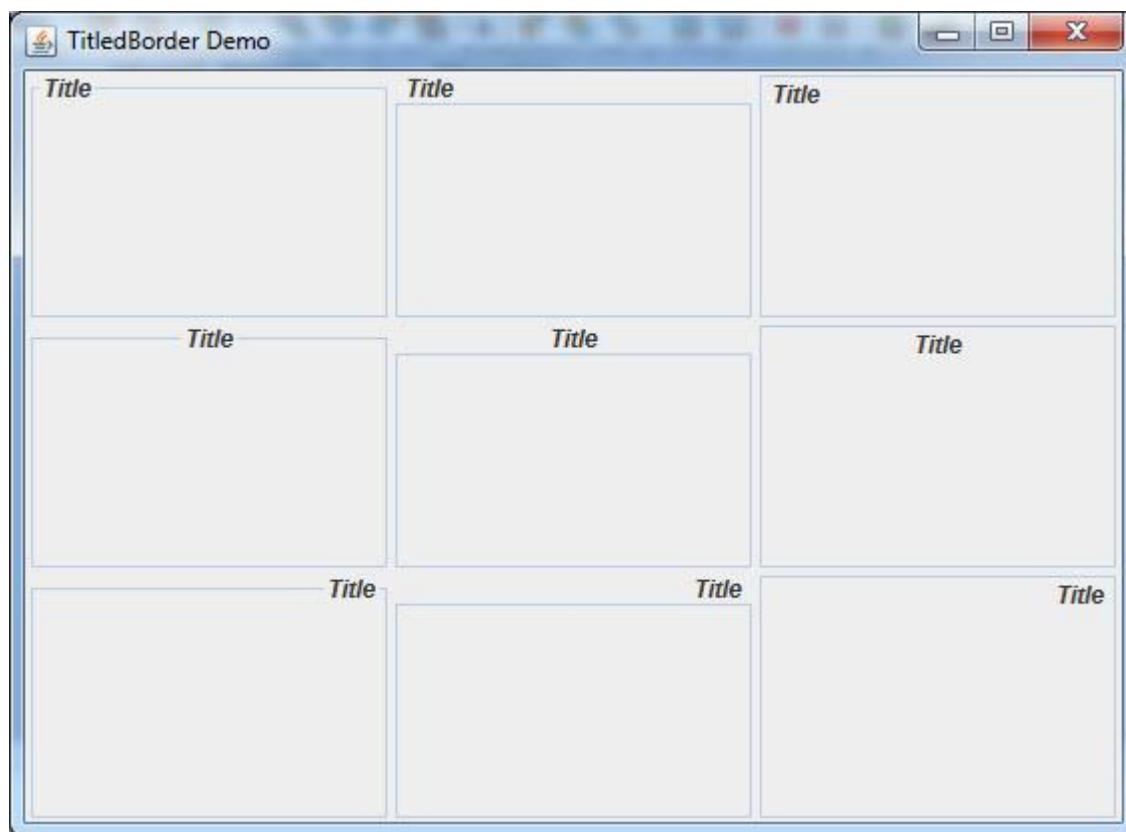
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.border.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        JFrame frame1 = new JFrame("TitledBorder Demo");
12        frame1.setLayout(new GridLayout(3, 3));
13        frame1.setSize(600 , 450);
14
15        JPanel Panel1 = new JPanel();
16        Panel1.setBorder(new TitledBorder
17                         (null, "Title", TitledBorder.LEFT, TitledBorder.TOP));
18        JPanel Panel2 = new JPanel();
19        Panel2.setBorder(new TitledBorder
20                         (null, "Title", TitledBorder.LEFT, TitledBorder.ABOVE_TOP));
21        JPanel Panel3 = new JPanel();
22        Panel3.setBorder(new TitledBorder
23                         (null, "Title", TitledBorder.LEFT, TitledBorder.BELOW_TOP));
24        JPanel Panel4 = new JPanel();
25        Panel4.setBorder(new TitledBorder
26                         (null, "Title", TitledBorder.CENTER, TitledBorder.TOP));
27        JPanel Panel5 = new JPanel();
28        Panel5.setBorder(new TitledBorder
29                         (null, "Title", TitledBorder.CENTER, TitledBorder.ABOVE_TOP));
30        JPanel Panel6 = new JPanel();
31        Panel6.setBorder(new TitledBorder
32                         (null, "Title", TitledBorder.CENTER, TitledBorder.BELOW_TOP));
33        JPanel Panel7 = new JPanel();
34        Panel7.setBorder(new TitledBorder
35                         (null, "Title", TitledBorder.RIGHT, TitledBorder.TOP));

```

```

36     JPanel Panel18 = new JPanel();
37     Panel18.setBorder(new TitledBorder(
38             null, "Title", TitledBorder.RIGHT, TitledBorder.ABOVE_TOP));
39     JPanel Panel19 = new JPanel();
40     Panel19.setBorder(new TitledBorder(
41             null, "Title", TitledBorder.RIGHT, TitledBorder.BELOW_TOP));
42
43     frame1.add(Panel1);
44     frame1.add(Panel2);
45     frame1.add(Panel3);
46     frame1.add(Panel4);
47     frame1.add(Panel5);
48     frame1.add(Panel6);
49     frame1.add(Panel7);
50     frame1.add(Panel8);
51     frame1.add(Panel9);
52
53     frame1.setVisible(true);
54 }
55 }
```

در کد بالا عنوان ۹ پنل را به حالت مختلف مشخص کردہ‌ایم. خروجی کد بالا به صورت زیر است :



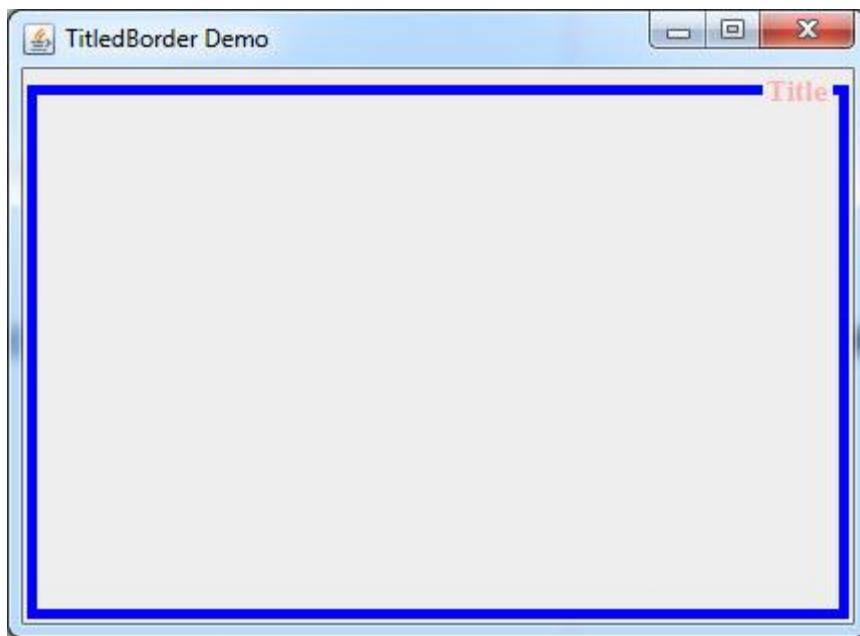
با توجه به شکل متوجه می‌شوید که اگر بخواهید مثلاً عنوان شما در بالای پنل و سمت راست آن نمایش داده شود باید از کد زیر استفاده

کنید :

```
new TitledBorder(null, "Title", TitledBorder.RIGHT, TitledBorder.TOP);
```

کلمه‌های TOP در کد ابتدای درس را به BOTTOM تغییر داده و برنامه را اجرا و نتیجه را مشاهده کنید. رنگ و نوع فونت عنوان و حاشیه کنترل را هم می‌توان مشخص کرد :

```
Panel1.setBorder(new TitledBorder(new MatteBorder(5, 5, 5, 5, Color.BLUE),
    "Title", TitledBorder.RIGHT, TitledBorder.TOP, new Font("Serif", Font.BOLD,
    15), Color.PINK));
```



کلاس MatteBorder

با استفاده از کلاس MatteBorder هم می‌توان یک خط به دور کنترل کشید و هم به جای خط می‌توان از آیکون استفاده کرد. به این نکته توجه کنید که قسمتی از فضای اصلی کنترل به وسیله خط یا آیکون اشغال می‌شود. سازنده این کلاس به روش‌های زیر مقدار دهی می‌شود :

```
MatteBorder(Icon)
MatteBorderInset, Color)
MatteBorderInset, Icon)
```

```
MatteBorder(top, left, bottom, right, Color)
MatteBorder(top, left, bottom, right, Icon)
```

برای درک عملکرد این کلاس به کد زیر توجه کنید :

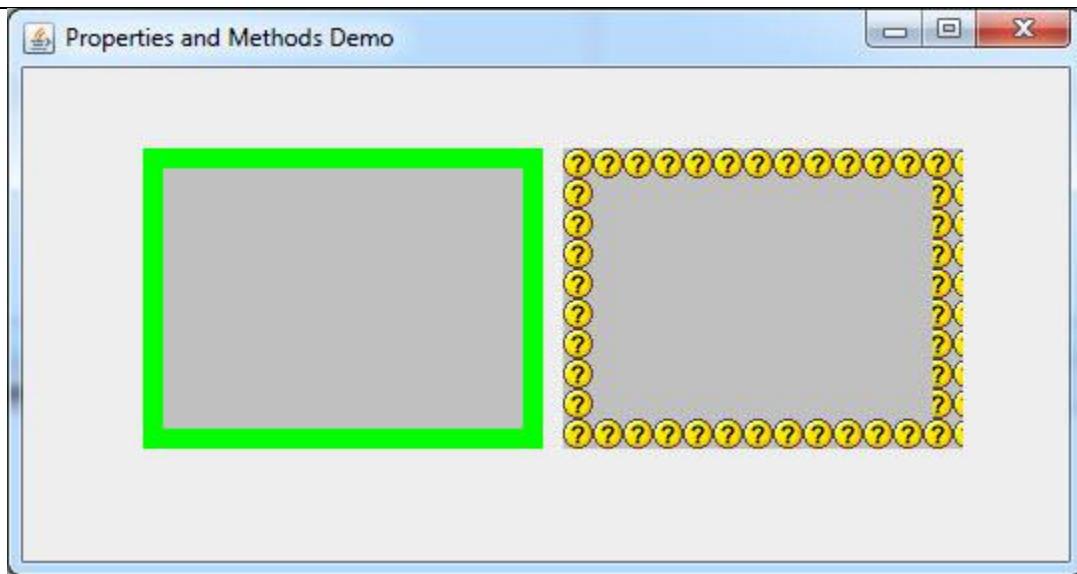
```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.border.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        JFrame frame1 = new JFrame("Properties and Methods Demo");
12        frame1.setLayout(null);
13        JPanel panel1 = new JPanel();
14        JPanel panel2 = new JPanel();
15
16        panel1.setBounds(60, 40, 200, 150);
17        panel1.setBackground(Color.LIGHT_GRAY);
18
19        panel2.setBounds(270, 40, 200, 150);
20        panel2.setBackground(Color.LIGHT_GRAY);
21
22        frame1.add(panel1);
23        frame1.add(panel2);
24
25        panel1.setBorder(new MatteBorder(10, 10, 10, 10, Color.GREEN));
26
27        panel2.setBorder(new MatteBorder(new ImageIcon("C:\\FrameDemo\\icon.gif")));
28
29        frame1.setSize(540 , 284);
30        frame1.setVisible(true);
31    }
32 }
```

در خطوط ۱۱-۲۳ کد بالا یک Frame ایجاد کرده و دو Panel با اندازه و پس زمینه یکسان به آن اضافه کرده‌ایم. مهم‌ترین خطوط کد بالا

خطوط ۲۵ و ۲۷ می‌باشند که در خط ۲۵ یک خط سبز با پهنای ۱۰ پیکسل به دور Panel اولی کشیده‌ایم. در خط ۲۷ هم به جای خط از

آیکون استفاده کرده‌ایم. آیکون را من از قبل در پوشه برنامه قرار داده‌ام. برنامه را اجرا و نتیجه را مشاهده کنید :



JButton

وقتی که بر روی کنترل `Button` کلیک می‌کنیم دستوراتی اجرا می‌شود، و این معمول‌ترین استفاده‌ای است که از این کنترل می‌شود. دکمه‌ها به نوعی برای تأیید یا لغو یک عمل و یا باز کردن کادرهای محاوره‌ای به کار می‌روند. سازنده این کلاس به صورت‌های زیر مقداردهی

می‌شود :

```
JButton(Action);
JButton(String);
JButton(Icon);
JButton(String, Icon);
```

کنترل `Button` دارای خواصی است که در جدول زیر تعدادی از این آن‌ها ذکر شده است :

خواص	توضیح
<code>enabled</code>	اگر مقدار این خاصیت <code>false</code> باشد بر روی <code>button</code> نمی‌توان کلیک کرد.
<code>text</code>	عنوانی است که به کنترل اختصاص داده می‌شود.
<code>visible</code>	مشخص می‌کند که آیا کنترل <code>button</code> در روی فرم قابل رویت باشد یا نه

فقط با ویرایش خواص یک button نمی‌توان از آن استفاده کرد. بلکه کنترل دکمه برای انجام برخی اعمال نیاز به واکنش به رویدادها

دارد. در زیر معمول ترین رویدادهای کنترل button آمده است.

رویداد	توضیح
actionPerformed	وقتی روی می‌دهد که بر روی دکمه کلیک کنید.
mouseExited	وقتی روی می‌دهد که ماوس کنترل را ترک کند.
stateChanged	وقتی روی می‌دهد که مکان دکمه تغییر کند.
mousePressed	وقتی روی می‌دهد که نشانگر ماوس بر روی کنترل باشد و دکمه ماوس رو به پایین فشار داده شود.
mouseEntered	وقتی روی می‌دهد که ماوس وارد کنترل دکمه می‌شود.
mouseReleased	وقتی روی می‌دهد که دکمه ماوس فشار داده شده و رها شود.

همانطور که قبلاً مشاهده کردید رویداد پیشفرض کنترل دکمه رویداد actionPerformed می‌باشد. اجازه دهید که با ایجاد برنامه‌هایی با رویدادهای دیگر نیز آشنا شویم. می‌خواهیم برنامه‌ای بنویسیم که در آن وقتی که با ماوس بر روی یک دکمه رفته‌یم و یا نشانگر ماوس را از آن خارج کردیم متن داخل دکمه تغییر کند. به کد زیر توجه کنید:

```

1 package myfirstprogram;
2
3 import javax.swing.*;
4 import java.awt.event.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         JFrame frame1 = new JFrame("Button Demo");
11         frame1.setLayout(null);
12         final JButton button1 = new JButton("buttonSample");
13
14         button1.setBounds(85, 80, 157, 60);
15
16         MouseAdapter MouseAdapter1 = new MouseAdapter()
17         {
18             @Override
19             public void mouseEntered(MouseEvent evt)
20             {
21                 button1.setText("Mouse has entered!");
22             }
23         }
24     }
25 }
```

```

22
23
24
25     @Override
26     public void mouseExited(MouseEvent evt)
27     {
28         button1.setText("Mouse has Left!");
29     }
30
31     button1.addMouseListener(MouseAdapter1);
32
33     frame1.add(button1);
34
35     frame1.setSize(356 , 263);
36     frame1.setVisible(true);
37
38 }
```

در خط ۱۰ و ۱۱ یک فریم ایجاد کرده و هیچ لایه بندی برای آن در نظر نمی‌گیریم. در خط ۱۲ یک دکمه ایجاد کرده و در خط ۱۴ آن را نسبت به گوشه بالا و سمت چپ فریم تراز می‌کنیم. حال نوبت به قسمت مهم برنامه می‌رسد. همانطور که می‌دانید برای اینکه رویدادی اتفاق بیفتد باید یک Interface توسط یک کلاس پیاده سازی شود. کلاسی که قرار است رابط را پیاده سازی کند کلاس MouseAdapter است که سه رابط MouseMotionListener، MouseListener و MouseWheelListener را پیاده سازی می‌کند. در نتیجه این پیاده سازی باید کدهایی برای بدنه متدهای این رابطها هم در نظر گرفته شود. چون ما با دو متدهای MouseEntered() و MouseExited() در خطوط ۱۶ و ۱۷ ایجاد کرده و کار داریم پس باید کدهایی بدنه این دو متدهای فراهم کنیم. پس در کل یک شیء از کلاس MouseAdapter در خطوط ۲۱ و ۲۲ متن داخل دکمه را تغییر می‌دهیم. حال می‌خواهیم این شیء را در دو متدهای MouseEntered() و MouseExited() در خطوط ۲۳ و ۲۴ اجرا کنیم. این کار را با استفاده از متد addMouseListener() در خط ۲۵ انجام می‌دهیم. در خط ۲۶ هم دکمه را به فریم اضافه می‌کنیم. حال برنامه را اجرا کنید. مشاهده می‌کنید که با قرار گرفت اشاره گر ماوس بر روی کنترل button خاصیت text آن تغییر می‌کند. و از طرف دیگر وقتی که اشاره گر از کنترل دکمه دور می‌شود این خاصیت دوباره تغییر می‌کند.



کنترل JLabel

کنترل `JLabel` برای اضافه کردن یک متن به فرم بکار می‌رود. این متن می‌تواند حاوی یک پیام و یا توضیحی درباره عملکرد سایر کنترل‌ها باشد. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
JLabel(Icon)
JLabel(String)
JLabel(Icon, horizontalAlignment)
JLabel(String, horizontalAlignment)
JLabel(String, Icon, horizontalAlignment)
```

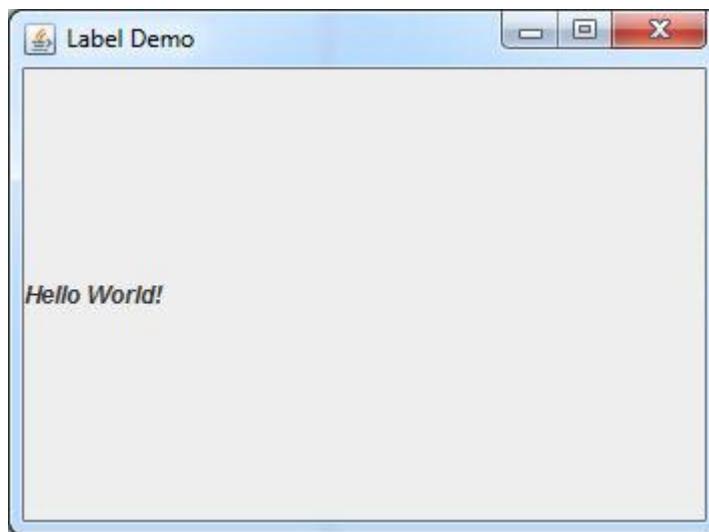
خاصیت `text` مهم‌ترین خاصیت آن می‌باشد چون هدف اصلی از استفاده از این کنترل نشان دادن متن در فرم است. این متن را هم

می‌توانیم در سازنده فرم بنویسیم :

```
JLabel Label1 = new JLabel("Hello World!");
```

و هم با استفاده از متد `setText()` به آن اختصاص دهیم :

```
JLabel Label1 = new JLabel();
Label1.setText("Hello World!");
```



برای ترازنی `Label` هم می‌توان از رابط `SwingConstants` به صورت زیر استفاده کرد :

```
JLabel Label1 = new JLabel("Hello World!", SwingConstants.CENTER);
```

و یا به صورت

```
JLabel Label1 = new JLabel();
Label1.setText("Hello World!");
Label1.setHorizontalAlignment(SwingConstants.CENTER);
```



رویدادهایی هم برای این کنترل وجود دارد که اکثر موقعیت آن نیازی نیست و در نتیجه در مورد آنها بحث نمی‌کنیم.

کنترل JTextField و JPasswordField

کنترل TextField ابتدایی‌ترین وسیله برای ورود اطلاعات در یک فرم ویندوزی می‌باشد. این کار را با تایپ آن‌ها (اطلاعات) در

TextField انجام می‌دهید. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
JTextField()
JTextField(String)
JTextField(Column)
JTextField(String,Column)
JTextField(Document,String,Column)
```

می‌خواهیم با یک مثال نحوه استفاده از TextField را به شما نشان دهیم. برنامه از شما می‌خواهد که دو عدد را وارد کنید و با زدن

دکمه جمع آن‌ها در TextField سومی نشان دهید. حال به طراحی برنامه می‌پردازیم. به کد زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import javax.swing.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        //Create Frame
12        JFrame frame1 = new JFrame("TextFiled Demo");
13        frame1.setLayout(null);
```

```

15     //Create Panel
16     JPanel panel1 = new JPanel();
17     panel1.setLayout(new GridLayout(4,2));
18     panel1.setBounds(40, 60, 270, 80);
19
20     //Create Textfield and Button
21     final JTextField firstNumber = new JTextField(15);
22     final JTextField SecondNumber = new JTextField(15);
23     final JTextField Result = new JTextField(15);
24     JButton Calculate = new JButton("Calculate");
25
26     //Add Textfield and Button to Panel
27     panel1.add(new Label("First Number"));
28     panel1.add(firstNumber);
29     panel1.add(new Label("Second Number"));
30     panel1.add(SecondNumber);
31     panel1.add(new Label("SUM"));
32     panel1.add(Result);
33     panel1.add(Calculate);
34
35     //Add Panel to Frame
36     frame1.add(panel1);
37
38     class MyActionListener implements ActionListener
39     {
40         @Override
41         public void actionPerformed(ActionEvent e)
42         {
43             double firstnumber = Double.parseDouble(firstNumber.getText());
44             double secondnumber = Double.parseDouble(SecondNumber.getText());
45             Result.setText(String.valueOf(firstnumber + secondnumber));
46         }
47     }
48
49     Calculate.addActionListener(new MyActionListener());
50
51     //Show Frame
52     frame1.setSize(356, 263);
53     frame1.setVisible(true);
54 }
55 }
```

در کد بالا و در خط ۱۶ یک Frame ایجاد کردہ‌ایم. در خط ۱۷ یک Panel ایجاد و در خط ۱۸ آن را به چهار سطر و دو ستون تقسیم نموده‌ایم.

در خط ۱۸ پنل را تقریباً به وسط فریم انتقال داده‌ایم. در خطوط ۲۱-۳۳ هم TextField ها و دکمه محاسبه را ایجاد و به Panel اضافه کردہ‌ایم.

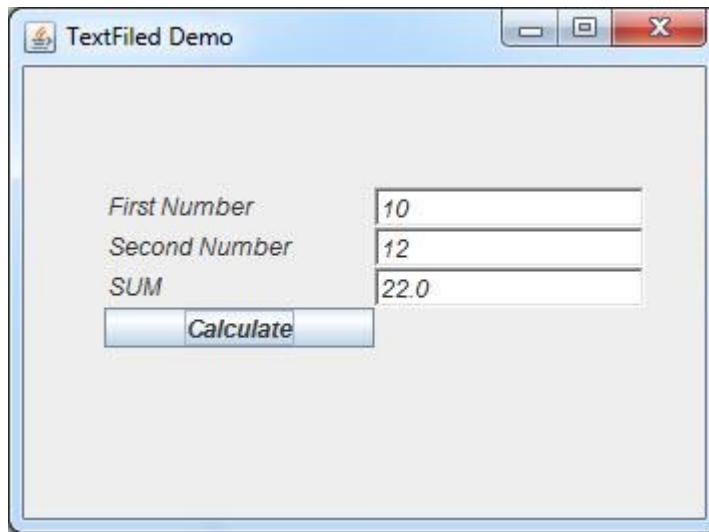
بخس مهم کد بالا خطوط ۳۸-۴۹ می‌باشد. ابتدا در خطوط ۳۸-۴۷ یک کلاس به نام MyActionListener تعریف کردہ‌ایم که

رابطه ایجاد شده سازی می‌کند. این رابطه دارای متodi به نام () actionPerformed() که زمانی فراخوانی می‌شود که بر

روی کنترلی کلیک شود. حال وظیفه کلاس MyActionListener فراهم کردن کدھایی برای بدنے این متod می‌باشد. این کدھا که در

خطوط ۴۳-۴۵ مشاهده می‌کنید همان عملیان جمع TextField ها می‌باشند. در خطوط ۴۳ و ۴۴ با استفاده از متod () getText()

ورودی کاربر را دریافت کرده و برای اینکه بتوان بر روی آن‌ها عملیات جمع را انجام داد با استفاده از متد `parseDouble()` از کلاس `Double` آن‌ها را به نوع اعشار تبدیل می‌کنیم. چون قرار است که نتیجه در `TextField` نمایش داده شود در خط ۴۵ و با استفاده از متد `String.valueOf()` حاصل جمع را به نوع رشته تبدیل و در `TextField` سوم نمایش می‌دهیم. در خط ۴۹ هم شیی از کلاس `MyActionListener` را به متد `addActionListener()` ارسال می‌کنیم تا دکمه درست کار کند. حال برنامه را اجرا و با وارد کردن دو عدد نتیجه را مشاهده کنید :



در ادامه آموزش کنترل `JTextField` می‌خواهیم رویداد `TextChanged` زبان سی شارپ را به وسیله جاوا پیاده سازی کنیم. برنامه به این صورت است که وقتی متنی در داخل `JTextField` نوشته می‌شود، همزمان در داخل یک کنترل `JLabel` هم نوشته شود. به کد زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        JFrame frame1 = new JFrame("TextFiled Demo");
12        frame1.setLayout(null);
13        JPanel panel1 = new JPanel();
14        panel1.setLayout(new GridLayout(2,1));
15        panel1.setBounds(40, 80, 270, 50);
16        final JTextField TextField1 = new JTextField();

```

```

17 final JLabel Label1 = new JLabel();
18 panel1.add(TextField1);
19 panel1.add(Label1);
20 frame1.add(panel1);
21
22 class MydocumentListener implements DocumentListener
23 {
24     public void changedUpdate(DocumentEvent documentEvent)
25     {
26     }
27     public void insertUpdate(DocumentEvent documentEvent)
28     {
29         Label1.setText(TextField1.getText());
30     }
31     public void removeUpdate(DocumentEvent documentEvent)
32     {
33         Label1.setText(TextField1.getText());
34     }
35 }
36
37 TextField1.getDocument().addDocumentListener(new MydocumentListener());
38
39 frame1.setSize(356 , 263);
40 frame1.setVisible(true);
41
42 }
43 }
```

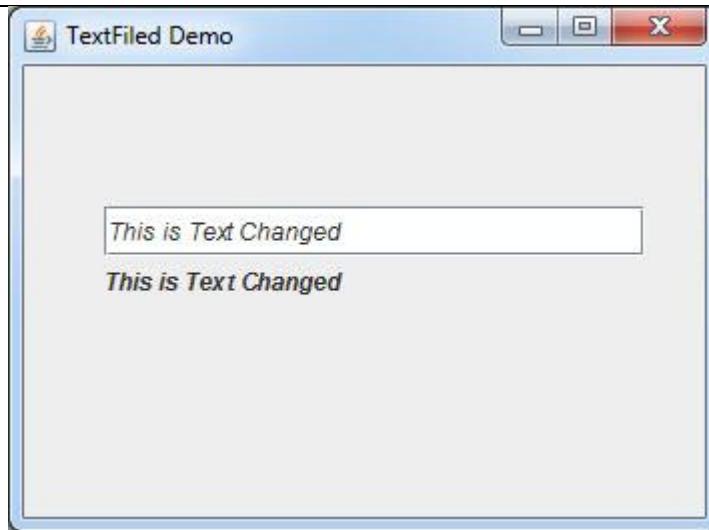
در خطوط ۱۱-۲۰ یک فریم ایجاد و یک کنترل Panel که حاوی یک TextField و یک Label است را به آن اضافه کردہ‌ایم. برای کار با متن یا به عبارتی تغییرات متن باید رابط DocumentListener توسط یک کلاس پیاده سازی شود که ما این کار را در خط ۲۲ انجام داده‌ایم. این رابط دارای سه متد زیر می‌باشد :

- changedUpdate()
- insertUpdate()
- removeUpdate()

که ما برای پیاده سازی رویداد مذکور به دو متد آخر نیاز داریم. در داخل بدنه دو متد آخر کد زیر را می‌نویسیم :

```
Label1.setText(TextField1.getText());
```

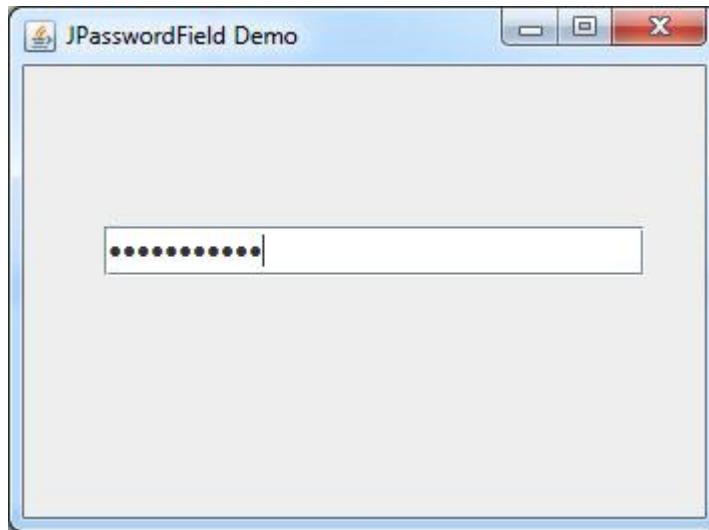
کاری که در خطوط ۳۴ و ۳۰ انجام داده‌ایم. سپس یک نمونه از کلاس MydocumentListener را در خط ۳۸ به متد addDocumentListener() ارسال می‌کنیم. حال برنامه را اجرا و متنی را در داخل TextField نوشته و نتیجه را مشاهده نمایید :



اگر شما می‌خواهد یک پسورد را بپذیرد، شما باید از کنترل JTextField یا JPasswordField به جای استفاده کنید. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JPasswordField()
public JPasswordField(String text)
public JPasswordField(int columnWidth)
public JPasswordField(String text, int columnWidth)
public JPasswordField(Document model, String text, int columnWidth)
```

هر کاراکتری را که در داخل این کنترل بنویسید آن را به صورت ستاره (*) نمایش می‌دهد :



JTextArea

کنترل `JTextArea` را به نوعی می‌توان یک `JTextField` در نظر گرفت با این تفاوت که در `JTextField` ورودی‌های کاربر در یک خط `text` نوشته می‌شوند در حالیکه در `JTextArea` می‌توان ورودی‌ها را در چند خط نوشت. `JTextArea` دارای یک خاصیت و آن هم `text` می‌باشد. در این کنترل به صورت توکار دارای اسکرول نیست و اگر متن وارد شده توسط کاربر زیاد باشد برای مشاهده تمام متن باید از کنترل `JScrollPane` استفاده کرد. سازنده این کلاس به روش‌های زیر مقدار دهی می‌شود :

```
public JTextArea()
public JTextArea(Document document)
public JTextArea(String text)
public JTextArea(int rows, int columns)
public JTextArea(String text, int rows, int columns)
public JTextArea(Document document, String text, int rows, int columns)
```

این کنترل دارای خاصیت‌های مهم زیر می‌باشد :

خاصیت	توضیح
Columns	تعداد ستون‌های Textarea را مشخص می‌کند
lineCount	تعداد خطوط Textarea را مشخص می‌کند
lineWrap	مشخص می‌کند که زمانیکه به آخر یک سطر در Textarea رسیدیم بقیه متن در سطر بعدی نوشته شود یا نه
rows	تعداد سطرهای Textarea را مشخص می‌کند
tabSize	مشخص می‌کند که با هر بار زدن دکمه Tab نشانگر ماوس چند کاراکتر به جلو رانده شود.

متدهای مهم این کنترل هم عبارت‌اند از :

متدها	توضیح
<code>setSelectedTextColor()</code>	رنگ متن انتخاب شده را مشخص می‌کند.
<code>setSelectionColor()</code>	رنگ پس زمینه متن انتخاب شده را مشخص می‌کند.

قسمتی از متن `Textarea` را انتخاب می‌کند. این متدها دو عدد می‌گیرد که یکی نشان دهنده نقطه شروع و دیگری نشان دهنده نقطه پایان متن انتخابی است.

`select()`

بیشتر متدهای کنترل `Textarea` برای متن انتخاب شده استفاده می‌شوند. برای مثال، متدهای `select()` و `setSelectionStart()` و `setSelectionEnd()` را می‌توان استفاده کرد.

شده است را مشخص می‌کند :

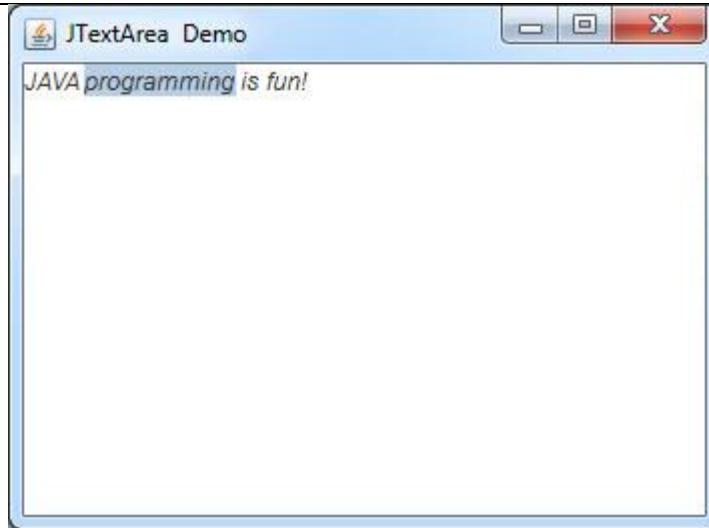
```
select(selectionStart, selectionEnd)
```

به کد زیر توجه کنید :

```

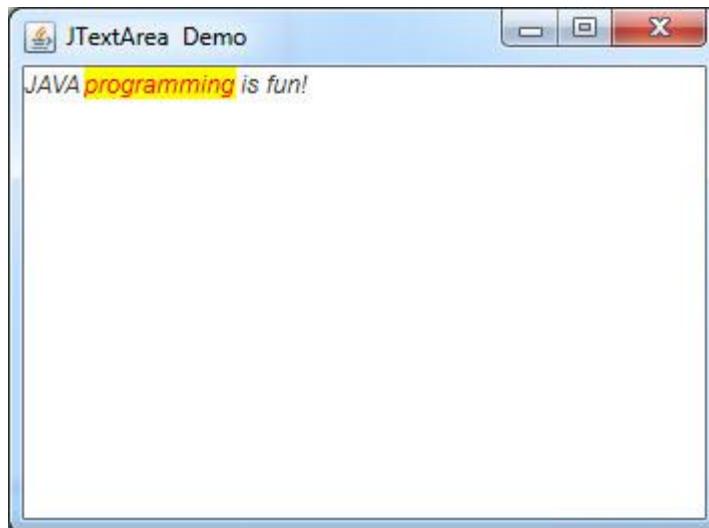
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         JFrame frame1 = new JFrame("JTextArea Demo");
11
12         JTextArea TextArea1 = new JTextArea("JAVA programming is fun!");
13
14         TextArea1.select(5, 16);
15
16         JPanel Panel1 = new JPanel();
17         Panel1.setLayout(new BorderLayout());
18         Panel1.add(TextArea1, BorderLayout.CENTER);
19         frame1.add(Panel1);
20         frame1.setSize(356, 263);
21         frame1.setVisible(true);
22     }
23 }
```

در خط ۱۲ کد بالا یک متن پیشفرض به `JTextArea` اضافه و در خط ۱۴ از اندیس ۵ تا اندیس ۱۶ آن را انتخاب کرده‌ایم. برنامه را اجرا کنید :



وقتی ما یک متن را انتخاب می‌کنیم، می‌توانیم متن انتخاب شده را به وسیله‌ی تعدادی متده استفاده کرد که بر روی متن انتخاب شده کار می‌کنند. مثلاً برای تغییر رنگ متن انتخاب شده از متده setSelectionColor() استفاده کنید. همچنین می‌توانید برای تغییر رنگ پس زمینه‌ی آن از متده setSelectionColor() استفاده کنید. بعد از خط ۱۴ کد بالا کدهای زیر را نوشته و برنامه را اجرا کنید :

```
TextArea1.setSelectedTextColor(Color.RED);
TextArea1.setSelectionColor(Color.YELLOW);
```



کنترل JRadioButton

کنترل JRadioButton یا دکمه رادیویی دکمه‌ای است که دارای دو حالت خاموش و روشن می‌باشد. دکمه‌ی JRadioButton یک دکمه‌ی دایره‌ای شکل به همراه یک برچسب است. شما با کلیک کردن بر روی دکمه‌ی JRadioButton می‌توانید آنرا از حالت خاموش به روشن و یا بلعکس تغییر دهید. وقتی که یک دکمه‌ی JRadioButton روشن باشد، یک نقطه در وسط آن قرار می‌گیرد، و زمانی که خاموش باشد، دایره‌ی آن خالی است.

دکمه‌های رادیویی معمولاً زمانی استفاده می‌شوند که یک کاربر می‌بایست از بین چند گزینه یکی از آن‌ها را انتخاب کند. برای مثال: زمانی که شما بخواهید جنسیت کاربر را مشخص کنید، می‌توانید از دو دکمه رادیویی با نام‌های مرد و زن استفاده کنید. وقتی که شما از دکمه‌های رادیویی استفاده کردید، فقط می‌توانید یکی از آن دو را انتخاب کنید. سازنده کلاس JRadioButton به روش‌هایی زیر مقداردهی می‌شود :

```
public JRadioButton()
public JRadioButton(Icon icon)
public JRadioButton(Icon icon, boolean selected)
public JRadioButton(String text)
public JRadioButton(String text, boolean selected)
public JRadioButton(String text, Icon icon)
public JRadioButton(String text, Icon icon, boolean selected)
public JRadioButton(Action action)
```

برای آشنایی با کاربرد دکمه‌های رادیویی به مثال زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        final JFrame frame1 = new JFrame("JRadioButton Demo");
12
13        JPanel Panel1 = new JPanel();
14        Panel1.setLayout(new GridLayout(3,1));
15
16        final JRadioButton radioButtonYes = new JRadioButton("Yes");
17        final JRadioButton radioButtonNo = new JRadioButton("NO");
18        JButton buttonShow = new JButton("Show Choice");
19
20        ButtonGroup ButtonGroup1 = new ButtonGroup();
21        ButtonGroup1.add(radioButtonYes);
```

```

22     ButtonGroup1.add(radioButtonNo);
23
24     Panel1.add(radioButtonYes);
25     Panel1.add(radioButtonNo);
26     Panel1.add(buttonShow);
27
28     frame1.add(Panel1);
29
30     class SelectedRadio implements ActionListener
31     {
32         @Override
33         public void actionPerformed(ActionEvent e)
34         {
35             if (radioButtonYes.isSelected())
36             {
37                 JOptionPane.showMessageDialog(frame1, "You choosed Yes!");
38             }
39             else if (radioButtonNo.isSelected())
40             {
41                 JOptionPane.showMessageDialog(frame1, "You choosed No!");
42             }
43         }
44     }
45
46     buttonShow.addActionListener(new SelectedRadio());
47
48     frame1.setSize(356 , 263);
49     frame1.setVisible(true);
50
51 }
```

هدف از کد بالا این است که با کلیک بر روی یک دکمه یک پیام مبنی بر انتخاب یک دکمه به کاربر نمایش داده شود. در خطوط ۱۱-۱۴ یک `Panel` و یک `Frame` و در خطوط ۱۶-۱۸ دو عدد دکمه‌ی رادیویی با نام‌های `radioButtonYes` و `radioButtonNo` و یک کنترل `buttonShow` ایجاد کردند. این دکمه‌ها را در خطوط ۲۴-۲۶ به `Panel` اضافه می‌کنیم. در جاوا به طور همزمان می‌توان دو دکمه رادیویی را در حالت انتخاب قرار داد، در حالیکه انتظاری که ما از دکمه‌های رادیویی داریم این است که همزمان یکی از آن‌ها قابل انتخاب باشد. برای رفع این مشکل در جاوا کلاسی به نام `ButtonGroup` وجود دارد که با اضافه کردن دکمه‌ها به یک شیء از این کلاس (خطوط ۳۰-۴۴) این امکان را به وجود می‌آوریم که همزمان یکی از دکمه‌ها انتخاب شود. برای اداره رویداد کلیک دکمه در خطوط ۳۵-۴۲ یک کلاس ایجاد می‌کنیم که رابط `ActionListener` را پیاده سازی می‌کند. سپس در داخل بدنه متده `actionPerformed()` مربوط به این رابط هم کدهای خطوط ۳۵-۴۲ را می‌نویسیم.

وقتی که شما بر روی `buttonShow` کلیک می‌کنید، برنامه تعیین می‌کند که کدام `JRadioButton` انتخاب شده است. شما می‌توانید این کار را به وسیله‌ی متده `isSelected()` انجام دهید. این متده در صورتیکه یک کنترل انتخاب شده باشد مقدار `true` در غیر اینصورت

را بر می‌گرداند. ما از یک عبارت شرطی if برای تعیین اینکه `JRadioButton` انتخاب شده است یا خیر استفاده می‌کنیم. اگر آن `false` انتخاب نشده باشد، پس دکمه‌ی `JRadioButton` ی دیگر انتخاب شده است. چون ما فقط دو دکمه‌ی `JRadioButton` بر روی فرم دارم و نهایتاً یکی از آن‌ها انتخاب می‌شود.



JCheckBox کنترل

کنترل `JCheckBox` یک دکمه است و به شکل یک جعبه‌ی خالی به همراه یک برچسب در کنار آن نمایش داده می‌شود. در حالت عادی، زمانی که بر روی جعبه‌ی خالی کلیک شود، یک تیک در داخل جعبه نمایان می‌شود که به ما می‌گوید کنترل `JCheckBox` در حالت

قرار دارد. برخلاف دکمه‌ی Radio که فقط اجازه‌ی انتخاب یکی از Radio های فرم را به ما می‌داد، شما می‌توانید چند عدد Selected

و یا همه‌ی آن‌ها را تیک بزنید. سازنده کلاس JCheckBox به روش‌های زیر مقداردهی می‌شود :

```
public JCheckBox()
public JCheckBox(Icon icon)
public JCheckBox(Icon icon, boolean selected)
public JCheckBox(String text)
public JCheckBox(String text, boolean selected)
public JCheckBox(String text, Icon icon)
public JCheckBox(String text, Icon icon, boolean selected)
public JCheckBox(Action action)
```

isSelected() دو حالت On یا Off (تیک خورده و تیک نخورده) را قبول کند، در نتیجه شما به سادگی می‌توانید از متدهای

که دو مقدار True به معنی اینکه JCheckBox تیک خورده و False به معنی اینکه JCheckBox تیک نخورده است را بر می‌گرداند،

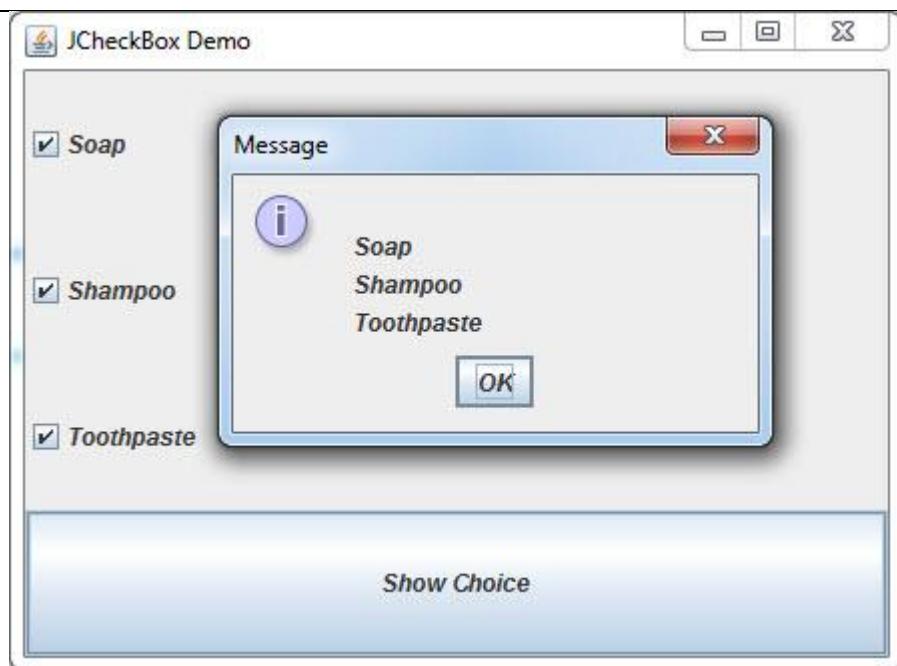
استفاده کنید. در مثال زیر نحوه‌ی کاربرد کنترل CheckBox را مشاهده می‌کنید :

```
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        final JFrame frame1 = new JFrame("JCheckBox Demo");
12
13        JPanel Panel1 = new JPanel();
14        Panel1.setLayout(new GridLayout(4,1));
15
16        final JCheckBox CheckBox1 = new JCheckBox("Soap");
17        final JCheckBox CheckBox2 = new JCheckBox("Shampoo");
18        final JCheckBox CheckBox3 = new JCheckBox("Toothpaste");
19        JButton buttonShow = new JButton("Show Choice");
20
21        Panel1.add(CheckBox1);
22        Panel1.add(CheckBox2);
23        Panel1.add(CheckBox3);
24        Panel1.add(buttonShow);
25
26        frame1.add(Panel1);
27
28        class SelectedCheckBox implements ActionListener
29        {
30            @Override
31            public void actionPerformed(ActionEvent e)
32            {
33                String items = "";
34                if (CheckBox1.isSelected())
```

```

35         {
36             items += "\n Soap";
37         }
38         if (CheckBox2.isSelected())
39         {
40             items += "\n Shampoo";
41         }
42         if (CheckBox3.isSelected())
43         {
44             items += "\n Toothpaste";
45         }
46         JOptionPane.showMessageDialog(frame1, items);
47     }
48 }
49
50 buttonShow.addActionListener(new SelectedCheckBox());
51
52 frame1.setSize(356 , 263);
53 frame1.setVisible(true);
54 }
55 }
```

در کد بالا قرار است که وقتی کاربر چک باکس‌ها را انتخاب و سپس بر روی دکمه کلیک کرد نام چک باکس‌های انتخاب شده نمایش داده شود. در خطوط ۱۱-۲۶ یک Frame ایجاد کرده‌ایم. سپس سه کنترل JCheckBox و یک دکمه را به یک Panel و خود Panel را به اضافه نموده‌ایم. برای اداره رویداد کلیک دکمه در خطوط ۳۰-۴۴ یک کلاس ایجاد می‌کنیم که رابط ActionListener را پیاده سازی می‌کند. سپس در داخل بدنه متده (actionPerformed) مربوط به این رابط هم کدهای خطوط ۳۳-۴۶ را می‌نویسیم. در این کد، یک متغیر از نوع رشته‌ای تعریف کرده‌ایم و آنرا با یک رشته‌ی تهی مقدار دهی کرده‌ایم. سپس چک می‌کنیم که کدام یک از چک باکس‌ها تیک خورده‌اند، هر کدام از آن‌ها که تیک خورده‌اند نامشان به وسیله‌ی عملگر + در رشته‌ای که قبلًاً به صورت تهی تعریف کرده‌ایم قرار می‌گیرد. سپس به وسیله‌ی یک پیغام (JOptionPane) نتایج را در خروجی نمایش می‌دهیم.



JPanel کنترل

کنترل JPanel معمولاً برای گروه بندی کنترل‌های فرم بکار می‌رود. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JPanel()
public JPanel(boolean isDoubleBuffered)
public JPanel(LayoutManager manager)
public JPanel(LayoutManager manager, boolean isDoubleBuffered)
```

یکی از کاربردهای خوب JPanel این است که شما می‌خواهید دکمه‌های Radio را گروه بندی کنید. با توجه به اینکه فقط یکی از دکمه‌های JPanel در فرم می‌تواند فعال باشد، شما می‌توانید با گروه بندی کردن آنها بیش از یک دکمه‌ی JRadioButton باشید. وقتی که شما کنترلی را در داخل یک JPanel قرار می‌دهید، آن کنترل به فرزند JPanel تبدیل می‌شود و JPanel به والد کنترل تبدیل می‌شود. برای درک این مساله کافیست به عنوان مثال، JPanel را تغییر اندازه دهید. هر کنترلی که در داخل JPanel قرار دارد به همراه آن حرکت می‌کند. به کد زیر توجه کنید :

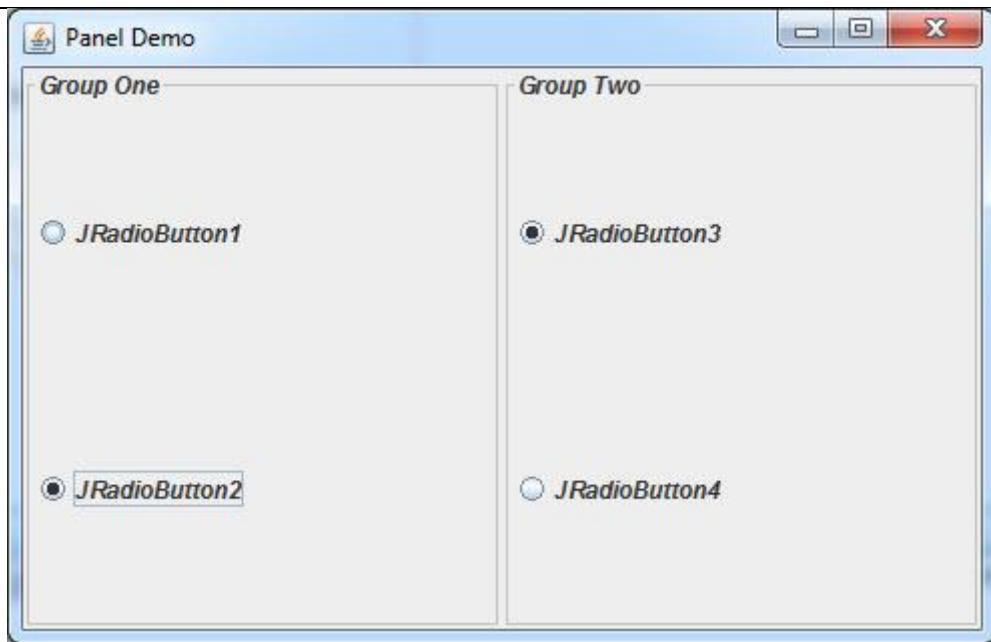
```
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.border.*;
6
7 public class MyFirstProgram
```

```

8
9 {
10     public static void main(String[] args)
11     {
12         JFrame frame1 = new JFrame("JCheckBox Demo");
13         frame1.setLayout(new GridLayout(1,2));
14
15         JPanel Panel1 = new JPanel();
16         Panel1.setLayout(new GridLayout(2,1));
17         Panel1.setBorder(new TitledBorder(
18             new MatteBorder(1, 1, 1, 1, Color.LightGray), "Group One"));
19
20         JPanel Panel2 = new JPanel();
21         Panel2.setLayout(new GridLayout(2,1));
22         Panel2.setBorder(new TitledBorder(
23             new MatteBorder(1, 1, 1, 1, Color.LightGray), "Group Two"));
24
25         JRadioButton JRadioButton1 = new JRadioButton("JRadioButton1");
26         JRadioButton JRadioButton2 = new JRadioButton("JRadioButton2");
27         JRadioButton JRadioButton3 = new JRadioButton("JRadioButton3");
28         JRadioButton JRadioButton4 = new JRadioButton("JRadioButton4");
29
30         ButtonGroup ButtonGroup1 = new ButtonGroup();
31         ButtonGroup1.add(JRadioButton1);
32         ButtonGroup1.add(JRadioButton2);
33
34         ButtonGroup ButtonGroup2 = new ButtonGroup();
35         ButtonGroup2.add(JRadioButton3);
36         ButtonGroup2.add(JRadioButton4);
37
38         Panel1.add(JRadioButton1);
39         Panel1.add(JRadioButton2);
40
41         Panel2.add(JRadioButton3);
42         Panel2.add(JRadioButton4);
43
44         frame1.add(Panel1);
45         frame1.add(Panel2);
46
47         frame1.setSize(356 , 263);
48         frame1.setVisible(true);
49     }
}

```

در کد بالا و در خطوط ۱۱-۲۲ یک Frame و دو Panel و در خطوط ۱۶ و ۲۱ دو حاشیه هم به Panel ها همراه با عنوان برای آنها ایجاد کردہ‌ایم. در خطوط ۲۴-۳۵ چهار دکمه رادیویی ایجاد کرده و آنها را در دو گروه، گروه بندی می‌کنیم (خطوط ۲۹-۳۵). گروه اول می‌دهیم (خطوط ۴۱-۴۷). حال برنامه را اجرا و نتیجه را با انتخاب دکمه‌ها مشاهده کنید:



JComboBox کنترل

کنترل `JComboBox` (نوار کرکرهای) روش دیگری است که به کاربر اجازه دهد از بین گزینه‌های مختلف یکی را انتخاب کند. کنترل `JComboBox` شبیه یک کنترل `TextBox` است که در سمت راست آن یک دکمه قرار دارد. وقتی بر روی دکمه‌ی آن کلیک شود، `JComboBox` یک نوار کرکرهای را که حاوی یک لیست از گزینه‌های مختلف است را نمایش می‌دهد. کاربر می‌تواند از بین این گزینه‌ها یکی را انتخاب کند. مورد انتخاب شده به متن داخل `ComboBox` تبدیل می‌شود. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JComboBox()
public JComboBox(Object listData[])
public JComboBox(Vector listData)
public JComboBox(ComboBoxModel model)
```

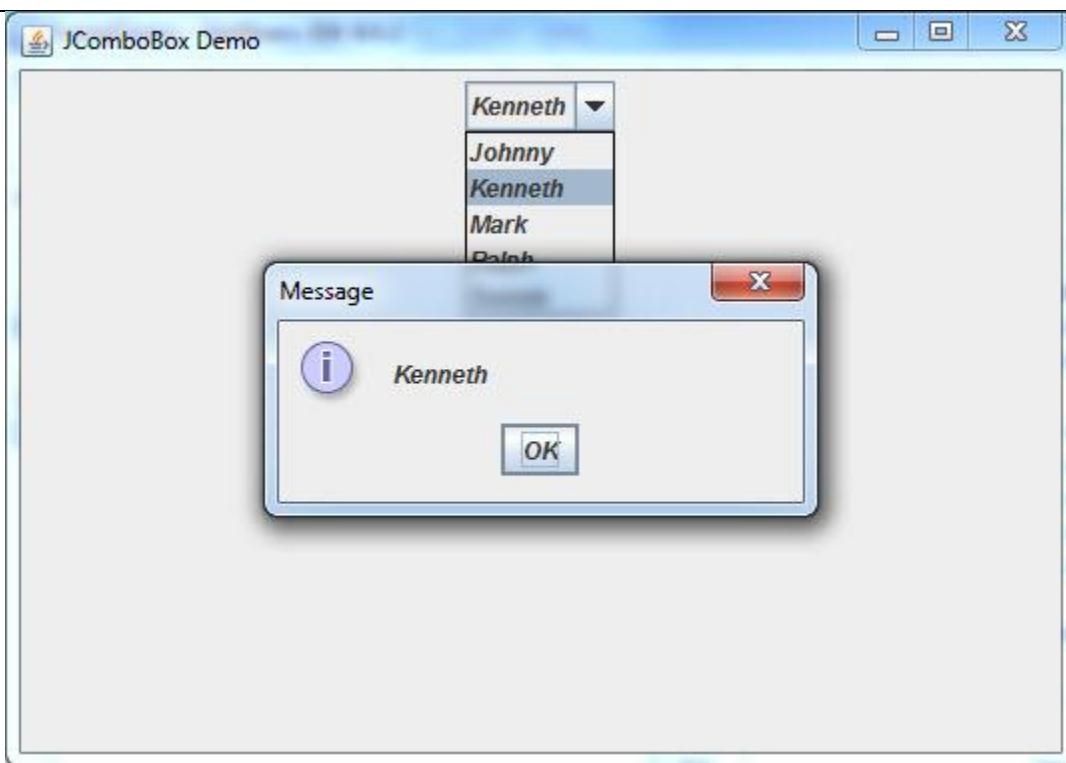
در مثال زیر با کاربرد `JComboBox` آشنا می‌شوید :

```
1 package myfirstprogram;
2
3 import javax.swing.*;
4 import java.awt.event.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         final JFrame frame1 = new JFrame("JComboBox Demo");
11         JPanel Panel1 = new JPanel();
```

```

12     final JComboBox ComboBox1 = new JComboBox();
13     Panel1.add(ComboBox1);
14     frame1.add(Panel1);
15
16     ComboBox1.addItem("Johnny");
17     ComboBox1.addItem("Kenneth");
18     ComboBox1.addItem("Mark");
19     ComboBox1.addItem("Ralph");
20     ComboBox1.addItem("Sussie");
21
22     class MyItemListener implements ActionListener
23     {
24         public void actionPerformed(ActionEvent e)
25         {
26             String selectedName = ComboBox1.getSelectedItem().toString();
27             JOptionPane.showMessageDialog(frame1, selectedName);
28         }
29     }
30
31     ComboBox1.addActionListener(new MyItemListener());
32
33     frame1.setSize(430 , 315);
34     frame1.setVisible(true);
35
36 }
```

کلاً هدف از برنامه بالا این است که با کلیک بر روی هر آیتم از `JComboBox` مقدار آن به صورت پیام نمایش داده شود. در خطوط ۱۰-۱۴ کد بالا یک `Frame`, یک `Panel` و یک `ComboBox` ایجاد کرده‌ایم. در خطوط ۱۶-۲۰ آیتم‌هایی را که قرار است به `JComboBox` اضافه شوند را با استفاده از متدهای `addItem()` اضافه می‌کنیم. چون که قرار است با کلیک بر روی آیتم‌های کمبوباکس پیام نمایش داده شود پس باید رابط `ActionListener` توسط یک کلاس پیاده سازی شود. این کار را در خط ۲۲ انجام داده‌ایم. در خطوط ۲۶-۲۷ هم کدهای بدنه متدهای `actionPerformed()` و `getSelectedItem()` را فراهم می‌کنیم. در خط ۲۶ یک رشته تعریف کرده‌ایم و با استفاده از متدهای `showMessageDialog()` و `String.toString()` کنترل `JComboBox` مقدار آیتم انتخاب شده را در داخل این رشته می‌بیزیم. در خط ۲۷ هم با استفاده از کلاس `JOptionPane` این رشته که همان آیتم انتخاب شده کمبوباکس است را نمایش می‌دهیم :



برای اضافه کردن آیتمها به کمبوباکس به صورت زیر هم می‌توان عمل کرد :

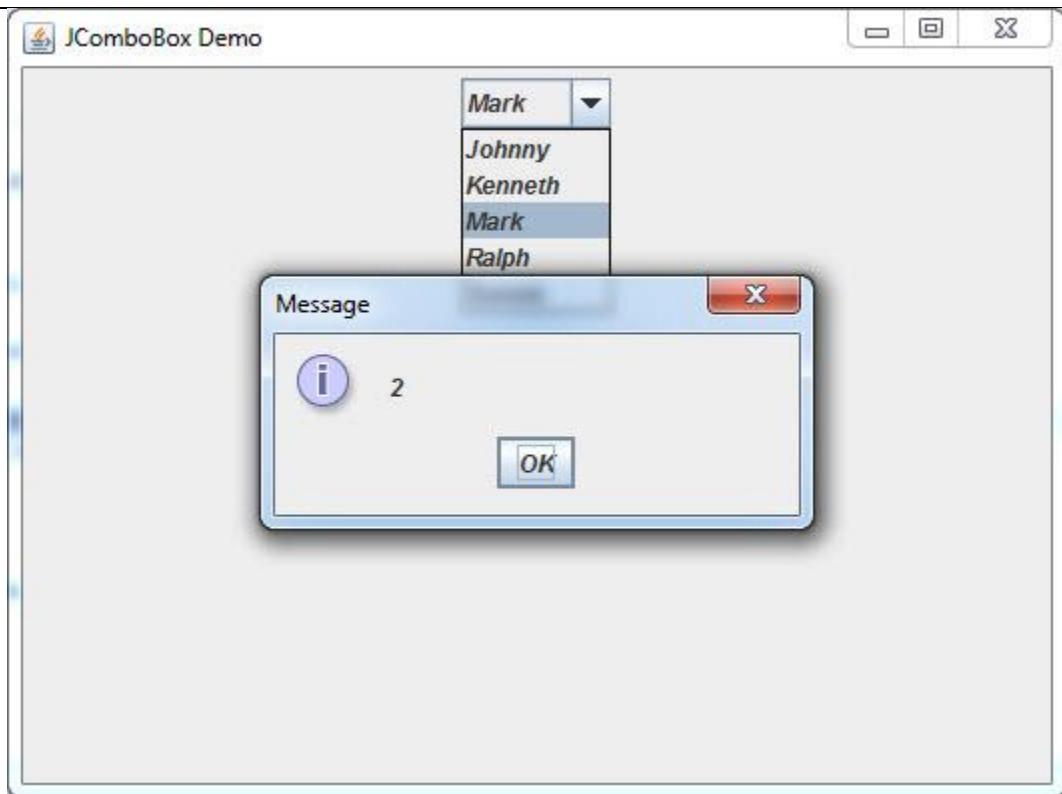
```
String[] items = { "Johnny", "Kenneth", "Mark", "Ralph", "Sussie" };
JComboBox ComboBox1 = new JComboBox(items);
```

در این روش یک آرایه از نوع رشته تعریف و سپس آیتم‌های مورد نظرتان را به آن اضافه می‌کنید. سپس نام آرایه را به سازنده کلاس `JComboBox` ارسال می‌کنید. آیتم‌های کمبوباکس دارای اندیس هستند و این اندیس‌ها از صفر شروع می‌شوند. اندیس اولین آیتم،^۰ اندیس دومین آیتم `۱` و ... برای به دست آوردن اندیس آیتم انتخاب شده از متده استفاده می‌شود. خط `getSelectedIndex()`

کد بالا را به صورت زیر تغییر دهید :

```
String selectedName = String.valueOf(ComboBox1.getSelectedIndex());
```

برنامه را اجرا و با انتخاب یک آیتم نتیجه را مشاهده کنید :



کنترل JList

کنترل `JList` برای نمایش لیستی از آیتم‌ها که قابل انتخاب هستند استفاده می‌شود. این آیتم‌ها را می‌توانید به صورت تکی و گروهی انتخاب کنید. کنترل `JList` بهترین گزینه برای موقعی است که شما می‌خواهید تعداد زیادی آیتم را در ستون‌های افقی و عمودی نمایش دهید. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JList()
public JList(Object listData[])
public JList(Vector listData)
public JList(ListModel model)
```

برای آشنایی بیشتر با خصیت‌ها و متدهای این کنترل به مثال زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import javax.swing.*;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {
9         JFrame frame1 = new JFrame("JList Demo");
```

```

10     JPanel Panel1 = new JPanel();
11     JList List1 = new JList();
12     Panel1.add(List1);
13     frame1.add(Panel1);
14
15     String[] items = { "Johnny", "Kenneth", "Mark", "Ralph", "Sussie" };
16     List1.setListData(items);
17
18     frame1.setSize(430 , 315);
19     frame1.setVisible(true);
20 }
21 }
```

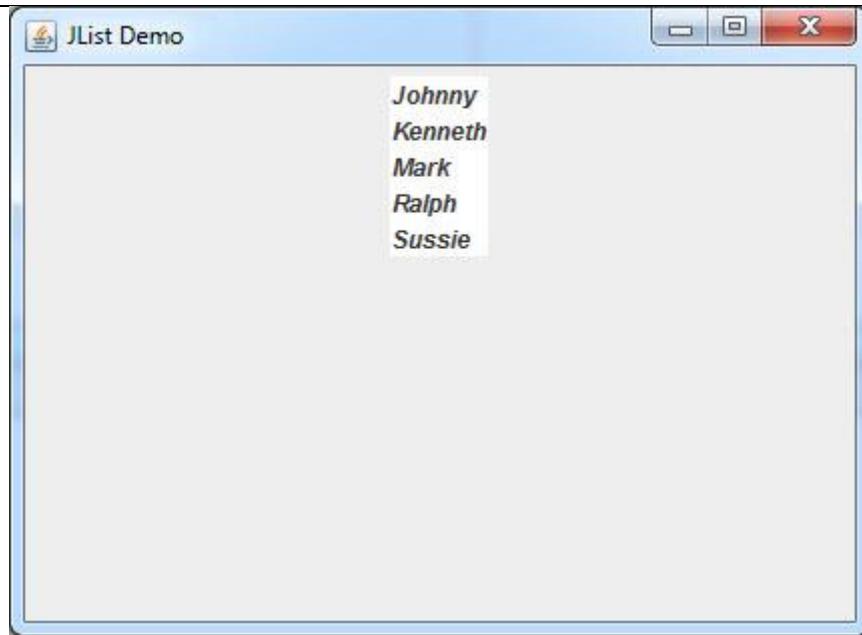
در کد بالا و در خطوط ۹-۱۳ سه کنترل Frame، Panel و List ایجاد کرده‌ایم. حال نوبت به اضافه کردن آیتم‌ها به List می‌رسد. در خط ۱۵ یک آرایه رشته‌ای تعریف و آیتم‌ها را به آن اضافه می‌کنیم. در خط ۱۶ این آرایه را به متدهای setListData() ارسال می‌کنیم و این متدهای آیتم‌ها را به List اضافه می‌کند. به جای خطوط ۱۵ و ۱۶ می‌توان از کد زیر استفاده کرد :

```

DefaultListModel DefaultListModel1 = new DefaultListModel();
DefaultListModel1.addElement("Johnny");
DefaultListModel1.addElement("Kenneth");
DefaultListModel1.addElement("Mark");
DefaultListModel1.addElement("Ralph");
DefaultListModel1.addElement("Sussie");

List1.setModel(DefaultListModel1);
```

کلاسی است که دارای متدهایی برای حذف و اضافه و در کل دستکاری آیتم‌های List می‌باشد. با استفاده از addElement() آیتم‌ها را به یک شیء از کلاس مذکور اضافه و سپس این شیء را به متدهای setListData() ارسال می‌کنیم. برنامه را اجرا و نتیجه را مشاهده کنید :



کلاس DefaultListModel دارای متد remove() برای حذف یک آیتم می‌باشد این متد اندیس آیتمی را که می‌خواهید حذف کنید را دریافت می‌کند :

```
DefaultListModel1.remove(2);
```

برای حذف چند آیتم از JList هم می‌توان از متد removeRange() که اندیس شروع و پایان آیتم‌هایی که پشت سر هم هستند و می‌خواهید حذف کنید را می‌گیرد :

```
DefaultListModel1.removeRange(2,4);
```

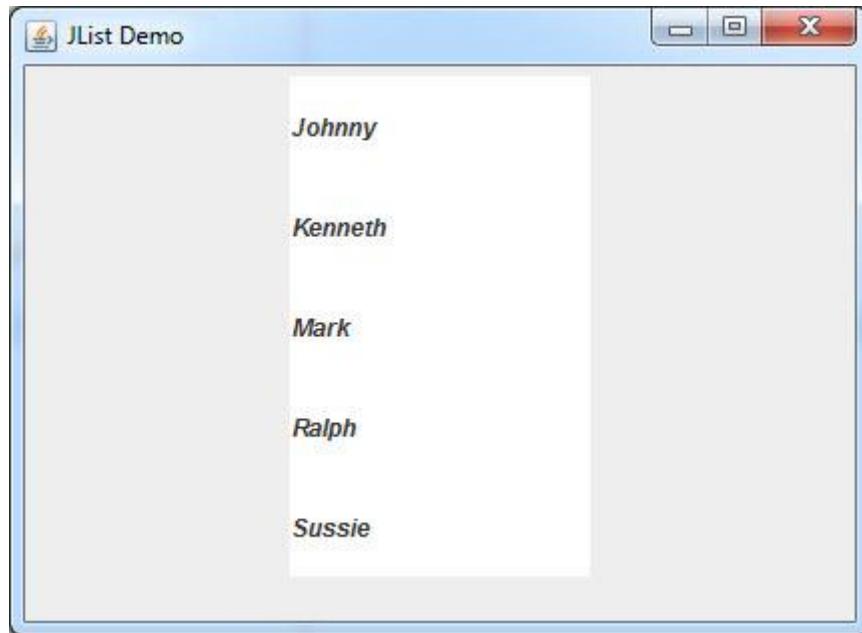
در حالت پیشفرض با گرفتن دکمه Ctrl کیبورد و کلیک بر روی آیتم‌ها می‌توان چند آیتم را به طور همزمان انتخاب کرد ولی اگر بخواهید فقط یک آیتم قابل انتخاب باشد می‌توانید از متد setSelectionMode() و ارسال مقدار ثابت SINGLE_SELECTION از رابط ListSelectionModel به آن استفاده کرد. همان کد ابتدای درس را در در نظر بگیرید. در خط ۱۷ کد فوق کدهای زیر را بنویسید :

```
List1.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

برای تعیین عرض و ارتفاع آیتم‌ها از متدهای setFixedCellHeight() برای تعیین عرض و setFixedCellWidth() برای تعیین ارتفاع سلول‌ها استفاده می‌شود :

```
List1.setFixedCellWidth(100);
```

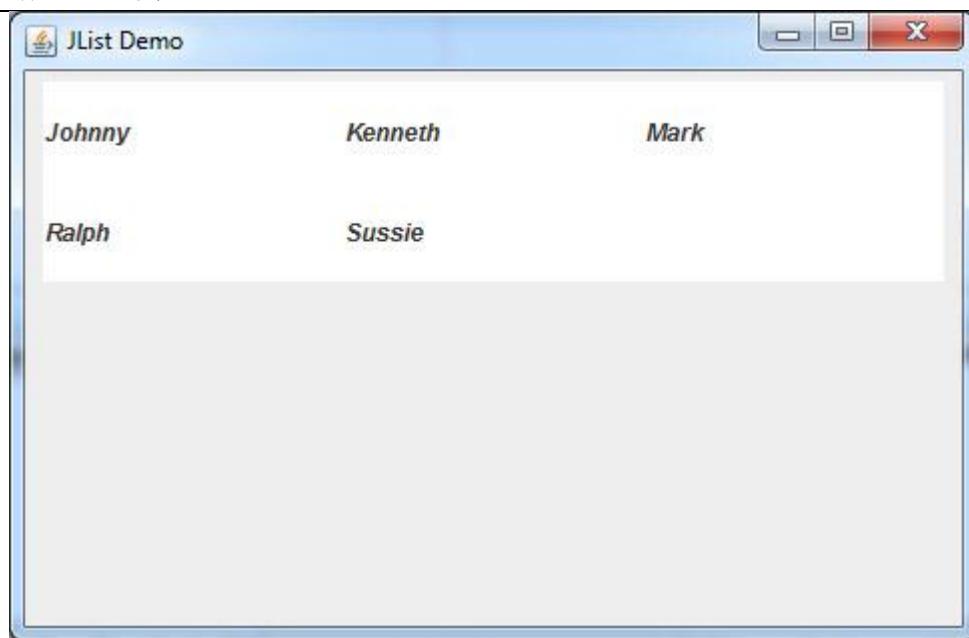
```
List1.setFixedCellHeight(50);
```



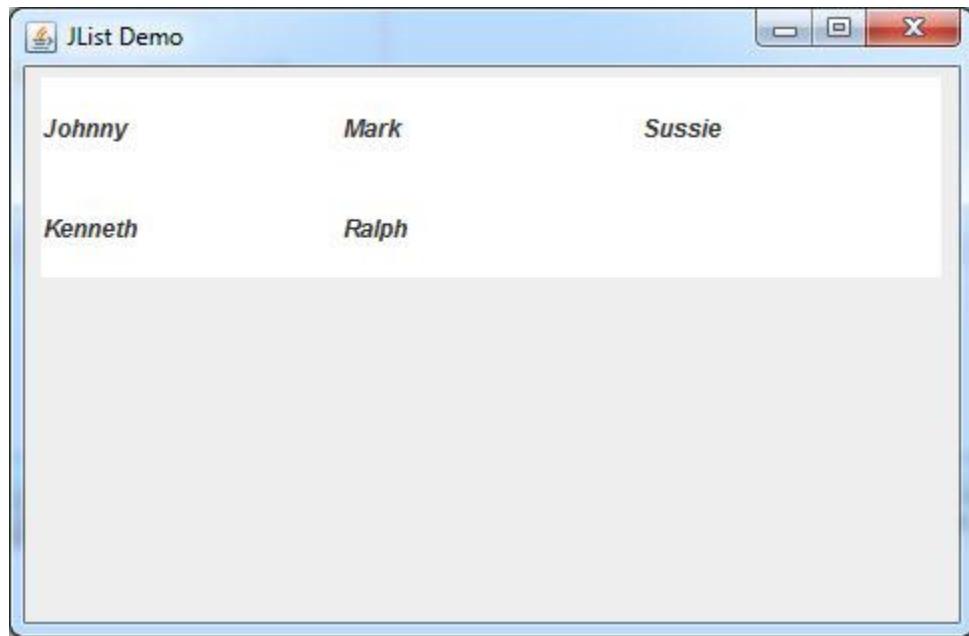
برای نمایش سطري و ستونی آیتمها می‌توان از متدهای `setLayoutOrientation()` برای تعیین نحوه چیدمان افقي یا عمودي آیتمها و `setVisibleRowCount()` برای تعیین سطرهای List که قرار است آیتمها در آنها نمایش داده شوند استفاده کرد. در ادامه کدهای بالا کدهای زیر را بنویسید :

```
List1.setLayoutOrientation(JList. HORIZONTAL_WRAP);
List1.setVisibleRowCount(2);
```

مقدار `HORIZONTAL_WRAP` باعث چینش افقي آیتمها می‌شود. آیتمها به طور خودکار طوری در ستونها قرار می‌گيرند که تعداد سطرهای بيشتر از آن مقداری که تعیين کرده‌ایم نشود :



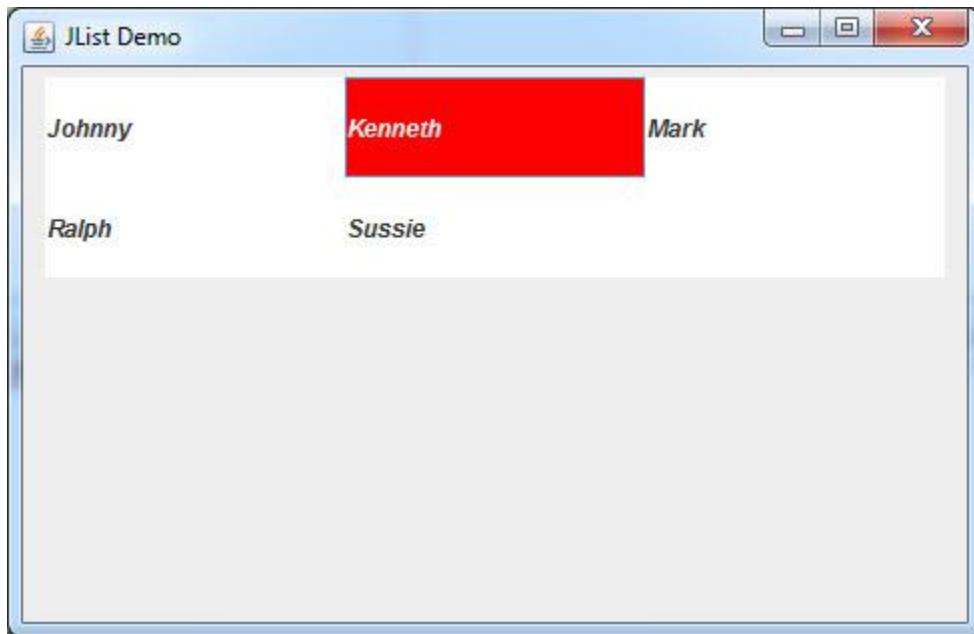
حال مقدار را به VERTICAL_WRAP تغییر داده و برنامه را اجرا و نتیجه را مشاهده کنید :



برای تعیین رنگ پس زمینه و رنگ فونت آیتمها هم از متدهای setSelectionBackground() و setSelectionForeground() استفاده می شود. کدهای زیر را هم به کد بالا اضافه کنید :

```
List1.setSelectionBackground(Color.RED);
```

```
List1.setSelectionForeground(Color.WHITE);
```



JSpinner کنترل

کنترل JSpinner عموماً برای دریافت اعداد از ورودی و محدود کردن کاربران برای وارد کردن مقادیر عددی بکار می‌رود. کنترل JSpinner از لحاظ شکل ظاهری شبیه به کنترل TextBox است با این تفاوت که دکمه‌هایی به شکل پیکان در سمت چپ یا راست آن برای افزایش و یا کاهش مقدار کنترل وجود دارند. مقدار عددی کنترل JSpinner می‌تواند از نوع اعشاری یا صحیح باشد. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JSpinner()
public JSpinner(SpinnerModel model)
```

حال می‌خواهیم یک برنامه بسازیم که کنترل JSpinner در آن بکار رفته باشد. به کد زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
```

```

11     final JFrame frame1 = new JFrame("JSpinner Demo");
12     frame1.setLayout(new BorderLayout());
13
14     JPanel Panel1 = new JPanel();
15     JPanel Panel2 = new JPanel();
16     Panel1.setLayout(new GridLayout(2,2));
17
18     JLabel label1 = new JLabel("Price");
19     JLabel label2 = new JLabel("Quantity");
20
21     SpinnerNumberModel Pricemode1          = new SpinnerNumberModel(0, 0, 10000, 0.5);
22     final JSpinner    SpinnerUpDownPrice   = new JSpinner(Pricemode1);
23     SpinnerNumberModel Quantitymodel      = new SpinnerNumberModel(0, 0, 100, 1);
24     final JSpinner    SpinnerUpDownQuantity = new JSpinner(Quantitymodel);
25
26     JButton Button1 = new JButton("Culculate");
27
28     Panel1.add(label1);
29     Panel1.add(SpinnerUpDownPrice);
30     Panel1.add(label2);
31     Panel1.add(SpinnerUpDownQuantity);
32     Panel2.add(Button1);
33     frame1.add(Panel1, BorderLayout.CENTER);
34     frame1.add(Panel2, BorderLayout.SOUTH);
35
36     class MyItemListener implements ActionListener
37     {
38         public void actionPerformed(ActionEvent e)
39         {
40             double price = (double)SpinnerUpDownPrice.getValue();
41             int quantity = (int)SpinnerUpDownQuantity.getValue();
42             double total;
43
44             total = price* quantity;
45
46             JOptionPane.showMessageDialog(frame1,"The total price is " + total);
47         }
48     }
49
50     Button1.addActionListener(new MyItemListener() );
51
52     frame1.setSize(364 , 146);
53     frame1.setVisible(true);
54 }
55 }
```

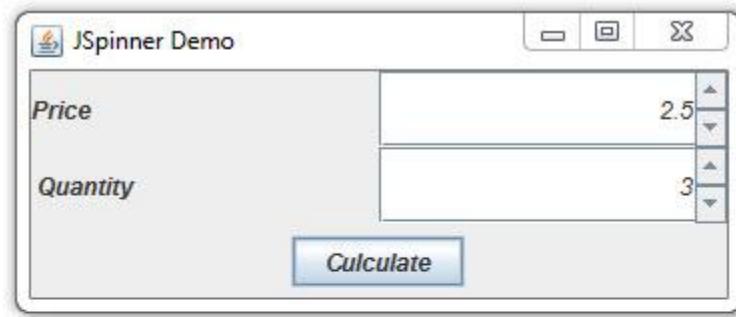
در کد بالا و در خطوط ۱۱-۳۴ کار خاصی انجام ندادهایم. فقط یک Frame، دو Panel، دو Label و دو JSpinner با نام‌های ۲۱ و ۲۲ نهفته است. در حالت پیشفرض با زدن دکمه‌های پایین و بالای JSpinner یک واحد از مقدار پیشفرض آن یعنی صفر، کم یا یک واحد به آن اضافه می‌شود. برای محدود کردن کاربران به یک محدود خاص از اعداد از کلاس SpinnerNumberModel استفاده می‌شود. این کلاس ورودی‌هایی که توسط کاربر در JSpinner وارد می‌شود را محدود می‌کند. همانطور که در کد بالا مشاهده می‌کنید سازنده این کلاس ۴ پارامتر می‌گیرد که اولین عدد مبنای شروع افزایش و کاهش عدد JSpinner، دو عدد بعدی محدودهای است که ورودی‌های کاربر در آن قرار می‌گیرند و چهارمین عدد هم مقدار افزایش و کاهش عدد JSpinner است. این اعداد همگی می‌توانند هم double و

هم int باشند. در خطوط ۲۲ و ۲۴ هم اشیاء ساخته شده از این کلاس را به سازنده کنترل JSpinner ارسال کرده‌ایم تا محدودیت‌ها را

بر روی آن‌ها اعمال کند.

در خط ۲۱ آخرین پارامتر Pricemode1 برابر با $\text{价}/\text{量}$ قرار داده‌ایم، بنابراین مقدار SpinnerUpDownPrice به اندازه $\text{价}/\text{量}$ واحد کم یا زیاد می‌شود. مقدار سومین پارامتر یعنی نهایت عددی که کاربر می‌تواند وارد کند ۱۰۰۰ است، پس قیمت (Price) به آن محدود می‌شود و کاربر نمی‌تواند رقمی بالاتر از آنرا وارد کند.

از آنجاییکه نهایت مقدار Quantitymodel را برابر ۱۰۰ قرار داده‌ایم بنابراین شما می‌توانید فقط مقادیری نهایتاً تا ۱۰۰ را در SpinnerUpDownQuantity وارد کنید. چون می‌خواهیم با وارد کردن دو مقدار در دو JSpinner و سپس زدن دکمه پیامی نمایش داده شود باید رویداد کلیک دکمه را مدیریت کنیم. این کار را در خطوط ۳۶-۵۰ انجام داده‌ایم. در خطوط ۴۰ و ۴۱ مقادیری را که در دو JSpinner توسط کاربر وارد شده است را توسط متده است `getValue()` به دست می‌آوریم و سپس با عمل cast به نوع double و int تبدیل می‌کنیم. سپس در خط ۴۴ حاصلضرب این دو مقدار را در متغیر total ریخته و مقدار این متغیر را در خط ۴۶ چاپ می‌کنیم :



JSlider کنترل

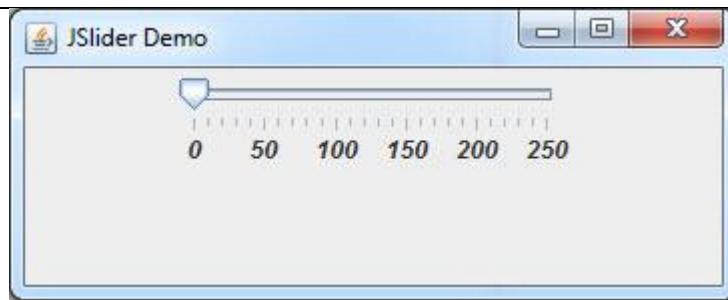
کنترل JSlider شبیه یک نوار لغزنده با یک دستگیره است که با استفاده از این دستگیره می‌توان مقدار آن را تعیین کنید. ناحیه‌ای که دستگیره به آن اشاره می‌کند نشان دهنده مقدار جاری کنترل است. کاربر می‌تواند دستگیره به درجه افقی حرکت دهد و همچنین می‌توان جهت حرکت را به صورت عمودی تغییر داد. مقدار این کنترل در حالت افقی از سمت چپ به راست و در حالت عمودی از پایین به بالا افزایش می‌یابد. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JSlider()
public JSlider(int orientation)
public JSlider(int minimum, int maximum)
public JSlider(int minimum, int maximum, int value)
public JSlider(int orientation, int minimum, int maximum, int value)
public JSlider(BoundedRangeModel model)
```

در زیر نحوه ایجاد و استفاده از JSlider و استفاده از متدهای آن آمده است :

```
1 package myfirstprogram;
2
3 import javax.swing.*;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {
9         final JFrame frame1 = new JFrame("JSpinner Demo");
10        JPanel Panel1 = new JPanel();
11        JSlider Slider = new JSlider(JSlider.HORIZONTAL, 0, 255, 0);
12        Panel1.add(Slider);
13        frame1.add(Panel1);
14
15        Slider.setMinorTickSpacing(10);
16        Slider.setMajorTickSpacing(50);
17
18        Slider.setPaintLabels(true);
19        Slider.setPaintTicks(true);
20
21        frame1.setSize(364 , 146);
22        frame1.setVisible(true);
23    }
24 }
```

در خط ۱۱ کد بالا یک Slider ایجاد کردہ‌ایم و در داخل سازنده نحوه نمایش Slider به صورت افقی (JSlider.HORIZONTAL) مشخص شده است. عدد ۰ و ۲۵۵ هم بازه‌ای است که اسلایدر می‌گیرد و عدد ۰ آخر هم نقطه شروع اسلایدر می‌باشد. در خطوط ۱۵-۱۹ هم این اسلایدر را درجه بندی کردہ‌ایم. از متد setMajorTickSpacing() ایجاد درجه‌های بزرگ بر روی Slider و از متد setMinorTickSpacing() برای ایجاد درجه‌های کوچک در بین دو درجه بزرگ و از متدهای setPaintLabels() و setPaintTicks() برای نمایش درجه‌ها و برچسب‌های آن‌ها استفاده می‌شود. برنامه را اجرا و نتیجه را مشاهده کنید :



توصیه می‌شود که مقادیر موجود در سازنده (خط ۱۱) و متدهای خطوط ۱۵ و ۱۶ را تغییر داده تا عملکرد این آن‌ها را بهتر درک کنید. همچنین مقادیر true خطوط ۱۹ و ۱۸ را به false تغییر داده تا کاربرد این دو متده را متوجه شوید. حال اجازه دهید که یک برنامه بنویسیم که دارای سه Slider باشد و با تغییر مقادیر Slider‌ها رنگ پس زمینه یک Panel تغییر کند. به کد زیر توجه کنید :

```

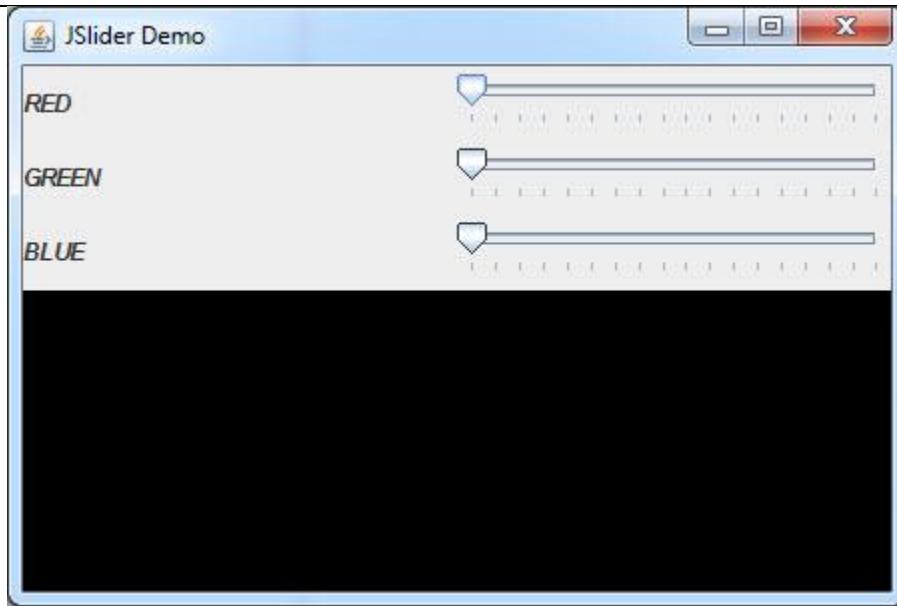
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import javax.swing.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        final JFrame frame1 = new JFrame("JSlider Demo");
12        frame1.setLayout(new BorderLayout());
13
14        JPanel Panel1 = new JPanel();
15        Panel1.setLayout(new GridLayout(3,2));
16
17        JLabel Label1 = new JLabel("RED");
18        JLabel Label2 = new JLabel("GREEN");
19        JLabel Label3 = new JLabel("BLUE");
20
21        final JSlider SliderGreen = new JSlider(JSlider.HORIZONTAL,0,255,0);
22        final JSlider SliderRed = new JSlider(JSlider.HORIZONTAL,0,255,0);
23        final JSlider SliderBlue = new JSlider(JSlider.HORIZONTAL,0,255,0);
24        SliderGreen.setMinorTickSpacing(15);
25        SliderRed.setMinorTickSpacing(15);
26        SliderBlue.setMinorTickSpacing(15);
27        SliderGreen.setPaintTicks(true);
28        SliderRed.setPaintTicks(true);
29        SliderBlue.setPaintTicks(true);
30
31        Panel1.add(Label1);
32        Panel1.add(SliderRed);
33        Panel1.add(Label2);
34        Panel1.add(SliderGreen);
35        Panel1.add(Label3);
36        Panel1.add(SliderBlue);
37

```

```

38     final JPanel Panel2 = new JPanel();
39     Panel2.setPreferredSize(new Dimension(0,150));
40     Panel2.setBackground(Color.BLACK);
41
42     frame1.add(Panel1, BorderLayout.CENTER);
43     frame1.add(Panel2, BorderLayout.SOUTH);
44
45     class ColorPicker implements ChangeListener
46     {
47         public void stateChanged(ChangeEvent changeEvent)
48         {
49             int red   = SliderRed.getValue();
50             int green = SliderGreen.getValue();
51             int blue  = SliderBlue.getValue();
52             Color color = new Color(red, green, blue);
53             Panel2.setBackground(color);
54         }
55     }
56
57     SliderGreen.addChangeListener(new ColorPicker());
58     SliderRed.addChangeListener(new ColorPicker());
59     SliderBlue.addChangeListener(new ColorPicker());
60
61     frame1.setSize(450 , 300);
62     frame1.setVisible(true);
63 }
64 }
```

در خط ۱۱ کد بالا یک Frame ایجاد کرده‌ایم و در خط ۱۲ نحوه لایه بندی آن را BorderLayout قرار داده‌ایم تا بتوانیم دو پنل را در وسط و پایین آن قرار دهیم. در خط ۱۴ اولین پنل را ایجاد و در خط ۱۵ آن را به ۳ سطر و دو ستون تقسیم کرده‌ایم. در خطوط ۱۷-۱۹ سه Label ایجاد کرده‌ایم. در خطوط ۲۱-۲۹ سه Slider ایجاد و مقادیر آن‌ها را بین ۰-۲۵۵ قرار داده‌ایم. چون کد رنگ‌ها در همین بازه قرار دارند. مثلاً رنگ کد سیاه (۰,۰,۰) و کد رنگ سفید (۲۵۵,۲۵۵,۲۵۵) می‌باشد. در خطوط ۳۱-۳۶ Label و Slider ها را به ۱۱ اولی اضافه می‌کنیم. در خطوط ۳۸-۴۰ Panel دوم را ایجاد کرده و یک ارتفاع به ان می‌دهیم و رنگ پیشفرض پس زمینه آن را سیاه می‌دهیم. چون زمانی که برنامه را اجرا می‌کنیم در حالت پیشفرض کل Slider ها مقدار ۰ دارند و این کد رنگ سیاه است :

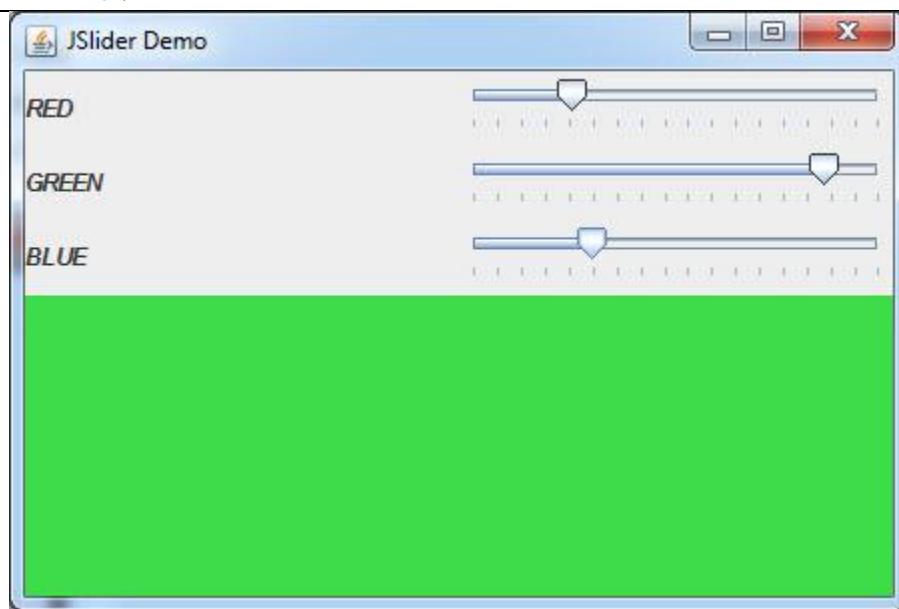


هنگام که دستگیره Slider به دو طرف حرکت می‌دهیم در اصل رابط ChangeListener پیاده سازی می‌شود. این رابط را در خط ۴۵

توسط یک کلاس پیاده سازی می‌کنیم و در داخل بدنه متده استChanged() این رابط کدهای زیر را می‌نویسیم :

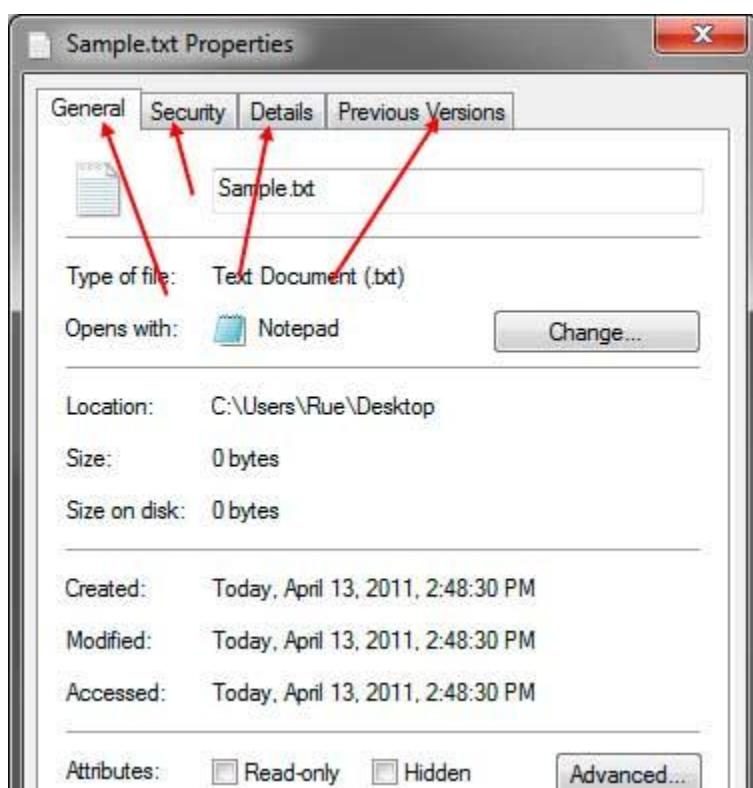
```
int red = SliderRed.getValue();
int green = SliderGreen.getValue();
int blue = SliderBlue.getValue();
Color color = new Color(red, green, blue);
Panel2.setBackground(color);
```

در کدهای بالا با استفاده از متده getValue() مقادیر هر یک از Slider ها را می‌گیریم. این مقادیر از نوع صحیح هستند. سپس همین مقادیر را در خط ۵۲ به به سازنده کلاس Color ارسال می‌کنیم. وظیفه این کلاس ایجاد رنگ است. سپس یک شیء از این کلاس را به متده setBackground() در خط ۵۳ می‌دهیم تا رنگ را بر روی Panel دومی اعمال کند. و در آخر یک شیء از کلاسی که رابط را پیاده سازی می‌کند یعنی ColorPicker (خط ۴۵) به متده addChangeListener() در خطوط ۵۷-۵۹ ارسال می‌کنیم. برنامه را اجرا و دستگیره‌های Slider ها را جایه جا و نتیجه را مشاهده کنید.



کنترل JTabbedPane

کنترل JTabbedPane به شما اجازه می‌دهد که برای پنجره‌های خود سربرگ (Tab) بسازید، شما در برنامه‌های زیادی این نوع پنجره‌ها را مشاهده کرده‌اید. برای مثال پنجره‌ی Properties فایل‌ها.



JTabbedPane می‌تواند حاوی کنترل‌های دیگر باشد. با کلیک بر روی هر سربرگ (Tab) می‌توانید محتویات آنرا مشاهده کنید. ظاهر

یک سربرگ فعال با دیگر سربرگ‌ها متفاوت است، بنابراین شما می‌توانید متوجه شوید که کدام یک از سربرگ‌ها فعال است. زمانی که شما بر روی یک سربرگ کلیک می‌کنید، کنترل‌هایی که به آن تعلق دارند نمایش داده می‌شوند. JTabbedPane به شما اجازه می‌دهد که یک فرم را در داخل سربرگ‌های مختلف سازماندهی کنید، به طوری که هر سربرگ یک دسته را نمایش دهد. برای مثال، می‌توانید یک فرم که حاوی اطلاعات شخصی را در سربرگ Personal Info و پیش زمینه‌ی آموزشی Educational Background قرار دهید.

سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

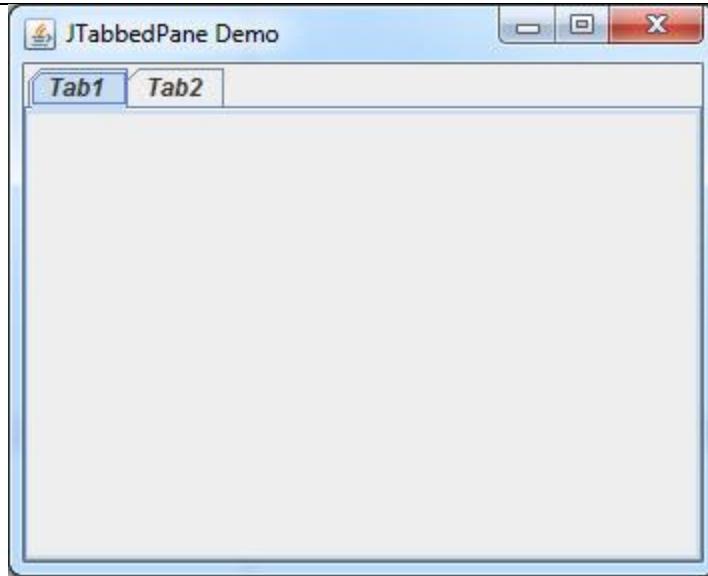
```
public JTabbedPane()
public JTabbedPane(int tabPlacement)
public JTabbedPane(int tabPlacement, int tabLayoutPolicy)
```

برای اضافه کردن یک کنترل TabControl می‌توان به صورت زیر عمل نمود :

```
1 package myfirstprogram;
2
3 import java.awt.GridLayout;
4 import javax.swing.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10        JFrame frame1 = new JFrame("JTabbedPane Demo");
11        JPanel panel1 = new JPanel();
12        JPanel panel2 = new JPanel();
13
14        JTabbedPane TabbedPane1 = new JTabbedPane();
15        TabbedPane1.addTab("Tab1", panel1);
16        TabbedPane1.addTab("Tab2", panel2);
17
18        frame1.add(TabbedPane1);
19        frame1.setSize(355, 286);
20        frame1.setVisible(true);
21    }
22}
```

همانطور که در خط ۱۴ کد بالا مشاهده می‌کنید ابتدا یک شیء از JTabbedPane ایجاد و سپس در خطوط ۱۵ و ۱۶ دو سربرگ یا tab به آن با استفاده از متد () addTab می‌کنیم. هر tab حداقل باید یک عنوان داشته باشد و یک کنترل دیگر را در خود جای دهد. که ما

در هر tab یک کنترل Panel قرار داده‌ایم :



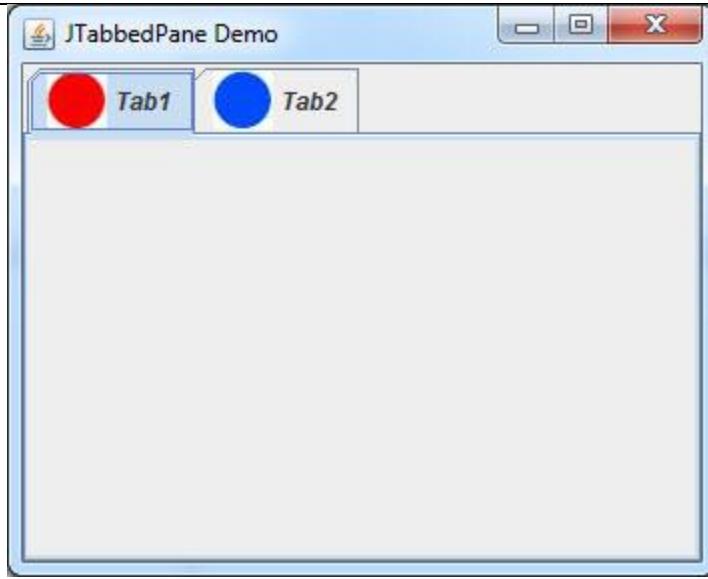
کلاس ImageIcon به شما اجازه می‌دهد که برای هر tab آیکونی را مشخص کنید. ابتدا دو عکس زیر را در درایو D ذخیره کنید :



سپس خطوط ۱۴ تا ۱۶ کد بالا را با کدهای زیر جایگزین کنید :

```
JTabbedPane TabbedPane1 = new JTabbedPane();
ImageIcon Icon1 = new ImageIcon("D:\\\\img1.jpg");
ImageIcon Icon2 = new ImageIcon("D:\\\\img2.jpg");
TabbedPane1.addTab("Tab1",Icon1,panel1);
TabbedPane1.addTab("Tab2",Icon2,panel2);
```

حال برنامه را اجرا و نتیجه را مشاهده کنید :



سازنده کلاس `JTabbedPane` (خط ۱۴) دو پارامتر می‌گیرد :

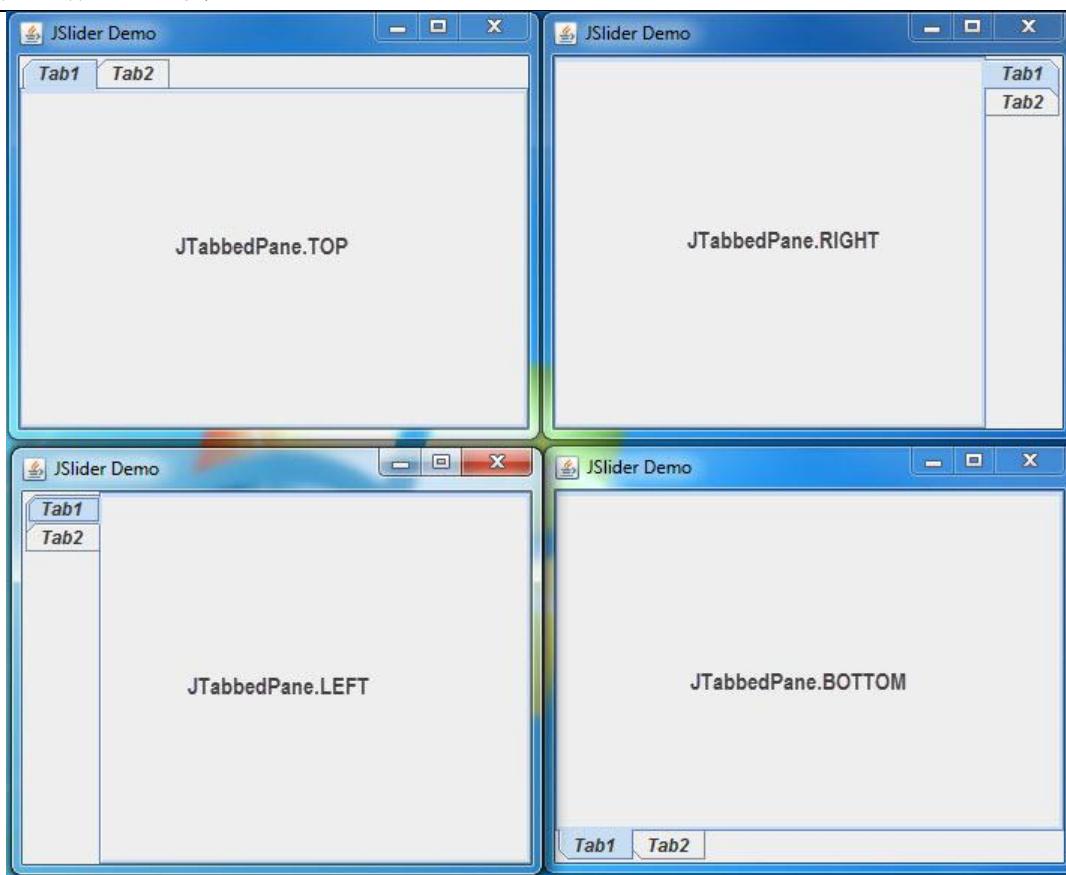
```
JTabbedPane TabbedPane1 = new JTabbedPane(tabPlacement, tabLayoutPolicy);
```

با استفاده از پارامتر اول می‌توان کنترل `JTabbedPane` را در قسمت بالا، راست، چپ و پایین فریم قرار داد. برای این کار کافیست یکی از

مقادیر زیر را در خط ۱۴ و در داخل پرانتز بنویسید :

- `JTabbedPane.TOP`
- `JTabbedPane.RIGHT`
- `JTabbedPane.LEFT`
- `JTabbedPane.BOTTOM`

خروجی پس از قرار دادن این مقادیر می‌تواند هر یک از حالات زیر باشد :

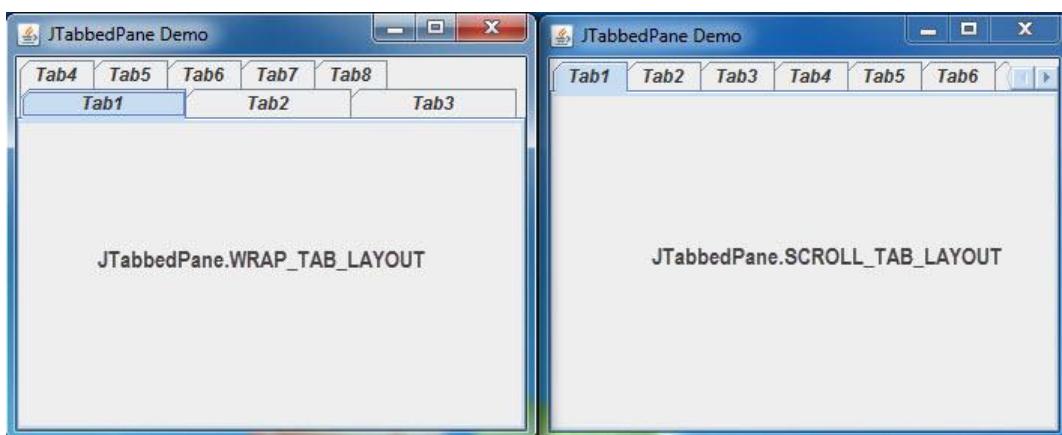


با استفاده از پارامتر دوم هم می‌توان Tab ها را در یک یا چند خط نمایش داد. پارامتر دوم می‌تواند شامل مقادیر زیر باشد :

- JTabbedPane.SCROLL_TAB_LAYOUT

- JTabbedPane.WRAP_TAB_LAYOUT

خروجی پس از قرار دادن این مقادیر می‌تواند هر یک از حالات زیر باشد :



حال فرض کنید که می‌خواهیم کنترل‌های دیگری در داخل این دو tab قرار دهیم. چون از قبل دو Panel به هر دو tab اضافه کردایم

کافیست که کنترل‌های دیگر را در این دو tab قرار دهیم. در خط ۱۳ کد بالا کدهای زیر را بنویسید :

```
panel1.setLayout(new GridLayout(2,2,10,10));
panel2.setLayout(new GridLayout(2,2,10,10));

JButton button1 = new JButton("Button1");
JButton button2 = new JButton("Button2");
JButton button3 = new JButton("Button3");
JButton button4 = new JButton("Button4");

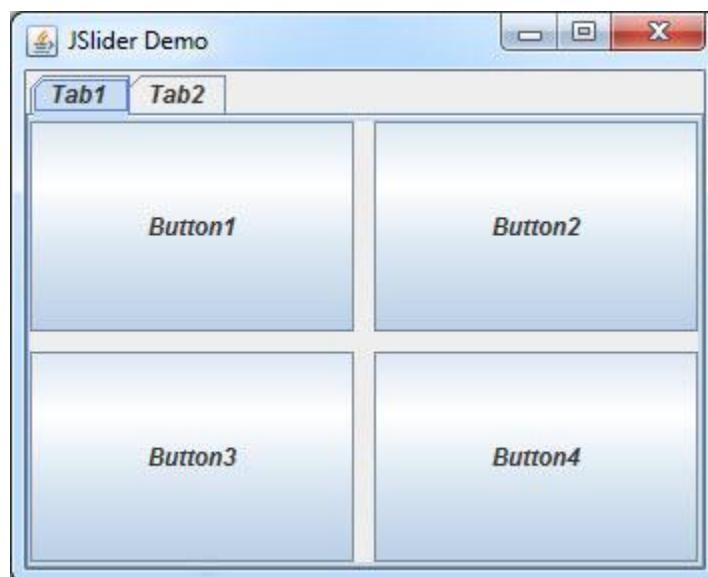
JCheckBox checkbox1 = new JCheckBox("CheckBox1");
JCheckBox checkbox2 = new JCheckBox("CheckBox2");
JCheckBox checkbox3 = new JCheckBox("CheckBox3");
JCheckBox checkbox4 = new JCheckBox("CheckBox4");

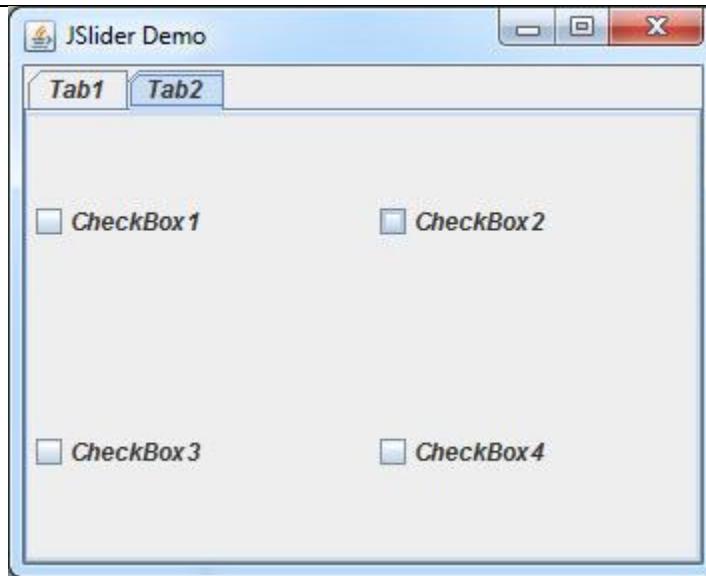
panel1.add(button1);
panel1.add(button2);
panel1.add(button3);
panel1.add(button4);

panel2.add(checkbox1);
panel2.add(checkbox2);
panel2.add(checkbox3);
panel2.add(checkbox4);
```

در این کدهای دو Panel را به دو سطر و دو ستون تقسیم کرده و در داخل panel1 چهار دکمه و در داخل panel2 چهار چک باکس قرار

دادهایم :



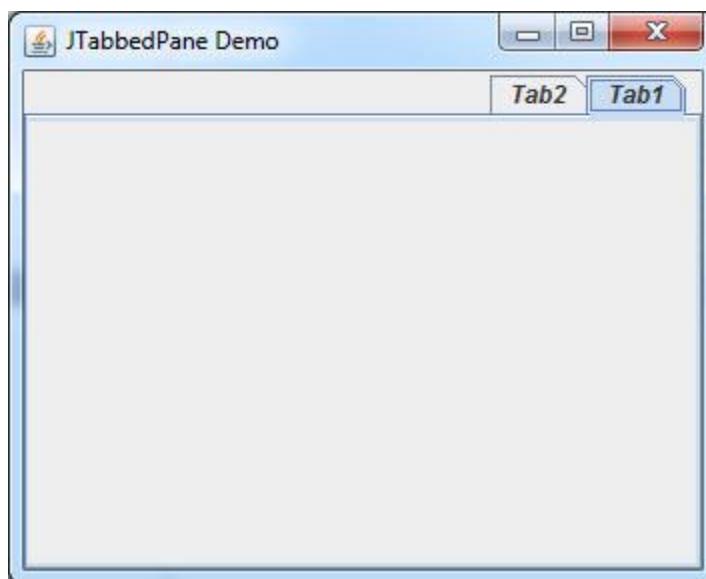


و در آخر هم این نکته را یاد آور شویم که برای راست چین کردن کنترل JTabbedPane می‌توان از متد (`setComponentOrientation()`)

به صورت زیر استفاده کرد :

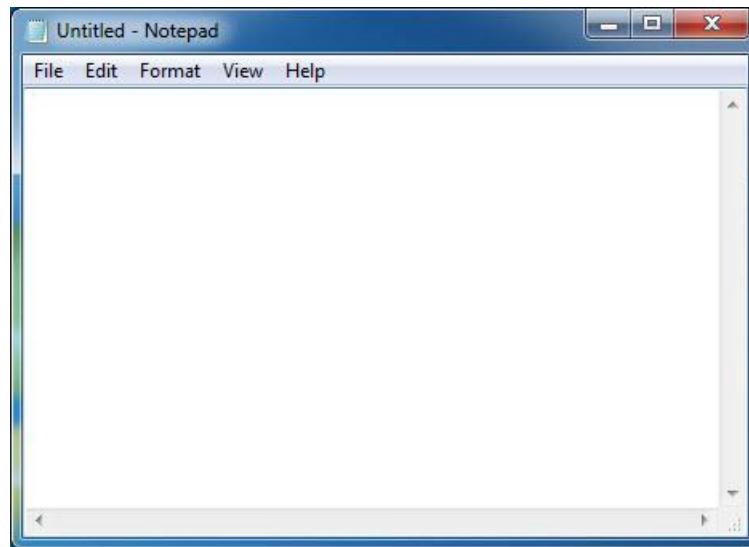
```
TabbedPane1.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
```

خروجی بعد از اعمال کد بالا می‌تواند به صورت زیر باشد :

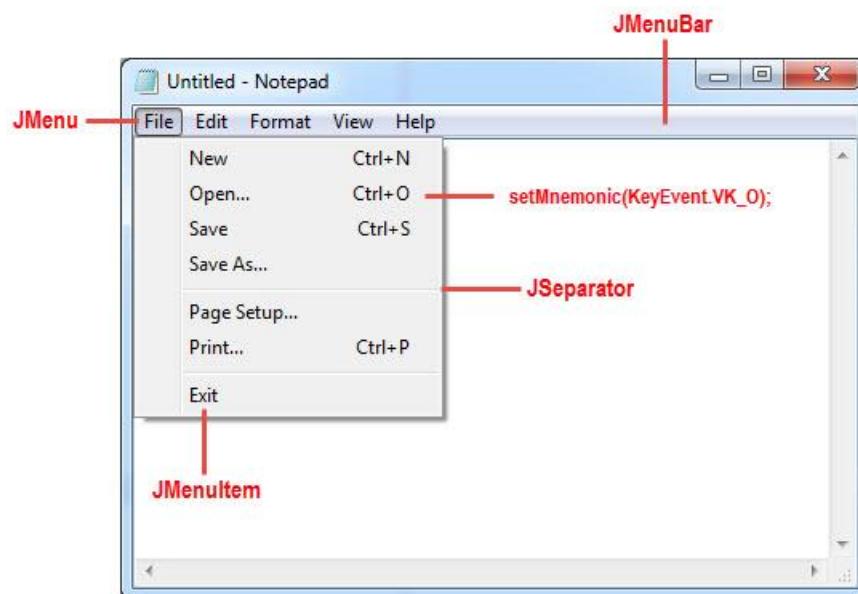


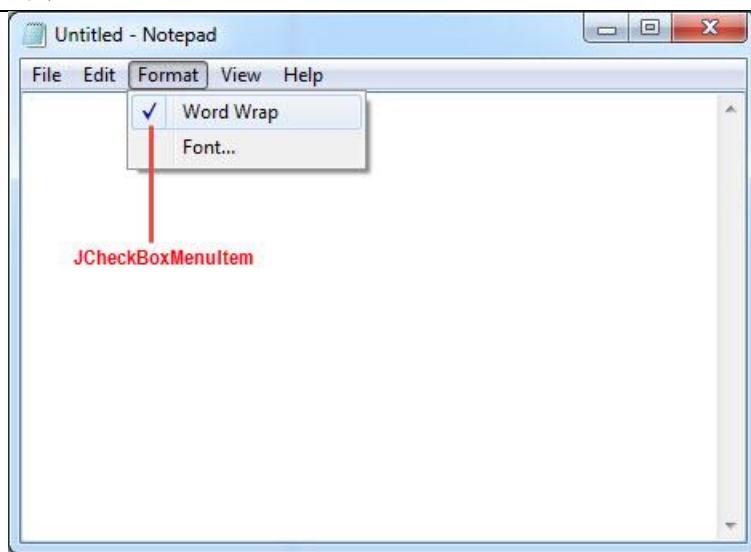
کنترل JMenuBar

از کنترل JMenuBar برای ساخت نوار منو استفاده می‌شود. در شکل زیر نمایی کلی از یک نوار منو را مشاهده می‌کنید :



در جاوا برای ایجاد یک نوار منو کامل از کلاس‌هایی که در شکل زیر مشاهده می‌کنید استفاده می‌شود :





حال اجازه دهید که وارد قسمت کدنویسی شده و نوار منوی بالا را شبیه سازی کنیم. به کد زیر توجه کنید :

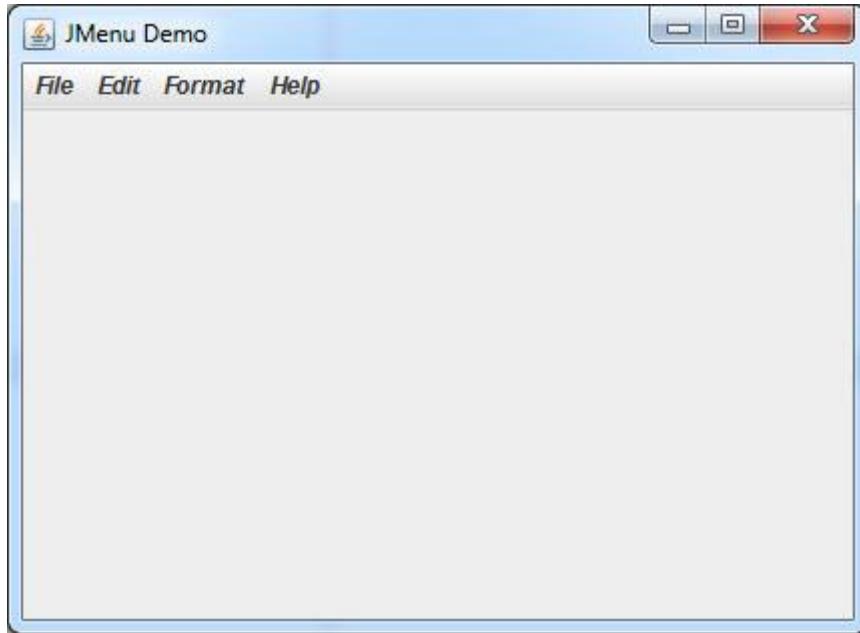
```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        JFrame frame1 = new JFrame("JMenu Demo");
12        frame1.setLayout(new BorderLayout());
13
14        JMenuBar menubar1 = new JMenuBar();
15        frame1.add(menubar1, BorderLayout.NORTH);
16
17        JMenu menu1 = new JMenu("File");
18        JMenu menu2 = new JMenu("Edit");
19        JMenu menu3 = new JMenu("Format");
20        JMenu menu4 = new JMenu("Help");
21
22        menubar1.add(menu1);
23        menubar1.add(menu2);
24        menubar1.add(menu3);
25        menubar1.add(menu4);
26
27        frame1.setSize(430 , 315);
28        frame1.setVisible(true);
29    }
30 }
```

در کد بالا و در خطوط ۱۱ یک فریم ایجاد کرده و در خط ۱۲ از کلاس BorderLayout برای لایه بندی آن استفاده می‌کنیم. در خط ۱۴ یک

نوار منو با استفاده از کلاس JMenuBar ایجاد کرده و آن را در خط ۱۵ به بالای فریم می‌چسبانیم. اگر خطوط ۱۷-۲۵ را پاک کرده و برنامه

را اجرا کنید مشاهده می‌کنید که هنوز کار خاصی انجام نشده است. برای اضافه کردن منو به منوبار از کلاس `JMenu` استفاده می‌شود. در خطوط ۱۷-۲۰ چهار منو ایجاد کرده و در خطوط ۲۲-۲۵ آن‌ها را به منوبار اضافه می‌کنیم. حال برنامه را اجرا و نتیجه را مشاهده کنید :

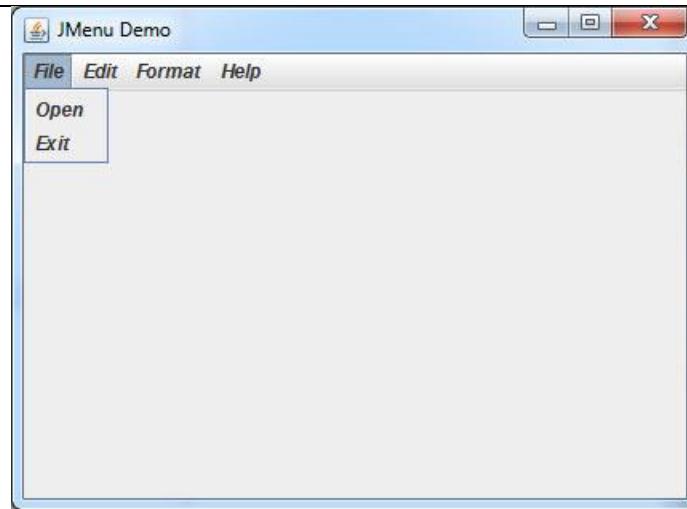


اکنون نوبت به زیر منوها می‌رسد. برای ایجاد زیر منو از کلاس `JMenuItem` استفاده می‌شود. فرض کنید که می‌خواهیم دو زیر منو به منوی `File` اضافه کنیم. برای این کار کدهای زیر از ۲۱ به بعد کد بالا بنویسید (تمامی کدهایی که در ادامه آموزش داده می‌شود را از همین خط به بعد بنویسید. شما می‌توانید بعد از تکمیل کدها آن‌ها را منظم کنید) :

```
JMenuItem menuItem1 = new JMenuItem("Open");
menu1.add(menuItem1);

JMenuItem menuItem2 = new JMenuItem("Exit");
menu1.add(menuItem2);
```

حال برنامه را اجرا و نتیجه را مشاهده کنید :



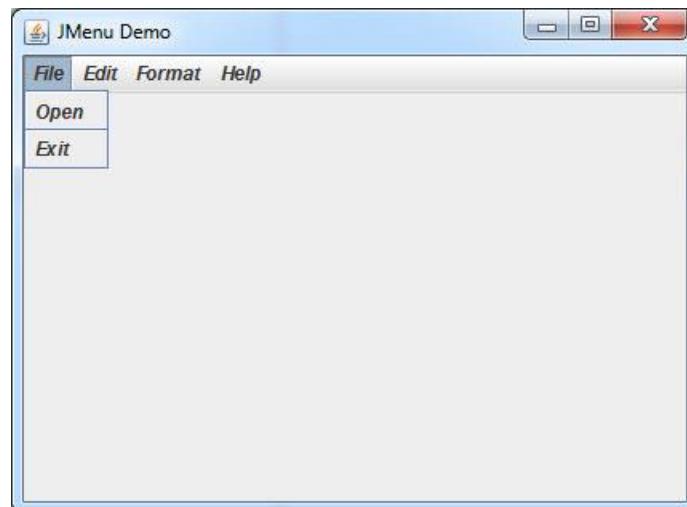
برای ایجاد خط جدا کننده هم از کلاس `JSeparator` استفاده می‌شود. کافیست که یک شیء از این کلاس را در بین دو زیر منو قرار دهید

```
JMenuItem menuItem1 = new JMenuItem("Open");
menu1.add(menuItem1);

menu1.add(new JSeparator());

JMenuItem menuItem2 = new JMenuItem("Exit");
menu1.add(menuItem2);
```

: برنامه را دوباره اجرا و نتیجه را مشاهده کنید



همانطور که مشاهده کردید برای ایجاد زیر منو از کلاس `JMenuItem` استفاده می شود ولی اگر بخواهیم برای یک زیر منو، زیر منوی دیگری

تعریف کنیم باید زیر منوی اصلی را با استفاده از کلاس `JMenu` ایجاد کنیم. برای درک بهتر کد بالا را با کدهای زیر جایگزین کنید :

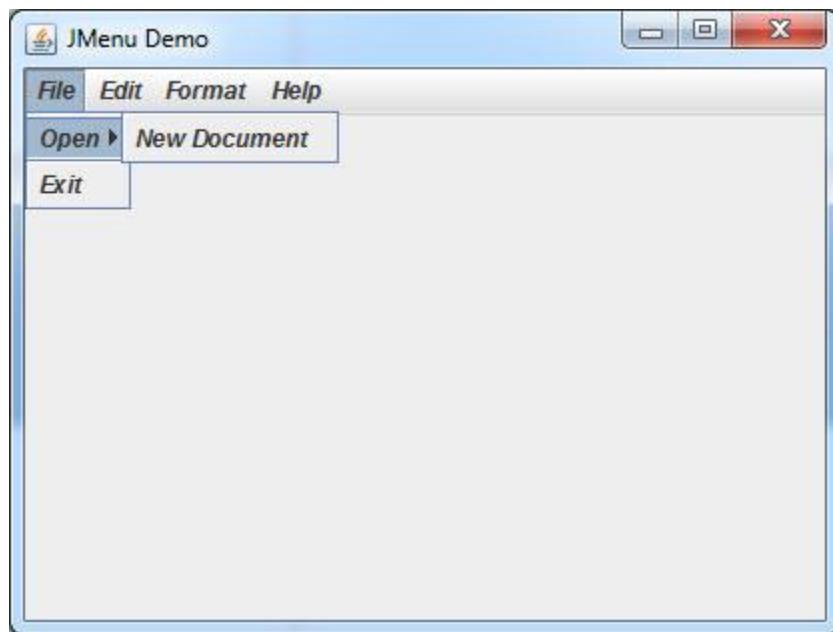
```
JMenu menuItem1 = new JMenu("Open");
menuItem1.add(menuItem1);
JMenuItem submenuItem1 = new JMenuItem("New Document");
menuItem1.add(submenuItem1);

menuItem1.add(new JSeparator());

JMenuItem menuItem2 = new JMenuItem("Exit");
menuItem1.add(menuItem2);
```

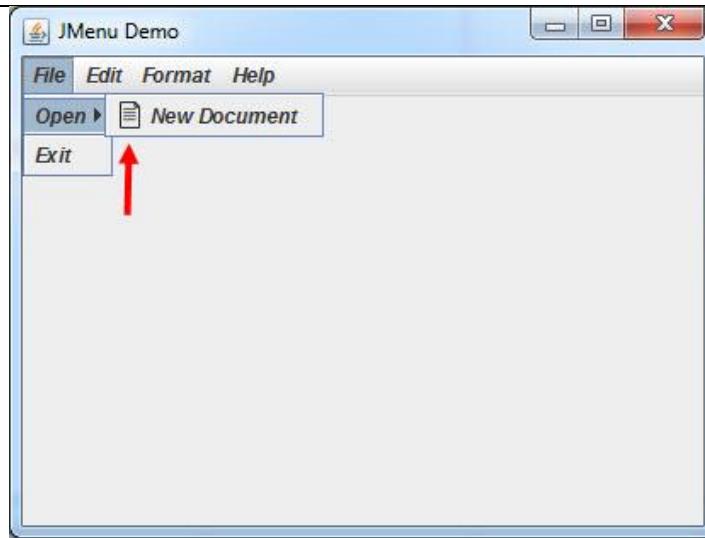
همانطور که در کد بالا مشاهده می کنید ابتدا منوی `Open` را با استفاده از کلاس `JMenu` ایجاد و سپس با استفاده از کلاس `JMenuItem`

زیر منوی `New Document` را به آن اضافه کرده ایم :



سازنده کلاس `JMenuItem` یک پارامتر دوم هم دریافت می کند که با استفاده از آن می توان یک آیکون به زیر منو اختصاص داد :

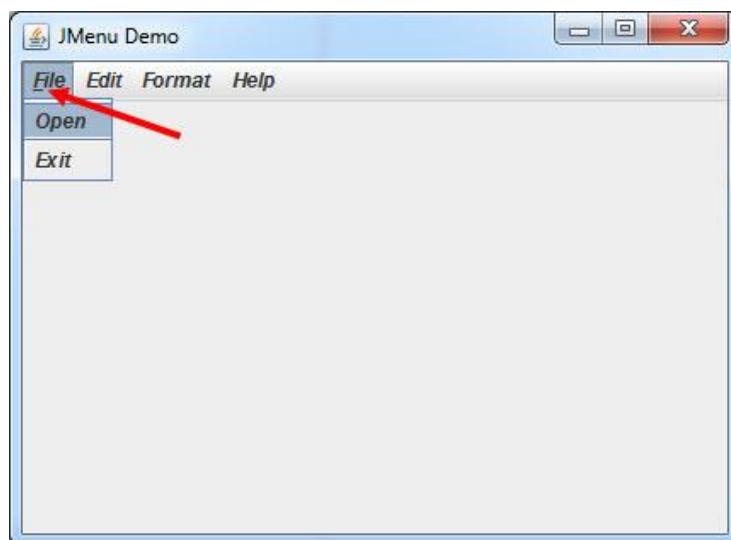
```
JMenuItem submenuItem1 = new JMenuItem("New Document", new
ImageIcon("D:\\images\\\\new.png"));
```



برای ایجاد کلید میانبر و استفاده از کلیدهای ترکیبی هم از متده استفاده می‌شود. فرض کنید که می‌خواهیم با زدن دکمه‌های Alt+F زیر منوهای منوی فایل نمایش داده شود. برای این کار کد زیر را می‌نویسید :

```
menu1.setMnemonic(KeyEvent.VK_F);
```

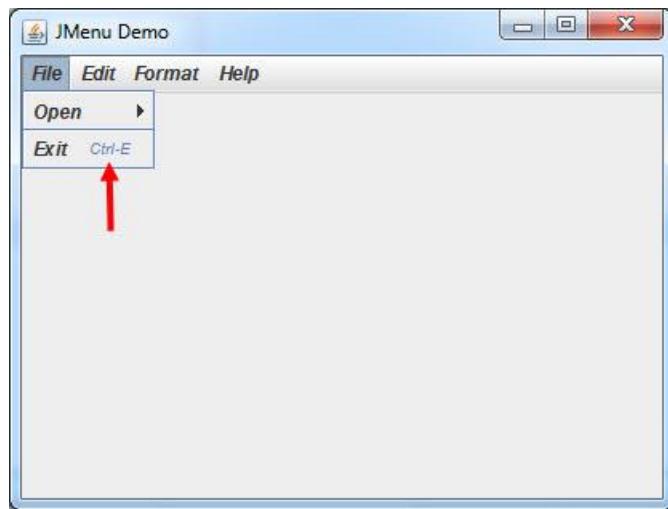
اگر برنامه را اجرا کنید مشاهده می‌کنید که یک خط تیره زیر حرف F کشیده شده است که نشان دهنده این است که شما می‌توانید از کلید Alt به همراه حرف F برای باز شدن منو استفاده کنید. برنامه را اجرا و دکمه F را بزنید :



برای ایجاد ایجاد کلید میانبر با استفاده از دکمه کنترل (Ctrl) هم می‌توانید از کلاس KeyStroke استفاده کنید. فرض کنید که می‌خواهیم با زدن دکمه‌های Ctrl+O زیر منوی Exit (یا همان menuItem2) عمل کند. برای این کار از کد زیر استفاده می‌کنیم :

```
KeyStroke keystroke = KeyStroke.getKeyStroke("control X");
menuItem2.setAccelerator(keystroke);
```

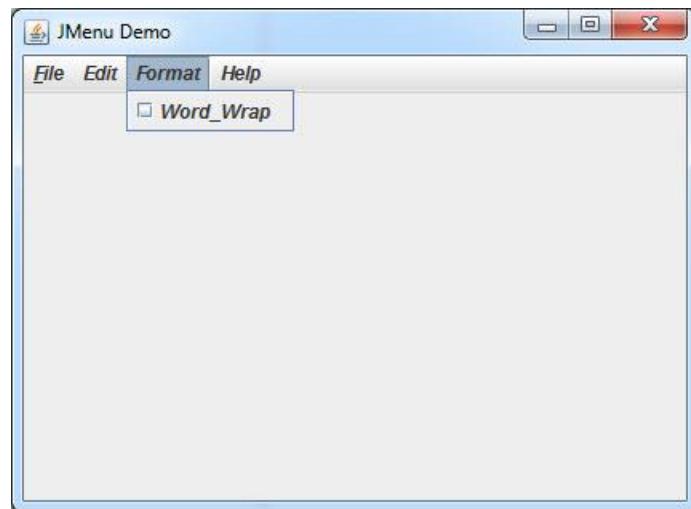
برنامه را اجرا و نتیجه را مشاهده کنید :



برای ایجاد یک زیر منو از نوع `JCheckBoxMenuItem` هم از کلاس `JCheckBoxMenuItem` استفاده می‌شود. فرض کنید که می‌خواهیم یک زیر منو از همین نوع به منوی `Format` (یا همان `menu3`) اضافه کنیم. برای اینکار به صورت زیر عمل می‌کنیم :

```
JCheckBoxMenuItem checkboxmenuItem1 = new JCheckBoxMenuItem("Word_Wrap");
menu3.add(checkboxmenuItem1);
```

برنامه را اجرا و بر روی منوی `Format` کلیک کنید :



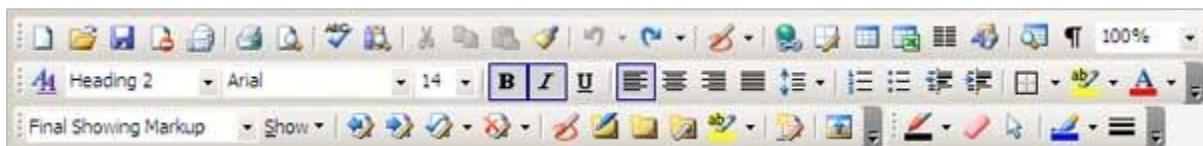
کنترل JToolBar

از کنترل `JToolBar` برای ایجاد نوار ابزار استفاده می‌شود. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
JToolBar()
JToolBar(int orientation)
JToolBar(String name)
JToolBar(String name, int orientation)
```

نوار ابزار شامل دکمه‌ها و اجزای مفیدی برای راحتی کار کاربر می‌باشد. مانند منو، نوار ابزار هم در تعداد زیادی از نرم افزارهای مشهور مانند

Microsoft Office 2003 به چشم می‌خورد. در زیر نوار ابزار Microsoft Office 2003 را مشاهده می‌کنید :



در اصل نوار ابزار را می‌توان منو و زیر منوهایی فرض کرد که به دلیل کاربرد زیاد آن‌ها، برای در دسترس بودن همیشگی، آن‌ها را به صورت

دکمه و در زیر نوار قرار می‌دهند. قبل از هر چیز ابتدا آیکون‌های برنامه را از لینک زیر دانلود کرده و در پوشه‌ای به نام `images` در درایو D

ذخیره کنید :

```
http://www.w3-farsi.com/?p=15248
```

برای درک بهتر کار با نوار ابزار و کنترل `JToolBar` در جاوا به کد زیر توجه کنید :

```
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         JFrame frame1 = new JFrame("JToolBar Demo");
11         frame1.setLayout(new BorderLayout());
12
13         JToolBar firstToolbar = new JToolBar();
14         JToolBar secondToolbar = new JToolBar();
15
16         JPanel panel = new JPanel();
17         panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
18
19         JButton New = new JButton(new ImageIcon("D:\\images\\new.png"));
```

```
20 JButton Open      = new JButton(new ImageIcon("D:\\images\\open.png")));
21 JButton Save     = new JButton(new ImageIcon("D:\\images\\save.png"));
22 JButton Cut      = new JButton(new ImageIcon("D:\\images\\cut.png"));
23 JButton Copy     = new JButton(new ImageIcon("D:\\images\\copy.png"));
24 JButton Print    = new JButton(new ImageIcon("D:\\images\\print.png"));
25 JButton Left2right = new JButton(new ImageIcon("D:\\images\\left2right.png"));
26 JButton Right2left = new JButton(new ImageIcon("D:\\images\\right2left.png"));
27 JButton Center   = new JButton(new ImageIcon("D:\\images\\center.png"));
28 JButton Right    = new JButton(new ImageIcon("D:\\images\\right.png"));
29 JButton Left     = new JButton(new ImageIcon("D:\\images\\left.png"));

30
31 firstToolbar.add(New);
32 firstToolbar.add(Open);
33 firstToolbar.add(Save);
34 firstToolbar.add(Cut);
35 firstToolbar.add(Copy);
36 firstToolbar.add(Print);
37 firstToolbar.setAlignmentX(0);

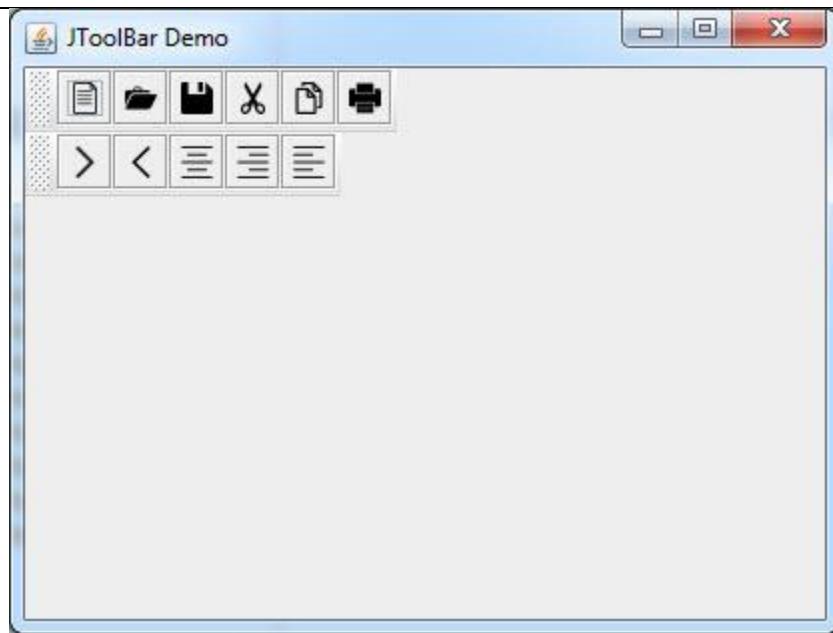
38
39 secondToolbar.add(Left2right);
40 secondToolbar.add(Right2left);
41 secondToolbar.add(Center);
42 secondToolbar.add(Right);
43 secondToolbar.add(Left);
44 secondToolbar.setAlignmentX(0);

45
46 panel.add(firstToolbar);
47 panel.add(secondToolbar);
48 frame1.add(panel, BorderLayout.NORTH);

49
50 frame1.setSize(416, 313);
51 frame1.setVisible(true);
52
53 }
```

در کد بالا و در خطوط ۱۳ و ۱۴ دو نوار ابزار ایجاد کردہ‌ایم. در خط ۱۶ و ۱۷ یک Panel ایجاد و نحوه لایه بندی آن را BoxLayout راستای محور ۷ ها قرار داده‌ایم. چون که می‌خواهیم دو نوار در زیر هم قرار گیرند. در خطوط ۱۹-۲۹ چندین دکمه ایجاد کرده و با استفاده از کلاس ImageIcon به هر کدام از آن‌ها یکی از عکس‌هایی که در درایو D ذخیره کردیم را اختصاص می‌دهیم. در خطوط ۳۱-۳۶ چندین دکمه را در نوار ابزار اول و در خطوط ۳۹-۴۳ مابقی دکمه‌ها را در نوار ابزار دوم قرار می‌دهیم. بدون خطوط ۳۷ و ۴۴ نوار ابزارها در وسط قرار می‌گیرند که ما در این دو خط و با استفاده از متده استفاده از متد setAlignmentX آن‌ها را به سمت چپ Panel منتقل می‌کنیم. در خطوط ۴۶ و ۴۷ دو نوار ابزار را به Panel و در خط ۴۸ خود Panel را به قسمت بالای Frame اضافه می‌کنیم. برنامه را اجرا و نتیجه را

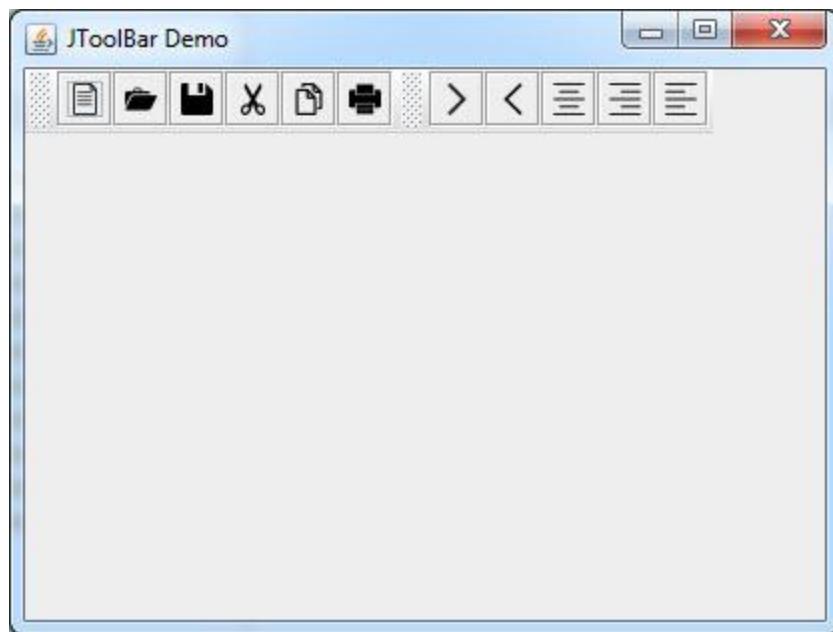
مشاهده کند :



اگر بخواهید که نوارهای ابزارتان در یک خط نمایش داده شوند کافیست که خط ۱۷ کد بالا را به صورت زیر تغییر دهید :

```
panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));
```

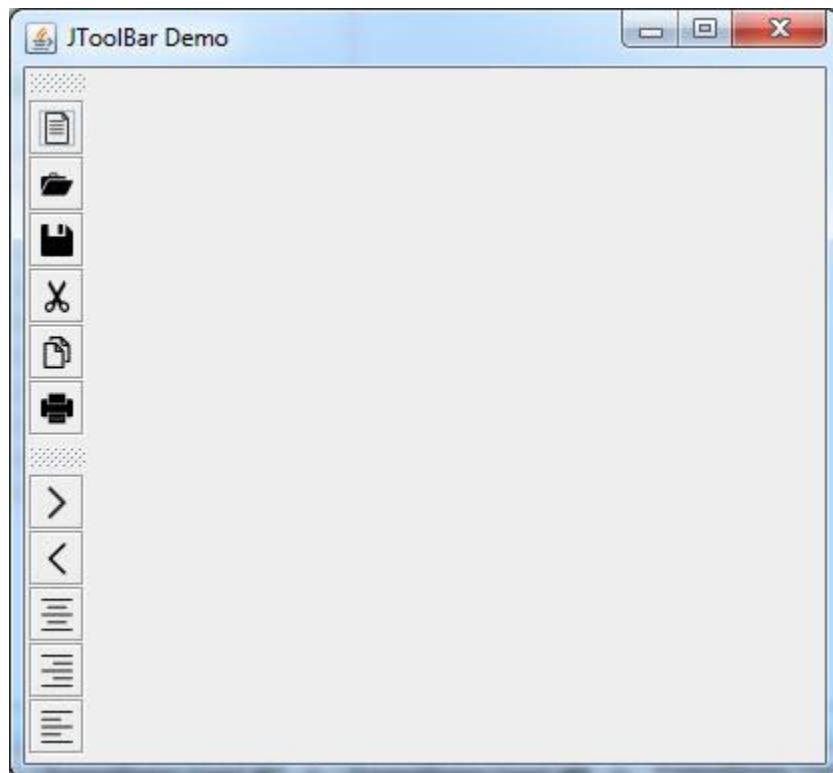
حال برنامه را اجرا و نتیجه را مشاهده کنید :



نوار ابزار را به صورت عمودی هم می‌شود ایجاد کرد. برای این کار کافیست که خطوط ۱۳ و ۱۴ کد بالا را به صورت زیر تغییر دهید :

```
JToolBar firstToolbar = new JToolBar(JToolBar.VERTICAL);
JToolBar secondToolbar = new JToolBar(JToolBar.VERTICAL);
```

برنامه را اجرا کنید :

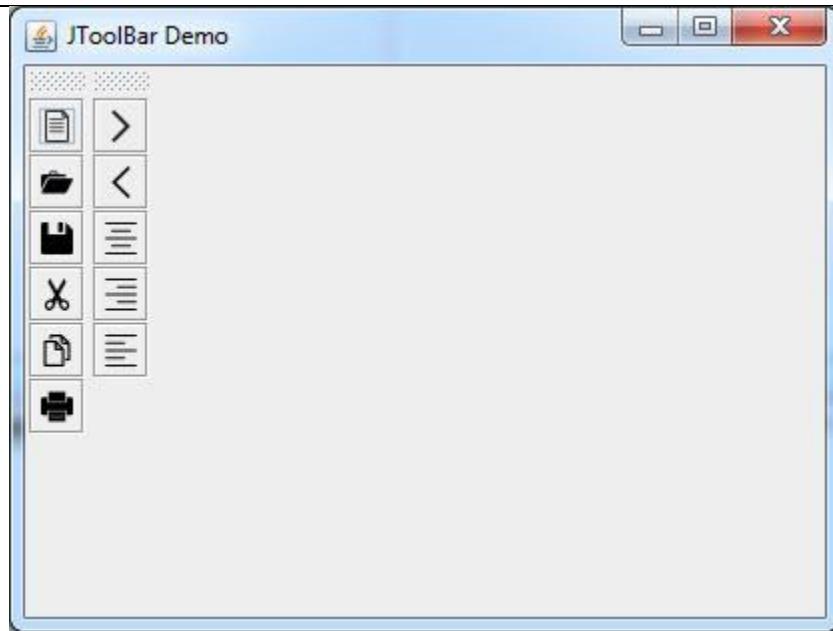


مشاهده می‌کنید که نوار ابزارها در زیر هم قرار گرفتند و این به خاطر لایه بندی Panel در خط ۱۷ می‌باشد. حال خطوط ۱۷، ۳۷ و ۴۴ را به

صورت زیر تغییر دهید :

```
panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));
firstToolbar.setAlignmentY(0);
secondToolbar.setAlignmentY(0);
```

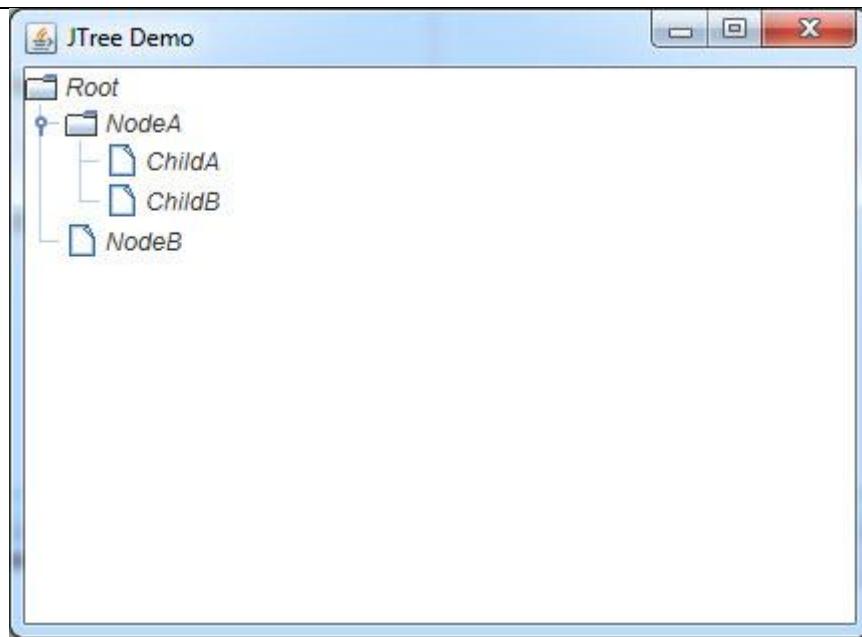
بار دیگر برنامه را اجرا و نتیجه را مشاهده کنید :



JTree کنترل

کنترل JTree برای نمایش درختی آیتم‌ها به کار می‌رود. برای درک بهتر عملکرد این کنترل می‌توان به فهرست مطالب یک کتاب و یا نمایش فایل‌ها و یا پوشه‌های درون هارد اشاره کرد. هر شاخه از کنترل JTree یک گره یا Node می‌نامند. پس به جای شاخه می‌توان کلمه گره را هم به کار برد.

هر شاخه در این کنترل شامل یک برجسب و یک عکس اختیاری است و هر شاخه خود می‌تواند شامل زیرشاخه‌های دیگر باشد. با کلیک بر روی علامت کنار شاخه، کاربر می‌تواند زیرشاخه‌های مربوطه را مشاهده کند. در شکل زیر یک کنترل JTree ساده با چند شاخه نمایش داده شده است. همانطور که مشاهده می‌کنید شاخه اصلی یا Root Node، شاخه اصلی (Root Node) است که دارای دو زیرشاخه به نام‌های ChildA و ChildB می‌باشد. NodeA نیز به نوبه خود دارای دو زیرشاخه به نام‌های NodeB و NodeC است :



برای ایجاد یک درخت در جاوا از کنترل `JTree` و برای ایجاد شاخه‌های آن از کلاس `DefaultMutableTreeNode` استفاده می‌شود.

برای روشن شدن مطلب به مثال زیر توجه کنید. در مثال زیر می‌خواهیم همین شکل بالا را ایجاد کنیم:

```

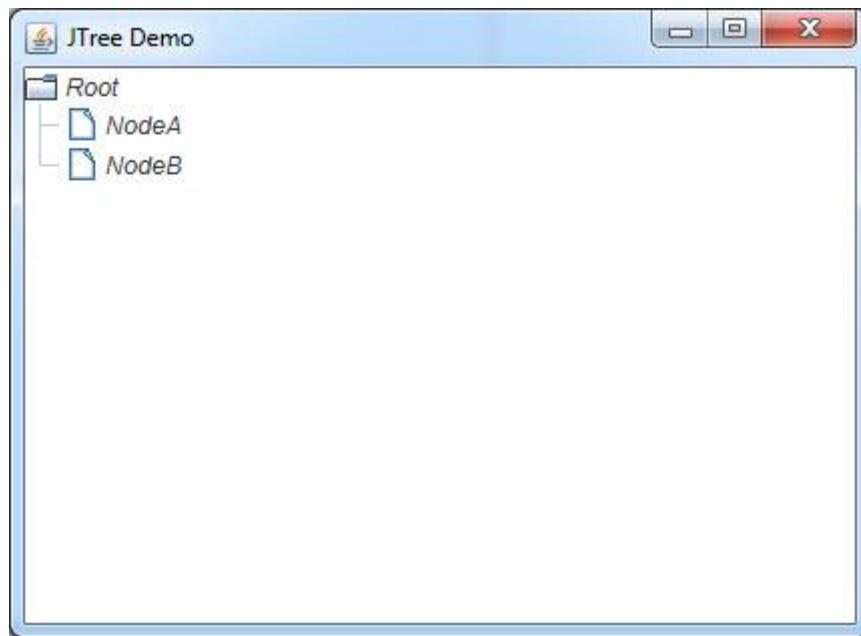
1 package myfirstprogram;
2
3 import javax.swing.*;
4 import javax.swing.tree.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args)
9     {
10         JFrame frame = new JFrame("JTree Demo");
11
12         DefaultMutableTreeNode RootNode = new DefaultMutableTreeNode("Root");
13
14         DefaultMutableTreeNode NodeA = new DefaultMutableTreeNode("NodeA");
15         RootNode.add(NodeA);
16
17         DefaultMutableTreeNode NodeB = new DefaultMutableTreeNode("NodeB");
18         RootNode.add(NodeB);
19
20         JTree Tree = new JTree(RootNode);
21
22         frame.add(Tree);
23         frame.setSize(430 , 315);
24         frame.setVisible(true);
25     }
26 }
```

همانطور که در کد بالا مشاهده می‌کنید ابتدا یک شاخه اصلی یا گره ریشه به نام RootNode ایجاد کردہ‌ایم. حال می‌خواهیم دو زیر شاخه

به آن اضافه کنیم. با استفاده از کلاس DefaultMutableTreeNode دو شاخه به نام‌های NodeA و NodeB ایجاد (خطوط ۱۶ و ۱۷) و

این دو شاخه را با استفاده از متد (add) در خطوط ۱۵ و ۱۸ به شاخه اصلی اضافه می‌کنیم. سپس در خط ۲۰ شاخه اصلی را به درخت و

خود درخت را در خط ۲۲ به فرم اضافه می‌کنیم :



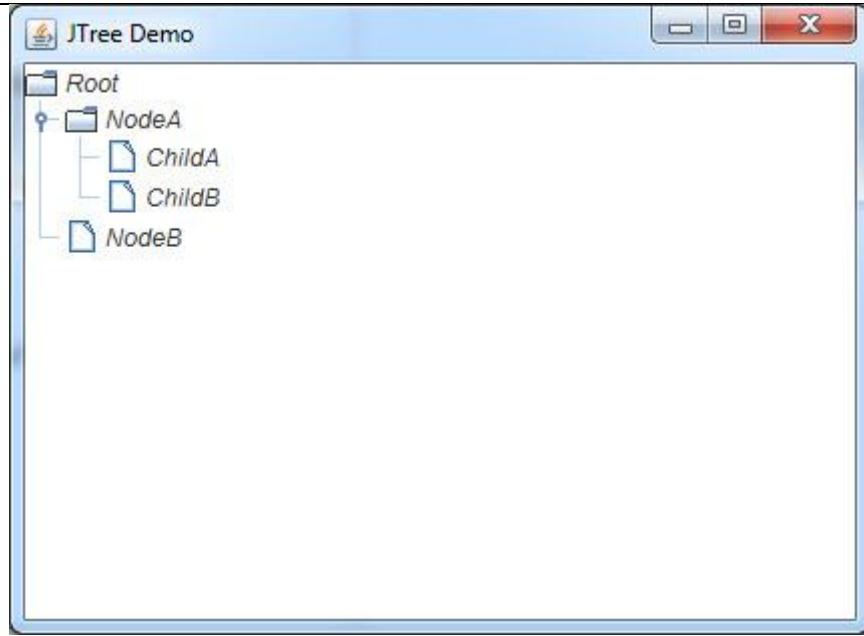
حال فرض کنید که می‌خواهیم دو زیر شاخه به شاخه NodeA در شکل بالا اضافه کنیم :

```
DefaultMutableTreeNode ChildA = new DefaultMutableTreeNode("ChildA");
NodeA.add(ChildA);

DefaultMutableTreeNode ChildB = new DefaultMutableTreeNode("ChildB");
NodeA.add(ChildB);
```

همانطور که در کد بالا مشاهده می‌کنید ابتدا با استفاده از کلاس DefaultMutableTreeNode دو زیر شاخه ایجاد و سپس با استفاده

از متد (add) آن‌ها را به NodeA اضافه می‌کنیم :



اگر بخواهید به هر یک از شاخه‌های بدون زیر شاخه و شاخه‌هایی که دارای زیر شاخه هستند یک آیکون اختصاص دهید می‌توانید از

: به همراه سه متده زیر استفاده کنید DefaultTreeCellRenderer

- برای نمایش آیکون شاخه‌های بدون زیر شاخه به کار می‌رود . setLeafIcon()
- برای نمایش آیکون شاخه‌های دارای زیر شاخه در حالت بسته به کار می‌رود . setClosedIcon()
- برای نمایش آیکون شاخه‌های دارای زیر شاخه در حالت باز به کار می‌رود . setOpenIcon()

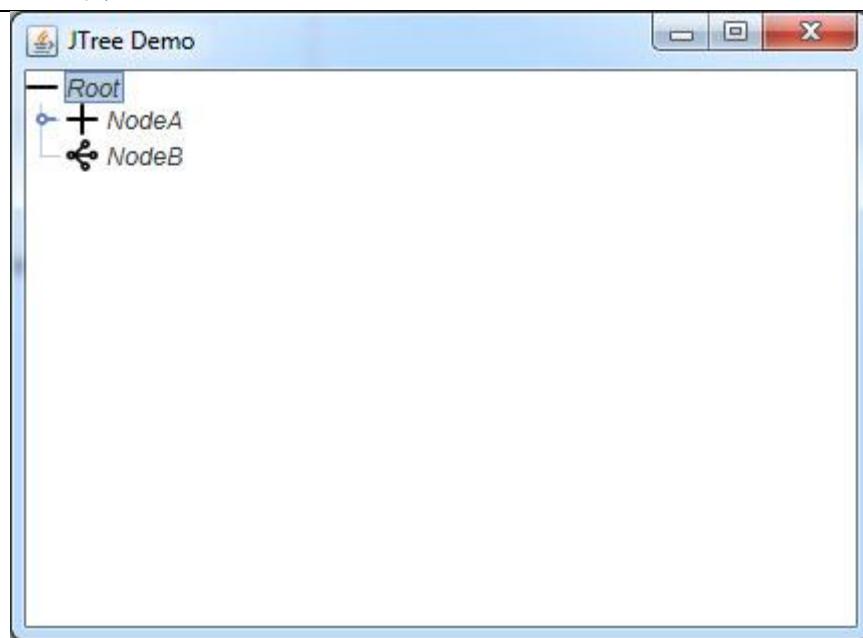
بعد از خط ۲۰ کد ابتدای درس کدهای زیر را بنویسید :

```
 ImageIcon leafIcon    = new ImageIcon("D:\\images\\node.png");
 ImageIcon openIcon   = new ImageIcon("D:\\images\\sub.png");
 ImageIcon closedIcon = new ImageIcon("D:\\images\\sum.png");

 DefaultTreeCellRenderer DefaultTreeCellRenderer1 =
 (DefaultTreeCellRenderer)Tree.getCellRenderer();
 DefaultTreeCellRenderer1.setLeafIcon(leafIcon);
 DefaultTreeCellRenderer1.setClosedIcon(closedIcon);
 DefaultTreeCellRenderer1.setOpenIcon(openIcon);
```

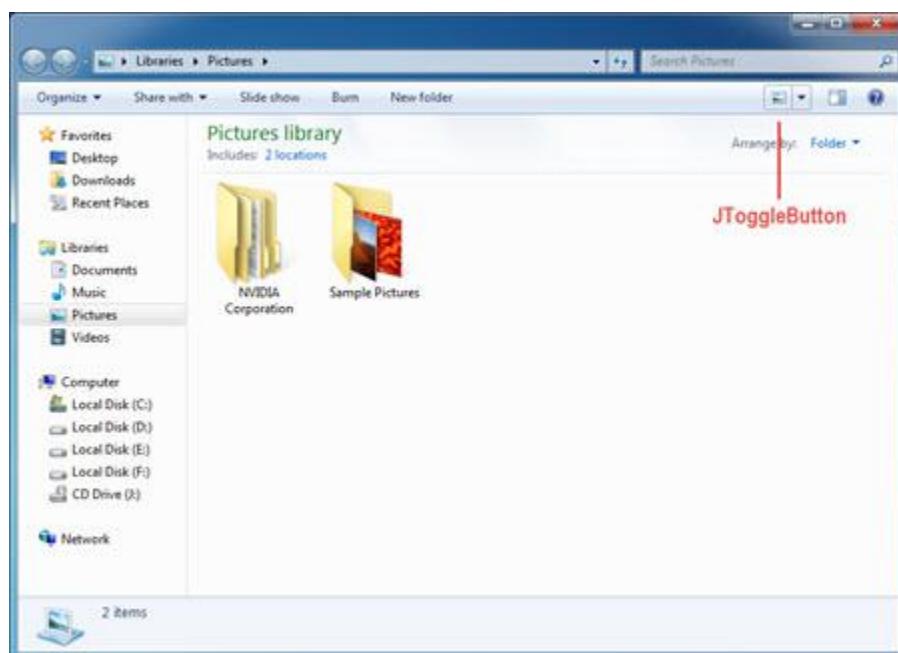
در خطوط بالا ابتدا مسیر سه آیکون را در سه شیء از کلاس ImageIcon قرار داده‌ایم. سپس این سه شیء را به سه متند از کلاس

: ارسال کرده‌ایم. برنامه را اجرا کنید DefaultTreeCellRenderer



JToggleButton کنترل

کنترل JToggleButton دو دکمه‌ای دو انتخابه یا دو حالت می‌باشد. این دکمه‌ها با کلیک بر روی آن‌ها به حالت انتخاب درآمد و رنگ پس زمینه آن‌ها تغییر می‌کند و اگر بخواهید آن‌ها را از حالت انتخاب خارج کنید باید دوباره بر روی آن‌ها کلیک کنید. یک نمونه از این نوع دکمه‌ها را در شکل زیر مشاهده می‌کنید :



سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JToggleButton()
public JToggleButton(Icon icon)
public JToggleButton(Icon icon, boolean selected)
public JToggleButton(String text)
public JToggleButton(String text, boolean selected)
public JToggleButton(String text, Icon icon)
public JToggleButton(String text, Icon icon, boolean selected)
public JToggleButton(Action action)
```

برای درک بهتر عملکرد کنترل JToggleButton به مثال زیر توجه کنید :

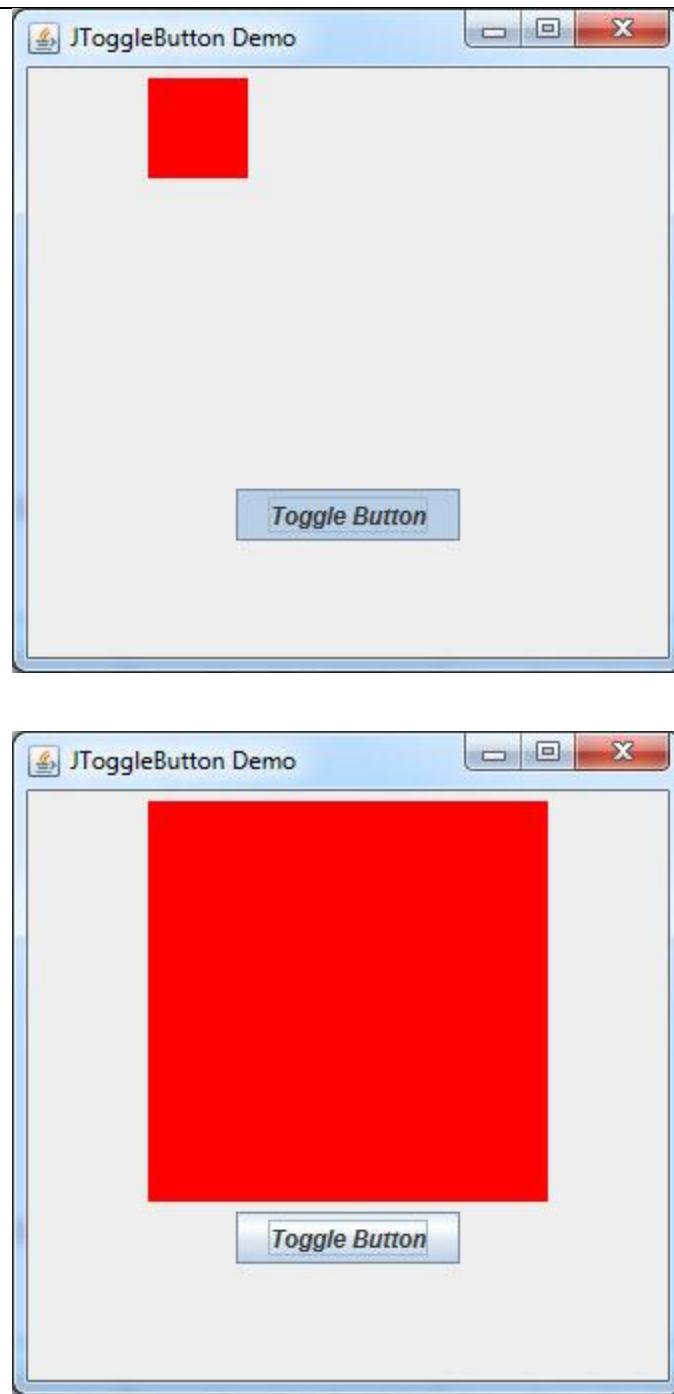
```
1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        JFrame frame = new JFrame("JToggleButton Demo");
12
13        JPanel panel = new JPanel();
14
15        final JLabel label = new JLabel();
16        label.setPreferredSize(new Dimension(200, 200));
17        label.setBackground(Color.red);
18        label.setOpaque(true);
19
20        JToggleButton toggleButton = new JToggleButton("Toggle Button");
21
22        panel.add(label);
23        panel.add(toggleButton);
24
25        frame.add(panel);
26
27        class stateChange implements ItemListener
28        {
29            public void itemStateChanged(ItemEvent itemEvent)
30            {
31                int state = itemEvent.getStateChange();
32
33                if (state == ItemEvent.SELECTED)
34                {
35                    label.setSize(50, 50);
36                }
37                else
38                {
39                    label.setSize(200, 200);
40                }
41            }
42        }
43    }
44}
```

```

42 }
43
44     toggleButton.addItemListener(new stateChange());
45
46     frame.setSize(500, 500);
47     frame.setVisible(true);
48 }
49 }
```

هدف از برنامه بالا این است که با کلیک بر روی یک دکمه `JToggleButton` اندازه `Label` تغییر کرده و با کلیک دوباره به حالت قبل بر گردد. در خطوط ۱۵-۱۸ یک `Label` ایجاد کرده و یک اندازه و رنگ پس زمینه هم به آن اختصاص می‌دهیم. در خط ۲۰ یک کنترل `JToggleButton` ایجاد کرده و این دو کنترل را در خطوط ۲۲ و ۲۳ به `Panel` و `Panel` را در خط ۲۵ به `Frame` اضافه می‌کنیم. همانطور که اشاره کردیم این دکمه‌ها دو وضعیت (`state`) دارند: ۱- انتخاب شده ۲- انتخاب نشده.

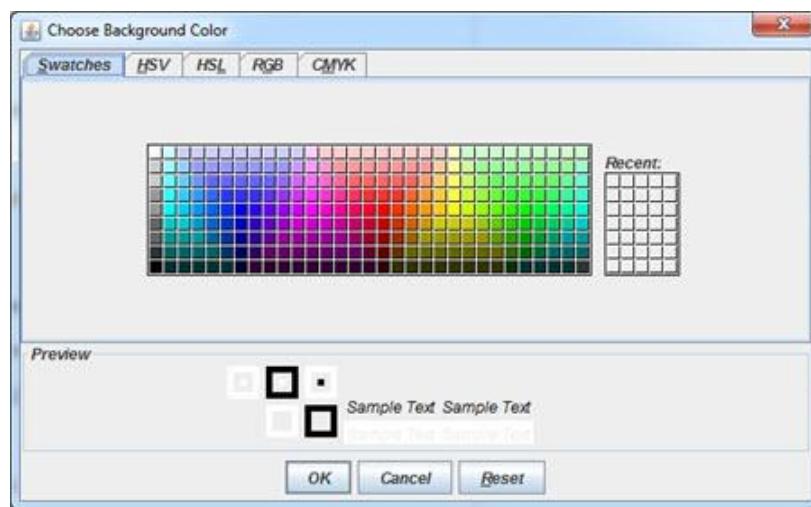
برای کار با این کنترل می‌توان رابط `ItemListener` را پیاده سازی کرد. در خطوط ۲۷-۴۲ یک کلاس با نام `stateChanege` ایجاد کرده و رابط مذکور را پیاده سازی می‌کنیم. این رابط دارای متodi به نام `itemStateChanged()` که باید پیاده سازی شود. همانطور که در خط ۲۹ مشاهده می‌کنید این متodi یک شیء از کلاس `ItemEvent` در یافت می‌کند. کلاس `ItemEvent` دارای `ItemEvent` متodi به نام `getStateChange()` که وضعیت دکمه را می‌توان با آن مشخص کرد. در خط ۳۱ وضعیت دکمه را به دست می‌آوریم. در خطوط ۳۳-۴۰ با استفاده از یک دستور `if` چک می‌کنیم که آیا دکمه انتخاب شده است؟ اگر انتخاب شده باشد اندازه `Label` را تغییر داده و در غیر اینصورت به همان اندازه قبل که در خط ۱۶ مشخص کرده‌ایم بر می‌گردانیم:



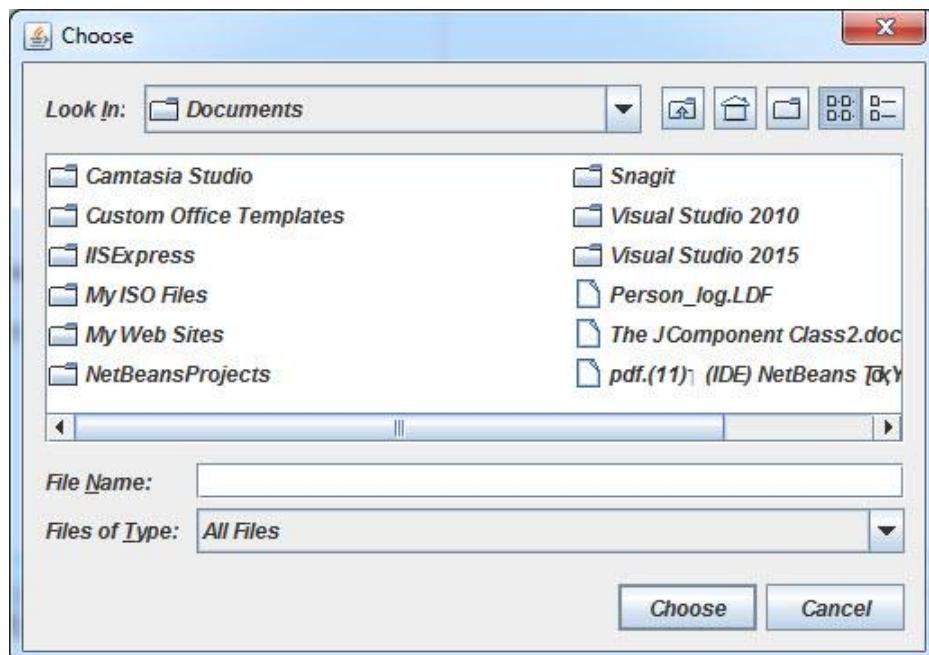
همانطور که در اشکال بالا مشاهده می‌کنید با انتخاب دکمه (شکل اول) رنگ پس زمینه JToggleButton تغییر می‌کند که نشان دهنده در حالت انتخاب بودن آن است و با کلیک دوباره بر روی آن یعنی خروج از حالت انتخاب رنگ پس زمینه به حالت عادی بر می‌گردد.

کادرهای محاوره‌ای (Dialogs)

کادرهای محاوره‌ای یا Dialog ها پنجره‌هایی هستند که دارای وظایف مشخصی مانند ذخیره یا باز کردن یک فایل، انتخاب یک رنگ و چاپ یک سند می‌باشند. جاوا دارای کنترل‌هایی می‌باشد که به شما اجازه می‌دهند با فراخوانی کادرهای محاوره‌ای امور خاصی را انجام دهید. کادرهای محاوره‌ای را می‌توان با استفاده از خواص این کنترل‌ها سفارشی کرد.



کادر محاوره‌ای انتخاب رنگ



کادر محاوره‌ای جستجوی پوشه‌ها و فایل‌ها

بنابراین می‌توانید با استفاده از این کادرهای محاوره‌ای برنامه‌های استانداردتری ایجاد کنید. این کادرها معمولاً در تمام برنامه‌های

ویندوزی مورد استفاده قرار می‌گیرند.

JFileChooser کنترل

از کنترل JFileChooser برای کار با فایل‌ها و پوشش‌ها استفاده می‌شود. با استفاده از این کنترل می‌توان یک فایل یا یک پوشش را از بین لیستی از فایل‌ها و پوشش‌ها انتخاب کرد. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JFileChooser()
public JFileChooser(File currentDirectory)
public JFileChooser(File currentDirectory, FileSystemView fileSystemView)
public JFileChooser(FileSystemView fileSystemView)
public JFileChooser(String currentDirectoryPath)
public JFileChooser(String currentDirectoryPath, FileSystemView fileSystemView)
```

کادر انتخاب یا ذخیره فایل، به وسیله سه متده نمایش داده می‌شود :

- showDialog()
- showOpenDialog()
- showSaveDialog()

در جدول زیر هم لیست متدهایی که برای سفارشی سازی کادر محاوره‌ای JFileChooser به کار می‌روند به همراه کاربرد آن‌ها ذکر شده است :

متده	کاربرد
setMultiSelectionEnabled	به کاربر اجازه انتخاب همزمان چند فایل یا پوشش را می‌دهد و دو مقدار true و false می‌گیرد.
setCurrentDirectory	به وسیله این متده می‌توان پوشش یا درایو بیشفرضی که هنگام باز شدن کادر محاوره‌ای نمایش داده می‌شود را مشخص کرد. اگر مقدار null به آن بدهیم بسته به نوع سیستم عامل یک مسیر بیشفرض تعیین می‌شود که مثلاً در ویندوز My Document است.

<p>این متد سه مقدار می‌گیرد که در حالتی که کاربر اجازه انتخاب چند فایل یا پوشه را دارد کاربر را محدود می‌کند :</p> <ul style="list-style-type: none"> • FILES_AND_DIRECTORIES : کاربر اجازه انتخاب همزمان فایل و پوشه را دارد. • FILES_ONLY : کاربر اجازه انتخاب فقط فایل را دارد. • DIRECTORIES_ONLY : کاربر اجازه انتخاب فقط پوشه را دارد. 	setFileSelectionMode
<p>به وسیله این متد می‌توان پسوند فایل‌های قابل نمایش در پنجره را مشخص کرد.</p>	setFileFilter
<p>مشخص می‌کند که گزینه AllFiles در بین لیست پسوندها نمایش داده شود یا نه؟</p>	setAcceptAllFileFilterUsed

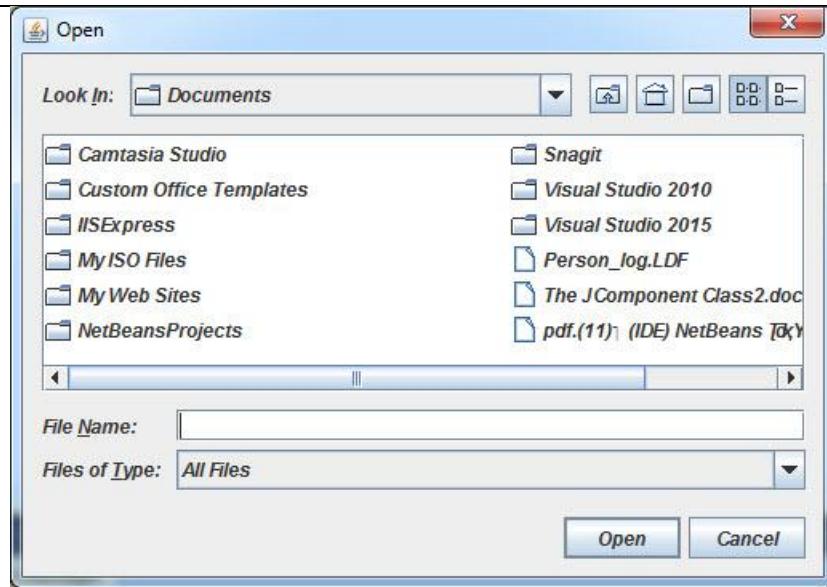
از متد () showDialog به همراه یک شیء از کلاس JFileChooser می‌توان یک کادر محاوره‌ای سفارشی برای باز یا ذخیره کردن یک فایل ایجاد کرد. این متد دو آرگومان می‌گیرد که اولین آرگومان کنترل والد و دومین آرگومان عنوان دکمه و کادر را مشخص می‌کند. برای روشن شدن نحوه استفاده از این متد، کد زیر را در یک فایل با پسوند جاوا نوشته و اجرا کنید :

```
package myfirstprogram;

import javax.swing.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        JFileChooser filechooser = new JFileChooser();
        filechooser.showDialog(null, "Open");
    }
}
```

نتیجه اجرای کد بالا به صورت زیر است :

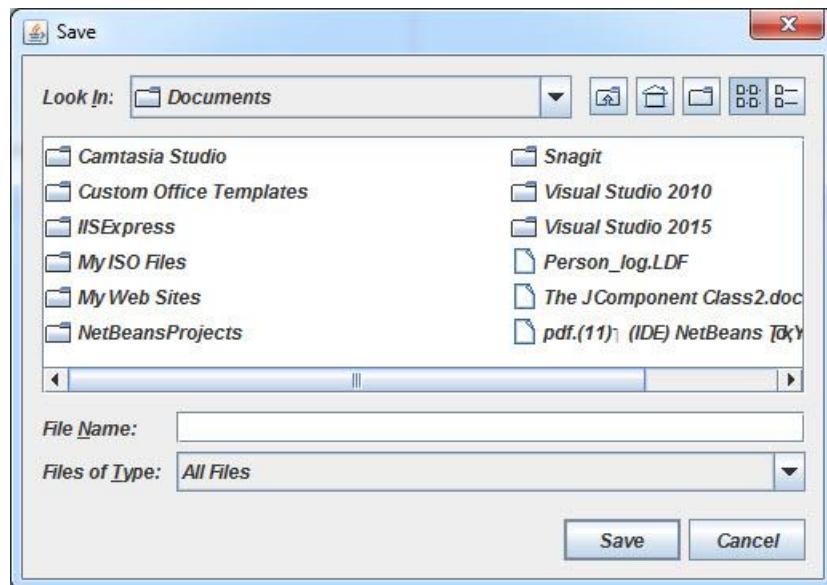


در شکل بالا آرگومان دوم متده `showDialog()` هم به جای نام دکمه و هم به جای عنوان پنجره به کار رفته است. حال آرگومان‌های متده

`showDialog()` کد بالا را به صورت زیر تغییر دهید :

```
filechooser.showDialog(null, "Save");
```

برنامه را اجرا و نتیجه را مشاهده کنید :



همانطور که احتمالاً متوجه شده‌اید می‌توان از این متند هم به جای کادر محاوره‌ای باز و هم به جای کادر محاوره‌ای ذخیره فایل استفاده

کرد. البته کلاس `JFileChooser` دارای متندی به نام `setDialogType` می‌باشد که با دریافت سه مقدار ثابت زیر نوع کادر محاوره‌ای را

مشخص می‌کند :

`JFileChooser.OPEN_DIALOG` •

`JFileChooser.SAVE_DIALOG` •

`JFileChooser.CUSTOM_DIALOG` •

در کد بالا ما لازم بود که یک عنوان به پنجره و دکمه اختصاص دهیم ولی با استفاده از این متند و ثابت‌ها به طور خودکار نوع کادر محاوره‌ای

تشخیص داده شده و مقادیری برای دکمه و عنوان اختصاص داده می‌شود. به کد زیر توجه کنید :

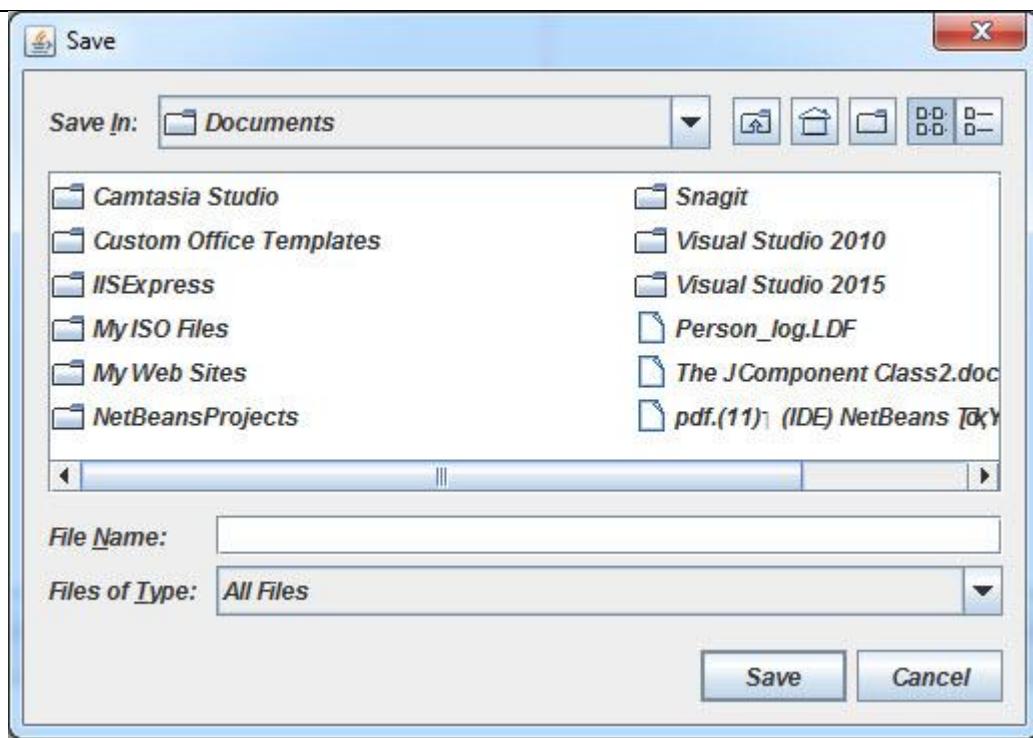
```
package myfirstprogram;

import javax.swing.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        JFileChooser filechooser = new JFileChooser();
        filechooser.setDialogType(JFileChooser.SAVE_DIALOG);
        filechooser.showDialog(null, null);
    }
}
```

در کد بالا مقدار عنوان پنجره و دکمه را `null` قرار داده‌ایم ولی چون در خط قبل از آن نوع پنجره را `JFileChooser.SAVE_DIALOG` را

گذاشته‌ایم در نتیجه عنوان دکمه و پنجره به `Save` تغییر می‌کند :



تشخیص اینکه چه دکمه‌ای توسط کاربر فشرده شده است بسیار مهم است. متد `() showDialog` وقتی که کاربر بر روی دکمه‌های `Open` یا `Save` کلیک می‌کند مقدار `0` و وقتی بر روی `Cancel` کلیک می‌کند، مقدار `1` و اگر خطایی در هنگام کار با کادر محاوره‌ای رخ دهد مقدار `-1` را بر می‌گرداند. با مقایسه این سه مقدار برگشتی با سه ثابت از کلاس `JFileChooser` می‌توان کدهای کارامد تری نوشت :

- `JFileChooser.APPROVE_OPTION`
- `CANCEL_OPTION`
- `JFileChooser.ERROR_OPTION`

به برنامه زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import javax.swing.*;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {
9         final JFileChooser filechooser = new JFileChooser();
10        filechooser.setDialogType(JFileChooser.SAVE_DIALOG);
11    }
12 }
```

```

12     int selectedNumber = filechooser.showDialog(null, null);
13
14     switch (selectedNumber)
15     {
16         case JFileChooser.APPROVE_OPTION :
17             JOptionPane.showMessageDialog(null, "You Clicked on Open or Save!");
18             break;
19         case JFileChooser.CANCEL_OPTION :
20             JOptionPane.showMessageDialog(null, "You Clicked on Cancel!");
21             break;
22         case JFileChooser.ERROR_OPTION :
23             JOptionPane.showMessageDialog(null, "An error occurred !");
24             break;
25     }
26 }
27 }
28 }
```

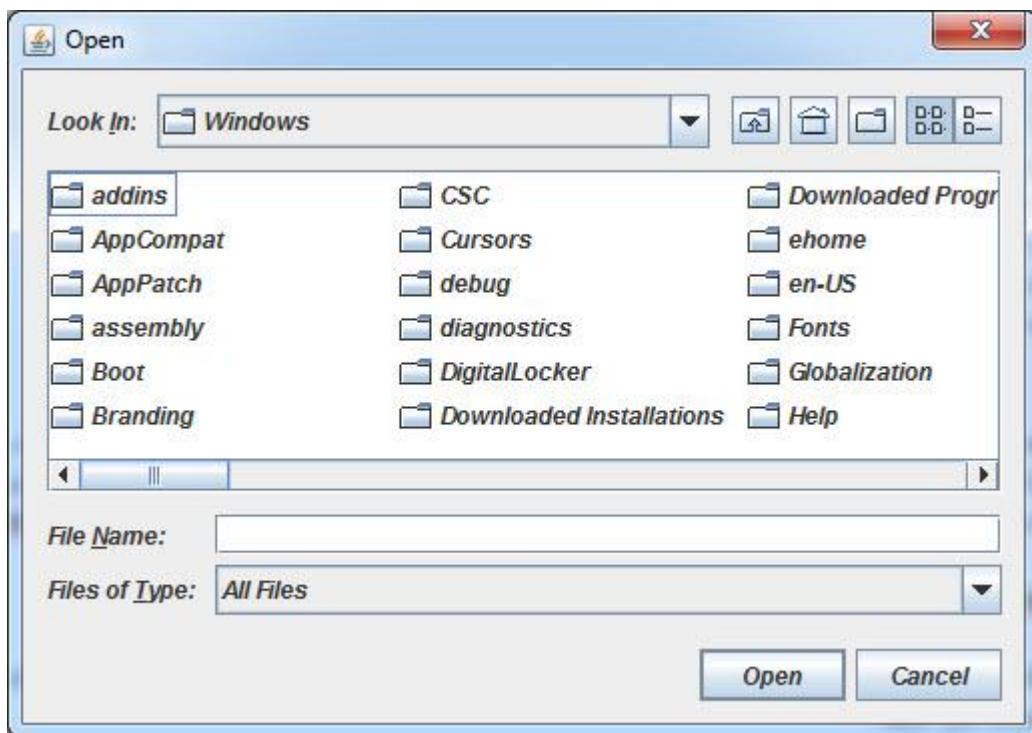
در خط ۱۳ کد بالا مقدار برگشتی از متند را در یک متغیر از نوع صحیح (selectedNumber) قرار داده‌ایم. سپس همین مقدار را در ۱۷ و ۲۰ و ۲۳ با مقادیر ثابت گفته شده مقایسه کرده و در صورت مساوی بودن با هر یک از این مقادیر پیغام‌های مختلفی نمایش داده‌ایم. حال اجازه دهید که با یک برنامه به شما نحوه استفاده از این کادرها و متدهای آن‌ها را به شما آموزش دهیم. به کد زیر توجه کنید:

```

1 package myfirstprogram;
2
3 import java.io.File;
4 import javax.swing.*;
5 import java.awt.event.*;
6 import javax.swing.filechooser.*;
7
8 public class MyFirstProgram
9 {
10     public static void main(String[] args)
11     {
12         final JFrame frame = new JFrame("JFileChooser Demo");
13         JPanel panel = new JPanel();
14         JButton button = new JButton("Open File");
15         panel.add(button);
16         frame.add(panel);
17
18         final JFileChooser filechooser = new JFileChooser();
19
20         button.addActionListener(new ActionListener()
21         {
22             public void actionPerformed(ActionEvent e)
23             {
24                 filechooser.showDialog(null,null);
25             }
26         });
27
28         frame.setSize(430,315);
29         frame.setVisible(true);
30     }
31 }
```

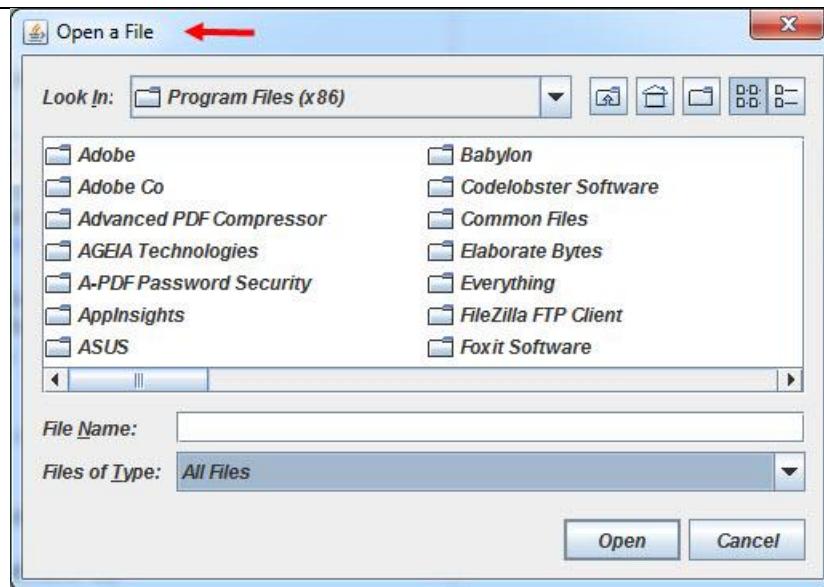
در کد بالا یک دکمه به Panel و Panel را به اضافه کرده‌ایم (خطوط ۱۶-۲۰). حال می‌خواهیم با کلیک بر روی دکمه، کادر محاوره‌ای باز کردن فایل (OpenFileDialog) نمایش داده شود. این کار را در خطوط ۲۶-۳۰ انجام داده‌ایم. با اجرای برنامه و زدن بر روی دکمه مشاهده می‌کنید که کادر محاوره‌ای نمایش داده می‌شود. از این به بعد کدهایی را که توضیح می‌دهیم بین خطوط ۱۸ و ۲۰ بنویسید. ابتدا نوع کادر محاوره‌ای را با استفاده از متده استفاده می‌کنیم :

```
filechooser.setDialogType(JFileChooser.OPEN_DIALOG);
```



مشاهده می‌کنید که عنوان دکمه و پنجره هر دو Open می‌باشد. اگر بخواهید عنوان پنجره را تغییر دهید از متده استفاده کنید :

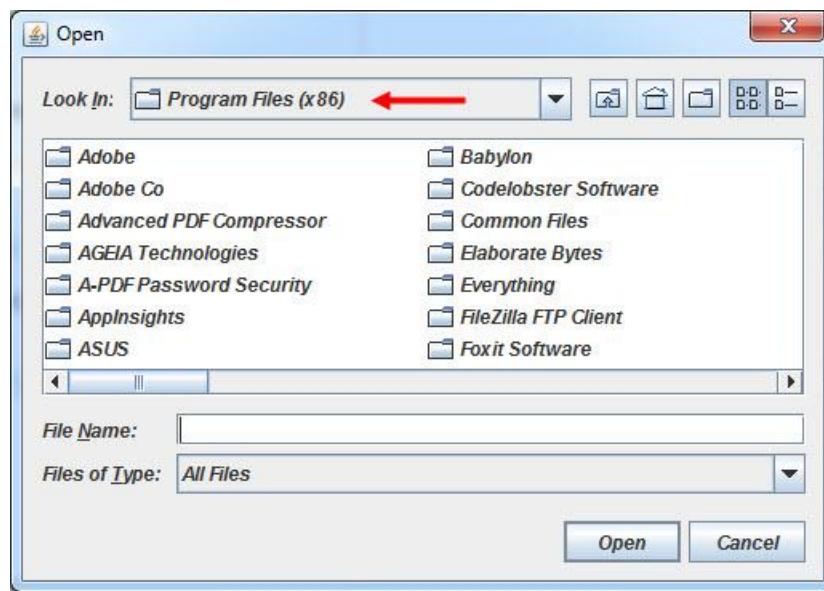
```
filechooser.setDialogTitle("Open a File");
```



فرض کنید که می‌خواهیم با باز شدن کادر محاوره‌ای مثلًا پوشه Program files درایو C به عنوان پوشه پیشفرض نمایش داده شود.

برای این کار () به صورت زیر استفاده می‌کنیم :

```
filechooser.setCurrentDirectory(new File("C:\\\\Program Files (x86)"));
```



اگر بخواهیم کاربر را مجبور به انتخاب یک یا چند فایل بکنیم یعنی کاربر نتواند با گرفتن دکمه Ctrl کیبورد هم پوشه انتخاب کند و هم

فایل از کد زیر استفاده می‌کنیم :

```
filechooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
filechooser.setMultiSelectionEnabled(true);
```

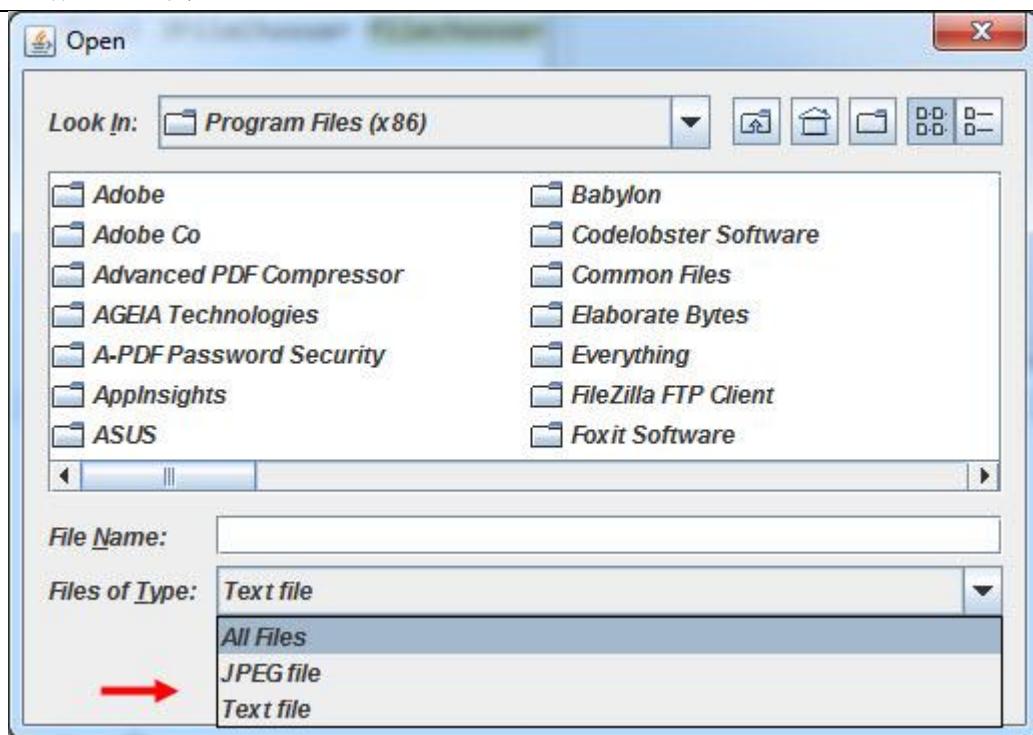
در خط اول مقدار `JFileChooser.FILES_ONLY` را به متدهای `setFileSelectionMode()` ارسال کرده‌ایم، در نتیجه فقط فایل‌ها قابل انتخاب هستند و برای حق انتخاب چند فایل هم مقدار `true` را به متدهای `setMultiSelectionEnabled()` ارسال می‌کنیم. با اجرای برنامه و گرفتن دکمه `Ctrl` متوجه می‌شوید که نمی‌شود همزمان هم فایل انتخاب کرد و هم پوشه. در حالت پیشفرض تمامی پسوندها در کادر محاوره‌ای نمایش داده می‌شود (`All files`). ولی اگر بخواهیم گزینه‌های دیگری به کادر محاوره‌ای اضافه کنیم که مثلًاً به وسیله آن‌ها فقط فایل‌های متنی و یا فقط عکس‌ها نمایش داده شوند از کلاس‌های `FileNameExtensionFilter` و `FileFilter` به صورت زیر برای تعیین پسوندهای دلخواه و اضافه کردن آن‌ها به کادر محاوره‌ای استفاده می‌کنید :

```
FileFilter ImageFiles = new FileNameExtensionFilter("JPEG file", "jpg", "jpeg");
FileFilter TextFiles = new FileNameExtensionFilter("Text file", "txt");
```

در کدهای بالا، اولین پارامتر داخل پرانتز توضیحی درباره پسوندها و پارامترهای بعدی هم خود اسم خود پسوندها می‌باشد. تا اینجا ما فقط پسوندها را مشخص کرده‌ایم. برای اضافه کردن آن‌ها به کادر محاوره‌ای از متدهای `setFileFilter()` به صورت زیر استفاده می‌کنیم :

```
filechooser.setFileFilter(ImageFiles);
filechooser.setFileFilter(TextFiles);
```

حال برنامه را اجرا کنید :



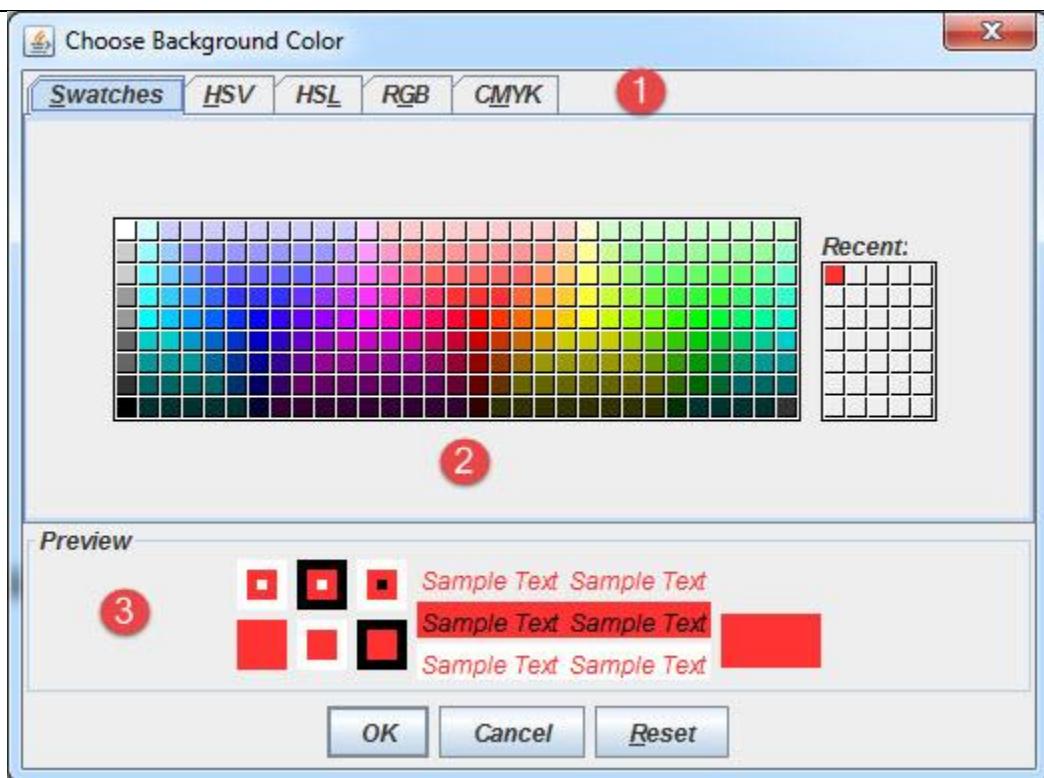
مشاهده می‌کنید که در جلوی کادر Files of Type دو گروه پسوند دیگر اضافه شده است که با انتخاب هر یک از آن‌ها فقط فایل‌های با آن پسوند در کادر محاوره‌ای نمایش داده می‌شود.

JColorChooser کنترل

از کنترل JColorChooser وقتی استفاده می‌شود که شما بخواهید یک رنگ انتخاب کنید. به عنوان مثال وقتی بخواهید رنگ یک فونت یا رنگ پس زمینه فرم را تغییر دهید می‌توانید از این کنترل استفاده کنید. سازنده این کلاس به روش‌های زیر مقداردهی می‌شود :

```
public JColorChooser()
public JColorChooser(Color initialColor)
public JColorChooser(ColorSelectionModel model)
```

نمای کلی کنترل JColorChooser در شکل زیر مشاهده می‌کنید :



پنجره `JColorChooser` از رنگ‌های از پیش تعریف شده تشکیل شده است (شماره ۲). شما می‌توانید با کلیک بر روی سربرگ‌های این پنجره (شماره ۱) تعداد بیشتری رنگ مشاهده کنید به طوری که در پنجره ظاهر شده هر رنگی که دوست دارید را انتخاب نمایید. همچنین با استفاده از پنجره Preview (شماره ۳) می‌توانید پیش نمایشی از اعمال رنگ مورد نظرتان بر قسمت‌های مختلف یک کنترل (متن، پس زمینه متن، خاشیه کنترل و ...) را مشاهده نمایید. برای نشان دادن عملکرد کادر محاوره‌ای `JColorChooser` به کد زیر توجه کنید :

```

1 package myfirstprogram;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.event.*;
6
7 public class MyFirstProgram
8 {
9     public static void main(String[] args)
10    {
11        JFrame frame1 = new JFrame("JColorChooser Demo");
12        final JPanel panel1 = new JPanel();
13        JButton button1 = new JButton("Change Panel Color");
14
15        panel1.add(button1);
16        frame1.add(panel1);
17
18        button1.addActionListener(new ActionListener()

```

```

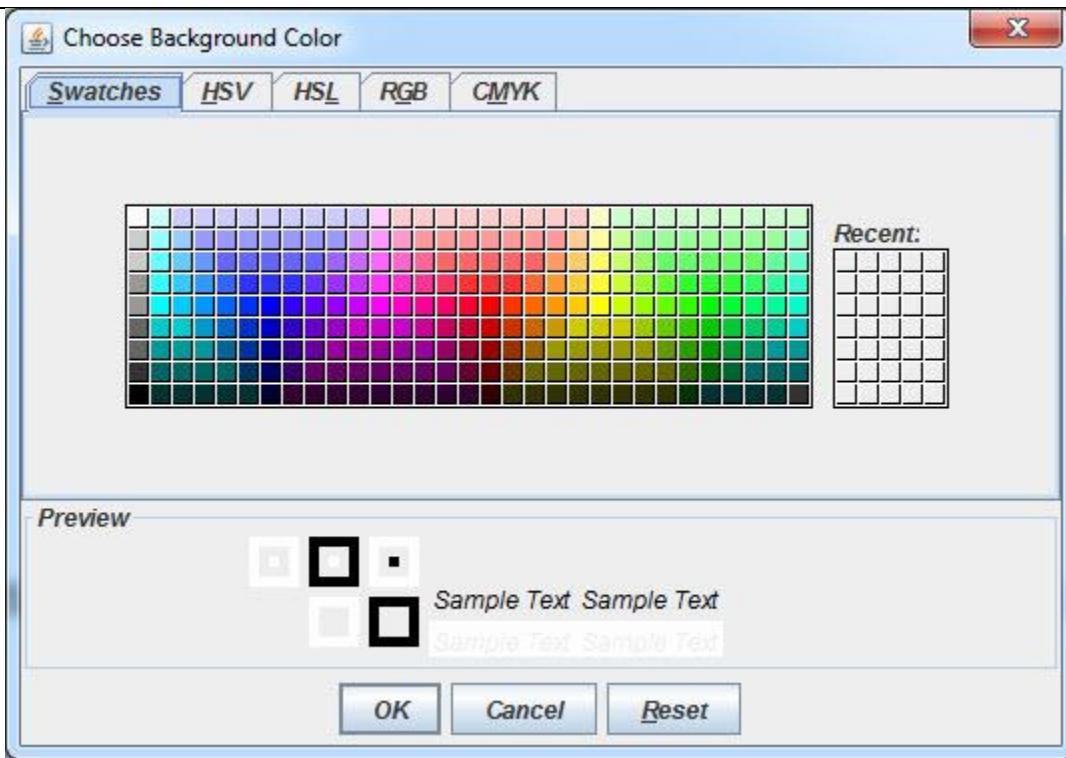
19     {
20         public void actionPerformed(ActionEvent e)
21         {
22             Color selectedColor = JColorChooser.showDialog(
23                 panel1, "Choose Background Color", panel1.getBackground());
24
25             if(selectedColor != null)
26             {
27                 panel1.setBackground(selectedColor);
28             }
29         }
30     );
31
32     frame1.setSize(416, 313);
33     frame1.setVisible(true);
34 }
35 }
```

هدف برنامه بالا اعمال رنگ به پس زمینه یه Panel میباشد. در خطوط ۱۱-۱۶ یک کنترل دکمه به Panel و Panel را به یک اضافه کرده‌ایم. چون که قرار است با کلیک بر روی دکمه کادر محاوره‌ای JColorChooser نمایش داده شود، پس باید رابط ActionListener را پیاده سازی کنیم. در خطوط ۱۸-۲۹ این کار را انجام داده و سپس در بدنه متده actionPerformed را بخطوط ۲۲-۲۸ را مینویسید. در خطوط ۲۲-۲۳ ابتدا یک رنگ را انتخاب میکنیم. در این خط ابتدا یک شی از کلاس Color ایجاد میکنیم. این کلاس یک رنگ را در اختیار ما میگذارد. در ادامه همین خط چون ما قرار است که با استفاده از کادر محاوره‌ای JColorChooser یک رنگ انتخاب کنیم باید ابتدا کادر را نمایش دهیم. این کار را با استفاده از متده استاتیک showDialog() انجام می‌دهیم. این متده سه پارامتر می‌گیرد :

```
JColorChooser.showDialog(parent,title,color);
```

پارامتر اول کنترل والد، پارامتر دوم عنوان کادر محاوره‌ای و پارامتر سوم رنگ پیشفرضی که هنگام نمایش کادر مشخص می‌شود، می‌باشد.

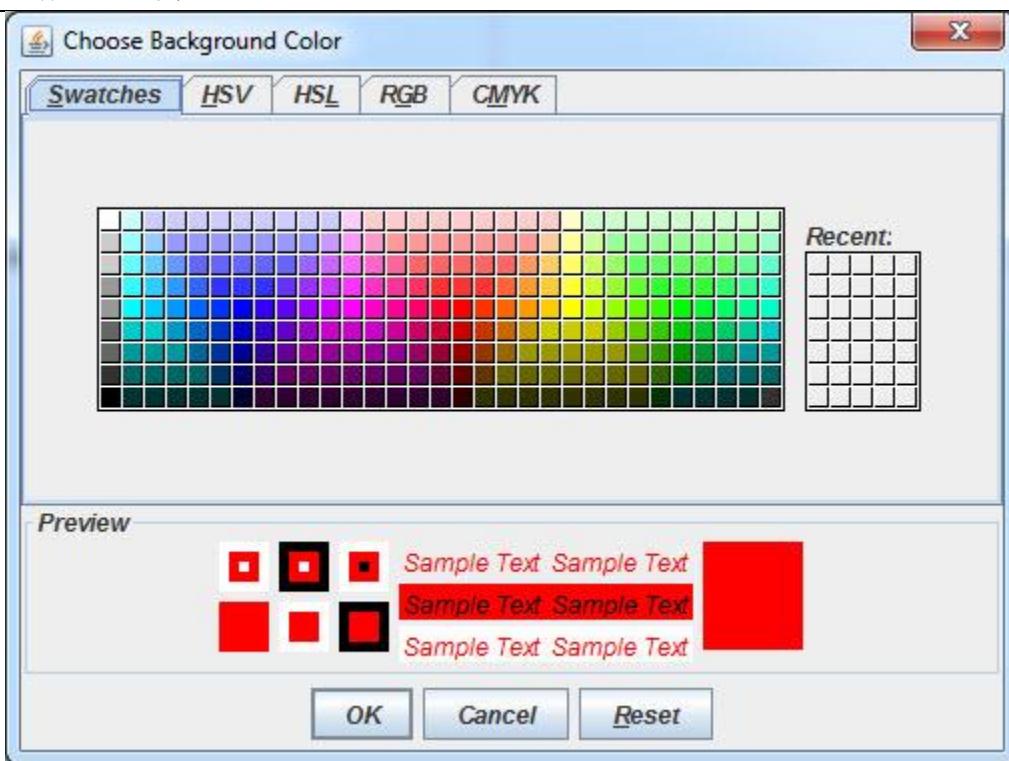
برنامه را اجرا کنید :



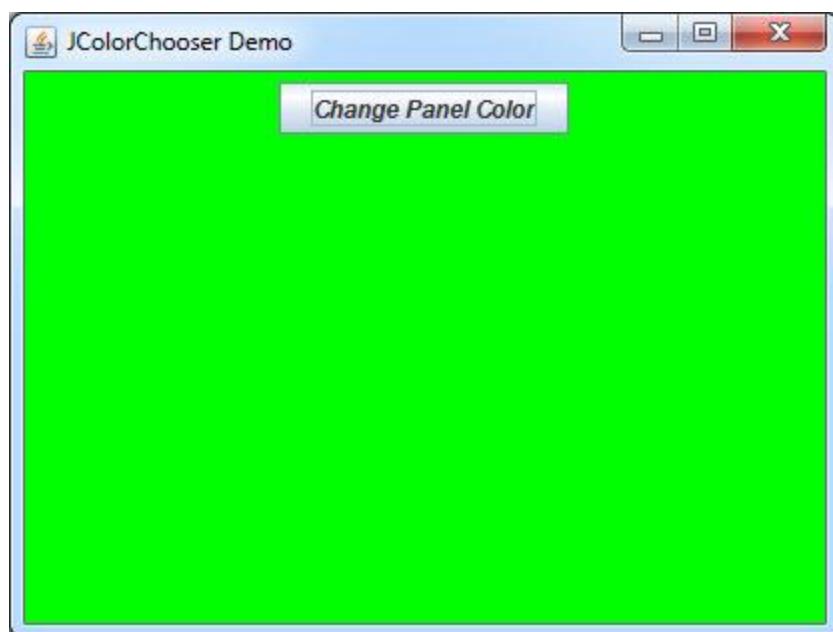
مشاهده می‌کنید که عنوان پنجره Choose Background Color و در قسمت پیش‌نمایش هم رنگ پیش‌فرض سفید می‌باشد. چون ما هیچ رنگی را از قبل به پس زمینه panel1 اختصاص نداده‌ایم در نتیجه متده است (getBackground) که رنگ جاری کنترل را بر می‌گرداند، رنگ سفید را بر می‌گرداند. استفاده از این متده یک حسن دارد و آن این است که اگر یک رنگ انتخاب کرده و بر روی دکمه OK کلیک کنید و دوباره کادر را باز کنید، همان رنگ قبلی را که انتخاب کرده‌اید به عنوان رنگ پیش‌فرض کادر به شما نمایش می‌دهد. حال همین خط ۲۲ را به صورت زیر بنویسید :

```
Color selectedColor = JColorChooser.showDialog(panel1, "Choose Background Color", Color.RED);
```

مشاهده می‌کنید که به خاطر پارامتر سوم بعد از اجرای برنامه و زدن دکمه، رنگ پیش‌فرض کادر محاوره‌ای قرمز می‌باشد :



حال اگر چندین بار رنگ انتخاب کرده و در هر بار دکمه OK را بزنید و دوباره کادر را باز کنید همیشه رنگ قرمز به شما نمایش داده می‌شود.
در خطوط ۲۷-۳۴ چک می‌کنیم که اگر رنگی انتخاب شده باشد آن را به پس زمینه Panel اعمال کند. برنامه را اجرا، یک رنگ را انتخاب و
بر روی دکمه OK کلیک کنید کنید :



کار با تاریخ، فایل و رشته

کلاس Date

کلاس Date کلاسی از جاوا است که در پکیج `java.util` قرار دارد و به شما اجازه استفاده، ذخیره، و دستکاری ساعت و تاریخ را می‌دهد. این کلاس دارای متدهایی برای دستکاری تاریخ مانند اضافه و کم کردن روزها، ماهها، یا سالها و مطابق کردن آن‌ها با تاریخ جاری را می‌دهند. همچنین دارای متدهایی است که تاریخ را به اشکال متفاوتی نشان می‌دهند. کد زیر نحوه استفاده از کلاس Date را نشان می‌دهد.

```
package myfirstprogram;

import java.util.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Date today = new Date();
        System.out.print(today);
    }
}
```

Wed Jul 08 17:52:38 IRDT 2015

در این مثال تاریخ و ساعتی که بنده این آموزش را قرار داده‌ام نمایش داده شده است. همانطور که در کد بالا مشاهده می‌کنید برای استفاده از این کلاس باید پکیج مربوطه (`java.util`) را با کلمه کلیدی `import` وارد برنامه کنید. کلاس Date دارای چندین متده است که در زیر کاربرد آن‌ها نشان داده شده است :

متده	توضیحات
<code>after()</code>	مقدار <code>true</code> را در صورتی که تاریخ سیستم بعد از تاریخ داده شده بصورت پارامتر باشد برمی‌گرداند.
<code>before()</code>	مقدار <code>true</code> را در صورتی که تاریخ سیستم قبل از تاریخ داده شده بصورت پارامتر باشد برمی‌گرداند.
<code>clone()</code>	برای کپی کردنشی تعریف شده.
<code>compareTo()</code>	مقایسه تاریخ سیستم با تاریخ پارامتر. در صورتی که برابر باشند مقدار <code>0</code> در صورتی که تاریخ سیستم قبل باشد مقدار منفی و در صورتی که بعد باشد مقدار مثبت بر می‌گرداند.

چک می‌کند که آیا کلاس داده شده در پارامتر از نوع تاریخ هست یا نه .	<code>compareTo()</code>
مقایسه تاریخ موجود با تاریخ و زمان پارامتر و بازگشت مقدار <code>true</code> در صورت برابر بودن .	<code>equals()</code>
بازگرداندن تعداد میلی ثانیه از تاریخ ۱ ژانویه ۱۹۷۰ .	<code>getTime()</code>
بازگرداندن کد هش شده شیء تعریف شده .	<code>hashCode()</code>
تنظیم زمان با دادن تعداد میلی ثانیه از ۱ ژانویه ۱۹۷۰	<code>setTime()</code>

به مثال زیر درباره کاربرد متدهای بالا توجه کنید :

```
package myfirstprogram;

import java.util.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println(new Date(2010).after(new Date(2015)));
        System.out.println(new Date(2010).before(new Date(2015)));
        System.out.println(new Date().clone());
        System.out.println(new Date(2010).compareTo(new Date(2015)));
        System.out.println(new Date(2010).equals(new Date(2015)));
        System.out.println(new Date().getTime());
        System.out.println(new Date().hashCode());
    }
}

false
true
Wed Jul 08 18:44:03 IRDT 2015
-1
false
1436364843809
1845766767
```

قالب بندی تاریخ

برای قالب بندی یا نمایش دلخواه تاریخ در جاوا هم می‌توان از متد `printf()` و هم از کلاس `SimpleDateFormat` استفاده کرد. قبل از خواندن ادامه آموزش توصیه می‌شود که ابتدا این پست را بخوانید.

<http://www.w3-farsi.com/?p=8321>

قالب بندی تاریخ با متده printf

هنگامی که یک شی Date ایجاد می‌کنیم برای نمایش تاریخ و زمان به صورت سفارشی می‌توان از متده printf و کارکترهای خاصی که در جدول زیر لیست شده‌اند استفاده کرد:

مثال	توضیح	کارکتر
Mon May 04 09:51:52 CDT 2015	نمایش تاریخ و زمان بصورت کامل	c
2004-02-09	نمایش تاریخ با فرمت iso 8601	F
10/09/2015	نمایش تاریخ با فرمت (month/day/year)	D
19:25:19	نمایش زمان با فرمت ۲۴ ساعته	T
10:05:20 pm	نمایش زمان با فرمت ۱۲ ساعته	r
20:05	نمایش زمان با فرمت ۲۴ ساعته بدون ثانیه	R
2015	نمایش سال با ۴ رقم	Y
06	نمایش سال با ۲ رقم آخر	y
20	نمایش ۲ رقم اول سال	C
February	نمایش نام کامل ماه	B
Feb	نمایش خلاصه شده نام ماه	b
04	نمایش ماه با ۲ رقم	m
07	نمایش روز با ۲ رقم	d
7	نمایش روز بودن عدد ۰ در صورت یک رقمی بودن	e

کار با تاریخ، فایل و رشته

۴۴۱

		نمایش نام کامل روز در هفته	A
	Mon	نمایش نام اختصاری روز در هفته	a
	071	نمایش شماره ۳ رقمی روز در سال	j
	19	نمایش شماره ۲ رقمی نشان دهنده ساعت در فرمت ۲۴ ساعته ۰۰ تا ۲۴	H
	17	نمایش ساعت بدون گذاشتن ° قبل از ساعت یک رقمی ° تا ۲۳	k
	07	شماره ۲ رقمی نشان دهنده ساعت در فرمت ۱۲ ساعته ۰۰ تا ۱۲	I
	7	نمایش ساعت بودن گذاشتن ° قبل از ساعت یک رقمی ° تا ۱۲	l
	07	نمایش دقیقه ۲ رقمی	M
	20	نمایش دقیقه بدون گذاشتن ° قبل از آن در صورت ۱ رقمی بودن	S
	051	نمایش ۳ رقمی میلی ثانیه	L
	051000000	نمایش ۹ رقمی نانو ثانیه	N
	PM	نمایش نوع زمان بصورت حروف بزرگ	P
	pm	نمایش نوع زمان با حروف کوچک	p
	-0800	نمایش عددی انحراف از زمان واحد GMT	z
	PST	منطقه زمانی	Z
	1084524320	تعداد ثانیه‌ها از تاریخ ۱/۱/۱۹۷۰	s
	1052334319051	تعداد میلی ثانیه از زمان ۰۱/۰۱/۱۹۷۰ GMT سال ۰۰:۰۰:۰۰	Q

برای اینکه با کاربرد کاراکترهای بالا در متند و همچنین قالب بندی آشنا شوید به مثال زیر توجه کنید :

```
Date today = new Date();
System.out.printf("%tY", today);
2015
```

در مثال بالا از `%tY` برای قالب بندی تاریخ استفاده کرده‌ایم. قالب بندی همیشه با علامت % شروع می‌شود. حرف t بعد از آن به معنای آن است که ما می‌خواهیم یک مقدار زمانی را قالب بندی کنیم و حرف Y هم، همانطور که در جدول مشاهده می‌کنید برای نمایش سال به صورت چهار رقمی است. حال فرض کنید که می‌خواهیم یک زمان را به صورت کامل نمایش دهیم :

```
Date today = new Date();
System.out.printf("%1$s %2$tB %2$td, %2$tY", "Due Today : ", today);
Due Today : July 08, 2015
```

همانطور که در کد بالا مشاهده می‌کنید، Due Today یک رشته است و اولین آرگومان بعد از رشته قالب بندی شده، در نتیجه از `%1$s` برای قالب بندی آن استفاده کرده‌ایم. عدد یک به معنای اولین آرگومان، علامت \$ محدود کننده است و برای قالب بندی‌های چند بخشی کاربرد دارد یا به عبارت دیگر اندیس آرگومان‌ها باید در داخل دو علامت \$ قرار بگیرد. علامت t کوچک هم به معنای رشته می‌باشد. دومین آرگومان بعد از رشته قالب بندی شده هم به صورت سه بخشی قرار است نمایش داده شود پس هر بخش با `%t` شروع و بعد از این عبارت هم یکی از کarakترهای موجود در جدول بالا آمده است. t به معنای تاریخ است.

قالب بندی تاریخ با کلاس SimpleDateFormat

کلاس `SimpleDateFormat` هم که در پکیج `java.text` قرار دارد با دریافت کarakترهای خاصی تاریخ را فرمات بندی می‌کند. لیست کarakترهایی که می‌توانیم به سازنده این کلاس بدھیم در جدول زیر آمده است :

کاراکتر	توضیحات	مثال
G	نوع فرمات تاریخ	AD
y	چهار رقم سال	2015
M	نام یا شماره ماه در سال	July or 07

کار با تاریخ، فایل و رشته

۴۴۲

12	روز در ماه	d
10	ساعت در فرمت ۱۲ ساعته	h
22	ساعت در فرمت ۲۴ ساعته	H
30	دقیقه از ساعت	m
55	ثانیه از دقیقه	s
234	میلی ثانیه	S
Tuesday	نام روز هفته	E
360	روز در سال	D
2 (second Wed. in July)	روز از هفته در ماه	F
40	هفته از سال	w
1	هفته از ماه	W
PM	نوشتمن قبل یا بعد ظهر	a
24	ساعت از روز ۲۴ ساعته	k
10	ساعت از روز ۱۲ ساعته	K
Eastern Standard Time	منطقه زمانی	z
Delimiter	نمایش متن	,
'	نمایش تک کوپیشن	"

به مثال زیر توجه کنید :

```
Date today = new Date();
SimpleDateFormat SDF = new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
System.out.println("Current Date: " + SDF.format(today));
```

Current Date: Wed 2015.07.08 at 10:39:19 PM IRDT

کلاس Math

از کلاس Math برای انجام محاسبات ریاضی استفاده می‌شود. از این کلاس می‌توان برای گرد کردن اعداد، گرفتن جذر یا نتیجه توان یک عدد استفاده کرد. این کلاس یک کلاس final است و دارای تعدادی متدهای static بوده که از آن‌ها برای انجام اعمال مختلف ریاضی از جمله به دست آوردن توان، جذر گرفتن و ... استفاده می‌شود. برای روشن شدن مطلب برخی از متدهای این کلاس را توضیح می‌دهیم.

متدها	توضیح
abs	قدرت مطلق یک عدد را برابر می‌گرداند.
ceil	کوچک‌ترین مقدار صحیحی که بزرگ‌تر یا مساوی با عدد مورد نظر ما باشد را برابر می‌گرداند.
cos	کوسینوس یک زاویه مشخص را برابر می‌گرداند.
floor	بزرگ‌ترین مقدار صحیحی که کوچک‌تر یا مساوی با عدد مورد نظر ما باشد را برابر می‌گرداند.
log10	لگاریتم یک عدد در مبنای ۱۰ را برابر می‌گرداند.
max	بزرگ‌ترین عدد در بین چندین عدد را برابر می‌گرداند.
min	کوچک‌ترین عدد در بین چندین عدد را برابر می‌گرداند.
pow	برای به توان رساندن یک عدد به کار می‌رود.
round	گرد کردن یک عدد اعشار به نزدیک‌ترین مقدار صحیح.
sin	سینوس یک زاویه مشخص را برابر می‌گرداند.
sqrt	جذر یک عدد را برابر می‌گرداند.

تائزنات یک زاویه را بر می‌گرداند.

tan

گرد کردن اعداد با استفاده از کلاس Math

می‌توان با استفاده از Math.floor() و Math.ceil() یک عدد با قسمت اعشار را گرد کرد. متدهای Math.ceil() و Math.floor() یک عدد از نوع double را گرفته و یک مقدار از نوع double گرد شده را بر می‌گرداند. نتیجه این متدهای بزرگتر یا مساوی آرگومان دریافت شده است. برای گرفته و نتیجه کوچکتر یا مساوی آرگومان گرفته شده است. برای روشن شدن مطلب به مثال زیر توجه کنید :

```
double number = 34.567;
double ceil = Math.ceil(number);
double floor = Math.floor(number);

System.out.println(MessageFormat.format("Math.Ceiling({0}) = {1}", number, ceil));
System.out.println(MessageFormat.format("Math.Floor({0}) = {1}", number, floor));

Math.Ceiling(34.567) = 35
Math.Floor(34.567) = 34
```

اگر بخواهید یک عدد اعشاری را گرد کنید می‌توانید از متدهای Math.round() استفاده کنید :

```
double firstNumber = 3.4;
double secondNumber = 3.6;

System.out.println(Math.round(firstNumber));
System.out.println(Math.round(secondNumber));

3
4
```

همانطور که در کدبالا مشاهده می‌کنید اگر اولین رقم اعشار از ۵ کوچکتر باشد عدد اعشار رو به پایین و اگر بزرگتر از ۵ باشد عدد اعشار رو به بالا گرد می‌شود.

به توان رساندن یک عدد با استفاده از کلاس Math

برای به توان رساندن یک عدد از متدهای Math.pow() استفاده می‌شود. این متدهای دو آرگومان از نوع double قبول کرده که اولین آرگومان پایه و دومی توان می‌باشد. مقدار برگشتی از این متدهای double است. به کدبالا توجه کنید :

```
for (int i = 0; i < 10; i++)
{
```

```
        System.out.println(MessageFormat.format("2^{0} = {1}", i, Math.pow(2, i)));
    }
```

```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
```

یافتن ریشه یک عدد

برای محاسبه ریشه یک عدد از متدهای Math.sqrt() استفاده می‌شود. این متدهای مقدار برگشتی از این متدهم double می‌باشد.

```
System.out.println(Math.sqrt(25));
```

```
5.0
```

یافتن بزرگترین و کوچکترین عدد با استفاده از کلاس Math

کلاس Math دارای متدهای Math.max() و Math.min() برای یافتن بزرگترین و کوچکترین عدد از بین چندین عدد می‌باشد. هر دو متدهای آرگومان از نوع عددی قبول می‌کنند. در حالت پیشفرض دو عدد را می‌توانید با هم مقایسه نمایید.

```
System.out.println(Math.min(1,2));
System.out.println(Math.max(1,2));
```

```
1
2
```

برای جلوگیری از محدودیت این متدها می‌توان به صورت تو از آنها به صورت زیر استفاده کرد:

```
//Get the maximum and minimum of 3 numbers
int max = Math.max(Math.max(1, 2), 3);
int min = Math.min(Math.min(1, 2), 3);

System.out.println(MessageFormat.format("Max = {0}", max));
System.out.println(MessageFormat.format("Min = {0}", min));
```

```
Max = 3
Min = 1
```

همچنین می‌توانید یک تابع تعریف کنید که کوچکترین و بزرگترین هر تعداد عدد را به شما معرفی کند :

```

package myfirstprogram;

public class MyFirstProgram
{
    static int GetMax(int[] numbers)
    {
        int maximum = numbers[0];

        for (int i = 1; i < numbers.length; i++)
        {
            maximum = Math.max(maximum, numbers[i]);
        }

        return maximum;
    }

    static int GetMin(int[] numbers)
    {
        int minimum = numbers[0];

        for (int i = 1; i < numbers.length; i++)
        {
            minimum = Math.min(minimum, numbers[i]);
        }

        return minimum;
    }

    public static void main(String[] args)
    {
        int[] numbers = { 32, 17, 45, 10, 5 };
        int max = GetMax(numbers);
        int min = GetMin(numbers);
        System.out.println(max);
        System.out.println(min);
    }
}

```

45

5

توابع `GetMax()` و `GetMin()` آرایه‌ای از اعداد صحیح قبول می‌کنند (هر تعداد عدد را قبول می‌کنند). در داخل توابع ما فرض را بر این گذاشته‌ایم که اولین مقدار بزرگ‌ترین و آخرین مقدار کوچک‌ترین عدد است. سپس یک حلقه `for` ایجاد کرده که با اندیس ۱ شروع می‌شود. با استفاده از متدهای `Math.max()` و `Math.min()` تعیین می‌کنیم که آیا عنصر جاری حلقه از مقدار جاری اعداد بزرگ‌تر است یا کوچک‌تر. که در این صورت مقدار جاری را جایگزین متغیرهای بزرگ یا کوچک می‌کنیم. سپس مقادیر نتیجه را برگشت می‌دهیم.

Math.PI

از ثابت PI که مقدار $\pi = 3.14159265358979323846$ را در خود ذخیره دارد، زمانی که بخواهید محیط یا مساحت یک دایره را پیدا کنید،

استفاده می‌شود. برای یافتن محیط یک دایره به صورت زیر عمل می‌شود که در آن radius شعاع می‌باشد :

```
double area = Math.PI * Math.pow(radius, 2);
```

کلاس Math دارای متدهای بیشتری است که ما به توضیح همین چند متده را بسند کردیم.

ایجاد عدد تصادفی

برای تولید عدد تصادفی در جاوا از کلاس Random که در پکیج java.util قرار دارد استفاده می‌شود. پس در ابتدای برنامه لازم است

که این کلاس را import کنید :

```
import java.util.Random;
```

ساده‌ترین روش برای تولید اعداد تصادفی استفاده از متدهای nextDouble(), nextInt() و ... است. برای این کار ابتدا یک شیء از

کلاس Random می‌سازیم و سپس از متدهای مذکور برای تولید اعداد تصادفی استفاده می‌کنیم. به مثال زیر توجه کنید :

```
package myfirstprogram;

import java.util.Random;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Random generator = new Random();

        boolean booleanValue = generator.nextBoolean();
        int intValue = generator.nextInt();
        double doubleValue = generator.nextDouble();
        float floatValue = generator.nextFloat();

        System.out.println(booleanValue);
        System.out.println(intValue);
        System.out.println(doubleValue);
        System.out.println(floatValue);
    }
}
```

```
false
-240333896
0.07659793195249542
0.6878707
```

حال اگر برنامه بالا را چندین بار اجرا کنید نتیجه‌های مختلفی در خروجی مشاهده خواهید کرد. برای اینکه نحوه کار با این کلاس و متدهای

آن را راحت درک کنید در مثال‌های بعدی با اعداد صحیح کار می‌کنیم. فرض می‌کنیم که شما می‌خواهید ده عدد از نوع int تولید کنید :

```
Random generator = new Random();

//Generate 10 random numbers
for (int i = 1; i <= 10; i++)
{
    System.out.println(generator.nextInt());
}
```

```
306495489
861232443
1265448050
-30370747
499374488
-1865142177
1196801248
-1061328848
62018178
1609249893
```

خروجی شما مطمئناً متفاوت است چون همه اعداد تصادفی می‌باشند. یکی دیگر از نسخه‌های متدهای nextInt() به شما اجازه می‌دهد که تعدادی عدد تصادفی که کوچکتر از یک عدد خاص هستند، تولید کنید. فرض کنید که می‌خواهید یک عدد صحیح در یک بازه تولید کنید، مثلاً عددی بین ۰ و ۴ (یعنی ۰ و ۴ هم جزء اعداد تولید شده باشند). برای این کار باید به صورت زیر عمل کنید :

```
Random generator = new Random();

//Generate 10 random numbers
for (int i = 1; i <= 10; i++)
{
    System.out.println(generator.nextInt(5));
}
```

```
1
4
1
0
3
1
4
4
4
0
```

همانطور که در مثال بالا مشاهده می‌کنید عدد ۵ را به عنوان آرگومان به متدهای nextInt() داده‌ایم و سپس با استفاده از حلقه for ده بار خروجی این متدهای را چاپ کرده‌ایم. مشاهده می‌کنید که عدد ۵ هرگز چاپ نمی‌شود، چون مقداری که این کلاس و متدهای می‌کنند عددی

بزرگ‌تر مساوی با \circ و کوچک‌تر از 5 است نه خود عدد 5 . پس برای اینکه عدد 5 هم چاپ شود باید به دو روش عمل کنیم، یا به جای عدد

: عدد 6 را بنویسیم

```
int intValue = generator.nextInt(6);
```

یا به صورت زیر عمل کنیم :

```
Random generator = new Random();

//Generate 10 random numbers
for (int i = 1; i <= 10; i++)
{
    System.out.println(generator.nextInt(5) + 1);
```

```
3
1
1
4
2
3
4
2
5
```

کد بالا باعث تولید اعداد $1, 2, 3, 4$ و 5 می‌شود. یعنی مفهوم کد بالا با اضافه کردن عدد 1 این است که کوچک‌ترین عدد تولیدی 1

بزرگ‌ترین عدد 5 باشد. حال فرض کنید به جای عدد 1 از عدد 2 یا عدد دیگر استفاده کنیم. در مثال زیر از عدد 2 استفاده می‌کنیم :

```
Random generator = new Random();

//Generate 10 random numbers
for (int i = 1; i <= 10; i++)
{
    System.out.println(generator.nextInt(5) + 2);
```

در کد بالا کوچک‌ترین عدد تولیدی 2 و بزرگ‌ترین عدد تولیدی $(2+5)-1$ خواهد بود :

```
6
3
6
2
5
4
3
3
6
6
```

می‌توان از یک مقدار هم برای تولید یک توالی تکراری از اعداد تصادفی استفاده کرد. بدین معنی که با هر بار اجرای برنامه توالی و نوع اعداد تولید شده مانند سری قبلی باشد که برنامه اجرا شده است. این مقدار هر عددی می‌تواند باشد. برای اختصاص این مقدار باید آن را در داخل پرانتز سازنده کلاس Random قرار دهید. مانند عدد صفر در مثال زیر :

```
Random generator = new Random(0);

//Generate 10 random numbers
for (int i = 1; i <= 10; i++)
{
    System.out.println(generator.nextInt(5));
```

```
0
3
4
2
0
3
1
1
4
4
```

با هر بار اجرای برنامه بالا همین اعداد را مشاهده خواهید نمود. حال اجازه بدهید که یک مثال ساده را توضیح دهیم. این برنامه پیغام‌های متفاوتی در هر بار اجرای آن به شما نشان می‌دهد.

```
package myfirstprogram;

import java.util.Random;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Random generator = new Random();

        int messageNumber = generator.nextInt(3) + 1;

        switch (messageNumber)
        {
            case 1:
                System.out.println("Hello to you my friend.");
                break;
            case 2:
                System.out.println("Good day to you sir/mam.");
                break;
            case 3:
                System.out.println("Have a happy day.");
```

```

        break;
    }
}
}

```

برنامه مقادیر ۱ تا ۳ را تولید می‌کند. سپس با استفاده از دستور `switch` برای نشان دادن یک پیغام برای هر مقدار ممکن استفاده می‌کنیم.

بعد از چندین بار اجرای برنامه متوجه می‌شوید که پیغام خوش آمد گویی تغییر می‌کند.

رشته‌ها و عبارات با قاعده

رشته‌ها معمول‌ترین انواع داده‌ای هستند که در تقریباً همه زبانهای برنامه نویسی یافت می‌شوند. یک رشته گروهی از کاراکترها مانند حروف، اعداد یا نشانه‌ها می‌باشد. رشته به وسیله کلمه کلیدی `String` تعریف می‌شود. اهمیت این نوع داده‌ای زمانی مشخص می‌شود که شما بخواهید اطلاعاتی از قبیل نام، آدرس، جنسیت یا ایمیل خودتان را ذخیره کنید. با استفاده از رشته‌ها می‌توان فوراً به کاربر اطلاعاتی ارائه و یا در مورد یک خطای برنامه هشدار داد. از عبارات با قاعده در اعتبارسنجی اطلاعات استفاده می‌شود. در درس‌های آینده در مورد خصوصیات مختلف رشته‌ها و متدهایی که به وسیله آن‌ها شما می‌توانید این خصوصیات را دستکاری کنید توضیح داده خواهد شد.

کلاس String

رشته‌ها در جاوا با کلاس `String` سرو کار دارند. برای ایجاد یک رشته چندین راه وجود دارد:

```

String myString;
myString = "An example of a string.";

```

به این نکته توجه کنید که رشته‌ها را باید در داخل دابل کوتیشن ("") قرار دهید. دابل کوتیشن به کمپایلر نشان می‌دهد که داده‌ای را که شما می‌خواهید در متغیر ذخیره کنید یک رشته است. برای تعریف و مقدار دهی به یک رشته می‌توان به صورت زیر هم عمل کرد:

```

String myString = "This is another string";

```

همچنین می‌توان از کلمه کلیدی `new` و سازنده `String` برای تخصیص یک مقدار به رشته استفاده کرد:

```

String myString = new String("This is a string.");

```

یک رشته مجموعه‌ای از کاراکترها می‌باشد. کد زیر نحوه جدا سازی کاراکترهای یک رشته را به وسیله حلقه `for` نشان می‌دهد.

```

String myString = "This is a string";

```

```
for (int i = 0; i < myString.length(); i++)
{
    System.out.println(myString.charAt(i));
}
```

T
h
i
s

i
s

a

s
t
r
i
n
g

در حقیقت، نسخه دیگر سازنده String یک آرایه از نوع کاراکتر قبول می‌کند.

```
char[] charArray = { 'H', 'e', 'l', 'l', 'o' };
String myString = new String(charArray);
```

در درس آینده نحوه مقایسه و مرتب سازی رشته‌ها را نشان خواهیم داد.

مقایسه رشته‌ها

می‌توان رشته‌ها را به روش‌های مختلف با هم مقایسه کرد. به عنوان مثال با استفاده از عملگر == می‌توان تست کرد که آیا دو رشته با هم برابرند یا نه.

```
String str1 = "Hello";
String str2 = "Hello";
String str3 = "Goodbye";

System.out.println(MessageFormat.format("str1 == str2 : {0}", str1 == str2));
System.out.println(MessageFormat.format("str1 == str3 : {0}", str1 == str3));
```

str1 == str2 : True
str1 == str3 : False

همچنین از متدهای compareTo() هم برای مقایسه دو رشته استفاده می‌شود. به کد زیر توجه کنید:

```
String str1 = "Hello";
String str2 = "Hello";

System.out.println(str1.compareTo(str2));
```

0

متده است `compareTo()` یک رشته را میگیرد و آن را با رشته مورد نظر ما مقایسه میکند و در صورت مساوی بودن هر دو مقدار صفر را بر میگرداند. بنابراین برای تشخیص مساوی دو رشته با استفاده از متده است `compareTo()` باید مانند کد زیر تست کنید که آیا مقدار برگشتی برابر صفر است یا نه :

```
if(str1.compareTo(str2) == 0)
{
    // some code
}
```

چطور تشخیص بدھیم که یک رشته از رشته دیگر کوچکتر یا بزرگتر است؟ هر کاراکتر در رشته به یونیکد معادل خود تبدیل میشود. اولین کاراکتر اولین رشته با اولین کاراکتر دومین رشته مقایسه میشود. اگر برابر بودند سپس دومین کاراکترها و به همین ترتیب بقیه کاراکترها با هم مقایسه میشوند. متده است `compareTo()` دو رشته یکسان را که فقط در بزرگی و کوچکی حروف با هم متفاوت باشند را در رشته متفاوت در نظر میگیرد. برای مقایسه دو رشته بدون در نظر گرفتن بزرگی و کوچکی حروف از متده است `compareToIgnoreCase()` :

استفاده میشود :

```
String str1 = "HELLO";
String str2 = "Hello";

System.out.println(str1.compareToIgnoreCase(str2));
```

0

همانطور که مشاهده میکنید با وجودیکه رشته اول با حروف بزرگ نوشته شده است ولی متده است `compareToIgnoreCase()` باز هم مقدار صفر را بر میگرداند که نشان دهنده مساوی بودن دو رشته است. یکی دیگر از متدهای مقایسه رشتهها متده است `equals()` که در صورت مساوی بودن دو رشته مقدار `true` را بر میگرداند :

```
String str1 = "Hello";
String str2 = "Hello";

System.out.println(str1.equals(str2));
```

true

متدهای `equalsIgnoreCase()` هم دو رشته را بدون در نظر گرفتن کوچکی و بزرگی حروف با هم مقایسه و مقدار `true` یا `false` را بر می‌گرداند.

الحق یا چسباندن رشته‌ها

چندین راه برای الحق رشته‌ها به هم وجود دارد. الحق به معنای چسباندن چندین رشته به هم و تبدیل آن‌ها به یک رشته است. در جاوا یکی از راههای ساده الحق رشته‌ها استفاده از عملگر `+` است:

```
String str1 = "Happy ";
String str2 = "New Year";
String result = str1 + str2;

System.out.println(result);
```

Happy New Year

مشاهده می‌کنید که استفاده از این عملگر چطور باعث ترکیب دو عملوند رشته‌ای شد. به این نکته توجه کنید که استفاده از عملگر `+` باعث ترکیب یک رشته با یک نوع داده‌ای دیگر مانند `int` می‌شود که در این حالت نوع `int` به صورت خودکار به نوع رشته‌ای تبدیل می‌شود.

```
System.out.println("Number of Guests: " + 100);
```

Number of Guests: 100

راه دیگر برای چسباندن دو رشته استفاده از متدهای `concat()` می‌باشد. شما می‌توانید تعداد بی‌نهایت رشته یا شیء به عنوان آرگومان به این متدها ارسال کنید. در زیر نحوه استفاده از این متدها نشان داده شده است:

```
String str1 = "Hello";
String str2 = " World";

String str3 = str1.concat(str2);
System.out.println("String concat using String concat method : " + str3);
```

Hello World

استفاده از متدهای `join()` نیز یکی دیگر از روش‌های الحق رشته‌ها است. از این متدهای برای ترکیب رشته‌ها و ذخیره آن‌ها در یک آرایه از نوع رشته استفاده می‌شود.

```
String[] words = { "Welcome", "to", "my", "site." };
String result = String.join(" ", words);
```

```
System.out.println(result);
```

```
Welcome to my site.
```

متدهای `join()` دو آرگومان قبول می‌کند، اولین آرگومان، رشته‌ای است که در داخل هر یک از رشته‌هایی که قرار است با هم ترکیب شوند قرار می‌گیرد. که در مثال بالا یک رشته که در اصل یک فضای خالی است ("") را بین کلمات قرار می‌دهیم. دومین آرگومان خود آرایه رشته‌ای است. سربارگذاری دیگر این متدهای شما اجازه می‌دهد به جای آرایه رشته‌ای، لیست رشته‌ها را ارسال کنید.

```
String result = String.join(" ", "Welcome", "to", "my", "site.");
```

```
System.out.println(result);
```

```
Welcome to my site.
```

رشته‌ها در جاوا تغییر ناپذیر هستند. این بدین معنی است که متغیر رشته‌ای یکبار مقدار دهی می‌شود و مقدار آن تغییر نمی‌کند. وقتی مقدار یک رشته را در جاوا اصلاح می‌کنید یک نسخه جدید از رشته ایجاد و نسخه قبلی دور انداده می‌شود. بنابراین همه متدهایی که با رشته‌ها سروکار دارند یک نسخه از رشته اصلاح شده را بر می‌گردانند و نسخه اصلی دست نخورد باقی می‌ماند.

تکه تکه کردن رشته‌ها

اگر بخواهید یک رشته را به چند رشته تکه تکه کنید می‌توانید از متدهای `split()` استفاده نمایید. اجازه دهید نگاهی به سربارگذاری‌های مختلف این متدها بیندازیم. متدهای `split()` آرایه‌ای از رشته‌ها را بر می‌گردانند که هر عنصر از این آرایه شامل یک زیررشته است. اولین سربارگذاری این متدهای از کarakترها را قبول می‌کند و بر اساس آن‌ها تشخیص می‌دهد که رشته باید در چه جایی به قسمت‌های مختلف تقسیم شود.

```
String message = "The quick brown fox jumps over the lazy dog.";
String[] substrings = message.split(" ");

for (String s : substrings)
{
    System.out.println(s);
}
```

```
The
quick
brown
fox
jumps
over
the
```

```
lazy
dog.
```

در مثال بالا از کاراکتر فاصله (‘ ’) برای جدا کردن کلمات در رشته بالا استفاده کردہایم چون دو کلمه متوالی به وسیله فاصله از هم جدا می‌شوند. کلمات در آرایه substrings ذخیره می‌شوند. سپس با استفاده از دستور foreach آن‌ها را در خطوط جداگانه چاپ می‌کنیم. می‌توان تعداد زیررشته‌های برگشتی را به وسیله سربارگذاری دیگر متدهای Split() محدود کرد.

```
String message = "The quick brown fox jumps over the lazy dog.";
String[] substrings = message.split(" ", 3);

for (String s : substrings)
{
    System.out.println(s);
}
```

```
The
quick
brown fox jumps over the lazy dog.
```

همانطور که در مثال بالا مشاهده می‌کنید دومین آرگومان برای تشخیص تعداد زیررشته‌ها به کار می‌رود. خروجی نشان می‌دهد که دو کلمه اول از رشته جدا شده‌اند و مابقی رشته در عنصر آخر آرایه ذخیره می‌شود.

جستجوی رشته‌ها

جستجوی رشته‌ها به وسیله متدهای جاوا بسیار راحت است. اجازه بدھید که نگاهی به متدهای مختلفی که محل وقوع یک رشته خاص را پیدا می‌کنند بیندازیم. متدهای lastIndexOf() و indexOf() محل یک رشته خاص را در رشته دیگر نشان می‌دهند. اگر رشته مورد نظر پیدا نشود متدهای فوق مقدار -1 را بر می‌گردانند. به مثالی در مورد متدهای indexOf() توجه کنید :

```
String str = "The quick brown fox jumps over the lazy dog.";
int index = str.indexOf("quick");

System.out.println(str);
System.out.println("quick was found at position " + index);
```

```
The quick brown fox jumps over the lazy dog.
quick was found at position 4
```

متد `lastIndexOf()` یک رشته را که شما به دنبال آن هستید را دریافت می‌کند. در مثال بالا متدهای `length()` و `charAt()` را بر می‌گرداند چون اندیس کلمه `quick` عدد `4` است. به یاد داشته باشید که اندیس یا مکان از صفر شروع شده و تا `-1`- طول ادامه می‌باید بنابراین کاراکتر پنجم دارای اندیس `4` است. متدهای `lastIndexOf()` کمی متفاوت است :

```
String str = "A very very very good day.";
int index = str.lastIndexOf("very");

System.out.println(str);
System.out.println("Last occurrence of very was found at position " + index);

A very very very good day.
Last occurrence of very was found at position 12
```

متدهای `lastIndexOf()` با این تفاوت که اندیس آخرین محل وقوع رشته را بر می‌گرداند. در کد بالا آخرین محل وقوع کلمه `"very"` اندیس `12` است. اگر یک رشته خاص در هنگام جستجو پیدا نشود این دو متدهای `length()` و `lastIndexOf()` را بر می‌گردانند.

```
String str1 = "This is a sample string.";

System.out.println(str1);

if (str1.indexOf("Whatever") == -1)
{
    System.out.println("\"Whatever\" was not found in the string.");

}

This is a sample string.
"Whatever" was not found in the string.
```

یکی دیگر از سربارگذاری های این دو متدهای این است که یک آرگومان دومی را قبول می‌کنند محل شروع جستجو را تعیین می‌کند. به تکه کد زیر توجه کنید :

```
String str1 = "This is a sample string.";

System.out.println(str1);

if (str1.indexOf("This", 5) == -1)
{
    System.out.println("\"This\" was not found in the string.");

}

This is a sample string.
"This" was not found in the string.
```

همانطور که مشاهده می‌کنید با وجودیکه کلمه this در جمله بالا وجود دارد ولی چون ما محل شروع جستجو در رشته را از اندیس ۵

انتخاب کرده‌ایم، کلمه پیدا نمی‌شود. اگر بخواهیم تمام محل‌های وقوع یک کلمه را پیدا کنیم می‌توان به صورت زیر عمل کنیم :

```
int position = 0;
String str1 = "This is a long long long string...";

do
{
    position = str1.indexOf("long", position);

    if (position != -1)
        System.out.println(position);

    position++;
}
while (position > 0);
```

```
10
15
20
```

متدهم می‌تواند چک کند که آیا یک رشته در داخل رشته دیگر وجود دارد یا نه.

```
String str1 = "This is a sample string.";

System.out.println(str1);

if (str1.contains("sample"))
{
    System.out.println("\\"sample\\" exists in the string.");
```

```
This is a sample string.
"sample" exist in the string.
```

این متدهم مورد نظر وجود داشته باشد مقدار true و اگر وجود نداشته باشد مقدار false را بر می‌گرداند. هر دو متدهم برای یافتن یک رشته در ابتدا و انتهایی یک رشته خاص به کار می‌روند.

`startsWith()` و `endsWith()` برای

```
String str1 = "Apple";

System.out.println(str1);

if (str1.startsWith("A"))
{
    System.out.println("The word starts with A.");
}
if (str1.endsWith("e"))
{
    System.out.println("The word ends with e.");
```

```
}
```

```
Apple
The word starts with A.
The word ends with e.
```

تغییر بزرگی و کوچکی حروف یک رشته

می‌توان بزرگی و کوچکی حروف یک رشته را تغییر داد. به عنوان مثال یک رشته که متشکل از حروف کوچک است را می‌توان به حروف بزرگ تبدیل کرد. با استفاده از متدهای `toLowerCase()` و `toUpperCase()` می‌توان حروف رشته را بزرگ یا کوچک کرد.

```
String lowercase = "abc";
String uppercase = "ABC";

System.out.println("lowercase.toUpperCase() = " + lowercase.toUpperCase());
System.out.println("uppercase.toLowerCase() = " + uppercase.toLowerCase());

lowercase.toUpperCase() = ABC
uppercase.toLowerCase() = abc
```

به این نکته توجه کنید که اگر بخواهیم یک رشته به عنوان مثال یک جمله که از حروف بزرگ و کوچک تشکیل شده است، مثلاً حرف اول هر کلمه بزرگ و بقیه حروف کوچک نوشته شده باشند را با استفاده از متدهای `toUpperCase()` و `toLowerCase()` تغییر دهیم فقط حروف کوچک آن تغییر کرده و بزرگ می‌شوند و حروفی که از قبل بزرگ بوده‌اند تغییر نمی‌کنند. این نکته در مورد متدهای `toTitleCase()` نیز صدق می‌کند. حال بباید یک متدهای ایجاد کنیم که حرف اول کلمات هر رشته‌ای که به آن ارسال می‌شود را به صورت بزرگ و بقیه را به صورت کوچک تبدیل کند. در این برنامه از متدهای دستکاری رشته که تا به حال یاد گرفته‌ایم استفاده شده است.

```
1 package myfirstprogram;
2
3 import java.util.Scanner;
4
5 class MyFirstProgram
6 {
7     static String ToTitleCase(String str)
8     {
9         String[] words = str.split(" ");
10
11         for(int i = 0; i<words.length; i++)
12         {
13             String firstLetter = words[i].substring(0, 1);
14             String rest      = words[i].substring(1);
15             String result    = firstLetter.toUpperCase() + rest.toLowerCase();
16             words[i] = result;
17         }
18
19         return String.join(" ", words);
20     }
21 }
```

```

20
21
22 public static void main(String[] args)
23 {
24     Scanner input = new Scanner(System.in);
25
26     String str1;
27
28     System.out.print("Enter a string: ");
29     str1 = input.nextLine();
30
31     System.out.println("Converting to Title Case...");
32     str1 = ToTitleCase(str1);
33
34     System.out.println(str1);
35 }
36 }
```

در مثال بالا یک متدهای ایجاد کردند (خطوط ۲۰-۲۷) که یک رشته را به عنوان آرگومان قبول می‌کند. ابتدا با استفاده از متدهای split() رشته را به کلمات تشکیل دهنده‌اش تقسیم بندی می‌کنیم، بنابراین می‌توان هر رشته را به صورت جداگانه دستکاری کرد.

```
String[] words = str.split(" ");
```

سپس با استفاده از یک حلقه for در میان کلمات گردش می‌کنیم (خطوط ۱۷-۱۱).

```
String firstLetter = words[i].substring(0, 1);
String rest = words[i].substring(1);
```

اولین حرف هر کلمه را با استفاده از متدهای substring() استخراج و در یک متغیر برای استفاده‌های بعدی ذخیره می‌کنیم (خط ۱۵). باقیمانده حروف کلمات را استخراج می‌کنیم.

```
String result = firstLetter.ToUpper() + rest.ToLower();
```

حال رشته‌ها را با هم ترکیب می‌کنیم البته از متدهای toUpperCase() برای بزرگ کردن حروف اول و از متدهای toLowerCase() برای بزرگ کردن بقیه حروف کلمات استفاده می‌کنیم. بعد از این کار عناصر آرایه را با کلمات اصلاح شده جایگزین می‌کنیم.

```
return String.join(" ", words);
```

سپس کلمات اصلاح شده را با استفاده از متدهای join() ترکیب کرده و در بین آنها فضای خالی قرار می‌دهیم. سپس آنها را به متدهای فراخوان برگشت می‌دهیم.

استخراج و جایگزین کردن رشته‌ها

برای استخراج قسمتی از یک رشته می‌توان از متده استفاده کرد. این متده دو آرگومان قبول می‌کند که یکی اندیس شروع و دیگری طولی از رشته را که می‌خواهیم استخراج کنیم را نشان می‌دهد. به مثال زیر توجه کنید.

```
String str1 = "This is a sample string.";
String str2 = str1.substring(10, 16);
System.out.println("str1 = " + str1);
System.out.println("str2 = " + str2);

str1 = This is a sample string.
str2 = sample
```

استخراج را از اندیس ۱۰ شروع کرده‌ایم (آرگومان اول که عدد ۱۰ است). همانطور که مشاهده می‌کنید کلمه "sample" از اندیس ۱۰ شروع شده است (کاراکتر یازدهم). آرگومان دوم نشان می‌دهد که ما تا چندمین کاراکتر را می‌خواهیم استخراج کنیم. از آنجاییکه قرار است کلمه sample را استخراج کنیم و این کلمه ۶ حرفی است پس باید آرگومان دوم را ۱۶ بنویسیم یعنی قرار است از ۱۰ تا ۱۶ را استخراج کنم. اگر نخواهید که مکان قرار گرفتن کلمه "sample" را به صورت دستی شمارش کنید می‌توانید با استفاده از متده indexOf() این کار را انجام دهید.

```
String str2 = str1.substring(str1.indexOf("sample"), 16);
```

یکی دیگر از سربارگذاریهای متده substring() فقط یک آرگومان که برای تعیین کردن اندیس شروع استخراج به کار می‌رود را قبول می‌کند در نتیجه استخراج از این اندیس شروع شده و تا پایان رشته ادامه می‌یابد.

جایگزین کردن رشته‌ها با استفاده از متده replace

با استفاده از متده replace() می‌توان یک رشته خاص را با یک رشته دیگر عوض کرد. به عنوان مثال در کد زیر می‌توان کلمه "dog" را با کلمه "cat" عوض کرد.

```
String str1 = "That dog is a lovely dog.";
System.out.println(str1);
System.out.println("Replacing all dogs with cats...");
str1 = str1.replace("dog", "cat");
System.out.println(str1);
```

```
That dog is a lovely dog.
Replacing all dogs with cats...
```

```
That cat is a lovely cat.
```

متد (`replace()`) دو آرگومان قبول می‌کند. اولین آرگومان رشته‌ای است که می‌خواهیم آن را با یک رشته جدید جایگزین کنیم (رشته قدیم) و دیگری رشته جدید است. متد (`replaceFirst()`) تمام کلمات "dog" واقع در رشته را پیدا کرده و کلمه "cat" را جایگزین آن‌ها می‌کند. حال اگر بخواهید فقط اولین کلمه `dog` به وسیله کلمه `cat` جایگزین شود می‌توانید از متد (`replaceFirst()`) استفاده کنید :

```
String str1 = "That dog is a lovely dog.";
System.out.println(str1);
System.out.println("Replacing all dogs with cats...");
str1 = str1.replaceFirst("dog", "cat");
System.out.println(str1);
```

```
That dog is a lovely dog.
Replacing all dogs with cats...
That cat is a lovely dog.
```

متد (`replaceAll()`) هم یک سری کاراکتر به عنوان الگو می‌گیرد و در هر جای رشته که آن‌ها را پیدا کند با رشته یا کاراکتری که خودمان می‌خواهیم جایگزین می‌کند. فرض کنید که در داخل رشته‌ای علائمی مانند +، - و * قرار دارند و می‌خواهیم آن‌ها را حذف کنیم برای این کار به صورت زیر عمل می‌کنیم :

```
String str1 = "---+Hello +World!***";
String result = str1.replaceAll("[*+-]", "");
System.out.println(result);
```

```
Hello World!
```

همانطور که در کد بالا مشاهده می‌کنید این متد دو آرگومان می‌گیرد که اولین آرگومان کاراکترهایی هستند که آن‌ها را در داخل جفت کروشه به صورت رشته قرار داده‌ایم و دومین آرگومان هم که "" (جفت کوتیشن بدون فاصله) می‌باشد که برای حذف کردن کاراکترها به کار برده‌ایم.

فرمت بندی رشته‌ها و اعداد

از متد (`printf()`) هم می‌توان برای قالب بندی رشته‌ها و اعداد استفاده کرد. این متد بر اساس یک الگو و با استفاده از کاراکترهای خاصی که در جدول زیر آمده‌اند، قالب بندی را انجام می‌دهد. الگوی کلی قالب بندی رشته‌ها و اعداد به صورت زیر است :

```
% [flags] [width] [.precision] conversion-character
```

در الگوی بالا اجزایی که در داخل کروشه هستند اختیاری می‌باشند. در حالت عادی الگو با علامت % شروع می‌شود و بعد از آن یکی از

کاراکترهای جدول زیر می‌آید :

نمایش کاراکتر	%c
نمایش اعداد در مبنای ده (صحیح)	%d
نمایش اعداد اعشاری نمایی	%e
نمایش اعداد اعشاری	%f
نمایش اعداد صحیح	%i
نمایش اعداد مبنای ۸	%o
نمایش رشته	%s
نمایش اعداد ده دهی مثبت	%u
نمایش اعداد مبنای ۱۶	%x
چاپ علامت درصد	%%
چاپ علامت درصد	\%

حال فرض کنید که می‌خواهیم یک عدد اعشاری را فرمت بندی کنیم :

```
System.out.printf("%f", 34.789456);
```

همانطور که مشاهده می‌کنید در مثال بالا از % برای نمایش عدد اعشار استفاده کردہ‌ایم. همین مثال، مثال خوبی است که به شما نحوه

استفاده از اجزایی که در داخل کروشه هستند را نشان دهیم. ابتدا کاربرد flag یا نشانه را می‌گوییم. نشانه‌هایی که بعد از علامت درصد

می‌توان استفاده کرد در جدول زیر آمده‌اند :

نشانه	کاربرد
-	با اضافه کردن فاصله به سمت راست یک عدد یا رشته آن را به سمت چپ می‌کشد.
+	با اضافه کردن فاصله به سمت چپ یک عدد یا رشته آن را به سمت راست می‌کشد.
0	تعدادی صفر که خودمان تعیین کرده‌ایم به سمت راست یا چپ نوشته یا عدد اضافه می‌کند
فاصله	تعدادی فاصله که خودمان تعیین کرده‌ایم به سمت راست یا چپ نوشته یا عدد اضافه می‌کند

به کار بردن نشانه‌های بالا به تنها ی و بدون اینکه تعیین کنیم چه تعداد فاصله یا صفر می‌خواهیم به ابتدا یا انتهای عدد یا رشته اضافه کنیم بی معنی می‌باشد. در این صورت باید از جزء بعدی که `width` یا پهنا هست استفاده کنیم. `[precision].` هم در صورتی که متغیر از نوع اعداد اعشاری باشد، برای تعیین تعداد ارقام اعشار، و اگر از نوع رشته باشد تعداد کارکترهایی که قرار است نمایش داده شوند را مشخص می‌کند. حال به مثال بر می‌گردیم. فرض کنید که می‌خواهیم سه رقم از ارقام بعد از ممیز عدد اعشار مثال بالا را نشان داده و قبل از بخش صحیح آن سه عدد صفر قرار دهیم یعنی `00034.789`. انجام این کار بسیار راحت است :

```
System.out.printf("%09.3f", 34.789456);
```

```
00034.789
```

همانطور که در کد بالا مشاهده می‌کنید الگو با علامت % شروع می‌شود. سپس نشانه را می‌نویسیم که در اینجا ۰ است. اما اینکه چرا عدد ۹ را نوشتیم دلیلش این است که `34/789` با احتساب ممیز آن برای نمایش نیاز به شش جای خالی دارد و چون قرار است که ما سه صفر هم قبل از عدد `34` قرار دهیم پس باید ۹ جای خالی ایجاد کنیم. و اما `[۳]` هم به معنای سه رقم اعشار است و `f` هم که برای نمایش اعداد اعشاری به کار می‌رود. فرض کنید که می‌خواهیم سه کارکتر اول رشته `Programming` را نمایش دهیم، برای این منظور به صورت زیر عمل می‌کنیم :

```
System.out.printf("%.3s", "Programming");
```

```
Pro
```

در زیر هم مثال‌هایی از نحوه استفاده از متدها `printf` و کارکترهای خاص آن آمده است :

```
System.out.printf("Hello Welcome to JAVA Programming");
```

Hello Welcome to JAVA Programming

چاپ کاراکترهای کنترلی

```
System.out.printf("Hello Welcome to\t JAVA Programming");
```

Hello Welcome to JAVA Programming

همانطور که مشاهده می‌کنید در کد بالا استفاده از \t باعث ایجاد فاصله بین رشته‌ها می‌شود.

چاپ متغیرها

```
System.out.printf("Addition of two Numbers : %d", sum);
```

قالب بندی چند قسمتی

```
System.out.printf("I Love %c %s",'c',"Programming");
```

I Love c Programming

در کد بالا کاراکتر C جایگزین %c و رشته یا کلمه Programming جایگزین %s می‌شود.

قالب بندی اعداد اعشاری

```
System.out.printf("%-12s%-12s\n","Column 1","Column 2");
System.out.printf("%-12.5f%.20f", 12.23429837482,9.10212023134);
```

Column 1	Column 2
12.23430	9.10212023134000000000

درباره کد بالا یک نکته را یادآور می‌شویم و آن این است که اگر تعداد کاراکترهای یک رشته از تعدادی که ما برای قالب بندی آن استفاده کردہ‌ایم کمتر باشد تعدادی فاصله در سمت چپ یا راست رشته قرار می‌گیرد. مثلًاً تعداد کاراکترهای 1 Column 1 هشت عدد می‌باشد و ما برای قالب بندی و ایجاد فاصله در سمت راست آن عدد ۱۲- را به کار برده‌ایم. با این کار ۴ فاصله در سمت راست رشته قرار می‌گیرد و باعث فاصله آن با رشته بعدی می‌شود. حال اگر تعداد کاراکترهای رشته بیشتر از تعداد باشد که ما تعیین کردہ‌ایم، کل رشته یا عدد بدون هیچ گونه فاصله‌ای در سمت چپ یا راست، نمایش داده می‌شود. همین نکته در مورد مثال‌های زیر صدق می‌کند.

```
System.out.printf("%d", 1234);
System.out.printf("%3d", 1234);
System.out.printf("%6d", 1234);
System.out.printf("%-6d", 1234);
System.out.printf("%06d", 1234);
```

```
1 2 3 4
1 2 3 4- -
1 2 3 4
1 2 3 4 - -
0 0 1 2 3 4
```

نمایش رشته‌ها در قالب خاص

```
String str = "Programming";
System.out.printf("%s", str);
System.out.printf("%10s", str);
System.out.printf("%15s", str);
System.out.printf("%-15s", str);
System.out.printf("%15.5s", str);
System.out.printf("%-15.5s", str);
```

```
Programming
Programming
----Programming
Programming----
-----Progr
Progr-----
```

در دو کد آخر خطهای تیره در اصل فاصله هستند که برای روشن شدن موضوع آن‌ها را به صورت خط تیره نمایش داده‌ایم. برای روشن شدن کاربرد اعداد منفی و مثبت بعد از علامت % فرض کنید که شما می‌خواهید بین دو رشته JAVA و Programming چهار فاصله قرار دهید. این کار به دو صورت امکان پذیر است. یا بعد از کلمه JAVA چهار فاصله قرار دهید :

```
System.out.printf("%-8s%s", "JAVA", "Programming");
```

```
JAVA      Programming
```

که در مثال بالا -8- بع معنای این است که 8 مکان ایجاد شود که چهار تا از آن‌ها توسط کلمه JAVA اشغال می‌شود و چهار تای دیگر به خاطر علامت منفی در سمت راست کلمه JAVA قرار می‌گیرند. حالت دوم هم این است که در سمت چپ کلمه Programming چهار فاصله قرار بدهیم :

```
System.out.printf("%s%15s", "JAVA", "Programming");
```

JAVA Programming

که در این صورت باید عدد ۱۵ را بنویسیم. چونکه کلمه Programming یازده حرفی است، پس چهار مکان دیگر به خاطر مثبت بودن علامت ۱۵ در سمت چپ آن قرار می‌گیرند. روش دیگر برای قالب بندی رشته‌ها استفاده از متدهای String.format() از کلاس String است. این متدهای شیوه‌به‌شیوه به متدهای printf() عمل می‌کنند. به مثال زیر توجه کنید:

```
String Str1 = "JAVA Programming";
String Str2 = String.format("Str1 is %s", Str1);
String Str3 = String.format("value is %f", 32.33434);
String Str4 = String.format("value is %32.12f", 32.33434);
```

```
System.out.println(Str2);
System.out.println(Str3);
System.out.println(Str4);
```

Str1 is JAVA Programming
value is 32.334340
value is 32.334340000000

نکته‌ای که فراموش شد و الان یادآور می‌شویم در مورد اعداد اعشار است. اگر تعدادی که ما برای نمایش رقم اعشار مشخص کردہ‌ایم بیشتر از ارقام اعشار عدد باشد بسته به تعدادی که ما مشخص کردہ‌ایم عدد صفر بعد از ارقام اعشار قرار می‌گیرد. مثلًاً در مثال بالا [12.1.0] یعنی دوازده رقم اعشار نمایش داده شود و چون تعداد ارقام اعشار عدد در مثال بالا ۵ تا می‌باشد در نتیجه هفت عدد صفر بعد از آن نمایش داده می‌شود.

کلاس StringBuilder

کلاس String می‌تواند با استفاده از عملگر + دو رشته را به هم متصل کند. اما این عملگر برای الحال دو رشته مختلف کارا نیست. چون شیء رشته در جاوا تغییر ناپذیر است یعنی وقتی که یک متغیر از نوع رشته را تعریف می‌کنیم مقدار آن تغییر نمی‌کند. هنگامی که یک رشته را به یک رشته موجود می‌چسبانید مقدار قبلی حذف و یک شیء جدید که شامل دو رشته به هم چسبیده است به وجود می‌آید، مثلًاً:

```
String Animal = "Dog";
Animal = "Cat";
System.out.println(Animal);
```

Cat

با هر بار انجام این فرایند اشیاء موقتی ایجاد و اشیاء قدیمی از بین می‌روند و این ایجاد و حذف شدن‌ها هم زمان برنده و هم حافظه را

اشغال می‌کنند. به مثال زیر توجه کنید، مثلاً اگر بخواهید همه اعداد ۰ تا ۹۹۹۹ را به هم بچسبانید :

```
int counter = 9999;
String s = "";

for (int i = 0; i <= counter; i++)
{
    s += i;
}
System.out.println(s);
```

ممکن است با نگاه کردن به کد متوجه هیچ مشکلی نشوید اما اگر از متند System.currentTimeMillis() کلاس استفاده کنید متوجه

می‌شود که اجرای این برنامه چقدر زمان می‌برد. کد بالا را به صورت زیر اصلاح می‌کنیم :

```
1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int counter = 9999;
8         String s = "";
9
10        long startTime = System.currentTimeMillis();
11
12        for (int i = 0; i <= counter; i++)
13        {
14            s += i;
15        }
16
17        long endTime = System.currentTimeMillis();
18        long totalTime = endTime - startTime;
19        System.out.println(totalTime);
20    }
21 }
```

هدف از برنامه بالا تبدیل ۱۰۰۰ عدد به رشته و چسباندن آن‌ها به یکدیگر است. با استفاده از متند currentTimeMillis() کلاس System که زمان جاری را نشان می‌دهد یک کرونومتر ایجاد می‌کنیم که زمان شروع آن قبل از اجرای حلقه for (خط ۱۰) و زمان پایان آن بعد از اتمام حلقه (خط ۱۷) می‌باشد. به طور میانگین اجرای برنامه بالا ۱۵۵۳ زمان می‌برد (البته ممکن است این زمان در سیستم شما متفاوت باشد). حال اجازه دهید که با استفاده از کلاس StringBuilder این اعداد را به هم وصل کنیم. این کار را با استفاده از متند append() این کلاس انجام می‌دهیم.

```

1 package myfirstprogram;
2
3 public class MyFirstProgram
4 {
5     public static void main(String[] args)
6     {
7         int counter = 9999;
8         String s = "";
9
10        long startTime = System.currentTimeMillis();
11
12        StringBuilder sb = new StringBuilder();
13
14        for (int i = 0; i <= counter; i++)
15        {
16            sb.append(i);
17        }
18
19        long endTime = System.currentTimeMillis();
20        long totalTime = endTime - startTime;
21        System.out.println(totalTime);
22    }
23 }
```

مشاهده می‌کنید که اجرای برنامه حدود ۳۵۰۰ میلی‌ثانیه طول می‌کشد. همانطور که مشاهده کردید کارایی برنامه به طور چشمگیری بهبود یافت.

کارایی برنامه با افزایش متغیر حلقه بیشتر می‌شود. کلاس `StringBuilder` برای کار و اتصال رشته‌های زیاد به هم بسیار قدرتمند است.

این کلاس دارای متدهای مختلفی است که در زیر به برخی از آن‌ها اشاره شده است.

متدها	توضیحات
<code>append()</code>	برای چسباندن رشته‌ها یا کارکترها به رشته جاری به کار می‌رود.
<code>charAt()</code>	یک کاراکتر خاص از یک رشته را بر می‌گرداند.
<code>delete()</code>	برای حذف یک توالی از کارکترها از رشته اصلی به کار می‌رود.
<code>getChars()</code>	برای کپی یک توالی از کارکترهای یک رشته در یک آرایه کاراکتری به کار می‌رود.
<code>indexOf()</code>	محل‌های وقوع یک زیررشته در رشته اصلی را بر می‌گرداند.
<code>insert()</code>	برای قرار دادن یک توالی از کارکترها در یک رشته به کار می‌رود.
<code>lastIndexOf()</code>	آخرین محل وقوع یک زیررشته در رشته اصلی را بر می‌گرداند.

تعداد کاراکترهای یک رشته را بر می‌گرداند.	<code>length()</code>
یک توالی از کاراکترها را با توالی دیگر از کاراکترها جایگزین می‌کند.	<code>replace()</code>
برای معکوس کردن یک رشته به کار می‌رود.	<code>reverse()</code>
قرار دادن یک کاراکتر در یک اندیس خاص از رشته	<code>setCharAt()</code>
طول یک توالی از کاراکترها را بر می‌گرداند.	<code>setLength()</code>
برای جدا کردن یک توالی از کاراکترها از رشته اصلی به کار می‌رود.	<code>substring()</code>

File System

بیشتر اوقات لازم است که داده‌هایتان را در یک فایل دائمی ذخیره کنید. همچنین لازم است که به سیستم فایل‌ها یا دایرکتوری کامپیوتر دسترسی داشته باشید به طوری که بتوانید فایل‌ها را در آن‌ها قرار داده و ذخیره کنید، مکانی را که می‌خواهید فایل در آن ذخیره شود را نشان داده، فایلی که در یک دایرکتوری ذخیره شده را مشاهده کرده و به طور خلاصه جزئیاتی در مورد یک دایرکتوری خاص را به دست آورید.

شما حتی می‌توانید چک کنید که اندازه فایل چقدر است و آیا فایل از نوع فقط خواندنی است یا نه؟ اگرچه استفاده از پایگاه داده امروزه بیشتر ترجیح داده می‌شود، اما گاهی اوقات برای برنامه‌های کوچک استفاده از یک فایل متنی کارتر است. همچنین برخی از برنامه‌های کاربردی هنوز وجود دارند که داده‌هایشان را در یک فایل متنی ذخیره می‌کنند و بنابراین لازم است که اطلاعات در این فایل نوشته و از آن خوانده شود. جاوا دارای کلاس‌هایی است که شما با استفاده از آن‌ها می‌توانید محتویات فایل‌ها را بخوانید، در داخل آن‌ها بنویسید و صفات فایل‌ها و دایرکتوری‌ها را چک کنید. در درس‌های بعدی نحوه نوشتن یک داده باینری و فشرده سازی فایل‌ها که باعث کاهش حجم آن‌ها می‌شود را خواهید آموخت.

Java IO پکیج

پکیج Java IO دارای کلاس‌های مختلفی است که از آن‌ها برای انجام عملیاتی مانند ایجاد و حذف فایل‌ها، خواندن از فایل و نوشتن در آن و همچنین باز و بسته کردن فایل‌ها استفاده می‌شود، همچنین از این پکیج در مباحث پیشرفته برای کار در شبکه نیز استفاده می‌شود. قبل از توضیح این کلاس‌ها ابتدا بهتر است با دو کلمه `file` و `stream` آشنا شوید.

یک File مجموعه‌ای از داده‌های ذخیره شده در دیسک با یک نام خاص، پسوند و یک مسیر می‌باشد. وقتی فایل را با JAVA برای خواندن

نوشتن باز می‌کنید، فایل تبدیل به Stream می‌شود. Stream در لغت به معنای جریان است و در مجموع به یک توالی یا سری از بایت‌ها

که بین دو مسیر مبدأ و مقصد در حال حرکت هستند گفته می‌شود. برای کار با کلاس‌های پکیج Java IO ابتدا باید این پکیج را

import کنید:

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

در جدول زیر لیست کلاس‌های معمول این پکیج آمده است :

کلاس	توضیح
Reader	از این کلاس abstract برای خواندن کاراکتر از یک جریان بایتی به کار می‌رود.
Writer	از این کلاس abstract برای نوشتن کاراکتر در یک جریان بایتی به کار می‌رود.
InputStream	از این کلاس abstract برای خواندن از یک جریان بایتی به کار می‌رود.
OutputStream	از این کلاس abstract برای نوشتن در یک جریان بایتی به کار می‌رود.
InputStreamReader	برای خواندن داده از یک جریان بایتی به کار می‌رود.
OutputStreamWriter	برای نوشتن در یک جریان بایتی به کار می‌رود.
File	برای ایجاد، حذف، انتقال و انجام اعمال مختلف بر روی فایل‌ها و پوشش‌ها به کار می‌رود.
FileInputStream	برای خواندن از فایل به کار می‌رود.
FileOutputStream	برای نوشتن در فایل به کار می‌رود.
RandomAccessFile	برای خواندن از فایل و نوشتن در فایل به شیوه دسترسی تصادفی به کار می‌رود.
ByteArrayInputStream	برای خواندن از یک آرایه که حاوی داده‌های یک جریان بایتی است به کار می‌رود.

برای نوشتن در یک آرایه به عنوان یک جریان بایتی به کار می‌رود	ByteArrayOutputStream
برای خواندن یک شیء جاوای از یک جریان بایتی به کار می‌رود.	ObjectInputStream
برای نوشتن یک شیء جاوای در یک جریان بایتی به کار می‌رود. س	ObjectOutputStream
برای خواندن کاراکتر از یک جریان بایتی به کار می‌رود.	BufferedReader
برای نوشتن کاراکترها در یک جریان به کار می‌رود.	BufferedWriter
برای خواندن از یک استرینگ استفاده می‌شود.	StringReader
برای نوشتن روی یک استرینگ استفاده می‌شود.	StringWriter
کلاسی قدرتمند و کاربردی است که برای نوشتن در انواع مختلف خروجی استفاده می‌شود.	PrintWriter

حال که با کلاس‌های معمول این پکیج آشنا شدید در درس‌های آینده در مورد کار با آن‌ها توضیح می‌دهیم.

کلاس‌های Writer و Reader

اگرچه با استفاده از جریان‌های بایتی می‌توان هر نوع عملیات ورودی و خروجی را پیاده سازی کرد ولی استفاده مستقیم آن‌ها برای کار با داده‌های متنی بر یونیکد چندان آسان نیست و نیازمند تبدیلات میانی است برای راحتی کار با داده‌های متنی در جاوا از زیر کلاس‌های Reader برای خواندن و از زیر کلاس‌های Writer برای نوشتن استفاده می‌شود.

کلاس Reader

کلاس Reader یک کلاس انتزاعی (abstract) است که مستقیماً آن را به کار نمی‌بریم بلکه از زیر کلاس‌های آن که کابرد های متفاوتی دارند استفاده می‌کنیم. در صورتی که قصد داشته باشید برای یک کاربرد خاص ویژگی جدیدی به کلاس Reader اضافه کنید باید یک زیر کلاس از آن ایجاد کنید. در جدول زیر لیست زیر کلاس‌های مهم این کلاس که در آموزش‌های بعدی با آن‌ها آشنا می‌شویم آمده است :

کلاس	توضیح
InputStreamReader	برای خواندن داده از یک جریان بایتی به کار می‌رود.
BufferedReader	برای خواندن کاراکتر از یک جریان بایتی به کار می‌رود.
StringReader	برای خواندن از یک رشته استفاده می‌شود.

Writer

برای نوشتن کاراکتر در یک جریان بایتی از زیر کلاس‌های Writer استفاده می‌کنیم. کلاس Writer یک کلاس انتزاعی (abstract) است که مستقیماً آن را به کار نمی‌بریم بلکه از زیر کلاس‌های آن که کابرد های متفاوتی دارند استفاده می‌کنیم. در صورتی که قصد داشته باشید برای یک کاربرد خاص ویژگی جدیدی به کلاس Writer اضافه کنید باید یک زیر کلاس از آن ایجاد کنید. در جدول زیر لیست زیر کلاس‌های مهم این کلاس که در آموزش‌های بعدی با آن‌ها آشنا می‌شویم آمده است:

کلاس	توضیح
OutputStreamWriter	برای نوشتن در یک جریان بایتی به کار می‌رود.
BufferedWriter	برای نوشتن کاراکترها در یک جریان به کار می‌رود.
StringWriter	برای نوشتن روی یک رشته استفاده می‌شود.
PrintWriter	کلاسی قدرتمند و کاربردی است که برای نوشتن در انواع مختلف خروجی استفاده می‌شود.

حال که با کلاس‌های معمول این پیچید آشنا شدید در درس‌های آینده در مورد کار با آن‌ها توضیح می‌دهیم.

کلاس‌های OutputStream و InputStream

برای خواندن از یک جریان بایتی از زیر کلاس‌های InputStream استفاده می‌شود. کلاس InputStream یک کلاس انتزاعی (abstract) است که مستقیماً آن را به کار نمی‌بریم بلکه از زیر کلاس‌های آن که کابرد های متفاوتی دارند استفاده می‌کنیم. در صورتی که قصد داشته باشید برای یک کاربرد خاص ویژگی جدیدی به کلاس InputStream اضافه کنید باید یک زیر کلاس از آن ایجاد کنید. در جدول زیر لیست زیر کلاس‌های مهم این کلاس که در آموزش‌های بعدی با آن‌ها آشنا می‌شویم آمده است:

کلاس	توضیح
FileInputStream	برای خواندن از فایل به کار می‌رود.
ByteArrayInputStream	برای خواندن از یک آرایه که حاوی داده‌های یک جریان بایتی است به کار می‌رود.
ObjectInputStream	برای خواندن یک شیء جاوا‌بی از یک جریان بایتی به کار می‌رود.

برای نوشتن یک جریان بایتی از زیر کلاس‌های `InputStream` استفاده می‌شود. کلاس `OutputStream` یک کلاس انتزاعی (`abstract`) است که مستقیماً آن را به کار نمی‌بریم بلکه از زیر کلاس‌های آن که کابرد های متفاوتی دارند استفاده می‌کنیم. در صورتی که قصد داشته باشید برای یک کاربرد خاص ویژگی جدیدی به کلاس `OutputStream` اضافه کنید باید یک زیر کلاس از آن ایجاد کنید. در جدول زیر لیست زیر کلاس‌های مهم این کلاس که در آموزش‌های بعدی با آن‌ها آشنا می‌شویم آمده است:

کلاس	توضیح
FileOutputStream	برای نوشتن در فایل به کار می‌رود.
ByteArrayOutputStream	برای نوشتن در یک آرایه به عنوان یک جریان بایتی به کار می‌رود
ObjectOutputStream	برای نوشتن یک شیء جاوا‌بی در یک جریان بایتی به کار می‌رود.

کلاس File

در جاوا از کلاس `File` برای دستکاری فایل‌های موجود در یک دایرکتوری استفاده می‌شود. این کلاس دارای متدهایی است که به شما اجازه کپی، حذف، بازکردن، انتقال و ایجاد یک فایل را می‌دهند. کلاس `File` در پکیج `java.io` قرار دارد و باید آن را در برنامه وارد کنید

:

```
import java.io.File;
```

برخی از متدهای پر کاربرد کلاس `File` در جدول زیر آمده است :

متدها	کاربرد
createNewFile()	ایجاد یک فایل در یک مسیر خاص
delete()	حذف فایل
isFile()	چک می‌کند که آیا مسیر داده شده، یک فایل است یا نه؟
isDirectory()	چک می‌کند که آیا مسیر داده شده، یک پوشه است یا نه؟
exists()	چک کردن اینکه آیا یک فایل در یک مسیر خاص وجود دارد یا نه؟
renameTo()	برای انتقال و تغییر نام یک فایل به کار می‌رود.
getName()	نام فایل را بر می‌گرداند.
getParent()	پوشه یا درایوی که فایل در آن قرار دارد را بر می‌گرداند.
getPath()	مسیر فایل را بر می‌گرداند.
length()	حجم فایل را بر حسب بایت بر می‌گرداند.
lastModified()	زمان آخرین دستکاری فایل را بر می‌گرداند.
mkdir()	برای ایجاد یک پوشه به کار می‌رود
mkdirs()	برای ایجاد یک پوشه‌های تو در تو به کار می‌رود

ایجاد یک فایل

با استفاده از متدهای `createNewFile()` می‌توان یک فایل جدید ایجاد کرد. این متدهای یک مقدار بولی را بر می‌گردانند. اگر فایل ایجاد شود `true` و در غیر اینصورت `false` را بر می‌گردانند. البته قبل از استفاده از این متدهای یک نمونه از کلاس `File` ایجاد شده و سپس مسیری که قرار است فایل در آنجا ساخته شود به عنوان پارامتر به سازنده کلاس `File` ارسال شود. به مثال زیر توجه کنید :

```
File f = new File("C:\\\\Sample.txt");
```

```

try
{
    if(f.createNewFile())
    {
        System.out.println("The File is created!");
    }
}
catch(IOException e)
{
    System.out.println(e.getMessage());
}

```

نکته اینجاست که اگر فایلی از قبل به این نام وجود داشته باشد حذف نشده و فایل جدید جایگزین نمی‌شود. همانطور که در کد بالا مشاهده می‌کنید باید خطاهای ایجاد و یا عدم ایجاد فایل را مدیریت کنید. روش دیگر برای ایجاد فایل ایجاد `IOException` به صورت زیر است :

```

public static void main(String[] args) throws IOException
{
    File f = new File("C:\\\\Sample.txt");

    f.createNewFile();
}

```

با استفاده از متده `exists` می‌توان تست کرد که آیا مسیری که مورد نظر شماست وجود دارد، که در صورت پیدا کردن مسیر فایل یا دایرکتوری، مقدار `true` و در غیر اینصورت مقدار `false` را بر می‌گرداند.

```

File f = new File("C:\\\\text.txt");

if (f.exists())
{
    System.out.println("The File found!");
}

```

حذف یک فایل

برای حذف یک فایل از متده `delete()` استفاده می‌شود :

```
f.delete();
```

این متده هم یک مقدار بولی را بر می‌گرداند.

انتقال و تغییر نام یک فایل

برای انتقال یک فایل یک از مسیر به مسیر دیگر می‌توانید از متده `renameTo()` استفاده می‌شود :

```
File Sourcesample = new File("C:\\\\Sample.txt");
Sourcesample.renameTo(new File("D:\\\\" + Sourcesample.getName()));
```

در خط دوم کد بالا ما در داخل متدهای `renameTo()` یک شیء جدید ایجاد کرده‌ایم و مسیر جدید را که درایو D است به آن داده‌ایم. برای اینکه نام فایل هم تغییر نکند از متدهای `getName()` برای به دست آوردن نام فایل مبدأ استفاده کرده‌ایم. در کل معنی کد بالا این است که فایل اصلی را به درایو D و با همین نام منتقل کن. برای روشن شدن مطلب به کد زیر توجه کنید :

```
File Sourcesample = new File("C:\\\\Sample.txt");
File Destsample = new File("D:\\\\Sample.txt");
Sourcesample.renameTo(Destsample);
```

در کد بالا ما یک فایل مبدأ داریم با نام `Sourcesample` و می‌خواهیم آن را به درایو D منتقل کنیم. برای این کار یک شیء جدید که نشان دهنده مکان جدید است با نام `Destsample` ایجاد کرده و به عنوان پارامتر به متدهای `renameTo()` ارسال می‌کنیم. برای انتقال و تغییر نام همزمان هم می‌توانیم به صورت زیر مسیر و نام جدید را تعریف کنیم :

```
File Destsample = new File("D:\\\\NewName.txt");
```

به دست آوردن مسیر و والد یک فایل

برای به دست آوردن مسیر و والد یک فایل از متدهای `getName()` و `getParent()` به صورت زیر استفاده می‌شود :

```
File f = new File("C:\\\\Sample.txt");
System.out.println(f.getName());
System.out.println(f.getParent());
System.out.println(f.getPath());
```

```
Sample.txt
C:\\
C:\\\\Sample.txt
```

همانطور که در کد بالا مشاهده می‌کنید نام فایل، مکان و مسیر فایل نشان داده شده است. کلاس `File` دارای متدهای بیشتری برای کار با فایل است که در این درس به همین چند متدهای اکتفا می‌کنیم.

ایجاد یک پوشه و پوشه‌های تو در تو

برای ایجاد یک پوشه از متدهای `mkdir()` استفاده می‌شود :

```
File f = new File("C:\\\\Folder1");
f.mkdir();
```

کد بالا یک پوشه با نام Folder1 در درایو C ایجاد می‌کند. و برای ایجاد پوشه‌های تو در تو از متده استفاده کرد :

```
File f = new File("C:\\\\Folder1\\\\Folder2");
f.mkdirs();
```

کد بالا یک پوشه با نام Folder1 در درایو C ایجاد می‌کند که در داخل آن یک پوشه دیگر Folder2 قرار دارد.

به دست آورن فایل‌ها و پوشه‌های موجود در یک پوشه

برای به دست آوردن نام پوشه‌ها و فایل‌های موجود در یک پوشه از متده listFiles() استفاده می‌شود. یک پوشه به نام Folder در داخل درایو C ایجاد کرده و دو فایل و یک پوشه در داخل آن قرار دهید. سپس کدهای زیر را بنویسید :

```
package myfirstprogram;

import java.io.File;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File file = new File("C:\\\\Folder");
        if (file.isDirectory())
        {
            File[] files = file.listFiles();
            for (File f : files)
            {
                System.out.println(f.getName());
            }
        }
    }
}
```

در کد بالا ابتدا مسیر پوشه‌ای که در درایو C ایجاد کردہ‌ایم را در سازنده کلاس File نوشته‌ایم. در خط بعد با استفاده از متده isDirectory() چک می‌کنیم که آیا مسیر متعلق به یک پوشه است یا نه؟ اگر چنین بود در داخل بدن دستور if ابتدا یک آرایه از کلاس فایل ایجاد می‌کنیم، چون متده listFiles() یک آرایه از فایل‌ها را بر می‌کرداند. سپس با استفاده از یک حلقه foreach نام فایل‌ها را چاپ می‌کنیم.

به دست آوردن لیست زیر پوشه‌ها

همانطور که در مثال قبل دیدیم با استفاده از متدهای `listFiles()` و `FileFilter` می‌توانیم لیست فایل‌ها و زیر پوشه‌های یک پوشه خاص را به دست بیاوریم ولی گاهی نیاز داریم تنها زیر پوشه‌ها را لیست کنیم برای اینکار می‌توانیم از `FileFilter` استفاده کنیم. به مثال زیر دقت کنید:

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File file = new File("G:/MyFolder");
        FileFilter myFilter = new FileFilter()
        {
            @Override
            public boolean accept(File f)
            {
                return f.isDirectory();
            }
        };

        if(file.isDirectory())
        {
            File[] files = file.listFiles(myFilter);
            for(File f : files)
            {
                System.out.println(f.getName());
            }
        }
    }
}
```

مثال بالا لیست زیر پوشه‌های پوشه `MyFolder` را چاپ می‌کند. برای استفاده از `FileFilter` ابتدا یک شیء از آن ایجاد می‌کنیم، چون این کلاس یک `Interface` است مجبور به پیاده سازی متدهای `accept()` و `FileFilter` آن خواهیم بود، در داخل متدهای `accept()` مشخص می‌کنیم که چه نوع فایل‌هایی باید لیست شوند؟ در اینجا فقط فایل‌هایی که شرط `isDirectory()` برای آنها `true` باشد (به عبارتی پوشه باشند و نه یک فایل معمولی) لیست خواهند شد. برای استفاده از `myFilter` کافی است آن را به عنوان پارامتر به متدهای `listFiles()` ارسال کنیم. در مثال بعد با کاربرد `FileFilter` بیشتر آشنا خواهید شد. گاهی نیاز داریم تا فایل‌های خاصی را لیست کنیم، به عنوان مثال در یک برنامه مخصوص ویرایش متن ممکن است لازم داشته باشیم تنها فایل‌های با پسوند `.txt` چاپ شوند، خوشبختانه در این حالت هم می‌توانیم از `FileFilter` استفاده کنیم. مثال زیر تمام فایل‌های با پسوند `.txt` که در داخل پوشه `MyFolder` قرار دارند را چاپ می‌کند:

```
package myfirstprogram;
```

```

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File file = new File("G:/MyFolder");
        FileFilter myFilter = new FileFilter()
        {
            @Override
            public boolean accept(File f)
            {
                return f.getName().endsWith(".txt");
            }
        };

        if(file.isDirectory())
        {
            File[] files = file.listFiles(myFilter);
            for(File f : files)
            {
                System.out.println(f.getName());
            }
        }
    }
}

```

همانطور که مشاهده می‌کنید این مثال بسیار شبیه مثال قبلی است با این تفاوت که در اینجا به جای استفاده از شرط `isDirectory()` از شرط `endsWith()` استفاده می‌کند، متدهای `getName()` و `listFiles()` نام فایل را بر می‌گرداند و متدهای `accept()` و `FileFilter` به صورت مشخص می‌کند آیا نام فایل با پسوند `.txt` چاپ شوند. گاهی نیاز پیدا می‌کنیم برای لیست کردن فایل‌ها شرط‌های متفاوتی داشته باشیم باعث می‌شود تنها فایل‌هایی با پسوند `.txt` چاپ شوند. مثلاً زیر پوشه‌هایی که عناوان پارامتر ورودی `listFiles()` به عنوان ممکن است بخواهیم تمام فایل‌های `.txt` و `.exe` را لیست کنیم برای این کار کافی است از عملگر `||` برای ترکیب شرط‌ها استفاده کنیم. مثلاً زیر تمام فایل‌های `.exe` و همچنین تمام زیر پوشه‌هایی که عناوan `MyFolder` را لیست می‌کند :

```

package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File file = new File("G:/MyFolder");
        FileFilter myFilter = new FileFilter()
        {

```

```

@Override
public boolean accept(File f)
{
    return f.getName().endsWith(".exe") || f.isAbsolute();
}

if(file.isDirectory())
{
    File[] files = file.listFiles(myFilter);
    for(File f : files)
    {
        System.out.println(f.getName());
    }
}
}

```

برای به دست آوردن نام درایوهای اصلی سیستم هم از متده استفاده می کنیم :

```

package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File[] roots = File.listRoots();
        for (File f : roots)
        {
            System.out.println(f);
        }
    }
}

```

همانطور که مشاهده می کنید متده listRoots() یک آرایه از نوع File بر می گرداند که با یک حلقه for می توانیم اشیای داخل آن را مورد بررسی قرار دهیم. دقت کنید که متده static listRoots() یک متده است و برای استفاده از آن نیاز به ساختن یک شیء جدید از کلاس File نداریم و می توانیم مستقیماً آن را فراخوانی کنیم.

تفاوت متدهای listFiles و list

برای لیست فایلها و پوشش‌های داخل یک پوشه خاص می توانیم از یکی از متدهای list() یا listFiles() استفاده کنیم، تفاوت این دو متده در این است که متده list() تنها آرایه‌ای از String ها بر می گرداند که حاوی نام فایل‌هاست و مناسب برنامه‌هایی است که تنها به نام فایل نیاز داریم ولی متده listFiles() آرایه‌ای از نوع File بر می گرداند و این قابلیت را به ما می دهد که مستقیماً با زیر فایل‌ها و زیر پوشش‌های پوشش مورد نظر به سادگی کار کنیم.

تعیین حجم کلی و فضای خالی یک درایو

برای تعیین فضای خالی یک درایو می‌توانیم از متدهای `getFreeSpace()` استفاده کنیم، این متدها تعداد بایت‌های آزاد یک درایو را مشخص می‌کند، فرآخوانی این متدها فقط باید بر روی درایوهای اصلی انجام شود و بر روی سایر فایل‌ها نتیجه‌ای نخواهد داشت. مثال زیر فضای خالی درایو G را مشخص می‌کند :

```
File f = new File("G:");
System.out.println(f.getFreeSpace());
```

برای تعیین حجم یک درایو می‌توانیم از متدهای `getTotalSpace()` استفاده کنیم، این متدها حجم یک درایو را بحسب بایت در اختیار می‌دارند، مشابه متدهای قبلی این متدهای نیز باید روی درایوهای اصلی فرآخوانی شود. مثال زیر حجم درایو G را مشخص می‌کند :

```
File f = new File("G:/");
System.out.println(f.getTotalSpace());
```

کلاس InputStreamReader

در آموزش‌های قبلی با کلاس‌های انتزاعی `InputStream` و `Reader` آشنا شدیم، کلاس `InputStreamReader` به عنوان واسطه بین این دو کلاس عمل می‌کند، این کلاس داده‌ها را به صورت جریان بایتی از ورودی می‌خواند و بر اساس یک `charset` مشخص آنها را به جریان کاراکتری تبدیل می‌کند، مهم‌ترین کاربرد این کلاس استفاده از آن در کاربردهای شبکه است بدین صورت که اطلاعات به صورت جریان بایتی از شبکه خوانده می‌شوند و سپس به جریان متنی تبدیل می‌شوند. از این کلاس می‌توان برای خواندن از فایل، خواندن از وروردی سیستم (کنسول) و یا خواندن از یک حافظه بافر نیز استفاده کرد. برای استفاده از این کلاس باید از یکی از سازنده‌های زیر برای ساخت شیء جدید از آن استفاده کنیم :

```
InputStreamReader(InputStream in)
InputStreamReader(InputStream in, Charset cs)
InputStreamReader(InputStream in, String charsetName)
```

متدهای پر کاربرد `InputStreamReader` در جدول زیر آمده است :

متدها	کاربرد
close ()	جریان ورودی را بسته و منابع استفاده شده را آزاد می‌کند
read()	یک کاراکتر از جریان ورودی خوانده و آن را به صورت یک int بر می‌گرداند، در صورتی که به انتهای جریان ورودی رسیده باشیم خروجی این متده مقدار -1 خواهد بود.
read(char [] cbuf,int offset,int length)	کاراکترها را از جریان ورودی خوانده و در آرایه cbuf ذخیره می‌کند و تعداد کاراکترهای خوانده شده را برمی‌گرداند، در صورتی که کاراکتری برای خواندن وجود نداشته باشد خروجی این متده مقدار -1 خواهد بود. آرایه‌ای است که کاراکترها در آن بافر می‌شوند offset: اندیس شروع نوشتمن در آرایه مورد نظر length: حداکثر کاراکترهایی که باید خوانده شوند
ready ()	مشخص می‌کند که قادر به خواندن از جریان ورودی هستیم یا خیر، خروجی این متده به صورت true یا false است.

خواندن از فایل با متده read

در مثال زیر با استفاده از متده read() محتوی یک فایل متنی را کاراکتر به کاراکتر خوانده و چاپ می‌کنیم.

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        InputStreamReader inputstreamreader = null;
        FileInputStream fileinputstream = null;
        try
        {
            fileinputstream = new FileInputStream(myFile);
            inputstreamreader = new InputStreamReader(fileinputstream);
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        while ((ch = inputstreamreader.read()) != -1)
        {
            System.out.print((char) ch);
        }
    }
}
```

```
        int c;
        while ((c = inputstreamreader.read()) != -1)
        {
            System.out.print((char)c);
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    finally
    {
        try
        {
            inputstreamreader.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

همانطور که در کد بالا مشاهده می‌کنید ابتدا باید فایلی که قصد خواندن از آن داریم را با یک شیء از کلاس `File` مشخص کنیم (شی `myFile` در مثال بالا) سپس با استفاده از `FileInputStream` فایل مورد نظر را به یک جریان بایتی تبدیل کنیم (شی `fileInputStream` در مثال بالا) و در آخرین مرحله از تبدیلات، جریان بایتی را با استفاده از `InputStreamReader` به جریان متنی `inputStreamReader` تبدیل می‌کنیم (شی `inputStreamReader` در مثال بالا). در ادامه با استفاده از متده `read` کاراکتر به کاراکتر از `inputStreamReader` می‌خوانیم و تا زمانی که نتیجه فراخوانی این متده - نباشد آن را در خروجی چاپ می‌کنیم. از آنجایی که ممکن است فایل وجود نداشته باشد یا در هنگام عملیات خواندن با خطای پیش بینی نشده مواجه شویم باید حتماً از بلوک `try-catch` برای به دام انداختن خطاهای استفاده کنیم.

خواندن از فایل، با کمک یک آرایه

همانطور که در ابتداء دیدیم متند `read()` دیگری نیز وجود دارد که از یک آرایه بافر برای خواندن از جریان ورودی استفاده می‌کند، در بسیاری از کاربردها خواندن کاراکتر به کاراکتر زمان بر بوده و بهتر است با استفاده از یک آرایه بافر عملیات خواندن را تسريع بخشیم. مثال زیر به شرح این موضوع مربوط دارد:

```
package myfirstprogram;
```

```

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        InputStreamReader inputstreamreader = null;
        FileInputStream fileinputstream = null;

        char charbuffer[] = new char[200];
        int readed;
        try
        {
            fileinputstream = new FileInputStream(myFile);
            inputstreamreader = new InputStreamReader(fileinputstream);

            readed = inputstreamreader.read(charbuffer, 0, charbuffer.length);

            for (int i = 0; i < readed; i++)
            {
                System.out.print(charbuffer[i]);
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                inputstreamreader.close();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

متغیر readed تعداد کاراکترهای خوانده شده از فایل را مشخص می‌کند.

خواندن از ورودی سیستم (knssoul)

خواندن از ورودی سیستم مشابه خواندن از فایل است با این تفاوت که در این حالت دیگری نیازی به FileInputStream نداریم و با استفاده از System.in مستقیماً از ورودی سیستم می‌خوانیم. مثال زیر متنی را که کاربر در ورودی سیستم تایپ می‌کند (و در پایان کلید اینتر را می‌زند) در خروجی استاندارد چاپ می‌کند.

```

package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        InputStreamReader inputstreamreader = null;

        char charbuffer[] = new char[200];
        int readed;
        try
        {
            inputstreamreader = new InputStreamReader(System.in);

            readed = inputstreamreader.read(charbuffer, 0, charbuffer.length);

            for (int i = 0; i < readed; i++)
            {
                System.out.print(charbuffer[i]);
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                inputstreamreader.close();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }
}

```

خواندن از یک حافظه بافر

جريان ورودی می‌تواند مقادیر ذخیره شده در یک آرایه نیز باشد. در مثال زیر از یک آرایه به عنوان جريان ورودی استفاده می‌کنیم :

```

package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        String test = "Test String!";

```

```
byte[] memoryBuffer = test.getBytes();

InputStreamReader inputstreamreader = null;
ByteArrayInputStream bytearrayinputstream = null;

char charbuffer[] = new char[200];
int readed;
try
{
    bytearrayinputstream = new ByteArrayInputStream(memoryBuffer);

    inputstreamreader = new InputStreamReader(bytearrayinputstream);

    readed = inputstreamreader.read(charbuffer, 0, charbuffer.length);

    for (int i = 0; i < readed; i++)
    {
        System.out.print(charbuffer[i]);
    }
}
catch (IOException e)
{
    e.printStackTrace();
}
finally
{
    try
    {
        inputstreamreader.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
}
```

در مثال بالا از یک حافظه بافر (memoryBuffer در کد) به عنوان جریان ورودی استفاده کردیم، برای ساخت حافظه بافر ابتدا رشته test را به آرایه‌ای از بایت‌ها (متغیر memoryBuffer) تبدیل کردیم، سپس این آرایه از بایت‌ها را با استفاده از ByteArrayInputStream به یک جریان ورودی بایتی (متغیر bytearrayinputstream) تبدیل کردیم و در نهایت با استفاده از InputStreamReader جریان ورودی، را به یک جریان کاراکتری (inputstreamreader) تبدیل کردیم. مانند قسمت‌ها مشابه کدهای قبلی است.

OutputStreamWriter کلاس

در آموزش‌های قبلی با کلاس‌های انتزاعی `Writer` و `OutputStreamWriter` آشنا شدیم، کلاس `OutputStream` به عنوان واسطه بین این دو کلاس عمل می‌کند، با استفاده از این کلاس می‌توانیم یک جریان کاراکتری را به جریان پایتی تبدیل و ذخیره کنیم. بیشترین کاربرد این کلاس زمانی است که بخواهیم رشته‌هایی با فرمتهایی همچون `UTF-16`, `UTF-8` و ... را به صورت پایتی ذخیره کنیم و در عین حال

مطمئن باشیم که عمل decode به درستی صورت می‌گیرد. کار با این کلاس دشوار است و معمولاً از این کلاس به عنوان ابزار کمکی در کتاب کلاس‌های پیشرفته‌تر استفاده می‌شود. برای استفاده از این کلاس باید از یکی از سازنده‌های زیر برای ساخت شیء جدید از آن استفاده کنیم:

```
OutputStreamWriter (OutputStream out)
OutputStreamWriter (OutputStream out, Charset cs)
OutputStreamWriter (OutputStream out, String charsetName)
```

متدهای پر کاربرد OutputStreamWriter در جدول زیر آمده است:

متدها	کاربرد
close()	جریان ورودی را بسته و منابع استفاده شده را آزاد می‌کند
flush()	داده‌های بافر شده در حافظه را در استریم خروجی ارسال می‌کند.
write(char c)	کarakتر c را در استریم خروجی می‌نویسد.
write(char[] cbuf, int offset, int length)	قسمتی از آرایه cbuf را با شروع از اندیس offset به تعداد length کarakتر در استریم خروجی می‌نویسد.
write(String str, int offset, int length)	مشابه متدهای قبلی است با این تفاوت که به جای آرایه‌ای از کarakترها از یک String استفاده می‌کند.

نوشتن در فایل

مثال زیر یک متن آزمایشی را با فرمات UTF-8 در فایل Test.txt ذخیره می‌کند.

```
package myfirstprogram;
import java.io.*;
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        String sampleData = "Sample Text to Store";
```

```
FileOutputStream fileoutputstream = null;
OutputStreamWriter outputstreamwriter = null;

try
{
    fileoutputstream = new FileOutputStream(myFile);

    outputstreamwriter = new OutputStreamWriter(fileoutputstream, "UTF-8");

    outputstreamwriter.write(sampleData, 0, sampleData.length());

    outputstreamwriter.flush();
}
catch (IOException e)
{
    e.printStackTrace();
}
finally
{
    try
    {
        outputstreamwriter.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
}
```

RandomAccessFile کلاس

کلاس RandomAccessFile همانطور که از نام آن پیداست کلاسی با قابلیت "دسترسی تصادفی" به فایل است که به ما اجازه می‌دهد به صورت همزمان هم از فایل بخوانیم و هم در آن بنویسیم، با استفاده از این کلاس با فایل به مثابه یک آرایه بزرگ برخورد می‌کنیم، این فایل دارای ساختاری اندیسی یا اشاره گر گونه است که در هر لحظه به آخرین مکانی که عمل خواندن یا نوشتمن در آن انجام شده است اشاره می‌کند، این کلاس با متدهای مختلفی که در اختیار ما قرار می‌دهد کار با انواع داده‌های مختلف اعم از اعداد صحیح، اعداد ممیز شناور، رشته‌ها و ... را بسیار آسان می‌سازد. برای استفاده از این کلاس باید از یکی از سازنده‌های زیر برای ساخت شیء چدید از آن

استفاده کنیم :

```
RandomAccessFile (File file, String mod)  
RandomAccessFile (String fileName, String mod)
```

پارامتر mod

قبل از ادامه باید با یارامتر mod آشنا شویم، این یارامتر نحوه دسترسی به فایل را مشخص می‌کند و می‌تواند یکی از موارد زیر باشد:

معنی	مقدار
مشخص می‌کند که فقط قصد خواندن از فایل را داریم.	r
مشخص می‌کند که قصد خواندن و نوشتمن را داریم.	rw
مشخص می‌کند که قصد خواندن و نوشتمن داریم و تغییرات فایل یا متا دیتا باید سریعاً بر روی فایل اعمال شوند.	rws
مشابه مورد قبلی است با این تفاوت تغییر سریع متادیتا الزامی نیست.	rwd

متدهای پر کاربرد RandomAccessFile در جدول زیر آمده است:

متدها	کاربرد
close ()	جریان ورودی خروجی را بسته و منابع استفاده شده را آزاد می‌کند
length()	اندازه فایل را بر حسب بایت بر می‌گرداند.
read()	یک بایت از فایل می‌خواند.
read(byte[] b)	یک آرایه از بایتها را از فایل می‌خواند.
read(byte[] b,int offset,int length)	قسمتی از آرایه b را با شروع از آندیس offset به تعداد length بایت را از فایل می‌خواند.
readBoolean()	سعی می‌کند یک مقدار boolean از فایل بخواند.
readByte()	یک بایت علامت دار از فایل می‌خواند.
readChar()	یک کاراکتر از فایل می‌خواند.
readDouble()	یک مقدار double از فایل می‌خواند.
readFloat()	یک مقدار float از فایل می‌خواند.

متندی فوق العاده سریع برای خواندن یک آرایه از بایت است، بهتر است از این متند برای خواندن کل فایل در عملیاتی همچون کپی کردن فایل یا مشابه آن استفاده کنیم.	<code>readFully(byte[] b)</code>
یک مقدار <code>int</code> از فایل می‌خواند.	<code>readInt()</code>
یک خط کامل از فایل می‌خواند.	<code>readLine()</code>
یک مقدار <code>short</code> از فایل می‌خواند.	<code>readShort()</code>
یک هشت بیتی بی علامت از فایل می‌خواند.	<code>readUnsignedByte()</code>
یک شانزده بیتی بی علامت از فایل می‌خواند.	<code>readUnsignedShort()</code>
یک رشته با فرمت UTF را از فایل می‌خواند.	<code>readUTF()</code>
مکان خوان از فایل را به موقعیت <code>pos</code> تغییر می‌دهد.	<code>seek(long pos)</code>
اندازه فایل را تغییر می‌دهد.	<code>setLength(long length)</code>
یک آرایه از بایتها را در فایل می‌نویسد.	<code>write(byte[] b)</code>
یک آرایه از بایتها را با شروع از انديس <code>off</code> و به اندازه <code>len</code> در فایل می‌نویسد.	<code>write(byte[] b,int off,int len)</code>
یک مقدار <code>boolean</code> در فایل می‌نویسد.	<code>writeBoolean(boolean v)</code>
یک بایت در فایل می‌نویسد.	<code>writeByte(int v)</code>
یک رشته را به صورت آرایه‌ای از بایتها در فایل می‌نویسد.	<code>writeBytes(String s)</code>
یک کاراکتر در فایل می‌نویسد.	<code>writeChar(int v)</code>
یک رشته را به صورت آرایه‌ای از کاراکترها در فایل می‌نویسد.	<code>writeChars(String s)</code>

یک double در فایل می‌نویسد.	writeDouble(double v)
یک مقدار float در فایل می‌نویسد.	writeFloat(float v)
یک مقدار int در فایل می‌نویسد.	writeInt(int v)
یک مقدار long در فایل می‌نویسد.	writeLong(long v)
یک مقدار short در فایل می‌نویسد.	writeShort(int v)
یک رشته با فرمت UTF را در فایل می‌نویسد.	writeUTF(String s)

خواندن از فایل

در مثال زیر از کلاس RandomAccessFile برای خواندن از فایل استفاده می‌کنیم، قبل از اجرای این مثال یک فایل به نام Test.txt برای خواندن از فایل استفاده می‌کنیم، قبل از اجرای این مثال یک فایل به نام

ایجاد کرده و یک محتوای دلخواه در داخل آن نوشته و ذخیره کنید. سپس از کد زیر برای خواندن فایل استفاده نمایید :

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        RandomAccessFile randomaccessfile = null;

        try
        {
            randomaccessfile = new RandomAccessFile(myFile, "r");

            String line;
            while ((line = randomaccessfile.readLine()) != null)
            {
                System.out.println(line);
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

در کد بالا پارامتر دوم `RandomAccessFile` مشخص می‌کند که تنها قصد خواندن از فایل را داریم، در ادامه در یک حلقه `while` با استفاده از متدهای `readLine()` یک خط از فایل می‌خوانیم و آن را چاپ می‌کنیم. زمانی که متدهای `readLine()` مقدار `null` برگرداند به انتهای فایل رسیده‌ایم و از حلقه `while` خارج می‌شویم.

نوشتن در فایل

در مثال زیر از `RandomAccessFile` برای نوشتن مقادیر مختلف در فایل استفاده می‌کنیم.

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        RandomAccessFile randomaccessfile = null;

        try
        {
            randomaccessfile = new RandomAccessFile(myFile, "rw");
            randomaccessfile.writeUTF("A B C\\n");
            randomaccessfile.writeInt(20);
            randomaccessfile.writeFloat(3.14f);
            randomaccessfile.writeInt(80);
            randomaccessfile.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

در مثال بالا پارامتر دوم `RandomAccessFile` برابر با `rw` است و این بدان معنی است که قصد خواندن و نوشتن در فایل را داریم. با استفاده از متدهای `writeUTF()` یک رشته در فایل می‌نویسیم بعد از آن یک مقدار `int` و سپس یک مقدار `float` و سپس یک مقدار `int` دیگر در فایل می‌نویسیم. اگر فایلی که در مثال قبلی ذخیره شد را در یک ادیتور متنی ساده مثل `notepad` باز کنید مشاهده می‌کنید که محتوای آن خوانا نیست، برای اینکه از صحت کد قبل مطمئن شویم کد دیگری می‌نویسیم و مقادیر نوشته شده در فایل را می‌خوانیم و چاپ می‌کنیم.

```
package myfirstprogram;

import java.io.*;
```

```

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        RandomAccessFile randomaccessfile = null;

        try
        {
            randomaccessfile = new RandomAccessFile(myFile, "rw");

            String s = randomaccessfile.readUTF();
            System.out.println(s);

            int a = randomaccessfile.readInt();
            System.out.println(a);

            float f = randomaccessfile.readFloat();
            System.out.println(f);

            int b = randomaccessfile.readInt();
            System.out.println(b);

            randomaccessfile.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}

```

کد بالا مشابه کد قبلی است با این تفاوت که این بار به جای عمل نوشتن از عمل خواندن برای تست استفاده می‌کنیم. باید دقت کنید که ترتیب خواندن باید مشابه ترتیب نوشتن باشد. اگر قصد استفاده از RandomAccessFile را دارید برای ذخیره اعداد در فایل از ادبیتور های متنی مانند notepad استفاده نکنید زیرا این مقادیر در حقیقت متن‌های ASCII هستند و نه اعداد واقعی و اگر با خوانده شوند مقادیری بی معنی خواهند بود. راه حل صحیح نوشتن اعداد در فایل این است که از خود کلاس RandomAccessFile برای نوشتن عدد استفاده کنید.

کلاس **ByteArrayInputStream**

کلاس **ByteArrayInputStream** به ما اجازه می‌دهد تا با یک آرایه از بایت‌ها که در RAM قرار دارد به عنوان یک جریان ورودی عمل کنیم و بتوانیم از کلاس‌هایی که زیر کلاس **InputStream** هستند در کنار آن بهره ببریم. این کلاس بیشتر به عنوان یک واسطه در کنار کلاس‌های دیگر استفاده می‌شود. کاربرد اصلی این کلاس در شبکه است، بدین صورت که اطلاعات به صورت آرایه‌ای از بایت‌ها از طریق

شبکه دریافت می‌شوند و سپس با استفاده از `ByteArrayInputStream` به یک جریان ورودی تبدیل می‌شوند. برای استفاده از این

کلاس باید از یکی از سازنده‌های زیر برای ساخت شیء جدید از آن استفاده کنیم :

```
ByteArrayInputStream (byte[] b)
ByteArrayInputStream (byte[] b,int offset,int length)
```

متدهای پر کاربرد `ByteArrayInputStream` در جدول زیر آمده است:

متدهای پر کاربرد	کاربرد
<code>close ()</code>	جریان ورودی را بسته و منابع استفاده شده را آزاد می‌کند
<code>read()</code>	یک بایت از جریان ورودی خوانده و آن را به صورت یک <code>int</code> بر می‌گرداند، در صورتی که به انتهای جریان ورودی رسیده باشیم خروجی این متده مقدار <code>-1</code> خواهد بود.
<code>read(byte[] b,int offset,int length)</code>	بایتها را از جریان ورودی خوانده و در آرایه <code>b</code> ذخیره می‌کند و تعداد بایتها خوانده شده را بر می‌گرداند، در صورتی که بایتی برای خواندن وجود نداشته باشد خروجی این متده مقدار <code>-1</code> خواهد بود. آرایه‌ای است که بایتها در آن بافر می‌شوند <code>offset</code> : اندیس شروع نوشتمن در آرایه مورد نظر <code>length</code> : حداکثر بایتهايی که باید خوانده شوند
<code>available()</code>	تعداد بایتهايی که باقی ماند اند را مشخص می‌کند.

خواندن از یک آرایه

در مثال زیر با استفاده از متده `read()` محتوی یک آرایه را بایت به بایت خوانده و چاپ می‌کنیم.

```
package myfirstprogram;
import java.io.*;
public class MyFirstProgram
{
    public static void main(String[] args)
```

```
{
    byte[] buffer = { 72, 101, 108, 108, 111 };

    ByteArrayInputStream byteinputarraystream;

    try
    {
        byteinputarraystream = new ByteArrayInputStream(buffer);

        int v;
        while ((v = byteinputarraystream.read()) != -1)
        {
            System.out.print((char)v);
        }

        byteinputarraystream.close();
    }
    catch (IOException e) { }
}
}
```

در مثال فوق آرایه مورد نظر از قبل آماده است در کاربردهای پیشرفته ممکن است این آرایه در زمان اجرا تولید شود (مثلاً در کاربردهای شبکه)، بعد از ساخت `ByteArrayInputStream` میتوانیم با آن به صورت یک `InputStream` معمولی بر خورد کنیم، در داخل حلقه `while` هر بار یک بایت از استریم ورودی میخوانیم در صورتی که مقدار خوانده شده (`v`) برابر با ۱ نباشد آن را به کاراکتر تبدیل و سپس چاپ میکنیم. خروجی کد بالا عبارت "Hello" خواهد بود.

تبدیل رشته به آرایه

در بسیاری از موارد قبل از استفاده از `ByteArrayInputStream` نیاز پیدا میکنیم تا یک رشته متنی را به آرایه‌ای از بایت‌ها تبدیل کنیم که در این صورت میتوانیم از متدهای `getBytes()` استفاده کنیم:

```
String s = "Hello Java";
byte[] arr = s.getBytes();
```

کلاس `ByteArrayOutputStream`

کلاس `ByteArrayOutputStream` کلاسی است که به ما اجازه می‌دهد یک استریم خروجی را به آرایه‌ای از بایت‌ها تبدیل کنیم، این کلاس یک کلاس واسط است و به عنوان یک ابزار کمکی در کنار کلاس‌های پیشرفته‌تر استفاده می‌شود، بیشترین کاربرد این کلاس در شبکه است که به ما اجازه می‌دهد انواع مختلفی از `OutputStream` ها را به آرایه‌ای از بایت تبدیل و در شبکه ارسال کنیم. برای استفاده از این کلاس باید از یکی از سازنده‌های زیر برای ساخت شیء جدید از آن استفاده کنیم:

```
ByteArrayOutputStream ()
ByteArrayOutputStream (int bufferSize)
```

متدهای پر کاربرد `ByteArrayOutputStream` در جدول زیر آمده است:

متدهای پر کاربرد	جزئیات
<code>close ()</code>	جریان خروجی را بسته و منابع استفاده شده را آزاد می‌کند.
<code>toByteArray()</code>	مهمترین متدهای این کلاس است و در انتهای عملیات می‌توانیم از آن برای تبدیل استریم به آرایه استفاده کنیم.
<code>toString()</code>	استریم را به رشته تبدیل می‌کند.
<code>toString(String charsetName)</code>	بر اساس یک Charset خاص استریم را به رشته تبدیل می‌کند.
<code>write(int byte)</code>	یک بایت در استریم می‌نویسد.
<code>write(byte[] b)</code>	یک آرایه از بایتها را در استریم می‌نویسد.
<code>write(byte[] b,int offset,int length)</code>	قسمتی از یک آرایه از بایتها را در استریم می‌نویسد.
<code>write(OutputStream out)</code>	محتوای استریم را در OutputStream دیگری می‌نویسد.

مثال

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        ByteArrayOutputStream baytearrayoutputstream = null;
        baytearrayoutputstream = new ByteArrayOutputStream();

        try
        {
            baytearrayoutputstream.write("Hello Java".getBytes());
            baytearrayoutputstream.write(" IO".getBytes());
        }
    }
}
```

```

        System.out.println(baytearrayoutputstream.toString());
        byte[] result = baytearrayoutputstream.toByteArray();
        baytearrayoutputstream.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
}

```

در مثال بالا ابتدا یک `ByteArrayOutputStream` ایجاد می‌کنیم. سپس چند بار از متدهای `write()` برای نوشتن در آن استفاده می‌کنیم، از آنجایی که متدهای `getBytes()` آرایه‌ای از بایت‌ها را می‌پذیرد لذا باید رشته‌ها را با استفاده از متدهای `getBytes()` به آرایه‌ای از بایت‌ها تبدیل کنیم. با استفاده از متدهای `toString()` محتوای کل استریم را به دست می‌آوریم و سپس آن را چاپ می‌کنیم و در انتها کل استریم را به آرایه‌ای از بایت‌ها (`result`) تبدیل می‌کنیم (که البته در این مثال از استفاده خاصی نمی‌کنیم و هدف فقط آشنایی با نحوه استفاده از متدهای `toByteArray()` بود).

کلاس‌های `ObjectOutputStream` و `ObjectInputStream`

در این آموزش با کلاس‌های `ObjectOutputStream` و `ObjectInputStream` برای خواندن و نوشتن شیء از فایل آشنا می‌شویم، از آنجایی که این دو کلاس مکمل یکدیگر هستند با هر دوی آن‌ها در یک آموزش آشنا می‌شویم. اگر از `ObjectOutputStream` برای نوشتن شیء در فایل استفاده کنیم حتماً باید از `ObjectInputStream` برای خواندن شیء از آن فایل استفاده کنیم و اگر قصد استفاده از `ObjectInputStream` را داشته باشیم باید مطمئن شویم که فایل مورد نظر قبلًا با `ObjectOutputStream` ایجاد شده است، لذا آموزش این دو کلاس در دو مبحث جدا ممکن نیست و در این آموزش با هر دو کلاس آشنا می‌شویم. مهم‌ترین کاربرد این دو کلاس ذخیره شیء و خواندن شیء از فایل است با این حال این دو کلاس متدهای کمکی دیگری نیز دارند که کار با داده‌های مختلف را ساده‌تر می‌کنند.

کلاس `ObjectOutputStream`

از این کلاس برای ذخیره سازی شیء در فایل استفاده می‌شود. برای استفاده از این کلاس باید از سازنده زیر برای ساخت شیء جدید از آن استفاده کنیم :

```
ObjectOutputStream (OutputStream out)
```

متدهای پر کاربرد `ObjectOutputStream` در جدول زیر آمده است:

متدها	کاربرد
جریان خروجی را بسته و منابع استفاده شده را آزاد می‌کند.	<code>close ()</code>
داده‌های بافر شده را به سمت خروجی ارسال می‌کند.	<code>flush()</code>
یک آرایه از بایت‌ها را در جریان خروجی می‌نویسد.	<code>write(byte[] b)</code>
یک قسمت از یک آرایه از بایت‌ها را در جریان خروجی می‌نویسد.	<code>write(byte[] b,int offset,int length)</code>
یک بایت در جریان خروجی می‌نویسد.	<code>write(int v)</code>
یک مقدار <code>boolean</code> در جریان خروجی می‌نویسد.	<code>writeBoolean(boolean b)</code>
یک بایت هشت بیتی را در جریان خروجی می‌نویسد.	<code>writeByte(int v)</code>
یک رشته را به آرایه‌ای از بایت‌ها تبدیل کرده و در جریان خروجی می‌نویسد.	<code>writeBytes(String str)</code>
یک کاراکتر شانزده بیتی را در جریان خروجی می‌نویسد.	<code>writeChar(int v)</code>
یک مقدار <code>double</code> را در جریان خروجی می‌نویسد.	<code>writeDouble(double d)</code>
یک مقدار <code>float</code> را در جریان خروجی می‌نویسد.	<code>writeFloat(float f)</code>
یک مقدار <code>int</code> را در جریان خروجی می‌نویسد.	<code>writeInt(int i)</code>
یک مقدار <code>long</code> را در جریان خروجی می‌نویسد.	<code>writeLong(long l)</code>
مهم‌ترین متدهای این کلاس است و یک شیء جاوای را در جریان خروجی می‌نویسد.	<code>writeObject(Object obj)</code>
یک مقدار <code>short</code> را در جریان خروجی می‌نویسد.	<code>writeShort(int v)</code>

کلاس ObjectInputStream

از این کلاس برای خواندن یک شیء از یک جریان ورودی استفاده می‌شود. برای استفاده از این کلاس باید از سازنده زیر برای ساخت شیء

جدید از آن استفاده کنیم :

```
ObjectInputStream (InputStream in)
```

متدهای پر کاربرد ObjectInputStream در جدول زیر آمده است:

کاربرد	متدها
جریان ورودی را بسته و منابع استفاده شده را آزاد می‌کند	close ()
یک بایت از جریان ورودی می‌خواند.	read()
یک آرایه از بایت‌ها را از جریان ورودی می‌خواند.	read(byte[] b)
به تعداد length بایت از جریان ورودی می‌خواند و با شروع از اندیس offset در آرایه b می‌نویسد.	read(byte[] b,offset,length)
یک مقدار boolean را از جریان ورودی می‌خواند.	readBoolean()
یک بایت هشت بیتی را از جریان ورودی می‌خواند.	readByte()
یک کاراکتر از جریان ورودی می‌خواند.	readChar()
یک مقدار double را از جریان ورودی می‌خواند.	readDouble()
یک مقدار float را از جریان ورودی می‌خواند..	readFloat()
یک مقدار را از جریان ورودی می‌خواند.	readInt()
یک مقدار long را از جریان ورودی می‌خواند.	long readLong()

مهمترین متد این کلاس است و یک شیء جاوایی را از جریان ورودی می‌خواند.	readObject()
یک مقدار short را از جریان ورودی می‌خواند.	readShort()
یک رشته متنی را از جریان ورودی می‌خواند.	readUTF()

نوشتن شیء در فایل

در اولین مثال از این بخش یک شیء ساده جاوایی از یک کلاس اختصاصی را در فایل می‌نویسیم. ابتدا یک کلاس ساده به نام Person

به صورت زیر ایجاد می‌کنیم :

```
package myfirstprogram;

import java.io.*;
import java.io.Serializable;

class Person implements Serializable
{
    String name;
    String lastName;
    int score;

    public Person(String n, String ln, int i)
    {
        this.name = n;
        this.lastName = ln;
        this.score = i;
    }

    @Override
    public String toString()
    {
        return "[" + name + " : " + lastName + " , " + score + "]";
    }
}
```

کلاس‌هایی که قسمت داریم اشیایی از آن را در فایل ذخیره کنیم باید حتماً از نوع Serializable باشند، در کلاس بالا متد () باشد، در فایل ذخیره می‌کنیم :

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
```

```
Person person = new Person("Jack", "Statham", 15);

File myFile = new File("D:\\Test.obj");
FileOutputStream fileoutputstream = null;
ObjectOutputStream objectoutputstream = null;

try
{
    fileoutputstream = new FileOutputStream(myFile);
    objectoutputstream = new ObjectOutputStream(fileoutputstream);
    objectoutputstream.writeObject(person);
    objectoutputstream.flush();
    objectoutputstream.close();
}
catch (IOException e)
{
    e.printStackTrace();
}
}
```

در کد بالا ابتدا یک شیء نمونه از کلاس مورد نظر ایجاد می‌کنیم (شی person)، چون قصد نوشتن این شیء در یک فایل را داریم باید یک نمونه از کلاس File ایجاد کرده (شی myFile) و سپس از روی آن یک FileOutputStream ایجاد کنیم (شی FileOutputStream)، سپس از روی FileOutputStream یک ObjectOutputStream جدید ایجاد می‌کنیم (شی ObjectOutputStream) و در نهایت با استفاده از متده writeObject شیء مورد نظر را در فایل می‌نویسیم. در انتها نیز متدهای flush() و close() را فراخوانی می‌کنیم.

خواندن شیء از فایل

و به جای آنها کدهای زیر را می‌نویسیم :

```
File myFile = new File("D:\\Test.obj");
FileInputStream fileInputStream = null;
ObjectInputStream objectInputStream = null;

try
{
    fileInputStream = new FileInputStream(myFile);
    objectInputStream = new ObjectInputStream(fileInputStream);
    Person objInFile = (Person)objectInputStream.readObject();
    System.out.println(objInFile);
    objectInputStream.close();
}
catch (IOException | ClassNotFoundException e)
{
```

```

    e.printStackTrace();
}

```

از آنجایی که قصد خواندن از فایل را داریم باید ابتدا یک نمونه از کلاس `FileInputStream` ایجاد کنیم، سپس یک `ObjectInputStream` جدید از روی `fileInputStream` ایجاد می‌کنیم (شی `fileInputStream`) و بعد از آن یک `ObjectInputStream` جدید از روی `fileInputStream` ایجاد می‌کنیم (شی `objectInputStream`). در نهایت با استفاده از متدهای `readObject()` و `readInt()` یک شیء از جریان ورودی می‌خوانیم ولی چون خروجی این متدهای نوع کلاس `Object` است باید آن را به کلاس مورد نظر `cast` کنیم. در انتها شیء خوانده شده را چاپ می‌کنیم و جریان ورودی را می‌بندیم. خروجی کد بالا به صورت زیر خواهد بود :

```
[Jack : Statham , 15]
```

نوشتن و خواندن چند شیء در فایل

با استفاده از متدهای `writeObject()` و `writeInt()` می‌توانیم چندین شیء در فایل بنویسیم ولی هنگام خواندن از فایل باید به همان ترتیبی که اشیا را

در فایل نوشتیم آن‌ها را از فایل بخوانیم. برای درک بهتر یک کلاس جدید به نام `Book` به صورت زیر ایجاد می‌کنیم :

```

package myfirstprogram;

import java.io.*;
import java.io.Serializable;

class Person implements Serializable
{
    String name;
    String lastName;
    int score;

    public Person(String n, String ln, int i)
    {
        this.name = n;
        this.lastName = ln;
        this.score = i;
    }

    @Override
    public String toString()
    {
        return "[" + name + " : " + lastName + " , " + score + "]";
    }
}

class Book implements Serializable
{
    String title;
    int pages;
}

```

```

public Book(String t, int p)
{
    this.title = t;
    this.pages = p;
}

@Override
public String toString()
{
    return "(" + this.title + " , " + this.pages + " )";
}
}

```

کلاس فوق مشابه کلاس Person است و به توضیح خاصی نیاز ندارد. کد نوشتن در فایل را به صورت زیر تغییر می‌دهیم :

```

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        Person firstperson = new Person("John", "Scith", 15);
        Person secondperson = new Person("Jason", "Obimiang", 20);

        Book book = new Book("Core Java", 400);

        File myFile = new File("D:\\Test.obj");
        FileOutputStream fileoutputstream = null;
        ObjectOutputStream objectoutputstream = null;

        try
        {
            fileoutputstream = new FileOutputStream(myFile);
            objectoutputstream = new ObjectOutputStream(fileoutputstream);

            objectoutputstream.writeObject(firstperson);
            objectoutputstream.writeObject(secondperson);
            objectoutputstream.writeObject(book);

            objectoutputstream.flush();
            objectoutputstream.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}

```

مشابه کد قبلی است با این تفاوت که این بار دو شیء از کلاس Person و یک شیء از کلاس Book در فایل نوشتم. کد خواندن شیء از

فایل را به صورت زیر تغییر می‌دهیم :

```

FileInputStream fileinputstream = null;
ObjectInputStream objectinputstream = null;

```

```

try
{
    File myFile = new File("D:\\Test.obj");
    fileInputStream = new FileInputStream(myFile);
    objectInputStream = new ObjectInputStream(fileInputStream);

    Person firstperson = (Person)objectInputStream.readObject();

    Person secondperson = (Person)objectInputStream.readObject();

    Book book = (Book)objectInputStream.readObject();

    System.out.println(firstperson);
    System.out.println(secondperson);
    System.out.println(book);

    objectInputStream.close();
}
catch (IOException | ClassNotFoundException e)
{
    e.printStackTrace();
}

```

این کد نیز بسیار مشابه کد خواندن شیء از فایل قبلی است، تنها نکته مهم این است که اشیا به همان ترتیبی که در فایل نوشته شده‌اند

باید از فایل خوانده شوند. خروجی کد بالا به صورت زیر خواهد بود :

```
[John : Scith , 15]
[Jason : Obimiang , 20]
( Core Java , 400 )
```

کلاس BufferedReader

همانطور که می‌دانیم کلاس‌های Reader برای خواندن از جریان کاراکتری به کار می‌روند، از آنجایی که خواندن کاراکتر به کاراکتر چندان بهینه نیست باید از تکنیک بافر کردن برای بهینه سازی و افزایش سرعت عمل خواندن استفاده کنیم، کلاس BufferedReader با استفاده از تکنیک بافر کردن عمل خواندن را بهبود می‌بخشد. از این کلاس می‌توان برای خواندن از فایل، خواندن از وروردی سیستم (کنسول) و یا خواندن از یک حافظه بافر نیز استفاده کرد. برای استفاده از این کلاس باید از یکی از سازنده‌های زیر برای ساخت شیء جدید از آن استفاده کنیم :

```

BufferedReader (Reader in)

BufferedReader (Reader in,int bufferSize)
```

متدهای پر کاربرد BufferedReader در جدول زیر آمده است:

متدها	کاربرد
close ()	جریان ورودی را بسته و منابع استفاده شده را آزاد می‌کند
read()	یک کاراکتر از جریان ورودی خوانده و آن را به صورت یک بر می‌گرداند، در صورتی که به انتهای جریان ورودی رسیده باشیم خروجی این متدها مقدار ۱ خواهد بود.
read(char[] cbuf)	یک آرایه از کاراکترها را از جریان ورودی می‌خواند.
read(char[] cbuf, offset, length)	کاراکترها را از جریان ورودی خوانده و در آرایه cbuf ذخیره می‌کند و تعداد کاراکترهای خوانده شده را بر می‌گرداند، در صورتی که کاراکتری برای خواندن وجود نداشته باشد خروجی این متدها مقدار ۱ خواهد بود. cbuf : آرایه‌ای است که کاراکترها در آن بافر می‌شوند offset : اندیس شروع نوشتن در آرایه مورد نظر length : حداقل کاراکترهایی که باید خوانده شوند
String readLine()	یک خط از جریان ورودی می‌خواند.
boolean ready ()	مشخص می‌کند که قادر به خواندن از جریان ورودی هستیم یا خیر، خروجی این متدها به صورت true یا false است.

خواندن از فایل با متدهای read()

در مثال زیر با استفاده از متدهای read() محتوی یک فایل متنی را کاراکتر به کاراکتر خوانده و چاپ می‌کنیم، قبل از ادامه یک فایل متنی با نام Test.txt در درایو D ایجاد کرده و یک متن دلخواه در داخل آن نوشته و از کدهای زیر برای خواندن آن استفاده کنید :

```
package myfirstprogram;
import java.io.*;
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        try
        {
            BufferedReader reader = new BufferedReader(new FileReader("D:\\Test.txt"));
            String str;
            while((str = reader.readLine()) != null)
                System.out.println(str);
            reader.close();
        }
        catch(FileNotFoundException e)
        {
            System.out.println("File not found");
        }
        catch(IOException e)
        {
            System.out.println("Error reading file");
        }
    }
}
```

```

{
    File myFile = new File("D:\\Test.txt");

    InputStreamReader inputstreamreader = null;
    FileInputStream fileinputstream = null;
    BufferedReader bufferreader = null;

    try
    {
        fileinputstream = new FileInputStream(myFile);
        inputstreamreader = new InputStreamReader(fileinputstream);
        bufferreader = new BufferedReader(inputstreamreader);

        int c;
        while ((c = bufferreader.read()) != -1)
        {
            System.out.print((char)c);
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
}

```

همانطور که در کد بالا مشاهده می‌کنید ابتدا باید فایلی که قصد خواندن از آن داریم را با یک شیء از کلاس `File` مشخص کنیم (شی `myFile` در مثال بالا) سپس با استفاده از `FileInputStream` فایل مورد نظر را به یک جریان بایتی تبدیل کنیم (شی `fileinputstream` در مثال بالا) و سپس جریان بایتی را با استفاده از `InputStreamReader` به جریان متنی تبدیل می‌کنیم (شی `inputstreamreader` در مثال بالا) ایجاد می‌کنیم. در ادامه با استفاده از متدهای `read()` کاراکتر به کاراکتر از `inputstreamreader` و `bufferreader` می‌خوانیم و تا زمانی که نتیجه فراخوانی این متدها نباشد آن را در خروجی چاپ می‌کنیم. از آنجایی که ممکن است فایل وجود نداشته باشد یا در هنگام عملیات خواندن با خطای پیش بینی نشده مواجه شویم باید حتماً از بلوک `try-catch` برای به دام انداختن خطاهای استفاده کنیم.

خواندن از فایل با کمک یک آرایه

همانطور که در ابتداء دیدیم متدهای `read()` دیگری نیز وجود دارد که از یک آرایه بافر برای خواندن از جریان ورودی استفاده می‌کند، در بسیاری از کاربردها خواندن کاراکتر زمان بر بوده و بهتر است با استفاده از یک آرایه بافر عملیات خواندن را تسریع بخشیم. مثال زیر به شرح این موضوع می‌پردازد :

```
package myfirstprogram;
```

```

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        InputStreamReader inputstreamreader = null;
        FileInputStream fileinputstream = null;
        BufferedReader bufferreader = null;

        char cbuf[] = new char[200];
        int readed;

        try
        {
            fileinputstream = new FileInputStream(myFile);
            inputstreamreader = new InputStreamReader(fileinputstream);
            bufferreader = new BufferedReader(inputstreamreader);
            readed = bufferreader.read(cbuf, 0, cbuf.length);

            for (int i = 0; i < readed; i++)
            {
                System.out.print(cbuf[i]);
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}

```

خروجی کد بالا مشابه مثال قبلی است. متغیر `readed` تعداد کاراکترهای خوانده شده از فایل را مشخص می‌کند.

خواندن با استفاده از متدهای `readLine()`

کد زیر مشابه کدهای قبلی است با این تفاوت که این بار از متدهای `readLine()` برای خواندن خط به خط استفاده کنیم.

```

package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        InputStreamReader inputstreamreader = null;
        FileInputStream fileinputstream = null;
        BufferedReader bufferreader = null;

        try

```

```
    {
        fileinputstream = new FileInputStream(myFile);
        inputstreamreader = new InputStreamReader(fileinputstream);
        bufferreader = new BufferedReader(inputstreamreader);

        String line;
        while ((line = bufferreader.readLine()) != null)
            System.out.println(line);

    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
```

استفاده از BufferedReader این است که داده‌های ورودی را در حافظه بافر می‌کند و سرعت خواندن را به شدت افزایش می‌دهد.

BufferedWriter کلاس

همانطور که می‌دانیم کلاس‌های `Writer` برای نوشتن استریم‌های متنی به کار می‌روند، با استفاده از کلاس `BufferedWriter` می‌توانیم عمل نوشتن را با تکنیک بافر کردن سریع‌تر سازیم، این کلاس ابتدا داده‌ها را در یک بافر در حافظه اصلی نگه داری می‌کند و پس از هر بار پر شدن بافر آن‌ها را به صورت گروهی و یک جا در جریان خروجی می‌نویسد که این عمل سرعت نوشتن در جریان خروجی را به شدت افزایش می‌دهد. برای استفاده از این کلاس باید از یکی از سازنده‌های زیر برای ساخت شیء جدید از آن استفاده کنیم:

```
BufferedWriter (Writer out)  
BufferedWriter (Writer out,int bufferSize)
```

متدهای پر کاربرد BufferedWriter در جدول زیر آمده است:

متدها	کاربرد
close ()	جریان خروجی را بسته و منابع استفاده شده را آزاد می‌کند.
flush()	داده‌های بافر شده در حافظه را در استریم خروجی ارسال می‌کند.
voir write(char c)	کاراکتر c را در استریم خروجی مینویسد.

آرایه‌ای از کاراکترها را در استریم خروجی می‌نویسد.	<code>write(char[] cbuf)</code>
قسمتی از آرایه <code>cbuf</code> را با شروع از اندیس <code>offset</code> به تعداد <code>length</code> کاراکتر در استریم خروجی می‌نویسد.	<code>write(char[] cbuf, offset, length)</code>
مشابه متد قبلی است با این تفاوت که به جای آرایه‌ای از کاراکترها از یک <code>String</code> استفاده می‌کند.	<code>write(String str, offset, length)</code>
یک رشته متنی را در استریم خروجی می‌نویسد.	<code>write(String s)</code>
یک جدا کننده خطوط در استریم خروجی می‌نویسد.	<code>newLine()</code>

نوشتن در فایل

مثال زیر یک متن آزمایشی را با فرمت UTF-8 در فایل Test.txt ذخیره می‌کند.

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        try
        {
            File myFile = new File("D:\\Test.txt");
            FileOutputStream fileoutputstream = new FileOutputStream(myFile);
            OutputStreamWriter outputstreamwriter = new OutputStreamWriter(fileoutputstream, "UTF-8");

            BufferedWriter bufferedwriter = new BufferedWriter(outputstreamwriter);

            bufferedwriter.write("Hello BufferedWriter");
            bufferedwriter.newLine();
            bufferedwriter.write("Second Line");
            bufferedwriter.newLine();
            bufferedwriter.write("Third Line");

            bufferedwriter.flush();
            bufferedwriter.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

در مثال بالا ابتدا یک شیء از کلاس `File` ایجاد می‌کنیم و سپس با استفاده از `FileOutputStream` یک جریان خروجی (fileoutputstream) بر روی آن ایجاد می‌کنیم، سپس یک `OutputStreamWriter` از روی `OutputStream` (شی outputstreamwriter) بر روی `fileoutputstream` ایجاد می‌کنیم تا بتوانیم به صورت کاراکتری بر روی جریان خروجی بنویسیم (شی outputstreamwriter)، در نهایت از روی `OutputStreamWriter` یک `BufferedWriter` ایجاد می‌کنیم تا مطمئن شویم عمل نوشتن با سرعت زیادی انجام می‌شود، در ادامه کد با استفاده از متدهای `write()` و `newLine()` بر روی `bufferedWriter` می‌نویسیم.

مهم‌ترین دلیل استفاده از `BufferedWriter` این است که قبل از نوشتن اطلاعات بر روی جریان خروجی آن‌ها را در حافظه اصلی (RAM) بافر می‌کند و داده‌ها را به صورت گروهی در جریان خروجی می‌نویسد که این عمل سرعت نوشتن را به شدت افزایش می‌دهد.

کلاس `StringReader`

می‌دانیم که کلاس‌های `Reader` برای خواندن کاراکتر از استریم‌های کاراکتری به کار می‌روند ولی همیشه منبع استریم فایل، منبعی در شبکه یا کنسول سیستم نیست گاهی نیاز داریم تا از یک رشته متنی از نوع `String` به عنوان یک استریم استفاده کنیم، در چنین شرایطی می‌توانیم از کلاس `StringReader` استفاده کنیم. برای استفاده از این کلاس باید از سازنده زیر برای ساخت شیء جدید از آن استفاده کنیم:

```
StringReader (String s)
```

متدهای پر کاربرد `StringReader` در جدول زیر آمده است:

متدها	کاربرد
<code>close ()</code>	جریان ورودی را بسته و منابع استفاده شده را آزاد می‌کند
<code>read()</code>	یک کاراکتر از جریان ورودی خوانده و آن را به صورت یک <code>int</code> بر می‌گرداند، در صورتی که به انتهای جریان ورودی رسیده باشیم خروجی این متده مقدار ۱ خواهد بود.
<code>read(char [] cbuf,int offset,int length)</code>	کاراکترها را از جریان ورودی خوانده و در آرایه <code>cbuf</code> ذخیره می‌کند و تعداد کاراکترهای خوانده شده را بر می‌گرداند، در

صورتی که کاراکتری برای خواندن وجود نداشته باشد خروجی این متده مقدار -1 خواهد بود.	
cbuf : آرایه‌ای است که کاراکترها در آن بافر می‌شوند offset : اندیس شروع نوشتمن در آرایه مورد نظر length : حداکثر کاراکترهایی که باید خوانده شوند	
یک آرایه از کاراکترها را از جریان ورودی می‌خواند.	read(char[] cbuf)
مشخص می‌کند که قادر به خواندن از جریان ورودی هستیم یا خیر، خروجی این متده به صورت true یا false است.	ready ()
از ns کاراکتر بعدی در استریم صرف نظر می‌کند.	skip(long ns)

در مثال زیر با استفاده از متده read() محتوی یک رشته متنی را کاراکتر به کاراکتر خوانده و چاپ می‌کنیم.

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        String data = "Hello String Reader!";
        StringReader sr = new StringReader(data);

        int c;
        try
        {
            while ((c = sr.read()) != -1)
            {
                System.out.print((char)c);
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

ابتدا از روی رشته متنی `data` یک `StringReader` جدید ایجاد می‌کنیم، سپس در داخل یک حلقه `while` با استفاده از متند `()`

یک کاراکتر خوانده و آن را در متغیر `c` ذخیره می‌کنیم، اگر مقدار این متغیر برابر با ۱- نباشد آن را در خروجی چاپ می‌کنیم.

کلاس `StringWriter`

کلاس `StringWriter` یک استریم در حافظه اصلی (RAM) در اختیار ما قرار می‌دهد که می‌توانیم بر روی آن بنویسیم و در نهایت استریم مورد نظر را به رشته متنی تبدیل کنیم، معمولاً زمانی از این کلاس استفاده می‌شود که ما از تعداد عملیات نوشتمن در زمان کامپایل آگاه نیستیم. این کلاس معمولاً به عنوان یک ابزار کمکی در کنار کلاس‌های دیگر استفاده می‌شود. برای استفاده از این کلاس باید از سازنده زیر برای ساخت شیء جدید از آن استفاده کنیم :

```
StringWriter()
```

متدهای پر کاربرد `StringWriter` در جدول زیر آمده است:

متند	کاربرد
<code>close()</code>	جریان خروجی را بسته و منابع استفاده شده را آزاد می‌کند
<code>flush()</code>	داده‌های بافر شده در حافظه را در استریم خروجی ارسال می‌کند.
<code>write(char c)</code>	کاراکتر <code>c</code> را در استریم خروجی می‌نویسد.
<code>write(char[] cbuf,int offset,int length)</code>	قسمتی از آرایه <code>cbuf</code> را با شروع از اندیس <code>offset</code> به تعداد <code>length</code> کاراکتر در استریم خروجی می‌نویسد.
<code>write(String str,int offset,int length)</code>	مشابه متند قبلی است با این تفاوت که به جای آرایه‌ای از کاراکترها از یک <code>String</code> استفاده می‌کند.
<code>write(char[] cbuf)</code>	آرایه‌ای از کاراکترها در استریم خروجی می‌نویسد.
<code>write(String s)</code>	رشته متنی <code>s</code> را در استریم خروجی می‌نویسد.

مهمترین متد این کلاس است و معمولاً در پایان کار استفاده می‌شود، این متد کل محتوای استریم را به `String` تبدیل می‌کند.

`toString()`

متداول‌ترین کاربرد `StringWriter` این است که چندین رشته متنی مختلف را در آن بنویسیم و در نهایت یک رشته متنی واحد از روی آن‌ها ایجاد کنیم، مثال زیر به شرح این موضوع می‌پردازد:

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        StringWriter stringwriter = new StringWriter();
        try
        {
            stringwriter.write("Hello \n");
            stringwriter.write("Java \n");
            stringwriter.write("IO\n");

            String result = stringwriter.toString();
            System.out.println(result);

            stringwriter.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

```
Hello
Java
IO
```

در کد بالا ابتدا یک `StringWriter` ایجاد می‌کنیم و چند خط در آن می‌نویسیم، در نهایت `stringwriter` را به یک رشته متنی (`result`) تبدیل و چاپ می‌کنیم. قبل‌دیدیم که برای خواندن استریم متنی از فایل باید از کلاس‌های `Reader` استفاده کنیم ولی در مثال زیر تنها با استفاده از کلاس `FileInputStream` و به کمک `StringWriter` محتوای یک فایل متنی را می‌خوانیم و چاپ می‌کنیم.

```
package myfirstprogram;
```

```

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");

        FileInputStream fileInputStream = null;
        StringWriter stringWriter = new StringWriter();

        try
        {
            fileInputStream = new FileInputStream(myFile);

            int c;
            while ((c = fileInputStream.read()) != -1)
            {
                stringWriter.write(c);
            }

            fileInputStream.close();

            System.out.println(stringWriter.toString());

            stringWriter.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}

```

در کد بالا ابتدا یک شیء از کلاس `File` ایجاد می‌کنیم، برای خواندن هر نوع جریانی از این فایل به یک `FileInputStream` نیاز خواهیم داشت (شیء `fileInputStream`)، در ادامه در داخل یک حلقه `while` کاراکتر به کاراکتر از `fileInputStream` می‌خوانیم و اگر مقدار خوانده شده برابر با `-1` نبود با استفاده از متدهای `write` و `toString` آن را در `stringWriter` می‌نویسیم، در نهایت با فراخوانی متدهای `close` و `printStackTrace` را به رشته متنی تبدیل می‌کنیم و سپس آن را چاپ می‌کنیم.

کلاس PrintWriter

کلاس `PrintWriter` کلاسی کمکی است که امکانات زیادی را در اختیار ما قرار می‌دهد و به ما اجازه می‌دهد به صورت مستقیم بر روی فایل یا خروجی کنسول سیستم یا هر جریان خروجی دیگری کار کنیم، به علاوه این کلاس از کلاس‌های بهینه سازی شده است و سرعت فوق العاده ای دارد. برای استفاده از این کلاس باید از یکی از سازنده‌های زیر برای ساخت شیء جدید از آن استفاده کنیم :

```

PrintWriter (File file)

PrintWriter(File file, String charsetName)

PrintWriter(OutputStream out)

PrintWriter(OutputStream out, boolean autoFlush)

PrintWriter(String fileName)

PrintWriter(String fileName, String charset)

PrintWriter(Writer out)

PrintWriter(Writer out, boolean autoFlush)

```

متدهای پر کاربرد `PrintWriter` در جدول زیر آمده است:

متدهای پر کاربرد	توضیح	نام
جریان خروجی را بسته و منابع استفاده شده را آزاد می کند.		<code>close ()</code>
داده های بافر شده در حافظه را در استریم خروجی ارسال می کند.		<code>flush()</code>
یک مقدار <code>boolean</code> را در استریم خروجی می نویسد.		<code>print(boolean b)</code>
یک کاراکتر را در جریان خروجی می نویسد.		<code>print(char c)</code>
یک آرایه از کاراکترها را در جریان خروجی می نویسد.		<code>print(char[] str)</code>
یک مقدار <code>double</code> را در جریان خروجی می نویسد.		<code>print(double d)</code>
یک مقدار <code>float</code> را در جریان خروجی می نویسد.		<code>print(float f)</code>
یک مقدار <code>int</code> را در جریان خروجی می نویسد.		<code>print(int i)</code>
یک مقدار <code>long</code> را در جریان خروجی می نویسد.		<code>print(long l)</code>

یک شیء را در جریان خروجی می‌نویسد.	<code>print(Object o)</code>
یک رشته متнی را در جریان خروجی می‌نویسد.	<code>print(String s)</code>
برای قالب بندی خروجی استفاده می‌شود.	<code>PrintWriter format(String format, Object..o)</code>
یک جدا کننده خط جدید در جریان خروجی می‌نویسد.	<code>println()</code>
یک مقدار <code>boolean</code> را در استریم خروجی می‌نویسد.	<code>println(boolean b)</code>
یک کاراکتر را در جریان خروجی می‌نویسد.	<code>println(char c)</code>
یک آرایه از کاراکترها را در جریان خروجی می‌نویسد.	<code>println(char[] str)</code>
یک مقدار <code>double</code> را در جریان خروجی می‌نویسد.	<code>println(double d)</code>
یک مقدار <code>float</code> را در جریان خروجی می‌نویسد.	<code>println(float f)</code>
یک مقدار <code>int</code> را در جریان خروجی می‌نویسد.	<code>println(int i)</code>
یک مقدار <code>long</code> را در جریان خروجی می‌نویسد.	<code>println(long l)</code>
یک شیء را در جریان خروجی می‌نویسد.	<code>println(Object o)</code>
یک رشته متنی را در جریان خروجی می‌نویسد.	<code>println(String s)</code>
یک کاراکتر را در جریان خروجی می‌نویسد.	<code>write(char c)</code>
آرایه‌ای از کاراکترها در جریان خروجی می‌نویسد.	<code>write(char[] cbuf)</code>
قسمتی از یک آرایه را در جریان خروجی می‌نویسد.	<code>write(char[] cbuf, int offset, int length)</code>
یک رشته متنی را در جریان خروجی می‌نویسد.	<code>write(String s)</code>
قسمتی از یک رشته متنی را در جریان خروجی می‌نویسد.	<code>write(String s, int offset, int length)</code>

از `PrintWriter` می‌توان برای قالب بندی خروجی استفاده کرد، این قالب بندی می‌تواند بر روی فایل خروجی، خروجی شبکه یا حتی خروجی کنسول سیستم باشد، مثال زیر نحوه استفاده از متده است `printf()` برای قالب بندی را نشان می‌دهد.

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        PrintWriter printwriter = new PrintWriter(System.out, true);

        printwriter.printf("Hello %5d\n", 7);
        printwriter.printf("Hello %5d\n", 77);
        printwriter.printf("Hello %5d\n", 777);
        printwriter.printf("Hello %5d\n", 7777);
        printwriter.printf("Hello %5d\n", 77777);
    }
}

Hello      7
Hello     77
Hello    777
Hello   7777
Hello  77777
```

با استفاده از `%d` مشخص می‌کنیم که قصد داریم یک عدد صحیح را چاپ کنیم، `%nd` مشخص می‌کند که می‌خواهیم به اندازه `n` کاراکتر فاصله ایجاد کنیم. در مثال بالا به جای مقدار ۵ اعداد دیگری را نیز امتحان کنید.

```
package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        PrintWriter printwriter = new PrintWriter(System.out, true);

        printwriter.printf("Hello %.3f\n", 5.87878787);
    }
}

Hello 5.879
```

در مثال بالا `.f` مشخص می‌کند که قصد چاپ کردن یک مقدار ممیز شناور را داریم، `.3f` مشخص می‌کند که قصد داریم تا سه رقم بعد از اعشار را چاپ کنیم، دقت کنید که ارقام گرد می‌شوند. در مثال زیر از `PrintWriter` برای نوشتن در فایل استفاده می‌کنیم.

```

package myfirstprogram;

import java.io.*;

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        File myFile = new File("D:\\Test.txt");
        PrintWriter printwriter = null;
        try
        {
            printwriter = new PrintWriter(myFile, "UTF-8");

            printwriter.println("Hello");
            printwriter.println("Java");
            printwriter.println("IO");

            printwriter.flush();
            printwriter.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}

```

در مثال بالا چون قصد نوشتمن در یک فایل را داریم ابتدا یک شیء از کلاس `File` ایجاد می‌کنیم، سپس از روی آن یک `PrintWriter` ایجاد می‌کنیم و در پارامتر دوم مشخص می‌کنیم که قصد داریم با فرمات UTF-8 در فایل بنویسیم، در ادامه با استفاده از متدهای `println()` سه خط در فایل می‌نویسیم و در انتهای `printwriter` را می‌بندیم.

زبان نشانه گذاری توسعه پذیر (XML)

زبان نشانه گذاری توسعه پذیر (XML) به شما اجازه می‌دهد که داده‌ها را در یک متن و قالب ساخت یافته ذخیره کنید. این زبان به طور گسترده به عنوان یک دیتابیس جایگزین و برای ذخیره اطلاعات مربوط به پیکربندی نرم افزارها به کار می‌رود. XML از لحاظ دستوری شبیه به HTML بوده و اگر با آشنایی داشته باشدید یادگیری این زبان برایتان راحت‌تر است. در زیر یک سند XML را مشاهده می‌کنید :

```

<Persons>
  <Person>
    <Name>John Smith</Name>
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person>
    <Name>Mike Folly</Name>
    <Age>25</Age>
    <Gender>Male</Gender>

```

```
</Person>
<Person>
  <Name>Lisa Carter</Name>
  <Age>22</Age>
  <Gender>Female</Gender>
</Person>
</Persons>
```

سند XML ترکیبی از عناصر XML می‌باشد. یک عنصر XML شامل یک تگ آغازی، یک تگ پایانی و داده‌ای است که در بین این دو تگ قرار می‌گیرد.

```
<open>data</close>
```

می‌توان بر اساس داده‌ای که یک عنصر XML در خود نگهداری می‌کند یک نام برای عنصر انتخاب کرد. به این نکته توجه کنید که عناصر به حروف بزرگ و کوچک حساس‌اند، بنابراین دو کلمه Person و person با هم متفاوت‌اند. XML فضاهای خالی را نادیده می‌گیرد، بنابراین به جای نوشتن یک فایل در یک خط می‌توانید آن را در چند خط بنویسید تا خوانایی آن بالاتر رود. بین عناصر XML ممکن است رابطه پدر-فرزندی وجود داشته باشد

```
<parent>
  <child1>data</child1>
  <child2>
    <grandchild1>data</grandchild1>
  </child2>
</parent>
```

سند XML بالا دارای اطلاعاتی برای سه شخص می‌باشد. هر سند XML باید دارای یک عنصر ریشه (root) باشد. در مثال اول این درس، عنصر Person، عنصر ریشه (پدر) و دیگر عناصر داخل آن در حکم فرزندان آن می‌باشند. جزئیات هر شخص در داخل عنصر Person قرار دارند. عناصر فرزند عنصر Person عبارت‌اند از Name، Age و Gender. صفات XML، روشی دیگر برای اضافه کردن داده به یک عنصر می‌باشند.

```
<Person name="John Smith">some data</Person>
```

عنصر بالا یک خاصیت به نام name دارد که مقدار آن John Smith می‌باشد. مقادیر باید در داخل کوتیشن (‘’) یا دابل کوتیشن (‘“’) قرار بگیرند. در زیر روش اضافه کردن صفات نشان داده شده است.

```
<element att1="value1" att2="value2" ... attN="valueN">data</element>
```

همانطور که مشاهده می‌کنید، می‌توان به یک عنصر چندین صفت اضافه کرد.

```
<Person name="John Smith" age="30" gender="Male">some data</Person>
```

اجازه دهید که به عناصر مثال ابتدای درس صفاتی اضافه کنیم.

```
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Mike Folly">
    <Age>25</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Lisa Carter">
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
</Persons>
```

عنصر Name هر شخص (person) را حذف و صفت معادل آن (name) را برای هر عنصر می‌نویسیم. اسناد XML می‌توانند دارای یک تعریف XML باشند. تعریف XML شامل اطلاعاتی درباره سند XML مانند نسخه (همیشه نسخه ۱/۰ پیشنهاد می‌شود) و نوع رمزگذاری (encode) متن آن می‌باشد.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

این تعریف در بالاترین بخش سند و درست قبل از عنصر اصلی نوشته می‌شود. برای فایل XML می‌توان توضیحات نیز نوشت. نحوه نوشتן توضیحات در XML به صورت زیر است.

```
<!-- This is an XML comment -->
```

می‌توان با استفاده از یک ویرایشگر متن ساده فایل‌های XML تولید کرد.

مدیریت فایل‌های XML

برای خواندن و نوشتمن فایل‌های XML در جاوا روش‌ها و کتابخانه‌های توکار و خارجی زیادی وجود دارند که با استفاده از آن‌ها می‌توانیم فایل‌های XML را بخوانیم، ایجاد کنیم و یا تحت پرس و جو قرار دهیم، به صورت کلی روش‌های تجزیه فایل‌های XML در جاوا را می‌توان به دو دسته زیر تقسیم کرد :

- روش‌های مبتنی بر درخت تجزیه
- روش‌های مبتنی بر استریم

قبل از اینکه با این دو روش آشنا شویم باید با مفهوم تجزیه (parsing) آشنا شویم.

تجزیه (Parser) فایل‌های XML

به روشهای گفته می‌شود که در طی آن در داخل یک فایل یا استریم مبتنی بر XML حرکت می‌کنیم و به المان‌ها، ویزگی‌ها و صفت‌های عناصر XML دسترسی پیدا می‌کنیم و می‌توانیم با آن‌ها به صورت اشیای قابل استفاده در جاوا رفتار کنیم.

تجزیه گر (Parser)

تجزیه گر (Parser) ابزاری است که عمل تجزیه را انجام می‌دهد، این ابزار می‌تواند بر اساس الگوریتم‌های استریمی یا الگوریتم‌های مبتنی بر درخت تجزیه پیاده سازی شود، از تجزیه گرهای مبتنی بر درخت تجزیه می‌توان به Parser DOM و از تجزیه گرهای مبتنی بر استریم می‌توان به SAX Parser اشاره کرد. کتابخانه‌های دیگری همچون JDOM، StAX و DOM4J نیز تجزیه گرهای دیگری هستند که می‌توان از آن‌ها نام برد. دو راهکار اصلی تجزیه و تحلیل فایل‌های XML در جاوا عبارت‌اند از :

- راهکار مبتنی بر DOM
- راهکار مبتنی بر SAX

راهکار مبتنی بر DOM

در این روش درخت تجزیه بر اساس Document Object Model DOM یا انحصاراً W3C است ساخته می‌شود، درخت تجزیه ساختمان داده‌ای پیچیده‌ای است که عناصر فایل XML را به صورت درختی مبتنی بر DOM نگه داری می‌کند.

در این روش یک بار کل محتوای XML خوانده می‌شود و سپس Parser آن را به یک درخت (درخت تجزیه) تبدیل می‌کند، بعد از آن می‌توانیم با دسترسی به گره‌های این درخت فایل XML مورد نظر را بخوانیم، در هنگام استفاده از این روش باید به نکات زیر دقت کنیم :

- در این روش یک بار محتوای فایل XML به صورت کامل خوانده می‌شود لذا تنها برای مواردی قابل استفاده است که محتوای XML را به صورت کامل در اختیار داریم (مثلاً فایل) و برای مواردی که داده به مرور زمان قابل تغییر باشد قابل استفاده نیست (مثلاً یک استریم قابل تغییر آنلاین)

- در این روش کل فایل یک بار از ابتدا تا انتها خوانده می‌شود لذا برای مواردی که حجم فایل بزرگ باشد مفید نیست.

- در این روش یک درخت تجزیه ساخته شده و این درخت در حافظه اصلی (RAM) قرار می‌گیرد که حجم آن ممکن است زیاد باشد لذا ممکن است مصرف حافظه زیادی داشته باشد.
- مرحله خواندن و تجزیه اولیه در این روش کند است.
- پس از ساخت درخت تجزیه به دلیل استفاده از ساختار درختی دسترسی به عناصر XML با سرعت بالایی صورت می‌گیرد.

راهکار مبتنی بر SAX

این راهکار یک راهکار استریمی است و معمولاً سریع‌تر از راهکار قبلی است، در این روش درخت تجزیه ساخته نمی‌شود و در نتیجه مصرف حافظه کمتری دارد، این روش برای مواردی که استریم به مرور زمان بزرگ‌تر می‌شود قابل استفاده است در هنگام استفاده از این روش باید به موارد زیر دقت کنیم :

- این روش هم با فایل و هم استریم‌های قابل تغییر سازگار است.
- برای مواردی که حجم فایل بزرگ باشد قابل استفاده است.
- نسبت به DOM مصرف حافظه کمتری دارد.
- مرحله خواندن و تجزیه اولیه در این روش سریع‌تر است.
- در این روش دسترسی به صورت بالا به پایین است و از حالت Random Access پشتیبانی نمی‌کند و اگر قصد داشته باشیم چندین بار به گره‌هایی خاص دسترسی داشته باشیم مفید نیست.

نکته: به صورت کلی DOM Parser از SAX Parser سریع‌تر است و تنها در مواردی که دسترسی محلی زیادی داشته باشیم از DOM به جای SAXParser استفاده می‌کنیم.

خواندن فایل‌های XML با DOM Parser

در این آموزش با خواندن فایل‌های XML با روش مبتنی بر DOM آشنا می‌شویم. قبل از هر کار یک فایل xml جدید به نام sample.xml در درایو D ایجاد می‌کنیم و محتوای زیر را در آن می‌نویسیم :

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
```

```

<Person name="Mike Folley">
  <Age>25</Age>
  <Gender>Male</Gender>
</Person>
<Person name="Lisa Carter">
  <Age>22</Age>
  <Gender>Female</Gender>
</Person>
<Person name="Jerry Frost">
  <Age>27</Age>
  <Gender>Male</Gender>
</Person>
<Person name="Adam Wong">
  <Age>35</Age>
  <Gender>Male</Gender>
</Person>
</Persons>

```

این فایل یک فایل XML ساده است که تگ ریشه آن Persons است. در داخل این تگ یک تگ Person و در داخل این تگ نیز دو تگ Gender و Age قرار دارند. با استفاده از کد زیر محتوای این فایل را می‌خوانیم و خروجی مناسب را برای آن ایجاد می‌کنیم:

```

1 package myfirstprogram;
2
3 import java.io.*;
4 import org.w3c.dom.*;
5 import javax.xml.parsers.*;
6
7 public class MyFirstProgram
8 {
9     static void printNode(Node n)
10    {
11        System.out.println("-----");
12        Element e = (Element)n;
13
14        // Name
15        System.out.println(n.getNodeName() + " Name = " + e.getAttribute("name"));
16        // Age
17        String Age = e.getElementsByTagName("Age").item(0).getTextContent();
18        System.out.println(n.getNodeName() + " Age: " + Age);
19        // Gender
20        String Gender = e.getElementsByTagName("Gender").item(0).getTextContent();
21        System.out.println(n.getNodeName() + " Gender: " + Gender);
22    }
23    public static void main(String[] args) throws Exception
24    {
25        File myXMLFile = new File("D:\\sample.xml");
26        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
27
28        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
29
30        Document doc = dBuilder.parse(myXMLFile);
31
32        String rootName = doc.getDocumentElement().getNodeName();

```

```

33     System.out.println(rootName);
34
35     NodeList list = doc.getElementsByTagName("Person");
36     int listSize = list.getLength();
37     System.out.println(listSize);
38
39     for (int i = 0;i < listSize; i++)
40     {
41         Node nodeI = list.item(i);
42         printNode(nodeI);
43     }
44 }
45
46 }
```

```

Persons
5
-----
Person name: John Smith
Person Age: 30
Person Gender: Male
-----
Person name: Mike Folley
Person Age: 25
Person Gender: Male
-----
Person name: Lisa Carter
Person Age: 22
Person Gender: Female
-----
Person name: Jerry Frost
Person Age: 27
Person Gender: Male
-----
Person name: Adam Wong
Person Age: 35
Person Gender: Male
```

در ادامه با بخش بخش کد بالا آشنا می‌شویم. قبل از هر کار باید ابزارهای مورد نیاز برای کار با XML، تجزیه گر مبتنی بر DOM و کتابخانه io را import کنیم (خطوط ۵-۷). در داخل تابع main که نقطه شروع برنامه است یک شیء از کلاس فایل ایجاد می‌کنیم و به فایلی که قصد خواندن از آن را داریم اشاره می‌کنیم (خط ۲۵). در خطوط ۲۶-۳۰ یک Document ایجاد می‌کنیم که همان درخت تجزیه مورد نظر ماست. دقت کنید که شیء myXMLFile پارامتر ورودی متده است. در خط ۳۲ گره ریشه (root) را به دست می‌آوریم. متده استخراج می‌شود. سپس در خط ۳۶ لیست تمام گره‌هایی که از نوع Person هستند را به دست می‌آوریم.

متده getElementsByTagName() لیستی از تمام تگ‌ها را بر اساس نامی که به آن ارسال می‌کنیم ب صورت یک لیست به ما می‌دهد. کلاس NodeList یک لیست برای نگه داری Node هاست. با استفاده از متده getLength() می‌توانیم اندازه این لیست را به دست

بیاوریم، خطوط ۳۷-۳۸ تعداد تگ‌های Person را چاپ می‌کند. با استفاده از یک حلقه for می‌توانیم به آیتم‌های داخل list دسترسی پیدا می‌کنیم و آن را در یک شیء از کلاس Node نگه داری کنیم (خطوط ۴۰-۴۴). خط ۴۲ هر آیتم از list را در یک شیء از نوع Node می‌کنیم و آن را در خط ۴۳ متدهای printNode را بر روی آن فراخوانی می‌کنیم. متدهای printNode و name از خودمان آن را نوشته‌ایم (۹-۱۲)، در داخل این متدهای ابتدا یک خط به عنوان جدا کننده چاپ می‌کنیم و سپس شیء ای از نوع Element تبدیل می‌کنیم (خطوط ۱۱-۱۲). می‌دانیم که تگ Person در فایل مورد نظر دارای صفتی به نام name است، با استفاده از متدهای getAttribute() و getElementsByTagName() می‌توانیم تمام تگ‌های داخلی را بر اساس نامشان لیست کنیم. با استفاده از متدهای item() و getAttribute() می‌توانیم یک گره با استفاده از متدهای getTextContent() و getAttributeValue() محتوای متنهای داخلی گره را چاپ کنیم.

خواندن فایل‌های XML با SAX Parser

در این آموزش با خواندن فایل‌های XML با روش مبتنی بر SAX آشنا می‌شویم. قبل از هر کار یک فایل xml جدید به نام sample.xml در درایو D ایجاد می‌کنیم و محتوای زیر را در آن می‌نویسیم :

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Mike Folley">
    <Age>25</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Lisa Carter">
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
  <Person name="Jerry Frost">
    <Age>27</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Adam Wong">
    <Age>35</Age>
    <Gender>Male</Gender>
  </Person>
</Persons>
```

این فایل یک فایل XML ساده است که تگ ریشه آن Persons است. در داخل این تگ یک تگ Person و در داخل این تگ نیز دو تگ

Age و Gender قرار دارند. روش مبتنی بر SAX اگرچه سریع‌تر است ولی برای خواندن درست نیاز داریم که یک Handler برای تعبیر تگ‌ها

و الامان‌ها ایجاد کنیم، برای این کار یک کلاس جدید به نام MyHandler ایجاد می‌کنیم و کد زیر را در آن می‌نویسیم (خطوط ۹-۴۵):

```

1 package myfirstprogram;
2
3 import org.xml.sax.Attributes;
4 import org.xml.sax.SAXException;
5 import org.xml.sax.helpers.DefaultHandler;
6
7 import java.io.File;
8 import javax.xml.parsers.SAXParser;
9 import javax.xml.parsers.SAXParserFactory;
10
11 class MyHandler extends DefaultHandler
12 {
13     boolean Age = false;
14     boolean Gender = false;
15
16     @Override
17     public void startElement(String uri, String localName, String qName, Attributes attr)
18     throws SAXException
19     {
20         if (qName.equalsIgnoreCase("Person"))
21         {
22             System.out.println("Person Name: " + attr.getValue("name"));
23         }
24         else if (qName.equalsIgnoreCase("Age"))
25         {
26             Age = true;
27         }
28         else if (qName.equalsIgnoreCase("Gender"))
29         {
30             Gender = true;
31         }
32     }
33
34     @Override
35     public void characters(char[] ch, int start, int length) throws SAXException
36     {
37         if (Age)
38         {
39             System.out.println("Person Age: " + new String(ch, start, length));
40         }
41         else if (Gender)
42         {
43             System.out.println("Person Gender: " + new String(ch, start, length));
44             System.out.println("-----");
45         }
46
47         Age = false;
48         Gender = false;
49     }
50 }
51
52 public class MyFirstProgram
53 {

```

```

54 public static void main(String[] args) throws Exception
55 {
56     File myXMLFile = new File("D:/sample.xml");
57
58     SAXParserFactory sFactory = SAXParserFactory.newInstance();
59
60     SAXParser saxParser = sFactory.newSAXParser();
61
62     MyHandler mHandler = new MyHandler();
63
64     saxParser.parse(myXMLFile, mHandler);
65 }
66 }
```

```

Person Name: John Smith
Person Age: 30
Person Gender: Male
-----
Person Name: Mike Folley
Person Age: 25
Person Gender: Male
-----
Person Name: Lisa Carter
Person Age: 22
Person Gender: Female
-----
Person Name: Jerry Frost
Person Age: 27
Person Gender: Male
-----
Person Name: Adam Wong
Person Age: 35
Person Gender: Male
-----
```

کلاس `MyHandler` زیر کلاس `DefaultHandler` است و برای کار با آن باید حداقل دو متده است: `startElement()` و `characters()`. در هنگام تجزیه استریمی از این هندر استفاده شده و در هر زمان که عمل تجزیه به یک تگ آغازین می‌رسد متده `startElement()` فراخوانی می‌شود و هر گاه که تجزیه گر وارد قسمت‌های متنی داخل یک تگ (مثلًا `Gender` در تگ `Male`) می‌شود متده `characters()` فراخوانی می‌شود. پس با توجه به موارد فوق متدهای `startElement()` و `characters()` را به صورتی که در کد بالا مشاهده می‌کنید، پیاده سازی می‌کنیم (خطوط ۴۷-۴۹).

در داخل متده استفاده از پارامتر `qName` می‌توانیم نام گره را مشخص کنیم، اگر نام گره `person` بود مقدار صفت `id` را چاپ می‌کنیم، پارامتر `attr` در هر گره صفت‌های گره را نگه داری می‌کند که با فراخوانی متده `getValue()` بر روی آن می‌توانیم به

یک صفت خاص دسترسی پیدا کنیم، در خط زیر به صفت `id` دسترسی پیدا می‌کنیم و مقدار آن را چاپ می‌کنیم. اگر تگی که وارد آن

می‌شویم Person نباشد یکی از دو `if` بعدی ممکن است اجرا شوند :

- اگر نام تگ `Age` باشد مقدار بولی `Age` را برابر `true` قرار می‌دهیم.

- اگر نام تگ `Gender` باشد مقدار بولی `Gender` را برابر `true` قرار می‌دهیم.

می‌دانیم که متدهای `character()` و `startElement()` بعد از متدهای `Character` و `StartElement` فراخوانی می‌شود، زیرا ابتدا یک تگ باز می‌شود، متن در داخل آن قرار می‌گیرد و سپس تگ بسته می‌شود مثل `<Gender>Male</Gender>`. پس هنگامی که وارد متدهای `Character` و `StartElement` می‌شویم. تنها یکی از موارد `Age` یا `Gender` می‌تواند برابر با `true` باشد. با استفاده از ساختار شرطی خطوط ۳۷-۴۵ مورد برابر با `true` را پیدا می‌کنیم و متن داخل آن را چاپ می‌کنیم.

نکته: در این پارامتر رشته مورد نظر به صورت یک آرایه از کاراکترها قرار دارد برای چاپ متن باید یک شیء جدید از `String` ایجاد کنیم و آرایه مورد نظر را به همراه پارامترهای `start` و `length` به آن ارسال کنیم.

در پایان تمام متغیرهای بولی را برابر با `false` قرار می‌دهیم (خطوط ۴۷ و ۴۸). می‌دانیم که با رسیدن به تگ آغازین بعدی مجدداً متدهای `startElement()` فراخوانی می‌شود و مجدداً موارد فوق تکرار می‌شود. حال که `Handler` مان آماده است می‌توانیم کد لازم برای خواندن و تجزیه فایل `xml` را بنویسیم (خطوط ۵۶-۶۴). در این خطوط، یک `SAXParser` ایجاد می‌کنیم و فایل و هندر مورد نظر را به آن می‌دهیم، مابقی کارها به عهده تجزیه گر خواهد بود.

نکته: در صورتی که فایل `xml` دارای ساختار پیچده‌ای باشد و تگ‌های تو در توی زیادی داشته باشیم نوشتن `Handler` دشوار می‌شود و احتمال خطأ افزایش می‌یابد، در چنین موارد استفاده از `DomParser` بهینه‌تر و کم خطاطر است.

ساخت XML با روش مبتنی بر DOM

در این آموزش با ساخت XML مبتنی بر DOM آشنا می‌شویم. در این آموزش قصد داریم فایل `xml` ساده‌ای به صورت زیر را با کد جاوا ایجاد کنیم :

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
```

```
</Person>
</Persons>
```

با استفاده از کد زیر می‌توانیم این ساختار را در فایلی به نام sample.xml ایجاد کنیم :

```

1 package myfirstprogram;
2
3 import java.io.File;
4 import javax.xml.parsers.*;
5 import javax.xml.transform.*;
6 import javax.xml.transform.dom.DOMSource;
7 import javax.xml.transform.stream.StreamResult;
8 import org.w3c.dom.*;
9
10 public class MyFirstProgram
11 {
12     public static void main(String[] args)
13         throws TransformerException, ParserConfigurationException
14     {
15         File myFile = new File("D:\\sample.xml");
16
17         DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
18         DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
19
20         Document doc = dBuilder.newDocument();
21
22         //Persons Root
23         Element Persons = doc.createElement("Persons");
24         doc.appendChild(Persons);
25
26         //Person Node
27         Element Person = doc.createElement("Person");
28         Persons.appendChild(Person);
29
30         //Set attribute for Person
31         Attr attr = doc.createAttribute("name");
32         attr.setValue("John Smith");
33         Person.setAttributeNode(attr);
34
35         // Age Node
36         Element Age = doc.createElement("Age");
37         Age.appendChild(doc.createTextNode("30"));
38         Person.appendChild(Age);
39
40         //Gender Node
41         Element Gender = doc.createElement("Gender");
42         Gender.appendChild(doc.createTextNode("Male"));
43         Person.appendChild(Gender);
44
45         //write to file
46         TransformerFactory tFactory = TransformerFactory.newInstance();
47         Transformer transformer = tFactory.newTransformer();
48
49         DOMSource source = new DOMSource(doc);
50         StreamResult result = new StreamResult(myFile);
```

```

51
52     //transform
53     transformer.transform(source, result);
54 }
55 }
```

در ادامه با بخش‌های مختلف کد بالا آشنا می‌شویم. ابتدا ابزارهای مورد نیاز را import می‌کنیم (خطوط ۸-۳). سپس در خط ۱۵ فایلی که قصد نوشتمن در آن را داریم مشخص می‌کنیم. در خط ۱۷ یک Document خالی ایجاد می‌کنیم. با استفاده از متده استفاده از متده createElement() یک المان جدید به نام Persons ایجاد می‌کنیم و با استفاده از متدهappendChild() آن را به doc اضافه می‌کنیم، چون doc خالی است یک المان جدید همان گره ریشه خواهد بود (خطوط ۲۳-۲۴). در ادامه گره دیگری به نام Person ایجاد می‌کنیم، می‌دانیم که گره Person زیر گره، گره Persons باشد لذا این بار متدهappendChild() را بر روی شیء Persons فراخوانی می‌کنیم (خط ۲۷). حال نوبت به ایجاد زیر گره‌های Person یعنی Age و Gender می‌رسد. همانطور که مشاهده می‌کنید خطوط ۳۶-۳۸ و ۴۱-۴۳ شبیه هم هستند. ابتدا با استفاده از متدهcreateTextNode() یک مقدار متنی بین این دو تگ ایجاد می‌کنیم و سپس با فراخوانی متدهappendChild() را بر روی شیء Person، آنها را به صورت زیر گره این گره در می‌آوریم. اگر برنامه را اجرا کرده و به درایو D بروید مشاهده می‌کنید که یک فایل XML به صورت ابتدای درس ایجاد شده است. تنها کاری که باقی مانده است ذخیره ساختار فوق در یک فایل است. برای این کار از کلاس Transformer استفاده می‌کنیم، از روی doc یک شیء از کلاس DOMSource ایجاد می‌کنیم و از روی فایلی که قصد نوشتمن در آن را داریم نیز شیء ای از کلاس StreamResult ایجاد می‌کنیم.

اشیای source و result را به متده transform ارسال می‌کنیم و کار تمام می‌شود. حال اگر مثلاً زیر گره Age دارای خاصیتی به نام age و مقدار Young باشد، کدهای زیر را جایگزین نمایید :

```

Element Age = doc.createElement("Age");
Attr attr1 = doc.createAttribute("age");
attr1.setValue("Young");
Age.setAttributeNode(attr1);
Age.appendChild(doc.createTextNode("30"));
Person.appendChild(Age);
```

همانطور که در کد بالا مشاهده می‌کنید، با استفاده از متده createAttribute() خاصیت را ایجاد کرده، با استفاده از متده setValue() یک مقدار به آن اختصاص داده و در نهایت با استفاده از متده setAttributeNode() آن را به گره مورد نظر اختصاص می‌دهیم. حال دوباره برنامه را اجرا و نتیجه را مشاهده کنید.

ساخت XML با روش مبتنی بر Stream

در این آموزش با ساخت XML مبتنی بر استریم آشنایی شویم. این روش نسبت به روش مبتنی بر DOM سریع‌تر و ساده‌تر است، به علاوه نوشتگاه‌ها و بستن آن‌ها در کد جاوا‌ای با ساختار XML مطابقت بیشتری دارد. در این آموزش قصد داریم فایل XML ساده‌ای به صورت زیر را با کد جاوا‌ای ایجاد کنیم:

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
</Persons>
```

با استفاده از کد زیر می‌توانیم این ساختار را در فایلی به نام sample.xml ایجاد کنیم:

```
1 package myfirstprogram;
2
3 import java.io.*;
4 import javax.xml.stream.*;
5
6 public class MyFirstProgram
7 {
8     public static void main(String[] args) throws Exception
9     {
10         File myFile = new File("D:\\sample.xml");
11         try (FileOutputStream fos = new FileOutputStream(myFile))
12         {
13             XMLOutputFactory xmlOFactory = XMLOutputFactory.newInstance();
14             XMLStreamWriter xmlSWriter = xmlOFactory.createXMLStreamWriter(fos, "UTF-8");
15
16             xmlSWriter.writeStartDocument("UTF-8", "1.0");
17
18             xmlSWriter.writeStartElement("Persons");
19             xmlSWriter.writeStartElement("Person");
20             xmlSWriter.writeAttribute("name", "John Smith");
21
22                 xmlSWriter.writeStartElement("Age");
23                 xmlSWriter.writeCharacters("30");
24                 xmlSWriter.writeEndElement();
25
26                 xmlSWriter.writeStartElement("Gender");
27                 xmlSWriter.writeCharacters("Male");
28                 xmlSWriter.writeEndElement();
29
30             xmlSWriter.writeEndElement();
31             xmlSWriter.writeEndElement();
32
33             xmlSWriter.writeEndDocument();
34
35             xmlSWriter.flush();
36             xmlSWriter.close();
37         }
38     }
39 }
```

در ادامه با بخش‌های مختلف کد بالا آشنا می‌شویم. ابتدا ابزارهای مورد نیاز را `import` می‌کنیم، در این روش به ابزارهای زیادی نیاز نداریم (خطوط ۴-۳). با استفاده از خط ۱۰ فایلی که قصد نوشتمن در آن را داریم مشخص می‌کنیم. برای نوشتمن با این روش نیاز به یک استریم خواهیم داشت، چون قصد ما نوشتمن در فایل باید از `FileOutputStream` استفاده کنیم، برای همین در خط ۱۱ `fos` را تعریف کرده‌ایم. با استفاده از خطوط ۱۳ و ۱۴ یک `XMLStreamWriter` ایجاد می‌کنیم که همانطور که از نام آن پیداست محتوای XML را بر روی یک استریم خروجی می‌نویسد. با استفاده از متدهای `writeStartDocument()` و `writeStartElement()` ابتدای فایل XML را مشخص می‌کنیم، منتظر با این متدهای انتهای کار باید متدهای `writeEndDocument()` و `writeEndElement()` را فراخوانی کنیم تا مطمئن شویم فایل به درستی بسته می‌شود (خط ۳۳). برای نوشتمن یک تگ آغازین از متدهای `writeStartElement()` استفاده می‌کنیم (۱۸). به عنوان مثال خط زیر تگ آغازین `<Person>` را ایجاد می‌کند. در هر زمان که نیاز به بستن یک تگ داشتیم از متدهای `writeEndElement()` و `writeEndDocument()` استفاده می‌کنیم و تگ مورد نظر بسته می‌شود. پس از باز کردن یک تگ با استفاده از متدهای `writeAttribute()` و `writeCharacters()` به سادگی می‌توانیم به تگ مورد نظر صفت جدید اضافه کنیم، پارامتر اول نام صفت و پارامتر دوم مقدار صفت است. به عنوان مثال در خط ۲۰ صفت `name` را با مقدار `John Smith` به تگ `Person` اضافه کرده‌ایم. برای نوشتمن مقدار متنی داخل یک تگ به سادگی از متدهای `writeStartElement()` و `writeEndElement()` استفاده می‌کنیم، به عنوان مثال در خط ۲۳ و ۲۷ مقدار `Male` و `Gender` را در بخش متن تگ‌های `Age` و `Gender` نوشته‌ایم. در انتها برای اینکه مطمئن شویم استریم خروجی در فایل نوشته شده است متدهای `flush()` و `close()` را فراخوانی می‌کنیم و سپس استریم را با فراخوانی متدهای `close()` می‌بندیم.

پرس و جوی محتوای XML با XPath

XPath پیشنهاد رسمی W3C برای پرس و جو و جست و جو در محتوای XML است، در حقیقت XPath استانداری است که زبانی را در اختیار ما قرار می‌دهد که با استفاده از آن می‌توانیم المان‌ها و صفات‌ها مختلف یک فایل XML را مورد بررسی قرار دهیم. XPath دارای ویژگی‌ها و امکانات زیر است:

- برای قسمت‌های مختلف یک فایل XML مانند المان‌ها، صفات‌ها، متن‌ها، توضیحات و غیره ساختارهای مناسب را در اختیار ما قرار می‌دهد.
- با استفاده از `Path Expression` این امکان را به ما می‌دهد که گره یا گره‌هایی که دارای شط خاصی هستند را به سادگی انتخاب کنیم.
- کتابخانه استاندار و کاملی را در اختیار ما قرار می‌دهد که از طریق قادر به کار کردن با انواع داده‌ای مختلف همچون اعداد، تاریخ، رشته‌ها و غیره خواهیم بود، به علاوه توابع استانداری برای تغییر و یا مقایسه نیز در اختیار ما قرار می‌دهد.

- از مهمترین استانداردهای XSLT است و یادگیری XSLT نیازمند یادگیری XPath نیز می‌باشد.
- که از ویژگی‌های Path Expression است به ما امکان می‌دهد تا از بین لیست گره‌ها فقط گره‌های خاص را انتخاب کنیم.
- استاندار پیشنهادی W3C است.

Path Expression

عبارت متنی است که به ما اجازه می‌دهد گره یا گره‌ای را از داخل یک محتوای XML استخراج کنیم، جدول زیر عبارات مهم و کاربردی Path Expression را نشان می‌دهد.

عملکرد	عبارة
به کار بردن نام گره باعث می‌شود تا تمام گره‌های با نام مورد نظر انتخاب شوند.	node-name
مشخص می‌کند که انتخاب و جست و جو از ابتدای فایل صورت گیرد.	/
تمام گره‌ها صرف نظر از مکان آن‌ها انتخاب می‌شوند.	//
گره فعلی را انتخاب می‌کند.	.
پدر گره فعلی را انتخاب می‌کند.	..
برای انتخاب بر حسب صفت به کار می‌رود.	@

Predicates

پس از ساخت یک لیست توسط Path Expression با استفاده از Predicate می‌توانیم گره یا گره‌های خاصی را انتخاب می‌کنیم، با عبارت [...] مشخص می‌شود که به جای [...] می‌توانیم از توابع استاندارد، اندیس یا عبارات پیچیده‌تر برای انتخاب گره‌های خاص استفاده کنیم، در ادامه با مثال با این موضوع آشنا می‌شویم:

عملکرد	عبارت
اولین گره student که زیر گره class باشد را انتخاب می‌کند.	/class/student[1]
سومین گره student که زیر گره class باشد را انتخاب می‌کند.	/class/student[3]
آخرین گره student که زیر گره class باشد را انتخاب می‌کند.	/class/student[last()]
گره ماقبل آخر را با شرایط قبلی انتخاب می‌کند.	/class/student[last()-1]
تمام گرههای student که دارای sid برابر با ۳۳۳ هستند را انتخاب می‌کند.	//student[@sid='333']
تمام گرههای student زیر گره class که اندیس آنها از ۳ کمتر است را انتخاب می‌کند (یعنی دو گره ابتدایی انتخاب می‌شوند). در اینجا اندیس از یک شروع می‌شود.	/class/student[position()<3]

XPath استفاده از

خوبی‌خانه در جاوا با استفاده از کتابخانه‌های توکار می‌توانیم به سادگی از امکانات XPath استفاده کنیم، به صورت کلی استفاده از

شامل مراحل زیر است:

۱. کتابخانه‌های مورد نیاز را import کنیم که معمولاً موارد زیر کافی هستند:

```
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import javax.xml.xpath.*;
import java.io.*;
```

۲. برای خواندن از فایل یک DocumentBuilder ایجاد کنیم:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
```

۳. یک Document ایجاد کنیم و محتوای XML را از یک فایل یا استریم بخوانیم

```
Document doc = builder.parse(input);
```

۴. یک شیء از کلاس XPath ایجاد کنیم:

```
XPath xPath = XPathFactory.newInstance().newXPath();
```

.۵ مورد نظر را ساخته و آن را با متدها compile() کامپایل کنیم.

```
XPathExpression expr = xPath.compile(XPATH_EXPRESSION_STRING);
```

.۶ با استفاده از متدها evaluate() نتیجه مورد نظر را محاسبه کنیم.

```
NodeList nodeList = (NodeList) expr.evaluate(doc, XPathConstants.NODESET);
```

در ادامه با استفاده از چند مثال به صورت عملی با موارد فوق آشنا می‌شویم. قبل از ادامه یک فایل XML به نام sample.xml ایجاد می‌کنیم

و محتوای زیر را در آن می‌نویسیم و در درایو D ذخیره می‌کنیم:

```
<?xml version="1.0" encoding="utf-8" ?>
<Persons>
  <Person name="John Smith">
    <Age>30</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Mike Folley">
    <Age>25</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Lisa Carter">
    <Age>22</Age>
    <Gender>Female</Gender>
  </Person>
  <Person name="Jerry Frost">
    <Age>27</Age>
    <Gender>Male</Gender>
  </Person>
  <Person name="Adam Wong">
    <Age>35</Age>
    <Gender>Male</Gender>
  </Person>
</Persons>
```

برای انتخاب گرهای از فایل بالا که خاصیت Name آن برابری John Smith است به صورت زیر عمل می‌کنیم :

```
1 package myfirstprogram;
2
3 import org.w3c.dom.*;
4 import javax.xml.parsers.*;
5 import javax.xml.xpath.*;
6 import java.io.*;
7
8 public class MyFirstProgram
9 {
10     static void printNode(Node n)
```

```

11  {
12      System.out.println("-----");
13      Element e = (Element)n;
14
15      //Parent Node
16      System.out.println(n.getNodeName() + " Name: " + e.getAttribute(" name"));
17
18      //Age
19      String Age = e.getElementsByTagName("Age").item(0).getTextContent();
20      System.out.println("Person Age: " + Age);
21
22      //Gender
23      String Gender = e.getElementsByTagName("Gender").item(0).getTextContent();
24      System.out.println("Person Gender: " + Gender);
25  }
26
27  public static void main(String[] args) throws Exception
28  {
29      String XPATH_EXPRESSION_STRING = "//Person[@name='John Smith']";
30
31      File myXMLFile = new File("D:/sample.xml");
32
33      DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
34      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
35
36      Document doc = dBuilder.parse(myXMLFile);
37
38      XPath xPath = XPathFactory.newInstance().newXPath();
39      XPathExpression expr = xPath.compile(XPATH_EXPRESSION_STRING);
40
41      NodeList list = (NodeList)expr.evaluate(doc, XPathConstants.NODESET);
42
43      int listSize = list.getLength();
44      System.out.println(listSize);
45      for (int i = 0; i<listSize; i++)
46      {
47          Node nodeI = list.item(i);
48          printNode(nodeI);
49      }
50  }
51 }
```

```

1
-----
Person Name: John Smith
Person Age: 30
Person Gender: Male
```

برای انتخاب گرههای از نوع Person که جنسیت (Gender) در آنها زن (Female) باشد، کافی است که خط ۲۹ کد بالا را به صورت

زیر تغییر دهیم:

```
String XPATH_EXPRESSION_STRING = "//Person[Gender='Female']";
```

```
1
```

```
-----  
Person Name: Lisa Carter  
Person Age: 22  
Person Gender: Female
```

برای انتخاب گرههای از نوع Person که سن (Age) آنها بیشتر از ۲۵ سال باشد، کافی است که خط ۲۸ کد بالا را به صورت زیر تغییر

دهیم:

```
String XPATH_EXPRESSION_STRING = "//Person[Age > 25]";
```

```
3  
-----  
Person Name: John Smith  
Person Age: 30  
Person Gender: Male  
-----  
Person Name: Jerry Frost  
Person Age: 27  
Person Gender: Male  
-----  
Person Name: Adam Wong  
Person Age: 35  
Person Gender: Male
```

مطالب جدید جاوا را از سایت W3-farsi.com بخوانید

کار با بانک اطلاعاتی

MySQL چیست؟

بیشتر برنامه‌های امروزی از روش‌های مختلفی برای ذخیره سازی داده‌ها استفاده می‌کنند. یک برنامه می‌تواند دارای انواع مختلفی از منابع داده مانند فایل text و فایل XML و همچنین یک Database از همان‌طوری که معمولاً برای ذخیره انواع داده مانند نام، آدرس، سن، جنس و شغل یک شخص، آهنگ، تصویر و بسیاری چیزهای دیگر استفاده می‌شود. یک دیتابیس مجموعه‌ای است از انواع مختلف داده‌های ساخت یافته در داخل جداولی که شامل فیلدها و رکوردها می‌باشند. بیشتر برنامه‌های امروزی از دیتابیس برای ذخیره اطلاعات استفاده می‌کنند. دیتابیس‌های رابطه‌ای شامل داده‌های هستند که به صورت سازمان یافته با همدیگر در ارتباط هستند. این نوع دیتابیس‌ها شامل یک یا چند جدول به هم پیوسته هستند. جداول شامل سطر و ستون هستند. در دیتابیس‌ها یک سطر نشان دهنده یک رکورد است. به عنوان مثال در یک دیتابیس که شامل رکوردهای کارمند است، یک سطر نشان دهنده یک رکورد از یک کارمند است. ستون نشان دهنده فیلدها یا خواص و صفت می‌باشد. به عنوان مثال یک کارمند دارای یک فیلد مثلاً FirstName (نام) و یک فیلد LastName (نام خانوادگی) و یک فیلد Age (سن) است. می‌توانید بین چندین جدول ارتباط برقرار کنید. به عنوان مثال یک جدول کارمند می‌تواند دارای یک فیلد به نام City_ID باشد. سپس جدول دیگر به نام Cities می‌تواند شامل فیلدهای City_ID و CityName باشد. شما می‌توانید بین این دو جدول ارتباط برقرار کنید. MySQL یکی از استانداردترین راههای برقراری ارتباط با دیتابیس می‌باشد. این زبان دارای دستوراتی است که شما به وسیله آن‌ها می‌توانید داده‌های دیتابیس را بروز کرده، بازیابی، اضافه و حذف کنید. همچنین به شما اجازه ایجاد و تغییر دیتابیس و جداول و ایجاد ارتباط بین جداول مختلف را می‌دهد. DBMS یا Database Management Systems مانند MySQL به شما اجازه دسترسی سریع به داده‌های دیتابیس را می‌دهد و شامل ابزارهای مختلفی برای پرس و جو، ایجاد، حذف و بروز رسانی دیتابیس می‌باشد. بیشتر DBMS‌ها یک محیط گرافیکی جهت انجام امور مختلف برای شما فراهم می‌آورند. مثال‌هایی از DBMS مانند Oracle، Access، SQL SERVER و MySQL توسط شرکت اوراکل توسعه، توزیع، و پشتیبانی می‌شود.

از آنجاییکه در این سری آموزشی می‌خواهیم در مورد نحوه ارتباط با بانک اطلاعاتی MySQL در زبان جاوا و انجام عملیات مختلف از جمله ثبت و حذف و ویرایش و ... بر روی داده‌های آن، بحث کنیم، پس در درس بعد مباحثت مقدماتی این زبان را به شما آموزش می‌دهیم.

مبانی MySQL

قبل از اینکه اقدام به دسترسی و تغییر دیتابیس‌ها کنید لازم است در مورد مبانی MySQL اطلاعاتی داشته باشید. از این زبان برای پرس و جو در دیتابیس استفاده می‌شود. درک MySQL بسیار ساده و آسان است. از MySQL تنها برای پرس و جوی داده استفاده نمی‌شود، بلکه

تقریباً از آن برای انجام هر کاری مثلاً ایجاد، بروزرسانی و حذف دیتابیس و جداول استفاده می‌شود. MySQL همانند سایر زبان‌های برنامه

نویسی دارای انواعی از داده‌های است که به سه نوع متنی، عددی و تاریخی تقسیم می‌شوند و لیست آن‌ها در جدول زیر نمایش داده شده

است :

نوع	توضیح	داده
عددی	داده عددی معمولی که می‌تواند با نشانه یا بدون نشانه باشد (مثبت یا منفی). اگر از نوع نشانه دار باشد محدوده مجاز اعداد بین ۲۱۴۷۴۸۳۶۴۸ - ۲۱۴۷۴۸۳۶۴۸ می‌باشد، اگر بدون نشانه باشد محدوده مجاز اعداد از ۰ تا ۴۲۹۴۹۶۷۲۹۵ می‌باشد. می‌توانید تعداد ارقام اعداد را تا ۱۱ عدد مشخص کنید .	INT
عددی	عدد صحیح خیلی کوچک که می‌تواند نشانه دار یا بدون نشانه (مثبت یا منفی) باشد. اگر نشانه دارد باشد محدوده مجاز اعداد بین ۱۲۷ - ۱۲۸ می‌باشد. اما اگر بدون نشانه باشد محدوده مجاز اعداد بین ۰ تا ۲۵۵ می‌باشد. می‌توانید تعداد ارقام را تا ۴ رقم مشخص کنید .	TINYINT
عددی	عدد صحیح کوچک می‌تواند نشانه دار یا بدون نشانه باشد. اگر نشانه دار باشد محدوده مجاز اعداد بین ۳۲۷۶۸ - ۳۲۷۶۷ می‌باشد. اگر بدون نشانه باشد محدوده مجاز اعداد بین ۰ تا ۶۵۵۳۵ می‌باشد. می‌توانید تعداد ارقام را تا ۵ رقم مشخص کنید .	SMALLINT
عددی	عدد صحیح متوسط که می‌تواند نشانه دار یا بدون نشانه باشد. اگر نشانه دار باشد محدوده مجاز اعداد بین ۸۳۸۸۶۰۸ - ۸۳۸۸۶۰۷ می‌باشد. اگر بدون نشانه باشد محدوده مجاز اعداد بین ۰ تا ۱۶۷۷۷۲۱۵ می‌باشد. می‌توانید تعداد ارقام را تا ۹ رقم مشخص کنید	MEDIUMINT
عددی	عدد صحیح بزرگ می‌تواند نشانه دار یا بدون نشانه باشد. اگر نشانه دار باشد محدوده مجاز اعداد بین ۹۲۲۳۳۷۷۰۳۶۸۵۴۷۷۵۸۰۷ - ۹۲۲۳۳۷۷۰۳۶۸۵۴۷۷۵۸۰۸ می‌باشد. اگر بدون نشانه باشد محدوده مجاز اعداد بین ۰ تا ۹۲۲۳۳۷۷۰۳۶۸۵۴۷۷۵۸۰۷ می‌باشد. تعداد ارقام را می‌توانید تا ۲۰ رقم مشخص کنید .	BIGINT
عددی	عدد اعشاری که نمی‌تواند بدون علامت باشد. می‌توان طول ارقام (M) و تعداد اعشار (D) را تعریف کرد. تعریف طول و تعداد اعشار اجباری نمی‌باشد و در صورت عدم وجود، مقدار پیش‌فرض ۱۰,۲	FLOAT

	اشتفاده می‌شود که ۲ تعداد ارقام اعشار و ۱۰ تعداد کل ارقام می‌باشد (با احتساب ارقام اعشار). دقت اعشار می‌تواند تا ۲۴ عدد برای FLOAT باشد .	
DOUBLE	عدد اعشاری با دقت مضاعف نمی‌تواند بدون علامت باشد. می‌توان طول ارقام (D) و تعداد اعشار (D) را تعریف کرد. تعریف طول و تعداد اعشار اجباری نمی‌باشد و در صورت عدم تعریف مقدار پیش‌فرض ۴،۶ اشتفاده می‌شود که ۴ تعداد ارقام اعشار و ۱۶ تعداد کل ارقام می‌باشد (با احتساب اعداد اعشار). دقت اعشار می‌تواند تا ۵۴ عدد برای DOUBLE باشد. REAL متراffد برای DOUBLE است .	عددی
DECIMAL	عدد اعشاری با ممیز شناور که نمی‌تواند بدون علامت باشد. در اعشار ممیز شناور هر عدد اعشار معادل یک بایت می‌باشد. تعریف طول ارقام (M) و تعداد اعشار (D) می‌باشد. NUMERIC متراffد برای DECIMAL می‌باشد .	عددی
DATE	تاریخ با فرمت DD-MMM-YYYY، که محدوده آن بین ۰۱-۰۱-۹۹۹۹ تا ۳۱-۱۲-۹۹۹۹ می‌باشد. برای مثال ۳۰-۱۲-۱۹۷۳ به صورت December 30th, 1973 ذخیره می‌شود .	تاریخ
DATETIME	ترکیبی از تاریخ و زمان با فرمت HH:MM:SS DD-MM-YYYY می‌باشد، محدوده آن بین ۰۱-۰۱-۱۰۰۰ و ۳۱-۱۲-۹۹۹۹ می‌باشد. برای مثال، ۰۰:۰۰:۰۰ ۲۳:۵۹:۵۹ ۱۲-۱۲-۹۹۹۹ به صورت December 30th, 1973 ۱۵:۳۰:۰۰ ۳۰-۱۲-۱۹۷۳ ذخیره می‌شود .	تاریخ
TIMESTAMP	محدوده زمانی بین نیمه شب ۱، January 1970 و زمانی در ۲۰۳۷ می‌باشد. فرمت آن شبیه فرمت DATETIME می‌باشد بودن خط تیره در میان اعداد. برای مثال، ۱۵:۳۰:۰۰ ۱۲-۱۲-۹۹۹۹ به صورت December 30th, 1973 ذخیره می‌شود .	تاریخ
TIME	زمان را با فرمت HH:MM:SS ذخیره می‌کند .	تاریخ
YEAR	سال را با فرمت ۲ رقم یا ۴ رقم ذخیره می‌کند. اگر طول ۲ تعیین شود، YEAR می‌تواند بین ۱۹۷۰ تا ۲۰۶۹ (۶۹ تا ۷۰) باشد. اگر طول ۴ تعیین شود YEAR می‌تواند بین ۱۹۰۱ تا ۲۱۵۵ باشد. طول پیش‌فرض ۴ می‌باشد .	تاریخ

CHAR	رشته با طول ثابت که طول آن بین ۱ تا ۲۵۵ می‌باشد (مثال: CHAR(5) مقدار خالی سمت راست با فضای خالی پر می‌شود تا زمانی که به حداقل طول تعیین شده برسد. تعریف طول رشته اجباری نیست اما مقدار پیش‌فرض ۱ می‌باشد .	رشته
VARCHAR	رشته با طول منغیر که طول آن می‌تواند بین ۰ تا ۲۵۵ باشد. (مثال: VARCHAR(25). تعریف مقدار طول رشته اجباری می‌باشد .	رشته
BLOB / TEXT	فیلد با حداقل مقدار ۶۵۵۳۵ کاراکتر . BLOB مخفف "Binary Large Object" می‌باشد و برای ذخیره مقادیر بزرگ داده باینری استفاده می‌شود، از قبیل تصویر یا انواع مختلف فایل‌ها. فیلهایی که عنوان TEXT ذخیره شده‌اند نیز مقدار بزرگی از داده را ذخیره می‌کنند. تفاوت آن‌ها در این می‌باشد که مرتب سازی و مقایسه در فیلهای BLOB بین حروف کوچک و بزرگ تفاوت قائل می‌شود اما در فیلهای TEXT تفاوتی قائل نمی‌شود. برای نوع‌های TEXT و BLOB طول تعریف نمی‌شود .	رشته
TINYBLOB / TINYTEXT	یک ستون TEXT یا BLOB با حداقل مقدار ۲۵۵ کاراکتر. برای نوع‌های TINYTEXT و TINYBLOB طول تعریف نمی‌شود .	رشته
MEDIUMBLOB / MEDIUMTEXT	یک ستون TEXT یا BLOB با حداقل مقدار ۱۶۷۷۷۲۱۵ کاراکتر. برای نوع‌های MEDIUMTEXT و MEDIUMBLOB طول تعریف نمی‌شود .	رشته
LONGBLOB / LONGTEXT	یک ستون TEXT یا BLOB با حداقل مقدار ۴۲۹۴۹۶۷۲۹۵ کاراکتر. برای نوع‌های LONGTEXT و LONGBLOB طول تعریف نمی‌شود .	رشته
ENUM	یک نوع شمارشی می‌باشد، که یک اصطلاح عامیانه برای لیست است. هنگامی که یک فیلد ENUM تعریف می‌کنید، شما لیستی از گزینه‌ها ایجاد می‌کنید که یک مورد باید انتخاب شود (یا می‌تواند NULL باشد). برای مثال شما می‌خواهید فیلد دارای مقدار "A" یا "B" یا "C" باشد، فیلد ENUM باید به صورت ('C', 'B', 'A') ENUM() تعریف شود و تنها همان مقادیر تعیین شده (یا NULL) می‌تواند برای آن فیلد جمع آوری شود .	رشته

SET	نوع SET نیز همانند ENUM می‌باشد و تنها تفاوت آن در این می‌باشد که امکان انتخاب چند گزینه از لیست را نیز میسر می‌سازد.	رشته
-----	---	------

حال اکه با انواع داده‌ها آشنا شدید، در ادامه در دستورات پایه‌ای MySQL توضیح می‌دهیم.

دستورات MySQL

MySQL دارای دستوراتی برای کار با بانک اطلاعاتی مانند ایجاد بانک، ایجاد و حذف جداول و کار با داده‌های ذخیره شده در بانک می‌باشد که در جدول زیر به مهم‌ترین آن‌ها اشاره شده است :

دستور	نحوه استفاده	کاربرد
ALTER TABLE	ALTER TABLE table_name ADD column_name datatype or ALTER TABLE table_name DROP COLUMN column_name	برای اضافه کردن، حذف کردن یا تغییر ستون‌ها در جدول موجود استفاده می‌شود.
CREATE DATABASE	CREATE DATABASE database_name	یک دیتابیس جدید ایجاد می‌کند.
CREATE TABLE	CREATE TABLE table_name (column_name1 data_type, column_name2 data_type, column_name3 data_type, ...)	یک جدول جدید ایجاد می‌کند.
DELETE	DELETE FROM table_name WHERE some_column=some_value or DELETE FROM table_name (Note: Deletes the entire table!!) DELETE * FROM table_name (Note: Deletes the entire table!!)	برای حذف یک رکورد یا همه رکوردهای جدول به کار می‌رود.
DROP DATABASE	DROP DATABASE database_name	برای حذف دیتابیس به کار می‌رود.
DROP TABLE	DROP TABLE table_name	برای حذف جدول به کار می‌رود.

INSERT INTO	INSERT INTO table_name VALUES (value1, value2, value3,...) or INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,...)	برای اضافه کردن یک یا چند رکورد به جدول به کار می‌رود.
LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern	به همراه دستور WHERE برای پیدا کردن یک الگوی خاص در یک ستون استفاده می‌شود.
ORDER BY	SELECT column_name(s) FROM table_name ORDER BY column_name [ASC DESC]	برای مرتب کردن نتیجه پرس و جو بر اساس ستون مشخص شده به صورت صعودی یا نزولی به کار می‌رود.
SELECT	SELECT column_name(s) FROM table_name	برای استخراج داده‌های ستون‌های خاصی از جدول به کار می‌رود.
SELECT *	SELECT * FROM table_name	برای استخراج داده‌های یک جدول به کار می‌رود.
UPDATE	UPDATE table_name SET column1=value, column2=value,... WHERE some_column=some_value	برای به روز رسانی رکوردهای موجود در یک جدول استفاده می‌شود.
WHERE	SELECT column_name(s) FROM table_name WHERE column_name operator value	برای استخراج رکوردهایی که در شرط خاصی صدق می‌کنند کاربرد دارد.

MySQL به تنها یی زبان پیشرفته و بزرگی است. بیشتر از این در مورد آن توضیح نمی‌دهیم. برای مشاهده لیست کامل دستورات MySQL

به لینک زیر مراجعه کنید :

http://www.w3schools.com/sql/sql_quickref.asp

حال که با مفاهیم پایه‌ای آن آشنا شدید، در درس بعد با کاربرد این دستورات به صورت عملی آشنا می‌شوید.

نصب سرور MySQL

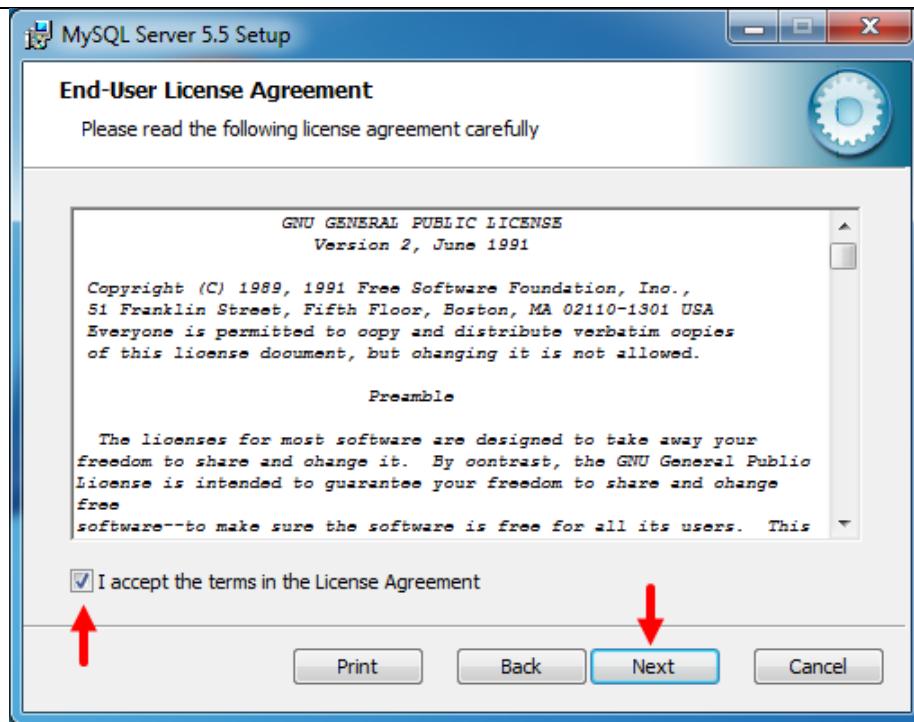
همانطور که در درس قبل اشاره شد، در این سری آموزشی قصد داریم که از MySQL به عنوان منبع اصلی داده‌هایمان استفاده کنیم. پس لازم است که ابتدا سرور آن را نصب کنیم. در این درس از نسخه 5.5 MySQLSERVER استفاده می‌کنیم. دو نسخه ۶۴ و ۳۲ بیت آن را از لینک زیر دانلود کرده و بسته به نوع ویندوزتان یکی از آن‌ها را نصب کنید:

<http://www.d1.w3-farsi.com/Software/Java/MySQL5.5.19.zip>

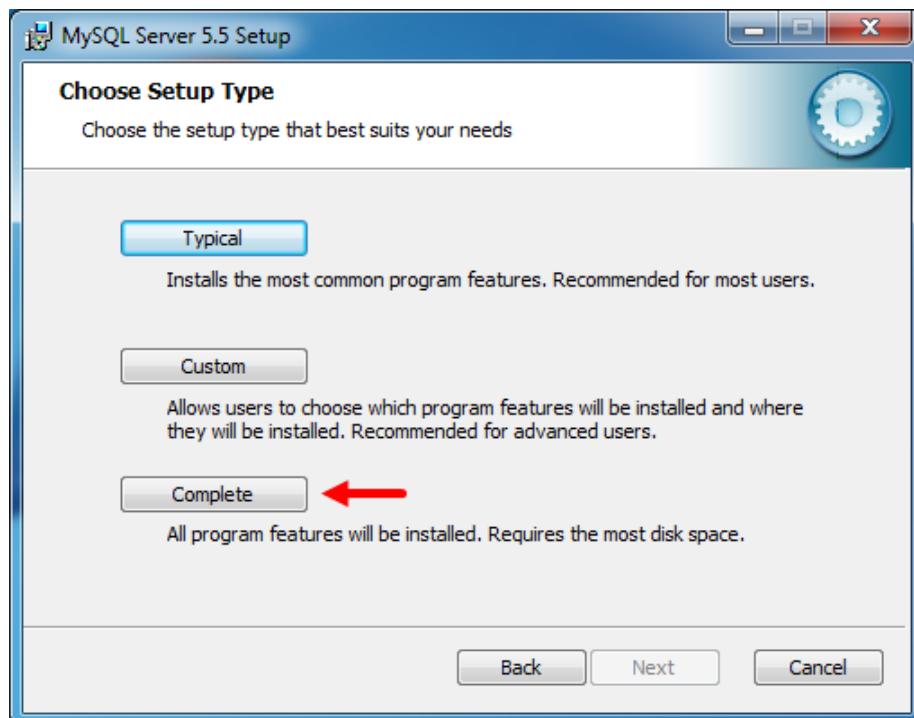
بعد از دانلود فایل بالا، آن را از حالت فشرده خارج کنید. با کلیک بر روی فایل اجرایی یا exe صفحه‌ای به صورت زیر نمایان می‌شود که دکمه Next را می‌زنیم:



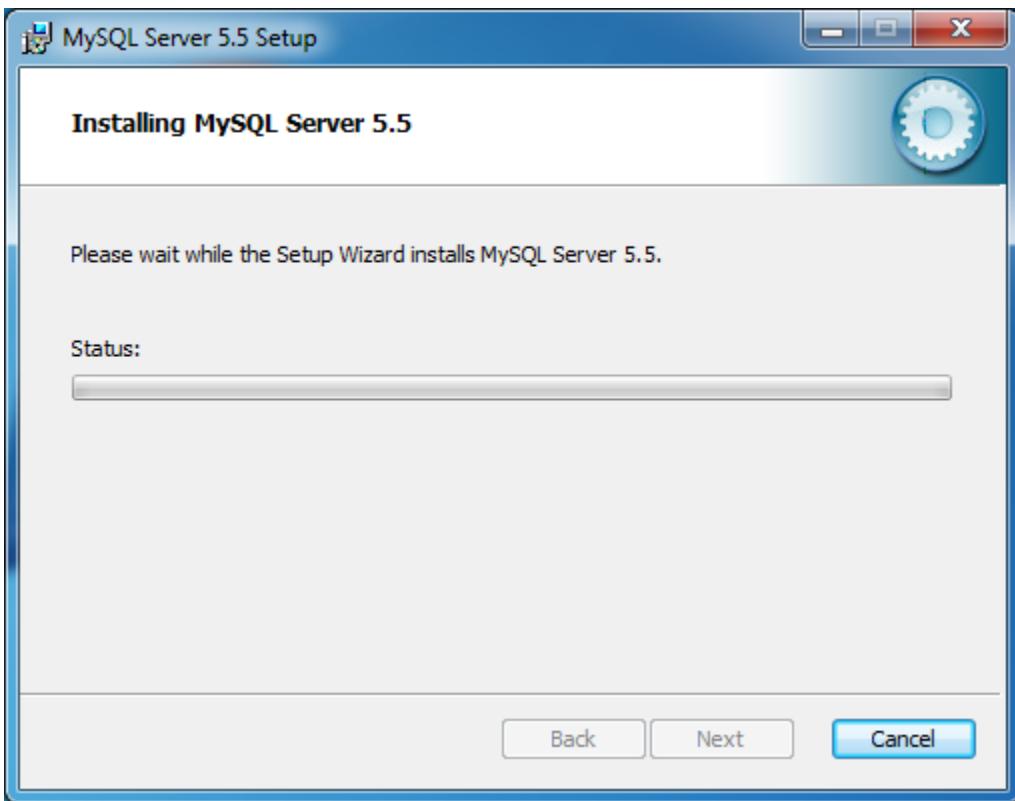
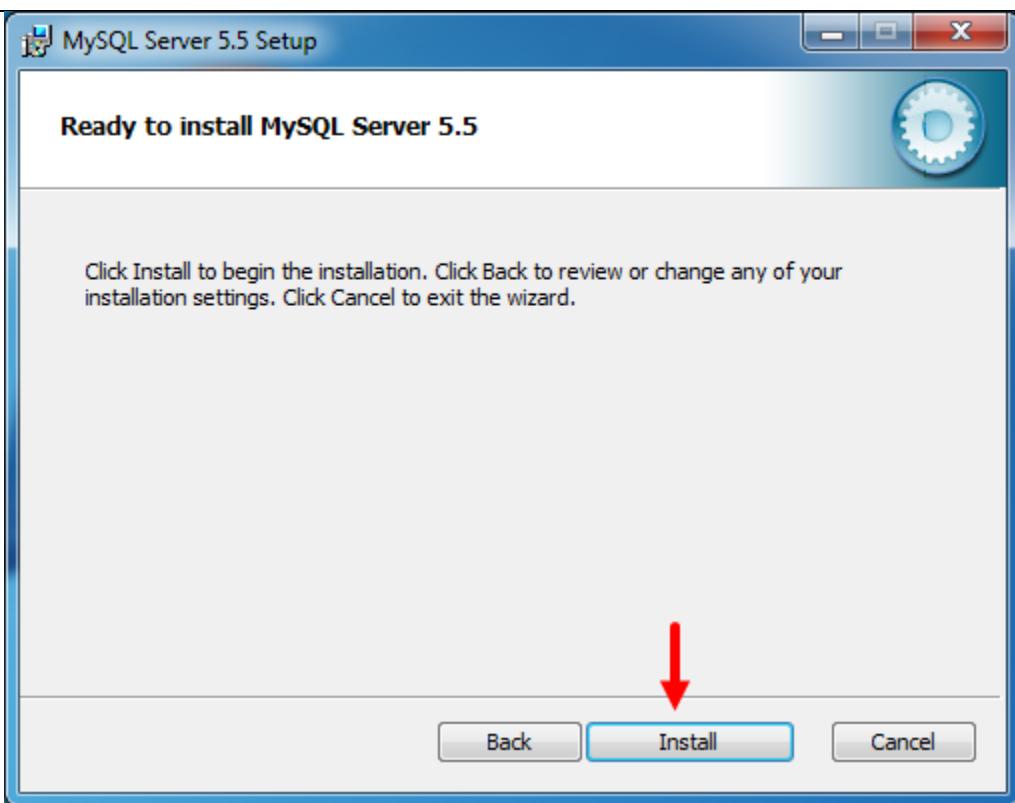
در مرحله بعد گزینه‌ی Accept I را انتخاب می‌کنیم و سپس Next می‌زنیم:



در مرحله بعد گزینه‌ی Complete را انتخاب می‌کنیم :



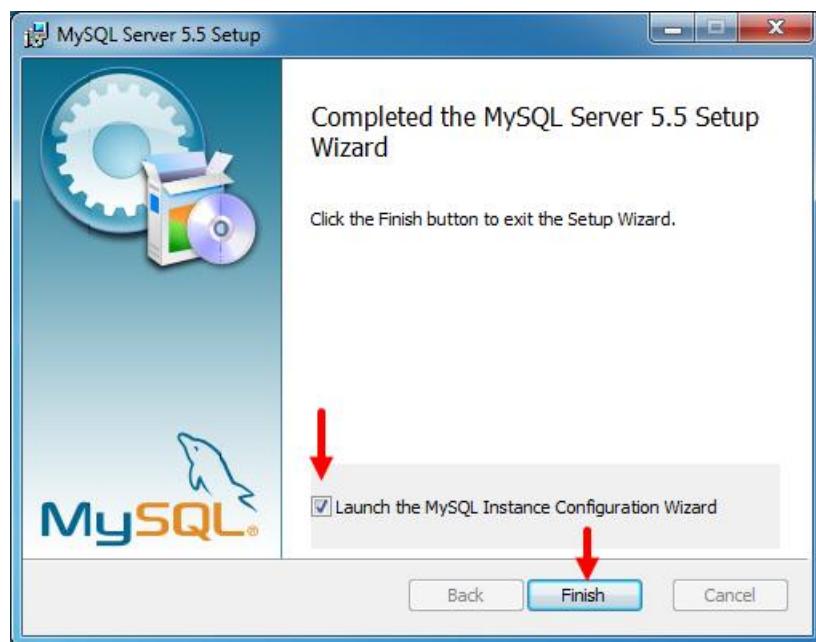
با فشردن دکمه Install مراحل نصب شروع می‌شود و بعد از زدن چند دکمه Next متوالی نصب به پایان می‌رسد :





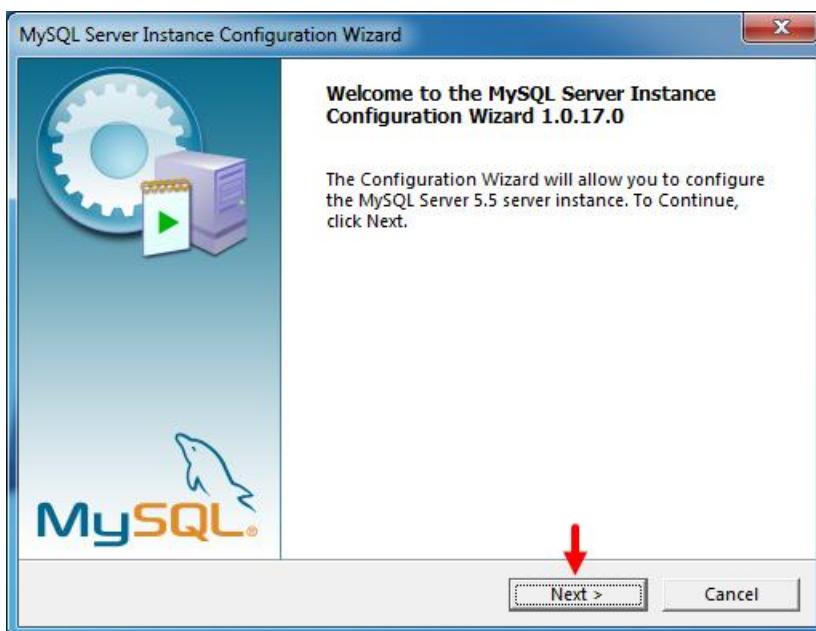
در شکل پایین همانطور که مشاهده می‌کنید گزینه Launch the MySQL Instance Configuration Wizard را تیک زده و سپس

دکمه Finish را می‌زنیم :

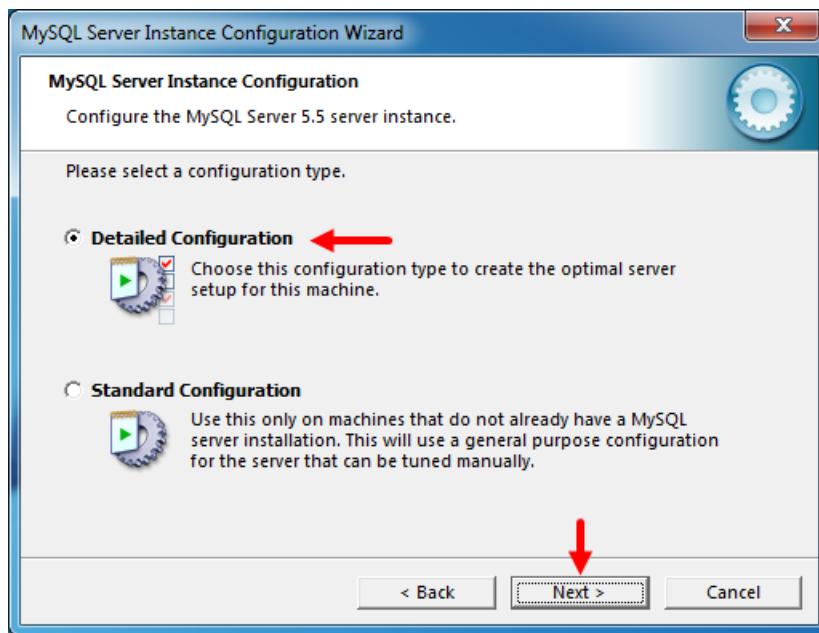


با تیک زدن گزینه Launch the MySQL Instance Configuration Wizard زیر باز می‌شود که با زدن دکمه

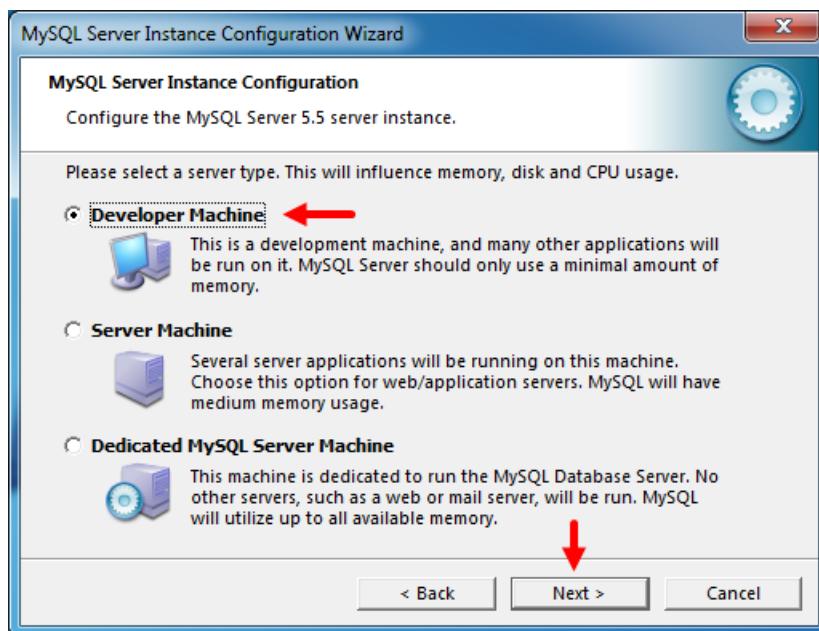
Next ما را وارد مراحل تنظیم زبان، پورت اتصال و ... می‌کند :



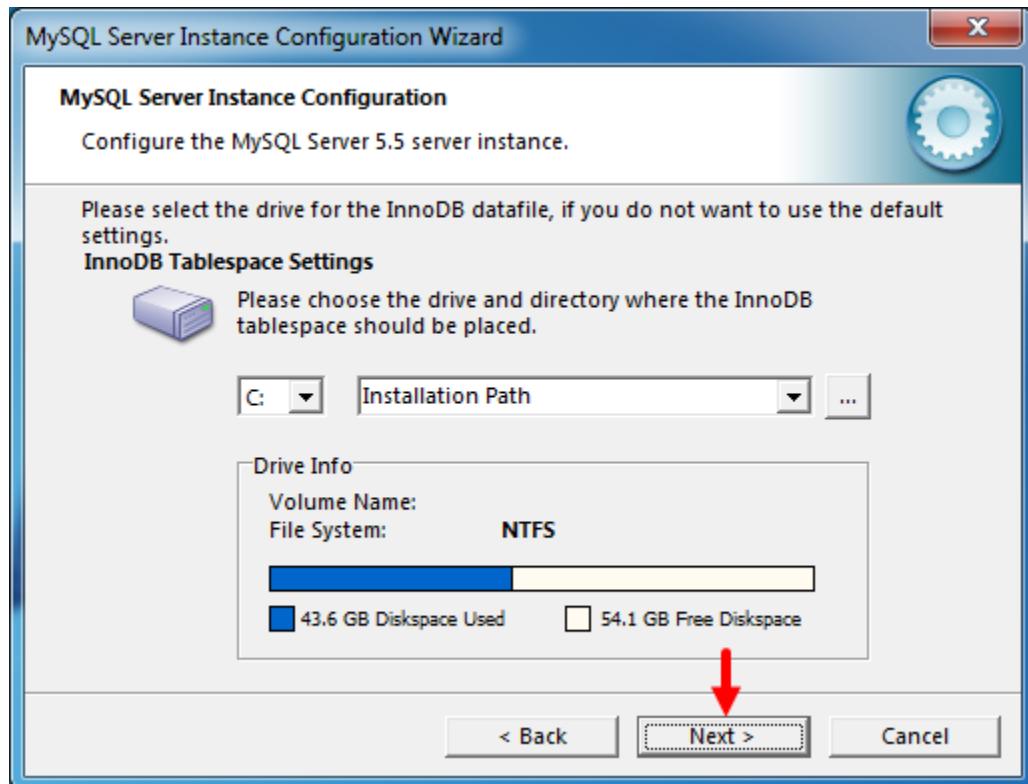
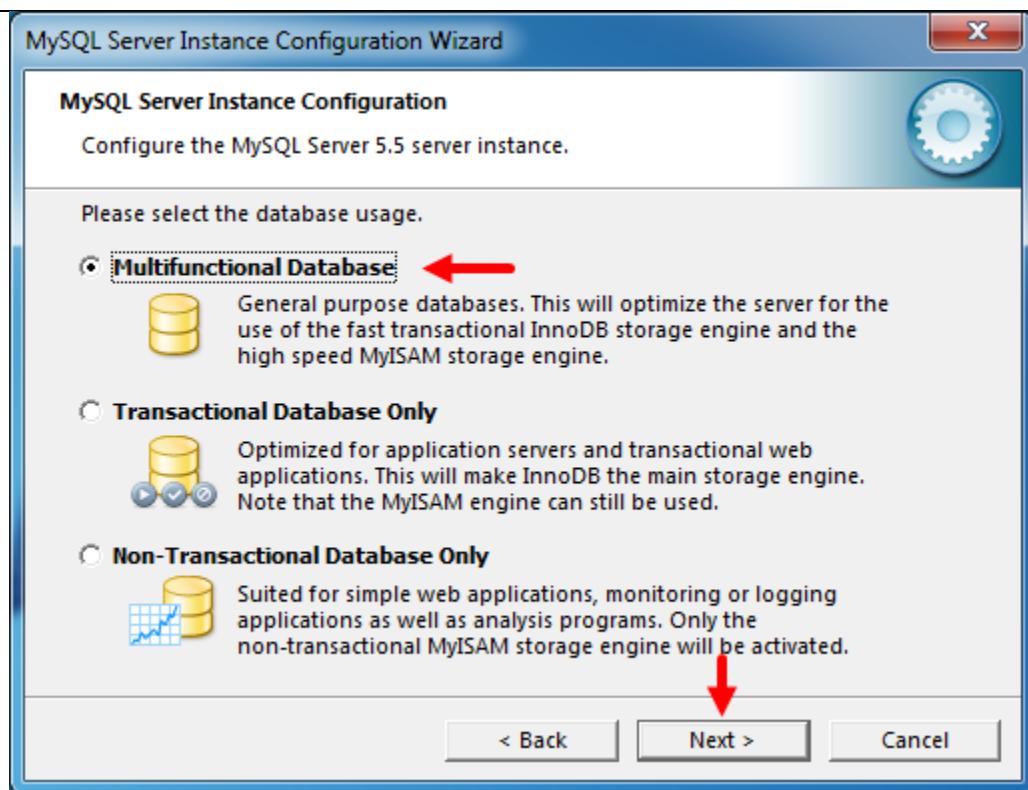
بعد از زدن دکمه Next پنجره زیر ظاهر می‌شود که با انتخاب گزینه اول پیکریندی به صورت استاندارد انجام می‌شود :

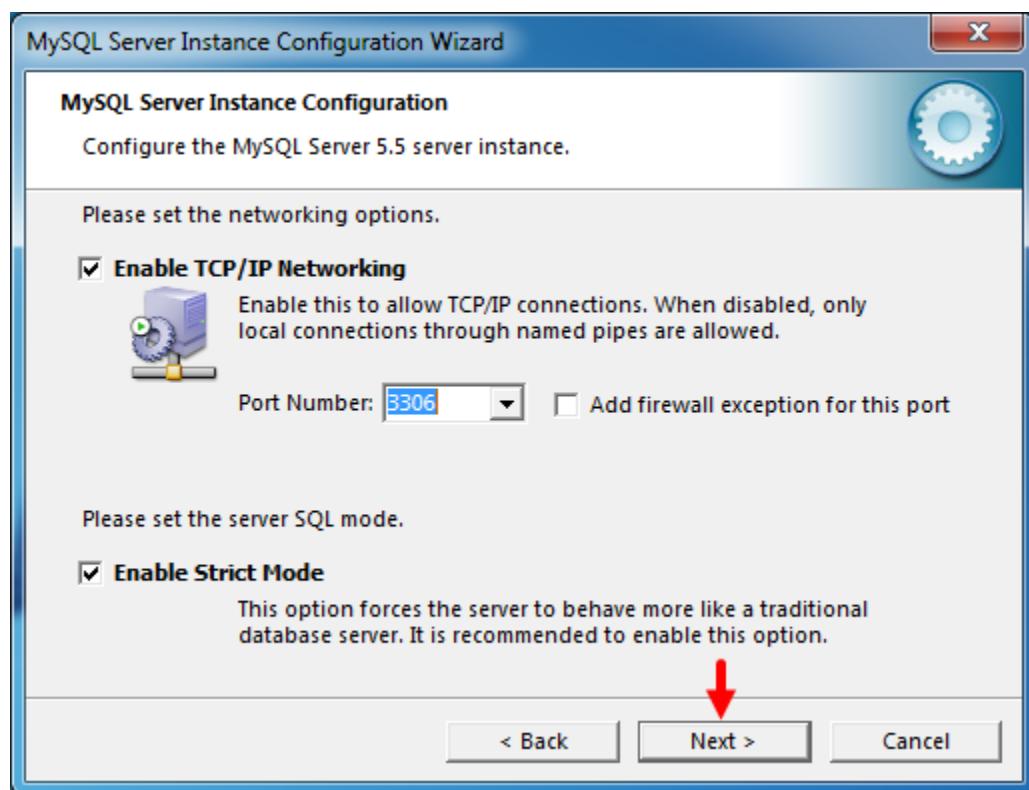
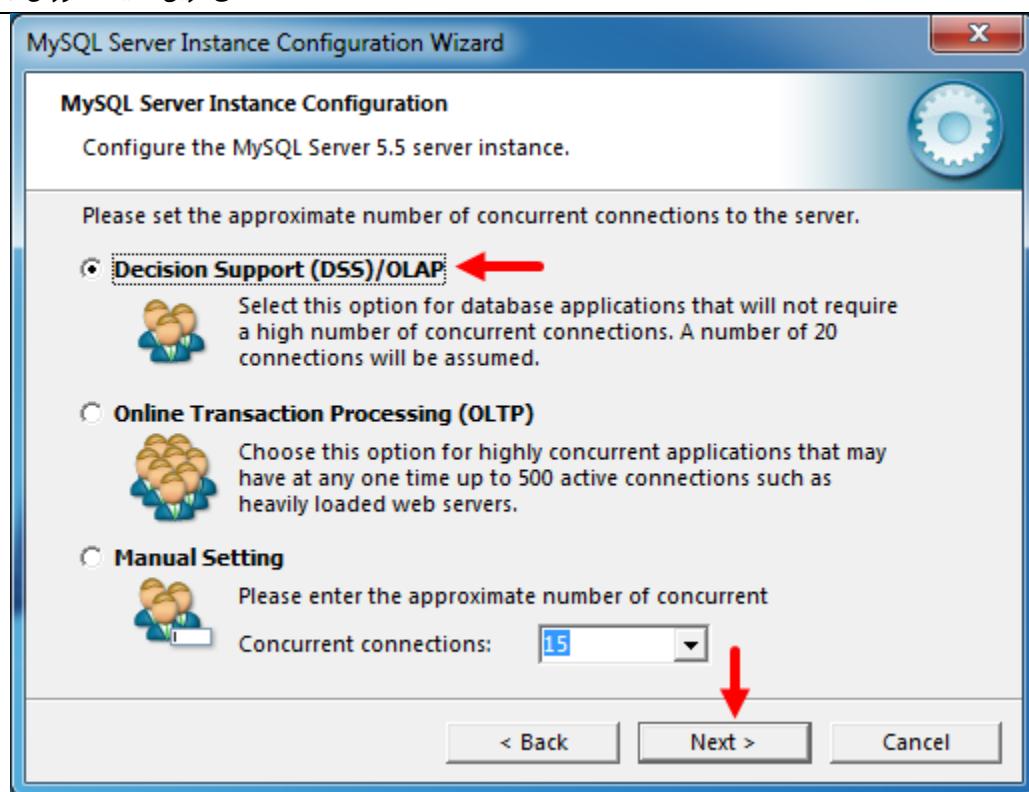


در مرحله بعد گزینه اول را انتخاب می‌کنیم که مقدار حافظه را نسبت به حالت‌های دیگر مصرف می‌کند :



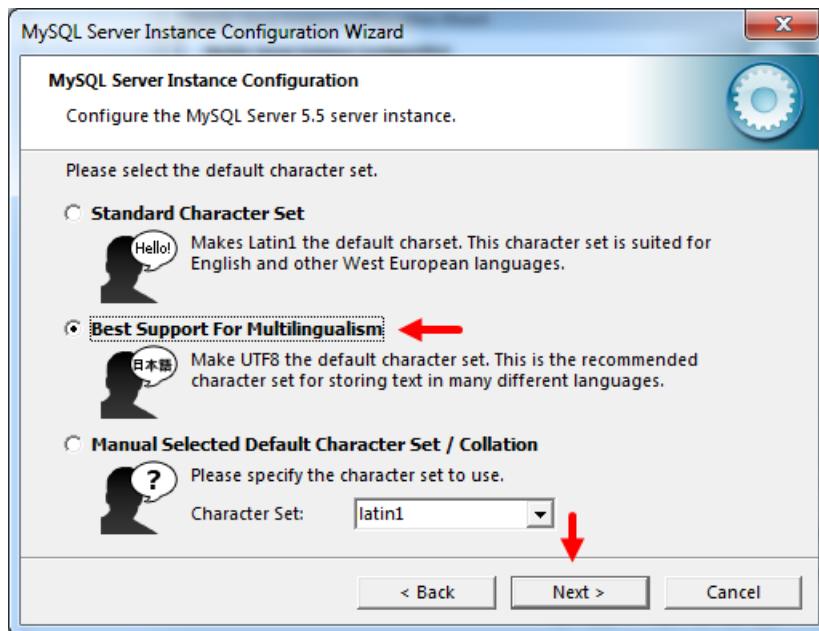
مراحل زیر را به صورت پیشفرض انجام می‌دهیم :



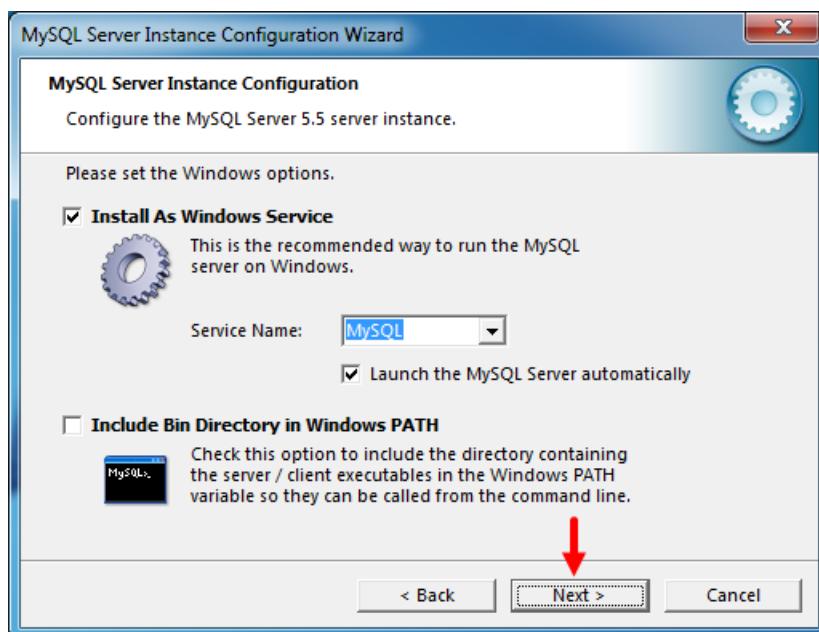


در شکل زیر نوع کاراکتر (character set) را مشخص کنیم که ما گزینه‌ی دوم که UTF8 را پشتیبانی می‌کند را انتخاب می‌کنیم. یعنی

چند زبان از جمله فارسی را پشتیبانی می‌کند و سپس Next را می‌زنیم :



در این مرحله نامی را اختصاص می‌دهیم که زمانی که این سرویس Run می‌شود با این نام در لیست سرویس‌های ویندوز دیده شود که MySQL 5.5 را در نظر گرفتیم و گزینه‌ی زیر آن را تیک می‌زنیم تا زمان بالا آمدن ویندوز به طور اتوماتیک راه اندازی شود:



در مرحله بعد از آنجاییکه نمیخواهیم هنگام اتصال به سرور پسورد داشته باشیم، تیک گزینه Modify Security Settings را بر

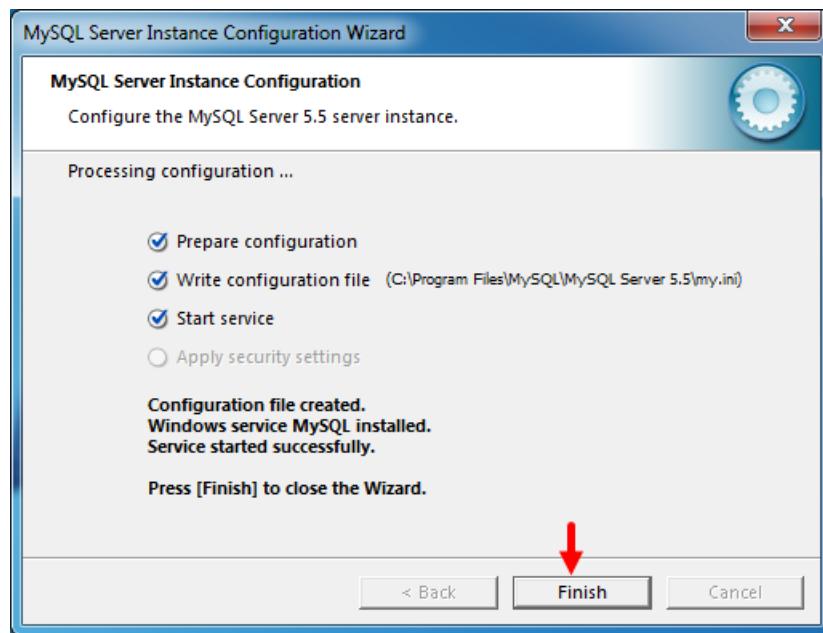
می‌داریم :



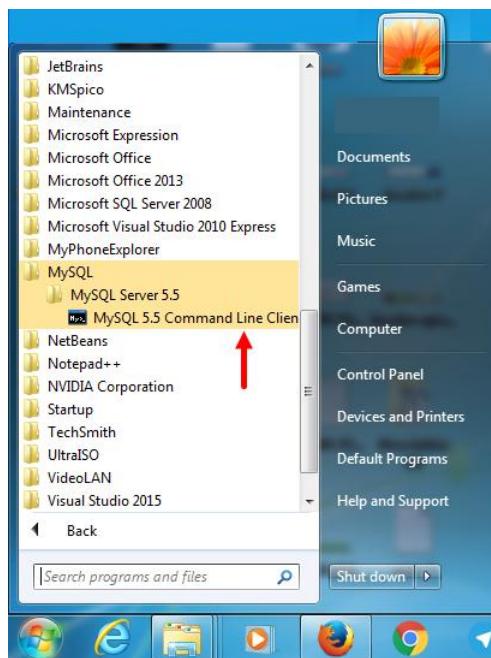
البته این کار اختیاری است و شما ممکن است که دوست داشته باشید هنگام اتصال از شما پسورد درخواست شود، در نتیجه می‌توانید در دو کادر رویروی این گزینه پسورد دلخواه را وارد کرده و سپس در برنامه استفاده کنید (به صورت زیر) :



در مرحله آخر بر روی Execute کلیک کرده تا تنظیمات اعمال شود :

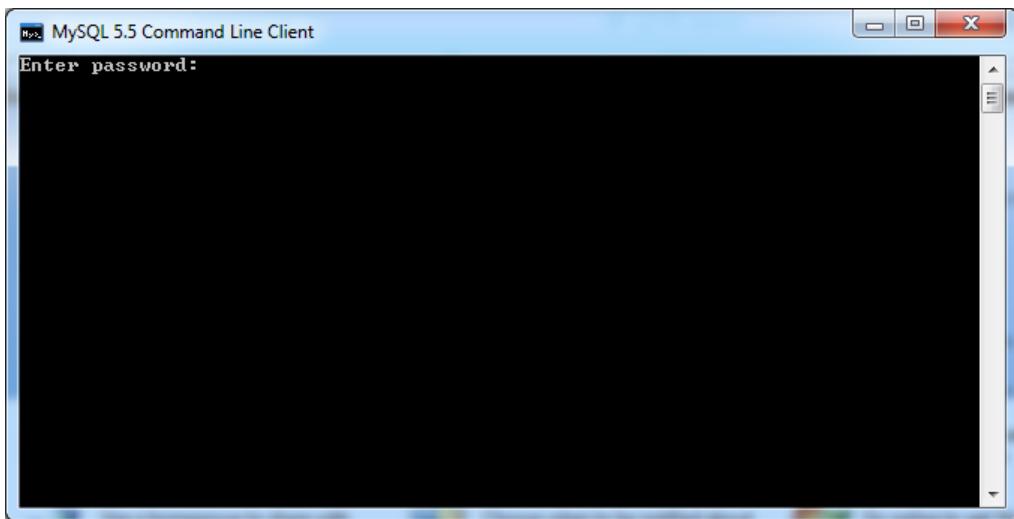


برای اطمینان از صحت نصب MYSQLSERVR به منوی Start ویندوز رفته و در بین برنامه های نصب شده به شکل زیر MySQL را پیدا و بر روی گزینه MySQL 5.5 Command Line Client کلیک کنید :

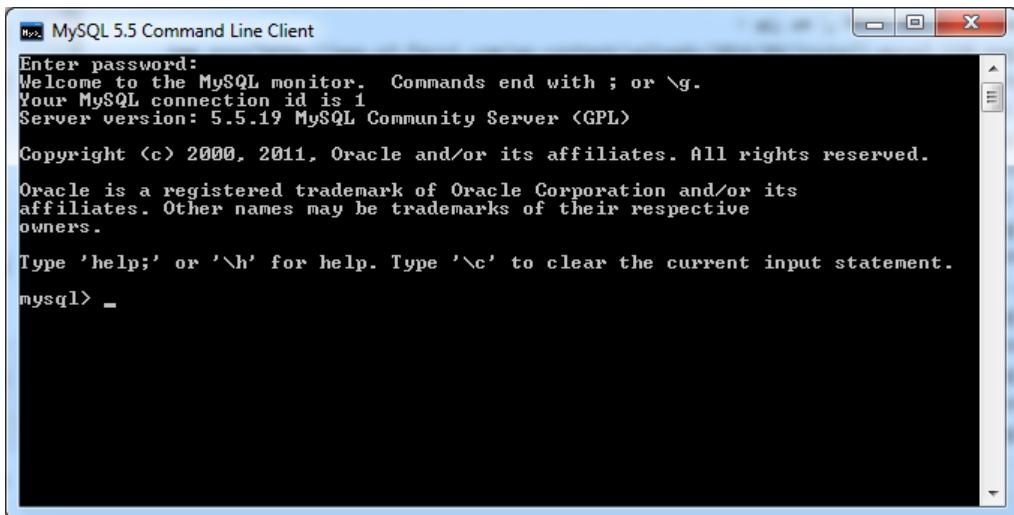


با کلیک بر روی گزینه فوق صفحه‌ای به صورت زیر نمایش داده می‌شود که از شما پسورد می‌خواهد و چون در مراحل قبل ما هیچ پسوردی

برای اتصال به سرور اختصاص ندادیم با زدن دکمه Enter از این قسمت رد می‌شویم :



در مرحله نهایی بعد زدن دکمه Enter، پنجره زیر نمایان می‌شود که نشان از نصب کامل MYSQSERVER است :



نصب نرم افزار MySQL Administrator و آشنایی با محیط آن

برای اجرای دستورات MySQL می‌توان از محیط غیر گرافیکی یا همان محیط کنسول SERVER MySQL استفاده کرد. کار با محیط کنسول و کدنویسی در آن در بسیاری از مواقع کسل کننده و زمان بر می‌باشد. در این درس می‌خواهیم مراحل نصب نرم افزاری را به شما آموزش دهیم که شما را از کدنویسی رها می‌کند. نرم افزار MySQL Administrator دارای محیط گرافیکی برای کار با پایگاه داده، جدول و

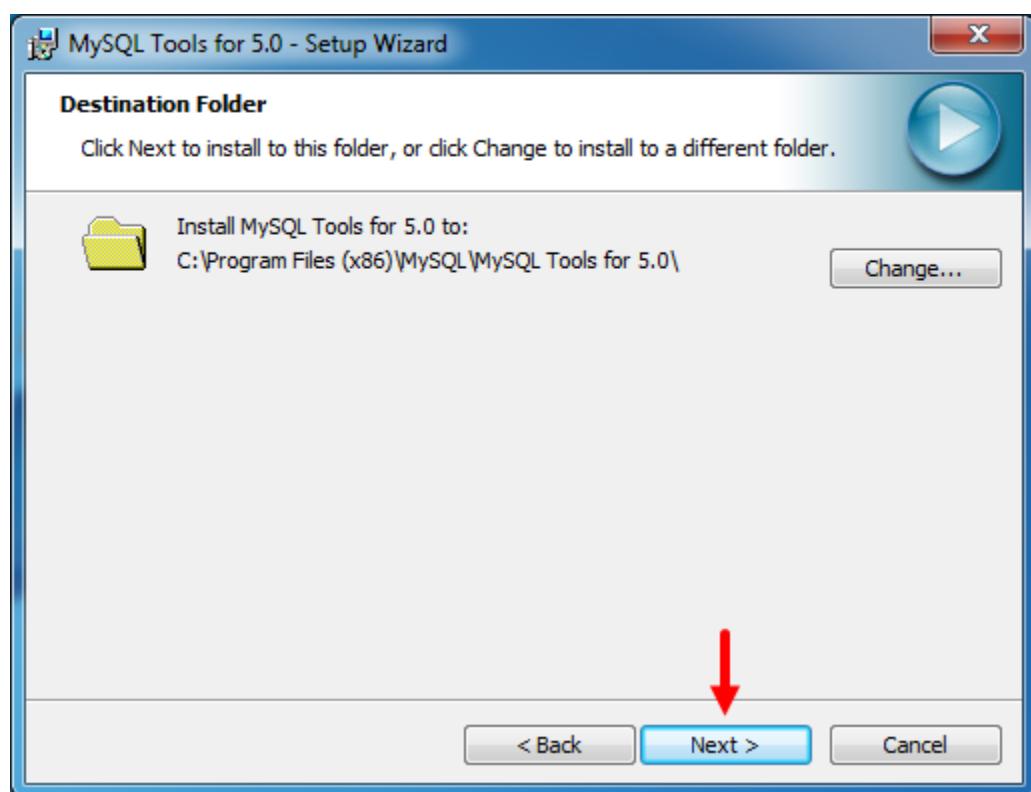
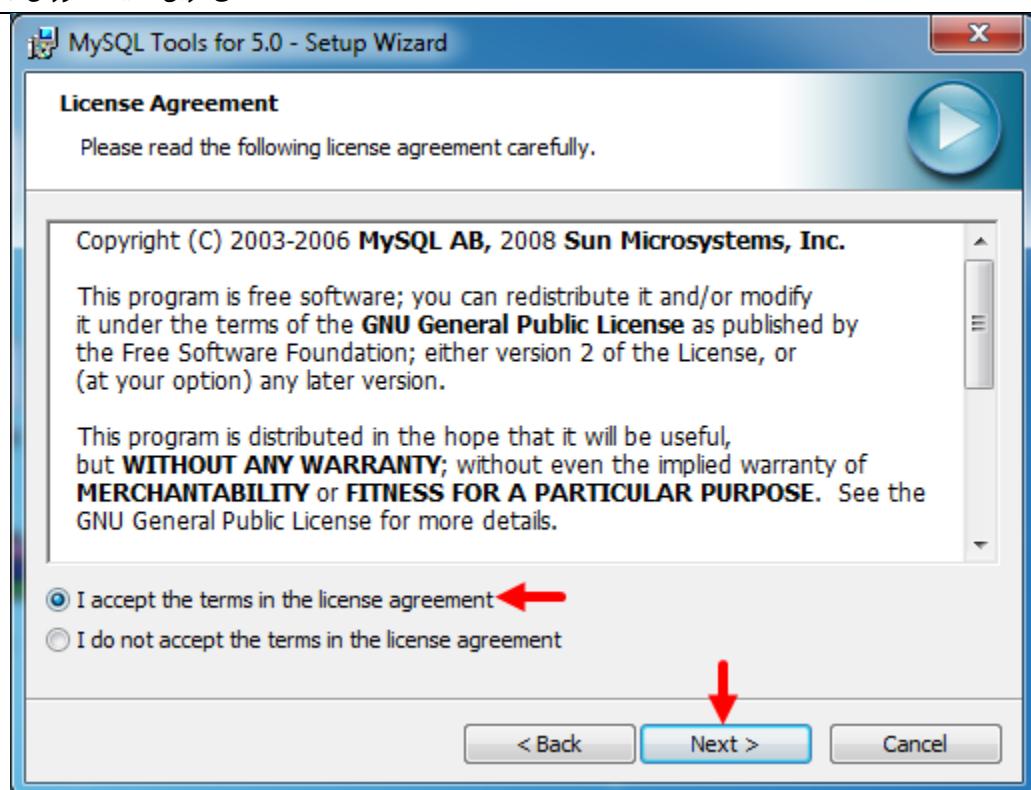
داده‌های آن می‌باشد. با استفاده از این محیط گرافیکی شما تنها با چند کلیک ساده می‌توانید دیتابیس و جدول ایجاد کرده و آن‌ها را

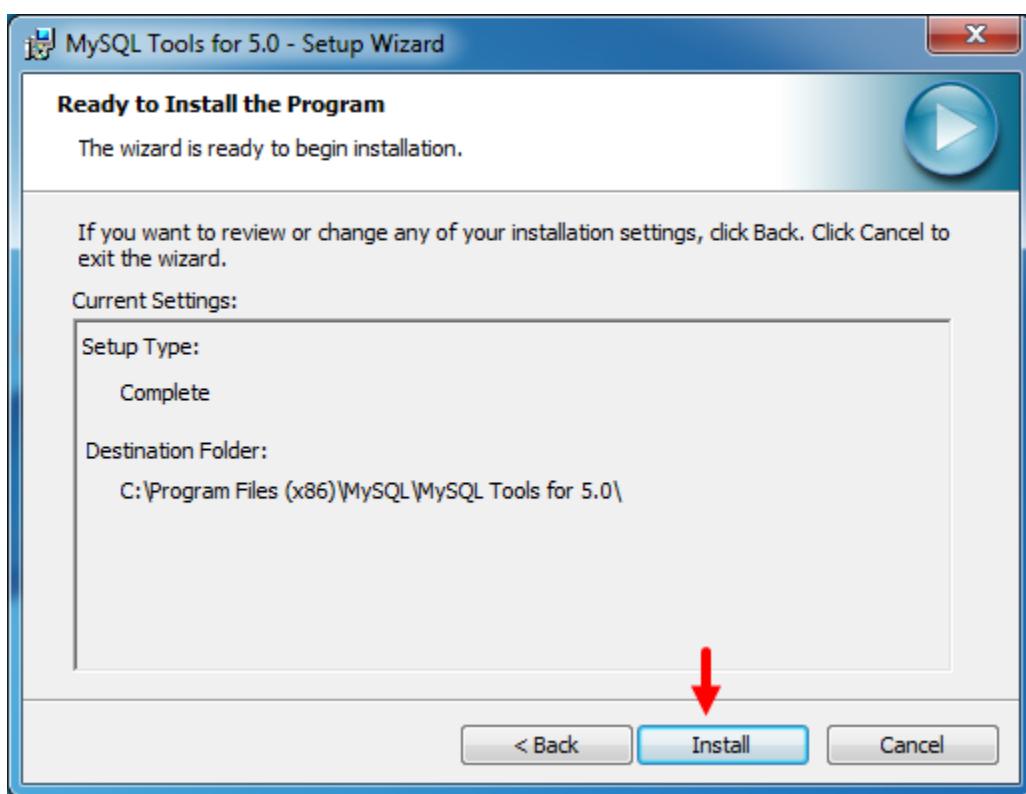
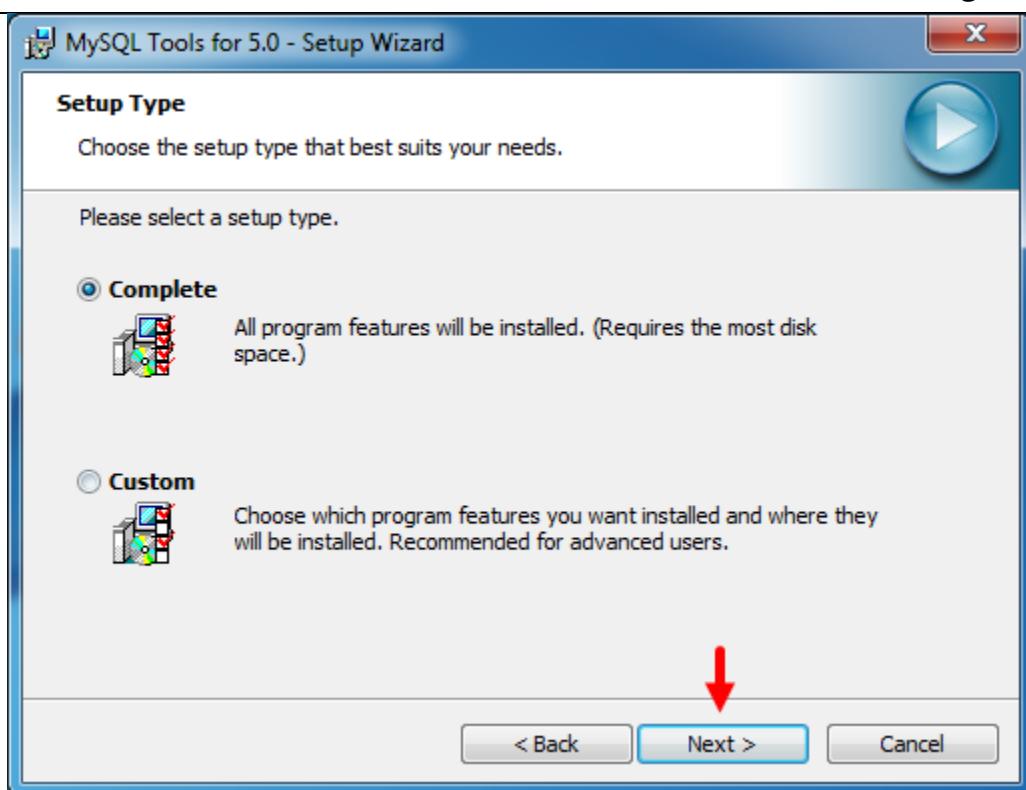
مدیریت کنید. برای دانلود این نرم افزار از طریق لینک زیر اقدام کنید :

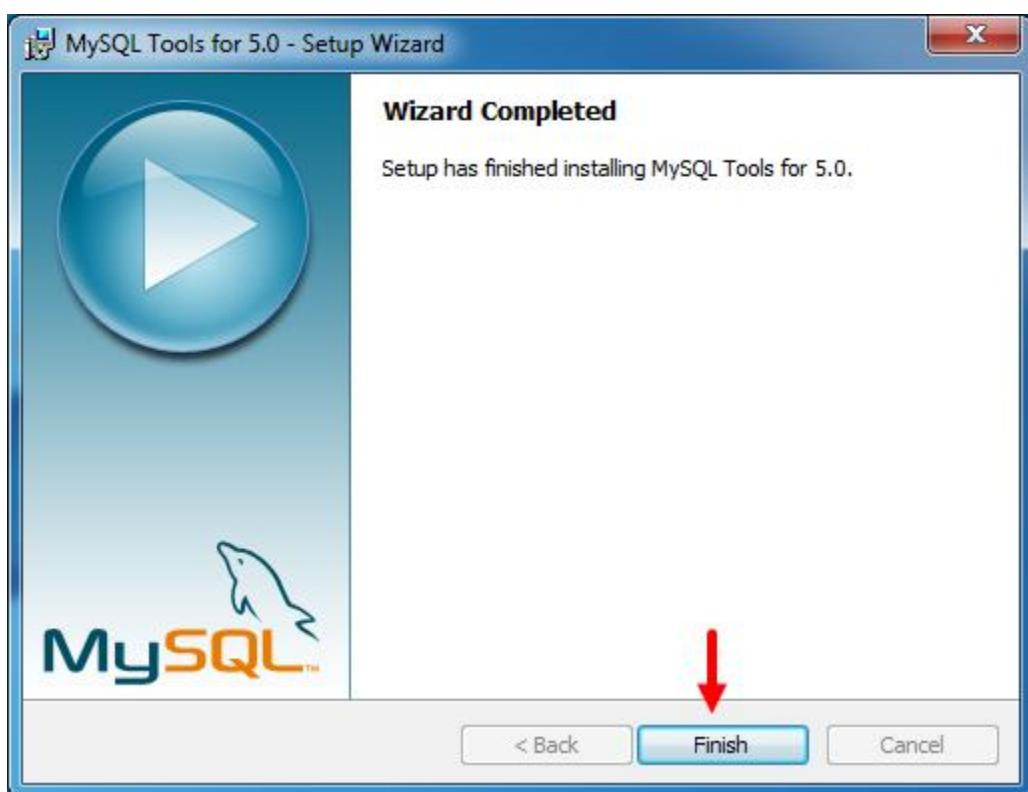
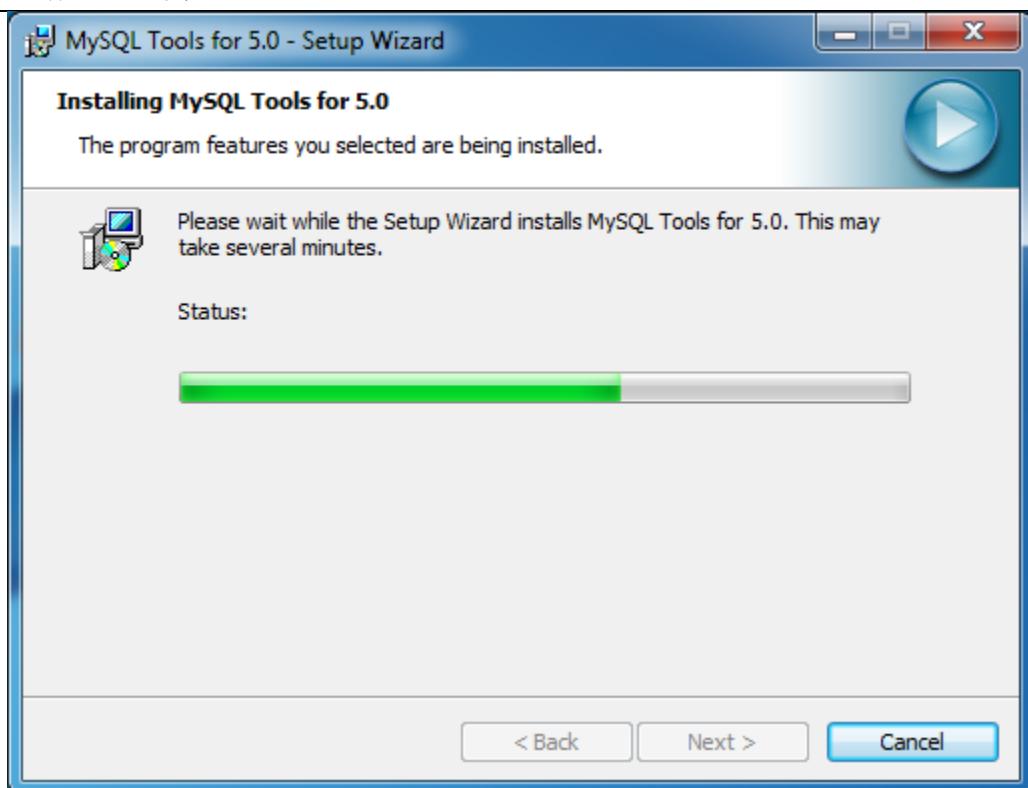
<http://www.dl.w3-farsi.com/Software/Java/MYSQL%20Administrator.msi>

بعد از دانلود بر روی فایل با پسوند .msi دوبار کلیک کرده و تمام مراحل زیر را طی کنید :





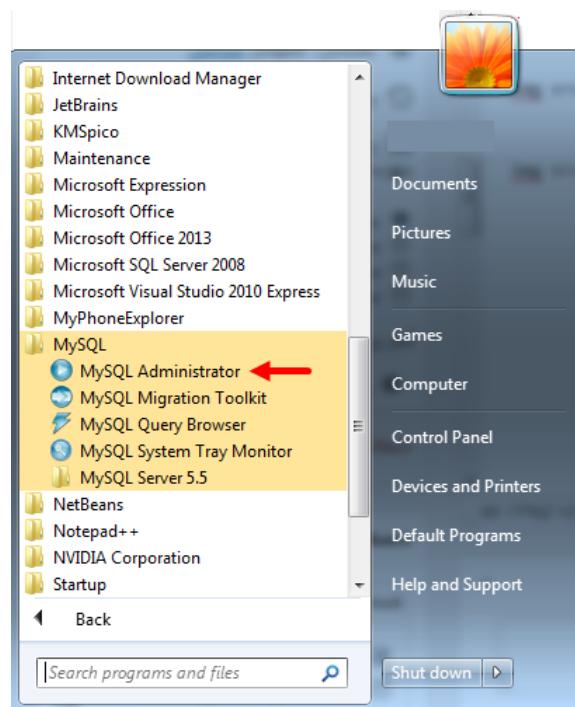




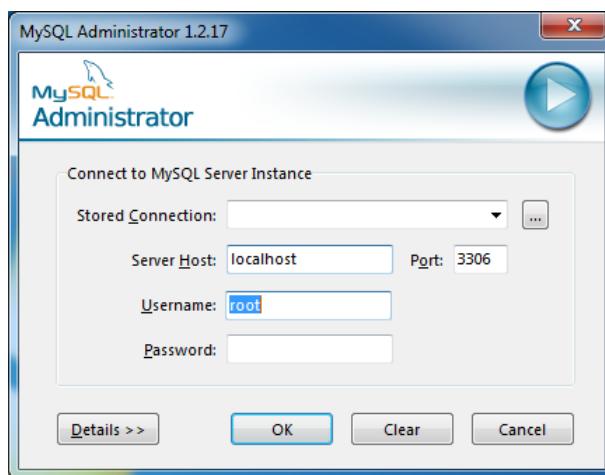
آشنایی با محیط MySQL Administrator

قبل از آشنایی با محیط MySQL اجازه دهید، نحوه اجرا و ورود به آن را توضیح دهیم. ابتدا از منوی Start بر روی

گزینه MySQL Administrator کلیک کرده :

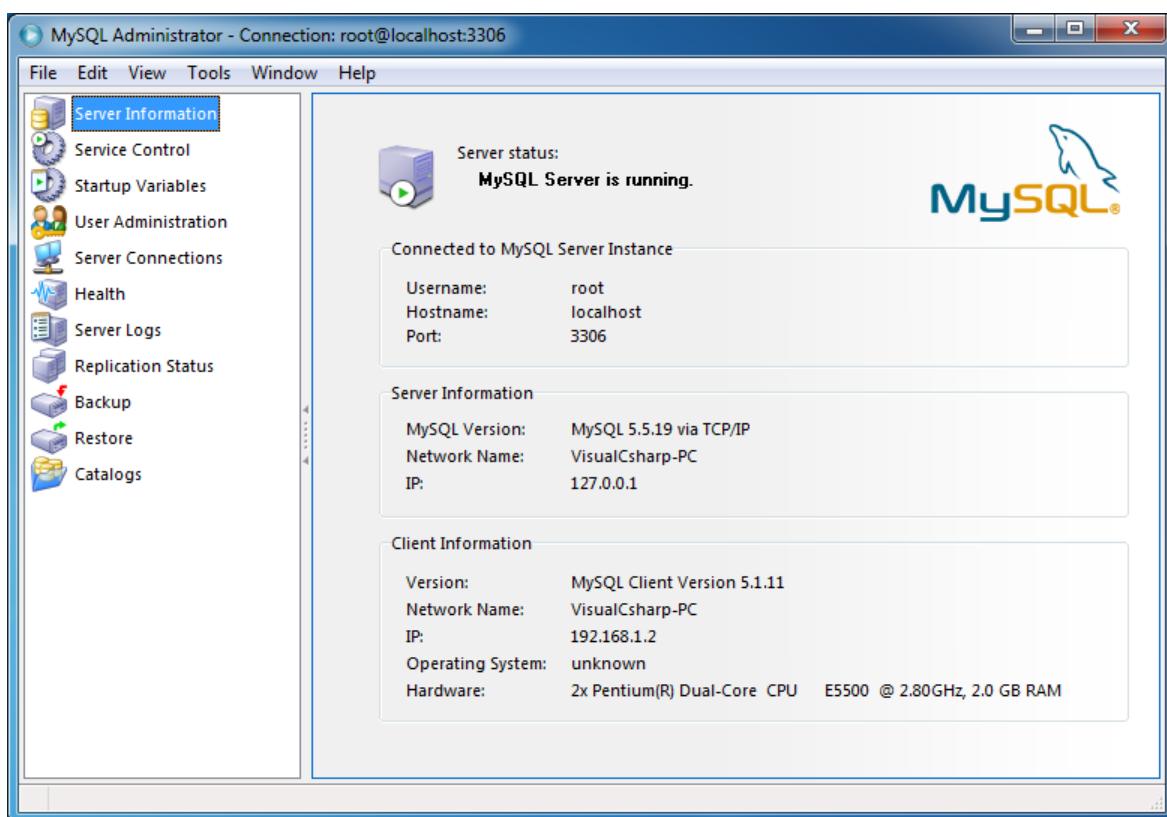


تا صفحه‌ای به صورت زیر ظاهر شود :



در صفحه باز شده در داخل کادر Host Server کلمه‌ی localhost یا 127.0.0.1 و در داخل کادر Username کلمه root را می‌نویسیم.

در کادر پسورد هم هیچ چیزی نمی‌نویسیم. چون اگر به یاد داشته باشید در هنگام نصب MySQL پسوردی برای اتصال به سرور تعريف نکردیم. سپس بر روی دکمه OK کلیک کرده نا وارد محیط برنامه شوید :



توضیح منوهای سمت چپ شکل بالا در جدول زیر آمده است :

منو	کاربرد
Server Information	اطلاعاتی درباره‌ی سرور، port، IP و... را به ما می‌دهد.
Service Control	از طریق این بخش می‌توان سرویس Mysql را start و stop و یا stop کنیم.
Startup Variable	تنظیمات امنیتی، شبکه و ... در این گزینه انجام می‌شود.

در این گزینه می‌توانیم کاربرانی با سطح دسترسی مختلف به دیتابیس ایجاد کنیم. کاربر root دسترسی کامل به دیتابیس دارد.	User Administration
تعداد ارتباطات که به سرور متصل است را می‌توان در اینجا مشاهده کرد.	Server Connection
وضعیتی از عملکرد CPU و Ram و ... به ما نشان می‌دهد.	Health
اتفاقاتی که برای سیستم افتاده، مثلًاً چه زمانی سرویس Start و Stop شده و یا چه دیتابیسی و در چه زمانی ساخته شده و را در یک فایل متنی ذخیره می‌کند.	Server Logs
از طریق چند سرور اطلاعات را بر روی دیتابیس ذخیره می‌کنند.	Replication Status
برای پشتیبان گیری از دیتابیس و بازگردانی آن مورد استفاده قرار می‌گیرند.	Restore و Backup
لیست بانک‌های اطلاعاتی موجود را می‌توانیم از اینجا مشاهده کنیم.	Catalogs

در شکل و جدول بالا ما بیشتر با دو گزینه User Administration و Catalogs برای ایجاد دیتابیس و جدول و کاربر سرکار داریم. که در درس‌های آینده در مورد آن‌ها توضیح می‌دهیم.

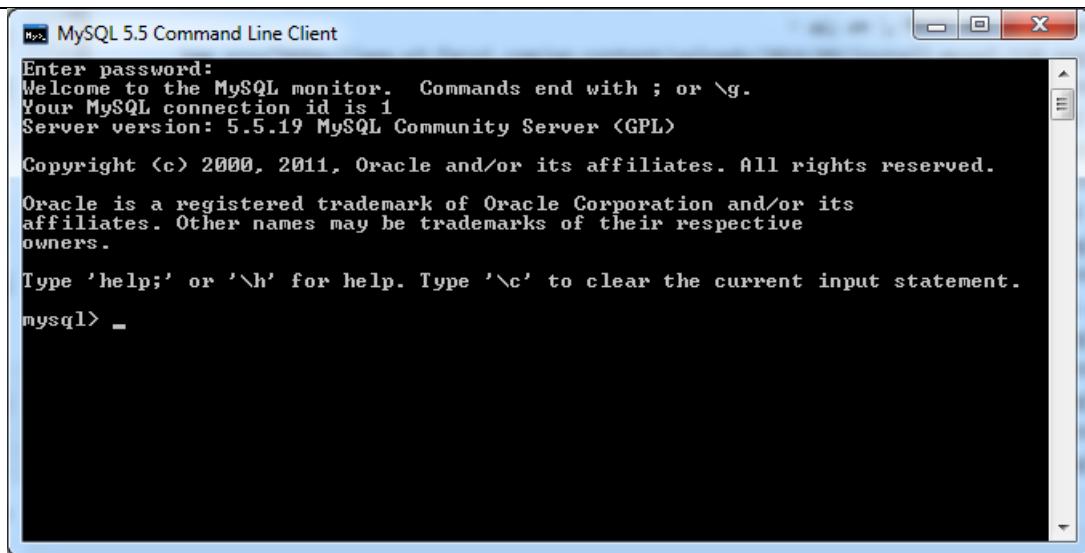
ایجاد جدول و دیتابیس با استفاده از محیط کنسول MySQL

در این درس می‌خواهیم یک دیتابیس که شامل یک جدول و چندین رکورد است را ایجاد کنیم. قصد داریم که ایجاد دیتابیس و جدول را از دو راه شما آموزش دهیم :

- محیط کنسول MySQL

- محیط گرافیکی MySQL Administrator

ابتدا روش ایجاد دیتابیس و جدول و کار با داده‌های دیتابیس در محیط کنسول را توضیح می‌دهیم. برای ورود به محیط کدنویسی به منوی استارت رفته و بر روی گزینه MySQL 5.5 Command Line Client کلیک کنید تا پنجره محیط کنسول به صورت زیر نمایش داده شود :



```
MySQL 5.5 Command Line Client
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.5.19 MySQL Community Server <GPL>

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

ایجاد دیتابیس

در این بخش میخواهیم یک دیتابیس با نام University که شامل جدول Students است را ایجاد کنیم. برای این کار در پنجره بالا

دستور MySQL زیر را مینویسیم :

```
mysql> CREATE DATABASE University;
```

با فشار دادن دکمه Enter، اگر همه چیز به خوبی پیش رود و برنامه پیغام خطأ ندهد کد زیر نمایش داده میشود :

```
mysql> CREATE DATABASE University;
Query OK, 1 row affected (0.01 sec)
```

ایجاد جدول

برای ایجاد جدول student که شامل جزئیاتی در مورد چندین دانش آموز است ابتدا باید از ایجاد دیتابیس مطمئن شوید برای این کار

دستور زیر را تایپ کنید :

```
mysql> USE University;
```

معنای دستور بالا این است که ما میخواهیم تغییراتی در دیتابیسی به نام University بدهیم. همچنین یک پیغام مبنی بر اعمال

تغییرات به شما نشان داده میشود. اجازه دهید که جدول Students را ایجاد کنیم. دستور MySQL زیر را مینویسیم :

```
mysql> CREATE TABLE Students
-> (
->     StudentID int AUTO_INCREMENT primary key,
->     FirstName varchar(50),
->     LastName varchar(50),
->     Gender varchar(10),
->     Age int,
->     Address varchar(50)
-> );
```

جدول ایجاد شده دارای ۶ ستون می‌باشد. ستون اول StudentID است، که آن را به عنوان کلید اصلی و از نوع int تعریف کرده‌ایم. دیگر ستون‌های آن به ترتیب عبارت‌اند از FirstName، LastName، Gender، Age و Address که نوع آن‌ها مشخص شده است. با اجرای دستور فوق جدول ایجاد می‌شود. نکته‌ای که در کد بالا وجود دارد و بهتر است که در همینجا به آن اشاره کنیم مربوط به خط ۳ و کلمات primary key و AUTO_INCREMENT می‌باشد.

- primary key، مقداری است که یک سطر یا رکورد را نشان می‌دهد. این مقدار باید برای هر رکورد، یکتا باشد و در رکورد دیگری عین همین مقدار نباید وجود داشته باشد. یکی از روش‌های ایجاد کلید اصلی استفاده از یک فیلد شمارشی به عنوان کلید اصلی است. به هر رکورد یک فیلد شمارشی اختصاص دهید که با اضافه شدن هر رکورد یک مقدار به آن اضافه شود. بدین معنا که اگر شما یک رکورد به دیتابیس اضافه کردید عدد ۱، به ازا دومین رکورد عدد ۲ و... به فیلد شمارشی اضافه شود.

- AUTO_INCREMENT، فیلدی است که به طور خودکار هنگام اضافه شدن یک رکورد جدید، مقدار می‌گیرد (ممکن‌آور است که آن اضافه می‌شود). به عنوان مثال اگر یک رکورد جدید را اضافه کنیم، به طور خودکار مقدار ۱ به آن اختصاص داده می‌شود، و در هر بار اضافه شدن یک رکورد جدید این مقدار نیز یک واحد اضافه می‌شود. می‌توان یک کلید اصلی را به عنوان فیلد شناسه تعریف کرد.

اضافه کردن رکوردها

حال که جدولمان را ایجاد کرده‌ایم اجازه دهید اطلاعات مربوط به چند دانش آموز را به آن اضافه کنیم. در زیر اطلاعات مربوط به چند

دانش آموز که به جدول اضافه کرده‌ایم نشان داده شده است :

StudentID	FirstName	LastName	Gender	Age	Address
1	Edward	Lyons	Male	17	Spencer Street
2	Jimmie	Vargas	Male	18	Blue Bay Avenue
3	Monica	Ward	Female	16	Maple Street

4	Joann	Jordan	Female	17	Spencer Street
5	Cheryl	Swanson	Female	17	Wacky Street
6	Clara	Webb	Female	18	Spooner Street
7	Zack	Norris	Male	19	Blue Bay Avenue
8	Randall	May	Male	18	Golden Street
9	Jessica	Cole	Female	17	Maple Street
10	Oscar	Manning	Male	18	Maple Street

حال که ۱۰ رکورد داریم اجازه دهید آنها را با استفاده از دستورات SQL به جدول Students اضافه کنیم :

```
mysql> INSERT INTO Students(FirstName, LastName, Gender, Age, Address) VALUES
-> ('Edward' , 'Lyons' , 'Male' , 17, 'Spencer Street'),
-> ('Jimmie' , 'Vargas' , 'Male' , 18, 'Blue Bay Avenue'),
-> ('Monica' , 'Ward' , 'Female', 16, 'Maple Street'),
-> ('Joann' , 'Jordan' , 'Female', 17, 'Spencer Street'),
-> ('Cheryl' , 'Swanson', 'Female', 17, 'Wacky Street'),
-> ('Clara' , 'Webb' , 'Female', 18, 'Spooner Street'),
-> ('Zack' , 'Norris' , 'Male' , 19, 'Blue Bay Avenue'),
-> ('Randall', 'May' , 'Male' , 18, 'Golden Street'),
-> ('Jessica', 'Cole' , 'Female', 17, 'Maple Street'),
-> ('Oscar' , 'Manning', 'Male' , 18, 'Maple Street');
```

از یک نسخه اصلاح شده دستور INSERT INTO که به ما اجازه وارد کردن چندین رکورد را به صورت یکجا می‌دهد، استفاده می‌کنیم. هر رکورد در داخل پرانتز قرار دارد و رکوردها به وسیله کاما از هم جدا شده‌اند. بعد از اجرای دستورات MySQL، جدول Students باید دارای ۱۰ رکورد باشد. اگر همه چیز به درستی انجام شود، یک پیغام نشان داده می‌شود که نشان دهنده تعداد رکوردهایی است که دستکاری

(اضافه) شده‌اند. همه کدهایی که تا الان برای ایجاد دیتابیس University نوشته‌ایم به صورت زیر است :

```
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.19 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE University;
Query OK, 1 row affected (0.01 sec)

mysql> USE University;
Database changed
mysql> CREATE TABLE Students
-> (
->     StudentID int AUTO_INCREMENT primary key,
->     FirstName varchar(50),
->     LastName varchar(50),
->     Gender varchar(10),
->     Age int,
->     Address varchar(50)
-> );
Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO Students(FirstName, LastName, Gender, Age, Address) VALUES
->     ('Edward', 'Lyons', 'Male', 17, 'Spencer Street'),
->     ('Jimmie', 'Vargas', 'Male', 18, 'Blue Bay Avenue'),
->     ('Monica', 'Ward', 'Female', 16, 'Mapple Street'),
->     ('Joann', 'Jordan', 'Female', 17, 'Spencer Street'),
->     ('Cheryl', 'Swanson', 'Female', 17, 'Wacky Street'),
->     ('Clara', 'Webb', 'Female', 18, 'Spooner Street'),
->     ('Zack', 'Norris', 'Male', 19, 'Blue Bay Avenue'),
->     ('Randall', 'May', 'Male', 18, 'Golden Street'),
->     ('Jessica', 'Cole', 'Female', 17, 'Mapple Street'),
->     ('Oscar', 'Manning', 'Male', 18, 'Mapple Street');
Query OK, 10 rows affected (0.02 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

حال می‌خواهیم با توجه به مطالبی که در درس قبل یاد گرفتیم، عملیات مختلفی بر روی بانک انجام دهیم. دستور INSERT را که در بالا

برای وارد کردن اطلاعات انجام داده‌ایم. حال برای نمایش اطلاعات می‌خواهیم از دستور SELECT استفاده کنیم. در ادامه کدهای بالا دستور

زیر را بنویسید :

```
mysql> SELECT* FROM Students;
```

کد بالا تمامی داده‌های جدول Students را استخراج کرده و در خروجی نمایش می‌دهد :

StudentID	FirstName	LastName	Gender	Age	Address
1	Edward	Lyons	Male	17	Spencer Street
2	Jimmie	Vargas	Male	18	Blue Bay Avenue
3	Monica	Ward	Female	16	Mapple Street
4	Joann	Jordan	Female	17	Spencer Street
5	Cheryl	Swanson	Female	17	Wacky Street
6	Clara	Webb	Female	18	Spooner Street

```

7           Zack          Norris        Male      19     Blue Bay Avenue
8           Randall       May           Male      18     Golden Street
9           Jessica       Cole          Female    17     Maple Street
10          Oscar         Manning      Male      18     Maple Street
(10 rows affected)
1>

```

اگر بخواهید اطلاعات یک یا چند ستون خاص را نمایش دهید کافیست از کد زیر استفاده کنید :

```
mysql> SELECT FirstName, LastName FROM Students;
```

کد بالا فقط اطلاعات مربوط به ستون‌های FirstName و LastName را نمایش می‌دهد :

FirstName	LastName
Edward	Lyons
Jimmie	Vargas
Monica	Ward
Joann	Jordan
Cheryl	Swanson
Clara	Webb
Zack	Norris
Randall	May
Jessica	Cole
Oscar	Manning

حال فرض کنید که می‌خواهیم نام خانوادگی و سن شخصی که اسمش Edward است را از جدول استخراج کنیم. برای این کار از دستور

WHERE به صورت زیر استفاده می‌کنیم :

```
mysql> SELECT LastName, Age FROM Students WHERE FirstName = 'Edward';
```

خروجی کد بالا به صورت زیر است :

LastName	Age
Lyons	17

حال می‌خواهیم اشخاصی که آن‌ها کمتر از ۵ StudentID را استخراج کرده و به صورت نزولی مرتب کنیم. برای این کار به صورت

زیر عمل می‌کنیم :

```
mysql> SELECT StudentID, FirstName, LastName FROM Students WHERE StudentID < 5 ORDER BY studentID DESC;
```

کد بالا ۴ نفر اول را استخراج کرده و بر اساس ستون StudentID از بزرگ به کوچک مرتب می‌کند. خروجی کد بالا به صورت زیر است :

StudentID	FirstName	LastName
4	Joann	Jordan
3	Monica	Ward
2	Jimmie	Vargas
1	Edward	Lyons

فرض کنید می‌خواهید تمامی نامهایی که در داخل آن‌ها ar وجود دارد را استخراج کنید. در این صورت از عبارت LIKE به صورت زیر

استفاده کنید :

```
mysql> SELECT FirstName FROM Students WHERE FirstName LIKE '%ar%';
```

کد بالا تمامی نامهایی که در آن‌ها دو حرف a و r پشت سر هم آمده‌اند را استخراجی می‌کند :

FirstName
Edward
Clara
Oscar

اگر بخواهیم مثلاً سن شخصی به نام Edward را از ۱۷ به ۲۰ تغییر دهیم می‌توانیم از دستور UPDATE استفاده کنیم :

```
mysql> UPDATE Students SET Age = 20 WHERE StudentID = 1;
```

در دستور بالا گفته‌ایم که سن شخصی از جدول Students را که آن برابر ۱ است را به ۲۰ تغییر بده. حال اگر با استفاده از

یک دستور SELECT همه داده‌ها را نمایش دهیم مشاهده می‌کنید که سن Edward به ۲۰ تغییر کرده است. همانطور که مشاهده می‌کنید

استفاده از دستورات MySQL بسیار آسان است. شما می‌توانید بقیه دستورات را که در درس قبل آموختید را به صورت بالا امتحان کنید.

ایجاد جدول و دیتابیس با استفاده از محیط MySQL Administrator

در درس قبل با نحوه ایجاد دیتابیس و جدول و دستکاری اطلاعات با استفاده از محیط کدنویسی آشنا شدید. در این درس دقیقاً می‌خواهیم

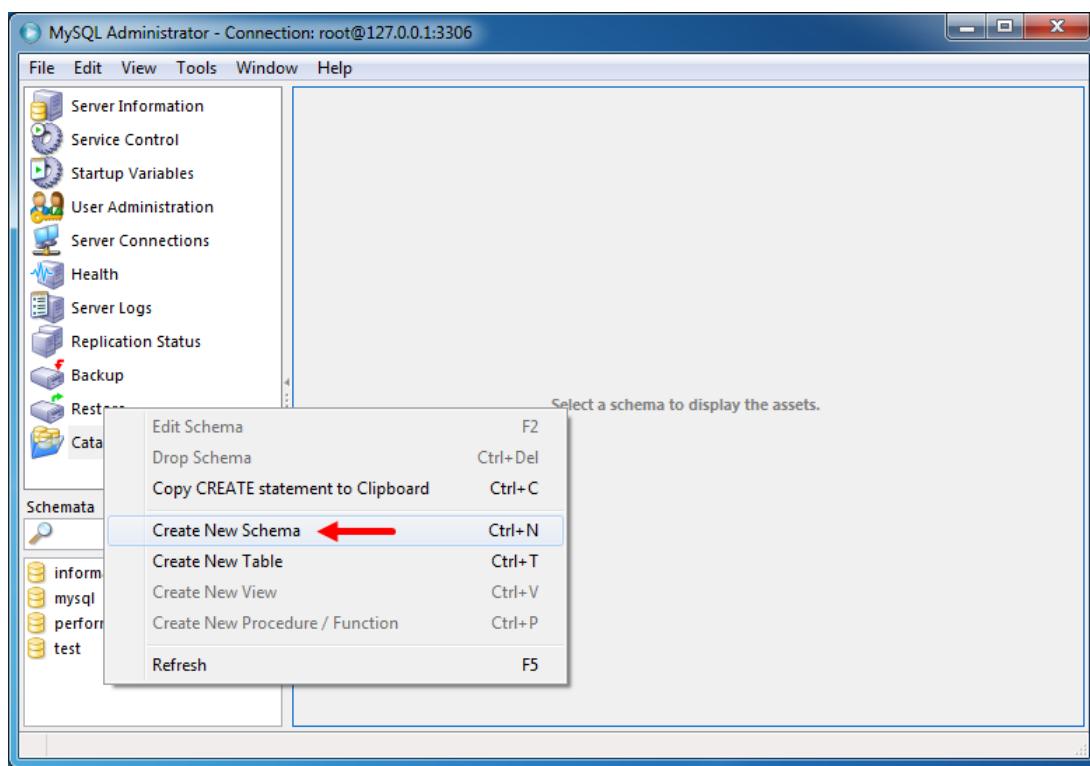
همان کارها را با استفاده محیط گرافیکی MySQL Administrator انجام دهیم. برنامه را اجرا کرده و وارد محیط آن شوید. برای ایجاد

دیتابیس بر روی گزینه Catalogs کلیک کنید.

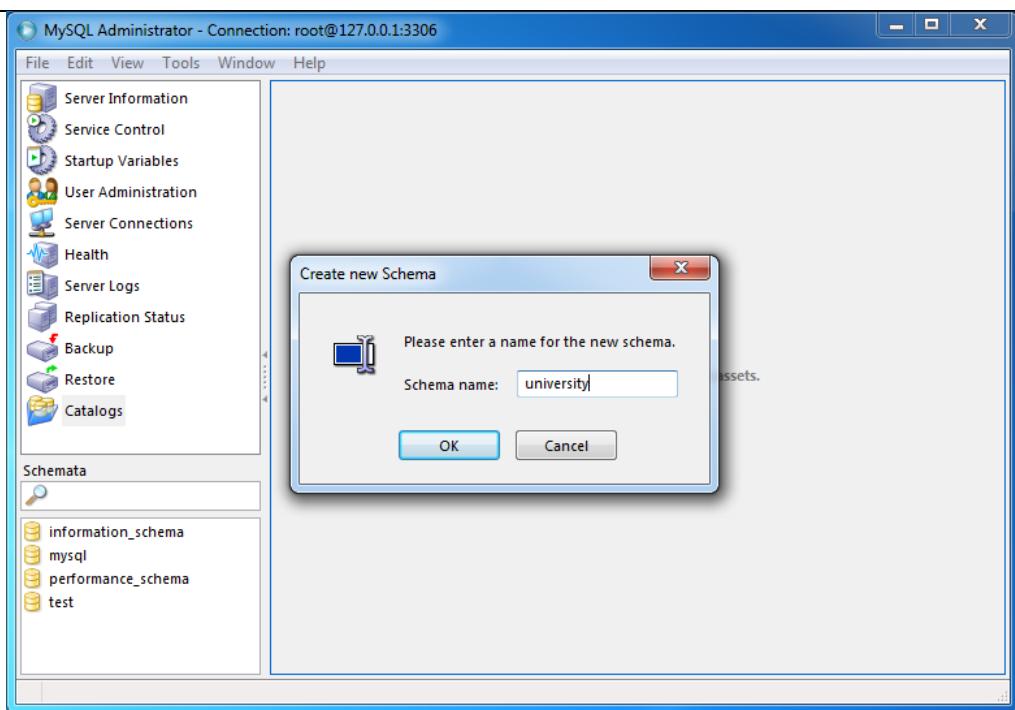
نکته: چون در درس قبل ما دیتابیس university را ایجاد کرده‌ایم، با کلیک بر روی گزینه Catalogs نام آن در داخل لیست دیتابیس‌ها وجود دارد. بر روی university راست کلیک کرده و سپس گزینه Schema را بزنید تا دیتابیس حذف شود.

ایجاد دیتابیس

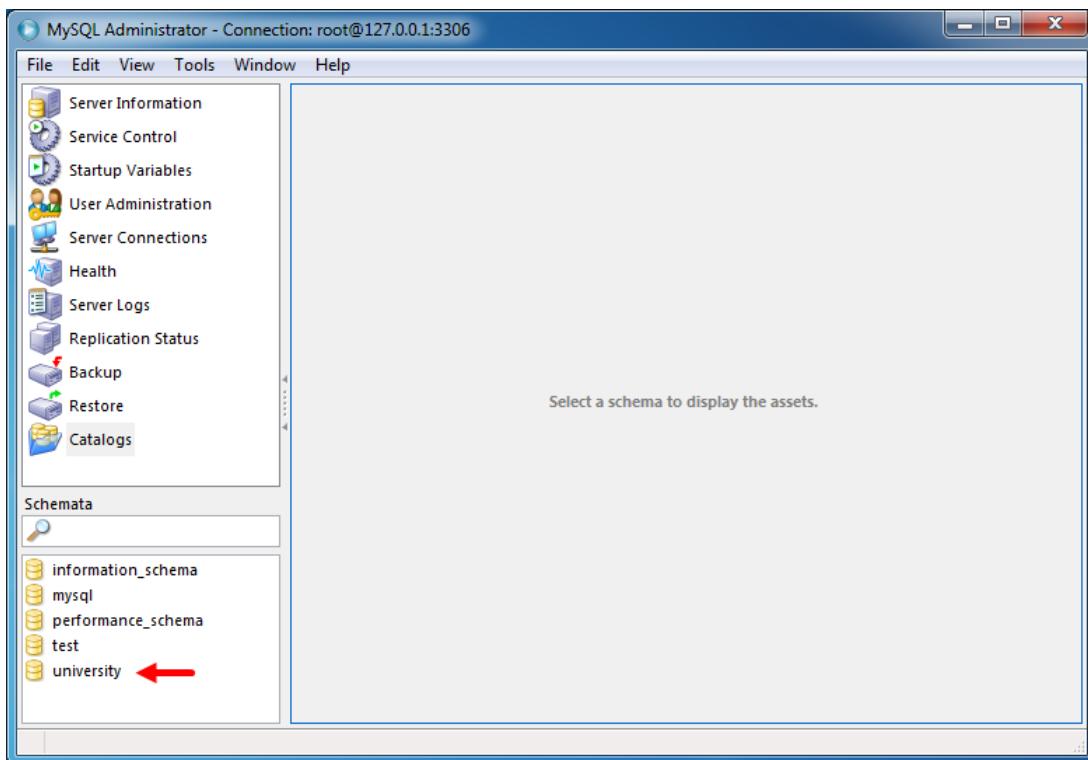
برای ایجاد دیتابیس، بر روی گزینه Catalogs کلیک کرده و سپس در یک قسمت خالی به صورت زیر راست کلیک کرده و گزینه Create New Schema را بزنید:



با کلیک بر روی گزینه Create New Schema یک کادر نمایان می‌شود که شما می‌توانید نام دیتابیستان را در داخل آن بنویسید:

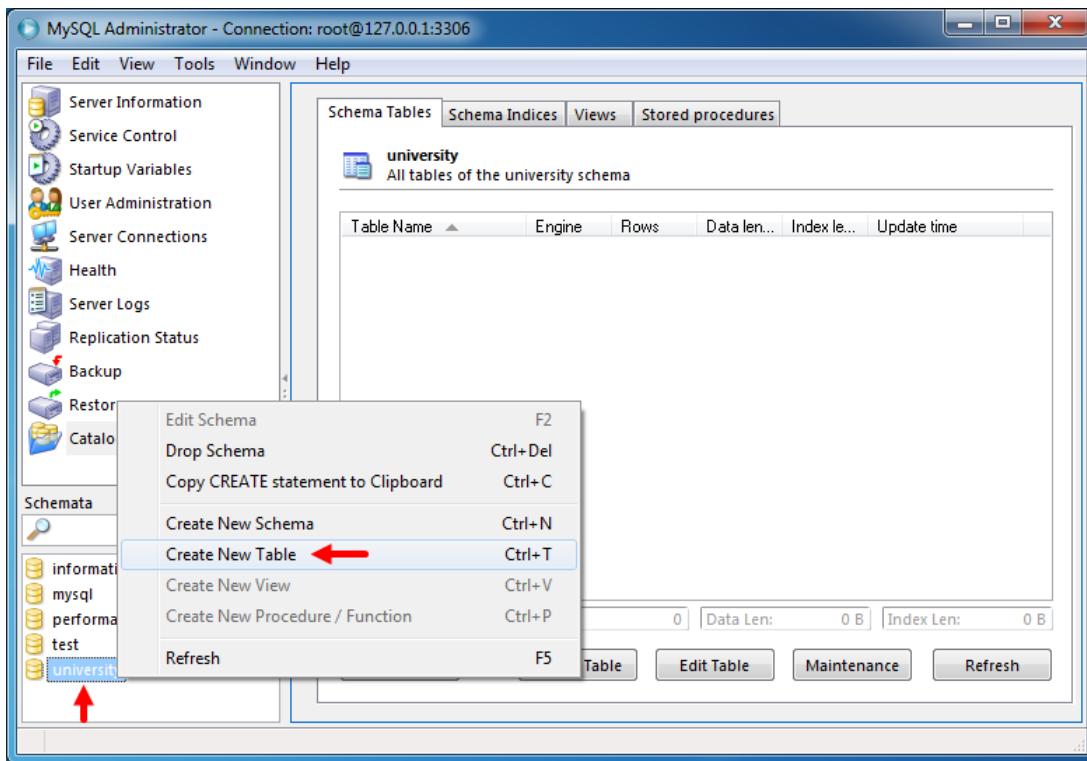


بعد از نوشتن نام دیتابیس (در اینجا university) و زدن دکمه OK، دیتابیس ساخته شده با لیست دیتابیس‌ها اضافه می‌شود :

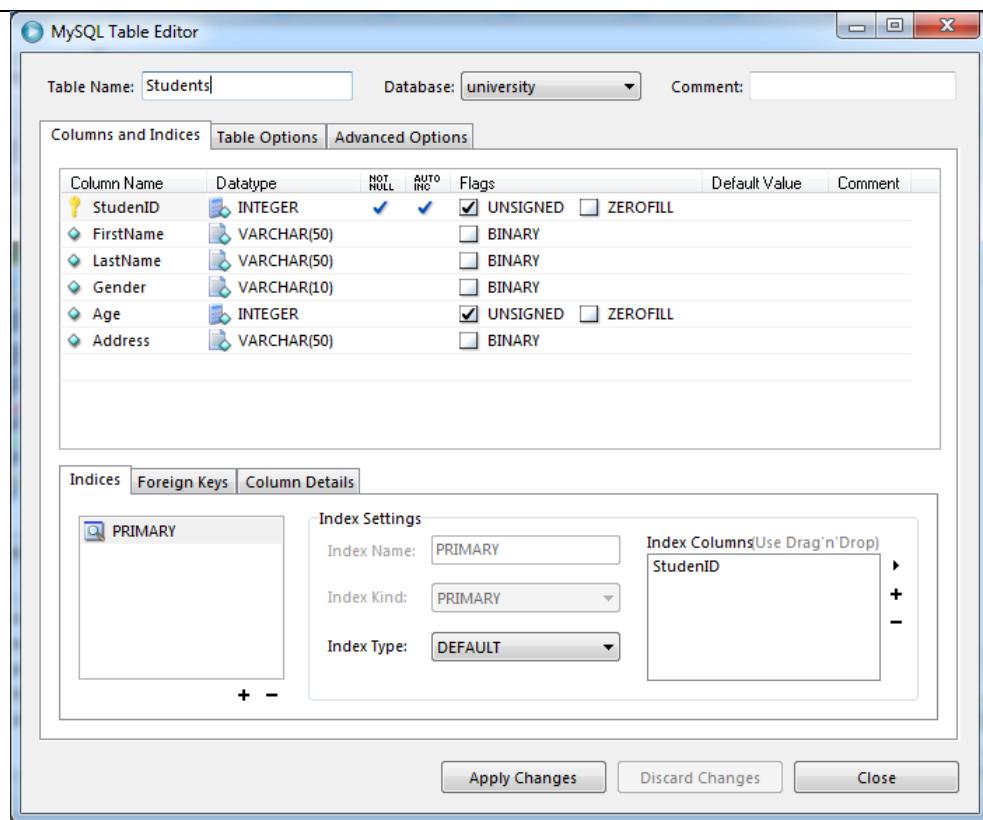


ایجاد جدول

برای ایجاد جدول بر روی نام دیتابیس راست کلیک کرده و گزینه Create New Table را بزنید :

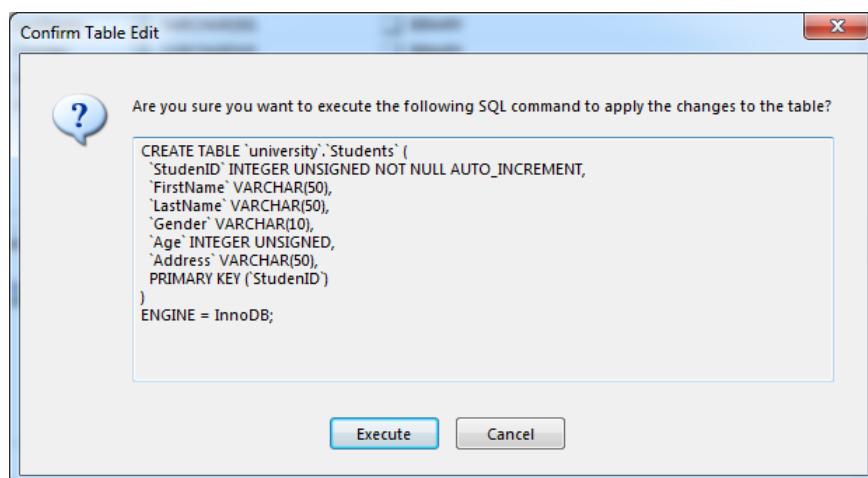


بعد از کلیک بر روی گزینه Create New Table، صفحه‌ای به صورت زیر ظاهر می‌شود. بعد از نوشتتن نام جدول در کادر مربوطه دکمه Enter را بزنید تا وارد قسمت نامگذاری ستون‌ها یا همان Column Name در شکل بالا شوید. برای نامگذاری و وارد کردن سایر مشخصات ستون‌ها کافیست بر روی سلوک‌های مربوطه دوبار کلیک کنید تا نشانگر ماوس به حالت چشمک زن درآید و شما بتوانید تغییرات را به صورت زیر اعمال کنید :

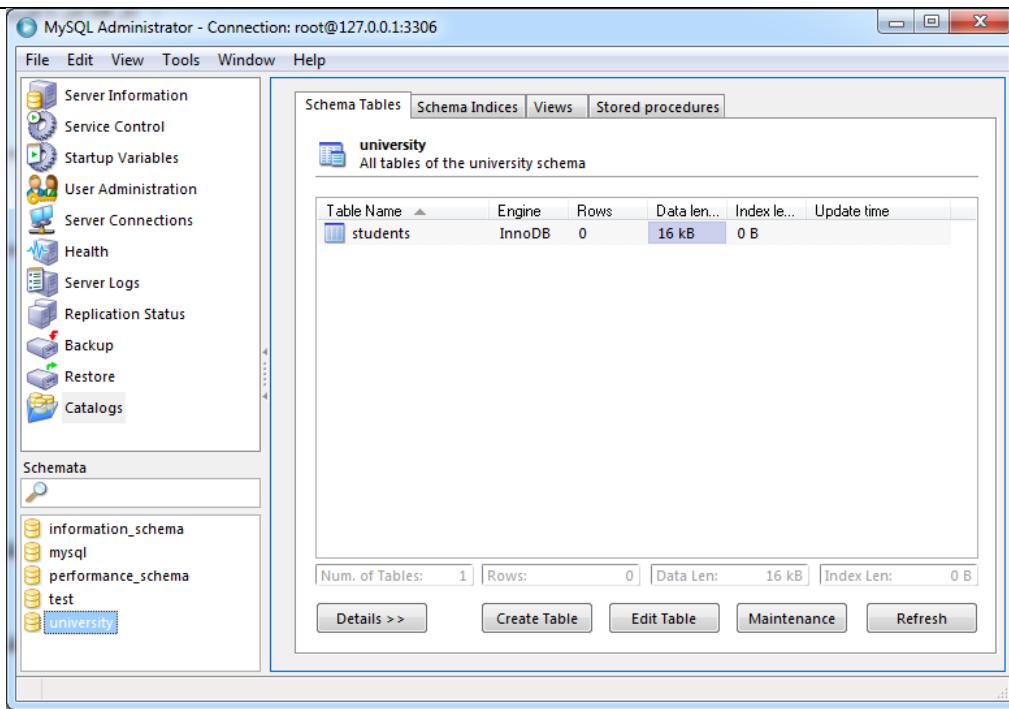


بعد از ردن دکمه Apply changes در شکل بالا کادری به صورت زیر نمایش داده می‌شود که نشان دهنده مشخصات کلی جدول است،

بر روی دکمه Execute کلیک کنید تا جدول و ستون‌های آن در داخل دیتابیس ایجاد شوند :

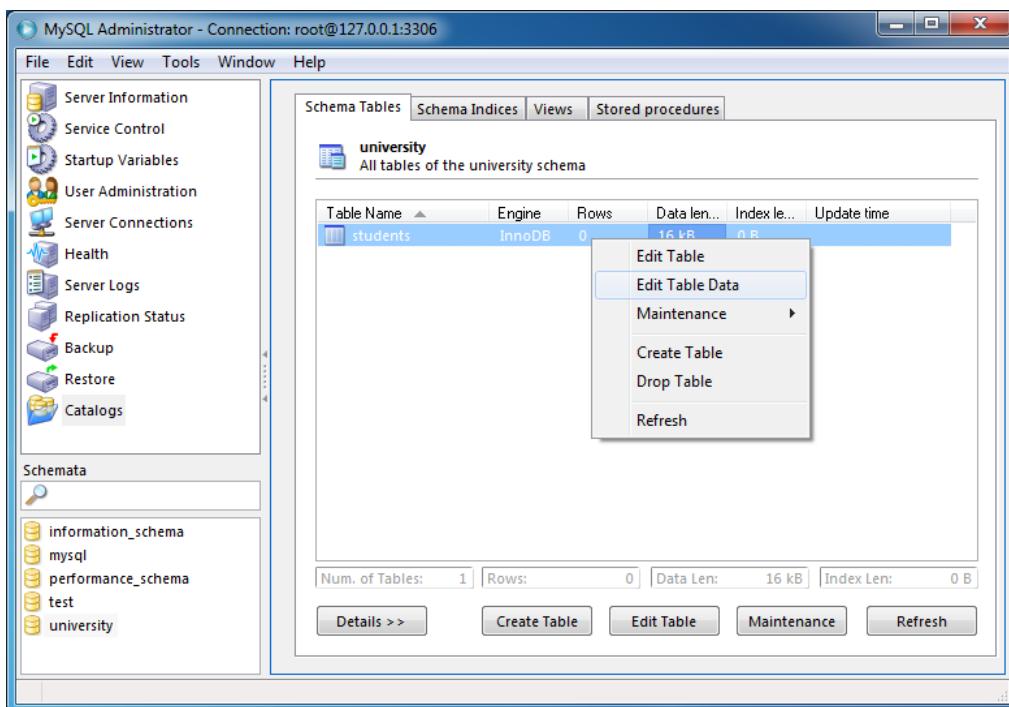


حال اگر بر روی دیتابیسمان یعنی university کلیک کنیم، مشاهده می‌کنیم که حاوی یک جدول به نام Students است :



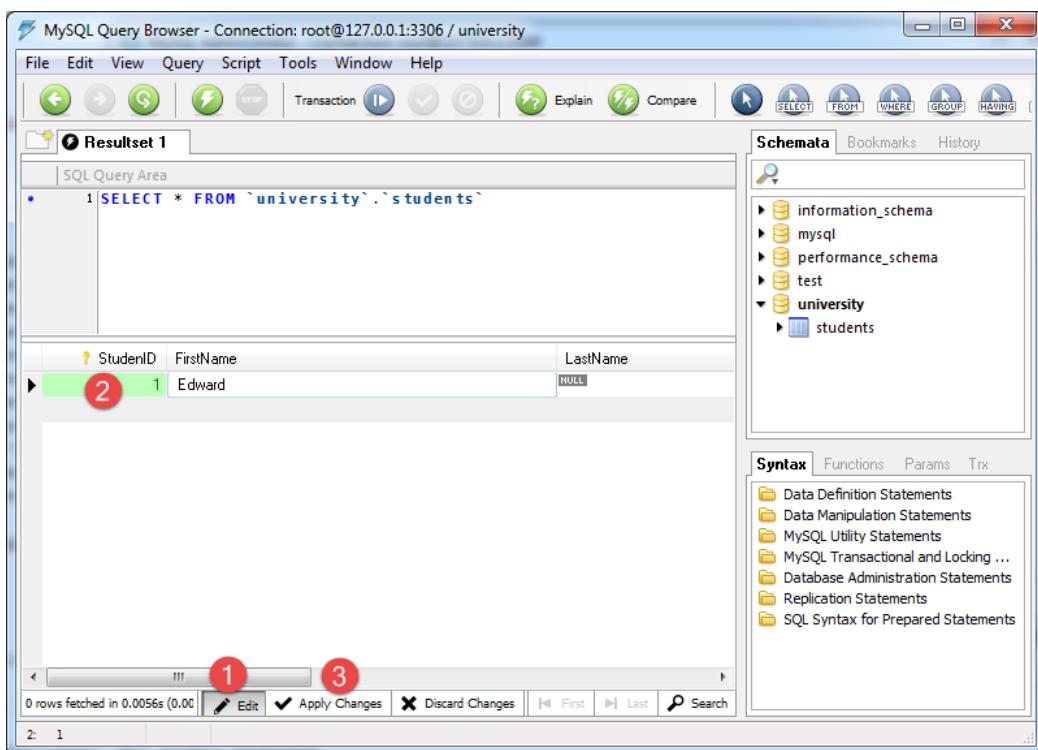
وارد کردن داده‌ها و اجرای دستورات MySQL

برای وارد کردن داده‌ها در جدول هم بر روی نام جدول راست کلیک کرده و سپس گزینه Data را بزنید :

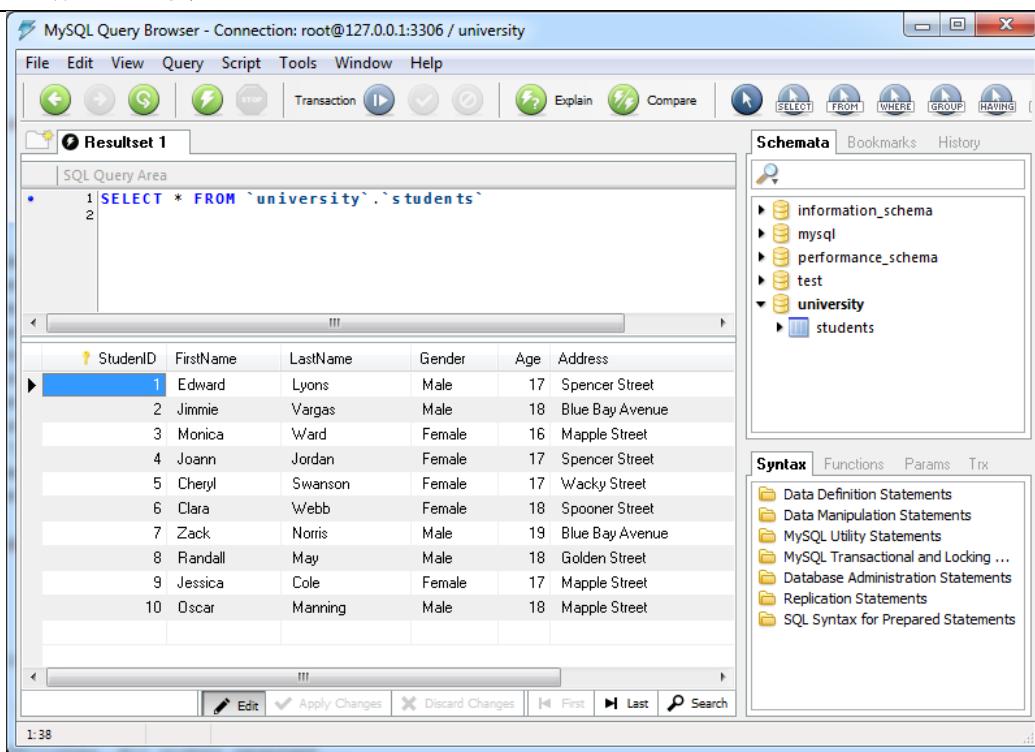


بعد از کلیک بر روی گزینه Edit Table Data صفحه‌ای به صورت زیر ظاهر می‌شود. برای وارد کردن داده‌ها همانند شکل زیر ابتدا گزینه

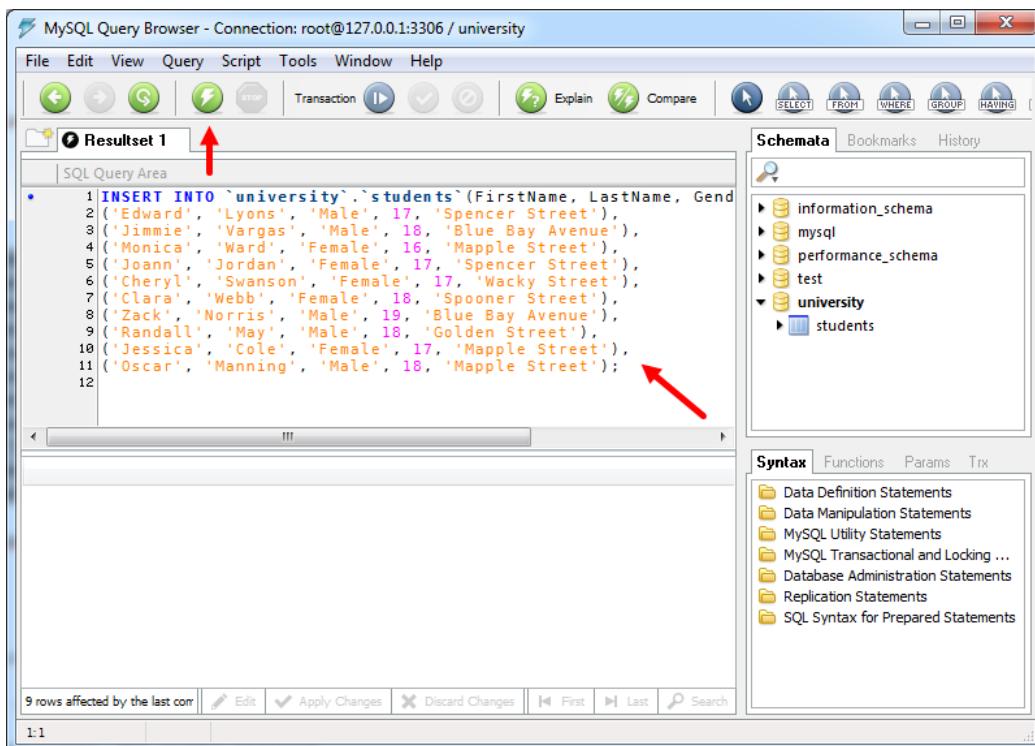
را بزنید تا به شما اجازه داده شود که اطلاعات را وارد نمایید:



در شکل بالا همانطور که احتمالاً متوجه شده‌اید با دوبار کلیک بر روی هر سلول به شما اجازه داده می‌شود که اطلاعات را وارد کنید و بعد از اتمام کار یک سلول با زدن دکمه Tab اطلاعات سلول بعد را وارد کنید و این کار را تا وارد کردن همه داده‌ها تکرار کرده و سپس گزینه Apply Changes را بزنید:

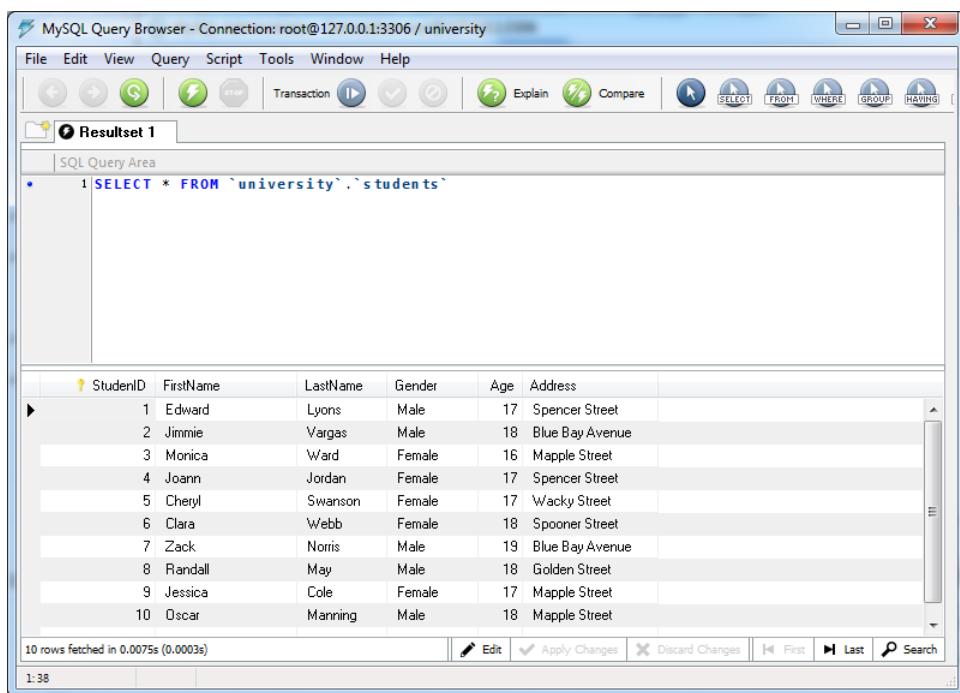


الته به این نکته توجه کنید که در این محیط می‌توان با استفاده از کدنویسی و به صورت زیر هم داده را وارد کرد :



برای این منظور کافیست که مانند شکل بالا دستورات را در کادر بالا نوشته و دکمه جرقه شکل را بفشارید. حال برای اطمینان از اینکه کار

با دیتابیس در این محیط با محیط کنسول تفاوت ندارد چند دستور از دستورات درس قبل را در این محیط آزمایش کنید:



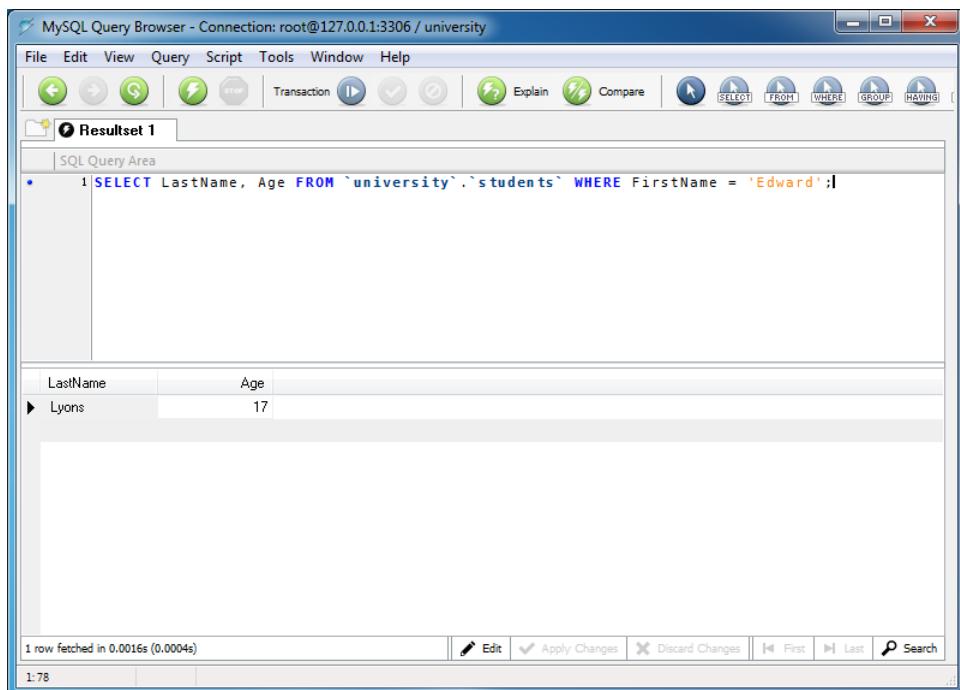
The screenshot shows the MySQL Query Browser interface. The SQL Query Area contains the following query:

```
1 | SELECT * FROM `university`.`students`
```

The results are displayed in a table:

	StudentID	FirstName	LastName	Gender	Age	Address
▶	1	Edward	Lyons	Male	17	Spencer Street
	2	Jimmie	Vargas	Male	18	Blue Bay Avenue
	3	Monica	Ward	Female	16	Maple Street
	4	Joann	Jordan	Female	17	Spencer Street
	5	Cheryl	Swanson	Female	17	Wacky Street
	6	Clara	Webb	Female	18	Spooner Street
	7	Zack	Norris	Male	19	Blue Bay Avenue
	8	Randall	May	Male	18	Golden Street
	9	Jessica	Cole	Female	17	Maple Street
	10	Oscar	Manning	Male	18	Maple Street

10 rows fetched in 0.0075s (0.0003s)



The screenshot shows the MySQL Query Browser interface. The SQL Query Area contains the following query:

```
1 | SELECT LastName, Age FROM `university`.`students` WHERE FirstName = 'Edward';
```

The results are displayed in a table:

LastName	Age
▶ Lyons	17

1 row fetched in 0.0016s (0.0004s)

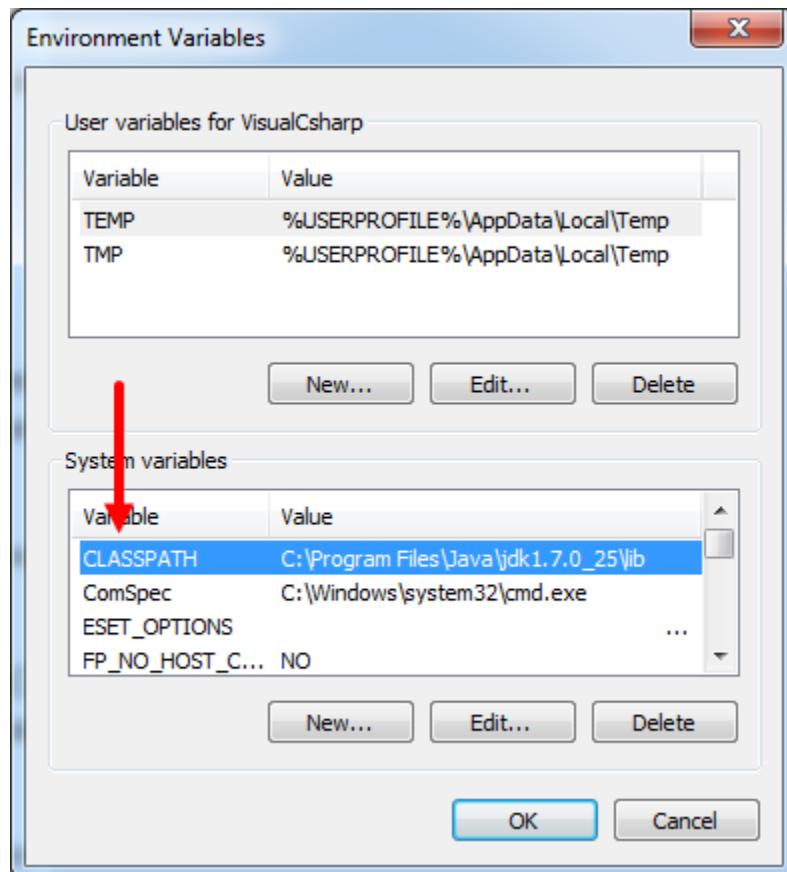
JDBC چیست؟

JDBC یا Java Database Connectivity یک رابط نرم افزاری (API) است، که به برنامه های جاوا اجازه اتصال و تبادل اطلاعات با بانک اطلاعاتی را می دهد. این رابط مجموعه ای از interface ها و کلاس های نوشته شده توسط زبان جاوا می باشد و با استفاده از آن می توان دستورات SQL و PL/SQL را تقریباً بر روی هر نوع پایگاه داده ای اجرا کرد. JDBC برای اولین بار به عنوان بخشی از JDK ۱.۱ به برنامه نویسان معرفی شد. البته JDBC ورژن مختص خود و مستقل از جاوا را دارد آخرين ورژن موجود از JDBC ورژن ۴/۰ می باشد که در SE7 وجود دارد. قبل از شروع کار با JDBC باید متغیرهای زیر را در پنجره Environment Variables با توجه به آموزش [ایجاد یک](#)

[برنامه ساده در جاوا](#)، وارد کنید :

```
JAVA_HOME: C:\Program Files\Java\jdk1.7.0_25
CLASSPATH: C:\Program Files\Java\jdk1.7.0_25\jre\lib.
PATH:      C:\Program Files\Java\jre1.7.0_25\bin.
```

به عنوان مثال متغیر CLASSPATH را به صورت زیر تعریف کنید :



به این نکته توجه کنید که من در سیستم خودم نسخه `jdk1.7.0_25` را نصب کرده‌ام و ممکن است که شما از نسخه دیگر استفاده کنید. پس مراقب باشید که نسخه خود را جایگزین کنید. بعد از نصب JDK دو پکیج `javax.sql` و `java.sql` به طور خودکار در دسترس شما قرار می‌گیرد. وارد کردن یکی از این دو Package در برنامه هنگام کار با دیتابیس الزامی است.

مراحل ارتباط با بانک در جاوا

در این درس می‌خواهیم یک مراحل ارتباط با بانک اطلاعاتی در جاوا را با استفاده از یک برنامه ساده به شما آموزش دهیم. این ارتباط شامل مراحل زیر است :

۱. وارد کردن پکیج `*.java.sql.*`
۲. ثبت یا بارگذاری یک راه انداز (Driver) در حافظه رم.
۳. ارتباط با بانک.
۴. اجرای دستورات بر روی بانک.
۵. پردازش نتایج حاصل از اجرای دستورات بر روی بانک.
۶. پاک کردن اشیاء بی استفاده و آزاد کردن حافظه.

در زیر یک مثال ساده با توجه به مراحل بالا ذکر شده است. نگران کدهای زیر نباشید. در درس‌های آینده در مورد آن‌ها توضیح می‌دهیم

:

```
package myfirstprogram;

import java.sql.*; //Step 1

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver").newInstance(); //Step 2
            String url = "jdbc:mysql://localhost:3306/University?user=root&password="; //Step 3
            Connection connection = DriverManager.getConnection(url);

            Statement statement = connection.createStatement(); //Step 4
            statement.executeQuery("SELECT * FROM Students");

            ResultSet result = statement.getResultSet(); //Step 5
            result.beforeFirst();
            while (result.next())

```

```
        {
            System.out.println(
                result.getString("StudentID") + "\t" +
                result.getString("FirstName") + "\t" +
                result.getString("LastName") + "\t" +
                result.getString("Gender") + "\t" +
                result.getString("Age") + "\t" +
                result.getString("Address")
            );
        }

        result.close();
        statement.close();
        connection.close();
    }
    catch (Exception ex)
    {
        System.out.println(ex.getMessage());
    }
}
```

در کد بالا هر مرحله را با ایجاد یک خط فاصله از مراحل دیگر جدا کرده‌ایم. در درس‌های آینده شما را به جزئیات بیشتر کدهای بالا آشنا می‌کنیم. چون در مرحله ۳ و ۴ دستورات را بر روی جدول Students که در درس‌های قبل ایجاد کردیم، اجرا کرده‌ایم، خروجی به صورت

زیر خواهد بود :

1	Edward	Lyons	Male	17	Spencer Street
2	Jimmie	Vargas	Male	18	Blue Bay Avenue
3	Monica	Ward	Female	16	Mapple Street
4	Joann	Jordan	Female	17	Spencer Street
5	Cheryl	Swanson	Female	17	Wacky Street
6	Clara	Webb	Female	18	Spooner Street
7	Zack	Norris	Male	19	Blue Bay Avenue
8	Randall	May	Male	18	Golden Street
9	Jessica	Cole	Female	17	Mapple Street
10	Oscar	Manning	Male	18	Mapple Street

چیست JDBC Driver

JDBC Driver یا کانکتورها، فایل‌های با پسوند jar هستند، که توسط شرکت‌های سازنده پایگاه داده تولید می‌شوند و از آن‌ها برای برقراری ارتباط بین برنامه‌های Java و پایگاه داده استفاده می‌شود. چون در این سری آموزشی ما می‌خواهیم ارتباط بین برنامه‌هایمان با بانک اطلاعات، MySQL، Oracle، کنیم، سس، ناپد JDBC Driver مربوط به این بارگاه داده را دانلود کنیم. در اینجا دانلود را انجام دهیم.

۱۰۵

<http://www.dl.w3-farsi.com/Software/java/mysql-connector-java-5.1.39-bin.jar>

بعد از دانلود فایل بالا یک پوشه به نام MySQLConnector در درایو C ایجاد کرده و فایل را در داخل آن بگذارید. پس مسیر فایل شما به

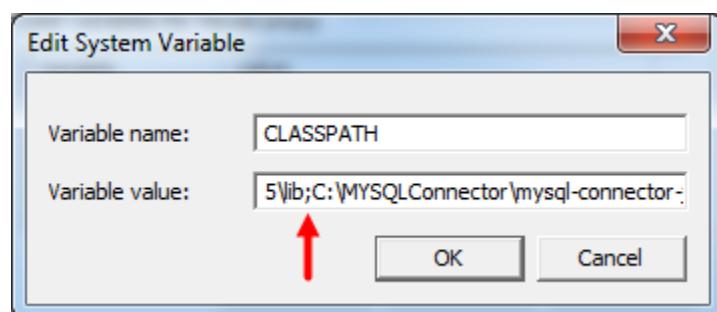
صورت زیر خواهد بود :

```
C:\MySQLConnector\mysql-connector-java-5.1.39-bin.jar
```

همانطور که گفتیم برای ارتباط با بانک ما به این فایل احتیاج داریم. برای استفاده از این فایل باید مسیر آن را به متغیر CLASSPATH اضافه کنیم. برای این کار قبل و بعد از مسیر فوق دو علامت ویرگول (;) به صورت زیر اضافه کرده :

```
;C:\MySQLConnector\mysql-connector-java-5.1.39-bin.jar;
```

و سپس آن را متغیر مذکور مانند شکل زیر کپی می‌کنیم :



اگر این کار را انجام ندهید، مجبور می‌شوید که در هنگام کامپایل برنامه مسیر این فایل را با استفاده از دستور cp - به صورت زیر مشخص کنید :

```
java -cp C:\MySQLConnector\mysql-connector-java-5.1.39-bin.jar; ProgramName
```

نام برنامه شما است. حال که فایل JDBC Driver را دانلود و پیکربندی کردید نوبت به اتصال به دیتابیس می‌رسد. قبل از اتصال به دیتابیس باید یک کلاس را ثبت یا رجیستر کنیم. منظور از رجیستر کردن بارگذاری کلاس در داخل حافظه رم است. کلاسی که می‌خواهیم بارگذاری کنیم، کلاس Driver می‌باشد که در پکیج com.mysql.jdbc و این پکیج هم به نوبه خود در داخل فایل mysql-connector-java-5.1.39-bin.jar قرار دارد. برای ثبت کلاس Driver دو راه وجود دارد :

- استفاده از متدها forName() مربوط به کلاس java.lang.Class

- استفاده از متدهای registerDriver() مربوط به کلاس java.sql.DriverManager

دو کد زیر برای ثبت کلاس Driver معادل هم هستند :

```

try
{
    Class.forName("com.mysql.jdbc.Driver").newInstance();

    //Some Code
}
catch (ClassNotFoundException ex)
{
    System.out.println(ex.getMessage());
}
try
{
    DriverManager.registerDriver(new com.mysql.jdbc.Driver());

    //Some Code
}
catch (SQLException ex)
{
    System.out.println(ex.getMessage());
}

```

همانطور که احتمالاً از دو کد بالا متوجه شده‌اید، استفاده از دو متد () و registerDriver() ممکن است به ترتیب باعث ایجاد دو استثنای SQLException و ClassNotFoundException شود. پس بهتر است که آن‌ها را در داخل دستور try catch برویسید. به عنوان آخرین نکته یادآور می‌شویم که در همین حد کافیست که بدانید، قبل از اتصال به بانک اطلاعاتی باید کلاس Driver مربوط به بانک را در حافظه RAM بارگذاری کنید. البته این کلاس در پایگاه‌های داده مختلف دارای نام متفاوتی است. مثلاً بارگذاری کلاس Oracle در پایگاه داده داده به صورت زیر است :

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

حال که با کاربرد JDBC آشنا شدید در درس بعد در مورد نحوه اتصال به بانک اطلاعاتی توضیح می‌دهیم.

ارتباط با بانک

بعد از بارگذاری Driver در حافظه، نوبت به ارتباط با بانک اطلاعاتی می‌رسد. برای این کار از متد دیگر کلاس DriverManager به نام getConnection() استفاده می‌کنیم. این متد دارای سربارگذاری‌های زیر است :

```

getConnection(String url)
getConnection(String url, Properties prop)
getConnection(String url, String user, String password)

```

همانطور که در کدهای بالا مشاهده می‌کنید، این متدهای رشته که در اصل مسیر بانک اطلاعاتی و مشخصات اتصال به آن است را دریافت

می‌کند. حالت کلی این رشته به صورت زیر است :

```
jdbc:RDBMS://hostname:port Number/databaseName
```

کد بالا نمای کلی رشته و در جدول زیر شکل دقیق‌تر آن برای اتصال به دیتابیس‌های مختلف نشان داده شده است :

RDBMS	JDBC Driver	URL
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname:port Number/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:tdin:@hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:hostname: port Number/databaseName

حال که با سربارگذاری های این متدهای مختلفی که این متدهای رشته ارتباط با دیتابیس‌ها دریافت می‌کند آشنا شدید، اجازه دهید

که با استفاده از آن به دیتابیس و جدولی که در درس‌های قبل ایجاد کردیم، متصل شویم. برای این کار به صورت زیر عمل می‌کنیم :

```
Connection connection =
    DriverManager.getConnection("jdbc:mysql://localhost:3306/university?user=root&password=");
```

کد بالا را به صورت خوانتری هم می‌توان نوشت :

```
String url = "jdbc:mysql://localhost:3306/university?user=root&password=";
Connection connection = DriverManager.getConnection(url);
```

در تعریف رشته یا url مشاهده می‌کنید که مقدار پورت را ۳۳۰۶، نام کاربری را root و پسورد را خالی قرار داده‌ایم. این مقادیر همان

مقادیری هستند که در هنگام نصب سرور MySQL مشخص کردیم. همانطور که در کد بالا مشاهده می‌کنید :

برای اتصال نهایی به بانک باید یک شیء از Interface Connection یا رابط ایجاد کرده و مسیر بانک را از طریق متدهای

getconnection() به آن معرفی کنیم. کد درس قبل را به صورت زیر کامل‌تر می‌کنیم :

```
try
```

```
{
    DriverManager.registerDriver(new com.mysql.jdbc.Driver());

    String url = "jdbc:mysql://localhost/university?user=root&password=";
    Connection connection = DriverManager.getConnection(url);

    //some code

    connection.close();
}
catch (SQLException ex)
{
    System.out.println(ex.getMessage());
}
```

تا اینجای کار ارتباط ما با بانک برقرار شد. یک نکته را که بهتر است در همین جا یادآور شویم این است، بعد از اتصال به بانک و انجام عملیات مختلف بر روی آن بهتر است اتصال را قطع کنیم. همانطور که در کد بالا مشاهده می‌کنید این کار را با استفاده از متدها close() و finally() مربوط به Connection انجام می‌دهیم، که همان مرحله ۶ یعنی پاک کردن اشیاء بلا استفاده و آزاد کردن حافظه می‌باشد. حتی اگر شما متدهای close() را فراخوانی نکنید، زباله روب یا garbage collector جاوا اشیاء بی استفاده را پاک می‌کند ولی از لحاظ برنامه نویسی بهتر است که خودتان این کار را انجام دهید. برای اطمینان از بسته شدن ارتباط می‌توان متدهای close() و finally() در بلوک فراخوانی کرد، چون کدهای این بلوک چه خطایی در برنامه رخ دهد و چه ندهد، اجرا می‌شوند :

```
try
{
    //some code
}
catch (SQLException ex)
{
    //some code
}
finally
{
    connection.close();
}
```

اجرای دستورات بر روی بانک

بعد از برقراری ارتباط با بانک، نوبت به ارسال دستورات SQL به آن می‌رسد. برای این منظور سه رابط وجود دارند که دارای متدها و خواص مفیدی هستند که شما را قادر می‌سازند دستورات را به بانک ارسال کرده و نتایج حاصل از آنها را دریافت و پردازش کنید. در جدول زیر اسم این رابطها و کاربرد آنها ذکر شده است:

رابط	کاربرد
Statement	برای اجرای دستورات ساده SQL به کار می‌رود.
PreparedStatement	برای اجرای دستورات SQL به صورت پویا یا دینامیک به کار می‌رود. یا به نوعی می‌توان گفت که برای اجرای دستوراتی از SQL به کار می‌رود که در زمان اجرا پارامتر دریافت می‌کنند.
CallableStatement	در هنگام کار با رویه‌های ذخیره شده یا stored procedures به کار می‌رود. این رابط هم می‌تواند در زمان اجرا پارامتر قبول کند.

قبل از استفاده از هریک از رابطهای بالا باید یک شیء از آن‌ها ایجاد و نتایج حاصل از یک متاد از رابط Connection را در داخل آن قرار داد. کدهای زیر نحوه ایجاد شیء از این رابطهای فراخوانی متدهای رابط Connection مربوط به هریک از آن‌ها نمایش داده شده است

:

```
Statement statement = connection.createStatement();
PreparedStatement preparedstatement = connection.prepareStatement();
CallableStatement callablestatement = connection.prepareCall();
```

رابط Statement

همانطور که در بالا اشاره شد قبل از استفاده از رابط Statement از رابط Connection createStatement() باید متاد () از رابط Statement باشد که در بالا اشاره شد. همچنان که در اینجا مذکور شد قبل از این کار باید متاد Statement را فراخوانی کنید :

```
Statement statement = connection.createStatement();
```

زمانیکه یک شیء از رابط Statement ایجاد کردید می‌توانید دستورات SQL را با استفاده از سه متاد زیر اجرا کنید :

متاد	توضیح	نوع برگشتی
execute()	(در مورد شیء ResultSet در درس آینده توضیح می‌دهیم)	boolean

تعداد سطرهایی را که تحت تأثیر دستور SQL قرار گرفته‌اند را برمی‌گرداند. مثلاً تعداد سطرهایی از دیتابیس را که UPDATE، INSERT و یا DELETE شده‌اند را برمی‌گرداند.	int	executeUpdate()
از این متدها معمولاً برای اجرای دستور INSERT استفاده می‌شود و نتایج را به صورت یک <code>ResultSet</code> بر می‌گرداند. این شیء نماینده سطرهای برگشت داده شده، حاصل از دستور SELECT است که در درس بعد درباره آن توضیح می‌دهیم.	ResultSet	executeQuery()

رابط PreparedStatement

رابط PreparedStatement از رابط Statement ارث بری می‌کند و این توانایی را به شما می‌دهد که در زمان اجرای برنامه پارامترهایی را برای تکمیل دستور SQL ارسال کنید. قبل از استفاده از رابط Statement باید متدهای `prepareStatement()` از رابط Connection را

فراخوانی کنید :

```
try
{
    String SQL = "Update Students SET Age = ? WHERE StudentID = ?";
    PreparedStatement preparedstatement = connection.prepareStatement(SQL);
    //some code
}
catch (SQLException e)
{
    //some code
}
```

همانطور که در کد بالا مشاهده می‌کنید دستور SQL دارای علامت‌های ? می‌باشد. این علامت‌های سؤال با مقادیر یا پارامترهایی که ما ارسال می‌کیم جایگزین می‌شوند. هر کدام از این علامتها دارای یک اندیس می‌باشد. مثلاً اولین علامت سؤال دارای اندیس ۱، دومین علامت دارای اندیس ۲ و ... می‌باشد. ارسال پارامترها با استفاده از متدهایی از رابط PreparedStatement که به صورت `setXXX()` هستند، صورت می‌گیرد، که XXX نوع داده‌ای که قرار است توسط متدهای ارسال شود را نشان می‌دهد. مثلاً برای ارسال یک پارامتر از نوع عدد صحیح از متدهای `setInt()` استفاده می‌شود. در زیر لیست این متدها ذکر شده است :

متدها	کاربرد
<code>setInt(int paramInt, int value)</code>	یک مقدار صحیح را جایگزین علامت ? می‌کند.
<code>setString(int paramInt, String value)</code>	یک مقدار رشته‌ای را جایگزین علامت ? می‌کند.

یک مقدار float را جایگزین علامت ؟ می‌کند.	setFloat(int paramIndex, float value)
یک مقدار double را جایگزین علامت ؟ می‌کند.	setDouble(int paramIndex, double value)

همانطور که مشاهده می‌کنید، این متدها دو پارامتر دریافت می‌کنند که اولین پارامتر اندیس علامت سؤال و دومین پارامتر مقداری که قرار است به جای علامت سؤال قرار بگیرد، می‌باشد. از آنجاییکه رابط از رابط ارث بری می‌کند، در نتیجه می‌تواند از متدهای آن یعنی executeUpdate() و executeQuery() برای اجرای دستورات بر روی بانک هم استفاده کند. کد ابتدایی درس را به صورت زیر کامل‌تر می‌کنیم :

```

try
{
    DriverManager.registerDriver(new com.mysql.jdbc.Driver());

    String url = "jdbc:mysql://localhost:3306/university?user=root&password=";
    Connection connection = DriverManager.getConnection(url);

    Statement statement = connection.createStatement();
    statement.executeQuery("SELECT * FROM Students");

    //some code
}
catch (Exception ex)
{
    System.out.println(ex.getMessage());
}

```

در مورد رابط CallableStatement در درس‌های آینده به طور کامل توضیح می‌دهیم.

پردازش نتایج حاصل از اجرای دستورات بر روی بانک

دستورات SQL که داده‌ها را از دیتابیس می‌خوانند، آن‌ها را در یک ResultSet به نام Interface ResultSet می‌ریزند. در واقع یک سری از مجموعه نتایج است و در آن واحد فقط یک سطر را نمایش می‌دهد. می‌توان آن را به صورت یک جدول به صورت زیر تصور کرد :

StudentID	FirstName	LastName	Gender	Age	Address
1	Edward	Lyons	Male	17	Spencer Street
2	Jimmie	Vargas	Male	18	Blue Bay Avenue
3	Monica	Ward	Female	16	Mapple Street
4	Joann	Jordan	Female	17	Spencer Street
5	Cheryl	Swanson	Female	17	Wacky Street
6	Clara	Webb	Female	18	Spooner Street
7	Zack	Norris	Male	19	Blue Bay Avenue
8	Randall	May	Male	18	Golden Street
9	Jessica	Cole	Female	17	Mapple Street
10	Oscar	Manning	Male	18	Mapple Street

با اجرای یک Statement یا PreparedStatement یعنی فراخوانی متدهای executeQuery() به روش‌های زیر ایجاد ResultSet می‌شود :

```
Statement statement = connection.createStatement();
ResultSet result = statement.executeQuery("SELECT * FROM Students");
```

یا

```
String sql = "SELECT * FROM Students";
PreparedStatement statement = connection.prepareStatement(sql);
ResultSet result = statement.executeQuery();
```

دارای یک نشانگر (cursor) است که به سطر جاری اشاره می‌کند. متدهای این رابط به سه گروه زیر تقسیم می‌شوند :

- متدهای پیمایش گر: برای حرکت نشانگر به کار می‌روند.

- متدهای دریافت کننده مقادیر: برای مشاهده داده یک ستون از سطر جاری به کار می‌روند.

- متدهای به روز کننده مقادیر: برای به روز کردن داده یک ستون از سطر جاری به کار می‌روند.

نشانگر بر طبق خصیصت‌های ResultSet در بین سطرهای حرکت می‌کند. این خواص در زمان ایجاد دستور یا همان Statement یعنی

قبل از ایجاد ResultSet تعریف می‌شوند و تعیین می‌کنند که مثلاً ResultSet فقط خواندنی باشد یا قابل بروزرسانی و در زیر

نحوه تعریف خصوصیات ResultSet در متدهای رابط Connection آمده است :

```
createStatement(int RSType, int RSConcurrency);
prepareStatement(String SQL, int RSType, int RSConcurrency);
prepareCall(String sql, int RSType, int RSConcurrency);
```

در کدهای بالا RSType نوع و RSConcurrency فقط‌خواندنی یا قابل به روزرسانی بودن ResultSet را مشخص می‌کند.

ResultSet انواع

انواع ResultSet در جدول زیر آمده است. اگر هیچ نوعی برای ResultSet تعریف نکنید، برای آن به طور خودکار نوع TYPE_FORWARD_ONLY در نظر گرفته می‌شود.

نوع	توضیح
ResultSet.TYPE_FORWARD_ONLY	نشانگر فقط می‌تواند رو به جلو در داخل مجموعه نتایج حرکت کند.
ResultSet.TYPE_SCROLL_INSENSITIVE	نشانگر می‌تواند به عقب و جلو حرکت کند و مجموعه جواب یا ResultSet بعد از اینکه ایجاد شد، نسبت به تغییرات دیتابیس حساس نیست.
ResultSet.TYPE_SCROLL_SENSITIVE	نشانگر می‌تواند به عقب و جلو حرکت کند و مجموعه جواب یا ResultSet بعد از اینکه ایجاد شد، نسبت به تغییرات دیتابیس حساس است.

دومین خاصیتی هم که برای ResultSet می‌توان تعریف کرد، دارای مقادیر زیر است :

Concurrency	توضیح
ResultSet.CONCUR_READ_ONLY	مجموعه نتایج را فقط‌خواندنی می‌کند.
ResultSet.CONCUR_UPDATABLE	با وجود این خاصیت می‌توان مجموعه نتایج را بروز کرد.

با توجه به مطالب بالا برای ایجاد یک ResultSet که بتوان در آن فقط رو به جلو حرکت کرد و داده‌های آن فقط‌خواندنی و غیرقابل ویرایش باشد باید به صورت زیر عمل کنید :

```
try
```

```
{
    Statement statement =
        connection.createStatement(resultSet.TYPE_FORWARD_ONLY, resultSet.CONCUR_READ_ONLY);

    //some code
}
catch(Exception ex)
{
    //some code
}
```

گردش در میان سطرهای ResultSet

دارای متدهایی است که با استفاده از آن‌ها می‌توان در میان مجموعه جواب یا همان سطرهای ResultSet حرکت کرد.

لیست این متدها در جدول زیر آمده است :

متدهایی	توضیح
beforeFirst()	نشانگر را به قبل از اولین سطر منتقل می‌کند.
afterLast()	نشانگر را به بعد از آخرین سطر منتقل می‌کند.
first()	نشانگر را به اولین سطر منتقل می‌کند.
last()	نشانگر را به آخرین سطر منتقل می‌کند.
absolute()	نشانگر را به سطر خاصی منتقل می‌کند.
relative()	یک عدد دریافت می‌کند که نشان می‌دهد چه تعداد سطر را نسبت به مکان فعلی نشانگر می‌توان رو به جلو یا عقب پرش کرد.
previous()	نشانگر را به سطر قبل منتقل می‌کند و اگر سطر قبلی وجود نداشته باشد مقدار false را بر می‌گرداند.
next()	نشانگر را به سطر بعد منتقل می‌کند و اگر سطر بعدی وجود نداشته باشد مقدار false را بر می‌گرداند.
getRow()	شماره سطر جاری را بر می‌گرداند.

یک سطر جدید ایجاد می‌کند و نشانگر را در ابتدای آن قرار می‌دهد. البته جای قبلی نشانگر در حافظه می‌ماند.	<code>moveToInsertRow()</code>
وقتی که یک سطر توسط متدهای <code>moveToInsertRow()</code> ایجاد و مقداردهی شد، با فراخوانی این متدهای نشانگر به جای قبلی خود باز می‌گردد.	<code>moveToCurrentRow()</code>

ResultSet مشاهده مقادیر

رابط `ResultSet` دارای دوازده متده است که از آن‌ها برای دریافت مقادیر از سطر جاری استفاده می‌کند. به ازای هر نوع داده، دو نسخه از هر متده موجود است:

- متدهایی که نام ستون را می‌گیرند و مقدار آن را بر می‌گردانند.

- متدهایی که اندیس ستون را می‌گیرند و مقدار آن را بر می‌گردانند.

به عنوان مثال برای دریافت مقدار یک ستون از نوع `int` باید از متده `getInt()` استفاده کنید.

متده	توضیح
نام یک ستون از سطر جاری را دریافت می‌کند و مقدار آن را بر می‌گرداند. این متده برای ستون‌هایی که مقادیر آن‌ها عددی است کاربرد دارد. مثلًاً ستون سن افراد.	<code>getInt(String columnName)</code>
اندیس یک ستون از سطر جاری را دریافت می‌کند و مقدار آن را بر می‌گرداند. اندیس ستون‌ها از ۱ شروع می‌شود. این متده برای ستون‌هایی که مقادیر آن‌ها عددی است کاربرد دارد. مثلًاً ستون سن افراد.	<code>getInt(int columnIndex)</code>

همانند متدهای بالا رابط `ResultSet` دارای متدهای دیگری است که برای برگرداندن هشت نوع اصلی دیگر جاوا به کار می‌روند. مثلًاً متده `getString()`، همانند متده `getInt()` اندیس یا نام یک ستون از سطر جاری را می‌گیرد و مقدار آن را بر می‌گرداند. `ResultSet` دارای متدهای مشابه دیگری برای دریافت سایر انواع داده SQL مانند `Date`، `Time`، `TimeStamp`، `Clob` و `Blob` می‌باشد.

بروزرسانی مقادیر ResultSet

دارای مجموعه‌ای از متدهاست که از آن‌ها برای بروزرسانی مقادیر استفاده می‌کند. به ازای هر نوع داده دو متده برای بروزرسانی

آن وجود دارد :

- متدهایی که نام ستون را می‌گیرند و عملیات بروزرسانی را انجام می‌دهند.
 - متدهایی که اندیس ستون را می‌گیرند و عملیات بروزرسانی را انجام می‌دهند.
- مثلاً برای ستون‌هایی که دارای نوع رشته‌ای هستند، از این متدها به صورت زیر استفاده می‌شود :

متدها	توضیح
اندیس ستون مورد نظر ما را دریافت کرده و مقدار داخل آن را با چیزی که مد نظر ما است، تغییر می‌دهد.	<code>updateString(int columnIndex, String s)</code>
نام ستون مورد نظر ما را دریافت کرده و مقدار داخل آن را با چیزی که مد نظر ما است، تغییر می‌دهد.	<code>updateString(String columnName, String s)</code>

متدهای دیگری همانند متدهای بالا برای به روزرسانی هشت داده اصلی مانند `Object`, `String`, `URL` و انواع داده‌های SQL واقع در پکیج `java.sql`, وجود دارند. به روزرسانی یک سطر از مجموعه نتایج، به منزله تغییر ستون‌های سطر جاری شیء `ResultSet` می‌باشد.

برای اعمال این تغییرات به یک سطر از دیتابیس اصلی باید یکی از متدهای زیر را فراخوانی کنید :

متدها	توضیح
وقتی یک سطر از دیتابیس را بروزرسانی می‌کنیم، با فراخوانی این متده سطر معادل آن در <code>ResultSet</code> به روز می‌شود.	<code>updateRow()</code>
سطر جاری را از دیتابیس حذف می‌کند.	<code>deleteRow()</code>

با Refresh داده‌های ResultSet، هر گونه تغییر در آن‌ها را به دیتابیس اعمال می‌کند.	refreshRow()
تمام بروزرسانی‌های سطر جاری را لغو می‌کند.	cancelRowUpdates()
یک سطر جدید به دیتابیس اضافه می‌کند.	insertRow()

خلاصه‌ای از کار ResultSet

قبل از ایجاد ResultSet، خصوصیات از آن را که مد نظرمان است را در داخل متدهای createStatement() و executeQuery() می‌نویسیم:

```
Statement statement1 =
    connection.createStatement(resultSet.FETCH_FORWARD, resultSet.CONCUR_READ_ONLY);
```

برای ایجاد ResultSet، متدهای executeQuery() از رابط Statement را فراخوانی می‌کنیم. در کل کد زیر به منزله ایجاد یک کوئری از جدول Student می‌باشد، یعنی تمام مقادیر موجود در جدول را در داخل یک شیء ResultSet می‌ریزیم:

```
ResultSet result = statement1.executeQuery("SELECT * FROM Students");
```

تا اینجای کار ما ResultSet را ایجاد کردیم. برای پیمایش و نمایش مقادیر هم از دستور while و متدهای شیء ResultSet استفاده می‌کنیم. به کد زیر توجه کنید:

```
result.beforeFirst();

while (result.next())
{
    System.out.println(
        result.getInt("StudentID") + "\t" +
        result.getString("FirstName") + "\t" +
        result.getString("LastName") + "\t" +
        result.getString("Gender") + "\t" +
        result.getInt("Age") + "\t" +
        result.getString("Address")
    );
}
```

در کد بالا ابتدا نشانگر را با استفاده از متدهای beforeFirst() و next() به قبل از سطر اول منتقل می‌کنیم. در شرط حلقه هم با استفاده از متدهای getInt() و getString() از نوع عددی هست برای دریافت مقادیر آن‌ها از متدهای StudentID و Age و برای دریافت مقادیر سایر ستون‌ها از متدهای get() استفاده می‌کنیم. حال فرض کنید که می‌خواهیم از یک سطر خاص به بعد را پیمایش کنیم. برای اینکار ابتدا نشانگر را با

استفاده از متدهای absolute() و beforeFirst() به سطر مورد نظر می‌بریم و سپس عملیات پیمایش را انجام می‌دهیم. به جای متدهای absolute() و beforeFirst() کد زیر را نوشتیم و کد را اجرا کنید:

```
result.absolute(7);

while (result.next())
{
    //some code
}

8    RandallMay    Male   18      Golden Street
9    JessicaCole   Female  17      Mapple Street
10   Oscar Manning Male   18      Mapple Street
```

حال در همین حالت به جای متدهای next() و absolute() در قسمت شرط متدهای relative() را به صورت زیر نوشتیم و کد را اجرا کنید:

```
result.absolute(7);

while (result.relative(3))
{
    //some code
}

10   Oscar Manning Male   18      Mapple Street
```

در کد بالا نشانگر در سطر ۷ قرار دارد. وقتی که ما متدهای relative() را به صورت بالا فراخوانی می‌کنیم، عدد ۳ یعنی نشانگر سه سطر پرشنی داشت. در نتیجه نشانگر روی سطر ۱۰ قرار می‌گیرد. حال اگر همین عدد را منفی بدهیم، پرسن نشانگر رو به عقب خواهد بود:

```
result.absolute(7);

while (result.relative(-3))
{
    //some code
}

4    Joann Jordan Female 17      Spencer Street
1    Edward Lyons  Male   17      Spencer Street
```

ResultSet به روز رسانی، حذف و اضافه کردن سطرهای

همانطور که قبلًا هم اشاره کردیم، برای بروز رسانی مقادیر موجود در ستون‌های یک سطر از ResultSet، از متدهایی که با کلمه update شروع می‌شوند استفاده می‌شود. فرض کنید که می‌خواهیم نام Monica که در سطر سوم قرار دارد را به Zhila تغییر دهیم. برای این کار

به صورت زیر عمل می‌کنید :

```

1 package myfirstprogram;
2
3 import java.sql.*;
4
5 public class MyFirstProgram
6 {
7     public static void main(String[] args)
8     {
9         try
10        {
11            DriverManager.registerDriver(new com.mysql.jdbc.Driver());
12
13            String url = "jdbc:mysql://localhost/university?user=root&password=";
14            Connection connection = DriverManager.getConnection(url);
15
16            Statement statement1 = connection.createStatement(
17                                ResultSet.TYPE_FORWARD_ONLY,
18                                ResultSet.CONCUR_UPDATABLE
19                                );
20            ResultSet result = statement1.executeQuery("SELECT * FROM Students");
21
22            if (result.getConcurrency() == ResultSet.CONCUR_UPDATABLE)
23            {
24                result.absolute(3);
25                result.updateString("FirstName", "Zhila");
26                result.updateRow();
27            }
28            else
29            {
30                System.out.println("ResultSet non-updatable.");
31                return;
32            }
33
34            result.beforeFirst();
35
36            while (result.next())
37            {
38                System.out.println(result.getString("FirstName"));
39            }
40
41            result.close();
42            statement1.close();
43            connection.close();
44        }
45        catch (Exception ex)
46        {
47            System.out.println(ex.getMessage());
48        }
49    }
50 }
```

```
Jimmie
Zhila
Joann
Cheryl
Clara
Zack
Randall
Jessica
Oscar
```

درباره خطوط ۱۱-۲۰ کد بالا توضیح نمی‌دهیم چون قبلاً درباره آن‌ها صحبت کردہ‌ایم. تنها نکته موجود در این خطوط مربوط به خط ۱۸ است که در آنجا به برنامه اعلام کرده‌ایم که می‌خواهیم یک ResultSet ایجاد کنیم که بتوانیم مقادیر موجود در آن را به روز رسانی یا ویرایش کنیم. در خط ۲۲ برای اطمینان از قابل ویرایش بودن ResultSet با استفاده از متدهای getConcurrency() چک می‌کنیم که آیا ResultSet قابل بروزرسانی است است یا نه؟ اگر بود، ابتدا با استفاده از متدهای absolute()، نشانگر را به سطر ۳ می‌بریم و سپس با استفاده از متدهای updateString() می‌گوییم که قرار است مقدار ستون FirstName را به Zhila تغییر دهیم. سپس با فراخوانی متدهای updateRow() و updateRow() تغییرات نهایی را به بانک اعمال می‌کنیم. در غیر اینصورت یعنی اگر ResultSet قابل ویرایش نبود در قسمت (خطوط ۲۸-۳۲) پیغام non-updatable را به کاربر نمایش داده و با استفاده از دستور return از اجرای سایر کدها جلوگیری می‌کنیم. با فرض اینکه ستون مورد نظر ما ویرایش شده است، برای نمایش این ستون و مشاهده تغییرات، ابتدا با استفاده از متدهای beforeFirst()، نشانگر را به ابتدای سطرهای ResultSet آورده و سپس با استفاده از یک دستور while که قبلاً توضیح داده شد، مقادیر را نمایش می‌دهیم (خطوط ۳۴-۳۹). مشاهده می‌کنید که مقدار ستون اول از سطر سوم تغییر کرده است. درباره خطوط ۴۱ در درس آینده توضیح می‌دهیم. برای اضافه کردن سطر به ResultSet از متدهای insertRow() و moveToInsertRow() استفاده می‌شود. خطوط ۲۴-۲۶ کد بالا را پاک کرده و کدهای زیر را به جای آن‌ها بنویسید :

```
result.moveToInsertRow();

result.updateString("FirstName", "John");
result.updateString("LastName", "Scith");
result.updateString("Gender", "Male");
result.updateInt("Age", 25);
result.updateString("Address", "Golden Street");

result.insertRow();
```

در کد بالا ابتدا با استفاده از متدهای moveToInsertRow() و insertRow() یک سطر خالی ایجاد و سپس مقادیری را در داخل هر یک از سلول‌های آن قرار می‌دهیم و در نهایت با فراخوانی متدهای deleteRow() آن را حذف کنیم. برای حذف یک سطر خاص هم کافیست که نشانگر را به محل سطر برد و با فراخوانی متدهای deleteRow() آن را حذف کنیم. مثلاً گذید زیر را به جای کدهای بالا بنویسید :

```
result.absolute(3);
result.deleteRow();
```

با اجرای برنامه سطر سوم از جدول اصلی حذف می‌شود.

پاک کردن اشیاء بی استفاده و آزاد کردن حافظه

همانطور که در درس‌های قبل مشاهده کردید، برای کار با رابطه‌ای `Connection`, `Statement` و `ResultSet` از آن‌ها اشیایی ایجاد کردیم. این اشیا به طور خودکار بعد از مدتی توسط زباله روب یا `Garbage Collector` از حافظه پاک می‌شوند ولی بهتر است که خودتان این کار را به صورت دستی و با فراخوانی متدهای `close()` و به صورت زیر انجام دهید.

```
package myfirstprogram;

import java.sql.*;
//Step 1

public class MyFirstProgram
{
    public static void main(String[] args)
    {
        try
        {
            //some code

            statement.close();
            result.close();
            connection.close();
        }
        catch (Exception ex)
        {
            System.out.println(ex.getMessage());
        }
    }
}
```

فراخوانی متدهای `close()` باعث حذف اشیاء ایجاد شده از رابطه‌ای مذکور و آزاد شدن حافظه می‌شود. روش بهتر برای پاک کردن اشیاء فراخوانی همین متدهای `finally` در بلوک `finally` می‌باشد. چون کدهای این بلوک، چه برنامه با خطا مواجه شود و چه نشود، در هر صورت اجرا می‌شوند:

```
package myfirstprogram;

import java.sql.*;

public class MyFirstProgram
```

```

{
    public static void main(String[] args) throws SQLException
    {
        Connection connection = null;
        Statement statement = null;
        ResultSet result = null;

        try
        {
            //some code
        }
        catch (Exception ex)
        {
            System.out.println(ex.getMessage());
        }
        finally
        {
            statement.close();
            result.close();
            connection.close();
        }
    }
}

```

همانطور که در کد بالا مشاهده می‌کنید، باید از رابط‌ها در خارج از دستور try catch شیئ ایجاد کرد، تا این اشیاء در بلوک finally باشند. همچنین در جلوی متده main() هم باید عبارت throws SQLException را بنویسیم، چون که استفاده از متده close() ممکن است باعث ایجاد استثناء شود. کد بالا را به صورت زیر هم می‌توان نوشت :

```

package myfirstprogram;

import java.sql.*;
//Step 1

public class MyFirstProgram
{
    public static void main(String[] args) throws SQLException
    {
        Connection connection = null;
        Statement statement = null;
        ResultSet result = null;

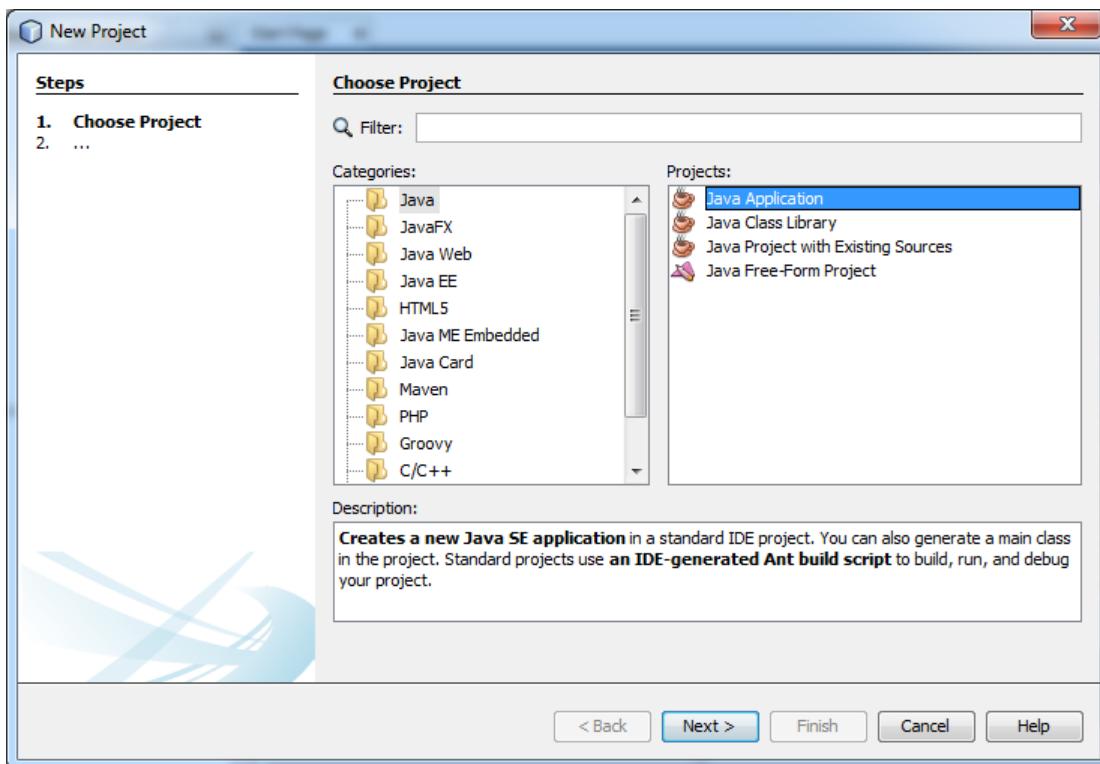
        try
        {
            //some code
        }
        catch (Exception ex)
        {
            System.out.println(ex.getMessage());
        }
        finally
        {
            try
            {

```

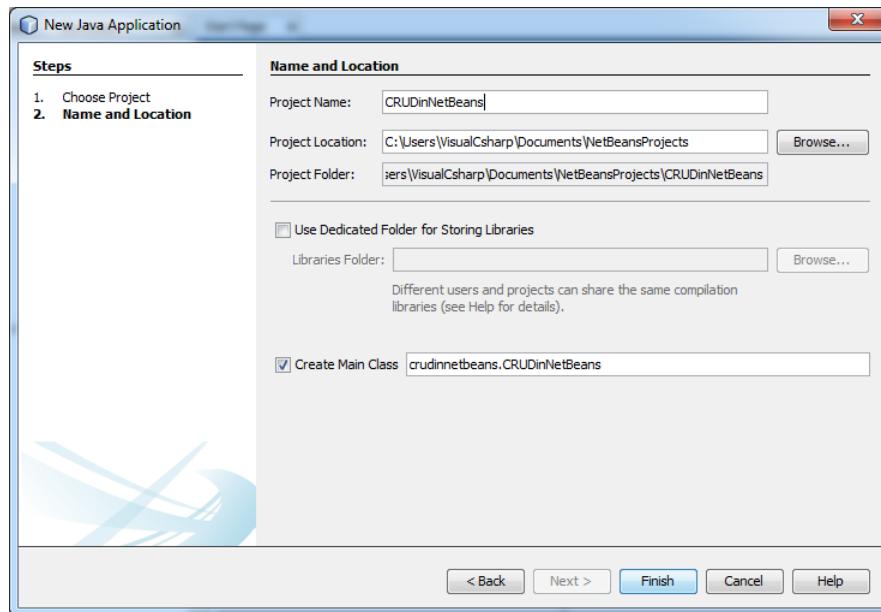
```
        statement.close();
        result.close();
        connection.close();
    }
    catch (SQLException ex)
    {
        System.out.println(ex.getMessage());
    }
}
```

ثبت، حذف، ویرایش و انتخاب اطلاعات با استفاده از NetBeans

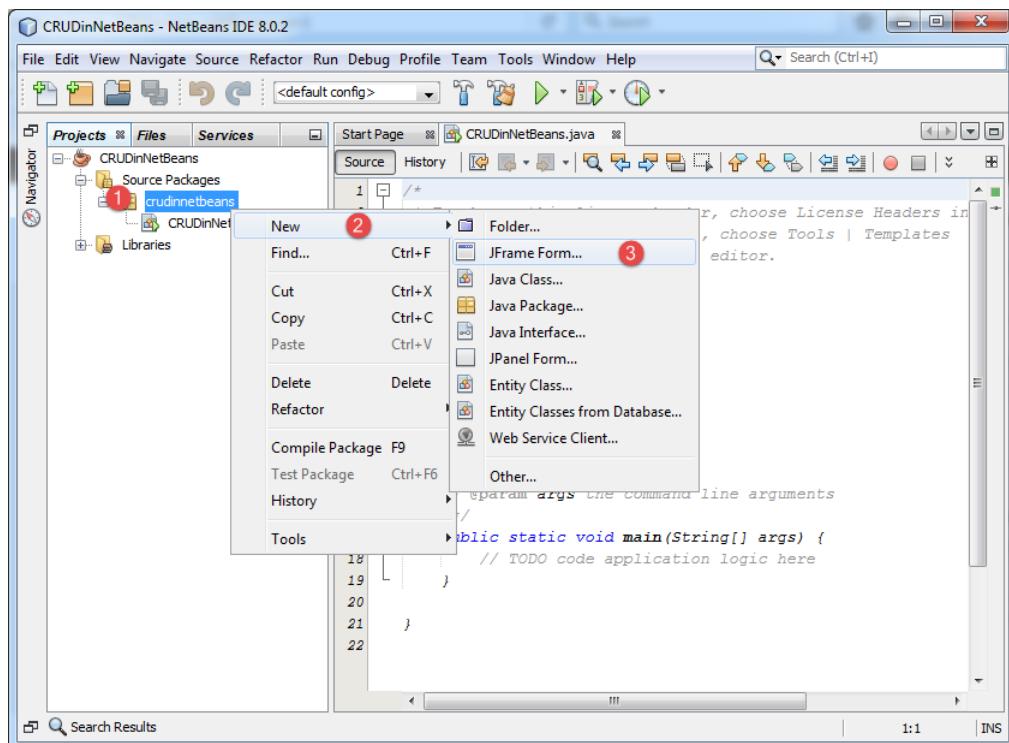
در این درس می‌خواهیم سیستم ساده مدیریت دانشجو را طراحی کنیم. در واقع تمام مطالبی که تاکنون گفته شده را در یک مثال و در محیط بصری NetBeans توضیح می‌دهیم. در درس‌های قبلی یک دیتابیس به نام University و یک جدول با نام Students در داخل آن ایجاد کردیم. این جدول دارای یک ستون به نام StudentID می‌باشد. در این مثال تمرکز اصلی بر روی این فیلد است، چون که برای هر دانشجو دارای یک مقدار منحصر به فرد است و می‌توان بر اساس آن عملیات ثبت، حذف، درج و ویرایش اطلاعات را انجام داد. برنامه NetBeans را باز کرده و به صورت زیر یک پروژه جدید به ایجاد می‌کنیم:



بر روی دکمه Next کلیک کرده و نام پروژه را CRUDinNetBeans بگذارید :

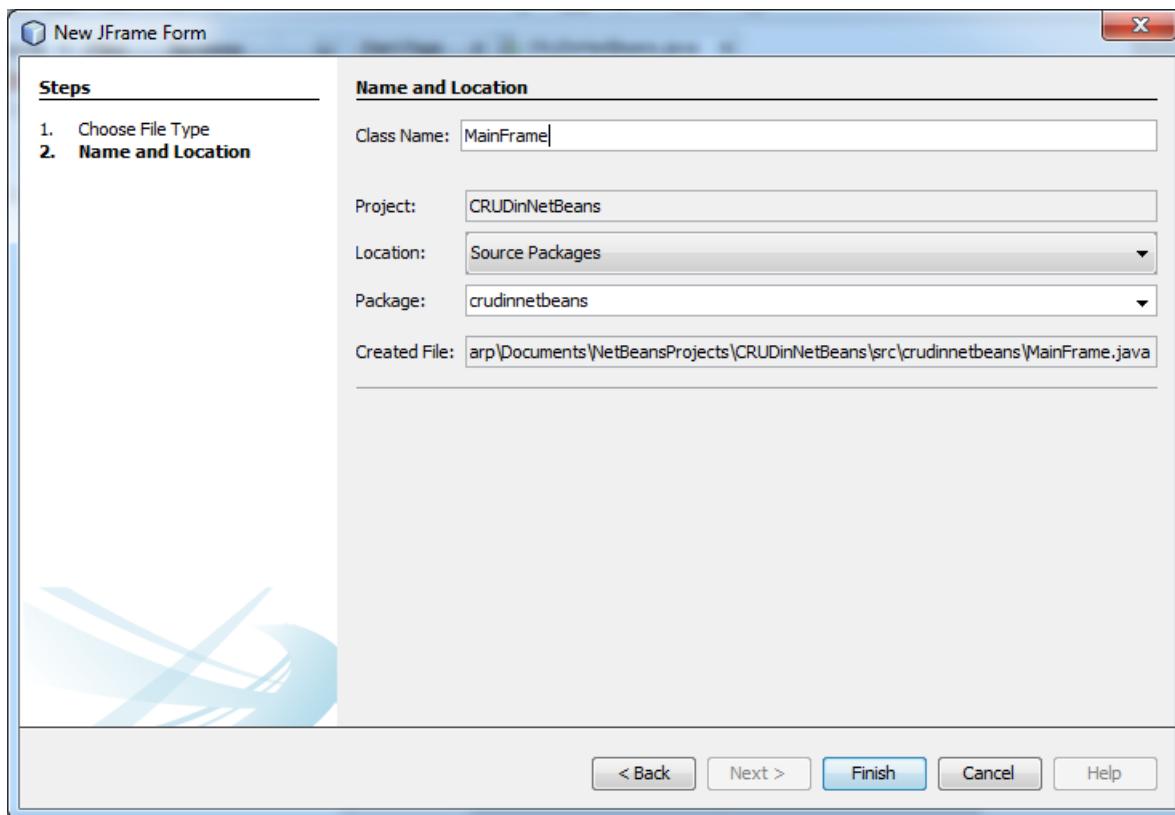


بر روی دکمه finish کلیک کرده تا وارد پروژه شوید. در صفحه باز شده به صورت زیر یک Frame جدید به برنامه اضافه کنید :



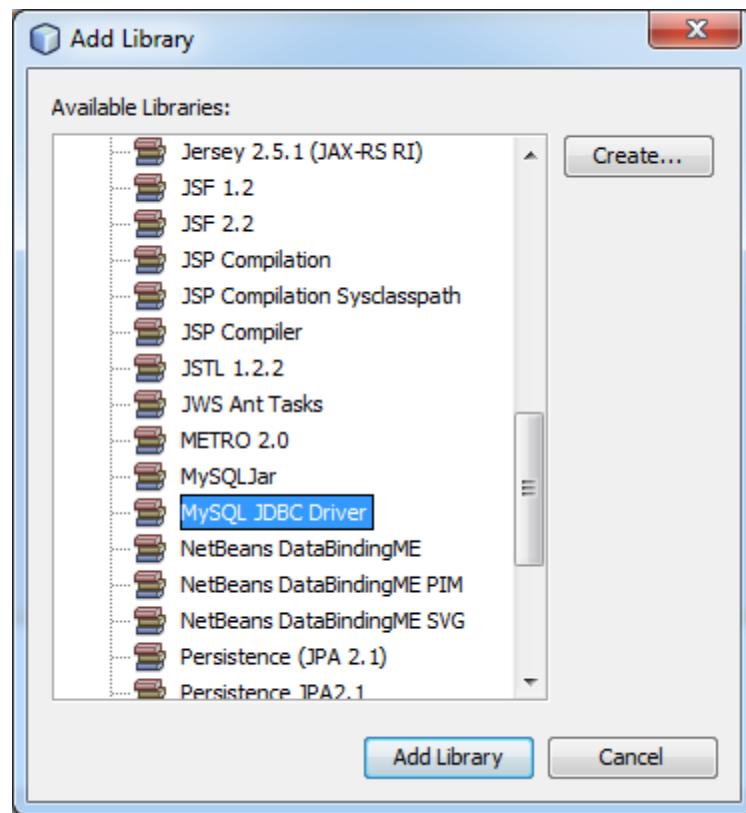
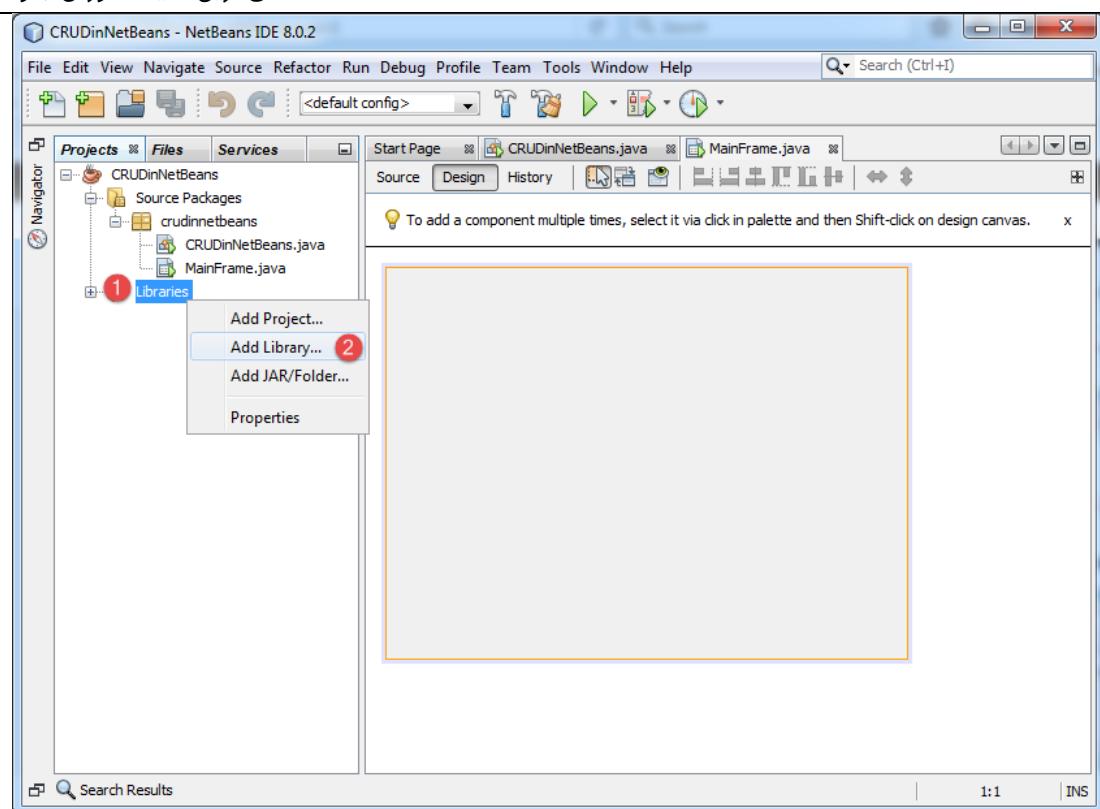
بعد از زدن بر روی گزینه Form JFrame صفحه‌ای به صورت زیر باز می‌شود که شما می‌توانید با استفاده از آن یک نام برای Frame انتخاب

کنید، که ما در این مثال نام آن را MainFrame می‌گذاریم و بر روی دکمه Finish کلیک می‌کنیم:

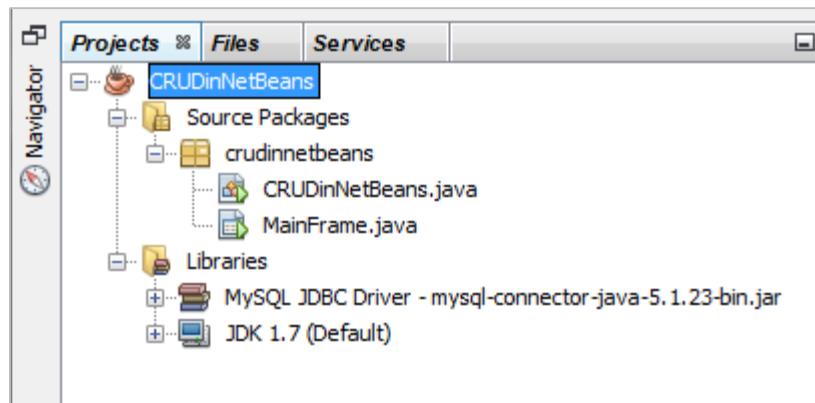


چون در این مثال می‌خواهیم از بانک MySQL استفاده کنیم پس باید Connector این بانک راه به برنامه اضافه کنیم. برای این منظور به

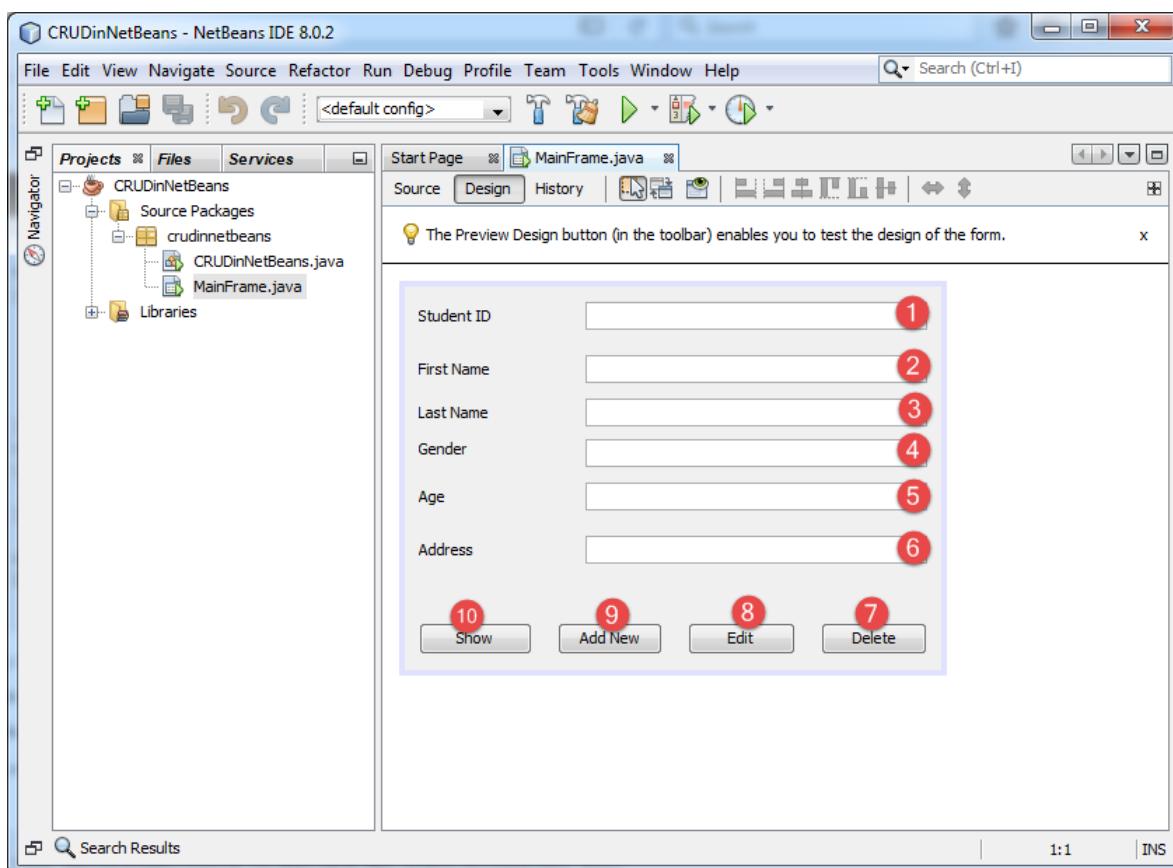
صورت زیر بر روی Libraries کلیک راست کرده و Connector را به برنامه اضافه می‌کنیم :



بعد از اتمام مراحل فوق، شکل نهایی پروژه به صورت زیر می‌شود:



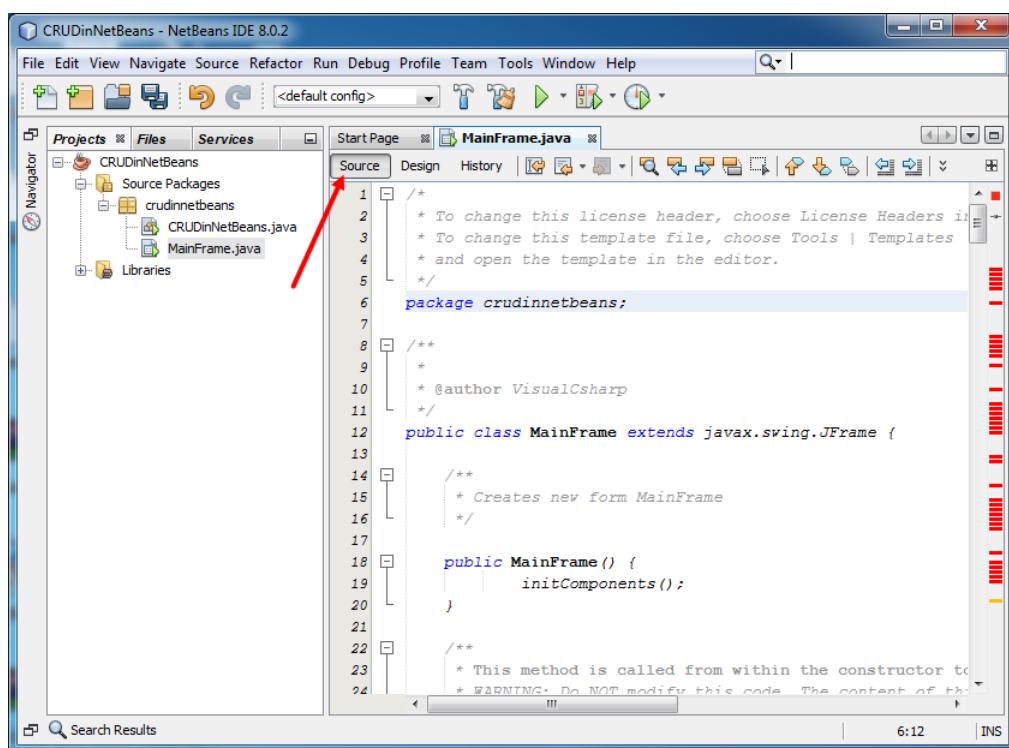
حال در حالی که سربرگ MainFrame.java فعال است، چند کنترل JTextField و JButton بر روی Frame قرار دهید:



سپس طبق جدول زیر خاصیت‌های این کنترل‌ها را تنظیم کنید:

Label	Variable Name	Properties
1	textBoxStudentId	
2	textBoxFirstName	enabled: False
3	textBoxLastName	enabled: False
4	textBoxGender	enabled: False
5	textBoxAge	enabled: False
6	textBoxAddress	enabled: False
7	buttonDelete	label: Delete
8	buttonEdit	label: Edit
9	buttonAddNew	label: Add New
10	buttonShow	label: Show

بعد از تنظیم خواص کنترل‌ها بر روی دکمه Source کلیک کرده تا وارد محیط کدنویسی شوید :



بعد از ورود به محیط کدنویسی، کدهای زیر را به برنامه اضافه کنید :

```

1 package crudinnetbeans;
2
3 import java.sql.*;
4 import javax.swing.*;
5
6 public class MainFrame extends javax.swing.JFrame
7 {
8     Connection connection = null;
9     PreparedStatement statement = null;
10    ResultSet result = null;
11
12    private boolean updateReady = false;
13    private boolean insertReady = false;
14
15    public MainFrame()
16    {
17        try
18        {
19            initComponents();
20
21            DriverManager.registerDriver(new com.mysql.jdbc.Driver());
22            String url = "jdbc:mysql://localhost:3306/University?user=root&password=";
23            connection=DriverManager.getConnection(url);
24        }
25        catch (SQLException ex)
26        {
27            System.out.println(ex.getMessage());
28        }
29    }
30}

```

به یک نکته خیلی مهم توجه کنید و آن این است که، در کد بالا برخی از خطوط مانند خطوط ۱، ۶ و ۱۵ از قبل توسط NetBeans تولید شده‌اند و شما باید بقیه خطوط را بنویسید. در خطوط ۳ و ۴ کد بالا از آنچاییکه ما با کنترل‌های Swing و بانک اطلاعاتی سر و کار داریم، این دو پکیج را وارد برنامه کرده‌ایم. برای جلوگیری از تکرار کدنویسی، کلاس‌ها یا متغیرهایی را که در طول برنامه با آن‌ها زیاد سر و کار داریم، در خطوط ۸-۱۳ تعریف کرده‌ایم. درباره متغیرهای خطوط ۱۲ و ۱۳ در ادامه توضیح می‌دهیم. در این مثال دوست داریم که هنگامی که برنامه اجرا می‌شود ارتباط با بانک برقرار شود. برای اینکه به محض اجرای برنامه ارتباط با بانک برقرار شود، در بدنه سازنده فرم خطوط ۱۷-۲۸ را می‌نویسیم. درباره این کدها در درس‌های قبل توضیح داده‌ایم.

انتخاب اطلاعات

برای انتخاب یا نمایش اطلاعات از دکمه Show استفاده می‌کنیم. هدف از ایجاد این دکمه این است که کاربر با وارد کردن یک ID در جعبه متن StudentID و زدن این دکمه، اطلاعات مربوط به یک دانشجو را نمایش دهد. بر روی دکمه Show دو بار کلیک کرده و در رویداد ActionPerformed آن کدهای زیر را بنویسید :

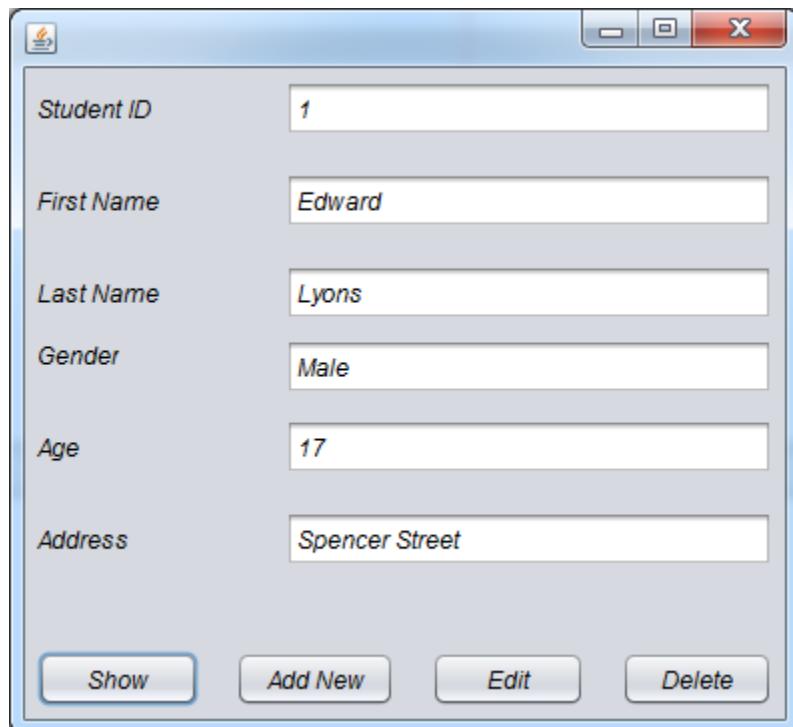
```

1  private void buttonShowActionPerformed(java.awt.event.ActionEvent evt)
2  {
3      try
4      {
5          String SQL = "SELECT * FROM Students WHERE StudentID = ?";
6
7          statement = connection.prepareStatement(SQL);
8          statement.setInt(1, Integer.parseInt(textBoxStudentID.getText()));
9
10         result = statement.executeQuery();
11
12         if(result.next())
13         {
14             textBoxFirstName.setText(result.getString("FirstName"));
15             textBoxLastName.setText(result.getString("LastName"));
16             textBoxGender.setText(result.getString("Gender"));
17             textBoxAge.setText(result.getString("Age"));
18             textBoxAddress.setText(result.getString("Address"));
19         }
20         else
21         {
22             JOptionPane.showMessageDialog(null, "StudentID not found!");
23         }
24     }
25     catch (NumberFormatException | SQLException ex)
26     {
27         JOptionPane.showMessageDialog(null, "Please Enter a number or check your
28 connection");
29     }
}

```

در خط ۵ ابتدا کد SQL مورد نظرمان را می‌نویسیم. در این خط می‌گوییم که دانشجویی انتخاب شود که آن برابر عددی باشد که در خط ۸ و از طریق جعبه متن وارد شده است. نکته‌ای که در خط ۸ وجود دارد استفاده از متدها Integer.parseInt() برای تبدیل مقدار وارد شده از جعبه متن به عدد صحیح است. چون جعبه متن فقط رشته دریافت می‌کند پس باید آن را با استفاده از این متدها عدد تبدیل کنیم. استفاده از این متدها باعث بروز استثناء می‌شود چون ممکن است که کاربر اشتباهًا به جای عدد، حرف تایپ کند. پس در خط ۲۵ و در قسمت catch این استثناء (NumberFormatException) را اداره می‌کنیم. در خط ۱۰ دستور SQL را اجرا می‌کنیم و نتایج را در شیء ResultSet می‌ریزیم. اگر عدد وارد شده توسط کاربر با ID یکی از دانشجوها برابر باشد. در این صورت ResultSet دارای یک رکورد هست. پس شرط خط ۱۲ درست بود و وارد بدنه if می‌شویم. در آنجا مشخص می‌کنیم که مثلًاً ستون نام

(First Name) این رکورد در جعبه متن textBoxFirstName نمایش داده شود و و اگر عدد وارد شده توسط کاربر در بین ID دانشجوها نباشد پیغام StudentID not found نمایش داده می‌شود. حال اگر کلًّا یک مقدار غیر عددی توسط کاربر وارد شود و یا در ارتباط با بانک اختلالی به وجود آید قسمت catch اجرا می‌شود. به عنوان مثال عدد ۱ را وارد کرده و دکمه Show را بزنید و نتیجه را مشاهده کنید :



ثبت اطلاعات

کار بعدی که می‌کنیم اضافه کردن کد buttonAddNew است که کار آن اضافه کردن یا ثبت دانشجو است (متلاً ثبت نام دانشجو). بر

روی دکمه دو بار کلیک کرده و در رویداد actionPerformed آن کدهای زیر را بنویسید :

```

1  private void ClearFields()
2  {
3      textBoxFirstName.setText("");
4      textBoxLastName.setText("");
5      textBoxGender.setText("M");
6      textBoxAge.setText("");
7      textBoxAddress.setText("");
8  }
9
10 private int GetNextStudentID()
11 {
12     String sql = "SELECT MAX(StudentID) as StudentID FROM Students";
13     try

```

```

14     {
15         statement = connection.prepareStatement(sql);
16         statement.execute();
17         result = statement.getResultSet();
18
19         if (result.next())
20         {
21             int nextID = Integer.valueOf(result.getString("StudentID"));
22             return nextID + 1;
23         }
24     }
25     catch (SQLException ex)
26     {
27         JOptionPane.showMessageDialog(null, ex.getMessage());
28     }
29     return 0;
30 }
31
32 private void buttonAddNewActionPerformed(java.awt.event.ActionEvent evt)
33 {
34     if (!insertReady)
35     {
36         buttonAddNew.setText("Enroll");
37         ClearFields();
38         textBoxStudentID.setText(String.valueOf(GetNextStudentID()));
39         textBoxStudentID.setEnabled(false);
40         textBoxFirstName.setEnabled(true);
41         textBoxLastName.setEnabled(true);
42         textBoxGender.setEnabled(true);
43         textBoxAge.setEnabled(true);
44         textBoxAddress.setEnabled(true);
45         buttonShow.setEnabled(false);
46         buttonEdit.setEnabled(false);
47         buttonDelete.setEnabled(false);
48         insertReady = true;
49     }
50     else
51     {
52         String Query = "INSERT INTO Students (FirstName, LastName, Gender, Age, Address) VALUES
53                               (?, ?, ?, ?, ?) ";
54         try
55         {
56             buttonAddNew.setText("Add New");
57             statement = connection.prepareStatement(Query);
58
59             statement.setString(1, textBoxFirstName.getText());
60             statement.setString(2, textBoxLastName.getText());
61             statement.setString(3, textBoxGender.getText());
62             statement.setInt(4, Integer.parseInt(textBoxAge.getText().trim()));
63             statement.setString(5, textBoxAddress.getText());
64
65             int affectedRows = statement.executeUpdate();
66
67             if(affectedRows > 0)
68                 JOptionPane.showMessageDialog(null, "Student successfully enrolled.");
69             else
70                 JOptionPane.showMessageDialog(null, "Failed to enroll student.");
71         }
72         catch (NumberFormatException | SQLException ex)
73         {
74             JOptionPane.showMessageDialog(null, "Exception happened!");
75         }
76
77         textBoxStudentID.setEnabled(true);
78         textBoxFirstName.setEnabled(false);

```

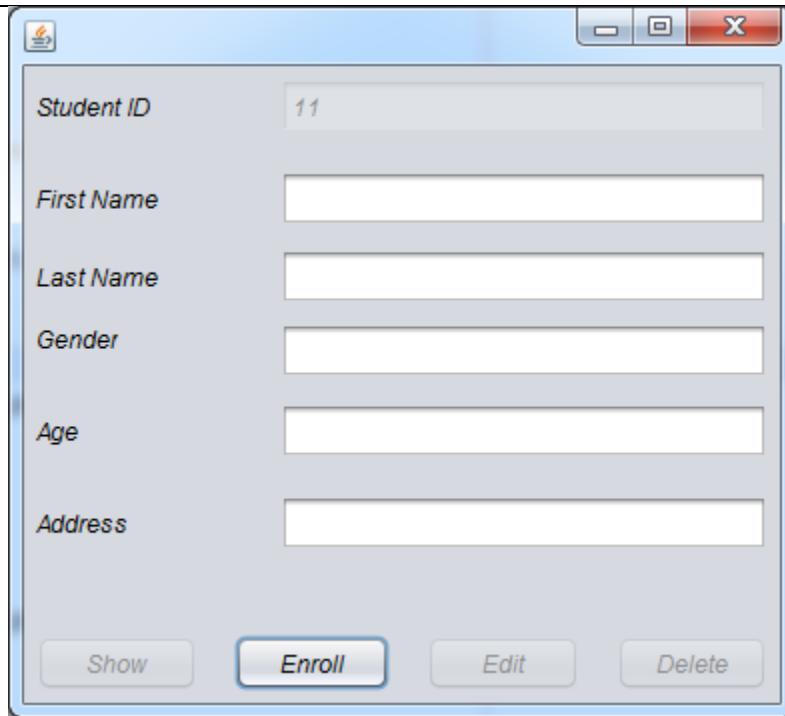
```

79   textBoxLastName.setEnabled(false);
80   textBoxGender.setEnabled(false);
81   textBoxAge.setEnabled(false);
82   textBoxAddress.setEnabled(false);
83   buttonShow.setEnabled(true);
84   buttonEdit.setEnabled(true);
85   buttonDelete.setEnabled(true);
86   insertReady = false;
87 }

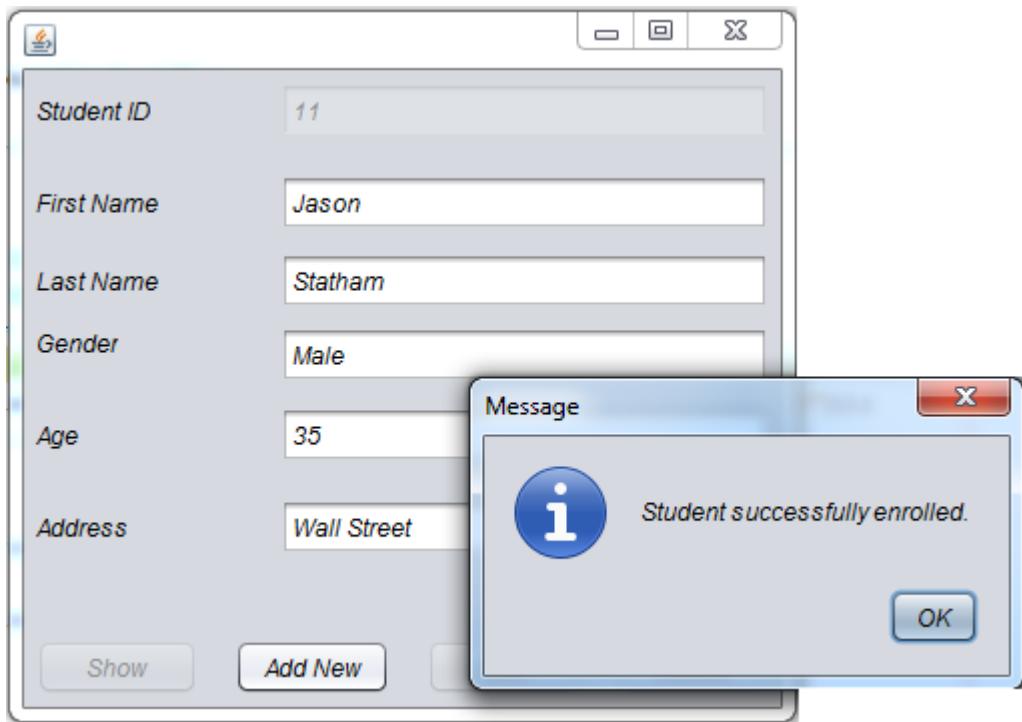
```

در کد بالا ابتدا دو متند، یکی برای پاک کردن محتویات جعبه‌های متن (ClearFields) و دیگری برای به دست آوردن ID آخرین رکورد ثبت شده در بانک (GetNextStudentID) تعریف می‌کنیم. در بدنه متند (ClearFields) یعنی خطوط ۳-۷ مشاهده می‌کنید که برای پاک کردن یک جعبه متن کافیست، یک رشته خالی را با استفاده از متند (setText) به جعبه متن اختصاص دهیم. در بدنه متند هم ابتدای یک دستور SQL می‌نویسیم که آخرین ID وارد شده در جدول را به ما برگرداند (خط ۱۲ MySQL). GetNextStudentID() یک تابع به نام (MAX) است که بزرگ‌ترین مقدار یک ستون عددی را بر می‌گرداند. در خط ۱۲ نام ستون StudentID را به این تابع می‌دهیم. در خطوط ۱۵-۱۷ دستور SQL را جدا و نتیجه را در شیء ResultSet می‌ریزیم. در خط ۱۹ هم چک می‌کنیم که آیا این شیء دارای رکوردي هست، اگر رکوردي داشت در داخل بدنه if مقدار ستون StudentID آن را در داخل یک متغیر به نام nextID می‌ریزیم. این متغیر در اصل بزرگ‌ترین یا آخرین ID موجود در بانک را در خود ذخیره می‌کند. ولی چون ما می‌خواهیم رکورد جدیدی اضافه کنیم پس باید ID رکورد جدید ۱ واحد بیشتر از آخرین ID موجود در بانک باشد. پس در خط ۲۲ یک واحد به این متغیر اضافه می‌کنیم. حال نوبت به کدهای موجود در بدنه کنترل کننده رویداد ActionPerformed می‌رسد. همانطور که در ابتدای درس مشاهده کردید ما یک فیلد با نام insertReady با مقدار false تعریف کردیم. با استفاده از این فیلد نحوه اجرای دستورات موجود در کنترل کننده رویداد true insertReady را کنترل می‌کنیم. ابتدادر خط ۳۴ مقدار موجود در شرط را با گذاشتن علامت! در کنار نام ActionPerformed تغییر می‌دهیم. این کار باعث می‌شود که وقتی برای اولین بار بر روی دکمه New کلیک شد، کدهای خطوط ۳۶-۴۸ اجرا شوند. در خط ۳۶ ابتدامتن روی دکمه را به Enroll (ثبت نام) تغییر می‌دهیم. در خط ۳۷ محتویات تمام جعبه‌های متن را با فراخوانی متند GetNextStudentID پاک می‌کنیم. در خط ۳۸ یک واحد بیشتر از آخرین ID موجود در بانک را با فراخونی متند (ClearFields) در جعبه متن textBoxStudentID قرار می‌دهیم. سپس دکمه‌ها و جعبه‌های متنی که برای کاربر کاربردی دارند را غیر فعال می‌کنیم.

شکل نهایی برنامه بعد از اولین کلیک به صورت زیر است :



کردن فیلد insertReady در خط ۴۸ باعث می‌شود که وقتی برای بار دوم بر روی دکمه New کلیک کنیم مقدار موجود در شرط خط ۳۴ برابر false شده و خطوط موجود در خطوط ۵۲-۸۵ اجرا شوند. اینکه در درس‌های قبل گفتیم که با استفاده از PrepareStatement می‌توان دستوراتی را اجرا کرد که در زمان اجرای برنامه تکمیل می‌شوند، را می‌توان در خطوط ۵۲-۶۲ به طور کامل درک کرد. در خط ۵۲ ابتدا یک دستور وارد کردن اطلاعات (INSERT) می‌نویسیم و به جای مقادیر علامت سؤال می‌گذاریم. این علامت سؤال‌ها به ترتیب توسط مقادیر که توسط کاربر وارد جعبه‌های متن می‌شوند پر می‌شوند. مثلًاً مقدار وارد شده در جعبه متن textBoxFirstName در اولین علامت سؤال، مقدار وارد شده در جعبه متن دوم در دومین علامت سؤال و الی آخر قرار می‌گیرد. در خط ۶۴ دستور را با فراخوانی متدها executeUpdate() اجرا می‌کنیم. این متدها در صورتیکه سطری به بانک اضافه شود مقدار را بر می‌گرداند. در نتیجه در خط ۶۶ چک می‌کنیم که آیا مقدار بزرگتر از ه است یا نه؟ اگر بزرگ‌تر بود به این معناست که سطری به بانک اضافه شده است و این اضافه شدن را با نشان دادن یک پیغام به کاربر اعلام می‌کنیم (خط ۶۷) :



بعد از اتمام کارها `insertReady` را `false` می‌کنیم که دفعه بعد قسمت `if` اجرا شود.

ویرایش اطلاعات

دکمه `Edit` هم مانند دکمه `Add New` کار می‌کند.

```

1  private void buttonEditActionPerformed(java.awt.event.ActionEvent evt)
2  {
3      if (!updateReady)
4      {
5          buttonEdit.setText("Update");
6          textBoxStudentID.setEnabled(false);
7          textBoxFirstName.setEnabled(true);
8          textBoxLastName.setEnabled(true);
9          textBoxGender.setEnabled(true);
10         textBoxAge.setEnabled(true);
11         textBoxAddress.setEnabled(true);
12         buttonShow.setEnabled(false);
13         buttonAddNew.setEnabled(false);
14         buttonDelete.setEnabled(false);
15         updateReady = true;
16     }
17     else
18     {
19         String Query =
20             "UPDATE Students SET FirstName=?, LastName=?, Gender=?, Age=?, Address=? WHERE StudentID = ?";
21         try
22         {

```

```

23     buttonEdit.setText("Edit");
24     statement = connection.prepareStatement(Query);
25
26     statement.setString(1, textBoxFirstName.getText());
27     statement.setString(2, textBoxLastName.getText());
28     statement.setString(3, textBoxGender.getText());
29     statement.setInt(4, Integer.parseInt(textBoxAge.getText().trim()));
30     statement.setString(5, textBoxAddress.getText());
31     statement.setInt(6, Integer.parseInt(textBoxStudentID.getText().trim()));
32
33     int affectedRows = statement.executeUpdate();
34
35     if(affectedRows>0)
36         JOptionPane.showMessageDialog(null, "Student details successfully updated.");
37     else
38         JOptionPane.showMessageDialog(null, "Update failed.");
39     }
40     catch (NumberFormatException | SQLException ex)
41     {
42         JOptionPane.showMessageDialog(null, "Exception happened!");
43     }
44
45     textBoxStudentID.setEnabled(true);
46     textBoxFirstName.setEnabled(false);
47     textBoxLastName.setEnabled(false);
48     textBoxGender.setEnabled(false);
49     textBoxAge.setEnabled(false);
50     textBoxAddress.setEnabled(false);
51     buttonShow.setEnabled(true);
52     buttonAddNew.setEnabled(true);
53     buttonDelete.setEnabled(true);
54     updateReady = false;
55 }
56 }
```

کار با این دکمه به این صورت است که ابتدا کاربر در جعبه متن textBoxStudentID یک عدد که همان ID رکوردی از بانک است که قرار است ویرایش شود را وارد می‌کند. سپس در داخل جعبه‌های متن مقادیر مورد نظر را وارد کرده و در نهایت با زدن دکمه تغییرات را به بانک ارسال می‌کند. وقتی برای اولین بار بر روی دکمه Edit کلیک می‌کنیم همه textbox‌ها فعال می‌شوند و تمام دکمه‌ها به جز دکمه Edit غیر فعال می‌شوند. با کمی توجه به کدهای بالا متوجه می‌شوید که کدنویسی این دکمه بسیار شبیه به دکمه Add New است و فقط تفاوت در دستور SQL است.

حذف اطلاعات

و در نهایت کد مربوط به پاک کردن دانشجو را می‌نویسی. دکمه Delete این کار را برای ما انجام می‌دهد. بر روی این دکمه دو بار کلیک کرده و کدهای زیر را بنویسید :

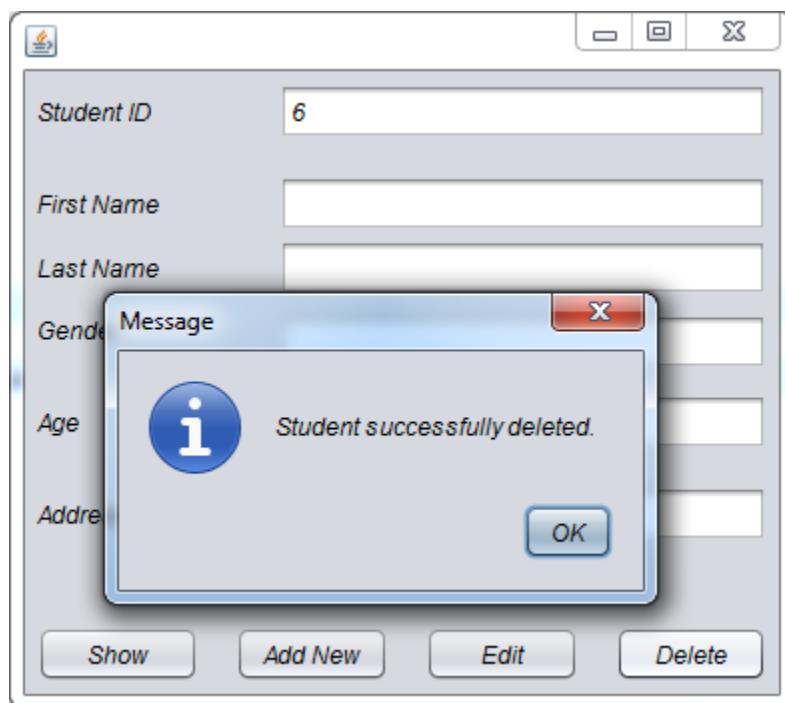
```

1  private void buttonDeleteActionPerformed(java.awt.event.ActionEvent evt)
2  {
3      String Query = "DELETE FROM Students WHERE StudentID = ?";
4      try
5      {
```

```

6   statement = connection.prepareStatement(Query);
7   statement.setInt(1, Integer.parseInt(textBoxStudentID.getText().trim()));
8
9   int affectedRows = statement.executeUpdate();
10
11  if(affectedRows > 0)
12      JOptionPane.showMessageDialog(null, "Student successfully deleted.");
13  else
14      JOptionPane.showMessageDialog(null, "Failed to delete student.");
15 }
16 catch (NumberFormatException | SQLException ex)
17 {
18     JOptionPane.showMessageDialog(null, "Exception happened!");
19 }
20 }
```

توضیح کد بالا هم تکراری است. تنها توضیح مربوط به خطوط ۳ و ۷ است. در خط ۳ یک دستور نوشته‌ایم که در آن گفته‌ایم سطری از جدول حذف شود که آن برابر عددی باشد که توسط کاربر در جعبه متن textBoxStudentID وارد شده است. یعنی علامت سؤال خط ۳ با مقدار گرفته شده از کاربر در خط ۷ جایگزین می‌شود. در نتیجه بعد از زدن دکمه Delete سطر مذکور حذف می‌شود :



برای مطالعه مطالب جدید جاوا به سایت w3-farsi.com مراجعه فرمایید.