

```
public class PingPongThread extends Thread {  
  
    private String word; // what word to print  
    private int delay; // how long to pause  
  
    public PingPongThread(String name,  
                          String whatToday,  
                          int delayTime) {  
        super(name);  
        word = whatToday;  
        delay = delayTime;  
    }  
    public void run() {  
        try {  
            while (true) {  
                System.out.print(getName() + ":" + word + " ");  
                Thread.sleep(delay); // wait until next time  
            }  
        } catch (InterruptedException e) {  
            return; // exit this thread  
        }  
    }  
  
    public static void main(String[] args) {  
        PingPongThread player1 =  
            new PingPongThread(name: "Player1", whatToday: "ping", delayTime: 33);  
        PingPongThread player2 =  
            new PingPongThread(name: "Player2", whatToday: "PONG", delayTime: 100);  
        player1.start(); // 1/30 second  
        player2.start(); // 1/10 second
```

برنامه نویسی به زبان جاوا

احمدرضا صدیقی

برنامه نویسی به زبان جاوا

مؤلف:

احمدرضا صدیقی

این کتاب با رضایت نویسنده و ناشر به صورت رایگان در اختیار همگان قرار گرفته است. هرگونه تغییر در کتاب یا دخل و تصرف در محتوای آن، انتشار تمام یا بخشی از آن (چه به صورت انتفاعی یا غیرانتفاعی) خلاف اخلاق و قانون است.

انتشار این کتاب در هر کanal یا وبسایتی، در همین قالب PDF و بدون افزودن هرگونه تبلیغ و معرفی محسولی آزاد است.
اگر از مطالعه کتاب راضی بودید، مبلغی را به صلاحیت خود به خیریه مورد اعتماد خود اهدا کنید.

موفق باشد
احمدرضا صدیقی

نام کتاب: برنامه نویسی به زبان جاوا

مؤلف: احمد رضا صدیقی

ناشر: کانون نشر علوم

طرح جلد: مائدۀ فکری

تیراژ: ۱۰۰۰ جلد

تقدیم به شهیدان راه وطن

هจده سال پیش درست زمانی که جاوا به تازگی در دنیا معرفی شده بود و خصوصیات جذاب و پرقابلیت این زبان نقل محافل بود، من و برخی دوستانم جزو اولین افرادی بودیم که توسعه بر پایه زبان جاوا را در ایران آغاز کردیم. در آن زمان، هیچ آموزشگاه یا معلمی که بتواند این زبان را به من بیاموزد وجود نداشت، حتی هیچ کتابی هم نبود، اینترنت هم به شکل امروزی کامل و غنی نبودا بنابراین یادگیری را با حداقل منابعی که در اختیار داشتم شروع کردم و این زبان را آموختم.

سالها بعد وقتی مدیریت فنی پروژه های جاوای را به عهده داشتم، کمیود نیروی برنامه نویس، پروژه هایم را تحت فشار قرار داده بود، تا اینکه ایده تالیف کتاب فارسی جاوا به ذهنم خطور کرد تا فارغ التحصیلان دانشگاه به سادگی و در کوتاه ترین زمان این زبان را بیاموزند وارد بازار کار شوند.

با هر رحمتی که بود یک مجموعه سه جلدی تالیف و چاپ کردم. از زمان چاپ اولین مجموعه کتابهای جاوای من زمان زیادی گذشته و در این مدت این کتابها بارها بروزرسانی و تجدید چاپ شده اند و هربار با استقبال بی نظیر شما دوستان مواجه شده اند. تا اینکه تصمیم گرفتم با توجه به تغییرات زیادی که در سالهای اخیر در حوزه نرم افزار رخ داده است محتوای هرسه کتاب را بازبینی کنم و مجموعه جدیدی شامل پنج جلد کتاب را تالیف کنم که جوابگوی نیازهای فعلی و آینده برنامه نویسان جاوا باشد. کتاب حاضر اولین کتاب از این مجموعه است و کتابهای بعدی به تدریج آماده و چاپ می شوند. همچنین در وبسایت اطلس سافت وضعیت چاپ کتابهای بعدی را اطلاع رسانی می کنم و در همانجا می توانید سورسهای این کتاب و کتابهای دیگر را دانلود کنید.

در طی این سالها ایمیلیهای تشکر و قدردانی زیادی از شما دوستان دریافت کرده ام که در اینجا لازم است از این همه لطف و محبت شما سپاسگزاری کنم. همانند گذشته می توانید سوالات، انتقادات یا پیشنهادات خود را از طریق ایمیل زیر با من در میان بگذارید.

ahmadseddighi@yahoo.com

واز طریق لینکدین اینجانب به آدرس زیر

<https://www.linkedin.com/in/ahmadseddighi/>

با من در ارتباط باشید. همچنین از شما تقاضامندم به جای ارسال ایمیل های تشکر و قدردانی که صرفاً توسط اینجانب قابل مشاهده است، نقطه نظرات خود را در لینکدین اینجانب به آدرس فوق، یا در سایت goodreads.com در صفحه مربوط به همین کتاب مطرح نمایید تا برای افراد دیگری که به دنبال کتاب و منابع جاوای هستند راهگشا باشد.

احمدرضا صدیقی

فهرست مطالب

۱	جاوا.....	جاوا.....
۱	تاریخچه جاوا.....	با جاوا چه نوع برنامه هایی می توان نوشت؟.....
۲	با جاوا چه نوع برنامه هایی می توان نوشت؟.....	در بازار کار بیشتر چه برنامه هایی تولید می شوند؟.....
۳	در بازار کار بیشتر چه برنامه هایی تولید می شوند؟.....	چگونه باید برنامه ریزی برای یادگیری جاوا را شروع کنم؟.....
۴	چگونه باید برنامه ریزی برای یادگیری جاوا را شروع کنم؟.....	بازار کار جاوا در ایران و خارج از ایران چگونه است؟.....
۵	بازار کار جاوا در ایران و خارج از ایران چگونه است؟.....	اندروید چیست؟.....
۶	اندروید چیست؟.....	برای یادگیری جاوا باید لینوکس بلد باشم؟.....
۷	برای یادگیری جاوا باید لینوکس بلد باشم؟.....	برای شروع برنامه نویسی جاوا به چه چیزهایی نیاز دارم؟.....
۸	برای شروع برنامه نویسی جاوا به چه چیزهایی نیاز دارم؟.....	کلاسهای آموزشی چقدر مفید هستند؟.....
۹	کلاسهای آموزشی چقدر مفید هستند؟.....	چه کتابهایی بخوانم؟.....
۱۰	چه کتابهایی بخوانم؟.....	آماده شدن برای برنامه نویسی
۱۱	آماده شدن برای برنامه نویسی	جاوا چگونه کار می کند؟.....
۱۲	جاوا چگونه کار می کند؟.....	خانواده جاوا.....
۱۳	خانواده جاوا.....	یادگیری زبان جاوا را از کجا شروع کنم؟.....
۱۴	یادگیری زبان جاوا را از کجا شروع کنم؟.....	بسترِ جاوا.....
۱۵	بسترِ جاوا.....	نصب Java SDK
۱۶	نصب Java SDK	دریافت، نصب و راه اندازی JDK
۱۷	دریافت، نصب و راه اندازی JDK	قرار دادن مسیر شاخه bin در PATH
۱۸	قرار دادن مسیر شاخه bin در PATH	تنظیم متغیر PATH در ویندوز
۱۹	تنظیم متغیر PATH در ویندوز	تنظیم متغیر PATH در لینوکس
۲۰	تنظیم متغیر PATH در لینوکس	شروع جاوا با یک برنامه ساده
۲۱	شروع جاوا با یک برنامه ساده	ذخیره و کامپایل برنامه
۲۲	ذخیره و کامپایل برنامه	اجرای برنامه
۲۳	اجرای برنامه	ورودی و خروجی استاندارد
۲۴	ورودی و خروجی استاندارد	ورود یک رشته به برنامه از طریق ورودی استاندارد

۱۸ System.out.print()
۱۸	پیاده سازی و فراخوانی یک متده جدید
۱۹	استفاده از یک متغیر عمومی (property) بین چندین متده
۲۰ استفاده از System.out.printf()
۲۱	پیاده سازی یک کلاس جدید
۲۱	استفاده از عملگرهای محاسباتی
۲۲	مثالی دیگر
۲۲ پیاده سازی کلاس Mathematics
۳۰	متده
۳۳	استفاده از یک کلاس دیگر
۳۴ استفاده از متدهای کلاس Math
۳۵	تولید اعداد تا سی
۳۶	محاسبه قطر مستطیل
۳۷	متدهای تودر تو
۳۸ استفاده از پارامترهای متده main()
۴۱	متعلقات (property)
۴۴	خلاصه
۴۵	تمرینات
۴۷	آشنایی با محیط برنامه نویسی IntelliJ IDEA
۴۷	نصب IntelliJ IDEA
۴۸	اجرای IntelliJ
۴۹	شروع کار با IntelliJ
۴۹	مفاهیم پایه در برنامه نویسی جاوا
۴۹	انواع داده ها
۵۱	متغیرها
۵۲	محدوده قابل مشاهده یک متغیر
۵۳	محدوده یک بلوك
۵۴	محدوده یک متده
۵۵	محدوده یک کلاس
۵۶	نامگذاری متغیرها
۵۷	عملگرهای انتساب
۵۸	عملگرهای محاسباتی

۱۸	عملگرهای بیتی
۱۹	عملگرهای مقایسه ای
۲۰	عملگرهای منطقی
۲۱	عملگر ?
۲۲	اولویت بندی عملگرها
۲۳	ساختارهای کنترلی
۲۴	ساختار کنترلی if
۲۵	ساختار if-else
۲۶	ساختار if-else چندگانه
۲۷	ساختار switch
۲۸	ساختار while
۲۹	ساختار do-while
۳۰	ساختار for
۳۱	کنترل اجرای حلقه ها با استفاده از break و continue
۳۲	حلقه بی نهایت
۳۳	کلمه کلیدی return
۳۴	آرایه ها
۳۵	معرفی و ایجاد آرایه
۳۶	دسترسی به عناصر آرایه و مقداردهی آنها
۳۷	آرایه های دو بعدی
۳۸	آرایه های چند بعدی
۳۹	پارامترهای متغیر
۴۰	خلاصه
۴۱	آشنایی با مفاهیم شی گرایی
۴۲	استفاده از کلاس برای تعریف یک داده جدید
۴۳	افزودن متدهای کلاس
۴۴	حذف دسترسی خارجی به داده های کلاس
۴۵	افزودن متدهای setter
۴۶	«سازنده» یا «constructor»
۴۷	تعریف یک داده دیگر
۴۸	متدهای toString()
۴۹	متدهای equals()
۵۰	خلاصه

۱۶۱	شی گرایی
۱۶۱	شی گرایی چیست؟
۱۶۲	اشیاء و کلاس
۱۶۳	رابطه کلاس و شی
۱۶۷	طراحی کلاس
۱۶۸	تعریف کلاس
۱۶۹	پیاده سازی کلاس
۱۷۱	ایجاد آبجکت
۱۷۳	مقداردهی فیلدهای آبجکت
۱۷۴	پیاده سازی کلاس مکعب
۱۷۴	اضافه کردن یک متدهای کلاس Box
۱۷۷	سازنده
۱۷۹	سازنده های دارای پارامتر
۱۸۰	کلمه کلیدی this
۱۸۲	Garbage Collection
۱۸۳	متدهای finalize()
۱۸۴	Overloading
۱۸۵	سازنده های overload شده
۱۸۷	استفاده از آبجکت به عنوان پارامتر
۱۹۰	نگاهی دقیقتر به پارامترهای متدها
۱۹۵	برگرداندن آبجکت توسط یک متند
۱۹۶	استفاده از پکیج برای دسته بندی کلاسهایها
۲۰۱	حق دسترسی به متدها و فیلدهای یک کلاس
۲۰۳	static
۲۰۵	تعریف فیلدهای final
۲۰۵	کلاسهای داخلی و تودرتو
۲۰۹	خلاصه
۲۱۲	تمرینات
۲۱۶	ارث بری
۲۱۹	حق دسترسی و ارث بری
۲۲۰	مثالی دیگر
۲۲۳	تبديل آبجکتها به یکدیگر
۲۲۵	استفاده از متدها و فیلدهای کلاس پدر در کلاس فرزند
۲۲۶	کلمه کلیدی super

۲۳۶	استفاده از <code>super</code> برای فراخوانی سازنده کلاس پدر
۲۳۷	استفاده از <code>super</code> برای فراخوانی متدها یا فیلد های کلاس پدر
۲۳۸	ایجاد کلاسهای سلسله مراتبی
۲۳۹	سازنده کلاس پدر چه زمانی فراخوانی می شود؟
۲۴۰	مثال کاربردی ۱
۲۴۱	Overriding
۲۴۲	کاربرد overriding
۲۴۳	کلاسهای abstract
۲۴۴	مثالی دیگر از abstract
۲۴۵	کلمه کلیدی final
۲۴۶	استفاده از final برای جلوگیری کردن از overriding
۲۴۷	استفاده از final برای جلوگیری از ارث بری
۲۴۸	کلاس Object
۲۴۹	خلاصه
۲۵۰	اینترفیس
۲۵۱	تعریف اینترفیس
۲۵۲	پیاده سازی اینترفیس
۲۵۳	پیاده سازی ناقص یک اینترفیس
۲۵۴	متغیرهایی که درون اینترفیس تعریف می شوند
۲۵۵	ارث بری در میان اینترفیس ها
۲۵۶	یک مثال کاربردی
۲۵۷	خلاصه
۲۵۸	تمرینات
۲۵۹	Enumeration
۲۶۰	یک مثال ساده
۲۶۱	تعریف enum
۲۶۲	تعریف مقادیر ثابت در enum ها
۲۶۳	چند نکته پایانی
۲۶۴	تمرینات
۲۶۵	کنترل خطا و استشنا
۲۶۶	استفاده از Exception در کنترل خطا
۲۶۷	توسعه کلاس Exception
۲۶۸	تولید بیش از یک استشنا در یک مت

۳۰۵ finally بلوک
۳۰۶ اینترفیس
۳۰۷ متد های کلاس Exception
۳۰۸ تولید خطاهای زنجیره ای
۳۰۹ متد getMessage()
۳۱۰ متد getCause()
۳۱۱ متد printStackTrace()
۳۱۲ استثنای «زمان اجرا» و «غیر زمان اجرا»
۳۱۳ خلاصه
۳۱۴ تمرینات
۳۱۵ پکیج های جاوا
۳۱۶ پکیج java.lang
۳۱۷ پکیج java.util
۳۱۸ پکیج java.nio و java.io
۳۱۹ پکیج java.lang
۳۲۰ کلاس های پوشش دهنده داده های نوع اولیه
۳۲۱ کلاس های Double و Float
۳۲۲ کلاس های Long و Integer
۳۲۳ کلاس Short
۳۲۴ کلاس Byte
۳۲۵ کلاس Character
۳۲۶ کلاس Boolean
۳۲۷ کلاس System
۳۲۸ متد long currentTimeMillis()
۳۲۹ متد void arrayCopy()
۳۳۰ استفاده از متد setProperty() و getProperty()
۳۳۱ کلاس Object
۳۳۲ اینترفیس Cloneable و استفاده از متد Clone()
۳۳۳ کلاس Class
۳۳۴ کلاس Math
۳۳۵ خلاصه
۳۳۶ تمرینات
۳۳۷ کاراکترها و رشته ها
۳۳۸ سازنده های کلاس String
۳۳۹ استفاده از کاراکتر های کنترلی

۳۵۷	متصل کردن رشته ها به یکدیگر.....
۳۵۸	ترکیب یک رشته با داده های دیگر.....
۳۵۹	متدهای کلاس String.....
۳۶۰	مقایسه کردن دو String.....
۳۶۱	انطباق ابتدا یا انتهای رشته با رشته دیگر.....
۳۶۲	مکان یابی کاراکترهای داخل String.....
۳۶۳	استخراج رشته های زیر مجموعه یک رشته.....
۳۶۴	اتصال دو String.....
۳۶۵	متدهای دیگر String.....
۳۶۶	کلاس StringBuffer.....
۳۶۷	متدهای دیگر StringBuffer.....
۳۶۸	خلاصه.....
۳۶۹	تمرینات.....

۳۷۰ ساختمان داده ها و پکیج java.util

۳۷۱	ساختمان داده ها.....
۳۷۲	اینترفیس های Collection.....
۳۷۳	اینترفیس Collection.....
۳۷۴	اینترفیس List.....
۳۷۵	اینترفیس Set.....
۳۷۶	اینترفیس SortedSet.....
۳۷۷	اینترفیس Queue.....
۳۷۸	کلاس های Collection.....
۳۷۹	کلاس ArrayList.....
۳۸۰	کلاس LinkedList.....
۳۸۱	کلاس HashSet.....
۳۸۲	اینترفیس Iterator.....
۳۸۳	حلقه for.....
۳۸۴	Map.....
۳۸۵	اینترفیس های Map.....
۳۸۶	اینترفیس Map.....
۳۸۷	اینترفیس SortedMap.....
۳۸۸	اینترفیس Map.Entry.....
۳۸۹	کلاس های Map.....
۳۹۰	کلاس HashMap.....
۳۹۱	کلاس Arrays.....

FIV	کلاسها و اینترفیس های قدیمی در پکیج <code>java.util</code>
FIV	اینترفیس <code>Enumeration</code>
FIV	کلاس <code>Vector</code>
FIV	کلاس <code>Stack</code>
FIV	کلاس <code>Hashtable</code>
FIV	کلاس <code>Properties</code>
FIV	کلاس <code>Date</code>
FIV	کلاس <code>Calendar</code>
FIV	کلاس <code>Random</code>
FIV	خلاصه
FIV	تمرینات

FVI	Generics
FVI	نام گذاری پارامترهای <code>Generic</code>
FVI	متدها و سازنده های <code>Generic</code>
FVI	استفاده از نشانه های <code>? extends</code> و <code>super</code>
FVI	ارث بری
FVI	تمرینات

FVII	ورودی و خروجی
FVII	<code>File</code>
FVII	<code>Stream</code>
FVII	کلاسها و اینترفیس های ورودی و خروجی داده ها در جاوا
FVII	کلاس های <code>Byte Stream</code>
FVII	کلاس <code>InputStream</code>
FVII	کلاس <code>OutputStream</code>
FVII	<code>FileInputStream</code>
FVII	<code> FileOutputStream</code>
FVII	کلاس <code>ByteArrayInputStream</code>
FVII	کلاس <code>ByteArrayOutputStream</code>
FVII	کلاس های <code>Buffered Byte Streams</code>
FVII	<code>BufferedInputStream</code>
FVII	<code>BufferedOutputStream</code>
FVII	<code>DataOutputStream</code> و <code>DataInputStream</code>
FVII	کلاس های <code>Character Stream</code>
FVII	<code>Reader</code>

F9۳	Writer
F9۴	FileReader
F9۵	FileWriter
F9۶	CharArrayReader
F9۷	CharArrayWriter
۸۰۱	BufferedReader
۸۰۲	BufferedWriter
۸۰۳	استفاده از ورودی و خروجی Stream
۸۰۴	Serialization
۸۰۵	اینترفیس Serializable
۸۰۶	کلاس Scanner
۸۰۷	خلاصه
۸۰۸	تمرینات
۸۱۰	Thread ها و همزمانی
۸۱۱	پیاده سازی و اجرای Thread
۸۱۲	کلاس Thread
۸۱۳	باری پینگ پنگ
۸۱۴	اصلی Thread
۸۱۵	همزمانی (Synchronization)
۸۱۶	بلوک synchronized
۸۱۷	مدیریت ارتباط بین Thread ها
۸۱۸	اجرا کننده ها
۸۱۹	اینترفیس Callable
۸۲۰	اینترفیس Future
۸۲۱	کلاس های TimerTask و Timer
۸۲۲	خلاصه
۸۲۳	تمرینات
۸۲۴	ارتباط با پایگاه داده
۸۲۵	SQL
۸۲۶	Driver
۸۲۷	JDBC Driver
۸۲۸	JDBC API
۸۲۹	نصب PostgreSQL
۸۳۰	آشنایی با SQL

۵۵۳	ارتباط با پایگاه داده.....
۵۷۱	جستجو در پایگاه داده با دستور select
۵۷۴	مدیریت تراکنش
۵۷۸	کلاس Entity
۵۷۹	خلاصه
۵۸۱	تمرینات

۵۷۳ عبارتهای باقاعده

۵۷۴	ساده ترین عبارت باقاعده
۵۷۵	کاراکترهای خاص
۵۷۶	محدوده کاراکترها
۵۷۷	میانبر برخی محدوده های پر استفاده
۵۷۸	تکرار
۵۷۹	انطباق رشته های خالی
۵۸۰	گروه بندی کاراکترها
۵۸۱	ارجاع به عقب
۵۸۲	انطباق در نقاط مرزی
۵۸۳	پردازش عبارت باقاعده در جاوا
۵۸۴	ایجاد آبجکت Pattern
۵۸۵	کلاس Pattern
۵۸۶	متدهای split(String input)
۵۸۷	متدهای split(String intput, int limit)
۵۸۸	متدهای matcher(String input)
۵۸۹	برخی متدهای کلاس String
۵۹۰	کلاس Matcher
۵۹۱	متدهای معادل در کلاس java.lang.String
۵۹۲	کلاس PatternSyntaxException
۵۹۳	خلاصه
۵۹۴	Annotation
۷۰۰	یک مثال ساده از Annotation
۷۰۳	انواع Annotation
۷۰۵	چند نکته در تعریف Annotation
۷۰۶	عناصر Annotation
۷۰۷	محدود کردن محل اعمال annotation
۷۰۹	سیاست های ماندگاری

۷۱۰ کار کردن با annotationها
۷۱۲ خلاصه
۷۱۳ تمرینات
۷۱۴ واسط کاربری
۷۱۵ JFrame
۷۱۶ Layout، چیدمان نمایش اجزا
۷۱۷ یک مثال کاربردی
۷۱۸ چندین LayoutManager دیگر
۷۱۹ منوها
۷۲۰ خلاصه
۷۲۱
۷۲۲ Logging
۷۲۳ Logging چیست؟
۷۲۴ JDK Logging
۷۲۵ Logger آبجکت
۷۲۶ سطح لاغ
۷۲۷ Handler آبجکت
۷۲۸ الگوی نامگذاری فایل لاغ
۷۲۹ Formatter آبجکت
۷۳۰ LogManager آبجکت
۷۳۱ تمرینات
۷۳۲ عبارتهای لامبدا و کلاس‌های تو در تو
۷۳۳ موارد کاربرد کلاس داخلی
۷۳۴ کلاس داخلی استاتیک
۷۳۵ کلاس غیراستاتیک داخلی
۷۳۶ یک مثال
۷۳۷ کلاس‌های محلی
۷۳۸ دسترسی به اعضای کلاس بیرونی
۷۳۹ شباهت کلاس محلی به کلاس داخلی غیراستاتیک
۷۴۰ کلاس‌های بی نام (Anonymous)
۷۴۱ Listener
۷۴۲ عبارتهای لامبدا (Lambda)
۷۴۳ اینترفیس functional
۷۴۴ کلاس Stream

۷۹۱	متدهای پیش فرض و متدهای استاتیک
۷۹۲	قواعد متدهای پیش فرض
۷۹۳	متدهای استاتیک
۷۹۴	تمرینات
۷۹۵	Reflection
۷۹۶	کلاس
۷۹۷	وآکاوی جزییات کلاس با استفاده از آبجکت Class
۷۰۰	دسترسی به اجزای داخلی کلاسها (فیلدها، متدها و سازنده ها).
۷۰۱	مثال کاربردی
۷۰۵	تمرینات
۷۰۷	چندزبانی (Internationalization)
۷۱۰	مرحله اول، تولید فایلهای ResourceBundle
۷۱۱	مرحله دوم، ایجاد آبجکت Locale
۷۱۲	مرحله سوم، ایجاد آبجکت ResourceBundle
۷۱۳	مرحله چهارم، واکشی متن از آبجکت ResourceBundle
۷۱۴	کلاس Locale
۷۱۵	متنها و پیامها
۷۱۶	فرمت دهی
۷۱۷	استفاده از فرمتهای آماده
۷۱۸	فرمتهای دلخواه برای نمایش اعداد
۷۱۹	تغییر نمادها
۷۱۹	فرمتهای دلخواه برای تاریخ و زمان
۷۲۰	تغییر نمادهای تاریخ و زمان
۷۲۱	پیام ها
۷۲۵	JAR ابار
۷۲۶	فایل JAR
۷۲۷	فایل Manifest
۷۲۸	اجرایی کردن فایل JAR
۷۲۹	تمرینات
۷۲۹	javadoc
۷۳۰	تولید اسناد javadoc
۷۳۱	چند نکته

جاوا

جاوا یکی از زبانهای محبوب برنامه نویسی در دنیاست. زمانیکه این زبان در دنیا معرفی شد زبان C و C++ محبوبترین زبانهای برنامه نویسی بودند اما جاوا خصوصیتی داشت که آنها نداشتند، جاوا مستقل از سخت افزار و سیستم عامل بود. در اصل، هدف طراحی جاوا هم همین بود. برنامه ای که روی یک سیستم عامل و یک سخت افزار اجرا می شد روی یک سیستم عامل دیگر یا سخت افزار دیگر قابل اجرا نبود. از آنجائیکه کامپیوترهای آن زمان از نظر سیستم عامل و سخت افزار کاملا متنوع بودند همین موضوع باعث افزایش هزینه تولید نرم افزار شده بود تا اینکه شرکت Sun Microsystems زبان جاوا را با شعار «برنامه تان را یک بار بنویسید، همه جا اجرا کنید» به دنیا معرفی نمود.

تاریخچه جاوا

در سال ۱۹۹۱ تیمی از مهندسان شرکت Sun Microsystems تصمیم گرفتند زبانی طراحی کنند که برای لوازم و وسائل الکترونیکی خانگی مورد استفاده قرار گیرد که تا قبل از پیدایش جاوا، با زبان C برای آنها برنامه نویسی می شد. تلاش این متخصصان منجر به طراحی زبانی مستقل از سخت افزار شد این زبان ابتدا Oak نامگذاری شد اما بعدا در سال ۱۹۹۵ به «جاوا» تغییر نام داد. قواعد و دستور زبان جاوا از زبانهای C و C++ گرفته شده است. زبان جاوا ابتدا برای لوازم خانگی، اسباب بازیها و وسائل الکترونیکی طراحی شد.

مدتی کوتاه پس از ارایه زبان جاوا، اینترنت به شکل تجاری در آمد. نکته جالب اینجا بود که کامپیوترهای اینترنت نیز همانند لوازم خانگی، متفاوت و متنوع بودند بنابراین نیاز به زبانی بود که بتوان با استفاده از آن برنامه‌هایی نوشت که در تمام کامپیوترهای اینترنت قابل اجرا باشد، این بار نیز جاوا گزینه مناسبی بود به یکباره، زبان جاوا به یکی از محبوب‌ترین زبانهای برنامه نویسی اینترنت تبدیل شد و طراحی مناسب آن باعث فرآگیر شدن آن گردید امروزه جاوا به عنوان یکی از زبانهای پرطرفدار و پراستفاده دنیا شناخته می‌شود.

با جاوا چه نوع برنامه‌هایی می‌توان نوشت؟

زبان جاوا یک زبان همه منظوره است یعنی با استفاده از آن می‌توانید انواع بسیار مختلف و متنوعی از برنامه‌ها را بنویسید از برنامه‌های رومیزی که فقط روی یک کامپیوتر اجرا می‌شوند تا برنامه‌های وب و برنامه‌های توزیع شده، برنامه‌هایی برای موبایل، لوازم خانگی و حتی بازیهای کامپیوتری. از جاوا برای ارتباط با سخت افزار استفاده نمی‌شود، مثلاً از جاوا برای ارتباط با پرینتر و اسکنر استفاده نمی‌شود زیرا وقتی صحبت سخت افزار به میان می‌آید نیازمند زبانی هستیم که بی‌واسطه بتواند با سیستم عامل یا حتی سخت افزار ارتباط برقرار کند. چون جاوا توسط JVM اجرا می‌شود و JVM نیز خود توسط سیستم عامل اجرا می‌شود، ارتباط با سخت افزار از طریق جاوا خیلی منطقی نیست هرچند غیرممکن نیز نیست.

در بازار کار بیشتر چه برنامه‌هایی تولید می‌شوند؟

در بازار کار —چه داخل ایران و چه خارج از ایران— از زبان جاوا بیشتر برای تولید برنامه‌های تحت وب و برنامه‌های توزیع شده استفاده می‌شود. برنامه‌های رومیزی سهم بسیار کمی از بازار جاوا را در اختیار دارند.

چگونه باید برنامه ریزی برای یادگیری جاوا را شروع کنم؟

یادگیری زبان جاوا نیاز به علاقه، پشتکار، و برنامه ریزی دقیق است. وقتی از علاقه و پشتکار خود مطمئن شدید، بدانید که کار سختی در پیش رو ندارید فقط لازم است برنامه ریزی داشته باشید و فرد با تجربه ای در کنار شما باشد تا شما را راهنمایی کند. خیلی خوب است اگر بتوانید در یک دوره آموزشی شرکت کنید ولی باید بدانید که صرف شرکت در یک دوره آموزشی شما را برنامه نویس نخواهد کرد. در کنار دوره آموزشی باید مرتباً تمرین کنید. مطمئناً در انجام تمرینات سوالات بسیاری برای شما پیش خواهد آمد، مشکلات زیادی را تجربه خواهید کرد، و بسیاری از جنبه‌های زبان جاوا برای شما روش خواهد شد. بنابراین در یک جمله ساده، شما باید علاقه و پشتکار داشته باشید و ضمن شرکت در یک دوره آموزشی، به صورت مدوا و مستمر تمرین کنید. مطمئن باشید برنامه نویسی قدرتمندی خواهید شد.

بازار کار جاوا در ایران و خارج از ایران چگونه است؟

بازار کار جاوا در ایران بسیار خوب است. در ایران حجم بسیار بالای از پژوهه‌های دولتی با جاوا پیاده سازی می‌شوند از آنجاییکه پژوهه‌های بزرگ همگی دولتی هستند برنامه نویس جاوا بسیار بیش از گذشته مورد نیاز

است. نوعاً حقوق برنامه نویسان جاوا تا چندین برابر زبانهای برنامه نویسی دیگر است. وضعیت جاوا در خارج ایران از داخل هم بهتر است کشورهای استرالیا، انگلستان، و کانادا از عمدۀ سرمایه گذاران در زمینه جاوا هستند در این کشورها پروژه‌ها به صورت پیش فرض با جاوا تولید می‌شوند. در کشورهای دیگر صاحب فناوری مثل آمریکا نیز وضع تقریباً به همین منوال است. گذشته از این، دانشگاهها و مراکز آکادمیک از گذشته تا کنون جاوا را به عنوان زبان مرجع برای کامپیوتر می‌شناسند.

اندروید چیست؟

اندروید یک سیستم عامل برای گوشیهای هوشمند و تبلت هاست که توسط گوگل و از روی هسته لینوکس و برخی ابزارهای منبع باز تولید شده است. بسیاری از شرکتهای سازنده گوشیهای هوشمند از قبیل Samsung، HTC، و Sony از اندروید به عنوان سیستم عامل گوشیهای خود استفاده می‌کنند. با این حساب ممکن است بپرسید اندروید چه ارتباطی با جاوا دارد؟ جواب این است که شرکت گوگل که تولید کننده اندروید است یک «بسته توسعه اندروید» به صورت رایگان و منبع-باز نیز برای برنامه نویسان جاوا تولید کرده است تا برنامه نویسان جاوا بتوانند با استفاده از این بسته برنامه‌های جاوای بنویسند که روی گوشی‌هایی با سیستم عامل اندروید اجرا شود. برنامه نویسی اندروید مبتنی بر جاوا استاندارد است، ولی مفاهیم، جزئیات و تکنیکهای برنامه نویسی خاص خود را دارد.

برای یادگیری جاوا باید لینوکس بلد باشم؟

برای یادگیری جاوا هیچ پیش نیازی وجود ندارد. البته اگر با یک زبان برنامه نویسی آشنا باشید یا مفاهیم شی گرایی را بدانید مطمئناً خیلی سریع تر جاوا را یاد خواهید گرفت. ولی همانطور که گفته شد جاوا مستقل از سخت افزار و سیستم عامل است، بنابراین می‌توانید روی لینوکس، ویندوز، یا مک برنامه نویسی کنید.

برای شروع برنامه نویسی جاوا به چه چیزهایی نیاز دارم؟

اولین چیزی که به آن احتیاج دارید JDK است. JDK مخفف عبارت Java Development Kit است و در فارسی به آن «بسته توسعه جاوا» گفته می‌شود. این بسته شامل مجموعه‌ای از ابزارها شامل کامپایلر، دیباگر، اجرakannde جاوا، کتابخانه‌های جاوا، چندین برنامه نمونه و هر آنچیزی است که برای برنامه نویسی به زبان جاوا به آن نیاز دارید. بنابراین قبل از شروع برنامه نویسی جاوا باید JDK را نصب کنید. بعد از اینکه JDK را روی سیستم خود نصب کردید باید یک ویرایشگر جاوای داشته باشید تا بتوانید کدهای جاوا را در آن تایپ کنید و در کامپایل و اجرای برنامه به شما کمک کند. اگرچه شما می‌توانید بدون ویرایشگر، یک برنامه جاوا را بنویسید، اما استفاده از یک ویرایشگر کمک شایانی به برنامه نویسی سریع و بی عیب و نقص شما خواهد کرد. ویرایشگرهای معروف جاوای عبارتند از NetBeans، IntelliJ IDEA، eclipse، و IntelliJ IDEA.

کلاسیای آموزشی چقدر مفید هستند؟

خیلی خوب است که در یک دوره آموزشی ثبت نام کنید، داشتن یک معلم خوب می‌تواند کمک شایانی به شما بکند. متاسفانه این روزها بسیاری از موسسات آموزشی با شیوه‌های تبلیغاتی از علاقمندان سوء استفاده می‌کنند. حتی برخی دانشگاه‌های پرnam نشان نیز در این مسیر باطل افتاده اند تا با تبلیغات فراوان و اعطای مدرک با نشان دانشگاه، علاقمندان را به شرکت در دوره‌های خود ترغیب کنند. بنابراین فراموش نکنید که اگرچه شرکت در یک دوره آموزشی سیار لازم و مفید است اما در این مسیر، گرفتار شرکتها، موسسات، یا حتی دانشگاه‌های سودجو نشوید. اگر چنین احساسی داشتید، یا اگر به هر دلیل امکان شرکت در دوره‌ای را ندارید باز هم می‌توانید با تلاش و پشتکار به هدف خود برسید.

چه کتابهای بخوانم؟

خوشبختانه امروزه کتابهای مفید و مناسب فراوانی در زمینه برنامه نویسی جاوا وجود دارند اما همه آنها به زبان انگلیسی هستند اگر دنبال کتاب فارسی هستید فقط این کتاب و کتابهای دیگر این نویسنده پیشنهاد می‌شود اما باید بدانید که با خواندن این کتابها، کار شما تمام نشده است، بلکه تازه آغاز شده است بلاعسله پس از خواندن کتابهای فارسی باید به سراغ کتابهای انگلیسی بروید. اصولاً برای انجام هر کار با کامپیوتر به دانستن زبان انگلیسی نیاز دارید و برنامه نویسی کامپیوتر نیز از این نظر مستثنی نیست. دانش زبان انگلیسی شما باید در اندازه‌ای باشد که حداقل بتوانید متون فنی کامپیوتر را مطالعه کنید. سعی براین است که وبسایت اطلس سافت در حد امکان راهنمایی شما برای حل مشکلات و یادگیری زبان جاوا باشد. اگر سوالی دارید می‌توانید از طریق atlassoft.ir@gmail.com مطرح کنید.

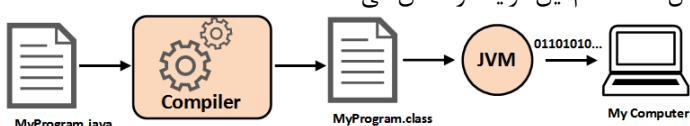


آماده شدن برای برنامه نویسی

قبل از اینکه برنامه نویسی جاوا را شروع کنید، لازم است با نحوه برنامه نویسی، کامپایل و اجرای برنامه های جاوا آشنا شوید. این همان موضوعی است که در این فصل به آن می پردازیم و در انتهای فصل نصب نرم افزارهای مورد نیاز برای برنامه نویسی جاوا را توضیح می دهم.

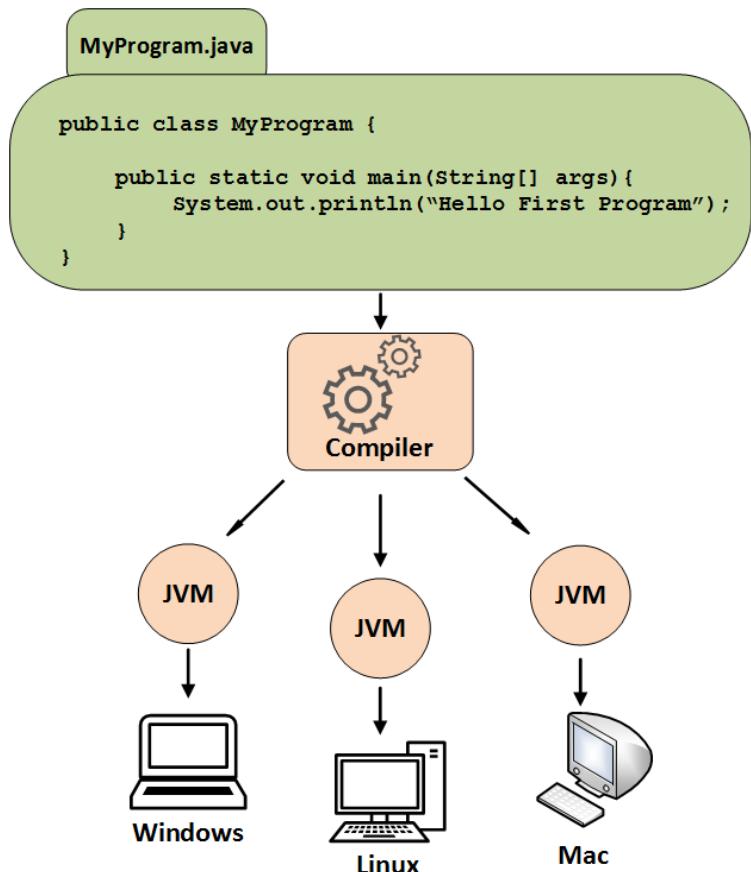
جاوا چگونه کار می کند؟

شیوه کار کرد جاوا بسیار جالب است. وقتی برنامه ای به زبان جاوا می نویسید آنرا در یک فایل با پسوند `.java` ذخیره می کنید سپس با استفاده از کامپایلر جاوا آنرا کامپایل می کنید تا خطاهای گرامری برنامه کشف شود و آنها را رفع کنید درصورتیکه هیچ اشکال گرامری در برنامه وجود نداشته باشد، کامپایلر جاوا از روی سورسهای برنامه (یعنی همان فایلهای `.java`) فایلهای `.class` تولید می کند. فایلهای تولید شده حاوی کدهای میانی هستند که به آنها «بایت کد» گفته می شود. حال نوبت «اجرا کننده جاوا» است تا بایت کدهای برنامه (یعنی همان فایلهای `.class`) را اجرا کند. به اجرای کننده جاوا، «ماشین مجازی جاوا» یا `JVM` گفته می شود. شکل ۱-۱ تمام این فرایند را نشان می دهد.



شکل ۱-۱. تولید، کامپایل و اجرای یک برنامه جاوا

جاوا یک شعار معروف دارد، «یک بار بنویسید همه جا اجرا کنید». منظور این شعار این است که وقتی شما برنامه‌ای می‌نویسید به محیطی که قرار است برنامه در آن اجرا شود (مثلاً ویندوز، لینوکس، مک،...) فکر نمی‌کنید. در زمان اجرا، این وظیفه ماشین مجازی جاواست که برنامه شما را منطبق بر محیطی (سیستم عامل و سخت افزار) که قرار است اجرا شود اجرا کند. شکل ۱-۲ این موضوع را نشان می‌دهد.



شکل ۱-۲. وجود نسخه‌های مختلف JVM برای هر سیستم عامل

همانطور که از شکل مشخص است، برای هر سیستم عامل، یک ماشین مجازی متفاوت طراحی شده است که کار تطبیق و اجرای برنامه‌های جاوا روی آن سیستم عامل را برعهده دارد.

توجه: برخی از JVM‌ها عملیات بیشتری را برای اجرای یک برنامه انجام می‌دهند، مثلاً نقاط ناکارآمد هر برنامه را کشف می‌کنند و راهبردی را برای بهبود و افزایش کارایی آن نقاط اتخاذ می‌کنند. حتی ممکن است برنامه را به کدهای محلی سیستم عامل کامپایل کنند تا سرعت اجرای برنامه افزایش یابد.

یکی از اینگونه JVMs Java HotSpot Virtual Machine است.

خانواده جاوا

اگرچه مکانیسم کامپایل، دیباگ و اجرای برنامه های جاوا برای هر برنامه ای که به زبان جاوا نوشته می شود یکسان است، اما به لحاظ انواع برنامه هایی که با جاوا تولید می شوند، جاوا در سه نسخه مختلف ارایه می شود که هر نسخه برای تولید نوع خاصی از برنامه ها استفاده می شود. تفاوت نسخه های جاوا در در کتابخانه های آنهاست و گرنه در کامپایلر، دیباگر، اجرائکننده جاوا و بسیاری از ابزارهای دیگر یکسان هستند. جدول ۱-۱ نسخه های جاوا و تفاوتها آنها را توضیح می دهد.

نحوه جاوا	مخفف	کاربرد
Java Standard Edition	Java SE	نسخه پایه جاواست. برای نوشتن هر برنامه جاوا به این نسخه نیاز دارد. در این نسخه کتابخانه های اصلی جاوا قرار دارند. ارتباط با پایگاه داده، برنامه نویسی XML تحت شبکه، کار با فایلها، Threadها، واسط کاربری رومیزی، پردازش همگی با این نسخه از جاوا امکانپذیر است.
Java Micro Edition	Java ME	از این نسخه برای نوشتن برنامه ها روی سخت افزارهای خاص مثل موبایل، لوازم خانگی، ... استفاده می شود.
Java Enterprise Edition	Java EE	نسخه مدرن و سازمانی جاواست که برای تولید برنامه های وب، برنامه های توزیع شده، و برنامه های روی Server اجرا می شوند.

جدول ۱-۱. نسخه های جاوا

Java SE یعنی همان نسخه استاندارد جاوا اصلی ترین نسخه جاواست. با استفاده از این نسخه می توانید برنامه هایی برای کامپیوترهای شخصی بنویسید مثلاً می توانید یک برنامه رومیزی، یک اپلت، یا یک برنامه اندروید بنویسید. می توانید به پایگاه داده متصل شوید و داده هایی را از آن بخوانید یا در آن ذخیره کنید یا یک فایل روی کامپیوترتان ایجاد کنید و داده هایی را در آن ذخیره کنید. همچنین می توانید به شبکه متصل شوید و داده هایی را با شبکه رد و بدل کنید. با استفاده از جاوای استاندارد (Java SE) برنامه های متنوعی می توان نوشت. در زیر توضیحات بیشتری در مورد این برنامه ها داده شده است.

۱. برنامه های رومیزی (Desktop Applications) به برنامه هایی گفته می شوند که روی کامپیوترهای شخصی اجرا می شوند خصوصیت اصلی این برنامه ها این است که فقط یک کاربر با آنها کار می کند. مثلاً برنامه Photoshop یک برنامه رومیزی است.

۲. اپلت ها (Applets) برنامه های جاوای هستند که توسط جستجوگر وب اجرا می شوند اپلتها یکی از اصلی ترین عوامل محبوبیت جاوا در جهان بوده اند اما امروزه با وجود رقبای قدر تمندی چون Ajax و Adobe Flash و مشکلات امنیتی که اخیراً رسانه ای شده اند، به ندرت استفاده می شوند.

۳. برنامه های اندروید (Android Applications) برنامه هایی هستند که به زبان جاوا برای گوشیهای هوشمند که سیستم عامل اندروید دارند نوشته می شوند.

برای نوشتن برنامه هایی برای موبایل، لوازم خانگی، و سخت افزارهای خاص استفاده می شود دقیق کنید که برنامه هایی که با Java ME نوشته می شوند را با برنامه های اندروید اشتباہ نگیرید. برنامه هایی که برای گوشیهای هوشمند اندروید نوشته می شوند در اصل برنامه های Java SE هستند که روی گوشی های هوشمند اندروید اجرا می شوند. اما برنامه های Java ME می توانند روی هر سخت افزاری که جاوا را پشتیبانی می کند –از جمله روی گوشیهای نسل قدیمی که هوشمند نیستند هم- اجرا شوند. برنامه های اندروید پیشرفته تر از برنامه های Java ME هستند، واسط کاربری زیباتر و معمولًا منطق پیچیده تری نیز دارند. امروزه از برنامه نویسی Java ME استقبال کمتری می شود و در عوض برنامه نویسی اندروید رونق زیادی دارد.

Java EE نسخه پیشرفته جاواست که برای نوشتن برنامه های Server استفاده می شود، برنامه های تحت وب و سرویس های ایمیل از جمله این برنامه ها هستند. Java EE از بخش های مختلفی تشکیل شده است که به هر کدام یک «تکنولوژی» گفته می شود و برای پیاده سازی یک سرویس خاص استفاده می شود. مثلاً تکنولوژیهای JSP و Servlet برای تولید و اسțط کاربری وب استفاده می شوند. از Java Mail API برای پیاده سازی سرویس ایمیل استفاده می شود. JMS برای پیاده سازی نوع خاصی از برنامه ها که برنامه های «مبتنی بر پیغام» نامیده می شوند استفاده می شود. در این نوع برنامه ها، قسمتهای مختلف یک برنامه می توانند با ارسال و دریافت پیغام با یکدیگر کار کنند. مبحث Java EE نسبت به دیگر مباحث جاوا بسیار گسترده است و یادگیری آن زمان بر و دشوارتر نیز هست.

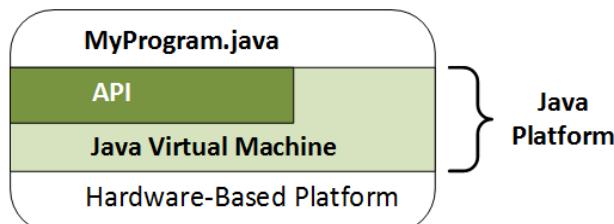
توجه: به برنامه های کاربردی که امروزه تولید می شوند و عموماً جزئیات مفصلی دارند گفته می شود واژه «Program» بیشتر در متون قدیمی و به برنامه های کوچک و ساده گفته می شد. در این کتاب از کلمه «برنامه» به عنوان معادل فارسی هر دو واژه استفاده شده است.

یادگیری زبان جاوا را از کجا شروع کنم؟

طبعی است که برای یادگیری جاوا باید از Java SE شروع کنید زیرا زیربنای نسخه های دیگر جاواست. اما هدف نهایی شما باید یادگیری Java EE باشد زیرا اغلب پروژه هایی که امروزه در ایران و خارج از ایران تعریف می شوند تحت سرور هستند و طبیعتاً باید با استفاده از Java EE پیاده سازی شوند. Java EE بسیار وسیع، گسترده و تا حدودی پیچیده است و به همین دلیل، یادگیری و تسلط به آن به زمان بیشتری نیاز دارد. اگر به برنامه نویسی موبایل علاقه دارید خوب است بدانید سهم اندروید در بازار نرم افزار در حال افزایش است هر چند یادگیری آن زمان زیادی نیاز ندارد و البته درآمد آن در مقایسه با Java EE کمتر است. Java ME شاید آخرین گزینه برای یادگیری باشد، زیرا این روزها خیلی خواهان ندارد.

بسترِ جاوا

در اینجا مناسب است که با اصطلاح «بسترِ جاوا» که معادل عبارت انگلیسی Java Platform است آشنا شویم. در بسیاری از مستندات و کتابهای جاوا از این اصطلاح برای اشاره به محیط اجرای برنامه های جاوا استفاده می شود که حداقل شامل کامپایلر جاوا، اجراکننده جاوا (ماشین مجازی جاوا) و کتابخانه های جاوا هست اما در نسخه های مختلف جاوا ممکن است چیزهای دیگری نیز در این بستر وجود داشته باشند. شکل ۱-۳ بسترِ جاوا استاندارد را نشان می دهد.



شکل ۱-۳. بسترِ جاوا استاندارد

توجه: اولین نسخه جاوا ۱.۰ و نسخه های بعدی آن به ترتیب ۱.۱، ۱.۲، ...، ۱.۸ هستند. معماری جاوا از نسخه ۱.۲ به صورت بنیادین تغییر کرد و انسجام و قوام پیدا کرد به همین خاطر نسخه ۱.۲ و نسخه های بعدی جاوا، ۲ Java نامیده شدند بدین ترتیب در متون جاوا و کنفرانس هایی جاوا از نامهای J2EE و J2ME استفاده می شد. از نسخه ۱.۵ به بعد شیوه نامگذاری تغییر کرد و نام "J2SE" به "Java SE" ، نام "J2EE" به "Java EE" و "J2ME" به "Java ME" تغییر نام دادند.

نصب Java SDK

Java Software Development که در فارسی «بسته توسعه جاوا» -معادل عبارت انگلیسی Java SDK - و به صورت مختصر JDK نامیده می شود در واقع بسته نرم افزاری است که برای برنامه نویسی Java SE به آن نیاز دارد. این بسته، شامل مجموعه ای از ابزارها (برای کامپایل، اجرا، مستندسازی، بسته بندی، و)، کتابخانه های جاوا (Java SE APIs)، و سورس کتابخانه های جاوا است. این بسته، که اصلی ترین نرم افزار برای برنامه نویسی جاواست را با مراجعه به وبسایت اوراکل دانلود و نصب کنید.

در مواقعي که صرفاً قصد اجرای یک برنامه جاوا را دارید (مثلاً در محیط عملیاتی که قرار است برنامه ای که قبل تولید شده است اجرا شود) به جای JDK می توان JRE را نصب کرد. Java Runtime Environment است و در فارسی به آن «محیط اجرایی جاوا» گفته می شود زیر مجموعه ای از JDK است و فقط شامل قسمتهایی از JDK است که برای اجرای برنامه های جاوا نیاز است. این بدین معنی است که JRE فاقد کامپایلر یا دیباگر است. شکل ۱-۴ ارتباط بین SDK، JDK، JRE و JVM را نشان می دهد. دقت کنید که مستطیل های داخلی زیرمجموعه مستطیل های خارجی هستند.



شکل ۴-۱. ارتباط بین SDK، JRE، JDK و JVM

توجه: جاوا یک زبان «منبع-باز» است. این بدین معناست که سورس کتابخانه های جاوا به عنوان بخشی از JDK در اختیار همگان قرار داده می شود. به طور کلی، محصولات منبع-باز، محصولاتی هستند که کدهای آن در اختیار همگان قرار دارد و بر اساس قوانینی که تولید کننده آن محصول وضع می کند دیگران می توانند از کدهای آن محصول استفاده کنند، یا آنرا تغییر دهد و محصول شخصی یا تجاری جدیدی تولید کنند.

توجه: جاوا یک «مشخصه» است. منظور از مشخصه این است که شرکت اوراکل به عنوان پشتیبانِ جاوا مسئولیت دارد تا خصوصیات، رفتار و جزئیات جاوا را تعریف و مشخص کند که به آن مشخصه گفته می شود. مشخصه به منزله سندی است که جزیبات و عملکرد تمام اجزای جاوا را توصیف و تشریح می کند. شرکت اوراکل سپس بر اساس مشخصه ای که تولید کرده است، نسخه ای از جاوا را تولید و در اختیار همگان قرار دهد. وجود مشخصه به عنوان مبنای پیاده سازی جاوا این امکان را به هر شرکت، موسسه، یا گروه می دهد تا براساس آن یک نسخه دیگر از جاوا را پیاده سازی کند. نتیجه این است که علاوه بر JDK که توسط شرکت اوراکل ارایه می شود JDK لهای دیگری از قبیل JamaicaVM (IBM، J9)، و OpenJDK نیز تولید شوند که توسط گروه ها و شرکتهای دیگر پشتیبانی می شوند. JDK لهایی که توسط شرکتهای دیگر تولید می شوند از تمام مشخصه جاوا که توسط اوراکل تدوین شده است پیروی می کنند و ممکن است در بخشهایی از قبیل «حذف آجکتهای اضافه»، «استراتژی کامپایل» و «تکنیکهای بهینه سازی» با یکدیگر متفاوت باشند. انگیزه این شرکتها برای تولید یک نسخه دیگر JDK از آنجا ناشی می شود که این شرکتها محصولات نرم افزاری مبتنی بر جاوا دارند و در تلاش هستند تا به منظور کارایی بهتر محصولات خود، نسخه بهینه ای از JDK را تولید و ارایه کنند.

توجه: اگرچه گفته می شود که جاوا منبع-باز است اما در JDK اوراکل، بخشهای کوچکی از کتابخانه های جاوا منبع-باز نیستند و کدهای آن در اختیار همگان قرار ندارد. علت این است که اوراکل طی

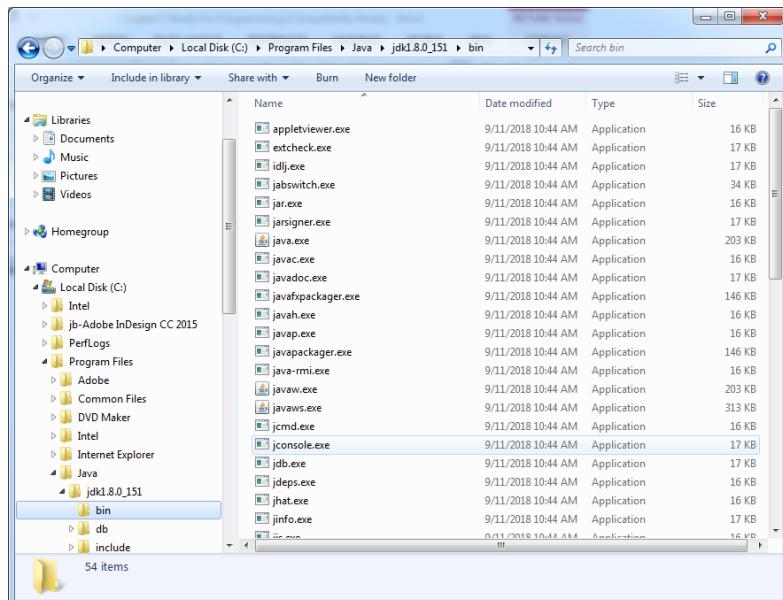
آماده شدن برای برنامه نویسی

قراردادی با برخی شرکتهای دیگر، تولید آن بخشها را به عهده آنها گذاشته است و براساس توافقات و قرارداد بین اوراکل و آن شرکتها، اوراکل حق نشر سورس آن بخشها را ندارد. این موضوع دلیلی شده است تا برنامه نویسان آزاد که به صورت مستقل فعالیت می کنند OpenJDK را تولید کنند که تمام بخش‌های آن «منبع باز» است.

دربیافت، نصب و راه اندازی JDK

برای آنکه برنامه نویسی جاوا را آغاز کنید لازم است JDK را دریافت، نصب و راه اندازی کنید. برای این منظور می توانید با مراجعه به وب سایت شرکت اوراکل، آخرین نسخه JDK را دریافت کنید. در حال آخرين نسخه جاوا 1.9 است.

پس از دانلود JDK روی آن دابل کلیک کنید و از طریق ویزاردش آنرا نصب نمایید. وقتی آنرا نصب کردید، شاخه ها و فایلهایی بصورتی که در شکل ۱-۵ نشان داده شده است در محل نصب شده وجود خواهد داشت.



شکل ۱-۵. شاخه ها و زیرشاخه های JDK نصب شده

در محل نصب شده -همانطور که ملاحظه می کنید- زیرشاخه های `.bin`, `.jre`, `.demo`, `.lib`, `.include` وجود خواهد داشت. شاخه `bin` حاوی ابزارهای JDK است که برای کامپایل، اجراء، اشکالزدایی، مستندسازی و ... استفاده می شوند. برخی از این ابزارها عبارتند از:

• کامپایلر جاواست که فایلهای `.java` را به `.class` کامپایل می کند. اگر کدهای جاوا اشکالات گرامی داشته باشد آنها را کشف و گزارش می کند و در صورتیکه اشکالی وجود نداشته باشد بایت کدهای جاوا (فایلهای `.class`) را تولید می کند.

- **java** ماشین مجازی جاواست که برنامه‌ای را که کامپایل شده است اجرا می کند.
- **jar** از روی مجموعه ای از فایلهای `.jar` می سازد. فایلهای `.jar` فایلهای بسته بندی شبیه فایلهای `.zip` هستند که در جاوا استفاده می شوند. این فایلهای پسوند `.jar`* دارند.
- **javadoc** برای تولید مستندات استفاده می شود.
- **appletviewer** ابزاری برای تست و مشاهده اپلت هاست، زمانیکه در حال نوشتن یک اپلت هستید، با استفاده از این ابزار می توانید آنرا اجرا نموده و عملکرد صحیح آنرا بررسی کنید.

نگران ابزارهای فوق نباشید! به مرور در این کتاب به درک کاملی از جاوا و ابزارهای جاوا خواهید رسید.

قرار دادن مسیر شاخه bin در PATH

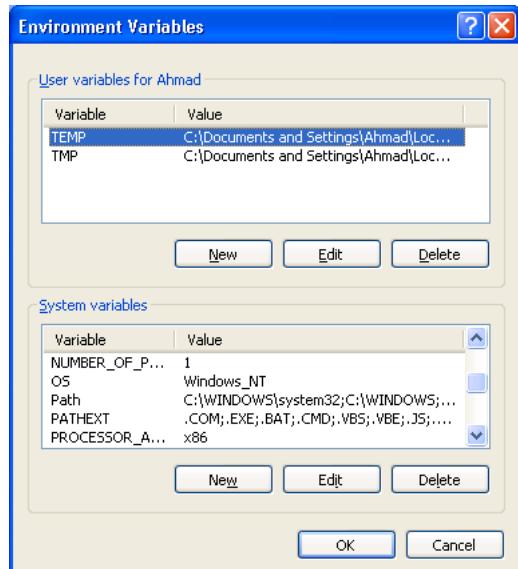
وقتی JDK را روی سیستم خود نصب کردید، آمده اید تا برنامه ای به زبان جاوا بنویسید، کامپایل و اجرا کنید. هرچند همانطور که در فصلهای بعدی خواهید دید ما به یک ویرایشگر هوشمند نیاز داریم تا کدهای خود را در آن بنویسیم، خطایابی و اجرا کنیم اما بدون وجود یک ویرایشگر نیز صرفا با داشتن JDK می توانیم شروع به برنامه نویسی جاوا کنیم، در فصلهای بعدی بعد از اینکه برخی مفاهیم پایه ای جاوا را آموختید و با برنامه نویسی در یک ویرایشگر متنی ساده (Notepad) آشنا شدید با معرفی IntelliJ IDEA نحوه کار با محیط های عملی برنامه نویسی را خواهید آموخت.

قبل از اینکه برنامه نویسی جاوا را شروع کنیم بهتر است با تنظیم متغیر سیستمی PATH امکان اجرا کردن کامپایلر و اجراکننده جاوا را در هرجایی از سیستم فراهم کنیم. وقت کنید که کامپایلر و اجراکننده جاوا دو ابزار JDK هستند که در شاخه `bin` قرار دارند و برای اجرا کردن آنها می بایست مسیر آنها قید شود، با تنظیم متغیر PATH دیگر برای اجرای آنها نیازی به قید شدن مسیر آنها نیست و صرفا با نام بردن آنها در هر مسیری از سیستم، اجرا می شوند.

تنظیم متغیر PATH در ویندوز

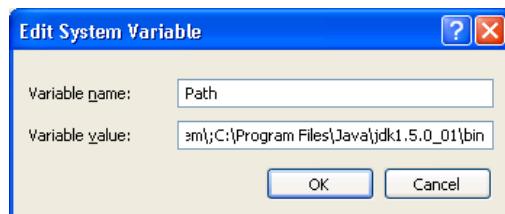
۱. روی My Computer کلیک راست کنید و گزینه Properties را انتخاب نمایید.
۲. در دیالوگی که به شما نشان داده می شود روی Advanced کلیک کنید.
۳. روی دکمه Environment Variables کلیک کنید.

آماده شدن برای برنامه نویسی



شکل ۶-۱. متغیرهای محیطی ویندوز

۴. در قسمت System Variables متغیر Path را پیدا کنید آنرا انتخاب کنید و سپس با کلیک روی Edit، آنرا ویرایش کنید.



شکل ۷-۱. ویرایش متغیر Path

۵. در دیالوگی که نشان داده می شود در انتهای رشته ای که به عنوان Path مشخص شده است یک ; بگذارید و مسیر bin مربوط به JDK ای که نصب کرده اید را وارد کنید
۶. تمام دیالوگ هایی که باز هستند را OK کنید.

تنظیم متغیر PATH در لینوکس

در لینوکس می توان با دستور `export` در پنجره دستور به صورت زیر این متغیر را تنظیم کرد.
`export PATH=$PATH:/usr/java/jdk1.9.0_01/bin`

در اینجا فرض شده است که جاوا در مسیر `/usr/java/jdk1.9.0_01` نصب شده است.

تنظیم مسیر جاوا از پنجره دستور بیشتر زمانی کاربرد دارد که می خواهید یک برنامه جاوا را آنهم فقط یکبار از پنجره دستور اجرا کنید. اگر می خواهید مسیر جاوا را به صورت دائمی روی سیستم خود تنظیم کنید (مشابه آنچه برای ویندوز انجام دادید) کافی است فایل /etc/profile را ویرایش کنید و خط

```
export PATH=$PATH:/usr/java/jdk1.9.0_01/bin
```

را به انتهای آن اضافه نمود. که در اینصورت به صورت دائمی و عمومی برای تمام کاربران سیستم این متغیر تنظیم شده است.

پ

شروع جاوا با یک برنامه ساده

در این فصل با نوشتن یک برنامه ساده با حال و هوای برنامه نویسی جاوا آشنا خواهید شد. پس از نوشتن برنامه، جزئیات آن را تشریح، سپس آنرا کامپایل و اجرا می کنیم.
کد ۲-۱ یک برنامه بسیار ساده به زبان جاوا را نشان می دهد.

```
1 //This is the first program in the Java language
2 public class FirstProgram {
3     public static void main(String[] args) {
4         System.out.println("First Program in Java");
5     }
6 }
```

کد ۲-۱

وقتی این برنامه را اجرا کنید جمله

First Program in Java

روی صفحه نمایش نشان داده خواهد شد.
اجازه دهید خط به خط برنامه را بررسی کنیم تا ببینیم این برنامه چگونه نوشته شده و چگونه کار می کند.

خط اول این برنامه

`//This is the first program in the Java language`

که با `//` شروع شده است فقط یک جمله توضیحی است و جنبه اجرایی ندارد. از علامت `//` برای افراد توپیخات یک خطی به برنامه استفاده می شود تمام جملاتی که بعد از `//` و در همان خط نوشته می شوند توسط کامپایلر نادیده گرفته می شوند. به این جملات توضیحی `comment` گفته می شود.

توجه: جملات توضیحی خوانایی برنامه را افزایش می دهند و به افراد دیگری که کدهای برنامه را می خوانند کمک می کنند تا عملکرد برنامه را سریعاً درک کنند. حتی ممکن است در آینده به خود شما نیز کمک کنند تا عملکرد برنامه ای که قبلاً نوشته اید را درک کنید!

جدول ۲-۱ سه علامت برای مشخص کردن جملات توضیحی را نشان می دهد.

علامت	توضیح	مثال
<code>//</code>	جملات توضیحی یک خطی که ابتدای جمله با <code>//</code> شروع می شود.	<code>//First Program in Java</code>
<code>/* ... */</code>	جملات توضیحی چند خطی که ابتدای توپیخات با <code>/*</code> شروع می شود و انتهاي جملات به <code>*/</code> ختم می شود.	<code>/* This is a simple java Program just for demonstrating basic structure of Java programs */</code>
<code>/** ... */</code>	مستندات جاوای که همانند توپیخات چند خطی معمولاً در چند خط بیان می شوند. این جملات در واقع مستندات جاوا هستند و این امکان وجود دارد تا در پایان انجام پروژه، با استفاده از ابزار <code>javadoc</code> که یکی از ابزارهای <code>JDK</code> است از روی این مستندات فایلهای <code>HTML</code> تولید کنیم.	<code>/** *An <code>FirstProgram</code> object shows how a simple Java program can be written. * * @version 1.0 * @author Ahmad R. Seddighi * @since 1.0 */</code>

جدول ۲-۲. انواع توپیخات در جاوا

خط ۲

شروع جاوا با یک برنامه ساده

```
class FirstProgram {
```

یک کلاس با نام `FirstProgram` تعریف می‌کند. هر برنامه جاوا از حداقل از یک کلاس تشکیل شده است در برنامه‌های بزرگ ممکن است تعداد کلاسها به چندهزار نیز برسد. اگر بخواهیم یک تعریف ابتدایی از کلاس ارایه کنیم، باید بگوییم «کلاس، مکانیسمی برای سازماندهی کدهای برنامه است». بر این مبنای، کدهایی که کارکرد و کارایی مشابه یا مرتبطی دارند در قالب یک کلاس تعریف می‌شوند. به عنوان نمونه، کدهای یک برنامه که مربوط به چاپ داده‌ها هستند در کلاسی با نام `Printer` تعریف و پیاده سازی می‌شوند. به همین ترتیب، کدهایی که داده‌های برنامه را پردازش می‌کنند در قالب کلاس `Processor` پیاده سازی می‌شوند. در مثال فوق نیز، کلاسی با نام `FirstProgram` تعریف کرده ایم زیرا این کلاس حاوی کدهای اولین برنامه‌ما به زبان جاواست.

توجه: تعریفی که از کلاس در اینجا مطرح کردیم یک تعریف ابتدایی و مقدماتی است و همانطور که در فصلهای بعدی خواهید دید برای کلاس که مهم‌ترین مفهوم در طراحی شی گرایی است، تعریف بسیار کامل‌تری وجود دارد.

از کلمه `class` برای تعریف یک کلاس استفاده می‌کنیم. هر کلاسی که تعریف می‌شود دارای یک نام (در مثال ما `FirstProgram`) هم است که بعد از کلمه `class` می‌آید. نام کلاس، یک کلمه دلخواه است که توسط برنامه نویس برای کلاس انتخاب می‌شود و امکان می‌دهد تا در هر جایی از برنامه به آن کلاس ارجاع دهیم. نام کلاس باید منطبق بر قوانین نامگذاری جاوا انتخاب شود.

پس از نام کلاس، از `{` برای مشخص کردن محتوای کلاس استفاده می‌شود و در پایان نیز از علامت `}` برای مشخص کردن انتهای کلاس استفاده می‌شود. به محتوای کلاس که بین علامتهای `{` و `}` قرار می‌گیرند «بدنه کلاس» نیز گفته می‌شود.

کلمه `class` که برای تعریف کلاس استفاده می‌شود یک کلمه رزو شده است. این بدین معنی است که این کلمه معنی خاصی در زبان جاوا دارد و شما نمی‌توانید آنرا به عنوان نام کلاس یا متغیر استفاده کنید. به کلمه `class` و کلمات دیگر که رزو شده هستند «کلمه کلیدی» یا `keyword` گفته می‌شود. جدول زیر لیست کلمات کلیدی در جاوا را نشان میدهد دقیق کنید که تمام این کلمات از حروف کوچک الفبای انگلیسی تشکیل شده‌اند.

abstract	double	int	strictfp	boolean
super	break	extends	long	switch
native	synchronized	case	finally	new
float	package	throw	char	for
class	goto *	protected	transient	const *
try	continue	implements	return	void
short	volatile	do	instanceof	static

interface	final	catch	throws	public
else	byte	this	private	if
default	while	import		

جدول ۲-۳. کلمات کلیدی

کلمات کلیدی که با * مشخص شده اند در حال حاضر در جاوا استفاده نمی شوند.
هر نامی که برای یک کلاس یا متغیر انتخاب می شود می بایست از قواعد نامگذاری جاوا پیروی کند که به صورت مختصر عبارت است از:

یک نام می تواند با ترکیبی از کاراکترهای عددی و غیرعددی الفبای انگلیسی (حروف کوچک یا بزرگ) ساخته شود با شرط اینکه کاراکتر عددی در ابتدای نام نباشد (نام با عدد شروع نشود). همچنین به جز \$ (علامت دلار) و _ (underline) نمی توان از هیچ کاراکتر خاص دیگری (مثلا *, %, يا &) در نام استفاده کرد. هرچند بهتر است از \$ و _ هم استفاده نشود.

در نامگذاری کلاسهای جاوا بهتر است حرف اول نام کلاس از حرف بزرگ الفبای انگلیسی و بقیه حروف از حرف کوچک انتخاب شوند. اگر نام کلاس دو کلمه ای باشد (مثلا FirstProgram) شروع کلمات از حروف بزرگ انتخاب شود.

توجه: جاوا یک زبان حساس به متن (case-sensitive) است یعنی به حروف کوچک و بزرگ حساس است به عنوان مثال، در جاوا A1 و a1 با یکدیگر متفاوت هستند.

توجه: تمام کلمات کلیدی جاوا با حروف کوچک هستند.

به برنامه خودمان بازگردیدم. در خط ۳، جمله

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

مشخص کننده یک متد (Method) است. مفهوم متد مشابه مفهوم «تابع» (function) است که در برنامه نویسی رویه ای استفاده می شود. اگر آشنایی قبلی با هیچ زبان برنامه نویسی ندارید، توضیحات زیر در کمتری از مفهوم متد به شما می دهد.

متدها از اجزای کلاسهای هستند، و می توان آنها را به ماشینی تشبیه کرد که داده هایی را دریافت می کنند، پردازشی را روی داده های دریافتی انجام می دهند و در نهایت نتیجه های را بر میگردانند.



شکل ۲-۴. کارکرد متد ها

شروع جاوا با یک برنامه ساده

اینها چیزهایی است که در تعریف متدهای لحاظ کنید، یعنی باید مشخص کنید که متدهای کدام نوع داده هایی را دریافت می کند، چه نوع داده ای را تولید می کند و بر میگرداند و از همه مهمتر نام متدهای چیست. شکل عمومی تعریف متدهای صورت زیر است.

```
returnType methodName(parameter-list) {  
    //method's body  
}
```

در اینجا، returnType نام متدهای برجسته، parameter-list نوع داده برجسته، و methodName داده های ورودی متدهای برجسته که به آنها «پارامترهای متدهای برجسته» گفته می شود. یک متدهای تواند پارامترهای ورودی نداشته باشد که در اینصورت به جای parameter-list چیزی نوشته نمی شود. همچنین می تواند مقدار برجسته نداشته باشد که در اینصورت به جای returnType کلمه void نوشته می شود. اما یک متدهای نام دارد زیرا بدون آن امکان فراخوانی و اجرای آن متدهای وجود ندارد. کد زیر یک متدهای نام perform را نشان می دهد که هیچ پارامتر ورودی ندارد و مقدار برجسته آن از جنس int (عدد صحیح) است.

```
int perform() {  
    .  
    .  
    return 1;  
}
```

در تعریف هر متدهای برجسته که داخل پرانتز مشخص می شوند، از علامت {} برای مشخص کردن شروع محتوای متدهای برجسته که خاتمه می شود. به محتوای متدهای برجسته، یعنی همان کدهایی که منطق متدهای برجسته را شکل می دهند و بین {} و {} قرار می گیرند، «بدنه متدهای برجسته» می شود. البتہ اگر یک متدهای برجسته نداشته باشد (یعنی void به عنوان نوع داده برجسته مشخص شده باشد) می توان از کلمه return بدون مشخص کردن یک مقدار استفاده کرد مثلاً متدهای process که مقدار برجسته آن void است.

```
void process(int input) {  
    .  
    .  
}
```

نیازی به `return` ندارد اما اگر در جایی از بدنه متدهای خواهیم اجرای متدهای خاتمه دهیم، از کلمه `return` بدون مشخص کردن مقداری استفاده می‌کنیم.

```
void process(int input) {
    .
    .
    .
    if(valid)
        return;
    .
    .
}
```

برای اجرای یک متدهای کافیست که نام آن متدهای را به همراه پارامترهای ورودی آن متدهای صدا بزنیم، اگر آن متدهای برگشتی داشته باشد میتوان در زمان فرآخوانی انتظار مقدار برگشتی را داشته باشیم، در غیر اینصورت اگر نوع برگشتی `void` تعیین شده باشد انتظار مقدار برگشتی نداریم.
اجازه دهید به برنامه ساده‌ای که نوشتیم برگردیم. در آن کلاس داخل بدنه کلاس، یک متدهای به صورت زیر تعریف و پیاده سازی شده است

```
public static void main(String[] args) {
    System.out.println("First Program in Java");
}
```

نام این متدهای `main`، پارامترهای ورودی آن آرایه‌ای از رشته (`String[] args`)، مقدار برگشتی `void` تعیین شده که به معنی نداشتن مقدار برگشتی است. در تعریف این متدهای از کلمات `public` و `static` استفاده شده که از کلمات کلیدی جاوا هستند و در فصلهای آینده در مورد آنها صحبت خواهیم کرد.

توجه: در فصلهای آینده در مورد رشته‌ها و آرایه‌ها صحبت خواهیم کرد.

اگرچه در انتخاب نام، پارامترهای ورودی و مقدار برگشتی متدها آزاد هستید و بسته به نیاز می‌توانید متدهای دلخواه تعریف و پیاده سازی کنید اما متدهای `main` که در مثال فوق پیاده سازی شده است همواره با همین شکل و شمایل تعریف و پیاده سازی می‌شود. در واقع هر برنامه جاوا، برای اینکه قابل اجرا باشد باید یک متدهای `main` دقیقاً به شکل فوق داشته باشد. وقتی یک کلاس جاوا را اجرا می‌کنیم، ماشین مجازی جاوا به دنبال متدهای با شکل فوق در کلاس مذبور می‌گردد تا آنرا اجرا کند، اگر چنین متدهای را یافت نکند پیغام خطای «متدهای `main` یافت نشد» را نمایش می‌دهد!

در بدنه متدهای `main` جمله

```
System.out.println("First Program in Java");
```

شروع جاوا با یک برنامه ساده

نوشته شده که رشتہ "First Program in Java" را در کنسول برنامه نمایش می دهد. اما کنسول برنامه چیست؟ منظور از کنسول برنامه، همان پنجره دستور یا **Command Prompt** در ویندوز است! خوب حتماً می پرسید این چه ارتباطی با برنامه جاوا دارد؟ جواب این است که برنامه های جاوا عموماً از طریق پنجره دستور یا **Command Prompt** اجرا می شوند. بنابراین **System.out.println()** نمایش رشتہ های دلخواه در کنسول برنامه است.

توجه: به «پنجره دستور» که خروجی اجرای برنامه در آن نمایش می یابد «کنسول برنامه» یا «خروجی استاندارد» گفته می شود.

در پایان جمله **System.out.println()** از علامت ; برای مشخص کردن انتهای آن جمله استفاده شده است. علامت ; انتهای هر جمله جاوا را مشخص می کند، فراموش کردن ; در انتهای جمله باعث خطای کامپایل خواهد شد.

ذخیره و کامپایل برنامه

کد فوق را در یک ویرایشگر متنی (مثلا **Notepad**) تایپ کنید و در یک فایل همان کلاس با پسوند **.java** ذخیره کنید (در اینجا چون نام کلاس **FirstProgram** است نام فایل نیز باید **FirstProgram.java** انتخاب شود).

حال باید آنرا کامپایل کنید برای این منظور ابتدا پنجره دستور (**Command Prompt**) را باز کنید و به مسیری که فایل برنامه را ذخیره کرده اید بروید. در این مثال با فرض اینکه فایل **FirstProgram.java** را در **c:\java-program** ذخیره کرده باشید به شکل زیر می توانید به شاخه **c:\java-program** بروید.

```
Microsoft Windows [Version 6.0.6001]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.
```

```
C:\Users\ABC>cd\
C:\>cd java-programs
C:\>java-programs>
```

حال برای کامپایل کلاس، جمله **javac FirstProgram.java** را تایپ کنید.



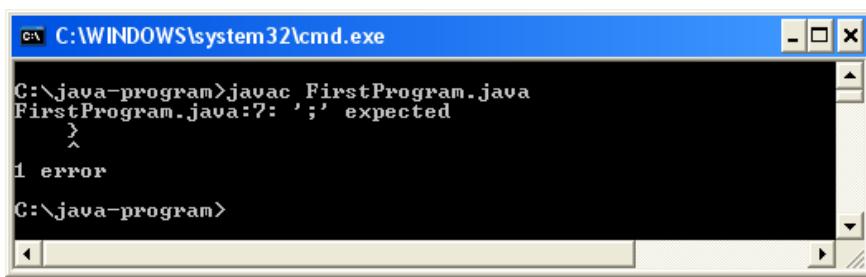
شکل ۲-۲. دستور کامپایل کلاس

اگر کدی که نوشته‌اید هیچ اشکال گرامری نداشته باشد با موفقیت کامپایل خواهد شد و یک فایل FirstProgram.class در کنار FirstProgram.java ایجاد می‌شود. در غیر اینصورت ممکن است با پیام خطایی مواجه شوید. دقت کنید که پیغام زیر نشان دهنده خطای نیست و صرفاً اعلان می‌کند متغیر PATH را که به مسیر نصب JDK اشاره می‌کند و در فصل اول آرا توضیح دادیم به درستی تنظیم نکرده‌اید.



شکل ۲-۳. پیغام خطای عدم نصب یا تنظیم JDK

شکل ۲-۴، یک پیغام خطای کامپایل را نشان می‌دهد که حاکی از قلم افتادن یک ; در انتهای یک جمله از برنامه است.



شکل ۲-۴. گزارش یک خطای گرامری

اجرای برنامه

پس از اینکه برنامه کامپایل و فایل `class`. تولید شد می‌توان آن را اجرا کرد. اجرای برنامه، از طریق پنجره دستور (Command Prompt) و با استفاده از دستور `java` انجام می‌شود (برای کامپایل از دستور `javac` استفاده شد).

`java FirstProgram`

دقت کنید که برای اجرا، نام کلاس بدون پسوند آمده است. در اینصورت نتیجه زیر را در کنسول ملاحظه خواهید کرد.

`First Program in Java`

اکنون که اولین برنامه را با موفقیت کامپایل و اجرا کردید، فرصت مناسبی است که نگاهی دقیقتر به آن بیندازیم و مطالب بیشتری در رابطه با جاوا بیاموزید.

ورودی و خروجی استاندارد

پیام‌ها و نتیجه اجرای یک برنامه ممکن است در یک فایل ثبت شود، روی صفحه نمایش نشان داده شود، یا روی شبکه ارسال شود. در این بین، صفحه نمایش، خروجی اصلی برنامه‌های جاواست و به همین دلیل است که به آن «خروجی استاندارد» گفته می‌شود.

دسترسی به خروجی استاندارد در برنامه‌های جاوا از طریق جملات `System.out.println()` و `System.out.print()` امکانپذیر است. در اولی، پس از نمایش یک متن، نشانگر به ابتدای خط بعدی می‌رود اما در دومی نشانگر در انتهای متن باقی می‌ماند.

داده‌های یک برنامه ممکن است از طریق موس، صفحه کلید، فایل، حافظه کامپیوتر، یا حتی شبکه وارد برنامه شوند در بین تمام این ورودی‌ها، به کنسول برنامه که کاراکترهای صفحه کلید را دریافت می‌کند «ورودی استاندارد» گفته می‌شود. برای ورود یک کارکتر صفحه کلید در کنسول برنامه از جمله `System.in.read()` استفاده می‌شود.

توجه کنید که کنسول برنامه اینترفیس مشترک ورودی استاندارد و خروجی استاندارد است و در حالیکه کاراکترهای صفحه کلید از طریق کنسول برنامه وارد برنامه می‌شوند، پیامهای برنامه از طریق همین کنسول نمایش می‌یابند.

ورود یک رشته به برنامه از طریق ورودی استاندارد

جمله `System.in.read()` کاراکتری که توسط کاربر تایپ می شود را از طریق کنسول دریافت می کند. مقداری که این جمله برمی گرداند یک عدد است که کد اسکی کاراکتری است که توسط کاربر تایپ شده است که از طریق جمله زیر می توانید به کاراکتر متناظر آن دسترسی پیدا کنید.

```
char c = (char) System.in.read();
```

در زیر برنامه ای نوشته شده است که از کاربر می خواهد تا رشته ای را تایپ کند و در انتهای دکمه `Enter` را فشار دهد. به محض اینکه `Enter` زده شود (کد اسکلی دکمه `Enter`، `'\r'` است)، خواندن کاراکترها خاتمه می یابد و رشته وارد شده در کنسول برنامه چاپ می شود.

```
1 //this program shows reading from console
2 public class ReadingFromConsole {
3     public static void main(String[] args) throws Exception{
4         String str = "";
5         char ch = ' ';
6         System.out.println("Enter Any String Please!");
7         while(ch != '\r'){
8             ch = (char) System.in.read();
9             str = str + ch ;
10        }
11        System.out.println();
12        System.out.println("Your entered String is: "+str);
13    }
14 }
```

کد ۲

نمونه ای از اجرای این برنامه را در زیر ملاحظه می کنید.

```
Enter Any String Please!
Java Programming Fundamental
```

```
Your entered String is: Java Programming Fundamental
```

اگر در کد فوق با دقت نگاه کنید متوجه خواهید شد که در تعریف متدهای `main()` از عبارت `throws Exception` استفاده شده که قبلاً با آن برخورد نکرده اید. لطفاً صبر کنید تا در فصل کنترل خطای در مورد آن صحبت کنیم.

System.out.print()

شروع جاوا با یک برنامه ساده

قبل استفاده از جمله `System.out.println()` برای نمایش یک عبارت در کنسول برنامه را آموختید. جمله `System.out.print()` نیز عملکرد مشابهی دارد با این تفاوت که () پس از نمایش یک رشته، اشاره گر را به خط بعدی می برد تا رشته های بعدی در خطوط بعدی نمایش یابند در حالیکه () اشاره گر را در انتهای همان خط نگه می دارد تا رشته های بعدی در انتهای خط فعلی نمایش داده شوند.

نکته: اگر رشته حاوی کاراکتر `\n` باشد این کاراکتر نمایش داده نمی شود و به جای آن، بقیه رشته که بعد از `\n` قرار دارند در خط بعدی چاپ می شوند مثلا با اجرای جمله زیر

```
System.out.println("Welcome\nto\nJava\nProgramming");
```

خروجی زیر نمایش می یابد.

```
Welcome  
To  
Java  
Programming
```

به کاراکتر `\n` که با معنی خاصی تفسیر می شود یک کاراکتر کنترلی گفته می شود. در فصل «رشته ها و کاراکترها» با کاراکترهای کنترلی دیگری نیز آشنا خواهید شد.

پیاده سازی و فراخوانی یک متده جدید

هر برنامه جاوا از مجموعه ای از کلاسها، و هر کلاس از مجموعه ای از متدها تشکیل شده است. خاصیت مهم متدها این است که می توانند یکدیگر را فراخوانی کنند. در این قسمت با نحوه فراخوانی یک متده توسط متده دیگر آشنا می شویم.

فرض کنید متده `print()` را به صورت زیر پیاده سازی کرده ایم تا یک متن را در کنسول برنامه نمایش دهد.

```
static void print(String text){  
    System.out.println(text);  
}
```

نام این متده `print()`، پارامتر آن رشته ای با نام `text` است و مقداری نیز برنمی گرداند چراکه مقدار برگشتی آن `void` است. بدنه این متده حاوی یک جمله `System.out.println()` است که مقدار پارامتر متده (رشته `text`) را در کنسول برنامه نمایش می دهد.

حال مثال `FirstProgram` که را قبل از نوشته ایم را برای فراخوانی متده `print()` به صورت زیر تغییر می دهیم.

```
1 //using a new method, print()
```

```

2  public class UsingPrintMethod {
3      public static void main(String[] args) {
4          print("First Program in Java");
5      }
6      static void print(String text) {
7          System.out.println(text);
8      }
9  }

```

کد ۲-۳

خروجی اجرای برنامه فوق متنابه قبل است، با این تفاوت که در اینجا متد `print()` را برای نمایش رشته `First Program in Java` فراخوانی کرده است.

استفاده از یک متغیر عمومی (property) بین چندین متد

یک کلاس می‌تواند علاوه بر متدها حاوی متغیرهای عمومی نیز باشد. منظور از متغیر عمومی، متغیری است که درون کلاس اما خارج از بدنه متدها تعریف می‌شود و درون متدهای کلاس یا حتی کلاس‌های دیگر برنامه قابل استفاده است. برای درک متغیر عمومی، کلاس زیر را ملاحظه کنید.

```

1 //this example shows how a class can have a property
2 public class HavingProperty {
3     static int number=0;
4
5     public static void main(String[] args) {
6         print("This is ");
7         print("your First Program in java");
8         print("You called method print "+number+" Times!");
9     }
10    static void print(String text) {
11        number++;
12        System.out.println(text);
13    }
14 }

```

کد ۲-۴

در این کلاس، خارج از بدنه متدها، متغیری با نام `number` تعریف شده است که هدف آن شمارش تعداد دفعاتی است که متد `print` فراخوانی می‌شود (هر بار که متد `print` فراخوانی می‌شود یک واحد به مقدار

شروع جاوا با یک برنامه ساده

این متغیر افزوده می شود). در پایان متد main تعداد دفعات فراخوانی متد print فراخوانی شده است چاپ گردیده است.

توجه داشته باشید که متد print مقدار متغیر number را افزایش می دهد و متد main مقدار آنها را نمایش می دهد.

توجه: از اینجا به بعد، برای اینکه مطالب کتاب خواناتر شود، هرجایی که نامی از یک متد می بریم در انتها نام از () استفاده می کنیم تا مشخص کنیم که از یک متد نام می بریم. مثلاً می گوییم متدهای () و print() اما می گوییم متغیر number. وقت کنید که () که بعد از نام متد آورده ایم پارامترهای متد را مشخص نمی کند و صرفاً جهت خوانایی آورده شده است کما اینکه در اینجا متدهای main و print هر دو دارای پارامتر هستند.

استفاده از System.out.printf()

در متد main() که در مثال اخیر مشاهده کردید از جمله

"**You called method print "+number+" Times!**

تعداد دفعاتی که متد print فراخوانی شده است را نشان می دهد. در اینجا مجبور شده ایم با استفاده از عملگر + دو رشته و یک عدد را به یکدیگر متصل کنیم.

You called method print

number

Times!

متد printf() در چنین مواردی که کار ایجاد و قالبدهی یک رشته دشوار است کاربرد دارد. به عنوان مثال، رشته فوق را با استفاده از جمله زیر می توان در کنسول برنامه نمایش داد.

```
System.out.printf("You called method print %d Times!", number);
```

در فراخوانی printf()، پارامتر اول، رشته ای است که می خواهیم در کنسول برنامه نمایش یابد. در این رشته، مکانهایی که باید با مقدار عدد مناسب جایگزین شوند با علامت % مشخص می شوند. علاوه بر % از علامتهاي %s, %c و %n به ترتیب برای مشخص کردن مکان یک رشته، یک کاراکتر و شروع خط جدید نیز می توانید استفاده کنید. پارامترهای بعدی متد printf()، اعداد، کاراکترها، یا رشته هایی هستند که باید جایگزین شوند. از آنجا که در مثال فوق فقط عدد number باید جایگزین %d شود، number به عنوان پارامتر دوم متدد شخص شده است.

پیاده سازی یک کلاس جدید

متدها می توانند در کلاسهای مختلفی سازماندهی شوند این بدین معنی است که متدهی که در یک کلاس تعریف و پیاده سازی می شود می تواند متدهی را که در یک کلاس دیگر قرار دارد فراخوانی کند. مثال زیر دو کلاس و چگونگی فراخوانی یک متدهای کلاس توسط یک متدهای دیگر را نشان می دهد.

```

1 public class FirstClass {
2     public static void main(String[] args) {
3         SecondClass.print("This is ");
4         SecondClass.print("your First Program in java");
5     }
6 }
7 public class SecondClass {
8     static void print(String text) {
9         System.out.println(text);
10    }
11 }
```

۲-۴ کد

در این مثال، کلاس FirstClass متدهای print() از کلاس SecondClass را فراخوانی کرده است توجه کنید که برای فراخوانی متدهای از کلاس دیگر، از پیشوند نام آن کلاس و نقطه (.) قبل از نام آن متده استفاده شده است.

استفاده از عملگرهای محاسباتی

در جاوا مجموعه ای از عملگرهای محاسباتی برای کار با اعداد و متغیرهای عددی پیش بینی شده اند. جدول ۲-۳ این عملگرهای محاسباتی را نشان می دهد.

عملگر	علامت	معنی	مثال
جمع	+	دو عدد را یکدیگر جمع می کند	$f + 7$
تفريق	-	دو عدد را از یکدیگر تفريقي می کند	$f - 7$
ضرب	*	دو عدد را در یکدیگر ضرب می کند	$f * 7$
تقسيم	/	یک عدد را بر عدد دیگری تقسيم می کند	$f / 7$
خارج قسمت تقسيم	%	باقیمانده تقسيم یک عدد را بر عدد دیگری بدست می دهد.	$f \% 7$

جدول ۲-۳. عملگرهای محاسباتی

تمرین: برنامه ای بنویسید که از طریق کنسول برنامه، دو عدد از کاربر دریافت کند، سپس حاصل جمع، حاصل ضرب، حاصل تفریق، خارج قسمت تقسیم و باقیمانده تقسیم یکی بر دیگری را نمایش دهد.

مثالی دیگر

هر برنامه جاوا از مجموعه ای از کلاسها و هر کلاس از مجموعه ای از متدها تشکیل شده است. تا اینجا با مفاهیم اولیه کلاس و متدها آشنا شدید و تعریف و استفاده از متغیر را در کلاس آموختید. به این متغیر که خارج از بدن متدها تعریف می شود و همه متدها به آن دسترسی دارند، **property** گفته می شود و مشابه متند، جزئی از کلاس محسوب می شود. در ادامه این فصل تلاش می کنم تا با ارایه مثالهای دیگر شما را با مفاهیم «کلاس»، «متده» و «property» بیشتر آشنا کنم. در کنار آن، برخی موضوعات جدید را نیز توضیح می دهم.

پیاده سازی کلاس Mathematics

هر برنامه جاوا از یک یا چندین کلاس جاوا تشکیل شده است و هر کلاس نیز شامل مجموعه ای از متدهای مرتبط است. اینکه چه متدهایی مرتبط محسوب می شوند و می بایست در یک کلاس تعریف شوند به منطق آنها مربوط می شود، متدهایی که عملکردشان از نظر منطقی در یک گروه قرار می گیرند طبیعتاً در یک کلاس نیز تعریف می شوند.

تصور کنید که در یک برنامه بخواهیم چهار عمل اصلی محاسباتی (جمع، تفریق، ضرب، تقسیم) را روی دو عدد انجام داده و نتیجه را نمایش دهیم. کلاس **Mathematics** این برنامه را نشان می دهد.

```
1 public class Mathematics {  
2     public static void main(String[] args) {  
3         int a = 4;  
4         int b = 7;  
5         int sum = a + b;  
6         int sub = a - b;  
7         int div = a/b;  
8         int mul = a*b;  
9         System.out.println("sum: "+sum);  
10        System.out.println("sub: "+sub);  
11        System.out.println("div: "+div);  
12        System.out.println("mul: "+mul);  
13    }  
14 }
```

در این مثال، ابتدا دو متغیر به نامهای `a` و `b` که مقادیر `4` و `7` را نگهداری می‌کنند تعریف شده‌اند. تعریف دو متغیر `a` و `b` این مزیت را دارد که با تعیین مقادیر آنها برنامه برای اعداد دیگر هم کار می‌کند. بعد از تعریف متغیرها، حاصل جمع، تفریق، تقسیم و ضرب دو متغیر محاسبه شده و در متغیرهای دیگری ذخیره شده است. تا توسط جملات `System.out.println()` به صورت جداگانه در خروجی استاندارد نمایش داده شوند. دقیق کنید که در زمان نمایش نتایج، از عملگر `+` برای چسباندن یک رشته به یک عدد صحیح و نمایش آن در کنسول برنامه استفاده شده است.

توجه: از عملگر `+` برای جمع اعداد یا مقادیر متغیرهای عددی استفاده می‌شود. از این عملگر همچنین برای چسباندن رشته‌ها به داده‌های دیگر استفاده می‌شود. حاصل چسباندن یک رشته به هر داده‌ای (یک رشته، یک داده عددی یا غیرعددی، ...) یک رشته خواهد بود.

کد فوق اگرچه کار می‌کند اما از نظر طراحی اشکالات عمدی دارد. اولین اشکال آن، وجود قسمتهای تکراری در کلاس است. مثلاً برای نمایش نتایج محاسبات از چهار جمله `System.out.println()` استفاده شده است. در نتیجه اگر از ما بخواهند که به جای نمایش نتایج در کنسول برنامه، نتایج را در یک پنجره نمایش دهیم مجبور خواهیم بود تا چهار جمله `System.out.println()` را جمله دیگری جایگزین کنیم تا به جای کنسول، نتایج در پنجره نمایش یابند. راه حل این مسئله، تعریف یک متدهد در این کلاس است که در قسمت بعدی توضیح داده می‌شود.

توجه: هر برنامه باید طوری طراحی شود که تعیین برنامه همیشه با حداقل تغییراتِ کد امکانپذیر باشد.

متد

براساس یک اصل طراحی، لازم است قسمتهای تکراری برنامه در قالب متدهد جداگانه پیاده سازی شود. اگرچه نمایش پیغام‌ها در این مثال آنقدر عمل ساده‌ای محسوب می‌شود که می‌توان از تکراری بودن آن صرف‌نظر کرد، اما تمرین خوبی است تا لزوم حذف کد تکراری را بیاموزید و مزیت استفاده از متدهد به جای کد تکراری را درک کنید.

طبق آنچیزی که گفته شد می‌توان متدهد جداگانه‌ای برای نمایش پیغام‌های برنامه پیاده سازی نمود. کد `4-2` کلاس تغییریافته قبلی را که اکنون حاوی یک متدهد جدید است را نشان می‌دهد.

```

1 public class Mathematics2{
2     public static void main(String[] args) {
3         int a = 4;

```

```
4      int b = 7;
5      int sum = a+b;
6      int sub = a-b;
7      int div = a/b;
8      int mul = a*b;
9      show("sum: "+sum);
10     show("sub: "+sub);
11     show("div: "+div);
12     show("mul: "+mul);
13 }
14 public static void show(String message) {
15     System.out.println(message);
16 }
17 }
```

۲-۶ کد

همانطور که ملاحظه می کنید متد `show()` به کلاس اضافه شده و در تعریف آن از کلمات کلیدی `void`, `public`, `static` و `String` استفاده شده است. این متد چیزی برنمی گرداند (مقدار برگشتی `void` دارد)، یک رشته دریافت می کند (پارامتر آن `String str` است) و آنرا در کنسول برنامه نمایش می دهد. واضح است که اگر بخواهیم نتایج محاسبات را در جای دیگری به غیر از کنسول برنامه نمایش دهیم، کافیست متد `show()` را تغییر دهیم بدون اینکه نیاز به تغییر دیگری در برنامه باشد.

تمرین: یک متد به نام `merge` بنویسید که دو رشته را به عنوان ورودی دریافت کند آن دو رشته را با یکدیگر متصل می کند و نتیجه حاصل شده را برمی گرداند.

در طراحی کلاس `Mathematics2` یک اشکال دیگر نیز وجود دارد. متد `main()` بیش از اندازه طولانی و مفصل است! اگر در یک برنامه یک متد طولانی وجود دارد که مجموعه ای از عملکردهای مختلف در آن پیاده سازی شده است، بهتر است با تجزیه آن متد به متدهای کوچکتر، ضمن افزایش خوانایی آن متد، تغییر و نگهداری آن را نیز آسان کنیم. البته اگر بخواهیم صادق باشیم، متد `main()` آنقدرها هم طولانی و مفصل نیست، اما در اینجا تمرین خوبی است تا طراحی صحیح متدها را بیاموزید.

توجه: در هر برنامه می توان قسمتهای تکراری برنامه را در قالب یک متد تعریف نمود اینکار ضمن کاهش کدهای برنامه، خوانایی آنها را بالا می برد و تغییر و نگهداری آنها را آسان می کند.

در حال حاضر عملکرد متدهای main() شامل محاسبه چهار عمل اصلی و نمایش نتایج در کنسول برنامه است. با تجزیه این متدها به چهار متدهای جداگانه که هر یک عمل اصلی را انجام می‌دهند کد تمیز و خواناتر زیر تولید می‌شود.

```

1 // This class has a static method, just prints a message
2 public class Mathematics3 {
3
4     public static void main(String[] args) {
5         int a = 4;
6         int b = 7;
7         printSum(a, b);
8         printSub(a, b);
9         printDiv(a, b);
10        printMul(a, b);
11    }
12
13    public static void show(String message) {
14        System.out.println(message);
15    }
16
17    public static void printSum(int x, int y) {
18        int result = x + y;
19        show("sum: "+result);
20    }
21
22    public static void printSub(int x, int y) {
23        int result = x - y;
24        show("sub: "+result);
25    }
26
27    public static void printDiv(int x, int y) {
28        int result = x/y;
29        show("div: "+result);
30    }
31
32    public static void printMul(int x, int y) {
33        int result = x * y;
34        show("mul: "+result);
35    }

```

همانطور که ملاحظه می کنید کلاس Mathematics3 حاوی چهار متده است که به ترتیب مجموع، تفاضل، تقسیم و حاصلضرب دو عدد را محاسبه و در خروجی استاندارد نمایش می دهند. هر چهار متده دو عدد صحیح به عنوان پارامتر دریافت می کنند و مقدار برگشتی هر چهار متده نیز void است (یعنی هیچ مقداری را به فراخواننده خود برنمی گردانند) مشابه متدهای دیگری که تاکنون آموخته اید این متدها نیز با کلمات کلیدی static و public تعریف شده اند. با تغییر فوق، عملکرد متده main فراخوانی چهار متده جدآگانه ای است که هر کدام یک عمل اصلی را انجام می دهند و نتیجه را در کنسول برنامه نمایش می دهند.

توجه: هر چهار متده دو عدد صحیح با نامهای `x` و `y` به عنوان پارامتر دریافت می کنند این دو نام، در حقیقت دونام مستعار هستند که به دو پارامتر متده انتساب داده می شود تا در بدنۀ متده بتوان به مقادیر پارامترها دسترسی بپیدا نمود. به عبارت دیگر برای فراخوانی این دو متده کافی است دو پارامتر int به آنها ارسال کرد همانطور که ملاحظه می کنید نام این پارامترها در جایی که متدها فراخوانی شده اند `a` و `b` است. از `x` و `y` تنها داخل متدها برای دسترسی به مقادیر استفاده می شود.

استفاده از یک کلاس دیگر

اگرچه یک کلاس می تواند به تعداد دلخواه متده داشته باشد، در بسیاری از مواقع لازم است متدها را به جای یک کلاس در کلاسهای مختلفی تعریف کنیم. تا اینجا با فراخوانی یک متده توسط متده دیگری از همان کلاس آشنا شدید. در اینجا می خواهیم در مورد فراخوانی متدهای یک کلاس توسط متدهای کلاس دیگر صحبت کنیم.

برای توضیح این موضوع اجازه دهید طراحی کلاس Mathematics3 را تغییر دهیم و متدهای عملیات اصلی را در یک کلاس دیگر پیاده سازی کنیم. انجام این کار توجیه درستی دارد زیرا در آنصورت کلاسی خواهیم داشت که منحصرا در برگیرنده «منطق چهار عمل اصلی» باشد و هیچ کد غیرمرتبه دیگری در آن وجود نداشته باشد.

توجه: اینکه در هر برنامه چه کلاسهایی باید پیاده سازی شود و در هر کلاس چه متدهایی باید وجود داشته باشد از مباحث طراحی شی گرایی است که در فصلهای آینده به صورت دقیق و مفصل خواهد آموخت. در این فصل کلاس را در ساده ترین شکل آن یعنی «روشی برای دسته بنده متدهای یک برنامه» تصور کنید.

کد ۲-۸ کلاس جدید MathOperations را نشان می دهد که منحصرا چهار متده برای انجام چهار عمل اصلی در آن پیاده سازی شده است.

```

1  public class MathOperations {
2
3      public static void printSum(int x, int y){
4          int result = x + y;
5          System.out.println("sum: "+result);
6      }
7
8      public static void printSub(int x, int y){
9          int result = x - y;
10         System.out.println("sub: "+result);
11     }
12
13     public static void printDiv(int x, int y){
14         int result = x/y;
15         System.out.println("div: "+result);
16     }
17
18     public static void printMul(int x, int y){
19         int result = x * y;
20         System.out.println("mul: "+result);
21     }
22 }
```

۲-۸

با داشتن کلاس فوق، کلاس Mathematics4 حاوی فقط یک متده است که متدهای کلاس main() را فراخوانی کرده است.

```

1  /* Using mathematics operations
2  defined in the MathOperations class*/
3  public class Mathematics4 {
4      public static void main(String[] args){
5          int a = 4;
6          int b = 7;
7          MathOperations.printSum(a, b);
8          MathOperations.printSub(a, b);
```

شروع جاوا با یک برنامه ساده

```
9     MathOperations.printDiv(a, b);  
10    MathOperations.printMul(a, b);  
11 }  
12 }
```

کد ۲-۹

توجه: هر کلاس ممکن است یک متدهای `main()` داشته باشد اما برنامه فقط از طریق یکی از آنها اجرا می شود.

به فراخوانی متدهای کلاس `MathOperations` توسط متدهای `printDiv()` و `printMul()` از کلاس `Mathematics` کنید. برای فراخوانی هر متدهای `MathOperations`، پیشوند نام کلاس و یک نقطه قبل از نام متدهای پارامترهای آن ظاهر شده است. مثلا برای فراخوانی متدهای `printSum()` و `printAvg()` از جمله زیر استفاده شده است.

```
MathOperations.printSum(a, b);
```

توجه: این روش فراخوانی متدهای `static` امکانپذیر است و همانطور که در فصل شی گرایی خواهید دید برای متدهای غیر `static` روش فراخوانی متفاوت خواهد بود.

استفاده از متدهای کلاس Math

کلاس `Math`، یکی از کلاسهای جاواست که حاوی متدهایی برای انجام اعمال محاسباتی و مثلثاتی است. متدهای `random()` یکی از متدهای این کلاس است که یک عدد تصادفی اعشاری تولید می کند. عددی که تولید می شود در محدوده ۰ و ۱ است که شامل صفر می شود اما شامل یک نمی شود.

تولید اعداد تاس

فرض کنید می خواهیم برنامه ای بنویسیم که یک تاس را شبیه سازی کند. وقتی یک تاس را پرتاب می کنید یکی از اعداد ۱ تا ۶ به صورت تصادفی تولید می شود. حال فرض کنید بخواهید متدهای `random()` که تاس را شبیه سازی کند، یعنی وقتی آن متدهای `random()` را فراخوانی کنیم به صورت تصادفی یکی از اعداد ۱ تا ۶ را به ما برگرداند.

متدهای `play()` که عدد تصادفی تاس را تولید می کند به صورت زیر تعریف می شود.

```
public static int play(){  
    ...  
}
```

این متدهای پارامتر ورودی ندارد و این کاملاً منطقی است زیرا در پرتاب تاس هیچ داده اولیه‌ای وجود ندارد. مقدار برگشتی این متدهای عدد صحیح int است که طبیعتاً در زمان اجرا باید یکی از اعداد ۱ تا ۶ باشد.

برای تولید عدد تصادفی ۱ تا ۶ می‌توانیم از متدهای random() کلاس Math استفاده کنیم

```
double d = Math.random();
```

از آنجاییکه عدد تولید شده یک عدد اعشاری بین ۰ و ۱ است اگر حاصل فوق را در ۶ ضرب کنیم به یک عدد اعشاری بین ۰ تا ۶ شود که شامل ۰ می‌شود اما شامل ۶ نمی‌شود.

```
double d = Math.random()*6;
```

حال اگر حاصل عبارت فوق را یک واحد افزایش دهیم به یک عدد اعشاری تصادفی در محدوده ۱ تا ۷ می‌رسیم که شامل ۱ می‌شود اما شامل ۷ نمی‌شود.

```
double d = Math.random()*6 + 1;
```

حال با حذف قسمت اعشاری از عبارت فوق، به یکی از اعداد ۱ تا ۶ خواهیم رسید.

```
public static int play() {
    double d = Math.random()*6+1;
    int result = (int) d;
    return result;
}
```

برای حذف قسمت اعشاری و تبدیل حاصل عبارت به یک عدد صحیح از d (int) استفاده کرده‌ایم. به این کار تبدیل یک عدد اعشاری به عدد صحیح گفته می‌شود و در قسمتهای آینده با آن بیشتر آشنا خواهید شد.

محاسبه قطر مستطیل

فرض کنید می‌خواهیم متدهای پیاده سازی کنیم که اندازه طول و عرض یک مستطیل را دریافت کند و اندازه قطر مستطیل را محاسبه و برگرداند. بنابراین این متدهای تعریفی به شکل زیر داشته باشد:

```
public static double diameter(double a, double b) {
```

```
}
```

و a و b که دو عدد اعشاری double هستند اندازه‌های طول و عرض مستطیل هستند، مقدار برگشتی این متدهم که یک عدد double است اندازه قطر محاسبه شده توسط این متدهم کند که توسط متدهای سراغ بدنه می‌شود. حال اجازه دهید سراغ بدنه متدهای sqrt() و diameter() به شکل زیر باشد.

برای محاسبه مربع یک عدد به سادگی می‌توانیم آنرا در خودش ضرب کنیم. برای محاسبه جذر یک عدد می‌توان از متدهای sqrt() در کلاس Math استفاده نمود. با این توضیحات، شکل زیر از متدهای diameter() به شکل زیر در خواهد آمد.

```
public static double diameter(double a, double b) {
    double r = Math.sqrt(a*a + b*b);
    return r;
```

شروع جاوا با یک برنامه ساده

}

در کد فوق، ابتدا قطر مستطیل محاسبه و در متغیر `r` قرار داده شده و سپس توسط جمله `return r;` برگردانده شده است. اما می‌توان از تعریف متغیر `r` صرف نظر کرد و متند را به صورت زیر خلاصه کرد.

```
public static double ghotr(double a, double b) {
    return Math.sqrt(a*a + b*b);
}
```

جدول ۲-۴ متدهای دیگر کلاس `Math` را نشان می‌دهد.

مثال	توضیح	متند
اگر $x > 0$ باشد مقدار <code>abs(x)</code> برابر x خواهد بود اگر $x == 0$ باشد مقدار <code>abs(x)</code> صفر خواهد بود. اگر $x < 0$ باشد مقدار <code>abs(x)</code> برابر $-x$ خواهد بود	قدرمطلق x را بر می‌گرداند (x می‌تواند یک مقدار <code>long</code> , <code>int</code> یا <code>float</code> باشد)	<code>abs(x)</code>
مقدار <code>ceil(9.2)</code> برابر 10.0 و مقدار <code>ceil(-9.8)</code> برابر -9.0 خواهد بود.	نزدیکترین عدد صحیح بزرگتر از x را برمی‌گرداند	<code>ceil(x)</code>
مقدار <code>cos(0.0)</code> برابر 1.0 خواهد بود	مقدار کسینوس x را بر می‌گرداند	<code>cos(x)</code>
مقدار <code>exp(1.0)</code> برابر $e^{1.0} \approx 2.71828$ خواهد بود.	مقدار e^x را بر می‌گرداند	<code>exp(x)</code>
مقدار <code>floor(9.2)</code> برابر 9.0 و مقدار <code>floor(-9.8)</code> برابر -10.0 خواهد بود.	نزدیکترین عدد صحیح کوچکتر از x را برمی‌گرداند	<code>floor(x)</code>
$\log(2.71828)$ برابر 1.0 و $\log(7.389056)$ برابر 2.0 خواهد بود.	لگاریتم x را بر مبنای نپر بر می‌گرداند	<code>log(x)</code>
بین x و y بزرگترین مقدار را بر می‌گرداند. $x = 67.3$ و $y = 12.4$ برابر $\max(12.4, 67.3)$ $x = -12.4$ و $y = -67.3$ برابر $\min(-12.4, -67.3)$ خواهد بود	بین x و y بزرگترین مقدار را بر می‌گرداند.	<code>max(x, y)</code>
بین x و y کوچکترین مقدار را بر می‌گرداند. $x = 12.4$ و $y = -67.3$ برابر $\min(12.4, 67.3)$ $x = -12.4$ و $y = 67.3$ برابر $\max(-12.4, -67.3)$ خواهد بود	بین x و y کوچکترین مقدار را بر می‌گرداند.	<code>min(x, y)</code>

مقدار x به توان y را برمی گرداند.	$\text{pow}(x, y)$
مقدار سینوس x را برمی گرداند.	$\text{sin}(x)$
جذر عدد x را برمی گرداند.	$\text{sqrt}(x)$
مقدار تانژانت x را برمی گرداند.	$\text{tan}(x)$

جدول ۴-۴. متدهای کلاس Math

متدهای تودرتو

در جاوا این امکان وجود دارد تا یک متدهای خود را فراخوانی کند به فراخوانی یک متدهای خودش، «فراخوانی بازگشتی» یا **recursion** گفته می شود. بعضی مواقع برای انجام بعضی محاسبات لازم است تا عملیات متدهای خود را تکرار شود. محاسبه فاکتوریل یک عدد یکی از این نمونه عملیات هاست. فاکتوریل هر عدد برابر حاصلضرب تمام اعداد صحیح بین ۱ و خود آن عدد است برای مثال

```
1! = 1
2! = 2 * 1
3! = 3 * 2 * 1
4! = 4 * 3 * 2 * 1
n! = n * (n-1) * ... * 3 * 2 * 1
```

(فاکتوریل هر عددا را که با ! نشان می دهد)

برای محاسبه فاکتوریل یک عدد، می توانیم با شمارش از عدد ۱ تا آن عدد، هر یک از آن اعداد را درهم ضرب کنیم.

```
public static long fact(int n){
    long result=1;
    for(int i=1; i<n; i++) {
        result = result*i;
    }
    return result;
}
```

اما یک راه دیگر برای پیاده سازی فاکتوریل یک عدد فراخوانی بازگشتی است. در پیاده سازی این روش، این قاعده که فاکتوریل هر عدد برابر با حاصلضرب همان عدد در فاکتوریل عدد قبلی آن است استفاده می شود.

```
1! = 1
n! = n * (n - 1)!
```

متدهای که این قاعده را پیاده سازی می کند به صورت زیر خواهد بود

```
public static long fact(int n) {
    int result;
```

شروع جاوا با یک برنامه ساده

```
if(n==1)
    return 1;
return fact(n-1) * n;
}
```

در اینجا متدهای fact، حاصل فاکتوریل عدد n (پارامتر متدهاست) را محاسبه و بر می‌گرداند. این متده برای محاسبه فاکتوریل عدد n، اگر n برابر 1 باشد 1 را بر می‌گرداند و اگر n بزرگتر از 1 باشد حاصلضرب n در فاکتوریل n-1 را بر می‌گرداند.

تمرین: متدهای بنویسید که به روشهای بازگشتی و غیربازگشتی دنباله فیبوناچی را محاسبه کند. دنباله فیبوناچی که برای اعداد صفر و بزرگتر از صفر تعریف می‌شود دارای قانون زیر است:

```
fibonacci(0) = 0
fibonacci(1) = 1
fibonacci(2) = 1
Fibonacci(3) = 2
fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
```

فیبوناچی اعداد صفر و یک برابر خود این اعداد است اما فیبوناچی هر عدد دیگری برابر با مجموع فیبوناچی دو عدد قبلی آن است.

استفاده از پارامترهای متدهای main()

در جاوا مقدار پیش فرض برای پارامترهای متده وجود ندارد یعنی برای فراخوانی یک متده باید حتماً مقدار تمام پارامترهای متده را به ترتیب مشخص کنیم.

حال با دو سوال روبرو هستیم. اولاً متدهای main() که نقطه شروع اجرای برنامه است توسط کجا فراخوانی می‌شود؟ ثانیاً این متده، آرایه‌ای از رشته‌ها (String[] args) را به عنوان پارامتر دریافت می‌کند در حالیکه ما هیچگاه مقداری را به عنوان مقادیر پارامترهای این متده مشخص نکرده‌ایم.

پاسخ سوال اول این است که متدهای main() اساساً توسط برنامه نویس فراخوانی نمی‌شود بلکه «ماشین مجازی جاوا» این متده را فراخوانی و برنامه را اجرا می‌کند.

در پاسخ به سوال دوم باید بگوییم بله، متدهای main() یک آرایه از رشته‌ها را به عنوان پارامتر دریافت می‌کند. هر چند تاکنون از پارامترهای متدهای main() استفاده نکرده‌ایم و در تمام برنامه‌هایی که تاکنون نوشته‌ایم مقدار این پارامتر پوچ است اما می‌توان در پنجه فرمان جایی که برنامه اجرا می‌شود مقادیر این پارامترها را مشخص کرد.

توجه: از آنجاییکه متدهای main() مستقیماً توسط JVM فراخوانی می‌شود، در فراخوانی آن می‌توانید

هیچ پارامتری را مشخص نکنید، در اینصورت JVM متد main() را با یک پارامتر پوچ (null) فراخوانی می کند.

برای درک چگونگی انجام اینکار، تصور کنید که در یک برنامه بخواهیم پارامترهای متد main() را در صورت وجود نمایش دهیم. کلاس زیر انجام اینکار را نشان می دهد. به مثال زیر توجه کنید.

```

1  /*
2   this program shows using of main method's arguments
3   */
4  public class MainArgumantsDemo {
5      public static void main(String[] args) {
6          if(args!=null) {
7              for(int i=0; i<args.length; i++) {
8                  System.out.println(args[i]);
9              }
10         }
11     }
12 }
```

۲-۱۰ کد

پارامترهای متد main() با رشتہ هایی که با بعد از نام کلاس در پنجره دستور وارد می شوند مشخص می شوند. به عنوان مثال، اجرای کلاس فوق در پنجره دستور به صورت زیر

java MainArgumantsDemo Java Language Programming

نتیجه زیر را به همراه خواهد داشت.

Java
Language
Programming

توجه: یکی از کاربردهای پارامترهای متد main() تنظیمات برنامه است. مثلا در برخی برنامه ها، وقتی برنامه را اجرا می کنید این گزینه را دارد تا ظاهر برنامه (رنگ و شکل و شمایل برنامه)، زبان برنامه (فارسی، انگلیسی,...)، یا حتی نام کاربری و رمز ورود برنامه را از طریق پارامترهای متد main() مشخص کنید.

متعلقات (property)

احازه دهید مثال خود را اندکی تغییر دهیم. فرض کنید بخواهیم کلاس MathOperations را به گونه ای تغییر دهیم که ابتدا دو عدد را مشخص کنیم سپس متدهای جمع، تفریق، ضرب و تقسیم را برای انجام چهار عمل اصلی روی دو عدد فراخوانی کنیم. برای این منظور لازم است دو متغیر `x` و `y` داخل کلاس MathOperations تعریف کنیم که مقادیر دو عدد را نگهداری می کنند و توسط تمام متدهای کلاس قابل دسترسی هستند. کد ۴-۷ کلاس تغییریافته MathOperations را نشان می دهد.

```

1  public class MathOperations2 {
2
3      static int x;
4      static int y;
5
6      public static void printSum() {
7          int result = x + y;
8          System.out.println("sum: "+result);
9      }
10
11     public static void printSub() {
12         int result = x - y;
13         System.out.println("sub: "+result);
14     }
15
16     public static void printDiv() {
17         int result = x/y;
18         System.out.println("div: "+result);
19     }
20
21     public static void printMul() {
22         int result = x * y;
23         System.out.println("mul: "+result);
24     }
25 }
```

کد ۱۱

همانطور که قبلا هم گفته شد، به متغیرهایی که در یک کلاس و خارج از محدوده متدها تعریف می شوند «متعلقات» یا property های آن کلاس گفته می شود. دقت کنید که در تعریف متغیرهای `x` و `y` از کلمه کلیدی static استفاده شده است اگرچه در بیشتر مواقع متعلقات یک کلاس غیر static هستند اما در

این مثال ما مجبور هستیم که این متغیرها را static تعریف کنیم زیرا متدهایی که از این property‌ها استفاده می‌کنند خود به صورت static تعریف شده‌اند. در مورد این موضوع در فصلهای آینده به صورت مفصل صحبت خواهیم کرد. متدهای کلاس فوق، که قبلًا مقادیر عددی را به عنوان پارامتر دریافت می‌کردنند اینجا هیچ پارامتری دریافت نمی‌کنند و انجام محاسبات را روی متعلقات کلاس انجام می‌دهند.

توجه: متعلقات یک کلاس، متغیرهایی هستند که در محدوده کلاس و بیرون از بدنه متدها تعریف می‌شوند.

در تعریف یک property، همانند تعریف یک متدهایی است از کلمات کلیدی public و static استفاده شود معنی این کلمات را در مبحث طراحی شی گرا خواهید آموخت. اما حال که کلاس به شکل فوق تغییر کرد، لازم است کلاس Mathematics نیز به شکل زیر تغییر کند.

```

1 // Using mathematics operations defined in MathOperations2
2 public class Mathematics5 {
3
4     public static void main(String[] args) {
5         if(args == null) {
6             //no main parameters!
7             System.out.println("Provide two integer arguments");
8             return;
9         }else if(args.length < 2) {
10            //less main parameters!
11            System.out.print("We need two integer arguments");
12            return;
13        }
14
15        String value1 = args[0];
16        String value2 = args[1];
17        int a = Integer.parseInt(value1);
18        int b = Integer.parseInt(value2);
19        MathOperations2.x = a;
20        MathOperations2.y = b;
21        MathOperations2.printSum();
22        MathOperations2.printSub();
23        MathOperations2.printDiv();

```

شروع جاوا با یک برنامه ساده

```
24     MathOperations2.printMul();  
25 }  
26 }
```

کد ۲-۱۳

همانطور که ملاحظه می کنید لازم است قبل از فراخوانی متدها، **property** های کلاس مقداردهی می شوند. اینکار مشابه فراخوانی یک متدهای کلاس انجام می شود که پس از نام کلاس، یک نقطه، و سپس نام **property**، مقداری به آن انتساب داده شده است.

در کلاس فوق، همچنین از متدهای **parseInt()** از کلاس **Integer** استفاده شده است تا یک رشته عددی را به یک عدد صحیح تبدیل کنیم. **Integer** یکی از کلاس‌های جاواست.

خاصصه

در این فصل با ساختار اصلی یک برنامه و تعریف کلاس آشنا شدید. هر کلاس با یک نام که بعد از کلمه کلیدی `class` می‌آید تعریف می‌شود بعد از آن بدنئ کلاس با } شروع و با { خاتمه می‌یابد. هر کلاس، باید در یک فایل متنی با نام همان کلاس و با پسوند `.java` ذخیره شود.

هر کلاس ممکن است حاوی یک یا چندین متده باشد، هر متده با یک نام، تعدادی داده‌های ورودی (پارامترها)، و یک مقدار برگشتی مشخص می‌شود اما متده می‌تواند داده ورودی نداشته باشد یا مقدار برگشتی آن `void` باشد که به معنی نداشتن مقدار برگشتی است.

برای اجرای هر برنامه جاوا، لازم است یک متده `main()` به صورت زیر در یک کلاس تعریف شده باشد:

```
public static void main(String[] args) {
    .
    .
    .
}
```

در حقیقت نقطه شروع اجرای هر برنامه جاوا، یک متده `main` دقیقاً به صورت فوق است این متده تعدادی رشته را به عنوان ورودی دریافت می‌کند هریک از این رشته‌ها از طریق کنسول و در هنگام اجرای برنامه به آن ارسال می‌شوند در صورتیکه هیچ رشته‌ای به این متده ارسال نشود `args` مقدار `null` یا تهی خواهد داشت.

یکی از کاربردهای کلاس، دسته بندهای مشابه یا مرتبط است. فراخوانی یک متده (در صورتیکه در تعریف آن متده کلیدی `static` استفاده شده باشد) با نام بردن از نام کلاس و یک نقطه (.) نام آن متده انجام می‌شود.

متده `parseInt()` در کلاس `Integer` قرار دارد و کار آن دریافت یک رشته و تبدیل آن به معادل عددی آن رشته است این رشته باید قابل تبدیل به عدد باشد، یعنی تنها حاوی کاراکترهای عددی باشد. همچنین در هر برنامه می‌توان متغیرهای رشته‌ای یا عددی تعریف کرد، هر متغیر در حقیقت قسمتی از حافظه است که به آن یک نام اختصاص داده شده است و از آن می‌توان به منظور نگهداری یک مقدار استفاده نمود متغیرهای عددی امکان می‌دهند تا روی یک داده پردازشی انجام شده و مقدار آن در طول یک فرایند دستکاری شود.

نامگذاری کلاسهای، متدها و متغیرها از قوانین خاص نامگذاری پیروی می‌کنند. در نامگذاری می‌توان از تمام کاراکترهای حروف الفبای زبان انگلیسی، کاراکترهای عددی و کاراکترهای خاص _، \$ استفاده نمود کاراکترهای عددی نمی‌توانند به عنوان اولین کاراکتر در یک نام استفاده شوند و بهتر است که از کاراکترهای خاص نیز استفاده نشود.

در نامگذاریها همواره بهتر است نام کلاس با حرف بزرگ الفبای انگلیسی، نام متده یا متغیر نیز با حرف کوچک الفبای انگلیسی شروع شود این قراردادی است که تقریباً تمام برنامه نویسان جاوا از آن پیروی می‌کنند.

تمرینات

کلاسی بنویسید و در آن متدهای زیر را پیاده سازی کنید:

۱) متدهای بنویسید که دو عدد را از کاربر دریافت کند سپس

در صورتیکه دو عدد مساوی باشند پیغام

"These numbers are equals"

را در خروجی استاندار چاپ کند.

در صورتیکه عدد اول بزرگتر از عدد دوم باشد پیغام "First is bigger" را چاپ

کند.

در صورتیکه عدد دوم بزرگتر از عدد اول باشد پیغام "First is smaller" را

چاپ کند.

۲) برنامه ای بنویسید که دو عدد را بصورت پارامتر دریافت کند و مجموع آنها را در خروجی استاندارد

چاپ کند در صورتیکه یکی از اعداد وارد نشود آنرا صفر فرض کند و در صورتیکه دو عدد وارد نشود

با یک پیغام مناسب از برنامه خارج شود.

۳) برنامه ای بنویسید که مربع و مکعب اعداد ۰ تا ۱۰ را بصورت زیر در خروجی استاندارد چاپ کند.

Number	Square	Cube
1	1	1
2	4	8
3	9	27
.	.	.
.	.	.
10	100	1000

۴) برنامه ای بنویسید که یک عدد بزرگتر از صفر را دریافت کند و سپس مقدار فاکتوریل آن عدد را محاسبه و در خروجی استاندارد چاپ کند (فاکتوریل هر عدد برابر حاصلضرب تمام اعداد صحیح کوچکتر از آن عدد و بزرگتر از ۱ است).

- (۵) برنامه ای بنویسید که حقوق و دستمزد کارگران یک شرکت را محاسبه و چاپ می کند. هر کارگر، ساعت کار و دستمزد مشخصی دارد حقوق هر کارگر در ماه از طریق حاصلضرب تعداد ساعت کار در دستمزد ساعتی وی محاسبه می شود، برنامه ای بنویسید که اطلاعات ساعت کار و دستمزد ۱۰ کارگر را دریافت کند و پس از محاسبه حقوق ماهیانه، نتیجه را در خروجی استاندارد چاپ کند.
- (۶) برنامه ای بنویسید که تعداد نامشخصی از اعداد را دریافت کند و کوچکترین و بزرگترین بین آنها را در خروجی استاندارد چاپ کند (برای این کار می توانید از عدد اول برای تعیین تعداد اعدادی که دریافت می شوند استفاده کنید)
- (۷) متدهای به نام `isSquare()` بنویسید که دو عدد به عنوان پارامتر دریافت کند، بررسی کند که آیا عدد دوم مربع عدد اول هست یا خیر، در صورتیکه مربع عدد اول باشد "is square" و در غیر اینصورت "is not square" برگرداند.
- (۸) متدهای بنویسید که یک عدد صحیح به عنوان پارامتر دریافت می کند سپس مربعی با ضلعی به طول همان عدد (با استفاده از کاراکتر های `*`) در خروجی استاندارد چاپ می کند.
- (۹) متدهای فوق را بگونه ای تغییر دهید که این متدها علاوه بر عدد صحیح یک پارامتر کاراکتری نیز دریافت کنند سپس مربعی با ضلعی به طول همان عدد و با استفاده از کاراکتری که دریافت نموده در خروجی استاندارد چاپ کند.
- (۱۰) یک عدد `Perfect` به عددی گفته می شود که از مجموع اعداد کمتر از خودش محاسبه شود مثلاً ۶ یک عدد `Perfect` است زیرا $1 + 2 + 3 = 6$ یا 10 یک عدد `Perfect` است زیرا $1 + 2 + 3 + 4 = 10$
- متدهای بنویسید که یک عدد صحیح را به عنوان پارامتر دریافت کند و تعیین کند آیا آن عدد `Perfect` است یا خیر.
- (۱۱) متدهای بنویسید که یک عدد به عنوان پارامتر دریافت کند و تعیین کند آیا آن عدد، اول است یا خیر.
- (۱۲) متدهای بنویسید که دو عدد صحیح دریافت کند و تعیین کند که آیا آن دو عدد را محاسبه کرده و برگرداند.

۳

آشنایی با محیط برنامه نویسی IntelliJ IDEA

اگرچه برای برنامه نویسی جاوا از هر ویرایشگر متنی (مثلا Notepad) می‌توانید استفاده کنید اما نوشتن کدهای برنامه در یک ویرایشگر ساده همانند Notepad کار سختی خواهد بود زیرا Notepad هیچ کمکی در کدنویسی شما نمی‌کند شما باید قوانین و دستورات زبان جاوا را حفظ باشید اگر چیزی را فراموش کنید باید با مراجعه به یک کتاب یا منابع دیگر آنرا با خاطر آورید. در عمل هیچ برنامه نویس جاوابی از Notepad یا ویرایشگرهای مشابه برای کدنویسی جاوا استفاده نمی‌کند در عوض محیطهای حرفه‌ای و تخصصی تولید شده اند که کدنویسی جاوا را بیش از پیش برای شما آسان می‌کنند، اشتباهات سهوی در کدنویسی شما را کشف و به شما گزارش می‌کنند، امکانی به نام «تکمیل کد» دارند که می‌توانند به صورت خودکار کدهای شما را تکمیل و به این ترتیب سرعت کدنویسی شما را بالا ببرند و برنامه نویسی را به کاری بسیار لذت بخش و جذاب تبدیل کنند. به این محیطهای اصطلاحاً IDE گفته می‌شود.

برای زبان جاوا IDE‌های مختلفی وجود دارند که سه تای آنها که از بقیه پرطرفدارتر و رایج‌تر هستند در جدول ۱-۳ توضیح داده شده‌اند.

نام ویرایشگر	تولید کننده	مجوز استفاده	توضیح
IntelliJ	JetBrains	تجاری و رایگان، نسخه تجاری امکانات بسیار فراوانی نسبت به بهترین ویرایشگر و محیط برنامه	به نظر نویسنده، این ویرایشگر

نویسی جاواست. هوش فوق العاده، سرعت بالا و سبکی از مزایای این محیط برنامه نویسی جاواست.	نسخه رایگان دارد اما برای یادگیری جاوا استاندارد، نسخه رایگان کفایت می کند.		IDEA
امکانات فراوان و فوق العاده از مزیت های این ویرایشگر است، اما استفاده از آن و منوهای آن به سادگی IntelliJ نیست برای یادگیری آن نسبتاً زمان بیشتری نیاز است.	رایگان	Oracle	NetBeans
این محیط برنامه نویسی، منبع-باز و رایگان است و محبوبترین محیط توسعه جاواست این محیط زبانهای دیگر از قبیل C و PHP را نیز پشتیبانی می کند. اشکال این ویرایشگر وجود باگهای فراوان آن است. البته هسته آن تقریباً بدون باگ است و باگها بیشتر در افزونه های آن (plugin‌های آن) ظاهر می شود.	رایگان	Eclipse	Eclipse

جدول ۱-۳. ابزارهای رایج جاوا

اگر با چند برنامه نویس جاوا مشورت کنید هر کدام ممکن است با سلیقه خود یکی از محیطهای فوق یا حتی محیط های دیگر را پیشنهاد کند و در این مورد نیز ممکن است بسیار تعصب داشته باشد. واقعیت این است که اگر هر یک از ویرایشگرهای فوق را انتخاب کنید پس از مدتی با آن مانوس می شوید و پس از اینکه چم و خم و جزئیات آن را یادگرفتید دیگر حاضر نیستید آنرا با ویرایشگر دیگری جایگزین کنید. در ادامه با محیط روپایی IntelliJ IDEA آشنا می شویم.

نصب IntelliJ IDEA

IntelliJ IDEA را می توانید مستقیماً از وبسایت <http://www.jetbrains.com> دانلود کنید یا اگر این کتاب را به همراه سی دی تهیه کرده اید از نسخه ای که در سی دی کتاب قرار دارد استفاده کنید. برای

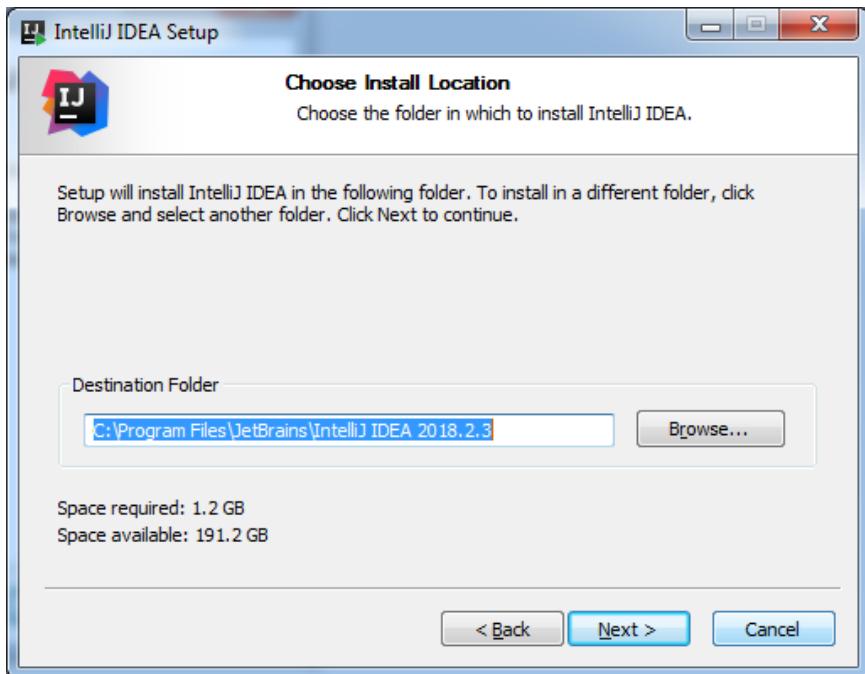
آشنایی با محیط برنامه نویسی IntelliJ IDEA

استفاده از IntelliJ IDEA قبل از هرچیز باید آنرا روی سیستم خود نصب کنید، نصب آن کار بسیار آسانی است و لازم است با کلیک کردن روی فایل اجرایی **Setup.exe** نصب آنرا شروع کنید که در ابتدا با پنجره زیر شروع می شود.



شکل ۱-۳. مرحله اول

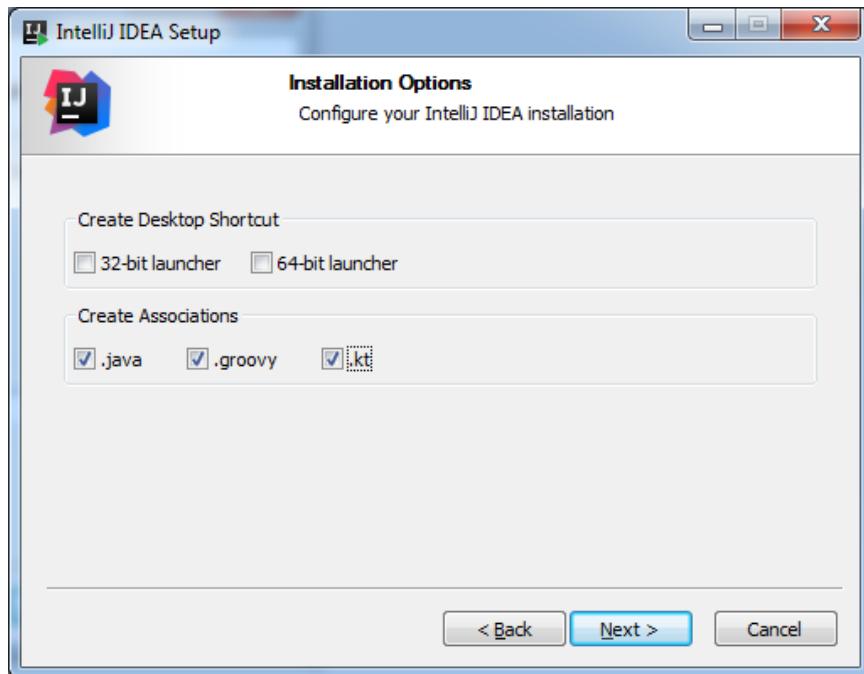
با کلیک روی گزینه **Next** پنجره زیر به شما نشان داده می شود که از شما می خواهد مسیر نصب را مشخص کنید در اغلب مواقع مسیر پیش فرضی که به شما پیشنهاد می کند مناسب است مگر اینکه درایو شما پُر باشد و لازم باشد تا مسیر دیگری را در درایو دیگری مشخص کنید.



شکل ۲-۳. انتخاب مسیر نصب

بعد از انتخاب مسیر نصب، با کلیک روی **Next** پنجره زیر نمایش می‌یابد که با تغییر تنظیمات ویندوز فایلهای جاوا و Groovy و Kotlin را به خود اختصاص می‌دهد. به این ترتیب بعداً با کلیک روی فایلهایی با پسوند `.java`, `.kt`, `.groovy` و `.xml` برای نمایش و ویرایش آنها اجرا شود.

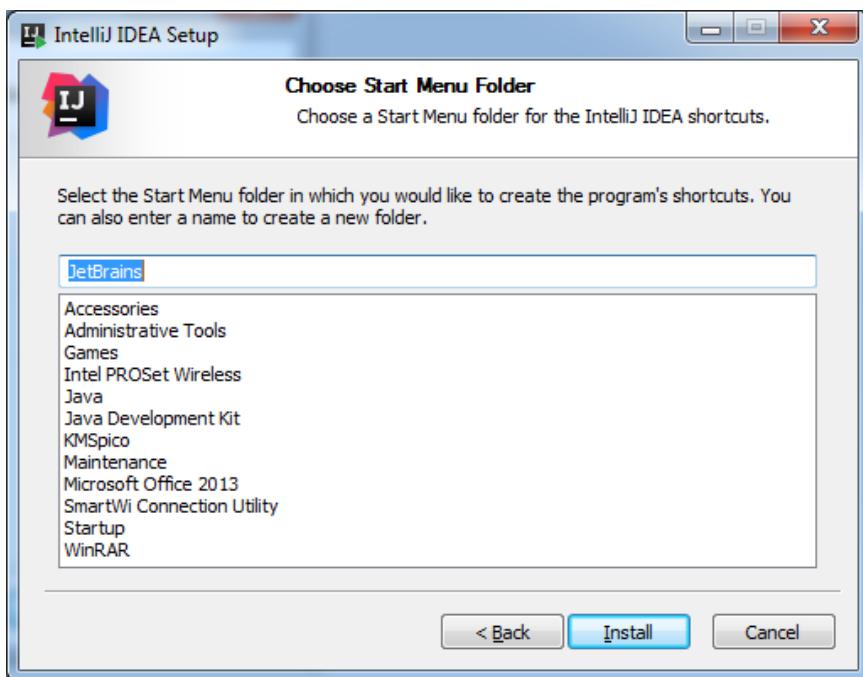
آشنایی با محیط برنامه نویسی IntelliJ IDEA



شکل ۳-۳. اختصاص فایلهای Java، groovy و kotlin به IntelliJ

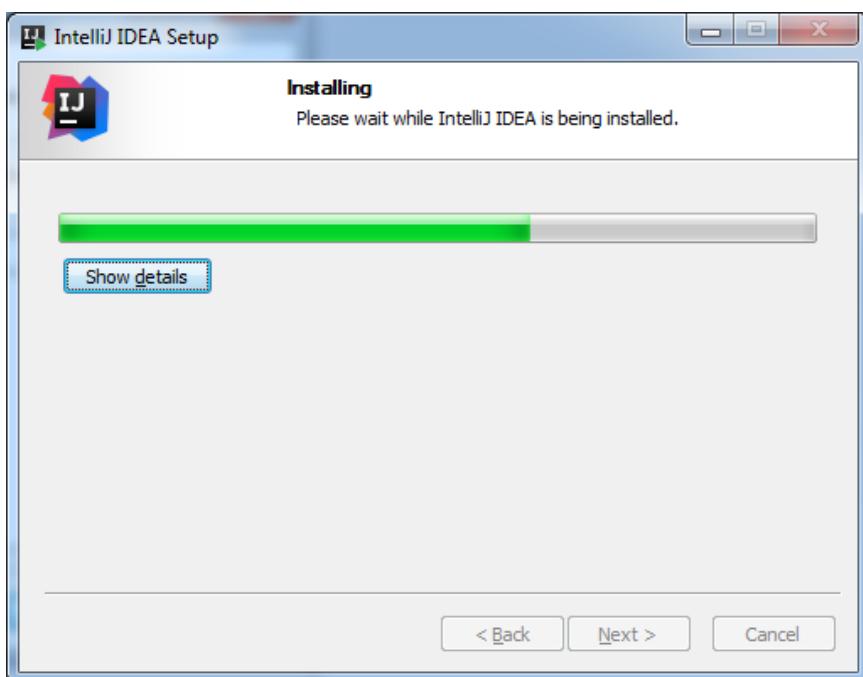
توجه: یکی از خصوصیات ماشین مجازی Java این است که می‌تواند برنامه‌هایی که به برخی زبانهای دیگر نوشته می‌شوند را اجرا کند. kotlin، groovy، Java اسکریپت، و اسکالا (Scala) چهار زبان اسکریپتی مختلف هستند که توسط ماشین مجازی Java اجرا می‌شوند.

پس از کلیک روی گزینه Next پنجره شکل ۳-۴ نمایش می‌یابد تا نام شاخه‌ای در Start Menu مشخص کنید تا IntelliJ IDEA در آن شاخه نصب شود. به نظر می‌رسد نام پیش‌فرض پیشنهادی مناسب باشد!



شکل ۴-۴. انتخاب نام شاخه Start Menu که IntelliJ در آن نصب می شود.

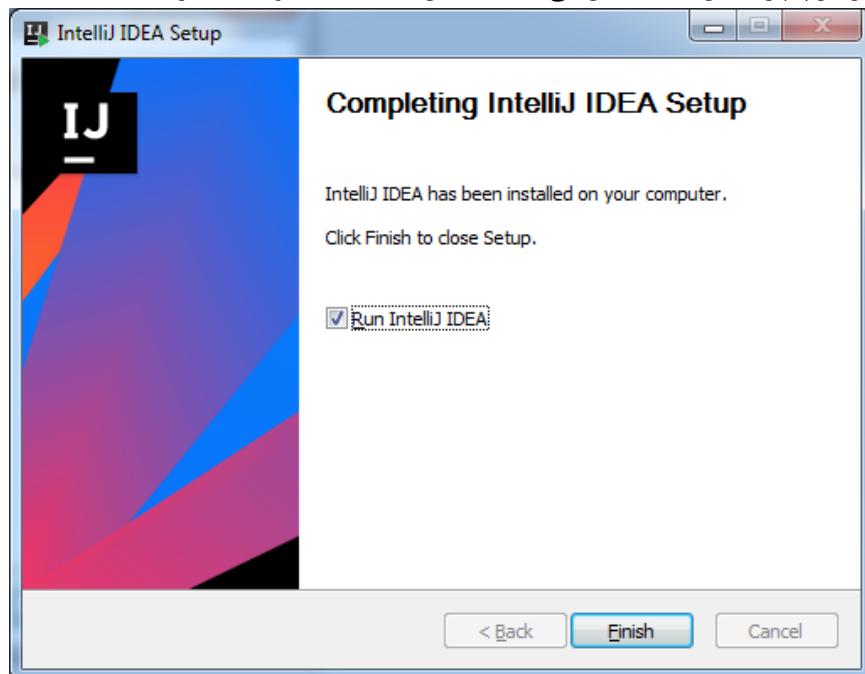
با انتخاب گزینه Install، نصب IntelliJ شروع می شود و پنجره ۴-۵ نمایش می یابد.



آشنایی با محیط برنامه نویسی IntelliJ IDEA

شکل ۵-۳. در حال نصب

در آخر نیز پنچرۀ شکل ۳-۶ نمایش می یابد که نشان دهنده نصب موفقیت آمیز IntelliJ IDEA است.

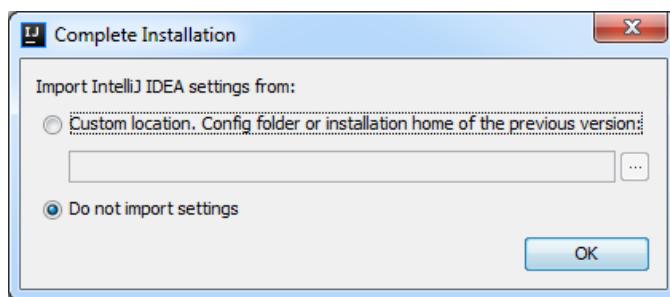


شکل ۶-۳. نصب موفقیت آمیز IntelliJ IDEA

با انتخاب گزینه Run IntelliJ IDEA، اجرا می شود.

اجرای IntelliJ

با اجرای IntelliJ که از طریق Start Menu و یافتن شاخه ای که در آن نصب کرده اید امکانپذیر است، پنجرۀ زیر نمایش می یابد.



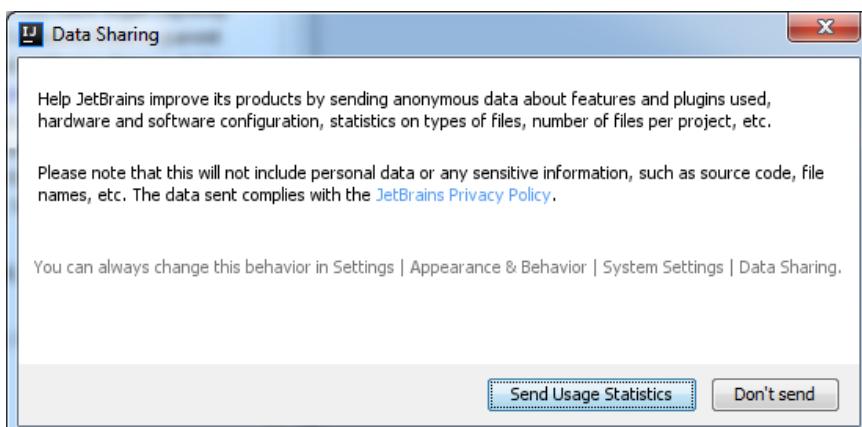
شکل ۷-۳. بارگذاری یا عدم بارگذاری تنظیمات قبلی

که از شما می خواهد در صورتیکه قبل از IntelliJ را نصب کرده اید و اینک می خواهید تنظیمات نسخه قبلی را به نصب جدید منتقل کنید در این مرحله آنرا انجام دهید. از آنجاییکه اولین بار است که IntelliJ را نصب می کنید گزینه دوم را مطابق شکل فوق انتخاب کنید و گزینه OK را کلیک کنید تا پنجره ۳-۸ نمایش یابد.



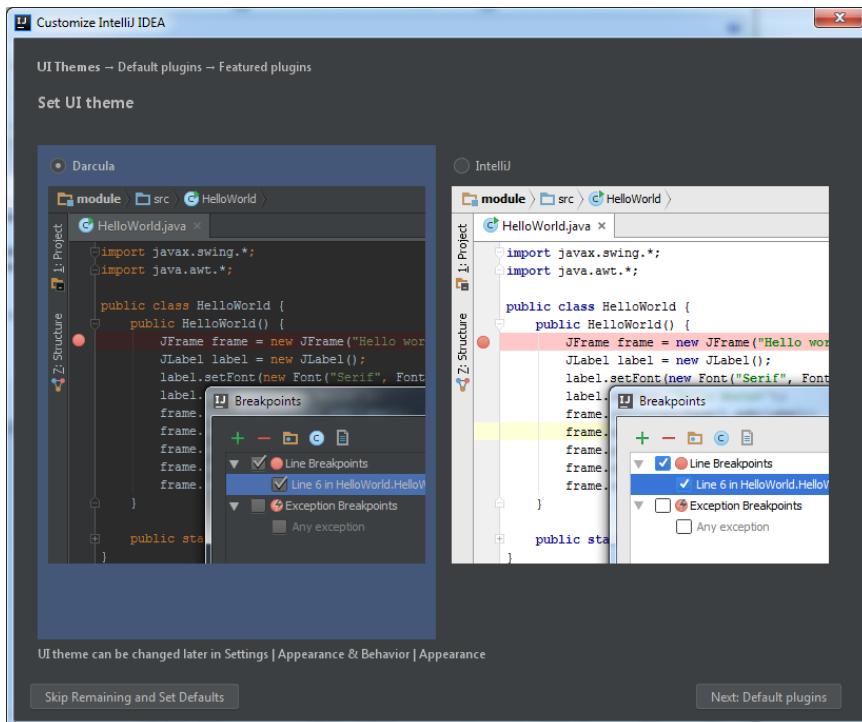
شکل ۷-۳. پذیرش قرارداد استفاده از IntelliJ IDEA

در ابتدا گزینه Accept غیرفعال است، اسکرول را به پایین بکشید (که به معنی این است که تمام مفاد قرارداد استفاده از IntelliJ را مطالعه کرده اید) و در اینصورت با فعال شدن گزینه Accept آنرا کلیک کنید تا پنجره شکل ۳-۹ نمایش یابد.



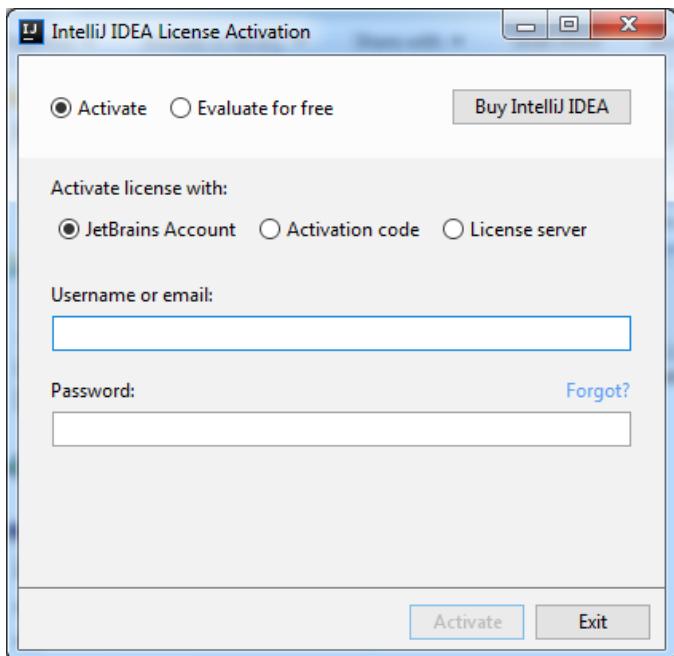
شکل ۹-۳. ارسال یا عدم ارسال بازخورد به شرکت JetBrains

در این مرحله IntelliJ از شما می خواهد تا در صورت امکان به آن اجازه دهید تا اطلاعات آماری برنامه هایی که با آن می نویسید را به شرکت JetBrains که سازنده IntelliJ است ارسال کند. بهتر است گزینه Don't Send را انتخاب کنید. به هر حال پنجره شکل ۹-۳ نمایش می یابد.



شکل ۹-۳. انتخاب تم

که امکان می دهد شما تم مورد نظرتان را انتخاب کنید. IntelliJ از دو تم سفید و خاکستری پشتیبانی می کند که هر کدام علاوه‌بر خودش را دارد. تم مورد نظرتان را در این مرحله انتخاب کنید هر چند بعداً می توانید از طریق تنظیمات IntelliJ نیز آنرا تغییر دهید. با انتخاب گزینه Skip Remaining and Set Defaults تنظیمات پیش فرض را قبول کنید و به مرحله بعد، شکل ۱۱-۳ بروید.



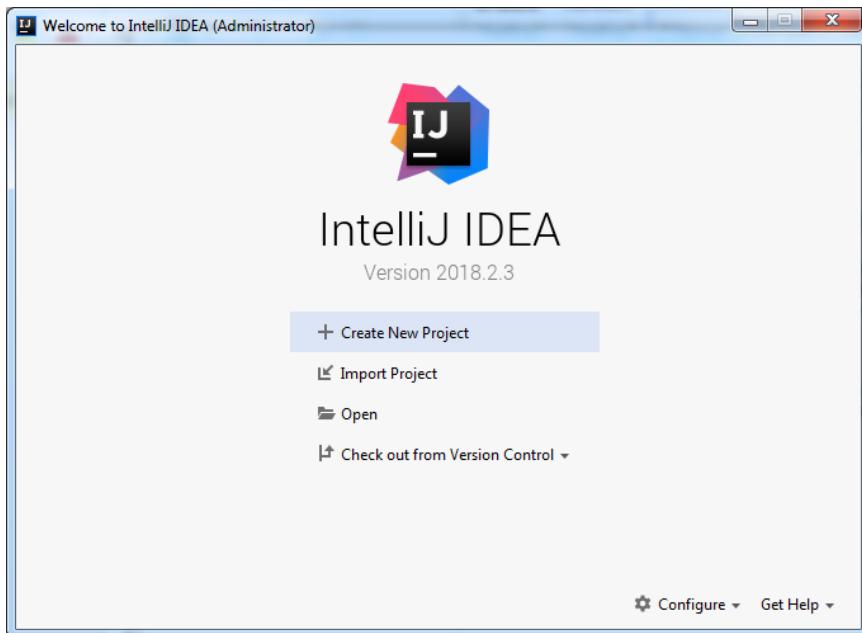
شکل ۱۱-۳. ورود شماره سریال خرید

در اینجا از شما خواسته می شود که کد فعالسازی را وارد کنید اگر آنرا خریداری کرده باشد، شرکت JetBrains به شما کد فعال سازی را می دهد اما از آنجاییکه در ایران امکان خرید این نرم افزار وجود ندارد Evaluate for free تقریبا همه آنرا crack می کنند. اگر نخواهید آنرا Crack کنید می توانید با انتخاب گزینه Crack به مدت ۳۰ روز از آن استفاده کنید.

شروع کار با IntelliJ

به هر حال وقتی این برنامه را اجرا کنید پنجره ۱۲-۳ نمایش داده می شود.

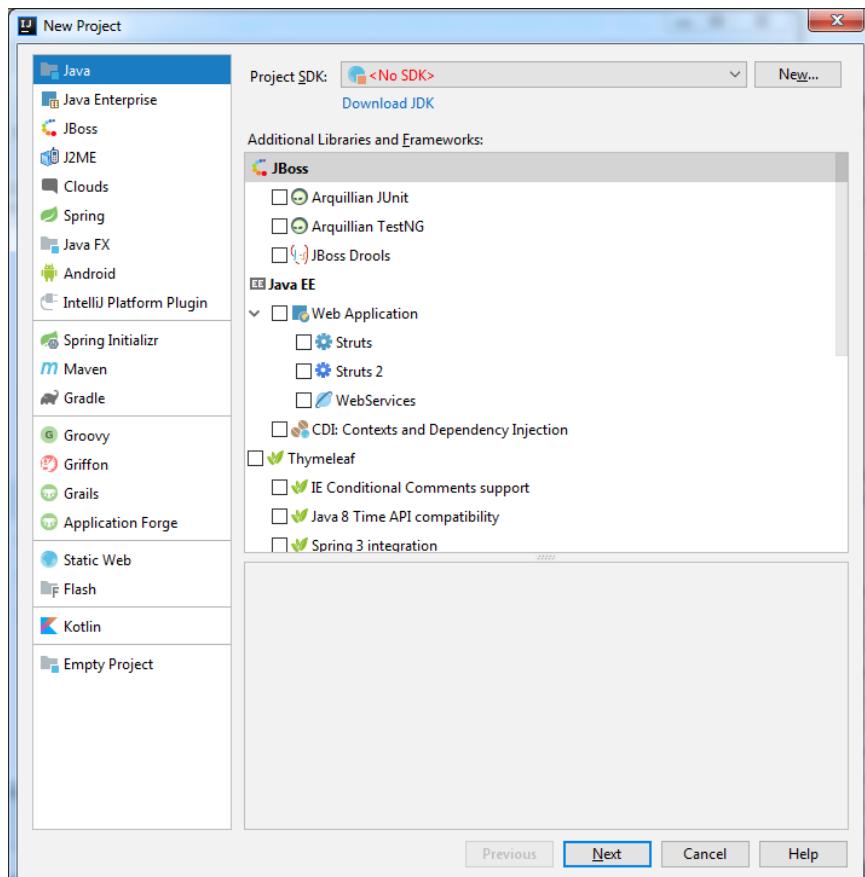
آشنایی با محیط برنامه نویسی IntelliJ IDEA



شکل ۳-۱۲. ایجاد یک پروژه جدید

برای شروع کدنویسی در ابتدا باید یک پروژه ایجاد کنید پروژه اصلاحی است که به کدهای یک برنامه و تنظیمات آن برنامه گفته می شود. وقتی با IntelliJ یک پروژه ایجاد می کنید، یک دایرکتوری روی سیستم شما ایجاد می شود که تمام فایلهای برنامه در آن ذخیره می شوند. علاوه بر فایلهای برنامه (از قبیل کلاس‌های برنامه یا هر فایل دیگری که در برنامه ایجاد می کنید)، IntelliJ یک فایل مخصوص به خود با پسوند .iml نیز در شاخه پروژه ایجاد می کند که در آن تنظیمات برنامه را نگهداری می کند.

برای ایجاد یک پروژه روی **Create New Project** در پنجره فوق کلیک کنید تا پنجره شکل ۳-۱۳ نمایش یابد.

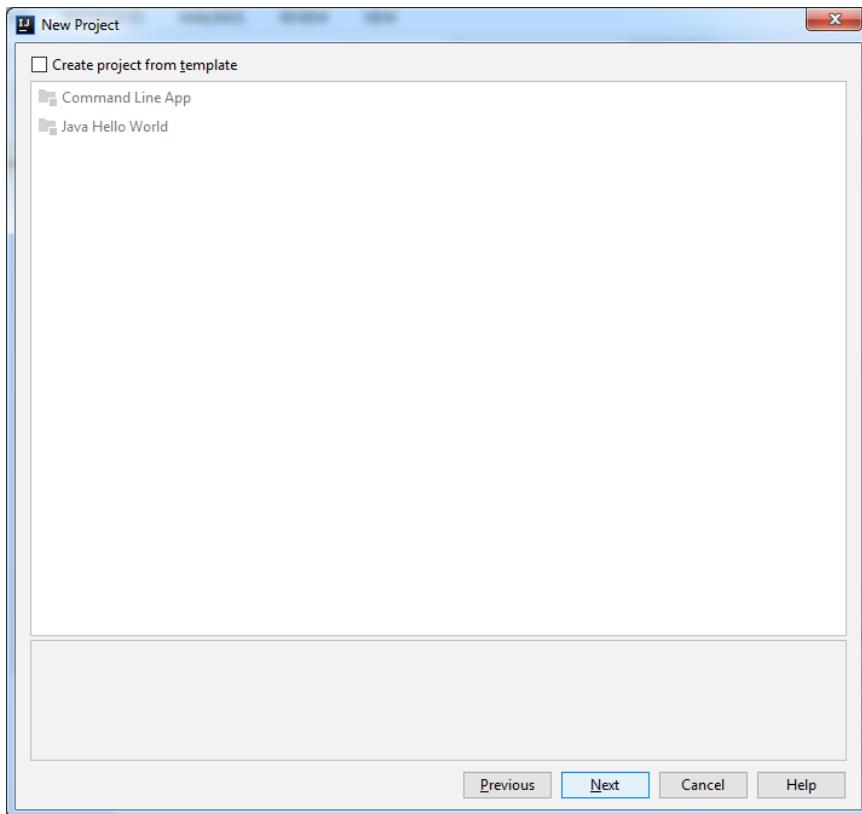


شکل ۱۳-۳. مشخص کردن مسیر JDK و انتخاب نوع پروژه

در قسمت Project SDK باید با کلیک کردن روی دکمه New مسیری که JDK را نصب کرده اید مشخص کنید.

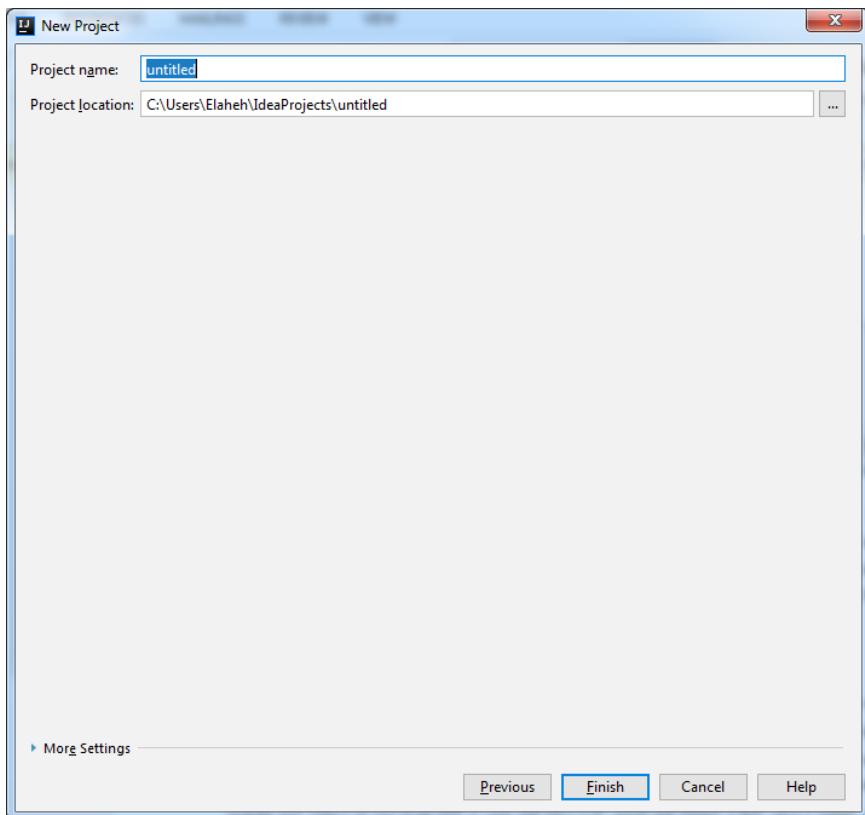
در قسمت سمت چپ لیستی از انواع پروژه هایی که می توان با IntelliJ تولید کرد آمده است که می توان انتخاب نمود. اما از آنجاییکه ما در این کتاب صرفا برنامه نویسی جاوا استاندارد را می آموزیم با گزینه های دیگر کاری نداریم و با کلیک روی Next به مرحله بعدی می رویم.

آشنایی با محیط برنامه نویسی IntelliJ IDEA



شکل ۱۴-۳. استفاده از قالب‌های آماده برای پروژه جدید

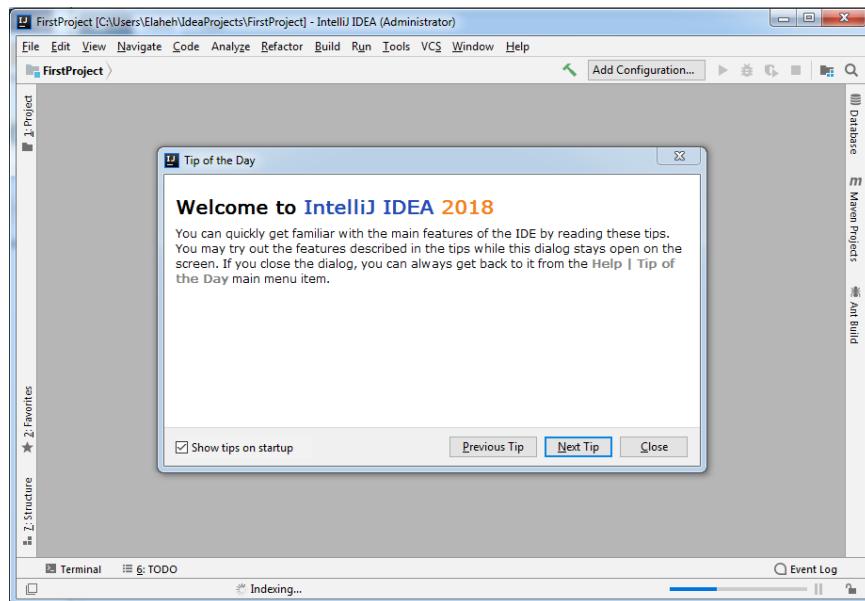
در این مرحله از شما خواسته می‌شود تا در صورت تمایل از قالب‌های آماده برای پروژه خود استفاده کنید. قالب‌هایی که در اینجا لیست شده‌اند چیز قابل توجه‌ای به پروژه اضافه نمی‌کنند و ما از خیر آن می‌گذریم اما شما برای تفمن و رفع حس کنجکاوی می‌توانید آنها را انتخاب کنید و نتیجه را مشاهده کنید. با انتخاب گزینه **Next** به مرحله بعدی یعنی انتخاب نام و مسیر پروژه می‌رویم.



شکل ۱۵-۳. انتخاب نام و مسیر برای پروژه

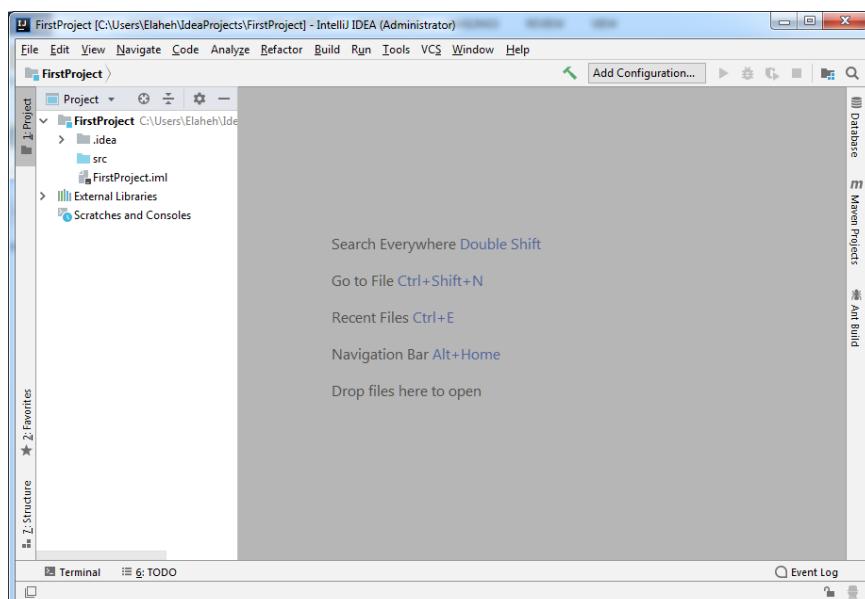
پس از این مرحله و با انتخاب گزینه **Finish** پروژه ایجاد می شود و پنجره زیر نمایش می یابد.

آشنایی با محیط برنامه نویسی IntelliJ IDEA



شکل ۱۶-۳. محیط پروژه

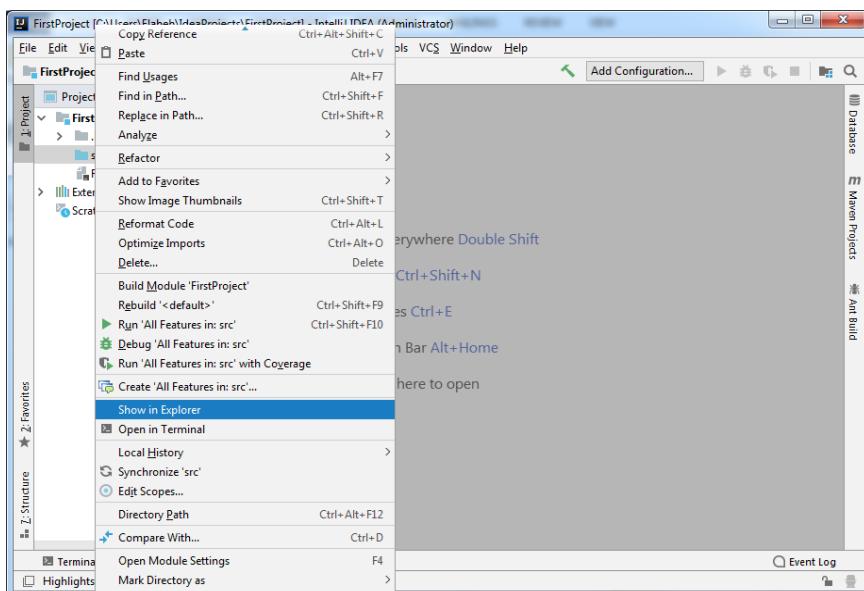
روی Close کلیک کنید تا دیالوگ وسط بسته شود و سپس با کلیک روی کلمه Project در قسمت بالای سمت چپ پنجره، شکل پنجره به صورتیکه در شکل ۱۷-۳ نشان داده شده نمایش می یابد.



شکل ۱۷-۳. ساختار پروژه

در سمت چپ، درختی را مشاهده می کنید که ریشه آن نام پروژه و مسیری که پروژه در آن ذخیره شده است را نشان می دهد. داخل این شاخه همانطور که مشخص است، شاخه های idea و src و فایل FirstProject.iml وجود دارند. داخل این درخت، فایلها و شاخهای پروژه را سازماندهی می کنید. مثلا تمام کلاس‌های جاوا (که همگی در فایل‌های با پسوند .java ذخیره می شوند) در شاخه src قرار داده می شوند. همچنین بنا به نیاز ممکن است شاخه های دیگری در کنار src ایجاد کنید و فایل‌هایی از انواع مختلف را در آنها قرار دهید. مثلا می توانید یک شاخه با نام img ایجاد کنید و فایل تمام عکسها و تصاویری که در پروژه استفاده می شوند را در آن قرار دهید.

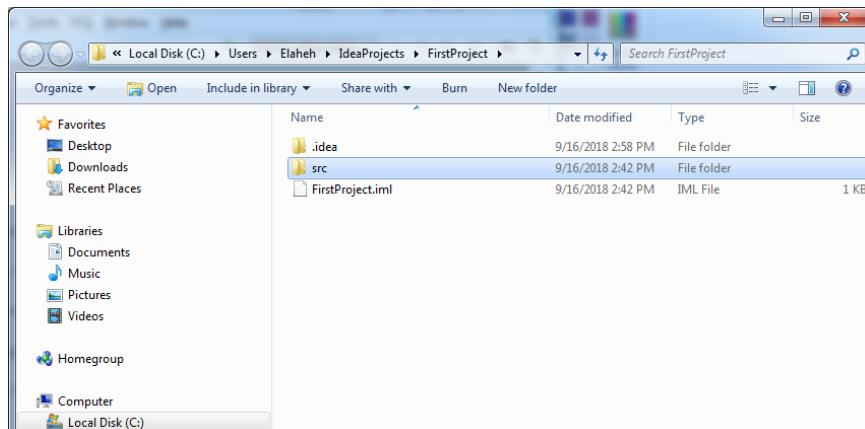
اگر روی گره ریشه یا هر فایل یا شاخه ای که در پروژه وجود دارد به صورتی که در شکل زیر نشان داده شده است کلیک راست کنید و گزینه Show In Explorer را انتخاب کنید محل فیزیکی که آن فایل یا شاخه در آن ذخیره شده است به شما نشان داده خواهد شد



شکل ۱۶-۳. نمایش محل فیزیکی پروژه در سیستم

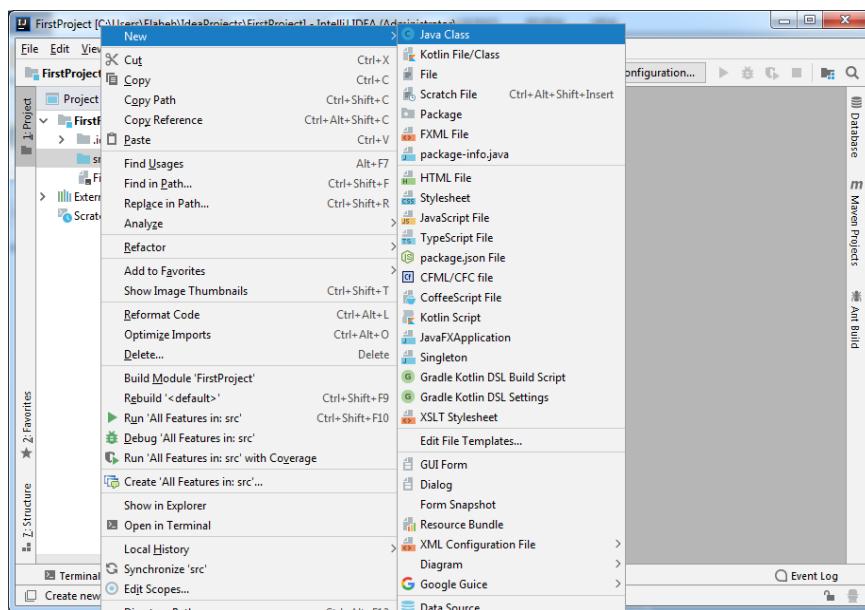
در این شاخه فایلهایی که در درخت پروژه مشاهده می شوند را می توانید بباید

آشنایی با محیط برنامه نویسی IntelliJ IDEA



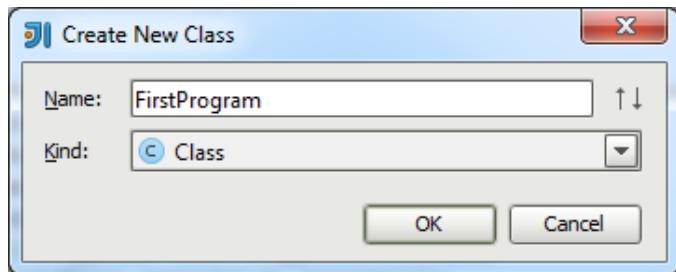
شکل ۱۹-۳. محل فیزیکی پروژه در سیستم

حال اجازه دهید برنامه ساده‌ای که در فصل پیش نوشته‌یم را با استفاده از IntelliJ IDEA کامپایل و اجرا کنیم. برای این منظور روی شاخه src کلیک راست کنید و گزینه New و از لیست بعدی گزینه Java Class را انتخاب کنید.



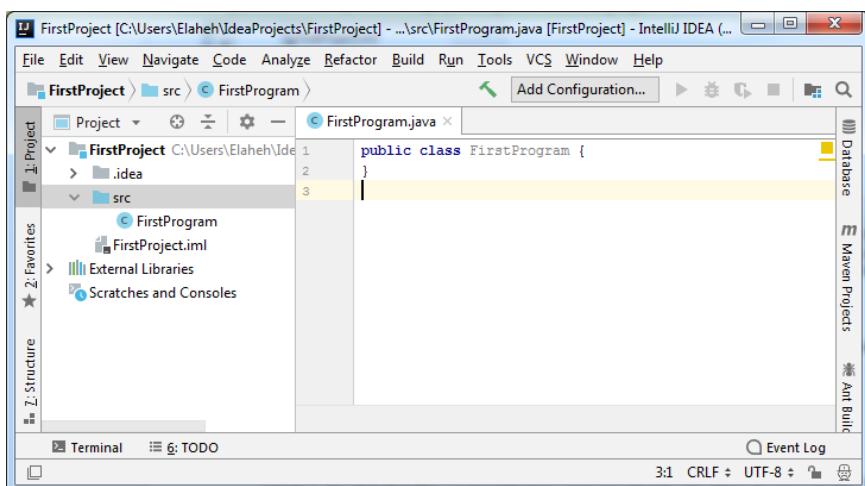
شکل ۲۰-۳. ایجاد کلاس

از شما می‌خواهد نام کلاسی که قصد ایجاد آنرا دارید وارد کنید.



شکل ۲۱-۳. انتخاب نام برای کلاس

در اینجا ما نام اولین کلاسی که نوشته ایم یعنی FirstProgram را وارد نموده ایم، تا IntelliJ به صورتی که در شکل زیر می بینید



شکل ۲۲-۳. کدهای تولید شده توسط IntelliJ IDEA

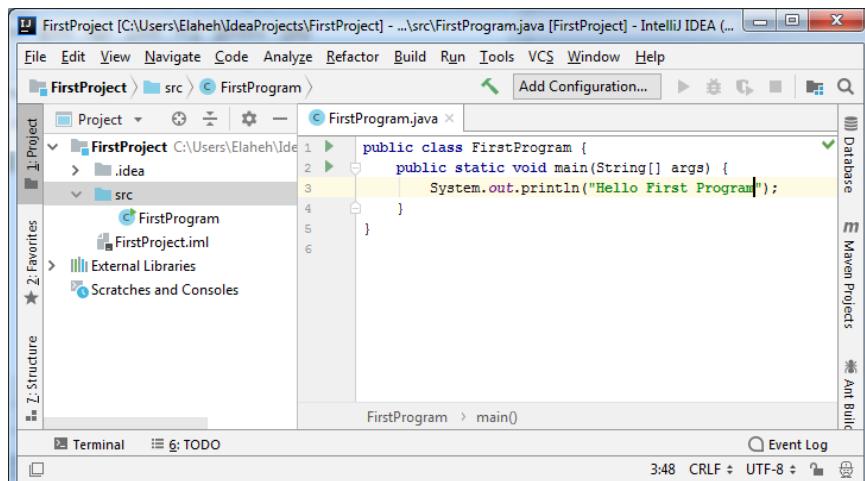
همانطور که ملاحظه می کنید، IntelliJ یک فایل با نام src در شاخه FirstProgram.java ایجاد می کند و بدنه اصلی کلاس یعنی جملة

```
public class FirstProgram {  
}
```

را نیز در آن ذخیره می کند تا شما آنرا تکمیل کنید. در بالای سمت راست محیط ویرایشگر، یک مربع رنگی وجود دارد که وضعیت کدهای آن کلاس را به شما گزارش می دهد. اگر این مربع به رنگ سبز باشد، نشان می دهد که کدهای شما در بهترین وضعیت هستند یعنی هیچ خطای کامپایل در آنها وجود ندارد و شما تمام نکات و شرایط کدنویسی آن کلاس را رعایت کرده اید. اگر این مربع زردرنگ باشد (به صورتیکه در شکل فوق مشاهده می کنید) نشاندهنده این است که هیچ خطای گرامری در کد شما وجود ندارد و به درستی کامپایل

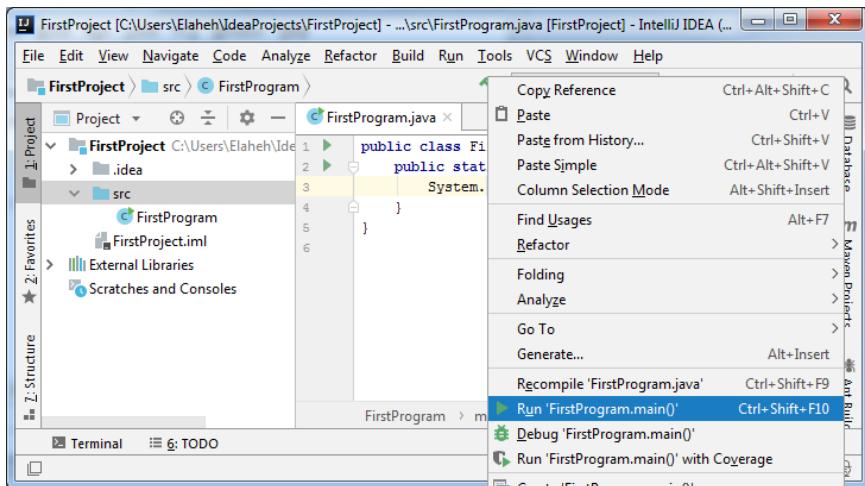
آشنایی با محیط برنامه نویسی IntelliJ IDEA

خواهد شد اما در مورد شکل کدنویسی، رعایت نکات و شرایط کدنویسی برخی اخطارها وجود دارد. در اغلب مواقع این اخطارها جدی نیستند و کد شما با موفقیت کامپایل و اجرا خواهد شد اما در برخی موارد خاص ممکن است که شما در زمان اجرا خطأ بدهد یا نتیجهٔ مورد نظر شما را نداشته باشد با گذشت زمان و کسب تجربه، شما متوجه خواهید شد که کدام اخطارها جدی و کدامیک غیرهمم هستند اما حال که کم تجربه هستید به آنها توجه کنید و با حرکت موس به خطی که اخطار در آن وجود دارد به متن اخطار و نحوه رفع آن توجه کنید. اگر این مرتع، قرمزرنگ باشد بدین معناست که در گُد شما خطای گرامری وجود دارد و کامپایل نخواهد شد با حرکت موس به خطی از برنامه که در آن خطای خط قرمز قرار دارد به پیغام خطأ و نحوه رفع آن توجه کنید. حال شروع به تایپ کنید و کدهای برنامه را بنویسید.



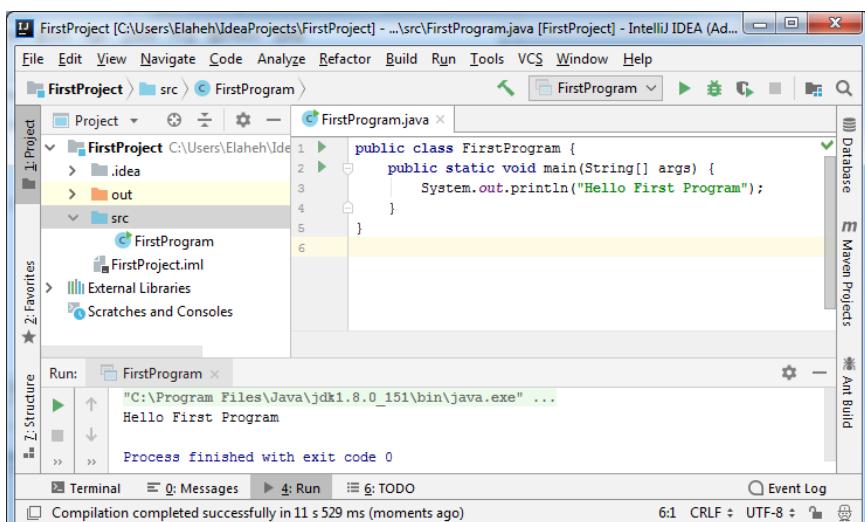
شکل ۳-۲۳. پیاده سازی کلاس FirstProgram

برای اجرای برنامه کافیست روی کدهای برنامه کلیک راست کنید و گزینه Run را به صورت زیر انتخاب کنید.



شکل ۲۴-۳. اجرای کلاس در IntelliJ IDEA

در این صورت برنامه شما در صورتیکه خطای کامپایل نداشته باشد اجرا می شود و نتیجه ای به صورت زیر در کنسول برنامه مشاهده خواهد کرد.



شکل ۲۵-۳. نتیجه اجرای کلاس در کنسول

از اینجا به بعد، شما باید بتوانید کدهای خود را در این محیط پیاده سازی کنید. در انتها این فصل توجه شما را به برخی میانبرهای IntelliJ جلب می کنم.

آشنایی با محیط برنامه نویسی IntelliJ IDEA

توضیح	کلید میانبر
تکمیل کد	Ctrl +Space
تولید جمله System.out.println()	sout + TAB
تولید متدهای public static void main(String[] args)	psvm + TAB
اجرا کلاس	Ctrl + Shift + F10
منویی باز می شود که هر گزینه آن می تواند کدهایی را تولید کند (سازنده کلاس، متدهای getter و setter,...)	Alt + Insert
لیست متدها و فیلدات کلاس و جابجا شدن روی آنها	Ctrl + F12
محل های استفاده شده از یک فیلد یا متدهای در برنامه را نشان می دهد	Alt + F7
یک خط از برنامه را حذف می کند	Ctrl + Y
یک خط از برنامه را Cut می کند	Ctrl + X
یک خط از برنامه را تکرار می کند	Ctrl + D

جدول ۲-۳. میانبرهای IntelliJ IDEA



مفاهیم پایه در برنامه نویسی جاوا

قبل از اینکه وارد مباحث پیشرفته برنامه نویسی شویم لازم است با قواعد و دستور زبان جاوا آشنا شوید در دو فصل پیش با شکل کلی برنامه های جاوا و نحوه اجرای آنها در محیط IntelliJ آشنا شدید. در این فصل با انواع داده ها، آرایه ها، ساختارهای کنترلی و حلقه ها آشنا خواهید شدید. این مفاهیم در دیگر زبانهای برنامه نویسی هم وجود دارد بنابراین اگر با یک زبان دیگر آشنا باشید نیازی به صرف وقت زیادی برای مطالب این فصل نیست و می توانید به صورت سریع و گذرآین فصل را مطالعه کنید.

انواع داده ها

داده ها پراهمیت ترین بخش هر نرم افزار هستند. هر برنامه داده هایی را دریافت می کند آنها را پردازش یا دستکاری می کند و ممکن است آنها را در یک حافظه دائمی ذخیره کند. یک برنامه حسابداری را تصور کنید، این برنامه داده های عددی را از کاربر (حسابدار) دریافت می کند روی آنها پردازش و محاسباتی انجام می دهد و نتایجی را به کاربر یا کاربران خود نشان می دهد.

داده ها ممکن است عدد، رشته، تاریخ و زمان، یا هر قالب دیگری داشته باشند اما هر قالبی که داشته باشند با ترکیبی از ۱ و ۰ ها نگهداری می شود به هر یک از این ۰ و ۱ ها، یک بیت گفته می شود. به هر ۸ بیت یک بایت گفته می شود چون هر بیت دو حالت ۰ یا ۱ دارد بنابراین با ۸ بیت می توان^۸ حالت مختلف داشت که

برابر ۲۵۶ است بنابراین با هر بایت می‌توان از اعداد ۱۲۷ تا ۱۲۸ را نشان داد. در جاوا از داده byte برای تعریف یک عدد ۸ بیتی استفاده می‌شود.

یکی از داده‌های پایه در زبان جاواست. داده‌های پایه، داده‌هایی هستند که داده‌های پیچیده از روی آنها ساخته می‌شوند به داده‌های پایه، داده‌های نیز گفته می‌شود. علاوه بر byte داده‌های پایه دیگری نیز در زبان جاوا وجود دارند که عبارتند از: داده‌های کاراکتری، داده‌های عددی صحیح، داده‌های عددی اعشاری، داده بولی (که مقدار آن «درست» یا «نادرست» است). جدول زیر لیست این داده‌ها را به همراه تعداد بیت و محدودیت قابل نمایش آنها نشان می‌دهد.

نوع داده	نام داده	اندازه به بیت	محدوده قابل نمایش
اعداد صحیح	long	64	9,223,372,036,854,775,807 تا -9,223,372,036,854,775,808
	int	32	-2,147,483,648 تا 2,147,483,647
	short	16	-32768 تا 32767
	byte	8	-128 تا 127
اعداد اعشاری	double	64	1.79769313486231570E+308 تا -1.79769313486231570E+308
	float	32	-3.4029237E+38 تا 3.40292347E+38
کاراکتر	char	16	'\u0000' تا '\uFFFF'
بولی	boolean	8	false تا true

جدول ۱-۴. انواع داده‌های پایه

همانطور که در جدول فوق ملاحظه می‌کنید داده‌های long، int، short و byte معرف اعداد صحیح هستند، double و float معرف اعداد اعشاری، char داده کاراکتری و boolean داده بولی هستند. توجه داشته باشید که اگرچه int، long و short همگی برای کار با اعداد صحیح استفاده می‌شوند اما محدوده اعدادی که توسط هریک از این داده‌ها پشتیبانی می‌شود متفاوت است این موضوع در مورد double و float نیز صحیح است.

هر داده‌ای که در لیست فوق وجود نداشته باشد با ترکیب دو یا چند داده فوق ساخته می‌شود. به عنوان نمونه در یک برنامه حقوق و دستمزد اغلب داده‌ها عددی هستند اما داده‌های محدودی نیز وجود دارند که ممکن است عددی نباشند، مثلاً سابقه کارمندان، آدرس یا اطلاعات شناسایی آنها داده‌هایی مرکب هستند. در فصل شی گرایی با چگونگی تعریف داده‌های مرکب از روی داده‌های پایه آشنا خواهد شد.

توجه: String که معرف یک رشته است نمونه‌ای از یک داده مرکب است که از روی داده‌های پایه

ساخته شده است. یک String از یک یا ترکیب چند کاراکتر ساخته می شود. در جاوا برای نمایش یک کاراکتر از یک کوتیشن و برای نمایش یک رشته از دو کوتیشن استفاده می شود. مثلا 'A' یک کارکتر و "A" یک رشته است.

متغیرها

هر گاه بخواهید یک داده را در چند جای مختلف از برنامه استفاده کنید، مقدار آنرا در جاهای مختلفی از برنامه تغییر دهید یا در محاسبات از آن استفاده کنید باید یک متغیر تعریف کنید. متغیر همانطور که از نام آن مشخص است، داده ای است که مقدار آن می تواند تغییر کند. یک متغیر در حقیقت شبیه یک ظرف است، که در آن مقداری قرار داده می شود سپس در قسمتهای مختلف برنامه می توان مقدار درون ظرف را تغییر داد یا دستکاری کرد.

هر متغیر دارای یک نام است تا با استفاده از آن در قسمتهای مختلف برنامه بتوان به آن دسترسی پیدا کرد یا مقدار آنرا تغییر داد. برای تعریف کردن یک متغیر، علاوه بر نام، باید نوع داده ای که آن متغیر نگهداری می کند را نیز مشخص نمود.

؛ نام متغیر نوع متغیر

به عنوان مثال

```
int i;  
float f;  
double d;  
char c;
```

هر کدام از نمونه های فوق مثالهایی از تعریف متغیر از یک نوع داده اولیه است اولی متغیری به نام `a` از جنس int، موارد بعدی به ترتیب متغیرهایی از جنس `float`، `double` و `char` و با نامهای `f` و `d` و `c` تعریف کرده اند.

تعریف متغیرهای هم نوع نیز می تواند به صورت همزمان باشد مثلا:

```
int i, j;
```

دو متغیر `j` و `i` از نوع int تعریف می کند.

تعریف متغیر می تواند با مقداردهی اولیه همراه باشد

؛ مقدار اولیه = نام متغیر نوع متغیر

مثلاً:

```
int i = 5;  
double d=0.562;  
char c='A';  
int a = 5;
```

توجه: اگر متغیری از جنس `float` تعریف می کنید برای مقداردهی باید بعد از مقدار از حرف `f` استفاده کنید

```
float f= 0.5f;
```

از آنجاییکه `float` و `double` هر دو اعداد اعشاری هستند هر مقدار اعشاری که به یک متغیر انتساب می دهید به صورت پیش فرض یک مقدار `double` است مگر اینکه در انتهای مقدار آن به صورت فوق از حرف `f` استفاده کنید.

مقدار دهی متغیرها نیز همانند تعریف آنها می تواند همزمان باشد مثلا

```
int i = 5, j =6;
```

دو متغیر `i` و `j` را از نوع `int` با مقادیر اولیه ۵ و ۶ تعریف می کند. اما مقداردهی را می توانید در جملات جداگانه ای نیز انجام دهید، درینصورت اگر متغیر قبل مقداری داشته باشد، مقدار آن تغییر می کند و اگر قبل مقداری نداشته باشد مقدار داده می شود کدهای زیر مقداردهی متغیرها بعد از تعریف آنها را نشان می دهد.

```
int i = 5;
float f= 0.5f;
double d=0.562;
char c = 'A';

i = 15;
f= 0.72f;
d=0.744;
c = 'B';
```

وقتی متغیر در برنامه تعریف شد، در خطوط بعدی برنامه می توان مقدار جدیدی به آن متغیر انتساب داد یا مقدار آن متغیر را تغییر داد. به تکه کدهای زیر توجه کنید:

```
String s;
s = " This is the first program in java";
s = s + ", enjoy!";
System.out.println (s);
```

در این مثال، یک متغیر رشته‌ای به نام `s` تعریف شده است. `String` مشخص کننده نوع متغیر (رشته ای) و `s` مشخص کننده نام متغیر است سپس به متغیر `s` مقدار `"This is the first program in java"` داده شده است. در خط بعد عبارت `, enjoy!` به انتهای مقدار متغیر `s` چسبانده شده است. و در پایان نیز مقدار این متغیر در خروجی استاندارد چاپ شده است.

محدوده قابل مشاهده یک متغیر

هر متغیری که در برنامه تعریف می شود فقط در قسمتهای خاصی از برنامه، قابل دستیابی و استفاده است. به محدوده ای که یک متغیر، قابل دستیابی است و می توان به مقدار آن متغیر دسترسی پیدا کرد یا مقدار آنرا تغییر داد، محدوده قابل مشاهده یا scope آن متغیر گفته می شود. بسته به اینکه یک متغیر در چه قسمی از یک کلاس یا متند تعریف شده باشد، محدوده قابل مشاهده آن متغیر نیز متفاوت خواهد بود. قبل از اینکه محدوده قابل مشاهده یک متغیر را بیاموزید لازم است با چند تعریف آشنا شوید.

در برنامه های جاوا از جملات { و } برای مرزبندی کدهای برنامه استفاده می شود مثلا هر کلاس و هر متند با { شروع و با } خاتمه می یابد به محدوده داخل یک متند که بین { و } قرار دارد «بلوک آن متند» و به محدوده داخل یک کلاس که بعد از { و قبل از { قرار دارد «بلوک آن کلاس» گفته می شود علاوه بر بلوک متند و بلوک کلاس، به هر محدوده دیگری که داخل { و { قرار داشته باشد، اعم از جملات while، if، ...، یک «بلوک» گفته می شود.

هر متغیر بسته به اینکه در چه بلوکی از برنامه تعریف شده باشد فقط در بلوک یا بلوکهای خاصی از برنامه قابل دسترسی است. در قسمت بعدی خواهیم دید که اگر یک متغیر در محدوده یک «بلوک»، یک «بلوک متند» یا یک «بلوک کلاس» تعریف شود در چه بلوک هایی قابل دسترسی است.

به صورت یک قانون کلی اولا هر متغیر بعد از جایی که تعریف می شود یعنی در خطوط بعدی برنامه قابل دسترسی خواهد بود و ثانیا هر متغیر محدود به نزدیکترین بلوکی است که در آن تعریف شده است. مثال زیر را در نظر بگیرید.

```
while(condition) {  
    int i=5;  
}
```

در این مثال متغیر i خارج از حلقه while در دسترس نخواهد بود زیرا نزدیکترین بلوکی که آن را محصور کرده است، بلوک حلقه while است. اگر متغیر i به شکل زیر خارج از حلقه while و داخل متند تعریف شود

```
public void process() {  
    int i=5;  
    while(condition) {  
        ...  
    }  
}  
public void perform() {  
    ...  
}
```

در اینصورت نزدیکترین بلوکی که آنرا محدود کرده است، متده `process` خواهد بود این بدین معناست که متغیر در متده `process` یعنی هم داخل حلقة `while` و هم خارج از حلقة `while` قابل دسترس خواهد بود. ولی این متغیر داخل متده `perform` در دسترس نخواهد بود. برای اینکه متغیر `a` در هر دو متده قابل دسترس باشد می بایست این متغیر را به صورت زیر تعریف کنید

```
class Test{
    int i=5;
    public void process() {
        while(condition) {
            ...
        }
    }
    public void perform() {
        ...
    }
}
```

اینک متغیر `a` محدود به نزدیکترین بلوک یعنی بلوک کلاس `Test` شده است و در سراسر کلاس `Test` که شامل هر دو متده می شود قابل دسترسی است. همانطور که در فصلهای آینده خواهید دید به متغیری که محدوده قابل مشاهده آن کلاس باشد `Property` گفته می شود.

محدوده یک بلوک

هر متغیر در هر بلوکی که تعریف می شود، فقط در همان بلوک قابل دسترسی و استفاده است به عبارت دیگر محدوده قابل مشاهده یک متغیر، بلوکی است که آن متغیر در آن تعریف شده است. اگر از یک متغیر، خارج از بلوکی که آن متغیر در آن تعریف شده است استفاده شود، یک خطای کامپایل ایجاد خواهد شد و برنامه کامپایل نخواهد شد. به مثال زیر توجه کنید.

```
1 //Block scope demonstrating
2 //this program shows different blocks in a typical class
3 public class BlockScopeDemonstrate {
4
5     public static void main(String[] args) {
6         int a = 10 ;
7         if(a==5){   //block scope starts
8             int c = 40;
9             System.out.println("a="+a);
```

```
10         System.out.println("c="+c);
11     } //block scope ends
12     System.out.println("a="+a);
13     System.out.println("c="+c);
14 }
15 }
```

کد ۱

همانطور که ملاحظه می کنید متغیر `c` داخل بلوک `if` تعریف شده است بنابراین نمی توان از این متغیر خارج از محدوده بلوک `if` استفاده کرد. در صورتی که کلاس فوق را کامپایل کنید پیغام خطای کامپایل زیرا را مشاهده خواهید کرد.

```
BlockscopeDemonstrate.java:13 cannot resolve symbol
symbol : variable c
location: class BlockscopeDemonstrate
    System.out.println("c="+c);
                                         ^
1 error
```

نکته جالب در مورد مثال فوق این است که متغیر `a` که بیرون بلوک `if` تعریف شده داخل بلوک `if` قابل دسترسی است.

توجه: در داخل هر بلوک می توان به متغیرهایی که خارج و قبل از آن بلوک تعریف شده اند دسترسی پیدا کرد.

توجه داشته باشید که پس از اجرای یک بلوک از برنامه، متغیرهای محلی که در آن بلوک تعریف شده اند غیر قابل دسترس می شوند، و توسط JVM به صورت اتوماتیک از حافظه پاک می شوند.

محدوده یک مت

محدوده یک مت شامل همه کدهایی می شود که بالاصله پس از تعریف مت داچل `{` و `}` قرار دارند به این محدوده «بدنه مت» نیز گفته می شود هر متغیری که در این محدوده تعریف شود بعد از نقطه ای که در آن تعریف شده تا انتهای بدنه مت قابل دسترسی است اما بیرون متدى که در آن تعریف شده-مثلًا در متدهای دیگر- قابل دسترسی نیست. به مثال زیر دقت کنید.

```

1 //Method scope demonstrating
2 //this program shows two methods with individual scopes
3 public class MethodScopeDemonstrate {
4     public static void main(String[] args) {
5         int i = 5;
6         System.out.println("i="+i);
7     }
8
9     public static void print() {
10        System.out.println("i="+i); //won't compile
11    }
12 }
```

کد ۳

این کلاس دارای یک اشغال گرامری است، زیرا در حالیکه متغیر **i** داخل متده است **main()** تعریف شده و محدوده مشاهده آن فقط متده است **main()** است داخل متده است **print()** نیز استفاده شده است. کامپایل این کلاس، پیغام خطای کامپایل زیر را تولید خواهد کرد.

```

MethodScopeDemonstrate.java:11: cannot resolve symbol
symbol   : variable i
location: class MethodScopeDemonstrate
        System.out.println("i="+i);
                                         ^
1 error
```

توجه: به محض اتمام اجرای یک متده تمام متغیرهایی که در بدنۀ آن متده تعریف شده‌اند غیرقابل دسترس شده و توسط JVM به صورت اتوماتیک از حافظه پاک می‌شوند.

محدوده یک کلاس

تعریف هر کلاس با **{** شروع و با **}** خاتمه می‌یابد که محدوده کلاس را مشخص می‌کند به این محدوده «بدنۀ کلاس» نیز گفته می‌شود. در بدنۀ کلاس منظور خارج بدنۀ متدها- می‌توانید هر متغیری تعریف کنید این متغیرها در سراسر محدوده کلاس و در تمام متدها قابل دسترسی‌اند. به کد ۵-۳ توجه کنید.

```

1 //this program shows using variables in class scope
2 public class ClassScopeDemonstrate {
3     int i = 5;
```

```
4  
5     public void print(){  
6         System.out.println("i="+i);  
7     }  
8  
9     public int add(int j){  
10        return i+j;  
11    }  
12 }
```

کد ۴-۳

در این کلاس متغیر `i` داخل بدنۀ کلاس و بیرون تمام متدها تعریف شده است بنابراین تمام متدها به آن دسترسی دارند به این متغیرها، خصوصیتها یا `property` های کلاس نیز گفته می‌شود که بعداً در مبحث شی گرایی مفصل‌ا درباره آن صحبت خواهیم کرد.

داخل یک «محدوده» نمی‌توانید دو یا چند متغیر همنام داشته باشید ولی اگر محدوده متغیرها با یکدیگر متفاوت باشند به گونه‌ای که داخل هیچ‌کدام از محدوده‌ها نتوان به متغیر تعریف شده در محدوده دیگر دسترسی داشت، می‌توان در دو محدوده متغیرهای همنام تعریف نمود برای درک این مطلب به مثال زیر توجه کنید.

```
1 //defining different variable in independent scopes  
2 public class IndependentScopeDemo {  
3  
4     public void printValues(int i){  
5  
6         if(i == 2){  
7             int j=12;  
8             System.out.println(j);  
9         }else{  
10            int j=24;  
11            System.out.println(j);  
12        }  
13    }  
14 }
```

کد ۴-۴

در این کلاس در بلوک `if` متغیری به نام `j` تعریف شده است در بلوک `else` نیز متغیر دیگری با نام `j` تعریف شده است از آنجاییکه بلوکهای `if` و `else` مستقل از یکدیگر هستند این برنامه بدون هیچ خطایی کامپایل خواهد شد. اینک به کلاس زیر دقت کنید.

```

1 //defining different variable in dependent scopes
2 public class IndependentScopeDemo1 {
3
4     public void printValues(int i){
5
6         int j = 23 ;
7         if(i == 2){
8             int j=12; //error
9             System.out.println(j);
10        }else{
11            int j=24; //error
12            System.out.println(j);
13        }
14    }
15 }
```

۴-۵ کد

در این کلاس، بیرون بلوکهای `if` و `else` متغیری به نام `j` تعریف شده است چون درون بلوک `if` یا `else` می توان به متغیری که بیرون بلوک تعریف شده است دسترسی پیدا کرد بنابراین نمی توان متغیری با نام متغیری که بیرون بلوک تعریف شده است داخل بلوک نیز تعریف نمود بنابراین این کلاس دارای دو خطایی کامپایل خواهد بود.

توجه: متغیرهایی که در محدوده کلاس تعریف می شوند را متغیرهای «عمومی» و متغیرهایی که در محدوده متند یا بلوکهای داخل متند تعریف می شوند را متغیرهای «محلي» می نامند.

متغیرهای عمومی و محلی می توانند روی هم افتادگی داشته باشند این بدین معنی است که یک کلاس می تواند حاوی یک متغیر محلی و یک متغیر عمومی با نام های یکسان از یک جنس یا از جنسهای مختلف باشد برای درک این مطلب به مثال زیر دقت کنید.

```
1 //defining different variable class
2 //scope and block(or method) scope
3 public class IndependentScopeDemo2 {
4
5     int j = 26;
6
7     public void printValues(int i) {
8
9         if(i == 2) {
10            int j=12;
11            System.out.println(j);
12        }else{
13            int j=24;
14            System.out.println(j);
15        }
16    }
17 }
```

کد ۴

در این مثال، در حالیکه در بدنه کلاس متغیری به نام `j` تعریف شده، در بلوکهای `if` و `else` نیز متغیرهای دیگری با همان نام تعریف شده است. این برنامه بدون هیچگونه خطایی کامپایل خواهد شد.

توجه: در محدوده هایی از کلاس که دسترسی همزمان به یک متغیر محلی و یک متغیر عمومی همنام امکان پذیر است با استفاده از نام متغیر می توانید به متغیر محلی و با بکارگیری پیشوند `this` قبل از نام متغیر می توانید به متغیر عمومی دسترسی پیدا کنید.

کد زیر استفاده از کلمه `this` برای دسترسی به متغیر عمومی در مثال قبل را نشان می دهد.

```
1 //defining different variable class
2 //scope and block(or method) scope
3 public class IndependentScopeDemo3 {
4
5     int j = 26;
6
7     public void printValues(int i) {
```

```

9         if(i == 2) {
10            int j=12;
11            System.out.println("Value of Local j is: "+j);
12        }else{
13            int j=24;
14            System.out.println("Value of Local j is: "+j);
15        }
16
17        System.out.println("Value of Global j is: "+this.j);
18    }
19 }
```

۴-۷ کد

دقت کنید که چگونه با استفاده از کلمه `this` و یک نقطه به متغیر عمومی `j` دسترسی شده است. نکته مهم دیگری که لازم است بدانید این است که پارامترهای یک متاد نیز به عنوان متغیر محلی در آن متاد محسوب می شوند برای درک این مطلب به مثال زیر توجه کنید.

```

1 //defining different variable class
2 //scope and block(or method) scope
3 public class IndependentScopeDemo4 {
4
5     int i = 26;
6
7     public void aMethod(int i){
8         System.out.println("Value of Local i is: "+i);
9         System.out.println("Value of Global i is: "+this.i);
10    }
11 }
```

۴-۸ کد

همانطور که در این مثال مشاهده می کنید، متاد `aMethod` دارای یک پارامتر با نام `i` و از جنس `int` است در این کلاس یک متغیر عمومی با نام `i` نیز تعریف شده است بنابراین در متاد `aMethod` برای دسترسی به مقدار پارامتر فقط کافیست از نام `i` استفاده کنید اما برای دسترسی به مقدار متغیر عمومی `i` باید از عبارت `this.i` استفاده نمایید. در مورد مفهوم و کاربرد کلمه `this` در فصل شی گرایی بیشتر خواهید آموخت.

نامگذاری متغیرها

نام یک متغیر از قواعد نامگذاری جاوا پیروی می کند. بر این اساس نام یک متغیر:

- ۱- می تواند ترکیبی از اعداد و کاراکترهای لاتین باشد اما نمی تواند با عدد شروع شود.
 - ۲- می تواند از کاراکتر زیرخط (_)، که به آن underscore گفته می شود، و علامت دلار (\$) تشکیل شود اما نمی تواند حاوی هیچ کاراکتر خاص دیگری از قبیل *، /، ، %، !، ؟ و ... باشد. (البته بهتر است از این علامتها استفاده نشود)
 - ۳- نام یک متغیر نمی تواند حاوی فاصله (space) باشد.
- جدول زیر مثالهایی از نامهای مجاز و غیرمجاز برای متغیرها را نشان می دهد.

نام مجاز	نام غیرمجاز
x1	1x
\$first_number	first number
count\$number	x.no
firstIdentifier	4456

جدول ۴-۲. نمونه هایی از نامهای مجاز و غیرمجاز در جاوا

عملگرها

در جاوا همانند هر زبان برنامه نویسی دیگر امکان انجام انواع محاسبات و مقایسه ها بین داده ها امکان پذیر است. اعمالی که بین دو یا چند داده انجام می شوند از طریق علایمی نشان داده می شوند که به آنها «عملگر» گفته می شود به هریک از داده نیز عملوند گفته می شود. عملگرها در جاوا به پنج دسته زیر قابل تقسیم بندی هستند.

- ۱ عملگر انتساب
- ۲ عملگرهای محاسباتی
- ۳ عملگرهای بیتی
- ۴ عملگرهای مقایسه ای
- ۵ عملگرهای منطقی

عملگر انتساب

به عملگر = عملگر انتساب گفته می شود با استفاده از این عملگر می توانید به یک متغیر مقدار جدیدی انتساب دهید. به مثال زیر توجه کنید:

```
int a = 2000;
```

متغیرها را می توانید به صورت همزمان نیز مقداردهی کنید مثلا

```
int p,q,r,s ;
p=q=r=s=a ;
```

در مقداردهی متغیرها، مقداردهی آنها از راست به چپ انجام می شود در مثال فوق ابتدا مقدار متغیر a به s ، سپس مقدار s به r ، مقدار r به q و... انتساب داده می شود.

عملگرهای محاسباتی

از این عملگرها برای انجام محاسبات ریاضی (از قبیل جمع، تفریق، ضرب و تقسیم،...) بین دو متغیر عددی یا کارکتری استفاده می شود جدول ۴-۲ لیست این عملگرها را نشان می دهد.

ارزیابی	مثال	معنی	عملگر
	$c = a + b$	جمع	+
	$c = a - b$	تفریق	-
	$c = a * b$	ضرب	*
	$c = a / b$	تقسیم	/
	$c = a \% b$	باقیمانده تقسیم	%
$c = c + 1$	$c++$	یک واحد اضافه	++
$c = c - 1$	$c--$	یک واحد کم	--
$c = c + a$	$c += a$	جمع و انتساب	$+=$
$c = c - a$	$c -= a$	تفریق و انتساب	$-=$
$c = c * a$	$c *= a$	ضرب و انتساب	$*=$

مفاهیم پایه در برنامه نویسی جاوا

c = c / a	c/a	تقسیم و انتساب	/=
c = c % a	c%a	تقسیم و انتساب	%=
	c=-c	منفی	-

جدول ۳-۴. عملگرهای محاسباتی

مثال زیر نحوه بکارگیری عملگرهای محاسباتی و نتیجه هر عملگر را نشان می دهد.

```
1 public class ArithmeticOperationDemo {  
2  
3     public static void main(String[] args) {  
4         int p = 5, q = 12, r = 20, s;  
5  
6         System.out.println("p=" + p);  
7         System.out.println("q=" + q);  
8         System.out.println("r=" + r);  
9  
10        //addition  
11        s = p + q;  
12        System.out.println("p + q = " + s);  
13  
14        //subtraction  
15        s = p - q ;  
16        System.out.println("p - q = "+s);  
17  
18        //modules  
19        s = p % q ;  
20        System.out.println("p % q = "+s);  
21  
22        //multiplication  
23        s = p * q ;  
24        System.out.println("p * q = "+s);  
25    }
```

```

26         //division
27         s = p / q ;
28         System.out.println("p / q = "+s);
29
30         //increment
31         p++ ;
32         System.out.println("p++ = "+p);
33
34         //decrement
35         p--;
36         System.out.println("p-- = "+p);
37
38         //addition and assignment
39         r+=q;
40         System.out.println("r after r+=q is "+r);
41
42         //subtraction and assignment
43         r-=q ;
44         System.out.println("r after r-=q is "+r);
45
46         //multiplication and assignment
47         r*=q ;
48         System.out.println("r after s*=q is "+r);
49
50         //division and assignment
51         s/=q ;
52         System.out.println("r after s/=q is "+r);
53
54         //modules
55         s%=q ;
56         System.out.println("r after r%=q is "+r);
57
58         //negation and assignment
59         s = -s ;
60         System.out.println("r after r=-r is "+r);
61
62     }
63 }
```

مفاهیم پایه در برنامه نویسی جاوا

اجرای برنامه فوق، نتیجه زیر را خواهد داشت.

```
p=5  
q=12  
r=20  
p + q = 17  
p - q = -7  
p % q = 5  
p * q = 60  
p / q = 0  
p++ = 6  
p-- = 5  
r after r+=q is 32  
r after r-=q is 20  
r after s*=q is 240  
r after s/=q is 240  
r after r%=q is 240  
r after r=-r is 240
```

عملگرهای بیتی

از این عملگرها برای انجام محاسبات بیتی روی داده های نوع int، short، long و byte استفاده می شود. جدول زیر این عملگرها را نشان می دهد.

معنی	نحوه استفاده	معنی	عملگر
تمام بیتهای 1 تبدیل به 0 و تمام بیتهای 0 تبدیل به 1 می شوند	<code>int a = 60; /* 60 = 0011 1100 */ int b = 13; /* 13 = 0000 1101 */ int c = 0; c = ~ a; /* -61 = 1100 0011 */</code>	NOT	~
اگر دو بیت 1 باشد حاصل آنها 1 و در غیراینصورت حاصل آنها 0 خواهد شد AND	<code>c = a & b; /* 12 = 0000 1100 */</code>	AND	&
اگر یکی از بیتها 1 باشد حاصل OR آنها 1 و در غیراینصورت حاصل آنها 0 خواهد شد.	<code>a b = 61 /* 61 = 0011 1101 */</code>	OR	

اگر دو بیت مخالف باشند، حاصل آنها ۱ و در غیر اینصورت ۰ خواهد بود.	<code>c = a ^ b; /* 49 = 0011 0001 */</code>	Exclusive OR	\wedge
با اضافه کردن یک ۰ در سمت چپ، یک بیت از سمت راست برداشته می شود	<code>c = a >> 2; /* 215 = 1111 */</code>	Shift Right	$>>$
با اضافه کردن یک ۱ در سمت راست، یک بیت از سمت چپ برداشته می شود	<code>c = a >> 2; /* 215 = 0000 1111 */</code>	Shift Left	$<<$

جدول ۴-۴. عملگرهای بیتی

با استفاده از این عملگرها می توانید روی بیتهای هر عدد، عملیات بیتی انجام دهید جدول زیر نتیجه عملگرهای مختلف بیتی را روی دو بیت A و B نشان می دهد:

A	B	A B	A&B	A^B	$\sim A$
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

جدول ۵-۵. حاصل عملگرهای بیتی روی دو بیت A و B

برنامه زیر اعمال عملگرهای بیتی را روی بیتهای مختلف نشان می دهد.

```
1 // Demonstrate the bitwise logical operators.
```

مفاهیم پایه در برنامه نویسی جاوا

```
2  public class BitLogic {  
3  
4      public static void main(String args[]) {  
5  
6          String binary[] = {"0000", "0001", "0010",  
7              "0011", "0100", "0101",  
8              "0110", "0111", "1000",  
9              "1001", "1010", "1011",  
10             "1100", "1101", "1110",  
11             "1111"};  
12  
13         int a = 3; // 0 + 2 + 1 or 0011 in binary  
14         int b = 6; // 4 + 2 + 0 or 0110 in binary  
15         int c = a | b;  
16         int d = a & b;  
17         int e = a ^ b;  
18         int f = (~a & b) | (a & ~b);  
19         int g = ~a & 0x0f;  
20  
21         System.out.println("a = " + binary[a]);  
22         System.out.println("b = " + binary[b]);  
23         System.out.println("a|b = " + binary[c]);  
24         System.out.println("a&b = " + binary[d]);  
25         System.out.println("a^b = " + binary[e]);  
26         System.out.println("~a&b|a&~b = " +  
27             binary[f]);  
28         System.out.println("~a = " + binary[g]);  
29     }  
30 }
```

۴-۱-۰ کد

اجرای برنامه فوق نتیجه عملیات بیتی روی اعداد را نشان می دهد:

```
a = 0011  
b = 0110  
a|b = 0111  
a&b = 0010  
a^b = 0101  
~a&b|a&~b = 0101  
~a = 1100
```

عملگرهای مقایسه ای

عملگرهای مقایسه ای برای ارزیابی نسبت دو عملوند بکار می‌روند نتیجه ارزیابی یک مقدار boolean است که مقدار آن true یا false است این ارزیابی معمولاً در ساختارهای کنترلی همانند if استفاده می‌شوند جدول زیر عملگرهای مقایسه ای را نشان می‌دهد.

عملگر	معنی	نحوه استفاده	معنی
==	مساوی	int a = 5; int b = 3; if(a==b){ System.out.println("Equals"); }	تساوی دو متغیر را بررسی می کند و حاصل ارزیابی یک boolean مقدار است.
!=	مخالف (بزرگتر یا کوچکتر)	if(a==b){ System.out.println("Equals"); }	عدم تساوی دو متغیر را بررسی می کند و حاصل ارزیابی یک boolean مقدار است.
>	بزرگتر از	if(a>b){ System.out.println("Greater"); }	بزرگتر بودن یک متغیر را بررسی می کند و حاصل ارزیابی یک boolean مقدار است.
<	کوچکتر از	if(a<b){ System.out.println("Less"); }	کوچکتر بودن یک متغیر را بررسی می کند و حاصل ارزیابی یک boolean مقدار است.
>=	بزرگتر و مساوی	if(a>=b){ System.out.println("Greater or equals"); }	بزرگتر بودن یا مساوی بودن یک متغیر با یک مقدار را بررسی می کند و حاصل ارزیابی یک boolean مقدار است.

مفاهیم پایه در برنامه نویسی جاوا

کوچکتر بودن یا مساوی بودن یک متغیر با یک مقدار را بررسی می کند و حاصل ارزیابی یک boolean مقدار است.	if(a<=b) { System.out.println("Less or equals"); }	کوچکتر و مساوی	<=
---	--	----------------	----

جدول ۴-۶. عملگرهای بیتی

عملگرهای منطقی

عملگرهای منطقی با عملوندهای boolean کار می کنند این عملگرها بین دو مقدار boolean بکار می روند و نتیجه ارزیابی آنها نیز یک مقدار boolean است. جدول ۴-۷ عملگرهای منطقی را نشان می دهد.

معنی	نحوه استفاده	معنی	عملگر
حاصل این عبارت وقتی است که حاصل ارزیابی A و B به طور همزمان true باشد.	A&B	AND	&
حاصل این عبارت وقتی است که حاصل ارزیابی A و B به طور همزمان true باشد.	A&&B	Short-circuit AND	&&
حاصل این عبارت وقتی است که حاصل ارزیابی A، B یا A باشد.	A B	OR	
حاصل این عبارت وقتی است که حاصل ارزیابی A، B یا A باشد.	A B	Short-circuit OR	
حاصل این عبارت وقتی است که حاصل true	A^B	XOR	^

ارزیابی A با مقدار ارزیابی B متفاوت باشد.			
حاصل این عبارت وقتی است که حاصل true false ارزیابی A باشد.	A !	NOT	!
درصورتیکه حاصل true ارزیابی A برابر باشد Value1 و در Value2 غیراینصورت به متغیر D انتساب داده می شود.	D=A?Value1:Value2	Ternary if-else	? :

جدول ۴-۷. عملگرهای منطقی

وقتی از & یا | استفاده می کنیم ابتدا دو طرف این عملگرها ارزیابی می شوند سپس از روی نتیجه ارزیابی طرفین این عملگرها، نتیجه نهایی استنتاج می شود مثلا در هنگام ارزیابی عبارت:

```
if ((a==5) | (b != -1)) {  
...  
}
```

ابتدا هر دو عبارت `a==5` و `b != -1` ارزیابی می شوند سپس درصورتیکه حداقل مقدار یکی از آنها باشد، حاصل ارزیابی true و در غیراینصورت false استنتاج می شود.
اما در OR short-circuit AND و short-circuit short-circuit وضعیت متفاوت است، در این عملگرها ابتدا عملوند سمت چپ ارزیابی می شود و سپس در صورت نیاز عملوند سمت راست ارزیابی می شود مثلا در هنگام ارزیابی عبارت:

```
if ((a==5) || (b != -1)) {  
...  
}
```

ابتدا عبارت `a==5` ارزیابی می شود در صورتی که حاصل ارزیابی این عبارت مقدار true باشد، نیازی به ارزیابی `b != -1` نیست زیرا برای محاسبه OR، درست بودن یکی از گزاره ها کافیست. جمله `b != -1` تنها زمانی ارزیابی می شود که حاصل ارزیابی `a==5` false باشد.

توجه: توصیه می شود از `&&` به جای `||` و از `||` به جای `|` استفاده شود.

جدول ۴-۸ نتیجه عملگرهای منطقی را روی مقادیر مختلف دو متغیر `A` و `B` که از نوع `boolean` هستند نشان می دهد.

<code>A</code>	<code>B</code>	<code>A B</code>	<code>A&B</code>	<code>A^B</code>	<code>!A</code>
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

جدول ۴-۸. حاصل عملگرهای منطقی روی دو مقدار `A` و `B`

کد زیر کلاسی را نشان می دهد که حاصل عملگرهای منطقی را روی دو متغیر در کنسول برنامه چاپ می کنند.

```

1 // Demonstrate the boolean logical operators.
2 public class BoolLogic {
3
4     public static void main(String args[]) {
5
6         boolean a = true;
7         boolean b = false;
8         boolean c = a | b;
9         boolean d = a & b;
10        boolean e = a ^ b;
11        boolean f = (!a & b) | (a & !b);
12        boolean g = !a;
13

```

```

14     System.out.println("a = " + a);
15     System.out.println("b = " + b);
16     System.out.println("a|b = " + c);
17     System.out.println("a&b = " + d);
18     System.out.println("a^b = " + e);
19     System.out.println("!a&b|a&!b = " + f);
20     System.out.println("!a = " + g);
21 }
22 }
```

کد ۱۱

نتیجه اجرای این برنامه به صورت زیر خواهد بود:

```

a = true
b = false
a|b = true
a&b = false
a^b = true
!a&b|a&!b = true
!a = false
```

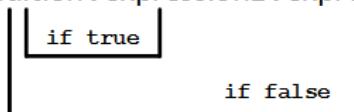
عملگر ?

به جای نوشتن یک ساختار if-else برای انتساب مقدار به یک متغیر می توانید از این عملگر ساده استفاده کنید کلی استفاده از این عملگر به صورت زیر است:

`variable = condition?expression1:expression2`

در صورتیکه حاصل ارزیابی condition مقدار expression1 باشد true و در غیر این صورت به متغیر variable انتساب داده می شود.

`variable = condition?expression1:expression2`



کد ۱۲-۴- بکارگیری این عملگر را نشان می دهد.

```

1 //Using Ternary operator
2 public class TernaryDemo {
3     public static void main(String[] args) {
4 }
```

```
5      //get a random number between 0 to 10
6      double d = Math.random()*10;
7
8      //assign int value of d to b
9      int b = (int) d ;
10     System.out.println(b);
11
12     //calculate remind of random number to 2
13     b = b%2 ;
14
15     String str ;
16     //if remind value is 0 then b is even
17     //otherwise it is odd
18     str=b==0?"even":"odd";
19
20     //print result String to console
21     System.out.println(str);
22 }
23 }
```

کد ۱۲

در این برنامه ابتدا از طریق `Math.random()` یک عدد تصادفی محاسبه شده است از آنجاییکه عدد حاصل شده عددی اعشاری بین ۰ و ۱ است آنرا در ۱۰ ضرب کرده ایم که حاصل آن عددی اعشاری بین ۰ تا ۱۰ خواهد بود سپس این عدد به نزدیک ترین عدد صحیح گرد شده و باقیمانده آن بر ۲ محاسبه شده است. اگر باقیمانده این عدد بر ۲ صفر باشد کلمه `even` و در غیراینصورت کلمه `odd` در کنسول برنامه چاپ خواهد شد.

خروجی برنامه فوق به صورت زیر خواهد بود.

```
3
Odd
```

اولویت بندی عملگرهای ریاضی

برای انجام محاسبات ریاضی می‌توانید از ترکیبی از عملگرهای استفاده کنید مثلاً جمله زیر چندین عدد را با یکدیگر جمع می‌کند و در یک متغیر عددی با نام `A` قرار می‌دهد.

```
int i = 12 + 35 + 17 + 1 ;
```

همچنین جمله

```
int j = 3 * 21 * 12 ;
```

حاصلضرب 3 و 21 و 1 را در متغیر عددی `j` قرار می‌دهد. اما این جملات می‌توانند ترکیبی از اعمال مختلف ریاضی باشند به عنوان مثال:

```
int k = 12*21 + 35/16 + 14 - 8 ;
```

در اینصورت نحوه محاسبه عبارات فوق از روی اولویت عملگرها انجام خواهد شد از آنجاییکه اولویت ضرب و تقسیم بالاتر از اولویت جمع و تفریق است ابتدا حاصل ضرب 12 و 21 و حاصل تقسیم 35 و 16 محاسبه می‌شود سپس حاصل

```
int k = 252 + 2 + 14 - 8;
```

محاسبه می‌شود که نهایتاً حاصل `k` برابر 260 خواهد شد.

در صورتی که در حالت خاصی بخواهیم به نحوی اولویت‌های از پیش تعیین شده را تغییر دهیم باید از عملگر پرانتز استفاده کنیم مثلاً در عبارت

```
int h = (12+5)*6 + (3-1)/2 ;
```

ابتدا حاصلجمع 5 و 12 محاسبه می‌شود و سپس مقدار آن در 6 ضرب می‌شود از طرفی ابتدا حاصل تفریق 3 و 1 محاسبه می‌شود و نتیجه آن بر 2 تقسیم می‌شود و نهایتاً حاصل آنها جمع شده و به `h` انتساب داده می‌شود.

جدول زیر اولویت عملگرهای ریاضی را از بالا به پایین نشان می‌دهد.

عملگر
()
++ -- !
* / %
+ -

جدول ۴-۹. اولویت عملگرهای ریاضی

به صورت معمول می‌توان مقدار یک متغیر را در متغیر دیگری قرار داد این کار با استفاده از عملگر انتساب بین دو متغیر انجام می‌شود، اما اگر بخواهید مقدار یک متغیر را به یک متغیر دیگر که از جنس دیگریست انتساب دهید ممکن است کار شما به سادگی امکانپذیر نباشد. در جاوا فقط داده‌هایی را می‌توانید به یکدیگر انتساب دهید که با یکدیگر سازگار باشند، در غیراینصورت تبدیل داده‌ها امکان پذیر نمی‌باشد. به تبدیل دو داده متفاوت اما سازگار اصطلاحاً Cast کردن یک داده به داده دیگر گفته می‌شود در صورتیکه دو داده ناسازگار باشند برنامه شما به درستی کامپایل خواهد شد اما در زمان اجرا چهار خطای خواهد شد. اما چه داده‌هایی سازگار و چه داده‌هایی ناسازگار هستند؟

در میان داده‌های پایه، داده‌های عددی با یکدیگر سازگار محسوب می‌شوند بنابراین شما می‌توانید اعداد اعشاری float و double را به یکدیگر و اعداد صحیح int، long و short را نیز به یکدیگر تبدیل

کنید حتی می توانید اعداد اعشاری را به اعداد صحیح و بالعکس تبدیل کنید. در میان داده های پایه، `char` و `boolean` با یکدیگر و با هیچکدام از داده های عددی سازگار نمی باشند. بنابراین هرگاه صحبت `Cast` کردن داده های پایه به میان می آید منظور داده های سازگار یا همان داده های عددی هستند. یک مقدار `byte` را می توان به تمام انواع داده های عددی تبدیل کرد همچنین تمام داده های عددی را نیز می توان به `byte` تبدیل نمود.

برای تبدیل کردن دو نوع داده سازگار به یکدیگر دو حالت وجود دارد

الف- اندازه داده مقصد از داده مبدأ بزرگتر است به این نوع تبدیل داده ها به یکدیگر، «تبدیل اتوماتیک» گفته می شود زیرا می توانید به سادگی مقدار متغیرها را به یکدیگر انتساب دهید:

```
int i=27;
long l=138;
l = i;
```

در این مثال اندازه `A` کوچکتر از `A` است زیرا `A` یک عدد `int` (32 بیتی) ولی `l` یک عدد `long` (64 بیتی) است.

ب- داده مقصد از داده مبدأ کوچکتر است در اینصورت باید صریحا در برنامه خود مشخص کنید که می خواهید یک داده نوع بزرگتر را به داده نوع کوچکتر تبدیل کنید:

```
int i=27;
long l=12;
i =(int) l;
```

طبعی است که در چنین موقعی ممکن است قسمتی از داده های پایه از دست برود (قسمت پارازش داده از بین می رود) زیرا به علت اندازه کوچکتر مقصد، ممکن است فقط قسمتی از داده اولیه به داده مقصد منتقل شود (داده مقصد کوچکتر از داده مبدأ است). وقتی داده ای از نوع `long` به یک داده از نوع `int` انتساب داده می شود قسمت با ارزشتر داده از بین خواهد رفت.

توجه: جاوا هر عدد صحیح را به صورت پیش فرض `int` تفسیر می کند. اما با افزودن کاراکتر '`'L`' به انتهای یک مقدار عددی صحیح، آنرا از جنس `long` قلمداد کرد.

مثال زیر تبدیل داده ها را نشان می دهد.

```
1 // Demonstrate casts.
2 public class Conversion {
3
```

```

4     public static void main(String args[]) {
5         byte b;
6         int i = 257;
7         double d = 323.142;
8
9         System.out.println(
10            "\nConversion of int to byte.");
11         b = (byte) i;
12         System.out.println("i and b " + i + " " + b);
13
14         System.out.println(
15            "\nConversion of double to int.");
16         i = (int) d;
17         System.out.println("d and i " + d + " " + i);
18
19         System.out.println(
20            "\nConversion of double to byte.");
21         b = (byte) d;
22         System.out.println("d and b " + d + " " + b);
23     }
24 }
```

کد ۱۳

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود:

Conversion of int to byte.
i and b 257 1

Conversion of double to int.
d and i 323.142 323

Conversion of double to byte.
d and b 323.142 67

توجه: جاوا هر عدد اعشاری را به صورت پیش فرض به عنوان double تفسیر می کند.

به این ترتیب، جمله

مفاهیم پایه در برنامه نویسی جاوا

```
float var = 12.34; //compile error
```

خطای کامپایل می دهد زیرا عدد 12.34 را یک عدد double فرض می کند و اجازه نمی دهد تا به یک متغیر float انتساب داده شود. در اینجا دو راه حل وجود دارد اولی cast کردن عدد 12.34 به float به صورت زیر است

```
float var = (float)12.34;
```

و دومی، افزون یکی از کاراکترهای f یا F به انتهای مقدار عددی 12.34 است.

```
float var = 12.43f;
```

تبديل داده ها، تنها شامل داده های پایه نمی شود در فصل های بعدی، «مقدمه ای بر مفاهیم شی گرایی» و «برنامه نویسی شی گرا»، با مطالب بیشتری در این زمینه آشنا خواهید شد.

ساختارهای کنترلی

زبان جاوا همانند هر زبان برنامه نویسی دیگری به شما امکان می دهد تا روند اجرای برنامه را بسته به شرایط فیزیکی کنترل کنید. به عنوان مثال شما می توانید بسته به اینکه آیا یک شرط خاص برقرار است یا خیر عملیات خاصی را انجام دهید، یا تا زمانیکه یک شرط برقرار است مجموعه ای از عملیات را تکرار کنید. در این بخش به صورت مختصر با ساختارهای کنترلی جاوا برای کنترل روند اجرای برنامه آشنا خواهید شد و در فصلهای آینده به صورت مفصل آنها را فرا خواهید گرفت. جدول زیر ساختارهای کنترلی جاوا را به صورت مختصر توضیح می دهد.

نام ساختار کنترلی	هدف استفاده
if-else	با برقراری یک شرط، و با عدم برقراری شرط، عملیاتی انجام می شود
if-else if-else	با برقراری یک شرط یک عمل، و با عدم برقراری یک شرط، شرط دیگری برای انجام عمل دیگری برسی می شود.
While	تا مادامیکه یک شرط برقرار باشد، عملیاتی تکرار می شود.
do-while	تا مادامیکه یک شرط برقرار باشد، عملیاتی تکرار می شود حتی اگر شرط برقرار نباشد یکبار عملیات انجام می شود.
for	عملیات به تعداد مشخصی تکرار می شوند این تعداد لزوما نباید در زمان کامپایل برنامه مشخص باشند
switch-case	بسته به اینکه یک متغیر چه مقداری داشته باشد عمل خاصی انجام می شود.

جدول ۴-۱۰. ساختارهای کنترلی

ساختار کنترلی if

در نوشتن یک برنامه، مواردی پیش می‌آید که لازم است با برقراری یا عدم برقراری یک شرط، کارخاصی انجام شود. اینگونه جملات شرطی را با استفاده از ساختار `if` به صورت زیر پیاده سازی می‌کنند.

```
if (condition) {
    statement;
```

```
}
```

جمله‌ای است که می‌خواهیم با برقرار بودن شرط اجرا شود، `condition` نیز همان شرط اجرای `statement` است که نتیجه ارزیابی آن مقدار `true` یا `false` خواهد بود که به معنی برقراری یا عدم برقراری شرط خواهد بود.

در صورتیکه جملات داخل بلوک `if` فقط شامل یک جمله باشد می‌توان از گذاردن `{` و `}` در ابتداء و انتهای بلوک صرفنظر کرد:

```
if (condition)
```

```
    statement
```

همچنین می‌تواند عبارتی باشد که حاصل ارزیابی آن یک مقدار `boolean` است. مثلاً در تکه کد زیر:

```
int a = 5;
```

```
.
```

```
.
```

```
.
```

```
if( a == 6) {
```

```
.
```

```
.
```

```
}
```

در صورتیکه مقدار متغیر `a` برابر 6 باشد بلوک `if` اجرا می‌شود و در غیر اینصورت از آن صرفنظر می‌شود. برای ساختن شرط می‌توانید از عملگرهای شرطی که در جدول زیر آمده است استفاده کنید.

عملگرهای جبری در ریاضیات	مثال	معنی	عملگرهای شرطی در زبان جاوا
=	x == y	x برابر y است	==

مفاهیم پایه در برنامه نویسی جاوا

\neq	$x \neq y$	x برابر y نیست	$!=$
$>$	$x > y$	x بزرگتر از y است	$>$
$<$	$x < y$	x کوچکتر از y است	$<$
\geq	$x \geq y$	x بزرگتر یا مساوی y است	\geq
\leq	$x \leq y$	x کوچکتر یا مساوی y است	\leq

جدول ۴-۱۱. عبارتهای شرط

مثال زیر، برنامه‌ای را نشان می‌دهد که در آن از طریق دو دیالوگ، دو عدد از کاربر دریافت می‌شود بسته به اینکه دو عدد مساوی، کوچکتر یا بزرگتر هستند پیغام مناسبی نمایش داده می‌شود.

```

1 //this program shows using of condition statements
2 public class UsingConditions {
3
4     public static void main(String[] args) {
5
6         int num1 = 23;
7         int num2 = 12;
8
9         if(num1 > num2)
10             show("First number is greater");
11
12         if(num1 == num2)
13             show("Both are equal");
14
15         if(num1 < num2)
16             show("First number is smaller");
17
18         System.exit(0);
19     }
20
21     public static void show(String str){
22         System.out.println(str);
23     }
24 }
```

کد ۱۳

توجه: عبارت شرط الزاماً نباید یک متغیر boolean باشد بلکه هر عبارتی که حاصل ارزیابی آن یک مقدار boolean باشد نیز می تواند به عنوان شرط استفاده شود. به عنوان مثال، عبارت `a==6` یا عبارت `a!=6` و عبارتی از این قبیل نیز می توانند به عنوان شرط استفاده شوند.

ساختار if-else

در برنامه ها موارد فراوانی وجود دارد که لازم است با برقرار بودن یک شرط، عملیات خاصی و با عدم برقراری آن شرط نیز عملیات دیگری انجام شود برای این منظور می توانید از ساختار if-else استفاده کنید
شکل کلی این ساختار به صورت زیر است:

```
if( condition ) {
    //if-operations
} else {
    //else-operations
}
```

در صورتیکه حاصل ارزیابی condition مقدار true باشد جملات داخل بلوک if (operations) و در صورتیکه مقدار false باشد جملات داخل بلوک else (operations) اجرا می شود.
هر کدام از بلوکهای if یا else که تنها شامل یک جمله باشد را می توان بدون {} و {} نیز نوشت.

توجه: توصیه می شود حتی اگر بلوکهای if یا else فقط از یک جمله تشکیل شده اند بازهم بلوکهای if یا else را با {} و {} بنویسید. این کار از خطاهای سهوی در برنامه نویسی جلوگیری می کند زیرا ممکن است در آینده جمله ای را به یک بلوک if یا else اضافه کنید در اینصورت تحت هیچ شرایطی بلوک if یا else صحبت خود را از دست نخواهد داد.

ساختار if-else چندگانه

ساختار if-else می تواند به اندازه دلخواه به صورت تو-در-تو ادامه پیدا کند مثال زیر، یک ساختار if-else تو در تو را نشان می دهد.

```
if(condition1) {
    //operations-1
} else{
    if(condition2) {
```

مفاهیم پایه در برنامه نویسی جاوا

```
//operations-2
}else{
    //operations-3
}
}
```

در صورتیکه شرط condition1 برقرار باشد، //operations-1 انجام میشود و در صورتیکه شرط condition2 برقرار نباشد بلوك else اجرا می شود که آن خود نیز شامل یک بلوك if-else دیگر است. حال اگر شرط condition2 برقرار باشد، //operations-2 انجام می شود و در صورت عدم برقراری شرط، if-else //operations-3 انجام خواهد شد. جملات فوق را میتوان به یک ساختار ساده‌تر به نام if به صورت زیر تبدیل نمود:

```
if(condition1) {
    //operations-1
}else if(condition2) {
    //operations-2
}else{
    //operations-3
}
```

در ساختار فوق، می‌توان از جملات "else if" بیشتری استفاده نمود اما در هر صورت همواره فقط یک else if وجود دارد.

```
if(condition1) {
    //operations-1
}else if(condition2) {
    //operations-2
}else if(condition3) {
    //operations-3
.
.
.
}else{
    //operations-n
}
```

مثال زیر استفاده از بلوکهای if-else چندگانه را نشان می‌دهد.

```
1 // Demonstrate if-else-if statements.
2 public class IfElse {
3
4     public static void main(String args[]) {
5         if((args==null) || (args.length==0)) {
6             System.out.println("Please, provide an integer");
7         }
8     }
9 }
```

```

7         return;
8     }
9     int month = Integer.parseInt(args[0]);
10
11    String season;
12
13    if (month==12 || month==1 || month==2)
14        season = "Winter";
15
16    else if(month==3 || month==4 || month==5)
17        season = "Spring";
18
19    else if(month==6 || month==7 || month==8)
20        season = "Summer";
21
22    else if(month==9 || month==10 || month==11)
23        season = "Autumn";
24    else
25        season = "Bogus Month";
26
27    System.out.println("April is in the " + season + ".");
28 }
29 }
```

کد

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود:

April is in the Spring.

switch ساختار

ساختار switch مشابه if else چندگانه است که براساس مقدار یک متغیر یا حاصل یک عبارت، جملات مختلفی اجرا می شوند. شکل کلی این ساختار به صورت زیر است.

```
switch(متغیر){
    case value1:
        عملیات ۱;
        break;
    case value2:
```

مفاهیم پایه در برنامه نویسی جاوا

```
    عمليات ۲;  
    Break;  
    .  
    .  
    .  
  default:  
    عمليات n;  
}
```

برای آشنایی با عملکرد این ساختار به مثال زیر توجه کنید که در آن از if-else چندگانه استفاده شده است.

```
String retStr = "";  
if( i==0 ) {  
    retStr = "Zero";  
} else if( i==1 ) {  
    retStr = "One";  
} else if( i==2 ) {  
    retStr = "Two";  
} else if( i==3 ) {  
    retStr = "Three";  
} else {  
    retStr = "Four";  
}
```

همانطور که ملاحظه می کنید، تمام شرطهای این ساختار if-else چندگانه براساس مقدار متغیر i تعریف شده اند. در مواقعي این چنین می توانید به جای ساختار فوق، از فرم ساده تر switch به صورت زیر استفاده کنید.

```
int i = 2;  
  
switch(i){  
    case 0:  
        System.out.println("Good Luck");  
        break;  
    case 1:  
        System.out.println("Bad Luck");  
        break;  
    case 2:  
        System.out.println("SoSo");  
        break;  
    default:  
        System.out.println("Invalid Value!");  
}
```

این ساختار با کلمه کلیدی `switch` آغاز می‌شود و بعد از آن، داخل پرانتز یک متغیر یا یک عبارت می‌آید، سپس بلوک `switch` با علامت `{` آغاز می‌شود.

سپس `case` های مختلف قرار می‌گیرند که هر کدام یکی از مقادیر متغیر را به همراه عملیات مربوط به آن مشخص می‌کنند. پس از کلمه کلیدی `case` علامت `:` می‌آید که مشخص کننده شروع جملاتی است که با برقرار بودن آن `case` از کلمه کلیدی `break` برای خروج و اتمام `switch` استفاده می‌شود.

در مثال فوق در صورتی که مقدار `i` برابر صفر باشد: `0` اجرا می‌شود عبارت `Good Luck` در کنسول برنامه نمایش داده می‌شود دستور `break` اجرا می‌شود و ساختار `switch` پایان می‌یابد و اجرای برنامه به بعد از بلوک `switch` منتقل می‌شود. در مورد بقیه مقادیر `i` وضع به همین شکل است. در پایان هر `case` از کلمه کلیدی `default` استفاده شده است، در صورتیکه هیچیک از `case` های قبلی اتفاق نیفتد `default` اجرا خواهد شد استفاده از گزینه `default` اختیاریست و می‌توانید از آن استفاده نکنید مثال زیر استفاده از `switch` را نشان می‌دهد.

```

1 // A simple example of the switch.
2 public class SampleSwitch {
3
4     public static void main(String args[]) {
5
6         for (int i = 0; i < 6; i++) {
7
8             switch (i) {
9                 case 0:
10                     System.out.println("i is zero.");
11                     break;
12                 case 1:
13                     System.out.println("i is one.");
14                     break;
15                 case 2:
16                     System.out.println("i is two.");
17                     break;
18                 case 3:
19                     System.out.println("i is three.");
20                     break;
21                 default:
22                     System.out.println(
23                         "i is greater than 3.");

```

```
24 }  
25 }  
26 }  
27 }
```

F-۱۵ کد

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود.

```
i is zero.  
i is one.  
i is two.  
i is three.  
i is greater than 3.  
i is greater than 3.
```

در صورتیکه کلمه `break;` را در پایان جملات `case` ننویسید جملات داخل `case` بعدی نیز اجرا می شوند برای اینکه این مطلب را کاملا درک کنید به مثال زیر توجه کنید.

```
1 // In a switch, break statements are optional.  
2 public class MissingBreak {  
3  
4     public static void main(String args[]) {  
5  
6         for (int i = 0; i < 12; i++) {  
7             switch (i) {  
8                 case 0:  
9                 case 1:  
10                case 2:  
11                case 3:  
12                case 4:  
13                    System.out.println("i is less than 5");  
14                    break;  
15                case 5:  
16                case 6:  
17                case 7:  
18                case 8:  
19                case 9:  
20                    System.out.println("i is less than 10");  
21                    break;
```

```

22         default:
23             System.out.println("i is 10 or more.");
24     }
25 }
26 }
27 }
```

کد F-۱۶

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود.

```
i is less than 5
i is less than 10
i is 10 or more.
i is 10 or more.
```

در واقع اگر کلمه `break` را ننویسید، بعدی هم به عنوان قسمتی از `case` فعلی فرض می شود. کلاس IfElse را که در مبحث مربوط به `if-else` مطرح کردیم باز دیگر ملاحظه کنید، آن کلاس را با استفاده از `switch` و با حذف برخی از جملات `break;` می توان به صورت زیر نوشت.

```

1 // An improved version of the season program.
2 public class Switch {
3
4     public static void main(String args[]) {
5
6         int month = 4;
7         String season;
8
9         switch (month) {
10            case 12:
11            case 1:
12            case 2:
13                season = "Winter";
14                break;
```

```
15      case 3:
16      case 4:
17      case 5:
18          season = "Spring";
19          break;
20      case 6:
21      case 7:
22      case 8:
23          season = "Summer";
24          break;
25      case 9:
26      case 10:
27      case 11:
28          season = "Autumn";
29          break;
30      default:
31          season = "Bogus Month";
32      }
33
34      System.out.println("April is in the " + season + ".");
35  }
36 }
```

کد F-۱۷

توجه: متغیر یا عبارتی که از آن به عنوان شرط switch استفاده می شود فقط می تواند از انواع داده های اعداد صحیح (long, int, short و byte)، کاراکتر (char) یا بولی (boolean) باشد. وقت کنید که از اعداد اعشاری (float و double) و همچنین رشته (String) نمی توان به عنوان شرط switch استفاده کرد.

ساختار while

با استفاده از ساختار while می توان تا مدامیکه یک شرط برقرار است عمل خاصی را تکرار کرد. شکل کلی این ساختار به صورت زیر است.

```
while(condition) {
    .
    .
    .
```

}

فرض کنید می خواهیم برنامه ای بنویسیم که کاربر از طریق آن، نمرات ۱۰ نفر از دانشجویان را وارد می کند و در پایان میانگین نمرات توسط برنامه محاسبه شده و نشان داده می شود. چنین برنامه ای با استفاده از حلقه while در زیر نوشته شده است.

```

1 //Calculation of average by while loop
2 public class Average {
3     public static void main(String[] args) {
4         if((args==null)|| (args.length==0)) {
5             System.out.println("Please, provide an integer");
6             return;
7         }
8         int total, //sum of grades
9             gradeValue, //grade value
10            average ; //average of all grades
11            String grade ; //grade typed by user
12
13        //initialization phase
14        total = 0;
15
16
17        int gradeCounter=0;
18        while( gradeCounter <= args.length) {
19
20            //accepts grade from user
21            grade = args[gradeCounter];
22
23            //convert String grade to int grade
24            gradeValue = Integer.parseInt(grade);
25
26            //add new grade to sum of last grades
27            total = total + gradeValue ;
28
29            //increment gradeCounter
30            gradeCounter = gradeCounter + 1 ;
31
32        }
33

```

```

34     //perform division
35     average = total/10 ;
36
37     //display average of exam grades
38     System.out.println("Class average is "+average);
39
40 }
41 }
```

کد ۱۱

در ابتدای برنامه فوق، `average` متغیر عددی تعریف شده است،
مقدار عددی که توسط هر کاربر وارد می شود را نگهداری می کند
`gradeValue`
مجموع اعدادی را که توسط کاربر وارد می شود را نگهداری می کند.
`total`
به عنوان یک شمارنده استفاده می شود از این متغیر برای کنترل تعداد
`gradeCounter`
اعدادی که توسط کاربر وارد شده است استفاده می شود.
میانگین اعدادی که کاربر وارد کرده است را نگهداری می کند.
`average`

در خط ۱۸ از برنامه، از یک حلقه `while` استفاده شده، که شرط آن کوچکتر بودن شمارنده از ۱۰ است
بدین ترتیب تازمانیکه مقدار شمارنده کوچکتر یا مساوی ۱۰ باشد حلقه `while` ادامه می یابد.
درون حلقه `while`، رشتة عددی در متغیر `grade` قرار داده شده است. سپس با فراخوانی
`parseInt()` از کلاس `Integer`، رشتة عددی تبدیل به یک مقدار عددی شده، و با مقدار
متند `total` که در ابتدا صفر است جمع گردیده است. در پایان حلقه نیز به `gradeCounter` یک واحد
افزوده شده است.

بدین ترتیب با اجرای حلقه `while`، ۱۰ عدد بواسطه یک دیالوگ از کاربر دریافت می شود و مجموع آنها در
متغیر `totalValue` نگهداری می شود.
بعد از حلقه `while`، میانگین محاسبه شده، توسط جمله `System.out.println()` نمایش داده
شده است.

اگر بلوک حلقه `while` تنها از یک جمله تشکیل شده باشد می توان از نوشتن `{}` در این حلقه صرفنظر
کرد.
تمرین: برنامه فوق را بگونه ای تغییر دهید که تازمانیکه عدد وارد شده توسط کاربر یک عدد غیر منفی است
حلقه `while` ادامه یابد در صورتیکه عدد وارد شده توسط کاربر یک عدد منفی باشد، حلقه پایان پذیرد و
میانگین اعداد وارد شده توسط کاربر محاسبه و به وی نشان داده شود.

ساختار do-while

```
do {
    .
    .
    .
} while(condition);
```

ساختار do-while نیز هم از نظر استفاده و هم از نظر کاربرد مشابه ساختار while است. تنها تفاوت آن در این است که شرط حلقه در انتهای حلقه بررسی می شود بنابراین حتی اگر شرط حلقه برقرار هم نباشد، حلقه حداقل یکبار اجرا می شود.

ساختار do-while با کلمه کلیدی do شروع می شود سپس بلوک حلقه، داخل {} قرار مشخص می شود و بعد از آن، شرط حلقه داخل پرانتز و بعد از کلمه کلیدی while می آید در انتهای نیز؛ انتهای حلقه را مشخص می کند. در هنگام اجرای برنامه، جملات داخل بلوک do اجرا می شوند با رسیدن به انتهای حلقه، شرط تکرار حلقه بررسی می شود تا در صورت برقراری شرط، اجرای حلقه تکرار شود و در غیراینصورت حلقه خاتمه یابد و کنترل به بعد از حلقه منتقل شود.

مثالی را که با حلقه while پیاده سازی نمودیم می توان با استفاده از حلقه do-while به صورت زیر پیاده سازی نمود.

```

1 //Calculation of average by do-while loop
2 public class Average1 {
3
4     public static void main(String[] args) {
5         if((args==null) || (args.length==0)) {
6             System.out.println("Please, provide an integer");
7             return;
8         }
9         int total, //sum of grades
10            gradeCounter, //number of grades entered
11            gradeValue, //grade value
12            average; //average of all grades
13         String grade; //grade typed by user
14
15         //initialization phase
16         total = 0;
17         gradeCounter = 0; //prepare to loop
18
19         //processing phase
```

```

20      do {
21          //accepts grade from user
22          grade = args[gradeCounter];
23
24          //convert String grade to int grade
25          gradeValue = Integer.parseInt(grade);
26
27          //add new grade to sum of last grades
28          total = total + gradeValue ;
29
30          //increament gradeCounter
31          gradeCounter = gradeCounter + 1 ;
32
33      } while(gradeCounter <= 10);
34
35      //perform divition
36      average = total/10 ;
37
38 //display average of exam grades
39     System.out.println(average);
40
41 }
42 }
```

۴-۱۹ کد

خطوط متفاوت این برنامه با برنامه قبلی -که از حلقه while استفاده شده بود- به صورت پرنگ نشان داده شده است مقدار اولیه شمارنده به جای ۱ از ۰ شروع شده، همچنین شرط حلقه نیز در انتهای حلقه بررسی می شود.

در این مثال، در ابتدا بدون هیچ شرطی، جملات داخل بلوک do اجرا می شوند در انتهای بلوک، مقدار متغیر gradeCounter یک واحد افزایش می یابد و سپس توسط جمله while(gradeCounter>0) مقدار عددی gradeCounter بررسی می شود در صورتیکه مقدار آن کوچکتر یا مساوی ۱۰ باشد حلقه یکبار دیگر ادامه می یابد.

تفاوت while با do-while این است که از آنجاییکه در while، شرط حلقه در پایان حلقه بررسی شود حتی با برقرار نبودن شرط نیز، حلقه حداقل یک بار اجرا می شود.

ساختار for

```

        عبارت ۳؛ عبارت ۲؛ عبارت ۱
for( ) {
    //any operations
}

```

از حلقه `for` نیز همانند `do-while` و `while` برای تکرار مجموعه ای از عملیات استفاده می شود با این تفاوت که حلقه `for` به شما امکان می دهد تا جملات حلقه را به تعداد دفعات مشخصی تکرار کنید مثلاً جملاتی را ۱۰ بار تکرار کنید. حلقه `for` از یک متغیر `int` به عنوان شمارنده استفاده می کند، تا با هر بار اجرای حلقه مقدار شمارنده افزایش یابد و هر بار بتوان مشخص نمود که حلقه چندبار تکرار شده و آیا به تعداد مورد نظر اجرا شده است یا خیر.

توسط «عبارت ۱» مشخص می شود «عبارت ۳» مشخص می کند که با هر بار اجرای حلقه، چند واحد به مقدار شمارنده اضافه شود. «عبارت ۲» شرطی است که در هر بار اجرای حلقه بررسی می شود و در صورتیکه شرط برقرار باشد اجرای حلقه ادامه می یابد. این شرط معمولاً تعداد دفعات حلقه را بررسی می کند تا به محض اینکه تعداد دفعات اجرای حلقه با تعداد مورد نظر برابر شد اجرای حلقه را خاتمه یابد.

همانطور که در بالا ملاحظه می کنید این ساختار با کلمه کلیدی `for` به همراه سه عبارت مشخص می شود:

عبارت ۱: مشخص کننده وضعیت آغاز حلقه است، این عبارت معمولاً شمارنده حلقه را تعریف می کند.

عبارت ۲: مشخص کننده شرط ادامه حلقه است، این عبارت معمولاً شمارنده حلقه را با یک مقدار ثابت یا با یک متغیر دیگر مقایسه می کند تا مشخص کند آیا حلقه به تعداد مورد نظر اجرا شده است یا خیر.

عبارت ۳: جملاتی است که با هر بار اجرای حلقه انجام می شود. این عبارت معمولاً یک واحد به شمارنده حلقه اضافه می کند.

توجه کنید که همه عبارتهای فوق با؛ از یکدیگر جدا شده اند و ممکن است هر کدی در آنها نوشته شود. در مثال زیر، از حلقه `for` برای چاپ شمارش معکوس از ۱۰ تا صفر استفاده شده است.

```

1 //Demonstrate the for loop.
2 public class ForTick {
3
4     public static void main(String args[]) {
5         int n;
6         for(n=10; n>0; n--)
7             System.out.println("tick " + n);
8     }
9 }
```

کد ۴-۲۰

در این حلقه ابتدا وضعیت شروع حلقه $n=0$ تعیین شده است هر بار با بررسی شرط $n > 0$ اجرای حلقه ادامه می یابد و با هر بار اجرای حلقه، توسط جمله $n--$ مقدار n یک واحد کاهش می یابد. نتیجه اجرای برنامه فوق، مشابه آنچه در `while` و `do-while` دیدید خواهد بود.

در حلقه `for` هر کدام از عبارات ۱، ۲ و ۳ می تواند وجود نداشته باشند کدهای ۵-۲۱ و ۵-۲۲ مشابه کد ۵-۲۰ هستند که در آنها به ترتیب عبارت اول و عبارت سوم حذف شده است.

```
1 public class ForTick1 {
2     public static void main(String args[]) {
3         int n=10;
4
5         for(; n>0; n--)
6             System.out.println("tick " + n);
7     }
8 }
9 public class ForTick2 {
10    public static void main(String args[]) {
11        int n=10;
12
13        for(n=10; n>0;) {
14            System.out.println("tick " + n);
15            n--;
16        }
17    }
18 }
```

کد ۴-۲۱

توجه: در یک حلقه `for`، هر سه عبارت می توانند وجود نداشته باشند در اینصورت حلقه `for` به صورت زیر در خواهد آمد.

```
for(;;) {  
}
```

به این حلقه `for` که هیچ شرط خاتمه‌ای ندارد حلقه‌ی بی‌نهایت گفته می‌شود.

کنترل اجرای حلقه‌ها با استفاده از `break` و `continue`

به طور معمول برای خروج و اتمام هر حلقه‌ای، باید شرط آن حلقه `false` شود. اما در مواقعی ممکن است بخواهید درون حلقه (مثلاً با رخداد یک حادثه، برقراری یک شرط دیگر،...) اجرای حلقه را خاتمه دهید. کلمه `break` برای این منظور پیش‌بینی شده است. به تکه کد زیر توجه کنید.

```
while (b) {
    .
    .
    .
    if (i==10) {
        break;
    }
    System.out.println("Counter is : "+i);
    .
    .
    .
}
```

برای اینکه حلقه `while` فوق خاتمه یابد باید مقدار متغیر `b` برابر `false` شود اما همانطور که ملاحظه می‌کنید در نقطه‌ای از حلقه `while` شرط `i == 10` بررسی شده است تا درصورتیکه این شرط برقرار باشد، با اجرای `break` حلقه `while` بلافاصله خاتمه یابد.

کلمه `continue` نیز کارکردی مشابه `break` دارد با این تفاوت که اجرای برنامه را به ابتدای حلقه منتقل می‌کند. برای درک عملکرد `continue` به مثال زیر توجه کنید.

```
int i=0;
while(true){
    if(i !=10){
        i++;
        continue ;
    }else{
        System.out.println("Counter is : "+i);
        break;
    }
}
```

همانطور که ملاحظه می‌کنید، شرط حلقه `while` مقدار `true` است بنابراین علی‌الظاهر این حلقه همواره اجرا می‌شود. اما در حلقه `while` تا مادامیکه شرط `i != 10` برقرار باشد یک واحد به `i` افزوده می‌شود و با استفاده از `continue` کنترل به ابتدای حلقه منتقل می‌شود زمانیکه `i` برابر 10 شود جمله `System.out.println("Counter is : "+i);`

مفاهیم پایه در برنامه نویسی جاوا

اجرا خواهد شد. بنابراین نتیجه اجرای کد فوق، عبارت Counter is 10 در کنسول برنامه خواهد بود.
کلمات کلیدی break و continue برای تمام حلقه های while، do-while و for قابل استفاده هستند.

توجه: حلقه های do-while و while، for می توانند به صورت تودرتو نیز استفاده شوند. به عبارت دیگر داخل هر حلقه do-while یا while، for یا while، for می توان یک حلقه while، for یا do-while داشت.

در صورتیکه در یک برنامه، حلقه های تودرتو داشته باشید استفاده از continue یا break را روی اجرای نزدیکترین حلقه اثر می گذارند برای درک این مطلب به مثال زیر دقت کنید.

```
1 public class BreakLoop {  
2     public static void main(String args[]) {  
3           
4             for (int i = 0; i < 3; i++) {  
5                 System.out.print("Pass " + i + ": ");  
6                 for (int j = 0; j < 100; j++) {  
7                     // terminate loop if j is 10  
8                     if (j == 10)  
9                         break;  
10                     System.out.print(j + " ");  
11                 }  
12                 System.out.println();  
13             }  
14             System.out.println("Loops complete.");  
15         }  
16     }
```

۴-۲۲ کد

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.

```
Pass 0: 0 1 2 3 4 5 6 7 8 9  
Pass 1: 0 1 2 3 4 5 6 7 8 9  
Pass 2: 0 1 2 3 4 5 6 7 8 9  
Loops complete.
```

```

1 // Using continue with a label.
2 public class ContinueLabel {
3
4     public static void main(String args[]) {
5         for (int i = 0; i < 10; i++) {
6             for (int j = 0; j < 10; j++) {
7                 if (j > i) {
8                     System.out.println();
9                     continue;
10                }
11                System.out.print(" " + (i * j));
12            }
13        }
14        System.out.println();
15    }
16 }
```

کد ۲۳

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود:

0									
0	1								
0	2	4							
0	3	6	9						
0	4	8	12	16					
0	5	10	15	20	25				
0	6	12	18	24	30	36			
0	7	14	21	28	35	42	49		
0	8	16	24	32	40	48	56	64	
0	9	18	27	36	45	54	63	72	81

حلقه بی نهایت

در صورتیکه شرط حلقه `while` یا `do-while` همواره برقرار (true) باشد اجرای حلقه تا همیشه ادامه می یابد به حلقه ای که شرط آن همیشه برقرار است حلقة بی نهایت گفته می شود عموما در چنین برنامه ای، شرطی درون حلقه گنجانده می شود تا با برقراری آن شرط، اجرای حلقه خاتمه یابد. حلقه های زیر نمونه هایی از حلقه های بی نهایت هستند:

```

while(true) {
    //statements
}

do{
    //statements
```

مفاهیم پایه در برنامه نویسی جاوا

```
}while(true);  
  
for(int i=0;;i++){  
    //statements  
}
```

کلمه کلیدی return

یکی دیگر از جملات کنترلی، جمله return است فراخوانی این جمله باعث خاتمه اجرای متد می شود اگر متدی مقدار برگشتی نداشته باشد (مقدار برگشتی آن void باشد) از عبارت return می توان برای خاتمه اجرای آن متد در هر نقطه ای از آن متد استفاده نمود. اگر متد مقدار برگشتی نداشته باشد (مقدار برگشتی آن void نباشد) از عبارتی مشابه var که در آن return var استفاده می شود طبیعی است که در نقطه ای که مقدار برگشتی متد برگردانده می شود اجرای متد نیز خاتمه می یابد و مقدار به فراخواننده آن متد داده می شود.

در هر نقطه ای از متد و به هر شکلی که استفاده شود باعث خاتمه اجرای آن متد خواهد شد.

توجه: اگر return در هر نقطه ای از متد اجرا شود، اجرای متد خاتمه می یابد و جملات بعد از آن اجرا نخواهند شد.

```
1 //Demonstrate return.  
2 public class Return {  
3  
4     public static void main(String args[]) {  
5         boolean t = true;  
6  
7         System.out.println("Before the return.");  
8  
9         if(t)  
10             return; // return to caller  
11  
12         System.out.println("This won't execute.");  
13     }  
14 }
```

اجرای کلاس فوق نتیجه زیر را در بر خواهد داشت.

Before the return.

توجه: عملکرد break بسیار شبیه return است در حالیکه از break برای خاتمه اجرای حلقه استفاده می شود return اجرای متدهد را خاتمه می دهد.

آرایه‌ها

آرایه‌ها متغیرهایی هستند که بیش از یک مقدار را نگهداری می‌کنند. مثلاً جمله زیر:

```
String[] strs;
```

متغیری به نام `strs` تعریف می‌کند که حاوی مجموعه‌ای از مقادیر است که همگی از جنس `String` هستند. به متغیرهایی شبیه این متغیر که به جای یک مقدار، مجموعه‌ای از مقادیر از یک نوع را نگهداری می‌کنند، آرایه گفته می‌شود.

تعریف و ایجاد آرایه

جمله `String[] strs;` فقط آرایه‌ای به نام `strs` را تعریف می‌کند که همه عناصر آن از جنس `String` هستند اما تعداد و مقدار هریک از عناصر را مشخص نمی‌کند. برای مشخص کردن تعداد عناصر آرایه باید با استفاده از کلمه کلیدی `new` استفاده کنید به جملات زیر توجه کنید.

```
String[] strs ;  
strs = new String[4];
```

جمله اول، آرایه‌ای به نام `strs` تعریف می‌کند که عناصر آن از جنس `String` هستند جمله دوم، مشخص می‌کند که این آرایه چهار عنصر دارد (آن آرایه را با چهار عنصر در حافظه می‌سازد). همانطور که ملاحظه می‌کنید برای ساختن آرایه از کلمه کلیدی `new` استفاده شده است کلمه `new`، از کلمات مهم در جاواست که در مورد معنی و مفهوم آن در فصلهای بعدی صحبت خواهیم کرد.

توجه: در جاوا محدودیتی در تعداد عناصر آرایه وجود ندارد بنابراین یک آرایه می‌تواند حداقل تا اندازه‌ای که داده نوع `int` اجازه می‌دهد عنصر داشته باشد.

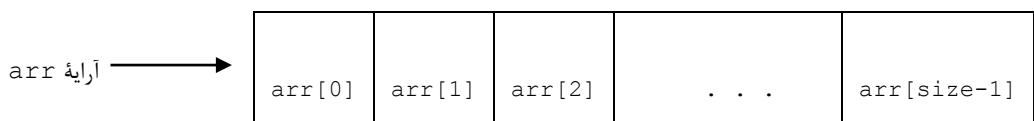
به هریک از متغیرهایی که توسط آرایه نگهداری می‌شوند یک «عنصر» گفته می‌شود. برای دسترسی به هریک از عناصر آرایه از موقعیت فیزیکی آن عنصر در آرایه - که به آن ایندکس عنصر در آرایه نیز گفته می‌شود- استفاده می‌شود

در مثال فوق موقعیت اولین عنصر آرایه `0` و موقعیت دومین عنصر آرایه `1` است عنصر اول آرایه فوق تحت عنوان `strs[0]` و عنصر دوم نیز تحت عنوان `strs[1]` قابل دسترس می‌باشند. دقت کنید از آنجائیکه موقعیت عناصر آرایه از `0` شروع می‌شود همیشه موقعیت آخرین عنصر یک واحد کمتر از تعداد عناصر آرایه

است در این مثال ایندکس آخرین عنصر 3 است در حالیکه آرایه 2 عنصر دارد. بنابراین اگر arr معرف یک آرایه با n عنصر باشد عناصر آرایه با نامهای

```
arr[0]
arr[1]
arr[2]
.
.
.
arr[n-1]
```

قابل دسترس خواهد بود. (چون ایندکس آرایه از 0 شروع می‌شود همواره آخرین عنصر آرایه، n-1 امین عنصر خواهد بود).



توجه: اگرچه «تعریف کردن یک آرایه» و «تعیین تعداد عناصر آرایه» می‌توانند به عنوان دو عمل در دو خط متمایز از برنامه انجام شوند، اما آنها را نیز می‌توان در یک خط برنامه به صورت زیر ادغام نمود:

```
String[] strs = new String[2];
```

معمولًا تعریف کردن آرایه و ایجاد آن در حافظه، همزمان انجام می‌شود (همانند آنچه در فوق ملاحظه می‌کنید)

دسترسی به عناصر آرایه و مقداردهی آنها

برای مقدار دهی کردن عناصر یک آرایه می‌توانید عناصر آنرا جداگانه مقدار دهی کنید، یا تمام آرایه را در زمان ایجاد، مقدار دهی کنید:

روش اول:

```
String[] str = new String[2];
str[0] = "name";
str[1] = "family";
```

روش دوم

مفاهیم پایه در برنامه نویسی جاوا

```
String[] str = new String[]{"name", "family"};
```

روش سوم

```
String[] str = {"name", "family"};
```

تفاوت روش اول و روش‌های دوم و سوم در این است که در روش‌های دوم و سوم تعداد عناصر آرایه مستقیماً مشخص نشده است در عوض از روی تعداد مقادیر تعداد عناصر آرایه مشخص می‌شود.

برای ساختن آرایه‌هایی از جنس دیگر نیز می‌توان به همین روش عمل کرد مثلاً

```
int[] a = new int[5];  
float[] f = new float[12];
```

هر کدام به ترتیب آرایه‌ای 5 تایی از `int` و آرایه‌ای 12 تایی از `float` می‌سازند.

تمرین: یک آرایه 5 تایی از جنس `String` تعریف کنید و به عناصر آن مقادیر دلخواه بدهید، سپس با استفاده از یک حلقه `for`، مقادیر آنرا در خروجی استاندارد چاپ کنید.

برای بدست آوردن تعداد عناصر آرایه می‌توان از متغیر `length` که در هر آرایه تعریف شده است استفاده کرد مثلاً اگر آرایه‌ای با نام `c` تعریف کرده باشد عبارت `c.length` مشخص کننده طول آرایه یا همان تعداد عناصر آرایه است

```
.  
. .  
String[] c = new String[25];  
System.out.println("length of c is="+c.length);  
. .  
.
```

نتیجه اجرای تکه کد فوق چنین است:

```
length of c is=25
```

در زیر مثالی ارایه شده است که در آن تعداد روزهای هر ماه از سال توسط آرایه ای از اعداد `int` نشان داده شده است سپس تعداد روزهای ماه آوریل در خروجی استاندارد چاپ شده است.

```
1 // Demonstrating days of month in array format  
2 public class DaysOfMonthArray {  
3  
4     public static void main(String args[]) {  
5         int month_days[] = { 31, 28, 31, 30,  
6                             31, 30, 31, 31,
```

```

7           30, 31, 30, 31 };
8
9     System.out.println("April has " +
10    month_days[3] + " days.");
11   }
12 }
```

کد ۱۹

در زیر مثال دیگری نشان داده شده است که در آن آرایه ای از اعداد اعشاری double ساخته شده است سپس میانگین این اعداد محاسبه شده و در خروجی استاندارد چاپ شده است.

```

1 // Average an array of values.
2 public class Average2 {
3
4     public static void main(String args[]) {
5         double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};
6         double result = 0;
7         int i;
8
9         for(i=0; i<5; i++)
10            result = result + nums[i];
11
12         System.out.println("Average is " + result/5);
13     }
14 }
```

کد ۱۹

آرایه‌های دو بعدی

آرایه‌ای که تاکنون با آن آشنا شدید به آرایه یک بعدی مشهور است زیرا هر عنصر آرایه با یک عدد که همان ایندکس است مشخص می‌شود. اگر بخواهید متغیری تعریف کنید و با استفاده از آن متغیر، داده‌های داخل یک ماتریس یا یک جدول را مقدار دهی، نمایش و یا دستکاری کنید می‌توانید از آرایه دو بعدی استفاده کنید.

Column0 Column1

ColumnN

مفاهیم پایه در برنامه نویسی جاوا

Row0	a [0] [0]			
Row1				
RowM				a [M-1] [N-1]

هر عنصر ماتریس با استفاده از دو عدد مشخص می‌شود که همان طول و عرض ماتریس است. هر عنصر آرایه دو بعدی نیز با استفاده از دو عدد مشخص می‌شود. جمله زیر

```
String[][] strs = new String[4][3];
```

یک آرایه دو بعدی به طول 4 و عرض 3 تعریف می‌کند.
مشابه آرایه یک بعدی، برای مقدار دهی یا دستکاری مقادیر عناصر آرایه دو بعدی از ایندکس عناصر استفاده می‌شود منتها عناصر آرایه دو بعدی دارای ایندکس دوتایی هستند.

```
strs[0][0]  
strs[1][0]  
strs[2][0]  
. . .  
strs[3][2]
```

به عنوان یک قانون کلی، در صورتیکه آرایه‌ای $[m][n]$ تعریف کرده باشیم این آرایه $n \times m$ عنصر دارد که عناصر آن از $[0][0]$ شروع می‌شود و با $[m-1][n-1]$ خاتمه می‌یابد. برای مقدار دهی کردن آرایه دو بعدی همانند آرایه یک بعدی، سه روش وجود دارد:
روش اول:

```
int[][] a = new int[2][3];
a[0][0] = 1;
a[0][1] = 2 ;
a[0][2] = 0;
a[1][0] = -1;
a[1][1] = 1;
a[1][2] = 2;
```

روش دوم:

```
int[][] a = new int[][]{{1, 0, -1},{2, 1, 2}};
```

روش سوم:

```
int[][] a = {{1, 0, -1},{2, 1, 2}};
```

در زیر مثالی نشان داده شده است که در آن یک آرایه دو بعدی با ابعاد ۴ و ۵ ساخته می شود سپس ایندکس هر عنصر به عنوان مقدار آن عنصر قرار داده شده است در پایان، مقادیر داخل آرایه در خروجی استاندارد (به شکل یک ماتریس) چاپ شده است.

```
1  /**
2   * This program demonstrate using of two dimensional array
3   */
4  public class TwoDimArray {
5
6      public static void main(String[] args){
7          String[][] twoDim = new String[4][5];
8
9          //assinging values
10         for (int i=0; i<twoDim.length; i++) {
11             for (int j=0; j<twoDim[i].length; j++) {
12                 twoDim[i][j] = "["+i+"]["+j+"]";
13             }
14         }
15
16         //printing values
17         for(int i=0; i<twoDim.length; i++) {
18             for(int j=0; j<twoDim[i].length; j++) {
19                 System.out.print(twoDim[i][j]+" ");
20             }
21         }
22     }
23 }
```

مفاهیم پایه در برنامه نویسی جاوا

```
21         System.out.println();
22     }
23
24 }
25 }
```

کد ۱۹

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود:

```
[0][0] [0][1] [0][2] [0][3] [0][4]
[1][0] [1][1] [1][2] [1][3] [1][4]
[2][0] [2][1] [2][2] [2][3] [2][4]
[3][0] [3][1] [3][2] [3][3] [3][4]
```

در زیر مثال دیگری نشان داده است که در آن آرایه ای از اعداد **double** تعریف شده اند مقادیر هر عنصر این آرایه برابر حاصلضرب ایندکسهای آن عنصر است.

```
1 // Initialize a two-dimensional array.
2 public class Matrix {
3
4     public static void main(String args[]) {
5         int m[][] = {{ 0*0, 1*0, 2*0, 3*0 },
6                     { 0*1, 1*1, 2*1, 3*1 },
7                     { 0*2, 1*2, 2*2, 3*2 },
8                     { 0*3, 1*3, 2*3, 3*3 }};
9         int i, j;
10
11         for(i=0; i<4; i++) {
12             for(j=0; j<4; j++)
13                 System.out.print(m[i][j] + " ");
14             System.out.println();
15         }
16     }
17 }
```

کد ۲۰

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود:

```
0.0  0.0  0.0  0.0
0.0  1.0  2.0  3.0
0.0  2.0  4.0  6.0
0.0  3.0  6.0  9.0
```

آرایه‌های چند بعدی

همانند آرایه‌های دو بعدی، آرایه‌هایی با ابعاد بیشتر نیز وجود دارند که برای آدرس دهی هر عنصر آنها باید از ایندکس چندگانه استفاده نمود.

برای مثال جملهٔ

```
String[][][] strs = new String[2][3][5];
```

یک آرایه از جنس String با طول 2، عرض 3، و ارتفاع 5 تعریف می‌کند عناصر این آرایه به صورت

```
str[0][0][0]
```

```
str[0][0][1]
```

```
.
```

```
.
```

```
.
```

```
str[0][0][4]
```

```
str[0][1][0]
```

```
str[0][1][1]
```

```
.
```

```
.
```

```
.
```

```
str[0][1][4]
```

```
str[0][2][0]
```

```
str[0][1][1]
```

```
.
```

```
.
```

```
.
```

```
str[1][2][3]
```

```
str[1][2][4]
```

است.

در تعریف آرایه‌های چند بعدی محدودیتی وجود ندارد و شما می‌توانید آرایه‌های n بعدی داشته باشید در صورتیکه آرایه ای با ابعاد $[k][m][n]$ تعریف کنید ایندکس اولین عنصر آن $[0][0][0]$ و ایندکس آخرین عنصر آن $[n-1][m-1][k-1]$ خواهد بود.

مفاهیم پایه در برنامه نویسی جاوا

در برنامه زیر آرایه ای سه بعدی با ابعاد 3، 4 و 5 ساخته شده است سپس هر عنصر با حاصلضرب ایندکسهای همان عنصر مقداردهی می شود در پایان مقادیر عناصر آرایه به صورت چندین ماتریس در خروجی استاندارد چاپ می شوند.

```
1 // Demonstrate a three-dimensional array.  
2 public class ThreeDMatrix {  
3  
4     public static void main(String args[]) {  
5         int threeD[][][] = new int[3][4][5];  
6         int i, j, k;  
7  
8         for (i = 0; i < 3; i++)  
9             for (j = 0; j < 4; j++)  
10                for (k = 0; k < 5; k++)  
11                    threeD[i][j][k] = i * j * k;  
12  
13        for (i = 0; i < 3; i++) {  
14            for (j = 0; j < 4; j++) {  
15                for (k = 0; k < 5; k++)  
16                    System.out.print(threeD[i][j][k]);  
17                System.out.println();  
18            }  
19            System.out.println();  
20        }  
21    }  
22 }
```

۱-۱۹ کد

خروجی اجرای این برنامه به صورت زیر خواهد بود:

```
0 0 0 0 0  
0 0 0 0 0  
0 0 0 0 0  
0 0 0 0 0
```

```
0 0 0 0 0  
0 1 2 3 4  
0 2 4 6 8  
0 3 6 9 12
```

```
0 0 0 0 0  
0 2 4 6 8  
0 4 8 12 16
```

0 6 12 18 24

پارامترهای متغیر

وقتی یک متند با تعدادی پارامتر تعریف می شود، در هنگام فراخوانی آن متند می بایست مقادیر تمام پارامترهای آن متند مشخص شود. به عنوان مثال اگر در یک کلاس متندی به صورت زیر تعریف شده باشد

```
public void execute(int a, String b) {
    ...
}
```

برای فراخوانی متند `execute()` باید مقدار هر دو پارامتر `a` و `b` مشخص شود بنابراین فراخوانیهای زیر با خطای کامپایل مواجه خواهد شد.

```
execute(1);
execute(1, "some strings");
```

در جاوا مفهومی با نام «مقدار پیش فرض» برای پارامترهای متند وجود ندارد. به شکل مشابه اگر متندی به صورت زیر تعریف شود

```
public void process(String[] args) {
    ...
}
```

در فراخوانی این متند نیز می بایست مقدار پارامتر `args` مشخص شود. دقت کنید که در مواقعي که نوع یک پارامتر از جنس مقادیر پایه نیست، (از جنس یک کلاس است یا آرایه است) می توان مقدار `null` نیز به عنوان مقدار پارامتر مشخص کرد. مثلاً متند `process()` را می توان به صورت زیر فراخوانی کرد

```
process(null); //correct
```

امکان جالبی که در جاوا وجود دارد تعریف پارامترهای متغیر (`varargs`) است. کلاس `Summer` را با دو متند `sum()` ملاحظه کنید.

```
1 class Summer {
2
3     public int sum(int a, int b) {
4         return a+b;
5     }
6
7     public int sum(int a, int b, int c) {
```

```
8     return a+b+c;
9 }
10}
11}
```

کد F-۱۹

هر دو متدهای این کلاس مجموع مقادیر پارامترهایشان را برمی‌گردانند. یعنی اگر بخواهیم مجموع دو عدد را محاسبه کنیم اولین متدهای این کلاس است، و اگر بخواهیم مجموع سه عدد را محاسبه کنیم دومین متدهای این کلاس است. حال سوال اینجاست که اگر بخواهیم مجموع چهار عدد را محاسبه کنیم باید چه متدهای فراخوانی کنیم؟ ممکن است به نظرتان برسد که متدهای جدیدی به کلاس اضافه کنیم که چهار عدد دریافت می‌کنند و مجموع چهار عدد را برمی‌گردانند. اما راه حل بهتر استفاده از پارامترهای متغیر است. با حذف دو متدهای sum() فوق، متدهای جدید را به صورت زیر به کلاس Summer اضافه می‌کنیم.

```
1 public class Summer1 {
2     public int sum(int... args) {
3         int sum = 0;
4         for(int x: args) {
5             sum += x;
6         }
7         return sum;
8     }
9 }
10}
```

کد F-۱۹

پارامتر int... args به معنی، «صفر، یک، یا بیش از یک عدد int است» که در نهایت در متدهای sum() به صورت یک آرایه دریافت می‌شود و قابل پردازش است. با تعریف متدهای sum() به صورت فوق، می‌توان آنرا به آشکال زیر فراخوانی کرد.

```
sum();
sum(1);
sum(5, 2);
sum(6, 2, 1, 0);
```

(تمام فراخوانی‌های فوق صحیح هستند:-)

توجه داشته باشید که هر متدهایی که بر پارامتر متغیر، پارامتر دیگری نیز داشته باشد مشروط بر اینکه پارامتر دیگر اولاً خودش پارامتر متغیر نباشد و ثانیاً باعث نامفهوم شدن متدهای کامپایلر نشود. به این ترتیب متدهایی که زیر صحیح است ممکن است باشند.

```
public int sum(int...args, String msg) {
    ...
}
```

اما متدهایی که زیر صحیح نیستند و خطای کامپایل تولید می‌کنند.

```
public int sum(int... args, int a) {
    ...
}

public int sum(int...args1, String...args2) {
    ...
}
```

خلاصه

در این فصل، با انواع داده های پایه آشنا شدید داده های پایه شامل `long`, `char`, `int`, `byte`, `float` و `short` و `double` هستند از آنجاییکه از این داده ها برای تعریف داده های پیچیده تر استفاده می شود به آنها «داده های پایه» گفته می شود.

در مواردی که نیاز به نگهداری یک مقدار، جمع و تفریق کردن یا هرگونه محاسبات دیگری روی آن است، می توان از متغیر استفاده کرد. متغیر در حقیقت قسمتی از حافظه کامپیوتر است که یک نام به آن انتساب داده می شود و یک مقدار در آن نگهداری می شود. در طول یک برنامه می توان مقدار متغیر را تغییر داد و آنرا با یک مقدار جدید جایگزین نمود.

برای تعریف یک متغیر ضمن اینکه باید یک نام برای آن در نظر گرفته شود، باید نوع داده ای که توسط آن متغیر نگهداری می شود را نیز مشخص نمود. همچنین در هنگام تعریف متغیر می توان به آن مقدار اولیه انتساب داد یا بدون مقدار اولیه، آنرا تعریف نمود.

هر متغیر در محدوده خاصی از برنامه قابل استفاده است، محدوده های داخل یک برنامه به سه دسته قابل دسته بندی است:

الف-محدوده یک بلوک

ب-محدوده یک متند

ج-محدوده یک کلاس

هر بلوک به قسمتی از برنامه گفته می شود که بین { و } قرار دارند بدین ترتیب کدهایی که بین { و } در یک جمله `if` یا عباراتی مشابه قرار دارند یا کدهایی که داخل یک متند یا داخل یک کلاس نوشته می شوند در محدوده آن بلوک محسوب می شوند.

برای انجام محاسبات روی داده ها، در جاوا پنج دسته عملگر تعریف شده اند: عملگرهای انتساب، محاسباتی، بیتی، مقایسه ای و منطقی. عملگر «انتساب» برای انتساب یک مقدار به یک متغیر استفاده می شود، عملگر «محاسباتی» برای انجام عملیات محاسباتی روی مقدار داده انجام می شود، عملگر «بیتی» برای انجام عملیات روی بیت‌های داده ها استفاده می شود، عملگر «مقایسه ای» برای مقایسه مقدار دو داده (کوچکتر، مساوی، بزرگتر,...) بکار می رود و عملگر «منطقی» برای انجام اعمال منطقی (`and`, `or`, `not`) استفاده می شوند. عملگرهایی که در یک عبارت قرار دارند اولویت های متفاوتی هستند در بین عملگرهای، ضرب و تقسیم بالاترین اولویت و جمع و تفریق پایین ترین اولویت را دارا هستند با استفاده از پرانتز می توان اولویت عملگرهای را تغییر داد.

تبديل یک داده به داده دیگر (انتساب مقدار یک متغیر به یک متغیر دیگر که از نوع دیگری تعریف شده است) فقط برای داده های سازگار امکان پذیر است، در بین داده های پایه، تنها داده های عددی سازگار محسوب می شوند. در این موقع اگر نوع داده مقصود (متغیری که مقدار به آن انتساب داده می شود) بزرگتر از نوع داده مبدأ باشد می توان با استفاده از عملگر «انتساب»، داده مبدأ را به داده مقصود انتساب داد، اما اگر نوع

داده مقصد کوچکتر از داده مبدأ باشد ممکن است در حین انتساب، قسمتی از داده‌ها از بین برود به همین دلیل باید با مشخص کردن نوع داده مقصد، به کامپایلر و اجرا کننده جاوا بگویید که با علم به از بین رفتن داده‌ها، تبدیل داده را انجام داده اید.

از ساختارهای کنترلی، برای کنترل اجرای برنامه بر اساس مقدار متغیرهای داخل برنامه استفاده می‌شود. ساختارهای `if`, `if-else`, `if-else if-else`, `if` یک شرط یا مجموعه‌ای از شروط، عملیات مختلفی در برنامه انجام شود.

ساختار `switch` شبیه ساختار `if-else` است که در آن براساس مقدار یک متغیر از جنس `char`, `byte`, `int` از حلقه‌های `for`, `while` و `do-while` برای اجرای مکرر قسمتی از یک برنامه استفاده می‌شود. حلقه‌های `while` و `do-while` تا هر زمان که شرط حلقه برقرار باشد اجرا می‌شوند تفاوت `do-while` در این است که از آنجاییکه در حلقه `do-while` شرط حلقه در انتهای حلقه بررسی می‌شود، حتی با برقرار نبودن شرط حلقه، حداقل یکبار اجرا می‌شود.

از حلقه `for` برای اجرای قسمتی از برنامه، به تعداد دفعات مشخص استفاده می‌شود. کلمات کلیدی `break` و `continue` برای تغییر روند اجرای یک حلقه استفاده می‌شوند، باعث انتقال اجرای حلقه به ابتدای حلقه و `break` باعث اتمام اجرای حلقه و خروج از حلقه می‌شود. `return` نیز یکی دیگر از جملات کنترلی است که باعث اتمام اجرای یک متند می‌شود جملاتی از متند که بعد از `return` قرار دارند اجرا نمی‌شوند.

از آرایه‌ها برای تعریف یک متغیر با چندین مقدار استفاده می‌شود، به عبارت دیگر، یک آرایه به مجموعه‌ای از مقادیر از یک نوع گفته می‌شود. هریک از مقادیر داخل آرایه براساس موقعیت آن عنصر شناسایی می‌شود، که به آن ایندکس گفته می‌شود ایندکس یک عدد است که مشخص کننده موقعیت یک عنصر داخل آرایه است، این بدین معناست که برای دسترسی یا تغییر مقدار هر یک از عناصر آرایه باید از ایندکس آن عنصر استفاده شود.

برای تعریف هر آرایه، نام آن به همراه نوع آن آرایه مشخص می‌شود مقداردهی عناصر آرایه نیز می‌تواند در زمان تعریف و ایجاد آرایه، یا بعد از آن صورت گیرد. آرایه‌ها می‌توانند چندبعدی باشند که در اینصورت برای مقداردهی یا بدست آوردن مقدار یک عنصر درون آرایه باید بیش از یک ایندکس را مشخص نمود.

تمرینات

- ۱) آرایه‌ای ۱۰ تایی از اعداد بسازید و مقادیر آنرا در خروجی استاندارد چاپ کنید.
- ۲) آرایه‌۱۰ تایی را که ساخته‌اید مقدار دهی کنید و سپس مقادیر آنرا در خروجی استاندارد چاپ کنید.
- ۳) متدهی بنویسید که آرایه‌ای از اعداد را به عنوان پارامتر دریافت می‌کند و مقادیر آنرا در خروجی استاندارد چاپ می‌کند سپس آنرا از طریق متدهی `main` فراخوانی کنید.
- ۴) آرایه‌ای ۵ تایی از رشته بسازید و مقادیر پیش‌فرض آنرا در خروجی استاندارد چاپ کنید.
(نکته: وقتی آرایه‌ای از آبجکت می‌سازید مقادیر پیش‌فرض تمام آنها `null` است)
- ۵) متدهی بنویسید که آرایه‌ای از اعداد دریافت می‌کند و میانگین آنها را بصورت یک عدد `double` برمی‌گرداند.
- ۶) آرایه‌ای از اعداد بسازید و عناصر آنرا با اعداد دلخواه خود مقدار دهی کنید. در پایان به اندازه مقدار عددی هر کدام از عناصر آرایه، یک `Progress Bar` در خروجی استاندارد بسازید. مثلا اگر عدد ۵ باشد پنج ستاره در خروجی چاپ کنید:

- ۷) متدهی بنویسید که آرایه‌ای از اعداد را دریافت نموده و سپس عناصر آرایه را به صورت معکوس درون آرایه قرار دهد.
- ۸) متدهی بنویسید که یک آرایه دو بعدی از `int` را دریافت می‌کند و مقادیر آنرا در خروجی استاندارد چاپ می‌کند.
- ۹) متدهی بنویسید که آرایه‌ای دو بعدی از `int` دریافت می‌کند و بزرگترین عنصر آن آرایه را برمی‌گرداند.
- ۱۰) متدهی بنویسید که آرایه‌ای دو بعدی از `int` دریافت می‌کند و میانگین عناصر آنرا برمی‌گرداند.

۱۱) متدى بنويسيد که يك آرایه از اعداد و يك عدد را دريافت کند، سپس با جستجو در آرایه، تعداد دفعاتي را که آن عدد در آرایه تکرار شده است را برگرداند.

۱۲) برنامه اي بنويسيد که يك آرایه دو بعدی از اعداد دريافت کند و يك آرایه يك بعدی برگرداند به گونه اي که هر عنصر آرایه تولید شده برابر با مجموع عناصر هر ستون آرایه اوليه باشد.

۱۳) متدى بنويسيد که يك عدد `int` به عنوان پaramتر دريافت می کند در صوريکه باقيمانده آن بر ۵ عدد صفر باشد، رشتة "zero" ، "one" باشد رشتة "two" باشد رشتة "three" باشد رشتة "four" باشد رشتة را به عنوان خروجي برگرداند.

۱۴) برنامه اي بنويسيد که در يك حلقه `while` ، به صورت مداوم يك عدد را از کاربر دريافت کند و آنرا در خروجي استاندارد چاپ کند در صوريکه عدد زوج باشد، حلقه ادامه يابد و در صوريکه عدد وارد شده فرد باشد، اجرای حلقة خاتمه يابد.

۱۵) متدى بنويسيد که دو آرایه از اعداد که تعداد عناصر آنها يکسان است را به صورت پaramتر دريافت کند، سپس عناصر با مقاييسه عناصر آرایه، آرایه جديدي توليد کند که مقدار هر عنصر آن، عنصر بزرگتر در هر يك از دو آرایه باشد.

۱۶) متدى بنويسيد که يك آرایه از اعداد و يك عدد را دريافت کند، سپس با جستجو در آرایه، در صوريکه آن عدد در آرایه وجود داشته باشد `true` و در غيرايصوريت `false` برگرداند.

۱۷) متدى بنويسيد که دو آرایه از اعداد را دريافت کند سپس حاصلضرب دو آرایه را برگرداند.

۱۸) متدى بنويسيد که دو آرایه از رشتة دريافت کند و آرایه اى توليد کند که هر عنصر آن مجموع عناصر نظير آن در هر دو آرایه باشد.

۱۹) متدى بنويسيد که دو آرایه از اعداد دريافت کند و با مقاييسه آنها، آرایه جديدي توليد کند در صوريکه عناصر نظير يكديگر در آرایه با يكديگر مساوي باشند، ۰ در صوريکه اولي بزرگتر از دومي باشد ۱، و اگر دومي بزرگتر از اولي باشد ۱- در آرایه قرار دهد.

مناهیم پایه در برنامه نویسی جاوا

۲۰) در کلاس زیر مکانهایی را که خطای کامپایل دارند پیدا کنید:

```
public class AccessScopeandCastExercise {  
  
    int j = 0;  
    int i = 2;  
  
    public int getValue(int j) {  
        int j = 12;  
        return j;  
    }  
  
    public void printValue() {  
        int j = 12;  
        System.out.println(j);  
    }  
  
    public void calculate(int j) {  
        if (j == i) {  
            int i = 12;  
            System.out.println(i + j);  
        }  
    }  
  
    public int convert(char c) {  
        return c;  
    }  
  
    public float convert(double d) {  
        return d;  
    }  
  
    public double convert(float d) {  
        return d;  
    }  
  
    public int convert(short s) {  
        return s;  
    }  
  
    public int convert(long l) {  
        return l;  
    }  
}
```

۲۱) در زیر داده های مختلفی تعریف شده اند، داده هایی را که به یکدیگر قابل تبدیل هستند را به یکدیگر تبدیل کنید:

```
int i = 4;  
long j = 3 ;  
float k = 23.0f ;  
double d = 12.12 ;  
char ch = 'c';  
boolean b = false ;
```



آشنایی با مفاهیم شی گرایی

در فصل دوم با تعریف و پیاده سازی کلاس و متدها آشنا شدیم. استفاده از کلاس یک روش صحیح و کارآمد برای تقسیم برنامه به اجزای منطقی کوچکتر است که بالطبع پیاده سازی، تست و نگهداری برنامه را آسان می کند. اگرچه آنچه که تاکنون در مورد نقش، عملکرد و کاربرد کلاس آموخته اید صحیح است، اما کلاس پایه و اساس برنامه نویسی شی گرایی است که موضوع این فصل است. در این فصل با برخی مفاهیم مقدماتی کلاس و شی گرایی آشنا می شویم و در فصل بعد به صورت مفصل مبحث شی گرایی را ادامه خواهیم داد.

استفاده از کلاس برای تعریف یک داده جدید

تاکنون از کلاس به عنوان محلی برای تعریف متدها استفاده کردیم، اما یکی از کاربردهای اصلی کلاس، تعریف داده های جدید است. در فصل پیش، با داده های اولیه (`int`, `float`, `double` و...) آشنا شدیم. علت نامگذاری این داده ها به «داده های اولیه» این است که از ترکیب آنها می توان داده های پیچیده تر تعریف نمود. اما چگونه؟ اگر ما به داده «زمان» نیاز داشته باشیم که جزو داده های اولیه نیست چگونه می توانیم چنین داده ای را با ترکیب داده های اولیه ایجاد کنیم؟

چوب این سوال، تعریف یک کلاس و متغیرهایی درون آن کلاس است که به منزله اجزای آن داده قلمداد می شوند. کلاس `Time` را به صورت زیر ملاحظه کنید.

```

1 public class Time {
2     int hour ; //0-23
3     int minute ; //0-59
4     int second; //0-59
5 }
```

کد ۱

در این کلاس سه متغیر `hour`, `minute` و `second` تعریف شده اند که همگی از نوع عدد صحیح `int` و معرف اجزای زمان یعنی ساعت، دقیقه و ثانیه هستند. به این ترتیب با تعریف کلاس `Time` و با ترکیب سه داده اولیه، داده جدیدی تعریف کرده ایم که معرف زمان است.

حال برای تعریف متغیری از جنس زمان، مشابه تعریف متغیر از داده های اولیه عمل می کنیم.
`Time now;`

در اینجا `now` نام متغیر، و `Time` جنس متغیر را مشخص می کند.
 ممکن است بپرسید که متغیر `now` معرف چه زمانی است و چه مقداری دارد؟ جواب این سوال این است که تا اینجا مقدار `now` پوچ است یعنی هیچ مقداری ندارد اصطلاحاً گفته می شود که مقدار آن `null` است. معنی این جمله این است که متغیر `now` هنوز در حافظه ایجاد نشده است که بتوان به آن مقداری انتساب داد این اصلی ترین تفاوت بین داده های اولیه و داده های از جنس کلاس است. وقتی متغیری از نوع داده های اولیه تعریف می شود همان موقع در حافظه با یک مقدار پیش فرض مقداردهی می شود اما متغیری که از جنس کلاس است به صورت پیش فرض مقدار `null` دارد و در حافظه وجود ندارد. برای مقداردهی آن، باید با استفاده از کلمه کلیدی `new` از ماشین مجازی جاوا بخواهیم تا آنرا در حافظه ایجاد کند. به این کار «ایجاد آبجکت» گفته می شود و به صورت زیر انجام می شود.

```
now = new Time();
```

دقت کنید که برای ایجاد آبجکت بعد از کلمه کلیدی `new`، نام کلاس (`Time`) به همراه `()` در انتهای آن آورده شده است.

حال که آبجکت `now` در حافظه ایجاد شد، می توان اجزای آن یعنی ساعت، دقیقه و ثانیه را مقداردهی نمود. برای این منظور باید ابتدا نام آبجکت، یک نقطه و سپس نام آن جزء آورده شود تکه کد زیر انجام اینکار را نشان می دهد.

```

now.hour = 12 ;
now.minute = 46 ;
now.second = 9;
```

به این ترتیب آبجکت `now` که از جنس کلاس `Time` است معرف زمان ۱۲:۴۶:۹ است.

آشنایی با مفاهیم شی گرایی

برای خواندن جزییات زمان، همانند مقداردهی آن، کافی است نام متغیر، یک نقطه (.) و نام آن جزء را مشخص کنید.

```
System.out.println("Hour:" + now.hour);
System.out.println("Minute:" + now.minute);
System.out.println("Second:" + now.second);
```

اجرای کد فوق نتیجه زیر را در کنسول برنامه به همراه خواهد داشت.

Hour:12

Minute:46

Second:9

به متغیر now و تمام متغیرهایی که از جنس کلاس هستند آبجکت گفته می شود همچنین به داده های یک کلاس، Property های آن کلاس گفته می شود. در مثال فوق، hour، minute و second از جمله Property های کلاس Time هستند. در برخی از کتب فنی، به جای کلمه Property از کلمه فیلد استفاده می شود در این کتاب نیز به خاطر ساده تر بودن نگارش فارسی فیلد از این کلمه استفاده خواهیم کرد.

افزودن متدهای کلاس

کلاس Time همانند هر کلاس دیگری می تواند متدهای خود را داشته باشد که ممکن است به صورت استاتیک یا غیراستاتیک تعریف شوند. در فصلهای پیشین با متدهای استاتیک که در تعریف آنها از کلمه static استفاده می شود آشنا شدید، این متدها به فیلدهای استاتیک کلاس دسترسی دارند و بدون ایجاد آبجکت از روی یک کلاس می توان آنها را فراخوانی کرد. اما متدهای غیراستاتیک که موضوع آن ماست اهمیت و کاربرد بیشتری دارند زیرا به داده های استاتیک و غیراستاتیک کلاس دسترسی دارند و تنها با داشتن آبجکتی از روی کلاس می توان آنها را فراخوانی کرد.

با این توضیح، متدهای Time را به کلاس setTime اضافه کرده ایم که وظیفه آن مقداردهی اجزای زمان است. این متدهای سه پارامتر int دریافت می کنند و مقادیر پارامترها را به فیلدهای کلاس انتساب می دهد.

```
1 public class Time {
2     int hour ; //0-23
3     int minute ; //0-59
4     int second; //0-59
5
6     public void setTime(int h, int m, int s){
7         hour = h;
8         minute = m ;
```

```

9         second = s;
10    }
11 }
```

کد ۵-۳

از آنجاییکه متده است، برای فراخوانی آن نیازمند آبجکتی از روی کلاس Time هستیم. برای فراخوانی آن باید نام آبجکت، یک نقطه (.) و نام متده را به همراه پارامترهای آن داخل پرانتز برای مشخص کنیم (اگر یک متده فاقد پارامتر باشد نام متده به همراه () آورده می شود). کد زیر فراخوانی متده () setTime() را نشان می دهد.

```

1 public class UsingTime {
2     public static void main(String[] args) {
3
4         Time now = new Time();
5         now.setTime(12, 46, 9);
6         System.out.println("Time is:" +now.hour+ ":" +
7                         now.minute+ ":" +now.second);
8     }
9 }
```

کد ۵-۴

نتیجه اجرای کلاس فوق، مشابه قبل خواهد بود. در کلاس فوق، برای نمایش مقادیر زمان، فیلدهای آبجکت now با رشته های " " به هم چسبانده شده تا توسط System.out.println() در کنسول برنامه نمایش یابد. ممکن است به ذهن شما برسد که متده در کلاس Time پیاده سازی کنید که این رشته را تولید کند. به این ترتیب هر جایی که به نمایش یا چاپ داده های آبجکت زمان نیاز داشته باشید آن متده را فراخوانی کنید. کلاس Time که متده جدید toString() در آن پیاده سازی شده است را در زیر ملاحظه می کنید.

```

1 public class Time {
2     int hour ; //0-23
3     int minute ; //0-59
4     int second; //0-59
5
6     public void setTime(int h, int m, int s){
7         hour = h;
8         minute = m ;
9         second = s;
```

آشنایی با مفاهیم شی گرایی

```
10     }
11
12     public String toString() {
13         return hour + ":" + minute + ":" + second ;
14     }
15 }
```

کد ۵-۴

با وجود متد (`toString()`) در کلاس `Time`، کلاس `UsingTime` به صورت زیر در خواهد آمد.

```
1 public class UsingTime1 {
2     public static void main(String[] args) {
3
4         Time now = new Time();
5         now.setTime(12, 46, 9);
6         System.out.println("Time is:" + now.toString());
7     }
8 }
```

کد ۵-۵

خروجی کد فوق مشابه قبل است.
در اینجا لازم است به دو نکته اشاره کنیم. اول اینکه رشته ها را می توان با عملگر `+` به رشته های دیگر، داده های اولیه، یا هر آبجکت دیگری متصل کرد که نتیجه آن یک رشته می شود مثلا حاصل رشته `str` در زیر

```
String str = "Java "+2;
```

رشته "Java 2" خواهد بود. اما اگر یک رشته را توسط عملگر `+` به یک آبجکت متصل کنیم، حاصل رشته تولید شده چه خواهد بود؟ مثلا اگر در کلاس `TimeUsing` رشته زیر را تعریف کنیم

```
String str = "Time is:" +now;
```

چه مقداری خواهد داشت؟ حاصل آن دقیقا با رشته زیر یکسان خواهد بود.

```
String str = "Time is:" +now.toString();
```

علت آن این سوال است که وقتی آبجکتی را به یک رشته متصل می کنید، جاوا به صورت خودکار متد (`toString()`) آن آبجکت را فراخوانی می کند تا بتواند آن آبجکت را به `String` تبدیل کند و به رشته دیگر متصل کند. در واقع ما همواره با پیاده سازی متد (`toString()`) در هر کلاس، نحوه تبدیل شدن آبجکت های آن کلاس به یک رشته `String` را تعریف می کنیم. این دقیقا چیزی است که جاوا برای نمایش یا چاپ یک آبجکت به آن نیاز دارد.

توجه: متد () در هر کلاس باید همیشه به شکل `toString()`

```
public String toString() {
}
```

نوشته شود در غیراینصورت برای جاوا قابل شناسایی نخواهد بود. از یاد نماید که جاوا به حروف کوچک و بزرگ حساس است.

حال اجازه دهید کلاس `Time` را بازهم اصلاح کنیم، یکی از ایراداتی که می توان به این کلاس گرفت، این است که اجزای زمان می توانند هر مقداری بگیرند، مثلاً می توانید مقدار `hour` را برابر ۵۴ قرار دهید، این در حالیست که مقدار ساعت نمی تواند بیشتر از ۲۳ باشد.

از آنجاییکه مقادیر زمان در یک محدوده خاصی معتبر است (ساعت بین ۰ تا ۲۳ و ثانیه و دقیقه بین ۰ تا ۶۰) لازم است در متد `setTime()` (که از طریق آن، اجزای زمان مقداردهی می شوند) صحت پارامترهای ورودی بررسی شود. درصورتیکه مقداری خارج از محدوده معتبر مشخص شده باشد، باید یک مقدار پیش فرض (مثلاً ۰) به آن داده انتساب داده شود. متد `setTime()` را می توان به صورت زیر اصلاح نمود.

```
1 public class Time {
2
3     int hour ; //0-23
4     int minute ; //0-59
5     int second; //0-59
6
7     //set a new time value and perform validity checks
8     //on the data, set invalid values to zero
9     public void setTime(int h, int m, int s) {
10         hour = (h >= 0 && h < 24) ? h : 0;
11         minute = (m >= 0 && m < 60) ? m : 0;
12         second = (s >= 0 && s < 60) ? s : 0;
13     }
14
15     public String toString(){
16         return hour + ":" + minute + ":" + second ;
17     }
18 }
```

حذف دسترسی خارجی به داده های کلاس

حال که متدهای `setTime()` برای مقداردهی و صحت سنجی فیلدهای کلاس پیاده سازی شد بهتر است مانع از مقداردهی مستقیم فیلدهای کلاس شویم. در غیراینصورت، ممکن است سهوا به فیلدهای آبجکت `Time` مقادیر نامعتبر انتساب داده شود.

برای اینکه مانع دسترسی به فیلدهای کلاس شویم، کافیست کلمه کلیدی `private` را قبل از تعریف هر یک از فیلدها اضافه کنیم.

```

1 public class Time {
2     private int hour ; //0-23
3     private int minute; //0-59
4     private int second; //0-59
5
6     //set a new time value and perform validity checks
7     //on the data, set invalid values to zero
8     public void setTime(int h, int m, int s) {
9         hour = (h >= 0 && h < 24) ? h : 0;
10        minute = (m >= 0 && m < 60) ? m : 0;
11        second = (s >= 0 && s < 60) ? s : 0;
12    }
13
14    public String toString() {
15        return hour + ":" + minute + ":" + second ;
16    }
17 }
```

کد ۵-۷

وقتی یک فیلد یا متدهای صورت `private` تعریف می شود به این معناست که آن فیلد یا متدهای فقط در داخل آن کلاس قابل استفاده یا فراخوانی است بنابراین اگر یک فیلد یا متدهای `private` را از خارج کلاس مقدار دهی یا فراخوانی کنیم با خطای کامپایل مواجه خواهیم شد. مثلاً کد زیر

```
Time now = new Time();
now.hour = 5;
```

با خطای کامپایل مواجه می شود زیرا `hour` به صورت `private` تعریف شده و نمی توان آنرا بیرون کلاس `Time` مقداردهی کرد.

کلمه کلیدی `private` نقطه مقابل است و بدین معناست که اگر یک فیلد یا متند به صورت `public` تعریف شود در هر نقطه‌ای از کلاس یا بیرون کلاس قابل فراخوانی و استفاده است. در مورد کلمات `public` و `private` در فصل بعدی به صورت مفصل صحبت خواهیم کرد.

افزودن متدهای setter

حال که داده‌های کلاس `Time` همگی به صورت `private` تعریف شده‌اند، امکان دسترسی از خارج از کلاس به آنها وجود ندارد و فقط از طریق متند `setTime()` می‌توان آنها را مقداردهی کرد. اما متند `setTime()` فقط در موقعی مناسب است که بخواهیم هر سه داده کلاس `Time` (ساعت، دقیقه و ثانیه) را مقداردهی کنیم، در مواردی که بخواهیم فقط یک داده از آبجکت کلاس (مثل `hour`) را مقداردهی کنیم نمی‌توان از این متند استفاده نمود. برای مقداردهی تک تک داده‌های کلاس، می‌توان متدهایی مشابه `setTime()` به کلاس اضافه کرد. این متدها باید مشابه متند `setTime()` صحت داده‌ها را قبل از مقداردهی بررسی کنند. به این متدها که نام آنها با `set` شروع می‌شود `setter` گفته می‌شود. کلاس `Time` که در آن متدهای `setter` پیاده‌سازی شده‌اند در زیر نشان داده شده است.

```

1  public class Time {
2      private int hour ; //0-23
3      private int minute ; //0-59
4      private int second; //0-59
5
6      //set a new time value and perform validity checks
7      //on the data, set invalid values to zero
8      public void setTime(int h, int m, int s) {
9          setHour(h);
10         setMinute(m);
11         setSecond(s);
12     }
13
14     public void setHour(int h){
15         hour = (h >= 0 && h < 24) ? h : 0;
16     }
17
18     public void setMinute(int m){
19         minute = (m >= 0 && m < 60) ? m : 0;
20     }
21
22     public void setSecond(int s){
23         second = (s >= 0 && s < 60) ? s : 0;

```

```

24     }
25
26     public String toString() {
27         return hour + ":" + minute + ":" + second ;
28     }
29 }
```

کد ۱

نکته جالب در مورد کلاس فوق، این است که متدهای `setTime()` خود از متدهای `toString()` برای مقداردهی فیلدهای کلاس استفاده نموده است.

«سازنده» یا «constructor»

وقتی آبجکتی از کلاس `Time` ایجاد می کنیم، با فراخوانی متدهای `setTime()` فیلدهای زمان را مقداردهی می کنیم. این بدين معناست که ساخته شدن آبجکت در حافظه و مقداردهی داده های آبجکت در دو مرحله مجزا انجام می شود. سوالی که در اینجا مطرح می شود این است که آیا می توان مقداردهی فیلدهای زمان را همزمان با ایجاد آبجکت انجام داد؟ قبل از جواب دادن به سوال، باید بگوییم که چنین نیازی کاملا منطقی است زیرا در دنیای واقعی وقتی زمان یک ملاقات را مشخص می کنیم در واقع همزمان آنرا ایجاد و مقداردهی کرده ایم.

در جوا این امکان با فراهم کردن سازنده یا `constructor` فراهم شده است. وقتی آبجکتی از یک کلاس ساخته می شود (با بکارگیری کلمه `new`، ماشین مجازی جوا متدی را از کلاس فراخوانی می کند که به آن «سازنده» یا «constructor» گفته می شود. بنابراین اگر می خواهید در زمان ساخته شدن آبجکت، کاری نیز انجام شود (مثلاً مقداردهی اولیه داده های آبجکت)، لازم است برای کلاس، سازنده ای پیاده سازی شود که حاوی عمل موردنظر (مثلاً مقداردهی اولیه داده های آبجکت) است.

اما متد سازنده با متدهایی که تا به حال آنها را پیاده سازی کرده اید کمی فرق دارد.

- متد سازنده، همنام کلاس است
- متد سازنده، هیچ مقدار برگشتی ندارد (این متد هیچ نوع داده ای را برنمی گرداند، حتی `void`)
- متد سازنده، می تواند هر تعداد پارامتر داشته باشد یا نداشته باشد.

کلاس `Time` را که در آن یک سازنده پیاده سازی شده است به صورت زیر تصور کنید.

```

1  public class Time {
2      private int hour ; //0-23
3      private int minute ; //0-59
4      private int second; //0-59
5 }
```

```

6     Time() {
7         System.out.println("Object is being created.");
8     }
9
10    //remaining of the class
11 }
```

کد ۵-۹

در کلاس فوق، یک سازنده پیاده سازی شده است همانطور که مشخص است نام این متده همنام کلاس است، هیچ نوع برگشتی ندارد. در این مثال، سازنده بدون پارامتر پیاده سازی شده است، اما یک سازنده می تواند پارامتر هم داشته باشد. به این ترتیب وقتی با استفاده از جملة

Time now = new Time();

آجکتی از کلاس Time ساخته می شود، جمله Object is being created. در کنسول برنامه نمایش می یابد.

از سازنده کلاس می توان برای هر منظور دلخواه استفاده نمود، هر عملی که لازم است در زمان ساخته شدن آجکت انجام شود را می توان در سازنده پیاده سازی کرد.

یکی از رایج ترین کارهایی که سازنده کلاسها آن را انجام می دهند، مقداردهی فیلدهای آجکت است که اینکار را با پارامترهایی که دریافت می کنند انجام می دهند. کلاس تغییر یافته Time که مقادیر داده های کلاس را از طریق پارامترهای سازنده دریافت می کند و نحوه ایجاد آجکت از آن در زیر نشان داده شده است.

```

1 public class Time {
2     private int hour; //0-23
3     private int minute; //0-59
4     private int second; //0-59
5
6     Time(int h, int m, int s) {
7         hour = h;
8         minute = m;
9         second = s;
10    }
11
12    //set a new time value and perform validity checks
13    //on the data, set invalid values to zero
14    public void setTime(int h, int m, int s) {
15        setHour(h);
16        setMinute(m);
17        setSecond(s);
```

آشنایی با مفاهیم شی گرایی

```
18     }
19
20     public void setHour(int h){
21         hour = (h >= 0 && h < 24) ? h : 0;
22     }
23
24     public void setMinute(int m){
25         minute = (m >= 0 && m < 60) ? m : 0;
26     }
27
28     public void setSecond(int s){
29         second = (s >= 0 && s < 60) ? s : 0;
30     }
31
32     public String toString(){
33         return hour + ":" + minute + ":" + second ;
34     }
35 }
```

کد ۱۰

```
1 import javax.swing.*;
2
3 public class UsingTime2 {
4     public static void main(String[] args){
5
6         Time now = new Time(12, 46, 9);
7         JOptionPane.showMessageDialog(null,
8             "Time is:" +now);
9     }
10 }
```

کد ۱۱

همانطور که ملاحظه می کنید در کلاس Time یک سازنده جدید پیاده سازی شده است که سه پارامتر int (که در واقع اجزای زمان هستند) را دریافت می کند و مقادیر آنها را به فیلدهای کلاس انتساب می دهد. همانطور که در کلاس TimeUsing نشان داده شده است، ایجاد آبجکت Time با جمله new Time(12, 46, 9) انجام شده است که مقادیر 12، 46 و 9 به عنوان پارامترهای سازنده کلاس Time مشخص می کند

```
Time(int h, int m, int s)
```

بدین ترتیب اجزای زمانی این آبجکت با مقادیر ۱۲، ۶ و ۰ مقداردهی می شوند.
با اضافه شدن سازنده به کلاس Time، مناسب است تا کدهای کلاس را تمیز کنیم.
با دقت در عملکرد سازنده و متدهای setTime() متوجه می شوید که هر دو متدهای عمل یکسانی انجام می
دهند (هر دو متدهای setTime() می کنند) بنابراین سازنده کلاس می تواند متدهای
setTime() را به صورتی که در زیر مشاهده می کنید فراخوانی کند.

```

1  public class Time {
2      private int hour ; //0-23
3      private int minute ;//0-59
4      private int second; //0-59
5
6      Time(int h, int m, int s){
7          setTime(h, m, s);
8      }
9
10     //set a new time value and perform validity checks
11     //on the data, set invalid values to zero
12     public void setTime(int h, int m, int s) {
13         setHour(h);
14         setMinute(m);
15         setSecond(s);
16     }
17
18     public void setHour(int h){
19         hour = (h >= 0 && h < 24) ? h : 0;
20     }
21
22     public void setMinute(int m){
23         minute = (m >= 0 && m < 60) ? m : 0;
24     }
25
26     public void setSecond(int s){
27         second = (s >= 0 && s < 60) ? s : 0;
28     }
29
30     public String toString(){
31         return hour + ":" + minute + ":" + second ;

```

```
32 }  
33 }
```

کد ۵-۱۲

توجه: همواره در کدنویسی می بایست از پیاده سازی کدهای تکراری خودداری کنید فراخوانی متدهای `setTime` در سازنده کلاس `Time` دقیقا با همین هدف انجام شده است. رعایت این موضوع باعث می شود، تغییرات احتمالی برنامه در آینده به حداقل برسد مثلا اگر در آینده منطق مقداردهی `setter` کلاس تغییر کند، تنها با تغییر متدهای `setter` می توان شیوه مقداردهی را به سازنده کلاس نیز سرایت داد.

تعريف یک داده دیگر

در قسمت قبل با استفاده از داده های اولیه، داده جدید `Time` را تعریف کردیم که معرف زمان بود. در اینجا بحث خود را با تعریف یک کلاس دیگر، که معرف «کسر اعداد» است ادامه می دهیم. همانطور که می دانیم در بین داده های اولیه، داده ای به نام «داده کسری» وجود ندارد که با استفاده از آن بتوان متغیرهای کسر اعداد تعریف نمود. برای داشتن چنین داده جدیدی، همانند حالت قبل کلاس جدیدی تعریف می کنیم که معرف کسر اعداد باشد.

با دقت در کسر اعداد متوجه می شویم که هر کسر از یک صورت و یک مخرج که هر دو عدد صحیح هستند تشکیل شده است. بنابراین کلاس `Rational` را با دو فیلد عددی به صورت زیر پیاده سازی می کنیم.

```
1 public class Rational{  
2     int top ;  
3     int bottom ;  
4 }
```

کد ۵-۱۳

همانند کلاس `Time` می توان برای این کلاس نیز متدهای سازنده و `toString()` را به صورت زیر پیاده سازی نمود.

```
1 public class Rational{  
2     int top ;  
3     int bottom ;  
4  
5     public Rational(int t, int b) {
```

```

6      top = t;
7      bottom = b;
8  }
9
10     public String toString(){
11         return top + " / " + bottom;
12     }
13 }
```

کد ۱۴

با پیاده سازی کلاس Rational، آبجکتهای این کلاس به صورت زیر ایجاد می شوند.

```
Rational rational = new Rational(2, 3);
```

این آبجکت معرف کسر ۲/۳ است.

یکی از عملیاتی که روی کسر اعداد انجام می شود جمع کردن آنهاست. مثلاً می توان مقدار یک کسر را با یک عدد جمع یا تفريط کرد. به عنوان مثال فرض کنید بخواهیم مقدار آبجکت فوق را که معرف کسد ۲/۳ است را با کسر دیگری که مقدار آن ۲/۳ است جمع کنیم از آنجائیکه این دو کسر مخرج یکسانی دارند نیازی به مخرج مشترک نیست و با جمع صورت آنها می توان به حاصل جمع دو کسر رسید.

```
rational.top = rational.top + 2;
```

اگر مقدار کسری که می خواهیم با آبجکت rational جمع کنیم $\frac{t}{b}$ باشد در اینصورت ناگزیر هستیم مخرج مشترک بگیریم و حاصل جمع دو کسر را از طریق فرمول زیر محاسبه کنیم.

$$\frac{\text{top}}{\text{bottom}} + \frac{t}{b} = \frac{\text{top} * b + t * \text{bottom}}{b * \text{bottom}}$$

معنی این جمله این است که باید برای جمع دو کسر کدهایی به صورت زیر پیاده سازی کنیم.

```
rational.top= rational.top*b + a* rational.bottom ;
```

```
rational.bottom = b* rational.bottom ;
```

مثلاً برای جمع آبجکت کسر که در بالا ساخته شد با کسر کدهای زیر نوشته شوند.

```
rational.top= rational.top*5 + 2* rational.bottom ;
```

```
rational.bottom = 5* rational.bottom ;
```

اگرچه جمع دو کسر به شکلی که در بالا انجام شد نتیجه مورد نظر ما را برآورده می کند، اما دارای یک اشکال اساسی است، و آن اینکه در هر جایی از برنامه که نیاز به حاصل جمع دو کسر داشته باشیم می بایست

آشنایی با مفاهیم شی گرایی

کدهایی تکراری مشابه کدهای فوق پیاده سازی کنیم. راه حل این مسئله افزودن متدهای add() به کلاس Rational است تا برای جمع یک کسر با کسر دیگر آنرا فراخوانی کنیم.

```
1 public class Rational{  
2  
3     int top ;  
4     int bottom ;  
5  
6     public Rational(int t, int b){  
7         top = t;  
8         bottom = b;  
9     }  
10  
11    void add(int a, int b){  
12        top=top*b + a*bottom ;  
13        bottom = b*bottom ;  
14    }  
15 }
```

کد ۱۵

متدهای add() دارای دو پارامتر t و b از جنس int است که معرف صورت و مخرج کسر دیگری هستند که قرار است با کسر فعلی جمع شود. به این ترتیب برای جمع کسر k با کسر دیگری که مقدار آن $\frac{2}{3}$ باید به صورت زیر عمل نمود.

```
Rational k =new Rational(3, 7);  
k.add(2,3);
```

با کمی دقت روی متدهای add() متوجه می شوید که پارامترهای متدهای add() اگرچه دو عدد int هستند اما در واقع داده های یک کسر دیگر هستند. بنابراین چرا پارامتر متدهای add() خود آبجکتی از Rational نباشد؟! از نظر طراحی شی گرا بهتر است به جای استفاده از دو پارامتر عددی که به منزله صورت و مخرج کسر دوم هستند یک آبجکت از کلاس کسر را به عنوان پارامتر برای متدهای add() تعریف کنیم.

```
1 public class Rational{  
2  
3     int top ;
```

```

4     int bottom ;
5
6     public Rational(int t, int b){
7         top = t;
8         bottom = b;
9     }
10
11    void add(Rational k){
12        top=top*k.bottom + k.top*bottom ;
13        bottom = k.bottom*bottom ;
14    }
15
16 }
```

کد ۵-۱۶

در متدهای add() و k در متد add()، به جای دو پارامتر عددی، یک کسر k به عنوان پارامتر مشخص گردیده است داخل متدهای صورت و مخرج کسر دوم دریافت شده و حاصل جمع دو کسر محاسبه گردیده است. حال برای اینکه یک کسر را با کسر دیگری جمع کنید باید کدی مشابه آنچه در زیر آمده است داشته باشید.

```
//create first Rational object
Rational k1 =new Rational(3, 7);
```

```
//create second Rational object
Rational k =new Rational(2, 3);

k1.add(k2);
```

تمرین: متدهای ضرب (multiply)، تفکیق (divide) و تفریق (subtract) را به کلاس Rational اضافه کرده و آنها را در کلاس MainClass فراخوانی کنید.

متدهای toString()

در ابتدای این فصل متدهای toString() را در کلاس Time پیاده سازی کردیم. همانطور که گفته شد این متدهای String برای گرداندن و کاربرد آن زمانی است که می خواهیم آبجکتها را نمایش دهیم. طبیعی است که نمایش آبجکتها از طریق یک رشته امکانپذیر است. حال احاجاه دهید یک آبجکت Rational را که در حال حاضر قادر به ایجاد متد toString() است در کنسول برنامه نمایش دهیم تا ببینیم خروجی آن چیست!

```
Rational k = new Rational(2, 3);
```

آشنایی با مفاهیم شی گرایی

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

نتیجه کد فوق، نمایش جمله عجیب و غریب زیر در کنسول برنامه است.

```
Rational@1507383
```

ترکیب نام کلاس، علامت @ و یک عدد (1507383) نمایش یافته است. طبیعی است که ما انتظار داشته باشیم که آبجکت Rational فوق که معرف عدد $\frac{2}{3}$ است دقیقاً به صورت $\frac{2}{3}$ نمایش یابد. برای این منظور متدهای پیاده سازی می‌کنیم تا به خواسته خود برسیم.

توجه: همانطور که در فصل ارث بری خواهید دید در تمام کلاسهای جاوا متدهای `toString()` را به پیاده سازی پیش فرض زیر وجود دارد.

```
public String toString() {  
    return getClass().getName() + "@" +  
    Integer.toHexString(hashCode());  
}
```

خروجی پیش فرض متدهای فوق، "نام کلاس@یک عدد" یعنی همان چیزی که در بالا مشاهده شد است. بنابراین برای نمایش آبجکت Rational لازم است متدهای `toString()` را به شکل مناسب در کلاس Rational پیاده سازی کنیم.

```
public String toString() {  
    return top + "/" + bottom;  
}
```

متدهای `equals()`

فرض کنید در برنامه دو آبجکت `k1` و `k2` به صورت زیر تعریف و ایجاد شده باشند.

```
Rational k1 = new Rational(2, 3);
```

```
Rational k2 = new Rational(2, 3);
```

اگرچه هر دو آبجکت معرف کسر $\frac{2}{3}$ هستند ولی مقایسه آنها با استفاده از عملگر `==` نشان می‌دهد که از نظر جاواین دو آبجکت با هم برابر نیستند!

```
if (k1 == k2) {  
    System.out.println("equals");
```

```

} else {
    System.out.println("not equals");
}

```

نتیجه اجرای کد فوق، نمایش "not equals" در کنسول برنامه خواهد بود.

مسئله از اینجا ناشی می شود که از نظر جاوا، محتوای آبجکتها اهمیتی ندارد بلکه موقعیت آبجکتها در حافظه مهم است. از آنجاییکه `k1` و `k2` دو آبجکت جداگانه هستند که در محل های متفاوتی از حافظه قرار دارند، از نظر جاوا متفاوت هستند. اما دو آبجکت ممکن است تحت شرایطی به یک محل از حافظه اشاره کنند مثلا در کد زیر دو آبجکت `k1` و `k2` هرچند نامهای متفاوتی دارند اما دو آبجکت جداگانه نیستند.

```

Rational k1 = new Rational(2, 3);
Rational k2 = k1;

```

این دو آبجکت به به یک محل از حافظه اشاره می کنند و هر تغییری روی مقادیر `k1` روی مقادیر `k2` نیز تاثیر خواهد داشت و بالعکس. در این حالت، دو آبجکت `k1` و `k2` مساوی هستند زیرا هر دو آبجکت به یک موقعیت از حافظه اشاره می کنند.

در برنامه ها موارد زیادی وجود دارند که می خواهیم دو آبجکت را براساس محتوا (به جای موقعیت آنها در حافظه) مقایسه کنیم. در اینصورت راهکار آن استفاده از متدهای `equals()` به جای عملگر `==` است.

```

if(k1.equals(k2)) {
    System.out.println("equals");
} else {
    System.out.println("not equals");
}

```

همانطور که ملاحظه می کنید متدهای `equals()` از یکی از آبجکتها فراخوانی شده و آبجکت دیگر به عنوان پارامتر آن مشخص شده است. در واقع شکل متدهای `equals()` که باید پیاده سازی شود به صورت زیر است.

```

public boolean equals(Object obj) {
    ...
}

```

این متدهای آبجکتی را به صورت پارامتر دریافت می کند، آنرا با آبجکت فعلی (یعنی همان آبجکتی که متدهای `equals()` از آن فراخوانی شده است) مقایسه می کند و مقدار `boolean` بر می گرداند. مقدار `true` به معنی مساوی بودن دو آبجکت، و مقدار `false` به معنی نامساوی بودن دو آبجکت خواهد بود.

همانطور که دقت کرده اید، پارامتر این متدهای آبجکتی از کلاس `Object` است. کلاس `Object` یکی از کلاسهای اصلی جاواست. این کلاس، یک کلاس عمومی است و هر داده غیر اولیه ای (مثلا آبجکتها) از

آشنایی با مفاهیم شی گرایی

کلاس Rational، Time، ... آبجکتی از کلاس Object محسوب می شوند. در فصل های آینده با کلاس Object بیشتر آشنا می شوید.
اگر بخواهیم متدهای Rational را برای کلاس Rational پیاده سازی کنیم باید این متدها را به صورت زیر بنویسیم.

```
public boolean equals(Object obj) {  
    if(obj instanceof Rational){  
        Rational k = (Rational) obj;  
        if(k.top*bottom - k.bottom*top == 0){  
            return true;  
        } else{  
            return false ;  
        }  
    } else{  
        return false;  
    }  
}
```

در این متدها از عملگر instanceof استفاده کرده ایم تا بررسی کنیم که آیا آبجکت `j` از جنس کلاس Rational است یا خیر. اگر `j` از جنس کلاسی به جز کلاس Rational باشد مقایسه آن با یک آبجکت Rational بی معناست و در این حالت این متدها برمی گردانند. اما اگر `j` از جنس کلاس Rational باشد ابتدا آنرا با استفاده از جمله

```
Rational k = (Rational) obj;
```

به یک آبجکت Rational با نام `k` تبدیل کرده ایم. از آنجاییکه `j` از جنس کلاس Object است نمی توان به داده های کسر که در این آبجکت قرار دارند دسترسی پیدا کرد. جمله فوق به ما امکان می دهد تا آبجکتی از جنس Rational در اختیار داشته باشیم که در نتیجه بتوان به داده های کسر و متدهای آن دسترسی داشت. با داشتن آبجکت Rational می توان دو آبجکت را با یکدیگر مقایسه کرد، تنها وقتی دو آبجکت با یکدیگر مساوی هستند که طرفین وسطین دو آبجکت با یکدیگر مساوی باشد. حاصلضرب صورت یک کسر در مخرج کسر دوم برابر با حاصلضرب صورت کسر دوم در مخرج کسر اول باشد. (کسرهای $\frac{3}{2}$ و $\frac{6}{4}$ با یکدیگر مساوی هستند زیرا $2 \times 3 = 4 \times 6$ است).

حال برای مقایسه دو آبجکت کسر به جای عملگر `==`، از متدهای `equals()` به صورت زیر استفاده می کنیم:

```
Rational k1 = new Rational(2, 3);
```

```
Rational k2 = new Rational(4, 6);
```

```
if(k1.equals(k2)) {
    System.out.println("equals");
} else{
    System.out.println("not equals");
}
```

در این صورت پیغام زیر را در کنسول برنامه ملاحظه خواهید کرد.

equals

این بدين معناست که مقایسه به درستی انجام گرفته است.

در مورد عملگر == و متدهای equals() چندین نکته زیر قابل توجه است:

برای مقایسه داده های اولیه (از قبیل int, long, float, double, byte, boolean, char) می توانید از عملگر == استفاده کنید چون اساساً داده های اولیه بر اساس محتوا با یکدیگر مقایسه می شوند.

۱. برای مقایسه هر داده ای که از جنس کلاس باشد از متدهای equals() استفاده کنید. از آنجاییکه

یک کلاس است، برای مقایسه آبجکتهاي String می توانید از یکی از متدهای

equals() یا equalsIgnoreCase() استفاده کنید. متدهای equals()

کاراکترهای دو رشته را (هم از لحاظ تعداد، ترتیب و هم از لحاظ کوچک و بزرگ بودن حروف) با

یکدیگر مقایسه می کند متدهای equalsIgnoreCase() متشابه است و عمل می کند با

این تفاوت که در مقایسه کردن دو رشته، کوچک و بزرگ بودن آنها را با یکدیگر لحاظ نمی کند.

(در مورد این دو متدهای String در فصل «رشته ها و کاراکترها» صحبت خواهیم

کرد).

خلاصه

در این فصل با تعریف و استفاده از متدهای static آشنا شدید تمام متدهای static را می‌توان با استفاده از نام کلاس، یک نقطه(.) و سپس نام متدهای پارامترهای آن متدهای فراخوانی کرد (به متدهای static گفته می‌شود که قبل از مشخص کردن نوع مقدار برگشتی متدهای static از کلمه کلیدی static استفاده شده باشد). کلاس Math که در پکیج java.lang قرار دارد حاوی بسیاری از متدهای مفید برای انجام محاسبات ریاضی و هندسی است.

یکی از کاربردهای کلاس، تعریف یک داده جدید است در این فصل کلاس Time که معرف داده جدید زمان است، تعریف شد این داده دارای سه جزء ساعت، دقیقه و ثانیه است که به صورت سه متغیر int داخل کلاس Time تعریف شده اند به این اجزا که داخل کلاس تعریف می‌شوند فیلدهای کلاس گفته می‌شود. فیلدها و متدهای کلاس می‌توانند به صورت public یا private تعریف شوند در صورتیکه به صورت private تعریف شوند از خارج کلاس قابل استفاده یا فراخوانی نیستند و در صورتیکه public تعریف شوند هم در داخل و هم در خارج از کلاس قابل استفاده و فراخوانی خواهند بود.

برای تغییر مقدار فیلدهای private می‌توان متدهایی در کلاس تعریف کرد که مقدار جدیدی به فیلد انستاب می‌دهد به این متدها که با نام set شروع می‌شوند گفته می‌شود.

برای مقدار دهی کردن فیلدهای آبجکت (یا انجام هرگونه عملیات دیگری) در حین ساخته شدن آبجکت، می‌توان متدهای constructor یا «سازنده» یا «آبجکت» می‌شود در هنگامی که با استفاده از کلمه new از روی یک کلاس آبجکت ساخته می‌شود سازنده آن کلاس فراخوانی می‌شود و می‌توان از طریق پارامترهایی که به سازنده ارسال می‌شوند، اجزای آن آبجکت را مقداردهی کرد.

با پیاده سازی متدهای toString() در یک کلاس می‌توان نحوه نمایش آبجکتها را از آن کلاس را مشخص نمود.

برای مقایسه دو آبجکت نباید از عملگر == استفاده کنید، این عملگر برای مقایسه دو آبجکت، ارجاع آنها را با یکدیگر مقایسه می‌کند، به جای این عملگر می‌توانید متدهای equals() را در دو کلاس پیاده سازی کنید.

تمرینات

(۱) متدهای بنویسید که یک آرایه از Object دریافت کند و عناصری از آرایه را که معرف عدد هستند

(از قبیل Integer, Long, Byte) را از بقیه جدا کند و در قالب یک آرایه از Object برگرداند.

(۲) کدی به صورت زیر نوشته شده است:

```
Book book = new Book();
Book book1 = new Book("Java Programming", "James Gosling",
                     "Prentice Hall", 1999);
Book book2 = new Book("Java Programming");
```

کلاس Book را پیاده سازی کنید به صورتی که کد فوق کامپایل شود!

(۳) متدهای بنویسید که یک String دریافت کند، و تمام کاراکترهایی که بعد از "کاراکترهای حروف بزرگ" قرار دارند را به یکدیگر بچسباند و در قالب یک رشته جدید برگرداند مثلاً

"t4YiouRrlsUaiqQn" → "iran"

(۴) متدهای بنویسید که دو رشته را دریافت کند و به صورت تصادفی دو رشته را با یکدیگر ترکیب نماید (راهنمایی، از متدهای Math.random() برای تولید یک عدد تصادفی ".+" یا ".") استفاده کنید حال به تعداد کل کاراکترهای دو رشته، عدد تصادفی ۰ یا ۱ تولید کنید و در هر بار تصمیم بگیرید که کاراکتری را از هریک از دو رشته انتخاب کنید)

(۵) فرض کنید قرار است یک برنامه انبارداری پیاده سازی کنید که در آن کاربر اپراتور می‌تواند کالا تعريف کنید (اطلاعات یک کالا را ثبت کند)، تولید کننده کالا را تعریف کند (اطلاعات یک تولید کننده کالا را ثبت کند)، محصولی را برای ورود به انبار ثبت کند، یا محصولی را برای خروج از انبار ثبت کند. چه کلاسهایی در این پروژه باید تعریف شوند حتی الامکان سعی کنید با دانشی که از شی گرایی دارید این کلاسها را طراحی و پیاده سازی کنید.

(۶) کلاس Tас را پیاده سازی کنید این کلاس باید Tас (بازی تخته نرد) را مدل کند. (راهنمایی: قبل در کلاس متدهای throwDice() را پیاده سازی کرده ایم که پرتاب Tас را مدل می‌کرد، اینکه با دانشی که از شی گرایی دارید کلاسی بنویسید که خود Tас را مدل کند)

(۷) متدهای بنویسید که یک آرایه از String دریافت کند و عناصر تکراری را حذف کند مثلاً ["Iran", "Tehran", "Tehran", "Iran", "Java"] → ["Iran", "Tehran", "Java"]

(۸) کلاسی بنویسید که دارای متدهای زیر باشد

- متدهای که دو آرایه از اعداد int را دریافت کند و مشترکات آنها را برگرداند
- متدهای که ب.م. دو عدد را محاسبه کند و برگرداند

آشنایی با مفاهیم شی گرایی

• متدهای که ک.م.دو عدد را برگردانند

۹) فرض کنید می خواهید برنامه ای برای شهرداری تهران بنویسید که موقعیت فیزیکی هر ساختمان Building را ذخیره می کند. کلاس کلاس که معرف یک ساختمان است را پیاده سازی کنید.

(راهنمایی: فرض کنید همه ساختمان ها شمالی یا جنوبی هستند و از هر طرف به یک ساختمان دیگر یا خیابان منتهای می شوند)

۱۰) کلاسهایی که معرف اشیای یک موسسه آموزشی هستند (همراه با جزئیات آنها) را ایجاد نمایید.

۱۱) فرض کنید که در تمرین اول، کلاسی به نام Course (یک دوره آموزشی) تعریف شده است که یک فیلد number (تعداد دانشجویان دوره) دارد. متدهای بنویسید که دو آبجکت Course

دریافت کند و بر اساس اینکه تعداد دانشجویان کدامیک بیشتر باشد یکی از اعداد ۱ (به معنی اولی بیشتر)، -۱ (به معنی دومی بیشتر) یا صفر (به معنی هر دو مساوی) را برگرداند.

۱۲) متدهای با نام merge پیاده سازی کنید که دو آرایه از آبجکت دریافت کند و عنصری غیرمشترک دو آرایه را با یکدیگر ترکیب کند و آرایه جدیدی تولید کرده و آنرا برگرداند.

۱۳) متدهای بنویسید که یک آرایه String دریافت کند و کاراکتری که بیش از بقیه در تمام آرایه تکرار شده است را برگرداند.

۱۴) برنامه ای بنویسید که بازی منج را مدل کند در این برنامه:

a. دو بازیکن، سه بازیکن، یا چهار بازیکن با یکدیگر بازی می کنند

b. هر بازیکن چهار مهره در اختیار دارد

c. هر بازیکن تاس را پرتاب می کند تا شش بیاید و شروع کند

d. مهره های هر بازیکن ممکن است توسط مهره های بازیکن دیگر خورده شود

e. هر بازیکنی که زودتر از بقیه بتواند مهره های خود را در سبد خود قرار دهد برنده است

۱۵) کلاسی بنویسید که آسانسور را در یک ساختمان مدل کند. توضیح: هر آسانسور در ساختمان در یک محدوده طبقات کار می کند (مثالاً از ۱ تا ۵ برای یک ساختمان که یک پارکینگ و پنج طبقه دارد). فردی که ممکن است در یک طبقه ساختمان (مثالاً طبقه ۴) باشد از آسانسور درخواست

جابجایی می کند وقتی آسانسور مقابل وی قرار گرفت وی وارد آسانسور می شود و طبقه ای را

انتخاب می کند آسانسور به آن طبقه می رود. در آسانسور قوانین زیر وجود دارد:

a. فرد نمی تواند خارج از محدوده آسانسور را درخواست کند.

b. هر طبقه ای که زودتر درخواست دهد زودتر هم باید سرویس داده شود مگر اینکه در

حین سرویس دادن از طبقه ای عبور کند که فرد دیگر درخواست دهنده در آن قرار دارد

که باید با باز شدن در، به فرد جدید هم سرویس دهد.

۱۶) کلاسهای مورد نیاز برای مدل سازی یک شهر را پیاده سازی کنید. در این مدل، بزرگراهها، خیابانها، کوچه ها، ساختمانها، بارکها، بیمارستانها، ادارات دولتی، آتش نشانی، مدارس، مغازه ها و

- فروشگاهها، مدل می شوند. توضیح، فرض کنید که می خواهید یک نقشه شهر را مدل کنید باید کلاس هایی را تعریف کنید که برای ایجاد یک نقشه به آن نیاز دارید
- ۱۷) کلاس Customer را که معرف یک مشتری در یک بانک است را پیاده سازی کنید. توجه کنید که هر مشتری علاوه بر مشخصات فردی (شامل نام، نام خانوادگی، تاریخ تولد، تاریخ عضویت، شماره ملی، و آدرس) حداقل یک حساب دارد (ممکن است بیش از یک حساب هم داشته باشد). سپس متدهای بنویسید که آرایه ای از Customer و یک آبجکت Date دریافت کند و تمام مشتریانی که در آن آرایه تاریخ عضویت آنها قبل از آن تاریخ باشد را در قالب یک آرایه جدید برگرداند.
- ۱۸) متدهای بنویسید که دو آبجکت Rational را با یکدیگر مقایسه کند، اگر اولی بزرگتر باشد ۱+ و اگر مساوی باشند ۰ و در غیراینصورت وقتی دومی بزرگتر باشد ۱- برگرداند.

۴

شی گرایی

در فصل پیش با چگونگی پیاده سازی کلاس برای تعریف داده جدید و ایجاد آجکت از روی کلاس آشنا شدید. همچنین تعریف متدها، فیلد و سازنده و نحوه فراخوانی و دسترسی به آنها را آموختید. در این فصل با مفاهیم بیشتری در رابطه با کلاس آشنا می شوید.

شی گرایی چیست؟

از زمانیکه اولین کامپیوترها ساخته شدند زمان زیادی می گذرد در آن روزها کامپیوترها کارهای ساده ای انجام می دادند مثلا می توانستند محاسبات عددی را انجام دهد، داده های آماری را پردازش، یا اطلاعات آب و هوایی را تحلیل کنند. در تمام این موارد، مجموعه ای از داده ها وجود داشتند و کامپیوتر می بایست با انجام محاسبات ساده یا پیچیده یک خروجی را تولید می کرد. نوشتن اینگونه برنامه ها کار پیچیده ای نبود زیرا برنامه نویس می بایست الگوریتم محاسباتی یا تحلیلی خود را در قالب صدها یا هزاران خط پیاده سازی کند. برنامه ای که تولید می شد خط به خط از ابتدتا انتهای به ترتیب اجرا می شد تا الگوریتم محاسباتی اجرا شود و نتیجه ای بدست آید. البته برنامه نویس ها با پیاده سازی توابع، کدهای خود را سازماندهی و تمیز می کردند.

با گذشت زمان، و با ورود کامپیوترها به زندگی روزمره انسانها، نرم افزارها نیز به مرور پیچیده تر شدند در واقع انتظارات از نرم افزارها افزایش یافت تا جایی که انتظار می رفت تحصیل، تجارت، کسب و کار، بهداشت، بانک، و حتی زندگی خصوصی و روزمره انسانها نرم افزاری شود!

نسل جدید برنامه ها، بسیار پیچیده تر از نسل اولیه نرم افزارها بودند. نسل جدید برخلاف نسل قدیم که متمرکز بر عملیات محاسباتی یا کنترلی بود، بر شبیه سازی دنیای واقعی متمرکز بودند. به عنوان نمونه، نرم افزار مدیریت فروشگاه که توسط پرسنل یک فروشگاه استفاده می شود را تصور کنید. این نرم افزار بخش عده ای از کار پرسنل را انجام می دهد و باعث تسهیل و تسريع کار فروشگاه می شود آنچه در این برنامه انجام می شود در واقع شبیه سازی آن چیزی است که قبلاً به صورت دستی توسعه صندوقدار، فروشنده، انباردار، مسئول خرید، مدیر فروشگاه،... صورت می گرفته است.

به عنوان مثالی دیگر، یک برنامه جامع حسابداری را تصور کنید کار این برنامه این است که محاسباتی را که قبلاً به صورت دستی صورت می گرفته را بصورت نرم افزاری انجام دهد، گزارشاتی که قبلاً به صورت دستی تولید می شده را به صورت خودکار تولید کند، و مستنداتی که قبلاً به صورت کاغذی ثبت و در بایگانی نگهداری می شد را به صورت الکترونیکی ثبت و نگهداری کند. همه اینها، به معنی شبیه سازی آن چیزی است که قبلاً به صورت دستی انجام می شده و اینکه به صورت خودکار انجام می شود و بدین ترتیب باعث تسهیل و تسريع عملیات می شود.

با مقایسه دو مثال فوق با برنامه های اولیه که تولید می شدند به وضوح مشخص است که این نرم افزارها ماهیت متفاوتی دارند. در حالیکه نرم افزارهای اولیه روی انجام یک الگوریتم یا منطق کاری تمرکز می کردند، نرم افزارهای جدید روی شبیه سازی (مدل سازی) دنیای واقعی و تعامل بین موجودات دنیای واقعی تمرکز دارند. وجود همین تفاوت باعث شده روش جدیدی برای طراحی و تولید نرم افزارهای جدید ابداع شود که به آن «شی گرایی» (معادل فارسی عبارت Object-Oriented) گفته می شود و مبنای آن موجوداتی هستند که در تعامل با یکدیگر منطق برنامه را شکل می دهند.

براساس آنچه شی گرایی پیشنهاد می کند، برای تسهیل و ساده کردن فرایند تولید نرم افزار، لازم است محیط واقعی و موجوداتی که در محیط واقعی در تعامل با یکدیگر هستند را شبیه سازی کنیم. برای درک موضوع به یک مثال توجه کنید. اگر بخواهید برنامه ای برای یک بانک بنویسید باید ببینید در یک بانک چه اشیایی وجود دارند، هریک از این اشیاء چه کارهایی انجام می دهند، و هر شی چه رابطه ای با شی دیگر دارد. نتیجه این تحقیق شناسایی اشیایی از قبیل «مشتری»، «بانک»، «حساب»، «حسابدار»، «صندوقدار»، «پرانتور» ... و آعمالی به نام «افتتاح حساب»، «افزایش موجودی»، «انتقال پول»، ... خواهد بود. حال اگر بتوانید کدهایی بنویسید که اشیا فوق، ارتباط بین آنها و کارهایی که انجام می دهند را شبیه سازی کند در واقع یک برنامه مبتنی بر شی گرایی نوشته اید.

به مثال دیگری توجه کنید، فرض کنید بخواهیم برنامه ای برای یک کتابخانه بنویسیم. بر اساس شی گرایی باید اشیایی که در یک کتابخانه وجود دارند، ارتباط آنها و کارهایی که انجام می دهند را شناسایی کنیم. این اشیاء شامل «کتابخانه»، «عضو»، «کتابدار»، «کتاب» و شامل آعمال «امانت کتاب»، «رزرو کتاب»، «تحویل کتاب»، «ثبت کتاب جدید» است.

تمرین : لیستی از ۱۰۰ شی مختلف که در اطراف شما وجود دارد تهیه کنید.

اشیاء و کلاس

هر شی در دنیای واقعی دو دسته خصوصیات متمایز دارد: ۱- متعلقات ۲- رفتارها متعلقات، چیزهایی است که یک شیء مالک آنهاست و رفتار، کارهایی است که آن شیء انجام می‌دهد. برای مثال یک اتومبیل را تصور کنید فرمان، درها، کلاچ، موتور و چرخها از متعلقات اتومبیل محسوب می‌شوند و حرکت کردن، ایستادن، روشن شدن، خاموش شدن از رفتارهای اتومبیل محسوب می‌شوند. به عنوان مثالی دیگر، یک انسان را تصور کنید هر انسان نام، نام خانوادگی، شماره شناسنامه، شماره ملی، تاریخ تولد، ... دارد که همگی از متعلقات انسان هستند هر انسان ممکن است بخوابد، بیدارشود، راه برود، مطالعه کند، صحبت کند، ... که اینها همگی از جمله رفتارهای انسان هستند. دقت کنید که در هر دو مثال فوق، ما صرفا مجموعه ای از اشیا را توصیف کرده ایم بدون این که به شی خاصی اشاره کنیم. به عبارت دیگر، توضیح داده ایم که خانواده خودروها و خانواده انسانها چه متعلقات و رفتارهایی دارند بدون اینکه به یک خودرو یا یک فرد خاص اشاره کنیم و مثلاً بگوییم خودرویی که در جلوی منزل پارک شده است یک پژو ۴۰۵ خاکستری، با روکش صندلی مشکی، ... است! یا انسانی وجود دارد که نام آن سه راب سپهری متولد کاشان از توضیحات فوق اینطور می‌توان نتیجه گیری کرد که اگر بخواهیم برنامه ای را مبتنی بر شی گرایی پیاده سازی کنیم، باید اولاً اشیاء یک خانواده را شناسایی کنیم. مثلاً خودروها، انسانها، ساختمانها، کتابها ... که هر کدام معرف خانواده ای از اشیا هستند را شناسایی کنیم. بعد از شناسایی خانواده ها، باید متعلقات و رفتارهای مشترک بین اعضای هر خانواده را مشخص کنیم. مثلاً می‌دانیم که همه خودروها رنگ، حداکثر سرعت، ظرفیت، سال تولید و رفتارهایی از قبیل حرکت کردن، ایستادن، خاموش شدن دارند یا همه کتابها متعلقاتی از قبیل عنوان، نویسنده، ناشر، تاریخ نشر و تیراز و رفتارهایی از قبیل انتشار یافتن، توقف یافتن انتشار، ویرایش شدن دارند. به خانواده اشیاء که در واقع توصیفی از متعلقات و رفتارهای مشترک بین اعضای آن خانواده است، «کلاس» گفته می‌شود.

با دانستن کلاس یک خانواده، می‌توانیم اشیایی در برنامه ایجاد کنیم که معرف اشیاء دنیای واقعی باشد. مثلاً یک شی خودرو ایجاد کنیم که رنگ آن خاکستری، حداکثر سرعت آن ۲۱۰ کیلومتر بر ساعت، ظرفیت آن ۴ نفر، سال تولید آن ۱۳۹۷ باشد. به عنوان مثالی دیگر یک شی کتاب ایجاد کنیم که عنوان آن «آشپزی ایرانی» نویسنده آن «علی فرهمند»، ناشر آن «نشر خانه»، تاریخ نشر آن «مهرماه ۱۳۹۷»، و تیراز آن «۵۰۰۰ جلد» باشد.

برای درک بهتر و بیشتر مفاهیم اشیاء و کلاسها به مثالهای زیر دقت کنید.

کلاس کامپیوتر	
رفتار	متعلقات
انجام محاسبات، روشن شدن، خاموش شدن	مانیتور، صفحه کلید، کیس، مادربرد

یک شیء از کلاس کامپیوتر،

- اصول و مبانی یوگا نوشته محسن دانشور انتشارات گسترش علم ۱۳۸۲

ورزش و سلامتی از دیدگاه ملی پوشان نوشته علی مرادی انتشارات محسنی

کلاس کتاب	
رفتار	متعلقات
امانت بودن کتاب، در حال مطالعه بودن	نام، نویسنده، محتوا

اشیائی از کلاس کتاب،

- اصول و مبانی یوگا نوشته محسن دانشور انتشارات گسترش علم ۱۳۸۲

ورزش و سلامتی از دیدگاه ملی پوشان نوشته علی مرادی انتشارات محسنی ۱۳۸۴

کلاس کامیون	
رفتار	متعلقات
حرکت کردن، بارگیری کردن	نام، ظرفیت

یک شیء از کلاس کامیون،

- کامیون Volvo دیزل ۲۰ تن، کامیون لیلاند دیزل ۱۲ تن، ...

کلاس کارخانه	
رفتار	متعلقات
در حال تولید، آماده برای تولید، معلق	نام، نام محصول، خصوصی/ عمومی

شی گرایی

اشیائی از کلاس کارخانه،

- شرکت فولاد مبارکه
- شرکت تولید رب یک و یک
- کارخانه سیمان سپاهان

کلاس مدرسه	
رفتار	متعلقات
	نام، جنس دانشآموزان (دخترانه، پسرانه)، غیر تعطیل بودن/فعال بودن انتفاعی یا انتفاعی، ظرفیت

اشیائی از کلاس مدرسه،

- دبیرستان هاتف
- مدرسه پسرانه علوی
- مدرسه دخترانه حجاب

کلاس انسان	
رفتار	متعلقات
صحبت کردن، راه رفتن، خندیدن	دست، پا، سر، گردن، زبان

اشیائی از کلاس انسان،

- انشتاين
- هیتلر
- بتھوون
- شعبان
- مظفر

کلاس شهر	
رفتار	متعلقات
میزبان جام جهانی بودن، پایتخت بودن، تمیز بودن، زیبا بودن	جمعیت، مساحت، تعداد پارکها

اشیائی از کلاس شهر

- اصفهان
- تهران
- فرانکفورت
- سیدنی
- لس آنجلس

کلاس هواپیما	
رفتار	متعلقات
طول بال، تعداد موتور، ظرفیت، قدرت موتور، پرواز کردن، نشستن، سوخت گیری، تعمیر شدن حداکثر مسافت پرواز	

اشیائی از کلاس هواپیما

- یک هواپیمای بوئینگ
- یک هواپیمای ایرباس
- یک هواپیمای توپولوف
- یک هواپیمای C-130
- یک هواپیمای آنتونوف

کلاس میوه	
رفتار	متعلقات
فصلی است، درختی است	مزه(شیرین، ترش)، منطقه رویش

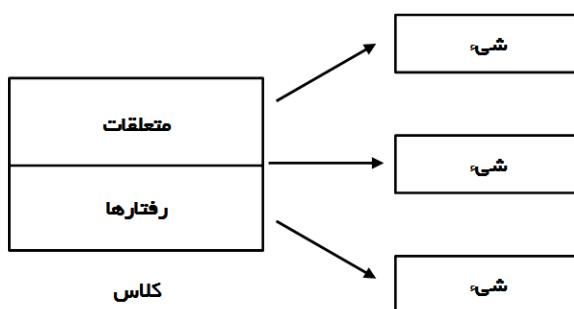
اشیائی از کلاس میوه

- یک گلابی
- یک هندوانه
- یک موز
- یک کیوی
- یک توت فرنگی

توجه: شی یک موجود واقعیست که وجود دارد در حالیکه کلاس فقط توصیفی از یک موجود واقعیست.

رابطه کلاس و شی

همچنان که گفته شد، هر کلاس به منزله یک نقشه یا طرح است که مجموعه‌ای از موجودات واقعی را توصیف می‌کند، و اشیاء نمونه‌هایی هستند که از روی کلاس ایجاد شده‌اند. به نظر می‌رسد دنیای واقعی هم بر اساس چنین منطقی بنا شده است، زیرا برای ساخت یک خودرو، ابتدا طرح و نقشه آن خودرو تهیه می‌شود (کلاس خودرو) و سپس از روی آن نقشه (آن کلاس) هزاران نمونه خودروی مشابه ایجاد می‌شود. شکل زیر رابطه کلاس و شی‌ه را در برنامه نویسی شی گرایی نشان می‌دهد.



شکل ۶-۱. ایجاد اشیاء از یک کلاس

اما کلاس در شی گرایی با طرح و نقشه یک شیء تفاوتهایی هم دارد، زیرا یک کلاس صرفاً توصیفی از اشیاء مشابه است و مقادیر متعلقات آنها را مشخص نمی‌کند. به عبارت دیگر، اگرچه اشیایی که از روی یک کلاس ایجاد می‌شوند، رفتار یکسانی دارند اما ممکن است مقادیر متعلقات آنها متفاوت باشد. به عنوان مثال با فرض

اینکه کلاس خودروی پژو ۲۰۶ دارای سه متعلقه به نام های «رنگ بدنه، تیپ خودرو، و شماره شاسی» باشد در مورد دو شی که از روی همین کلاس ساخته شده اند ممکن است رنگ بدنه یکی سفید و رنگ بدنه دیگری مشکی باشد، ممکن است تیپ یکی ۵ و دیگری تیپ ۶ باشد ممکن است، یکی از آنها شماره شاسی IR278329828 و دیگری IR3992929 را داشته باشند. در تمام این موارد اگرچه دو خودرو متعلقات یکسانی دارند ولی مقدار متعلقات هر خودرو متفاوت است.

تمرین: برای ۱۰ شیی که در تمرین قبل مشخص کردہاید کلاس‌های آنها را تعیین نموده و اشیای دیگری نیز از آن کلاس معرفی کنید.

طراحی کلاس

وقتی قرار است به روش شی گرایی برنامه ای بنویسید، قبل از هر چیز لازم است کلاس‌های آن برنامه را شناسایی کنید. اینکار با تحلیل نیازمندیهای آن نرم افزار انجام می شود که در نهایت منجر به طراحی کلاسها می شود. از تحلیل نیازمندیهای مشتری (کاربران نرم افزار)، اشیاء، تعلقات و رفتار آنها شناسایی می شوند و توسط طراح، کلاس‌های برنامه طراحی می شوند. برنامه نویسان ممکن است در بسیاری از موقع طراح هم باشند اما کمتر اتفاق می افتد که برنامه نویسان در گیر تحلیل نیازمندیهای مشتری شوند. به هر حال بعد از آنکه کلاس‌های برنامه طراحی و پیاده سازی شدن، منطق برنامه که شامل ایجاد اشیاء و تعامل بین آنهاست پیاده سازی می شوند. استفاده از روش شی گرایی در تحلیل، طراحی و پیاده سازی نرم افزار مزایای زیر را دارد.

- به آسانی می توان درک خود را از دنیای واقعی به دنیای کامپیوتر منتقل نمود (پیاده سازی کرد).
- اشکال‌زدایی، خطایابی و تست برنامه تسهیل می شود.
- تغییر و نگهداری برنامه آسان و کم هزینه می شود.

تعريف کلاس

همانطور که قبلاً آموخته اید، برای تعریف یک کلاس از کلمه کلیدی `class` و یک نام دلخواه (نام کلاس) استفاده می شود. مثلاً برای تعریف کلاسی که معرف دبیرستانهای است کلاس `HighSchool` به صورت زیر تعریف می شود.

```
class HighSchool {
    //class body
}
```

شی گرایی

به عبارت `class HighSchool` که کلاس و نام آنرا مشخص می کند «تعريف کلاس» گفته می شود. تعريف کلاس می تواند جزئیات بیشتری نیز داشته باشد که در ادامه این کتاب با آن آشنا می شوید. بلاfaciale بعد از تعريف کلاس، علامت { مشخص کننده شروع کلاس است و در انتهای علامت } خاتمه کلاس را مشخص می کند. به کدهایی که بین { و } نوشته می شوند «بدنه کلاس» گفته می شود. معمولاً قبل از کلمه `class` از کلمه کلیدی `public` استفاده می شود که در مورد مفهوم آن بعداً صحبت خواهیم کرد.

```
public class HighSchool{  
//class body  
}
```

هر کلاس که `public` باشد باید در یک فایل متنی با نام همان کلاس و با پسوند `.java` ذخیره شود. این بدین معنی است که کلاس فوق باید در فایل `HighSchool.java` ذخیره شود.

توجه: اگرچه هر نامی که با قوانین نامگذاری جاوا سازگار باشد (مثلاً با عدد شروع نشود، از کاراکترهای خاص تشکیل نشده باشد,...) را می توانید برای یک کلاس انتخاب کنید، اما بهتر است از «قاعده پاسکال» برای نامگذاری کلاسها پیروی کنید. براساس این قاعده، نام کلاس باید ضمن رعایت قوانین نامگذاری جاوا با حرف بزرگ انگلیسی شروع شود و اگر نام کلاس دو کلمه یا بیشتر باشد (مثلاً `HighSchool`) کلمات بعدی نیز با حروف بزرگ انگلیسی شروع شود. در قاعده پاسکال، استفاده از زیر خط (`_`) مجاز نیست. قاعده پاسکال که توسط همه برنامه نویسان جاوا استفاده می شود، باعث متحوالشکل کدهای جاوا و در نتیجه خوانashدن آنها می شود.

جدول ۶-۱ مثالهایی از نامگذاریهای مناسب، ناممناسب و نادرست را نشان می دهد:

مناسب	ناممناسب	نادرست
<code>SampleClassName</code>	<code>Sampleclassname</code>	<code>Sample Class Name</code>
<code>FirstWelcome</code>	<code>Welcome</code>	<code>1Welcome</code>
<code>Person</code>	<code>\$Value</code>	<code>First*Welcome</code>
<code>Country</code>	<code>_value</code>	<code>First/Welcome</code>

جدول ۶-۱. مثالهایی از نامهای معتبر و نامعتبر

پیاده سازی کلاس

بعد از تعريف کلاس، بدنه کلاس پیاده سازی می شود که شامل دو بخش «متعلقات» و «رفتار» می شود. متعلقات یک کلاس همان فیلهای کلاس و رفتار آن کلاس، متدهای آن کلاس هستند.

به عبارت دیگر و به ازای هریک از متعلقات می بایست یک فیلد و به ازای هریک از رفتار می بایست یک متدهای سازی شود. در زیر دو کلاس Aircraft و Car به همراه جزئیات آنها را مشاهده می کنید.

```

1 class HighSchool{
2
3     String name ;
4     String address;
5
6     void start(){
7         //do something
8     }
9     void finish(){
10        //do something
11    }
12 }
```

۶-۱

```

1 class Aircraft{
2
3     String name ;
4     int maxFlight ;
5     boolean inFlight ;
6     boolean inFixingMode;
7
8     void fly(){
9         //do something
10    }
11     void takeOff(){
12         //do something
13    }
14 }
```

۶-۲

```

1 class Car {
2     String name ;
3     String model ;
```

شی گرایی

```
4     String owner ;
5     String company ;
6
7     void start() {
8         //do something
9     }
10
11    void go() {
12        //do something
13    }
14
15    void doBreak() {
16        //do something
17    }
18
19    void stop() {
20        //do something
21    }
22
23 }
```

کد ۶-۳

تمرین : تمام کلاس‌هایی که در تمرینات قبل یافتید را پیاده سازی کنید.

ایجاد آبجکت

شیء یک نمونه واقعی از کلاس است. در حالیکه یک کلاس تنها توصیفی از یک موجود واقعی است، تا زمانیکه شیء ای از روی یک کلاس ایجاد نشود چیزی وجود ندارد. تصور کنید که بسیار گرسنه هستید سرمیز ناهار خوری نشسته‌اید و آنچه را دوست دارید روی کاغذ نوشته‌اید ولی نوشته‌ها، شما را سیر نمی کنند، باید آنچه را نوشته اید تجسم واقعی پیدا کند تا بتوانید آنرا میل کنید! تا اینجا هم شما فقط کلاس‌هایتان را نوشته اید ولی آنها فقط تعریفی از موجودات واقعی هستند، برای اینکه موجود واقعی داشته باشید باید از روی کلاس‌هایی که تعریف کرده‌اید شیء بسازید.

توجه: شیء، معادل فارسی کلمه Object است که در شی گرایی استفاده می شود. از اینجا به بعد، به جای کلمه شیء از کلمه آبجکت استفاده می کنیم که حس بهتر و دقیق تری نسبت به شیء منتقل می کند.

درست شبیه نام یک فرد که با آن می توان آن فرد را صدا زد، با او صحبت کرد یا خواسته ای را از وی مطرح نمود، هر آبجکت نیز دارای یک نام است تا بتوان به آن آبجکت دسترسی پیدا کرد. علاوه بر نام، هر آبجکت دارای جنس هم هست که در واقع همان کلاسی است که آبجکت از روی آن ایجاد می شود. ایجاد آبجکت در جاوا دو مرحله دارد.

الف-تعريف آبجکت

ب-ایجاد آبجکت

جمله

```
HighSchool alborz ;
```

یک آبجکت به نام alborz و از جنس کلاس HighSchool تعریف می کند. در حقیقت با تعریف یک آبجکت، یک نام (در این مثال alborz) و جنس آبجکت (در این مثال کلاس HighSchool) مشخص می شود بدون اینکه آن آبجکت به صورت فیزیکی در حافظه ایجاد شود. همانطور که در فصل قبل هم گفته شد، تا زمانیکه یک آبجکت ایجاد نشده باشد یعنی جایی در حافظه به آن اختصاص داده نشده است و مقدار آن پوچ (null) است.

برای ایجاد آبجکت در حافظه، یعنی اختصاص حافظه به آبجکت از کلمه کلیدی new استفاده می شود. به این ترتیب با اجرای جمله زیر، آبجکت alborz از کلاس Highschool در حافظه ایجاد می شود.

```
alborz = new HighSchool();
```

بعد از کلمه new با یک فاصله (space) نام کلاس به همراه () آورده می شود و در پایان نیز ; گذاشته می شود.

در بسیاری مواقع، تعریف و ایجاد آبجکت در یک خط به صورت زیر ادغام می شوند.

```
HighSchool alborz = new HighSchool();
```

مقداردهی فیلدهای آبجکت

بعد از اینکه یک آبجکت ایجاد شد، می‌توان فیلدهای آن را (در صورت وجود) مقدار دهی کرد. در مثال فوق که آبجکت `alborz` ایجاد شده است، نام و آدرس دبیرستان مشخص نیست! ما می‌خواهیم آبجکت `alborz` معرف دبیرستان البرز در خیابان انقلاب باشد در حالیکه فیلدهای `name` و `address` آبجکت `alborz` مقداری ندارند و برابر `null` هستند.

با نوشتن نام آبجکت، یک نقطه و سپس نام فیلد می‌توان به یک فیلد خاص دسترسی پیدا نمود، به آن مقداری انتساب داد یا مقدار آن را خواند. تکه کد زیر مقداردهی فیلدهای `name` و `address` از آبجکت `alborz` را نشان می‌دهد.

```
alborz.name = "دبیرستان البرز";
alborz.address="تهران، خیابان انقلاب";
```

تکه کد زیر دسترسی به مقادیر فیلدهای آبجکت `alborz` را نشان می‌دهد.

```
System.out.println("Name:" + alborz.name);
System.out.println("Address:" + alborz.address);
```

مشابه فیلدهای آبجکت می‌توان متدهای یک آبجکت را نیز فراخوانی کرد. کافی است بعد از نام آبجکت و پس از یک نقطه، با ذکر نام یک متدهای آنرا فراخوانی نمود.
جمله زیر، متدهای `start()` از آبجکت `alborz` را فراخوانی می‌کند.

```
alborz.start();
```

پیاده سازی کلاس مکعب

به عنوان مثال می‌خواهیم کلاسی طراحی کنیم که مشخص کننده یک مکعب باشد. تمام مکعب‌ها دارای سه بعد طول، عرض و ارتفاع هستند بنابراین کلاس مکعب دارای سه فیلد طول و عرض و ارتفاع است. به این ترتیب کلاس مکعب شکلی به صورت زیر خواهد داشت:

```
1 public class Box {
2     double width ;
3     double height ;
4     double depth;
5 }
```

در جای دیگری از برنامه (مثلا در یک کلاس دیگر)، از کلاس فوق یک آبجکت می سازیم و فیلدهای آن را مقدار دهی می کنیم در پایان با استفاده از مقادیر فیلدهای آبجکت، حجم آبجکت را محاسبه کرده و در خروجی استاندارد چاپ می کنیم.

```

1 public class BoxDemo {
2     public static void main(String args[]) {
3
4         Box mybox = new Box();
5         double vol;
6
7         // assign values to mybox's instance variables
8         mybox.width = 10;
9         mybox.height = 20;
10        mybox.depth = 15;
11
12        // compute volume of box
13        vol= mybox.width * mybox.height * mybox.depth;
14
15        System.out.println("Volume is " + vol);
16    }
17 }
```

کد ۷-۵

اضافه کردن یک متده کلاس Box

در کلاس BoxDemo برای محاسبه و چاپ حجم یک مکعب، حاصلضرب مقادیر فیلدهای آبجکت mybox محاسبه شده، سپس نتیجه آن در خروجی استاندار چاپ شده است. به نظر می رسد داشتن یک متده در کلاس Box که کار محاسبه حجم را انجام دهد مناسب است، همچنین می توان متده دیگری در نظر گرفت که کار چاپ مقدار محاسبه شده را انجام دهد. به این ترتیب کلاس Box به صورت زیر درمی آید.

```

1 public class Box {
2     double width ;
3     double height ;
4     double depth;
5
6     public double getVolume() {
7         return width * height * depth ;
8     }
}
```

شی گرایی

```
9  
10     public void printVolume() {  
11         System.out.println("Volume is "+getVolume());  
12     }  
13 }
```

کد ۴-۶

```
1 public class BoxWithMethodsDemo {  
2     public static void main(String args[]) {  
3  
4         Box box1 = new Box();  
5         Box box2 = new Box();  
6  
7         // assign values to box1's instance variables  
8         box1.width = 10;  
9         box1.height = 20;  
10        box1.depth = 15;  
11  
12        // assign values to box2's instance variables  
13        box2.width = 11;  
14        box2.height = 13;  
15        box2.depth = 8;  
16  
17        // print volume of box1  
18        box1.printVolume();  
19  
20        // print volume of box2  
21        box2.printVolume();  
22    }  
23 }
```

کد ۴-۷

در کلاس Box، متد () getVolume اضافه شده است که کار محاسبه حجم کلاس را انجام می دهد این متد یک مقدار double بر می گرداند که مشخص کننده حجم همان آبجکت است. متد () printVolume نیز حجم محاسبه شده توسط متد () getVolume را در خروجی استاندارد چاپ می کند. توجه کنید از آنجاییکه مقدار برگشتی متد () getVolume یک داده double است در

انتهای این متدهای return وجود دارد که مقدار محاسبه شده را برمی‌گرداند. متدهای printVolume() دارای مقدار برگشتی نیست بنابراین مقدار برگشتی آن void تعیین شده است. به کلاس Box می‌توان متدهای اضافه کرد که از طریق آن بتوان به تمامی فیلدهای کلاس Box مقادیر جدیدی انتساب داد در این صورت متدهای مزبور باید دارای پارامترهای ورودی باشد که مشخص کننده مقادیر جدید فیلدهای کلاس باشد مقدار برگشتی این متدهای void خواهد بود زیرا این متدهای تولید نمی‌کند. کلاس Box با اضافه شدن متدهای setDimension() به شکل زیر در خواهد آمد.

```

1  public class Box {
2      double width ;
3      double height ;
4      double depth;
5
6      public double getVolume(){
7          return width * height * depth ;
8      }
9
10     public void printVolume(){
11         System.out.println("Volume is "+getVolume());
12     }
13
14     public void setDimension(double w,double h,double d){
15         width = w ;
16         height = h ;
17         depth = d;
18     }
19 }
```

۷-۱ کد

استفاده از متدهای setDimension() در مثال زیر نشان داده شده است:

```

1  public class BoxWithMethodsDemo1 {
2      public static void main(String args[]){}
3
4          Box box1 = new Box();
5          Box box2 = new Box();
6
7          // assign values to box1's instance variables
8          box1.setDimension(10, 20, 15);
```

```
9  
10     // assign values to box2's instance variables  
11     box1.setDimension(11, 13, 8);  
12  
13     // print volume of box1  
14     box1.printVolume();  
15  
16     // print volume of box2  
17     box2.printVolume();  
18 }  
19 }
```

کد ۶-۹

اینک با استفاده از متد () setDimension می توان فیلدهای کلاس Box را مقداردهی کرد این متد سه پارامتر double دریافت می کند سپس مقادیر دریافت نموده را در فیلدهای کلاس Box قرار می دهد.

سازنده

در مثال های فوق برای مقداردهی کردن فیلدهای آبجکتی از کلاس Box از دو روش استفاده کردیم

الف- با دسترسی مستقیم به فیلدهای آبجکت، مقادیر آنها را مشخص نمودیم مثلا:

```
box1.width=11;
```

ب- متدهی نوشتمیم که با فراخوانی آن، مقادیری را که به صورت پارامتر دریافت می نمود به فیلدهای آبجکت انساب می داد:

```
box1.setDimension(10, 20, 15);
```

در دنیای واقعی، معمولاً با ساختن یک شی، خصوصیات آن شی نیز مقداردهی می شوند مثلاً وقتی شرکتی یک هواپیما می سازد ظرفیت، حداکثر مسافت پرواز ... این هواپیما در حین ساخت مشخص می شود و اینطور نیست که ابتدا هواپیما ساخته شود و سپس ظرفیت و حداکثر مسافت پرواز آن مشخص شود. طراحان جawa نیز امکان مقداردهی فیلدهای کلاس در حین ساختن آبجکت را فراهم آورده اند. هر گاه از روی یک کلاس (با استفاده از کلمه کلیدی new)، یک آبجکت می سازید متدهی از کلاس به نام «سازنده» فراخوانی می شود شما می توانید مقدار دهی اولیه فیلدهای کلاس است را در این متدهی قرار دهید به این ترتیب هرگاه آبجکتی از کلاس می سازید در همان هنگام فیلدهای آن آبجکت نیز مقدار دهی می شوند.
سازنده، متدهی هم نام کلاس است که هیچ مقدار برگشتی ندارد اما می تواند پارامتر داشته یا نداشته باشد مثلاً اگر بخواهید در هنگام ساختن آبجکت از کلاس Box، فیلدهای height، width و depth با اعداد 11، 12 و 9 مقداردهی شوند باید سازنده ای به صورت زیر به کلاس خود اضافه کنید.

```

1  public class Box {
2      double width ;
3      double height ;
4      double depth;
5
6      Box() {
7          System.out.println("Constructor is called!");
8          width = 11;
9          height = 12 ;
10         depth = 9 ;
11     }
12
13     public double getVolume() {
14         return width * height * depth ;
15     }
16
17     public void printVolume() {
18         System.out.println("Volume is "+getVolume());
19     }
20
21     public void setDimension(double w,double h,double d) {
22         width = w ;
23         height = h ;
24         depth = d;
25     }
26 }
```

۴-۱۰ کد

برای ساختن یک آبجکت از روی کلاس Box از دستور زیر استفاده می کنیم:

```
Box box = new Box();
```

با استفاده از دستور new Box()، یک آبجکت از کلاس Box ساخته می شود این دستور به صورت اتوماتیک باعث فراخوانی متده است () Box یعنی سازنده کلاس می شود.

در صورتیکه در یک کلاس هیچ سازنده ای ننویسید آن کلاس دارای یک سازنده است که به آن سازنده پیش فرض گفته می شود. یعنی حتی اگر برای کلاستان سازنده ای ننویسید باز هم می توانید با استفاده از جمله Box box = new Box();

شی گرایی

آبجکتی از کلاس `Box` بسازید. به سازنده ای که در کلاس پیاده سازی نمی شود اما در هنگام ساخته شدن آبجکت از یک کلاس، فراخوانی می شود سازنده پیش فرض گفته می شود.

توجه: در طراحی یک کلاس، در نامگذاری کلاس، متدها و فیلدها بهتر است نام کلاس با حرف بزرگ، نام متدها و فیلدها با حرف کوچک الفبای انگلیسی شروع شود از آنجائیکه نام کلاس با حرف بزرگ شروع می شود و سازنده نیز متندی با نام مشابه نام کلاس است بنابراین می توان گفت که سازنده تنها متندی در کلاس است که با حرف بزرگ الفبای انگلیسی آغاز می شود.

سازنده های دارای پارامتر

سازنده ای که تاکنون استفاده کردیم، فیلدهای آبجکت را با مقادیر پیش فرض مقدار دهی می کند به عبارت دیگر، تمام آبجکتها یی که توسط این سازنده ساخته می شوند شبیه هم هستند و فیلدهای آنها دارای مقادیر یکسان می باشند (اگر چه می توان با فراخوانی متند `setDimension()` مقادیر فیلدها را تغییر داد). در بسیاری از موارد لازم است در هنگام ساخته شدن آبجکت، فیلدهای آبجکت را با مقادیر متفاوتی مقدار دهی کنید. خوشبختانه این کار از طریق سازنده امکانپذیر است و از آنجائیکه سازنده یک متند است بنابراین می توانید برای آن پارامترهایی در نظر بگیرید. کلاس `Box` با استفاده از یک سازنده که دارای پارامترهایی برای مقدار دهی اولیه فیلدهای آبجکت است به صورت زیر درمی آید:

```
1 public class Box {  
2     double width ;  
3     double height ;  
4     double depth;  
5  
6     Box(double w, double h, double d){  
7         System.out.println("Constructor is called!");  
8         width = w;  
9         height = h ;  
10        depth = d ;  
11    }  
12  
13    public double getVolume(){  
14        return width * height * depth ;  
15    }  
16  
17    public void printVolume(){  
18        System.out.println("Volume is "+getVolume());  
19    }  
20}
```

```

19     }
20
21     public void setDimension(double w,double h,double d) {
22         width = w ;
23         height = h ;
24         depth = d;
25     }
26 }
```

کد ۷-۱۱

در هنگام ساخت آبجکت از این کلاس باید مقادیر پارامترهای سازنده را به صورت زیر به آن ارسال کنیم:

```

1 public class BoxParamConstDemo2 {
2     public static void main(String args[]) {
3
4         Box box1 = new Box(10, 20, 15);
5         Box box2 = new Box(11, 13, 8);
6
7
8         // print volume of box1
9         box1.printVolume();
10
11        // print volume of box2
12        box2.printVolume();
13
14    }
15 }
```

کد ۷-۱۲**کلمه کلیدی this**

در سازنده کلاس فوق که به صورت زیر پیاده سازی شده بود

```

Box(double w, double h, double d) {
    System.out.println("Constructor is called!");
    width = w;
    height = h ;
    depth = d ;
}
```

شی گرایی

نام پارامترهای سازنده `d`, `h` و `w` هستند در صورتیکه نام این پارامترها را به `width`, `height` و `depth` تغییر دهیم این متد به صورت زیر در می آید:

```
Box(double width, double height, double depth){  
    System.out.println("Constructor is called! ");  
    width = width;  
    height = height ;  
    depth = depth ;  
}
```

در اینجا نام پارامترها با نام فیلدهای کلاس یکسان است همچنان که حدس زده اید برای کامپایلر جاوا مشخص نیست که شما قصد دارید مقادیر داخل پارامترها را به فیلدها انتساب دهید یا قصد انتساب مقادیر فیلدهای آبجکت را به پارامترهای سازنده دارید!!!^۱ بنابراین باید به طریقی تفاوت بین پارامترها و فیلدها را مشخص کنید. این کار از طریق کلمه کلیدی `this` صورت می گیرد. هرگاه در یک کلاس بخواهید فیلد از همان کلاس را مشخص کنید باید با استفاده از کلمه کلیدی `this` یک ". قبل از نام فیلد آنرا از بقیه متغیرها یا فیلدها مجزا کنید. مثال فوق با بکارگیر کلمه کلیدی `this` به شکل زیر در می آید.

```
1 public class Box {  
2     double width ;  
3     double height ;  
4     double depth;  
5  
6     Box(double width, double height, double depth) {  
7         this.width = width;  
8         this.height = height ;  
9         this.depth = depth ;  
10    }  
11  
12    public double getVolume () {  
13        return width * height * depth ;  
14    }  
15  
16    public void printVolume () {  
17        System.out.println("Volume is "+getVolume());  
18    }
```

^۱ البته کامپایلر در این موقع فرض می کند که می خواهید مقادیر پارامترها را به همان پارامترها انتساب دهید!

```

19
20     public void setDimension(double w, double h, double d) {
21         width = w ;
22         height = h ;
23         depth = d;
24     }
25 }
```

کد ۷-۱۳

Garbage Collection

آجکتهايی که در يك برنامه ساخته مي شود با اتمام اجرای برنامه توسيط سيسitem عامل از حافظه پاک مي شوند اما در صورتیکه برنامه برای مدتی طولانی در حال اجرا باشد (مانند بسياري از برنامه هاي تحت وب) پس از گذشت مدت زمانی از عملكرد برنامه، حافظه سيسitem انباشته از آجکتهاي خواهد بود که بدون استفاده می مانند و دیگر در برنامه به آنها نيازی نیست. در برخی زيانها مثل C++ همانطور که برنامه نويس مسئول توليد آجکتهاي برنامه است (با استفاده از کلمه کليدي new) مسئوليت از بين بردن آجکتهاي بلااستفاده را نيز بر عهده دارد. اما در جاوا از بين بردن آجکتهاي بلااستفاده به صورت خودكار توسيط قسمتی از JVM به نام Garbage Collector انجام می شود. روش عملكرد Garbage Collector به اين شكل است که هر آجکتی که توسيط برنامه به آن دسترسی وجود ندارد بلااستفاده تلقی شده و باید از حافظه سيسitem پاک شود برای درک بهتر اين مسئله، به متذ زير توجه كنيد.

```

public void printVolume() {
    double volume = getVolume();
    String text = "Volume is " +volume ;
    System.out.println(text);
}
```

پس از اتمام اجرای متذ فوق، متغير text به هيج وجه قابل دسترسی نخواهد بود زيرا اين متغير داخل بدنۀ متذتعريف شده و خارج از آن قابل دسترس نخواهد بود بنابراین پس از خاتمه اجرای متذ، متغير text از نظر Garbage Collector يك آجکت بلااستفاده تلقی شده و باید از حافظه سيسitem پاک شود.

متذ finalize()

اگرچه کار پاک کردن يك آجکت بلااستفاده به صورت خودكار توسيط Garbage Collector صورت می گيرد اما ممکن است بخواهيد با از بين رفتن يك آجکت، کار خاصی نيز انجام شود برای مثال ممکن است

شی گرایی

یک آبجکت یکی از فایلهای سیستم را در اختیار گرفته باشد و در حال نوشتن یا خواندن داده از آن باشد اگر این آبجکت قبل از اینکه فایل اشغال شده آزاد شود توسط Garbage Collector از حافظه پاک شود فایل همچنان اشغال می‌ماند و توسط آبجکتها دیگر قابل دسترس نخواهد بود بنابراین مواردی پیش می‌آیند که لازم است با حذف شدن یک آبجکت از حافظه سیستم، عملیات خاصی نیز انجام شود برای این منظور متدهای `finalize()` پیش‌بینی شده است کافی است عملیات مورد نظر خود را داخل این متدهای سازی کنید.

```
1 public class Box {  
2     double width ;  
3     double height ;  
4     double depth;  
5  
6     Box(double width, double height, double depth){  
7         this.width = width;  
8         this.height = height ;  
9         this.depth = depth ;  
10    }  
11  
12    public double getVolume(){  
13        return width * height * depth ;  
14    }  
15  
16    public void printVolume(){  
17        System.out.println("Volume is "+getVolume());  
18    }  
19  
20    public void setDimension(double w,  
21                                double h,  
22                                double d){  
23        width = w ;  
24        height = h ;  
25        depth = d;  
26    }  
27    protected void finalize(){  
28        String text = "width: " +  
29                           width +",height: " +  
30                           height + ",depth: " + depth;  
31        System.out.println("a Box with " + text +  
32                           "is going to be destroyed");  
33    }  
34}
```

```
33     }
34 }
```

کد ۷-۱۴

در متد `finalize()` از کلاس فوق، فقط یک پیغام مبنی بر حذف آبجکتی از کلاس `Box` با ابعاد مشخص در خروجی استاندارد چاپ می شود. در این متد از کلمه کلیدی `protected` استفاده شده است که در مورد معنا و مفهوم آن در بخش‌های بعدی صحبت خواهیم کرد.

Overloading

در یک کلاس جاوا می توان چندین متد با یک نام و با یک نوع مقدار برگشتی داشت مشروط براینکه پارامترهای ورودی آنها متفاوت باشد. به متدهایی که دارای چنین خصوصیتی باشند متدهای `overloaded` گفته می شود و به چنین فرایندی `overloading` اطلاق می شود. وقتی یکی از متدهای `overloaded` فراخوانی می شود جاوا از طریق تعداد و نوع پارامترهای فراخوانی متد تشخیص می دهد که کدامیک از متدها را باید اجرا کند.

مثال زیر کلاسی را نشان می دهد که متدهای آن `overload` شده اند.

```
1 class OverloadedMethods {
2     void test() {
3         System.out.println("No parameters");
4     }
5     // Overload test for one integer parameter.
6     void test(int a) {
7         System.out.println("a: " + a);
8     }
9
10    // Overload test for two integer parameters.
11    void test(int a, int b) {
12        System.out.println("a and b: " + a + " " + b);
13    }
14    // overload test for a double parameter
15    double test(double a) {
16        System.out.println("double a: " + a);
17        return a * a;
18    }
19 }
```

کد ۷-۱۵

شی گرایی

فراخوانی متدهای این کلاس در کلاس زیر نشان داده شده است:

```
1  class TestOverloading {
2      public static void main(String args[]) {
3          OverloadedMethods om = new OverloadedMethods();
4          double result;
5
6          // call all versions of test()
7          om.test();
8          om.test(10);
9          om.test(10, 20);
10         result = om.test(123.25);
11         System.out.println("Result of ob.test(123.25): "+
12             result);
13     }
14 }
```

کد ۷-۱۴

با اجرای کلاس فوق، خروجی زیر تولید خواهد شد:

```
No parameters
a: 10
a and b: 10 20
double a: 123.25
Result of ob.test(123.25): 15190.5625
```

همانطور که ملاحظه می کنید متدهای `test()` چهار بار `overload` شده است اولین فرم این متدهای هیچ پارامتری دریافت نمی کند، دومی فقط یک پارامتر عدد `int`، سومین فرم دو عدد `int` و چهارمین یک پارامتر `double` دریافت می کند.

سازنده های `overload` شده

از آنجاییکه سازنده یک کلاس نیز از جمله متدهای کلاس محسوب می شود می توان در هر کلاس سازنده های مختلف با پارامترهای مختلف داشته باشید. برای درک این مطلب به آخرین نسخه از کلاس `Box` که در کد ۶-۱۴ نشان داده شده است توجه کنید. اگر بخواهیم آبجکتی از این کلاس ایجاد کنیم می بایست با استفاده از سازنده این کلاس که سه پارامتر `int` دریافت می کند به صورت زیر آن آبجکت را ایجاد کنیم.

```
Box box = new Box(12, 11, 10);
```

اما قطعاً موقعاً پیش می‌آیند که در هنگام ایجاد آبجکت فقط طول و عرض Box مشخص است یا حتی ممکن است هیچیک از فیلدهای Box مشخص نباشد.

برای حل این مسئله می‌توانید از overloading، برای داشتن چندین سازنده با پارامترهای مختلف در یک کلاس استفاده نمود در واقع داشتن چندین سازنده با پارامترهای مختلف در یک کلاس یکی از کارهای رایج در طراحی کلاس‌هاست.

کلاس Box که در آن سه سازنده با پارامترهای مختلف وجود دارد به صورت زیر خواهد بود.

```

1 public class Box {
2     double width ;
3     double height ;
4     double depth;
5
6     Box(double width, double height, double depth) {
7         this.width = width;
8         this.height = height ;
9         this.depth = depth ;
10    }
11
12    // constructor used when no dimensions specified
13    Box() {
14        width = -1; // use -1 to indicate
15        height = -1; // an uninitialized
16        depth = -1; // box
17    }
18
19    // constructor used when cube is created
20    Box(double len) {
21        width = height = depth = len;
22    }
23
24    //remaining part of the Box
25 }
```

کد ۷-۱۸

در کلاس زیر ساخت آبجکتهای مختلف از کلاس Box با استفاده از سازنده‌های مختلف آن ساخته شده است:

```
1 public class OverloadCons {  
2     public static void main(String args[]) {  
3         // create boxes using the various constructors  
4         Box mybox1 = new Box(10, 20, 15);  
5         Box mybox2 = new Box();  
6         Box mycube = new Box(7);  
7  
8         double vol;  
9  
10        // get volume of first box  
11        vol = mybox1.getVolume();  
12        System.out.println("Volume of mybox1 is " + vol);  
13  
14        // get volume of second box  
15        vol = mybox2.getVolume();  
16        System.out.println("Volume of mybox2 is " + vol);  
17  
18        // get volume of cube  
19        vol = mycube.getVolume();  
20        System.out.println("Volume of mycube is " + vol);  
21    }  
22 }
```

کد ۷-۱۹

استفاده از آبجکت به عنوان پارامتر

متدهایی که تاکنون نوشته ایم همگی دارای پارامترهای نوع اولیه بودند اما متدهای می توانند به جای داده های نوع اولیه، آبجکتی از کلاس های مختلف دریافت کنند. مثال زیر را در نظر بگیرید:

```
1 class Test {  
2     int a, b;  
3  
4     Test(int i, int j) {  
5         a = i;  
6         b = j;  
7     }  
8 }
```

```

9     // return true if o is equal to the invoking object
10    boolean equals(Test o) {
11        if(o.a == a && o.b == b)
12            return true;
13        else
14            return false;
15    }
16 }
```

کد ۴-۳۰

در فوق کلاسی با نام Test نوشته شده است در این کلاس متدهای equals() و hashCode() تعریف شده است که آبجکتی از جنس کلاس Test به عنوان پارامتر دریافت می‌کند و با مقایسه کردن فیلد های آبجکت دریافت کرده با فیلد های خود، مشخص می‌کند که آبجکت دریافت نموده با خودش برابر است یا خیر. در صورت تساوی فیلد های دو آبجکت مقدار true و در غیر اینصورت مقدار false برگرداند. در زیر استفاده از این متدها در یک کلاس دیگر نشان داده شده است.

```

1 public class PassOb {
2     public static void main(String args[]) {
3         Test ob1 = new Test(100, 22);
4         Test ob2 = new Test(100, 22);
5         Test ob3 = new Test(-1, -1);
6
7         System.out.println("ob1 == ob2: " + ob1.equals(ob2));
8
9         System.out.println("ob1 == ob3: " + ob1.equals(ob3));
10    }
11 }
```

کد ۴-۳۱

خروجی برنامه فوق به صورت زیر خواهد بود:

```
ob1 == ob2: true
ob1 == ob3: false
```

در سازنده های یک کلاس نیز می‌توان از آبجکت به عنوان پارامتر استفاده نمود یکی از سازنده های رایج که در کلاسها نوشته می‌شود سازنده ای است که پارامتری از جنس همان کلاس دریافت می‌کند و مقدار هر یک

شی گرایی

از فیلدهای آبجکت را به فیلدهای خود انتساب می دهد. کلاس Box که دارای چنین سازنده ای است به صورت زیر خواهد بود:

```
1 public class Box {  
2     double width ;  
3     double height ;  
4     double depth;  
5  
6     Box(Box box) {  
7         this.width = box.width;  
8         this.height = box.height ;  
9         this.depth = box.depth ;  
10    }  
11  
12    Box(double width, double height, double depth) {  
13        this.width = width;  
14        this.height = height ;  
15        this.depth = depth ;  
16    }  
17  
18    // constructor used when no dimensions specified  
19    Box() {  
20        width = -1; // use -1 to indicate  
21        height = -1; // an uninitialized  
22        depth = -1; // box  
23    }  
24  
25    // constructor used when cube is created  
26    Box(double len) {  
27        width = height = depth = len;  
28    }  
29  
30  
31    public double getVolume () {  
32        return width * height * depth ;  
33    }  
34 }
```

استفاده از این کلاس و سازنده جدید آن در کلاس زیر نشان داده شده است:

```

1  public class OverloadCons2 {
2      public static void main(String args[]) {
3
4          // create boxes using the various constructors
5          Box mybox1 = new Box(10, 20, 15);
6          Box mybox2 = new Box();
7          Box mycube = new Box(7);
8
9          Box myclone = new Box(mybox1);
10
11         double vol;
12
13         // get volume of first box
14         vol = mybox1.getVolume();
15         System.out.println("Volume of mybox1 is " + vol);
16
17         // get volume of second box
18         vol = mybox2.getVolume();
19         System.out.println("Volume of mybox2 is " + vol);
20
21         // get volume of cube
22         vol = mycube.getVolume();
23         System.out.println("Volume of cube is " + vol);
24
25         // get volume of clone
26         vol = myclone.getVolume();
27         System.out.println("Volume of clone is " + vol);
28     }
29 }
```

کد ۶-۲۳

نگاهی دقیقتر به پارامترهای متند

وقتی متند با یک یا بیش از یک پارامتر فراخوانی می شود، فراخوانی می تواند به دو صورت زیر انجام شود.

- فراخوانی با مقدار (call-by-value)
- فراخوانی با ارجاع (call-by-reference)

شی گرایی

به این معناست که وقتی متدهای فراخوانی می‌شود، کپی داده‌ها به جای اصل داده‌ها به متدهای ارسال می‌شود. این بدین معناست که اگر آن متدهای مقدار پارامتری را که دریافت کرده تغییر دهد اصل آن داده تغییر نخواهد کرد.

اما در مقابل، به این معناست که وقتی داده‌ای به عنوان پارامتر به یک متدهای ارسال می‌شود در حقیقت آن داده (همان داده و نه کپی آن) به متدهای ارسال شود تا اگر مقدار پارامتر توسط متدهای تغییر داده شود اصل نیز تغییر کند.

برای درک این دو مفهوم، به کلاس MethodInvocation که در زیر پیاده سازی شده است توجه کنید. در این کلاس یک متدهای changeParams() که دو پارامتر int دریافت می‌کند و مقدار پارامتر اولی را دو برابر و مقدار پارامتر دوم را نصف می‌کند.

```
1 class MethodInvocation {
2     void changeParams(int i, int j) {
3         i *= 2;
4         j /= 2;
5     }
6 }
```

کد ۷-۲۴

حال اجازه دهید، در جای دیگری متدهای changeParams() را فراخوانی کنیم و نتیجه آنرا مشاهده کنیم.

```
1 class TestMethodInvocation {
2     public static void main(String args[]) {
3         MethodInvocation mi = new MethodInvocation();
4         int a = 15, b = 20;
5
6         System.out.println("a and b before call: " + a +
7             " " + b);
8
9         mi.changeParams(a, b);
10
11        System.out.println("a and b after call: " + a +
12            " " + b);
13    }
14 }
```

کد ۷-۲۵

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.

```
a and b before call: 15 20
a and b after call: 15 20
```

همانطور که ملاحظه می کنید اگرچه مقدار داده های عددی که به متدهای ارسال شده اند درون متدهای تغییر یافته اند ولی این تغییر روی اصل داده اثربخش نداشته است. این همان چیزی است که به آن فراخوانی با مقدار (call-by-value) می گوییم زیرا در فراخوانی متدهای کپی داده ها به جای اصل داده به متدهای ارسال می شود.

برای درک call-by-reference، کلاسهای Holder و MethodInvocation را که به صورت زیر تغییر یافته تصور کنید.

```
1 public class Holder {
2     int a, b;
3
4     Holder(int a, int b) {
5         this.a = a;
6         this.b = b;
7     }
8 }
9 class MethodInvocation {
10     void changeParams(Holder holder) {
11         holder.a *= 2;
12         holder.b /= 2;
13     }
14 }
15 }
```

کد ۴-۳۶

متدهای changeParams() در کلاس MethodInvocation به گونه ای تغییر یافته که به جای دو عدد int، یک آبجکت از جنس کلاس Holder دریافت کند. این متدهای فیلد آبجکت Holder را که دو عدد int هستند تغییر می دهد. نتیجه فراخوانی این متدها در زیر ملاحظه کنید.

```
1 class TestMethodInvocation1{
2     public static void main(String args[]) {
3         Holder h = new Holder(15, 20);
```

```

4     MethodInvocation mi = new MethodInvocation();
5
6     System.out.println("h.a and h.b before call: " + h.a +
7                     " " + h.b);
8
9     mi.changeParams(h);
10
11    System.out.println("h.a and h.b after call: " +
12                     h.a + " " + h.b);
13 }
14 }
```

۶-۳۷

خروجی اجرای کد فوق به صورت زیر خواهد بود:

```

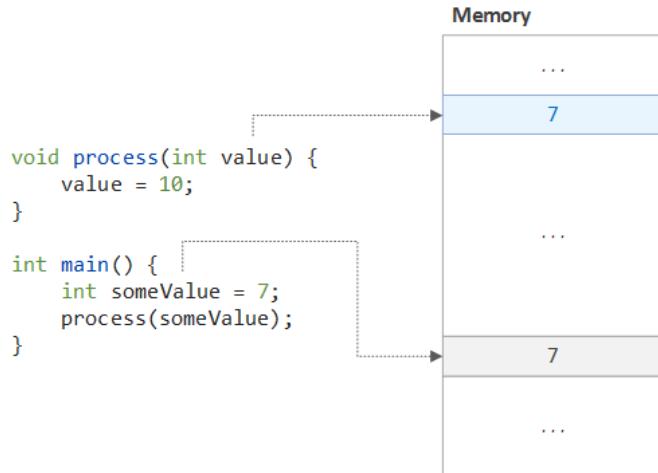
h.a and h.b before call:15 20
h.a and h.b after call: 30 10
```

همانطور که ملاحظه می کنید تغییراتی که متده بر روی پارامترها ایجاد کرده، روی اصل پارامترها تاثیر گذاشته و مقدار پارامتر نیز تغییر کرده است به همین دلیل گفته می شود که فراخوانی متده در اینجا-
call by-reference بوده است.

با مقایسه دو مثال فوق متوجه می شویم که متده changeParams () که پارامترهای int دارد-
call-by-value و متده Holder changeParams () که پارامتری از دریافت می کند-
reference است.

صبر کنید! اگرچه نتیجه گیری فوق درست است اما گفته می شود که جاوا call-by-value است! اما چرا
چنین ادعایی وجود دارد؟
برای درک این استدلال باید مفاهیم فوق را بیشتر توضیح دهیم.

وقتی متده با پارامتری از جنس داده های پایه (int, float, boolean,...) فراخوانی می شود، یک
کپی از داده پایه به متده ارسال می شود. درست شبیه این است که یک داده دیگر اما با مقدار یکسان به متده
ارسال شود بنابراین هر تغییری در مقدار پارامتر روی اصل آن داده تاثیری ندارد. شکل زیر این قاعده را نشان
می دهد.



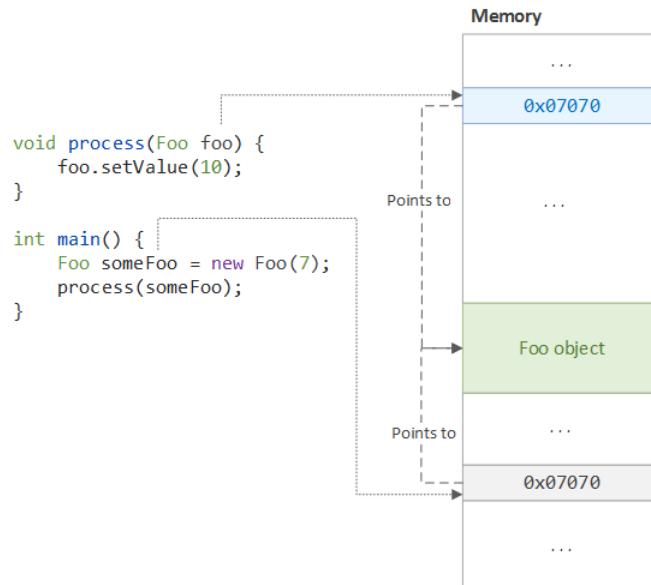
شکل ۷-۲. فراخوانی یک متده با پارامتر پایه

اما وقتی پارامتر متده، آبجکتی از یک کلاس باشد (به جای اینکه یک مقدار پایه باشد)، به جای یک کپی از آن داده یک کپی از «آدرس» آن آبجکت به متده ارسال می شود. به این ترتیب تا مادامی که متده بخواهد محتوای آدرسی که دریافت کرده است را تغییر دهد طبیعتاً اصل آن آبجکت نیز تاثیر می پذیرد و رفتاری مشابه call-by-reference از خود نشان می دهد اما در واقع call-by-reference نیست، زیرا اگر این طور می بود می بایست متده فراخوانی شده بتواند آدرس آن آبجکت را تغییر دهد و کدی که در متده changeParams() به در زیر نوشته شده تاثیری روی اصل آبجکت نمی گذارد زیرا در اینجا متده changeParams() آبجکت جدیدی در حافظه ایجاد می کند (که به معنی آدرس جدیدی در حافظه است).

```

class MethodInvocation {
    void changeParams(Holder holder) {
        holder = new Holder();
        holder.a=83;
        holder.b=18;
    }
}
  
```

تغییراتی که روی پارامتر holder ایجاد می شوند فقط در محدوده متده قابل مشاهده هستند و اصل آبجکت تاثیری نمی پذیرد. به شکل زیر دقت کنید.



شکل ۶-۳. فراخوانی یک متده با پارامتر آبجکت

وقتی متده پارامتری از آبجکت دریافت میکند، آدرس آن در حافظه را (به جای کپی آن آبجکت) دریافت میکند. به این ترتیب تا زمانی که محتوایی که آن ارجاع به آن اشاره دارد را تغییر دهد اصل آبجکت نیز تغییر میکند، اما به مجردی که با بکارگیری `new` به جای محتوای قبلی به محل جدیدی از حافظه اشاره میکند بدون اینکه اصل آبجکت تحت تاثیر قرار گیرد صرفاً مقدار پارامتر تغییر میکند به همین دلیل زبان جاوا یک زبان `call-by-value` خوانده میشود.

برگرداندن آبجکت توسط یک متده

مقدار برگشتی یک متده میتواند از هر نوع داده ای باشد نوع داده برگشتی میتواند از انواع اولیه (`int`, `char`, ..., `float`) یا آبجکتی از یک کلاس باشد برای مثال در کلاس `Test` یک متده به نام `incrByTen()` تعریف شده است که آبجکتی از کلاس `Test` برمی گرداند در این آبجکت، مقدار `a` در آبجکت جدید ۱۰ واحد بیشتر از قبلی است:

```

1 class DataWrapper {
2     int a;
3
4     DataWrapper(int i) {
5         a = i;
6     }
7 }
```

```

8     DataWrapper incrByTen() {
9         DataWrapper temp = new DataWrapper(a+10);
10        return temp;
11    }
12 }
```

کد ۷-۲۸

کد زیر استفاده از این متده در یک کلاس دیگر را نشان می دهد.

```

1 class TestDataWrapper {
2     public static void main(String args[]) {
3         DataWrapper ob1 = new DataWrapper(2);
4         DataWrapper ob2;
5
6         ob2 = ob1.incrByTen();
7         System.out.println("ob1.a: " + ob1.a);
8         System.out.println("ob2.a: " + ob2.a);
9
10        ob2 = ob2.incrByTen();
11        System.out.println("ob2.a after second increase: "
12                           + ob2.a);
13    }
14 }
```

کد ۷-۲۹

اجرای کلاس فوق نتیجه ای به صورت زیر خواهد داشت:

```
ob1.a: 2
ob2.a: 12
ob2.a after second increase: 22
```

همانطور که ملاحظه می کنید فراخوانی متده `incrByTen()`، آبجکت جدیدی از کلاس `Test` بر می گرداند مقدار `a` در این آبجکت ۱۰ واحد از مقدار `a` در آبجکتی که متده `incrByTen()` آن فراخوانی شده است بیشتر است.

استفاده از پکیج برای دسته‌بندی کلاسها

همانطور که گفته شد، هر کلاس جاوا در یک فایل متنی با همان نام کلاس و با پسوند `.java`. روی سیستم ذخیره می شود بنابراین با مطالعه که تاکنون آموخته ایم نمی توانیم دو کلاس همنام، اما متفاوت داشته

شی گرایی

باشیم زیرا امکان وجود دو فایل با نام یکسان در یک شاخه روی دیسک سخت وجود ندارد از این گذشته، در برنامه‌های بزرگ معمولاً تعداد کلاس‌های برنامه بسیار زیاد خواهد بود و از آنجاییکه هر کلاس نیز در یک فایل جداگانه همنام با نام کلاس روی سیستم ذخیره می‌شود، در یک برنامه بزرگ تعداد زیادی فایل وجود خواهد داشت که کار مدیریت و استفاده از آنها را دشوار می‌کند برای اینکه مدیریت و استفاده از کلاسها آسان شود و احیاناً امکان داشتن کلاس‌های مختلفی با نام یکسان در یک برنامه میسر شود مفهومی به نام پکیج ابداع شده است.

package که معنی لغوی آن در فارسی «بسته» یا «بسته بندی» است، امکانی برای بسته بندی کردن کلاس‌های یک برنامه است تا با قرار گرفتن کلاس‌های مختلف در بسته های مختلف، تداخل نامگذاری کلاسها از بین بود و امکان داشتن کلاس‌های همنام در برنامه بوجود بیاید ما در طول مطالب این کتاب از کلمه «پکیج» به جای کلمه انگلیسی package استفاده خواهیم کرد.
از نظر پیاده سازی، یک پکیج چیزی جز یک شاخه یا دایرکتوری روی کامپیوتر نیست، این بدین معناست که اگر بخواهید یک کلاس را داخل یک پکیج قرار دهید، تنها کافیست یک دایرکتوری با نام دلخواه خود بسازید و فایل `.java` را درون آن قرار دهید! البته برای اینکه شاخه ای که فایل کلاستان در آن قرار دارد به عنوان پکیج شناسایی شود، باید نام آن شاخه را در ابتدای تعریف کلاستان به عنوان پکیج معرفی کنید.

به عنوان مثال فرض کنید کلاس `MainClass`، که یکی از کلاس‌های برنامه است در مسیر

`C:\java-program\src\MainClass.java`

قرار دارد حال می‌خواهیم این کلاس را داخل یک پکیج با نام `mypack` قرار دهیم برای این منظور، ابتدا داخل شاخه `C:\java-program\src`: یک شاخه جدید با نام `mypack` ایجاد کرده و فایل `MainClass.java` را در آن قرار می‌دهیم:
`C:\java-program\src\mypack\MainClass.java`

سپس شاخه `mypack` را به عنوان پکیج برنامه، به صورت زیر در کلاس معرفی می‌کنیم.

```
package mypack ;  
  
public class MainClass {  
    //codes  
}
```

اینک شاخه `mypack` در کلاس `MainClass` به عنوان پکیج این کلاس در نظر گرفته می‌شود.
پکیج ها می‌توانند شاخه‌های متعدد و تودرتو باشند به عنوان مثال فرض کنید که بخواهیم کلاس `MainClass` را داخل پکیجی به صورت `mypack.pack1` قرار دهیم در اینصورت مشابه حالت قبل

شاخه جدیدی با نام pack1 داخل شاخه mypack ایجاد می‌کنیم و فایل MainClass.java را داخل آن قرار می‌دهیم.

C:\java-program\src\mypack\pack1\MainClass.java

سپس به ابتدای تعریف کلاس باید مسیر پکیج را به صورت زیر اضافه کنیم:

```
package mypack.pack1 ;  
  
public class MainClass {  
    //codes  
}
```

استفاده از پکیج، فایده‌های دیگری نیز دارد از آنجائیکه بسیاری از کلاس‌هایی که افراد یا شرکتها می‌نویسنند جنبه کتابخانه‌ای ۲ دارد و ممکن است توسط شرکتهای دیگر به صورت کتابخانه استفاده شود بهتر است نامگذاری پکیج‌ها به‌گونه‌ای باشد که اگر دو کلاس همان‌نم در دو کتابخانه مختلف وجود داشته باشند تداخلی از لحاظ اسم و پکیج رخ ندهد به عنوان مثال اگر شرکت A یک کلاس با نام Utility.java در یک پکیج طراحی کرده باشد و شرکت B نیز یک کلاس با همین نام و با همین پکیج طراحی کرده باشد و شما بخواهید همزمان از هر دو کلاس در برنامه‌تان استفاده کنید مشکل تداخل نامگذاری در استفاده از آنها را خواهید داشت به همین منظور شرکت Sun پیشنهاد کرده است که هر شرکت برای نامگذاری پکیج‌های خود از آدرس URL همان شرکت به صورت معکوس استفاده کند مثلاً در صورتی که شرکت A دارای URL <http://www.atlassoft.ir> است پکیج‌های خود را با نام ir.atlassoft شروع کند از آنجائیکه URL هر شرکت منحصر به فرد است، پکیج‌های تمام شرکتها منحصر بفرد خواهند ماند.

توجه: در نامگذاری پکیج بهتر است همواره از حروف کوچک الفبای انگلیسی استفاده شود.

در صورتیکه بخواهید در یک کلاس از کلاس دیگری که در پکیج متفاوتی قرار دارد استفاده کنید باید مسیر کامل آن کلاس را مشخص کنید مثلاً اگر بخواهید در برنامه‌تان از کلاس Date که در پکیج java.util قرار دارد استفاده کنید باید در زمان تعریف آبجکت از آن کلاس، مسیر کامل آنرا به صورت زیر مشخص کنید:

```
java.util.Date date ;
```

شی گرایی

اما از آنجاییکه نوشتمن مسیر کامل برای تمام آبجکتهايی که در یک برنامه ساخته می شوند کار خسته کننده و نازيبایی به نظر می رسد، راه دیگری نیز برای معرفی کلاسهایی که در پکیج قرار دارند پیش بینی شده است در این روش باید در ابتدای کلاس و بعد از نام پکیج، با استفاده از کلمه کلیدی `import` نام پکیج و کلاسی را که استفاده می کنید را مشخص نمایید. از فصل دوم به خاطر دارید که برای استفاده از کلاس `JOptionPane` دستور

```
import javax.swing.JOptionPane;
```

را به ابتدای تعریف کلاس اضافه کردیم، کلاس `JOptionPane` در پکیجی با نام `javax.swing` قرار دارد که از کلاسهای JDK محسوب می شود.

دانستن چندین نکته در مورد پکیج لازم به نظر می رسد:

اول اینکه کلاسهای استاندارد جاوا در پکیج ای به نام `java` تعریف شده اند عملیات اصلی در زبان جاوا نیز در کلاسهایی در پکیج دیگری به نام `lang` داخل `java` تعریف شده اند. به عبارت بهتر، کلاسهای پایه جاوا در پکیجی به نام `java.lang` تعریف شده اند برای استفاده از کلاسهایی که در این پکیج قرار دارند نیازی به `import` کردن آنها ندارید این بدین معناست که تمام کلاسهایی که در پکیج `java.lang` قرار دارند به صورت خودکار در تمام کلاسهای جاوا `import` می شوند.

همچنین اگر از دو کلاس همانم که هر کدام پکیج متفاوتی دارند در یک کلاس استفاده می کنید استفاده از `import` کفایت نمی کند و در هنگام استفاده از هر کلاس، لازم است مسیر کامل آن کلاس را مشخص کنید. به عنوان مثال در جاوا دو کلاس به نام های `Date` و `java.sql.Date` وجود دارند یکی از آنها در `java.util` و دیگری در `java.util.Date` قرار دارند در صورتیکه بخواهید همزمان از هر دو کلاس در یک کلاس استفاده کنید باید هنگام استفاده از آنها نام کامل هر کلاس را مشخص کنید مثلا

```
java.util.Date date1 ;  
java.sql.Date date2 ;
```

سومین مسئله اینکه، هنگامی که یک پکیج را `import` می کنید تنها کلاسهایی در آن پکیج قابل استفاده خواهند بود که به صورت `public` تعریف شده باشند همچنین تنها متدها و فیلدھایی از یک کلاس `import` شده را می توانید فراخوانی کنید که به صورت `public` تعریف شده باشند. برای مثال کلاس زیر را در نظر بگیرید.

```
1 package mypack;  
2  
3 /* Now, the Balance class, its constructor, and its  
4 show() method are public. This means that they can
```

```

5      be used by non-subclass code outside their package.
6  */
7 public class Balance {
8     String name;
9     double bal;
10
11    public Balance(String n, double b) {
12        name = n;
13        bal = b;
14    }
15
16    public void show() {
17        if(bal<0)
18            System.out.print("-->> ");
19        System.out.println(name + ": $" + bal);
20    }
21 }
```

کد ۴-۳۰

کلاس فوق به صورت **public** تعریف شده و متد **show()** در این کلاس نیز **public** است. بنابراین می‌توان از این کلاس در یک کلاس دیگر که در یک پکیج متفاوت قرار دارد استفاده نمود و متد **show()** آنرا فراخوانی کرد. کلاس زیر این مطلب را نشان می‌دهد.

```

1 package anotherPack;
2
3 import mypack.Balance;
4
5 class TestBalance {
6     public static void main(String args[]) {
7
8         /* Because Balance is public, you may use Balance
9          class and call its constructor. */
10        Balance test = new Balance("J. J. Jaspers", 99.88);
11
12        test.show(); // you may also call show()
13    }
14 }
```

کد ۴-۳۱

حق دسترسی به متدها و فیلدهای یک کلاس

همانطور که قبلاً گفته شد آبجکتها یکی از روی کلاسهای برنامه می‌سازید شبیه‌سازی اشیاء دنیای واقعی هستند، بالطبع خصوصیات و رفتار آنها نیز شبیه‌سازی خصوصیات و رفتار اشیاء دنیای واقعی خواهد بود. در دنیای انسانها هر فرد مسایل و خصوصیاتی دارد که برای تمام افراد دیگر قابل مشاهده است و می‌توانند به آن دسترسی پیدا کنند مثلاً میزان تحصیلات، اندازه قد، نام و جنسیت از خصوصیاتی است که هر فردی در اختیار هر فرد دیگری قرار می‌دهد اما همان فرد، خصوصیات و مسایلی دارد که فقط توسط افراد خاصی قابل مشاهده است مثلاً مسایل خانوادگی یک نفر که فقط توسط افراد خانواده آن فرد قابل مشاهده و دستیابی است اما دسته سومی از مشخصات و خصوصیات نیز وجود دارند که شخصی هستند و توسط هیچ فرد یا شخص دیگری، به جز خود آن شخص قابل مشاهده و دسترسی نیست و حتی در سطح خانواده نیز قابل دسترسی و مشاهده نیست.

این موضوع در مورد اشیای بی جان و غیرزنده نیز وجود دارد. یک مانیتور را تصور کنید، بخشی از اطلاعات و خصوصیات آن مانیتور در اختیار همگان قرار داده می‌شود مثلاً رنگ، اندازه، شرکت سازنده، فرکانس، ... در مقابل بخش دیگری از خصوصیات هستند که فقط در همکاران شرکت سازنده قرار می‌گیرد و از طرف شرکت اصلی در اختیار شرکتهای مونتاژ کننده آن مانیتور قرار می‌گیرد. مثلاً اطلاعات کلی از اجزای داخلی مانیتور، اندازه پیچ‌ها و قطعات داخلی و... و بخشی از خصوصیات مانیتور فقط در اختیار شرکت سازنده قرار دارد و در اختیار هیچ فرد یا شرکت دیگر قرار داده نمی‌شود مثلاً طراحی مدارات و چیپ‌های مانیتور. اگر قرار باشد، شی گرایی دنیای واقعی را به همان شکلی که هست مدل کند پس باید راهی برای مدل سازی این سه دسته خصوصیت نیز داشته باشد.

در شی گرایی به خصوصیات یا رفتارهایی که به صورت نامحدود در اختیار تمامی آبجکتها دیگر قرار می‌گیرد، `public` یا عمومی گفته می‌شود. به خصوصیات یا رفتاری نیز که فقط توسط کلاسهای همان پکیج قابل دسترسی است `protected` یا محافظت شده گفته می‌شود.

خصوصیات و رفتاری هم که توسط کلاس دیگری جز همان کلاس قابل دسترسی نیست `private` گفته می‌شود. به مثال زیر دقت کنید.

```

1 //This class is created just to show accessing
2 //properties and methods of a class
3 public class AccessDemo {
4     public int a;
5     protected int b;
6     private int c;
7
8     public void doAction1() {
9         //some codes

```

```

10     }
11
12     protected void doAction2() {
13         //some codes
14     }
15
16     private void doAction3() {
17         //some codes
18     }
19 }
```

کد ۶-۳۲

حال کلاس دیگری به نام AccessDemo را تصور کنید که از کلاس AccessDemoUse استفاده نموده است.

```

1 public class AccessUseDemo {
2     public static void main(String[] args) {
3
4         AccessDemo access = new AccessDemo();
5
6         // These are OK, a and b may be accessed           //
7         directly
8         access.a = 5 ;
9         access.doAction1();
10
11
12         access.b = 6;
13         access.doAction2();
14
15         // This is not OK and will cause an error
16         access.c = 7; // Error!
17         access.doAction3(); // Error!
18     }
19 }
```

کد ۶-۳۳

در کلاس AccessDemoUse جملات

```
access.c = 7;
access.doAction3();
```

شی گرایی

به هیچ وجه کامپایل نخواهد شد زیرا `c` و `doAction3()` به صورت `private` تعریف شده‌اند و خارج کلاس `AccessDemo` به هیچ وجه قابل دسترسی نیستند. جملات

```
access.a = 6;  
access.doAction1();
```

نیز تحت هر شرایطی کامپایل خواهد شد زیرا `a` و `doAction1()` به صورت `public` تعریف شده‌اند و در تمام کلاس‌های دیگر قابل دسترسی هستند.

از آنجائیکه `b` و `doAction2()` به صورت `protected` تعریف شده‌اند جملات

```
access.b = 6;  
access.doAction2();
```

تنها در شرایطی کامپایل خواهد شد که کلاس `AccessDemo` و `AccessDemoUse` در یک پکیج قرار داشته باشند.

جدول زیر خلاصه‌ای از حق دسترسی هر کلاس به خصوصیات و رفتار کلاس دیگر را نشان می‌دهد.

Specifier	class	subclass	package	world
<code>private</code>	X			
<code>protected</code>	X	X*	X	
<code>public</code>	X	X	X	X
<code>package</code>	X		X	

جدول ۶-۲. محدودیت‌های دسترسی

static

در برنامه نویسی شی گرا نمی‌توان بدون ساختن آبجکت از یک کلاس، متدهای آن کلاس را فراخوانی کرد اگر دقت کنید در بسیاری از متدهایی که قبل از این فصل تعریف و به کلاس اضافه نمودیم قبل نام متدهای کلمه کلیدی `static` استفاده نمودیم درصورتیکه بخواهید یک متدهای فیلد به کلاس خاصی وابستگی نداشته باشند به عبارت دیگر بدون ساختن آبجکت از آن کلاس قابل فراخوانی و استفاده باشند با استفاده از کلمه کلیدی `static` قبل از نام متدهای فیلد آنرا مشخص می‌کنیم. اگر دقت کرده باشید قبل از متدهای `main()` کلمه `static` استفاده می‌شود. علت تعریف کردن متدهای `static` به صورت `main()` به این خاطر است که قبل از اجرای برنامه هنوز هیچ آبجکتی ساخته نشده است بنابراین چاره‌ای جز اجرای برنامه از طریق یک متدهای `static` وجود ندارد.

از آنجائیکه متدهای `static` به آبجکتی که در آن هستند وابستگی ندارند بنابراین از داخل یک متدهای `static` نمی‌توان به فیلدهای کلاسی که در آن قرار دارد دسترسی پیدا کرد و یا از کلمه‌های `this` یا

برای دسترسی به فیلدهای آن کلاس یا کلاس پدر استفاده نمود. در این متدها فقط می‌توان از `super` متدها یا فیلدهای `static` آن کلاس استفاده نمود.

برای فراخوانی متدهای `static` کافی است پس از نام کلاس، نام متدها یا `static` مورد نظر را مشخص کنیم.

```
classname.method();
```

مثال زیر بکارگیری متدها و متغیرهای `static` را در یک کلاس نشان می‌دهد:

```

1 // Demonstrate static variables, methods, and blocks.
2 class UseStatic {
3     static int a = 3;
4     static int b;
5
6     static void meth(int x) {
7         System.out.println("x = " + x);
8         System.out.println("a = " + a);
9         System.out.println("b = " + b);
10    }
11
12    static {
13        System.out.println("Static block initialized.");
14        b = a * 4;
15    }
16
17    public static void main(String args[]) {
18        meth(42);
19    }
20 }
```

کد ۷-۳۴

اجرای کلاس فوق، نتیجه‌ای بصورت زیر خواهد داشت:

```
Static block initialized.
x = 42
a = 3
b = 12
```

تعريف فیلد های final

فیلدهایی که تاکنون تعریف کردیم، هر لحظه می‌توانند مقدار جدیدی قبول کنند، به عبارت دیگر، مقدار آنها قابل تغییر است. اما شما می‌توانید فیلد یک کلاس را بگونه‌ای تعریف کنید که مقدار آن همیشه ثابت باشد. به فیلدهایی که مقدار آنها ثابت است و نمی‌توان در طول برنامه مقدار آنها را تغییر داد، final گفته می‌شود برای اینکه یک فیلد را به صورت final تعریف کنید باید کلمه کلیدی final را قبل از نوع فیلد بنویسید:

```
final int increment = 5;
```

مثال فوق یک فیلدی به نام increment که مقدار آن 5 است را تعریف می‌کند همچنانکه ملاحظه می‌کنید این فیلد به صورت final تعریف شده یعنی مقدار آنرا نمی‌توانید تغییر دهید. اگر در جایی از برنامه بخواهید مقدار آنرا تغییر دهید (مثلاً با استفاده از دستور increment=12) با یک خطای کامپایل مواجه خواهید شد.

در زیر مثال دیگری را ملاحظه می‌کنید که فیلدهایی از انواع مختلف به صورت final تعریف شده‌اند.

```
public final int left = 5;  
protected final String name="Factory";  
private final char c = 'A' ;
```

توجه داشته باشید که همواره مقدار یک متغیر final را باید در هنگام تعریف متغیر مشخص کنید.

کلاس‌های داخلی و تودرتو

این امکان وجود دارد که داخل یک کلاس، کلاس دیگری تعریف کنید به این کلاس‌ها که داخل یک کلاس دیگر تعریف می‌شوند «کلاس داخلی» گفته می‌شود. محدوده دسترسی به یک کلاس داخلی، محدوده کلاس اصلی است مثلاً اگر کلاس B داخل کلاس A تعریف شود کلاس B فقط برای A شناخته شده است و خارج از A قابل دسترس نیست. کلاس B به تمام فیلدهای کلاس A (حتی فیلدهایی که به صورت private تعریف شده اند) دسترسی دارد اما کلاس A به هیچکدام از فیلدهای کلاس B دسترسی ندارد. کلاس‌های داخلی را می‌توان به دو صورت static و غیر static تقسیم کرد کلاس به کلاسی گفته می‌شود که قبل از نام آن از کلمه کلیدی static استفاده شده باشد از آنجائیکه این کلاس به صورت static تعریف شده است نمی‌تواند به فیلدها و متدهای غیر static کلاس اصلی دسترسی پیدا کند. به خاطر این محدودیت استفاده از کلاس‌های داخلی که به صورت static تعریف شده اند بسیار کم است.

مهمترین نوع کلاس‌های داخلی inner class هستند یک کلاس داخلی گفته می‌شود که غیر static باشد این گونه کلاس‌ها دسترسی کامل به فیلدها و متدهای غیر static کلاس اصلی دارند و

می توانند مستقیماً و بدون هیچگونه آدرس دهی، همانند دیگر متدهای کلاس اصلی، به آنها دسترسی پیدا کنند.

مثال زیر نحوه تعریف و بکارگیری یک inner class را نشان می دهد کلاس Outer دارای یک فیلد به نام outer_x و یک متده به نام test() و یک inner class به نام Inner است.

```

1  class Outer {
2      int outer_x = 100;
3
4      void test() {
5          Inner inner = new Inner();
6          inner.display();
7      }
8
9      // this is an innner class
10     class Inner {
11         void display() {
12             System.out.println("display: outer_x = " + outer_x);
13         }
14     }
15 }
```

۶-۳۵ کد

در کلاس Outer که داخل کلاس Inner تعریف شده است به تمام متدها و فیلدهای کلاس دسترسی وجود دارد در کلاس Inner یک متده به نام display() تعریف شده است این متده مقدار outer_x را در خروجی استاندارد چاپ می کند کلاس زیر بکارگیری کلاس Outer را نشان می دهد.

```

1  public class TestOuter {
2      public static void main(String args[]) {
3          Outer outer = new Outer();
4          outer.test();
5      }
6  }
```

۶-۳۶ کد

در متده main() یک آبجکت از کلاس Outer ساخته شده است و سپس متده test() آن فراخوانی شده است در اینجا از طریق آبجکت outer نمی توان به متده display() دسترسی پیدا کرد. خروجی اجرای برنامه فوق به صورت زیر خواهد بود:

```
display: outer_x = 100
```

نکته مهمی که باید به خاطر داشته باشید این است که کلاس Inner به تمام فیلدها و متدهای کلاس Outer دسترسی دارد ولی عکس این قضیه نادرست است یعنی کلاس Outer نمی تواند به هیچکدام از متدها یا فیلدهای کلاس Inner دسترسی پیدا کند مثال زیر این مطلب را بهتر نشان می دهد.

```
1 class Outer1 {
2     int outer_x = 100;
3
4     void test() {
5         Inner inner = new Inner();
6         inner.display();
7     }
8
9     // this is an innner class
10    class Inner {
11        int y = 10; // y is local to Inner
12        void display() {
13            System.out.println("display: outer_x = "
14                         + outer_x);
15        }
16    }
17
18    void showy() {
19        //error, y not known here!
20        System.out.println(y);
21    }
22 }
```

کد ۴-۳۷

همچنان که ملاحظه می کنید، y در کلاس Inner تعریف شده است بنابراین در خارج از کلاس Inner و در متدهای showy() قابل دسترسی نخواهد بود.
به عنوان نکته ای دیگر بد نیست بدانید کلاس داخلی می تواند در هر قسمتی و در هر بلوکی از کلاس اصلی تعریف شود مثال زیر کلاس Inner را که داخل یکی از متدهای کلاس اصلی تعریف شده است را نشان می دهد.

```

1  class Outer2 {
2      int outer_x = 100;
3
4      void test() {
5          for(int i=0; i<10; i++) {
6              class Inner {
7                  void display() {
8                      System.out.println("display: outer_x = " +
9                          outer_x);
10             }
11         }
12         Inner inner = new Inner();
13         inner.display();
14     }
15 }
16 }
```

۶-۳۹ کد

```

1 public class TestOuter2 {
2     public static void main(String args[]) {
3         Outer2 outer = new Outer2();
4         outer.test();
5     }
6 }
```

۶-۴۰ کد

خروجی اجرای این برنامه به صورت زیر خواهد بود.

```
display: outer_x = 100
```

خلاصه

در این فصل با مفاهیم کلاس و آبجکت آشنا شدید هر کلاس تعریفی از یک موجود واقعی یا آبجکت است که دو جزء خصوصیت و رفتار است خصوصیت مشخص کننده جزئی از کلاس است که آن موجود مالک آن است و رفتار عملکرد آن موجود را توصیف می کند. یک خصوصیت را با استفاده از تعریف یک فیلد و یک رفتار را با استفاده از یک متده است در یک کلاس تعریف می کنند.

ایجاد یک موجود مجازی از روی یک کلاس شامل دو مرحله است: تعریف موجود و سپس ساختن موجود. تعریف موجود همانند تعریف یک متغیر است که در آن نوع موجود (نام کلاس) و سپس نام آن موجود آورده می شود ولی برای ساختن یک موجود از روی یک کلاس از کلمه کلیدی `new` به همراه نام کلاس و () استفاده می شود مثلا:

```
Time now = new Time();
```

یک موجود با نام `now` از جنس کلاس `Time` تعریف می کند که با استفاده از () `new Time()` ایجاد شده است.

تعریف و ایجاد یک موجود می تواند در دو خط مجرما صورت پذیرد:

```
Time now;
now = new Time();
```

برای مقداردهی کردن فیلدهای یک آبجکت یا فراخوانی متدهای آن، کافی است نام آبجکت، یک نقطه () و سپس نام آن متده یا فیلد نوشته شود مثلا:

```
now.hour = 5;
now.toString();
```

متده `toString()` از این آبجکت را فراخوانی می کند.

سازنده یا `constructor` متده از کلاس است که در هنگام ساختن آبجکت از آن کلاس فراخوانی می شود، این متده همانم کلاس است و هیچ مقدار برگشتی ندارد تمام کلاسهها به صورت پیش فرض دارای یک سازنده هستند که هیچ پارامتری دریافت نمی کند اما می توان هر سازنده ای با هر تعداد پارامتر به کلاس افزود. از سازنده ها برای مقداردهی اولیه فیلدهای آبجکت استفاده می شود.

در صورتیکه در یک کلاس به فیلدها یا متدهای همان کلاس دسترسی پیدا کنیم می توان از کلمه کلیدی `this` استفاده نمود این کلمه خصوصا در زمانی کاربرد دارد که یک متده دارای پارامترهایی همان فیلدهای کلاس است، با استفاده از کلمه کلیدی `this` می توان بین آن پارامتر و فیلد کلاس تمایز قایل شد. مثلا در سازنده کلاس `Time`، که پارامترهایی همان فیلدهای کلاس وجود دارند می توان از `this` به صورت زیر استفاده نمود:

```
public Time(int hour, int minute, int second) {
    this.hour = hour;
    this.minute = minute;
```

```

        this.second = second ;
    }
}

```

در یک کلاس می توان متدهای مشابه، با یک نام، یک مقدار برگشتی اما پارامترهای مختلف داشت به این خاصیت متدها در یک کلاس overloading گفته می شود مثلا در کلاس Time می توان دو سازنده به صورت زیر داشت:

```

public Time() {
    this.hour = 0 ;
    this.minute = 0;
    this.second = 0;
}

public Time(int hour, int minute, int second) {
    this.hour = hour ;
    this.minute = minute;
    this.second = second ;
}

```

همانطور که ملاحظه می کنید در کلاس Time دو سازنده () تعريف شده است که یکی از آنها هیچ پارامتری دریافت نمی کند و دیگری سه پارامتر int دریافت می کند.
در فراخوانی هر متد ممکن است پارامترهایی یک متد، به صورت call-by-reference یا call-by-value ارسال شوند، در call-by-value، یک کپی از پارامترها به متد ارسال می شود این بدين معناست که اگر توسط متد فراخوانی شده، مقادیر پارامترها دستکاری شوند اصل داده ها تغییر نخواهند کرد اما در مقابل در call-by-reference اصل داده ها به صورت پارامتر به متد ارسال می شوند و اگر توسط متد فراخوانی شده، مقادیر پارامترها دستکاری شود اصل داده ها نیز تغییر خواهد کرد. تمام داده های نوع اولیه (float، int، char و ...) به صورت call-by-value به متد ها ارسال می شوند اما تمام داده هایی که از نوع کلاس هستند به صورت call-by-reference به متد ارسال می شوند.
در صورتیکه یک متد یا فیلد به صورت static تعريف شود می توان بدون ساختن آبجکت از آن کلاس به آن فیلد یا متد دسترسی پیدا کرد (قبل در فصول قبل، با تعريف و استفاده از یک متد static آشنا شده اید).

می توان یک فیلد را به صورت final تعريف نمود، در این صورت حتما باید این فیلد دارای یک مقدار اولیه باشد هیچ گاه نمی توان در هیچ نقطه ای از برنامه مقدار آنرا تغییر داد.
در این فصل با مفهوم پکیج آشنا شدید با استفاده از پکیج می توان کلاسهای یک برنامه را دسته بندی نمود و احیانا کلاسهای همانام در یک برنامه داشت. تعريف یک پکیج برای یک کلاس، طی دو مرحله انجام می شود ابتدا باید یک دایرکتوری با نام پکیج بسازید و فایل آن کلاس را داخل آن دایرکتوری قرار دهید و سپس داخل آن کلاس و قبل از نوشتن جملات import، مسیر آن دایرکتوری بعد از کلمه کلیدی package بنویسید.

شی گرایی

در صورتیکه بخواهید از کلاسی استفاده کنید که درون پکیج دیگری قرار دارد باید مسیر کامل کلاس (نام کلاس همراه با پکیج آن) را با استفاده از کلمه کلیدی `import` معرفی کنید این معرفی به کامپایلر و اجراءکننده برنامه جاوا امکان می دهد تا مسیر کلاسی را که قصد استفاده از آن را دارید پیدا کنند.

کلمات کلیدی `public`, `protected` و `private` حق دسترسی یک کلاس به خصوصیات و رفتار کلاس دیگر را مشخص می کنند وقتی یک متدها یا فیلد یک کلاس به صورت `public` تعریف می شوند آن فیلد یا متدها کلاس تمام کلاسهای دیگر قابل دسترسی است، `private` خصوصیات و رفتار شخصی یک کلاس را مشخص می کند وقتی یک فیلد یا متدها به صورت `private` تعریف می شوند آن فیلد یا متدها فقط داخل همان کلاس قابل دسترسی است، `protected` نیز که در فصل بعد با آن آشنا خواهید شد، مشخص کننده خصوصیات و رفتاری است که توسط همان کلاس و کلاسهای فرزند آن کلاس قابل دسترسی هستند.

اگر از هیچیکی از این کلمات استفاده نشود، آن فیلد یا متدها فقط توسط کلاسهایی که در پیکچ یکسان با آن کلاس قرار دارند قابل دسترسی و استفاده خواهد بود.

تمرینات

- ۱) کلاس Date طراحی کنید که معرف تاریخ باشد در این کلاس از سازنده برای مقداردهی به فیلدهای آن استفاده کنید همچنین در این کلاس متدهای `setDate()` و دیگر متدهای `setter` را نیز پیاده سازی کنید.
- ۲) کلاس Weather طراحی کنید که معرف وضعیت آب و هوا باشد، در این کلاس درجه حرارت فعلی، بیشترین درجه حرارت در ۲۴ ساعت، کمترین درجه حرارت در ۲۴ ساعت و میزان رطوبت را می توانید به عنوان فیلدهای کلاس در نظر بگیرید. متدهای `toString()` و `equals()` را پیاده سازی کنید.
- ۳) کلاس Line طراحی کنید که معرف خط در محورهای مختصات باشد، هر خط یک شیب و یک نقطه برخورد با محور x ها دارد دیگر متدهایی که برای این کلاس به نظرتان می رسد را پیاده سازی کنید.
- ۴) کلاس Car طراحی کنید که معرف یک خودرو باشد، فیلدها و متدهای این کلاس را پیاده سازی کنید، دو خودرو تنها در صورتیکه با یکدیگر مساوی هستند که بدون توجه به تاریخ ساخت، مدل و دیگر خصوصیات آنها تنها از نظر توان و سرعت با یکدیگر مساوی باشند.
- ۵) کلاس Course طراحی کنید که معرف یک «درس» باشد که در یک موسسه آموزشی تدریس می شود دو درس تنها زمانی با یکدیگر مساوی هستند که بدون توجه به استادی که آنرا ارایه می کند عنوان یکسان و زمان شروع و پایان یکسان داشته باشند.
- ۶) کلاس Mountain را که معرف کوه است را طراحی کنید، فیلدها و متدهای این کلاس را پیاده سازی کنید. دو کوه تنها زمانی با یکدیگر مساوی هستند که ارتفاع یکسان داشته باشند.
- ۷) کلاس Account را که معرف یک حساب بانکی است را طراحی و پیاده سازی کنید.
- ۸) کلاس Player را که معرف یک بازیکن ورزشی است را طراحی و پیاده سازی کنید دو بازیکن تنها زمانی یکسان قلمداد می شوند که قیمت یکسانی داشته باشند.
- ۹) کلاس مثلث را طراحی و پیاده سازی کنید، دو مثلث تنها زمانی با یکدیگر مساوی هستند که دو ضلع و زاویه بین آنها، سه ضلع آنها یا دو زاویه و ضلع بین آنها با یکدیگر مساوی باشند. (مسئله را ساده فرض کنید و هر مثلث را فقط با یکی از شرایط مثلا با دو ضلع و زاویه بین آنها مشخص کنید).
- ۱۰) کلاس City را که معرف شهر است طراحی و پیاده سازی کنید، دو شهر تنها زمانی با یکدیگر مساوی قلمداد می شوند که ارجاع یکسانی داشته باشند (که هر دو یک آجکت باشند!).
- ۱۱) کلاس Country را که معرف یک کشور است طراحی و پیاده سازی کنید، دو کشور تنها زمانی با یکدیگر مساوی هستند که جمیعت یکسان داشته و متعلق به یک قاره باشند.

شی گرایی

(۱۲) متدهی بنویسید که آرایه ای از آبجکت دریافت کند و جنس (کلاس) داده های درون آبجکت را برگرداند.

```
public Class[] getType(Object[] objects)
```

(۱۳) کلاسی بنویسید که دارای مجموعه ای از متدهای **overload** باشد که یک `consume()` باشد که یک `Object` را دریافت می کند و مجموع آنها را در قالب یک عدد `double` را نگهداری می کند.

(۱۴) متدهی بنویسید که دو آرایه `Object` دریافت کند و در صورتیکه تمام عناصر دو آرایه از یک جنس باشند دو آرایه را با یکدیگر ترکیب کند و برگرداند.

(۱۵) کلاسی بنویسید که یک رشته را از طریق سازنده خود دریافت کند. این کلاس دارای یک متدهی `randomize()` است که کاراکترهای رشته را به صورت تصادفی جابجا می کند و رشته بهم ریخته ای تولید کند. وقتی هر کاراکتر را در طول رشته جابجا می کند موقعیت اصلی آن کاراکتر را نیز در یک آرایه `int` متناظر نگهداری می کند به گونه ای که با داشتن آن آرایه بتوان رشته اصلی را بازیابی کرد. (آرایه عددی درست مثل کلید عمل می کند که می توان رشته بهم ریخته را رمزگشایی کرد)

V

ارث بری

موجودات زنده از نظر خصوصیات و رفتار شبیه والدین خود هستند مثلاً فرزند یک انسان از نظر رنگ پوست، قد، رنگ چشم، جنس مو و خصوصیات دیگری مشابه پدر یا مادر خود است. علاوه بر خصوصیات، رفتار فرزندان نیز مشابه رفتار والدین آنهاست. البته فرزندان ممکن است در مواردی با والدین خود متفاوت باشند اما در اصل و اکثر موارد مشابه آنها هستند.

از شبهاه فرزندان به والدین که بگذریم، می‌توان گفت که همواره «فرزندان نوع تکامل یافته‌ای از والدین خود هستند». وجود همین قاعده باعث شده که فرایند تکامل بشر در طول تاریخ شکل بگیرد و فرزندان بشر که روزی غارنشین بودند امروزه مدرن ترین زندگی را تجربه می‌کنند و برای سفر به فضا برنامه ریزی می‌کنند.

اما از انسان و موجودات زنده نیز که بگذریم، در مورد موجودات غیرزنده و اشیاء نیز می‌توان رابطه پدر و فرزندی را تصور کرد. مثلاً وقتی موتورسیکلت اختراع شد مدت‌ها پیش از آن دوچرخه اختراع شده بود. در واقع موتور سیکلت نوع تکامل یافته‌ای از دوچرخه بود که تمام خصوصیات و رفتار دوچرخه را داشت اما خصوصیات و رفتار کاملتر و پیشرفته‌تری را نیز از خود نشان می‌داد. به این ترتیب می‌توان موتور سیکلت را فرزند دوچرخه تصور کرد. رابطه پدر و فرزندی را در بین بسیاری از اشیاء دیگر نیز می‌توان تعمیم داد، مثلاً رابطه آموزشگاه و دانشگاه، یا رابطه شهر و شهرستان، خیابان و بزرگراه، ... به این خاصیت که یک شی خصوصیاتی مشابه شی دیگر دارد طوری که به منزله فرزند آن شی است «ارث بری» گفته می‌شود.

وقتی چنین رابطه‌ای بین موجودات زنده و غیرزنده دنیای واقعی وجود دارد، طبیعی است که انتظار داشته باشیم تا در شی گرایی نیز وجود داشته باشد چراکه هدف شی گرایی مدل سازی اشیا و مفاهیم دنیای واقعی مبتنی بر واقعیت است.

ارث بری یکی از مفاهیم پرکاربرد در طراحی شی گرایی به شمار می‌رود و ضمن جلوگیری از تکرار کد، توسعه، تست و نگهداری کد را تسهیل می‌کند.

برای درک مفهوم ارث بری در شی گرایی به مثال زیر توجه کنید. تصور کنید که قرار است برنامه‌ای برای شهرداری تولید شود که در آن خانه‌ها و آپارتمان‌ها، خیابانها، کوچه‌ها و اتوبانها ثبت می‌شوند. واضح است که نیاز به پیاده سازی کلاس‌های `House`, `Apartment`, `Street`, `Apartement`, `Highway` و `Alley` داریم، به عنوان نمونه، دو کلاس `House` و `Apartment` پیاده سازی شبیه لیست ۱-۷ خواهد داشت.

```

1 public class House {
2     int no;
3     double surface;
4 }
5
6 public class Apartment {
7     int no;
8     double surface;
9     int floor;
10 }
```

۱-۷ کد

هر خانه دارای شماره (`no`) و مساحت (`surface`) است. هر آپارتمان نیز علاوه بر شماره و مساحت، طبقه (`floor`) نیز دارد. کلاس `Apartment` علاوه بر تمام خصوصیات کلاس `House`, یک خصوصیت اضافه `floor` نیز دارد و از آنجاییکه هر دو کلاس از یک خانواده (مسکن) هستند می‌توان فرض کرد که هر آپارتمان فرزند یک خانه است.

در توصیف ارث بری از کلمه کلیدی `extends` به صورت زیر استفاده می‌شود.

```

1 public class House1 {
2     int no;
3     double surface;
4 }
5
6 public class Apartment1 extends House1 {
7     int floor;
8 }

```

کد ۷-۲

در کلاس Apartment خصوصیات no و surface حذف شده اند زیرا بواسطه جمله extends که در تعریف کلاس Apartment اضافه شده است، خصوصیات کلاس House به کلاس Apartment به ارث رسیده است و به صورت ضمنی در این کلاس وجود دارند.

ارث بری به طرز شگفت انگیزی طراحی برنامه را آسان می کند، پیاده سازی برنامه را سرعت می دهد و تغییر و نگهداری برنامه را تسهیل می کند. با بودن ارث بری شما با خیال آسوده می توانید خصوصیتی را به پدر اضافه یا کم کنید و مطمئن باشید که آن خصوصیت به فرزندان اضافه یا کم می شود بدون آنکه مجبور باشید کلاسهای فرزند را دستکاری کنید و چیزی را حذف یا اضافه کنید.

به کلاس پدر، که اشتقاق از آن صورت می گیرد superclass و به کلاس فرزند که از کلاس پدر مشتق می شود subclass گفته می شود. می توان گفت که کلاس فرزند (Subclass) نوع خاص تری از کلاس پدر (Superclass) است زیرا علاوه بر متدها و خصوصیاتی که کلاس پدر به ارث می برد متدها و فیلدهای اضافه تری دارد. همچنین گفته می شود کلاس subclass کلاس superclass را توسعه داده است.

شكل کلی تعریف ارث بری به صورت زیر است.

```

class subclass extends superclass {
    //class body
}

```

در مثال زیر به کلاسهای House و Apartment متدهایی اضافه شده است که در ک بهتری از ارث بری به شما می دهد.

```

1 public class House1 {
2     int no;
3     double surface;
4
5     void showHouse() {
6         System.out.println("House[" + no + ":" + surface+"]");

```

```

7         }
8     }
9
10    public class Apartment1 extends House1 {
11        int floor;
12
13        void showApartment() {
14            System.out.println("Apartment[" +
15                                no + ", "+
16                                floor+", "+
17                                surface+"]");
18        }
19        void showFloor(){
20            System.out.println("floor is: "+floor);
21        }
22    }
23 }
```

کد ۳

کد زیر ایجاد آبجکت‌هایی از `House` و `Apartment` و مقداردهی فیلدها و فراخوانی متدهای آنها را نشان می‌دهد.

```

1  public class SimpleInheritance {
2      public static void main(String args[]) {
3          House1 house = new House1();
4          Apartment1 apartment= new Apartment1();
5
6          // The superclass may be used by itself.
7          house.no = 10;
8          house.surface = 250;
9          house.showHouse();
10         System.out.println();
11
12         /* The subclass has access to all public members
13             of its superclass. */
14         apartment.no = 7;
15         apartment.floor = 2;
16         apartment.surface = 120;
17         apartment.showHouse();
```

ارث بری

```
18     apartment.showFloor();  
19     System.out.println();  
20     apartment.showApartment();  
21 }  
22 }
```

کد ۷-۴

از آنجائیکه کلاس Apartment کلاس House مشتق شده است هر آبجکتی که از کلاس Apartment ساخته می شود تمام متدها و فیلد های House را نیز دارد. کد فوق گواه این مدعاست، زیرا فیلدهای no و surface و متدهای showHouse و no تعریف و پیاده سازی شده اند در آبجکتهای کلاس Apartment قابل دسترسی و فراخوانی هستند. اینکه کلاس Apartment از کلاس House به ارث برده است هیچ تاثیری در عملکرد کلاس House نخواهد داشت. خروجی اجرای برنامه فوق به صورت زیر است:

```
House[10:250.0]
```

```
House[7:120.0]  
floor is: 2
```

```
Apartment[7, 2, 120.0]
```

کلاس فرزند می تواند پدر کلاس دیگری باشد و این سلسله تا هر سطحی ادامه پیدا کند اما هر کلاس فقط می تواند از یک کلاس ارث ببرد (مشتق شود).

توجه: هیچ کلاسی نمی تواند (مستقیم یا غیرمستقیم) از خودش ارث ببرد (از خودش مشتق شود). اگر چنین کدی پیاده سازی شود، کامپایلر خطای کامپایل می دهد.

حق دسترسی و ارث بری

اگرچه گفته می شود کلاس فرزند همه متدها و فیلد های کلاس پدر نیز را به ارث می برد ولی در واقع متدها و فیلد هایی از کلاس پدر که به صورت private تعریف شده اند به ارث برده نمی شوند. از مبحث شی گرایی به خاطر دارید که متدها یا فیلد هایی private فقط داخل همان کلاسی که تعریف شده اند قابل فراخوانی هستند. کلاس تغییر یافته House را ملاحظه کنید.

```
1 public class House2 {  
2     int no;
```

```

3     double surface;
4     private boolean hasGarden;
5
6     void showHouse() {
7         System.out.println("House in " + no +
8             " with surface: " + surface +
9             (hasGarden? " with garden":" has no garden")));
10    }
11 }
12
13 public class Apartment2 extends House2 {
14     int floor;
15
16     void showApartment() {
17         System.out.println("Apartment in " + no +
18             ", floor "+floor+
19             " with surface "+surface);
20    }
21     void showFloor(){
22         System.out.println("floor is: "+floor);
23    }
24 }
25

```

کد ۷-۵

در کلاس House یک مقدار بولی hasGarden به صورت private تعریف شده است که در کلاس فرزند یعنی Apartment و نه هیچ جای دیگری خارج از کلاس House قابل دسترس نخواهد بود.

مثالی دیگر

فصل پیش را به خاطر بیاورید که کلاس Box با سه فیلد به نامهای width، height و depth تعریف نمودیم حال فرض کنید بخواهیم Box جدیدی تعریف کنیم که علاوه بر سه فیلد دارای فیلد دیگری به نام weight هم باشد، برای داشتن چنین کلاسی می توانیم کلاس جدیدی با ۴ فیلد تعریف کنیم اما بر اساس آنچه از ارث بری خوانده اید می توان کلاس جدیدی ایجاد کرد که از کلاس Box مشتق شود و فقط حاوی فیلد اضافه weight باشد. کلاسهای BoxWeight و Box را در زیر ملاحظه می کنید.

```

1 public class Box {
2

```

```

3     double width;
4     double height;
5     double depth;
6
7     // construct clone of an object
8     Box(Box ob) { // pass object to constructor
9         width = ob.width;
10        height = ob.height;
11        depth = ob.depth;
12    }
13
14    // constructor used when all dimensions specified
15    Box(double w, double h, double d) {
16        width = w;
17        height = h;
18        depth = d;
19    }
20
21    // constructor used when no dimensions specified
22    Box() {
23        width = -1; // use -1 to indicate
24        height = -1; // an uninitialized
25        depth = -1; // box
26    }
27
28    // constructor used when cube is created
29    Box(double len) {
30        width = height = depth = len;
31    }
32
33    // compute and return volume
34    double volume() {
35        return width * height * depth;
36    }
37 }
```

V-T-S

```
1 public class BoxWeight extends Box {
```

```

1   double weight; // weight of box
2
3
4   // constructor for BoxWeight
5   BoxWeight(double w, double h, double d, double m) {
6       width = w;
7       height = h;
8       depth = d;
9       weight = m;
10      }
11  }

```

V-V دد

```

1  public class DemoBoxWeight {
2      public static void main(String args[]) {
3          BoxWeight mybox1 =
4              new BoxWeight(10, 20, 15, 34.3);
5          BoxWeight mybox2 =
6              new BoxWeight(2, 3, 4, 0.076);
7          double vol;
8
9          vol = mybox1.volume();
10         System.out.println("Volume of mybox1 is " + vol);
11         System.out.println("Weight of mybox1 is " +
12             mybox1.weight);
13         System.out.println();
14
15         vol = mybox2.volume();
16         System.out.println("Volume of mybox2 is " + vol);
17         System.out.println("Weight of mybox2 is " +
18             mybox2.weight);
19     }
20 }

```

V-V دد

خروجی اجرای برنامه فوق به صورت زیر خواهد بود:

```

volume of mybox1 is 3000.0
weight of mybox1 is 34.3

volume of mybox2 is 24.0
weight of mybox2 is 0.076

```

ارث بری

مهمترین فایده ارث بری در ساده سازی طراحی برنامه است در هر برنامه می توان رفتار مشترک مجموعه ای از کلاسها را در قالب یک کلاس عمومی طراحی کرد، سپس مجموعه کلاسها را از آن مشتق نمود. هریک از کلاسها یک کلاس مشتق می شوند، خود می توانند پدر مجموعه ای از کلاسها دیگر باشند و خصوصیات و رفتار مشترک فرزندان خود را در خود جمع کنند. برای نمونه، فرض کنید بخواهیم نوع دیگری از کلاس Box را تعریف کنیم که دارای فیلد اضافه ای به نام باشد همانند کلاس ColorBox، کلاس WeightBox را می توان از Box مشتق نمود و یک فیلد جدید به نام color در آن تعریف کرد.

```
1 public class ColorBox extends Box {  
2  
3     int color; // color of box  
4  
5     ColorBox(double w, double h, double d, int c) {  
6         width = w;  
7         height = h;  
8         depth = d;  
9         color = c;  
10    }  
11 }
```

کد ۷-۹

تبدیل آبجکتها به یکدیگر

در جوا این امکان وجود دارد که یک آبجکت را از کلاس پدر تعریف کنیم ولی از جنس کلاس فرزند ایجاد کنیم مثلا

```
Box box;  
box = new ColorBox();
```

کامپایلر جوا به ما اجازه می دهد تا آبجکت box را از جنس کلاس Box یا هر کلاسی که از کلاس Box مشتق شده است ایجاد کنیم. اگرچه آبجکتی که در زمان اجرا در حافظه ایجاد می شود از جنس ColorBox است اما در زمان توسعه و در خطوط بعدی برنامه، فقط به متدها و فیلدهایی که در کلاس Box تعریف شده اند می توان دسترسی پیدا نمود. هرگاه در یک برنامه، دو آبجکت از یک جنس تعریف شده باشند بدون توجه به اینکه از چه جنسی ایجاد شده اند، می توان مقدار یکی را به دیگری نسبت داد به جملات زیر توجه کنید:

```
Box b1 = new BoxWeight(3, 5, 7, 8.37);
Box b2 = new ColorBox(4, 5, 3, 2.21);
b1 = b2;
```

اما اگر دو آبجکت از دو نوع مختلف تعریف شده باشند -حتی اگر پدر یکسانی داشته باشند- انتساب مقدار یکی به دیگری باعث بروز خطای کامپایل خواهد شد. جملات زیر نادرست بوده و کامپایل نخواهد شد:

```
BoxWeight b1 = new BoxWeight(3, 5, 7, 8.37);
ColorBox b2 = new ColorBox(4, 5, 3, 2.21);
b1 = b2; //compile error
```

باید دقت کنید که آبجکت فرزند، آبجکتی از پدر محاسب می شود اما آبجکت پدر آبجکتی از جنس فرزند نیست. این بدین معناست که آبجکت فرزند را می توان به آبجکتی از جنس پدر انتساب داد در حالیکه آبجکت پدر را نمی توان به آبجکتی از فرزند انتساب داد. با این توضیح، تکه کد زیر صحیح است.

```
BoxWeight b1 = new BoxWeight(3, 5, 7, 8.37);
Box b2 = new Box(3, 5, 7);
b2 = b1; //correct!
```

اما کد زیر نادرست است و باعث خطای کامپایل می شود.

```
BoxWeight b1 = new BoxWeight(3, 5, 7, 8.37);
Box b2 = new Box(3, 5, 7);
b1 = b2; //compile error!
```

البته اگر مطمئن هستید که در زمان اجرا، آبجکتهاي ایجاد شده در حافظه قابل انتساب هستند، جاوا راه انتساب را برای شما بازگذارده است. مثلا آبجکت b2 که از جنس Box است را می توانید به صورت زیر به آبجکت b1 که از جنس BoxWeight است انتساب دهید.

```
BoxWeight b1 = new BoxWeight(3, 5, 7, 8.37);
Box b2 = new BoxWeight(4, 2, 4, 7.17);
b1 = (BoxWeight) b2; //correct
```

توجه داشته باشید که اگرچه آبجکت b2 از کلاس Box تعریف شده اما در حافظه از کلاس BoxWeight ایجاد شده است به همین دلیل بدون توجه به وضعیت زمان اجرا (که انتساب آنها امکانپذیر است) کامپایلر جاوا مانع انتساب مقدار b2 به b1 می شود. در چنین وضعیتی همانطور که مشخص است از قبل از b2 استفاده شده است که به آن cast کردن گفته می شود.

توجه کنید که حتی اگر آبجکت b2 از جنس BoxWeight ایجاد نشده باشد باز هم با `cast` کردن می توان مقادار آنرا به b1 انتساب داد و مانع خطای کامپایلر شد اما در زمان اجرا با خطا مواجه می شوید. کد زیر این موضوع را نشان می دهد.

```
BoxWeight b1 = new BoxWeight(3, 5, 7, 8.37);
Box b2 = new ColorWeight(4, 2, 4, 7.17);
b1 = (BoxWeight) b2; //runtime error!
```

توجه: یک آبجکت فرزند را می توان به تمام آبجکتها یی که از جنس پدر خود تعریف شده اند انتساب داد اما یک آبجکت پدر را تنها زمانی می توان به آبجکت فرزند انتساب داد که از جنس همان آبجکت فرزند یا فرزندان آن آبجکت فرزند ساخته شده باشد

استفاده از متدها و فیلدهای کلاس پدر در کلاس فرزند

وقتی آبجکتی از کلاس فرزند ساخته می شود می توانید تمام متدها و فیلدهایی را که در آن کلاس یا کلاس پدر آن به صورت `public` تعریف شده اند را فراخوانی کنید در مورد متدها و فیلدهای `private` مسئله واضح است (هیچ یک از آنها را نمی توان فراخوانی کرد) اما در مورد متدها و فیلدهای `protected` چطور؟ به مثال زیر که استفاده از کلاسهای `Box` و `BoxWeight` را نشان می دهد دقต کنید:

```
1 public class RefDemo {
2     public static void main(String args[]) {
3         BoxWeight weightbox =
4             new BoxWeight(3, 5, 7, 8.37);
5         Box plainbox = new Box();
6         double vol;
7
8         vol = weightbox.volume();
9         System.out.println("Volume of weightbox is " +
10             vol);
11        System.out.println("Weight of weightbox is " +
12             weightbox.weight());
13        System.out.println();
14
15        // assign BoxWeight reference to Box reference
16        plainbox = weightbox;
```

```

17
18     // OK, volume() defined in Box
19     vol = plainbox.volume();
20     System.out.println("Volume of plainbox is " +
21         vol);
22
23     /* The following statement is invalid because
24     plainbox does not define a weight member. */
25
26     //System.out.println("Weight of plainbox is "
27     // + plainbox.weight);
28 }
29 }
```

کد ۷-۱۰

کلمه کلیدی super

کلاس BoxWieght که در قسمت قبل ملاحظه کردید دارای اشکالاتی است برای مثال در سازنده این کلاس، فیلد های کلاس پدر یعنی `width`, `height` و `depth` مقداردهی شده اند بر اساس طراحی شی گرایی، بهتر است فیلد های همان کلاس توسط همان کلاس مقداردهی شود. اگر فرزند یک کلاس می خواهد فیلد های پدر خود را مقداردهی کند بهتر است اینکار را از طریق سازنده کلاس پدر یا یکی از متدهای کلاس پدر انجام دهید. اینکار یک فایده مهم دارد، اگر بخواهید نحوه مقداردهی را تغییر دهید هر دو کلاس را باید تغییر دهید دومین اشکال این کلاس این است که در اغلب موارد، فیلد های کلاس پدر به صورت `private` تعریف می شوند و بنابراین کلاس فرزند به آنها دسترسی ندارد (در این مثال، فیلد های کلاس پدر به صورت `private` تعریف نشده اند).

به هر دو دلیل فوق، بهتر است مقدار دهی فیلد های همان کلاس توسط یکی از متدهای تعریف شده در همان کلاس صورت گیرد به این ترتیب اگر بخواهیم فیلد های یک کلاس را در کلاس فرزند یا هرجای دیگری بیرون کلاس مقداردهی کنیم، می توان متدهای تعریف شده در کلاس را فراخوانی نمود.

برای فراخوانی یکی از متدهای کلاس پدر یا دسترسی به یکی از فیلد های کلاس پدر، از کلمه کلیدی `super` استفاده می شود از کلمه کلیدی `super` همچنین برای فراخوانی سازنده کلاس پدر از درون کلاس فرزند نیز استفاده می شود.

استفاده از `super` برای فراخوانی سازنده کلاس پدر

کلاس فرزند می تواند با استفاده از دستور زیر سازنده کلاس پدر را فراخوانی کند:

`super(parameter-list)`

parameter-list پارامترهایی هستند که در سازنده کلاس پدر وجود دارند فراخوانی سازنده کلاس پدر همیشه باید اولین جمله سازنده کلاس فرزند باشد.

برای اینکه بینند چگونه باید از `super` برای فراخوانی سازنده کلاس پدر استفاده نمود، کلاس `TutorialBox` را که در زیر نشان داده شده است ملاحظه کنید:

```

1 public class BoxWeight1 extends Box {
2
3     double weight; // weight of box
4
5     // initialize width, height, and depth using super()
6     BoxWeight1(double w, double h, double d, double m) {
7         super(w, h, d); // call superclass constructor
8         weight = m;
9     }
10 }
```

کد ۷-۱۱

همانطور که ملاحظه می کنید در سازنده کلاس فوق، با استفاده از کلمه `super` سازنده کلاس پدر (کلاس `Box`) فراخوانی شده است و مقادیر عددی `w`, `h` و `d` به آن ارسال شده اند به این ترتیب فیلدهای `width`, `height` و `depth` که توسعه سازنده کلاس پدر (که با جمله `super(w, h, d)` فراخوانی شده است) مقدار دهی شده اند.

بدین ترتیب برای مقداردهی فیلدهای کلاس پدر، بدون دسترسی مستقیم به هر یک از فیلدها از سازنده کلاس پدر استفاده شده است. از آنجائیکه کلاس پدر ممکن است دارای چندین سازنده مختلف باشد جواوا از طریق پارامترهایی که داخل `()` مشخص می کنید تشخیص می دهد که قصد فراخوانی کدامیک از سازنده ها را دارد. برای مثال کلاس `BoxWeight` در زیر آمده است در این کلاس سازنده های مختلفی قرار دارند که بالطبع سازنده های مختلفی از کلاس پدر را نیز فراخوانی کرده اند.

```

1 public class Box1 {
2     private double width;
3     private double height;
4     private double depth;
5
6     // construct clone of an object
7     Box1(Box1 ob) { // pass object to constructor }
```

```

8         width = ob.width;
9         height = ob.height;
10        depth = ob.depth;
11    }
12
13    // constructor used when all dimensions specified
14    Box1(double w, double h, double d) {
15        width = w;
16        height = h;
17        depth = d;
18    }
19
20    // constructor used when no dimensions specified
21    Box1() {
22        width = -1; // use -1 to indicate
23        height = -1; // an uninitialized
24        depth = -1; // Box1
25    }
26
27    // constructor used when cube is created
28    Box1(double len) {
29        width = height = depth = len;
30    }
31
32    // compute and return volume
33    double volume() {
34        return width * height * depth;
35    }
36
37 }
```

نحوه ایجاد

```

1 public class BoxWeight2 extends Box1 {
2     double weight; // weight of box
3
4     // construct clone of an object
5     // pass object to constructor
6     BoxWeight2(BoxWeight2 ob) {
```

```

7         super(ob);
8         weight = ob.weight;
9     }
10
11    // constructor when all parameters are specified
12    BoxWeight2(double w, double h, double d, double m) {
13        super(w, h, d); // call superclass constructor
14        weight = m;
15    }
16
17    // default constructor
18    BoxWeight2() {
19        super();
20        weight = -1;
21    }
22
23    // constructor used when cube is created
24    BoxWeight2(double len, double m) {
25        super(len);
26        weight = m;
27    }
28 }
```

V-IPR دک

```

1 public class DemoSuper {
2     public static void main(String args[]) {
3         BoxWeight2 mybox1 =
4             new BoxWeight2(10, 20, 15, 34.3);
5
6         BoxWeight2 mybox2 =
7             new BoxWeight2(2, 3, 4, 0.076);
8
9         BoxWeight2 mybox3 =
10            new BoxWeight2(); // default
11
12         BoxWeight2 mycube = new BoxWeight2(3, 2);
13         BoxWeight2 myclone = new BoxWeight2(mybox1);
14 }
```

```

15     double vol;
16
17     vol = mybox1.volume();
18     System.out.println("Volume of mybox1 is " +
19             vol);
20     System.out.println("Weight of mybox1 is " +
21             mybox1.weight);
22     System.out.println();
23
24     vol = mybox2.volume();
25     System.out.println("Volume of mybox2 is " +
26             vol);
27     System.out.println("Weight of mybox2 is " +
28             mybox2.weight);
29     System.out.println();
30
31     vol = mybox3.volume();
32     System.out.println("Volume of mybox3 is " +
33             vol);
34     System.out.println("Weight of mybox3 is " +
35             mybox3.weight);
36     System.out.println();
37
38     vol = myclone.volume();
39     System.out.println("Volume of myclone is " +
40             vol);
41     System.out.println("Weight of myclone is " +
42             myclone.weight);
43     System.out.println();
44
45     vol = mycube.volume();
46     System.out.println("Volume of mycube is " +
47             vol);
48     System.out.println("Weight of mycube is " +
49             mycube.weight);
50     System.out.println();
51 }
52 }
```

به سازنده زیر که در کلاس BoxWeight تعریف شده است کاملاً دقت کنید:

```
// construct clone of an object
BoxWeight(BoxWeight ob) { // pass object to constructor
    super(ob);
    weight = ob.weight;
}
```

توجه کنید که سازنده کلاس پدر با پارامتری از کلاس BoxWieght (به جای کلاس Box) فراخوانی شده است از آنجاییکه BoxWeight از جنس Box محسوب می شود هر متدهی که پارامتری از جنس Box دریافت می کند می توان هر یک از فرزندان Box را ارسال کرد.

توجه: اگر در سازنده کلاس فرزند، هیچ یک از سازنده های کلاس پدر فراخوانی نشود، سازنده پیش فرض کلاس پدر فراخوانی می شود.

استفاده از super برای فراخوانی متدها یا فیلد های کلاس پدر

super بسیار شبیه this است در صورتیکه بخواهید در کلاس فرزند یکی از متدها یا فیلد های کلاس پدر را فراخوانی کنید (خصوصا در مواردی که متدها یا فیلد های کلاس پدر و فرزند مشابه است این ممکن است باشد) باید از کلمه super که شکل کلی استفاده از آن به صورت زیر است استفاده کنید:

```
super.member
```

که در اینجا member نام فیلد یا متدهای کلاس پدر است. مثال زیر بکارگیری super را نشان می دهد

```
1 public class A {
2     int i;
3 }
```

کد ۷-۱۵

```
1 public class B extends A {
2
3     int i; // this i hides the i in A
4
5     B(int a, int b) {
6         super.i = a; // i in A
7         i = b; // i in B
8     }
9 }
```

```

10     void show() {
11         System.out.println("i in superclass: " +
12             super.i);
13         System.out.println("i in subclass: " + i);
14     }
15 }
```

کد ۷-۱۶

```

1 public class UseSuper {
2     public static void main(String args[]) {
3         B subOb = new B(1, 2);
4
5         subOb.show();
6     }
}
```

کد ۷-۱۷

اجرای برنامه فوق به صورت زیر خواهد بود.

```
i in superclass: 1
i in subclass: 2
```

همانطور که ملاحظه می کنید یک متغیر `i` در کلاس A و یک متغیر `A` در کلاس B تعریف شده است در کلاس B برای دستیابی به `A` که در کلاس A قرار دارد از کلمه `super` استفاده شده است.

ایجاد کلاس‌های سلسله مراتبی

تا این لحظه فقط کلاس‌هایی داشته ایم که در آن یک کلاس پدر و یک کلاس فرزند تعریف شده بودند. اما می توان کلاس‌هایی تعریف کرد که در آن کلاس‌های متعددی به صورت سلسله مراتبی از یکدیگر مشتق شوند به عنوان مثال سه کلاس A، B و C را تصور کنید که در آن کلاس A از کلاس B و کلاس C از کلاس B مشتق شده است وقتی یک کلاس از کلاس دیگری مشتق می شود تمام خصوصیات و رفتار کلاس پدر و اجداد خود را نیز به ارث می برد در مورد مثالی که ذکر شد، کلاس C حاوی تمام خصوصیات کلاس‌های A و B می باشد. به عنوانی مثالی دیگر، کلاس Shipment که از کلاس BoxWeight مشتق شده است را در نظر بگیرید کلاس Shipment تمام خصوصیات کلاس‌های Box و BoxWeight را به ارث می برد و خود دارای یک `cost` است که مشخص کننده هزینه فروش هر نوع است. در زیر کدهای مربوط به BoxWeight و Shipment را ملاحظه می کنید.

```
1 // Extend BoxWeight to include shipping costs.
2
3 // Start with Box.
4
5 class Box {
6     private double width;
7     private double height;
8     private double depth;
9
10    // construct clone of an object
11    Box(Box ob) { // pass object to constructor
12        width = ob.width;
13        height = ob.height;
14        depth = ob.depth;
15    }
16
17    // constructor used when all dimensions specified
18    Box(double w, double h, double d) {
19        width = w;
20        height = h;
21        depth = d;
22    }
23
24    // constructor used when no dimensions specified
25    Box() {
26        width = -1; // use -1 to indicate
27        height = -1; // an uninitialized
28        depth = -1; // box
29    }
30
31    // constructor used when cube is created
32    Box(double len) {
33        width = height = depth = len;
34    }
35
36    // compute and return volume
37    double volume() {
38        return width * height * depth;
```

```

39     }
40 }
```

۶-۱۸ کد

```

1 // Add weight.
2
3 class BoxWeight extends Box {
4     double weight; // weight of box
5
6     // construct clone of an object
7     // pass object to constructor
8     BoxWeight(BoxWeight ob) {
9         super(ob);
10        weight = ob.weight;
11    }
12
13    // constructor when all parameters are specified
14    BoxWeight(double w, double h, double d, double m) {
15        super(w, h, d); // call superclass constructor
16        weight = m;
17    }
18
19    // default constructor
20    BoxWeight() {
21        super();
22        weight = -1;
23    }
24
25    // constructor used when cube is created
26    BoxWeight(double len, double m) {
27        super(len);
28        weight = m;
29    }
30 }
```

۶-۱۹ کد

ارث بری

```
1 public class Shipment extends BoxWeight2 {  
2     double cost;  
3  
4     // construct clone of an object  
5     Shipment(Shipment ob) { // pass object to constructor  
6         super(ob);  
7         cost = ob.cost;  
8     }  
9  
10    // constructor when all parameters are specified  
11    Shipment(double w, double h, double d,  
12              double m, double c) {  
13        super(w, h, d, m); // call superclass constructor  
14        cost = c;  
15    }  
16  
17    // default constructor  
18    Shipment() {  
19        super();  
20        cost = -1;  
21    }  
22  
23    // constructor used when cube is created  
24    Shipment(double len, double m, double c) {  
25        super(len, m);  
26        cost = c;  
27    }  
28 }
```

۷-۱۰-۲۸

```
1 public class DemoShipment {  
2  
3     public static void main(String args[]) {  
4  
5         Shipment shipment1 =  
6             new Shipment(10, 20, 15, 10, 3.41);  
7         Shipment shipment2 =  
8             new Shipment(2, 3, 4, 0.76, 1.28);
```

۷-۱۰-۲۹

```

9
10    double vol;
11
12    vol = shipment1.volume();
13    System.out.println("Volume of shipment1 is " +
14        vol);
15    System.out.println("Weight of shipment1 is " +
16        + shipment1.weight);
17    System.out.println("Shipping cost: $" +
18        shipment1.cost);
19    System.out.println();
20
21    vol = shipment2.volume();
22    System.out.println("Volume of shipment2 is " +
23        vol);
24    System.out.println("Weight of shipment2 is " +
25        + shipment2.weight);
26    System.out.println("Shipping cost: $" +
27        shipment2.cost);
28}
29}

```

۷-۲۱ کد

خروجی اجرای برنامه فوق بصورت زیر خواهد بود:

```
Volume of shipment1 is 3000.0
Weight of shipment1 is 10.0
Shipping cost: $3.41
```

```
Volume of shipment2 is 24.0
Weight of shipment2 is 0.76
Shipping cost: $1.28
```

این مثال حاوی یک نکته بسیار مهم است و آن این که در هر کلاس می توانید با استفاده از جمله `super()` سازنده نزدیکترین کلاس پدر را فراخونی کنید از آنجاییکه در این مثال کلاس `Shipment` از کلاس `BoxWeight` مشتق شده است فراخوانی جمله `super()` در کلاس `Shipment` باعث فراخوانی سازنده کلاس `BoxWeight` می شود.

سازنده کلاس پدر چه زمانی فراخوانی می شود؟

سوال اینجاست که وقتی از روی یک کلاس آبجکتی ساخته می شود ترتیب فراخوانی سازنده های کلاس ها به چه شکل است مثلا فرض کنید که کلاس A از کلاس B مشتق شده باشد در اینصورت وقتی از کلاس B یک آبجکت ساخته می شود ترتیب فراخوانی سازنده های کلاس ها به چه شکل است؟

جواب این سوال این است که سازنده های کلاس ها به ترتیب از کلاس پدر به کلاس فرزند فراخوانی می شوند به همین دلیل هم هست که () super باید اولین دستور در سازنده کلاس فرزند باشد. در صورتیکه جمله () را در سازنده کلاس فرزند بکار نبرید، به صورت پیش فرض سازنده کلاس پدر که هیچ پارامتری ندارد (سازنده پیش فرض) فراخوانی می شود. مجموعه کلاسهای زیر ترتیب فراخوانی سازنده های کلاسهای مشتق شده را نشان می دهد.

```

1 public class A {
2     A() {
3         System.out.println("Inside A's constructor.");
4     }
5 }
```

کد ۷-۲۲

```

1 public class B extends A {
2     B() {
3         System.out.println("Inside B's constructor.");
4     }
5 }
```

کد ۷-۲۳

```

1 public class C extends B {
2     C() {
3         System.out.println("Inside C's constructor.");
4     }
5 }
```

کد ۷-۲۴

```

1 public class CallingCons {
```

```

2     public static void main(String args[]) {
3         C c = new C();
4     }
5 }
```

کد ۷-۳۵

خروجی اجرای برنامه فوق به صورت زیر خواهد بود.

```

Inside A's constructor.
Inside B's constructor.
Inside C's constructor.
```

همانطور که ملاحظه می کنید ترتیب فرآخوانی سازنده های کلاس ها، به همان ترتیبی است که مشتق شده اند.

مثال کاربردی ۱

فرض کنید می خواهیم برنامه ای بنویسیم که در آن مجموعه ای از اشکال هندسی تعریف و محاسباتی روی آنها انجام می شود در این برنامه اشکال هندسی همچون نقطه، دایره، استوانه، کره، مربع، و مکعب شبیه سازی و استفاده می شوند.

بالطبع در پیاده سازی یک چنین برنامه ای کلاسهای خواهیم داشت که هریک معرف یکی از اشکال هندسی هستند در این برنامه، کلاس Point معرف نقطه، کلاس Circle معرف دایره، کلاس Cylinder معرف استوانه، کلاس Square معرف مربع، و کلاس Cube معرف مکعب خواهد بود.

در ادامه این مسئله کلاسهای Point، Circle، و Cylinder را پیاده سازی خواهیم کرد و بقیه را به عهده خواننده می گذاریم.

هر نقطه در محورهای مختصات با دو مقدار طول و عرض مشخص می شود بنابراین می توان کلاس Point را با دو فیلد، x و y به صورت زیر در نظر گرفت.

```

1  public class Point{
2      int x ;
3      int y ;
4 }
```

کد ۷-۳۶

هر دایره نیز با مختصات مرکز و اندازه شعاع مشخص می شود، بنابراین می توان در کلاس Circle، سه فیلد به صورت زیر در نظر گرفت:

```
1 public class Circle {  
2     int x;  
3     int y;  
4     int radius;  
5 }
```

کد ۷-۳۷

با دقت در کلاس فوق، به این نتیجه می‌رسیم که `x` و `y` مشخص کننده یک نقطه است بنابراین می‌توان به جای تعریف `x` و `y` داخل کلاس `Circle` که معرف مختصات مرکز دایره هستند، یک فیلد از جنس `Point` تعریف نمود.

```
1 public class Circle1 {  
2     Point center;  
3     int radius;  
4 }
```

کد ۷-۳۸

به این خصوصیت، که یکی از فیلدهای یک کلاس از جنس کلاس دیگر است **Composition** گفته می‌شود.

اما به جای تعریف یک فیلد از جنس یک کلاس دیگر، می‌توان کلاس `Circle` را از کلاس `Point` مشتق کرد بدین ترتیب فیلدهای کلاس `Point` به کلاس `Circle` ارث برده می‌شوند.

```
1 public class Circle2 extends Point{  
2     int radius;  
3 }
```

کد ۷-۳۹

اکنون که کلاس‌های `Point` و `Circle` طراحی شده‌اند می‌توان بقیه متدهای این کلاسها را پیاده سازی نمود.

```
1 public class Point{  
2     private int x ;  
3     private int y ;  
4  
5     //construct a Point object by 0, 0 values  
6     public Point() {
```

```

7         setPosition(0, 0);
8     }
9
10    //construct a Point object by x, y values
11    public Point(int x, int y) {
12        setPosition(x, y);
13    }
14
15    //set new value to x, y
16    public void setPosition(int x, int y){
17        this.x = x;
18        this.y = y;
19    }
20
21    //construct a Point object by another Point
22    public Point(Point p){
23        setPosition(p.getX(), p.getY());
24    }
25
26    //getter methods
27    public int getX() {
28        return x;
29    }
30
31    public int getY() {
32        return y;
33    }
34
35    //setter methods
36    public void setX(int x) {
37        this.x = x;
38    }
39
40    public void setY(int y) {
41        this.y = y;
42    }
43
44    public String toString() {
45        return "["+x+","+y+"]";
46    }

```

```
1 public class Circle2 extends Point{
2     int radius;
3     public Circle2() {
4         super(0, 0);
5         this.radius = 0;
6     }
7
8     public Circle2(int radius) {
9         super(0, 0);
10        this.radius = radius;
11    }
12
13    public Circle2(int x, int y, int radius) {
14        super(x, y);
15        this.radius = radius;
16    }
17
18    public Circle2(Point center, int radius){
19        super(center);
20        this.radius = radius;
21    }
22
23    //getter method
24    public int getRadius() {
25        return radius;
26    }
27
28    //setter method
29    public void setRadius(int radius) {
30        this.radius = radius;
31    }
32
33    public String toString() {
34        return "("+radius+","+super.toString()+")";
35    }
```

```

36     public double getArea() {
37         return Math.PI * Math.pow(radius, 2);
38     }
39 }
```

کد ۷-۳۱

در کلاس Circle متدی به نام getArea() تعریف شده است که مساحت دایره را محاسبه و برمی گرداند، یک مقدار static Math.PI است که مقدار آن عدد پی (3.14159...) است همچنین متد pow() که در کلاس Math قرار دارد حاصل یک عدد به توان عدد دیگر را محاسبه کرده و برمی گرداند این دو عدد به صورت پارامتر به این متد ارسال می شوند. کلاس زیر بکارگیری این دو کلاس را نشان می دهد.

```

1 public class UsingCirclePoint {
2     public static void main(String[] args) {
3
4         Point p ;
5         Circle2 c1;
6         Circle2 c2;
7
8         p = new Point(5, 4);
9         c1 = new Circle2(2, 3, 2);
10        c2 = new Circle2(p, 2);
11
12        System.out.println("P: "+p);
13        System.out.println("C1: "+c1);
14        System.out.println("C2: "+c2);
15
16        System.out.println("Area of c1: "+c1.getArea());
17        System.out.println("Area of c2: "+c2.getArea());
18
19    }
20 }
```

کد ۷-۳۲

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود:

```
P: [5,4]
C1: (2,[2,3])
```

ارث بری

```
C2: (2,[5,4])
Area of c1: 12.566370614359172
Area of c2: 12.566370614359172
```

تمرین: تمام متدهای دو کلاس `Circle` و `Point` را یک به یک بررسی کرده و عملکرد هر کدام را به طور کامل درک کنید.

Overriding

فرض کنید که کلاس `Child` از کلاس `Parent` مشتق شده باشد اگر در هر دو کلاس `Child` و `Parent` متدهای با نام و پارامترهای یکسان پیاده سازی شده باشد گفته می شود که کلاس `Child` آن `override` شده کلاس `Parent` را `override` کرده است. اگر در کلاس فرزند بخواهیم متدهای کلاس `Parent` را `override` کنیم، باید قبل از نام متدهای `super` استفاده کنیم در صورتیکه قبل از نام متدهای `super` را مشخص نکنید در واقع متدهای کلاس فرزند را فراخوانی کرده اید. مثال زیر را در نظر بگیرید.

```
1 public class Parent {
2     int i, j;
3
4     Parent(int a, int b) {
5         i = a;
6         j = b;
7     }
8
9     // display i and j
10    void show() {
11        System.out.println("i and j: " + i + " " + j);
12    }
13 }
```

کد

```
1 public class Child extends Parent {
2     int k;
3
4     Child(int a, int b, int c) {
5         super(a, b);
6         k = c;
7     }
8 }
```

```

7     }
8
9     // display k -- this overrides show() in A
10    void show() {
11        System.out.println("k: " + k);
12    }
13 }
```

V-۳۴ کد

```

1 public class OverrideTest {
2     public static void main(String args[]) {
3         Child subOb = new Child(1, 2, 3);
4
5         subOb.show(); // this calls show() in B
6     }
7 }
```

V-۳۵ کد

خروجی اجرای برنامه فوق به صورت زیر خواهد بود.

k: 3

در این مثال متدهای show() که در کلاس Parent قرار دارد توسط کلاس Child **override** شده است همانطور که گفته شد در صورتیکه در کلاس Child بخواهید متدهای show() مربوط به کلاس Parent را فراخوانی کنید باید از کلمه **super** استفاده کنید.

```

1 public class Child extends Parent {
2     int k;
3
4     Child(int a, int b, int c) {
5         super(a, b);
6         k = c;
7     }
8
9     // display k -- this overrides show() in A
10    void show() {
```

ارث بری

```
11     super.show(); // this calls A's show()
12     System.out.println("k: " + k);
13 }
14 }
```

کد ۷-۳۶

در صورتیکه کلاس Child جدید را با کلاس Child قبلی جایگزین کنید با اجرای کلاس Override نتیجه زیر را به همراه خواهد داشت.

```
i and j: 1 2
k: 3
```

در اینجا متده است که در کلاس فرزند قرار دارد متده show() مربوط به کلاس پدر را با استفاده از کلمه super فراخوانی کرده است. متده override کردن تنها وقتی اتفاق می افتد که متده که در کلاس فرزند وجود دارد کاملا از لحاظ نام و پارامترها با متده موجود در کلاس پدر یکسان باشند اگر پارامترهای دو متده مشابه نباشند می توان گفت که دو متده overload شده اند. برای مثال نسخه تغییر یافته کلاسهای Parent و Child را به صورت زیر ملاحظه کنید.

```
1 public class Parent1 {
2     int i, j;
3
4     Parent1(int a, int b) {
5         i = a;
6         j = b;
7     }
8
9     // display i and j
10    void show() {
11        System.out.println("i and j: " + i + " " + j);
12    }
13 }
```

کد ۷-۳۷

```
1 public class Child1 extends Parent {
2     int k;
```

```

3
4     Child1(int a, int b, int c) {
5         super(a, b);
6         k = c;
7     }
8
9     // overload show()
10    void show(String msg) {
11        System.out.println(msg + k);
12    }
13 }
```

۷-۳۸ کد

```

1 public class OverrideTest1 {
2     public static void main(String args[]) {
3         Child1 subOb = new Child1(1, 2, 3);
4
5         // this calls show() in B
6         subOb.show("This is k: ");
7
8         // this calls show() in A
9         subOb.show();
10    }
11 }
```

۷-۳۹ کد

خروجی اجرای برنامه فوق به صورت زیر است.

```
This is k: 3
i and j: 1 2
```

متد (`show()`) در کلاس B وجود دارد برخلاف نسخه ای که در A وجود دارد یک رشته نیز دریافت می کند و این خود دلیل کافی برای `override` نشدن این متد توسط کلاس B خواهد بود.

کاربرد overriding

با استفاده از `overriding` می توانید دو کلاس داشته باشید که از یک کلاس مشتق شده اند هر دو کلاس فرزند، یکی از متدهای کلاس پدر را `override` کرده اند بنابراین در عین حال که هر دو از یک جنس

ارث بری

هستند (هر دو کلاس از جنس کلاس پدر هستند) ولی یکی از متدهای کلاس پدر خود را تغییر داده اند (یکی از متدهای کلاس پدر را به ارث نبرده اند) به چنین خصوصیتی اصطلاحاً پلی مورفیسم گفته می شود. به عنوان مثال برنامه زیر را تصور کنید در این برنامه یک کلاس به نام `Figure` طراحی شده است که مشخص کننده یک شکل هندسی دو بعدی است `dim1` و `dim2` مشخص کننده ابعاد این شکل هستند سپس در این کلاس متدهی `area()` تعریف شده است که مساحت شکل مشخص شده توسط `Figure` را محاسبه می کند.

دو کلاس `Triangle` و `Rectangle` که مشخص کننده مستطیل و مثلث هستند از این کلاس مشتق شده اند و هر دوی آنها متدهی `area()` را که در کلاس `Figure` وجود دارد `override` کرده اند. در زیر کدهای مربوط به این دو کلاس را مشاهده می کنید.

```
1 public class Figure {
2     double dim1;
3     double dim2;
4
5     Figure(double a, double b) {
6         dim1 = a;
7         dim2 = b;
8     }
9
10    double area() {
11        System.out.println("Area for Figure is undefined.");
12        return 0;
13    }
14 }
```

کد ۷-۴۰

```
1 public class Rectangle extends Figure {
2     Rectangle(double a, double b) {
3         super(a, b);
4     }
5
6     // override area for rectangle
7     double area() {
8         System.out.println("Inside Area for Rectangle.");
9         return dim1 * dim2;
10    }
```

```
11 }
```

V-FI کد

```
1 public class Triangle extends Figure {
2     Triangle(double a, double b) {
3         super(a, b);
4     }
5
6     // override area for right triangle
7     double area() {
8         System.out.println("Inside Area for Triangle.");
9         return dim1 * dim2 / 2;
10    }
11 }
```

V-FM کد

```
1 public class FindAreas {
2     public static void main(String args[]) {
3         Figure f = new Figure(10, 10);
4         Rectangle r = new Rectangle(9, 5);
5         Triangle t = new Triangle(10, 8);
6
7         Figure figref;
8
9         figref = r;
10        System.out.println("Area is " + figref.area());
11
12        figref = t;
13        System.out.println("Area is " + figref.area());
14
15        figref = f;
16        System.out.println("Area is " + figref.area());
17    }
18 }
```

V-FM کد

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود:

```
Inside Area for Rectangle.  
Area is 45.0  
Inside Area for Triangle.  
Area is 40.0  
Area for Figure is undefined.  
Area is 0.0
```

کلاس‌های abstract

فرض کنید قرار است برنامه‌ای برای بانک بنویسید، با مطالعه مستندات تحلیل به این نتیجه می‌رسید در آن بانک دو نوع حساب پس انداز وجود دارد حساب «پس انداز کوتاه مدت» و حساب «پس انداز بلند مدت». پس به این نتیجه می‌رسید که باید دو کلاس مختلف که هریک معرف یکی از این حسابهاست را طراحی و پیاده سازی کنید.

کلاس معرف حساب پس انداز کوتاه مدت به صورت زیر خواهد بود.

```
1 import java.util.Date;  
2  
3 public class ShortTimeAccount{  
4     String accountNumber;  
5     Date openDate;  
6     Double amount;  
7  
8     public double benefit(){  
9         return amount*0.12;  
10    }  
11 }
```

۷-۴۴ کد

و کلاس معرف حساب پس انداز بلند مدت، شکلی به صورت زیر خواهد داشت.

```
1 import java.util.Date;  
2  
3 public class LongTimeAccount{  
4     String accountNumber;  
5     Date openDate;  
6     Double amount;  
7  
8     public double benefit(){
```

```

9     return amount*0.16;
10    }
11 }
```

V-۴۵ کد

این دو کلاس شباهت خیلی زیادی با هم دارند و هر دو نیز معرف یک دسته شیء هستند در چنین موقعی براساس اصول طراحی شی گرایی لازم است کلاس پایه ای مثلًا با نام Account ایجاد کنید و خصوصیات رفتار مشترک هر دو کلاس فوق را به آن کلاس منتقل کنید.

```

1 import java.util.Date;
2
3 public class Account{
4     String accountNumber;
5     Date openDate;
6     Double amount;
7 }
8
9 public class ShortTimeAccount1 extends Account{
10    public double benefit(){
11        return amount*0.12;
12    }
13 }
14
15 public class LongTimeAccount1 extends Account{
16    public double benefit(){
17        return amount*0.16;
18    }
19 }
```

V-۴۶ کد

مزیت این کار در این است که هر تغییری در کلاس Account مثلاً اضافه کردن یک فیلد یا حذف یک فیلد روی هر دو کلاسی که از آن مشتق شده اند تاثیر می‌گذارد بدون اینکه نیاز باشد تا تغییرات برنامه را در هر دو کلاس اعمال کنیم.

هر دو کلاس ShortTimeAccount1 و LongTimeAccount1 دارای متد benefit() با هستند. ما بسیار تمایل داریم که این متد را نیز به کلاس پایه Account منتقل کنیم زیرا در آن صورت وقتی آبجکتی از Account به صورت زیر تعریف می‌کنیم

```
Account a = ...
```

ارث بری

```
a.benefit();
```

بدون توجه به اینکه آبجکت a از جنس چه کلاسی در حافظه ایجاد شود می توانیم متدها را از آن فراخوانی کنیم اما اگر متدهای benefit() در کلاس Account تعریف نشود فراخوانی این متدها آبجکتهاست امکان پذیر نخواهد بود اما از طرف دیگر ما نمی توانیم متدهای benefit() را از کلاسهای Account حذف کنیم و به کلاس LongTimeAccount1 و ShortTimeAccount1 منتقل کنیم زیرا بدنه این متدها در هریک از دو کلاس متفاوت است.

از طرف دیگر ما هیچ آبجکتی در برنامه نداریم که از کلاس Account ساخته شود یعنی آبجکتهاست که معرف حساب بانکی هستند از یکی از دو کلاس LongTimeAccount1 یا ShortTimeAccount1 یا ایجاد می شود. همه اینها را به یک نتیجه می رساند، کلاس Account فقط به ضرورت طراحی شی گرایی ایجاد شده و معادل خارجی ندارد به چنین کلاسی کلاس abstract یا انتزاعی یا غیرواقعی گفته می شود و با استفاده از کلمه کلیدی abstract مشخص می شود.

```
1 import java.util.Date;
2
3 public abstract class Account1{
4     String accountNumber;
5     Date openDate;
6     Double amount;
7 }
```

۷-۴۷

اضافه شدن کلمه abstract در مقابل نام کلاس Account مانع از ساخته شدن آبجکت از روی این کلاس می شود:

```
Account1 account = new Account1(); //error
```

اما می توان آبجکتهاست به صورت زیر ایجاد نمود

```
Account account = new ShortTimeAccount1();
```

یا

```
Account account = new LongTimeAccount1();
```

اما همچنان نمی توان متدهای benefit() را از آبجکتهاست فوق فراخوانی نمود.

اما abstract شدن کلاس Account امکان دیگری نیز می دهد و آن تعریف کردن متدهای

benefit() به صورت غیرواقعی است.

```
1 import java.util.Date;
2
3 public abstract class Account1{
```

```

4     String accountNumber;
5     Date openDate;
6     Double amount;
7     public abstract double benefit();
8 }
```

۷-۴۱ کد

طبعی است که چون کلاس Account یک کلاس abstract است نمی توان از روی آن آبجکت ساخت اما هر کلاسی که از کلاس Account مشتق شود می بایست متده benefit() را دقیقاً به همان شکلی که در کلاس Account تعریف شده است پیاده سازی کند.

مثالی دیگر از abstract

فرض کنید بخواهید یک برنامه حقوق و دستمزد پیاده سازی کنید تا با استفاده از آن، حقوق کارمندان شرکت محاسبه و پرداخت شود.

فرض کنید که شرکت دارای سه دسته کارمند باشد:

۱- یک دسته، کارمندان ثابت شرکت هستند این گونه کارمندان میزان ساعات مشخصی در ماه کار می کنند و میزان حقوق ثابتی را نیز دریافت می کنند (البته در صورتیکه بیش از ساعت موظف خود در شرکت حضور داشته باشند اضافه کار دریافت می کنند).

۲- دسته دیگر، کارمندان ساعتی هستند که بسته به میزان ساعتی که در شرکت کار می کنند حقوق دریافت می کنند به این گونه کارمندان، کارمندان ساعتی نیز گفته می شود

۳- کارمندان دیگری نیز در یک شرکت وجود دارند که به صورت پیمانکاری کار می کنند این گونه کارمندان، با شروع یک پروژه، میزان دستمزد مشخصی را مشخص می کنند و در پایان پروژه (که ممکن است هر باره زمانی را شامل شود) حقوق دریافت می کنند.

بنابراین به نظر می رسد که در این برنامه سه کلاس PermanentWorker، HourlyWorker و ContractWorker وجود داشته باشند.

```

class PermanentWorker{
}

class HourlyWorker{
}

class ContractWorker {
```

ارث بری

تمام کارمندان دارای نام، نام خانوادگی و میزان حقوق ماهیانه هستند بنابراین می توان کلاس های فوق را به صورت زیر تغییر داد.

```
1 public class PermanentWorker{  
2     private String firstName ;  
3     private String lastName ;  
4     private double monthlyPayment ;  
5  
6     public double getSalary(){  
7         return monthlyPayment ;  
8     }  
9 }
```

V-۴۹ کد

```
1 public class HourlyWorker{  
2     private String firstName ;  
3     private String lastName ;  
4     private double hourlyPayment ;  
5     private double hours;  
6  
7     public double getSalary(){  
8         return hours* hourlyPayment ;  
9     }  
10 }
```

V-۵۰ کد

```
1 public class ContractWorker {  
2     private String firstName ;  
3     private String lastName ;  
4     private double contractPayment ;  
5     private double contractNumbers ;  
6  
7     public double getSalary(){  
8         return contractNumbers* contractPayment ;  
9     }  
10 }
```

کد ۱-۵۱

با دقت در کلاس های فوق، متوجه خواهید شد که هر سه کلاس دارای فیلد های مشترک و متد مشترک هستند بنابراین می توان رفتار مشترک سه کلاس کارمندان را در یک کلاس پایه به نام Worker تعریف کرد و هر سه کلاس را از آن مشتق نمود.

```

1 public class Worker {
2     private String firstName ;
3     private String lastName ;
4
5     public double getSalary(){
6         return 0 ;
7     }
8 }
```

کد ۱-۵۲

```

1 public class PermanentWorker extends Worker{
2     private String firstName ;
3     private String lastName ;
4     private double monthlyPayment ;
5
6     public double getSalary(){
7         return monthlyPayment ;
8     }
9 }
```

کد ۱-۵۳

```

1 public class HourlyWorker extends Worker{
2     private String firstName ;
3     private String lastName ;
4     private double hourlyPayment ;
5     private double hours;
6
7     public double getSalary(){
8         return hours* hourlyPayment ;
9     }
```

10 }

۷-۵۴ کد

```

1 public class ContractWorker extends Worker{
2     private String firstName ;
3     private String lastName ;
4     private double contractPayment ;
5     private double contractNumbers ;
6
7     public double getSalary(){
8         return contractNumbers* contractPayment ;
9     }
10 }
```

۷-۵۵ کد

به این ترتیب، فیلد های مشترک در هر سه کلاس را به کلاس Worker منتقل نمودیم، متدهای getSalary() را نیز که در هر سه کلاس وجود دارد در کلاس Worker تعریف نمودیم ولی همانطور که ملاحظه می کنید این متدهای هیچ پیاده سازی ندارد و در آن فقط جمله return 0; تعریف شده است.

```

1 public class Worker {
2     private String firstName ;
3     private String lastName ;
4
5     public double getSalary(){
6         return 0 ;
7     }
8
9     public String toString(){
10        return firstName+":"+lastName;
11    }
12 }
```

۷-۵۶ کد

در این کلاس، یک متدهای toString() نیز تعریف شده است که نحوه نمایش اطلاعات کلاس کارمندان را مشخص می کند.

هر دو متدهای **getter** و **setter** به ارث برده می شوند. با افزودن سازنده و متدهای **getter** و **setter** به هر کدام از کلاسها، به صورت زیر درخواهند آمد.

```

1  public class Worker {
2      private String firstName ;
3      private String lastName ;
4
5
6      //default constructor
7      public Worker(){
8          this.firstName="";
9          this.lastName="";
10     }
11
12     public Worker(String firstName, String lastName){
13         this.firstName = firstName;
14         this.lastName = lastName ;
15     }
16
17
18
19     public double getSalary(){
20         return 0 ;
21     }
22
23     public String toString(){
24         return firstName+":"+lastName;
25     }
26 }
```

V-۶۷

```

1  public class PermanentWorker extends Worker{
2      private double monthlyPayment ;
3
4      //default constructor
5      public PermanentWorker() {
6          super();
7          this.monthlyPayment = 0;
```

ارث بری

```
8     }
9
10    public PermanentWorker(String firstName,
11                           String lastName,
12                           double monthlyPayment) {
13        super(firstName, lastName);
14        this.monthlyPayment = monthlyPayment;
15    }
16
17    public double getSalary() {
18        return monthlyPayment ;
19    }
20 }
```

V-ΔΠ ΣΣ

```
1  public class HourlyWorker extends Worker{
2      private double hourlyPayment ;
3      private double hours;
4
5      //default constructor
6      public HourlyWorker(){
7          super();
8          this.hourlyPayment = 0;
9          this.hours = 0 ;
10     }
11     public HourlyWorker(String firstName,
12                           String lastName,
13                           double hourlyPayment,
14                           double hours){
15        super(firstName, lastName);
16        this.hourlyPayment = hourlyPayment;
17        this.hours = hours ;
18    }
19    public double getSalary() {
20        return hours* hourlyPayment ;
21    }
22 }
```

V-ΔΨ ΣΣ

```

1  public class ContractWorker extends Worker{
2      private double contractPayment ;
3      private double contractNumbers ;
4
5      //default constructor
6      public ContractWorker() {
7          super();
8          this.contractPayment = contractPayment;
9          this.contractNumbers = contractNumbers ;
10     }
11
12     public double getSalary(){
13         return contractNumbers* contractPayment ;
14     }
15 }
```

۷-۶۰ کد

برای تمام فیلد های کلاسهای فوق می توان متدهای getter و setter تعریف نمود تا از طریق آنها بتوان مقدار جدیدی به فیلد ها انتساب داد یا مقدار فعلی آنها را دریافت نمود همچنین در متدهای setter می توان صحت داده را نیز بررسی نمود:

```

1  public class PermanentWorker extends Worker{
2      private double monthlyPayment ;
3
4      //default constructor
5      public PermanentWorker() {
6          super();
7          setMonthlyPayment(monthlyPayment);
8      }
9
10     public PermanentWorker(String firstName,
11                           String lastName,
12                           double monthlyPayment) {
13         super(firstName, lastName);
14         this.monthlyPayment = monthlyPayment;
15     }
16
17     public double getSalary() {
```

```

18         return monthlyPayment ;
19     }
20
21     public void setMonthlyPayment(double payment) {
22         this.monthlyPayment=(payment>=0) ? payment : 0;
23     }
24
25     public double getMonthlyPayment() {
26         return this.monthlyPayment ;
27     }
28 }
```

V-۷۱ آن

```

1  public class HourlyWorker extends Worker{
2      private double hourlyPayment ;
3      private double hours;
4
5      //default constructor
6      public HourlyWorker(){
7          super();
8          setHourlyPayment(hourlyPayment);
9          setHours(hours);
10     }
11
12     public HourlyWorker(String firstName,
13                         String lastName,
14                         double hourlyPayment,
15                         double hours){
16         super(firstName, lastName);
17         this.hourlyPayment = hourlyPayment;
18         this.hours = hours ;
19     }
20
21     public double getSalary(){
22         return hours* hourlyPayment ;
23     }
24
25     public void setHourlyPayment(double payment){
26         this.hourlyPayment = (payment>=0) ? payment : 0 ;
```

```

27     }
28
29     public double getHourlyPayment() {
30         return this.hourlyPayment;
31     }
32
33     public void setHours(double hours) {
34         this.hours = (hours>=0) ? hours: 0 ;
35     }
36
37     public double getHours() {
38         return this.hours;
39     }
40 }
```

۶-۴۲

```

1  public class ContractWorker extends Worker{
2      private double peicePayment ;
3      private double peiceNumbers ;
4
5      //default constructor
6      public ContractWorker(){
7          super();
8          setPeicePayment(peicePayment);
9          setPeiceNumbers(peiceNumbers);
10     }
11     public double getSalary(){
12         return peiceNumbers* peicePayment ;
13     }
14
15     public void setPeicePayment (double payment){
16         this.peicePayment = (payment>=0) ? payment : 0 ;
17     }
18
19     public double getPeicePayment () {
20         return this.peicePayment;
21     }
22
23 }
```

```

24     public void setPeiceNumbers (double pieces) {
25         this.peiceNumbers = (pieces >=0) ? pieces: 0 ;
26     }
27
28     public double getPeiceNumbers () {
29         return this.peiceNumbers;
30     }
31 }
```

۷-۶۳ کد

در هنگام استفاده از کلاس‌های فوق، هیچگاه آبجکتی از کلاس Worker ساخته نمی‌شود زیرا هیچ کارمندی در شرکت یافت نمی‌شود که مشخص کننده Worker باشد (تمام کارمندان از یکی از انواع PermanentWorker یا ContractWorker یا HourlyWorker هستند) از طرفی همانطور که در کلاس Employee مشاهده می‌کنید متد () getSalary مقدار صفر برمی‌گرداند که این مقدار نیز یک مقدار غیرمنطقی است.

از آنجائیکه هیچگاه آبجکتی از روی کلاس Worker ساخته نمی‌شود می‌توان این کلاس را به صورت تعریف نمود و متد () getSalary آنرا نیز abstract کرد. در اینصورت کلاس Worker به صورت زیر درخواهد آمد:

```

1  public abstract class Worker {
2      private String firstName ;
3      private String lastName ;
4
5
6      //default constructor
7      public Worker(){
8          this.firstName="";
9          this.lastName="";
10     }
11
12     public Worker(String firstName, String lastName){
13         this.firstName = firstName;
14         this.lastName = lastName ;
15     }
16
17
18
19     public abstract double getSalary();
```

```

20
21     public String toString() {
22         return firstName+":"+lastName;
23     }
24 }
```

کد ۴۴

کلمه کلیدی final

کلمه کلیدی final، سه کاربرد دارد یکی از کاربردهای آن این است که از آن می‌توان برای تعریف متغیرهای ثابت استفاده کرد این متغیرها در هنگام تعریف شدن مقدار می‌گیرند و هیچگاه مقدار آنها تغییر نمی‌کند. دو مورد کاربرد دیگر آن در ارث بری است که در زیر به آن می‌پردازیم.

استفاده از final برای جلوگیری کردن از overriding

مواردی در برنامه‌ها وجود دارند که مایلید مانع از override شدن یک متد توسط کلاس‌های فرزند شوید. با استفاده از کلمه final که قبل از نام متد بکار می‌رود می‌توان مانع از override شدن آن متد توسط کلاس‌های فرزند شد مثال زیر این مسئله را نشان می‌دهد.

```

1 public class One {
2     final void meth() {
3         System.out.println("This is a final method.");
4     }
5 }
```

کد ۴۵

```

1 public class Two extends One {
2     void meth() { // ERROR! Won't override.
3         System.out.println("Illegal!");
4     }
5 }
```

کد ۴۶

در کامپایل برنامه فوق، با خطای کامپایل مواجه خواهد شد زیر متد meth در کلاس One به صورت final تعریف شده است.

استفاده از final برای جلوگیری از ارث بری

برخی اوقات مایلید مانع از ارث بری یک کلاس شوید به عبارت دیگر می خواهید هیچ کلاسی از یک کلاس خاص مشتق نشود راه حل این مسئله استفاده از کلمه final قبل از نام کلاس است. بکارگیری کلمه های final و abstract به صورت همزمان در مقابل نام کلاس نامعتبر است زیرا کلاس final کلاسی است که نمی تواند صاحب فرزند شود و کلاس abstract کلاسی است که نمی تواند final بدون فرزند بماند!!! در زیر مثالی از کلاس final آورده شده است.

```

1 public final class One {
2     //...
3 }
4
5
6 // The following class is illegal.
7 public class Two extends One {
8 }
```

۷-۶۷

کلاس Object

در جاوا کلاس خاصی به نام Object وجود دارد که تمام کلاسهای جاوا فرزند آن هستند به عبارت دیگر، کلاس Object پدر تمام کلاسهای جاواست. در کلاس Object دو متد equals و toString وجود دارند از متد equals برای مقایسه دو آبجکت استفاده می شود در صورتیکه دو آبجکت با یکدیگر مساوی باشند این متد مقدار true و در غیر اینصورت مقدار false برمی گرداند. برای اینکه امکان مقایسه دو آبجکت از یک کلاس را داشته باشید لازم است این متد را override کنید.

متد toString برای نمایش یک آبجکت به صورت رشته استفاده می شود به عبارت بهتر هرگاه بخواهید یک آبجکت را به صورت یک رشته در خروجی استاندارد چاپ کنید یا آن را در صفحه نمایش، نشان دهید باید این متد را override کنید این متد یک رشته برمی گرداند که مشخص کننده نمایش رشته ای کلاس مورد نظر است.

خلاصه

از آنجاییکه برنامه های شی گرا، به نوعی شبیه سازی دنیای واقعی هستند و بسیاری از اشیای دنیای واقعی نیز مدل توسعه یافته ای از اشیای دیگر هستند، در زبان جاوا مفهومی به نام ارث بری طراحی شده است براین اساس یک کلاس می تواند مدل توسعه یافته ای از یک کلاس دیگر باشد.

با استفاده از کلمه کلیدی extends در تعریف یک کلاس می توان مشخص کرد که آن کلاس از چه کلاسی مشتق شود و خصوصیات و رفتار چه کلاسی را به ارث ببرد. در جاوا ارث بری چندگانه وجود ندارد و بنابراین هر کلاس می تواند تنها از یک کلاس مشتق شود.

برای دسترسی به متدهای فیلدی کلاس پدر، می توان از کلمه کلیدی super استفاده کرد، خصوصاً زمانی کاربرد دارد که کلاس فرزند دارای فیلد یا متدهای مشابه با فیلد یا متدهای کلاس پدر باشد. با استفاده از این کلمه می توان سازنده کلاس پدر را فراخوانی کرد، این فراخوانی که در اولین خط از سازنده کلاس فرزند انجام می شود شکلی به صورت زیر دارد:

```
super (parent-parameters);
```

parent-parameters لیست پارامترهایی است که سازنده کلاس پدر دریافت می کند. کلاس فرزند نوعی از کلاس پدر است، بنابراین می توان در زمان تعریف یک آبجکت فرزند آنرا از جنس کلاس آبجکت پدر تعریف نمود:

```
parentClass obj = new childClass();
```

کلاس پدر و childClass کلاس فرزند هستند. یک کلاس می تواند متدهای پیاده سازی نشده نیز داشته باشد، در اینصورت به آن کلاس گفته می شود این کلاس با استفاده از کلمه کلیدی abstract که قبل از تعریف کلاس می آید مشخص می شود، همچنین متدهای سازی ندارد باید با همین کلمه مشخص شود. هر کلاسی که از یک کلاس abstract مشتق می شود باید متدهای abstract آنرا پیاده سازی کند یا اینکه خود به صورت abstract تعریف شود.

کلاس فرزند می تواند متدهای کاملاً مشابهی با متدهای کلاس پدر داشته باشد، به این خصوصیت overriding گفته می شود اما اگر متدهای کلاس پدر به صورت final تعریف شده باشد، کلاس فرزند نمی تواند آن را override کند.

اگر بخواهید هیچ کلاسی نتواند از یک کلاس مشتق شود، کافیست آنرا به صورت final تعریف کنید. اگر یک کلاس را از هیچ کلاس دیگری مشتق نکنید، آن کلاس از کلاس Object مشتق خواهد شد بنابراین می توان گفت که تمام کلاس‌هایی که در جاوا تعریف می شوند به نوعی فرزند کلاس Object هستند.

تمرینات

- (۱) با استفاده از مفاهیم ارث بری که در این فصل با آنها آشنا شدید، مجموعه کلاسهایی طراحی کنید که در یک رستوران معرف غذاهای آن رستوران هستند.
- (۲) با استفاده از ارث بری و مفهوم abstraction، مجموعه کلاسهایی طراحی کنید که معرف خودروهای مختلف باشند.
- (۳) کلاسهایی را طراحی کنید که معرف خودروهایی هستند که توسط یک شرکت تولید می شوند.
- (۴) مجموعه کلاسهایی را طراحی کنید که معرف تمام شهرها و ابرشهرهای جهان هستند.
- (۵) مجموعه کلاسهایی طراحی کنید که معرف تمام خانه های مسکونی هستند.
- (۶) مجموعه کلاسهایی را طراحی کنید که معرف تمام موسسات آموزشی و دانشگاهها باشند.
- (۷) فرض کنید که قرار است برای یک شرکت حمل و نقل برنامه ای بنویسید تمام کلاسهایی که به نظرتان می رسد را طراحی و پیاده سازی کنید. در طراحی این کلاسها از مفاهیم abstraction و ارث بری نیز استفاده کنید.
- (۸) فرض کنید که می خواهید برای یک شرکت نرم افزاری که انواع مختلفی از نرم افزارها را تولید می کند برنامه ای بنویسید، کلاسهایی که فکر می کنید برای مدلسازی نرم افزارهای این شرکت به آنها نیاز دارید را طراحی و پیاده سازی کنید. (هرچه بگندند نمکش می زند وای به روزی...).
- (۹) فرض کنید که قرار است برای یک کتابخانه نرم افزاری نوشته شود، کلاسهایی که در این نرم افزار وجود خواهند داشت را طراحی و پیاده سازی کنید.
- (۱۰) کلاسهایی را که برنامه ای برای پرندگان یک باغ وحش به نظرتان می رسد را طراحی و پیاده سازی کنید.

۱

اینترفیس

در فصل پیش با کلاس **abstract** آشنا شدید، این کلاسها تجسم خارجی ندارند (یعنی در دنیای خارجی، هیچ شئ ای معادل این کلاسها وجود ندارد) و تعریف این کلاسها فقط به ضرورت طراحی شی گرایی انجام می شود. اگر سهوا آبجکتی از این کلاسها در برنامه ایجاد کنید، کامپایلر جاوا خطای کامپایل می دهد و مانع آن می شود. از طرف دیگر، از آنجاییکه هیچ گاه آبجکتی از روی این کلاسها ساخته نمی شود، جاوا اجازه داده تا برخی یا همه متدهای این کلاسها به صورت ناقص پیاده سازی شوند (بدنه نداشته باشند). در تعریف این کلاسها و در تعریف متدهای ناقص این کلاسها (متدهایی که بدنه ندارند) از کلمه **abstract** استفاده می شود. کد ۱-۸ یک نمونه از کلاس **abstract** را نشان می دهد.

```
1 public abstract class Employee {  
2     String firstName;  
3     String lastName;  
4  
5     public Employee(String firstName, String lastName) {  
6         this.firstName = firstName;  
7         this.lastName = lastName;  
8     }  
9  
10    public String getFirstName() {
```

```

11     return firstName;
12 }
13
14     public String getLastName() {
15         return lastName;
16     }
17
18     public abstract double getSalary();
19 }
```

کد ۸-۱

وقتی یک کلاس **abstract** تعریف می شود معنی آن این است که حداقل یک کلاس دیگر در برنامه وجود دارد که از این کلاس **abstract** مشتق شده است در غیر این صورت از آنجایی که نمی توان از روی کلاس **abstract** آبجکت ایجاد کرد، عملاً آن کلاس هیچ کاربردی نخواهد داشت.

علاوه بر کلاس (که ممکن است **abstract** باشد یا نباشد) در جاوا امکان تعریف موجود دیگری که به آن «اینترفیس» گفته می شود نیز امکان پذیر است. اینترفیس را می توان مشابه یک کلاس **abstract** تصور کرد که تمام متدهای آن **abstract** تعریف شده اند یعنی به صورت ناقص تعریف شده اند و بدنه ندارند. کد ۸-۲ یک نمونه اینترفیس را نشان می دهد.

```

1 public interface PersonTemplate {
2     public String getFirstName();
3     public String getLastName();
4 }
```

کد ۸-۲

حال یک کلاس می تواند از اینترفیس فوق پیروی کند که در این صورت می بایست تمام متدهای این اینترفیس را پیاده سازی کند. کد ۸-۳ این قاعده را نشان می دهد.

```

1 public class HourlyEmployee implements PersonTemplate {
2
3     String firstName;
4     String lastName;
5
6     public String getFirstName() {
7         return firstName;
8     }
9 }
```

اینترفیس

```
10     public String getLastName() {  
11         return lastName;  
12     }  
13 }
```

کد ۸-۳

ممکن است بپرسید که کاربرد اینترفیس چیست و چرا با وجود کلاس **abstract** به آن نیاز داریم؟ در جواب باید بگوییم که از آنجاییکه در جاوا ارث بری چندگانه وجود ندارد، اینترفیس می تواند خلاً آنرا پر کند. به این ترتیب، یک کلاس می تواند در عین حال که از یک کلاس دیگر مشتق می شود (ارث می برد) از یک اینترفیس نیز پیروی کند. کد ۸-۴ این قاعده را نشان می دهد.

```
1  public class HourlyEmployee1 extends Employee implements  
2  PersonTemplate {  
3  
4      double hours;  
5      double paymentPerHour;  
6  
7      public HourlyEmployee1(String firstName, String lastName) {  
8          super(firstName, lastName);  
9      }  
10  
11     public double getSalary() {  
12         return hours*paymentPerHour;  
13     }  
14  
15     public String getFirstName() {  
16         return firstName;  
17     }  
18  
19     public String getLastName() {  
20         return lastName;  
21     }  
22 }
```

کد ۸-۴

در اینجا، در حالیکه کلاس **HourlyEmployee** از کلاس **Employee** ارث می برد، از اینترفیس **PersonTemplate** نیز پیروی می کند.

این ویژگی، مهم ترین کاربرد اینترفیس این است زیرا می‌تواند مثل پلی، دو مجموعه کلاس، که هر مجموعه ارث بری‌های خودش را دارد، به هم مرتبط کند.

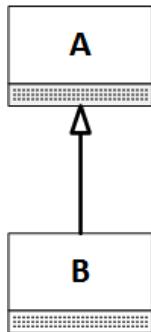
توجه: از نظر مفهومی نسبت کلاس به آبجکت، همان نسبت اینترفیس به کلاس است. به عبارت دیگر، از کلاس برای توصیف مجموعه‌ای از اشیاء مشابه و از اینترفیس برای توصیف مجموعه‌ای از کلاس‌های مشابه استفاده می‌شود.

همانطور که از مثال فوق نیز دریافت‌هه اید، از کلمه `interface` (به جای کلمه `class`) برای تعریف یک اینترفیس استفاده می‌شود. از آنجاییکه اینترفیسها مشخص کننده موجودات واقعی در دنیای خارج نیستند می‌توانند مشابه کلاس‌های `abstract` متدهای ناقص (متدهای بدون بدنه) داشته باشند.

یک کلاس ممکن است «از یک اینترفیس پیروی کند». پیروی کردن از یک اینترفیس مشابه ارث بری از یک کلاس است.

توجه: از آنجاییکه در جاوا از کلمه `implements` برای مشخص کردن پیروی از یک اینترفیس استفاده می‌شود که معنی آن «پیاده سازی» است، در این کتاب از عبارت «پیاده سازی اینترفیس» به جای عبارت «پیروی از اینترفیس» استفاده می‌کنیم.

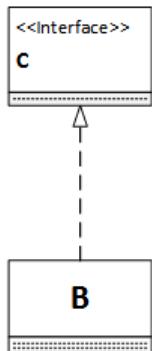
وقتی یک کلاس یک اینترفیس را پیاده سازی می‌کند باید تمام متدهای آن اینترفیس را پیاده سازی کند. ما در این کتاب برای اینکه بگوییم کلاس `B` از کلاس `A` مشتق شده است از علامت گذاری شکل ۱-۸ استفاده می‌کنیم.



شکل ۱-۸. ارث بری

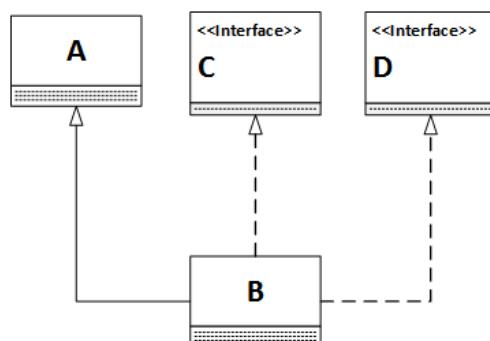
و برای اینکه بگوییم کلاس `B` اینترفیس `C` را پیاده سازی می‌کند علامت شکل ۱-۲ را به کار می‌بریم.

اینترفیس



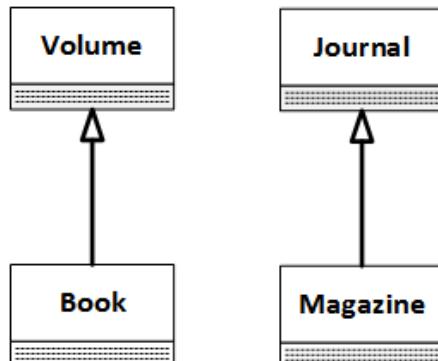
شکل ۸-۲. پیاده سازی اینترفیس

از آنجاییکه یک کلاس می تواند هم زمان از یک کلاس مشتق و یک یا بیش از یک اینترفیس را پیاده سازی کند می توانیم چنین فرم هایی از ارتباطات کلاسها نیز داشته باشیم



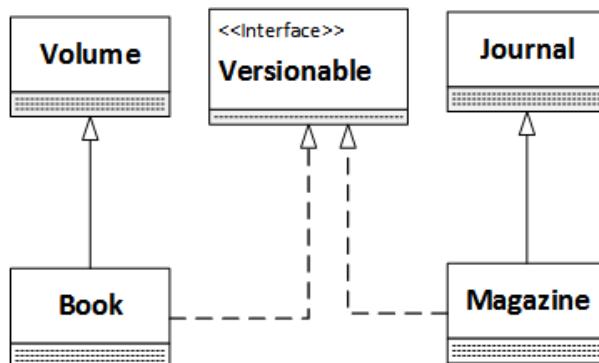
شکل ۸-۳. پیاده سازی بیش از یک اینترفیس

برای درک کاربرد اینترفیس به یک مثال توجه کنید. فرض کنید که کلاس‌های Book و Magazine که به ترتیب ترتیب معرف کتاب و مجله هستند با سلسله مراتبی که در شکل ۸-۴ نشان داده شده، در برنامه طراحی شده اند.



شکل ۴-۴. اشتاقاق در دو کلاس مختلف

با وجود اینکه کلاس‌های **Book** و **Magazine** کلاس‌های مستقلی هستند اما از آنجاییکه اشتراکاتی دارند تصمیم می‌گیرید که طراحی آنها را اصلاح کنید و مشترکات آنها را به گونه‌ای لحاظ کنید. متأسفانه یا خوشبختانه در جاوا ارث بری چندگانه وجود ندارد تا بتوان مشترکات کلاس‌های **Book** و **Magazine** را به یک کلاس جدید منتقل و این دو کلاس را از آن مشتق نمود. راه حل، استفاده از اینترفیس است تا اشتراکات را به یک اینترفیس (مثلاً اینترفیس **Versionable**) منتقل کرده و کلاس‌های کلاس‌های **Book** و **Magazine** را از آن پیاده سازی نمود.



شکل ۴-۵. استفاده از اینترفیس برای ارتباط دو سلسله مختلف

توجه: امروزه وقتی نرم افزاری تولید و توسعه داده می‌شود، طراحی آن برنامه نیز به صورت مداوم بازبینی می‌شود و تغییر می‌یابد. این بدين معنی است که فرایند توسعه نرم افزار همواره با تغییر در طراحی همراه است، به این قاعده **Refactoring** گفته می‌شود.

اینترفیس

وقتی در حال برنامه نویسی هستید، همواره حواستان به کدهای تکراری و اشتراکات بین کلاسها باشد. طراحی برنامه را به گونه ای تغییر دهید که کدهای تکراری نداشته باشید و اشتراکات کلاسها فقط در یک جا ظاهر (مثلاً یک کلاس یا اینترفیس) ظاهر شوند.

تعريف اینترفیس

تعريف اینترفیس شبیه تعريف کلاس است با این تفاوت که از کلمه `interface` به جای کلمه `class` استفاده می شود.

```
access interface name {  
    return-type method-name1 (parameter-list);  
    .  
    .  
    type final-varname1=value ;  
    .  
    .  
    .  
}
```

مقدار `public` می تواند `public` باشد یا اصلا وجود نداشته باشد وقتی که `access` مشخص نشده باشد اینترفیس تعريف شده فقط توسط اینترفیس ها و کلاسهايی که در همان پکیج قرار دارند قابل استفاده خواهد بود اما اگر به صورت `public` تعريف شود توسط هر کلاس یا اینترفیس دیگری قابل استفاده خواهد بود `name` نام اینترفیس را مشخص می کند و باید طبق قوانین نامگذاری در جوا انتخاب شود. اینترفیس می تواند به تعداد دلخواه متداشته باشد اما متدهای اینترفیس همگی بدون بدن هستند و به ; ختم می شوند. این متدها در حقیقت، متدهای `abstract` ای هستند که فاقد پیاده سازی هستند و هر کلاسی که اینترفیس را پیاده سازی کند باید تمام آنها را پیاده سازی کند.

درون یک اینترفیس می توان متغیر نیز تعريف نمود متغیرهایی که درون یک اینترفیس تعريف می شوند تماما به صورت `final` و `static` تعريف می شوند (حتی اگر این متغیرها را به صورت `final` و `static` تعريف نشوند، کامپایلر جوا در زمان کامپایل، `final` و `static` را به تعريف آنها اضافه می کند). بودن `static` و `final` بدين معناست که فيلدهای اینترفیس الزاما می بايست مقدار اولیه داشته باشند و هرجایی که اینترفیس در دسترس باشد می توان به آنها دست یافت. در ضمن به علت `final` بودن، مقدار آنها را نمی توان تغییر داد.

در صورتیکه متدها یا متغیرهای یک اینترفیس به صورت `public` تعريف نشده باشند این متدها و متغیرها از اینترفیس پیروی می کنند یعنی اگر اینترفیس به صورت `public` تعريف شده باشد متدها و متغیرهای داخل اینترفیس نیز `public` خواهند بود. در زیر مثال زیر ساده ای از یک اینترفیس نشان داده شده است در این اینترفیس یک متده به نام `Callback` دارد که یک عدد `int` بصورت پارامتر دریافت می کند.

```

1 public interface Callback {
2     void callback(int param);
3 }
```

کد ۱-۴

پیاده سازی اینترفیس

وقتی یک اینترفیس طراحی شد کلاسهای مختلف می‌توانند آنرا پیاده سازی کنند. پیاده سازی یک اینترفیس از طریق کلمه `implements` که بعد از نام کلاس نوشته می‌شود مشخص می‌شود شکل کلی پیاده سازی یک اینترفیس به صورت زیر است:

```

access class class-name implements interface-name1,
                     interface-name2,
                     ...,
                     interface-name3{
    //class body
}
```

مقدار `access` می‌تواند باشد یا اصلا هیچ مقداری نداشته باشد. یک کلاس می‌تواند بیش از یک اینترفیس را پیاده سازی کند که نام آنها با کاما (,) از یکدیگر جدا می‌شود و در آنصورت آن کلاس باید تمام متدهای تمام اینترفیس را پیاده سازی کند. اگر دو اینترفیس دارای یک متدهای کلی باشند در آنصورت کلاسی که آن دو اینترفیس را پیاده سازی می‌کند باید فقط یکبار آن متدهای کلی را پیاده سازی کند. کد ۱-۵ کلاس `Client` را نشان می‌دهد که اینترفیس `Callback` را که در کد ۱-۴ تعریف کردیم را پیاده سازی کرده است.

```

1 public class Client implements Callback {
2     // Implement Callback's interface
3     public void callback(int p) {
4         System.out.println("callback called with " + p);
5     }
6 }
```

کد ۱-۵

توجه: دقت کنید که متدهای اینترفیس را به صورت `public` تعریف شده است.

اینترفیس

کلاسی که یک اینترفیس را پیاده سازی می کند می تواند علاوه بر متدهای داخل اینترفیس، متدهای شخصی خودش را نیز داشته باشد کد ۶-۸، نسخه دیگری از کلاس Client را نشان می دهد که اینترفیس Callback را پیاده سازی کرده است و دارای یک متداضف به نام call است.

```
1 public class Client1 implements Callback {  
2  
3     // Implement Callback's interface  
4     public void callback(int p) {  
5         System.out.println("callback called with " + p);  
6     }  
7  
8     void call() {  
9         System.out.println(  
10            "Classes that implement interfaces " +  
11            "may also define other members, too.");  
12     }  
13 }
```

کد ۶

در ایجاد آبجکت، می توانید آن را از یک اینترفیس تعریف کنید و از یکی از کلاسهایی که آن اینترفیس را پیاده سازی کرده اند ایجاد کنید. به عنوان مثال، می توانید آبجکت c را از اینترفیس Callback تعریف و از کلاس Client به صورت زیر ایجاد کنید.

```
Callback c = new Client();  
c.callback(42);
```

توجه داشته باشید که از آنجا که آبجکت c از جنس Callback تعریف شده است، نمی توانید متداضف call را از این آبجکت فراخوانی کنید.

استفاده از اینترفیس، یکی از راههای پیاده سازی پلی مورفیسم در جاواست تا در حالیکه دو آبجکت از یک جنس هستند رفتار مختلفی از خود نشان دهند. به عنوان مثال، کد ۸-۷ کلاس LocalClient را نشان می دهد که اینترفیس Callback را پیاده سازی کرده است.

```
1 // Another implementation of Callback.  
2 public class LocalClient implements Callback {  
3     // Implement Callback's interface  
4     public void callback(int p) {  
5         System.out.println("Another version of callback");  
6         System.out.println("p squared is " + (p * p));  
7     }  
8 }
```

```

7     }
8 }
```

کد A-7

حال اگر دو آبجکت از اینترفیس Client و LocalClient اما از کلاس‌های Callback ایجاد کنیم، در هر دو آبجکت متدهای callback قابل فراخوانی است اما رفتار متفاوتی را از این دو متدهای دریافت خواهیم کرد.

```

Callback c = new Client();
Callback ob = new LocalClient();

c.callback(42);

c = ob; // c now refers to AnotherClient object
c.callback(42);
```

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود:

```
callback called with 42
Another version of callback
p squared is 1764
```

پیاده سازی ناقص یک اینترفیس

یک کلاس می‌تواند یک اینترفیس را به صورت ناقص پیاده سازی کند (برخی متدهای اینترفیس را پیاده سازی کند) که در اینصورت آن کلاس باید به صورت abstract تعریف شود. کد ۸-۱۰ کلاس AbstractClient را نشان می‌دهد که قادر به فاقد متدهای callback است بنابراین به صورت abstract تعریف شده است.

```

1 public abstract class AbstractClient implements Callback {
2     int a, b;
3     void show() {
4         System.out.println(a + " " + b);
5     }
6     // ...
7 }
```

کد A-8

توجه: نام کلاس فوق از کلمات Abstract و Client تشکیل شده است که ماهیت و هدف کلاس را توضیح می دهد. اصولا همیشه نام هر کلاس باید بتواند به تنها یعنی هدف اصلی کلاس بیان کند. توصیه می شود مشابه کلاس AbstractClient، نام کلاس های abstract را با کلمه abstract شروع کنید.

متغیرهایی که درون اینترفیس تعریف می شوند

هر اینترفیس می تواند دارای فیلد باشد که همگی مقادیر ثابت (static و final) هستند حتی اگر کلمات static final مقابله نام آنها ذکر نشود. کلاس SharedConstants این موضوع را نشان می دهد.

```
1 public interface SharedConstants {  
2     int NO = 0;  
3     int YES = 1;  
4     int MAYBE = 2;  
5     int LATER = 3;  
6     int SOON = 4;  
7     int NEVER = 5;  
8 }
```

۱-۹ کد

هر کلاسی که یک اینترفیس را پیاده سازی کند، می تواند مستقیماً و بدون ذکر نام اینترفیس به فیلدهای اینترفیس دسترسی یابد. کلاس Question این موضوع را نشان می دهد.

```
1 public class Question implements SharedConstants {  
2  
3     public int ask(int prob) {  
4         if (prob < 30)  
5             return NO;  
6         else if (prob < 60)  
7             return YES;  
8         else if (prob < 75)  
9             return LATER;  
10        else if (prob < 98)  
11            return SOON;  
12        else  
13            return NEVER;
```

```

14     }
15 }
```

۸-۱۰ لیست

در این مثال کلاس Question تعریف شده که اینترفیس SharedConstants را پیاده سازی کرده است و از مقادیر ثابت تعریف شده در اینترفیس در پیاده سازی متده است.

ارث بری در میان اینترفیس ها

اینترفیسها نیز مشابه کلاسها می توانند با استفاده از کلمه extends از یکدیگر مشتق شوند. به اینترفیس پدر، super-interface و به اینترفیس فرزند sub-interface گفته می شود. کلاسی که یک اینترفیس فرزند را پیاده سازی می کند می بایست تمام متدهای آن اینترفیس و تمام متدهایی که از اینترفیس پدر خود به ارث برده است را پیاده سازی کند.

توجه به این نکته ضروری است که مشابه ارث بری در کلاسها، متدهای فرزند نمی توانند در تضاد با متدهای پدر باشند. برای درک این موضوع به اینترفیس‌های EnhancedCallback و Callback به صورت زیر توجه کنید.

```

1 public interface Callback {
2     void callback(int param);
3 }
4 public interface EnhancedCallback extends Callback{
5     int callback(int param); //compile error
6 }
```

۸-۱۱ لیست

در اینجا متده callback() در اینترفیس فرزند تعریف مشابه متده callback() در اینترفیس پدر است با این تفاوت که مقدار برگشتی این متده در کلاس پدر void و در کلاس فرزند int است. این یک تضاد محسوب می شود زیرا متدها براساس نام و پارامترهایشان از یکدیگر تمیز داده می شوند. وجود دو متده callback() با پارامترهای یکسان در اینترفیس EnhancedCallback قابل قبول نیست.

یک تفاوت مهم در ارث بری اینترفیسها و کلاسها وجود دارد. در اینترفیسها ارث بری چندگانه وجود دارد بنابراین یک اینترفیس می تواند از یک یا بیش از یک اینترفیس دیگر مشتق شود.

یک مثال کاربردی

یکی از کاربردهای اینترفیس، زمانی است که می خواهیم دو بخش مختلف برنامه قرار است با یکدیگر در گیر شوند. طبیعی است که به دلایل مختلف از جمله نگهداری آسان تر و کم هزینه تر به دنبال کم کردن و به حداقل رساندن وابستگی بین دو بخش برنامه باشیم. استفاده از اینترفیس برای مرتبط کردن دو بخش برنامه بهترین راهکار است زیرا اینترفیسها بدون هیچ پیاده سازی صرفا ساختار متدها را مشخص می کنند و هر کلاسی که آنها را پیاده سازی کند می تواند جایگزین آنها شود.

به عنوان مثال تصور کنید که ما شرکتی هستیم که نرم افزاری برای تولید انواع گزارشات تولید کرده ایم و آنرا در قالب یک کتابخانه در اختیار شرکتها و سازمانهای مختلف قرار می دهیم تا با اضافه کردن آن به برنامه های خود ویژگی گزارش گیری را در برنامه هایشان اضافه کنند. در نرم افزاری که ما تولید کرده ایم، کلاس ReportGenerator به صورت زیر پیاده سازی شده است.

```

1 public class ReportGenerator {
2     public void report(Template template) {
3         //...
4     }
5 }
```

لیست ۸-۱۲

این کلاس یک متد () report دارد که آبجکتی از اینترفیس Template دریافت می کند.

```

1 public interface Template{
2     public String getTitle();
3     public String getDate();
4     public String getContent();
5     public String getFooter();
6 }
```

لیست ۸-۱۳

ما به مشتریان خود می گوییم برای استفاده از نرم افزار گزارش ساز، می بایست یک آبجکت از اینترفیس Template به متد () report ارسال کنند. اینکه آبجکت Template از چه کلاسی و با چه جزیاتی باشد به خود آنها بستگی دارد اما هر کلاسی که باشد می بایست اینترفیس Template را پیاده سازی کرده باشد و طبیعتاً حاوی متدهای این اینترفیس باشد.

خلاصه

در این فصل با مفهومی به نام اینترفیس آشنا شدید، کلاس تعریفی از یک موجود واقعی است و اینترفیس تعریفی از یک کلاس. تعریف اینترفیس همانند تعریف یک کلاس است با این تفاوت که در یک اینترفیس به جای کلمه `class` از کلمه `interface` استفاده می‌شود، هیچکدام از متدهایی که داخل اینترفیس تعریف می‌شوند بدنه ندارند و فقط ساختار متدها مشخص می‌کنند.

یک کلاس می‌تواند یک اینترفیس را پیاده سازی کند، در اینصورت باید متدهایی کاملاً منطبق بر متدهایی که داخل اینترفیس تعریف شده است داشته باشند. به این منظور باید در تعریف کلاس، نام اینترفیس بعد از کلمه `implements` که بعد از نام کلاس می‌آید اعلام شود.

کلاسی که یک اینترفیس را پیاده سازی می‌کند می‌تواند برخی یا حتی تمام متدهای اینترفیس را پیاده سازی نکند، که در اینصورت باید به صورت `abstract` تعریف شود.

ارث بری در میان اینترفیس‌ها هم وجود دارد هر اینترفیس می‌تواند از یک یا بیش از یک اینترفیس دیگر مشتق شود این کار با استفاده از کلمه `extends` انجام می‌شود.

هر کلاسی که اینترفیس فرزند را پیاده سازی می‌کند باید متدهای آن اینترفیس و اینترفیس پدران و اجداد آنها را نیز پیاده سازی کند.

تمام فیلدهای یک اینترفیس، به صورت `final` و `static` هستند حتی اگر از این کلمات کلیدی برای تعریف آنها استفاده نشود.

تمرینات

- (۱) یک اینترفیس Serial تعریف کنید که معرف دنباله اعداد صحیح است. به دنباله های زیر توجه کنید
- دنباله اعداد صحیح شامل ۰، ۱، ۲، ۳، ...
- دنباله اعداد اول شامل ۱، ۲، ۳، ۵، ۷، ...
- دنباله فیبوناچی شامل ۱، ۲، ۳، ۵، ۸، ...
- به این ترتیب برای هریک از دنباله های فوق، می توان یک کلاس پیاده سازی کرد که اینترفیس Serial را پیاده سازی کند.
- (۲) شیرینی ها و غذاها متفاوت هستند اما همگی با استفاده از یک مجموعه مواد اولیه تهیه می شوند، همگی دستور پخت دارند، و همگی شروع و پایان پخت دارند. اینترفیس Cookable را تعریف کنید تا بتوان برای تعریف کلاس های شیرینی و غذاها از آن استفاده کرد.
- (۳) اینترفیس Printable را تعریف کنید که هر کلاسی که آنرا پیاده سازی کند امکان پرینت گرفتن آن وجود داشته باشد.

۹

Enumeration

نوع خاصی از کلاس است که اساسا برای تعریف مقادیر ثابت طراحی شده است. اجازه دهید قبل از اینکه جزئیات پیاده سازی Enumeration را تشریح کنیم، با یک مثال فلسفه وجود آن را توضیح دهیم.

برنامه ای را تصور کنید که کارهای هفتگی شما را مدیریت و زمانبندی می کند. طبیعی است که در این برنامه، باید روزهای هفته تعریف شده باشند تا کاربر بتواند با انتخاب یک روز هفته، کارهای آن روز را مشخص و زمانبندی کند. در پیاده سازی این برنامه با دانشی که تا اینجا داریم، برای نشان دادن هر روز هفته از یک عدد استفاده می کنیم مثلا عدد ۰ به معنی شنبه، عدد ۱ به معنی یک شنبه، عدد ۲ به معنی دوشنبه،... اگرچه استفاده از اعداد برای نشان دادن روزهای هفته منطقی است اما مطمئن نیست! زیرا همیشه این امکان وجود دارد تا سهوا از عددی خارج از محدوده روزهای هفته استفاده شود (مثلا عدد ۱ - یا عدد ۹) و بدون اینکه کامپایلر، خطای کامپایل تولید کند، برنامه در زمان اجرا چار خطای منطقی خواهد شد. روزهای هفته (شنبه، یکشنبه,...) یک نمونه از موارد فراوانی است که در طول حیات برنامه تغییر نمی کنند و به همین دلیل به آنها مقادیر ثابت گفته می شود. جنسیت افراد (مرد، زن)، ماه های سال (فروردين، اردیبهشت,...)، میزان تحصیلات افراد (بی سواد، ابتدایی، متوسطه، کارданی، کارشناسی,...)، نوع شهروند (شهری، روستایی، عشاپری) چند نمونه دیگر از مقادیر ثابت هستند. Enumeration روشی برای تعریف کردن مقادیر ثابت در یک برنامه است که اشکالی که در تعریف عددی روزهای هفته وجود داشت را حل می کند. Enumeration مشابه کلاس تعریف می شود با این تفاوت که

به جای کلمه `enum` از کلمه `class` در تعریف آن استفاده می شود. برای سادگی در ادامه این فصل از کلمه `Enumeration` استفاده می شود.

یک مثال ساده

به مثالی که در مقدمه این فصل مطرح شد برگردیم که هر روز هفته با یک عدد نشان داده شد. فرض کنید به متدهی نیاز داریم که دو روز هفته را با یکدیگر مقایسه کند آیا اولی قبل از دومی قرار دارد.

```
public boolean before(int aDay, int bDay) {
    if(aDay-bDay<0) {
        return true;
    } else{
        return false;
    }
}
```

مثالا برای اینکه بررسی کنیم آیا روز دوشنبه قبل از روز سه شنبه قرار دارد این متده ب صورت زیر فراخوانی می شود.

```
int d1 = 2;
int d2 = 3;
if(before(d1, d2)) {
    ...
}
```

اگر چه کد فوق به درستی کار می کند، اما اگر سهوا به جای عدد ۳ که معرف روز سه شنبه است از عدد ۱۳ استفاده شده شود، برنامه خطای منطقی منطقی شده زیرا هیچ روزی از هفته با عدد ۱۳ نشان داده نمی شود. اگر روزهای هفته را به جای اعداد با `enum` نشان دهیم کامپایلر می تواند صحت داده هایی که به متده `before()` ارسال می شوند را بررسی کند تا خطاهای منطقی به زمان اجرا راه پیدا نکند و قبل از آن کشف و اصلاح شوند. کد زیر، یک `enum` را نشان می دهد که معرف روزهای هفته است.

```
1 package ir.atlassoft.javase.chapter9;
2
3 public enum Day{
4     FRIDAY, SATURDAY, SUNDAY, MONDAY, TUESDAY, THURSDAY,
5     WEDNESDAY
6 }
```

کد ۹-۱

Enumeration

در اینجا داده جدید Day را تعریف کرده ایم که حاوی مقادیر MONDAY، SUNDAY، SATURDAY، TUESDAY ... به عنوان روزهای هفتگی است. اگر بخواهیم روز دوشنبه را مشخص کنیم از Day.MONDAY استفاده می کنیم و Day.FRIDAY به ترتیب مشخص کننده روزهای یکشنبه و جمعه هستند.

متدهای before() که حالا با int کار می کند به صورت زیر بازنویسی شده است.

```
public boolean before(Day aDay, Day bDay) {  
    if(aDay.compareTo(bDay)<0) {  
        return true;  
    }  
    return false;  
}
```

در کد فوق از متدهای compareTo() که در هر enum به صورت پیش فرض وجود دارد برای مقایسه دو آبجکت Day استفاده شده است. مبنای کارکرد متدهای compareTo()، ترتیبی است که مقادیر ثابت تعریف شده اند، هر مقدار ثابتی که قبل از تعریف شده باشد کوچکتر است در مثال ما، مقدار ثابت FRIDAY کوچکتر از SATURDAY است زیرا قبل از آن تعریف شده است. حال به نحوه فراخوانی متدهای before() که در زیر آمده است توجه کنید.

```
Day d1 = Day.MONDAY;  
Day d2 = Day.TUESDAY;  
if(before(d1, d2) ) {  
    ...  
}
```

نکته جالبی که حتماً به آن توجه کرده اید این است که آبجکتهاي d1 و d2 هر کدام از جنس Day تعریف شده اند و در عین حال، هر کدام از آنها یکی از مقادیر ثابت تعریف شده در Day هستند. در واقع هر یک از فیلدهای یک enum از جنس همان enum نیز هست.

توجه: از new نمی توان برای ساخت آبجکت از روی یک enum استفاده کرد.

با تعریف Day برنامه وضوح و خوانایی بهتری پیدا کرده است زیرا موجودی با اسم Day بسیار بامعنی تر از یک عدد int است. علاوه بر این، کامپایلر از بکارگیری هر مقداری به جز مقادیر ثابت تعریف شده در Day ممانعت می کند. در متدهای before() پارامترها از جنس Day تعریف شده اند بنابراین هیچ مقداری به جز مقادیر ثابت تعریف شده در Day را نمی توان به این متدها ارسال نمود. البته مثل هر آبجکت دیگری، مقدار null را می توان به یک متغیر از جنس enum انتساب داد.

توجه: اگرچه در نامگذاری مقادیر ثابت یک enum آزاد هستید، ولی بهتر است از حروف بزرگ برای نام یک مقدار ثابت استفاده کنید.

علاوه بر متدهای compareTo() و equals()، به صورت پیش فرض متدهای equals() و hashCode() در هر enum نیز در آن برای بررسی تساوی دو مقدار enum استفاده می‌شود.

توجه داشته باشید که به جای == می‌توان از عملگر equals() نیز استفاده نمود که نسبت به () این مزیت را دارد که می‌توان از آن برای مقایسه یک متغیر null نیز استفاده نمود.

```
Day d = ...;
if(d == Day.MONDAY) {
    ...
}
```

تمرین: یک enum تعریف کنید که مشخص کننده وضعیت چراغ‌های راهنمای راهنمای باشد.

تعریف enum

یک enum یک نوع کلاس است اما از نظر تعریف و پیاده سازی با کلاس تفاوت‌هایی دارد. اولین تفاوت، استفاده از کلمه enum به جای کلمه class است. یک همانند یک کلاس می‌تواند فیلد، متدهای سازنده داشته باشد اما مقادیر ثابت یک enum همیشه باید قبل از دیگر اجزای enum باشد. در انتهای تعریف مقادیر ثابت enum که با کاما از یکدیگر جدا می‌شوند می‌تواند کاما یا؛ قرار گیرد اما اگر یک enum به جز مقادیر ثابت دارای متدهای فیلد یا سازنده باشد باید در انتهای تعاریف ثابت از ; استفاده شود. به عنوان مثال Day را که متدهای getSize() و toString() را به آن اضافه شده است را تصور کنید.

```
1 package ir.atlassoft.javase.chapter9;
2
3 public enum Day1 {
4     FRIDAY, SATURDAY, SUNDAY, MONDAY, TUESDAY, THURSDAY,
5     WEDNESDAY;
6
7     public static int getSize() {
8         return 7;
9     }
10 }
```

یک enum نمی تواند از یک کلاس یا enum دیگر مشتق شود زیرا به صورت تلویحی از java.lang.Enum مشتق شده است. همچنین هیچ کلاس یا enum دیگر نمی توانند یک enum دیگر را توسعه دهند زیرا به صورت تلویحی هر enum به صورت final تعریف می شود. اگرچه enum ها می توانند سازنده داشته باشند، اما از آنجاییکه در هیچ جایی از برنامه آبجکتی از روی آنها ایجاد نمی شود، سازنده های آن نمی توانند public یا protected باشد. هر enum به صورت تلویحی اینترفیس‌های Comparable و Serializable را پیاده سازی کند.

نکته: اگرچه یک enum می تواند خیلی پیچیده باشد یعنی یک یا چند اینترفیس را پیاده سازی کند، چندین متده سازنده داشته باشد، و...اما در عمل در اغلب موارد enum ها بسیار ساده هستند و تنها حاوی مقادیر ثابت می باشند.

تعريف مقادیر ثابت در enum

مقادیر ثابتی که در یک enum تعریف می شوند اگرچه در بیشتر مواقع مشابه آنچه در Day دیدید فقط از یک نام تشکیل شده اند. اما در واقع آبجکتها ای از جنس همان enum هستند. کد ۹-۳ یک کلاس را نشان می دهد که معادل enum فوق است. با دقت در این کلاس و مقایسه آن با enum فوق درک بهتری از مقادیر داخل enum بدست خواهید آورد.

```

1 package ir.atlassoft.javase.chapter9;
2
3 public class Day2{
4     public static final Day2 FRIDAY = new Day2();
5     public static final Day2 SATURDAY = new Day2();
6     public static final Day2 SUNDAY = new Day2();
7     public static final Day2 MONDAY = new Day2();
8     public static final Day2 TUESDAY = new Day2();
9     public static final Day2 THURSDAY = new Day2();
10    public static final Day2 WEDNESDAY = new Day2();
11 }
```

اگرچه در تعریف مقادیر داخل enum از کلمات static، final، public استفاده نمی شود اما این کلمات به صورت تلویحی در تعریف مقادیر enum وجود دارند. همچنین در تعریف مقادیر یک enum هیچگاه آبجکتی از روی آن enum ایجاد نمی شود اگرچه به صورت تلویحی مقادیر enum آبجکتهایی از جنس همان enum هستند.

مقادیر ثابت تعریف شده در enum در واقع فیلدهای استاتیکی از جنس همان enum هستند. اگر یک enum با نام E داشته باشیم دو متدهای static values() و valueOf(String name) در آن اضافه می شوند.

```
public static E[] values()

public static E valueOf(String name)
```

اولی، تمام مقادیر ثابت تعریف شده در همان enum را در قالب یک آرایه برمیگرداند و دومی با دریافت یک رشته، مقدار ثابت معادل آنرا برمی گرداند. اگر نامی که به عنوان پارامتر به این متدها ارسال می شود منطبق بر هیچ یک از مقادیر ثابت تعریف شده در enum نباشد خطای `IllegalArgumentException` تولید می شود.

توجه کنید که طول آرایه ای که توسط متدهای `values()` و `valueOf()` برگردانده می شود برابر تعداد مقادیر ثابت تعریف شده در آن enum است و به این ترتیب نیازی به پیاده سازی متدهای `getCount()` یا `getSize()` که پیش از این در بالا تعریف کردیم وجود ندارد.

مقادیر ثابت تعریف شده در یک enum فقط یک بار در حافظه ایجاد می شوند و تا پایان اجرای برنامه در حافظه باقی می مانند. به همین دلیل یک enum نمی تواند حاوی متدهای `finalize()` و `finalizer()` باشد. (پیاده سازی `finalize()` در یک کلاس امکان می دهد تا بتوانیم منطق مورد نظرمان را در زمان حذف یک آبجکت از حافظه تعریف کنیم تا درست قبل پاک شدن یک آبجکت از حافظه توسط ماشین مجازی جاوا فرداخوانی شود)

همچنین همه enum ها به صورت final تعریف می شوند هرچند امکان نوشتن final در تعریف enum وجود ندارد. و به همین دلیل که آنها final هستند نمی توان آنها را abstract تعریف کرد یا حتی توسعه داد.

یک enum می تواند فیلد، متدها یا سازنده داشته باشد که باید بعد از تعریف مقادیر ثابت آن enum بیاند. به عنوان نمونه، تصور کنید enum فوق یک فیلد name نیز داشته باشد

```
1 package ir.atlassoft.javase.chapter9;
2
3 public enum Day3{
4     FRIDAY,
5     SATURDAY,
```

Enumeration

```
6     SUNDAY,  
7     MONDAY,  
8     TUESDAY,  
9     THURSDAY,  
10    WEDNESDAY;  
11    String name;  
12 }
```

۹-۴ کد

طبعی است که هر کدام از مقادیر ثابت Day دارای یک فیلد name هستند که باید به شیوه ای مقداردهی شود که می تواند از طریق سازنده انجام به صورت زیر انجام شود.

```
1 package ir.atlassoft.javase.chapter9;  
2  
3 public enum Day4{  
4     FRIDAY("Friday") ,  
5     SATURDAY("Saturday") ,  
6     SUNDAY("Sunday") ,  
7     MONDAY("Monday") ,  
8     TUESDAY("Tuesday") ,  
9     THURSDAY("Thursday") ,  
10    WEDNESDAY("Wednesday") ;  
11  
12    String name;  
13  
14    Day4(String name){  
15        this.name=name;  
16    }  
17 }
```

۹-۵ کد

با اضافه شدن سازنده به enum فوق، تعریف مقادیر ثابت enum نیز تغییر کرده است و در مقابل هر مقدار ثابت، مقدار فیلد name نیز مشخص گردیده است. در واقع همانطور که پیش از این گفته شد، مقادیر ثابت هر آبجکتهايی از جنس همان enum هستند. کد ۹-۶ یک کلاس معادل enum فوق را نشان می دهد با دقت در این کلاس، شما درک بهتری از آنچه در یک enum رخ می دهد بدست می آورید.

```
1 package ir.atlassoft.javase.chapter9;  
2
```

```

3  public class Day5 {
4      public final static Day5 FRIDAY = new Day5("Friday");
5      public final static Day5 SATURDAY = new Day5("Saturday");
6      public final static Day5 SUNDAY = new Day5("Sunday");
7      public final static Day5 MONDAY = new Day5("Monday");
8      public final static Day5 TUESDAY = new Day5("Tuesday");
9      public final static Day5 THURSDAY = new Day5("Thursday");
10     public final static Day5 WEDNESDAY = new Day5("Wednesday");
11
12     String name;
13
14     Day5(String name) {
15         this.name = name;
16     }
17
18 }

```

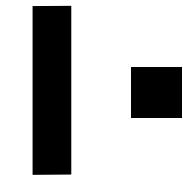
کد ۹-۶

چند نکتهٔ پایانی

امکان پیاده سازی کردن متدهای compareTo()، equals()، hashCode()، clone() وجود ندارد زیرا این متدها در کلاس java.lang.Enum به صورت final پیاده سازی شده‌اند متد () در هر enum ترتیب هر مقدار ثابت enum را برمی‌گرداند مثلاً ordinal() عدد ۲ برمی‌گرداند (اولین مقدار ثابت یک enum مقدار () صفر دارد) برای دسترسی به کلاس یک enum به جای متد () getClass() از متد () getDeclaringClass() استفاده کنید.

تمرینات

(۱) enum ای پیاده سازی کنید که معرف «شیوه پرداخت» باشد که مقادیر ثابت نقدی (Cash)، اعتباری (Credit)، چک (Cheque) را تعریف می کند. در این enum فیلد String name را در آن تعریف کنید که به هر مقدار enum یک نام اختصاص می دهد. برای آن سازنده و متدهای valueOf(String s) پیاده سازی کنید که با دریافت نام یک مقدار ثابت، مقدار ثابت متناظر با آنرا برمی گرداند.



کنترل خطای و استثنا

به هر حادثه‌ای که باعث توقف اجرای برنامه یا مانع عملکرد صحیح برنامه شود خطای گفته می‌شود به عنوان مثال، فرض کنید یک برنامه محاسباتی، در حین اجرا، برای انجام محاسبات خود به حافظه بیشتری نیاز داشته باشد و اندازه حافظه کامپیوتر شما نیز محدود باشد یا به هر دلیلی برنامه نتواند از حافظه بیشتری از سیستم استفاده کند، در این صورت یک خطای سیستمی رخ داده است، محاسبه به درستی انجام نشده و اجرای برنامه متوقف خواهد شد.

به عنوان مثالی دیگر، تصور کنید برنامه‌ای نوشته‌اید که محتویات یک فایل را که روی دیسک کامپیوتر قرار دارد نمایش می‌دهد اگر در زمان اجرای برنامه، این فایل روی دیسک کامپیوتر شما وجود نداشته باشد در این صورت، برنامه شما از یافتن آن فایل عاجز می‌ماند، و اجرای برنامه دچار اختلال می‌شود.

در دو مثال فوق، دو اتفاق مختلف اجرای برنامه را دچار اختلال کرده اند، در مثال اول اجرای برنامه به علت کمبود حافظه سیستم نمی‌تواند ادامه یابد اما در مثال دوم، یک شرایط ویژه برای برنامه بوجود آمده است که مانع عملکرد صحیح برنامه می‌شود اما برنامه ممکن است بتواند تحت این شرایط نیز به کار خود ادامه دهد مثلاً آن فایل اهمیت زیادی نداشته باشد و وجود نداشتن آن مانع عملکرد اصلی برنامه نشود. به «کمبود حافظه» در مثال اول که مانع عملکرد اصلی برنامه می‌شود «خطای گفته می‌شود اما به «نبودن فایل» که یک وضعیت غیرمعمول در اجرای برنامه دوم ایجاد می‌کند ولی منجر به توقف اجرای آن برنامه نمی‌شود یک «استثنا» گفته می‌شود.

به این ترتیب «خطا» و «استثنا» به دو رویداد غیرمعمول در اجرای برنامه‌ها گفته می‌شود. در حالیکه خطا باعث توقف اجرای برنامه می‌شود، استثنا باعث توقف اجرای برنامه نمی‌شود اما ممکن است مانع عملکرد صحیح برنامه می‌شود.

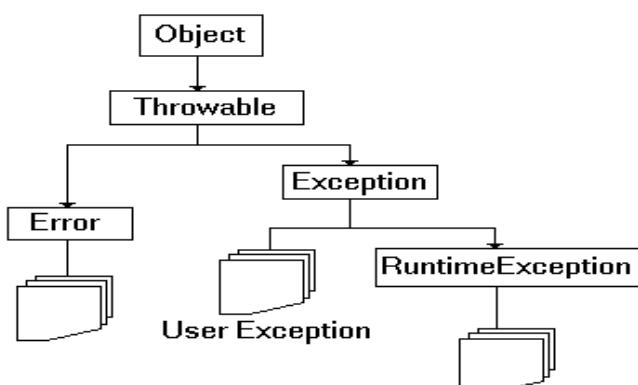
Exception Handling که ما در این کتاب به آن «کنترل استثنا» می‌گوییم در واقع مکانیسمی است که با استفاده از آن می‌توانید عکس العمل برنامه نسبت به وقوع یک استثنا را مشخص کنید.

توجه کنید که استثنا می‌تواند شامل هر حادثه ناخواسته‌ای باشد مثلاً حتی وارد کردن اطلاعات نادرست توسط کاربر را نیز می‌توان به عنوان یک استثنا در نظر گرفت و با استفاده از **Exception Handling** آنرا کنترل و مدیریت نمود.

از آنجاییکه جاوا یک زبان شیء گر است، کاملاً طبیعی است که انتظار داشته باشیم حوادث استثنا توسط کلاس یا کلاسهای جاوا تعریف شوند تا زمانیکه یک حادثه استثنا اتفاق می‌افتد با ایجاد آبجکتی از یک کلاس استثنا، رخداد آن حادثه را اعلان کنیم.

در جاوا کلاس `Exception` که در پکیج `java.lang` تعریف شده است کلاس اصلی حادثه استثناست. معنی این جمله این است که هر حادثه استثنا توسط یک آبجکت از کلاس `Exception` یا مشتقات این کلاس نشان داده می‌شود.

کلاسهایی که از کلاس `Exception` مشتق می‌شوند، هر کدام برای نشان دادن یک حادثه خاص پیاده سازی شده‌اند. شکل زیر کلاس `Exception` و برخی ارتباطات آنها را با یکدیگر نشان می‌دهد.



شگل ۱-۰۱. اشتراق کلاسهای خطأ در جاوا

کلاس `Throwable` پدر همه کلاسهای خطأ و استثناست، در واقع این کلاس در برگیرنده تمام خصوصیات مشترک دو فرزند خود یعنی کلاس `Error` و `Exception` است که به ترتیب معرف رخداد یک حادثه استثنا یا خطأ هستند.

کنترل خطای استثنای

از آنجاییکه خطای باعث توقف اجرای برنامه می شوند، برنامه نمی تواند نسبت به رخداد آن عکس العمل نشان دهد و بنابراین کنترل آنها از عهده برنامه نویسان خارج است. در نقطه مقابل، کنترل استثنایها که در زمان اجرا از طریق آبجکتهای Exception در برنامه ظاهر می شوند بخش مهمی از برنامه نویسی محسوب می شود. حال که با تئوری استثنای آشنا شدید، اجازه بدھید ببینیم چگونه باید یک استثنای تعريف کنیم و چگونه می توانیم رخداد آنرا کنترل کنیم.

استفاده از Exception در کنترل خطای

فرض کنید در برنامه شما کلاس Printer پیاده سازی شده است که کار چاپ مستندات را برعهده دارد. این کلاس حاوی یک متده است که کار چاپ اسناد را انجام می دهد. هر سند در قالب یک آبجکت Document به این متده ارسال می شود، تا توسط این متده به پرینتر ارسال شود. بدون اینکه وارد جزئیات کلاس فرضی Document شویم کلاس Printer را به صورت زیر پیاده سازی می کنیم.

```
1 package ir.atlassoft.javase.chapter10;  
2  
3 public class Printer{  
4     public void print(Document doc){  
5         //do print job  
6     }  
7 }
```

۱۰-۱ کد

به این ترتیب، برای چاپ یک سند، کدی به شکل زیر در برنامه پیاده سازی خواهیم کرد.

```
Document doc = ...  
Printer printer = new Printer();  
printer.print(doc);
```

تا اینجا همه چیز منطقی و صحیح است تنها ایرادی که به کد فوق وارد است این است که اگر پرینتر در دسترس نباشد (مثلا سیم آن قطع باشد، یا خاموش باشد، ...) برنامه باید چه عکس العملی نشان دهد در واقع متده print() که وظیفه چاپ اسناد را به عهده دارد، «وقوع خطای در چاپ سند» را چگونه باید گزارش کند؟ به این منظور، متده print() را به شکل زیر تغییر می دهیم تا اشکال در دسترس نبودن پرینتر را لاحظ کنیم.

```
1 package ir.atlassoft.javase.chapter10;  
2  
3 public class Printer{  
4     public void print(Document doc) {
```

```

5         if(isPrinterAvailable()) {
6             System.out.println(
7                 "Printer is Unavailable");
8         } else {
9             //do print job
10        }
11    }
12    private Boolean isPrinterAvailable() {
13        //check printer availability
14    }

```

کد ۱۰-۲

همانطور که ملاحظه می کنید، قبل از انجام چاپ در متدها `print()` و `isPrinterAvailable()` در دسترس بودن پرینتر بررسی شده است تا اگر پرینتر در دسترس نباشد، جمله `Printer is Unavailable` در کنسول برنامه نمایش داده می شود و در غیراینصورت سند `doc` برای چاپ به پرینتر ارسال شود.

اگرچه در گذشته، شرط استثناء یعنی «در دسترس نبودن پرینتر» بررسی و کنترل شده است اما نمایش پیغام در کنسول برنامه عکس العمل مناسبی نیست، زیرا در صورتی که پرینتر در دسترس نباشد، آبجکتی که متدهای `print()` را فراخوانی کرده است متوجه این موضوع نمی شود! وقتی اجرای متدهای `print()` به اتمام برسد، آبجکتی که این متدهای فراخوانی کرده است بدون توجه به اینکه در کنسول برنامه پیغامی نمایش داده شده است یا خیر، تصور می کند که کار چاپ با موفقیت خاتمه یافته است. بنابراین در روش صحیح کنترل استثناء، باید آبجکتی که متدهای `print()` را فراخوانی کرده است از بروز حادثه استثنای مطلع شود. با این توضیح، اجازه دهید متدهای `print()` را به گونه ای تغییر دهیم تا فراخواننده متدهای `print()` از رخداد استثنای مطلع شود.

```

1 package ir.atlassoft.javase.chapter10;
2
3 public class Printer2{
4     public int print(Document doc) {
5         if(!isPrinterAvailable()) {
6             return 0;
7         } else {
8             //do print job
9             return 1;
10        }
11    }
12    private Boolean isPrinterAvailable() {
13        //check printer availability

```

کنترل خطأ و استثناء

```
14 }  
15 }
```

کد ۱۰-۳

برای این منظور، مقدار برگشتی متد () print را از void به int تغییر داده ایم تا وقتی چاپ با موفقیت انجام شود عدد ۱ و هنگام دردسترس نبودن پرینتر عدد ۰ را برگرداند. آبجکتی که متد () print را فراخوانی می کند از روی مقداری که این متد برمی گرداند متوجه موفقیت یا شکست چاپ می شود. با کدی که در بالا نوشتمیم، شرط استثنای دردسترس نبودن پرینتر را کنترل کردیم ولی از مکانیسمی که جاوا برای کنترل استثنای فراهم کرده است استفاده نکردیم. در واقع به جای برگرداندن مقدار ۰ که نشانگر رخداد استثنای استثنای exception می توانیم از آبجکتی از کلاس Exception استفاده کنیم. کنترل استثنای در جاوا مکانیسم ساده ای دارد، هر حادثه استثنای را با آبجکتی از کلاس Exception نشان می دهیم.

```
Exception exception = new Exception();
```

و با استفاده از کلمه کلیدی throw به اجرای متد خاتمه می دهیم.

```
throw exception;
```

جمله فوق علاوه بر خاتمه اجرای متد، آبجکت exception را پرتاب می کند تا توسط فراخواننده متد print() دریافت شود. کد زیر متد () print را که براساس مکانیسم کنترل استثنای تغییر یافته است نشان می دهد.

```
1 package ir.atlassoft.javase.chapter10;  
2  
3 public class Printer3{  
4     public void print(Document doc)  
5         throws Exception{  
6             if(!isPrinterAvailable()) {  
7                 Exception ex = new Exception();  
8                 throw ex;  
9             } else{  
10                 //do print job  
11             }  
12         }  
13     private Boolean isPrinterAvailable(){  
14         //check printer availability
```

```

15      }
16
17 }
```

کد ۱۰-۴

به این ترتیب، آجکتی که متدها print() را فراخوانی می‌کند، ممکن است مقدار برگشتی متدها را دریافت کند (چاپ سند با موفقیت انجام شود) یا یک آجکت Exception دریافت کند (پرینتر در دسترس نباشد). با دققت در کد فوق متوجه تغییر دیگری در متدها print() می‌شوید.

```
public void print(Document doc) throws Exception{
```

در تعریف متدها print()، عبارت throws Exception اضافه شده است که همانند یک علامت عمل می‌کند و به کامپایلر جاوا اعلان می‌کند که فراخوانی متدها print() ممکن است باعث رخداد یک وضعیت استثنای شود. بنابراین هر جایی از برنامه که متدها print() فراخوانی می‌شود باید با کنترل استثناء، عکس العمل برنامه نسبت به رخداد احتمالی استثنای مشخص شود. اکنون که متدها print() تکمیل شده است اجازه دهید به جایی برویم که متدها print() فراخوانی می‌شود.

```
Document doc = ...
Printer3 printer = new Printer3();
try{
    printer.print(doc);
} catch(Exception ex){
    System.out.println("Printing Failed");
}
//program continues...
```

همانطور که ملاحظه می‌کنید متدها print() داخل بلوك try فراخوانی شده است

```
try{
    printer.print(doc);
}
try در تعریف آن وجود داشته باشد باید داخل بلوك
    فراخوانی شود. بلافصله بعد از بلوك try، بلوك catch قرار می گیرد.
```

```
catch(Exception ex){
    System.out.println("Printing Failed");
}
```

به بلوكهای try و catch که پشت سر هم قرار می‌گیرند به صورت مختصر بلوك try/catch گفته می‌شود و معنی آن است که اجرا کننده جاوا باید بلوك try را اجرا کند اما اگر در اجرای بلوك try با استثنای مواجه شود باید بلافصله اجرای بلوك try را متوقف و بلوك catch را اجرا کند. اگر در اجرای بلوك

کنترل خطا و استثنای

استثنایی اتفاق نیفتند بلوک try نادیده گرفته می شود و اجرای برنامه بعد از بلوک catch یعنی //program continues ادامه می یابد.

شکل کلی بلوک try/catch به صورت زیر است

```
try{  
    ...  
}catch(Exception ex){  
    ...  
}
```

با دقت در بلوک catch این سوال مطرح می شود که (Exception ex) چیست؟ این همان آبجکت است که در صورت رخداد استثنای در متدها print() توسط جمله throw پرتاب شد و در اینجا دریافت می شود.

آبجکت در جایی پرتاب و در بلوک catch دریافت می شود می تواند حاوی اطلاعات مفیدی باشد که در بلوک استفاده شود. به عنوان مثال، این آبجکت می تواند حاوی یک متن توضیحی باشد که جزئیات استثنای رخداده را شرح می دهد. به کلاس تغییر یافته Printer4 زیر توجه کنید.

```
1 package ir.atlassoft.javase.chapter10;  
2  
3 public class Printer4{  
4     public void print(Document doc)  
5         throws Exception{  
6             if(!isPrinterAvailable()) {  
7                 Exception ex = new  
8                     Exception("Printer is not available!");  
9                 throw ex;  
10            } else{  
11                //do print job  
12            }  
13        }  
14        private Boolean isPrinterAvailable()  
15            //check printer availability  
16        }  
17    }  
18 }
```

در اینجا، ایجاد آبجکت `Exception` همراه با یک متن توضیحی (پارامتری که به سازنده کلاس ارسال شده است) انجام شده است تا در بلوک `catch` که آبجکت `Exception` فوق دریافت می‌شود از متن توضیحی به صورت زیر استفاده شود.

```
Document doc = ...
Printer4 printer = new Printer4();
try{
    printer.print(doc);
} catch(Exception ex){
    System.out.println(ex.getMessage());
}
//program continues...
```

همانطور که ملاحظه می‌کنید از متدهای `getMessage()` از آبجکت `Exception` برای دسترسی به متن توضیحی آن استفاده شده است.

توسعه کلاس `Exception`

کلاس `Exception` یک کلاس عمومی است که از آن برای نشان دادن هر نوع شرایط استثنایی می‌توان استفاده نمود. در اغلب اوقات برای نشان دادن شرایط استثناء مختلف و تمایز بین شرایط استثناء کلاس `Exception` را توسعه می‌دهیم و کلاس استثناء خاص تری تعریف می‌کنیم. مثلاً برای نشان دادن شرط استثنای «پرینت ناموفق» کلاس استثناء `PrintException` را به صورت زیر تعریف می‌کنیم:

```
1 package ir.atlassoft.javase.chapter10;
2
3 public class PrintException extends Exception {
4 }
```

کد ۱۰-۶

اگرچه کلاس `PrintException` هیچ رفتار و فیلد اضافه‌ای نسبت به `Exception` ندارد و صرفاً آنرا توسعه داده است، اما بسته به نیاز می‌توان فیلدهایی را در آن تعریف نمود. اما حتی اگر هیچ فیلد یا متدهای در آن تعریف نشود، وجود چنین کلاسی گویای استثنای مشخصی است که در چاپ یک سند ایجاد می‌شود. با تعریف کلاس فوق، کلاسهای `Printer` و فراخواننده آن به صورت زیر تغییر می‌کنند.

```
1 package ir.atlassoft.javase.chapter10;
2
3 public class Printer5{
4     public void print(Document doc)
```

کنترل خطا و استثنای

```
5           throws PrintException{
6             if(isPrinterAvailable()){
7               throw new PrintException();
8             }else{
9               //do print job
10            }
11          }
12      private Boolean isPrinterAvailable(){
13        //check printer availability
14      }
15  }
```

کد ۱۰-۶

فراخوانی متد () print() تغییر می کند.

```
Document doc = ...
Printer5 printer = new Printer5();
try{
  printer.print(doc);
}catch(PrintException ex){
  System.out.println(ex.getMessage());
}
```

نکته مهم در مثال فوق این است که، اگرچه در متد () print() استثناء PrintException تولید می شود
اما متد () print() می تواند همانند قبل آبجکت Exception پرتاب کند

```
1 package ir.atlassoft.javase.chapter10;
2
3 public class Printer6{
4   public void print(Document doc)
5     throws Exception{
6       if(isPrinterAvailable()){
7         throw new PrintException();
8       }else{
9         //do print job
10      }
11    }
12  private Boolean isPrinterAvailable(){
13    //check printer availability
14  }
15 }
```

کد ۱۰-۷

چنین چیزی قابل قبول است زیرا هر کلاسی که از `Exception` مشتق شده باشد از جنس `Exception` هم محسوب می شود به این ترتیب با وجود `throws Exception` در تعریف متد، هر استثنایی که از `Exception` مشتق شده باشد می تواند رخ دهد. عکس این جمله درست نیست و مثلاً متد `()` `print` نمی تواند به صورت زیر تعریف شود

```

1 package ir.atlassoft.javase.chapter10;
2
3 public class Printer7{
4     public void print(Document doc)
5         throws PrintException {
6         if(!isPrinterAvailable()){
7             throw new Exception();
8         }else{
9             //do print job
10        }
11    }
12    private Boolean isPrinterAvailable() {
13        //check printer availability
14    }
15}
16}
```

۱۰-۹

در اینجا در حالیکه در متد `print()`، استثنایی از جنس `Exception` تولید شده است در تعریف متد استثنایی خاص تر از جنس `PrintException` نشان داده شده است. به عبارت ساده، تعریف متد انعکاس دهنده تمام انواع استثناهایی که در متد ممکن است اتفاق بیفتد نیست زیرا `PrintException` زیرمجموعه ای از `Exception` محسوب می شود.

توجه کنید که همواره استثناهایی که توسط یک متد گزارش می شوند باید در برگیرنده تمام استثناهای داخل متد باشند.

از طرف دیگر، جایی که متد `print()` فراخوانی می شود بلوک `catch` می تواند هر استثنایی را دریافت کند ولی استثنای هرچه که باشد باید در برگیرنده تمام انواع استثناهای گزارش شده در تعریف متد `print()` را شامل شود بنابراین اگر در متد `print()` استثنای `print` `Exception` گزارش شده باشد حتی با اطمینان از اینکه در بدنهٔ متد `print()` فقط استثنای `PrintException` تولید می شود نمی توان در بلوک `catch` استثنای `PrintException` را دریافت نمود، زیرا نادرست است و خطای کامپایل تولید می کند

کنترل خطا و استثنای

```
public void print(Document doc) throws Exception{  
...  
}  
  
try{  
    printer.print(doc);  
}catch(PrintException ex){  
    System.out.println(ex.getMessage());  
}
```

فراخوانی متد print()

اما کد زیر صحیح است

```
public void print(Document doc) throws PrintException{  
...  
}  
  
try{  
    printer.print(doc);  
}catch(Exception ex){  
    System.out.println(ex.getMessage());  
}
```

جایی که متد print() فراخوانی می شود:

تولید بیش از یک استثنای در یک متد

وقتی یک متد اجرا می شود ممکن است شرایط استثنای مختلفی اجرای صحیح متد را مختل کنند و ممکن است بخواهیم بسته به نوع استثنایی که رخ می دهد عکس العمل های متفاوتی نشان دهیم. مثلاً تصور کنید که متد print() با دو استثنای مختلف «در دسترس نبودن پرینتر» و «نبودن کاغذ در پرینتر» مواجه شود اگرچه هر دو استثنای مانع عملکرد پرینت و اجرای صحیح متد print() می شود اما لازم است پیغام های متفاوتی به کاربر نشان داده شود.

راه حل این مسئله، پیاده سازی دو کلاس استثنای مختلف است.

```
1 package ir.atlassoft.javase.chapter10;  
2 public class PrinterUnavailableException extends Exception{  
3 }  
4  
5 package ir.atlassoft.javase.chapter10;  
6 public class PrinterNoPaperException extends Exception{  
7 }
```

کد ۱۰-۱۰

حال در متد print() بسته به رخداد هر استثنای، آبجکتی از کلاس متناسب تولید و پرتاب می شود

```

1 package ir.atlassoft.javase.chapter10;
2
3 public class Printer8{
4     public void print(Document doc)
5             throws PrinterUnavailableException,
6             PrinterNoPaperException {
7         if(!isPrinterAvailable()){
8             throw new PrinterUnavailableException();
9         }else if(!printerHasPaper()){
10            throw new PrinterNoPaperException();
11        }else{
12            //do print job
13        }
14    }
15    private Boolean isPrinterAvailable() {
16        //check printer availability
17    }
18    private Boolean printerHasPaper() {
19        //check if printer has paper
20    }
21 }
```

کد ۱۰-۱۱

توجه کنید که در تعریف متدها print() دو خطأ گزارش شده اند معنی آن این است که جایی که متدهای print() فراخوانی می شود به صورتیکه در گذشت زیر ملاحظه می کنید می توان هر دو استثنای را دریافت نمود.

```

try{
    printer.print(doc);
} catch(PrinterUnavailableException ex){
    System.out.println("Printer Not Available");
} catch(PrinterNoPaperException ex){
    System.out.println("No Paper!");
}

البته ممکن است جاییکه متدهای print() فراخوانی می شود تصمیم بگیرید با هر دو استثنای یک شکل
یکسان برخورد کنید در اینصورت می توانید گذشت زیر فوق را به صورت زیر تغییر دهید.

try{
    printer.print(doc);
} catch(PrinterUnavailableException ex|PrinterNoPaperException ex){
    System.out.println("Printing Failed");
}
```

کنترل خطا و استثنای

در اینجا با استفاده از عملگر `|` مشخص کرده ایم که رخداد هریک از استثنایات `PrinterNoPaperException` یا `PrinterUnavailableException` باید به یک شکل برخورد شود. همچنین بلوکهای `try/catch` فوق را می توان به شکل زیر نیز پیاده سازی کرد.

```
try{
    printer.print(doc);
} catch(Exception ex) {
    System.out.println("Printing Failed");
}
```

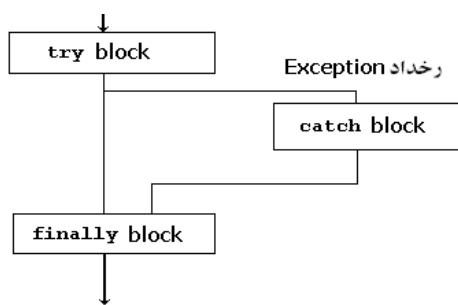
در اینجا بلوک `catch` مشخص می کند که با رخداد هر استثنایی از جنس کلاس `Exception` یا هر کلاسی که از کلاس `Exception` مشتق شده باشد باید به شکل یکسانی برخورد شود.

بلوک finally

در بلوکهای `try/catch` به صورت اختیاری می توان پس از آخرین بلوک `catch` یک بلوک `finally` به صورت

```
finally{
    //codes
}
```

اضافه کرد.
چه فراخوانی متده با موفقیت انجام شود(که در این صورت بلوک `try` اجرا خواهد شد) و چه فراخوانی متده منجر به بروز استثنای شود (که در این صورت یکی از بلوکهای `catch` اجرا خواهد شد)، در هر صورت بلوک `finally` اجرا خواهد شد.
یعنی بدون توجه به اینکه بلوک `try` با موفقیت اجرا شود یا استثنایی اتفاق بیفتد، بلوک `finally` اجرا خواهد شد.



شکل ۱۰-۳. روال اجرای بلوکهای `try` و `catch`

سوالی که در اینجا مطرح است این است که بدون وجود بلوک finally هم کدهایی که بعد از آخرین بلوک catch نوشته می شوند اجرا می شوند، در اینصورت چه لزومی به استفاده از بلوک finally وجود دارد؟! نکته اینجاست که در برخی سناریوهای کدهای بعد از این دو بلوک اجرا نمی شوند. برای درک این موضوع به متدهای showAndPrint() که در زیر نشان داده شده است توجه کنید

```
public void showAndPrint(){
    try{
        printer.print(doc);
        return;
    }catch(Exception ex){
        System.out.println("Printing Failed");
    }
    //some codes
}
```

در اینجا، دست بعد از پرینت یک سند که از طریق جمله `printer.print(doc);` در انتهای بلوک try قرار دارد. در اینجا اگر هیچ استثنایی در اجرای بلوک try رخ ندهد، جمله `return;` به اجرای متدهای `showAndPrint()` خاتمه می دهد و جملات `//some codes` که بعد از بلوک catch قرار دارند هرگز اجرا نخواهند شد. حتی اگر جمله `return;` در بلوک catch وجود داشته باشد، با رخداد یک استثنای `Exception` که بعد از بلوک catch قرار دارند اجرا نخواهند شد و متدهای `showAndPrint()` در بلوک catch خاتمه می یابد.

به این ترتیب کاملاً محتمل است که اجرای متدهای `showAndPrint()` در بلوک های `try` یا `catch` خاتمه پیدا کند و جملات بعد از بلوک `catch` اجرا نشوند.

بلوک `finally` تضمین می کند که فارغ از اینکه استثنایی رخ می دهد یا نمی دهد کدهای `//some codes` اجرا می شود بنابراین اگر به هر طریقی اجرای یک متدهای `showAndPrint()` خاتمه یابد درست قبل از خاتمه متدهای `finally` اجرا می شوند. آزاد کردن حافظه یا دیگر منابع سیستم از کاربردهای معمول بلوک `finally` است. در این مثال ممکن است از طریق بلوک `finally` بخواهیم پرینتر (فارغ از اینکه چاپ سند با موفقیت انجام شده یا استثنایی رخ داده است) برای پرینتهای بعدی آزاد کنیم.

```
try{
    printer.print(doc);
    return;
}catch(Exception ex){
    System.out.println("Printing Failed");
}finally{
    printer.close();
}
```

توجه: بلوک try می تواند بدون بلوک catch باشد اما در اینصورت حتماً باید بلوک finally داشته باشد. منطقی هم نیست که بلوک try همزمان بدون بلوک catch و finally باشد.

اینترفیس AutoCloseable

همانطور که گفته شد، یکی از کاربردهای بلوک finally، آزاد کردن حافظه و دیگر منابع برنامه است. این کار با فراخوانی `finally` در بلوک `printer.close()` در `finally` در مثال این فصل انجام شد. مشابه کلاس `Printer` که دارای متدهای `close()` برای آزاد کردن منابع پرینتر است و دقیقاً با هدف فراخوانی در بلوک `finally` طراحی و پیاده سازی شده است، کلاسهای مختلفی در جاوا وجود دارند که همگی دارای متدهای `close()` با همین هدف هستند. اینها باعث شده که بلوکهای `try/finally` در جاوا به شکل روتین و تکراری در آیند. همین موضوع طراحان جاوا را بر آن داشت تا مکانیسمی به جاوا اضافه کنند تا از کدنویسی اضافی و تکراری جلوگیری کند و برنامه نویسان را از کدهای تکراری و خسته کننده رهایی دهد. برای این منظور اینترفیس `java.lang.AutoCloseable` ارایه شد، تا در صورتیکه کلاسی این اینترفیس را پیاده سازی کند دیگر نیازی به نوشتن بلوک `finally` برای فراخوانی متدهای `close()` آن نباشد.

اگر بخواهیم مثال این فصل را با اینترفیس `AutoCloseable` ترکیب کنیم، اولاً باید کلاس `Printer` این اینترفیس را پیاده سازی کند که در اینصورت می‌بایست الزاماً متدهای `close()` را پیاده سازی کند.

```
1 package ir.atlassoft.javase.chapter10;
2
3 public class Printer9 implements AutoCloseable{
4     public void print(Document doc)
5             throws PrinterUnavailableException,
6             PrinterNoPaperException {
7         if(!isPrinterAvailable()){
8             throw new PrinterUnavailableException();
9         }else if(!printerHasPaper()){
10             throw new PrinterNoPaperException();
11         }else{
12             //do print job
13         }
14     }
15     private Boolean isPrinterAvailable(){
16         //check printer availability
17     }
```

```

18     private Boolean printerHasPaper(){
19         //check if printer has paper
20     }
21     public void close(){
22         //free memory and resources
23     }
24 }
```

کد ۱۰-۱۲

حال برای پرینت یک سند به صورت زیر عمل می کنیم.

```

try (Printer printer=new Printer()){
    printer.print(doc);
} catch(Exception ex){
    System.out.println("Printing Failed");
}
```

در اینجا اولاً ایجاد آبجکت **Printer** داخل پرانتز و در مقابل **try** انجام شده است. ثانیاً از نوشتن بلوک **finally** خودداری کرده ایم و انتظار داریم تا بعد از اجرای بلوکهای **try/catch** به صورت خودکار متند **()** از آبجکت **Printer** فراخوانی شود. جالب است بدانید که فقط در چنین شرایطی است که می توان از نوشتن بلوک **catch** خودداری کرد، زیرا در اینجا به صورت تلویحی بلوک **finally** وجود دارد. به این ترتیب کد زیر نیز صحیح است.

```

try (Printer printer=new Printer()){
    printer.print(doc);
}
```

کلاس‌های **InputStream** اینترفیس **AutoClosable** را پیاده سازی کرده اند.

متدهای کلاس Exception

اگر داخل بلوک **try** استثنایی رخد دهد، اجرای بلوک **try** متوقف می‌شود و بسته به نوع دریافت شده، بلوک **catch** متناظر با آن اجرا می‌شود. آبجکت **Exception** تولید شده در بلوک **catch** قابل دستیابی است:

```

catch(Exception e) {
    //codes
}
```

این آبجکت که نام آن **e** است در واقع همان آبجکت استثنایی است که در متند **print()** ایجاد و پرتاب شده است بنابراین از آن می‌توانید اطلاعات مورد نیاز در مورد علت و نحوه بروز خطأ را استخراج نموده و از روی آن پیغام مناسبی تولید کنید.

تولید خطاهای زنجیره ای

استثناهای سطح پایین باعث رخداد استثناهای سطح بالا می شوند برای درک این مطلب، به مثال زیر توجه کنید.

در مثال پرینت که تا اینجا در مورد آن صحبت کرده ایم، فرض کنید داده های برنامه در قالب متن ساده، عکس، ... باشد و بنابراین لازم است قبل از اینکه متدها print() را فراخوانی کنیم از روی داده هایی که قصد چاپ آنها را داریم آبجکت Document ایجاد کنیم. بدون اینکه بخواهیم در گیر نحوه تبدیل داده های متنوع برنامه را به آبجکت فرضی Document شویم، فرض کنید که این کار را در کلاس دیگری با نام ReportGenerator انجام می دهیم که به عنوان واسطه بین قسمتهای دیگر برنامه و کلاس Printer عمل می کند تا قبل از اینکه متدها print() فراخوانی شود داده های مورد نظر در قالب یک آبجکت Document برای فراخوانی متدها print() آماده شوند. کلاس ReportGenerator شکلی به صورت زیر خواهد داشت

```
1 package ir.atlassoft.javase.chapter10;  
2  
3 public class ReportGenerator{  
4  
5     Printer9 printer = new Printer9();  
6  
7     public void report(byte[] data) throws ReportException{  
8         Document doc = new Document(data);  
9         try{  
10             printer.print(doc);  
11         } catch(Exception ex){  
12             throw new ReportException(ex);  
13         }  
14     }  
15 }
```

کد ۱۰-۱۳

همانطور که ملاحظه می کنید در بلوک catch وقتی یک استثنای دریافت شده است، استثناء جدیدی از ReportException تولید و پرتاب شده است. معنی آن این است که رخداد یک استثنای خود می تواند باعث رخداد استثناء دیگری شود حال جایی که متدها report() فراخوانی شده است استثنایی از جنس ReportException دریافت می کند که ریشه آن استثنایی است که در کلاس Printer رخداده است.

توجه داشته باشید، که در مثال فوق، متدهای report() و print() دریافت می‌کند را خود پردازش کند (به شکلی که در فوق ملاحظه می‌کنید) یا اینکه مستقیماً آنرا به فراخواننده خود (یعنی فراخواننده متدهای report() و print()) ارسال کند. کد زیر را ملاحظه کنید:

```

1 package ir.atlassoft.javase.chapter10;
2
3 public class ReportGenerator1{
4     Printer9 printer = new Printer9();
5     public void report(byte[] data) throws Exception{
6         Document doc = new Document(data);
7         printer.print(doc);
8     }
9 }
10 }
```

کد ۱۰-۱۴

اینکه کدام گزینه را در طراحی خود انتخاب کنید کاملاً بستگی به برنامه و طراحی برنامه شما دارد.

متدهای getMethod()

همانطور که قبلاً دیدید، هر استثنایی که تولید می‌شود می‌تواند حاوی یک جمله توضیحی نیز باشد که در مورد علت بروز استثنای توضیح می‌دهد. این توضیح یکی از خصوصیات کلاس Exception است و بنابراین در هر کلاس استثنایی قابل استفاده است. با کمی تعییرات در کلاسهای استثنایی می‌توان جمله توضیحی را نیز در سازنده‌های کلاسهای استثنایی افزود.

```

1 package ir.atlassoft.javase.chapter10;
2
3 public class PrinterUnavailableException extends Exception{
4
5     public PrinterUnavailableException() {
6     }
7     public PrinterUnavailableException(String message) {
8         super(message);
9     }
10 }
```

کد ۱۰-۱۵

کنترل خطای استثنایی

به این ترتیب در ایجاد آبجکت استثنایی، می‌توان متن توضیحی مربوط به آن استثنایی را نیز مشخص نمود تا جاییکه استثنایی دریافت می‌شود از متن آن برای توضیح علت رخداد استثنایی استفاده نمود.

متدهای getCause()

از آنجاییکه استثنایی می‌توانند به صورت زنجیره ای تولید شوند، به طوریکه هر استثنایی عامل استثنایی دیگری باشد، برای کلاس Exception خصوصیت دیگری نیز با عنوان cause در نظر گرفته شده تا هر استثنایی بتواند علاوه بر متن توضیحی، عامل خود را نیز نگهداری کند. در نقطه شروع زنجیره، جایی که هیچ استثنایی وجود ندارد مقدار cause برای اولین استثنایی تهی (null) خواهد بود. ولی در ادامه زنجیره وقتی یک استثنایی باعث رخداد استثنایی بعدی می‌شود، هر استثنایی بعدی شناسایی می‌شود.

متدهای printStackTrace()

از آنجاییکه هر استثنایی می‌تواند باعث بروز یک استثنایی دیگر شود تا زنجیره ای از استثنایات پشت سر هم قرار گیرند، متدهای printStackTrace() در کلاس Exception پیاده سازی شده است تا با فراخوانی آن، لیست استثنایی که در زنجیره قرار دارند در خروجی استاندارد چاپ شوند. از این متدهای برای دیباگ برنامه ها استفاده می‌شود و عموماً در محیط عملیاتی کاربردی ندارد. حال با استفاده از مفاهیمی که تا این لحظه آموخته اید اجازه دهید، مثال چاپ خود را تکمیل کنیم.

استثنایات «زمان اجرا» و «غیر زمان اجرا»

اما استثنایات خود به دو دسته تقسیم می‌شوند اول، استثنایی که مربوط به زمان اجرای برنامه و شکل داده ها هستند و دوم، شرایط استثنایی که مربوط به منطق برنامه می‌باشند.

- Runtime Exceptions
- Non-Runtime Exceptions

استثنایات زمان اجرا (Runtime Exception) مربوط به دسترسی به داده ها هستند، مثلاً اگر در زمان اجرا عددی را بر صفر تقسیم کنید، یا بخواهید به خارج از محدوده یک آرایه دست پیدا کنید (مثلاً به عنصر ششم از یک آرایه پنج تایی مقدار بدھید)، یا بخواهید به خارج از محدوده یک رشته دست پیدا کنید (مثلاً به کاراکتر ششم از رشته ای که فقط ۵ کاراکتر دارد)، یا بخواهید آبجکتی را به یک کلاس ناسازگار cast کنید، ... همگی منجر به خطای زمان اجرا می‌شوند. کنترل این استثنایات به عهده برنامه نویس نیست، زیرا قبل از اجرای برنامه قابل پیش بینی هستند و برنامه نویسان باید مانع از رخداد چنین استثنایی شوند. استثنایات غیرزمان اجرا، شرایط استثنایی هستند که مربوط به منطق برنامه می‌شوند، اینها استثنایی هستند که باید برای نحوه عکس العمل برنامه در زمان رخداد آنها کدنویسی کنید، به عنوان مثالهایی از

استثناهای غیرزمان اجرا می توان به `SQLException` و `FileNotFoundException` اشاره کرد. زمانی رخ می دهد که بخواهید به یک فایل که وجود ندارد دسترسی پیدا کنید یا محتويات آنرا بخوانید. زمانی رخ می دهد بخواهید به یک پایگاه داده که در حال اجرا نیست، یا در دسترس نیست متصل شوید.

استثناهای زمان اجرا نیاز به بلوک `try/catch` ندارند و برنامه باید بگونه ای پیاده سازی شود که در هیچ شرایطی با خطای زمان اجرا مواجه نشویم. اما خطاهای غیر زمان اجرا همان استثناهایی هستند که در مودر آنها صحبت کردیم و رخداد آنها قبل از اجرای برنامه قابل پیش بینی است، در مورد رخداد آنها و نحوه عکس العمل برنامه باید در زمان توسعه برنامه تصمیم گیری کنیم. برای بسیاری از استثنایات معمول در جاوا کلاسهای `Exception` پیاده سازی شده است که برخی از آنها در جدول زیر لیست شده اند.

توضیح	Exception
خطای محاسباتی، از قبیل تقسیم بر صفر	<code>ArithmeticException</code>
دسترسی به عناصر خارج از محدوده یک آرایه	<code>ArrayIndexOutOfBoundsException</code>
انتساب یک مقدار ناسازگار به یک عنصر آرایه	<code>ArrayStoreException</code>
نادرست یک کلاس <code>cast</code>	<code>ClassCastException</code>
استفاده از یک آرگومان نادرست در فراخوانی یک متند	<code>IllegalArgumentException</code>
ساخت یک آرایه با اندازه منفی	<code>NegativeArraySizeException</code>
فراخوانی یک متند یا دسترسی به یک <code>null</code> از یک آبجکت <code>Property</code>	<code>NullPointerException</code>
تبديل یک رشته به یک عدد در صورتیکه در رشته کاراکتر غیر عددی وجود داشته باشد	<code>NumberFormatException</code>
خطای دسترسی غیرمجاز به یک فایل یا کلاس	<code>SecurityException</code>
دسترسی به کاراکترهای خارج از محدوده یک رشته	<code>StringIndexOutOfBoundsException</code>

جدول ۱۰. برخی استاندارد `Exception` های جاوا

خلاصه

اجرای هر قسمت از برنامه دو حالت دارد، یا آن قسمت بدون هیچ مشکل و نقصی اجرا می شود و همه چیز به خوبی پایان می یابد، یا اینکه در حین اجرای آن قسمت حادثه ای رخ می دهد و اجرای برنامه را دچار اختلال می کند. بحث این فصل روی حالت دوم بود که اجرای برنامه دچار اختلال می شود، در این وضعیت یکی از دو حالت زیر رخ می دهد:

الف- با رخداد حادثه اجرای برنامه متوقف می شود و نمی تواند ادامه یابد. (مثلاً حافظه کافی در اختیار برنامه نیست).

ب- رخداد خطا قسمتی از برنامه را دچار اشکال کرده است، و برنامه می تواند با عکس العمل مناسب همچنان به حیات خود ادامه دهد.

حالت «الف» عموماً در اختیار برنامه نویس نیست، و در این حالت، اجرای JVM با نمایش پیغام خاصی خاتمه می یابد تمام بحث ما در این فصل مربوط به خطاهای حالت «ب» هستند که اگرچه با رخداد آنها اجرای برنامه دچار اختلال می شود اما اجرای برنامه می تواند ادامه یابد. این خطاها نیز به نوبه خود به دو دسته خطاهاز Runtime و خطاهاز غیر Runtime تقسیم می شوند.

خطاهای Runtime، همانطور که از نام آن پیداست مربوط به زمان اجرای برنامه و شرایط اجرای برنامه می شود. این استثناهایی هستند که مربوط به شکل اشیا، یا نحوه دسترسی به آنهاست. مثلاً دسترسی به یک متاد از شیی که هنوز در حافظه ایجاد نشده است، دسترسی به عنصر ششم از یک آرایه پنج تایی، و موارد دیگری از این قبیل. یک برنامه باید به گونه ای نوشته شود که هیچگاه یک خطای Runtime رخ ندهد به عبارت بهتر برنامه نویس باید برنامه را به گونه ای بنویسد که این خطاها هیچ گاه رخ ندهند. در این فصل از کتاب، عمدۀ صحبت ما روی خطاهاز غیر Runtime است.

مکانیسمی برای کنترل استثناهای غیر Exception Handling است تا با استفاده از آن مشخص کنید که با رخداد خطا، عکس العمل برنامه در قبال آن چگونه باشد، مثلاً اجرای برنامه متوقف شود، برنامه به همان شکل معمول ادامه یابد، پیغام مناسبی به کاربر نشان داده شود، یا مهمتر از همه عمل خاص دیگری خارج از روال معمول برنامه انجام شود.

با استفاده از Exception Handling، خطایی که درون یک متاد رخ می دهد بپرون متاد کنترل می شود با استفاده از این مکانیسم با رخداد خطا در متاد، آبجکتی از کلاس Exception (یا یکی از کلاس‌های مشتق شده از آن) را تولید کرده و با استفاده از کلمه کلیدی throw به فراخواننده متاد تحویل می دهیم در تعریف متادی که در آن خط رخ می دهد با استفاده از کلمه کلیدی throws وقوع خطا را به فراخواننده آن متاد گوشزد می کنیم.

محلى که این متدهای فراخوانی می شود باید درون بلوک try نوشته شود در کنار بلوک try، بلوک دیگری به نام بلوک catch() قرار دارد که در صورت بروز خطا آن بلوک اجرا می شود. در صورتیکه در حین اجرای یک متدهای خطا رخ دهد بقیه متدهای خواهد شد و متدد مقدار برگشتی خواهد داشت همچنین بقیه بلوک try (که متدهای در آن بلوک فراخوانی شده است) صرفنظر می شود و بلوک catch() اجرا خواهد شد.

خطاهای در جاوا به دو دسته Runtime و غیر Runtime تقسیم می شوند خطاهای Runtime را می توان با بلوکهای try و catch کنترل کرد اما الزاماً در وجود بلوکهای try و catch ندارند روش درست در برخورد با این خطاهای این است که از بلوکهای try و catch استفاده نشود و به جای آن، این نوع خطاهای در برنامه پیش بینی کرد و مانع از وقوع آنها شد.

کلاس Exception دارای متدهای متعددی است که در موقع مختلف کاربرد دارند، متدهای toString() و getMessage() را به صورت یک رشته برمی گردانند، متدهای printStackTrace() و getLocalizedMessage() پیغامی که در هر خطای وجود دارد را برمی گردانند و متدهایی که باعث رخداد خطای نهایی شده اند را برمی گردانند.

تمرینات

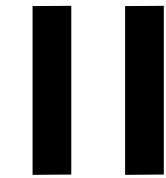
- (۱) متدى بنویسید که لیستی از اعداد را دریافت می کند و بزرگترین عدد داخل آرایه را برمی گرداند، در صورتیکه آرایه null باشد، یک Exception تولید کند، سپس این متدا در جای دیگری از برنامه فراخوانی کنید.
- (۲) یک کلاس Exception تعریف کنید حال متدى بنویسید که آرایه ای از آرایه های اعداد را دریافت می کند، سپس با فراخوانی متدى که در تمرین قبلی نوشته شد بزرگترین عنصره را آرایه را بدست می آورد اگر در این فراخوانی خطای رخ دهد، این متدا نیز خطایی از نوع کلاس Exception پیاده سازی شده تولید می کند. این متدا در جای دیگری از برنامه فراخوانی کنید.
- (۳) متدى بنویسید که دو String به صورت پارامتر دریافت کند، در صورتیکه رشتة اول درون رشتة دوم یافت شود، موقعیت آنرا برگرداند و در صورتیکه رشتة اول درون رشتة دوم وجود نداشته باشد یک Exception تولید کند این متدا در جای دیگری از برنامه فراخوانی کنید.
- (۴) کدهای زیر را به گونه ای اصلاح کنید که برنامه کامپایل شود:

```
public static void cat(File named) {
    RandomAccessFile input = null;
    String line = null;
    try {
        input = new RandomAccessFile(named, "r");
        while ((line = input.readLine()) != null) {
            System.out.println(line);
        }
        return;
    } finally {
        if (input != null) {
            input.close();
        }
    }
}
```

- (۵) یک کلاس `String` پیاده سازی کنید سپس یک متدا بنویسید که یک `String` دریافت می کند و بررسی می کند که آیا آن `String` یک آدرس پست الکترونیک معتبر (E-Mail) است یا خیر. در صورتیکه نامعتبر است خطایی از نوع `Exception` پیاده سازی شده تولید کند. این متدا را با `Email` های مختلف تست کنید.
- (۶) متدى بنویسید که یک عدد دریافت می کند و یک مقدار boolean برمی گرداند اگر عدد دریافت کرده بزرگتر از صفر باشد `true` و اگر کوچکتر از صفر باشد `false` برمی گرداند ولی اگر عدد صفر باشد یک `Exception` تولید کند. این متدا را با مقادیر مختلف فراخوانی کنید.
- (۷) متدى بنویسید که یک `String` به عنوان پارامتر دریافت کند، این متدا رشتة ای از کاراکترها را از طریق کنسول برنامه (وروڈی استاندارد) دریافت می کند و آنرا در انتهای `String` ای که به عنوان پارامتر دریافت نموده قرار می دهد، در صورتیکه کاربر کلید `Enter` را فشار دهد این

متدهای تولید کننده از این متدها در قسمت دیگری از برنامه برای خواندن و رودی از کاربر استفاده کنید.

۸) متدهای بنویسید که آرایه ای از int دریافت کنند و اعداد زوج آرایه را برگرداند در صورتیکه آرایه خالی یا null باشد ArrayEmptyException تولید کنند.



پکیج های جاوا

تا این قسمت از کتاب با گرامر زبان جاوا، داده های پایه، ساختارهای کنترلی و مفاهیم اولیه برنامه نویسی جاوا آشنا شدید. حال لازم است با کتابخانه های جاوا آشنا شوید واضح است که جاوا همانند هر زبان برنامه نویسی دیگر مجموعه ای از کتابخانه ها را در اختیار شما قرار می دهد تا بر اساس آن بتوانید برنامه خود را تولید کنید. کتابخانه های جاوا که در JDK وجود دارند در واقع مجموعه ای از کلاسها و اینترفیسها هستند که از آنها می توانید در برنامه های خود استفاده کنید به این کلاسها و اینترفیسها Java API (یا واسط برنامه نویسی جاوا) گفته می شود و بر اساس کارکردشان و اینکه مرتبط با چه مفهوم یا تکنولوژی هستند در پکیج های مختلفی سازماندهی شده اند. برای مثال، پکیج `java.net` شامل مجموعه ای از کلاسها و اینترفیسهاست که برای برنامه نویسی شبکه طراحی شده اند. جدول ۱۱-۱ تعدادی از این پکیجهای را نشان می دهد.

نام پکیج	کاربرد	مثالی از کلاسها و اینترفیسها
<code>java.lang</code>	اصلی ترین کلاس‌های جاوا	String, Integer, System
<code>java.util</code>	کلاس‌های پرکاربرد و با کاربرد	ArrayList, HashSet, Stack

	عمومی	
BigDecimal, BigInteger	کلاس‌هایی با کاربرد در محاسبات	java.math
Socket, URL, InetAddress	کلاس‌ها و اینترفیس‌های برنامه نویسی تحت شبکه	java.net
Reader, Writer, FileInputStream, CharBuffer	کلاس‌ها و اینترفیس‌هایی برای خواندن و نوشتن در فایل، یا دریافت و ارسال داده برای هر منبع داده	java.nio و java.io
Connection, Statement, Driver	کلاس‌ها و اینترفیس‌هایی برای کار با پایگاه داده	javax.sql و java.sql
JFrame, Button, JToolBar	کلاس‌هایی واسط کاربری	javax.swing و java.awt
UnicastRemoteObject, Remote, Naming	کلاس‌ها و اینترفیس‌هایی برای استفاده از تکنولوژی RMI (تکنولوژی RMI که در جلد دوم بررسی می‌شود برای برنامه نویسی توزیع شده طراحی شده است)	java.rmi
Applet, AudioClip	کلاس‌ها و اینترفیس‌هایی برای اپلت (اپلت‌ها برنامه‌های جاوا‌بی هستند که در وب و روی مرورگر وب اجرا می‌شوند)	java.applet

پکیج های جاوا

DateFormat, MessageFormat	شامل مجموعه ای از کلاسها برای نمایش متن و فرمت دهی به متن است. مثلاً نمایش تاریخ با فرمتهای مختلف، نمایش یک عدد با فرمتهای مختلف،...	java.text
Permission, Policy, AccessController	کلاسها و اینترفیسها برای برنامه نویسی امنیتی جاوا	java.security و javax.security
VetoableChangeListener, PropertyChangeListener, Introspector	کلاسها و اینترفیسها برای برنامه نویسی جاوابین. جاوابین یکی از تکنولوژیهای جاواست که امکان می دهد تا «تکه برنامه» های عمومی تولید کنیم. مثلاً با استفاده از جاوابین می توان یک قطعه نمایشی واسط کاربری را ایجاد نمود که در جاهای مختلف واسط کاربری استفاده شود.	java.beans

جدول ۱-۱۱. پکیج های اصلی جاوا

در این فصل قصد داریم تا برخی از پکیج های اصلی فوق را به شما معرفی کیم و نگاهی به کلاسها و اینترفیسها مهم آنها بیندازیم. در فصل های بعدی کتاب با جزئیات کامل تری از هر پکیج آشنا خواهید شد.

java.lang پکیج

این پکیج شامل اصلی ترین کلاسها جاوا می شود کلاسها یی که تقریباً در هر برنامه جاوا استفاده می شوند. کلاسها این پکیج می توان به دسته های زیر تقسیم نمود.

کلاس هایی که برای کار کردن با رشته های متنی استفاده می شوند از قبیل `StringBuffer`, `String`, `StringBuilder`

کلاسها یی که معرف اعداد و داده ها هستند از قبیل `Byte`, `Long`, `Double`, `Float`, `Integer`, `Boolean`

کلاس‌هایی که معرف خطا هستند (هر جای برنامه که خطای رخ دهد با آبجکتی از این کلاسها نشان داده می‌شود) از قبیل `Error` و `Exception`

کلاس‌هایی که برنامه نویسی همزمانی هستند از قبیل `Thread`, `ThreadGroup`, `ThreadLocal`

البته این پکیج شامل کلاسها و اینترفیس‌های دیگری نیز هست که برای جلوگیری از پیچیده شدن بحث از صحبت در مورد آنها خودداری می‌کنیم و بحث در مورد آنها را به جلد دوم کتاب موقول می‌کنیم.

پکیج `java.util`

این پکیج حاوی مجموعه‌ای از کلاس‌های عمومی و بسیار پرکاربرد است کلاس‌های این پکیج را می‌توان در دسته‌های زیر دسته بندی کرد:

کلاس‌هایی که تحت عنوان `Collection` ها شناخته می‌شوند و همانند ظرف هستند یعنی می‌توانند مجموعه‌ای از آبجکتها را در خود نگهداری کنند از قبیل `ArrayList`, `LinkedList`, `Queue`, `Stack`, `HashMap`, `HashSet`

کلاس‌های که معرف تاریخ و زمان هستند و برای کار با تاریخ و زمان استفاده می‌شوند از قبیل `Date` و `Calendar`

کلاس‌هایی که برای زمان ریز زمانی (مثلا در یک زمان مشخص کار خاصی انجام شود) استفاده می‌شوند شامل `TimerTask` و `Timer`

کلاس‌هایی که برای محلی سازی استفاده می‌شوند از قبیل `Locale`, `TimeZone` و `ResourceBundle`. پکیج `java.util` شامل سه زیر پکیج بسیار مهم نیز می‌شود که در جدول ۱۱-۲ توضیح داده شده‌اند.

نام پکیج	کاربرد	مثالی از کلاسها و اینترفیسها
<code>java.util.concurrent</code>	برای برنامه نویسی همزمانی استفاده می‌شود (مثلاً وقتی بخواهیم دو قسمت برنامه به صورت موازی و همزمان اجرا شوند)	<code>ConcurrentMap</code> , <code>Future</code> , <code>Executor</code>
<code>java.util.logging</code>	هر برنامه ممکن است از هزاران خط کد تشکیل شده باشد وقتی برنامه اجرا می‌شود برای شما سخت است بدانید کدام خط از برنامه در حال اجراست یا اجرای برنامه در چه	<code>LogManager</code> , <code>Logger</code> , <code>FileHandler</code>

پکیج های جاوا

	وضعیتی است. اما می توانید در هر خطی از برنامه که به نظرتان مناسب است یک جمله لاغ بنویسید. وقتی اجرای برنامه به نقطه لاغ می رسد متنی که در جمله لاغ مشخص کردید اید در فایل یا در کنسول برنامه نوشته می شود و از طریق آن متن می توانید متوجه شوید که آن نقطه از برنامه اجرا شده است.	
ZipFile ، ZipEntry ZipError	برای کار کردن با فایلهای ZIP استفاده می شود مثلا می توانید مجموعه ای از فایلهای در قالب یک فایل ZIP بسته بندی کنید یا یک فایل ZIP را باز کنید.	java.util.zip
JarFile ، JarEntry Attributes	برای کار کردن با فایلهای jar استفاده می شوند فایلهای jar که قالبی مشابه قالب فایلهای zip دارند برای بسته بندی کلاسها یک برنامه استفاده می شوند. وقتی یک برنامه در قالب یک فایل jar بسته بندی شد می توان آن فایل را اجرا نمود یا در یک محیط اجرایی قرار داد.	java.util.jar
Matcher ، Pattern MatchResult	برای کار با «عبارت‌های با قاعده» استفاده می شود. عبارتهای با قاعده در واقع متنها یی هستند که از یک قاعده مشخص پیروی می کنند. مثلا	java.util.regex

	دو رشته a13772 و b38383 را تصور کنید هر دو متن شش کارکتر دارند، با یک کاراکتر غیر عددی شروعی می شوند و با پنج کاراکتر عددی ادامه می یابند.
--	--

جدول ۲-۱۱. زیرپکیج های `java.util`

پکیج `java.nio` و `java.io`

این دو پکیج شامل مجموعه ای از کلاسها و اینترفیسها برای انجام عملیات ورود و خروج داده ها هستند. ممکن است در یک برنامه بخواهید داده های یک فایل را بارگذاری کنید (ورود داده به برنامه)، یا داده هایی را در یک فایل ذخیره کنید (خروج داده از برنامه). ورود و خروج داده ها به فایل محدود نمی شود و ممکن است سوکت شبکه، حافظه، یا هرجایی دیگری به عنوان مبدأ ورود داده ها یا مقصد خروج داده ها باشند. از مهمترین کلاسهایی که در این پکیج ها قرار دارند به کلاس‌های `File`, `InputStream`, `OutputStream` و `Serializable` و `Writer`, `Reader` می توان اشاره کرد.

۱۲

java.lang پکیج

در این فصل به معرفی و کاربرد کلاسها و اینترفیس‌هایی که در پکیج `java.lang` تعریف شده اند می‌پردازیم. کلاسها و اینترفیس‌هایی که در این پکیج قرار دارند اصلی ترین و مهم ترین کلاسها و اینترفیس‌های جاوا هستند زیرا معمولاً در هر جایی از برنامه استفاده می‌شوند و به همین دلیل کلاسها و اینترفیس‌های این پکیج نیاز به `import` ندارند و مجبور به اضافه کردن جمله

```
import java.lang.*;
```

برای استفاده کردن از آنها نیستید زیرا کامپایلر جاوا در زمان کامپایل برنامه، جمله فوق را به بالای تمام کلاس‌های برنامه اضافه می‌کند. برخی از مهم ترین کلاسها و اینترفیس‌های این پکیج در دو جدول زیر لیست شده اند.

<code>Boolean</code>	<code>Integer</code>	<code>System</code>
<code>Byte</code>	<code>Long</code>	<code>Thread</code>
<code>Character</code>	<code>Math</code>	<code>ThreadGroup</code>
<code>Class</code>	<code>Number</code>	<code>ThreadLocal</code>
<code>ClassLoader</code>	<code>Object</code>	<code>Throwable</code>
<code>Compiler</code>	<code>Package</code>	<code>Void</code>
<code>Double</code>	<code>Process</code>	<code>Character.SubSet</code>
<code>Enum</code>	<code>ProcessBuilder</code>	<code>Character.UnicodeBlock</code>
<code>Float</code>	<code>Short</code>	<code>Runtime</code>

String	StackTraceElement	RuntimePermission
StringBuffer	StrictMath	SecurityManager
StringBuilder		InheritableThreadLocal

جدول ۱۲-۱. کلاس‌های java.lang

Appendable	Comparable	Runnable
CharSequence	Iterable	
Cloneable	Readable	

جدول ۱۲-۲. اینترفیس‌های java.lang

با برخی از این کلاس‌ها در این فصل و با برخی دیگر در فصل‌های آینده آشنا می‌شوید.

کلاس‌های پوشش دهنده داده‌های نوع اولیه

در فصول قبل با داده‌های پایه (شامل int, char, float, boolean ...) آشنا شدید. اینها داده‌های اصلی جاوا هستند و هر نوع داده دیگری با ترکیب این داده‌ها و توسط یک کلاس تعریف می‌شود. به متغیری که از جنس یک داده پایه تعریف می‌شود «مقدار ساده» و به متغیری که از جنس یک کلاس تعریف می‌شوند «آبجکت» گفته می‌شود. در فراخوانی متدها، اگر پارامتر متدازن جنس یک داده پایه باشد همواره کپی آن مقدار ساده به متدازن ارسال می‌شود در حالیکه اگر پارامتر متدازن جنس کلاس باشد کپی ارجاع (reference) آن آبجکت به متدازن ارسال می‌شود. (در مورد این موضوع در فصل «برنامه نویسی شئ گرا» صحبت کرده ایم). همین تفاوت بین رفتار داده‌های پایه و آبجکتها، باعث شد تا طراحان جاوا از نسخه ۵ کلاس‌هایی را متناظر با داده‌های پایه به جاوا اضافه کنند که پوشش دهنده داده‌ای از همان جنس است. به عنوان نمونه، یک متغیر عددی را می‌توان از جنس داده پایه int یا از کلاس Integer تعریف کرد

```
int a = 12;
Integer b = new Integer(12);
```

متغیرهای a و b هر دو عدد ۱۲ را نشان می‌دهند اما از آنجاییکه Integer یک کلاس است b یک آبجکت است و رفتار آبجکتی از خود نشان می‌دهد اما a یک مقدار ساده است و رفتار مقداری دارد. کلاس Integer که امکان می‌دهد تا آبجکتهای عددی (به جای مقادیر عددی) تعریف کنیم، تعدادی متدهای کاربردی برای کار با اعداد int نیز در اختیار می‌گذارد. به عنوان نمونه، متداشتاتیک parseInt() یک رشته عددی را به یک عدد int تبدیل می‌کند.

```
String str = "93838";
int var = Integer.parseInt(str);
```

java.lang پکیج

جدول زیر داده های پایه و کلاس‌های پوشش دهنده آنها که همگی در پکیج java.lang قرار دارند را نشان می‌دهد.

int	Integer
long	Long
short	Short
boolean	Boolean
char	Character
byte	Byte
float	Float
double	Double

جدول ۱۲-۳. داده های پایه و کلاس‌های پوشش دهنده آنها

دقت کنید که تفاوت کلاس‌های پوشش دهنده همگی با حرف بزرگ الفبای انگلیسی و داده های پایه با حرف کوچک الفبای انگلیسی شروع می‌شوند بنابراین اگرچه float و Float کلمات یکسانی هستند اما از روی حرف اول آنها می‌توان کلاس یا داده پایه بودن آنها را تشخیص داد. کلاس‌های پوشش دهنده همگی دارای متدهای زیر هستند.

```
String toString()  
boolean equals(Object obj)  
int hashCode()
```

متدهای `toString()` داده ای که توسط آبجکت پوشش دهنده ارایه می‌شود را در قالب یک رشته بر می‌گرداند. (در مورد این متدهای فصلهای پیش آموخته اید، اینجا نیز همان مفهوم و کاربرد را دارد) متدهای `equals()` و `hashCode()` دو آبجکت پوشش دهنده را بررسی می‌کنند. `hashCode()` یک عدد صحیح int بر می‌گرداند. این عدد برای دو آبجکت که از یک کلاس ساخته شده‌اند و معرف یک داده هستند باید منحصر بفرد باشد. مثلاً اگر دو آبجکت Float وجود داشته باشد که هر دو معرف عدد ۱۲,۴۳ باشند انتظار داریم که عدد `hash code` یکسانی داشته باشند. که برای آبجکتها مخصوص بفرد از یک دسته منحصر است. این متدهای ساختمندان داده‌ها از قبیل Hashtable استفاده می‌شود.

کلاس‌های پوشش دهنده را می‌توان به دو بخش «عددی» و «غیر عددی» تقسیم کرد. کلاس‌های پوشش دهنده عددی شامل Integer، Long، Short، Byte، Double و غیر عددی شامل Character و Boolean هستند.

کلاس های پوشش دهنده عددی دارای متدهای مشترک زیر هستند که هریک می تواند برای تبدیل یک آبجکت عددی به یک نوع پایه فراخوانی شود.

```
byte byteValue()
double doubleValue()
float floatValue()
int intValue()
long longValue()
short shortValue()
```

پوشش دهنده های عددی همگی متدهایی استاتیک با ساختار

```
static x parseX(String str)
```

دارند که در آن `x` یک داده پایه است و این متدها می توانند یک رشته را به آن مقدار پایه تبدیل کنند. مثلاً کلاس `Integer` متد

```
static int parseInt(String str)
```

یک رشته عددی را به یک عدد `int` تبدیل می کند و در کلاس `Float` متد

```
static float parseFloat(String str)
```

یک رشته عدد اعشاری را به عدد `float` تبدیل می کند.
همچنین پوشش دهنده های عددی همگی متدهایی استاتیک با ساختار

```
static X valueOf(x value)
static X valueOf(String str)
```

دارند که معادل آبجکتی یک مقدار پایه یا یک رشته عددی را برمی گردانند. مثلاً کلاس `Integer` متد های

```
static Integer valueOf(int value)
static Integer valueOf(String str)
```

را دارد که از روی یک عدد `int` یا یک رشته عددی، یک آبجکت `Integer` ایجاد می کند و به شکل مشابه کلاس `Long` متد های

```
static Long valueOf(long value)
static Long valueOf(String str)
```

java.lang پکیج

را داراست که از روی یک عدد long یا یک رشته عددی، یک آبجکت Long برمی گرداند. در پوشش دهنده های عددی، مقادیر ثابتی نیز تعریف شده اند که فهرست آنها را در جدول زیر ملاحظه می کنید.

کمینه مقدار	MIN_VALUE
بیشینه مقدار	MAX_VALUE
تعداد بیت های کلاس	SIZE
نوع آبجکت	TYPE

جدول ۱۲-۴. مقادیر ثابت در کلاس های پوشش دهنده عددی

کلاس های float و Double

این کلاس ها به ترتیب داده های نوع اولیه float و double را نگهداری می کنند سازنده های کلاس float را در زیر می بینید:

```
Float(double num)
Float(float num)
Float(String str) throws NumberFormatException
```

همانگونه که ملاحظه می کنید، از روی داده های پایه float و double می توان آبجکتهايی از کلاس ساخت. float

سازنده های کلاس Double نیز در زیر نشان داده شده اند:

```
Double(double num)
Double(String str) throws NumberFormatException
```

از روی یک داده double و از روی یک رشته که معرف یک داده باشد(مثلا "12.1212") می توان یک آبجکت از کلاس Double ساخت. مثال زیر استفاده از این دو کلاس را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter12;
2
3 class DoubleDemo {
4
5     public static void main(String args[]) {
6         Double d1 = new Double(3.14159);
7         Double d2 = new Double("314159E-5");
8     }
}
```

```

9         System.out.println(d1 + " = " +
10        d2 + " -> " +
11        d1.equals(d2));
12    }
13 }
```

کد ۱۲-۱

همانطور که نتیجه اجرای این برنامه نشان می دهد هر دو سازنده نتیجه یکسانی از آبجکت Double ایجاد می کنند.

کلاس‌های `Long` و `Integer`، `Short`، `Byte`

از این کلاسها برای نگهداری داده های int، short، byte و long استفاده می شود سازنده های این کلاسها در زیر نشان داده شده اند:

```

Byte(byte num)
Byte(String str) throws NumberFormatException

Short(short num)
Short(String str) throws NumberFormatException

Integer(int num)
Integer(String str) throws NumberFormatException

Long(long num)
Long(String str) throws NumberFormatException
```

همانگونه که ملاحظه می کنید از روی داده های عددی یا رشته های معادل آنها می توان آبجکتها بی از کلاس‌های فوق ایجاد نمود.

کلاس Character

از این کلاس برای نگهداری داده نوع char استفاده می شود سازنده این کلاس در زیر نشان داده شده است:
Character(char ch)

ch مشخص کننده کاراکتری است که آبجکتی که از کلاس Character ساخته می شود آنرا نگهداری می کند.
با فراخوانی متده `charValue()` می توانید داده ای را که در این کلاس قرار دارد بدست آورید:

```
char charValue()
```

java.lang پکیج

در کلاس Character چندین ثابت تعریف شده اند که در جدول زیر آنها را ملاحظه می کنید. با استفاده از چندین متدهای static که در کلاس Character تعریف شده اند می توانید نوع یک کاراکتر و کوچکی یا بزرگی یک کاراکتر را مشخص کنید برنامه زیر فراخوانی این متدها را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter12;
2
3 public class IsDemo {
4     public static void main(String args[]) {
5         char a[] = {'a', 'b', '5', '?', 'A', ' '};
6
7         for (int i = 0; i < a.length; i++) {
8             if (Character.isDigit(a[i]))
9                 System.out.println(a[i] +
10                     " is a digit.");
11             if (Character.isLetter(a[i]))
12                 System.out.println(a[i] +
13                     " is a letter.");
14             if (Character.isWhitespace(a[i]))
15                 System.out.println(a[i] +
16                     " is whitespace.");
17             if (Character.isUpperCase(a[i]))
18                 System.out.println(a[i] +
19                     " is uppercase.");
20             if (Character.isLowerCase(a[i]))
21                 System.out.println(a[i] +
22                     " is lowercase.");
23         }
24     }
25 }
```

لیست ۱۲-۴

اجرای برنامه فوق نتیجه زیر را در خروجی استاندارد به دنبال خواهد داشت.

```
a is a letter.
a is lowercase.
b is a letter.
b is lowercase.
5 is a digit.
A is a letter.
A is uppercase.
is whitespace.
```

جدول زیر متدهای کلاس Character را نشان می دهد.

شرح	متد
چنانچه کاراکتر ch در یونی کد تعریف شده باشد، فراخوانی این متد مقدار true بر می گرداند و در غیراینصورت مقدار false بر می گرداند.	static boolean isDefined(char ch)
چنانچه کاراکتر ch یک رقم عددی باشد این متد true بر می گرداند و در غیر اینصورت مقدار false بر می گرداند.	static boolean isDigit(char ch)
چنانچه از کاراکتر ch در نام یک متغیر در برنام استفاده شود و وجود آن توسط کامپایلر صرفنظر شود (ممولاً اینگونه کاراکترها قابل مشاهده نیستند) فراخوانی این متد مقدار true بر می گرداند و در غیراینصورت مقدار false بر می گرداند.	static boolean isIdentifierIgnorable(char ch)
در صورتیکه از کاراکتر ch بتوان برای نامگذاری یک متغیر در جاوا استفاده کرد فراخوانی این متد مقدار true بر می گرداند و در غیر اینصورت مقدار false بر می گرداند.	static boolean isJavaIdentifierPart(char ch)
در صورتیکه بتوان از کاراکتر ch در ابتدای نام یک متغیر جاوا استفاده کرد فراخوانی این متد مقدار true بر می گرداند و در غیراینصورت مقدار false بر می گرداند.	static boolean isJavaIdentifierStart(char ch)
چنانچه ch یکی از حروف الفبا باشد فراخوانی این متد مقدار true بر می گرداند و در غیراینصورت مقدار false بر می گرداند.	static boolean isLetter(char ch)
در صورتیکه کاراکتر ch یکی از حروف الفبا یا یک رقم عددی باشد فراخوانی این متد	static boolean isLetterOrDigit(char ch)

java.lang پکیج

مقدار true برمی گرداند.	
در صورتیکه ch حرف کوچک الفبای انگلیسی باشد فراخوانی این متدها مقدار true برمی گرداند.	static boolean isLowerCase(char ch)
در صورتیکه ch کارکتر فاصله یونی کد باشد مقدار برگشتی این متدها true خواهد بود.	static boolean isSpaceChar(char ch)
در صورتیکه کارکتر ch حرف بزرگ الفبای انگلیسی باشد فراخوانی این متدها مقدار true برمی گرداند.	static boolean isUpperCase(char ch)
چنانچه ch کاراکتر "فاصله خالی" باشد این متدها مقدار true برمی گرداند.	static boolean isWhitespace(char ch)
این متدها ch را به حرف کوچک الفبای انگلیسی تبدیل کرده و برمی گردانند.	static char toLowerCase(char ch)
این متدها ch را به حرف بزرگ الفبای انگلیسی تبدیل کرده و برمی گردانند.	static char toUpperCase(char ch)

جدول ۱۲-۵. متدهای کلاس Character

Boolean کلاس

از این کلاس برای نگهداری داده های boolean استفاده می شود سازنده های این کلاس به صورت زیر می باشند:

```
Boolean(boolean boolValue)  
Boolean(String boolString)
```

در سازنده اول مقدار boolean یک مقدار است که از روی آن یک آبجکت ساخته می شود در سازنده دوم رشته است (مثلاً رشته "True") که مشخص کننده یک مقدار boolean است و از روی آن یک آبجکت معادل ساخته می شود توجه داشته باشید که هر کدام از کاراکترهای این رشته می توانند با حروف بزرگ یا کوچک الفبای انگلیسی نوشته شوند. در این کلاس دو مقدار ثابت TRUE و FALSE تعریف شده اند که هر کدام معرف مقادیر "درست" و "نادرست" هستند جدول زیر متدهای کلاس Boolean را نشان می دهد.

شرح	متدها
مقدار معادل boolean آبجکت را برمی گرداند.	boolean booleanValue()

<p>در صورتیکه مقدار boolean آبجکتی که متده compareTo() آن فراخوانی شده با مقدار آبجکت b یکسان باشد مقدار boolean برگشتی این متده صفر خواهد بود و در صورتیکه مقدار boolean آبجکت true و مقدار مقدار باشد مقدار برگشتی این متده مثبت و false در غیراینصورت منفی خواهد بود.</p>	<pre>int compareTo(Boolean b)</pre>
<p>در صورتیکه مقدار boolean آبجکتی که متده equals() آن فراخوانی شده با مقدار آبجکت boolean boolObj یکسان باشد مقدار برگشتی این متده true خواهد بود و در غیراینصورت false خواهد بود.</p>	<pre>boolean equals(Object boolObj)</pre>
<p>چنانچه متغیر سیستمی propertyName دارای مقدار true باشد فراخوانی این متده مقدار true برミگرداند و در غیراینصورت false برمی گردد.</p>	<pre>static boolean getBoolean(String propertyName)</pre>
<p>آبجکت را برمی گردد.</p>	<pre>int hashCode()</pre>
<p>اگر مقدار رشته ای str برابر "true" باشد حاصل فراخوانی این متده true و در غیراینصورت false خواهد بود (در این متده بزرگی و کوچکی حروف اهمیتی ندارد).</p>	<pre>static boolean parseBoolean(String str)</pre>
<p>رشته معادل آبجکت را برمی گردد.</p>	<pre>String toString()</pre>
<p>رشته معادل آبجکت boolValue را برمی گردد.</p>	<pre>static String toString(boolean boolValue)</pre>
<p>معادل Boolean مقدار boolValue را برمی گردد.</p>	<pre>static Boolean valueOf(boolean boolVal)</pre>
<p>در صورتیکه مقدار رشته boolString باشد (بدون توجه به کوچک بودن یا بزرگ بودن حروف) مقدار برگشتی این متده true آبجکتی از کلاس Boolean با مقدار برگردانده می شود و در غیراینصورت مقدار</p>	<pre>static Boolean valueOf(String boolString)</pre>

آبجکت false خواهد بود.

جدول ۶-۱۲. متدهای کلاس Boolean

کلاس System

کلاس System حاوی چندین فیلد و متدهای استاتیک است. «ورودی استاندارد»، «خروجی استاندارد» و همچنین «خروجی خطای مهترین فیلدهای استاتیک این کلاس هستند.

```
System.in
System.out
System.err
```

با ورودی و خروجی استاندارد قبل آشنا شده اید. خروجی خطای System.out برای نمایش‌های پیامهای خطای پیامهایی که می‌باشد مورد توجه کاربر قرار گیرند استفاده می‌شود. به صورت پیش‌فرض پیامهایی که در System.out قرار می‌گیرند مشابه System.out در کنسول برنامه نمایش می‌یابند اما این امکان وجود دارد تا آنها را از پیغام‌های System.out تفکیک کرد تا در جای دیگری نمایش یابند یا ذخیره شوند. برخی از متدهای کلاس System در ادامه تشریح شده‌اند.

متدهای System

این متدهای زمان فعلی را بر حسب میلی ثانیه بر می‌گردانند. عددی که بر می‌گرداند در واقع تعداد میلی ثانیه‌ای است که از نیمه شب اول ژانویه سال ۱۹۷۰ تا لحظه فعلی گذشته است.

از این متدهای زمانی دو اتفاق در برنامه، یا اندازه گیری مدت اجرای یک متدها، یا زمان اتمام یک پردازش استفاده می‌شود.

برای اندازه گیری فاصله زمانی کافیست در شروع و پایان یک حادثه این متدها را فراخوانی کنید و سپس با تفریق دو زمان، طول مدت اجرا را محاسبه کنید. مثال زیر بکارگیری این متدها نشان می‌دهد.

```

1 package ir.atlassoft.javase.chapter12;
2
3 public class Elapsed {
4     public static void main(String args[]) {
5         long start, end;
6
7         System.out.println("Timing a for loop from"+
8             " 0 to 10,000,000");
9
10        // time a for loop from 0 to 10,000,000

```

```

11         // get starting time
12         start = System.currentTimeMillis();
13         for (int i = 0; i < 10000000; i++) ;
14             // get ending time
15             end = System.currentTimeMillis();
16             System.out.println("Elapsed time: " +
17                 (end - start));
18     }
19 }
```

لیست ۱۲-۵

خروجی این برنامه به صورت زیر خواهد بود (توجه داشته باشید که بسته به مشخصات سخت افزاری و نرم افزاری سیستم شما از قبیل CPU و سیستم عامل، ممکن است نتیجه متفاوتی را مشاهده کنید)

```
Timing a for loop from 0 to 1,000,000
Elapsed time: 40
```

درصورتیکه دقیق میلی ثانیه برای شما کافی نباشد و به دقت نانوثانیه نیاز داشته باشید، میتوانید با فراخوانی `متد () زمان نانوثانیه را بدست آورید.` مثال زیر بکارگیری این متدها `nanoTime()` نشان می دهد:

```

1 package ir.atlassoft.javase.chapter12;
2
3 public class Elapsed2 {
4     public static void main(String args[]) {
5         long start, end;
6         System.out.println("Timing a for loop from"+
7             " 0 to 10,000,000");
8         // time a for loop from 0 to 10,000,000
9         start = System.nanoTime(); // get starting time
10        for (int i = 0; i < 10000000; i++) ;
11        end = System.nanoTime(); // get ending time
12
13        System.out.println("Elapsed time: " +
14            (end - start));
15    }
16 }
```

لیست ۱۲-۶

متدهای arrayCopy()

با استفاده از این متدهای می‌توانید یک آرایه را کپی کنید کارایی این متدهای بسیار بیشتر از روش‌های شخصی است که در آنها از حلقه (مثلاً حلقه for) برای کپی کردن یک آرایه استفاده می‌شود. در زیر از این متدهای برای کپی کردن عناصر آرایه a در آرایه b استفاده شده است. از این متدهای همچنین می‌توانید برای کپی کردن بخشی از یک آرایه در یک آرایه دیگر نیز استفاده کنید.

```
1 package ir.atlassoft.javase.chapter12;
2
3 public class ACDEmo {
4     static byte a[] = { 65, 66, 67, 68, 69,
5                         70, 71, 72, 73, 74 };
6     static byte b[] = { 77, 77, 77, 77, 77,
7                         77, 77, 77, 77, 77 };
8
9     public static void main(String args[]) {
10         System.out.println("a = " + new String(a));
11         System.out.println("b = " + new String(b));
12
13         System.arraycopy(a, 0, b, 0, a.length);
14         System.out.println("a = " + new String(a));
15         System.out.println("b = " + new String(b));
16
17         System.arraycopy(a, 0, a, 1, a.length - 1);
18         System.arraycopy(b, 1, b, 0, b.length - 1);
19         System.out.println("a = " + new String(a));
20         System.out.println("b = " + new String(b));
21     }
22 }
```

لیست ۱۲-۶

خروجی اجرای برنامه فوق بصورت زیر خواهد بود.

```
a = ABCDEFGHIJ
b = MMMMMMMMM
a = ABCDEFGHIJ
b = ABCDEFGHIJ
a = AABCDEFGH
b = BCDEFGHIJJ
```

استفاده از متدهای `setProperty()` و `getProperty()`

برنامه های جاوا ممکن است در محیط های مختلف (سیستم عامل های مختلف) و شرایط مختلفی اجرا شوند. کلاس `System` امکان می دهد تا به برخی اطلاعات محیطی که برنامه در آن اجرا می شود دسترسی پیدا کنیم. از جمله این اطلاعات، سیستم عامل، نسخه سیستم عامل، دایرکتوری کاربر، نسخه جاوا، و متغیر های محیطی است. متدهای `setProperty()` و `getProperty()` کلاس `Properties` در فصل ۱۴، «ساختمان» حاوی تمام اطلاعات محیطی است. (با کلاس `Properties` در فصل ۱۴، «ساختمان» داده ها و پکیج `java.util` آشنا می شوید). علاوه بر متدهای `setProperty()` و `getProperty()` دیگر جدول زیر، لیست متغیر های محیطی که از طریق متدهای `getProperties()` قابل دسترسی هستند را نشان می دهد.

<code>java.specification.vendor</code>	<code>java.vm.version</code>
<code>java.vendor</code>	<code>line.separator</code>
<code>java.vendor.url</code>	<code>os.arch</code>
<code>java.version</code>	<code>os.name</code>
<code>java.vm.name</code>	<code>os.version</code>
<code>java.vm.specification.name</code>	<code>path.separator</code>
<code>java.vm.specification.vendor</code>	<code>user.dir</code>
<code>java.vm.specification.version</code>	<code>user.home</code>
<code>java.vm.vendor</code>	<code>user.name</code>
<code>file.separator</code>	<code>java.io.tmpdir</code>
<code>java.class.path</code>	<code>java.library.path</code>
<code>java.class.version</code>	<code>java.home</code>
<code>java.specification.name</code>	<code>java.ext.dirs</code>
<code>java.specification.vendor</code>	<code>java.compiler</code>

جدول ۱۲-۷. متغیر های محیطی

مثال زیر نام و مقدار تمام اطلاعات محیطی سیستم را در خروجی استاندارد چاپ می کند.

```

1 package ir.atlassoft.javase.chapter12;
2
3 import java.util.Properties;
4 import java.util.Set;
5
6 public class UsingSystemProperties {
7
8     public static void main(String[] args) {
9         Properties props = System.getProperties();

```

java.lang پکیج

```
10         Set set = props.keySet();
11
12         String[] keys = (String[]) set.toArray(new
13 String[0]);
14         for (int i = 0; i < keys.length; i++) {
15             String key = keys[i];
16             String value = props.getProperty(key);
17             System.out.println(key+" : "+value);
18         }
19     }
20 }
```

۱۲-۷ لیست

خروجی اجرای این برنامه در زیر نشان داده شده است.

```
java.runtime.name : Java(TM) 2 Runtime Environment, Standard Edition
java.vm.version : 1.4.2_01-b06
java.vm.vendor : Sun Microsystems Inc.
java.vendor.url : http://java.sun.com/
.
.
.

java.runtime.version : 1.4.2_01-b06
java.vm.specification.vendor : Sun Microsystems Inc.
os.name : Windows XP
java.specification.name : Java Platform API Specification
user.name : Ahmad
java.vm.specification.version : 1.0
java.vendor : Sun Microsystems Inc.
```

مثال دیگری در زیر نشان داده شده که استفاده از متدها برای نمایش دایرکتوری کاربر (دایرکتوری که در آن، همین برنامه در حال اجراست را برمی‌گرداند) را نشان می‌دهد.

```
1 package ir.atlassoft.javase.chapter12;
2
3 public class ShowUserDir {
4     public static void main(String args[]) {
5         System.out.println(System.getProperty("user.dir"));
6     }
7 }
```

لیست ۱۲-۸

اجرای برنامه فوق نتیجه زیر را به همراه خواهد داشت.

```
F:\all-project\SystemDemo
```

در نقطه مقابل متد `getProperty()` که مقدار یک متغیر محیطی را برمی گرداند، متد `setProperty()` می تواند مقدار یک متغیر محیطی را تغییر دهد یا حتی متغیر محیطی جدیدی به متغیرهای موجود اضافه کند. مثلا ممکن است در شروع یک برنامه مجموعه ای از متغیرهای محیطی را تنظیم کنید تا در جاهای مختلفی از برنامه بتوانید با استفاده از متد `getProperty()` آنها را بخوانید و استفاده کنید.

کلاس Object

همانطور که قبلا گفته شد، کلاس `Object` پدر تمام کلاسهای جاواست. متدهایی که در این کلاس تعریف یا پیاده سازی شده اند در تمام کلاسهای جاوا به ارث برده می شوند. برخی از این متدها در بخش‌های قبلی بررسی شد در اینجا به استفاده از متد `clone()` و اینترفیس `Cloneable` خواهیم پرداخت.

اینترفیس Clone() و استفاده از متد Clone()

با استفاده از متد `clone()` می توانید یک کپی از آبجکت خود ایجاد کنید (به این عمل همتاسازی آبجکت گفته می شود). این بدین معناست که وقتی آبجکتی اصطلاحا `clone` می شود، تمام وضعیت آن آبجکت در جای دیگری از برنامه کپی می شود، این قابلیت به هیچ روش دیگری به جز `clone()` امکان‌پذیر نیست. متد `clone()` اگر چه در تمام کلاسهای جاوا به ارث برده می شود، اما فقط در کلاسهایی قابل فراخوانی است که اینترفیس `Cloneable` را پیاده سازی کرده باشند.

در اینترفیس `Cloneable` هیچ متدی تعریف نشده است که بخواهید آنرا پیاده سازی کنید، بلکه صرفا به منزله عالمتی است که به ماشین مجازی جاوا همتاسازی یک آبجکت را یادآوری می کند تا تمہیدات لازم برای همتاسازی آبجکتها یک کلاس را فراهم کند. در واقع از اینترفیس `Cloneable` برای نشان دادن اینکه یک کلاس کپی کردن آبجکتها خودش را مجاز می داند استفاده می شود.

اگر متد `clone()` را از کلاسی که اینترفیس `Cloneable` را پیاده سازی نکرده باشد فراخوانی کنید با خطای زمان اجرای `CloneNotSupportedException` مواجه خواهید شد.

توصیه می شود حتی الامکان از این متد استفاده نکنید زیرا همتاسازی یک آبجکت می تواند مشکل زا باشد. مثلا اگر آبجکتی به نام `obj` داشته باشید و یکی از فیلدهای آن، خود آبجکت دیگری به نام `ref` باشد، در کپی آبجکت `obj` فیلد `ref` کپی نمی شود و به همان آبجکت قبلی اشاره می کند.

java.lang پکیج

متدهای `clone()` در کلاس `Object` قرار دارد به صورت `protected` تعریف شده است بنابراین نمی‌توانید این متدها را از هر کلاس فراخوانی کنید این متدها فقط از داخل همان کلاس فراخوانی کرد یا اگر مایلید که توسعه کلاسهای دیگر نیز قابل فراخوانی باشد باید به صورت `public` آنرا `override` کنید. برنامه زیر استفاده از متدهای `clone()` و اینترفیس `Cloneable` را نشان می‌دهد.

```
1 package ir.atlassoft.javase.chapter12;
2
3 public class TestClone implements Cloneable {
4     int a;
5     double b;
6
7     // This method calls Object's clone().
8     TestClone cloneTest() {
9         try {
10             // call clone in Object.
11             return (TestClone) super.clone();
12         } catch (CloneNotSupportedException e) {
13             System.out.println("Cloning not allowed.");
14             return this;
15         }
16     }
17 }
18
19 class CloneDemo {
20     public static void main(String args[]) {
21         TestClone x1 = new TestClone();
22         TestClone x2;
23
24         x1.a = 10;
25         x1.b = 20.98;
26
27         x2 = x1.cloneTest(); // clone x1
28
29         System.out.println("x1: " + x1.a + " " + x1.b);
30         System.out.println("x2: " + x2.a + " " + x2.b);
31     }
32 }
```

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود.

```
x1: 10 20.98
x2: 10 20.98
```

همانطور که ملاحظه می کنید مقدار برگشتی متده است که باید آنرا به نوع مورد نظر خود Cast کنید در زیر مثال دیگری ارایه شده است که در آن با `clone()` کردن متده `override` فراخوانی آن در خارج از کلاس امکانپذیر شده است.

```

1 package ir.atlassoft.javase.chapter12;
2
3 class TestClone1 implements Cloneable {
4     int a;
5     double b;
6
7     // clone() is now overridden and is public.
8     public Object clone() {
9         try {
10             // call clone in Object.
11             return super.clone();
12         } catch (CloneNotSupportedException e) {
13             System.out.println("Cloning not allowed.");
14             return this;
15         }
16     }
17 }
18
19 class CloneDemo2 {
20     public static void main(String args[]) {
21         TestClone1 x1 = new TestClone1();
22         TestClone1 x2;
23
24         x1.a = 10;
25         x1.b = 20.98;
26
27         // here, clone() is called directly.
28         x2 = (TestClone1) x1.clone();
29 }
```

java.lang پکیج

```
30     System.out.println("x1: " + x1.a + " " + x1.b);
31     System.out.println("x2: " + x2.a + " " + x2.b);
32 }
33 }
```

لیست ۱۲-۱۰

حاصل اجرای برنامه فوق بصورت زیر خواهد بود:

```
x1: 10 20.98
x2: 10 20.98
```

شاید در نگاه اول نتوان از تاثیرات جانبی استفاده از همتاسازی مطلع شد اما در عمل استفاده از آن خطرات زیادی در عملکرد درست برنامه به همراه خواهد داشت.

کلاس Class

در پکیج `java.lang` کلاسی به نام `Class` وجود دارد. به این کلاس که خود توصیف کننده کلاسهای جاواست (مثلًا اینکه همه کلاسهای جاوا یک نام دارند، یک پکیج دارند، تعدادی سازنده، فیلد و متدهای...) به صورت مفصل در مبحث `Reflection` خواهیم پرداخت. اما از آنجاییکه این کلاس در پکیج `java.lang` قرار دارد بد ندیدم که در اینجا نیز نگاهی به این کلاس بیندازیم.
نکته ای که در این کلاس وجود دارد این است که با استفاده از کلمه کلیدی `new` نمی توان از روی این کلاس آبجکت ساخت، به جای آن، ماشین مجازی جاوا به ازای تمام کلاسهایی که در برنامه بارگذاری می شوند یک آبجکت `Class` ایجاد می کند که توصیف کننده آن کلاس است و می توان مستقیماً از طریق نام آن کلاس یا یکی از آبجکتهايی که از روی آن کلاس ایجاد شده است به آن آبجکت `Class` دست یافت.

```
Class clazz1 = String.class;
Class clazz2 = "Test String".getClass();
```

آبجکتهاي `clazz1` و `clazz2` هر دو یک آبجکت هستند و به یک جای حافظه اشاره می کنند.
حال که به آبجکتی از کلاس `Class` دست یافته ایم می توانیم جزئیات آن کلاس را اکلای کنیم. مثل اینکه نام آن کلاس چیست و از چه کلاسی مشتق شده است. مثال زیر انجام این کار را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter12;
2
3 public class Parent {
4     int a;
5     float b;
```

```

6   }
7
8 class Child extends Parent {
9     double c;
10 }
11
12 class UsingClass {
13     public static void main(String args[]) {
14         Parent object1 = new Parent();
15         Child object2 = new Child();
16         Class object1Class;
17
18         // get Class reference
19         object1Class = object1.getClass();
20         System.out.println("object1 is object of type: "
21                         + object1Class.getName());
22
23         // get Class reference
24         Class object2Class = object2.getClass();
25         System.out.println("object2 is object of type: "
26                         + object2Class.getName());
27         Class aClass = object2Class.getSuperclass();
28         System.out.println("object2's superclass is " +
29                         aClass.getName());
30     }
31 }
```

لیست ۱۲-۱۱

خروجی اجرای برنامه فوق بصورت زیر است.

```
object1 is object of type: Parent
object2 is object of type: Child
object2's superclass is Parent
```

Math کلاس

کلاس Math حاوی متدهایی برای انجام عملیات محاسباتی، مثلثاتی، هندسی و کار با اعداد اعشاری است. در این کلاس همچنین دو مقدار ثابت π و E تعریف شده اند که به ترتیب معرف اعداد «پی» و «پی» بوده و برابر ۳.۱۴ و ۲.۷۲ می باشند.

متدهای توابع مثلثاتی

متدهای زیر، مقدار مثلثاتی یک زاویه بر حسب رادیان را محاسبه می کنند و برمی گردانند.

متدها	شرح
static double sin(double arg)	سینوس زاویه arg را که بر حسب رادیان مشخص شده برمی گرداند.
static double cos(double arg)	کسینوس زاویه arg را که بر حسب رادیان مشخص شده برمی گرداند.
static double tan(double arg)	تانژانت زاویه arg را که بر حسب رادیان مشخص شده برمی گرداند.

جدول ۱۲-۸. متدهای توابع مثلثاتی

متدهای زیر وارون مقدار مثلثاتی یک زاویه بر حسب رادیان را برمی گردانند.

متدها	شرح
static double asin(double arg)	زاویه ای را بر حسب رادیان برمی گرداند که سینوس آن برابر arg است.
static double acos(double arg)	زاویه ای را بر حسب رادیان برمی گرداند که کسینوس آن برابر arg است.
static double atan(double arg)	زاویه ای را بر حسب رادیان برمی گرداند که تانژانت آن برابر arg است.
static double atan2(double x, double y)	زاویه را بر حسب رادیان برمی گرداند که تانژانت آن برابر y/x است.

جدول ۱۲-۹. متدهای وارون توابع مثلثاتی

متدهای زیر سینوس، کسینوس و تانژانت هیپربولیک یک زاویه را محاسبه می کنند و برمی گردانند.

شرح	متدها
سینوس هیپربولیک زاویه arg را که بر حسب رادیان مشخص شده برمی گرداند.	static double sinh(double arg)
کسینوس هیپربولیک زاویه arg را که بر حسب رادیان مشخص شده برمی گرداند.	static double cosh(double arg)
تانژانت هیپربولیک زاویه arg را که بر حسب رادیان مشخص شده برمی گرداند.	static double tanh(double arg)

جدول ۱۲-۱۰. متدهای توابع هیپربولیک مثلثاتی

متدهای توابع نمایی

متدهای زیر برای محاسبات اعداد تواندار استفاده می شوند

شرح	متدها
ریشه سوم عدد arg را برمی گرداند.	static double cbrt(double arg)
حاصل e^{\arg} را برمی گرداند.	static double exp(double arg)
لگاریتم طبیعی arg را برمی گرداند.	static double log(double arg)
لگاریتم arg را در مبنای ۱۰ برمی گرداند.(در J2SE 5 اضافه شده است)	static double log10(double arg)
لگاریتم طبیعی arg+1 را برمی گرداند(در J2SE 5 اضافه شده است)	static double log1p(double arg)
حاصل y^x را برمی گرداند	static double pow(double y, double x)
جذر arg را برمی گرداند.	static double sqrt(double arg)

جدول ۱۲-۱۱. متدهای توابع نمایی

توابع گرد کردن

در کلاس Math همچنین متدهای مختلفی برای انجام عملیات گرد کردن گنجانده شده اند در جدول زیر لیست این متدها را مشاهده می کنید در بین این متدها متدهای به نام ulp() قرار دارد ulp در اینجا مخفف جمله "units in the last place" است) با استفاده از این متدهای توانید، فاصله عددی بین یک مقدار و مقدار عددی بزرگتر از آنرا بدست آورید از این متدهای توانید برای ارزیابی دقت نتایج استفاده کنید.

java.lang پکیج

شرح	متدها
قدر مطلق arg را برمی گرداند.	static int abs(int arg)
قدر مطلق arg را برمی گرداند.	static long abs(long arg)
قدر مطلق arg را برمی گرداند.	static float abs(float arg)
قدر مطلق arg را برمی گرداند.	static double abs(double arg)
کوچکترین عدد صحیح بزرگتر یا مساوی arg را برمی گرداند.	static double ceil(double arg)
بزرگترین عدد صحیح کوچکتر یا مساوی arg را برمی گرداند.	static double floor(double arg)
بین دو عدد x و y ، عدد بزرگتر را برمی گرداند.	static int max(int x, int y)
بین دو عدد x و y ، عدد بزرگتر را برمی گرداند.	static long max(long x, long y)
بین دو عدد x و y ، عدد بزرگتر را برمی گرداند.	static float max(float x, float y)
بین دو عدد x و y ، عدد بزرگتر را برمی گرداند.	static double max(double x, double y)
بین دو عدد x و y ، عدد کوچکتر را برمی گرداند.	static int min(int x, int y)
بین دو عدد x و y ، عدد کوچکتر را برمی گرداند.	static long min(long x, long y)
بین دو عدد x و y ، عدد کوچکتر را برمی گرداند.	static float min(float x, float y)
بین دو عدد x و y ، عدد کوچکتر را برمی گرداند.	static double min(double x, double y)
نزدیکترین عدد صحیح به مقدار arg را برمی گرداند.	static double rint(double arg)
arg را به نزدیکترین عدد صحیح int گرد کرده و برمی گرداند.	static int round(float arg)
arg را به نزدیکترین عدد صحیح long گرد کرده و برمی گرداند.	static long round(double arg)
arg مربوط به ulp را برمی گرداند.	static float ulp(float arg)
arg مربوط به ulp را برمی گرداند.	static double ulp(double arg)

جدول ۱۲-۱۳. متدهای توابع گرد کردن

خلاصه

در این فصل با کلاس ها و اینترفیس‌هایی که در پکیج `java.lang` قرار دارند آشنا شدید این پکیج به صورت پیش فرض در تمام کلاس‌های جاوا `import` می‌شود و نیازی به اضافه کردن جمله `import java.lang.*;` در کلاسها نیست.

کلاس `String` یکی از کلاس‌های مهم این پکیج است که در فصل دیگری از این کتاب تشریح شده است. از جمله کلاس‌هایی که در این پکیج قرار دارند کلاس‌های «پوشش دهنده داده‌های پایه» هستند. از آنجاییکه داده های پایه، آبجکت محسوب نمی‌شوند، کلاس‌های پوشش دهنده می‌توانند داده‌های پایه را در قالب آبجکت `String` ارایه کنند. این کلاسها شامل کلاس‌های `Double`, `Long`, `Short`, `Float`, `Integer`, `Byte` و `Boolean`, `Character`, `long`, `double`, `byte`, `boolean`, `char`, `int`, `short`, `float` یکی دیگر از کلاس‌های این پکیج، کلاس `System` است که بسیاری از متدهای کاربردی در آن تعریف شده است از جمله این متدها می‌توان به، `currentTimeMillis()` که زمان فعلی سیستم را برمی‌گرداند، `setProperty()` و `getProperty()` که کار کپی یک آرایه را انجام می‌دهد، `arrayCopy()` که مقدار یک متغیر را از متغیرهای سیستم برمی‌گرداند یا مقداردهی می‌کند، اشاره کرد.

کلاس `Object` که تمام کلاس‌های جاوا به شمار می‌رود در این پکیج قرار دارد در این کلاس برخی متدهای پایه از قبیل `equals()`, `toString()` و `clone()` تعریف شده اند. کلاس `Math` و کلاس `Class` از دیگر کلاس‌های این پکیج هستند کلاس `Math` حاوی متدهایی برای انجام محاسبات ریاضی است و کلاس `Class` مشخص کننده اطلاعات مربوط به کلاس یک آبجکت است.

تمرینات

- (۱) متدهی بنویسید که آرایه ای از int دریافت کند و آرایه ای از Integer برگرداند در آرایه تولید شده، آبجکتهای Integer معادل عددی داده های int در آرایه وارد شده باشند.
- (۲) تمرین قبل را برای داده های char, boolean, double و long نیز انجام دهید.
- (۳) متدهی بنویسید که آرایه ای از int عدد int دریافت کند و یک آرایه int برگرداند این متده عناصر آرایه به اندازه عددی که به صورت پارامتر دریافت کرده است به راست شیف می دهد و آرایه حاصل شده را برمی گرداند به عنوان مثال اگر آرایه ورودی [1, 4, 5, 3, 7] و عددی که دریافت کرده ۲ باشد، آرایه تولید شده به صورت زیر خواهد بود: [0, 0, 1, 4, 5]
- (۴) (در پیاده سازی این متده از System.arraycopy استفاده کنید)
- (۵) برنامه ای بنویسید که زمان فعلی سیستم را به صورت hh:mm:ss در کنسول برنامه چاپ کند.
- (۶) متدهی بنویسید که یک عدد double به عنوان شعاع دایره دریافت می کند و مساحت دایره را به صورت یک عدد double برمی گرداند.
- (۷) متدهی بنویسید که آرایه ای از اعداد double دریافت کند و با گرد کردن عناصر آن، آرایه ای از اعداد int برگرداند.
- (۸) متدهی بنویسید که دو آرایه int به عنوان پارامتر دریافت کند، اگر دو آرایه دارای تعداد عناصر مساوی بوده و قدر مطلق عناصر نظیر آنها نیز با یکدیگر مساوی باشند true و در غیر این صورت false برگرداند.
- (۹) کلاس Rational که در فصل شی گرایی مطرح شد و معرف یک «کسر» است را در نظر بگیرید. برنامه ای بنویسید و یک آبجکت از روی این کلاس بسازید، سپس با فراخوانی متده clone آن، آبجکت جدیدی بسازید که معادل آبجکت قبلی است، آبجکت تولید شده را در خروجی استاندارد چاپ کنید. حال کلاس Rational را از اینترفیس Cloneable پیاده سازی کرده و متده clone آنرا نیز پیاده سازی کنید، حال مجددا برنامه خود را اجرا کنید و مقادیر آبجکت جدید را ملاحظه نمایید.
- (۱۰) متدهی بنویسید که یک آبجکت به عنوان پارامتر دریافت می کند و نام کلاس آن آبجکت، نام متدها و نام فیلد های آن کلاس را در کنسول برنامه چاپ می کند.
- (۱۱) متدهی بنویسید که دو آبجکت به صورت پارامتر دریافت کند و مشخص کند که آیا دو آبجکت در یک پکیج قرار دارند یا خیر.

۳

کاراکترها و رشته ها

به هر یک از حروفی که توسط صفحه کلید تایپ می‌کنید یک کاراکتر (character) گفته می‌شود البته گستره کاراکترها بیشتر از حروف صفحه کلید است و در واقع شامل رقم‌های اعداد، کاراکترهای خاص مانند (\$, \$+* , ، کاراکترهای کنترلی (" " ، "\r" و ...) که قابل مشاهده نیستند و حروف الفبای زبانهای مختلف (انگلیسی، فرانسوی، فارسی، عربی و ...) می‌شود.

در جاوا امکان تعریف متغیر از نوع کاراکتر وجود دارد برای تعریف متغیر کاراکتری از کلمه کلیدی char استفاده می‌شود. مقدار متغیر کاراکتری داخل تک کوپیشن (single quote) قرار می‌گیرد.

```
char a = 'c';
```

جمله فوق، متغیر a را از نوع کاراکتر و با مقدار اولیه c تعریف می‌کند.

می‌توانید به یک متغیر کاراکتری، یک مقدار unicode انتساب دهید (از مقادیر unicode برای نمایش کاراکترهای زبانهای مختلف استفاده می‌شود):

```
char a='\u1232';
```

اندازه داده کاراکتری در جاوا ۱۶ بیت است. این بدین معناست که هر کاراکتر با یک عدد ۱۶ بیتی مشخص می‌شود (اعداد ۰ تا ۶۵۵۳۶ مشخص کننده کاراکترهای مختلف است) به هر کدام از این اعداد که مشخص کننده یک کاراکتر هستند، کد اسکی آن کاراکتر گفته می‌شود. مثال‌های زیر این مطلب را نشان می‌دهد:

```

1 package ir.atlassoft.javase.chapter13;
2
3 public class CharDemo {
4     public static void main(String[] args) {
5
6         char ch1, ch2 ;
7
8         ch1 = 88;
9         ch2 = 'Y';
10
11        System.out.println("ch1 and ch2: ");
12        System.out.println(ch1 + "" + ch2);
13
14    }
15 }
```

کد ۱۳-۱

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.

```
ch1 and ch2: X Y
```

بنابراین همانطور که ملاحظه می کنید، 88 کد اسکی X است. به مثال دیگر زیر توجه کنید.

```

1 package ir.atlassoft.javase.chapter13;
2
3 public class CharDemo2 {
4     public static void main(String[] args) {
5
6         char ch1;
7         ch1 = 'Y';
8
9         System.out.println("ch1 Contains: "+ch1);
10
11        ch1++;
12        System.out.println("ch1 is now " + ch1);
13
14    }
```

به مجموعه ای از کاراکترها یک رشته گفته می شود، هر رشته را در جاوا با استفاده از کلاس String نشان می دهند

بر خلاف char که یک داده اولیه (primitive type) محسوب می شود یک کلاس جاواست که معرف مجموعه ای از کاراکترهاست. یک String می تواند ترکیبی از تمام کاراکترهای عددی (0,1,2,...,9)، غیر عددی (a,b,...) و کاراکترهای خاص (+,-,\$,...) باشد. مقدار متغیرهای String داخل دابل کوتبیش (double quotation) قرار می گیرد.

```
String str="sample string";
```

سازنده های کلاس String

کلاس String دارای ۹ سازنده است، مثال زیر، استفاده از ۷ سازنده از این کلاس را نشان می دهد.

```

1 package ir.atlassoft.javase.chapter13;
2
3 public class StringDemo{
4
5     public static void main (String[] args){
6         char[] charArray ={'b', 'i', 'r', 't',
7                            'h', 'd', 'a', 'y'};
8         byte[] byteArray =
9             { (byte) 'n', (byte) 'e', (byte) 'w',
10                  (byte) ' ', (byte) 'y', (byte) 'e',
11                  (byte) 'a', (byte) 'r' };
12
13         String s, s1, s2, s3,
14             s4, s5, s6, output;
15
16         s = new String ("hello");
17
18         //using string constructors
19         s1 = new String();
20         s2 = new String(s);
21         s3 = new String(charArray);
22         s4 = new String(charArray, 6,3);
```

```

23         s5 = new String(byteArray);
24         s6 = new String(byteArray, 4, 4);
25
26
27         output="s1=" + s1 +
28             "\ns2=" + s2 +
29             "\ns3=" + s3 +
30             "\ns4=" + s4 +
31             "\ns5=" + s5 +
32             "\ns6=" + s6;
33
34     }
35 }
```

کد ۱۳-۳

در جمله

s = new String ("hello");

یک آبجکت String با مقدار "hello" ساخته شده است این سازنده یک رشته را به عنوان پارامتر دریافت کرده است. در جمله

s1 = new String();

رشته ای ساخته شده که دارای هیچ کاراکتری نیست، سازنده این رشته هیچ پارامتری دریافت نکرده است. در جمله

s2 = new String(s);

رشته جدیدی از روی رشته موجود، ساخته شده است در این حالت مقدار رشته جدید، یک کپی از رشته قبلی خواهد بود سازنده این رشته، یک رشته دیگر را به عنوان پارامتر دریافت کرده است. در جمله

s3 = new String(charArray);

با استفاده از آرایه ای از کاراکترها، یک رشته ساخته شده است این رشته حاوی یک کپی از مقادیر داخل آرایه است. در جمله

s4 = new String(charArray, 6, 3);

از روی قسمتی از کاراکترهای درون آرایه ای از کاراکتر، یک رشته ساخته شده است سازنده این رشته، علاوه بر آرایه ای از کاراکتر، دو عدد int نیز به عنوان پارامتر دریافت کرده است عدد اول مشخص کننده موقعیت شروع (offset) کاراکتر در آرایه و عدد دوم مشخص کننده تعداد کاراکترهایی است که باید درون رشته کپی شوند به این ترتیب، مقدار رشته s4 برابر "day" خواهد بود در صورتیکه دو عدد مشخص شده خارج از

کاراکترها و رشته ها

محدوده طول آرایه باشند سازنده خطای StringIndexOutOfBoundsException تولید خواهد کرد. جمله

```
s5 = new String(byteArrey, 4, 4);
```

یک رشته با نام s5 ساخته می شود سازنده این رشته آرایه ای از بایت به همراه دو عدد int را به عنوان پارامتر دریافت می کند همانند سازنده قبلی، عدد اول مشخص کننده موقعیت شروع (offset) بایت در آرایه و عدد دوم مشخص کننده تعداد بایت هایی است که باید درون رشته کپی شوند مقدار این رشته "year" خواهد بود. همانند قبل، خارج از محدوده بودن دو عدد باعث بروز خطای StringIndexOutOfBoundsException خواهد شد.

توجه: همچنان که قبلاً گفته شد، هر کاراکتر 16 بیت یا دو بایت است، تمام کاراکترهای لاتین فقط دارای 8 بیت با معنی هستند و بقیه بیتهای آنها بلااستفاده است، بنابراین هر کاراکتر لاتین را می توانید با یک byte نشان دهید. مثال فوق نشان می دهد که با تبدیل هر کاراکتر لاتین به یک بایت، آن کاراکتر هیچ تغییری نمی کند.

```
S6= new String(byteArray);
```

از روی مقادیر داخل یک آرایه از String یک byte ساخته شده است. در جمله

```
S7 = new String(buffer);
```

از روی یک آبجکت از کلاس StringBuffer یک رشته ساخته شده است. کلاس StringBuffer مشابه کلاس String است و از آن برای کار کردن با رشته ها استفاده می شود. تفاوت و StringBuffer به طراحی داخلی آنها بر می گردد از نظر عملکرد و کارایی، StringBuffer نسبت به String مزیت دارد. در جمله

```
buffer = new StringBuffer("welcome to java programming");
```

یک آبجکت از کلاس StringBuffer و با مقدار "welcome to Java Programming" ساخته شده است.

استفاده از کاراکترهای کنترلی

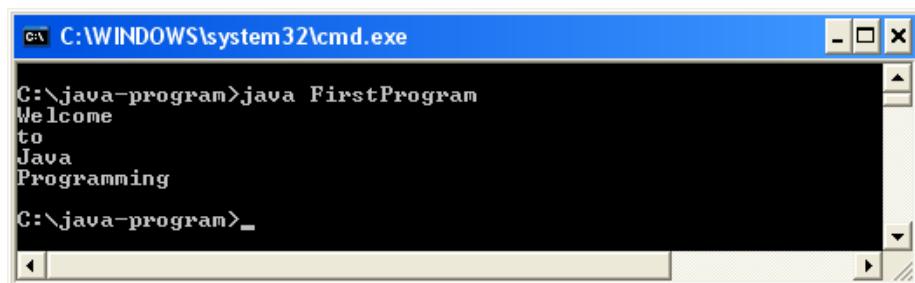
اگر رشته‌ای که توسط جملات `System.out.println()` و `System.out.print()` چاپ می‌شوند حاوی کاراکتر `\n` باشند این کاراکتر روی صفحه چاپ نخواهد شد و در عوض بقیه رشته که بعد از `\n` قرار دارند در خط بعدی چاپ می‌شوند مثلاً با اجرای برنامه زیر:

```

1 package ir.atlassoft.javase.chapter13;
2
3 public class FirstProgram {
4     public static void main(String[] args) {
5         System.out.println("Welcome\n\tto\nJava\nProgramming");
6     }
7 }
```

کد ۱۳-۴

خروجی نیز به صورت زیر خواهد بود:



شکل ۱۳-۴. خروجی اجرای کلاس FirstProgram

به کاراکترهایی مشابه `\n` که چاپ نمی‌شوند و در عوض با معنی خاصی تفسیر می‌شوند، «کاراکترهای کنترلی» گفته می‌شود جدول زیر لیستی از کاراکترهای کنترلی و نتیجه نمایش هریک را نشان می‌دهد

کاراکتر کنترلی	توضیح
\n	این کاراکتر مشخص کننده خط جدید است موقعیت کرسر را به ابتدای خط جدید منتقل می کند.
\t	این کاراکتر، کرسر را به اندازه یک tab به صورت افقی جابجا می کند.
\r	این کاراکتر، کرسر را به ابتدای خط فعلی منتقل می کند سپس هر کاراکتری که نوشته شود کاراکترهای خط فعلی را overwrite می کند به این کاراکتر carriage return می گویند
।।	برای چاپ یک \ در خروجی بکار می رود.
।"	برای چاپ یک " در خروجی بکار می رود

جدول ۱-۱۳. کاراکترهای کنترلی

تمرین: از کاراکترهای فوق در یک رشته استفاده کنید و آن رشته را توسط جمله System.out.println() در کنسول برنامه چاپ کنید.

متصل کردن رشته ها به یکدیگر

در جاوا، عملگر + برای متصل کردن دو یا چند رشته به یکدیگر استفاده می شود. مثال زیر چندین رشته را که با عملگر + به یکدیگر متصل شده اند، نشان می دهد.

```
String age = "9";
String s = "He is " + age + " years old. ";
System.out.println(s);
```

یکی از موارد کاربرد عملی اتصال رشته ها، وقتی است که در یک برنامه، رشته ای طولانی دارید در چنین مواردی به جای اینکه یک خط طولانی را برای مشخص کردن رشته بکار ببرید می توانید رشته را به چندین قسمت تقسیم نموده و آنها را با عملگر + به یکدیگر متصل کنید.

```
1 package ir.atlassoft.javase.chapter13;
2
3 //using concatenation to prevent long lines
4 public class ConCat {
5     public static void main(String[] args) {
6
7         String longStr = "This could have been " +
8             "a very long line that would have " +
```

```

9          "wrapped around. But string concatenation " +
10         "prevents this. " ;
11
12         System.out.println(longStr);
13
14     }
15 }
```

کد ۵

ترکیب یک رشته با داده های دیگر

رشته ها را می توان با انواع داده های دیگر ترکیب کرد به عنوان نمونه، مثال زیر دو رشته را با یک عدد int ترکیب کرده و از روی آنها رشته جدیدی ساخته است.

```

int age = 9;
String s = "He is " + age + " years old" ;
System.out.println(age);
```

در ترکیب رشته ها و انواع داده های دیگر کاملاً دقت کنید چون ممکن است با نتایج شگفت انگیزی مواجه شوید مثلاً برنامه زیر را تصور کنید:

```

String s = "four: " + 2 + 2 ;
System.out.println(s);
```

نتیجه اجرای جملات فوق به صورت زیر خواهد بود.

```
four: 22
```

علت این نتیجه، اولویت عملگرهای همانطور که می دانید برای محاسبه یک عبارت، آن عبارت از سمت چپ ارزیابی می شود برای محاسبه نتیجه String s = "four:" + 2 + 2؛ نیز، چون از هیچ پرانتزی استفاده نشده است، ابتدا حاصل 2 "four: " + 2 محاسبه شده که نتیجه آن رشته "four: 2" خواهد بود سپس مقدار آن با عدد 2 ترکیب می شود که نتیجه نهایی آن رشته "four: 22" خواهد شد.

در صورتیکه هدف شما از رشته فوق، رشته " four: 4" باشد باید جملات خود را به صورت زیر اصلاح کنید:

```
String s = "four: " + (2 + 2);
```

کاراکترها و رشته ها

به این ترتیب ابتدا حاصل `2 + 2` محاسبه شده و نتیجه آن با رشته "four:" ترکیب می شود.

متدهای کلاس String

در ادامه متدهای کلاس String و نحوه کار با آنها خواهید آموخت.

length()

مقدار برگشتی این متدهای int است که مشخص کننده طول String است (تعداد کاراکترهایی که در درون رشته قرار دارند) مثلاً با اجرای

```
String str = "sample";  
int len = str.length();
```

مقدار متغیر len برابر ۶ خواهد بود.

متدهای length را می توان مستقیماً روی یک رشته از کاراکترها فراخوانی کرد:

```
int length = "sample".length();  
System.out.println("length of sample is" + length);
```

charAt()

این متدهای int به عنوان پارامتر دریافت می کنند و کاراکتری را که در آن موقعیت از رشته قرار دارد را برمی گردانند مثلاً با اجرای

```
String str="sample";  
char c = str.charAt(4);
```

مقدار متغیر c برابر 'l' خواهد بود زیرا 'l' در موقعیت 4 از رشته فوق قرار دارد.
در صورتیکه عدد مشخص شده خارج از محدوده رشته باشد، فراخوانی این متدها باعث بروز خطای IndexOutOfBoundsException می شود.

getChars()

کاراکترهای یک رشته را به صورت آرایه ای از کاراکترها برمی گرداند. این متدهای چهار پارامتر دریافت می کنند.

```
void getChars(int sourceStart,  
             int sourceEnd,  
             char[] target,  
             int targetStart)
```

پارامتر اول که یک عدد int است نقطه شروع offset کاراکترهایی که قرار است کپی شوند را مشخص می‌کند. پارامتر دوم هم یک عدد int است مشخص کننده نقطه پایان کاراکترهایی است که قرار است کپی شوند. پارامتر سوم مشخص کننده آرایه‌ای است که کاراکترها در آن قرار می‌گیرند و پارامتر چهارم که یک عدد int است مشخص کننده نقطه شروع در آرایه مقصد است که قرار است کاراکترها در آنجا کپی شوند. مثلاً

```
String str = "hello world";
char[] charArray = new char[12];
str.getChars(7, 5, charArray, 3);
```

کارکترهای رشته "world" را با شروع از موقعیت 3 آرایه، در آن کپی می‌کند.

getBytes()

() getBytes() تمام کاراکترهای داخل رشته را با تبدیل به بایت، به صورت آرایه ای از بایت برمی‌گرداند در اینصورت هر کدام از بایتها، معادل یکی از کاراکترهای داخل رشته است در حقیقت این متده است در حقیقت این متده شکل این متده است:

```
byte[] getBytes()
```

یکی از مهمترین کاربردهای این متده وقتی است که می‌خواهید یک رشته Unicode را به محیطی منتقل کنید که Unicode را پشتیبانی نمی‌کند.

toCharArray()

در صورتیکه بخواهید تمام کاراکترهای داخل یک رشته را بدست آورید آسانترین راه آن، استفاده از متده toCharArray() است شکل این متده است:

```
char[] toCharArray()
```

این متده آرایه ای از کاراکتر را بر می‌گرداند که داخل رشته قرار دارند.

مقایسه کردن دو String

برای مقایسه کردن دو رشته String می‌توانید از متدهای equals(), equalsIgnoreCase() و regionMatches() استفاده کنید.

equals()

این متده یک String را با String دیگر مقایسه می‌کند شکل این متده بصورت زیر است:

```
boolean equals()
```

کاراکترها و رشته ها

در صورتیکه تمام کاراکترهای دو رشته با یکدیگر یکسان باشند (یعنی کاراکترها از همه نظر حتی از لحاظ کوچک و بزرگ بودن حرف با یکدیگر یکسان باشند) فراخوانی این متدها `true` و در غیر اینصورت `false` برمی‌گرداند.

```
String s1="sample";
String s2="sample";
boolean comparevalve= s1.equals(s2);
```

با اجرای جملات فوق مقدار متغیر `comparevalve` خواهد بود اما

```
String s1= "sample";
String s2= "Sample";
boolean comparevalve=s1.equals(s2);
```

با اجرای جملات فوق مقدار متغیر `comparevalve` خواهد شد.

`equalsIgnoreCase()`

دو رشته `String` را بدون توجه به بزرگ و کوچک بودن حروف با یکدیگر مقایسه می‌کند در صورتیکه دو رشته از نظر تعداد و نوع کاراکترها یکسان باشند مقدار `true` و در غیر این صورت `false` برمی‌گرداند.

```
String s1 = "sample";
String s2 = "Sample";
boolean comparevalve = s1.equalsIgnoreCase(s2);
```

با اجرای جملات فوق، مقدار متغیر `comparevalve` خواهد بود. مثال زیر بکارگیری متدهای `equals()` و `equalsIgnoreCase()` نشان می‌دهد.

```
1 package ir.atlassoft.javase.chapter13;
2
3 public class EqualsDemo{
4
5     public static void main(String[] args) {
6         String s1 = "Hello" ;
7         String s2 = "Hello";
8         String s3 = "Good-bye";
9         String s4 = "HELLO";
```

```

10
11     System.out.println(s1 +
12         " equals " + s2 + " => " +
13         s1.equals(s2));
14
15     System.out.println(s1 +
16         " equals " + s3 + " => " +
17         s1.equals(s3));
18
19     System.out.println(s1 +
20         " equals " + s4 + " => " +
21         s1.equals(s4));
22
23     System.out.println(s1 +
24         " equalsIgnoreCase " + s4 +
25         " => " +
26         s1.equalsIgnoreCase(s4));
27
28 }
29 }
```

کد ۶

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود:

```
s1 equals s2 => true
s1 equals s3 => false
s1 equals s4 => false
s1 equalsIgnoreCase s4 => true
```

regionMatches()

با استفاده از این متدهی می توانید قسمتهایی از دو رشته را با یکدیگر مقایسه کنید نحوه فراخوانی این متدهی بصورت زیر است:

```
regionMatches (int startIndex,
               String str2,
               int str2StratIndex,
               int numChars)
```

این متدهی چهار پارامتر دارد:

کاراکترها و رشته ها

پارامتر اول یک عدد int است که مشخص کننده موقعیت اولین کاراکتر در اولین رشته است (منظور از اولین رشته، همان رشته‌ای است که متد () regionMatches آن فراخوانی شده است)

پارامتر دوم، مشخص کننده رشته دوم است.

پارامتر سوم، مشخص کننده موقعیت اولین کاراکتر در دومین رشته است که مقایسه از آن نقطه شروع می‌شود.

پارامتر چهارم، یک عدد int است که مشخص کننده تعداد کاراکترهایی است که با یکدیگر مقایسه می‌شوند.

مثال فراخوانی

```
"sample string".regionMatches(7, "tri", 0, 3);
```

مقدار true بر می‌گرداند اما فراخوانی

```
"sample string".regionMatches(6, "str", 0, 3);
```

مقدار false بر می‌گرداند

نوع دیگری متد () regionMatches نیز وجود دارد که علاوه بر پارامترهایی که متد فوق دارد یک پارامتر boolean نیز دارد.

```
regionMatches(boolean caseIgnore,  
             int startIndex,  
             String str2,  
             int str2StartIndex,  
             int numChars)
```

در صورتیکه مقدار پارامتر true، caseIgnore باشد مقایسه دو بخش رشته، بدون توجه به بزرگی یا کوچکی حروف انجام خواهد شد.

مثال فراخوانی

```
"Sample string".regionMatches(true, 0, "sam", 0, 3);
```

مقدار true اما فراخوانی

```
"Sample string".regionMatches(false, 0, "sam", 0, 3);
```

مقدار false بر می‌گرداند.

توجه: برای مقایسه دو String هرگز از عملگر == استفاده نکنید این عملگر فقط برای مقایسه داده‌های نوع اولیه (مانند int, float, char, boolean,...) استفاده می‌شود در صورتیکه بخواهید دو آبجکت را مقایسه کنید باید از متدهای equals استفاده کنید (در اینصورت آن کلاس حتماً باید متدهای equals را پیاده‌سازی کرده باشد، کلاس String این متدها را پیاده‌سازی کرده است بنابراین می‌توانید از آن استفاده کنید).

انطباق ابتدای رشته با رشته دیگر

متدهای startsWith() و endsWith() امکان می‌دهند تا انطباق ابتدای رشته را با رشته دیگر بررسی کنیم.

```
boolean startsWith (String str)
boolean startsWith (String str, int i)
```

از متدهای startsWith برای بررسی اینکه آیا یک رشته با رشته دیگر شروع می‌شود یا خیر استفاده می‌شود. مثلاً:

```
"sample".startsWith ("sam");
```

مقدار برمی‌گرداند true

همچنین در صورتیکه از شکل دوم این متدها استفاده شود علاوه بر یک رشته، یک عدد int نیز دریافت می‌کند از این شکل متدهای توان برای تعیین اینکه آیا رشته‌ای که در موقعیت آم از یک رشته قرار دارد با رشته دیگر شروع می‌شود یا خیر استفاده کرد.

```
boolean endsWith(String str)
```

از متدهای endsWith برای بررسی اینکه آیا یک رشته با رشته دیگر خاتمه می‌یابد یا خیر، استفاده می‌شود مثلاً:

```
"sample".endsWith ("lea");
```

مقدار برمی‌گرداند false.

compareTo()

همچنان که ملاحظه کردید متدهای مختلفی برای مقایسه کردن دو رشته وجود دارد اما تمامی این متدها، دو رشته را از نظر یکسان بودن مقایسه می‌کنند. در برخی موارد ممکن است بخواهید دو رشته را از نظر اینکه کدامیک بزرگتر، کوچکتر یا مساوی هستند مقایسه نماییم یک رشته از رشته دیگر کوچکتر است اگر در

کاراکترها و رشته ها

دیکشنری لغات قبل از دیگری باید، یک رشته از رشته دیگر بزرگتر است اگر در دیکشنری لغات بعد از دیگری باید متده است compareTo () برای چنین موردی طراحی شده است. شکل این متده به صورت زیر است:

```
int compareTo(String str)
```

رشته ای است که با رشته دیگر (رشته ای که متده compareTo () از آن رشته فراخوانی شده است)، مقایسه می شود. نتیجه اجرای این متده به صورت زیر خواهد بود:

مقدار برگشتی	معنی
منفی (-1)	رشته ای که متده compareTo () از آن فراخوانی شده، کوچکتر از پارامتر متده است.
ثبت (+1)	رشته ای که متده compareTo () از آن فراخوانی شده، بزرگتر از پارامتر متده است.
صفر (0)	دو رشته با یکدیگر برابرند.

جدول ۱-۱۳. نتایج فراخوانی متده compareTo()

در زیر یک برنامه نمونه را ملاحظه می کنید که یک آرایه از رشته ها را مرتب (sort) می کند در این برنامه از متده compareTo () استفاده شده تا بر اساس الگوریتم bubble sort مرتب سازی صورت پذیرد.

```
1 package ir.atlassoft.javase.chapter13;
2
3 //a bubble sort for strings
4
5 public class SortString {
6
7     static String arr[] = {"Now", "is", "the", "time",
8                           "for", "all", "good", "men",
9                           "to", "come", "to", "the",
10                          "aid", "of", "their", "country"};
11
12    public static void main(String[] args) {
13        for (int i = 0; i < arr.length; i++) {
14            for (int j = 0; j < arr.length; j++) {
15                if (arr[i].compareTo(arr[j]) < 0) {
16                    String t = arr[j];
```

```

17             arr[j] = arr[i];
18             arr[i] = t;
19         }
20     }
21 }
22
23 //print array elements to standard output
24 for (int i = 0; i < arr.length; i++) {
25     System.out.println(arr[i]);
26
27 }
28 }
29 }
```

کد ۷-۱

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود:

```

Now
aid
all
come
country
for
good
is
men
of
the
the
their
time
to
To
```

همچنان که نتیجه اجرای برنامه فوق نشان می دهد متدهای compareTo() و compareToIgnoreCase() حروف بزرگ الفبای انگلیسی را مقدم بر حروف کوچک تصور می کند، به عبارت دیگر حروف بزرگ الفبای انگلیسی کوچکتر از حروف کوچک الفبای انگلیسی هستند (توجه داشته باشید که کد اسکی حروف بزرگ، کوچکتر از کد اسکی حروف کوچک است). در این مثال کلمه "Now" کوچکتر از کلمه "aid" نشان داده شده است. در صورتیکه می خواهید مقایسه بین دو رشته بدون توجه به کوچک بودن یا بزرگ بودن حروف الفبا صورت پذیرد می توانید از متدهای compareToIgnoreCase() استفاده کنید. شکل این متدها به صورت زیر است:

```
int compareToIgnoreCase(String str)
```

مکانیابی کاراکترهای داخل String

کلاس String دارای دو متده است که با استفاده از آنها می توانید یک کاراکتر یا یک رشته را درون یک رشته دیگر مکانیابی کنید.

- موقعیت اولین کاراکتر یا رشته مورد نظر را در رشته دیگر بر می گرداند.
- موقعیت آخرین کاراکتر یا رشته مورد نظر را در یک رشته دیگر بر می گرداند.

در هر دو متده فوق درصورتی که کاراکتر یا رشته مورد نظر در رشته یافت شود موقعیت آن رشته یا کاراکتر که یک عدد int است برگردانده می شود در غیر اینصورت اگر کاراکتر یا رشته مورد نظر یافت نشود مقدار 1 برگردانده می شود.
برای جستجوی یک کاراکتر در یک رشته، متدهای indexOf() و lastIndexOf() به صورت زیر استفاده می شوند

```
int indexOf(int ch)  
int lastIndexOf(int ch)
```

در متدهای فوق، ch مشخص کننده کاراکتری است که جستجو بر اساس آن صورت می پذیرد
برای جستجوی یک رشته داخل رشته دیگر، متدهای indexOf() و lastIndexOf() به صورت زیر استفاده می شوند:

```
int indexOf(String str)  
int lastIndexOf(String str)
```

رشته ای است که داخل رشته دیگر جستجو می شود.
همچنین شما می توانید با استفاده از متدهای زیر، جستجوی خود را از نقطه خاصی داخل رشته شروع کنید.
`int indexOf(int ch, int startIndex)`
`int lastIndexOf(int ch, int startIndex)`

```
int indexOf(String str, int startIndex)  
int lastIndexOf(String str, int startIndex)
```

در متدهای فوق startIndex موقعیت کاراکتری را در رشته مشخص می کند که جستجو از آن نقطه آغاز می شود. مثال زیر نحوه بکارگیری متدهای فوق را نشان می دهد.

```

1 //demonstrate indexOf() and lastIndexOf()
2
3 public class indexOfDemo {
4
5     public static void main(String[] args) {
6
7         String s = "Now is the time for all good men " +
8             "to come to the aid of their country." ;
9
10        System.out.println(s);
11        System.out.println("indexOf(t) = " +
12                           s.indexOf('t'));
13
14        System.out.println("indexOf(the) = " +
15                           s.indexOf("the"));
16
17        System.out.println("lastIndexOf(the) = " +
18                           s.lastIndexOf ("the"));
19
20        System.out.println("indexOf(t, 10) = " +
21                           s.indexOf ('t', 10));
22
23        System.out.println("lastIndexOf (t, 60) = " +
24                           s.lastIndexOf ('t', 60));
25
26        System.out.println("indexOf(the, 10) = " +
27                           s.indexOf ("the", 10));
28
29        System.out.println("lastIndexOf (the, 60) = " +
30                           s.lastIndexOf("the", 60));
31
32    }
33 }

```

کد A

خروجی برنامه فوق به صورت زیر خواهد بود.

```

Now is the time for all good men to come to the aid of their
country.
indexOf(t) = 7
indexOf(the) = 7
lastIndexOf(the) = 55

```

```
indexOf(t, 10) = 11  
lastIndexOf (t, 60) = 55  
indexOf(the, 10) = 44  
lastIndexOf (the, 60) = 55
```

استخراج رشته های زیر مجموعه یک رشته

با استفاده از متده است `substring()` می توانید یک رشته جدید بسازید که کاراکترهای آن برگرفته از رشته قبلی است به عنوان مثال رشته "ding" یک زیررشته از رشته "building" محسوب می شود یا رشته "oo" یک زیررشته از رشته "school" است.

```
String substring(int start)
```

با فراخوانی این متده با استفاده از کاراکترهایی که در موقعیت `start` به بعد قرار دارند یک رشته جدید بسازید. در صورتیکه موقعیت `start` خارج از محدوده رشته باشد فراخوانی این متده باعث بروز خطای `StringIndexOutOfBoundsException` خواهد شد. با اجرای جملات

```
String s1 = "Sample string";  
String s2= s1.substring(4);
```

مقدار `s2` برابر "e string" خواهد شد و اجرای

```
String s2= s1.substring (12);
```

باعث بروز `StringIndexOutOfBoundsException` خواهد شد زیرا 12 از طول رشته `s1` بیشتر است.

```
String substring (int start, int end)
```

با فراخوانی این متده با استفاده از کاراکترهایی که از موقعیت `start` تا موقعیت `end` در رشته قرار دارند یک رشته جدید بسازید (البته رشته جدید، شامل کاراکتری که در موقعیت `end` قرار دارد نمی شود) همانند قبل، در صورتیکه اعداد `start` یا `end` خارج از محدوده رشته باشد، خطای `StringIndexOutOfBoundsException` رخ خواهد داد.

```
1 package ir.atlassoft.javase.chapter13;  
2
```

```

3  public class StringReplace {
4
5      public static void main(String[] args) {
6
7          String org = "This is a test. This is, Too";
8          String search = "is";
9          String sub = "was";
10         String result = "";
11         int i ;
12
13     do{ //replace all matching substrings
14         System.out.println(org);
15         i = org.indexOf(search);
16         if(i != -1){
17             result = org.substring(0, i);
18             result = result + sub ;
19             result = result +
20                 org.substring(i + search.length());
21             org = result ;
22         }
23     }while(i != -1);
24
25 }
26 }
```

کد ۹

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.

```
This is a test. This is, Too
Thwas is a test. This is, Too
Thwas was a test. This is, Too
Thwas was a test. Thwas is, Too
Thwas was a test. Thwas was, Too
```

اتصال دو String

اگرچه می توانید با استفاده از عملگر `+` دو رشته را به یکدیگر متصل کنید اما در کلاس `String` متدهای `concat()` نیز به همین منظور تعریف شده است، با استفاده از این متدهای نیز می توانید دو رشته را به یکدیگر متصل کنید شکل این متدها به صورت زیر است:

```
String concat(String s)
```

کاراکترها و رشته ها

با فراخوانی این متدهای `s` به انتهای رشته اصلی چسبانده می شود.
مثلا با فراخوانی

```
String s = "First ".concat ("Second");
```

رشته `s` برابر "First Second" خواهد شد جمله زیر نیز همان نتیجه را خواهد داشت:

```
String s = "First "+"Second";
```

متدهای دیگر

```
replace (char c1,char c2)
```

فراخوانی این متدها، یک `String` جدید می سازد که در آن تمام کاراکترهای `c1` با کاراکتر `c2` جایگزین شده اند.

```
toUpperCase()
```

یک `String` جدید می سازد که در آن تمام کاراکترها به حرف بزرگ تبدیل شده اند.

```
toLowerCase()
```

یک `String` جدید می سازد که در آن تمام کاراکترها به حرف کوچک تبدیل شده اند.

```
trim()
```

فراخوانی این متدها، یک `String` جدید می سازد که در آن تمام کاراکترهای غیرقابل مشاهده از ابتداء و انتهای رشته حذف شده اند.

```
toCharArray()
```

یک آرایه از کاراکتر بر می گرداند که حاوی تمام کاراکترهای موجود در رشته است.

استفاده از متدهای `valueOf()`

در کلاس `String` چندین متدهای `static` تعریف شده است که از آنها می توانید برای تبدیل انواع داده ها به `String` استفاده کنید

```
valueOf (char[] charArray)
```

این متدهایی از کاراکتر را دریافت و از روی مقادیر آرایه، یک `String` می سازد.

```
valueOf (char[] charArray, int offset, int count)
```

این متدها از روی انواع داده‌ای که به صورت پارامتر دریافت می‌کند و از روی کاراکترهایی که در موقعیت `offset` تا `offset+count` قرار دارند یک `String` می‌سازد.

```
valueOf(boolean b)
valueOf(char c)
valueOf(int I)
valueOf(long l)
valueOf(float f)
valueOf(double d)
```

این متدها از روی انواع داده‌ای که به صورت پارامتر دریافت می‌کند یک `String` می‌سازد.

```
valueOf(Object o)
```

متدهای آبجکت `o` را به `String` تبدیل می‌کند (این کار را با فراخوانی متدهای `toString()` از آن آبجکت انجام می‌دهد).

کلاس StringBuffer

یکی دیگر از کلاسهای جاوا برای کار کردن با رشته‌ها، کلاس `StringBuffer` است. `StringBuffer` کلاسی شبیه `String` است اما قابلیت‌های بیشتری دارد کلاس `String` دارای طول ثابت و کاراکترهای غیرقابل تغییر است به عبارت دیگر هرگاه یک آبجکت از روی کلاس `String` ساخته می‌شود نمی‌توان مقدار آن رشته را تغییر داد (توجه داشته باشید که این مطلب با تغییر مقدار یک متغیر متفاوت است زیرا در آنجا مقدار یک متغیر رشته‌ای را با مقدار دیگر جایگزین می‌کنید ولی در مقدار اولی تغییری ایجاد نمی‌شود فقط مقدار قبلی آنرا گذاشته و مقدار جدیدی به آن انتساب می‌دهید). کلاس `StringBuffer` به شما امکان می‌دهد تا در هر قسمتی از رشته، کاراکترها یا رشته‌های جدیدی اضافه یا کم کنید.

برخی از سازنده‌های مهم کلاس `StringBuffer` را در زیر ملاحظه می‌کنید:

```
StringBuffer()
StringBuffer(int size)
StringBuffer(String str)
```

سازنده اول که پیش فرض است (سازنده‌ای که هیچ پارامتری دریافت نمی‌کند)، یک آبجکت از روی `StringBuffer` می‌سازد که طول آن ۱۶ کاراکتر است. سازنده دوم یک عدد `int` دریافت می‌کند و بنابراین آبجکتی با طول عدد مشخص شده می‌سازد. سازنده سوم که یک رشته دریافت می‌کند یک آبجکت با طول "طول رشته + ۱۶" ساخته و با رشته‌ای که دریافت نموده مقداردهی می‌کند.

کاراکترها و رشته ها

capacity() و length()

با فراخوانی متد capacity() می توان ظرفیت یک StringBuffer را بدست آورد ظرفیت مشخص کننده تعداد کل کاراکترهایی است که یک StringBuffer می تواند در خود جای دهد. در هر حال، هر آبجکت StringBuffer ممکن است قسمتی از ظرفیت خود پر شده باشد و قسمتی دیگر خالی باشد.

متد length() تعداد کاراکترهایی که در حال حاضر در StringBuffer قرار دارند را برمی گرداند شکل این دو متد به صورت زیر است:

```
int length()
int capacity()
```

مثال زیر نحوه بکارگیری این دو متد را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter13;
2
3 // StringBuffer length vs. capacity.
4 class StringBufferDemo {
5     public static void main(String args[]) {
6         StringBuffer sb = new StringBuffer("Hello");
7
8         System.out.println("buffer = " + sb);
9         System.out.println("length = " + sb.length());
10        System.out.println("capacity = " + sb.capacity());
11    }
12 }
```

کد ۱۳-۱۰

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.

```
buffer = Hello
length = 5
capacity = 21
```

ensureCapacity()

با استفاده از این متد می توانید ظرفیت یک آبجکت StringBuffer را تغییر دهید وقتی رشته یا کاراکتری را در آبجکتی از StringBuffer قرار می دهید، اگر فضای کافی در وجود نداشته باشد، به صورت اتوماتیک، ظرفیت آن به اندازه ۱۶ واحد افزایش می یابد

متدهای `ensureCapacity()` در مواردی کاربرد دارد که مدام رشته های کوچکی را به `StringBuffer` اضافه می کنید در چنین مواردی که افزایش مکرر ظرفیت `StringBuffer` باعث کاهش کارایی آن می شود، با استفاده از این متدهای توان ظرفیت `StringBuffer` را یکباره افزایش داد و باعث افزایش کارایی آن شد.
شکل این متدهای زیر است:

```
void ensureCapacity(int capacity)
```

مشخص کننده اندازه ظرفیت جدید `capacity` است.

setLength()

با فراخوانی این متدهای توانید طول `StringBuffer` را تغییر دهید، طول `StringBuffer` مشخص کننده تعداد کاراکترهایی است که در حال حاضر در `StringBuffer` وجود دارند در صورتیکه طول `StringBuffer` باشد کاراکترهای خالی به آن اضافه می شوند و در صورتیکه طول جدید بیشتر از طول فعلی `StringBuffer` باشد کاراکترهایی که خارج از محدوده طول جدید باشند از حذف می شوند.

setCharAt() و charAt()

با استفاده از این دو متدهای توانید کاراکترهایی را که در یک موقعیت خاص از `StringBuffer` قرار دارند را بدهست آورید یا تغییر دهید شکل این دو متدهای زیر است:

```
char charAt(int where)  
void setCharAt(int where, char ch)
```

مشخص کننده نقطه خاصی داخل `StringBuffer` است متدهای `charAt()` یک عدد دریافت می کند و کاراکتری را که در موقعیت آن عدد قرار دارد بر می گرداند، متدهای `setCharAt()` یک عدد و یک کاراکتر دریافت می کند و کاراکتر را در موقعیتی که توسط عدد مشخص شده است قرار می دهد. عدد `where` در دو متدهای فوق باید یک عدد غیر منفی و در محدوده طول `StringBuffer` باشد. مثال زیر بکارگیری این دو متدهای را نشان می دهد.

```

1 package ir.atlassoft.javase.chapter13;
2
3 // Demonstrate charAt() and setCharAt().
4
5 class setCharAtDemo {
6
```

کاراکترها و رشته ها

```
7  public static void main(String args[]) {  
8      StringBuffer sb = new StringBuffer("Hello");  
9      System.out.println("buffer before = " + sb);  
10     System.out.println("charAt(1) before = " +  
11         sb.charAt(1));  
12     sb.setCharAt(1, 'i');  
13     sb.setLength(2);  
14     System.out.println("buffer after = " + sb);  
15     System.out.println("charAt(1) after = " +  
16         sb.charAt(1));  
17 }  
18 }
```

کد ۱۳-۱۰

در زیر خروجی اجرای برنامه فوق را ملاحظه می کنید:

```
buffer before = Hello  
charAt(1) before = e  
buffer after = Hi  
charAt(1) after = i
```

getChars()

با استفاده از این متد می توانید تمام کاراکترهایی داخل StringBuffer را داخل آرایه ای از کاراکترها کپی کنید. شکل کلی متد () getChars به صورت زیر است:

```
void getChars(int sourceIndex,  
             int sourceEnd,  
             char[] target,  
             int targetIndex)
```

مشخص کننده دو نقطه ای هستند که کاراکترهای بین آن دو نقطه در آرایه کپی می شوند sourceEnd و sourceIndex آرایه کپی می شوند targetSource در آن کپی می شوند و نقطه ای در آرایه است که کپی کردن کاراکترها از آن نقطه شروع می شود.

append()

با استفاده از این متد می توان یک رشته را به StringBuffer اضافه نمود. اشکال مختلف این متد به صورت زیر هستند:

```
StringBuffer append(String str)
```

```
StringBuffer append(int num)
StringBuffer append(Object o)
```

برای پارامترهایی که رشته نیستند ابتدا با استفاده از متد `String.valueOf()`، رشته معادل آنها محاسبه شده، سپس رشته محاسبه شده به انتهای `StringBuffer` افزوده می شود مقدار برشگشتی متد `append()` همان `StringBuffer` ای است که متد `append()` آن فراخوانی شده است به این ترتیب می توان به صورت زنجیروار چندین متد `append()` را فراخوانی کرد. مثال زیر این موضوع را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter13;
2
3 // Demonstrate append().
4
5 class AppendDemo {
6
7     public static void main(String args[]) {
8         String s;
9         int a = 42;
10        StringBuffer sb = new StringBuffer(40);
11
12        s = sb.append("a =").append(a).append("!").toString();
13        System.out.println(s);
14    }
15
16 }
```

کد ۱۳-۱۱

insert()

با استفاده از این متد می توانید یک رشته را در یک موقعیت خاص از `StringBuffer` اضافه کنید (توجه داشته باشید که با استفاده از متد `append()` یک رشته به انتهای `StringBuffer` اضافه می شود) همانند متد `append()`، این متد پارامترهای غیر رشته ای نیز دریافت می کند. اشکال مختلف این متد به صورت زیر است:

```
StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
StringBuffer insert(int index, Object obj)
```

کاراکترها و رشته ها

برای پارامترهایی که رشته نیستند ابتدا با استفاده از متد `String.valueOf()`، رشته معادل آنها محاسبه شده، سپس رشته محاسبه شده در موقعیت `index` اضافه می شود مثال زیر استفاده از این متد را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter13;
2
3 // Demonstrate insert().
4
5 class insertDemo {
6
7     public static void main(String args[]) {
8         StringBuffer sb = new StringBuffer("I Java!");
9
10        sb.insert(2, "like ");
11        System.out.println(sb);
12    }
13
14 }
```

کد ۱۳-۱۲

نتیجه اجرای این برنامه به صورت زیر خواهد بود

I like Java!

reverse()

با استفاده از این متد ترتیب کاراکترهای درون `StringBuffer` معکوس می شود شکل این متد به صورت زیر است:

```
StringBuffer reverse()
```

این متد یک آجکت `StringBuffer` برمیگرداند که همان آجکتی است که متد `reverse()` آن فرآخوانی شده است. مثال زیر بکارگیری این متد را نشان می دهد:

```
1 package ir.atlassoft.javase.chapter13;
2
3 // Using reverse() to reverse a StringBuffer.
4
5 public class ReverseDemo {
```

```

6
7     public static void main(String args[]) {
8         StringBuffer s = new StringBuffer("abcdef");
9
10        System.out.println(s);
11        s.reverse();
12        System.out.println(s);
13    }
14
15 }
```

کد ۱۲-۱۳

اجرای این متدهای زیر را بدنبال خواهد داشت:

```
abcdef
fedcba
```

deleteCharAt() و delete()

با استفاده از این دو متدهای یک یا چندین کاراکتر را از StringBuffer حذف کنید شکل این دو متدهای زیر است:

```
StringBuffer delete(int startIndex, int endIndex)
StringBuffer deleteCharAt(int loc)
```

با استفاده از متدهای `delete()` می‌توانید مجموعه‌ای از کاراکترها را که در محدوده `startIndex` تا `endIndex` قرار دارند از StringBuffer حذف نمود. اما متدهای `deleteCharAt()` کاراکتری را که در موقعیت `loc` از StringBuffer قرار دارد از آن حذف می‌کند. هر دو متدهای `delete()` یا `deleteCharAt()` از آن فرآخوانی شده برمی‌گردانند که همان آبجکتی است که متدهای `delete()` یا `deleteCharAt()` از آن

فرآخوانی شده است.

مثال زیر بکارگیری این دو متدهای `delete()` و `deleteCharAt()` نشان می‌دهد:

```

1 package ir.atlassoft.javase.chapter13;
2
3 // Demonstrate delete() and deleteCharAt()
4
5 public class DeleteDemo {
```

کاراکترها و رشته ها

```
7  public static void main(String args[]) {  
8      StringBuffer sb =  
9          new StringBuffer("This is a test.");  
10  
11      sb.delete(4, 7);  
12      System.out.println("After delete: " + sb);  
13  
14      sb.deleteCharAt(0);  
15      System.out.println("After deleteCharAt: " +  
16          sb);  
17  }  
18  
19 }
```

۱۳-۱۴ کد

خروجی اجرای این برنامه به صورت زیر خواهد بود.

```
After delete: This a test.  
After deleteCharAt: his a test.
```

replace()

با استفاده از این متد می توانید کاراکترهایی که در محدوده خاصی از `StringBuffer` قرار دارند را با یک رشته دیگر جایگزین کنید.

شکل این متد به صورت زیر است:

```
StringBuffer replace(int startIndex,  
                     int endIndex,  
                     String str)
```

در متدهای `startIndex`- `endIndex`، `str`، رشته کاراکترهایی می شود که در موقعیت `startIndex` تا `endIndex` قرار دارند.

مثال زیر استفاده از این متد را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter13;  
2  
3 public class ReplaceDemo {  
4  
5     public static void main(String args[]) {
```

```

6      StringBuffer sb =
7          new StringBuffer("This is a test.");
8
9      sb.replace(5, 7, "was");
10     System.out.println("After replace: " + sb);
11 }
12
13 }
```

کد ۱۳-۱۵

نتیجه اجرای این مثال بصورت زیر خواهد بود:

After replace: This was a test.

substring()

با استفاده از این متدهای توانید قسمتی از کاراکترهای داخل StringBuffer را به صورت یک رشته با استخراج کنید این متدهای دو شکل زیر است:

```
String substring(int startIndex)
String substring(int startIndex, int endIndex)
```

متدهای اول که فقط یک عدد int دریافت می کند رشته ای را بر میگرداند که از موقعیت startIndex تا انتهای StringBuffer را شامل می شود اما متدهای دوم، دو عدد int دریافت می کند و کاراکترهایی را که از موقعیت startIndex تا endIndex-1 قرار دارند را بصورت یک رشته بر می گردانند.

متدهای دیگر

متدهای دیگر	شرح
int indexOf(String str)	موقعیت اولین نمونه از str را که درون StringBuffer قرار دارد، بر می گرداند در صورتی که str یافته نشود -1 بر می گرداند.
int indexOf(String str, int startIndex)	موقعیت اولین نمونه از str را با شروع از نقطه startIndex درون StringBuffer در صورتی که str درون این محدوده یافته نشود -1 بر می گرداند.
int lastIndexOf(String str)	موقعیت آخرین نمونه از str را درون

کاراکترها و رشته ها

برمی گرداند در صورتی که str یافته شود ۱- برمی گرداند.	StringBuffer	
موقعیت آخرین نمونه از str را با شروع از نقطه startIndex، درون StringBuffer، محدوده یافته نشود ۱- برمی گرداند.	int lastIndexOf(String str, int startIndex)	
اندازه StringBuffer را به اندازه تعداد کاراکترهای موجود در آن کاهش می دهد به عبارت دیگر فضاهای خالی را از StringBuffer حذف می کند.	void trimToSize()	

مثال زیر بکارگیری برخی از این متدها را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter13;
2
3 public class IndexOfDemo {
4
5     public static void main(String args[]) {
6         StringBuffer sb = new StringBuffer("one two one");
7         int i;
8
9         i = sb.indexOf("one");
10        System.out.println("First index: " + i);
11
12        i = sb.lastIndexOf("one");
13        System.out.println("Last index: " + i);
14    }
15
16 }
```

لیست ۱۴-۱۳

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.

```
First index: 0
Last index: 8
```

خاصه

در این فصل با رشته ها و کاراکترها آشنا شدید هر کدام از دکمه های صفحه کلید معرف یک کاراکتر است البته مجموعه کاراکترها به دکمه های صفحه کلید محدود نمی شود و بسیار بیشتر از آنهاست. در جاوا داده اولیه `char` برای تعریف یک متغیر از نوع کاراکتر وجود دارد. از روی یک یا ترکیب چندین کاراکتر یک رشته ساخته می شود کلاس `String` به منظور تعریف یک متغیر رشته ای در جاوا پیش بینی شده است. رشته ها را می توان به یکدیگر متصل کرد یا آنها را دستکاری نمود متدهای `length()` تعداد کاراکترهای درون یک رشته را برمی گرداند با استفاده از متدهای `charAt()` می توان کاراکتری را که در موقعیت خاصی درون رشته قرار دارد بدست آورد متدهای `getChars()` و `getBytes()` تمام کاراکترهای درون رشته را به صورت یک آرایه از کاراکتر برمی گرداند برای مقایسه کردن دو رشته می توان از `equals()` و `equalsIgnoreCase()` استفاده نمود متدهای `startsWith()`، `endsWith()` و `endsWith()` برای تبدیل انواع داده های اولیه به `String` استفاده می شود.

کلاس `StringBuffer` مشابه کلاس `String` است و امکان کارکردن با رشته ها را فراهم می کند در مواقعي که یک برنامه حجم عملیات زیادی روی رشته ها دارد و پردازش زیادی روی رشته ها انجام می دهد (کم زیاد کردن کاراکترها یا دیگر عملیات مربوط به رشته ها) بهتر است از این کلاس استفاده کنید زیرا کارایی این کلاس بالاتر از کلاس `String` است.

تمرینات

- (۱) برنامه ای بنویسید که یک رشته را دریافت کند، سپس تمام متدهای کلاس String که در این فصل با آن آشنا شدید را روی رشته فراخوانی کند و نتیجه آنرا در کنسول برنامه چاپ کند.
- (۲) متدى بنویسید که دو String دریافت کند و ایندکس تمام تکرارهای رشته دوم در رشته اول را چاپ کند
- (۳) متدى بنویسید که یک آرایه از رشته را دریافت کند، سپس آرایه جدیدی از رشته ها تولید کرده و برگرداند عناصر این آرایه باید معکوس عناصر آرایه ای باشند که به صورت پارامتر دریافت شده است به عنوان مثال اگر آرایه ای که دریافت کرده است به صورت زیر باشد:
- ```
["Home", "Window", "Computer"]
```
- باید آرایه زیر را برگرداند:
- ```
[ "emoH", "wodniW", "retupmoc" ]
```
- در صورتیکه یکی از عناصر آرایه null باشد، آن عنصر را به صورت null برگرداند.
- (۴) متدى بنویسید که آرایه ای از رشته ها و یک رشته را دریافت نموده و اولین موقعیت آن رشته در آرایه را پیدا کند و بر گرداند. اگر رشته در آرایه وجود نداشته باشد مقدار -۱ - را برگرداند.
- (۵) متدى بنویسید که یک آرایه از رشته ها دریافت می کند سپس آن آرایه را به صورت مرتب شده براساس حروف الفبا بر می گرداند.



ساختمان داده ها و پکیج `java.util`

`java.util` حاوی مجموعه ای از کلاس‌های عمومی و پرکاربرد است. از کلاس‌هایی که در این مجموعه قرار دارند برای کار کردن با زمان و تاریخ، تولید اعداد تصادفی، عملیات روی رشته ها، فرمت دهی داده ها استفاده می شود همچنین، برخی از مهمترین ساختمان داده های جاوا از قبیل لیست، صف و استک در این پکیج قرار دارند.

ساختمان داده ها

آرایه مهمترین ساختمان داده جاواست. اما آرایه ها به همان اندازه که مهم هستند ساده هم هستند، طول یک آرایه استاتیک است و پس از اینکه در حافظه ایجاد شد تغییر آن ممکن نیست، عناصر آرایه ها ذاتاً مرتب شده نیستند و جستجو در آنها می تواند پرهزینه باشد. عناصر آرایه ها می توانند تکراری باشند و هیچ کنترلی در آرایه برای جلوگیری از عناصر تکراری انجام نمی شود. بنابراین در مواقعي که دنبال ساختمان داده ای باشید که داده های تکراری را نپذیرد، یا عناصر خود را به صورت مرتب شده نگهداری کند، یا یا اندازه خود را به صورت دینامیک تغییر دهد آرایه نمی تواند نیاز شما را برآورده کند. در این صورت ممکن ساختمان داده هایی که در پکیج `java.util` قرار دارد سازی شده اند نیاز شما را برآورده کنند.

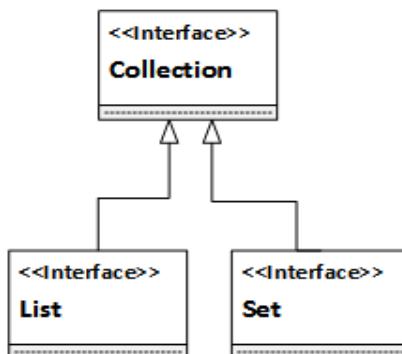
ساختمان داده های پکیج `java.util` تحت عنوان `collection` شناخته می شوند و به دو دسته `List` و `Set` قابل تقسیم هستند.

`List` ساختمان داده ای مشابه آرایه است که عناصر خود را بر اساس موقعیت نگهداری می کند مثلا مشابه آرایه ها، گفته می شود عنصر اول لیست، عنصر دوم لیست... عنصر `n`ام لیست. وقتی داده ای را به یک `List` اضافه می شود، این داده به صورت پیش فرض در انتهای لیست اضافه می شود، هرچند این امکان نیز وجود دارد تا آنرا در هر جایی از لیست اضافه کنید. `List` همانند آرایه، عناصر تکراری می پذیرد بنابراین یک داده می تواند به تعداد نامحدود در یک لیست ظاهر شود. لیست ذاتا مرتب شده نیست اما اگر داده های لیست قابل مرتب شدن باشند می توان با استفاده از کلاسهای دیگر ترتیب داده ها را مرتب نمود.

در مقابل لیست، `Set` قرار دارد که عناصر خود را به ترتیبی که خود صلاح می داند! نگهداری می کند بنابراین اگر چه می گوییم مثلا فلان `Set` دارای `n` عضو است اما عنصر اول یا عنصر `n`ام معنی ندارد چون نمی توان داده ای را در موقعیت `n`ام جای داد و نه می توان داده ای را که در موقعیت `n`ام قرار دارد بدست آورد. اصلی ترین خصوصیت یک `Set` این است که عنصر تکراری قبول نمی کند بنابراین اگر یک داده ای را چندین بار به یک `Set` اضافه کنید فقط یکبار در `Set` اضافه می شود.

اگرچه `List` و `Set` دو ساختمان داده با کاربرد متفاوت هستند اما چون ساختمان داده هستند، رفتار و خصوصیات مشترکی هم دارند مثلا هر دوی آنها `add()`, `remove()`, `size()` و `clear()` دارند که به ترتیب برای اضافه کردن یک داده، تشخیص تعداد داده های موجود، حذف یک داده، یا حذف تمام داده ها از ساختمان داده استفاده می شوند به همین دلیل، متدها مشترک `List` و `Set` در قالب یک اینترفیس دیگر با نام `Collection` تعریف شده است تا `List` و `Set` که آنها نیز اینترفیس هستند دارای پدر مشترک `Collection` باشند.

شكل ۱۴-۱ ارتباط اینترفیس سه اینترفیس `List`, `Set` و `Collection` را نشان می دهد.



شكل ۱۴-۱. ارتباط اینترفیس های `List`, `Set` و `Collection`

تا اینجا مشخص شد که `List` و `Set` اینترفیس هستند ممکن است بپرسید اگر این دو اینترفیس هستند پس نمی توانیم از روی آنها آجکت بسازیم، پس چگونه می توانیم از `List` و `Set` استفاده کنیم؟

ساختمان داده ها و پکیج `java.util`

اینترفیس `List` توسط کلاس `ArrayList` و اینترفیس `Set` توسط کلاس `HashSet` پیاده سازی شده است این بدین معنی است که برای استفاده از `List` می توانید به صورت زیر آبجکتی از کلاس `ArrayList` بسازید.

```
List list = new ArrayList();
```

و برای استفاده از `Set` به صورت زیر آبجکتی از `HashSet` ایجاد کنید.

```
Set set = new HashSet();
```

وقتی داده ای را به لیست اضافه کنید، همیشه آن داده به لیست اضافه می شود. اما اگر داده ای که قبلاً به

یک `Set` اضافه کرده اید مجدداً به آن `Set` اضافه کنید برای بار دوم اضافه نمی شود! ممکن است بپرسید

بر چه مبنایی می تواند تکراری بودن یک داده را تشخیص دهد؟

اگر داده ای که به `Set` اضافه می کنید، یک داده پایه (مثل `char`, `int`, ...) باشد به سادگی یکسان بودن

داده قابل تشخیص است اما اگر داده ای که به `Set` اضافه می شود یک آبجکت باشد (یعنی از روی یک

کلاس ایجاد شده باشد) تکراری بودن از روی متدهای `equals()` و `hashCode()` آن کلاس تشخیص

داده می شود در ادامه در مبحث `HashSet` در مورد نحوه پیاده سازی و کارکرد این دو متدهای بیشتر صحبت

خواهیم کرد.

اما همانطور که دقت کرده اید هیچ کدام از ساختمان داده های `ArrayList` و `HashSet` داده های خود

را به صورت مرتب شده نگهداری نمی کنند. `ArrayList` اساساً داده های خود را به همان ترتیب که در

آن اضافه می شوند نگهداری می کند.

اینترفیس `SortedSet` معرف یک مرتب شده است که علاوه بر اینکه عناصر تکراری را نمی پذیرد،

تمام عناصر خود را به صورت مرتب شده نگهداری می کند که البته لازمه آن این است که عناصری که به آن

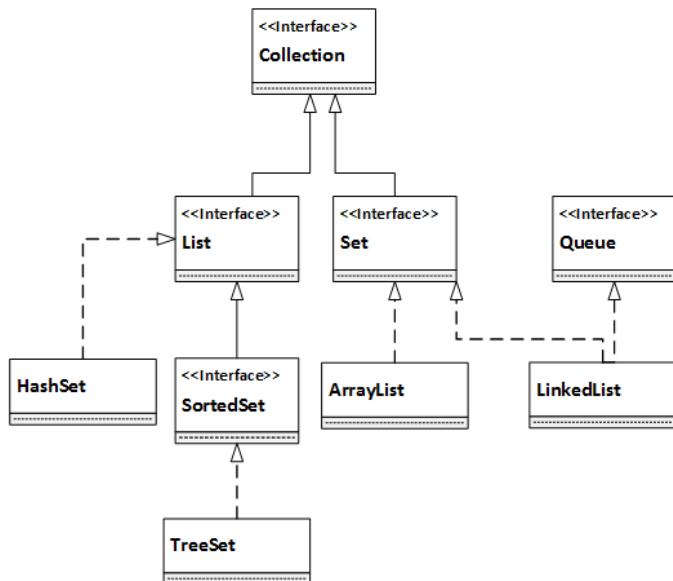
اضافه می شوند قابل مقایسه باشند تا `TreeSet` بتواند آنها را مرتب کند. داده های پایه (اعداد،

`char`, `boolean`) همگی قابل مقایسه اند ولی هر آبجکتی که در `TreeSet` قرار داده می شود باید

اینترفیس `java.lang.Comparable` را پیاده سازی کند.

از آنجاییکه `SortedSet` یک اینترفیس است، در عمل می بایست از کلاس `TreeSet` پیاده سازی کرده

است آبجکت ایجاد شود. ارتباط بین این کلاسها در شکل ۱۴-۲ نشان داده شده است.



شکل ۱۴-۲. ارتباط اینترفیس‌های Collection

دیاگرام فوق حاوی برخی کلاسها و اینترفیس‌های جدید است که تا به حال نامی از آنها نبرده ایم. مثلاً اینترفیس Queue نشان داده شده است که معرف یک صف است که هر آبجکتی که زودتر به آن وارد شود زودتر نیز از آن خارج می‌شود (دقیقاً مثل صف نانوایی). این اینترفیس توسط کلاس LinkedList پیاده سازی شده است که همزمان اینترفیس List را نیز پیاده سازی کرده است بنابراین کلاس LinkedList دوگانه دارد و علاوه بر اینکه می‌تواند به عنوان لیست استفاده شود می‌تواند به عنوان صف نیز استفاده گردد. دو جدول زیر لیست مهمترین کلاسها و اینترفیس‌های پکیج java.util را نشان می‌دهد.

AbstractCollection	Properties	PropertyResourceBundle
AbstractList	PriorityQueue	Random
AbstractMap	Formmater	ResourceBundle
AbstractQueue	GregorianCalendar	Scanner
Date	HashMap	UUID
AbstractSet	HashSet	Stack
ArrayList	Hashtable	StringTokenizer
Arrays	Observable	Timer
Vector	LinkedHashMap	TimerTask
Calendar	LinkedHashSet	TimeZone
Collections	LinkedList	TreeMap
TreeSet	ListResourceBundle	Locale

جدول ۱۴-۳. کلاس‌های Collection

Collection	List	Iterator
Comparator	ListIterator	Set

ساختمان داده ها و پکیج `java.util`

Enumeration	Map	SortedMap
EventListener	Map.Entry	SortedSet
Formattable	Queue	

جدول ۲-۱۴. اینترفیس های `java.util`

در مجموعه کلاسهای پکیج `java.util` ساختمان داده هایی از قبیل `Properties` و `Stack` نیز وجود دارند که به ترتیب به عنوان استک (آجکتی که دیرتر وارد شود زودتر خارج می شود) و دیکشنری متنی قابل استفاده هستند.

همچنین در این مجموعه، چندین کلاس و اینترفیس تعریف شده اند که تحت عنوان `Map` شناخته می شوند. از کلاسها و اینترفیسها `Map` برای نگهداری داده ها به صورت `key/value` استفاده می شود.

اینترفیس های `Collection`

هر کدام از کلاسها `Collection` یکی از اینترفیس های زیر را پیاده سازی کرده اند بنابراین می توان گفت، تمام کلاسها `Collection` متدهای مشترکی خواهند داشت.

اینترفیس	شرح
<code>Collection</code>	این اینترفیس که زیربنای تمام کلاسها تحت عنوان <code>Collection</code> است حاوی متدهایی برای مدیریت و نگهداری مجموعه ای از آجکتهاست.
<code>List</code>	این اینترفیس از اینترفیس <code>Collection</code> مشتق شده و در آن، متدهای دیگری برای کار کردن با لیستی از آجکتها طراحی شده است.
<code>Queue</code>	این اینترفیس که از <code>Collection</code> مشتق شده، متدهایی برای پیاده سازی کردن یک صف تعریف می کند. یک صف ساختمان داده ای است که داده ها از یک طرف به آن وارد و از طرف دیگر خارج می شوند.
<code>Set</code>	این اینترفیس از <code>Collection</code> مشتق شده، و برای کار کردن با مجموعه ای از آجکتها بکار می رود آجکتها باید در <code>Set</code> قرار می گیرند باید منحصر بفرد باشند به عبارت دیگر، <code>Set</code> آجکتها تکراری را نگهداری نمی کند.
<code>SortedSet</code>	این اینترفیس از <code>Set</code> مشتق شده تا برای مدیریت مجموعه ای از آجکتها غیرتکراری که مرتب نیز شده اند بکار رود.

جدول ۳-۱۴. اینترفیس های `Collection`

در ادامه، جزئیات هریک از اینترفیسها فوق توضیح داده می شود.

اینترفیس Collection

این اینترفیس زیربنای تمام کلاس‌های Collection محسوب می‌شود زیرا تمام کلاس‌های Collection این اینترفیس یا اینترفیس‌هایی که از این اینترفیس مشتق شده‌اند را پیاده سازی کرده‌اند. متد‌هایی که در اینترفیس Collection تعریف شده‌اند در جدول ۱۴-۴ لیست شده‌اند.

شرح	متد
با فراخوانی این متد می‌توانید آبجکت obj را به Collection اضافه کنید در صورتیکه Collection ای که این متد آنرا فراخوانی می‌کنید عناصر تکراری را مجاز ندانسته باشد و شما عنصر تکراری به آن اضافه کنید این متد مقدار false بر می‌گرداند و در غیراینصورت مقدار true بر می‌گرداند.	boolean add(Object obj)
تمام عناصر موجود در c را به Collection ای که متد addAll() آن فراخوانی شده است را اضافه می‌کند در صورتیکه این عمل با موفقیت همراه باشد این متد مقدار true و در غیراینصورت مقدار false بر می‌گرداند.	boolean addAll(Collection c)
تمام عناصر را از Collection حذف می‌کند.	void clear()
در صورتیکه آبجکت obj در Collection وجود داشته باشد فراخوانی این متد مقدار true بر می‌گرداند.	boolean contains(Object obj)
در صورتیکه تمام عناصر درون c، درون Collection جاری وجود داشته باشد فراخوانی این متد مقدار true بر می‌گرداند و در غیراینصورت مقدار false برخواهد گرداند.	boolean containsAll(Collection c)
چنانچه عناصر Collection جاری با عناصر آبجکت obj (که خود یک Collection است) یکسان باشند فراخوانی این متد مقدار true و در غیراینصورت مقدار false بر می‌گرداند.	boolean equals(Object obj)
«hash code» مربوط به Collection را بر می‌گرداند.	int hashCode()
در صورتیکه Collection جاری خالی باشد فراخوانی این متد مقدار true بر می‌گرداند.	boolean isEmpty()
Collection مربوط به iterator را بر می‌گرداند.	Iterator iterator()

ساختمان داده ها و پکیج `java.util`

گرداند.	
در صورتیکه آبجکت <code>obj</code> در <code>Collection</code> جاری وجود داشته باشد آنرا از <code>Collection</code> حذف نموده و مقدار <code>true</code> برmi گرداند در غیراینصورت (اگر آبجکت <code>obj</code> در <code>Collection</code> وجود نداشته باشد) فراخوانی این متده است <code>remove</code> برmiگرداند.	<code>boolean remove (Object obj)</code>
تمام عناصر داخل <code>c</code> را از <code>Collection</code> جاری حذف می کند در صورتیکه <code>Collection</code> جاری تغییر کند مقدار برگشتی این متده <code>true</code> خواهد بود و در غیراینصورت مقدار برگشتی <code>false</code> خواهد بود.	<code>boolean removeAll (Collection c)</code>
تمام عناصر داخل <code>Collection</code> جاری، به غیر از موارد موجود در <code>c</code> ، را از <code>Collection</code> جاری حذف می کند.	<code>boolean retainAll (Collection c)</code>
تعداد عناصر موجود در <code>Collection</code> جاری را برmi گرداند.	<code>int size()</code>
آرایه ای حاوی تمام عناصر موجود در <code>Collection</code> جاری را برmi گرداند عناصر آرایه از عناصر <code>Collection</code> کپی خواهند شد.	<code>Object[] toArray()</code>

جدول ۱۴-۴. متدهای اینترفیس `Collection`

با فراخوانی متده `add()` می توانید یک آبجکت به `Collection` اضافه کنید و با فراخوانی متده `addAll()` می توانید محتوای یک `Collection` را به `Collection` دیگر اضافه کنید متده `remove()` برای حذف یک آبجکت از `Collection` بکار می رود با استفاده از متده `clear()` می توانید یک `Collection` را خالی کنید و متدهای `removeAll()` برای حذف قسمتی از عناصر `Collection` و متده `retainAll()` برای باقی گذاردن قسمتی از عناصر و حذف بقیه بکار می رود. با استفاده از متده `contains()` می توانید مشخص کنید که آیا آبجکت خاصی درون `Collection` قرار دارد یا ندارد.

برای تعیین اینکه آیا تمام عناصر یک `Collection` درون `Collection` دیگر قرار دارد می توانید از متده `containsAll()` استفاده کنید که آیا `Collection` خالی تعیین می کند که آیا بوده یا حاوی عنصر یا عناصری است با فراخوانی متده `size()` می توانید تعداد عناصر داخل `Collection` را بدست آورید.

متدهای `toArray()`، عناصر داخل `Collection` را به صورت آرایه ای از آبجکت برmi گرداند در صورتیکه بخواهید مقدار برگشتی این متده آرایه ای از یک کلاس خاص باشد باید یک آرایه از کلاس مورد نظر نیز به عنوان پارامتر به این متده ارسال کنید در اینصورت می توانید آرایه برگشتی توسط متده را به نوع مورد `Cast` کنید.

متدهای equals()، تساوی یا عدم تساوی دو Collection را بررسی می کند.

اینترفیس List

این اینترفیس از Collection مشتق شده است و بنابراین متدهای تعریف شده در Collection را به ارث برده است عناصر داخل این List از طریق ایندکس آنها که از ۰ شروع می شوند قابل دستیابی هستند List می تواند حاوی عناصر تکراری باشد این اینترفیس علاوه بر متدهایی که از Collection می توانند حاوی عناصر تکراری باشند، دارای متد addAll() (علاوه بر دیگر نسخه های addAll() که در اینترفیس همچنین حاوی نسخه های دیگری از addAll() هستند) که از طریق آنها می توانید عناصر یک Collection دیگر را در ایندکس خاصی از List اضافه کنید.

در صورتیکه بخواهید به یک عنصر خارج از محدوده یک List دسترسی پیدا کنید خطای OutOfBoundsException رخ خواهد داد

شرح	متدهای
این متدهای آبجکت obj را در موقعیت index اضافه می کند اضافه شدن obj، تمام عناصری که در آن موقعیت به بعد قرار دارند را یک واحد جابجا می کند بنابراین داده ها به هیچ وجه از دست نخواهد رفت.	void add(int index, Object obj)
تمام عناصر c را در موقعیت index اضافه می کند چنانچه با فراخوانی این متدهای تغییری در داده های اعمال شود این متدهای true و در false غیراينصورت مقدار برمی گرداند.	boolean addAll(int index, Collection c)
آبجکت ذخیره شده در موقعیت index را برمی گرداند.	Object get(int index)
موقعیت اولین آبجکت obj در List را برمی گرداند در صورتیکه آبجکت مشخص شده در List وجود نداشته باشد -1 برمی گرداند.	int indexOf(Object obj)
موقعیت آخرین آبجکت obj در List را برمی گرداند در صورتیکه آبجکت مشخص شده در List وجود نداشته باشد -1 - برگردانده می شود.	int lastIndexOf(Object obj)
از روی عناصر داخل List، یک آبجکت Iterator برمی گرداند.	ListIterator listIterator()

ساختمان داده ها و پکیج `java.util`

از روی عناصر داخل <code>List</code> در موقعیت <code>index</code> به بعد قرار دارند یک آبجکت <code>Iterator</code> برگردانده می شود.	<code>ListIterator listIterator(int index)</code>
عنصر موجود در موقعیت <code>index</code> حذف شده و همان عنصر برگردانده می شود	<code>Object remove(int index)</code>
آبجکتی که در موقعیت <code>index</code> قرار دارد را با <code>obj</code> جایگزین می کند.	<code>Object set(int index, Object obj)</code>
یک <code>List</code> جدید که حاوی عناصر موجود در موقعیت <code>start</code> تا <code>end-1</code> از لیست فعلی قرار دارند برگردانده می شود.	<code>List subList(int start, int end)</code>

جدول ۱۴-۵. متدهای اینترفیس `List`

برای دستیابی به عنصر ذخیره شده در یک ایندکس خاص از `List` می توانید از متدهای `get()` استفاده کنید برای تغییر یا مقداردهی به یک ایندکس خاص درون `List` می توانید از متدهای `set()` استفاده کنید این متدها علاوه مقدار جدید، ایندکس عنصر در `List` را به عنوان پارامتر دریافت می کند. برای یافتن ایندکس یک آبجکت می توانید از متدهای `lastIndexOf()` و `indexOf()` استفاده کنید. از متدهای `subList()` می توانید برای ایجاد یک `List` جدید که زیرمجموعه ای از لیست فعلی است استفاده کنید برای اینکار باید ایندکس های عناصر ابتدایی و انتهایی را مشخص کنید.

اینترفیس `Set`

این اینترفیس نیز از `Collection` مشتق شده است، خصوصیت این اینترفیس آن است که عناصر تکراری را مجاز نمی داند بنابراین در صورتیکه با استفاده از متدهای `add()` و `addAll()` عنصری تکراری را به آن اضافه کنید آن عنصر اضافه نخواهد شد و فراخوانی آن مقدار `false` بر می گردد اند در این اینترفیس هیچ متدهای اضافه ای علاوه بر متدهایی که از `Collection` به ارث برده است) تعریف نشده است.

اینترفیس `SortedSet`

این اینترفیس از `Set` مشتق شده است بنابراین عناصر تکراری را نیز مجاز نمی داند عناصر داخل `SortedSet` به صورت صعودی مرتب می شوند. این اینترفیس علاوه بر متدهایی که در `Collection` تعریف شده اند دارای متدهای دیگری نیز می باشد که لیست آنها در جدول زیر نشان داده شده است. با فراخوانی متدهای `first()` و `last()` می توانید به ترتیب اولین و آخرین عنصر داخل `SortedSet` را بدست آورید.

با استفاده از متدهای `subset()` می توانید از روی `SortedSet` فعلی یک `SortedSet` جدید سازی که زیرمجموعه ای این `SortedSet` باشد برای این کار باید اولین و آخرین آبجکت مورد نظر در `SortedSet`

را مشخص کنید اگر به زیرمجموعه‌ای از عناصر داخل `SortedSet` نیاز دارید که از عنصر اول شروع می‌شوند می‌توانید از `headset()` استفاده کنید و اگر به زیرمجموعه‌ای نیاز دارید که به آخرین عنصر داخل `SortedSet` ختم می‌شود می‌توانید متدهای `tailSet()` را بکار ببرید.

شرح	متدها
نخستین عنصر موجود در <code>SortedSet</code> را بر می‌گرداند.	<code>Object first()</code>
یک <code>SortedSet</code> جدید بر می‌گرداند که زیرمجموعه‌ای از <code>SortedSet</code> فعلی است و عناصری که از اول تا موقعیت <code>end</code> وجود دارند را شامل می‌شود.	<code>SortedSet headset(int end)</code>
آخرین عنصر موجود در <code>SortedSet</code> را بر می‌گرداند.	<code>Object last()</code>
یک <code>SortedSet</code> بر می‌گرداند که حاوی عناصر موجود در موقعیت <code>start</code> تا <code>end-1</code> است.	<code>SortedSet subset(int start, int end)</code>
یک <code>SortedSet</code> بر می‌گرداند که حاوی عناصر موجود در موقعیت <code>start</code> تا انتهای <code>SortedSet</code> فعلی قرار دارد.	<code>SortedSet tailSet(int start)</code>

جدول ۱۴-۶. لیست متدهای `SortedSet`

اینترفیس Queue

این اینترفیس که در J2SE 5 اضافه شده است از `Collection` مشتق شده و عملکرد یک صف را شبیه سازی کرده است در صف همیشه اولین عنصر اضافه شده، اولین عنصری است که از صف خارج می‌شود (صف نانوایی را به خاطر بیاورید!) اما انواع دیگری از صفات وجود دارند که ورود و خروج داده‌ها از آنها با معیارهای متفاوتی صورت می‌گیرد.

در صف `Queue` عناصر را تنها می‌توان از یک سمت و به ترتیب از آن دریافت کرد برای دسترسی به یک عنصر در صف دو متدهای `poll()` و `remove()` پیش‌بینی شده اند تفاوت این دو متدها در این است که در صورتیکه صف خالی باشد فراخوانی متدهای `null` بر می‌گرداند در صورتیکه فراخوانی `remove()` باعث تولید خطای `NoSuchElementException` می‌شود از متدهای `element()` و `peek()` می‌توانید برای بدست آوردن عناصر اول یک صف استفاده کنید این دو متدهای `remove()` و `poll()` باعث حذف عنصر از صف نمی‌شوند تفاوت دو متدهای `element()` و `peek()` در این است که در صورت خالی بودن صف فراخوانی متدهای `peek()` و `element()` باعث تولید خطای `null` بر می‌گرداند در صورتیکه فراخوانی متدهای `element()` و `peek()` باعث تولید خطای `NoSuchElementException` می‌شود

ساختمان داده ها و پکیج `java.util`

با استفاده از متد `offer()` می توانید عناصری را به یک صف اضافه کنید از آنجاییکه طول برخی از صفها محدود است و ممکن است در زمان فراخوانی این متد، آن صف پر باشد ممکن است فراخوانی `offer()` با موفقیت همراه نباشد.

شرح	متد
عنصر ابتدای صف را برمی گرداند این عنصر از صف حذف نمی شود درصورتیکه صف خالی باشد خطای <code>NoSuchElementException</code>	<code>Object element()</code>
جز <code>obj</code> را به صف اضافه می کند اگر اضافه شدن با موفقیت همراه باشد مقدار برگشتی <code>true</code> و در غیراینصورت <code>false</code> خواهد بود.	<code>boolean offer(Object obj)</code>
عنصر ابتدای صف را برمی گرداند چنانچه صف خالی باشد، <code>null</code> را برمی گرداند این عنصر از صف حذف نمی شود.	<code>Object peek()</code>
عنصر ابتدای صف را برمی گرداند چنانچه صف خالی باشد، <code>null</code> را برمی گرداند این عنصر از صف حذف می شود.	<code>Object poll()</code>
عنصر ابتدای صف را برگردانده و آنرا از صف حذف می کند فراخوانی این متد درصورت خالی بودن صف، باعث تولید خطای <code>NoSuchElementException</code> خواهد شد.	<code>Object remove()</code>

جدول ۷-۱۴. متدهای اینترفیس `Queue`

کلاس‌های `Collection`

در اینجا کلاس‌های `Collection` را معرفی می کنیم، همانطور که در ابتدای این فصل گفته شد، کلاس‌های `Collection` یکی از اینترفیس‌های فوق را پیاده سازی کرده اند. این کلاس‌ها در جدول زیر خلاصه شده اند.

متد	شرح
اکثر متدهای اینترفیس <code>Collection</code> توسط این کلاس پیاده سازی شده اند.	<code>AbstractCollection</code>
این کلاس از <code>AbstractCollection</code> مشتق شده است و اکثر متدهای اینترفیس <code>List</code> را پیاده سازی کرده است.	<code>AbstractList</code>
از اینترفیس <code>Collection</code> مشتق می شود و بخش‌هایی از اینترفیس <code>Queue</code> را پیاده سازی کرده است (در ۵ J2SE	<code>AbstractQueue</code>

اضافه شده است	
این کلاس از <code>AbstractList</code> مشتق شده است و امکان دسترسی ترتیبی به عناصر <code>Collection</code> (به جای دسترسی تصادفی) را امکان‌پذیر می‌کند.	<code>AbstractSequentialList</code>
این کلاس از <code>AbstractSequentialList</code> مشتق شده و یک لیست پیوندی ایجاد می‌کند.	<code>LinkedList</code>
این کلاس از <code>AbstractList</code> مشتق شده و یک آرایه پویا ایجاد می‌کند.	<code>ArrayList</code>
از <code>AbstractCollection</code> مشتق شده و اکثر متدهای اینترفیس <code>Set</code> را پیاده سازی کرده است.	<code>AbstractSet</code>
این کلاس از <code>AbstractSet</code> مشتق شده و برای استفاده از عناصر <code>Enum</code> بکار می‌رود. (در ۵ J2SE اضافه شده است)	<code>EnumSet</code>
این کلاس از <code>AbstractList</code> مشتق شده و از یک جدول <code>hash</code> برای نگهداری داده‌ها استفاده می‌کند.	<code>HashSet</code>
این کلاس از <code>AbstractSet</code> مشتق شده و از یک ساختار درختی برای نگهداری داده‌ها استفاده می‌کند	<code>TreeSet</code>

جدول ۱۴-۱. کلاس‌های Collection

همچنان که قبلاً گفته شد چندین کلاس قدیمی به نامهای `Hashtable`, `Stack`, `Vector` در پکیج `java.util` وجود دارند که در نسخه‌های جدید بازنگری شده اند و در قسمت پایانی این فصل بررسی می‌شوند.

ArrayList کلاس

این کلاس، اینترفیس `List` را پیاده سازی نموده است در بسیاری از موارد به جای استفاده از آرایه می‌توانید از این کلاس استفاده کنید مشکلی که در استفاده از آرایه وجود دارد این است که در زمان ساخته شدن یک آجکت از آرایه، باید طول آنرا مشخص نمود، پس از ساخته شدن، در سراسر اجرای یک برنامه طول آن ثابت می‌ماند این در حالیست که طول `ArrayList` به صورت پویا تغییر می‌کند وقتی افزایش طول لازم باشد `ArrayList` به صورت خودکار طول خود را افزایش می‌دهد تا فضای لازم برای اضافه شدن عناصر جدید را فراهم کند و وقتی آجکتها از آن حذف می‌شوند به صورت خودکار طول آن کاهش می‌یابد. سازنده‌های کلاس `ArrayList` به صورت زیر می‌باشند:

`ArrayList()`

ساختمان داده ها و پکیج java.util

```
ArrayList(Collection c)
ArrayList(int capacity)
```

سازنده اول، یک ArrayList خالی با طول پیش فرض ۱۰ می سازد، سازنده دوم، یک ArrayList می سازد که حاوی عناصر ۰ است و سازنده سوم، یک ArrayList خالی با طول capacity می سازد.
برنامه زیر نحوه استفاده از این کلاس را نشان می دهد

```
1 package ir.atlassoft.javase.chapter14;
2
3 // Demonstrate ArrayList.
4 import java.util.*;
5
6 public class ArrayListDemo {
7
8     public static void main(String args[]) {
9
10         // Create an array list.
11         ArrayList al = new ArrayList();
12
13         System.out.println("Initial size of al: " +
14             al.size());
15
16         // Add elements to the array list.
17         al.add("C");
18         al.add("A");
19         al.add("E");
20         al.add("B");
21         al.add("D");
22         al.add("F");
23         al.add(1, "A2");
24
25         System.out.println("Size of al after additions:" +
26             al.size());
27
28         // Display the array list.
29         System.out.println("Contents of al: " + al);
30
31         // Remove elements from the array list.
32         al.remove("F");
```

```

33         al.remove(2);
34
35         System.out.println("Size of al after deletions:"+
36                         al.size());
37         System.out.println("Contents of al: " + al);
38     }
39 }
```

کد ۱۴-۱

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود:

```

Initial size of al: 0
Size of al after additions: 7
Contents of al: [C, A2, A, E, B, D, F]
Size of al after deletions: 5
Contents of al: [C, A2, E, B, D]
```

دقت کنید که از آنجاییکه عناصر داخل `ArrayList` هستند بنابراین فراخوانی

```
System.out.println("Size of al:" + al);
```

محتویات رشته ای داخل `ArrayList` را در خروجی استاندارد چاپ می کند اما در صورتیکه عناصر داخل `ArrayList` از جنس دیگری جز `String` باشند برای نمایش عناصر داخل `ArrayList` باید از روشهای دیگری استفاده نمود (مثلا با استفاده از حلقه `for` عناصر داخل یک `ArrayList` را چاپ کنید). طول یک `ArrayList` به صورت خودکار تغییر می کند اما می توانید با فراخوانی متدهای `ensureCapacity()` طول آنرا به اندازه دلخواه تغییر دهید (در مواردی که مطمئن هستید که طول آرایه باید به اندازه مشخصی افزایش یابد می توانید از این متدهای استفاده کنید افزایش خودکار و متوالی طول `ArrayList` کارایی کمتری نسبت به افزایش یکباره طول آن دارد) ساختار این متدها به صورت زیر است.

```
void ensureCapacity(int cap)
```

، طول جدید `ArrayList` است `cap`.

در صورتیکه بخواهید فضای آزاد یک `ArrayList` را از آن حذف کنید یعنی طول آن به اندازه تعداد عناصر داخل آن باشد می توانید از متدهای `trimToSize()` استفاده کنید:

```
void trimToSize()
```

ساخت یک آرایه از روی عناصر داخل یک **ArrayList**

اگر بخواهید از روی عناصر داخل یک **ArrayList**، یک آرایه بسازید می توانید متده است `toArray()` را فراخوانی کنید دو نسخه از این متده وجود دارد که به صورت زیر می باشند:

`Object[] toArray()`

شکل اول، یک آرایه از آبجکت برمه گرداند اما شکل دوم این متده، یک آرایه به عنوان پارامتر دریافت می کند و آرایه ای از جنس پارامتر دریافت نموده برمه گرداند.

مثال زیر بکارگیری این متده را نشان می دهد:

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 class ArrayListToArray {
6
7     public static void main(String args[]) {
8
9         // Create an array list.
10        ArrayList al = new ArrayList();
11
12        // Add elements to the array list.
13        al.add(new Integer(1));
14        al.add(new Integer(2));
15        al.add(new Integer(3));
16        al.add(new Integer(4));
17
18        System.out.println("Contents of al: " + al);
19
20        // Get the array.
21        Integer ia[] = new Integer[al.size()];
22        ia = (Integer[]) al.toArray(ia);
23
24        int sum = 0;
25
26        // Sum the array.
27        for (int i=0; i<ia.length; i++)
28            sum += ia[i].intValue();
29
30        System.out.println("Sum is: " + sum);
```

```

31     }
32 }
```

کد ۱۴-۲

خروجی اجرای این برنامه بصورت زیر خواهد بود.

```
Contents of al: [1, 2, 3, 4]
Sum is: 10
```

کلاس LinkedList

این کلاس، اینترفیس های Queue و List را پیاده سازی کرده است و دارای دو سازنده بصورت زیر است:

```
LinkedList()
LinkedList(Collection c)
```

سازنده اول یک LinkedList خالی می سازد ولی سازنده دوم از روی عناصر داخل Collection که به عنوان پارامتر دریافت می کند یک LinkedList می سازد.

این کلاس دارای دو متده addLast() و addFirst() می باشد که با استفاده از آنها می توانید یک

آجکت به ابتدایا یا به انتهای یک LinkedList اضافه کنید. شکل این دو متده بصورت زیر است:
void addFirst(Object o)
void addLast(Object o)

- مشخص کننده آجکتی است که به اضافه می شود
- همچنین دو متده getFirst() و getLast() در این کلاس تعریف شده است که با استفاده از آنها می توانید به عناصر اول و آخر درون LinkedList دسترسی پیدا کنید.

```
Object getFirst()
Object getLast()
```

برای حذف اولین یا آخرین عنصر درون LinkedList نیز می توانید از متدهای removeFirst() و removeLast() استفاده کنید. مثال زیر استفاده از این متدها را نشان می دهد:

```

1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
```

ساخته‌مان داده‌ها و پکیج `java.util`

```
4
5  public class LinkedListDemo {
6
7      public static void main(String args[]) {
8
9          // Create a linked list.
10         LinkedList ll = new LinkedList();
11
12         // Add elements to the linked list.
13         ll.add("F");
14         ll.add("B");
15         ll.add("D");
16         ll.add("E");
17         ll.add("C");
18         ll.addLast("Z");
19         ll.addFirst("A");
20
21         ll.add(1, "A2");
22
23         System.out.println("Original contents of ll:" +
24             ll);
25
26         // Remove elements from the linked list.
27         ll.remove("F");
28         ll.remove(2);
29
30         System.out.println("Contents of ll after " +
31             " deletion: " + ll);
32
33         // Remove first and last elements.
34         ll.removeFirst();
35         ll.removeLast();
36
37         System.out.println("ll after deleting first" +
38             " and last: " + ll);
39
40         // Get and set a value.
41         String val = (String) ll.get(2);
42         ll.set(2, val + " Changed");
43
```

```

44     System.out.println("11 after change: " + ll);
45 }
46 }
```

کد ۱۴-۳

خروجی اجرای برنامه زیر بصورت زیر می باشد.

```

Original contents of ll: [A, A2, F, B, D, E, C, Z]
Contents of ll after deletion: [A, A2, D, E, C, Z]
ll after deleting first and last: [A2, D, E, C]
ll after change: [A2, D, E Changed, C]
```

از آنجائیکه کلاس `LinkedList`، اینترفیس `List` را پیاده سازی کرده است فراخوانی `() add` باعث اضافه شدن عناصر به انتهای آن می شود (فراخوانی این متده در این کلاس مشابه متده `addLast()` است) در صورتیکه بخواهید عنصری را در موقعیت خاصی از `LinkedList` اضافه کنید می توانید از فرم دیگر متده `() add` استفاده کنید که در آن علاوه بر عنصری که قرار است اضافه شود ایندکس موقعیت آن در `LinkedList` نیز مشخص شده است. همچنین با استفاده از متدهای `() set` و `() get` می توانید عنصر موجود در یک موقعیت خاص از `LinkedList` را بدست آورید یا آنرا تغییر دهید.

کلاس HashSet

این کلاس اینترفیس `Set` را پیاده سازی نموده است در این کلاس برای ذخیره سازی عناصر از مکانیسمی به نام `hashing` استفاده می شود هر آبجکتی که درون `JVM` ساخته می شود یک کد منحصر بفرد دارد که به آن `hash code` گفته می شود `HashSet` عناصر خود را بر اساس `hash code` آن عناصر مرتب می کند مزیت چنین مکانیسمی، در زمان اجرای متدهای `() add`، `() remove`، `() contains` و `() size` است با این مکانیسم، زمان اجرای این متدها، حتی برای مجموعه های بزرگ نیز ثابت می ماند. سازنده های این کلاس به صورت زیر می باشند:

```

HashSet()
HashSet(Collection c)
HashSet(int capacity)
HashSet(int capacity, float fillRatio)
```

`fillRatio` که یک عدد اعشاری بین ۰.۰ و ۱.۰ است میزان پربودن `HashSet` را قبل از افزایش طرفیت مشخص می کند مثال زیر بکارگیری این کلاس را نشان می دهد:

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class HashSetDemo {
6
7     public static void main(String args[]) {
8
9         // Create a hash set.
10        HashSet hs = new HashSet();
11
12        // Add elements to the hash set.
13        hs.add("B");
14        hs.add("A");
15        hs.add("D");
16        hs.add("E");
17        hs.add("C");
18        hs.add("F");
19
20        System.out.println(hs);
21    }
22 }
```

کد ۱۴-۴

اجرای این برنامه نیز نتیجه‌ای بصورت زیر به همراه خواهد داشت:

[D, A, F, C, B, E]

اینترفیس Iterator

یکی از روش‌های دستیابی به عناصر داخل یک Collection، استفاده از Iterator است با استفاده از Iterator می‌توانید در یک Collection حرکت کرده و عناصر را بخوانید یا حذف کنید. قبل از اینکه نحوه دستیابی به عناصر Collection را مطرح کنیم لازم است با Iterator آشنا شویم. هر کلاسی که اینترفیس‌های Iterator یا ListIterator را پیاده سازی کرده باشد یک Iterator محسوب می‌شود دو جدول ۱۴-۹ و ۱۴-۱۰ متد‌هایی که در این دو اینترفیس تعریف شده‌اند را نشان می‌دهند.

شرح	متدها
برای خواندن هریک از داده های داخل <code>Iterator</code> استفاده می شود در صورتیکه داده دیگری داخل <code>Iterator</code> وجود داشته باشد مقدار <code>true</code> و در غیراینصورت <code>false</code> برمی گرداند.	<code>boolean hasNext()</code>
عنصر بعدی را برمی گرداند در صورتیکه عنصر بعدی داخل <code>Iterator</code> وجود نداشته باشد <code>NoSuchElementException</code> رخ می دهد.	<code>Object next()</code>
عنصر جاری را حذف می کند، چنانچه متدهای <code>next()</code> پیش از این متدهای <code>remove()</code> فراخوانده نشود، خطای <code>IllegalStateException</code> رخ می دهد.	<code>void remove()</code>

جدول ۱۴-۹. متدهای اینترفیس `Iterator`

شرح	متدها
آجکت <code>obj</code> را در موقعیتی قبل از عنصر حاصل از فراخوانی <code>next()</code> درج می کند.	<code>void add(Object obj)</code>
اگر عنصر دیگری وجود داشته باشد مقدار <code>true</code> و در غیراینصورت <code>false</code> برمی گرداند.	<code>boolean hasNext()</code>
اگر عنصر قبلی وجود داشته باشد مقدار <code>true</code> و در غیراینصورت <code>false</code> برمی گرداند.	<code>boolean hasPrevious()</code>
عنصر بعدی را برمی گرداند در صورتیکه عنصر بعدی وجود نداشته باشد خطای <code>NoSuchElementException</code> رخ می دهد.	<code>Object next()</code>
ایندکس عنصر بعدی را برمی گرداند در صورتیکه عنصر بعدی وجود نداشته باشد اندازه <code>List</code> را بر می گرداند.	<code>int nextIndex()</code>
عنصر قبلی را برمی گرداند در صورتیکه عنصر قبلی وجود نداشته باشد خطای <code>NoSuchElementException</code> رخ می دهد.	<code>Object previous()</code>
ایندکس عنصر قبلی را برمی گرداند اگر موجود نباشد -1 - بر می گرداند.	<code>int previousIndex()</code>
عنصر فعلی را از <code>List</code> حذف می کند در صورتیکه <code>remove()</code> قبل از فراخوانی <code>previous()</code> یا <code>next()</code> یا <code>remove()</code> فراخوانی شود خطای <code>IllegalStateException</code> رخ می دهد.	<code>void remove()</code>
آجکت <code>obj</code> را در موقعیت عنصر بعد از فراخوانی <code>next()</code> یا <code>previous()</code> قرار می دهد.	<code>void set(Object obj)</code>

جدول ۱۴-۱۰. متدهای اینترفیس `ListIterator`

استفاده از یک `Iterator`

در تمام کلاس‌های `Collection` متدهی به نام `iterator()` وجود دارد که یک آبجکت `Iterator` بر می‌گرداند سپس با استفاده از این آبجکت می‌توانید به تمامی عناصر داخل `Collection` دسترسی پیدا کنید. مراحل استفاده از `Iterator` بصورت زیر است:

الف) با فراخوانی متدهی `iterator()` از یک `Collection` یک آبجکت `Iterator` بدست آورید.

ب) حلقه‌ای ایجاد کنید که شرط آن حلقه، مقدار برگشتی متدهی `hasNext()` باشد این بدين معنایست که تا زمانیکه مقدار برگشتی این متدهی `true` باشد حلقه ادامه پیدا کند.

ج) با فراخوانی متدهی `next()` به هر یک از عناصر دسترسی پیدا کنید.

در مثال زیر بکارگیری `ListIterator` و `Iterator` نشان داده شده اند توجه داشته باشید که تنها در اختیار `Collection` هایی است که اینترفیس `List` را پیاده سازی کرده اند.

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class IteratorDemo {
6
7     public static void main(String args[]) {
8
9         // Create an array list.
10        ArrayList al = new ArrayList();
11
12        // Add elements to the array list.
13        al.add("C");
14        al.add("A");
15        al.add("E");
16        al.add("B");
17        al.add("D");
18        al.add("F");
19
20        // Use iterator to display contents of al.
21        System.out.print("Original contents of al: ");
```

```

22         Iterator itr = al.iterator();
23         while (itr.hasNext()) {
24             String element = (String) itr.next();
25             System.out.print(element + " ");
26         }
27         System.out.println();
28
29         // Modify objects being iterated.
30         ListIterator litr = al.listIterator();
31         while (litr.hasNext()) {
32             String element = (String) litr.next();
33             litr.set(element + "+");
34         }
35
36         System.out.print("Modified contents of al: ");
37         itr = al.iterator();
38         while (itr.hasNext()) {
39             String element = (String) itr.next();
40             System.out.print(element + " ");
41         }
42         System.out.println();
43
44         // Now, display the list backwards.
45         System.out.print("Modified list backwards: ");
46         while (litr.hasPrevious()) {
47             String element = (String) litr.previous();
48             System.out.print(element + " ");
49         }
50         System.out.println();
51     }
52 }
```

۱۴-۶

خروجی اجرای برنامه فوق بصورت زیر خواهد بود.

```
Original contents of al: C A E B D F
Modified contents of al: C+ A+ E+ B+ D+ F+
Modified list backwards: F+ D+ B+ E+ A+ C+
```

ساختمان داده ها و پکیج `java.util`

دقت کنید که برای بدست آوردن عناصر داخل `Collection` در جهت مستقیم از متدهای `next()` و `hasNext()` استفاده شده است و برای بدست آوردن عناصر داخل `Collection` در جهت معکوس از متدهای `previous()` و `hasPrevious()` استفاده شده است.

حلقه `for`

قبل استفاده از حلقة `for` برای دسترسی به عناصر یک `List` را آموخته اید

```
List list = ...
for(int i=0; i<list.size(); i++) {
    Object obj = list.get(i);
    ...
}
```

به جای استفاده از حلقة `for` و بدون تعریف یک شماره به صورت زیر می توانید به عناصر یک `List` دسترسی بیابید

```
List list= ...
for(Object obj:list) {
    ...
}
```

این حلقة `for` نسبت به نسخه قبلی خود ساده تر است و شمارنده آن حذف شده است اما نکته مهمتر در مورد این حلقة `for` این است که با استفاده از آن می توانید مقادیر یک `Set` را نیز بخوانید این چیزیست که با حلقة `for` سنتی امکانپذیر نیست. از این حلقة جدید می توان برای خواندن عناصر یک آرایه نیز استفاده نمود.

Map

`Map` همانند `List`، مجموعه ای از آبجکت ها را در خود نگهداری می کند. هر آبجکتی که در `Map` نگهداری می شود دارای یک «کلید» نیز هست که با استفاده از آن کلید می توان به آبجکت مرتبط با آن دسترسی پیدا کرد. به عبارت دیگر، با استفاده از `Map` می توانید مجموعه ای از آبجکتها و کلیدهای آنها را نگهداری کنید کلید هر آبجکت، خود یک آبجکت است. تمام آبجکتها یک `Map` قرار دارند با استفاده از کلیدشان شناسایی می شوند.

اینترفیس های `Map`

سه اینترفیس در ارتباط با `Map` وجود دارد که در جدول زیر لیست شده اند:

اینترفیس	شرح
<code>Map</code>	این اینترفیس هر آبجکت را به یک کلید مرتبط می کند به عبارت دیگر هر عنصر را با

یک کلید نگهداری می کند.	
مشخص کننده یک عنصر در Map است این کلاس، کلاس داخلی Map است.	Map.Entry
این اینترفیس از Map مشتق شده و کلیدهای عناصر را بصورت صعودی مرتب می کند.	SortedMap

جدول ۱۴-۱۱. اینترفیس های Map

اینترفیس Map

با استفاده از متدهایی که توسط این اینترفیس ارایه شده اند می توانید با مشخص کردن هر کلید برای هر آبجکت، آن آبجکت را به Map اضافه کنید. در Map ها دو متدهایی به نامهای `put()` و `get()` قرار دارند که با استفاده از آنها می توانید آبجکتی را به یک Map اضافه کنید یا مقدار آبجکت مرتبط با یک کلید را بدست آورید. این اینترفیس دارای متدهایی به نامهای `values()`، `keySet()` و `entrySet()` است که هر کدام یک Collection از آبجکتهای موجود در Map، کلیدها، یا مقادیر موجود در Map را بدست می دهند. متدهایی که در اینترفیس Map تعریف شده اند در جدول زیر نشان داده شده اند:

شرح	متدها
تمام مقادیر داخل Map را از Map پاک می کند (این مقادیر در حقیقت به صورت «کلید/مقدار» هستند)	<code>void clear()</code>
درصورتیکه آبجکتی با کلید k در Map وجود داشته باشد فراخوانی این متده را <code>true</code> و در غیراینصورت مقدار <code>false</code> بر می گرداند.	<code>boolean containsKey(Object k)</code>
در صورتیکه Map حاوی عنصری با مقدار v (نه کلید) باشد حاصل فراخوانی این متده را <code>true</code> و در غیراینصورت <code>false</code> خواهد بود.	<code>boolean containsValue(Object v)</code>
یک Set حاوی عناصر درون Map بر می گرداند هر کدام از این عناصر، آبجکتهایی از کلاس <code>Entry.Set</code> خواهد بود.	<code>Set entrySet()</code>
اگر یک obj با همان عناصر باشد فراخوانی این متده را <code>true</code> بر می گرداند در غیراینصورت مقدار <code>false</code> بر می گرداند.	<code>boolean equals(Object obj)</code>
مقدار متناظر با کلید k را بر می گرداند چنانچه چنین کلیدی وجود نداشته باشد مقدار برگشتی <code>null</code> خواهد بود.	<code>Object get(Object k)</code>

ساختمان داده ها و پکیج `java.util`

«hash code» مربوط به Map را برمی گرداند.	<code>int hashCode()</code>
چنانچه Map خالی باشد، فراخوانی این متدهای مقدار <code>true</code> و درغیراینصورت مقدار <code>false</code> بر می گردد.	<code>boolean isEmpty()</code>
یک Set حاوی تمام کلیدهای درون Map بر می گردد.	<code>Set keySet()</code>
مقدار مرتبط با کلید <code>k</code> را با مقدار جدید <code>v</code> جایگزین می کند و مقدار قدیمی را بر می گرداند در صورتی که کلید موجود نباشد جایگزینی صورت نخواهد گرفت و مقدار برگشتی <code>null</code> خواهد بود.	<code>Object put(Object k, Object v)</code>
تمام مقادیر موجود در <code>m</code> را به Map اضافه می کند.	<code>void putAll(Map m)</code>
مقدار مرتبط با کلید <code>k</code> را از Map حذف می کند.	<code>Object remove(Object k)</code>
تعداد عناصر درون Map را بر می گرداند (هر یک از این عناصر به صورت زوجهای «کلید/مقدار» خواهد بود)	<code>int size()</code>
یک Collection مشکل از تمام مقادیر موجود در Map بر می گردد.	<code>Collection values()</code>

جدول ۱۲-۱۴. متدهای اینترفیس Map

اینترفیس SortedMap

این اینترفیس مشخص کننده یک Map است که عناصر داخل آن، براساس کلیدها به صورت صعودی نگهداری می شوند. متدهای این اینترفیس در جدول زیر ارایه شده اند. برای بدست آوردن زیرمجموعه ای از Map می توانید از متدهای `()`، `tailMap()` یا `headMap()` استفاده کنید که تماماً یک Map بر می گردانند. همچنین برای بدست آوردن اولین و آخرین کلید می توانید از متدهای `firstKey()` و `lastKey()` استفاده کنید. متدهایی که در اینترفیس SortedMap تعریف شده اند:

شرح	متدها
نخستین کلید موجود در SortedMap را بر می گردد.	<code>Object firstKey()</code>
یک SortedMap بر می گرداند که حاوی عناصری است که کلید آنها کوچکتر از <code>endKey</code> هستند.	<code>SortedMap headMap(Object endKey)</code>

آخرین کلید SortedMap را برمی‌گرداند.	Object lastKey()
یک SortedMap برمی‌گرداند که حاوی عناصری است که کلید آنها بزرگتر یا مساوی start و کوچکتر از end می‌باشند.	SortedMap subMap(Object start, Object end)
یک SortedMap برمی‌گرداند که حاوی عناصری است که کلیدهای آنها بزرگتر یا مساوی startKey هستند.	SortedMap tailMap(Object startKey)

جدول ۱۳-۱۴. متدهای اینترفیس SortedMap

Map.Entry اینترفیس

با استفاده از این اینترفیس می‌توانید با عناصر داخل Map کار کنید به خاطر داشته باشید که متدهای entrySet() که در اینترفیس Map تعریف شده است، یک Set متشکل از عناصر Map برمی‌گرداند، هر یک از این عناصر در مجموعه حاصل، یک آبجکت از Map.Entry خواهد بود. این آبجکت از Map.Entry تعریف شده اند در جدول زیر نشان داده شده اند.

شرح	متدهای اینترفیس
در صورتی که <code>obj</code> یک آبجکت از جنس Map.Entry باشد، مقدار بزرگتر از <code>obj</code> فعلی باشد حاصل فراخوانی آن <code>true</code> و در غیر این صورت <code>false</code> خواهد بود.	boolean equals(Object obj)
کلید مربوط به این عنصر را در Map برمی‌گرداند.	Object getKey()
مقدار مربوط به این عنصر را در Map برمی‌گرداند.	Object getValue()
«hash code» مربوط به این عنصر را از Map برمی‌گرداند.	int hashCode()
مقدار این عنصر از Map را برابر <code>value</code> قرار می‌دهد.	Object setValue(Object value)

جدول ۱۵-۱۶. متدهای اینترفیس Map.Entry

کلاس‌های Map

چندین کلاس، اینترفیس‌های Map را پیاده سازی کرده اند برخی از آنها در جدول زیر نشان داده شده اند:

شرح	نام کلاس

ساختمان داده ها و پکیج `java.util`

اکثر متدهای داخل Map را پیاده سازی کرده است.	AbstractMap
این کلاس که از AbstractMap مشتق شده است برای نگهداری داده ها براساس hash code طراحی شده است.	HashMap
این کلاس از AbstractMap مشتق شده است و داده ها را در یک ساختار درختگونه نگهداری می کند.	TreeMap
این کلاس از HashMap مشتق شده و برای ایجاد امکان اضافه کردن عناصر تکراری طراحی شده است.	LinkedHashMap

جدول ۱۴-۱۵. پیاده سازیهای Map

کلاس `HashMap`

سازنده های این کلاس عبارتند از:

```
HashMap()
HashMap(Map m)
HashMap(int capacity)
HashMap(int capacity, float fillRatio)
```

همانند HashSet یک عدد اعشاری بین ۰.۰ و ۱.۰ است که میزان پربرودن را قبل از افزایش ظرفیت مشخص می کند استفاده از این کلاس در مثال زیر نشان داده شده است:

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class HashMapDemo {
6
7     public static void main(String args[]) {
8
9         // Create a hash map.
10        HashMap hm = new HashMap();
11
12        // Put elements to the map
13        hm.put("John Doe", new Double(3434.34));
14        hm.put("Tom Smith", new Double(123.22));
15        hm.put("Jane Baker", new Double(1378.00));
16        hm.put("Tod Hall", new Double(99.22));
17        hm.put("Ralph Smith", new Double(-19.08));
```

```

18
19      // Get a set of the entries.
20      Set set = hm.entrySet();
21      Iterator itr = set.iterator();
22
23      // Display the set.
24      while (itr.hasNext()) {
25          Map.Entry entry = (Map.Entry) itr.next();
26          System.out.print(entry.getKey() + ": ");
27          System.out.println(entry.getValue());
28      }
29
30      System.out.println();
31
32      // Deposit 1000 into John Doe's account.
33      double balance =
34          ((Double) hm.get("John Doe")).doubleValue();
35      hm.put("John Doe", new Double(balance + 1000));
36
37      System.out.println("John Doe's new balance: " +
38                      hm.get("John Doe"));
39  }
40 }
```

کد ۶

در این برنامه ابتدا آبجکتی از Map برای نگهداری اطلاعات حساب بانکی افراد، ساخته شده است سپس نام افراد به عنوان کلید و موجودی حساب آنها به عنوان مقدار به این آبجکت افزوده شده است در پایان محتوای این آبجکت با فراخوانی متده است entrySet() بدل است آمده است (مقدار برگشتی متده entrySet()، یک Map حاوی تمام عناصر داخل است هر کدام از عناصر داخل این Set، هر کدام آبجکتی از Map هستند با فراخوانی متدهای get() و get() می توان کلید یا مقدار هر کدام از این عناصر را بدل است آورده). همچنین با استفاده از متده put() مقدار پیشین مرتبط با یک کلید، با مقدار جدید جایگزین شده است. نتیجه اجرای برنامه فوق بصورت زیر خواهد بود.

```
Ralph Smith: -19.08
Tom Smith: 123.22
John Doe: 3434.34
Tod Hall: 99.22
Jane Baker: 1378.0
```

John Doe's new balance: 4434.34

Arrays کلاس

این کلاس حاوی متدهایی برای کار کردن با آرایه هاست.
با استفاده از متد () asList() می توانید از روی داده های درون یک آرایه، یک List بسازید شکل کلی این
متد به صورت زیر است:

```
static List asList(Array)
```

همان آرایه ای است که از روی آن یک List ساخته می شود.
متد () binarySearch() از یک جستجوی باینری برای پیدا کردن مقدار مشخص شده استفاده می کند
این متد باید برای آرایه های مرتب شده بکار برد شود. شکل های مختلف این متد به صورت زیر است:

```
static int binarySearch(byte array[], byte value)
static int binarySearch(char array[], char value)
static int binarySearch(double array[], double value)
static int binarySearch(float array[], float value)
static int binarySearch(int array[], int value)
static int binarySearch(long array[], long value)
static int binarySearch(short array[], short value)
static int binarySearch(Object array[], Object value)
static int binarySearch(byte array[], byte value)
```

array مشخص کننده آرایه ای است که جستجو باید روی آن انجام شود و value مشخص کننده
مقداری است که باید در آرایه جستجو شود.
همچنین از این کلاس برای مقایسه دو آرایه نیز می توانید استفاده کنید چنانچه دو آرایه مساوی باشند،
فراخوانی متد () equals() مقدار true بر می گردداند در غیر اینصورت مقدار آن false خواهد بود شکل
های مختلف متد () equals() در زیر نشان داده شده اند:

```
static boolean equals(boolean array1[], boolean array2[])
static boolean equals(byte array1[], byte array2[])
static boolean equals(char array1[], char array2[])
static boolean equals(double array1[], double array2[])
static boolean equals(float array1[], float array2[])
static boolean equals(int array1[], int array2[])
static boolean equals(long array1[], long array2[])
static boolean equals(short array1[], short array2[])
static boolean equals(Object array1[], Object array2[])
```

array1 و array2، دو آرایه ای هستند که تساوی آنها بررسی می شود.

با استفاده از متدهای `fill()` می توانید مقداری را به تمام عناصر موجود در آرایه انتساب دهید این متدهای دو شکل مختلف دارد یک شکل آن کل یک آرایه را با یک مقدار مشخص پر می کند و شکل دوم آن قسمتی از یک آرایه را با یک مقدار مشخص پر می کند. متدهای اول در زیر نشان داده شده است:

```
static void fill(boolean array[], boolean value)
static void fill(byte array[], byte value)
static void fill(char array[], char value)
static void fill(double array[], double value)
static void fill(float array[], float value)
static void fill(int array[], int value)
static void fill(long array[], long value)
static void fill(short array[], short value)
static void fill(Object array[], Object value)
```

شکل دوم این متدهای قسمتی از یک آرایه را با مقدار مشخصی پر می کند به صورت زیر است:

```
static void fill(boolean array[],
                 int start,
                 int end,
                 boolean value)

static void fill(byte array[],
                 int start,
                 int end,
                 byte value)

static void fill(char array[],
                 int start,
                 int end,
                 char value)

static void fill(double array[],
                 int start,
                 int end,
                 double value)

static void fill(float array[],
                 int start,
                 int end,
                 float value)

static void fill(int array[],
                 int start,
                 int end,
```

ساختمان داده ها و پکیج java.util

```
    int value)

static void fill(long array[],
                 int start,
                 int end,
                 long value)

static void fill(short array[],
                 int start,
                 int end,
                 short value)

static void fill(Object array[],
                 int start,
                 int end,
                 Object value)
```

به مقادیری از آرایه که در محدوده start تا end-1 قرار دارند تخصیص می یابد در صورتیکه بزرگتر از start باشد خطای IllegalArgumentException رخ می دهد و در صورتیکه start و خارج از محدوده آرایه باشند خطای ArrayIndexOutOfBoundsException رخ خواهد داد.

با استفاده از متده sort() می توانید یک آرایه را به صورت صعودی مرتب کنید همانند متده fill()، این متده نیز دو شکل دارد با استفاده از شکل اول، کل یک آرایه مرتب می شود و با استفاده از شکل دوم، قسمتی از آرایه مرتب می شود.
شکل اول این متده به صورت زیر است:

```
static void sort(byte array[])
static void sort(char array[])
static void sort(double array[])
static void sort(float array[])
static void sort(int array[])
static void sort(long array[])
static void sort(short array[])
static void sort(Object array[])
```

شکل دوم این متده نیز به صورت زیر است:

```
static void sort(byte array[], int start, int end)
static void sort(char array[], int start, int end)
static void sort(double array[], int start, int end)
static void sort(float array[], int start, int end)
static void sort(int array[], int start, int end)
static void sort(long array[], int start, int end)
```

```
static void sort(short array[], int start, int end)
static void sort(Object array[], int start, int end)
```

مثال زیر بکارگیری متدهای کلاس Arrays را نشان می دهد:

```

1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class ArraysDemo {
6
7     public static void main(String args[]) {
8
9         // Allocate and initialize array.
10        int array[] = new int[10];
11        for (int i = 0; i < 10; i++)
12            array[i] = -3 * i;
13
14        // Display, sort, and display the array.
15        System.out.print("Original contents: ");
16        display(array);
17        Arrays.sort(array);
18        System.out.print("Sorted: ");
19        display(array);
20
21        // Fill and display the array.
22        Arrays.fill(array, 2, 6, -1);
23        System.out.print("After fill(): ");
24        display(array);
25
26        // Sort and display the array.
27        Arrays.sort(array);
28        System.out.print("After sorting again: ");
29        display(array);
30
31        // Binary search for -9.
32        System.out.print("The value -9 is at location ");
33        int index =
34            Arrays.binarySearch(array, -9);
```

ساختمان داده ها و پکیج `java.util`

```
35
36     System.out.println(index);
37 }
38
39 static void display(int array[]) {
40     for (int i=0; i<array.length; i++)
41         System.out.print(i + " ");
42
43     System.out.println();
44 }
45 }
```

کد ۱۴-۷

خروجی اجرای برنامه فوق بصورت زیر است.

```
Original contents: 0 1 2 3 4 5 6 7 8 9
Sorted: 0 1 2 3 4 5 6 7 8 9
After fill(): 0 1 2 3 4 5 6 7 8 9
After sorting again: 0 1 2 3 4 5 6 7 8 9
The value -9 is at location 2
```

کلاسها و اینترفیس های قدیمی در پکیج `java.util`

همانگونه که قبلا گفته شد، `Collection` در نسخه های اولیه `java.util` وجود نداشت در عوض کلاس های دیگری در `java.util` تعریف شده بودند که در نسخه های فعلی `java.util` همچنان وجود دارند بعد از اضافه شدن `Collection` به `java.util`، ساختار این کلاس ها به گونه ای نیز تغییر کرده است تا تمام کلاس ها و اینترفیس های `java.util` یک ساختار پیروی کنند. کلاس های قدیمی که در `util` تعریف شده اند در جدول زیر نشان داده شده اند.

Dictionary	Hashtable	Properties	Stack	Vector
------------	-----------	------------	-------	--------

علاوه بر این کلاس ها، یک اینترفیس قدیمی نیز وجود دارد که در استفاده از این کلاس ها کاربرد دارد.

اینترفیس `Enumeration`

این اینترفیس حاوی متدهایی است که با استفاده از آنها می توانید عناصر موجود در `Collection` را جستجو کنید در حال حاضر `Iterator` جایگزین این اینترفیس شده است اگرچه این اینترفیس از رده خارج شده است ولی از آنجائیکه برخی کلاس های قدیمی (مثل `Vector` و `Properties`) از این اینترفیس استفاده می کنند، همچنان در `java.util` وجود دارد. در این اینترفیس دو متده است:

در صورتیکه که فراخوانی متدهای nextElement() و hasMoreElements() تعریف شده اند شکل این دو متده به صورت زیر است.

```
boolean hasMoreElements()
Object nextElement()
```

در صورتیکه که فراخوانی متدهای nextElement() و hasMoreElements() تعریف شده اند شکل این دو متده به صورت زیر است.

کلاس Vector

کلاس Vector شبیه کلاس ArrayList است که با استفاده از آن می توان آرایه ای با طول متغیر را شبیه سازی نمود در نسخه های اولیه Collection که java.util در آن قرار نداشتند این کلاس وجود داشت اما با ارایه Collection ها طراحی آن تغییر کرد و منطبق بر Collection ها گردید. سازنده های کلاس Vector به قرار زیر است:

```
Vector()
Vector(int size)
Vector(int size, int incr)
Vector(Collection c)
```

مشخص کننده تعداد افزایش ظرفیت incr در هنگام پر شدن آن است. در علاوه بر متدهایی که در List تعریف شده اند چندین متدهای قدیمی دیگر نیز دارد که در جدول زیر فهرست شده اند.

شرح	متده
آبجکت element را به Vector اضافه می کند.	void addElement(Object element)
ظرفیت Vector را برمی گرداند.	int capacity()
یک کپی از Vector برمی گرداند.	Object clone()
در صورتیکه Vector حاوی عنصر element باشد حاصل فراخوانی این متده true و در غیر این صورت false خواهد بود.	boolean contains(Object element)
عنصر موجود در Vector را درون آرایه array کپی می کند.	void copyInto(Object array[])

ساختمان داده ها و پکیج `java.util`

آبجکتی را که در موقعیت <code>index</code> قرار دارد برمی گردد.	<code>Object elementAt(int index)</code>
یک آبجکت <code>Enumeration</code> حاوی تمام عناصر موجود در <code>Vector</code> را برمی گردد.	<code>Enumeration elements()</code>
ظرفیت <code>Vector</code> را به اندازه <code>size</code> تغییر می دهد.	<code>void ensureCapacity(int size)</code>
اولین عنصر درون <code>Vector</code> را برمی گردد.	<code>Object firstElement()</code>
موقعیت آبجکت <code>element</code> را که در قرار دارد برمی گردداند در صورتیکه آبجکت <code>Vector</code> در <code>element</code> وجود نداشته باشد مقدار برگشتی <code>1 - خواهد بود.</code>	<code>int indexOf(Object element)</code>
موقعیت اولین آبجکت <code>element</code> را که در موقعیت <code>start</code> یا بعد از آن قرار دارد را برمی گردداند در صورتیکه آبجکت <code>element</code> در <code>Vector</code> وجود نداشته باشد مقدار برگشتی <code>1 - خواهد بود.</code>	<code>int indexOf(Object element, int start)</code>
آبجکت <code>index</code> را در موقعیت <code>element</code> به <code>Vector</code> اضافه می کند.	<code>void insertElementAt(Object element, int index)</code>
در صورتیکه <code>Vector</code> خالی باشد مقدار برگشتی <code>true</code> مت د مقدار <code>false</code> خواهد بود و در غیر این صورت <code>false</code> خواهد بود.	<code>boolean isEmpty()</code>
آخرین عنصر درون <code>Vector</code> را برمی گردد.	<code>Object lastElement()</code>
موقعیت آخرین عنصر <code>element</code> درون <code>Vector</code> را برمی گردداند در صورتیکه این عنصر <code>Vector</code> در <code>element</code> موجود نباشد مقدار برگشتی آن <code>1 - خواهد بود.</code>	<code>int lastIndexOf(Object element)</code>
موقعیت آخرین عنصر <code>element</code> را قبل از <code>index</code> قرار دارد را برمی گردداند در صورتیکه این عنصر <code>Vector</code> در <code>element</code> موجود نباشد مقدار برگشتی آن <code>1 - خواهد بود.</code>	<code>int lastIndexOf(Object element, int index)</code>
تمام عناصر را از <code>Vector</code> حذف می کند پس از فراخوانی این مت د اندازه <code>Vector</code> صفر خواهد	<code>void removeAllElements()</code>

بود.	
عنصر element را از Vector حذف می کند در صورتیکه عنصر در Vector وجود داشته باشد و حذف کردن با موفقیت به پایان برسد و در غیراینصورت false بر می گرداند.	boolean removeElement(Object element)
عنصری را که در موقعیت index قرار دارد از Vector حذف می کند.	void removeElement(int index)
عنصر موجود در موقعیت index را با عنصر element جایگزین می کند.	void setElement(Object element, int index)
طول آرایه را تغییر می دهد به این معنی که اگر size کمتر از تعداد عناصر فعلی درون Vector باشد با حذف تعدادی از عناصر طول کاهش می یابد و در صورتیکه size بزرگتر از تعداد عناصر فعلی باشد به تعداد لازم آجکتهاي null به انتهای Vector اضافه می شود.	void setSize(int size)
تعداد عناصر فعلی Vector را بر می گرداند.	int size()
معادل رشته ای Vector را بر می گرداند	String toString()
طول Vector را دقیقا برابر تعداد عناصر فعلی آن قرار می دهد (در بسیاری از موارد طول Vector ممکن است با تعداد عناصر درون Vector یکسان نباشد و تعدادی فضای خالی نیز درون Vector وجود داشته باشد فراخوانی این متدهای فضاهای خالی را از Vector حذف می کند)	void trimToSize()

جدول ۱۴-۱۶. متدهای کلاس Vector

از آنجائیکه کلاس Vector اینترفیس List را پیاده سازی کرده است می توانید همانند استفاده کنید مثلا می توانید با فراخوانی متدهای addElement() و elementAt() آجکت جدیدی به آن اضافه کنید و با استفاده از متدهای firstElement() و lastElement() استفاده کنید متدهای قرار دارد بدست آورید همچنین می توانید برای بدست آوردن نخستین و آخرین عنصر Vector از متدهای () و () استفاده کنید متدهای removeElementAt() و removeElement() برای حذف یک عنصر از Vector استفاده می

ساختمان داده ها و پکیج `java.util`

شود متدهای `lastIndexOf()` و `indexOf()` نیز برای بدست آوردن ایندکس یک آبجکت استفاده می شوند.

در مثال زیر از `Vector` برای ذخیره سازی عناصر داخل `Vector` استفاده شده است در این مثال استفاده از `Enumeration` نیز نشان داده شده است.

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 class VectorDemo {
6
7     public static void main(String args[]) {
8
9         // initial size is 3, increment is 2
10        Vector v = new Vector(3, 2);
11
12        System.out.println("Initial size: " + v.size());
13        System.out.println("Initial capacity: " +
14                           v.capacity());
15
16        v.addElement(new Integer(1));
17        v.addElement(new Integer(2));
18        v.addElement(new Integer(3));
19        v.addElement(new Integer(4));
20
21        System.out.println("Capacity after four" +
22                           " additions: " + v.capacity());
23
24        v.addElement(new Integer(5));
25        System.out.println("Current capacity: " +
26                           v.capacity());
27        v.addElement(new Integer(6));
28        v.addElement(new Integer(7));
29
30        System.out.println("Current capacity: " +
31                           v.capacity());
32        v.addElement(new Integer(9));
33        v.addElement(new Integer(10));
```

```

34
35     System.out.println("Current capacity: " +
36             v.capacity());
37
38     v.addElement(new Integer(11));
39     v.addElement(new Integer(12));
40
41
42     System.out.println("First element: " +
43             v.firstElement());
44     System.out.println("Last element: " +
45             v.lastElement());
46
47     if (v.contains(new Integer(3)))
48         System.out.println("Vector contains 3.");
49
50     // Enumerate the elements in the vector.
51     Enumeration vEnum = v.elements();
52
53     System.out.println("\nElements in vector:");
54     while (vEnum.hasMoreElements())
55         System.out.print(vEnum.nextElement() + " ");
56     System.out.println();
57 }
58 }
```

کد

نتیجه اجرای برنامه به صورت زیر خواهد بود.

```

Initial size: 0
Initial capacity: 3
Capacity after four additions: 5
Current capacity: 5
Current capacity: 7
Current capacity: 9
First element: 1
Last element: 12
Vector contains 3.

Elements in vector:
1 2 3 4 5 6 7 9 10 11 12
```

ساختمان داده ها و پکیج `java.util`

به جای استفاده از `Enumeration` می توانید از `Iterator` استفاده کنید در اینصورت قسمت مربوط به استخراج داده های `Vector` به صورت زیر تغییر خواهد کرد

```
// Use an iterator to display contents.  
Iterator vItr = v.iterator();  
System.out.println("\nElements in vector:");  
while (vItr.hasNext())  
    System.out.print(vItr.next() + " ");  
System.out.println();
```

Stack کلاس

این کلاس از کلاس `Vector` مشتق شده است و با استفاده از آن می توانید یک پشته `LIFO` را پیاده سازی کنید در `Stack`، آخرین داده ای که وارد می شود اولین داده ای خواهد بود که خارج می شود. علاوه بر تمامی متدهایی که در کلاس `Vector` تعریف شده اند حاوی متدهای دیگری نیز هست که در جدول زیر ملاحظه می کنید.

شرح	متد
در صورتیکه پشته خالی باشد، مقدار <code>true</code> و در غیراینصورت مقدار <code>false</code> بر می گرداند.	<code>boolean empty()</code>
عنصر بالای پشته را بر می گرداند اما آن را حذف نمی کند.	<code>Object peek()</code>
عنصر بالای پشته را بر می گرداند و آنرا از پشته حذف می کند.	<code>Object pop()</code>
این متد آبجکت <code>obj</code> را به پشته اضافه می کند و آنرا نیز بر می گرداند.	<code>Object push(Object obj)</code>
آبجکت <code>element</code> را در پشته جستجو می کند و موقعیت آن نسبت به بالای پشته را بر می گرداند در صورتیکه عنصر <code>element</code> در پشته یافته نشود <code>-1</code> بر می گرداند.	<code>int search(Object element)</code>

جدول ۱۶-۱۷. متدهای `Stack`

برای افزودن یک داده به `Stack` از متد `push()` استفاده می شود و برای استخراج یک داده که در بالای `Stack` قرار دارد از متد `pop()` استفاده می شود در صورتیکه `Stack` خالی باشد فراخوانی متد `pop()` باعث بروز خطای `EmptyStackException` خواهد شد. در صورتیکه بخواهید بدون حذف یک عنصر از `Stack`، داده ای را که در بالای `Stack` قرار دارد را بدست آورید می توانید از متد `peek()`

استفاده کنید. در صورتیکه Stack خالی باشد یعنی حاوی هیچ داده ای نباشد فراخوانی متده empty() مقدار true برموی گرداند.

با استفاده از متده search() نیز می توانید وجود یک آبجکت را درون Stack بررسی کنید. مثال زیر استفاده از یک Stack را نشان می دهد که با استفاده از آن داده هایی در بالای آن اضافه می شوند و سپس با استفاده از متده pop() از آن استخراج می شوند.

```

1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class StackDemo {
6
7     static void showpush(Stack st, int a) {
8         st.push(new Integer(a));
9         System.out.println("push(" + a + ")");
10        System.out.println("stack: " + st);
11    }
12
13    static void showpop(Stack st) {
14        System.out.print("pop -> ");
15        Integer a = (Integer) st.pop();
16        System.out.println(a);
17        System.out.println("stack: " + st);
18    }
19
20    public static void main(String args[]) {
21        Stack st = new Stack();
22
23        System.out.println("stack: " + st);
24        showpush(st, 42);
25        showpush(st, 66);
26        showpush(st, 99);
27        showpop(st);
28        showpop(st);
29        showpop(st);
30
31        try {
32            showpop(st);

```

ساختمان داده ها و پکیج java.util

```
33         } catch (EmptyStackException e) {  
34             System.out.println("empty stack");  
35         }  
36     }  
37 }
```

کد ۱۴-۹

نتیجه اجرای این برنامه در زیر نشان داده شده است:

```
stack: []  
push(42)  
stack: [42]  
push(66)  
stack: [42, 66]  
push(99)  
stack: [42, 66, 99]  
pop -> 99  
stack: [42, 66]  
pop -> 66  
stack: [42]  
pop -> 42  
stack: []  
pop -> empty stack
```

Hashtable کلاس

این کلاس از جمله کلاس‌های قدیمی `java.util.Collection` به شمار می‌رود اما با ظهور `HashMap` طراحی آن مورد بازنگری قرار گرفت و اینترفیس `Map` را پیاده سازی نمود این کلاس مشابه `HashMap` است. به هر داده ای که داخل `Hashtable` قرار داده می‌شود یک کلید اختصاص داده می‌شود این کلاس همانند `HashMap` هر داده را همراه با کلید آن در یک جدول نگهداری می‌کند از روی کلید هر داده `hash code` ساخته شده و سپس به عنوان ایندکس آن داده در نظر گرفته می‌شود.

سازنده‌های `Hashtable` در زیر نشان داده شده اند:

```
Hashtable()  
Hashtable(int size)  
Hashtable(int size, float fillRatio)  
HashMap(Map m)
```

همانند قبل، عددی اعشاری بین ۰.۰ و ۱.۰ است و مشخص می‌کند که چه مقدار از اندازه `fillRatio` باید پر شود تا طول آن افزایش یابد. `HashMap` این کلاس علاوه بر متدهایی که در اینترفیس `Map` تعریف شده اند دارای تعدادی متod قدیمی نیز می‌باشد که در جدول زیر نشان داده شده اند:

شرح	متد
عناصر موجود در Hashtable را پاک کرده و آنرا خالی می کند.	<code>void clear()</code>
یک کپی از روی Hashtable فعلی می سازد.	<code>Object clone()</code>
در صورتیکه آبجکت value درون Hashtable وجود داشته باشد این متاد مقدار true برمی گردداند.	<code>boolean contains(Object value)</code>
چنانچه کلید key در Hashtable وجود داشته باشد حاصل فراخوانی این متاد true و درغیراینصورت false خواهد بود.	<code>boolean containsKey(Object key)</code>
در صورتیکه آبجکت value در Hashtable وجود داشته باشد حاصل فراخوانی این متاد true خواهد بود.	<code>boolean containsValue(Object value)</code>
یک آبجکت Enumeration حاوی تمام عناصر موجود در Hashtable برمی گردداند.	<code>Enumeration elements()</code>
آبجکتی را که با کلید key مرتبط است را بر می گردداند.	<code>Object get(Object key)</code>
در صورتیکه Hashtable حاوی هیچ عنصری نباشد حاصل فراخوانی این متاد true خواهد بود.	<code>boolean isEmpty()</code>
یک آبجکت Enumeration حاوی تمام کلیدهای Hashtable برمی گردداند.	<code>Enumeration keys()</code>
آبجکت value را با کلید key در Hashtable اضافه می کند در صورتیکه کلید key قبلا در Hashtable وجود داشته باشد مقدار قبلی مرتبط با آن کلید را برمی گردداند و در صورتیکه قبلا وجود نداشته باشد null برمی گردداند.	<code>Object put(Object key, Object value)</code>
اندازه hash code را افزایش داده و تمام کلیدهای آنرا از نو می سازد.	<code>void rehash()</code>
کلید key و مقدار مرتبط با آنرا از Hashtable حذف می کند در صورتیکه کلید key در Hashtable موجود نباشد مقدار null برمی گردداند و در غیراینصورت مقدار مرتبط با آن کلید را	<code>Object remove(Object key)</code>

ساختمان داده ها و پکیج `java.util`

برمی گرداند.	
تعداد عناصر موجود در <code>Hashtable</code> را برمی گرداند.	<code>int size()</code>
معادل رشته ای <code>Hashtable</code> را برمی گرداند.	<code>String toString()</code>

جدول ۱۴-۱۱. متدهای کلاس `Hashtable`

در زیر مثال حسابهای بانکی که پیش از نشان داده شده است با استفاده از `Hashtable` پیاده سازی شده است.

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class HTDemo {
6     public static void main(String args[]) {
7         Hashtable balance =
8             new Hashtable();
9
10        Enumeration names;
11        String str;
12        double bal;
13
14        balance.put("John Doe", new Double(3434.34));
15        balance.put("Tom Smith", new Double(123.22));
16        balance.put("Jane Baker", new Double(1378.00));
17        balance.put("Tod Hall", new Double(99.22));
18        balance.put("Ralph Smith", new Double(-19.08));
19
20        // Show all balances in hashtable.
21        names = balance.keys();
22        while (names.hasMoreElements()) {
23            str = (String) names.nextElement();
24            System.out.println(str + ": " +
25                balance.get(str));
26        }
27
28        System.out.println();
29
30        // Deposit 1,000 into John Doe's account.
```

```

31         bal = ((Double) balance.get(
32             "John Doe")).doubleValue();
33
34         balance.put("John Doe", new Double(bal + 1000));
35         System.out.println("John Doe's new balance: " +
36             balance.get("John Doe"));
37     }
38 }
```

کد ۱۴-۱۰

خروجی اجرای برنامه به صورت زیر می باشد.

```

Tod Hall: 99.22
Ralph Smith: -19.08
John Doe: 3434.34
Jane Baker: 1378.0
Tom Smith: 123.22

John Doe's new balance: 4434.34
```

برای خواندن اطلاعات داخل Hashtable می توانید به جای استفاده از متدهای keys() و keySet() که یک Enumeration از کلیدهای Hashtable برمیگرداند از متدهای entrySet() و get() استفاده کنید برنامه زیر نسخه تغییریافته مثال فوق است که از آن به جای متدهای keys() و keySet() استفاده شده است.

```

1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class HTDemo2 {
6
7     public static void main(String args[]) {
8         Hashtable balance =
9             new Hashtable();
10
11     String str;
12     double bal;
13
14     balance.put("John Doe", new Double(3434.34));
```

```
15      balance.put("Tom Smith", new Double(123.22));
16      balance.put("Jane Baker", new Double(1378.00));
17      balance.put("Tod Hall", new Double(99.22));
18      balance.put("Ralph Smith", new Double(-19.08));
19
20      // Show all balances in hashtable.
21      // First, get a set view of the keys.
22      Set set = balance.keySet();
23
24      // Get an iterator.
25      Iterator itr = set.iterator();
26      while (itr.hasNext()) {
27          str = (String) itr.next();
28          System.out.println(str + ": " +
29                          balance.get(str));
30      }
31
32      System.out.println();
33
34      // Deposit 1,000 into John Doe's account.
35      bal = ((Double) balance.get(
36                  "John Doe")).doubleValue();
37      balance.put("John Doe", new Double(bal + 1000));
38      System.out.println("John Doe's new balance: " +
39                          balance.get("John Doe"));
40  }
41 }
```

کد ۱۴-۱۱

Properties کلاس

این کلاس از کلاس `Hashtable` مشتق شده است از این کلاس برای نگهداری داده های `String` استفاده می شود که کلید آن نیز یک رشته `String` است این کلاس یکی از کلاسهای پر کاربرد جاوا محسوب می شود به عنوان مثال می توانید با فراخوانی `System.getProperties()` آبجکتی از کلاس `Properties` بدست آورید که حاوی مقادیر محیطی سیستم است. این کلاس دارای دو سازنده به صورت زیر است:

```
Properties()
Properties(Properties props)
```

این کلاس علاوه بر متدهایی که در کلاس `Hashtable` تعریف شده اند حاوی متدهای دیگری نیز هست که در جدول زیر نشان داده شده اند.

شرح	متند
مقدار مرتبط با کلید key را برمی گرداند اگر کلید key در Properties وجود نداشته باشد null برگردانده می شود.	<code>String getProperty(String key)</code>
مقدار مرتبط با کلید key را برمی گرداند اگر کلید key در Properties وجود نداشته باشد defaultValue برگردانده می شود.	<code>String getProperty(String key, String defaultValue)</code>
Properties را به outStream ارسال می کند.	<code>void list(PrintWriter outStream)</code>
Properties را به outWriter ارسال می کند.	<code>void list(PrintStream outWriter)</code>
داده هایی را به صورت زوجهای «کلید/مقدار» از یک <code>InputStream</code> دریافت می کند.	<code>void load(InputStream streamIn) throws IOException</code>
داده هایی را به صورت زوجهای «کلید/مقدار» از یک سند XML که به صورت XML درآمده دریافت می کند.	<code>void loadFromXML(InputStream streamIn) throws IOException, InvalidPropertiesFormatException</code>
یک آبجکت <code>Enumeration</code> که حاوی تمام کلیدهای موجود در Properties است را برمی گرداند.	<code>Enumeration propertyNames()</code>
مقدار value را با کلید key به Properties اضافه می کند اگر چنین کلیدی قبلا وجود داشته باشد مقدار پیشین مرتبط با کلید را برمی گرداند و در غیراینصورت مقدار null برمی گرداند.	<code>Object setProperty(String key, String value)</code>
داده های درون Properties داخل یک OutputStream نوشته می شوند قبل از داده ها، description نیز به صورت یک توضیح اضافه می شود.	<code>void store(OutputStream streamOut, String description) throws IOException</code>

ساختمان داده ها و پکیج `java.util`

داده های درون <code>Properties</code> داخل یک فایل (که معرف یک <code>OutputStream</code> است) نوشته می شوند قبل از داده ها، نیز به صورت یک توضیح <code>description</code> اضافه می شود.	<code>void storeToXML(OutputStream outStream, String description) throws IOException</code>
داده های درون <code>Properties</code> داخل یک فایل (که معرف یک <code>OutputStream</code> است) نوشته می شوند قبل از داده ها، نیز به صورت یک توضیح <code>description</code> اضافه می شود در این متدهد <code>encoding</code> داده ها نیز توسط <code>enc</code> مشخص می شود.	<code>void storeToXML(OutputStream streamOut, String description, String enc)</code>

جدول ۱۴-۱۹. متدهای کلاس `Properties`

یکی از قابلیتهای این کلاس این است که می توانید برای یک کلید، یک مقدار پیش فرض تعريف کنید تا در صورتیکه آن کلید مقداردهی نشد قبلا با مقدار پیش فرض مقدار دهی شده باشد برای اینکار کافی است در متدهد `getProperties()` که با استفاده از آن مقدار یک کلید خوانده می شود مقدار پیش فرض را نیز مشخص کنید

```
getProperty("name", "default value")
```

اگر مقدار مرتبط با "name" پیدا نشود در اینصورت مقدار "default value" برگردانده خواهد شد. همچنین در هنگام ساخت آبجکت از این کلاس می توانید یک آبجکت `Properties` که حاوی مقادیر پیش فرض تعدادی کلید است به سازنده این کلاس ارسال کنید در این صورت اگر مثلا متدهد `getProperty("name")` را فراخوانی کنید و مقداری برای این کلید در `Properties` وجود نداشته باشد مقدار پیش فرض آن برگردانده خواهد شد. مثال زیر استفاده از این کلاس را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class PropDemo {
6     public static void main(String args[]) {
7         Properties capitals = new Properties();
8
9         capitals.put("Illinois", "Springfield");
```

```

10         capitals.put("Missouri", "Jefferson City");
11         capitals.put("Washington", "Olympia");
12         capitals.put("California", "Sacramento");
13         capitals.put("Indiana", "Indianapolis");
14
15         // Get a set-view of the keys.
16         Set states = capitals.keySet();
17
18         // Show all of the states and capitals.
19         Iterator itr = states.iterator();
20         while (itr.hasNext()) {
21             String name = (String) itr.next();
22             System.out.println("The capital of " +
23                     name + " is " +
24                     capitals.getProperty((String) name)
25                     + ".");
26
27         }
28
29         System.out.println();
30
31         // Look for state not in list -- specify default.
32         String str = capitals.getProperty("Florida",
33                         "Not Found");
34         System.out.println("The capital of Florida is "
35                         + str + ".");
36     }
37 }
```

کد

خروجی برنامه فوق در زیر نشان داده شده است.

```
The capital of Missouri is Jefferson City.
The capital of Illinois is Springfield.
The capital of Indiana is Indianapolis.
The capital of California is Sacramento.
The capital of Washington is Olympia.

The capital of Florida is Not Found.
```

ساختمان داده ها و پکیج `java.util`

چون Florida در فهرست موجود نیست از مقدار پیش فرض استفاده می شود. در برنامه فوق برای مشخص کردن مقدار پیش فرض از متده استفاده می شود. در برنامه فوق با یک مقدار پیش فرض استفاده شد اما همانطور که گفته شد می توان از یک آبجکت Properties که در آن مقادیر پیش فرض گنجانده شده است نیز استفاده نمود برنامه زیر مثال فوق را با استفاده از یک آبجکت Properties که حاوی مقادیر پیش فرض است را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.*;
4
5 public class PropDemoDef {
6
7     public static void main(String args[]) {
8         Properties defList = new Properties();
9         defList.put("Florida", "Tallahassee");
10        defList.put("Wisconsin", "Madison");
11
12        Properties capitals = new Properties(defList);
13
14        capitals.put("Illinois", "Springfield");
15        capitals.put("Missouri", "Jefferson City");
16        capitals.put("Washington", "Olympia");
17        capitals.put("California", "Sacramento");
18        capitals.put("Indiana", "Indianapolis");
19
20        // Get a set-view of the keys.
21        Set states = capitals.keySet();
22
23        // Show all of the states and capitals.
24        Iterator itr = states.iterator();
25        while (itr.hasNext()) {
26            String name = (String) itr.next();
27            System.out.println("The capital of " +
28                               name + " is " +
29                               capitals.getProperty((String) name)
30                               + ".");
31        }
32    }
}
```

```

33         System.out.println();
34
35     // Florida will now be found in the default list.
36     String str = capitals.getProperty("Florida");
37     System.out.println("The capital of Florida is "
38             + str + ".");
39 }
40 }
```

کد ۱۴-۱۳**استفاده از متدهای `store()` و `load()`**

یکی از قابلیتهای کلاس Properties این است که می توان مقادیر و کلیدهای آنها را از یک فایل متنی خواند یا در یک فایل متنی نوشت در بسیاری از برنامه های J2EE پیکربندی برنامه را درون یک فایل متنی properties می گنجانند سپس در زمان اجرای برنامه، اطلاعات فایل متنی را با استفاده از این کلاس خوانده و درون برنامه استفاده می کنند.

مثال زیر استفاده از این قابلیت کلاس Properties را برای خواندن اطلاعات دفتر تلفن افراد را از یک فایل متنی نشان می دهد این اطلاعات در یک فایل متنی با نام phonebook.dat ذخیره و بازیابی می شوند.

```

1 package ir.atlassoft.javase.chapter14;
2
3 /* A simple telephone number database that uses
4    a property list. */
5
6 import java.io.*;
7 import java.util.*;
8
9 public class PhoneBookBook {
10     public static void main(String args[])
11         throws IOException {
12         Properties ht = new Properties();
13         BufferedReader br = new BufferedReader(new
14             InputStreamReader(System.in));
15
16         String name, number;
17         FileInputStream fin = null;
18         boolean changed = false;
```

ساختمان داده ها و پکیج java.util

```
19
20      // Try to open phonebook.dat file.
21      try {
22          fin = new FileInputStream("phonebook.dat");
23      } catch (FileNotFoundException e) {
24          // ignore missing file
25      }
26
27      /* If phonebook file already exists,
28         load existing telephone numbers. */
29      try {
30          if (fin != null) {
31              ht.load(fin);
32              fin.close();
33          }
34      } catch (IOException e) {
35          System.out.println("Error reading file.");
36      }
37
38      // Let user enter new names and numbers.
39      do {
40          System.out.println("Enter new name" +
41                             " ('quit' to stop): ");
42          name = br.readLine();
43          if (name.equals("quit")) continue;
44
45          System.out.println("Enter number: ");
46          number = br.readLine();
47
48          ht.put(name, number);
49          changed = true;
50      } while (!name.equals("quit"));
51
52      // If phone book data has changed, save it.
53      if (changed) {
54          FileOutputStream fout = new
55                         FileOutputStream("phonebook.dat");
56
57          ht.store(fout, "Telephone Book");
58          fout.close();
```

```

59         }
60
61     // Look up numbers given a name.
62     do {
63         System.out.println("Enter name to find" +
64             " ('quit' to quit): ");
65         name = br.readLine();
66         if (name.equals("quit")) continue;
67
68         number = (String) ht.get(name);
69         System.out.println(number);
70     } while (!name.equals("quit"));
71 }
72 }
```

کد ۱۴-۱۴**کلاس Date**

از این کلاس برای نشان دادن تاریخ و زمان سیستم استفاده می شود دو سازنده این کلاس که هنوز از ردۀ خارج نشده اند (بسیاری از سازنده های این کلاس از ردۀ خارج شده اند) به صورت زیر می باشند:

Date()
Date(**long** millisec)

سازنده اول آبجکتی که معرف زمان و تاریخ فعلی سیستم است را مشخص می کند اما سازنده دوم یک عدد long دریافت می کند که برابر تعداد میلی ثانیه های سپری شده از نیمه شب اول ژانویه ۱۹۷۰ است برخی از متدهای این کلاس در زیر فهرست شده اند:

شرح	متدها
اگر date معرف تاریخی بعد از تاریخ آبجکتی باشد که متدها آن فراخوانی شده مقدار true و در غیر اینصورت false برمو گرداند.	boolean after(Date date)
اگر date معرف تاریخی قبل از تاریخ آبجکتی باشد که متدها آن فراخوانی شده مقدار true و در غیر اینصورت false برمو گرداند.	boolean before(Date date)
یک آبجکت جدید از روی آبجکت فعلی کپی می کند و آنرا برمو	Object clone()

ساختمان داده ها و پکیج `java.util`

گرداند.	
تاریخ آبجکتی را که متد <code>compareTo()</code> از آن فراخوانی شده را با تاریخ <code>date</code> مقایسه می کند در صورتیکه هر دو معرف یک تاریخ باشند حاصل فراخوانی این متد ۰ خواهد بود در صورتیکه تاریخ آبجکتی که متد <code>compareTo()</code> آن فراخوانی شده قبل از <code>date</code> باشد مقدار برگشتی منفی و در غیراینصورت مقدار برگشتی مثبت خواهد بود.	<code>int compareTo(Date date)</code>
در صورتیکه تاریخ و زمان آبجکتی که متد <code>equals()</code> آن فراخوانی شده با تاریخ و زمان <code>date</code> یکسان باشد مقدار <code>true</code> و در غیراینصورت مقدار <code>false</code> برمی گرداند.	<code>boolean equals(Object date)</code>
تعداد میلی ثانیه های تاریخ و زمان آبجکت، پس از نیمه شب اول ژانویه ۱۹۷۰ را برمی گرداند.	<code>long getTime()</code>
آبجکت را برمی گرداند.	<code>int hashCode()</code>
تاریخ و زمان آبجکت را تغییر می دهد مقدار جدید آبجکت برابر <code>time</code> میلی ثانیه پس از نیمه شب اول ژانویه سال ۱۹۷۰ است.	<code>void setTime(long time)</code>
آبجکت <code>Date</code> را به یک رشته تبدیل و آنرا برمی گرداند.	<code>String toString()</code>

جدول ۱۴-۲۰. متدهای کلاس Date

همانطور که در جدول ۱۴-۱۱ ملاحظه می کنید این کلاس اطلاعات مناسبی در مورد زمان یا تاریخ در اختیار شما قرار نمی دهد برای اینکه اطلاعات جامع تری درباره تاریخ و زمان سیستم بدست آورید از کلاس `Calendar` استفاده کنید. در کلاس `Date` متد `toString()` وجود دارد که زمان و تاریخ آبجکت را به صورت یک رشته تولید و برمی گرداند استفاده از این متد در مثال زیر نشان داده شده است.

```
1 package ir.atlassoft.javase.chapter14;  
2  
3 import java.util.Date;  
4  
5 public class DateDemo {  
6  
7     public static void main(String args[]) {  
8  
9         // Instantiate a Date object  
10        Date date = new Date();  
11  
12        // display time and date using toString()  
13    }  
14}
```

```

13         System.out.println(date);
14
15     // Display number of milliseconds since midnight,
16     // January 1, 1970 GMT
17     long msec = date.getTime();
18     System.out.println("Milliseconds since " +
19                     "Jan. 1, 1970 GMT = " + msec);
20 }
21 }
```

کد ۱۶-۱۷

نتیجه اجرای برنامه فوق بصورت زیر خواهد بود.

```
Fri Oct 06 21:59:42 IRST 2006
Milliseconds since Jan. 1, 1970 GMT = 1160159382364
```

Calendar کلاس

کلاس Calendar شبیه Date کلاس است یعنی برای کار کردن با زمان و تاریخ استفاده می شود اما این کلاس برخلاف کلاس Date، یک کلاس abstract است و شما نمی توانید مستقیماً از روی آن آبجکت بسازید. از آنجاییکه در زبانهای مختلف (مثلا فارسی، عربی، آلمانی...) از تاریخ متفاوتی استفاده می کنند و آنرا نیز با فرمات متفاوتی نشان می دهند از کلاس Calendar می توانید برای ارایه زمان و تاریخ در زبان مورد نظرتان استفاده کنید.

برای ساختن یک آبجکت از این کلاس، باید متدها getInstace() از کلاس Calendar را فراخوانی کنید این آبجکت مشخص کننده زمان فعلی سیستم است. برخی متدهای متداول این کلاس در جدول زیر فهرست شده اند:

شرح	متدها
<p>این متدها یک زمانی که آبجکت Calendar مشخص می کند مقداری را اضافه می کند. مشخص کننده یک جزء زمانی است این جزء ممکن است ساعت: Calendar.HOUR، دقیقه: Calendar.MINUTE، ثانیه: Calendar.SECOND یا ... باشد. val مشخص کننده مقداری است که قرار است به جزء زمانی که</p>	abstract void add(int which, int val)

ساختمان داده ها و پکیج **java.util**

توسط <code>which</code> مشخص شده است اضافه شود. برای تفیریق کردن یک مقدار منفی را باید وارد نمود.	
اگر تاریخ آبجکتی که متده است <code>after()</code> آن فراخوانی شده است <u>بعد</u> از تاریخ مشخص شده در <code>calendarObj</code> باشد حاصل فراخوانی این متده است <code>false</code> و در غیراینصورت <code>true</code> خواهد بود.	<code>boolean after(Object calendarObj)</code>
اگر تاریخ آبجکتی که متده است <code>before()</code> آن فراخوانی شده است <u>قبل</u> از تاریخ مشخص شده در <code>calendarObj</code> باشد حاصل فراخوانی این متده است <code>false</code> و در غیراینصورت <code>true</code> خواهد بود.	<code>boolean before(Object calendarObj)</code>
مقدار تمام اجزای زمان و تاریخ مشخص شده توسط آبجکت، را صفر می کند.	<code>final void clear()</code>
جزء زمانی که توسط <code>which</code> مشخص شده را صفر می کند.	<code>final void clear(int which)</code>
یک کپی از روی آبجکت می سازد.	<code>Object clone()</code>
اگر تاریخ مشخص شده توسط <code>calendarObj</code> با تاریخ و زمان آبجکتی که متده است <code>equals()</code> آن فراخوانی شده یکسان باشد فراخوانی این متده <code>true</code> برمی گرداند.	<code>boolean equals(Object calendarObj)</code>
مقدار جزء زمانی که توسط <code>calendarField</code> مشخص شده است را برمی گرداند.	<code>int get(int calendarField)</code>
آرایه ای از آبجکت های <code>Locale</code> را برمی گرداند که حاوی اطلاعات تقویمهای مختلف است.	<code>static Locale[] getAvailableLocales()</code>
یک آبجکت <code>Calendar</code> که معرف تاریخ و زمان <code>Locale</code> پیش فرض است را برمی گرداند.	<code>static Calendar getInstance()</code>
یک آبجکت <code>Calendar</code> که معرف تاریخ و زمان <code>locale</code> است را برمی گرداند.	<code>static Calendar getInstance(Locale locale)</code>
آبجکت <code>Date</code> که از نظر زمان و تاریخ معادل آبجکت <code>Calendar</code> فعلی است را برمی گرداند.	<code>final Date getTime()</code>
اگر جزء زمانی <code>which</code> دارای مقدار باشد فراخوانی این متده <code>true</code> برای می گرداند.	<code>final boolean isSet(int which)</code>
را به جزء زمانیکه توسط <code>val</code> مشخص	<code>void set(int which, int value)</code>

شده انتساب می دهد.	
سال، ماه و روز در یک ماه را در یک آبجکت Calendar مقداردهی می کند.	final void set(int year, int month, int dayOfMonth)
سال، ماه، روز در یک ماه، ساعت و دقیقه را در یک آبجکت Calendar مقداردهی می کند.	final void set(int year, int month, int dayOfMonth, int hours, int minutes)
سال، ماه، روز در یک ماه، ساعت، دقیقه و ثانیه را در یک آبجکت Calendar مقداردهی می کند.	final void set(int year, int month, int dayOfMonth, int hours, int minutes, int seconds)
اجزای تاریخ و زمانی که توسط آبجکت date مشخص شده را به اجزای تاریخ و زمانی آبجکت Calendar فعلی انتساب می دهد.	final void setTime(Date date)

جدول ۲۱-۱۴. متدهای کلاس Calendar

تعدادی مقادیر ثابت نیز در این کلاس تعریف شده اند که هنگام مقداردهی یا بدست آوردن یکی از مولفه های زمان کاربرد دارد لیست این ثابت ها در جدول ۱۴-۲۰ ملاحظه می کنید.

AM	FRIDAY	PM
AM_PM	HOUR	SATURDAY
APRIL	HOUR_OF_DAY	SECOND
AUGUST	JANUARY	SEPTEMBER
DATE	JULY	SUNDAY
DAY_OF_MONTH	JUNE	THURSDAY
DAY_OF_WEEK	MARCH	TUESDAY
DAY_OF_WEEK_IN_MONTH	MAY	UNDECIMBER
DAY_OF_YEAR	MILLISECOND	WEDNESDAY
DECEMBER	MINUTE	WEEK_OF_MONTH
DST_OFFSET	MONDAY	WEEK_OF_YEAR
EAR	MONTH	YEAR
FEBRUARY	NOVEMBER	ZONE_OFFSET
FIELD_COUNT	OCTOBER	

جدول ۲۰-۱۴. مقادیر ثابت کلاس Calendar

مثال زیر استفاده از این کلاس را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter14;
```

java.util ساختمان داده ها و پکیج

```
2
3 import java.util.Calendar;
4
5 public class CalendarDemo {
6
7     public static void main(String args[]) {
8
9         String months[] = {
10             "Jan", "Feb", "Mar", "Apr",
11             "May", "Jun", "Jul", "Aug",
12             "Sep", "Oct", "Nov", "Dec"};
13
14         // Create a calendar initialized with the
15         // current date and time in the default
16         // locale and timezone.
17         Calendar calendar = Calendar.getInstance();
18
19         // Display current time and date information.
20         System.out.print("Date: ");
21         System.out.print(months[calendar.
22             get(Calendar.MONTH)]);
23         System.out.print(" " +
24             calendar.get(Calendar.DATE) + " ");
25         System.out.println(calendar.get(Calendar.YEAR));
26
27         System.out.print("Time: ");
28         System.out.print(calendar.get(Calendar.HOUR) +
29             ":" );
30         System.out.print(calendar.get(Calendar.MINUTE) +
31             ":" );
32         System.out.println(calendar.get(
33             Calendar.SECOND));
34
35         // Set the time and date information
36         // and display it.
37         calendar.set(Calendar.HOUR, 10);
38         calendar.set(Calendar.MINUTE, 29);
39         calendar.set(Calendar.SECOND, 22);
40
41         System.out.print("Updated time: ");
```

```

42     System.out.print(calendar.get(Calendar.HOUR) +
43             ":" );
44     System.out.print(calendar.get(Calendar.MINUTE) +
45             ":" );
46     System.out.println(calendar.get(
47             Calendar.SECOND));
48 }
49 }
```

کد ۱۴-۱۵

خروجی این برنامه به صورت زیر خواهد بود.

```

Date: Oct 6 2006
Time: 10:0:57
Updated time: 10:29:22
```

در صورتیکه لازم است یک کلاس معرف یک تقویم منطقه‌ای یا شخصی داشته باشید باید کلاسی بنویسید که از این کلاس مشتق شده است و مشخصات تاریخی و زمانی آن منطقه را پیاده سازی کرده است به عنوان مثال کلاس GregorianCalendar که جزیی از `java.util.Date` است معرف تاریخ میلادی است.

کلاس Random

با استفاده از این کلاس می‌توانید اعداد تصادفی بسازید به اعدادی که توسط این کلاس تولید می‌شوند اعداد شبه تصادفی گفته می‌شود دلیل آن هم این است که اعداد تولید شده توسط این کلاس، یک سری عدد با توزیع یکنواخت هستند. سازنده‌های زیر برای این کلاس تعریف شده‌اند:

```

Random()
Random(long seed)
```

سازنده اول، در حقیقت تولید کننده‌ای است که از زمان فعلی سیستم به عنوان مقدار اولیه، یا `seed` استفاده می‌کند در سازنده دوم، امکان مشخص کردن دستی `seed` وجود دارد. متدهای این کلاس در جدول ۱۴-۲۲ شان داده شده‌اند.

شرح	متد
با فراخوانی این متد، هربار یک مقدار تصادفی <code>boolean</code> تولید شده و برگردانده می‌شود.	<code>boolean nextBoolean()</code>
با فراخوانی این متد، آرایه‌ای از بایت به صورت تصادفی تولیدشده و در <code>values</code> قرار می‌گیرد.	<code>void nextBytes(byte values[])</code>

ساختمان داده ها و پکیج `java.util`

فراخوانی این متده است، عدد تصادفی <code>double</code> بعدی را تولید کرده و برمی گرداند.	<code>double nextDouble()</code>
فراخوانی این متده است، عدد تصادفی <code>float</code> بعدی را تولید کرده و برمی گرداند.	<code>float nextFloat()</code>
فراخوانی این متده است، عدد تصادفی <code>Gauss</code> بعدی را تولید کرده و برمی گرداند.	<code>double nextGaussian()</code>
فراخوانی این متده است، عدد تصادفی <code>int</code> بعدی را تولید کرده و برمی گرداند.	<code>int nextInt()</code>
فراخوانی این متده است، عدد تصادفی <code>int</code> بعدی را در محدوده ۰ تا <code>n</code> تولید کرده و برمی گرداند.	<code>int nextInt(int n)</code>
فراخوانی این متده است، عدد تصادفی <code>long</code> بعدی را تولید کرده و برمی گرداند.	<code>long nextLong()</code>
برای تعیین کنید که این کار نقطه شروع برای تولید اعداد تصادفی را مشخص کرده اید اگر از همان <code>seed</code> استفاده کنید در آن صورت همان اعداد را بدست خواهید آورد اگر می خواهید سریهای مختلفی داشته باشید، از مقادیر متفاوتی به عنوان <code>seed</code> استفاده کنید.	<code>void setSeed(long newSeed)</code>

جدول ۱۴-۲۲. متدهای کلاس `Random`

اگر `seed` را تعیین کنید با این کار نقطه شروع برای تولید اعداد تصادفی را مشخص کرده اید اگر از همان `seed` برای ایجاد یک آبجکت `Random` استفاده کنید در آن صورت همان اعداد را بدست خواهید آورد اگر همانطور که در جدول فوق ملاحظه می کنید با استفاده از متدهای این کلاس می توانید انواع مختلف از اعداد تصادفی را بدست آورید به عنوان مثال `nextBoolean()` یک مقدار بولی تصادفی تولید می کند و `nextLong()` یک عدد `long` تصادفی تولید می کند `nextGaussian()` نیز برای اعداد تصادفی `double` استفاده می شود که انحراف از معیار استاندارد آنان نسبت به ۰،۰۱ است. مثال زیر استفاده از این کلاس را برای تولید اعداد تصادفی نشان می دهد در این مثال از متده `nextGaussian()` استفاده شده تا بوسیله آن ۱۰۰ عدد تصادفی محاسبه شود سپس میانگین اعداد بدست آمده محاسبه شده است

```
1 package ir.atlassoft.javase.chapter14;
2
3 import java.util.Random;
4
5 public class RandDemo {
6
7     public static void main(String args[]) {
```

```

8
9     Random r = new Random();
10    double val;
11    double sum = 0;
12    int bell[] = new int[10];
13
14    for (int i = 0; i < 100; i++) {
15        val = r.nextGaussian();
16        sum += val;
17        double t = -2;
18        for (int x = 0; x < 10; x++, t += 0.5)
19            if (val < t) {
20                bell[x]++;
21                break;
22            }
23    }
24    System.out.println("Average of values: " +
25                        (sum / 100));
26
27    // display bell curve, sideways
28    for (int i = 0; i < 10; i++) {
29        for (int x = bell[i]; x > 0; x--)
30            System.out.print("*");
31        System.out.println();
32    }
33}
34}

```

کد ۱۶

خروجی اجرای این برنامه به صورت زیر خواهد بود

```
Average of values: -0.1691477320134351
***  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
**  
*
```

خلاصه

پکیج `java.util` حاوی بسیاری از کلاس‌های پرکاربرد و عمومی است کلاس‌های ساختمان داده‌ها و برخی کلاس‌های ممکنی از جمله این کلاسها هستند. در این پکیج مجموعه‌ای از کلاسها و اینترفیسها تعریف شده اند که به آنها `Collection` گفته می‌شود. `Collection`‌ها همانند آرایه‌ها حاوی مجموعه‌ای از آبجکتها هستند اما برخلاف آرایه‌ها، طول ثابت ندارد و در هر قسمتی از برنامه می‌توان آبجکتها را جدیدی به آن اضافه کرد، آبجکتها را از آن حذف کرد یا به هر صورت طول آنرا تغییر داد.

کلاس‌هایی که تحت عنوان `Collection` نامیده می‌شوند هر کدام به منظور خاصی طراحی شده اند، در برخی از آنها می‌توان عنصر تکراری نیز قرار داد در حالیکه در برخی دیگر عنصر تکراری مجاز نیست، برخی از آنها آبجکتها را به شکل مرتب شده ای در خود نگهداری می‌کنند در حالیکه در برخی دیگر آبجکتها به همان ترتیبی که در `Collection` قرار داده شده اند نگهداری می‌شود.

از آنجاییکه این کلاسها دارای رفتار مشترک نیز هستند، اینترفیس `Collection` برای `Collection`‌ها معرفی رفتار مشترک این کلاسها معرفی شده است، تمام کلاس‌های `Collection` حاوی متدهای معرفی شده در این اینترفیس هستند.

اینترفیس `List`، یکی از اینترفیس‌های این مجموعه است که از `Collection` مشتق شده است که از `Collection` مشتق شده است، در این اینترفیس علاوه بر معرفی متدهای دیگری برای دستکاری آبجکتها، امکان قرار دادن آبجکتها را تکراری در آن وجود دارد.

اینترفیس `Set`، که از `Collection` مشتق شده است، دارای هیچ متدهای اضافه ای نیست، در این اینترفیس (کلاس‌هایی که این اینترفیس را پیاده سازی کرده اند) امکان قرار دادن آبجکتها را تکراری وجود ندارد.

اینترفیس `SortedSet` از `Set` مشتق شده و بنابراین عنصر تکراری را مجاز نمی‌دارد عنصری که در `SortedSet` نگهداری می‌شوند به صورت صعودی مرتب می‌شوند، در این اینترفیس علاوه بر متدهایی که در اینترفیس `Collection` معرفی شده است، متدهایی برای بدست آوردن زیرمجموعه‌های از عناصر آن، معرفی شده است.

یک ساختمان داده دیگر است که عناصر آن به همان ترتیبی که وارد شده اند خارج می‌شوند. `Queue` کلاس‌های `ArrayList` و `LinkedList`، اینترفیس `List` را پیاده سازی کرده اند کلاس `HashSet` نیز اینترفیس `Set` را پیاده سازی کرده است.

اینترفیس `Map` یکی دیگر از اینترفیس‌های معرفی شده در `java.util` است که عناصر را به صورت «کلید/مقدار» نگهداری می‌کند هر عنصری که در `Map` قرار داده می‌شود دارای یک کلید نیز هست که از طریق آن می‌توان به عنصر داخل `Map` دسترسی پیدا کرد.

علاوه بر اینترفیسها و کلاس‌های `Collection` که در فوق مطرح شدند، کلاس‌های ساختمان داده دیگری نیز در پکیج `java.util` معرفی شده اند از آنجا که این کلاسها از نسخه‌های اولیه جاوا در این اینترفیس

تعریف شده اند، به آنها کلاس‌های قدیمی یا سنتی گفته می‌شود. اینترفیس Enumeration و کلاس‌های Properties و Hashtable و Vector از این مجموعه به شمار می‌روند. Enumeration مشابه HashTable شبیه HashMap است.

کلاس Properties که خصوصیات یک Hashtable را دارد آبجکتهاي String را نگهداری می‌کند در این کلاس کلیدها و مقدار آنها تماماً String هستند از این کلاس برای خواندن تنظیمات یک برنامه که در یک فایل متنه properties نوشته می‌شوند استفاده می‌شود.

کلاس Random، برای تولید اعداد تصادفی استفاده می‌شود در نسخه‌های پیشین جاوا متدهای random() که در کلاس Math تعریف شده است برای تولید اعداد تصادفی استفاده می‌شد اگرچه کلاس Math همچنان دارای متدهای random() است و می‌توان از آن استفاده نمود، اما این متدهای در یک برنامه Multithread ممکن است نتواند جوابگو باشد به همین خاطر کلاس Random که در آن مجموعه ای از متدهای nextXXX() پیاده سازی شده اند تعریف شده است.

کلاس‌های Date و Calendar معرف آبجکت تاریخ هستند بسیاری از متدهای کلاس Date از رده خارج شده اند به همین علت استفاده از کلاس Calendar که تعریف تاریخ‌های مختلف در جاوا را امکان‌پذیر کرده است توصیه می‌شود.

تمرینات

- ۱) کلاسی پیاده سازی کنید، که در سازنده خود یک آرایه از آبجکت دریافت کند و دارای متدهای `toQueue()`, `toList()` و `toSet()` باشد که آبجکتهای `List`, `Set` و `Queue` را از روی عناصر آرایه برگرداند.
- ۲) کلاسی بنویسید که دارای دو فیلد `Queue` باشد این کلاس که رفتار `Queue` را شبیه سازی می کند از اولین `Queue` برای نگهداری عناصر در صف استفاده می کند اما عناصری که از اولین `Queue` خارج می شوند در دومین `Queue` ذخیره می شوند تا بدین ترتیب تاریخچه صف اول در صف دوم نگهداری شود صف دوم تنها ۱۰ عنصر را نگهداری می کند و در صورتیکه تعداد عناصر آن بیشتر از ۱۰ شود باید اولین عنصر را اخراج می کند. این کلاس را پیاده سازی کنید و از آن در یک برنامه استفاده نمایید.
- ۳) کلاسی بنویسید و متدهای زیر را در آن پیاده سازی کنید
- متدهای بنویسید که یک `List` و یک آبجکت دیگر را به صورت پارامتر دریافت می کند و تعداد دفعاتی که آن آبجکت در `List` تکرار شده است را برمی گرداند.
 - متدهای بنویسید که آبجکتی از `HashSet` دریافت کند و آرایه ای از آبجکت که عناصر آن همان عناصر `HashSet` هستند را برگرداند.
 - متدهای بنویسید که یک `ArrayList` دریافت کند و یک آرایه مرتب شده حاوی عناصر آن را برگرداند. (در پیاده سازی این تمرین از کلاس `Arrays` استفاده کنید)
- ۴) کلاسی بنویسید که دارای یک فیلد `Stack` و دو فیلد `HashMap` باشد از این کلاس می خواهیم به مثابه یک `Stack` حافظه دار استفاده کنیم، هر عنصری که به `Stack` اضافه می شود در اولین `HashMap` و هر عنصری که از `Stack` حذف می شود در دومین `HashMap` نگهداری می شود. کلیدهای هر دو `HashMap` ترتیب عملیاتی است که با `Stack` انجام می شود به عنوان مثال اولین عمل، اضافه شدن یک عنصر به `Stack` است بنابراین عنصری که به `Stack` افزوده می شود با کلید «1» به اولین `HashMap` اضافه می شود اگر عمل دوم حذف شدن آبجکت از `Stack` باشد عنصری که حذف می شود با کلید «2» به دوم افزوده می شود. به این

ترتیب پس از مجموعه ای از عملیات که با Stack انجام می شود می توان با مراجعه به آبجکتهای HashMap به تاریخچه عملیاتی که با Stack انجام شده است پی برد.

(۵) برنامه ای بنویسید که زمان فعلی سیستم را در یک فایل properties . به صورت جمله Now = 20 April 2007

ذخیره کند.

(۶) متدى بنویسید که سال، ماه، روز، ساعت، دقیقه و ثانیه را دریافت کند و آبجکت Date که معرف آن زمان باشد را برگرداند (راهنمایی: سازنده کلاس Date که برای این منظور می توانند استفاده شوند همگی deprecated هستند به جای آن از Calendar استفاده کنید)

(۷) متدى بنویسید که یک ArrayList و یک آرایه را دریافت کند و هریک از عناصر آرایه که در وجود ندارد را به ArrayList اضافه کند و برگرداند.

(۸) متدى بنویسید که دو ArrayList را دریافت کند و مشترکات آن دو را در قالب یک ArrayList جدید برگرداند.

(۹) کلاسی پیاده سازی کنید که در سازنده خود یک رشته String را دریافت کند و دارای متدهای زیر باشد

- متدى داشته باشد که تعداد کلمات رشته را برگرداند.

- متدى داشته باشد که رشته را برگرداند

- متدى که کلمه ای که بیش از همه تکرار شده است را برگرداند.

- متدى که یک Map برگرداند که در آن هر کلمه به عنوان کلید و تعداد تکرار آن کلمه به عنوان مقدار آن کلید در آن ذخیره شده باشد.

(۱۰) کلاسی بنویسید که دو آبجکت Date را از طریق سازنده خود دریافت کند و دارای متدهایی باشد که هر کدام تفاوت زمانی دو آبجکت را (تفاوت تعداد روزها، تفاوت ماه ها، تفاوت ساعتها، و تفاوت تعداد میلی ثانیه هایی که دو تاریخ با یکدیگر اختلاف دارند) را برگرداند.

(۱۱) متدى بنویسید که دو آبجکت HashMap دریافت کند و عناصر آنها را با یکدیگر ترکیب کند.

(۱۲) متدى بنویسید که آرایه ای از String دریافت کند و ضمن حذف تکرارها، عناصر آرایه را مرتب کند. (راهنمایی: از TreeSet استفاده کنید)

(۱۳) اینترفیس Reportable را تعریف کنید که دارای متدهای () void reportHistory() و () void clearHistory() باشد هر Collection ای که این اینترفیس را پیاده سازی

ساختمان داده ها و پکیج `java.util`

کرده باشد دارای تاریخچه باشد یعنی بتوان از عملکرد آن گزارش تهیه نمود یا گذشته آنرا پاک کرد.

۱۴) کلاسی پیاده سازی کنید که توسط سازنده خود دو آبجکت `Map` دریافت کند و متدهایی زیر را داشته باشد

- متدى که تعداد ردیفهای تکراری (کلیدهای تکراری) را برگرداند
- متدى که کلیدهای تکراری را در قالب یک `List` برگرداند.

۱۵) متدى بنویسید که یک آرایه `Person` دریافت کند و یک آرایه `Person` برگرداند. این متدى تکراری های آرایه اول را حذف می کند و برمیگرداند.

۱۶) متدى بنویسید که دو آبجکت `ArrayList` دریافت کند و تعداد آبجکتهای مشترک آنها را برگرداند.

۱۷) کلاسی بنویسید که یک آبجکت `Map` را دریافت کند و محتویات آبجکت را در قالب یک جدول درخروجی استاندارد نمایش دهد.

۱۸) کلاسی بنویسید که صفتانوایی را مدل کند در این کلاس دو صفت "یکی" و "چندتایی" وجود خواهند داشت.

۱۹) متدى بنویسید که یک آبجکت `ArrayList` دریافت کند، عناصر تکراری در آبجکت دریافت نموده را حذف کند و یک آبجکت `ArrayList` حاوی عناصری غیرتکراری برگرداند.

۲۰) متدى بنویسید که دو آبجکت `ArrayList` دریافت کند و عناصر مشترک آنها را در قالب یک `ArrayList` برگرداند.

۲۱) در یک برنامه نیاز به یک آبجکت `HashSet` که دارای تاریخچه باشد یعنی بعد از اضافه نمودن آبجکتهای مختلف بتوانیم مشخص کنیم که چه آبجکتهایی به چه ترتیبی اضافه شده اند چه عناصری تکراری بوده اند و اضافه نشده اند. (راهنمایی: لازم است قابلیتهای جدیدی به کلاس `HashSet` اضافه شود)

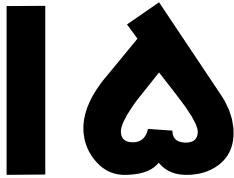
۲۲) کلاسی بنویسید که یک صفت تصادفی را مدل کند این صفت اگرچه عناصر صفت به ترتیب وارد می شود ولی خارج شدن آنها از صفت بر اساس فرمول زیر انجام می شود.

- اگر تعداد عناصر صفت فرد باشد، نوبت عنصر وسط است.

• اگر تعداد عناصر فرد زوج باشد، همانند صفت معمولی رفتار می شود و نوبت به عنصری که قبل از بقیه وارد صفت شده می رسد.

۲۳) کلاسی بنویسید که یک `Stack` با ویژگیهای زیر باشد
در این `Stack`، دو سطح بالا (`high`) و پایین (`low`) وجود داشته باشد. وقتی عنصری به `Stack` اضافه می شود به صورت پیش فرض به سطح بالا اضافه شود اگر بخواهیم عنصری به سطح پایین اضافه شود، متددیگری با نام `pushLow` را باید فراخوانی نمود این متدد کارکردی

- مشابه push دارد با این تفاوت که عنصر را به سطح پایین اضافه می کند. وقتی عنصری pop می شود به صورت پیش فرض عنصر از سطح بالا حذف می شود مگر اینکه سطح بالا خالی باشد.
- (۲۴) یک کلاس Stack پیاده سازی کنید که عناصر خود را «مرتب شده» نگهداری کند و وقتی pop انجام می شود، کوچکترین عنصر از Stack حذف شود.
- (۲۵) یک کلاس Stack «ورود-ممنوع و خروج-ممنوع» پیاده سازی کنید که مانع از وارد شدن یا مانع از خارج شدن عناصر خاصی می شود. این کلاس باید یک متده addBannedInput (Object) برای افزودن به لیست ورودیهای ممنوعه، و متده addBannedOutput (Object) برای افزودن به لیست خروجی های ممنوعه باشد. (توضیح: وقتی عنصر «ورود ممنوع» به استک وارد شود، استک آنرا به کلی نادیده می گیرد و آنرا اضافه نمی کند و وقتی یک عنصر «خروج ممنوع» از استک pop می شود استک مانع از خروج آن می شود و به جای آن عنصر بعدی را pop می کند)
- (۲۶) متده بنویسید که آرایه ای از عدد int را دریافت کند و آرایه را به صورت معکوس برگرداند در صورتیکه آرایه null یا خالی باشد خطای NotConversionException (کلاس خطایی که شما باید آنرا پیاده سازی کنید) تولید شود.
- (۲۷) متده بنویسید که یک String دریافت می کند و کاراکترهای عددی را در رشته قرار دارند را به هم می چسباند و از ترکیب آنها یک عدد برمی گرداند
 "1da219ds29re0" → 1219290
 اگر هیچ کاراکتر عددی در رشته وجود نداشته باشد (مثلا "aeislw, sw") خطای DigitNotFoundException تولید شود.
- (۲۸) یک کلاس Queue پیاده سازی کنید که دارای یک متده reverse () باشد به گونه ای که وقتی متده reverse () فراخوانی می شود ترتیب عناصر صف معکوس شود.



Generics

کلاس Container را که دارای دو فیلد data و description است و به ترتیب یک آبجکت و توضیحی مربوط به آن آبجکت است را به صورت زیر در نظر بگیرید.

```
1 package ir.atlassoft.javase.chapter15;
2
3 public class Container {
4     Object data;
5     String description;
6
7     public Container(Object data, String description) {
8         this.data = data;
9         this.description = description;
10    }
11
12    public Object getData() {
13        return data;
14    }
15
16    public void setData(Object data) {
```

```

17         this.data = data;
18     }
19
20     public String getDescription() {
21         return description;
22     }
23
24     public void setDescription(String description) {
25         this.description = description;
26     }
27 }
```

کد ۱۵-۱

از آنجاییکه فیلد data از جنس `java.lang.Object` تعریف شده، هر آبجکتی می‌تواند در این فیلد نگهداری شود. (تمام کلاسهای جاوا از کلاس `java.lang.Object` مشتق شده‌اند و بنابراین تمام آبجکتهای جاوا از جنس `java.lang.Object` محسوب می‌شوند) به این ترتیب هر دو کد زیر صحیح هستند:

```
Container h1 = new Container ("Generics", "A Java Feature!");
Container h2 = new Container (new Date(), "Current Time");
```

آبجکت h1 داده‌ای از جنس `String` و آبجکت h2 داده‌ای از جنس `java.util.Date` را نگهداری می‌کند.

اگر با فراخوانی متده `getData()` به آبجکتی که در h1 و h2 نگهداری می‌شود دسترسی پیدا کنیم به صورت زیر، هر دو آبجکتی از نوع `Object` بر می‌گردانند:

```
Object data1 = h1.getData();
Object data2 = h2.getData();
```

و از آنجاییکه ما می‌دانیم آبجکتی که در h1 قرار دارد از جنس `String` و آبجکتی که در h2 قرار دارد از جنس `Date` است می‌توانیم آنها را به صورت زیر تبدیل کنیم:

```
String data1 = (String)h1.getData();
Date data2 = (Date)h2.getData();
```

اگر این تبدیل را به اشتباه انجام دهیم کامپایلر جاوا قادر به تشخیص آن نیست اما در زمان اجرا، با خطای زمان اجرا مواجه می‌شویم:

```
Date data1 = (Date)h1.getData(); //runtime exception
```

Generics

مسایل و مشکلاتی از این نوع را از بین می برد. هدف Generic تولید کدهای بامفهوم و کم کردن خطاهای سهوی برنامه نویسان است. برای استفاده از خصوصیت Container، کلاس Generics را به صورت زیر تغییر می دهیم.

```
1 package ir.atlassoft.javase.chapter15;
2
3 public class Container1<E> {
4     E data;
5     String description;
6
7     public Container1(E data, String description) {
8         this.data = data;
9         this.description = description;
10    }
11
12    public E getData() {
13        return data;
14    }
15
16    public void setData(E data) {
17        this.data = data;
18    }
19
20    public String getDescription() {
21        return description;
22    }
23
24    public void setDescription(String description) {
25        this.description = description;
26    }
27 }
```

کد ۱۵-۲

همانطور که ملاحظه می کنید در تعریف کلاس Container1 از علامت <E> استفاده شده است و در این کلاس تمام Object ها با E جایگزین شده است. معنی این تعریف این است که در زمان ایجاد آبجکت باید نوع کلاسی که کلاس Container1 قرار است آبجکتهایی از آن را نگهداری کند مشخص کنیم. با تعریف فوق، ساختن آبجکتهای h1 و h2 به صورت زیر انجام می شود

```
Container1<String> h1 = new Container1<String>("Generics", "A Java Feature!");
Container1<Date> h2 = new Container1<Date>(new Date(), "Current Time");
```

در اینجا `Date` و `String` جایگزین `E` در تعریف کلاس `Container1` شده اند. از تعریف `Container1<String>` انتظار داریم که فیلد `data` در آبجکت `h1` از جنس `String` باشد و در آبجکت `h2` از جنس `Date` باشد. نکته جالب اینکه اگر متدهای `getData()` را از هر دو آبجکت `h1` و `h2` فراخوانی کنیم، مقدار برگشتی از جنس مورد نظر خواهد بود، بدون اینکه نگران تبدیل به جنس مورد نظر باشیم.

```
String data1 = h1.getData();
Date data2 = h2.getData();
```

به `<E>` در تعریف کلاس `Container1`، پارامتر کلاس گفته می شود. استفاده از فرم `Generic` در بسیاری از کلاس‌های جاوا استاندارد (Java SE) مشاهده می شود که مهمترین آنها کلاس‌های `HashSet`, `LinkedList`, `ArrayList`, `Set`, `List`, `Collection` و `HashMap` هستند. به عنوان مثال وقتی می خواهید لیستی از `String` ایجاد کنید می توانید به شکل زیر آنرا انجام دهید:

```
List<String> strings = new ArrayList<String>();
dr اینصورت فقط امکان افزودن آبجکتهاي String را به لیست خود دارد.
```

```
strings.add("Java Development");
strings.add(new Date()); //compile error
به شکل مشابه برای ایجاد یک آبجکت Set از اعداد Long به شکل زیر عمل می شود.
HashSet<Long> longSet = new HashSet<Long>();
```

نام کذاری پارامترهای Generic

در کلاس `Container` از کلمه `E` به عنوان نام پارامتر استفاده کردیم. در حقیقت در جاوا کلاسی با نام `E` یا داده ای با نام `E` وجود ندارد، و اشاره به یک کلاس یا نوع داده ای دارد که در زمان ساخته شدن آبجکت از کلاس `Container` مشخص می شود. ممکن است بپرسید چرا از کلمه `E` استفاده کردیم، آیا می توانیم از کلمات دیگری مثل `A`, `B` یا `C` استفاده کنیم؟ بله می توانید از هر نامی که بر اساس قواعد نامگذاری جاوا انتخاب شده باشد به جای `E` استفاده کنید هرچند در اغلب مواقع برنامه نویسان از `E` (شروع کلمه `Element`) استفاده می کنند.

همچنین توجه داشته باشید که یک کلاس می تواند بیش از یک پارامتر `Generic` داشته باشد و اساساً محدودیتی در تعداد پارامترهای `Generic` وجود ندارد. به عنوان نمونه، کلاس `Container` می تواند با دو پارامتر `Generic` به صورت زیر تعریف شود.

```

1 package ir.atlassoft.javase.chapter15;
2
3 public class Container2<E, F>{
4     E data1;
5     F data2;
6     //...
7 }

```

کد ۱۵-۳

متدها و سازنده های Generic

همانطور که گفته شد، E یک کلاس یا یک نوع داده نیست بلکه پارامتری است که در زمان ساخته شدن آبجکت با یک کلاس جایگزین می شود. معنی این جمله این است که وقتی آبجکتی از روی Container1 ساخته می شود

```
Container1<String> h1 = new Container1<String>();
```

در کلاس Container1 هرجایی که از E استفاده شده با String جایگزین می شود و سپس آبجکتی از کلاس Container1 ایجاد می شود. بنابراین یکی از وظایف کامپایلر، جایگزین کردن پارامترهای Generic با کلاسهای متناسب (مثلا Date یا String یا String) یا هر کلاسی دیگری که استفاده کرده اید) است. از این توضیحات می توان نتیجه گیری کرد که در زمان اجرای برنامه، اثری از پارامترهای Generic در آبجکتهاشان ساخته شده در حافظه وجود ندارد. با این حساب، آیا می توانیم متد () را createInstance که به صورت زیر نوشته شده به کلاس Container1 اضافه کنیم؟

```
public E createInstance() {
    return new E();
}
```

خیر! گُد فوق کامپایل نمی شود زیرا همانطور که در بخش‌های اول این کتاب آموختید ساخته شدن آبجکت با استفاده از عملگر new مربوط به زمان اجرای برنامه است و ما می دانیم که در زمان اجرا، اثری از پارامتر E وجود ندارد تا بتوان از روی آن آبجکت ساخت. استفاده از پارامترهای کلاس فقط در قسمتهایی مجاز است که توسط کامپایلر در زمان کامپایل قابل جایگزینی با نوع مورد نظر باشند، در غیر اینصورت با خطای کامپایل مواجه خواهید شد.

حال سوال دیگری را پاسخ دهید، آیا می توان مشابه کلاس Generic (کلاس Container1 با پارامتر E) در سطح متد نیز از Generic استفاده کرد؟ یعنی متدهای داشت که با پارامتر F کار کند؟ برای جواب به این سوال، باید نگاهی به متد () getdata در کلاس Container1<E> بیندازیم.

```
public E getData() {
    return data;
}
```

در اینجا، متدهای `getData()` جنسی از پارامتر کلاس را بر می گرداند. به عبارت دیگر مقدار برگشتی این متدهای جنس همان پارامتری است که در ایجاد آبجکت `Container1` استفاده شده است. سوال در اینجا این است که آیا یک متدهای پارامتری مستقل از پارامترهای کلاس داشته باشد؟ جواب این سوال نیز مثبت است. برای درک این موضوع به متدهای `logAndReturn` توجه کنید:

```
public <U> U logAndReturn(U u) {
    System.out.println(u.getClass());
    return u;
}
```

این متدهای یک پارامتر اختصاصی (متفاوت از پارامترهای کلاس) با نام `U` تعریف شده است. کار این متدهای دریافت یک آبجکت، چاپ نام کلاس آن آبجکت در کنسول برنامه، و سپس برگرداندن آن آبجکت است. در اینجا از علامت `<U>` برای مشخص کردن پارامتر استفاده شده است با این تفاوت که بخلاف پارامترهای کلاس که بلا فاصله بعد از نام کلاس قرار می گیرند، قبل از نوع برگشتی متدهای مشخص شده است. نحوه فراخوانی متدهای فوق به صورت زیر خواهد بود:

```
String v = this.<String>logAndReturn("");
Date l = this.<Date>logAndReturn(new Date());
```

در این گذشته، فرض بر این است که این فراخوانی از داخل کلاس `Container1` فراخوانی شده و به همین دلیل از پیشوند `this` قبل از فراخوانی استفاده شده است. اگر این متدهای جایی خارج از کلاس `Container1` فراخوانی شود طبیعی است که باید به جای کلمه `this` از نام آبجکت `Container1` استفاده شود. اگر متدهای استاتیک تعریف شده باشد باید به جای این متدهای `this` از نام کلاس استفاده کرد و اگر این متدهای درون یکی از کلاسهای فرزند فراخوانی شود از کلمه `super` به جای `this` استفاده می شود. نکته جالب اینکه، از آنجاییکه سازنده های کلاس نیز از جمله متدهای کلاس محسوب می شوند، آنها نیز می توانند پارامترهای `Generic` اختصاصی (مستقل از پارامترهای کلاس) داشته باشند:

```
public <U> Container1(U u) {
    ...
}
```

به این ترتیب برای ایجاد آبجکتی از روی کلاس `Container1` باید به صورت زیر عمل کنید:

```
Container1<String> container = new
<Date>Container1<String>(new Date());
```

Generics

در اینجا `<Date>` مشخص کننده پارامتر Generic سازنده، و `<String>` مشخص کننده پارامتر Generic کلاس است.

یعنی آیا می توان بدون داشتن یک کلاس Generic، یک متده است.

استفاده از نشانه های ?، super و extends

فرض کنید متده نوشته ایم که لیستی از اعداد Long را دریافت می کند و جمع عددی اعداد را بر می گرداند

```
static double sum(List<Long> list) {  
    double sum = 0.0;  
    for (Long n : list)  
        sum += n.doubleValue();  
    return sum;  
}
```

اما اگر بخواهیم متده فوق را به گونه ای تغییر دهیم که بتواند لیستی از هر نوع عددی (شامل Integer، Long، Float، ...) دریافت کند و جمع عددی آنها را برگرداند متده `sum` را به صورت زیر تغییر می دهیم

```
static double sum(List<Number> list) {  
    double sum = 0.0;  
    for (Long n : list)  
        sum += n.doubleValue();  
    return sum;  
}
```

تنها تغییری که داده ایم Long را به Number تغییر داده ایم این تغییر، منطقی به نظر می رسد زیر تمام کلاسهای عددی از قبیل Integer، Long،Float... همگی از کلاس Number مشتق شده اند.

اما متأسفانه فراخوانی متده `sum` با آبجکتی از List<Integer> با خطای کامپایل مواجه می شود:

```
List<Integer> l = new ArrayList<Integer>();  
l.add(1);  
l.add(4);  
l.add(9);  
double sum = sum(l); // INVALID: won't compile
```

نکته این خطا در اینجاست که اگرچه کلاس Integer فرزند کلاس Number است (هر آبجکت از جنس Number هم محسوب می شود) اما آبجکت List<Integer> از آبجکتی از جنس List<Number> محسوب نمی شود به همان دلیلی که [[آبجکتی از Number]] محسوب نمی شود.

بنابراین در تعریف آبجکت List ما به روشنی نیاز داریم که به جای لیستی از Number، لیستی از Number و هر کلاسی که از Number مشتق شده است را مشخص کنیم. اینکار از طریق نشانه ؟ امکان پذیر است

```
static double sum(List<? extends Number> list) {
    double sum = 0.0;
    for (Number n : list)
        sum += n.doubleValue();
    return sum;
}
```

در اینجا متدهای `sum`، آبجکت `List` ای دریافت می‌کند که می‌تواند حاوی `Number` یا هر آبجکتی که از `Number` مشتق شده است باشد. دقت کنید که کلمه `extends` که در اینجا بکار رفته است به معنی `implements` هم هست یعنی اگر بخواهیم لیستی از آبجکتها را مشخص کنیم که اینترفیس `List<? extends Serializable>` را پیاده سازی کرده اند آنرا به صورت `Serializable` می‌نویسیم. از این گذشته، با استفاده از نشانه `&` می‌توان آبجکتها را مشخص نمود که از یک کلاس مشتق، و یک اینترفیس را پیاده سازی کرده اند مثلاً `List<? extends Value & Serializable>` لیستی از آبجکتها را مشخص می‌کند که از کلاس `Value` مشتق شده اند و اینترفیس `Serializable` را پیاده سازی کرده اند.

اجازه دهید به مثال `sum` برگردیم. عبارت `List<? extends Number>` لیستی از همه آبجکتها را مشخص می‌کند که زیرمجموعه کلاس `Number` قرار داردند به همین دلیل گفته می‌شود عبارت «`? extends ?`» سقف آبجکتها را مشخص می‌کند. به شکل مشابه می‌توان با استفاده از عبارت «`super`» لیستی از آبجکتها را مشخص نمود که همگی پدر یک کلاس محسوب می‌شوند و به همین دلیل گفته می‌شود عبارت «`super ?`» کف آبجکتها را مشخص می‌کند.

برخلاف `List<Number>` که از جنس `Number` محسوب نمی‌شود و نمی‌توان به جای `List<? extends Integer>` از آن استفاده نمود، آبجکت `List<Integer>` از جنس `List<Number>` محسوب می‌شود. با تحلیل مشابه `List<? extends Integer>`

زیرمجموعه

`List<? extends Number>`

۹

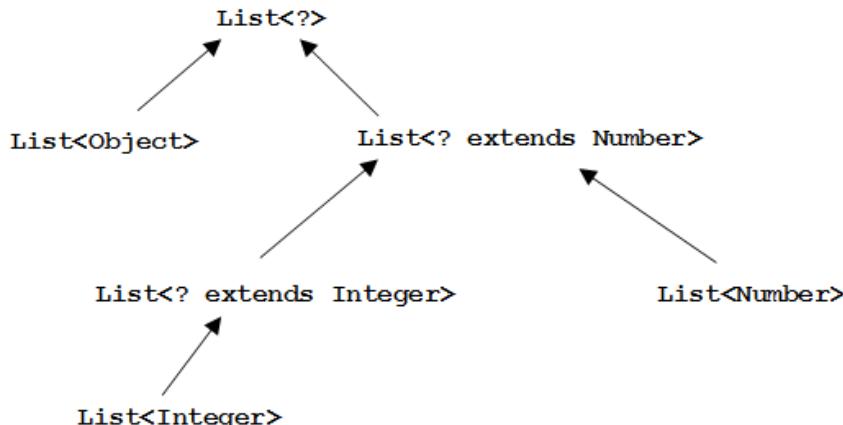
`List<? super Number>`

زیرمجموعه

`List<? super Integer>`

محسوب می‌شود. شکل زیر این مطلب را بهتر نشان می‌دهد.

Generics



شکل ۳-۱۵. ارث بری در Generics

دقت کنید که `List<? extends Object>` به معنی `List<Object>` است. همچنین توجه داشته باشید که استفاده از `? extends` فقط در تعریف پارامترهای متند امکانپذیر است و نمی‌توان از آن برای تعریف پارامترهای کلاس استفاده نمود.

در اینجا این سوال مطرح است که اگر یک کلاس به فرم **Generic** تعریف شده باشد، آیا می‌توان بدون مشخص کردن مقدار پارامتر از روی آن کلاس آبجکت ایجاد کرد؟ جواب این سوال مثبت است زیرا همانطور که گفته شد تمام کلاس‌های `Collection` از قبیل `ArrayList`, `List`, `Set`, `HashSet`, `ArrayList` دارای پارامترهای **Generic** هستند، در حالیکه تا قبل از این فصل، بدون مشخص کردن پارامتر از روی این کلاسها آبجکت می‌ساختیم.

```
List list = new ArrayList();
```

اما در اینصورت چه مقداری برای پارامترها در نظر گرفته می‌شود؟ جواب این این است که اگر مقداری برای پارامترها مشخص نکنید، «**صف آبجکتها**» برای پارامترها در نظر گرفته می‌شود. برای اینترفیس `List` که به صورت

```
public interface List<E> extends Collection<E> {  
    ...  
}
```

تعریف شده است، سقف پارامتر `Object` است و بنابراین کلاس `java.lang.Object` به جای `E` قرار می‌گیرد. اما اگر پارامتر کلاس `Container1` که ابتدای این فصل پیاده سازی کردیم را به صورت زیر تغییر دهیم

```
public class Container1<E extends Number> {  
    ...  
}
```

سقف مقدار پارامتر کلاس `java.lang.Number` خواهد بود که معنی آن این است که در صورت ایجاد آبجکت از کلاس `Container1` به شکل زیر

```
Container1 c = new Container1();
```

کلاس `java.lang.Number` به عنوان مقدار پارامتر `E` در نظر گرفته خواهد شد.

ارث بری

یک کلاس `Generic` می‌تواند از یک کلاس غیر `Generic` مشتق شود که کاملاً قابل تصور است زیرا همه کلاسهای جاوا به صورت پیش فرض از کلاس `Object` که یک کلاس غیر `Generic` است مشتق می‌شوند (کلاس `Container` چنین وضعیتی دارد).

یک کلاس غیر `Generic` می‌تواند از یک کلاس `Generic` مشتق شود که در اینصورت می‌تواند پارامتر کلاس پدر خود را مشخص کند یا مقدار پیش فرض آن پارامتر یعنی سقف پارامتر لحاظ می‌شود. به دو کلاس زیر توجه کنید.

```
public class CustomList extends ArrayList<String>{
...
}
public class CustomList extends ArrayList{
...
}
```

همچنین یک کلاس `Generic` می‌تواند از یک کلاس `Generic` مشتق شود که در اینصورت کلاس فرزند می‌تواند از پارامترهای `Generic` کلاس پدر خود استفاده کند.

```
public class CustomList<E> extends ArrayList<E>{
...
}
```

تمرینات

- (۱) کلاس ELinkedList که زنجیره ای از آبجکتها را مشخص می کند پیاده سازی کنید. کلاس ELinkedList با آبجکتهایی از جنس اینترفیس Linkable کار می کند که دارای دو متده است متده setNext() که از طریق آن می توان آبجکت Linkable بعدی لیست را مشخص کرد و متده getNext() که آبجکت head که آبجکتی از Linkable است و آبجکت آغازین لیست را مشخص می کند. این کلاس دو متده برای اضافه کردن و حذف کردن آبجکتهای در لیست دارد. کلاس ELinkedList و اینترفیس Linkable را به صورت generic پیاده سازی کنید.
- (۲) یک متده generic با نام findBigger() پیاده سازی کنید که یک، یا بیش از یک آبجکت با ترکیب آنها یک آبجکت Comparable را (در قالب یک آرایه) دریافت می کند و با مقایسه آنها بزرگترین آنها را برمی گرداند.
- (۳) یک متده generic پیاده سازی کنید که دو آبجکت Collection از یک جنس را دریافت می کند و با ترکیب آنها یک آبجکت Collection از همان جنس برمی گرداند.
- (۴) یک متده swapElements() پیاده سازی کنید که یک آرایه generic و دو عدد int با نام های `i` و `j` دریاف می کند، و ضمن جابجا کردن عنصر `i` و `j` آرایه را برمی گرداند. در پیاده سازی این متده، بررسی اینکه اعداد `i` و `j` در محدوده آرایه هستند می باشد با تولید `IllegalArgumentException` کنترل شود.
- (۵) یک کلاس NumberHolder به صورت generic پیاده سازی کنید که لیستی از اعداد (Number) را نگهداری کند و دارای متده add() باشد که بتوان یک عدد را به لیست آن اضافه نمود. این کلاس می باشد دارای متدهای `min()` و `max()` باشد که کوچکترین و بزرگترین عدد لیست را برمی گردانند. همچنین متدهای `first()` و `last()` اولین و آخرین اعداد لیست را برمی گردانند.

۱۴

ورودی و خروجی

با استفاده از کلاس‌هایی که در پکیج `iо.java` قرار دارند می‌توانید عملیات ورودی و خروجی انجام دهید، منظور از عملیات ورودی و خروجی، عملیاتی است که برای ورود داده از یک منبع خارجی به برنامه یا خروج داده از برنامه به منبع خارجی انجام می‌شود.

در اغلب برنامه‌های جاوا، لازم است تا داده‌ها را از یک منبع خارجی دریافت کنیم یا به یک منبع خارجی ارسال کنیم این منبع خارجی ممکن است یک دیسک، فایل، یک جایی درون شبکه، حافظه کامپیوتر یا حتی یک برنامه دیگر باشد. داده‌ها نیز ممکن است، آجکت، رشتة، عکس، صوت یا هرگونه اطلاعات دیگری باشند. به این منظور در جاوا کلاس‌هایی تعریف شده اند که با استفاده از آنها می‌توان عملیات ورودی و خروجی داده‌ها را انجام داد. ورودی و خروجی داده که به اختصاراً به آن `IO` گفته می‌شود یکی از قسمتهای پراهمیت در برنامه نویسی به شمار می‌رود.

File

یکی از کلاس‌های `iо.java`، کلاس `File` است در صورتیکه بخواهید در یک برنامه با سیستم فایل یا فایلهای درون سیستم ارتباط پیدا کنید یا روی فایلهای سیستم عملیاتی انجام دهید، باید از این کلاس استفاده کنید. با استفاده از این کلاس می‌توانید اطلاعات یک فایل را بخوانید یا آن را دستکاری کنید مثلاً می‌توانید حق دسترسی (permission)، تاریخ و زمان ایجاد یا تغییر در فایل یا دایرکتوری (directory) را

بخوانید یا اینکه در دایرکتوری های داخل سیستم جا به جا شوید. در جاوا دایرکتوری و فایل یکسان هستند و برای هر دوی آنها باید از کلاس `File` استفاده کنید. کلاس `File` دارای چندین سازنده است که در زیر تعدادی از آنها را مشاهده می کنید

```
File(String direcotryPath)
File(String directoryPath, String filename)
File(File dirObj, String filename)
File(URI uriObj)
```

`directoryPath` مشخص کننده مسیر یک دایرکتوری یا یک فایل است، `filename` نام یک فایل است، `dirObj` یک آبجکت از کلاس `File` است که مشخص کننده یک دایرکتوری است، `uriObj` مشخص کننده یک آبجکت از کلاس `URI` است که مشخص کننده یک فایل است. در زیر آبجکت هایی از کلاس `File` ساخته شده است:

```
File f1 = new File("/");
File f2 = new File("/", "autoexe.bat");
File f3 = new File(f1, "autoexe.bat");
```

`f1` مشخص کننده یک دایرکتوری است این دایرکتوری همان دایرکتوری است که کلاسهای برنامه ای که در حال اجراست در آن قرار دارند (" / " مشخص کننده دایرکتوری نسبی، نسبت به محل اجرای برنامه است) برای ساختن `f2` از دو آرگومان استفاده شده است آرگومان اول مسیر فایل و آرگومان دوم نام فایل است `f3` مشخص کننده یک فایل با نام `autoexe.bat` است که داخل دایرکتوری جاری قرار دارد.

توجه: در سیستم عامل «ویندوز» برای مشخص کردن مسیر یک فایل از `/` استفاده می شود اما اگر از `/` هم استفاده کنید، «ویندوز» بدرستی مشخص شده را شناسایی خواهد کرد، اما در سیستم عامل هایی همانند «بونیکس» یا «لینوکس» برای مشخص کردن مسیر یک فایل فقط از `/` استفاده می شود در این سیستم عامل ها استفاده از `/` برای مشخص کردن مسیر یک فایل معتبر نیست. بنابراین برای اینکه برنامه ای که می نویسید در تمام سیستم عامل ها قابل اجرا باشد باید از `/` برای مشخص کردن مسیر یک فایل استفاده کنید.

Linux, Unix, Solaris,... :	C:/Java_Programs/src/FirstClass.java
Windows :	C:\Java_Programs\src\FirstClass.java

توجه داشته باشید از آنجاییکه \ از جمله کاراکترهای کنترلی به شمار می رود نمی توانید مستقیما آنرا در یک رشته بکار ببرید در اینصورت باید به جای \ \ استفاده کنید مثلا برای مشخص کردن مسیر فایل فوق، باید آنرا به صورت زیر مشخص نمایید:

ورودی و خروجی

```
File f = new File("C:\\Java_Programs\\src\\FirstClass.java");
```

کلاس File دارای متدهایی است که از طریق آنها می‌توانید اطلاعات و مشخصه‌های مربوط به یک فایل یا دایرکتوری را استخراج کنید برای مثال متد getName() نام یک فایل را برمی‌گرداند، متد exists() مشخص می‌کند آیا فایل یا دایرکتوری مشخص شده در سیستم وجود دارد یا خیر. مثال زیر عملکرد برخی از متدهای کلاس File را نشان می‌دهد:

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.File;
4 import java.io.IOException;
5
6 public class UsingFile {
7
8     public static void main(String[] args)
9             throws IOException {
10
11         File f = new File("C:/temp/picture.jpg");
12         System.out.println("Path: "+f.getPath());
13         System.out.println("Name: "+f.getName());
14         System.out.println("Absolute Path: "+
15                             f.getAbsolutePath());
16         System.out.println("Parent: "+f.getParent());
17         System.out.println("Canonical Path: "+
18                             f.getCanonicalPath());
19     }
20 }
```

کد ۱۷۸

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.

```
Path: C:\\temp\\picture.jpg
Name: picture.jpg
Absolute Path: C:\\temp\\picture.jpg
Parent: C:\\temp
Canonical Path: C:\\temp\\picture.jpg
```

درصورتیکه آبجکتی که از روی کلاس File ساخته شده است مشخص کننده یک فایل باشد فراخوانی متده () مقدار isFile true برمی گردداند در غیر اینصورت نتیجه فراخوانی آن مقدار false خواهد بود. درصورتیکه آبجکتی که از روی کلاس File ساخته شده است از طریق یک آدرس نسبی ساخته شده باشد فراخوانی متده () isAbsolute false برمیگرداند در غیر اینصورت مقدار true برخواهد گرداند.

کلاس File دارای دو متده دیگر به نامهای

```
boolean renameTo(File newName)
boolean delete()
```

نیز می باشد .

با فراخوانی متده () renameTo می توانید نام یک فایل یا دایرکتوری را به نام جدید newName تغییر دهید، درصورتیکه تغییر نام با موفقیت انجام شود این متده مقدار true برمی گردداند و در صورت عدم موفقیت در تغییر نام، مقدار false برمیگرداند.

با فراخوانی متده () delete می توانید یک فایل یا دایرکتوری را از روی سیستم حذف کنید، با حذف موفقیت آمیز آن فایل یا دایرکتوری، این متده مقدار true بر میگرداند و در صورت عدم حذف، مقدار false برگردانده می شود.

جدول ۱۶-۱ برحی متدهای کلاس File را نشان می دهد.

توضیح	متده
وقتی اجرای JVM به صورت معمولی به پایان می رسد فایل مرتبط با آبجکتی که متده آن فراخوانی شده است از سیستم حذف می کند.	void deleteOnExist()
اگر فایل مرتبط با آبجکتی که متده آن فراخوانی شده است باشد فراخوانی این متده مقدار true و در غیر اینصورت مقدار false برمی گردداند.	boolean isHidden()
آخرین زمان تغییر فایل را به millisecond تغییر می دهد. millisecond مشخص کننده تعداد ثانیه های سپری شده از اول ژانویه ۱۹۷۰ تاکنون می باشد.	boolean setLastModified(long millisec)
فایل را به صورت فقط خواندنی تبدیل می کند.	boolean setReadOnly()

جدول ۱۶-۱. متدهای کلاس File

ورودی و خروجی

دایرکتوری ها

هر دایرکتوری در برنامه جawa همانند یک فایل با کلاس `File` مشخص می شود وقتی یک آبجکت از کلاس `File` ایجاد می شود که مشخص کننده یک دایرکتوری است فراخوانی متده است `isDirectory()` مقدار `true` برمنی گرداند یک دایرکتوری ممکن است خود حاوی فایلها یا دایرکتوری های دیگری باشد در این وضعیت می توانید با فراخوانی متده `list()`، لیست تمامی فایلها و دایرکتوری هایی که داخل این دایرکتوری قرار دارند را استخراج کنید:

```
String[] list()
```

برنامه زیر نحوه استفاده از متده `list()` را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.File;
4
5 public class DirList {
6
7     public static void main(String args[]) {
8
9         String dirname = "/Projects/Book-Samples";
10        File f1 = new File(dirname);
11
12        if (f1.isDirectory()) {
13            System.out.println("Directory of " + dirname);
14            String s[] = f1.list();
15            for (int i = 0; i < s.length; i++) {
16                File f = new File(dirname + "/" + s[i]);
17                if (f.isDirectory()) {
18                    System.out.println(s[i] +
19                            " is a directory");
20                } else {
21                    System.out.println(s[i] +
22                            " is a file");
23                }
24            }
25        } else {
26            System.out.println(dirname +
27                                " is not a directory");
28        }
29    }
30}
```

```
29     }
30 }
```

کد ۱۶-۲

اجرای برنامه فوق نتیجه‌ای مشابه زیر در پی خواهد داشت (بسته به اینکه چه دایرکتوری را مشخص کرده باشید)

```
Directory of /Projects/Book-Samples
src is a directory
Book-Samples.iml is a file
Book-Samples.ipr is a file
classes is a directory
Book-Samples.iws is a file
```

استفاده ازFilenameFilter

اگر بخواهید لیست تمامی فایلها یا دایرکتوری هایی را که در یک دایرکتوری قرار دارند و دارای شرایط خاصی هستند را بدست آورید، مثلاً بخواهید تمام فایلها مخفی (hidden) یا تمام فایلها با اندازه کمتر از یک مگابایت یا... که داخل یک دایرکتوری قرار دارند را بدست آورید، در اینصورت باید از متدهای list() به صورت زیر استفاده کنید:

```
String[] list(FilenameFilter fnfObj)
```

آبجکتی از کلاسی است که اینترفیس fnfObj را پیاده سازی کرده است این اینترفیس دارای یک متدهای accept که در حقیقت مشخص کننده فایلهاست که حائز شرایط شما هستند.

```
boolean accept(File directory, String filename)
```

در صورتیکه یک فایل حائز شرایط تعیین شده باشد این متدهای accept برمی‌گرداند و در غیر اینصورت مقدار false برخواهد گرداند. در حقیقت این کلاس شبیه غربال عمل می‌کند، وقتی فایلها داخل یک دایرکتوری را با استفاده از این غربال جدا می‌کنند تنها فایلهاست که حائز شرایط باشند از بقیه جدا می‌شوند. کدهای زیر یک کلاس فیلتر را نشان می‌دهد، که فایلهاست از یک نوع (با یک پسوند خاص) را مشخص می‌کند.

```
1 import java.io.*;
2
3 package ir.atlassoft.javase.chapter16;
4
```

ورودی و خروجی

```
5 import java.io.File;
6 import java.io.FilenameFilter;
7
8 public class OnlyExt implements FilenameFilter {
9     String ext;
10
11     public OnlyExt(String ext) {
12         this.ext = "." + ext;
13     }
14
15     public boolean accept(File dir, String name) {
16         return name.endsWith(ext);
17     }
18 }
```

کد ۱۶-۳

در زیر کلاسی نوشته شده که در آن با استفاده از این فیلتر، فایلهای تمام فایلها با پسوند `html` را که داخل یک دایرکتوری وجود دارند یافته می شوند:

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 class DirListOnly {
6
7     public static void main(String args[]) {
8         String dirname = "/java";
9         File f1 = new File(dirname);
10        FilenameFilter only = new OnlyExt("html");
11        String s[] = f1.list(only);
12        if(s == null) {
13            return;
14        }
15        for (int i = 0; i < s.length; i++) {
16            System.out.println(s[i]);
17        }
18    }
19 }
```

مجموعه متدهای listFiles()

مشابه متدهای list()، متدهای listFiles() وجود دارد که لیست فایلها درون یک دایرکتوری را بر می گرداند این متدهای listFiles() که آرایه ای از String بر می گردانند (این آرایه مشخص کننده نام تمام فایلها و دایرکتوری هایی است که در یک دایرکتوری وجود دارند) آرایه ای از File بر می گرداند، این آرایه معرف فایلها و دایرکتوری های داخل آن دایرکتوری است.

```
File[] listFiles()
File[] listFiles(FilenameFilter fnfObj)
File[] listFiles(FileFilter ffObj)
```

در متدهای فوق اولین listFiles()، لیست تمامی فایلها و دایرکتوری های داخل یک دایرکتوری را به صورت آرایه ای از File بر می گرداند.

دومین متدهای listFiles()، لیست تمامی فایلها را بر می گرداند که توسطFilenameFilter اینترفیس است که دارای یک متد accept() است که می تواند فایل را بازگشته باشد.

سومین listFiles()، لیست تمامی فایلها را بر می گرداند که مسیر آنها توسط FileFilter اینترفیس است که دارای یک متد accept() است که در پیش از این متد این فایل را بازگشته باشد. این متد به ازای تمامی فایلها داخل دایرکتوری فراخوانی می شود و مقدار boolean بر می گرداند، این مقدار true می شود تا از طریق آن مشخص شده اند، false می شود آیا فایل ورودی، حائز شرایط هست یا خیر. اینترفیس FileFilter که در پکیج java.io قرار دارد را در زیر ملاحظه می کنید.

```
package java.io;

public interface FileFilter {
    boolean accept(File pathname);
}
```

این متد در صورتی که یک فایل حائز شرایط باشد مقدار true بر می گرداند (مسیر آن با مسیر مورد نظر منطبق باشد) و در غیر اینصورت false بر می گرداند.

ایجاد دایرکتوری

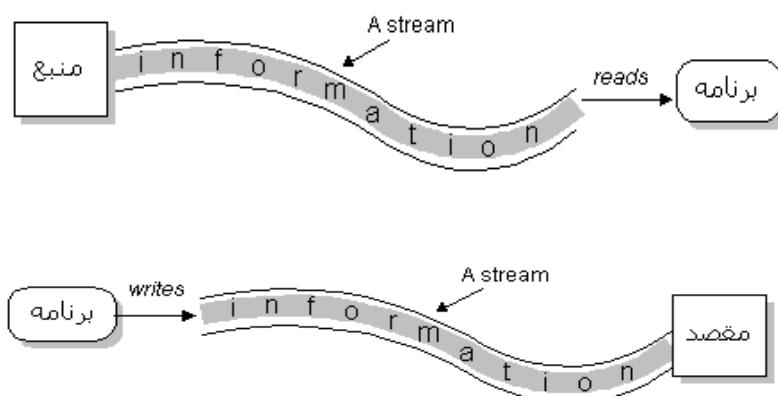
اگر بخواهید از طریق برنامه، روی سیستم یک دایرکتوری ایجاد کنید باید از متد mkdir() که از دیگر متد های کلاس File است استفاده کنید در صورتی که اجرای متد با موفقیت همراه باشد و دایرکتوری ساخته

ورودی و خروجی

شود نتیجه اجرای متدها true خواهد بود در غیر اینصورت اجرای متدهای false بر می گردد در صورتی که دایرکتوری قبل وجود داشته باشد یا مسیر آن دایرکتوری وجود نداشته باشد (دایرکتوری که قرار است دایرکتوری فعلی در آن ساخته شود وجود نداشته باشد) اجرای متدهای موقوفیت همراه نخواهد بود و نتیجه اجرای آن مقدار false تولید خواهد کرد. در صورتی که می خواهید یک دایرکتوری ایجاد کنید که ممکن است مسیر آن وجود نداشته باشد (مثلاً بخواهید یک دایرکتوری با نام newdir به صورت C:/java_programs/src/newdir بسازید و هیچکدام از دایرکتوری های src و makedirs وجود نداشته باشند) به جای () از متدهای mkdir() و java_programs استفاده کنید.

Stream

Stream چیست؟ معنی Stream در لغت به معنای جویبار یا جریان آب است اما از نقطه نظر تکنیکی، Stream به جریانی از داده ها گفته می شود که داده های برنامه توسط آن جریان منتقل می شوند در حقیقت Stream واسطه ای است که از طریق آن ورودی و خروجی داده ها از طریق آن صورت می گیرد Stream شبیه خطوط لوله ای هستند که داده ها را به منابع خارجی ارسال می کنند و یا داده ها را از منابع خارجی دریافت می کنند. از فصل دوم به خاطر بیاورید که با استفاده از جمله System.out.println() جملاتی را در خروجی استاندارد چاپ می کردید، در حقیقت مسئولیت System.out.println() انتقال یک String از برنامه به خروجی استاندارد است این انتقال توسط یک آبجکت Stream صورت می گیرد. هرگاه بخواهید داده هایی را از حافظه سخت (hard disk) به حافظه سیستم منتقل کنید لازم است با استفاده از یک برنامه به صورت بیت به بیت، داده ها را از روی حافظه سخت سیستم خوانده و آنها را به حافظه موقت سیستم منتقل کنید. به ارسال داده ها به منبع خارجی (که توسط Stream صورت می گیرد)، اصطلاحاً نوشتن (write) و به دریافت داده ها از منبع خارجی، اصطلاحاً خواندن (read) گفته می شود.



شکل ۱۶-۱. استفاده از Stream‌ها برای ورودی و خروجی داده‌ها به/از برنامه

در صورتیکه در حین خواندن یا نوشتن Stream خطای رخ دهد، IOException رخ خواهد داد، بنابراین هنگام نوشتن یا خواندن Stream نیاز به بلوکهای try/catch خواهد داشت.

در کلاس `java.lang.System`، دو `Stream` برای ورودی و خروجی استاندارد تعریف شده‌اند. یک `Stream` برای ارسال داده‌ها به خروجی و `System.out` برای دریافت داده‌ها از ورودی استاندارد هستند هرگاه قصد ارسال داده‌هایی را به خروجی استاندارد دارید لازم است داده‌های مورد نظرتان را درون `System.out` بنویسید و هرگاه قصد خواندن داده‌هایی را از ورودی استاندارد دارید لازم است داده‌های مورد نظرتان را از `System.in` بخوانید در مورد این دو آبجکت بعداً صحبت خواهیم کرد.

کلاسها و اینترفیس‌های ورودی و خروجی داده‌ها در جاوا

در جاوا دو نوع Stream تعریف شده است.

- Byte Stream
- Character Stream

به شما کمک می‌کنند تا بایتهایی از داده‌ها را در Stream بنویسید یا از Stream بخوانید به عنوان مثال صوت یا تصویر مجموعه‌ای از بایتهاست، هرگاه بخواهید آنها را از روی حافظه سخت بخوانید و توسط برنامه آنها را پردازش کنید، باید برای انتقال آنها از Byte Stream استفاده کنید. به طور مشابه، به شما کمک می‌کند تا مجموعه‌ای از کاراکترها را از یک منبع خارجی بخوانید یا در Character Stream یک منبع خارجی بنویسید مثلاً هرگاه قصد دارید تا داده‌های کاراکتری یا رشته‌ای را در یک فایل بنویسید یا از یک فایل بخوانید بهتر است از Character Stream ها استفاده کنید. انتقال داده‌ها در اصل توسط بابت صورت می‌گیرد و جدا کردن Byte Stream از Character Stream فقط برای ایجاد سهولت و کارایی بیشتر در کار کردن با کاراکترها است.

کلاس‌های Byte Stream

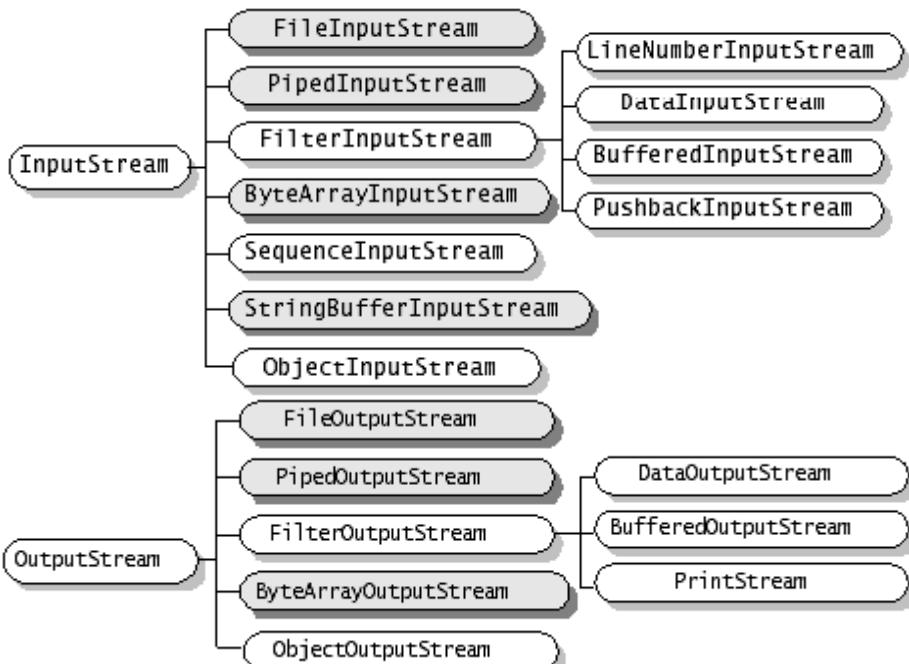
در جاوا دو کلاس `InputStream` و `OutputStream` برای کار کردن با Byte Stream طراحی شده‌اند.

کلاس `InputStream` مسئول انتقال بایتهایی از داده‌ها از منبع خارجی به برنامه است (ورود یا خواندن داده‌ها) و کلاس `OutputStream` مسئول انتقال بایتها از برنامه به منبع خارجی است (خروج یا نوشتن داده‌ها) هر دو کلاس `InputStream` و `OutputStream` abstract، `Inputstream` و `FileInputStream` با نوع خاصی از تعريف شده‌اند که از این دو کلاس مشتق شده‌اند. هر کدام از این کلاسها برای کار کردن با `FileInputStream` طراحی شده بایتها طراحی شده‌اند مثلاً برای خواندن اطلاعات از یک فایل، کلاس `FileInputStream` طراحی شده

ورودی و خروجی

است یا برای نوشتن اطلاعات در یک فایل، کلاس `FileOutputStream` طراحی شده است. از آنجاییکه دو کلاس `InputStream` و `OutputStream` **abstract** هستند، بالطبع متدهایی دارند که توسط کلاسهای فرزند آنها پیاده‌سازی شده اند. با اهمیت‌ترین این متدها، متد `(read)` در کلاس `InputStream` و متد `(write)` در کلاس `OutputStream` هستند. متد `(read)` یک بایت یا آرایه ای از بایت‌ها را از `Stream` می‌خواند و متد `(write)` یک بایت یا آرایه‌ای از بایتها را در `Stream` می‌نویسد.

شکل زیر کلاسهای `InputStream` و `OutputStream` و دیگر کلاسهایی که از این دو کلاس مشتق شده‌اند را نشان می‌دهد.



شکل ۱۷-۲. سلسله کلاسهای `Stream`

کلاس `InputStream`

کلاس `InputStream` که یک کلاس **abstract** است، چگونگی دریافت داده‌های بایتی را تعریف می‌کند آنچه اهمیت دارد منبع داده‌ها نیست مهم دسترسی و در اختیار گذاشتن داده‌هاست. داده‌ها ممکن است از هر منبعی بیایند ممکن است از کی برد، سیستم فایل، یا حتی از وب بیایند این کلاس، به عنوان **superclass** تمام کلاسهای ورودی داده در جاوا محسوب می‌شود (تمام کلاسهایی که کار ورود داده‌ها را به عهده دارند از این کلاس مشتق شده‌اند).

کلاس `InputStream` دارای متدهای متعددی است که بالطبع در تمام کلاسهایی که از این کلاس مشتق شده‌اند وجود دارد مطالعه این متدها به شما در ایجاد `Stream` خواندن و یا پردازش داده‌های `InputStream` کمک می‌کند.

توضیح	نحوه فراخوانی	متدها
این متده که مهمترین متده کلاس <code>InputStream</code> به شمار می‌رود یک <code>byte</code> را می‌خواند، اگر داده‌ای داخل <code>Stream</code> برای خواندن در دسترس نباشد، این متده در حالت انتظار باقی ماند تا داده‌ها از منبع فرا برستند این بدين معنی است که اجرای این متده تا نرسیدن داده‌ها ادامه می‌یابد و اجرای برنامه به خط بعدی منتقل نخواهد شد این متده بایت یا بایتهای مورد نظر را از <code>InputStream</code> می‌خواند و در صورتیکه <code>Stream</code> به آخر رسیده باشد و چیزی برای خواندن وجود نداشته باشد مقدار ۱- برمی‌گرداند. همچنین این متده با خداد خطأ <code>IOException</code> تولید می‌کند.	<code>read()</code> <code>//reads one byte byte abyte = aStream.read();</code> <code>//reads into an //array of bytes byte[] bar=new byte[1024]; aStream.read(bar);</code>	<code>read</code>
این متده تعداد بایتهایی را که در حال حاضر آماده برای خواندن هستند را برمی‌گرداند این متده قابل اعتماد نیست زیرا به محض اینکه تعداد بایتهای آماده را به شما می‌دهد، ممکن است بایتهای دیگری از راه برستند، و تعداد بایتهای آماده تغییر کند.	<code>available()</code>	<code>available</code>
این متده عملیات <code>Stream</code> را خاتمه داده و منابعی که در اختیار <code>Stream</code> قرار دارند را آزاد می‌کند. (به عنوان مثال اگر	<code>close ()</code> <code>// close the stream aStream.close();</code>	<code>close</code>

ورودی و خروجی

<p>با استفاده از یک Stream، محتویات یک فایل را منتقل می کنید با فراخوانی این متده، فایل از حالت بلوکه خارج می شود و برنامه های دیگر می توانند از آن استفاده کنند) بهتر است در پایان کار با Stream، متده close() آنرا فراخوانی کنید تا از به پایان رسیدن موقتیت آمیز عملیات Stream اطمینان یابید و منابع را آزاد کنید.</p> <p>رخداد خطأ در این متده، باعث بروز IOException می شود.</p>		
<p>این متده برای علامتگذاری موقعیت فعلی Stream بکار می رود.</p>	<pre>mark() //marks the 200 th byte aStream.mark(200);</pre>	mark
<p>در صورتیکه Stream ای که در حال mark کار کردن با آن هستیم متده () و () را پشتیبانی کند، markSupported() فراخوانی مقدار true برمی گرداند و در غیر این صورت false برمی گرداند.</p>	<pre>markSupported() //returns true if mark () supported aStream. markSupported();</pre>	markSupported
<p>این متده، موقعیت فعلی را به آخرین موقعیتی که قبلاً با متده mark() مشخص شده است انتقال می دهد رخداد خطأ در این متده، باعث بروز IOException می شود.</p>	<pre>reset(); //resets the last byte position and begin fromthere aStream.reset();</pre>	reset
<p>از تعداد مشخصی از کاراکترها عبور می کند این متده یک عدد int برمی گرداند که مشخص کننده تعداد واقعی کاراکترهایی است که از آنها عبور شده است.</p>	<pre>skip (int n); //skip next 200 //bytes aStream.skip(200);</pre>	skip

جدول ۱۶-۲. متدهای کلاس InputStream

کلاس OutputStream

کلاس OutputStream نیز یک کلاس abstract است، از آنجا که این کلاس، superclass تمام کلاسهای ارسال داده (output stream) می‌باشد، اساس نحوه نوشتن داده‌ها در تمام Stream‌ها را مشخص می‌کند در این کلاس مجموعه‌ای از متدها تعریف شده است که کار ایجاد output stream یا نوشتن و پردازش داده‌ها را در آن بر عهده دارند.

جدول زیر متدهای کلاس output stream را نشان می‌دهد.

توضیح	نحوه فراخوانی	متدها
این متده که مهمترین متده OutputStream کلاس است، یک یا آرایه‌ای از بایت را داخل Stream می‌نویسد. با خداد خطا در عمل نوشتن داده‌ها، این متده IOException می‌کند.	<pre>write() // writes a single byte write(int b); // writes an array of bytes write(byte[] bytes); //writes a particular //section of an array of //bytes write (byte[], int, int);</pre> <p>مثال:</p> <pre>//reads a keyboard aoutput //stream.write (mbyte); //writes it to a stream byte mbyte= System.in.read(); //writes array into a //stream aOutputStream.write(mbyte); //writes portion of an //array aOutputStream.write (byteArray,20,200);</pre>	write

ورودی و خروجی

<p>این متدهای داخل Stream را برای مقصود ارسال می کند تمام داده هایی که داخل Stream با فرایند با فراخوانی این متده به مقصد ارسال می شوند.</p> <p>اگر خطای در زمان ارسال داده ها به مقصد رخ دهد، این متده IOException تولید می کند.</p>	<p><code>flush()</code></p>	<p><code>flush</code></p>
<p>این متده، عملیات Stream را خاتمه می دهد و منابعی که در اختیار Stream قرار دارند را آزاد می کند.</p> <p>با رخداد خطای در این متده، IOException تولید می شود.</p>	<p><code>close()</code></p> <p><code>//closes the stream aOutputStream.close();</code></p>	<p><code>close</code></p>

جدول ۳-۱۶. متدهای کلاس OutputStream

FileInputStream

یکی از کلاسهای Input Stream که برای خواندن بایت‌های یک فایل طراحی شده است کلاس FileInputStream است با استفاده از این کلاس می‌توانید داده‌های یک فایل را خوانده و از آنها در برنامه استفاده کنید.

این کلاس دارای دو سازنده به صورت زیر است.

```
FileInputStream(String filePath)
FileInputStream(File fileObj)
```

مشخص کننده مسیر یک فایل است و `fileObj` آبجکتی از کلاس `File` است که فایل خاصی را مشخص می‌کند در صورتیکه هیچ فایلی با مسیر `filePath` یا با مشخصات `fileObj` وجود نداشته باشد یا به دلیل مسئله حق دسترسی (permission) یا به هر دلیل دیگری، `FileInputStream` نتواند فایل مورد نظر را پیدا کند یا به آن دسترسی پیدا کند، `FileNotFoundException` تولید خواهد شد مثال‌های زیر نحوه بکارگیری هر دو سازنده را نشان می‌دهد.

```
FileInputStream f0 = new FileInputStream("/autoexe.bat");
File f = new File("/autoexe.bat");
FileInputStream f1 = new FileInputStream(f);
```

در مثال زیر نحوه خواندن یک بایت، آرایه ای از یک آرایه از بایت از یک فایل نشان داده شده است، همچنین چگونگی استفاده از متدهای available() برای تعیین تعداد بایتهای باقیمانده نشان داده شده است در این مثال متدهای skip() نیز استفاده شده تا نحوه کردن بایتهای نا خواسته نشان داده شود.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 class FileInputStreamDemo {
6     public static void main(String args[])
7         throws IOException {
8     int size;
9     InputStream f = new FileInputStream(
10 "src/ir/atlassoft/javase/chapter16/FileInputStreamDemo.java");
11
12     System.out.println("Total Available Bytes: " +
13             (size = f.available()));
14     int n = size / 40;
15     System.out.println("First " + n +
16             " bytes of the file one read() at a time");
17
18     for (int i = 0; i < n; i++) {
19         System.out.print((char) f.read());
20     }
21
22     System.out.println("\nStill Available: " +
23 f.available());
24     System.out.println("Reading the next " + n +
25             " with one read(b[])");
26     byte b[] = new byte[n];
27     if (f.read(b) != n) {
28         System.err.println("couldn't read "
```

ورودی و خروجی

```
30                     + n + " bytes.");
31     }
32     System.out.println(new String(b, 0, n));
33     System.out.println("\nStill Available: " +
34         (size = f.available()));
35     System.out.println("Skipping half of remaining" +
36         " bytes with skip()");
37     f.skip(size/2);
38     System.out.println("Still Available: " +
39         f.available());
40     System.out.println("Reading " + n / 2 +
41         " into the end of array");
42     if (f.read(b, n/2, n/2) != n/2) {
43         System.err.println("couldn't read " +
44             n/2 + " bytes.");
45     }
46     System.out.println(new String(b, 0, b.length));
47     System.out.println("\nStill Available: " +
48     f.available());
49     f.close();
50 }
51 }
```

کد ۱۶-۵

خروجی اجرای برنامه فوق به صورت زیر خواهد بود:

```
Total Available Bytes: 1717
First 42 bytes of the file one read() at a time
package ir.atlassoft.javase.chapter16;
```

```
Still Available: 1675
Reading the next 42 with one read(b[])
import java.io.*;

class FileInputStream

Still Available: 1633
Skipping half of remaining bytes with skip()
Still Available: 817
Reading 21 into the end of array
import java.io.*;

println("couldn't read
```

Still Available: 796

FileOutputStream

مشابه کلاس `InputStream` که یک کلاس `FileInputStream` برای خواندن بایتهاي یک فایل است، کلاس `OutputStream` یک کلاس `FileOutputStream` برای نوشتن داده های بایتی داخل یک فایل است.

این کلاس دارای سازنده های زیادی است که برخی از مهمترین آنها عبارتند از:

```
FileOutputStream(String filePath)
FileOutputStream(File fileObj)
FileOutputStream(String filePath, boolean append)
FileOutputStream(File fileObj, Boolean append)
```

فایل را مشخص می کند، `append` نیز مشخص می کند که آیا داده هایی که قرار است در فایل نوشته شوند به داده های قبلی اضافه شوند یا داده های قبلی پاک شده و داده های جدید جایگزین آنها شوند سازنده های فوق خطاهای `FileNotFoundException` یا `SecurityException` را ممکن است تولید کنند.

در صورتیکه بخواهید با استفاده از آبجکتی از این کلاس، داده هایی را داخل یک فایل که روی سیستم وجود ندارد بنویسید، قبل از نوشتن، آن فایل ایجاد خواهد شد همچنین اگر بخواهید داده های بایتی را داخل یک فایل `Read-Only` بنویسید خطای `IOException` رخ خواهد داد.

در مثال زیر استفاده از کلاس `FileOutputStream` نشان داده شده است در این مثال یک رشته از کاراکترها ایجاد می شود و سپس رشته ایجاد شده با استفاده از متده `getBytes()` به آرایه ای از بایتها تبدیل می شود از روی این آرایه، سه فایل `file1.txt`، `file2.txt` و `file3.txt` ساخته شده است.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 class FileOutputStreamDemo {
6
7     public static void main(String args[])
8         throws IOException {
```

ورودی و خروجی

```
9
10     String source = "Now is the time for all good men\n" +
11         " to come to the aid of their country\n"
12         + " and pay their due taxes.";
13
14     byte buf[] = source.getBytes();
15     OutputStream f0 = new FileOutputStream("file1.txt");
16     for (int i=0; i < buf.length; i += 2) {
17         f0.write(buf[i]);
18     }
19     f0.close();
20
21     OutputStream f1 = new FileOutputStream("file2.txt");
22     f1.write(buf);
23     f1.close();
24
25     OutputStream f2 = new FileOutputStream("file3.txt");
26     f2.write(buf,
27             buf.length - buf.length/4,
28             buf.length/4);
29     f2.close();
30 }
31 }
```

۱۶-۴ د

محتويات سه فایل file3.txt و file2.txt و file1.txt به صورت زیر خواهد بود

file1.txt :
Nwi h iefralgo et oet h i ftercuty n a hi u ae.

file2.txt :
Now is the time for all good men to come to the aid of their country
and pay their due taxes.

file3.txt :
nd pay their due taxes.

کلاس ByteArrayInputStream

یکی دیگر از کلاس‌های `InputStream`، کلاس `ByteArrayInputStream` است که برای خواندن آرایه‌ای از بایتها که در حافظه قرار دارند استفاده می‌شود. این کلاس دو سازنده دارد هر کدام از آنها آرایه‌ای از بایتها را به صورت پارامتر دریافت می‌کنند.

```
ByteArrayInputStream(byte[] array)
ByteArrayInputStream(byte[] array, int start, int numBytes)
```

آرایه‌ای از بایت است که منبع داده برای `Stream` محاسبه می‌شود، سازنده دوم از روی `Array` زیرمجموعه‌ای از آرایه‌ها، یک `Stream` می‌سازد این زیرمجموعه از عنصر `start` تا `start+numBytes` را شامل می‌شود.

در مثال زیر یک رشته از کاراکترهای حروف الفبا ساخته شده است سپس از روی بایتهای این رشته، دو `ByteArrayInputStream` ساخته شده است.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 class ByteArrayInputStreamDemo {
6
7     public static void main(String args[])
8         throws IOException {
9
10        String tmp = "abcdefghijklmnopqrstuvwxyz";
11        byte b[] = tmp.getBytes();
12        ByteArrayInputStream input1 = new
13        ByteArrayInputStream(b);
14        ByteArrayInputStream input2 =
15            new ByteArrayInputStream(b, 0, 3);
16    }
17 }
```

کد ۱۶-۷

`input1` حاوی تمام حروف الفباست در حالیکه `input2` فقط شامل سه کاراکتر اول است. `ByteArrayInputStream` دارای دو متده است `reset()` و `mark()` را پیاده سازی کرده است در هر صورت در صورتیکه متده `mark()` فراخوانی نشده باشد متده `reset()` باعث می‌شود تا اشاره گری که موقعیت

ورودی و خروجی

آخرین بایت خوانده از Stream را نگهداری می کند به ابتدای Stream اشاره کند این بدین معناست که خواندن بایتهای Stream از ابتدا صورت خواهد گرفت.

در مثال زیر نحوه بکارگیری متده است به این ترتیب محتوای Stream دوبار خوانده شده است بار اول حروف "abc" خوانده شده و با حروف کوچک الفبای انگلیسی چاپ شده است و بار دیگر حروف "abc" خوانده شده و با حروف بزرگ الفبای انگلیسی چاپ شده است.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 public class ByteArrayInputStreamReset {
6
7     public static void main(String args[])
8             throws IOException {
9
10         String tmp = "abc";
11         byte b[] = tmp.getBytes();
12         ByteArrayInputStream in = new
13             ByteArrayInputStream(b);
14
15         for (int i = 0; i < 2; i++) {
16             int c;
17             while ((c = in.read()) != -1) {
18                 if (i == 0) {
19                     System.out.print((char) c);
20                 } else {
21                     System.out.print(
22                         Character.toUpperCase((char) c));
23                 }
24             }
25             System.out.println();
26             in.reset();
27         }
28     }
29 }
30 }
```

در این مثال ابتدا کاراکترهای داخل Stream خوانده شده و با حروف الفبای انگلیسی چاپ می شود سپس با reset کردن Stream، بار دیگر کاراکترهای داخل آن خوانده شده و با حروف بزرگ الفبای انگلیسی چاپ می شود نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.

```
abc
ABC
```

کلاس ByteArrayOutputStream

کلاس `ByteArrayOutputStream` یک کلاس است که می توان آرایه ای از بایتها را داخل آن نوشت در واقع این کلاس آرایه ای از بایتها را به صورت `stream` در داخل حافظه سیستم نگهداری می کند. این کلاس دارای دو سازنده به صورت زیر است:

```
ByteArrayOutputStream()
ByteArrayOutputStream(int numBytes)
```

در حالت اول، ارسال داده ها در Stream از طریق یک بافر ۳۲ بیتی صورت می گیرد ولی در سازنده دوم اندازه این بافر از طریق پارامتر `numBytes` مشخص می شود. مثال زیر نحوه بکارگیری `ByteArrayOutputStream` را نشان می دهد:

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 class ByteArrayOutputStreamDemo {
6
7     public static void main(String args[])
8         throws IOException {
9         ByteArrayOutputStream f = new ByteArrayOutputStream();
10        String s = "This should end up in the array";
11        byte buf[] = s.getBytes();
12
13        f.write(buf);
14        System.out.println("Buffer as a string");
15        System.out.println(f.toString());
16        System.out.println("Into array");
17        byte b[] = f.toByteArray();
```

ورودی و خروجی

```
18     for (int i=0; i<b.length; i++) {
19         System.out.print((char) b[i]);
20     }
21     System.out.println("\nTo an OutputStream()");
22     OutputStream f2 = new
23             FileOutputStream("test.txt");
24
25     f.writeTo(f2);
26     f2.close();
27     System.out.println("Doing a reset");
28     f.reset();
29     for (int i=0; i<3; i++)
30         f.write('X');
31     System.out.println(f.toString());
32 }
33 }
```

کد ۱۶-۹

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود توجه داشته باشید که بعد از فراخوانی متد () و reset() نوشتن سه X در Stream باعث پاک شدن بقیه جملات شده است.

```
Buffer as a string
This should end up in the array
Into array
This should end up in the array
To an OutputStream()
Doing a reset
XXX
```

در این مثال از متد () writeTo استفاده شده تا محتویات byteArrayOutputStream منقل شود. فایل test.txt بعد از اجرای برنامه محتوی رشته زیر خواهد بود.

This should end up in the array

(Buffered Byte Streams) بافر شده Stream

یک Stream است که دارای یک بافر حافظه است بافر، قسمتی از حافظه است که وجود آن باعث می شود تا عملیات ورودی/خروجی بیش از یک بیت در هر لحظه امکانپذیر باشد و به این ترتیب باعث افزایش کارایی می شود.

BufferedInputStream

بافر کردن داده های ورودی/خروجی یکی از روشهای رایج در افزایش کارایی است. کلاس BufferedInputStream امکان می دهد تا داده های یک InputStream داخل بافر قرار گرفته و باعث افزایش کارایی InputStream شود. کلاس BufferedInputStream دارای دو سازنده به صورت زیر است:

```
BufferedInputStream(InputStream inputStream)
BufferedInputStream(InputStream inputStream, int bufferSize)
```

اولین سازنده این کلاس از روی یک InputStream یک BufferedStream با اندازه پیش فرض ۲۰۴۸ بایت می سازد. در سازنده دوم، اندازه بافر از طریق پارامتر bufferSize مشخص می شود هر چه طول بافر بیش تر باشد کارایی BufferedStream بیشتر می شود، اما حداکثر اندازه بافر به سیستم عامل، حافظه در اختیار Virtual Machine و پیکر بندی سیستم بستگی دارد. در حالت کلی ۸۱۹۲ بایت، اندازه مناسبی به نظر می رسد، وقتی از BufferedInputStream برای خواندن داده ها استفاده می شود دادهایی که از منبع خارجی (دیسک سخت، شبکه یا...) خوانده می شوند در بافر ذخیره شده و در اینصورت حتی اگر داده ها را به صورت بایت به بایت از Stream بخوانید کارایی برنامه به نسبت بسیار بیشتر از وقتی خواهد بود که از بافر استفاده نمی کنید.

فایده دیگر بافر کردن داده ها این است که می توان روی داده های داخل Stream حرکت کرد یعنی می توان داده های قبلی را باز دیگر از Stream خواند.

در کلاس BufferedInputStream علاوه بر متدهای () و () skip() read()، متند () mark() نیز پیاده سازی شده است.

مثال زیر نحوه بکارگیری متدهای () و () mark() را نشان می دهد در این مثال، در یک رشته، کلمه "copy;" جستجو شده و با © جایگزین شده است.

```

1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 class BufferedInputStreamDemo {
6
7     public static void main(String args[])
8         throws IOException {
9
10        String s = "This is a &copy; copyright symbol " +
11          "but this is &copy not.\n";

```

```
12
13     byte buf[] = s.getBytes();
14     ByteArrayInputStream in =
15         new ByteArrayInputStream(buf);
16     BufferedInputStream f =
17         new BufferedInputStream(in);
18     int c;
19     boolean marked = false;
20
21     while ((c = f.read()) != -1) {
22         switch (c) {
23             case '&':
24                 if (!marked) {
25                     f.mark(32);
26                     marked = true;
27                 } else {
28                     marked = false;
29                 }
30                 break;
31             case ';':
32                 if (marked) {
33                     marked = false;
34                     System.out.print("(c)");
35                 } else
36                     System.out.print((char) c);
37                 break;
38             case ' ':
39                 if (marked) {
40                     marked = false;
41                     f.reset();
42                     System.out.print("&");
43                 } else
44                     System.out.print((char) c);
45                 break;
46             default:
47                 if (!marked)
48                     System.out.print((char) c);
49                 break;
50             }
51         }
```

```

52     }
53 }
```

کد ۱۰-۱۶

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود

```
This is a (c) copyright symbol but this is &copy not.
```

BufferedOutputStream

کلاس `BufferedOutputStream` که خود یک `OutputStream` است دارای یک بافر داده است تا داده هایی را که داخل آن نوشته می شوند در آن نگهداری کند به این ترتیب این کلاس با کاهش تعداد دفعاتی که داده ها در منبع خارجی نوشته می شوند باعث افزایش کارایی می شود با فراخوانی متده (`flush()`) از این کلاس می توانید تمام داده هایی که در بافر قرار دارند را به منبع خروجی ارسال کنید این کلاس دارای دو سازنده به صورت زیر است:

```
BufferedOutputStream(OutputStream outputStream)
BufferedOutputStream(OutputStream outputStream, int bufferSize)
```

سازنده اول تنها با استفاده از یک `OutputStream` با فری با `BufferedOutputStream` طول پیش فرض ۵۱۲ بایت می سازد. اما سازنده دوم علاوه بر `OutputStream`، اندازه بافر را نیز دریافت می کند.

DataOutputStream و DataInputStream

این دو کلاس امکان می دهند تا داده های اولیه ۱ را در یک `Stream` بنویسید یا از یک `Stream` بخوانید این کلاسها متدهایی برای تبدیل داده های اولیه به بایت یا بالعکس دارا هستند و کار ذخیره سازی داده های باینری (از قبیل اعداد صحیح و اعداد اعشاری) را در فایل آسان می کنند. کلاس `DataOutputStream` دارای سازنده ای به صورت زیر است.

```
DataOutputStream(OutputStream outputStream)
```

این `Stream` است که قرار است داده ها در آن نوشته شوند.

ورودی و خروجی

در کلاس DataOutputStream متدهایی تعریف شده است که از طریق آن می‌توانید داده‌های نوع اولیه را به دنباله‌ای از بایتها تبدیل کرده و داخل Stream بنویسید. در زیر نمونه‌ای از این متدها را ملاحظه می‌کنید.

```
final void writeDouble(double value) throws IOException
final void writeBoolean(boolean value) throws IOException
final void writeInt(int value) throws IOException
```

در متدهای فوق، value همان مقداری است که در Stream نوشته می‌شود. کلاس نقطه مقابل کلاس DataOutputStream است این کلاس فقط یک سازنده به صورت زیر دارد.

```
DataInputStream(InputStream inputStream)
```

مشخص کننده یک Stream است که داده‌ها از آن خوانده می‌شوند. در این کلاس نیز همانند DataOutputStream متدهایی تعریف شده است که بوسیله آن می‌توانید بایتها را به مقدار معادل آنها از انواع اولیه تبدیل کنید در زیر نمونه‌هایی از این متدها را ملاحظه می‌کنید:

```
final double readDouble() throws IOException
final boolean readBoolean() throws IOException
final int readInt() throws IOException
```

مثال زیر نحوه بکارگیری DataOutputStream و DataInputStream را نشان می‌دهد:

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 public class DataIODemo {
6
7     public static void main(String args[])
8         throws IOException {
9
10        FileOutputStream fout = new
11                  FileOutputStream("Test.dat");
12        DataOutputStream out = new DataOutputStream(fout);
13
14        out.writeDouble(98.6);
```

```

15         out.writeInt(1000);
16         out.writeBoolean(true);
17
18         out.close();
19
20         FileInputStream fin = new
21                 FileInputStream("Test.dat");
22         DataInputStream in = new DataInputStream(fin);
23
24         double d = in.readDouble();
25         int i = in.readInt();
26         boolean b = in.readBoolean();
27
28         System.out.println("Here are the values: " +
29                             d + " " + i + " " + b);
30
31         in.close();
32     }
33 }
```

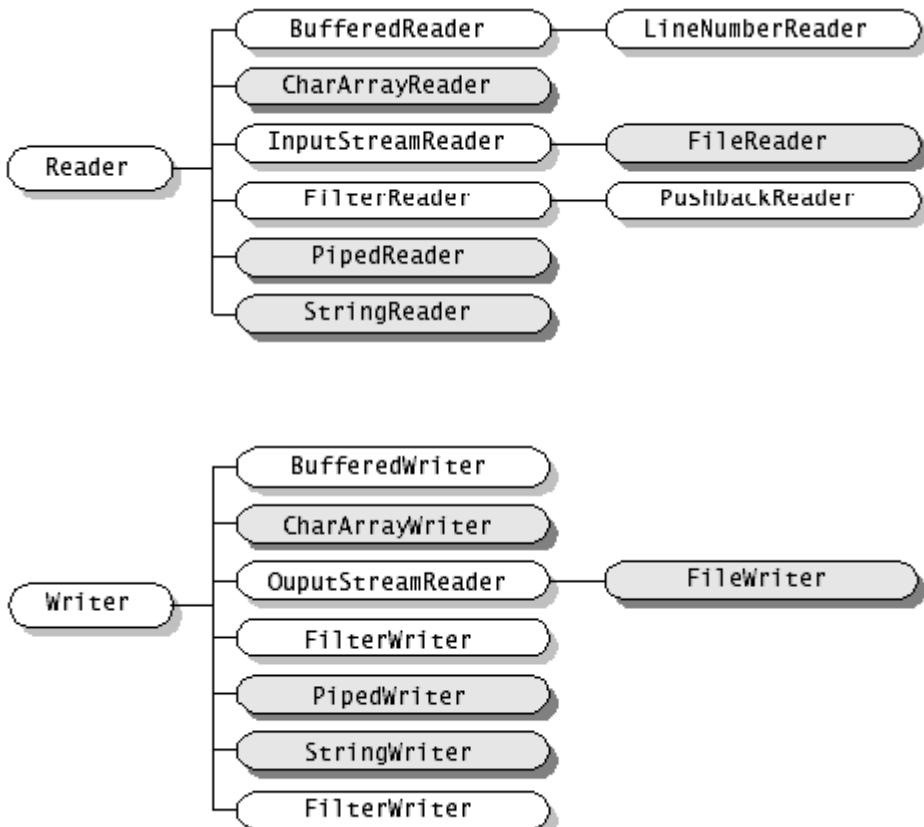
کد ۱۶-۱۱

کلاس‌های Character Stream

در قسمت قبل با کلاس‌های Byte Stream آشنا شدید این کلاسها انتقال داده‌های بایتی را به عهده دارند و نمی‌توانند مستقیماً با کاراکترهای Unicode کار کنند در مقابل Character Stream ها امکان ورودی و خروجی داده‌های کاراکتری را فراهم می‌کنند.

برای Stream‌های کاراکتری، دو کلاس abstract Reader و Writer به نامهای Reader و Writer طراحی و پیاده سازی شده است. تمام کلاس‌های Stream کاراکتری از این دو کلاس مشتق شده اند شکل زیر این کلاسها را نشان می‌دهد.

ورودی و خروجی



شکل ۳-۱۶. سلسله کلاس‌های Character Stream

Reader

که یک کلاس abstract است، برای خواندن داده‌های کاراکتری تعریف شده است در صورتیکه با فرآخوانی هریک از این متدها، خطایی رخ دهد، این متدها IOException تولید می‌کنند جدول زیر عملکرد متدهای کلاس Reader را نشان می‌دهد:

توضیح	نحوه فرآخوانی	متدها
این متده که مهمترین متده کلاس Reader به شمار می‌رود یک یا چندین کاراکتر از داده‌های داخل Stream را می‌خواند، اگر داده‌ای داخل Stream برای خواندن در دسترس نباشد، این متده در حالت انتظار	read() مثال: <pre>//reads one char char aChar = aStream.read();</pre>	read

<p>باقي ماند تا داده‌ها از منبع فرا بررسند این بدین معنی است که اجرای این متده ادامه می‌یابد و تا نرسیدن داده‌ها اجرای برنامه به خط بعدی منتقل نخواهد شد این متده بایت Reader باشته مورد نظر را از می‌خواند و در صورتیکه Stream به آخر رسیده باشد و چیزی برای خواندن وجود نداشته باشد مقدار ۱- برمی‌گردداند. همچنین رخداد خطأ در این متده باعث تولید IOException می‌شود.</p>	<pre>// reads data into //an array of chars char[] carr= new char[1024]; aStream.read(carr);</pre>	
<p>این متده تعداد کاراکترهایی را که در حال حاضر آماده برای خواندن هستند را برمی‌گرداند. اما این متده، قابل اعتماد نیست زیرا به محض اینکه تعداد کاراکترهای آماده را به شما می‌دهد، ممکن است داده‌های دیگری از راه بررسند، و تعداد کاراکترهای آماده تغییر کند.</p>	<p>available()</p>	<p>available</p>
<p>عملیات Stream را خاتمه داده و منابعی که در اختیار Stream قرار دارند را آزاد می‌کند. (به عنوان مثال اگر با استفاده از یک Stream، محتویات یک فایل را منتقل می‌کنید با فراخوانی متده close()، فایل از حالت بلوکه خارج می‌شود تا دیگر برنامه بتوانند از آن استفاده کنند. بهتر است در پایان عملیات با close()، متده Stream آنرا فراخوانی کنید تا از به پایان رسیدن موفقیت‌آمیز عملیات Stream اطمینان یافته و منابع را آزاد کنید رخداد خطأ در این متده باعث بروز IOException می‌شود.</p>	<p>close()</p> <p>مثال:</p> <pre>// close the stream aStream.close();</pre>	<p>close</p>
<p>این متده برای علامتگذاری موقعیت فعلی بکار می‌رود.</p>	<p>mark()</p> <p>مثال:</p>	<p>mark</p>

ورودی و خروجی

	//marks the 200 th //byte aStream.mark(200);	
در صورتیکه Stream ای که در حال کارکردن با آن هستیم متدهای mark() و reset() را پشتیبانی کند، فراخوانی true مقدار markSupported() بر می‌گرداند و در غیر این صورت false بر می‌گرداند.	markSupported () //returns true if //mark () supported aStream. markSupported();	markSupported
در صورتیکه داده خوانده نشده در Stream موجود باشد فراخوانی این متدهای مقدار true و در غیر این صورت false بر می‌گرداند.	ready () مثال: aStream.ready();	ready
این متدهای موقعيت فعلی را به آخرین موقعيتی که قبلاً با متدهای mark() مشخص شده است انتقال می‌دهد و خطا در این متدها باعث بروز IOException می‌شود.	reset (); مثال: // resets the last //char position and //begin from there aStream.reset();	reset
اشاره گری که به کاراکتر خوانده نشده اشاره می‌کند از تعداد مشخصی از کاراکترها عبور می‌کند (یا به عبارتی صرفنظر می‌کند) این متدهای int skip() بر می‌گرداند که مشخص کننده تعداد واقعی کاراکترهایی است که از آنها عبور شده است.	skip (int n); مثال: //skip next 200 //bytes aStream.skip(200);	skip

جدول ۱۶-۴. متدهای کلاس Reader

Writer

این کلاس، که یک کلاس abstract است برای نوشتندادهای کاراکتری درون یک Stream تعریف شده است. متدهای این کلاس که به صورت abstract تعریف شده اند عملیات نوشتندادهای کاراکتری را درون یک Stream به عهده دارند همگی مقدار برگشتی void دارند هرگونه خطا در این متدها باعث بروز IOException می‌شود.

متدهای کلاس	نحوه فراخوانی	توضیح
-------------	---------------	-------

<p>این متده که مهمترین متده است یک کلاس Writer است که یک بایت یا آرایه ای از بایت را داخل Stream می نویسد. بار خداد خطأ در این متده IOException تولید می شود.</p>	<pre>write() // writes a single //char write(int ch); // writes an array //of chars write(char[] chars); //writes a particular //section of an array //of chars write (char[] , int, int)</pre>	<p>write</p>
	<p>مثال:</p> <pre>//reads a keyboard //output //stream.write (mchar); //writes it to a stream char mchar= System.in.read(); aOutputStream.write (mchar); //writes array into a //stream aOutputStream.write (array); //writes portion of an //array aOutputStream.write (array,20,200);</pre>	
<p>کاراکتر ch را به انتهای Stream اضافه می کند.</p>	<p>append(ch)</p>	<p>append</p>
<p>این متده داده های داخل Stream را ارسال می کند تمام داده هایی که داخل بافر شده اند با فراخوانی این متده به مقصد ارسال می شوند با</p>	<p>flush()</p>	<p>flush</p>

ورودی و خروجی

رخداد خطأ در این متده است، IOException تولید می شود.		
این متده، عملیات Stream را خاتمه می دهد و منابعی که در اختیار Stream قرار دارند را آزاد می کند. بار خداد خطأ در این متده، IOException تولید می شود.	close() مثال: //closes the stream aOutputStream.close();	close

جدول ۱۶-۵. متدهای کلاس Writer

FileReader

از این کلاس می توان برای خواندن داده های کاراکتری که داخل یک فایل متنی قرار دارند استفاده کرد دو سازنده این کلاس عبارتند از :

```
FileReader(String filePath)  
FileReader(File fileObj)
```

رخداد خطأ در هر کدام از این سازنده ها، باعث بوجود آمدن FileNotFoundException می شود
مسیر کامل یک فایل است و filePath یک آبجکت از کلاس File است که مشخص کننده یک فایل روی سیستم است.

مثال زیر نحوه خواندن تمام خطوط متنی یک فایل را نشان می دهد و سپس محتويات خوانده شده در خروجی استاندارد چاپ می شود در این برنامه، سورس همین کلاس که در همین دایرکتوری قرار دارد خوانده شده و به خروجی استاندارد می رود.

```
1 package ir.atlassoft.javase.chapter16;  
2  
3 import java.io.*;  
4  
5 public class FileReaderDemo {  
6  
7     public static void main(String args[]) throws IOException {  
8         FileReader fr = new FileReader("FileReaderDemo.java");  
9         BufferedReader br = new BufferedReader(fr);  
10    }
```

```

11         String s;
12
13     while ((s = br.readLine()) != null) {
14         System.out.println(s);
15     }
16
17     fr.close();
18 }
19 }
```

کد ۱۷-۱۳

FileWriter

از این کلاس برای نوشتن داده های کاراکتری داخل یک فایل استفاده می شود رایج ترین سازنده های آن به قرار زیر است:

```

FileWriter(String filePath)
FileWriter(String filePath, boolean append)
FileWriter(File fileObj)
FileWriter(File fileObj, boolean append)
```

تمام این سازنده ها، `IOException` تولید می کنند، `filePath` مسیر کامل یک فایل را مشخص می کند و `fileObj` مشخص کننده یک آبجکت از کلاس `File` است که مشخص کننده یک فایل روی سیستم است. اگر مقدار `true`، `append` باشد داده هایی که داخل فایل نوشته می شوند به انتهای فایل اضافه می شوند در غیراینصورت داده های جدید، جایگزین داده های قبلی داخل فایل می شوند در صورتیکه فایل قبلا وجود نداشته باشد `FileWriter` فایل جدیدی ایجاد می کند در صورتیکه بخواهید داخل یک فایل `Read-Only`، داده هایی را بنویسید خطای `IOException` ایجاد خواهد شد. در مثال زیر ابتدا یک رشته کاراکتری تعریف شده، سپس با استفاده از متده `write()`، کاراکترهای رشته دریافت شده و با استفاده از کلاس `FileWriter` درون چندین فایل متنی نوشته شده است. فقط حاوی کاراکترهای زوج رشته است. `file1.txt` تمام کاراکترهای رشته است و `file3.txt` فقط ۱/۴ انتهایی کاراکترهای رشته را در بر می گیرد.

```

1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
```

ورودی و خروجی

```
5  public class FileWriterDemo {  
6  
7      public static void main(String args[])  
8          throws IOException {  
9  
10         String source = "Now is the time for \n" +  
11             " all good men to come to \n" +  
12             " the aid of their country\n" +  
13             " and pay their due taxes.";  
14  
15         char buffer[] = new char[source.length()];  
16         source.getChars(0, source.length(),  
17                         buffer, 0);  
18  
19         FileWriter f0 = new FileWriter("file1.txt");  
20         for (int i=0; i < buffer.length; i += 2) {  
21             f0.write(buffer[i]);  
22         }  
23  
24         f0.close();  
25  
26         FileWriter f1 = new FileWriter("file2.txt");  
27         f1.write(buffer);  
28         f1.close();  
29  
30         FileWriter f2 = new FileWriter("file3.txt");  
31  
32         f2.write(buffer,  
33                     buffer.length-buffer.length/4,  
34                     buffer.length/4);  
35         f2.close();  
36     }  
37 }
```

۱۷-۱۸ آن

CharArrayReader

از این کلاس برای خواندن آرایه‌ای از کاراکترها که در حافظه قرار دارند استفاده می‌شود این کلاس دارای دو سازنده است هر کدام از آنها آرایه‌ای از کاراکترها را به عنوان پارامتر دریافت می‌کند این آرایه به عنوان منبع داده هاست.

```
CharArrayReader(char[] array)
CharArrayReader(char[] array, int start, int numChars)
```

سازنده دوم علاوه بر آرایه‌ای از کاراکترها، دو عدد int دریافت می‌کند که زیرمجموعه‌ای از آرایه را مشخص می‌کند این زیرمجموعه، کاراکترهایی را از عنصر start تا start+numChars را از قرار دارند را شامل می‌شود.

مثال زیر، بکارگیری CharArrayReader را نشان می‌دهد.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 public class CharArrayReaderDemo {
6
7     public static void main(String args[])
8         throws IOException {
9
10        String tmp = "abcdefghijklmnopqrstuvwxyz";
11        int length = tmp.length();
12        char c[] = new char[length];
13
14        tmp.getChars(0, length, c, 0);
15        CharArrayReader input1 = new CharArrayReader(c);
16        CharArrayReader input2 =
17            new CharArrayReader(c, 0, 5);
18
19        int i;
20        System.out.println("input1 is:");
21        while ((i = input1.read()) != -1) {
22            System.out.print((char) i);
23        }
24        System.out.println();
```

ورودی و خروجی

```
25
26     System.out.println("input2 is:");
27     while ((i = input2.read()) != -1) {
28         System.out.print((char) i);
29     }
30     System.out.println();
31 }
32 }
```

۱۶-۱۴ کد

در این مثال، input1 شامل حروف کوچک الفبای انگلیسی است در حالیکه input2 تنها شامل ۵ حرف اول حروف الفباست خروجی اجرای برنامه فوق به صورت زیر است.

```
input1 is:
abcdefghijklmnopqrstuvwxyz
input2 is:
abcde
```

CharArrayWriter

از این کلاس برای نوشتن آرایه ای از کاراکترها در حافظه استفاده می شود این کلاس دارای دو سازنده به صورت زیر است.

```
CharArrayWriter()
CharArrayWriter(int numChars)
```

در سازنده اول، طول بافر مربوطه با مقدار پیش فرض ساخته می شود اما در حالت دوم طول این بافر توسط numChars مشخص می شود. مثال زیر نحوه بکارگیری کلاس CharArrayWriter را نشان می دهد قلا این برنامه را با استفاده از ByteArrayOutputStream نوشته ایم.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 public class CharArrayWriterDemo {
6
7     public static void main(String args[])
8         throws IOException {
9
10        CharArrayWriter f = new CharArrayWriter();
```

```

11      String s = "This should end up in the array";
12      char buf[] = new char[s.length()];
13
14      s.getChars(0, s.length(), buf, 0);
15      f.write(buf);
16      System.out.println("Buffer as a string");
17      System.out.println(f.toString());
18      System.out.println("Into array");
19
20      char c[] = f.toCharArray();
21      for (int i=0; i<c.length; i++) {
22          System.out.print(c[i]);
23      }
24
25      System.out.println("\nTo a FileWriter()");
26      FileWriter f2 = new FileWriter("test.txt");
27      f.writeTo(f2);
28      f2.close();
29      System.out.println("Doing a reset");
30      f.reset();
31      for (int i=0; i<3; i++)
32          f.write('X');
33      System.out.println(f.toString());
34  }
35 }
```

کد ۱۶-۱۵

BufferedReader

این کلاس با بافر کردن داده ها باعث افزایش کارایی می شود سازنده های این کلاس عبارتند از:

```
BufferedReader(Reader reader)
BufferedReader(Reader reader, int bufferSize)
```

اولین سازنده این کلاس از روش یک آبجکت Reader می سازد طول بافر این آبجکت دارای مقدار پیش فرض ۸۱۹۲ کاراکتر است اما دومین سازنده، آبجکت BufferedReader را با طول بافر bufferSize می سازد، از آنجاییکه () در این کلاس پشتیبانی می شود فرآخوانی

ورودی و خروجی

متدها markSupported() بر می‌گرداند با استفاده از متدها mark() و true مقدار روی کاراکترهای داخل بافر حرکت کرده، به عقب برگردید.

مثال زیر همانند مثال BufferedReader است اما به جای آن از BufferedInputStream استفاده شده است در این مثال رشته‌ای از کاراکترهای HTML پردازش شده و کلمه © با جایگزین شده است. خروجی این برنامه شبیه حالتی است که قبلاً در BufferedReader ملاحظه کردید.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 class BufferedReaderDemo {
6
7     public static void main(String args[])
8         throws IOException {
9
10        String s = "This is a &copy; copyright symbol " +
11            "but this is &copy not.\n";
12
13        char buf[] = new char[s.length()];
14        s.getChars(0, s.length(), buf, 0);
15        CharArrayReader in = new CharArrayReader(buf);
16        BufferedReader f = new BufferedReader(in);
17
18        int c;
19        boolean marked = false;
20
21        while ((c = f.read()) != -1) {
22            switch (c) {
23                case '&':
24                    if (!marked) {
25                        f.mark(32);
26                        marked = true;
27                    } else {
28                        marked = false;
29                    }
30                    break;
31                case ';':
```

```

32             if (marked) {
33                 marked = false;
34                 System.out.print(" (c) ");
35             } else
36                 System.out.print((char) c);
37             break;
38         case ' ':
39             if (marked) {
40                 marked = false;
41                 f.reset();
42                 System.out.print(" & ");
43             } else
44                 System.out.print((char) c);
45             break;
46         default:
47             if (!marked)
48                 System.out.print((char) c);
49             break;
50         }
51     }
52 }
53 }
```

کد ۱۶-۱۷

BufferedWriter

از این کلاس برای بافر کردن داده های کاراکتری که در یک منبع خاص نوشته می شوند استفاده می شود. این کلاس دارای متد `flush()` است که با فراخوانی آن، داده های داخل بافر به منبع خروجی ارسال شده و بافر خالی می شود.

از آنجائیکه این کلاس باعث کاهش دفعات نوشتن داده در `Stream` می شود استفاده از آن باعث افزایش کارایی می شود. `BufferedWriter` دارای دو سازنده به صورت زیر است:

```
BufferedWriter(Writer writer)
BufferedWriter(Writer writer, int bufSize)
```

اولین سازنده، از روی یک `Stream` با طول بافر پیش فرض ۸۱۹۲ می سازد اما در دومی طول بافر از طریق پارامتر `bufSize` مشخص می شود.

استفاده از ورودی و خروجی Stream

در زیر مثالی نشان داده شده که در آن از کلاسها و متدهای Stream های کاراکتری استفاده شده است. در این مثال تعداد کاراکترها، تعداد کلمات و تعداد خطوط موجود در یک فایل متنی، محاسبه شده و در خروجی استاندارد چاپ می شود در صورتیکه مسیر یک یا چندین فایل را به عنوان پارامترهای اجرای برنامه مشخص کنیم محاسبات کاراکتری را روی محتویات آن فایل ها انجام می دهد در غیر اینصورت (در صورتیکه مسیر هیچ فایلی را مشخص نکنید) محاسبات روی کلماتی که از ورودی استاندارد وارد می شوند صورت می گیرد.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4
5 public class WordCount {
6     public static int words = 0;
7     public static int lines = 0;
8     public static int chars = 0;
9
10    public static void wc(InputStreamReader isr)
11        throws IOException {
12        int c = 0;
13        boolean lastWhite = true;
14        String whiteSpace = "\t\n\r";
15
16        while ((c = isr.read()) != -1) {
17
18            // Count characters
19            chars++;
20
21            // Count lines
22            if (c == '\n') {
23                lines++;
24            }
25
26            //Count words by detecting the start of word
27            int index = whiteSpace.indexOf(c);
28            if (index == -1) {
29                if (lastWhite == true) {
```

```

30                     ++words;
31                 }
32             lastWhite = false;
33         } else {
34             lastWhite = true;
35         }
36     }
37     if (chars != 0) {
38         ++lines;
39     }
40 }
41
42 public static void main(String args[]) {
43     FileReader fr;
44     try {
45         if (args.length == 0) {
46             // We're working with stdin
47             wc(new InputStreamReader(System.in));
48         } else {
49             // We're working with a list of files
50             for (int i = 0; i < args.length; i++) {
51                 fr = new FileReader(args[i]);
52                 wc(fr);
53             }
54         }
55     } catch (IOException e) {
56         return;
57     }
58     System.out.println(lines + " " + words
59                         + " " + chars);
60 }
61
62 }
```

کد ۱۷-۱۸

وقتی این کلاس بدون هیچ آرگومانی اجرا شود با استفاده از `System.in` یک `InputStreamReader` ساخته می شود به این ترتیب ورودی استاندارد به عنوان منبع ورودی کاراکترها در نظر گرفته خواهد شد در صورتیکه مسیر یک یا چندین فایل به عنوان آرگومان مشخص شود، به ازای هر یک از فایلها، یک `FileReader` ساخته شده و محتویات کاراکتری هر یک از آن فایلها به عنوان

ورودی و خروجی

منبع ورودی کاراکترها در نظر گرفته خواهد شد. قلی از پایان برنامه، مقادیر محاسبه شده در خروجی استاندارد چاپ خواهد شد.

Serialization

به فرایندی گفته می شود که طی آن یک آبجکت به Stream تبدیل می شود در واقع Serialization به نوشتن یک آبجکت در یک Stream گفته می شود. از آنجائیکه Stream را می توان روی شبکه منتقل کرد، یا آنرا درون یک فایل روی دیسک سخت ذخیره نمود با استفاده از Serialization می توانید یک آبجکت را روی شبکه منتقل نموده با روی دیسک سخت ذخیره کنید.

توجه: RMI یکی از تکنولوژی های جاواست که از طریق آن، متدهای یک آبجکت که روی یک ماشین قرار دارد توسط آبجکت دیگر که روی ماشین دیگری قرار دارد فراخوانی می شود برای فراخوانی یک متدهای آن متد نیز به آن ارسال شود این پارامترها که به صورت آبجکت هستند از روی شبکه عبور می کنند تا به آبجکت دیگر برسند، بنابراین باید آبجکت های پارامتر را به Stream تبدیل نموده و در مقصد مجدد Stream را به آبجکت تبدیل کنیم.

اینترفیس Serializable

برای اینکه بتوان یک آبجکت را Serialize کرد باید این کلاس آن آبجکت اینترفیس Serializable را پیاده سازی کرده باشد. اینترفیس Serializable دارای هیچ property یا متدهای نیست، وقتی یک کلاس این اینترفیس را پیاده سازی می کند، در هنگام تولید آن آبجکت از روی آن کلاس، JRE آبجکت را به گونه ای می سازد که امکان ذخیره شدن روی دیسک سخت یا عبور از شبکه را داشته باشد. اگر یک کلاس Serializable باشد تمام کلاس هایی که از آن کلاس مشتق می شوند نیز Serializable خواهند بود.

ذخیره کردن آبجکتها در فایل

نوشتن آبجکت ها در Stream کار ساده‌ای است برای مثال در کد زیر، یک String و یک آبجکت از کلاس Date که درون یک Stream نوشته شده و Stream تولید شده به یک فایل ارسال شده است.

(توجه داشته باشید که هر گاه از طریق جمله

```
new Date();
```

از کلاس Date آبجکتی ساخته شود آن آبجکت معرف همان زمان و تاریخی است که آبجکت ساخته شده است).

```
FileOutputStream fileout=new FileOutputStream("D:/timefile");
ObjectOutputStream object = new ObjectOutputStream(out);
object.writeObject ("Today");
object.writeObject (new Date());
object.close();
fileOut.close();
```

در مثال فوق، ابتدا یک آبجکت از `FileOutputStream` ساخته شده است (هرگاه بخواهیم داده‌هایی را در یک فایل بنویسیم از این `Stream` استفاده می‌کنیم) در سازنده این کلاس رشته "D:/timefile" نوشته شده، این رشته مشخص کننده مسیر فایلی است که قرار است داده‌ها در آن نوشته شود. هر داده‌ای که در این `Stream` نوشته شود، به فایل مشخص شده ارسال و در آن نوشته خواهد شد.

سپس آبجکتی از `ObjectOutputStream` ساخته شده است هرگاه بخواهیم یک آبجکت را به صورت `Stream` درآوریم از این `Stream` استفاده می‌شود در سازنده این کلاس، `fileout` نوشته شده است. معنی آن این است که هر آبجکتی که در این `ObjectOutputStream` نوشته شود به `fileout` منتقل شده و از آنجا نیز به نوبه خود به فایل منتقل شود در خطوط بعدی، به ترتیب یک رشته و یک آبجکت از کلاس `Date` در `java.util.ObjectOutputStream` نوشته شده‌اند.

در انتهای، درست قبل از فراخوانی متدهای `close()` از `Stream`ها، متدهای `flush()` مربوط به فراخوانی شده است تا آخرین داده‌های درون `Stream` در فایل نوشته شوند در پایان، متدهای `close()` مربوط به دو `Stream` به ترتیب از آخرین `Stream` تولید شده به اولین `Stream` فراخوانی شده است.

کلاس `ObjectOutputStream` دارای متدهای دیگری از قبیل `writeUTF()` و `writeInt()` است که می‌توانند از آنها برای نوشتن داده‌های نوع اولیه در `Stream` استفاده کنید.

در صورتیکه آبجکتی که داخل `ObjectOutputStream` می‌نویسید `Serializable` نباشد، فراخوانی متدهای `writeObject()` باعث رخداد خطای `NotSerializableException` خواهد شد بنابراین فقط آبجکت‌هایی که اینترفیس `Serializable` را پیاده‌سازی کرده باشند را می‌توانید داخل `ObjectOutputStream` بنویسید.

بارگذاری آبجکت ذخیره شده در فایل

حال که آبجکت خود را به `Stream` تبدیل کرده‌اید بی‌شک مایلید بدانید چگونه می‌توانید با استفاده از `Stream` بار دیگر آبجکت خود را بازسازی کنید.

در زیر تکه کدی را ملاحظه می‌کنید که از روی فایل `timefile` (که قبلًا آبجکتها را در آن ذخیره کردید) آبجکت‌هایتان را مجددًا بازسازی می‌کند:

```
InputStream infile = new FileInputStream("d:/timefile");
ObjectInputStream inObject = new ObjectInputStream (infile);
String today=(String) inObject.readObject();
```

ورودی و خروجی

```
Date date= (Date) inobject.readObject ();
```

در این مثال، یک آبجکت از روی `FileInputStream` ساخته شده است و در سازنده آن، `D:/timefile` مشخص شده است این بدین معناست که می خواهیم محتویات فایل `D:/timefile` را از طریق `Stream` بخوانیم.

سپس یک آبجکت از روی `ObjectInputStream` ساخته شده است و در سازنده آن `infile` مشخص شده، این بدین معناست که می خواهیم از روی محتویات یک فایل، آبجکتها را بازسازی کیم بعد از آن، متده `readObject()` فراخوانی شده است این متده از روی `Stream` ساخته شده، آبجکتی را بازسازی کرده و برمه گرداند توجه کنید که مقدار برگشتی این متده `java.lang.Object` است و در مثال فوق به نوع مورد نظر `Cast` شده اند.

نکته دیگر اینکه، آبجکتها به همان ترتیبی که قبل از `Stream` نوشته شده اند باید از `Stream` خوانده شوند.

Scanner کلاس

در اغلب موارد، وقتی با ورود و خروج داده های متنی سروکار داریم، در گیر ترجمه داده های متنی به قالبی نزدیک به زبان انسانی و بالعکس است. برای این منظور، جاوا کلاس `java.util.Scanner` یک متن را به تکه های کوچک که به آنها `token` گفته می شود تجزیه می کند.

به صورت پیش فرض، کلاس `Scanner` متن ورودی را براساس «خط فاصله» به رشته های کوچک تر تجزیه می کند. منظور از خط فاصله، فضای خالی بین کلمات (`white space`)، `tab` و نقطه است که پایان جمله ها را مشخص می کند. برای درک عملکرد کلاس `Scanner` به کد ۱۶-۱۸ توجه کنید که محتوای یک فایل متنی با نام `readme.txt` را می خواند و آنرا در خروجی استاندارد چاپ می کند.

```
1 package ir.atlassoft.javase.chapter16;
2
3 import java.io.*;
4         import java.util.Scanner;
5
6 public class ScanReadme {
7     public static void main(String[] args) throws IOException {
8
9         Scanner scanner = null;
10
11     try {
12         scanner = new Scanner(new BufferedReader(new
13 FileReader("readme.txt")));
14     }
```

```

15         while (scanner.hasNext()) {
16             System.out.println(scanner.next());
17         }
18     } finally {
19         if (scanner != null) {
20             scanner.close();
21         }
22     }
23 }
24 }
```

کد ۱۷-۱۸

در حالیکه محتوای فایل `readme.txt` به صورت زیر است

```
#####
Full Source of the Book Java SE Development
Atlassoft Book Series.
#####
```

نتیجه اجرای کد ۱۶-۱۸ به صورت زیر خواهد بود.

```
#####
Full
Source
of
the
Book
Java
SE
Development
Atlassoft
Book
Series.
#####
```

برای جایگزین کردن هر کاراکتری به جای خط فاصله، تا جداسازی برمبنای آن انجام شود از متده استفاده `useDelimiter()` می شود. مثلا برای جایگزین کردن کاما به جای خط فاصله، از این متده استفاده زیر صورت می شود.

```
scanner.useDelimiter(", \\\\s*");
```

خلاصه

برای خواندن داده ها از یک منبع خارجی یا نوشتن آنها در یک منبع خارجی از مجموع کلاس هایی استفاده می شود که به آنها کلاس های `O/I` گفته می شود منبع خارجی ممکن است یک فایل، جایی در شبکه یا حافظه کامپیوتر باشد.

کلاس هایی `O/I` در جاوا به دو دسته عمده تقسیم می شوند:

- کلاس های کاراکتری
- کلاس های با ایتی

کلاس های کاراکتری برای انتقال داده های کاراکتری طراحی شده اند و کلاس های با ایتی برای انتقال داده های با ایتی پیاده سازی شده اند. اگرچه در عمل همه داده ها به صورت با ایت منتقل می شوند کلاس های کاراکتری تنها به منظور سهولت در انتقال داده و فراهم کردن امکان جابجایی کاراکترهایی با `encoding` متفاوت طراحی شده اند.

کلاس های `Writer` و `Reader` به عنوان `superclass` تمام کلاس های کاراکتری بوده و کلاس های `OutputStream` و `InputStream` تمام کلاس های با ایتی هستند کلاس های متعددی از این کلاس ها مشتق شده اند که هر یک به منظور انتقال نوع خاصی از داده ها طراحی شده اند. مثل کلاس های `FileWriter` و `FileReader` برای نوشتن و خواندن فایلهای متنی (`text`) و `HTML` و `XML` و `CSV` و `JSON` و `YAML` و `Properties` و `Properties` کلاس های `FileInputStream` و `FileOutputStream` برای نوشتن و خواندن فایلهای با ایتی (عکس، صوت، فیلم،...) طراحی شده اند.

آبجکتها برای برنامه (که از کلاس های جاوا ساخته شده اند) را می توان روی شبکه منتقل کرد یا آنها را روی دیسک سخت ذخیره نمود به این عمل `Serialization` گفته می شود برای اینکه یک آبجکت قابل باشد باید کلاس آن آبجکت اینترفیس `Serializable` را پیاده سازی کرده باشد.

تمرینات

- (۱) برنامه‌ای بنویسید که مسیر یک دایرکتوری را از طریق کنسول برنامه دریافت کند، و فایلها و دایرکتوریهای موجود در آن را نمایش دهد. در صورتیکه مسیر مشخص شده روی سیستم وجود نداشته باشد، با یک پیغام مناسب نادرست بودن مسیر را اعلام کند.
- (۲) برنامه‌ای بنویسید که کاربر ضمن مشخص نمودن مسیر یک فایل، متن مورد نظر خود را در آن فایل ذخیره کند.
- (۳) برنامه‌ای بنویسید که مسیر فایل یک عکس را از کاربر دریافت کند، و آنها در دایرکتوری دیگری که کاربر مشخص می‌کند کپی کند.
- (۴) برنامه‌ای بنویسید که مسیر یک فایل متنی همراه با یک کلمه را از کاربر دریافت کند، سپس فایل متنی را برای یافتن آن کلمه جستجو کند، در پایان تعداد کلمات یافته شده را به کاربر گزارش کند.
- (۵) برنامه‌ای بنویسید که یک رشته را از کاربر دریافت نموده و درون یک `ArrayList` قرار می‌دهد، سپس قبل از پایان برنامه آبجکت `ArrayList` را روی دیسک سخت ذخیره کند (Serialize) در دفعات بعدی که این برنامه اجرا می‌شود، قبل از اینکه کاربر رشته ای را وارد کند آبجکت `ArrayList` از روی دیسک سخت خوانده شده و رشته‌های قبلی که توسط وی وارد شده اند در کنسول برنامه چاپ شوند.
- (۶) برنامه‌ای بنویسید که از کاربر می‌خواهد تا مسیر یک دایرکتوری را مشخص کند، سپس از وی میخواهد تا نام فایلی که قصد جستجوی آنرا دارد نیز وارد کند، سپس تمام فایلهایی که در آن دایرکتوری یا زیردایرکتوریهای آن وجود داشته باشد را به وی گزارش می‌کند.
- (۷) برنامه‌ای بنویسید که مسیر دو دایرکتوری را دریافت کند و تمام فایلها و دایرکتوریهای داخل شاخه اول را درون شاخه دوم کپی کند.
- (۸) برنامه‌ای بنویسید که تعداد کلمات و تعداد خطوط یک فایل متنی را در کنسول برنامه چاپ کند.
- (۹) متدى بنویسید که مسیر یک فایل متنی را دریافت کند و کلمه‌ای که بیش از بقیه در آن فایل تکرار شده است را بیابد.
- (۱۰) متدى تمرین ۱ را به گونه‌ای تغییر بدهید که کلمات (نقطه، . ، is ، am ، are ، they ، he ، she) را در محاسبات خود قرار ندهد.
- (۱۱) متدى پیاده سازی کنید که مسیر یک دایرکتوری و پسوند فایل (.exe .txt .png .etc) را دریافت کند و مشخص کند که آیا فایلی با آن پسوند در آن دایرکتوری وجود دارد یا خیر.
- (۱۲) متدى پیاده سازی کنید که مسیر یک فایل متنی را دریافت کند و محتوای آن را در قالب یک `String` برگرداند.

وروڈی و خروجی

- (۱۳) متدى بنويسيد که مسیر يک فايل jpeg را دريافت کند و فايل جديدي توليد کند که اندازه هاي آن حداکثر ۱۰۰*۱۰۰ باشد. مثلا اگر يک فايل my-pic.jpeg با ابعاد ۲۰۰*۲۰۰ وجود داشته باشد عکس تغيير اندازه داده شده اين عکس که ابعاد آن ۶۰*۶۰ با نام my-pic-small.jpeg را توليد کند. (راهنمايی: برای اين منظور باید کمی در اينترنت جستجو کنيد و نحوه تغيير اندازه عکس را پيدا کنيد)
- (۱۴) متدى بنويسيد که مسیر يک دايركتوري و يک عدد long را دريافت کند و ليست تمام فايلهايی که اندازه آن بزرگتر از عدد long وارد شده باشد را برگرداند.
- (۱۵) کلاسي File را به گونه اي توسعه دهيد (extends) که هر بار فايلی را ايجاد، حذف، يا تغيير نام می دهيد در کنار آن فايل، فايل دیگری با همان نام و با پسوند history.txt - ايجاد شود که مشخص کند آن فايل در چه زمانهایی چه تغييری کرده است. (راهنمايی مثلا اگر يک فايل با نام project-history.txt داشته باشيم در کنار آن فايلی با نام project.doc ايجاد کند و در هر خط از آن توضيح دهد که چه تغييري - ايجاد، تغييرنام، يا حذف- در چه زمانی برای اين فايل اتفاق افتاده است)
- (۱۶) متدى بنويسيد که مسیر دو دايركتوري را دريافت کند و از تمام محتويات دايركتوري اول پشتيبان تهييه کند و آنرا در دايركتوري دوم قرار دهد درصورتیکه مسیر دايركتوري دوم null باشد پشتيبان را در کنار دايركتوري اول قرار دهد. (راهنمايی: مثلا اگر نام دايركتوري projects باشد دايركتوري پشتيبان آن با نام projects-back باشد)
- (۱۷) متدى بنويسيد که مسیر يک دايركتوري را دريافت کند و تعداد تمام فايلهايی که در آن دايركتوري يا زير دايركتوريهاي آن دايركتوري وجود دارند را برگرداند.
- (۱۸) يک متى جستجو بنويسيد که يک نام و مسیر يک دايركتوري را دريافت کند و ليست تمام فايلهايی که با آن نام مشابهت اسمی دارند را برگرداند نوع برگشتی اين متى آريه ای از File باشد.
- (۱۹) کلاس OutputStream را توسعه دهيد و نام آن را TeeOutputStream بگذاريid اين کلاس باید در سازنده خود دو آبجکت OutputStream دريافت کند و سپس در زمان نوشتن در آن محتويات خود را به دو OutputStream ارسال کند. برای تست اين کلاس، متنی را از کاربر دريافت کنيد (از طريق System.in) و متن دريافت شرده را در System.out و يک فايل متنی بنويسيد.
- (۲۰) يک کلاس InfiniteReader پياده سازی کنيد که در سازنده خود يک کاراكتر دريافت می کند و هربار که متى read() فراخوانی شود آن کاراكتر را برمی گرداند (این Stream بی انتها خواهد بود). يک کلاس NullWriter نيز پياده سازی کنيد که متى write() آن کاراكتر دريافت شده را صرفنظر می کند. (این Stream عملا هیچ خروجي ندارد). پياده سازی اين کلاسها ممکن است در تست برنامه ها کاربرد داشته باشند.

IV

Thread ها و همزمانی

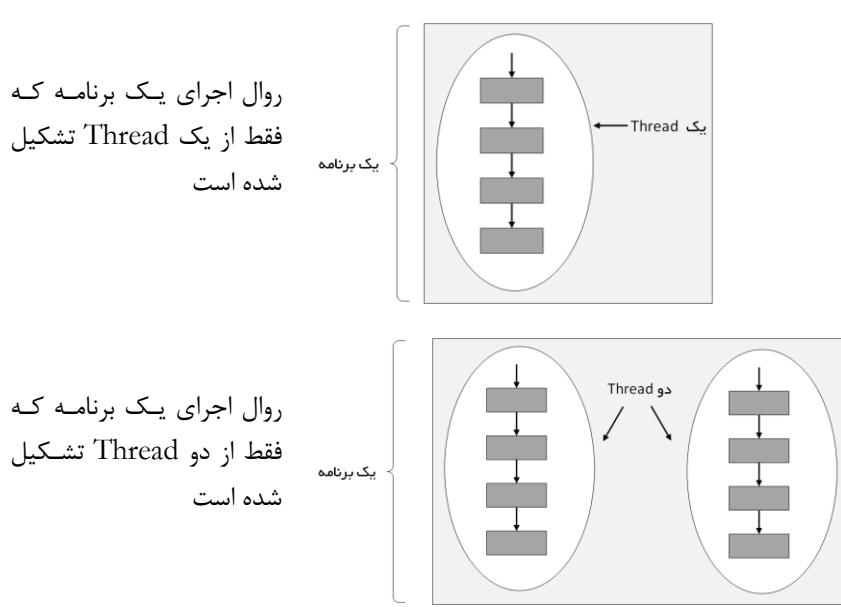
اگر دقت کرده باشید، کامپیوترتان می تواند چندین عمل را بصورت همزمان انجام دهد مثلا در حالیکه همزمان تعدادی سند را از طریق چاپگر چاپ می کنید، در حال تایپ در MS-Word هستید، به موسیقی که از کامپیوترتان پخش می شود گوش می کنید، و آنتی ویروس کامپیوترتان هم از طریق اینترنت درحال بروزرسانی است! به این قابلیت که یک کامپیوتر به صورت همزمان چندین عمل مختلف را انجام می دهد «چند پردازشی»^۱ گفته می شود چند پردازشیبکی از قابلیت های سیستم عامل های جدید است که در سیستم عامل های قدیمی از قبیل MS-DOS، پشتیبانی نمی شد.

همانطور که می دانید، تمام سخت افزار و سیستم شما توسط نرم افزاری که به آن سیستم عامل می گوییم کنترل و مدیریت می شود. وقتی چند برنامه به صورت همزمان اجرا شوند، سیستم عامل در بازه های زمانی کوتاه، CPU و منابع سیستم را به اجرای یک برنامه اختصاص می دهد تا به صورت چرخشی هر برنامه در مدت زمان کوتاهی اجرا شود و نوبت خود را به برنامه بعدی بدهد. بدین ترتیب چنین به نظر می رسد که گویا برنامه ها همزمان اجرا می شوند، در حالیکه در هر لحظه فقط یک برنامه در حال اجراست.

اما همانطور که برنامه ها می توانند به صورت موازی و همزمان اجرا شوند، اجزای کوچکتر یک برنامه هم می توانند به موازات یکدیگر و همزمان اجرا شوند. این موضوع که به آن Multi-Threading گفته می شود، موضوع این فصل است.

اگرچه Multi-Threading شباهتهایی با چندپردازشی دارد و از خصوصیات سیستم عامل های جدید است اما موضوع متفاوتی است. Multi-Threading امکان می دهد تا هر پردازش (برنامه) به بخش‌های کوچکتری که به آنها Thread گفته می شود تقسیم شود تا این بخش‌های کوچکتر بتوانند به صورت همزمان اجرا شوند و با این موازی سازی، از منابع سیستم به صورت بهینه استفاده شود و در مجموع کارایی برنامه ها افزایش یابد.

همانطور که می دانید، درگاه های ورودی/خروجی از قبیل شبکه، کیبرد، و هارددیسک نسبت به CPU سرعت بسیار پایین تری دارند که باعث می شود قسمت اعظم وقت یک برنامه در انتظار ورود یا خروج داده ها تلف شود. با وجود Multi-Threading یک برنامه می تواند وقت انتظار برای یک ورود یا خروج را صرف انجام کار دیگری کند. به عنوان مثال در حالیکه یک بخش برنامه در حال ارسال یک داده روی اینترنت است، همزمان بخش دیگر می تواند متنی که توسط کاربر در کیبرد تایپ می شود را دریافت کند. شکل زیر استفاده از Thread ها برای موازی سازی انجام کارها و پردازش‌های برنامه ها را نشان می دهد.



شکل ۱۷. اجرای Thread ها

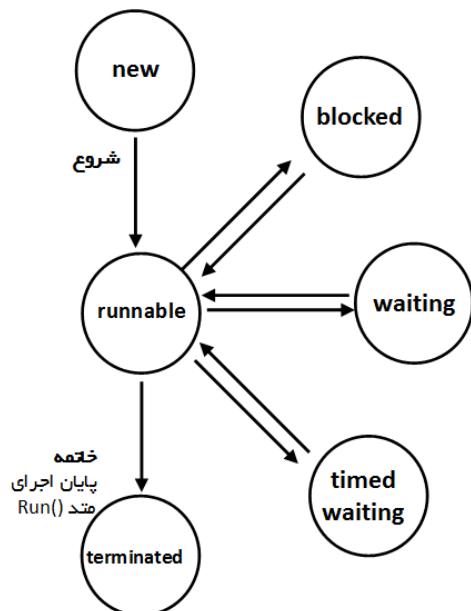
توجه داشته باشید که استفاده از Thread در طراحی یک برنامه فقط در مواردی که فرایندهای موازی در منطق آن برنامه وجود داشته باشد منطقی به نظر می رسد. بنابراین در یک برنامه که فرایندهایی را به ترتیب اجرا می کند و هر فرایند الزاماً می بایست بعد از اتمام فرایند قبلی خود شروع شود استفاده از Thread نه تنها مسئله ای را حل نمی کند بلکه پیچیدگیهای توسعه را بیشتر هم میکند.

Thread ها و همزمانی

از نظر پیاده سازی، هر Thread در واقع یک آبجکت جاواست که مثل هر آبجکت جاوای دیگری ممکن است متدها و فیلدهای خودش را داشته باشد. اصلی ترین تفاوت یک Thread با آبجکتها معمولی جاوا این است که وقتی متدهی از یک آبجکت معمولی فراخوانی می شود برنامه تا اتمام اجرای آن متدهای منتظر می ماند اما در ها وقتی متدهای آنها فراخوانی می برنامه در انتظار اتمام اجرای Thread نمی ماند و ادامه می باید. بنابراین وقتی در مورد برنامه نویسی Multi-Thread صحبت می کنیم در واقع از پیاده سازی نوع خاصی کلاس صحبت می کنیم که قاعده تا به شکل متفاوتی فراخوانی و اجرا می شوند.

از زمانی که یک آبجکت Thread ایجاد می شود تا زمانی که اجرای آن خاتمه می یابد ممکن است در وضعیت های مختلفی قرار بگیرد.

- - آبجکت new ایجاد شده اما اجرای آن هنوز شروع نشده است.
 - - در حال اجرا باشد.
 - - Thread منتظر آزادشدن یک منبع (مثلاً یک آبجکت) باشد که در حال حاضر در اختیار Thread دیگری است.
 - - براساس منطق برنامه تا برقراری یک شرط با فراخوانی متدهای wait به وضعیت «انتظار» برده شود.
 - - بر اساس منطق برنامه، با فراخوانی متدهای sleep برای مدت زمان مشخصی غیرفعال شود.
 - - اجرای Thread به پایان برسد. Thread ممکن است وظیفه ای که برایش مشخص شده را به صورت کامل انجام دهد و به پایان برسد یا ممکن است در یک شرایط غیرمنتظره قرار گیرد و اجرای آن خاتمه یابد. در هر دو حالت Thread به وضعیت terminated منتقل می شود.
- شکل زیر وضعیت‌های مختلف یک Thread را نشان می دهد.



شکل ۲-۱۷. وضعیت های مختلف یک Thread

پیاده سازی و اجرای Thread

- در جاوا دو روش برای پیاده سازی Thread وجود دارد:
- کلاسی بنویسید که اینترفیس java.lang.Runnable را پیاده سازی کند.
 - کلاسی بنویسید که از کلاس java.lang.Thread مشتق شده باشد.

پیاده سازی اینترفیس Runnable

ساده ترین و رایج ترین راه برای پیاده سازی یک Thread، پیاده سازی اینترفیس Runnable است که دارای یک متد () run() است که طبیعتاً باید پیاده سازی شود.

```

1 package ir.atlassoft.javase.chapter17;
2
3 public class SimpleRunnableThread implements Runnable {
4     public void run() {
5         //do something
6     }
7 }
```

کد ۱-۱۷

Thread ها و همزمانی

متدهای run() حاوی عملیاتی است که توسط Thread انجام می‌شود. وقتی اجرای متدهای run() به انتها بررس (یا با رخداد exception) خاتمه یابد، اجرای Thread خاتمه یافته و به وضعیت terminated منتقل می‌شود.

برای اجرای Thread که به شکل فوق پیاده سازی گردید، لازم است ابتدا آبجکتی از کلاس ساخته شود، و در یک آبجکت Thread پیچیده شود و متدهای start() و run() آن به صورت زیر فراخوانی شود.

```
SimpleRunnableThread srt = new SimpleRunnableThread();
Thread thread = new Thread(srt);
thread.start();
```

با فراخوانی متدهای start() و run() انتظار داریم تا منطقی که در متدهای run() پیاده سازی کردیم در موازات اجرای بقیه برنامه اجرا شود.

توسعه کلاس Thread

روش دوم برای پیاده سازی یک Thread، توسعه کلاس java.lang.Thread است. در این روش، کلاسی پیاده سازی می‌کنیم که از کلاس java.lang.Thread مشتق شده باشد و سپس متدهای run() که حاوی عملیاتی است که توسط Thread انجام می‌دهد پیاده سازی می‌کنیم.

```
1 package ir.atlassoft.javase.chapter17;
2
3 public class SimpleThread extends Thread{
4
5     public void run() {
6         //thread logic to execute
7     }
8 }
```

کد ۱۷-۳

برای اجرای این Thread، کافیست آبجکتی از روی آن ایجاد و متدهای start() و run() آن فراخوانی شود.

```
SimpleThread st = new SimpleThread();
st.start();
```

به این ترتیب انتظار داریم منطقی که در متدهای run() پیاده سازی شده است به موازات مابقی برنامه اجرا شود.

همانطور که ملاحظه کردید، از هر روشی که برای پیاده سازی یک Thread استفاده شود (پیاده سازی Runnable یا توسعه کلاس java.lang.Thread)، برای اجرای آن از کلاس

ج استفاده می شود. در حقیقت کلاس `java.lang.Thread` اصلی ترین کلاس در بحث برنامه نویسی Thread هاست. اجازه دهد قبل از اینکه وارد جزییات پیاده سازی Thread ها و اجرای آنها شویم کمی کلاس `java.lang.Thread` را واکاوی کنیم.

Thread کلاس

احاظه دهید ابتدا نگاهی به فیلدهای اصلی کلاس Thread بیندازیم و سپس متدهای آنرا بررسی کنیم.

فیلدهای Thread کلاس

در کلاس Thread سه فیلد مهم زیر تعریف شده است.

```
String name;
int priority;
boolean daemon;
```

name

هر آبجکت Thread دارای یک نام است. این نام ممکن است توسط برنامه نویس برای Thread مشخص شود یا توسط سازنده کلاس Thread مقداردهی شود. به عنوان مثال برای ایجاد آبجکت Thread می توانستیم به صورت زیر عمل کنیم.

```
SimpleRunnableThread srt = new SimpleRunnableThread();
Thread thread = new Thread(srt, "Simple-Runnable-Thread");
thread.start();
```

در اینجا نام Simple-Runnable-Thread به عنوان نام Thread انتخاب شده است. اگر مطابق قبل نامی برای آن انتخاب نمی شد، سازنده کلاس Thread یک نام که با فرمت Thread-number به آن اختصاص می دهد که number یک شمارنده است که با ایجاد هر آبجکت Thread یک واحد افزایش می یابد. وجود نام برای هر Thread امکان پیدا کردن و کنترل یک آبجکت Thread وقتی مجموعه ای از آبجکتها در حال اجرا هستند را فراهم می کند. دسترسی به نام یک Thread با فراخوانی متد getName() امکانپذیر است.

priority

هر Thread دارای یک اولویت است. وقتی مجموعه ای از Thread ها آمده برای اجرا باشند، مدیر اجرای Thread ها که بخشی از JVM است Thread ای که اولویت بالاتری داشته باشد را برای اجرا انتخاب می کند. به صورت پیش فرض، Thread ها اولویت خود را Thread ای که آنها را ایجاد کرده است به ارث می بردند. اما این امکان وجود دارد که با فراخوانی متد setPriority() Thread را تغییر داد.

Thread ها و همزمانی

متدهای Thread ها در Java برای ایجاد و مدیریت پارامتر دریافت می‌کند. متد `setPriority()` یک عدد int بین ۱ و ۱۰ است را به عنوان پارامتر دریافت می‌کند. بالاترین اولویت ۱ و پایین ترین اولویت از طریق مقادیر ثابت `MAX_PRIORITY` و `MIN_PRIORITY` در کلاس `java.lang.Thread` تعریف شده است. توجه داشته باشید که اگر چندین `thread` با اولویت بالا داشته باشید که تحت هیچ شرایطی نیز غیرفعال نشوند، ممکن است یک `thread` که اولویت پایینی دارد هیچگاه اجرا نشود! بنابراین استفاده از اولویت تنها زمانی منطقی است که در شرایطی `thread` های با اولویت پایین امکان اجرا شدن بیاند.

daemon

یک `thread` ممکن است به عنوان `daemon` تعیین شود که معنی آن این است که هیچ عملیات اصلی که بخشی از کارکرد اصلی برنامه است توسط آن انجام نمی‌شود. `daemon` های `Thread` در واقع عملیات کمکی را در برنامه انجام می‌دهند. مثلاً تایمر نمونه‌ای از این `Thread` هاست. خاصیت این `Thread` ها در این است که به عنوان کمکی برای کارکرد بقیه `Thread` ها استفاده می‌شوند. اگر اجرای برنامه خاتمه یابد، اما در حافظه `Thread` های فعال وجود داشته باشد که همگی `daemon` باشند JVM اجرای برنامه را خاتمه می‌دهد در غیراینصورت تا انمام آخرین `Thread` که `daemon` نباشد منتظر می‌ماند.

برای ایجاد `daemon` یک `Thread` با فراخوانی متد `dameon`

```
void setDaemon(boolean daemon)
```

از آبجکت `Thread` امکانپذیر است.

متدهای کلاس Thread

در زیر چند متد اصلی کلاس `Thread` لیست شده است.

متدها	توضیح
<code>public void run()</code>	همانطور که صحبت شد، متد <code>run()</code> عملیاتی که انجام می‌دهد را تعریف می‌کند.
<code>public void start()</code>	متد <code>start()</code> شروع اجرای <code>Thread</code> را مشخص می‌کند.
<code>public boolean isAlive()</code>	اگرچه ما با فراخوانی متد <code>start()</code> شروع کننده اجرای یک <code>Thread</code> هستیم اما ما خاتمه دهنده اجرای آن نیستیم. در واقع بسته به اینکه منطق داخل یک <code>Thread</code> چقدر پیچیدگی داشته باشد و بسته به شرایط محیطی مختلف (از لحظه سخت افزار، سیستم عامل، برنامه‌های در حال اجرا در سیستم، تعداد <code>Thread</code> های برنامه...) ممکن است زمان اجرای کوتاه یا بلندی داشته باشد.

<p>متدهای <code>isAlive()</code> امکان می‌دهد تا از پایان یافتن یا پایان نیافتن یک <code>thread</code> مطلع شویم.</p>	
<p>فراخوانی این متدها، به اجرای <code>thread</code> در هر وضعیتی که باشد خاتمه می‌دهد و آن را به وضعیت <code>terminated</code> منتقل می‌کند.</p>	<code>public void interrupt()</code>
<p>این متدهای <code>isInterrupted()</code> می‌کند که آیا اجرای <code>thread</code> نیمه کاره به پایان رسیده است یا خیر. دلیل نیمه کاره ماندن یک <code>thread</code> ممکن است فراخوانی متدهای <code>interrupt()</code> باشد یا شرایط ویژه‌ای باشد.</p>	<code>public boolean isInterrupted()</code>
<p>این متدهای <code>sleep()</code> را برای مدت زمان مشخصی غیرفعال می‌کند. دو نمونه از این متدهای کلاس <code>java.lang.Thread</code></p>	<code>public static void sleep(long ms)</code>
<pre><code>static void sleep(long milliseconds) throws InterruptedException static void sleep(long milliseconds, int nanoseconds) throws InterruptedException</code></pre> <p>اولی، تعداد میلی ثانیه‌هایی که <code>thread</code> باید غیرفعال شود را مشخص می‌کند و دومی تعداد میلی ثانیه‌ها و نانوثانیه‌های اضافی که <code>thread</code> باید غیرفعال بماند را مشخص می‌کند. در مواقعيتی که <code>thread</code> برای ادامه کار خود نیازمند شرایطی باشد که هنوز برقرار نیست، با فراخوانی متدهای <code>sleep()</code> می‌توانیم برای مدت زمان مشخصی آنرا غیرفعال کنیم تا منابعی (CPU، حافظه...) که توسط آن اشغال شده است به <code>thread</code> های دیگر اختصاص یابد و پس از سپری شدن آن زمان مجدداً فعالیت خود را از سر بگیرد.</p>	
<p>فراخوانی این متدهای <code>yield()</code> مشخص می‌کند که آماده است که وقت خود را در اختیار <code>thread</code></p>	<code>public static void yield()</code>

Thread ها و همزمانی

<p>دیگری قرار دهد. در اینصورت «هماهنگ کننده thread ها» که بخشی از jvm است و مسئولیت مدیریت و اجرای thread ها را بر عهده دارد ممکن است بر اساس سیاست خودش برای مدت زمانی thread را غیرفعال کند. از این متد بیشتر در زمان تست برنامه و مشاهده وضعیت اجرای thread ها در زمان تست استفاده می شود.</p>	
<p>متد <code>join()</code> که برخلاف متد <code>isAlive()</code> به ندرت استفاده می شود، امکان می دهد تا یک thread تا اتمام Thread دیگر منتظر بماند. در اینصورت حتی اگر thread اول به پایان برسد تا اتمام thread دوم در وضعیت اجرا باقی می ماند و به وضعیت <code>terminated</code> منتقل نمی شود.</p> <pre>final void join() final void join(long millis) final void join(long millis, int nanos)</pre> <p>در فراخوانی متد <code>join()</code> می توان حداکثر مدت زمانی را که thread باید منتظر اتمام thread دوم بماند را مشخص کنید.</p>	<p><code>public void join()</code></p>

جدول ۱۷. متد های کلاس Thread

بازی پینگ پنگ

در بازی پینگ پنگ، دو بازیکن وجود دارد که به نوبت هر کدام با راکت به توپ ضربه می زند. اگر بخواهیم هر بازیکن را با Thread شبیه سازی کنیم، کدی به صورت زیر خواهیم داشت.

```
1 package ir.atlassoft.javase.chapter17;
2
3 public class PingPongThread extends Thread {
4
5
6     private String word; // what word to print
7     private int delay; // how long to pause
```

```

8
9     public PingPongThread(String name,
10                         String whatToSay,
11                         int delayTime) {
12         super(name);
13         word = whatToSay;
14         delay = delayTime;
15     }
16     public void run() {
17         try {
18             while (true) {
19                 System.out.print(getName() + ":" + word + " ");
20                 Thread.sleep(delay); // wait until next time
21             }
22         } catch (InterruptedException e) {
23             return; // end this thread
24         }
25     }
26 }
```

کد ۱۷-۳

کلاس PingPongThread از کلاس Thread مشتق شده و متد run() را پیاده سازی کرده است و به این ترتیب به عنوان یک Thread قابل اجراست. علاوه بر name که فیلدی از کلاس java.lang.Thread است، این کلاس دارای دو فیلد word و delay است که به ترتیب صدای بازی هر بازیکن و تاخیر هر ضربه و ضربه بعدی همان بازیکن را نشان می دهد. در متد run() یک حلقه بینهایت، ابتدا صدای بازیکن در کنسول چاپ شده است و سپس به اندازه مدت زمان delay، اجرای Thread متوقف شده است. اجرای بازی پینگ با ایجاد آبجکتهایی از کلاس فوق و اجرای آنها به صورت زیر امکانپذیر است.

```

1 public static void main(String[] args) {
2     PingPongThread player1 =
3         new PingPongThread("Playe1", "ping", 33);
4     PingPongThread player2 =
5         new PingPongThread("Playe2", "PONG", 100);
6     player1.start(); // 1/30 second
7     player2.start(); // 1/10 second
8 }
```

در اینجا فرض شده که بازیکن اول صدای «پینگ» و بازیکن دوم صدای «پانگ» را تولید کند. بازیکن اول هر ۳۳ میلی ثانیه و بازیکن دوم هر ۱۰۰ میلی ثانیه به تپ ضربه می‌زنند. واضح است که این دو Thread یکدیگر همگام نیستند زیرا در حالیکه Thread اول سی بار در هر ثانیه صدای ping تولید می‌کند، فقط ۱۰ بار صدای pong فقط از Thread دوم شنیده می‌شود. در ادامه این فصل خواهید دید که چگونه Thread ها می‌توانند با یکدیگر همگام شوند.

Thread اصلی

فرض کنید برنامه‌ای نوشته اید که از چندین Thread تشکیل شده است، یعنی با اجرای برنامه آبجکتهاي Thread ایجاد و به موازی یکدیگر اجرا می‌شوند. واضح است که در شروع برنامه یعنی زمانیکه متند main() اجرا می‌شود هنوز هیچ آبجکتی در برنامه ایجاد نشده و طبیعتاً هیچ Thread ای در برنامه وجود ندارد. اما با اجرای متند main() آبجکتهاي Thread ها ایجاد شده و اجرا می‌شوند. نکته جالب اینجاست که تمام برنامه‌های جاوا در قالب Thread اجرا می‌شوند حتی اگر هیچ Thread ای توسط برنامه نویس ایجاد نشده باشد. به عبارت دیگر هر برنامه جاوا حداقل دارای یک Thread است که توسط JVM (نه توسط برنامه نویس) ایجاد و اجرا می‌شود. به این Thread که توسط JVM تولید می‌شود و مسئولیت اجرای متند main() را به عهده دارد «اصلی برنامه» گفته می‌شود. تمام Thread های دیگر برنامه، توسط Thread اصلی برنامه اجرا می‌شوند. به Thread هایی که توسط یک Thread ایجاد می‌شوند فرزندان Thread گفته می‌شود. اجرای هر Thread پس از اتمام اجرای کلیه Thread های فرزند پایان می‌یابد. کلاس java.lang.Thread دارای متند استاتیک currentThread() به صورت زیر است که آبجکتی از جنس Thread بر می‌گرداند.

```
static Thread currentThread()
```

در هر نقطه‌ای از برنامه که این متند را فراخوانی کنید، آبجکت Thread ای که در حال اجرای آن نقطه از برنامه است را بر می‌گرداند. به این ترتیب اگر متند فوق را در متند main() که برنامه از آن آغاز می‌شود فراخوانی کنید آبجکت Thread اصلی برنامه را بر می‌گرداند.

همزمانی (Synchronization)

وقتی دو یا چند Thread از یک آبجکت مشترک استفاده می‌کنند ممکن است باعث تداخل در کار یکدیگر شوند. بنابراین باید به رویی دسترسی آنها به منبع مشترک را کنترل کرد به گونه‌ای که دو Thread بدون تداخل در عملکرد یکدیگر به درستی به اجرای خود ادامه دهند. به این فرایند، کنترل همزمانی گفته می‌شود.

کنترل همزمانی در حقیقت از وظایف JVM است، آنچه به ما به عنوان طراح یا برنامه نویس مربوط می شود مشخص کردن نقاطی از آبجکت مشترک است که در زمان اجرای Thread‌ها ممکن است منجر به تداخل عملکرد آنها شود. برای درک این موضوع، کلاس Counter زیر را ملاحظه کنید.

```

1 package ir.atlassoft.javase.chapter17;
2
3 public class Counter {
4
5     int index=0;
6
7     public void incrementAndPrint() {
8         index++;
9         System.out.println(index);
10    }
11 }
```

کد ۱۷-۵

این کلاس دارای یک فیلد index و یک متده است که با هر فراخوانی incrementAndPrint () مقدار فیلد index را یک واحد افزایش داده و در کنسول برنامه چاپ می کند. حال تصور کنید که این متده همزمان توسط دو Thread فراخوانی شود. در فراخوانی اول، مقدار index یک واحد افزایش می دهد و درست قبیل از اینکه مقدار index در کنسول چاپ شود، فراخوانی دوم توسط Thread دوم شروع می شود و مقدار index یک واحد دیگر افزایش می یابد. سپس فراخوانی اول مقدار index را که در حال حاضر 2 است در کنسول برنامه چاپ می کند و خاتمه می یابد و سپس فراخوانی دوم نیز همین مقدار 2 را در کنسول نمایش می دهد. این درحالیست که انتظار داریم که با فراخوانی اول مقدار 1 و با فراخوانی دوم مقدار 2 در کنسول برنامه چاپ شوند.

در واقع برای اینکه متده incrementAndPrint () درست کار کند Thread‌ها نباید به صورت همزمان این متده را فراخوانی کنند و اگر فراخوانی کنند می باید با استفاده از روشی مانع از فراخوانی همزمان این متده شویم. این کار با افزودن کلمه کلیدی synchronized به تعریف متده incrementAndPrint () امکان‌پذیر است.

```

1 package ir.atlassoft.javase.chapter17;
2
3 public class Counter1 {
4
5     int counter = 0;
```

Thread ها و همزمانی

```
7     public synchronized int incrementAndReturn() {  
8         counter++;  
9         return counter;  
10    }  
11 }
```

کد ۱۷-۶

کلمه synchronized نشانه‌ای است که به JVM اعلان می‌کند متدهای incrementAndPrint() نمی‌توانند همزمان توسط دو یا چندین Thread فراخوانی شود و در صورتیکه فراخوانی شود باید Thread‌ها یکی یکی این متدهای فراخوانی کنند و تا زمانیکه یک فراخوانی به اتمام نرسیده Thread بعدی نمی‌تواند این متدهای فراخوانی کند.

کنترل همزمانی با استفاده از یک مکانیسم قفل گذاری توسط JVM انجام می‌شود. براساس این مکانیسم وقتی یک متدهای synchronized فراخوانی می‌شود، JVM روی آبجکت آن متدهای synchronized به آن تعلق دارد) یک قفل قرار می‌دهد، البته مشروط به اینکه قبل از آن آبجکت قفلی قرار نگرفته باشد. پس از پایان فراخوانی متدهای synchronized، قفل از روی آبجکت برداشته می‌شود. تا مادامیکه قفل روی آبجکت وجود داشته باشد، هیچ Thread دیگری نمی‌تواند آن متدهای synchronized را فراخوانی کند. حال که مفهوم synchronized و مکانیسم قفل گذاری را آموختید به کلاس ProcessorThread که در زیر پیاده سازی شده است توجه کنید

```
1 package ir.atlassoft.javase.chapter17;  
2  
3 public class ProcessorThread extends Thread {  
4  
5     Counter counter;  
6  
7     public ProcessorThread(Counter counter) {  
8         this.counter = counter;  
9     }  
10    public void run() {  
11        for(int i=0; i<10; i++) {  
12            counter.incrementAndPrint();  
13        }  
14    }  
15 }
```

کد ۱۷-۷

این کلاس Counter، فیلدی از Thread دارد و در اجرای خود ده بار متدهای incrementAndPrint را فراخوانی می‌کند. اگر دو آبجکت از روى این کلاس با یک آبجکت مشترک از کلاس Counter کار کنند انتظار داریم که تداخلی بین عملکرد آنها ایجاد نشود.

```
Counter counter = new Counter();
ProcessorThread thread1 = new ProcessorThread(counter);
ProcessorThread thread2 = new ProcessorThread(counter);
thread1.start();
thread2.start();
```

با اجرای کد فوق، انتظار داریم که اعداد 1 تا 20 در کنسول برنامه نمایش داده شوند. در صورتیکه اگر متدهای incrementAndPrint به صورت synchronized پیاده سازی نشده بود ممکن بود برخی اعداد در بازه 1 تا 20 اصلاً نمایش نیابند و برخی اعداد دوبار نمایش داده شوند. دقتهای همزنی برای متدهای غیراستاتیک وقتی مطرح است که یک آبجکت مشترک توسط دو یا چندین Thread استفاده شود. بنابراین اگر کد فوق به صورت زیر تغییر یابد اساساً مشکل همزنی وجود نخواهد داشت زیرا هر Thread با آبجکت Counter جدالگانه‌ای سروکار دارد.

```
Counter counter1 = new Counter();
Counter counter2 = new Counter();
ProcessorThread thread1 = new ProcessorThread(counter1);
ProcessorThread thread2 = new ProcessorThread(counter2);
thread1.start();
thread2.start();
```

همانطور که می‌دانید، می‌توان متدهای استاتیک یک کلاس را بدون ایجاد آبجکت از یک کلاس فراخوانی نمود. با این توصیف، آیا می‌توان متدهای استاتیک را به صورت synchronized تعریف کرد؟ جواب این سوال مثبت است. در این صورت و در شرایطی که آبجکتی از آن کلاس برای قفل گذاری ایجاد نشده است از چه آبجکتی برای قفل گذاری و هماهنگی بین Thread‌ها استفاده می‌شود؟ جواب این سوال، کلاسی است که متدهای استاتیک آن فراخوانی شده است. معنی آن این است که متدهای استاتیک از کلاسی که فراخوانی شده اند و متدهای غیراستاتیک از آبجکتی که فراخوانی شده اند برای قفل گذاری استفاده می‌کنند. به این ترتیب، از آنجاییکه محل قفل گذاری محل قفل گذاری متدهای استاتیک و غیراستاتیک متفاوت است، یک متدهای استاتیک می‌تواند همزمان با یک متدهای غیراستاتیک فراخوانی شود اما دو متدهای استاتیک یا دو متدهای غیراستاتیک نمی‌توانند به صورت همزمان فراخوانی شوند.

بلوک synchronized

اگرچه می‌توان با افزودن کلمه کلیدی synchronized به تعریف یک متدهای استاتیک دسترسی همزمان چندین Thread به آن متدهای را کنترل نمود، اما با اینکار مواری سازی و اجرای همزمان آن متدهای از بین رفته است. این

Thread‌ها و همزمانی

بدین معناست که synchronization نقطه مقابل Thread‌ها قرار دارد زیرا در حالیکه Thread‌ها تلاش می‌کنند تا با همزمان شدن اجرای قسمتهای مختلف برنامه، کارایی برنامه را افزایش دهند، synchronization با متوالی کردن اجرای بخش‌های برنامه، اگر نگوییم هدف Thread‌ها را بی اثر می‌کند اما لاقل کم اثر می‌کند. اگر متدى که synchronized می‌شود طولانی باشد، یعنی بدنه بزرگی داشته باشد در اینصورت حجم زیادی از کد از اجرای همزمان محروم شده است! خوشبختانه در چنین موقعی با بهبود طراحی می‌توان تاحدودی از اثرات همزمانی کاست. برای این منظور به جای اینکه کل متده به صورت synchronized تعریف شود می‌توان بخشی از متده که کنترل همزمانی در آن اهمیت دارد را در قالب یک بلوک synchronized تعریف نمود. در اینصورت بخش‌هایی از متده که خارج از بلوک قرار دارند می‌توانند به صورت همزمان توسط Thread‌های مختلف اجرا شوند اما بخش‌های داخل بلوک synchronized از اجرای همزمان توسط Thread‌های مختلف محافظت می‌شوند.

```
synchronized(object) {  
    //statements to be synchronized  
}
```

بلوک synchronized می‌تواند در هرجایی از بدنه یک متده که synchronized نیست ظاهر شود. در این بلوک، بلافاصله بعد از کلمه synchronized داخل پرانتز نام یک آبجکت آورده می‌شود و سپس جملاتی که باید همزمانی آنها کنترل شود داخل {} قرار می‌گیرند. آبجکتی که داخل پرانتز و بعد از کلمه synchronized مشخص می‌شود، محلی است که قفل گذاری روی آن انجام می‌شود. این امکانی است که در متده synchronized وجود ندارد زیرا در متدهای synchronized قفل روی آبجکتی که متده آن فراخوانی می‌شود قرار می‌گیرد، اما در بلوک synchronized این امکان وجود دارد تا هر آبجکتی برای قفل گذاری انتخاب شود.

وقتی یک بلوک synchronized اجرا می‌شود، JVM آبجکتی را که برای قفل گذاری انتخاب شده بررسی می‌کند تا مطئن شود قفلی روی آن وجود ندارد. درصورتیکه آن آبجکت آزاد باشد، روی آن قفل گذاشته می‌شود و بلوک synchronized اجرا می‌شود. در پایان اجرای بلوک، قفل از روی آبجکت برداشته می‌شود.

بلوک یا متده synchronized می‌تواند برای کنترل دسترسی همزمان و فراخوانی متدهای آبجکتها استفاده شود. اما سوالی که در اینجا مطرح می‌شود این است که چگونه می‌توان خواندن و نوشتمن متغیرها و دسترسی همزمان به آنها را کنترل نمود؟

در مورد آبجکتها، هر تغییری در وضعیت آبجکت برای تمام Thread‌های برنامه قابل مشاهده است، به عبارت دیگر تغییر مقدار هر یک از فیلد‌های یک آبجکت به مجردی که انجام شود توسط Thread‌های دیگر قابل مشاهده و دسترسی است و این لحاظ نگرانی وجود ندارد. اما در مورد متغیرهایی از جنس داده‌های پایه، به دلیل روشی که جاوا در سازماندهی این داده‌ها در حافظه استفاده می‌کند می‌بایست با استفاده از کلمه

کلیدی `volatile` که در تعریف آن فیلد استفاده می شود، دسترسی همزمان به آن فیلد برای خواندن یا تغییر مقدار آنرا کنترل نمود. به عبارت دیگر، اگر متغیری به صورت `volatile` تعریف شده باشد به مجرد اینکه مقدار آن تغییر کند مقدار آن توسط همه `Thread` های برنامه قابل مشاهده خواهد بود و در غیراینصورت تضمینی برای آن وجود ندارد. که زیر نحوه بکارگیری `volatile` را نشان می دهد.

```
public class SharedObject {
    public volatile int counter = 0;
}
```

مدیریت ارتباط بین Thread ها

فرض کنید دو `Thread` برای انجام عملیات خود نیاز به ارتباط با یکدیگر دارند یکی از آنها تولیدکننده و دیگری مصرف کننده داده هاست. تولید کننده و مصرف کننده ممکن است سرعت اجرای مختلفی داشته باشند به عبارت دیگر، ممکن است یکی سریعتر و دیگری کندر باشد، در اینصورت ممکن است برخی از داده های تولید شده توسط تولید کننده قبل از اینکه توسط مصرف کننده مصرف شوند دوباره نویسی (overwriting) شوند یا ممکن است به علت سرعت بالاتر مصرف کننده، یک داده تولید شده چندین بار خوانده شود (overreading). بنابراین نیازمند مکانیسمی هستیم که بواسطه آن، تولید کننده و مصرف کننده با یکدیگر همگام شوند.

برای درک این موضوع، اجازه دهید بحث خود را با پیاده سازی یک مثال ادامه دهیم. کدهای زیر کلاس های `Producer` و `Consumer` را که از یک آبجکت مشترک `Holder` برای نوشتن و خواندن داده ها استفاده می کنند نشان می دهن. کلاس `Producer` ده عدد اول (اعدادی که بر هیچ عددی به جز خودشان و یک بخش پذیر نیستند) را در `holder` قرار می دهد و `Consumer` اعداد مورد نظر را به ترتیب از `holder` دریافت می کند. اگر سرعت `Consumer` در مصرف داده های `Holder` بیشتر یا کمتر از سرعت `Producer` در تولید داده های `Holder` باشد در اینصورت ممکن است داده های تکراری مصرف شود یا برخی داده ها قبل از اینکه خوانده شوند از دست بروند.

```
1 package ir.atlassoft.javase.chapter17;
2
3 class Holder {
4     int data;
5
6     synchronized int get() {
7         System.out.println("Got: " + data);
8         return data;
9     }
}
```

Thread و همزمانی

```
10     synchronized void put(int data) {
11         this.data = data;
12         System.out.println("Put: " + data);
13     }
14 }
15
16 class Producer implements Runnable {
17     Holder holder;
18     int[] array = new int[]{2, 3, 5, 7, 11, 13, 17, 19, 23,
19     29};
20
21     Producer(Holder holder) {
22         this.holder = holder;
23     }
24
25     public void run() {
26         for(int i=0; i<10; i++) {
27             holder.put(array[i]);
28         }
29     }
30 }
31 class Consumer implements Runnable {
32     Holder holder;
33
34     Consumer(Holder holder) {
35         this.holder = holder;
36     }
37
38     public void run() {
39         for(int i=0; i<10; i++) {
40             holder.get();
41         }
42     }
43 }
44 class Main{
45     public static void main(String args[]) {
46         Holder holder = new Holder();
47         new Thread(new Producer(holder), "Producer").start();
48         new Thread(new Consumer(holder), "Consumer").start();
49 }
```

```

50     }
51 }
```

کد ۱۷-۸

اگر برنامه فوق را اجرا کنید، خواهید دید که Producer قبل از اینکه داده های تولید شده اش توسط Consumer مصرف شود آنها را بازنویسی می کند و Consumer نیز قبل از اینکه داده جدیدی نوشته شود داده های تکراری از holder برداشت می کند. نتیجه اجرای این برنامه با هریار اجرا (بسته به شرایط محیطی) ممکن است متفاوت باشد. خروجی زیر نتیجه یکی از این اجراهاست.

```

Put: 2
Put: 3
Put: 5
Put: 7
Put: 11
Put: 13
Put: 17
Put: 19
Put: 23
Put: 29
Got: 29
```

برای همگام کردن اجرای Thread‌ها، سه متدهای `notifyAll()`, `notify()` و `wait()` در کلاس `java.lang.Object` طراحی شده است که طبیعتاً از آنجاییکه کلاس‌های جاوا همگی از `java.lang.Object` مشتق شده‌اند در تمام کلاس‌های جاوا به ارث برده شده‌اند. فراخوانی متدهای `wait()` و `notify()` در هر متدهای از یک آبجکت، باعث توقف اجرای Thread‌ی می‌شود که آن متدهای فراخوانی کرده است. حال اگر در همان آبجکتی که قبلاً متدهای `wait()` و `notify()` فراخوانی شده، (همان متدهای `wait()` و `notify()`) از متدهای دیگر آن آبجکت متدهای این آبجکت تغییر نداشته باشند، این متدهای دیگر فراخوانی شده، متدهای `wait()` و `notify()` این آبجکت که متوقف شده است ادامه می‌یابند. مثال `Producer` و `Consumer` را با بکارگیری متدهای `wait()` و `notify()` می‌توان به شکل زیر اصلاح کرد.

```

1 package ir.atlassoft.javase.chapter17;
2
```

Thread و همزمانی

```
3  class Holder1 {
4      int data;
5      boolean newValue=false;
6
7      synchronized int get() throws InterruptedException {
8          try {
9              if (!newValue) {
10                  wait();
11              }
12              System.out.println("Got: " + data);
13              newValue = false;
14              return data;
15          }finally {
16              notify();
17          }
18      }
19      synchronized void put(int data) throws InterruptedException
20  {
21          try {
22              if (newValue) {
23                  wait();
24              }
25              this.data = data;
26              newValue = true;
27              System.out.println("Put: " + data);
28          }finally {
29              notify();
30          }
31      }
32  }
33  class Producer1 implements Runnable {
34      Holder1 holder;
35      int[] array = new int[]{2, 3, 5, 7, 11, 13, 17, 19, 23,
36      29};
37
38      Producer1(Holder1 holder) {
39          this.holder = holder;
40      }
41
42      public void run() {
```

```

43         for(int i=0; i<10; i++) {
44             try {
45                 holder.put(array[i]);
46             } catch (InterruptedException e) {
47                 e.printStackTrace();
48             }
49         }
50     }
51 }
52 class Consumer1 implements Runnable {
53     Holder1 holder;
54
55     Consumer1(Holder1 holder) {
56         this.holder = holder;
57     }
58
59     public void run() {
60         for(int i=0; i<10; i++) {
61             try {
62                 holder.get();
63             } catch (InterruptedException e) {
64                 e.printStackTrace();
65             }
66         }
67     }
68 }
69 class Main1{
70     public static void main(String args[]) {
71         Holder1 holder = new Holder1();
72         new Thread(new Producer1(holder), "Producer").start();
73         new Thread(new Consumer1(holder), "Consumer").start();
74
75     }
76 }
```

کد ۱۷-۹

کلاس Holder بیش از همه تغییر کرده است. در این کلاس یک فیلد newValue اضافه شده است که مشخص می کند آیا داده جدیدی توسط Producer تولید شده است یا خیر. وقتی متده put() توسط فراخوانی می شود علاوه بر مقداردهی فیلد data مقدار newValue به true تغییر داده

Thread ها و همزمانی

می شود که مشخص می کند data حاوی داده جدیدی است. در نقطه مقابل هرگاه متدهای get و Consumer فراخوانی می شود مقدار newValue به تغییر می کند تا مشخص شود که داده data خوانده شده و می تواند مقدار جدیدی در آن قرار گیرد. حال تصور کنید که قبل از تولید داده جدید توسط Producer، با فراخوانی متدهای get و Consumer خواندن اطلاعات را داشته باشد در اینصورت چون داده جدیدی تولید نشده و newValue مقدار دارد، فراخوانی () wait باعث تعليق Consumer می شود، تنها با نوشتن داده جدید در Holder (و بالطبع فراخوانی متدهای put و notify در آبجکت Consumer می تواند به اجرای خود ادامه دهد.

بالعکس تصور کنید که داده نوشته شده هنوز توسط Producer خوانده نشده و با فراخوانی متدهای put و Consumer فراخوانی شده هنوز نوشته شده دارد در این شرایط newValue مقدار true دارد و بنابراین () wait باعث توقف اجرای Producer می شود. تنها با اجرای متدهای get و خواندن داده توسط Producer، فراخوانی شده و notify ادامه می یابد. توجه داشته باشید که فراخوانی متدهای wait و InterruptedException استثنای را تولید می کند که ما آنرا به تعریف متدهای get و put اضافه کرده ایم که در نتیجه مجبور به استفاده از بلوکهای try/catch در کلاسهای Producer و Consumer شده ایم. اجرای کد فوق نتیجه صحیح زیر را به همراه خواهد داشت.

```
Put: 2
Got: 2
Put: 3
Got: 3
Put: 5
Got: 5
Put: 7
Got: 7
Put: 11
Got: 11
Put: 13
Got: 13
Put: 17
Got: 17
Put: 19
Got: 19
Put: 23
Got: 23
Put: 29
Got: 29
```

اجرا کننده ها

تا اینجا این فصل آموختید که چگونه یک Thread را پیاده سازی و اجرا کنید. اما شیوه ای که تا به حال برای اجرای Thread ها بکاربردیم (فراخوانی start) از آبجکت Thread فقط در برنامه های

کوچک که تعداد کمی Thread وجود دارد مناسب است و در برنامه های بزرگ با تعداد زیاد Thread نیازمند سازماندهی اجرای Thread ها و مدیریت منابع هستیم. برای این منظور، جاوا مجموعه ای از کلاسها را به منظور مدیریت اجرای Thread ها در اختیار می گذارد که تحت عنوان Executor شناخته می شوند. این کلاسها، عمدتاً از یکی از اینترفیس‌های ExecutorService یا ScheduledExecutorService پیاده سازی شده اند. جدول زیر تفاوت این سه اینترفیس را نشان می دهد.

توضیح	اینترفیس
این اینترفیس قابلیتهایی از قبیل مدیریت چرخه حیات Thread را در اختیار می گذارد.	ExecutorService
این اینترفیس که از اینترفیس ExecutorService مشتق شده است علاوه بر قابلیتهای آن، از اجرای زمانبندی شده Thread ها نیز پشتیبانی می کند.	ScheduledExecutorService

جدول ۲-۱۷. اینترفیس‌های Executor

تمام این اینترفیسها در پکیج java.util.concurrent قرار دارند.

توجه: اینترفیس ExecutorService که پدر اینترفیس است خودش فرزند اینترفیس迪گری با نام ScheduledExecutorService است، به عنوان یک Executor پایه شناخته می شود که با داشتن یک متدها execute() می تواند یک Thread را اجرا کند.

در عمل ما همیشه با کلاس‌های سروکار داریم که یکی از اینترفیس‌های ExecutorService یا ScheduledExecutorService را پیاده سازی کرده اند. قبل از اینکه با نحوه استفاده و کارکرد Executor ها آشنا شویم، لازم است با دو اینترفیس دیگر آشنا شویمCallable و Future.

اینترفیس Callable

اینترفیس Callable مشابه اینترفیس Runnable می تواند برای پیاده سازی Thread ها استفاده شود با این تفاوت که اینترفیس Callable می تواند مقداری را که حاصل اجرای Thread است برگرداند. این اینترفیس که در پکیج java.util.concurrent قرار دارد به صورت زیر است.

Thread و همزمانی

```
package java.util.concurrent;

public interface Callable<V> {
    V call() throws Exception;
}
```

توجه: این اینترفیس با `@FunctionalInterface` علامتگذاری شده است که ما برای نمایش ساده تر این اینترفیس، از نوشتن این علامت خودداری کرده ایم. در فصل بیست و سوم این کتاب، عبارتهای لامبدا و کلاسهای تودرتو با علامت `@FunctionalInterface` آشنا می شوید.

بنابراین برای پیاده سازی یک `Thread` می توانیم کلاسی پیاده سازی کنیم که اینترفیس `Callable` را پیاده سازی می کند و طبیعتاً متدهای `call()` آنرا نیز که حاوی منطق `Thread` است را پیاده سازی می کنیم. متدهای `run()` در اینترفیس `Runnable` یک مقدار برگشتی دارد که همان نتیجه اجرای `Thread` است. همچنین متدهای `call()` می توانند `Exception` هم تولید کند که این خاصیت در متدهای `run()` وجود ندارد.

Future

اینترفیس `Future` امکان بیشتری برای کنترل اجرای `Thread` فراهم می کند. این اینترفیس خصوصاً زمانی کاربرد می یابد که با `Thread` ای با یک منطق طولانی مواجه باشیم و با استفاده از آن می توان، وضعیت اجرای آن `Thread` را بررسی کرد یا به اجرای آن `Thread` خاتمه داد. این اینترفیس به صورت زیر در پکیج `java.util.concurrent` قرار دارد.

```
package java.util.concurrent;

public interface Future<V> {

    boolean cancel(boolean mayInterruptIfRunning);

    boolean isCancelled();

    boolean isDone();

    V get() throws InterruptedException, ExecutionException;

    V get(long timeout, TimeUnit unit)
        throws InterruptedException, ExecutionException,
    TimeoutException;
}
```

متدهای cancel() امکان خاتمه منطق Thread را فراهم می کند. متدهای isCancelled() و isDone() وضعیت خاتمه یافتن thread را توسط اجرا کننده Thread را مشخص کند. متدهای get() و isThread() خاتمه موفق Thread را مشخص می کند و متدهای execute() نتیجه اجرای Thread را برمی گردانند. متدهای start() و stop() بدون پارامتر، نتیجه اجرای Thread را برمی گردانند و درصورتیکه Thread خاتمه نیافته باشد منتظر می ماند تا Thread خاتمه یابد و نتیجه را برگرداند. به طور مشابه، متدهای shutdown() و shutdownNow() با پارامتر نیز نتیجه اجرای Thread را برمی گردانند اما به میزان زمان مشخصی (که توسط پارامترهای متدهای shutdown() و shutdownNow() مشخص می شوند) منتظر می ماند و درصورت گذشت آن میزان زمان و حاضر نبودن نتیجه Thread استثنای TimeoutException تولید می کند.

حال که با اینترفیس‌های Future و Callable آشنا شدید، اجازه دهید شما را با مفهوم Thread Pool که نبه آن «مخزن Thread» نیز گفته می شود و نوع خاصی از Executor هستند آشنا کنم.

Thread Pool «مخزن Thread» یک اجراکننده Thread است که هدف آن کاهش سریارهای ناشی از ایجاد آبجکتهاي Thread و اجرای آنهاست. مخزن Thread در واقع با فراهم کردن بستری می تواند اجرای Threadها به لحاظ منابعی که در اختیار می گیرند به صورت بهینه انجام دهد.

Thread Pool ها انواع مختلفی دارند، اما یک نوع بسیار رایج از آنها «مخزن با اندازه ثابت» نامیده می شود که در آن حداقل تعداد Thread هایی که در حال اجرا هستند ثابت است. هریک از آبجکتهاي Thread که در واقع آبجکتهايی هستند که یکی از اینترفیس‌های Callable یا Runnable را پیاده سازی کرده اند به ترتیب به مخزن اضافه می شوند و مخزن در صورتیکه ظرفیت داشته باشد آنها را اجرا می کند و درصورتیکه ظرفیت آن تکمیل باشد آنها را در صف قرار می دهد تا به محض خاتمه اجرای Thread های موجود، موارد صف را جایگزین کند. یکی از موارد کاربرد این گونه Thread Pool در وب-سرورهاست وقتیکه درخواست های مختلف از کاربران وب به سرور می رسد و سرور تصمیم دارد تا ضمن استفاده بهینه از منابع سیستم به تعداد مشخصی از کاربران با یک سقف مشخص سرویس دهد.

برای ایجاد Thread Pool می توان یکی از متدهای استاتیک کلاس Executors که در جدول زیر توضیح داده شده اند را فراخوانی کرد.

نام متدهای	توضیح
newFixedThreadPool(int nThreads)	یک مخزن با اندازه nThreads می سازد.
newCachedThreadPool()	یک مخزن با طول قابل افزایش می سازد. این مخزن نقطه مقابل مخزن با طول ثابت است و برای مواردی مناسب است که Thread ها منطق کوچکی دارند و در زمان کوتاهی به اتمام می رسند.
newSingleThreadExecutor()	یک مخزن ایجای می کند که در هر لحظه فقط یک

Thread ها و همزمانی

را اجرا می کند. Thread ها به صورت Thread متوالی و به ترتیبی که وارد صف می شوند اجرا می شوند.	
--	--

جدول ۳-۱۷. سه متد مهم از کلاس Executors

توجه: هر سه متدی که در جدول فوق توضیح داده شده اند، آبجکتهايی از جنس ExecutorService برمی گردانند. کلاس Executors متدهای دیگری نیز دارد که آبجکتهايی ScheduledExecutorService از آنها خارج از محدوده این کتاب است و در صورت علاقمندی می توانید از روی مستندات javadoc این کلاس آنها را مطالعه کنید.

حال اجازه دهید تا با یک مثال ارتباط اینترفیس‌های Future، Callable و آبجکت Thread Pool را توضیح دهم.

فرض کنید که می خواهید Thread ای پیاده سازی کنید که

```
1 package ir.atlassoft.javase.chapter17;
2
3 import java.util.ArrayList;
4 import java.util.Date;
5 import java.util.List;
6 import java.util.concurrent.Callable;
7 import java.util.concurrent.ExecutionException;
8 import java.util.concurrent.ExecutorService;
9 import java.util.concurrent.Executors;
10 import java.util.concurrent.Future;
11
12 public class MyCallable implements Callable<String> {
13
14     @Override
15     public String call() throws Exception {
16         Thread.sleep(1000);
17         //return the thread name executing this callable task
18         return Thread.currentThread().getName();
19     }
20
21     public static void main(String args[]){
22         //Get ExecutorService from Executors
```

```

23         // utility class, thread pool size is 10
24         ExecutorService executor =
25                 Executors.newFixedThreadPool(10);
26         //create a list to hold the Future
27         //object associated with Callable
28         List<Future<String>> list = new
29                         ArrayList<Future<String>>();
30         //Create MyCallable instance
31         Callable<String> callable = new MyCallable();
32         for(int i=0; i< 100; i++) {
33             //submit Callable tasks to
34             //be executed by thread pool
35             Future<String> future = executor.submit(callable);
36             //add Future to the list,
37             //we can get return value using Future
38             list.add(future);
39         }
40         for(Future<String> fut : list){
41             try {
42                 //print the return value of
43                 //Future, notice the output delay in console
44                 // because Future.get() waits
45                 //for task to get completed
46                 System.out.println(new Date()+" :: "+fut.get());
47             }catch(InterruptedException|ExecutionException e) {
48                 e.printStackTrace();
49             }
50         }
51         //shut down the executor service now
52         executor.shutdown();
53     }
54 }

```

کد ۴

کلاس‌های TimerTask و Timer

در برنامه های جاوا موارد فراوانی وجود دارند که لازم است یک Thread در یک زمان مشخصی اجرا شود. به عنوان نمونه تصور کنید که در برنامه نیاز دارید تا در ساعت ۲ نیمه شب که بار کمی روی برنامه هست یک

Thread ها و همزمانی

سری داده های آماری تولید شوند، یا فایلها بی از یک دایرکتوری کپی شوند، یا داده هایی در پایگاه داده ذخیره شوند. در همه این موارد قطعاً انتظار ندارید که یک اپراتور پای سیستم بنشیند تا در آن زمان با فشار Thread دادن یک کلید Thread مورد نظر را اجرا کند. کلاس `java.util.Timer` امکان اجرای یک Thread در یک زمان مشخص یا اجرای منظم یک Thread در زمان های مشخص (مثل ۲ نیمه شب هر شب) را می دهد.

اما عملیاتی که در قالب یک Thread برای اجرا به آجکت Timer خورانده می شود باید در قالب یک کلاس `java.util.TimerTask` یک کلاس `TimerTask` است و کافیست تا توسط یک کلاس دیگر که حاوی عملیات مورد نظر است توسعه داده شود. کد زیر یک کلاس TimerTask نمونه را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter17;
2
3 import java.util.Date;
4 import java.util.TimerTask;
5
6 public class MyTimerTask extends TimerTask {
7
8     @Override
9     public void run() {
10         System.out.println("Timer task started at:" +
11                         new Date());
12         completeTask();
13         System.out.println("Timer task finished at:" +
14                         new Date());
15     }
16
17     private void completeTask() {
18         try {
19             //assuming it takes 20 secs to complete the task
20             Thread.sleep(20000);
21         } catch (InterruptedException e) {
22             e.printStackTrace();
23         }
24     }
25 }
```

در این مثال، کلاس MyTimerTask صرفا زمان شروع و خاتمه خود را در کنسول برنامه نمایش می دهد. اجرای عملیاتی که در کلاس MyTimerTask تعریف شده اند با استفاده از آبجکتی از کلاس Timer به صورت زیر انجام می شود.

```
public static void main(String args[]) {
    TimerTask timerTask = new MyTimerTask();
    //running timer task as daemon thread
    Timer timer = new Timer(true);
    timer.scheduleAtFixedRate(timerTask, 0, 10*1000);
    System.out.println("TimerTask started");
    //cancel after sometime
    try {
        Thread.sleep(120000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    timer.cancel();
    System.out.println("TimerTask cancelled");
    try {
        Thread.sleep(30000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

قسمت بسیار مهم در کد فوق، فراخوانی متده است () scheduleAtFixedRate

```
timer.scheduleAtFixedRate(timerTask, 0, 10*1000);
```

که مشخص می کند MyTimerTask می بایست بلافاصله (پارامتر دوم با مقدار 0) اجرا شود و در بازه های زمانی ۱۰ ثانیه ای (پارامتر سوم با مقدار $10 * 1000$) تکرار شود. کلاس Timer دارای متدهای دیگری نیز هست که امکان اجرای یک TimerTask را فقط یکبار می دهد. این کلاس همچنین دارای متده است () cancel که امکان انصراف یا متوقف کردن عملیاتی که توسط TimerTask تعریف شده را می دهد.

خلاصه

Thread ها اجزای مستقل یک برنامه هستند که می توانند به صورت همزمان اجرا شوند منظور از اجرای همزمان این است که اجزای مستقل برنامه با مدیریت سیستم عامل و توسط CPU کامپیوتر لابالی یکدیگر اجرا می شوند.

هر برنامه ای که نوشته می شود حداقل شامل یک Thread است که به آن Thread اصلی گفته می شود، Thread اصلی شامل همان برنامه ای است که در حال اجراست اما ممکن است در عمل یک برنامه توسط سیستم عامل به Thread های جداگانه ای شکسته شود، به Thread هایی که در دل Thread اصلی متولد می شوند Thread های فرزند گفته می شود.

در جوا این امکان وجود دارد تا برنامه نویس بتواند به صورت دلخواه اجزای مستقل برنامه را به صورت Thread تعریف کند، بدین ترتیب آن اجزا می توانند به صورت همزمان اجرا شوند. برای اینکه یک کلاس به Thread یک Thread اجرا شود لازم است که از کلاس Thread مشتق شود یا اینترفیس Runnable را پیاده سازی کند. هر کلاسی که به عنوان Thread تعریف می شود باید دارای متدهای run() و

public void run()
باشد که حاوی عملیاتی است که آن Thread انجام می دهد.
برای اجرای یک Thread ابتدا باید آبجکتی از روی آن ساخته شده و سپس متد start() آن فراخوانی شود.
Thread ها می توانند با استفاده از متدهای () notifyAll() و () wait() با یکدیگر ارتباط برقرار نموده و همزمان شوند.

Thread و Deadlock از جمله مسایل مربوط به Thread هاست، در Deadlock دو منتظر آزاد سازی منبعی هستند که در اختیار طرف مقابل قرار دارد و هیچکدام نیز منبع مورد نظر طرف مقابل را آزاد نمی کنند. در Starvation Thread یک منبع را تا زمان طولانی در اختیار می گیرد و Thread دیگر برای مدتی طولانی منتظر نگه می دارد.
Thread ها را می توان معلق نمود، متوقف کرده یا اجرای آنها را پس از توقف ادامه داد.

تمرینات

- (۱) برنامه ای بنویسید که دو Thread در آن وجود دارد هر Thread یک مدت زمان تصادفی به خواب می رود و قبیل از خواب بیدار می شود، Thread دیگر را خواب می کند و جمله ای را در کنسول برنامه چاپ می کند سپس Thread دوم را بیدار می کند Thread دوم ضمن خواباندن Thread اول، همان کاری را انجام می دهد که Thread اول قبل انجام داده بود.
- (۲) برنامه ای بنویسید که با استفاده از سه Thread یک فایل را از یک شاخه در شاخه دیگر کمی می کند Thread اول مسئول خواندن داده های فایل است، Thread دوم مسئول نوشتن داده در فایل مقصد است و Thread سوم نیز میزان پیشرفت در کمی داده ها را گزارش می کند.
- (۳) مثال پینگ-پنگ را که در ابتدای این فصل دیدید با استفاده از مکانیسم `wait/notify` اصلاح کنید بگونه ای که بعد از هر پینگ فقط یک پنگ بیاید.
- (۴) برنامه ای بنویسید که به صورت بازگشتی محتوای یک دایرکتوری را داخل دایرکتوری دیگر کمی کند. به ازای هر فایل یک Thread ایجاد شود (حداکثر ۱۰ Thread) که کار کمی کردن فایلها را انجام می دهنند.
- (۵) برنامه ای بنویسید که لیستی از نام فایلها را از طریق خط فرمان دریافت می کند و تعداد خطوط هر فایل را چاپ می کند. برنامه به ازای هر فایل باید یک Thread ایجاد کند که کار بدهست آوردن تعداد خط هر فایل و چاپ آنرا به عهده دارد. برای بدهست آوردن تعداد خطوط یک فایل از کلاس `java.io.LineNumberReader` کمک بگیرید. برای انجام این تمرین باید کلاسی مثلاً با نام `LineCounter` پیاده سازی کنید که اینترفیس `Runnable` را پیاده سازی کرده است.

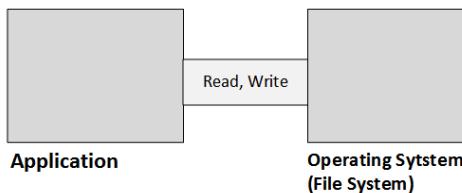


ارتباط با پایگاه داده

همانطور که اموال و دارایی عنصر مهم زندگی انسانها محسوب می‌شوند، «داده‌ها» نیز بخش بالهیت و حیاتی برنامه‌های کامپیوتری هستند. برنامه‌ها وظیفه دارند تا برای کاربران امکان ورود داده‌ها را فراهم کنند، داده‌ها در فرمت مناسب نمایش دهند، و امکان ویرایش و حذف داده‌ها را نیز به کاربران بدهند. به عنوان مثال، یک برنامه حسابداری را تصور کنید. حسابدار از این برنامه استفاده می‌کند تا اطلاعاتی را وارد کامپیوتر کند، آنها را پردازش و ذخیره کند و در صورت لزوم ویرایش یا حذف نماید. اما در میان همه فعالیتهایی که در ارتباط با داده‌ها انجام می‌شود، یک فعالیت مهم وجود دارد و آن «ذخیره و بازیابی» داده‌هاست زیرا اگر برنامه حسابداری یا هر برنامه دیگری نتواند اطلاعات وارد شده را به صورت دائمی روی کامپیوتر ذخیره کند با توقف برنامه یا خاموش شدن کامپیوتر، تمام اطلاعات از حافظه موقت (RAM) پاک شده و با اجرای مجدد برنامه، لازم است دوباره داده‌ها وارد شود.

نرم افزارهای قدیمی داده‌های خود را داخل فایل ذخیره می‌کردند. از آنجاییکه فایلها روی هارد دیسک به صورت دائمی (بلندمدت) ذخیره می‌شوند، با توقف برنامه یا خاموش شدن سیستم، داده‌ها روی هارد دیسک باقی می‌مانند.

اگرچه استفاده از فایل برای نگهداری داده‌ها راهکار خوبی بود اما به مرور که تکنولوژی نرم افزار توسعه پیدا کرد نرم افزارها هم از حالت «یک کاربری» به «چند کاربری» ارتقا یافتند و حجم داده‌ها و خصوصاً پیچیدگی آنها افزایش یافت. ظهور نرم افزارهای مدرن نیازمندیها و چالشهای جدیدی بوجود آورد که دیگر استفاده از فایل جوابگوی نیاز «ذخیره و بازیابی» داده‌ها نبود.



شکل ۱۱-۱. ذخیره سازی داده های یک برنامه در سیستم فایل

عمده ترین اشکالات استفاده از فایل به شرح زیر است.

۱- استقلال داده ها. داده هایی که در فایل نگهداری می شوند فقط توسط همان برنامه قابل فهم و پردازش هستند. در دنیایی که داده ها محوری ترین بخش نرم افزارها هستند مخفی کردن داده ها پشت نرم افزارها عیب بزرگی خواهد بود. ما به مکانیسمی نیاز داریم که به داده ها هویت مستقلی بدهد تا به جای اینکه توسط یک برنامه قابل فهم و استفاده باشد توسط هر برنامه ای فهمیده و استفاده شوند. این خصوصیت باعث انعطاف پذیری نرم افزارها شده و توسعه آنها را آسان و کم هزینه می کند.

۲- امنیت. در عین حال که میخواهیم داده های ذخیره شده توسط هر برنامه دیگری قابل فهم و پردازش باشد، انتظار داریم داده ها از دسترسی های غیرمجاز محافظت شوند و توسط افراد یا نرم افزارهای تایید نشده قابل استفاده نباشد.

۳- تراکنش. تراکنش یکی از کلیدی ترین ویژگیها در ذخیره و بازیابی داده هاست. برای درک مفهوم تراکنش، فرض کنید شخصی می خواهد مبلغی را از حساب خود به حساب دیگری واریز کند. برای این منظور باید آن مبلغ از حساب وی کسر شود و به همان اندازه حساب فرد دوم افزایش یابد اگر درست بعداز اینکه مبلغ از حساب وی کسر می شود و قبل از اینکه مبلغ حساب دوم افزایش یابد، سیستم دچار اختلال شود (مثلاً برق قطع شود)، پول از یکی از حسابها برداشت شده ولی به هیچ حساب دیگری واریز نشده است. تراکنش مکانیسمی است که با استفاده از آن تضمین می شود که دو عمل یا هردو با موقفيت انجام می شوند یا هیچ کدام انجام نمی شوند. چنین امکانی در ذخیره سازی فایل وجود ندارد، زیرا همچنانکه دیدید برداشت از حساب انجام شد و واریز به حساب دیگر اعمال نشد.

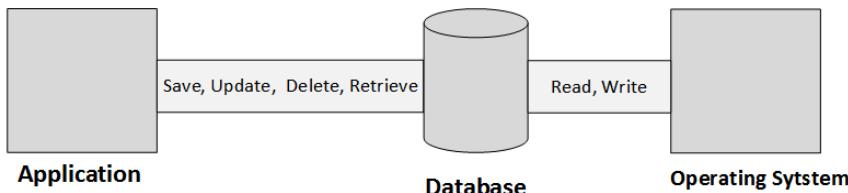
۴- زبان مشترک. اگر داده ها موجودات مستقلی باشند که بتوان با هر نرم افزاری به آنها دسترسی پیدا کرد و آنها را فهمید نیازمند یک زبان مشترک برای تعامل با آنها هستیم. زبان مشترک امكان می دهد تا برنامه ها مستقل از اینکه به چه زبان برنامه نویسی تولید شده اند یا روی چه پلتفرمی (ویندوز، لینوکس، ...) اجرا می شوند بتوانند با یک زبان مشترک با داده ها ارتباط برقرار کرده و آنها ذخیره، بروزرسانی، حذف، یا جستجو کنند.

ارتباط با پایگاه داده

چالشها و اشکالات فوق، باعث تولید نرم افزارهایی شده که به آنها «پایگاه داده» گفته می‌شود. کار پایگاه داده‌ها فراهم کردن سرویس‌هایی است که به صورت تخصصی و به بهترین شکل و کارایی، ذخیره و بازیابی داده‌ها را انجام می‌دهند و برنامه‌ها و برنامه نویسان را از مسایل و چالش‌های مربوط به نگهداری داده‌ها نجات می‌دهند.

پایگاه داده‌ها را می‌توان به دو نوع پایگاه داده‌های «رابطه‌ای» و «غیررابطه‌ای» تقسیم بندی کرد. در پایگاه داده‌های رابطه‌ای که موضوع این فصل است، داده‌های مختلف در قالب جداول داده سازماندهی می‌شوند. یک جدول پایگاه داده شبیه یک جدول در دنیای واقعی است که لیست افراد، کالاهای، یا هر داده دیگری در آن نمایش داده می‌شود. واضح است که مکانیسم ذخیره سازی جداول، چالش پایگاه داده است و به برنامه‌ها ارتباطی ندارد.

پایگاه داده‌های غیررابطه‌ای نسل جدیدتری از پایگاه داده‌ها هستند. اگرچه امروزه این نوع پایگاه داده‌ها هم طرفدار پیدا کرده و در جاوا امکان ارتباط با آنها وجود دارد اما به دلیل قدمت پایگاه داده‌های رابطه‌ای، ساختار قابل فهم و پشتیبانی از SQL، وجود متخصصان زیادی که در این حوزه فعالیت می‌کنند هنوز استفاده از پایگاه داده‌های رابطه‌ای انتخاب اول برنامه نویسان و معماران نرم افزار است.



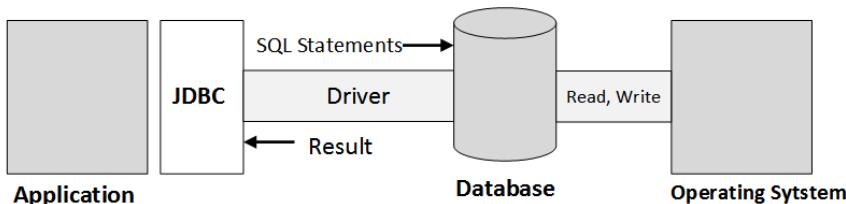
شکل ۱۱-۲. ذخیره سازی داده‌های یک برنامه با استفاده از پایگاه داده

SQL

پایگاه داده‌ها نرم افزارهایی هستند که ممکن است به یک زبان (جاوا، سی،...) نوشته و روی یک پلتفرم خاص (ویندوز، لینوکس،...) اجرا شوند تا سرویس‌های ذخیره و بازیابی داده‌ها را برای برنامه‌ها فراهم کنند. اما از آنجاییکه پایگاه داده و برنامه دو نرم افزار مستقل هستند و تلاش شده (به دلایلی که در بالا گفته شد)مستقل بمانند، زبانی با نام SQL ابداع شده است تا برنامه‌ها بتوانند با پایگاه داده‌ها صحبت کنند و منظور خود در ذخیره و بازیابی داده‌ها را مشخص نمایند. SQL برای ذخیره، ویرایش، حذف و جستجوی داده‌ها و به طور کلی کار با پایگاه داده‌ها یک گرامر و دستور زبان تعریف می‌کند که توسط هر پایگاه داده‌ای قابل درک است. به عنوان مثال عبارتی که با **INSERT INTO** شروع می‌شود یک دستور برای ذخیره داده است.

Driver

اگرچه با وجود SQL، برنامه‌ها می‌توانند با هر پایگاه داده ای (اوکل، DB2، MySQL) یا هر پایگاه داده دیگری) صحبت کننداما نیازمند واسطه ای هستیم که دستورات SQL ای که برنامه صادر می‌کند را به پایگاه داده منتقل کند و نتیجه اجرای دستورات را به برنامه برگرداند. به این واسطه، درایور (driver) گفته می‌شود.



شکل ۳-۱۷. جایگاه SQL و پایگاه داده

درایور یک برنامه کوچک است که بین برنامه و پایگاه داده قرار می‌گیرد و می‌تواند همانند پلی ارتباط این دو دنیای متفاوت (پایگاه داده و برنامه) را برقرار کند. وجود درایور باعث می‌شود که پایگاه داده و برنامه همچنان مستقل باقی بمانند، هرچند که خود درایور مستقل نیست و خاص یک پایگاه داده نوشته می‌شود. هر درایور عموماً توسط سازنده پایگاه داده تولید می‌شود و به پایگاه داده و حتی نسخه پایگاه داده و به زبان جاوا و حتی نسخه جاوا وابسته است. بنابراین اگر پایگاه داده خود را مثلاً از نسخه ۱۱ به نسخه ۱۲ یا جاوای برنامه را از ۱,۸ به ۱,۹ ارتقا دهید باید متناسب با آنها درایور برنامه را نیز تغییر دهید.

توجه: ممکن است بتوان در نسخه‌های بالاتر جاوا یا پایگاه داده از درایور نسخه پایین تر استفاده نمود اما قطعاً قابلیتهاي از قبیل کارایی، پشتیبانی از یک خصوصیت جدید پایگاه داده، ... را از دست می‌دهید.

JDBC Driver

اگر می‌خواهید در یک برنامه جاوا با یک پایگاه داده ارتباط برقرار کنید باید از JDBC Driver استفاده کنید. JDBC API تولید شده است که به زبان جاوا و منطبق بر JDBC Driver نوع خاصی از درایور است.

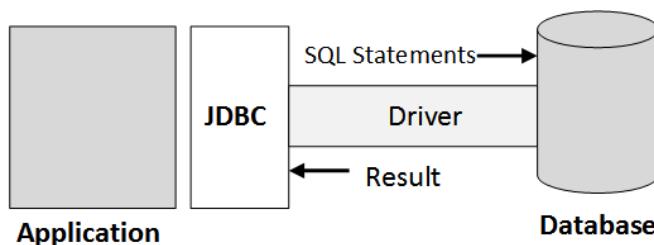
توجه داشته باشید که در برنامه‌های جاوا می‌توان از درایورهای غیرجاوایی از قبیل ODBC نیز استفاده کرد که در اینصورت می‌بایست با استفاده از کلاس واسطه JDBCBridge آنرا با برنامه جاوا سازگار کرد.

ارتباط با پایگاه داده

در برنامه های جاوایی ترجیح بر استفاده از JDBC Driver است زیرا مستقل از سیستم عامل است و می تواند در سیستم عامل غیر از «ویندوز» نیز استفاده شود. برای دریافت JDBC Driver به وب سایت پایگاه داده مورد نظرتان مراجعه کنید. با داشتن درایور، آماده برنامه نویسی برای ذخیره و بازبینی داده ها هستید. در جاوا مجموعه ای از کلاسها و اینترفیسها که تحت عنوان JDBC API شناخته می شوند برای ارتباط و تعامل با پایگاه داده معرفی شده اند. توجه داشته باشید که کلاسها و اینترفیسها JDBC API در پشت صحنه از درایور برای ارتباط با پایگاه داده استفاده می کنند هرچند این موضوع در کدهای ما ظهرور پیدا نمی کند.

JDBC API

همانطور که گفته شد، JDBC API مجموعه ای از کلاسها و اینترفیس های جاواست که برای تعامل با پایگاه داده به عنوان بخشی از کتابخانه جاوا طراحی و پیاده سازی شده است. هر برنامه جاوا از JDBC API برای ارتباط با پایگاه داده، اجرای جملات SQL، دریافت و پردازش نتایج حاصل از اجرای SQL استفاده می کند. همانطور که در شکل ۱۸-۴ مشاهده می کنید، JDBC API متکی به درایور است که در پشت آن قرار دارد به طوریکه باعث می شود کاملا جزئیات درایور از برنامه مخفی بماند.



شکل ۱۸-۴. نقش JDBC در ارتباط با پایگاه داده

نصب PostgreSQL

به عنوان اولین مرحله در برنامه نویسی برای کار با پایگاه داده لازم است پایگاه داده را روی سیستم خود نصب کنید.

ما در این کتاب از پایگاه داده PostgreSQL استفاده می کنیم که پرطرفدارترین و سریع ترین پایگاه داده منبع باز دنیاست. اما شما می توانید از هر پایگاه داده دیگری استفاده کنید.

توجه: PostgreSQL یک نرم افزار مدیریتی به نام PgAdmin دارد که امکان می دهد از طریق یک واسط کاربری خوش دست، پایگاه داده ای را ایجاد کنیم، جداول و ستونهایی به آن اضافه کنیم، و تمام

موجودیت‌های پایگاه داده (stored procedure, sequence, view, ...) را مدیریت کنیم. این نرم افزار هم‌زمان با نصب پایگاه داده PostgreSQL روی سیستم شما نصب می‌شود و از طریق منوی Start در ویندوز به سادگی می‌توانید آنرا بیابید.

آشنایی با SQL

در اینجا به صورت مختصر نگاهی اجمالی به برخی خصوصیات SQL خواهیم انداخت. بررسی و بحث در مورد جزییات SQL خارج از محدوده این کتاب است. در صورتی که دانش کافی نسبت به SQL دارید می‌توانید از این قسمت عبور کنید یا اگر شناخت کمی نسبت به SQL ندارید پس از مطالعه این بخش می‌توانید از کتابها و منابع آنلاین که بسیار ساده و مناسب هم هستند بهره ببرید.

جدول

اساس کارکرد پایگاه داده‌های رابطه‌ای، موجودی با نام «جدول» است که داده‌هایی مشابه و از یک جنس (مثلاً اشخاص، کالاهای...) را نگهداری می‌کنند. جدول در پایگاه داده مشابه جدولی که روی کاغذ ترسیم می‌شود از سطرهای (داده‌های جدول) و ستونها (جزییات داهای) تشکیل شده است. شکل زیر، جدول person را که معرف داده‌های اشخاص است را نشان می‌دهد.

id [PK] integer	first_name character var	last_name character var	ssn character var
1	Reza	Hasani	230923838
2	Mehdi	Mohammadi	337389020
3	Hesam	Nezampour	838388383

شکل ۵.۱۱-۵. جدول Person در پایگاه داده

برای اینکه هر ردیف جدول متمایز از ردیفهای دیگر باشد یک ستون شناسه (ID) در جداول وجود دارد که به آن «کلید اصلی» گفته می‌شود و مقدار آن در برای هر رکورد از جدول یک مقدار منحصر بفرد است. در واقع کلید اصلی، به هر ردیف هویت مستقل می‌بخشد تا با استفاده از آن بتوان به آن ردیف اشاره کرد، یا آنرا بازیابی، بروزرسانی، یا حذف نمود.

ایجاد جدول در پایگاه داده از طریق دستور SQL که قالبی به صورت زیر دارد انجام می‌شود.

ارتباط با پایگاه داده

```
CREATE TABLE table_name (
    column1 datatype1,
    column2 datatype2,
    column3 datatype3,
    ....
);
```

در اینجا `table_name` نام جدول، و `column3`, `column2`, `column1`..نام ستونهای آن و `datatype3`, `datatype2`, `datatype1` نیز به ترتیب جنس ستونها را مشخص می کنند. مثال زیر یک نمونه از ایجاد جدول را نشان می دهد.

```
CREATE TABLE person (
    id INTEGER NOT NULL,
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    ssn VARCHAR(30),
    PRIMARY KEY(id)
);
```

در اینجا، جدولی با نام `person` و با ستونهای `id`, `ssn` و `last_name`, `first_name`, `id` طراحی شده است که `id` از جنس عدد (INTEGER) و بقیه از جنس متن با طول ۳۰ کاراکتر هستند. علاوه بر INTEGER و VARCHAR انواع داده های دیگری نیز توسط SQL پشتیبانی می شوند که برخی از آنها در جدول زیر نشان داده شده اند.

داده معادل در جاوا	توضیح	داده SQL
<code>java.sql.Date</code>	تاریخ	<code>date</code>
<code>java.sql.Date</code>	زمان	<code>time</code>
<code>java.sql.Date</code>	تاریخ و زمان	<code>timestamp</code>
<code>Short</code>	یک بیت	<code>bit</code>
<code>Character</code>	یک کاراکتر	<code>char</code>
<code>String</code>	یک رشته	<code>varchar</code>
<code>Boolean</code>	یک مقدار <code>false</code> یا <code>true</code>	<code>boolean</code>
<code>Integer</code>	مقدار عدد صحیح	<code>integer</code>
<code>Long</code>	عدد صحیح	<code>bigint</code>
<code>String</code>	رشته یا متن	<code>text</code>

جدول ۱۱-۱. داده های SQL و معادل آنها در جاوا

جدوال پایگاه داده می توانند با یکدیگر ارتباط داشته باشند. مثلا ممکن است بخواهیم هر ردیف جدول افراد را با یک ردیف از جدول شهر مرتبط کنیم و به این ترتیب نشان دهیم که شهر محل سکونت آن فرد کدام شهر است. اینکار از طریق ایجاد یک ستون در جدول مبدا که مقدار آن برابر با «کلید اصلی» در جدول مقصد است انجام می شود. به این ستون «کلید خارجی» گفته می شود.

به عنوان مثال اگر جدول CITY که معرف داده های شهرهاست به صورت زیر ایجاد شده باشد

```
CREATE TABLE city (
    id INTEGER NOT NULL,
    name VARCHAR(30),
    PRIMARY KEY(id)
);
```

که در آن id کلید اصلی و name یکی از ستونهای آن است، میتوان جدول person را به صورت زیر ایجاد کرد تا هر شخص با یک شهر مرتبط شود.

```
CREATE TABLE person (
    id INTEGER NOT NULL,
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    ssn VARCHAR(30),
    city_id INTEGER,
    PRIMARY KEY(id),
    FOREIGN KEY (city_id) REFERENCES city(id)
);
```

در اینجا علاوه بر اضافه شدن ستون city_id به جدول person، با استفاده از جمله

```
FOREIGN KEY (city_id) REFERENCES city(id)
```

مشخص شده که ستون city_id از جدول person به ستون id از جدول city کلید خارجی دارد.

در پایگاه های رابطه ای مفاهیم دیگری از قبیل view و stored procedure هم وجود دارند که بحث در مورد آنها خارج از محدوده این کتاب است.

ذخیره یک رکورد

در SQL از عبارت INSERT INTO برای ذخیره یک رکورد در یک جدول استفاده می شود. شکل کلی این دستور به صورت زیر است

ارتباط با پایگاه داده

```
INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);
```

در اینجا `table_name` نام جدول است که بلافاصله بعد از آن نام ستونهای این جدول داخل پرانتز که با کاما جدا شده اند قرار گرفته است و مقادیر این ستونها بعد از کلمه `VALUES` به ترتیب داخل پرانتز مشخص شده اند.

به عنوان مثال جمله زیر یک رکورد در جدول `city` ذخیره می کند.

```
INSERT INTO city (id, name) values (1, 'تهران');
```

همچنین دستور زیر یک رکورد در جدول `person` ثبت می کند که به ردیف جدول `city` کلید خارجی دارد.

```
INSERT INTO person (id, first_name, last_name, ssn, city_id) values (1, 'حسین خانی', 'رضایی', 923882922, 1);
```

نکته حائز اهمیت در جملات فوق، استفاده از کوتیشن " در مشخص کردن مقادیر متغیر است. این در حالی است که مقدار `id` و `city_id` که عدد هستند بدون " نوشته شده اند.

بروزرسانی یک رکورد

عبارت `UPDATE` برای بروزرسانی یک جدول به صورت زیر استفاده می شود.

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

نام جدول است که بعد از آن از کلمه `SET` برای مشخص کردن مقادیر جدید ستونهای `column1`, `column2`, ..., `columnN` محدوده `WHERE` شده است. کلمه `WHERE` رکورد یا رکوردهایی را که باید بروزرسانی شوند را مشخص می کند. اگر از `WHERE` استفاده نشود، تمام رکوردهای جدول `table_name` با مقادیر جدید بروزرسانی می شوند.

به عنوان مثال برای بروزرسانی رکوردي از جدول `person` که کلید اصلی آن 1 است از جمله زیر استفاده می شود

```
UPDATE person  
SET first_name= 'Kamal', last_name = 'Jafari'  
WHERE id=1;
```

حذف یک رکورد

حذف یک رکورد از جدول با استفاده از دستور زیر امکانپذیر است.

```
DELETE FROM table_name
WHERE condition;
```

که در آن `table_name` نام جدول و `condition` محدوده رکوردهایی که باید حذف شوند را مشخص می کند. برای مثال، جمله زیر رکوردي که کلیداصلی آن 1 است را حذف می کند.

```
DELETE FROM person WHERE id=1;
```

دقت کنید که حذف این رکورد نباید قوانین پایگاه داده را نقض کند، که در اینصورت پایگاه داده آن رکورد را حذف نخواهد کرد و پیغام خطای مناسب صادر می کند. به عنوان مثال، اگر رکوردي از یک جدول دیگر به این رکورد کلید خارجی داشته باشد، پایگاه داده مانع حذف این رکورد می شود زیرا در غیراينصورت رکورد جدول دیگر به رکوردي کلید خارجی دارد که وجود ندارد و این قاعده خلاف قوانین پایگاه داده است.

جستجوی رکورد

در اینجا سه عمل اصلی «ایجاد، بروزرسانی، حذف» رکوردهای پایگاه داده را آموختید. اما چگونه می توانیم مقادیر رکوردهای موجود در یک جدول را مشاهده کنیم. اینکار از طریق دستور `SELECT` که قالبی به صورت زیر دارد امکانپذیر است.

```
SELECT column1, column2, ...
FROM table_name;
```

در اینجا `table_name` نام جدول، و `column1` و `column2` نام ستونهای جدول است که می خواهیم مقادیر آنها را مشاهده کنیم. به جای نام ستونهای جدول می توان از علامت * نیز استفاده کرد که در اینصورت معنی آن تمام ستونهای جدول است.

```
SELECT * FROM table_name;
```

به صورت اختیاری، می توان عبارت `WHERE conditions`

را به انتهای جمله فوق اضافه کرد تا رکوردهای خاصی که شرایط خاصی دارند را از جدول انتخاب نمود. به عنوان مثال، جمله زیر رکوردي از جدول `person` که مقدار ستون `ssn` آن برابر با 93282882 را برミ گردد.

```
SELECT * FROM person WHERE ssn='93282882';
```

استفاده از join

اگر یک جدول کلیدخارجی به جدول دیگر داشته باشد، در اینصورت جستجوی همزمان جداول از طریق `join` امکانپذیر است.

ارتباط با پایگاه داده

```
SELECT person.*, city.*  
FROM person  
INNER JOIN city ON person.city_id=city.id;
```

در اینجا تمام رکوردهای جدول `person` و رکوردهای متناظر با آنها که در جدول `city` قرار دارد و از طریق کلید خارجی `city_id` در جدول `person` به کلید اصلی `id` در جدول `city` ارتباط دارند بارگذاری شده است. با اجرای دستور SQL فوق، خروجی زیر را مشاهده خواهید کرد.

دقت کنید که از

```
INNER JOIN city ON person.city_id=city.id
```

در انتهای دستور SELECT استفاده شده است. JOIN انواع مختلفی دارد و JOIN INNER یکی از آنهاست.

ارتباط با پایگاه داده

اینک که پایگاه داده را نصب کرده اید و با مفاهیم SQL نیز آشنا شده اید آماده ایم تا در یک برنامه جاوا به پایگاه داده متصل شویم و جملات SQL را اجرا کنیم. کد زیر، مراحل ارتباط با یک پایگاه داده اجرای یک جمله SQL (ایجاد، بروزرسانی، یا حذف) را نشان می دهد.

```
import java.sql.* ;  
  
. . .  
  
Class.forName("...");  
String url = "...";  
String username = "...";  
String password = "...";  
  
Connection cn = DriverManager.getConnection(url, username,  
password);  
Statement st = cn.createStatement();  
  
String sql ="..."; //update sql  
  
st.executeUpdate(sql);  
st.close();  
cn.close();  
. . .
```

این مراحل را می توان به صورت زیر خلاصه کرد.

۱- پکیج `import java.sql;` را کنید.

```
import java.sql.*;
```

۲- کلاس درایور را بارگذاری کنید.

```
Class.forName("...");
```

۳- به پایگاه داده متصل شوید.

```
Connection cn = DriverManager.getConnection(url, username,
```

```
password);
```

۴- یک Statement بسازید

```
Statement st = cn.createStatement();
```

۵- دستور SQL را تعریف کنید.

```
String query = "INSERT INTO PERSON (NAME, FAMILY) "+
```

```
"VALUES ('ali', 'alavi')";
```

۶- دستور SQL خود را اجرا کنید.

```
int result = st.executeUpdate(query);
```

ارتباط با پایگاه داده

۷- نتایج اجرای Statement را بازیابی کنید (درصورتیکه نتیجه اجرای دستور SQL برایتان حائز اهمیت است).

```
System.out.println(result+" rows effected");
```

۸- با فراخوانی متد () close از آبجکتهای Statement و Connection منابع اشغال شده را آزاد کنید.

```
st.close();
```

```
cn.close();
```

در ادامه هریک از این مراحل با جزئیات تشریح شده اند.

پکیج import java.sql را کنید.
import java.sql.*;

* java.sql کلاسها و اینترفیس‌های JDBC API است، یعنی همه کلاسها و اینترفیس‌هایی که برای ارتباط با پایگاه داده، اجرای جملات SQL و دریافت نتایج نیاز است. مهمترین کلاس این مجموعه Connection و DriverManager اینترفیس‌های آن Statement و ResultSet هستند که با آنها بیشتر آشنا خواهید شد.

بارگذاری درایور

درایور پایگاه داده، یک فایل با پسوند jar است که از سایت پایگاه داده قابل دانلود است. فایل JAR نوعی فایل فشرده مثل فایل ZIP است که توسط جاوا پشتیبانی می شود و معمولاً برای بسته بندی کلاس‌های کامپایل شده جاوا و ارایه کتابخانه‌های جاوای استفاده می شود. درایور پایگاه داده نیز که یک کتابخانه جاوای است حاوی مجموعه‌ای از کلاس‌های جواست که در قالب یک فایل JAR بسته بندی شده و از سایت پایگاه داده قابل دانلود است. وقتی این فایل را دانلود کردید باید آنرا در classpath برنامه قرار دهید. classpath متغیری است که در سیستم تنظیم می شود و مسیر تمام کلاسها و فایلهای JAR خارجی (که بخشی از JDK نیستند) و برای اجرای برنامه به آن نیاز است را مشخص می کند. متغیر classpath در ویندوز از طریق اجرای دستور زیر در پنجره فرمان تنظیم می شود

```
set CLASSPATH=c:\jars\postgresql-9.1.jar;
```

تنظیم CLASSPATH در لینوکس به صورت زیر است

```
export CLASSPATH=/usr/jars/postgresql-9.1.jar;
```

اگر بخواهید بیش از یک فایل JAR را در CLASSPATH قرار دهید می باشد، مسیر فایلهای دیگر را به انتهای جملات فوق اضافه کنید و از ; برای جدا کردن مسیر فایلهای استفاده کنید. به موضوع پایگاه داده برگردیم! پس از اینکه درایور را دانلود کردید می باشد کلاس درایور (که یکی از کلاسهای داخل فایل JAR است) را به صورت زیر در برنامه بارگذاری کنید.

```
Class.forName("org.postgresql.Driver");
```

در این مثال org.postgresql.Driver کلاس درایور پایگاه داده Postgresql است که از آنجاییکه قصد داریم به این پایگاه داده متصل شویم آنرا انتخاب کرده ایم. نام کلاس درایور Driver و پکیج آن است که نام کامل آن یعنی org.postgresql.Driver به متند Class.forName() ارسال شده است. نتیجه این فراخوانی، بارگذاری کلاس درایور در حافظه است تا در خطهای بعدی برنامه بتوان از طریق آن به پایگاه داده متصل شد. وقت داشته باشید که درایور پایگاه داده هرچه که باشد، تاثیری در کدهای بعدی نخواهد داشت اما اجرای کدهای بعدی هرچند از دید ما مخفی است ولی تاثیر دارد.

ایجاد Connection برای ارتباط با پایگاه داده

بعد از اینکه درایور پایگاه داده را بارگذاری کردید آمده هستیم تا به پایگاه داده متصل شویم. اتصال به پایگاه داده با ایجاد آبجکت java.sql.Connection می شود. اما یک اینترفیس است و نمی توان از روی آن آبجکت ساخت، و باید از روی کلاسی که این اینترفیس را پیاده سازی کرده است آبجکت بسازیم. اما کلاسی که این اینترفیس را پیاده سازی کرده بخشی از درایور است (در فایل JAR درایور قرار دارد) و ترجیح ما این است که برنامه مان را مستقل از پایگاه داده و درایور آن نگهداریم تا در آینده بتوانیم با حداقل تغییرات در کدهای برنامه، پایگاه داده را در صورت لزوم تغییر دهیم. به همین دلیل، کلاس DriverManager کلاس java.sql.DriverManager می تواند در پشت صحنه متناسب با کلاس درایوری که در مرحله قبل بارگذاری شد آبجکت Connection مربوطه را ایجاد کند.

کلاس DriverManager مهمترین کلاس پکیج java.sql است و حاوی متدهایی برای مدیریت ارتباط با پایگاه داده است. این کلاس متد () getConnnection برای ایجاد آبجکت Connection ارایه

ارتباط با پایگاه داده

می کند. در این کلاس سه متد (`getConnection()` با پارامترهای مختلف) برای ایجاد آبجکت پیاده سازی شده است که بسته به شرایط باید یکی از آنها را فراخوانی کنید.

```
public static synchronized Connection  
    getConnection(String url) throws SQLException  
  
public static synchronized Connection getConnection (   
    String url,  
    String user,  
    String password) throws SQLException  
  
public static synchronized Connection  
    getConnection(String url, properties props)  
    throws SQLException
```

در هر سه متد، اولین پارامتر، رشته URL پایگاه داده است. اصطلاح URL را قبلا در اینترنت و آدرس‌های اینترنتی شنیده اید، که معرف آدرس یک صفحه یا سایت اینترنتی است. همانطور که URL در اینترنت یک آدرس منحصر‌بفرد برای یک فایل یا سایت اینترنتی نشان میدهد، در اینجا نیز URL معرف یک آدرس (نام) منحصر‌بفرد برای یک سرویس یا اپلیکیشن خاص است. به عنوان نمونه رشته

`jdbc:postgresql://localhost:5432/db`

URL یک پایگاه داده postgresql با نام db روی همین کامپیوتر (localhost) است که به پورت 5432 گوش می کند. اگر پایگاه داده روی کامپیوتر دیگری باشد باید به جای localhost، آی پی یا نام آن کامپیوتر را بیاورید.

در برنامه‌های عملیاتی که با داده‌های حیاتی سروکار داریم، امنیت داده‌ها اهمیت زیادی دارد. به همین دلیل پایگاه داده‌ها با فراهم کردن مفاهیم امنیتی از قبیل کاربر، رمز ورود، و مجوزها دسترسی به داده‌ها را کنترل می کنند و هر برنامه‌ای که بخواهد به پایگاه داده متصل شود، لازم است تا نام و رمز کاربری خود را وارد کند. در اینجاست که فرم دوم و سوم متد (`getConnection()`) که اجازه ورود نام و رمز را می دهد کاربرد پیدا می کند.

در فرم دوم، شما صریحاً می توانید نام کاربری و رمز مورد نیاز برای کار با پایگاه داده را مشخص کنید.

```
DriverManager.getConnection(   
    "jdbc:postgresql://localhost:5432/db",  
    "myusername",  
    "easy");
```

در فرم سوم، پارامتر `props` هرگونه اطلاعات اضافه‌ای که برای ایجاد آبجکت `Properties` است (از قبیل «نام کاربری»، «رمز»، «encoding»، ...) به آن نیاز است را فراهم می‌کند.

```
Properties props = new Properties();
props.put("username", "myusername");
props.put("password", "easy");
props.put("encoding", "UTF8");
Connection cn =
    DriverManager.getConnection("jdbc:postgresql://localhost:5432/db", props);
```

در صورت بروز خطا در تولید `Connection`، (مثلاً وقتی سرویس دیتابیس در دسترس نباشد، دیتابیس `db` وجود نداشته باشد، یا نام کاربری و رمز برای ارتباط نادرست باشد) تمام متدهای فوق `SQLException` تولید می‌کنند.

تعريف دستور SQL

مهمنترین مرحله در تعامل با پایگاه داده اجرای دستورات SQL است. در ابتدای این فصل با چندین دستور SQL آشنا شدید، در اینجا باید دو خبر خوب به شما بدهم، اول اینکه SQL ای که ما در برنامه‌های جاوا استفاده می‌کنیم ساده‌ترین فرم SQL است که به عنوان SQL-92 شناخته می‌شود هرچند می‌توان از نمونه‌های پیچیده‌تر SQL که ممکن است فقط روی پایگاه داده خاص اجرا می‌شوند نیز در یک برنامه جاوا استفاده کرد اما ترجیح می‌دهیم برنامه خود را به گونه‌ای توسعه دهیم که سازگار با پایگاه داده‌های مختلف باقی بماند تا در آینده همیشه گزینه تغییر پایگاه داده در برنامه امکان‌پذیر باشد. خبر خوب دوم هم این است که در برنامه‌های امروزی به جای استفاده مستقیم از JDBC که برنامه نویسان را درگیر نوشتن دستورات SQL می‌کند از ابزارها و فریم ورکهای سطح بالاتر (از قبیل Hibernate) استفاده می‌شود که تولید دستورات SQL را انجام می‌دهند و برنامه نویسان را از درگیر شدن با SQL رهایی می‌دهند. با این وجود هر برنامه نویس جاوا باید اصول و دستورات اصلی SQL و مفاهیم JDBC را بداند زیرا اولاً هنوز در راه حل‌های خاص از JDBC استفاده می‌شود و ثانیاً حتی اگر در آینده به صورت مستقیم درگیر JDBC نشود اما حداقل بتواند ارتباط با پایگاه داده را دیباگ کند.

یک دستور SQL یک رشته `String` است که به صورت زیر در جاوا قابل تعریف است.

```
String sql= "INSERT INTO city (id, name) values (1, 'تهران')";
```

ارتباط با پایگاه داده

نکته حائز اهمیت در دستور فوق این است که مقدار فیلد `name` که از جنس `VARCHAR` است داخل «تک کوتیشین»^۱ نوشته شده است. این نکته را همیشه به خاطر بسیارید که در تمام دستورات SQL، مقدار فیلدهایی که از جنس رشته (`String`) هستند باید داخل «تک کوتیشین» قرار گیرند، در مثال فوق تهران داخل «تک کوتیشین» قرار گرفته است.

آبجکت Statement

اجرای جملات SQL با استفاده از یکی از آبجکتهای `Statement` یا `PreparedStatement` انجام می‌شود. از آنجاییکه `Statement` و `PreparedStatement` هردو اینترفیس هستند، نمی‌توان از روی آنها آبجکت ایجاد کرد و باید از کلاس‌هایی که این اینترفیسها را پیاده سازی کرده اند آبجکت بسازیم. اما کلاس‌هایی که این اینترفیسها را پیاده سازی کرده اند، بخشی از درایور هستند و ترجیح ما بر این است که کدهای برنامه را همچنان از درایور و پایگاه داده مستقل نگاه داریم. با این کار، بعداً می‌توانیم بدون تغییر در کدهای برنامه، پایگاه داده برنامه را تعویض کنیم.

خوب‌خوانی آبجکت `Connection` که در قسمت قبل ایجاد شد متدهای `createStatement()` و `prepareStatement()` را به ترتیب برای ایجاد آبجکتهای `Statement` و `PreparedStatement` فراهم می‌کند.

```
Statement state=connection.createStatement();
PreparedStatement ps = connection.prepareStatement(sql);
```

توجه کنید که متدهای `prepareStatement()` که برای ایجاد آبجکت `PreparedStatement` فراخوانی شده، دستور SQL را به عنوان پارامتر دریافت می‌کند. همچنان که در قسمتهای بعدی این فصل خواهید دید `Statement` و `PreparedStatement` کارکرد مشابهی دارند اما به دلیل مزایای که در `PreparedStatement` وجود دارد، استفاده از `Statement` ترجیح داده می‌شود.

اجرای دستور SQL

آبجکت `Statement` یا `PreparedStatement` که قبلاً ایجاد شد دستور SQL را اجرا و نتیجه را برمی‌گرداند. در اینترفیس `Statement` متدهای مختلفی تعریف شده است که بسته به نوع دستور SQL باید یکی از آنها فراخوانی شود. مثلاً برای اجرای یک دستور SQL که باعث تغییر در پایگاه داده می‌شود، متدهای `executeUpdate()` فراخوانی می‌شود.

```
int result = statement.executeUpdate(sql);
```

^۱ single quote

همچنین اگر از آبجکت PreparedStatement برای اجرای SQL استفاده کنیم، متد executeUpdate() را به صورت زیر فراخوانی می کنیم.

```
int result = preparedStatement.executeUpdate();
```

در فراخوانی اخیر، پارامتر sql وجود ندارد زیرا اساساً آبجکت PreparedStatement براساس یک رشته sql ایجاد می شود و بنابراین رشته sql در آبجکت PreparedStatement مستتر است. از هریک از آبجکتهای Statement یا PreparedStatement که استفاده شود، مقدار برگشتی متد executeUpdate() یک عدد است که تعداد موجودیتها یا رکوردهایی که درنتیجه اجرای SQL تحت تاثیر قرار گرفته اند را مشخص می کند. با دستور SQL که در قسمت قبل آنرا تعریف کردیم را اجرا کنیم، انتظار داریم یک رکورد به جدول city اضافه شود و مقدار 1 برگردانده شود.

SQL پارامتردار

یکی از سختیهایی که در ایجاد جملات SQL وجود دارد تولید کردن پویای آنهاست. به عنوان مثال تصور کنید که بخواهیم نام شهر را از کاربر دریافت کنیم و از طریق جمله SQL زیر در پایگاه داده ذخیره کنیم.

```
String cityName = ...//imported from user interface
Integer cityId = ...//generated by the application
String sql = "INSERT INTO city (id, name) VALUES ("+
    cityId+", '"+cityName+"')";
```

واضح است که چسباندن مقادیر داده ها در لابلای یک جمله SQL بسیار دشوار و خطازاست و حتی در مواقعي غیرممکن است. از همه اینها گذشته، ساختن جمله SQL به شیوه فوق برنامه را در معرض آسیب پذیری SQL Injection قرار می دهد. کاربری که با برنامه کار می کند می تواند نام جداول پایگاه داده را حدس بزند و به جای اسم یک شهر که قرار است در جدول city ذخیره شود یک جمله SQL مثلاً جمله زیر را وارد کند.

```
delete from city;
```

حال بر اساس کد فوق، این رشته جای نام شهر داخل sql قرار می گیرد و با این کار تمام داده های جدول city را حذف می کند. برای حل این مسئله، از SQL های پارامتردار استفاده می شود که به جای مقادیر پویا علامت ? گذاشته می شود.

```
String sql = "INSERT INTO city (id, name) VALUES (?, ?) ";
```

ارتباط با پایگاه داده

برای اجرای SQL پارامتردار باید ابتدا مقادیر پارامترهای آن که با ? مشخص شده اند تعیین شود. این کار با بکارگیری آجکت PreparedStatement به جای Statement امکانپذیر است. کد زیر نحوه انجام اینکار را نشان می دهد.

```
String cityName = ...//imported from user interface
Integer cityId = ...//generated by the application
String sql = "INSERT INTO city (id, name) VALUES (?, ?) ";
PreparedStatement ps = cn.prepareStatement(sql);
ps.setInt(1, cityId);
ps.setString(2, cityName);
int result = ps.executeUpdate();
System.out.println(result+" rows effected");
```

دقت کنید که مقادیر پارامترها چگونه با فراخوانی متدهای setString() و setInt() مقداردهی شده اند. ps.setInt(1, cityId) مشخص می کند که مقدار اولین پارامتر که از جنس int است باید با مقدار cityId مقداردهی شود. ps.setString(2, cityName) مقدار دومین پارامتر که String است را مشخص می کند.

اینترفیس PreparedStatement علاوه بر متدهای setString() و setInt() متدهای setBytes(), setDate(), setLong(), ...، که برای پارامترهایی از نوع Long دیگری از قبیل byte[], Date، ... مناسب است. دقت کنید که برخلاف Statement در فراخوانی متدهای executeUpdate() ارسال نشده است. SQL مقدار executeUpdate() به متدهای executeQuery() و executeUpdate() کنده باید به جای () فراخوانی شود.

جستجو در پایگاه داده با دستور select

اگر دستور SELECT یک دستور SQL باشد که صرفا رکوردهایی را از پایگاه داده انتخاب و بارگذاری می کند باید به جای () فراخوانی شود.

```
String selectQuery="select * from city";
ResultSet result=statement.executeQuery(selectQuery);
```

با اجرای دستور SQL فوق، مقادیر تمام رکوردهای جدول city بارگذاری می شوند. مقدار برگشتی متدهای executeQuery() که آجکتی از اینترفیس ResultSet است، امکان دسترسی به رکوردهای انتخاب شده را می دهد.

اینترفیس ResultSet دارای یک نشانگر (cursor) است که در هر لحظه به یکی از رکوردهای انتخاب شده اشاره می کند. با فراخوانی متدهایی که اینترفیس ResultSet فراهم می کند و لیست آن در زیر نشان داده شده است می توان به جزئیات رکوردهای آن اشاره می شود دسترسی بیندازید.

```
byte getByte(String columnLabel)
```

```
short getShort(String columnLabel)
int getInt(String columnLabel)
long getLong(String columnLabel)
float getFloat(String columnLabel)
double getDouble(String columnLabel)
java.util.Date getDate(String columnLabel)
String getString(String columnLabel)
```

اگر فرض کیم که اشاره گر ResultSet به یک رکورد اشاره کند، بسته به اینکه مقدار کدامیک از ستونهای آن رکورد را بخواهیم می توانیم متناسب با جنس آن ستون (String، Date، long,...) یکی از متدهای فوق را فراخوانی کنیم.

هریک از متدهای فوق، نام ستون (columnLabel) را به عنوان پارامتر دریافت می کنند و مقدار آن ستون را برمی گردانند.

اما وقتی مقادیر یک رکورد خوانده شد، چگونه نشانه گر را به رکورد بعد منتقل کنیم؟ برای این منظور متدهای next() از اینترفیس ResultSet را فراخوانی می کنیم.

```
boolean next();
```

با فراخوانی متدهای next()، نشانگر به رکورد بعدی (در صورت وجود) اشاره می کند و مقدار برمی گرداند و در صورتیکه رکورد بعدی وجود نداشته باشد در وضعیت فعلی باقی می ماند و false برمی گرداند. با توضیحات فوق باید متوجه شده باشید که نحوه دسترسی به رکوردهای انتخاب شده چگونه است. کد زیر انجام اینکار را نشان می دهد.

```
ResultSet rs = st.executeQuery(selectQuery);
while (rs.next ()) {
    Integer id = result.getInt("id");
    String name = result.getString ("name");
    System.out.println("ID:"+id+", Name:"+name);
}
```

با فرض اینکه داده های جدول city به صورت زیر باشد.

	id	name
Row 0	1231	تهران
Row 1	3157	اصفهان
Row 2	4587	شیراز
Row 3	5312	کرمان

شکل ۱۱-۵. یک جدول نمونه در پایگاه داده

ارتباط با پایگاه داده

خروجی اجرای کد فوق به شکل زیر خواهد بود.

```
ID: 1231, name: تهران  
ID: 3157, name: اصفهان  
ID: 4587, name: شیراز  
ID: 5312, name: کرمان
```

در برخی درایورها، امکان حرکت معکوس نشانگر نیز وجود دارد. در این درایورها، با فراخوانی متدهای previous() از آبجکت ResultSet می‌توان نشانگر را به رکورد قبلی منتقل کرد.

```
boolean previous();
```

کارکرد متدهای previous() و next() مشابه است. با این تفاوت که با فراخوانی آن نشانگر به رکورد قبلی (در صورت وجود) اشاره می‌کند و true برای گرداندن و اگر رکورد قبلی وجود نداشته باشد در وضعیت فعلی می‌ماند و false برای گرداندن. برای اینکه از این نوع ResultSet در برنامه خود استفاده کنیم مشروط بر اینکه درایور پایگاه داده این نوع ResultSet را پشتیبانی کند) ایجاد آبجکت Statement می‌باشد. با دو پارامتر اضافه از آبجکت Connection فراخوانی شود.

```
public Statement createStatement(int rsType, int rsConcurrency)  
    throws SQLException;
```

ایجاد شود.
در اینجا پارامتر اول یکی از مقادیر

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE
- TYPE_SCROLL_SENSITIVE

و پارامتر دوم یکی از مقادیر

- CONCUR_READ_ONLY
- CONCUR_UPDATABLE

که همگی به عنوان مقدار ثابت در اینترفیس ResultSet تعریف شده اند. rsType، یعنی اولین پارامتر از متدهای createStatement() در فعل شدن previous() تاثیر دارد. مقدار TYPE_FORWARD_ONLY به معنی غیرفعال بودن previous() است و نشانگر فقط به جلو حرکت می‌کند. برای اینکه نشانگر به جلو و عقب حرکت کند، باید یکی از مقادیر TYPE_SCROLL_SENSITIVE یا TYPE_SCROLL_INSENSITIVE استفاده شوند. تفاوت آنها در

این است که اگر رکوردی که در آبجکت `ResultSet` است در حین حرکت نشانگر در پایگاه داده تغییر کند، مقدار `TYPE_SCROLL_SENSITIVE` مقدار تغییر یافته را نشان می دهد اما `TYPE_SCROLL_INSENSITIVE` همچنین اگر `CONCUR_READ_ONLY` را به عنوان مقدار پارامتر دوم بکار ببرید تغییر در داده های `ResultSet` روی داده های پایگاه داده تاثیری نمی گذارد اما با مقدار `CONCUR_UPDATABLE` تغییرات روی پایگاه داده منعکس می شوند.

مدیریت تراکنش

همانطور که در ابتدای این فصل گفته شد، یکی از خصوصیات تمام پایگاه داده ها «مدیریت و کنترل تراکنش» است. تراکنش به دو موضوع بسیار مهم نظر دارد. اول، وقتیکه مجموعه ای از عملیات به صورت «یک واحد غیرقابل تجزیه» می باشد همگی با موفقیت انجام شوند یا با شکست یا عدم انجام یکی، بقیه نیز به وضعیت قبل برگردانده شوند. به عنوان نمونه، وقتی که پولی از یک حساب بانکی به حساب دیگر منقل می شود، انتظار داریم که کاهش از حساب اول و افزایش موجودی حساب دوم، هر دو با موفقیت انجام شوند و درصورتیکه یکی از عملیات به هر دلیل با شکست مواجه شود دومی نیز به وضعیت قبل برگردانده شود. دومین خاصیت مهم یک تراکنش، «کنترل همزمانی عملیات» است. به عنوان مثال اگر دو دستور SQL همزمان بخواهند مقدار یک رکورد جدولی را تغییر دهند نباید تداخلی در کار کرد یکدیگر داشته باشند. به عنوان مثال، یک کاربر را تصور کنید که از طریق دو اپلیکیشن مجزا بخواهد مبلغی را از حساب خود خرج کند، در اینصورت نباید تداخلی بین کار کرد دو اپلیکیشن و مقدار نهایی حساب کاربر بوجود آید. البته تراکنشها دو خصوصیت مهم دیگر هم دارند که برنامه نویسان مربوط نمی شود و باید توسط تولید کنندگان پایگاه داده لحاظ شود.

اما در یک برنامه جاوای تراکنش چگونه باید کنترل شود؟ اولین نکته ای که در اینجا باید به آن اشاره کنیم این است که اساساً تراکنشها توسط پایگاه داده ها کنترل می شود، یک برنامه جاوا با استفاده از JDBC API درایور پایگاه داده می تواند درخواست خود در مورد تراکنش را به پایگاه داده منتقل کند. کنترل و مدیریت تراکنش توسط آبجکت `java.sql.Connection` انجام می شود، این آبجکت دارای متد `() setAutoCommit` به صورت زیر است.

```
public void setAutoCommit(boolean autoCommit) throws SQLException
```

اگر این متد با پارامتر `true` فراخوانی شود، معنی آن این است که هر اجرای `Statement` در قالب یک تراکنش جداگانه اجرا می شود و نمی توان دو یا چند اجرای متوالی `Statement` را در قالب یک تراکنش قرار داد. واضح است که این وضعیت به منزله حذف تراکنش از منطق برنامه است. دقت کنید که از منظر

ارتباط با پایگاه داده

پایگاه داده، حذف تراکنش ممکن نیست، بلکه تحت شرایط فوق هیچ کدی در برنامه برای کنترل تراکنش وجود نخواهد داشت و هر اجرای Statement به عنوان یک تراکنش در نظر گرفته خواهد شد. اما با مشخص کردن مقدار false برای پارامتر () setAutoCommit لازم است در انتهای چندین Connection.commit() اجرای Statement که میخواهیم در قالب یک تراکنش لحاظ شوند، از ; () به منظور مشخص کردن انتهای تراکنش استفاده کنیم.

Entity کلاس

جدول city که در قسمت قبل آنرا معرفی کردیم و اغلب جداولی که در برنامه های واقعی وجود دارند حاوی داده های حیاتی و مهمی هستند که توسط کاربران در برنامه وارد می شوند و توسط آنها ویرایش یا حذف می شوند. به عبارت بهتر رکوردهای این جداول، آبجکتها بی هستند که توسط کاربران سیستم ایجاد، ویرایش، و یا حذف می شوند. بنابراین وقتی از یک رکورد یک جدول صحبت می کنیم در واقع از یک آبجکت برنامه صحبت کرده ایم که در پایگاه داده ذخیره شده است. واضح است که همه آبجکتها ب برنامه در پایگاه داده ذخیره نمی شوند، بلکه فقط آبجکتها با اهمیت که حاوی داده هستند ذخیره و بازیابی می شوند. به این آبجکتها، آبجکتها گفته می شود.

نکته بسیار جالب این است که همانطور که یک آبجکت متناظر با یک رکورد از یک جدول است و بالعکس. هر جدول نیز متناظر با یک کلاس است. بنابراین اگر در برنامه ای جدولی با نام city وجود دارد انتظار داریم که کلاسی با نام City (یا هر نام دیگری) نیز در برنامه وجود داشته باشد که متناظر با جدول city باشد. وقتی آبجکتی از روی کلاس City ساخته می شود می تواند به عنوان رکوردي در جدول city ذخیره شود. به کلاس City «کلاس City» و به آبجکتی که از روی City ساخته می شود «آبجکت City» گفته می شود.

با این توضیح و برای تکمیل مثال قبل، کلاس City را به صورت زیر در برنامه پیاده سازی می کنیم.

```
1 package ir.atlassoft.javase.chapter18;
2
3 public class City {
4     int id;
5     String name;
6
7     public City() {
8     }
9
10    public City(int id, String name) {
11        this.id = id;
12        this.name = name;
```

```

13     }
14
15     public int getId() {
16         return id;
17     }
18
19     public void setId(int id) {
20         this.id = id;
21     }
22
23     public String getName() {
24         return name;
25     }
26
27     public void setName(String name) {
28         this.name = name;
29     }
30 }
```

کد ۱-۱۰. کلاس City

در کلاس City که در فوق پیاده سازی کرده ایم فیلدهای id و name را تعریف کرده ایم که متناظر با آنها ستونهای id و name در جدول city وجود دارند. در کلاس City فوق، دو سازنده و متدهای getter و setter تعریف شده اند و هیچ کار کرد خاص دیگری در این کلاس وجود ندارد.

توجه: در هر برنامه ای که با پایگاه داده کار می کند عملیات ذخیره و بازیابی آبجکتهاي Entity که هر کدام توسط یک دستور SQL مجزا انجام می شود در قالب متدهای مختلف پیاده سازی می شوند. اگر تعداد کلاسهای Entity در یک برنامه زیاد باشد در اینصورت متدهایی که برای ذخیره و بازیابی آبجکتهاي Entity پیاده سازی می شوند نیز زیاد خواهند بود به همین دلیل الگویی در طراحی کلاسها ارایه شده است که به آن الگوی DAO گفته می شود و براساس آن «تمام متدهایی که مربوط به ذخیره و بازیابی یک کلاس Entity خاص است باید در قالب یک کلاس جداگانه با نام کلاس DAO پیاده سازی شود». به عنوان نمونه، تمام عملیات ذخیره و بازیابی آبجکتهاي City باید در یک کلاس (مثلاً با نام CityDAO) پیاده سازی شود. با چنین نظمی، طراحی و پیاده سازی متدهای ذخیره و بازیابی آبجکتها سروسامان می یابد و مهمتر اینکه نگهداری برنامه بسیار آسان خواهد شد. کلاس CityDAO در زیر نشان داده شده است.

```

1 package ir.atlassoft.javase.chapter18;
2
```

ارتباط با پایگاه داده

```
3 import java.sql.*;
4 import java.util.ArrayList;
5
6 public class CityDAO {
7
8     private Connection cn;
9
10    static {
11        try {
12            Class.forName(
13                "org.postgresql.Driver");
14        } catch (ClassNotFoundException e) {
15            e.printStackTrace();
16        }
17    }
18
19    //opens database connection
20    public CityDAO() throws Exception {
21
22        cn = DriverManager.getConnection(
23            "jdbc:postgresql://localhost:5432/db");
24        cn.setAutoCommit(false);
25    }
26
27    //insert a new City to database
28    public void save(City city)
29        throws Exception {
30        String query = "INSERT INTO city " +
31            "(id, name) VALUES (?, ?)";
32        PreparedStatement ps = cn.prepareStatement(query);
33        ps.setInt(1, city.getId());
34        ps.setString(2, city.getName());
35        ps.executeUpdate();
36        close();
37    }
38
39    //remove an existing city from database
40    public void remove(City city)
41        throws Exception {
42        String query = "DELETE FROM city WHERE id=?";
```

```

43
44     PreparedStatement ps = cn.prepareStatement(query);
45     ps.setInt(1, city.getId());
46     ps.executeUpdate();
47     ps.close();
48     close();
49 }
50
51 //update an existing city with
52 //new fields values in database
53 public void update(City city)
54     throws Exception {
55     String query = "UPDATE city SET name=? WHERE ID=?";
56     PreparedStatement ps = cn.prepareStatement(query);
57     ps.setString(1, city.getName());
58     ps.setInt(2, city.getId());
59
60     ps.executeUpdate(query);
61     ps.close();
62     close();
63 }
64
65 //retrive all citys from database
66 public City[] findAll()
67     throws Exception {
68
69     ArrayList retList = new ArrayList();
70     String query = "SELECT * FROM city";
71     PreparedStatement ps = cn.prepareStatement(query);
72     ResultSet result = ps.executeQuery();
73     while (result.next()) {
74         int id = result.getInt("id");
75         String name = result.getString("name");
76         City city =
77             new City(id, name);
78         retList.add(city);
79     }
80     ps.close();
81     return (City[]) retList.toArray(new City[0]);
82 }
```

ارتباط با پایگاه داده

```
83  
84     private void close() throws SQLException {  
85         cn.commit();  
86     }  
87  
88 }
```

کد ۱۸-۱. کلاس CityDAO

کد زیر نحوه ذخیره کردن یک آبجکت City را با استفاده از کلاس CityDAO نشان می دهد.

```
1 package ir.atlassoft.javase.chapter18;  
2  
3 public class SavingCityDemo {  
4  
5     public static void main(String[] args) throws Exception {  
6         City e = new City(1, "Tehran");  
7         CityDAO cityDAO = new CityDAO();  
8         cityDAO.save(e);  
9     }  
10 }
```

کد ۱۸-۲. ذخیره آبجکت City

خلاصه

در صورتیکه نتوان داده های یک برنامه را در یک مکان دائمی ذخیره نمود، عملاً آن برنامه کاربرد چندانی ندارد به این منظور برنامه هایی نوشته شده اند که به آنها Database یا «پایگاه داده» گفته می شود این برنامه ها می توانند داده های هر برنامه ای را به صورت دائمی درون سیستم فایل کامپیوتر ذخیره کنند. برای استفاده از یک پایگاه داده نیاز به یک برنامه واسطه به نام راه انداز و یک زبان قابل فهم برای برنامه و پایگاه داده به نام SQL است راه انداز معمولاً توسط شرکت تولید کننده پایگاه داده نوشته می شود و برای هر مصرف کننده ای قابل استفاده خواهد بود. SQL نیز زبان استانداری است که از طریق آن می توان مفاهیم ذخیره سازی یا بازیابی داده ها را به تمام انواع پایگاه های داده فهماند! در جاوا مجموعه‌ای از کلاسها و اینترفیسها برای ارتباط، ذخیره و بازیابی داده ها در پایگاه داده ها طراحی شده اند که به آن اختصاراً JDBC گفته می شود مهمترین اینترفیس‌های این مجموعه، Statement، PreparedStatement و Connection هستند که عملیات مرتبط با پایگاه داده را انجام می دهند.

آبجکتی است که ارتباط با پایگاه داده را بر عهده دارد، Connection Statement و PreparedStatement اجرای جملات SQL را انجام می دهند استفاده از Statement مشابه است اما از این نظر که می توان مقادیر داخل SQL را با استفاده از ? مشخص نمود و سپس از طریق ایندکس آنها، هریک را مقداردهی کرد نسبت به PreparedStatement ارجحیت دارد.

تمرینات

- (۱) برنامه‌ای بنویسید که از طریق کنسول برنامه، اطلاعات هر کتاب را دریافت کند، سپس این اطلاعات را در پایگاه داده ذخیره کند. اطلاعات هر کتاب می‌تواند شامل نام، نام نویسنده، شابک، موضوع و ناشر باشد.
- (۲) برنامه‌ای بنویسید که اطلاعات کتابهایی را که در مثال قبل وارد پایگاه داده کرده‌اید بخواند، و آنها را در کنسول برنامه نمایش دهد.
- (۳) برنامه‌ای بنویسید که نام یک کتاب را از طریق کنسول برنامه دریافت کند، سپس اطلاعات کامل آن کتاب را از پایگاه داده خوانده و به کاربر نشان دهد، درصورتیکه کتابی با نام مورد نظر یافت نشود پیغام مناسبی به کاربر ارایه کند.
- (۴) برنامه‌ای بنویسید که با استفاده از کلاس `Property`، تمام `Property`‌های سیستم را بخواند و آنها را در جدولی در پایگاه داده ذخیره کند.
- (۵) فرض کنید که قرار است برنامه‌ای برای یک رستوران بنویسید، یکی از کلاس‌هایی که در این برنامه وجود خواهد داشت کلاس `Food` خواهد بود، مسلماً در پایگاه داده نیز جدولی به نام `FOOD` خواهید داشت که اطلاعات هر غذا در آن ذخیره می‌شود کلاسی بنویسید که دارای متدهای `Food getFood(int id)`
`void saveFood(Food food)`
`Food[] getAllFoods()`
`void updateFood(Food food)` برای ذخیره و بازیابی هریک از آبجکت‌های `Food` باشد.
- (۶) فرض کنید که می‌خواهیم برنامه‌ای بنویسیم که همواره یک عدد غیرتکراری تولید می‌کند برای این منظور جدولی در پایگاه داده طراحی می‌کنیم که یک ستون `UNIQUE_ID` دارد این جدول یک سطر دارد که در آن یک عدد ذخیره شده است هرگاه نیاز به یک عدد غیر تکراری داشته باشیم می‌توان با مراجعه به جدول، عدد ذخیره شده را بازیابی کرده و در اختیار برنامه قرار داد علاوه بر این، یک واحد به آن اضافه نموده و مجدداً آنرا ذخیره کرد تا در مراجعات بعدی عدد متفاوتی دریافت شود این برنامه را پیاده سازی کنید.

۷) یک پایگاه داده ایجاد کنید و در آن جدولهای مختلف با نام های دلخواه همراه با ستون های دلخواه ایجاد کنید. سپس برنامه ای بنویسید که نام یک جدول را از طریق کنسول برنامه دریافت می کند و تمام اطلاعات آن جدول (نام ستونها و اطلاعات ذخیره شده در آن جدول) را در کنسول برنامه نمایش می دهد.

۸) برنامه ای بنویسید که نام دو جدول را از کاربر دریافت کند، سپس اطلاعات ذخیره شده در یک جدول را در جدول دیگر ذخیره کند. (فرض براین است که دو جدول ستونهای یکسان دارند)

۹) فرض کنید که در یک پایگاه داده جدول PERSON وجود دارد که اطلاعات افراد مختلف را نگهداری می کند ضمن ایجاد پایگاه داده و جدول PERSON، برنامه ای بنویسید که به صورت تصادفی یکی از افراد داخل جدول را انتخاب کند و آنرا نمایش دهد.

۱۰) فرض کنید در پایگاه داده ای که متعلق به یک بانک است، جدولی به نام ACCOUNT وجود دارد که اطلاعات حسابهای مشتریان را نگهداری می کند اولاً کلاسی بنویسید که از طریق متدهای آن بتوان اطلاعات حساب مشتریان را ذخیره و بازیابی کرد متدهای این کلاس به صورت زیر خواهند بود:

```
Account getAccount(int id)
void saveAccount(Account account)
Account[] getAllAccounts()
void updateAccount(Account account)
```

حال جدول دیگری در این پایگاه داده ذخیره کنید که تاریخچه هر حساب در آن نگهداری شود، اطلاعات این جدول می تواند شامل میزان کاهش یا افزایش موجودی و تاریخ آن باشد. حال متدهای updateAccount و saveAccount را به گونه ای تغییر دهید که در زمان ذخیره یا بروزرسانی اطلاعات حساب، اطلاعات در این جدول نیز ثبت شود.

۱۱) برنامه ای بنویسید که با هر بار اجرا شدن زمان اجرا و زمان توقف خود را در پایگاه داده ثبت کند.

۱۹

عبارت‌های باقاعدۀ

یک «عبارت با قاعده»، یک رشته است که مجموعه ای از رشته‌های دیگر را توصیف می‌کند. رشته‌هایی که توسط یک «عبارت با قاعده» توصیف می‌شوند همگی از یک قالب و الگو پیروی می‌کنند. برای درک این مطلب به رشته‌های زیر توجه کنید.

alireza@mysite.com
mehdi@domain.ir
iraj@server.net

رشته‌های فوق همگی آدرس ایمیل هستند، این رشته‌ها اگرچه بایکدیگر متفاوتند اما از الگوی یکسانی پیروی می‌کنند مثلاً همه آنها با یک نام شروع می‌شوند که از کاراکترهای حروف الفبای انگلیسی تشکیل شده‌اند، بعد از آن کاراکتر @ می‌آید سپس یک نام دیگر، یک نقطه، و در انتها یک پسوند .com، .ir یا .net قرار می‌گیرد. برای ایمیل‌های فوق می‌توان عبارت باقاعدۀ

`^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,6}$`

را مشخص نمود که قالب و الگوی آدرسهای ایمیل را مشخص می‌کند.

نوشتن عبارتهای باقاعده خود قواعدی دارد که در این فصل با آنها آشنا خواهید شد. وقتی یک عبارت باقاعده را نوشtid می توانید از زبان جاوا برای پردازش آن استفاده کنید. مثلاً می توانید با استفاده از زبان جاوا انطباق مجموعه ای از رشته ها را با یک عبارت باقاعده بررسی کنید، می توانید تمام رشته های منطبق بر یک عبارت باقاعده را از یک متن استخراج کنید یا ویرایش یا دستکاری کنید.

کلاسها و اینترفیسها مربوط به کار با عبارتهای باقاعده در پکیج `java.util.regex` قرار دارد.

توجه: «عبارت‌های باقاعدۀ» ترجمۀ عبارت Regular Expressions است. در کتب زبان انگلیسی به عبارتهای باقاعدۀ به صورت مختصر RegEx می‌گویند. در این فصل نیز ما از همین کلمۀ RegEx برای نامیدن عبارتهای باقاعدۀ استفاده می‌کینم.

در این فصل ابتدا نوشتun عبارتهای باقاعده را خواهید آموخت و سپس چگونگی استفاده از کلاسها و اینترفیسها جاوا را برای پردازش عبارتهای باقاعدۀ فرا خواهید گرفت.

ساده ترین عبارت باقاعدۀ

ساده ترین شکل یک عبارت باقاعدۀ یک رشته است که دقیقاً رشته هایی معادل خودش را توصیف می‌کند.

عبارت باقاعدۀ `cat` را تصور کنید در اینصورت:

رشته "cat" منطبق بر این عبارت خواهد بود.

رشته "my cat is white" در یک نقطه و آنهم از کاراکتر ۳ تا ۶ منطبق بر عبارت باقاعدۀ فوق است.

رشته "my cat and your cat" در دو نقطه منطبق بر عبارت باقاعدۀ فوق است.

رشته "cats" در یک نقطه، از کاراکتر ۰ تا ۴ منطبق بر عبارت باقاعدۀ فوق است.

کاراکترهای خاص

برخی کاراکترها هستند که اگر از آنها در یک عبارت باقاعدۀ استفاده کنید با معنی خاصی تفسیر می‌شوند.

کاراکترنقطه (.) یکی از این کاراکترهای خاص است و اگر در یک عبارت باقاعدۀ استفاده کنید به معنی «هر کاراکتری» تفسیر می‌شود. مثلاً عبارت `cat.` یعنی هر رشته ای که با `cat` شروع می‌شود و در انتهای هر کاراکتری بیاید. با این تعریف، رشته های `cat`, `cati`, `cats`, و `cat.` منطبق بر عبارت `cat.` هستند.

کاراکترهای `.+*?^$[\]{}` همگی جزو کاراکترهای خاص محسوب می‌شوند.

توجه: به لیست فوق دقت کنید، کاراکترهای `!`, `@`, `#` در لیست کاراکترهای خاص قرار ندارند.

اگر بخواهید کاراکترهای فوق را در شکل و قالب خودشان استفاده کنید (نه با معنی و مفهوم خاص شان) می‌توانید آنها را به یکی از دو شکل زیر استفاده کنید:

- ۱) قبل از آنها از کاراکتر \ استفاده کنید
- ۲) قبل از آن \Q و بعد از آن \E استفاده کنید.

در ادامه این فصل با کاراکترهای خاص آشنا خواهید شد.

محدوده کاراکترها

عبارت‌های باقاعدۀ بسیار انعطاف پذیر هستند با استفاده از آنها می‌توانید رشته‌هایی را مشخص کنید که با یکی از چند کاراکتر مورد نظرتان شروع می‌شوند یا به یکی از چند کاراکتر موردنظرتان خاتمه می‌یابند. ممکن است بخواهید رشته‌هایی را مشخص کنید که با یکی از چند کاراکتر مورد نظرتان شروع نشوند. این کار از طریق گروه بندی کاراکترها امکان‌پذیر است. مثلاً اگر بخواهید رشته‌هایی را مشخص کنید که حرف اول آنها یکی از کاراکترهای a، b یا c باشد ولی حرف دوم و سوم آنها at باشد عبارت باقاعدۀ [abc]at را می‌توانید تعریف کنید. کاراکترهای [] که از جمله کاراکترهای خاص به شمار می‌روند برای دسته بندی کردن کاراکترها استفاده می‌شوند فقط یکی از کاراکترهایی که داخل [] قرار می‌گیرند می‌توانند در رشته مورد نظر ظاهر شوند. با این توصیف، رشته‌های aat و bat و cat منطبق بر عبارت باقاعدۀ فوق هستند. جدول زیر خلاصه‌ای از گروه بندی کاراکترها را نشان می‌دهد

محدوده کاراکترها	
یکی از کاراکترهای a، b یا c	[abc]
هر کاراکتری به جز کاراکترهای a، b و c	[^abc]
از کاراکتر a تا z یا از کاراکتر Z تا a	[a-zA-Z]
از کاراکتر a تا d یا از کاراکتر m تا p (شکل دیگر این عبارت [a-dm-p] است)	[a-d[m-p]]
اشتراک کاراکترهای a تا z و کاراکترهای d، e و f (نتیجه اشتراک دو مجموعه، کاراکترهای d، e و f است)	[a-z&&[def]]
کاراکترهای a تا z به جز کاراکترهای b و c	[a-zA&&[^bc]]
کاراکترهای a تا z به جز کاراکترهای m تا p	[a-zA&&[^m-p]]

جدول ۱۹-۱۹. گروه بندی کاراکترها

توجه کنید که در انطباق یک رشته با یک عبارت با قاعده، فقط یک کاراکتر رشته با یک گروه منطبق می‌شود به عبارت ساده‌تر، [abc] مشخص کننده یک کاراکتر است که آن کاراکتر می‌توان a، b یا c باشد. به مثالهای زیر توجه کنید

منطبق نیستند	منطبق هستند	عبارت باقاعدۀ
Hat	rat .cat .bat	[bcr]at
rat و cat .bat	hat	[^bcr]at

D	a . b . c	[a-c]
cat6	cat4 .cat5 .cat1	cat[1-5]
cat4 .cat5 .cat1	cat7 .cat6	cat[^1-5]
5 .9	0 , 3.6 , 8	[0-4 [6-8]]
6 .2	5 .4 , 3	[0-9&&[345]]
3 .7	4 .5 , 6	[2-8&&[4-6]]
3 .4 , 5	2 .6 , 9	[0-9&&[^345]]

جدول ۱۹-۳. مثالیایی از عبارتهای باقاعدۀ

میانبر برخی محدوده های پراستفاده

برای آسان شدن تعریف عبارتهای باقاعدۀ، برای برخی گروه کاراکترهای پراستفاده، میانبرهایی طراحی شده است. مثلا در یک عبارت با قاعده می توان برای مشخص کردن یک کاراکتر عددی به جای [0-9] از \d استفاده نمود. جدول زیر لیست میانبرهای گروه های پراستفاده و گروه کاراکترهای معادل آن را نشان می دهد.

میانبر	گروه معادل آن
.	هر کاراکتری
\d	هر کاراکتری ([0-9])
\D	هر کاراکتر غیرعددی ([^0-9])
\s	فاصله (space) و کاراکترهای غیرقابل مشاهده ([\t\n\x0B\f\r])
\S	کاراکتری به جز کاراکتر «فاصله» و کاراکترهای غیرقابل مشاهده ([^\s])
\w	اعداد و کاراکترهای حروف الفبای انگلیسی ([a-zA-Z_0-9])
\W	کاراکترهای به جز اعداد و کاراکترهای حروف الفبا ([^\w])

جدول ۱۹-۴. میانبرهای پراستفاده

همانطور که ملاحظه می کنید تمام میانبرها با \ شروع می شوند. در ابتدای این فصل نیز در بخش "کاراکترهای خاص"، استفاده از \ برای منتفی کردن معنی کاراکترهای خاص را آموختید. نکته ای که وجود \ دارد این است که در زبان جاوا اگر بخواهید رشته ای حاوی \ را مشخص کنید باید از \\ استفاده کنید زیرا \ یک کاراکتر کنترلی در رشته ها نیز محسوب می شود. نتیجه اینکه اگر بخواهید یک عبارت باقاعدۀ برای مشخص کردن یک کاراکتر عددی را بیان کنید باید رشته ای به صورت زیر را مشخص کنید

\d

یا اگر بخواهید کاراکتر) را مشخص کنید باید عبارت باقاعدۀ)) را استفاده کنید.

عبارت‌های با قاعده

جدول زیر مثالهایی را از بکارگیری میانبرها و کاراکترهای کنترلی را نشان می‌دهد.

منطبق نمی‌شوند	منطبق می‌شوند	عبارت باقاعده
	@, e, 1	.
a, w, k, p	1, 5, 7	\d
1, 8, 3	a, h	\D
a, 3, t	space	\s
Space	R, 8, u	\S
@, #, *	A, r, p, e	\w
h, q, 9, x	!, #, &, ^	\W

جدول ۴-۱۹. مثالهایی از بکارگیری میانبرها

بد نیست در انتهای این بخش، یکبار دیگر میانبرهای محدوده‌های پراستفاده را مرور کنیم.

الف) میانبرهای با حروف کوچک:

\d هر کاراکتر عددی

\D هر کاراکتر فاصله (یا کاراکترهای غیرقابل مشاهده از قبیل کاراکتر خط جدید)

\s هر کاراکتر عددی و حروف الفبای انگلیسی

ب) میانبرهای با حروف بزرگ

\D هر کاراکتر غیرعددی

\S هر کاراکتر قابل مشاهده

\W هر کاراکتر غیرعددی و غیر حروف الفبای انگلیسی

تکرار

عبارت‌های باقاعده‌ای که تا این لحظه آموختید فقط بر یک کاراکتر محدودیت ایجاد می‌کردند، حتی وقتی در عبارت باقاعده از گروه کاراکترها یا میانبر گروهها استفاده کردیم باز هم فقط محدوده یک کاراکتر را مشخص نمودیم. اما در بسیاری موارد می‌خواهیم میزان تکرار یک کاراکتر را نیز مشخص کنیم (مثلاً می‌خواهیم یک عدد ده رقمی را مشخص کنیم (یعنی رشته‌ای که از ۱۰ کاراکتر عددی ساخته شده باشد و رقم سمت چپ آن صفر نباشد) یا می‌خواهیم اعداد بین ۱ تا ۱۰۰ را مشخص کنیم (یعنی رشته‌ای که از ۱ تا سه کاراکتر عددی تشکیل شده باشد). در هر دو مورد باید علاوه بر مشخص کردن کاراکتر عددی (\d) تعداد تکرار آن کاراکتر را نیز باید مشخص کنیم.

از نشانه های $?$ ، $*$ ، $+$ و $\{ \}$ برای مشخص کردن میزان تکرار یک کاراکتر یا مجموعه کاراکترها استفاده می شود. جدول زیر معنی این نشانه ها را توضیح می دهد

میزان تکرار	توضیح
$?$	صفر یا یک
$*$	صفر یا هر تعداد
$+$	حداقل یک و حداقل هر تعداد
$\{n\}$	دقیقا n بار
$\{n,\}$	حداقل n بار
$\{n,m\}$	حداقل n و حداقل m بار

جدول ۱۹-۵. نشانه های تکرار

رشته خالی یا رشته ای که فقط حاوی یک کاراکتر a باشد بر عبارت باقاعدۀ $a?$ منطبق می شود زیرا از نظر باقاعدۀ $?$ کاراکتر قبل از $?$ باید وجود نداشته باشد یا حداکثر یکبار وجود داشته باشد. رشته خالی و رشته های a ، aa ، aaa ... بر عبارت باقاعدۀ a^* منطبق می شوند زیرا باقاعدۀ * بیان می دارد که کاراکتر قبل از $*$ می تواند وجود نداشته باشد یا به تعداد دلخواه ظاهر شود به شکل مشابه، همچنین رشته های a^+ ، a^{+} ... بر عبارت a^+ منطبق می شوند اما رشته خالی بر آن منطبق نمی شود زیرا بر اساس قاعدۀ $+$ ، کاراکتر قبل از $+$ حداقل باید یکبار و حداکثر به تعداد نامحدود می تواند وجود داشته باشد. رشته $"aa"$ بر عبارت a^2 منطبق می شود اما رشته های $"a"$ و $"aa"$ بر آن منطبق نمی شوند. رشته $"aaa"$ ، $"aa^+$... بر عبارت $\{2\}$ منطبق می شوند اما رشته های $"a"$ و $"aa"$ بر آن منطبق نمی شوند. و نهایتا رشته های $"aa"$ و $"aaa"$ بر عبارت $\{2,3\}$ منطبق می شوند اما رشته های $"a"$ و $"aaa"$ بر آن منطبق نمی شوند.

حال که نشانه های تکرار را آموخته اید باید بدانید که این نشانه ها نیز در سه شکل مختلف قابل استفاده هستند به ساده ترین شکل این نشانه ها که تا این لحظه با آنها آشنا شده اید، نشانه های «پرخور» یا **Greedy** گفته می شود. برای درک دلیل این نامگذاری، عبارت Cat^* . را تصور کنید هر رشته ای که با یک یا بیش از یک کاراکتر شروع شود و در انتها به رشته Cat ختم شوند بر این عبارت منطبق می شوند. مثلا رشته های $1Cat$ ، $OneCat$ ، $ACat$... همگی بر این عبارت منطبق هستند اما همه این رشته ها فقط در یک نقطه بر عبارت Cat^* . منطبق می شوند. رشته $ACatWithAnotherCat$ را تصور کنید این رشته در چند نقطه بر عبارت Cat^* . منطبق می شود؟ واضح است که در این رشته دو کلمه Cat وجود دارند و قبل از هر کدام از آنها نیز حداقل یک کاراکتر قرار دارد پس انتظار ما این است که این رشته در دو نقطه بر عبارت فوق منطبق باشد. اما برخلاف انتظار، این رشته فقط در یک نقطه (از ابتدا تا انتهای رشته) بر عبارت فوق منطبق است. دومین Cat موجود در رشته بر Cat موجود در عبارت Cat^* . منطبق می شود.

علت این است که وقتی پردازندۀ عبارت باقاعدۀ می خواهد انطباق رشته فوق را بر روی عبارت **Cat***. تشخیص دهد کل رشته را به صورت یکجا دریافت می کند (اصطلاحا گفته می شود کل رشته را یکجا می خورد) و سپس انطباق تمام رشته را روی عبارت **Cat***. بررسی می کند اگر انطباق رخ دهد کار خاتمه می یابد اما اگر انطباق رخ ندهد، کاراکتر آخر حذف می شود و انطباق رشته بدست آمده بر روی عبارت بررسی می شود، این کار تا بدست آمدن یک انطباق ادامه می یابد و به محض حصول یک انطباق خاتمه می یابد اما اگر انطباقی رخ ندهد جستجو خاتمه می یابد بدون اینکه هیچ انطباقی رخ داده باشد. به این ترتیب وقتی پردازندۀ عبارت با قاعده، انطباق **ACatWithAnotherCat** را روی **Cat***. بررسی می کند ابتدا انطباق تمام رشته را روی عبارت **Cat***. بررسی می کند و از آنجاییکه تمام رشته به کلمۀ **Cat** ختم می شود منطبق می شود و خاتمه می یابد.

در مقابل نشانه های «پرخور»، نشانه های «کم اشتها» یا **Reluctant** قرار دارند. برخلاف نشانه های پرخور، در نشانه های کم اشتها پردازندۀ تمام رشته را یکجا نمی خورد بلکه کاراکتر به کاراکتر از ابتدای رشته شروع به خوردن رشته می کند تا نقاط انطباق رشته بر عبارت را پیدا کند در صورتیکه انطباقی رخ دهد، پردازش از کاراکتر بعد از انطباق کار خود را ادامه می دهد. شکل این نشانه ها مشابه نشانه های پرخور است با این تفاوت که از علامت ؟ بعد از نشانه استفاده می شود مثلا شکل کم اشتهای عبارت **Cat**?* به صورت **Cat**?*. است. اگر انطباق رشته **ACatWithAnotherCat** را بر **Cat**?*. بررسی کنیم در اینصورت به دو انطباق خواهیم رسید زیرا اگر از کاراکتر اول رشته جستجو کنیم، اولین انطباق در **ACat** خواهد بود (کاراکتر ۰ تا ۴) بعد از آن، انطباق دوم روی رشته **WithAnotherCat** (از کاراکتر ۴ تا ۱۸) رخ می دهد.

در مقابل دو نوع نشانه های پرخور و کم اشتها، نوع دیگری از نشانه های تکرار هم وجود دارند که به آنها نشانه های «انحصار طلب» یا **Possessive** گفته می شود این نشانه ها، مشابه نشانه های پرخور تمام رشته را یکجا می خورند اما برخلاف نشانه های پرخور، اگر انطباق پیدا نشوند رشته را به عقب برنمی گردند. شکل این نشانه ها مشابه نشانه های پرخور است با این تفاوت که از علامت + در انتهای این نشانه ها استفاده می شود. اگر انطباق رشته **ACatWithAnotherCat** را بر **Cat**+*+. بررسی کنیم با نهایت شگفتی هیچ انطباقی وجود نخواهد داشت علت این است که نشانه انحصار طلب تمام رشته را به صورت یکجا می خورد و سپس انطباق را بررسی می کند تمام رشته بر عبارت +*. منطبق می شود حال چیزی از رشته باقی نمانده تا بر **Cat** (که در انتهای عبارت **Cat**+*. وجود دارد) منطبق شود. به این ترتیب این رشته بر این عبارت انحصار طلب منطبق نمی شود. جدول زیر انواع نشانه های تکرار را شرح می دهد.

معنی	نشانه های تکرار		
	انحصار طلب	کم اشتها	پرخور
X صفر یا یک بار	X?+	X??	X?
X صفر یا هر تعداد	X*+	X*?	X*+
X یک یا بیشتر	X++	X+?	X++

X دقيقا n بار	X{ n } +	X{ n } ?	X{ n }
X حداقل n بار	X{ n, } +	X{ n, } ?	X{ n, }
X حداقل n بار و حداکثر m بار	X{ n, m } +	X{ n, m } ?	X{ n, m }

جدول ۱۹-۶. انواع نشانه های تکرار

انطباق رشته های خالی

دو عبارت باقاعدۀ a^* و $a?$ را تصور کنید. رشته خالی (رشته‌ای که حاوی هیچ کاراکتری نباشد) بر هر دو این عبارتها منطبق است زیرا هر دو اجازه وجود نداشتن a را در رشته می‌دهند. از آنجائیکه طول رشته خالی صفر است، انطباق برای رشته خالی از کاراکتر ۰ تا کاراکتر ۰ اتفاق می‌افتد. به این نوع انطباق که در حقیقت یک رشته خالی بر عبارت باقاعدۀ منطبق می‌شود «انطباق رشته خالی» گفته می‌شود.

انطباق رشته خالی در چندین حالت مختلف ممکن است اتفاق بیفتد.

- (۱) در رشته خالی (مثال فوق)
 - (۲) بعد از آخرین کاراکتر رشته (می‌توان فرض کرد هر رشته با ترکیب همان رشته و یک رشته خالی ساخته شده است، بنابراین رشته خالی انتهای هر رشته می‌تواند یک نقطه انطباق باشد)
 - (۳) یا بین دو کاراکتر رشته (می‌توان بین هر دو کاراکتر یک رشته، یک رشته خالی را تصور کرد که می‌تواند نقطه انطباق باشد)
- برای درک انطباق رشته خالی به مثالهای زیر توجه کنید:

انطباق	رشته	عبارت
در دو نقطه، کاراکتر ۰ تا ۱ (کاراکتر a) و کاراکتر ۱ تا ۱ (رشته خالی)	a	a?
در دو نقطه، کاراکتر ۰ تا ۱ (کاراکتر a) و کاراکتر ۱ تا ۱ (رشته خالی)	a	a*
در یک نقطه، کاراکتر ۰ تا ۱ (کاراکتر a)	a	a+

جدول ۱۹-۷. مثالیابی از انطباق رشته های خالی (۱)

در هر سه نشانه تکرار، کاراکتر a بر روی عبارت باقاعدۀ منطبق می‌شود اما در $a?$ و a^* رشته خالی که در انتهای رشته تصور می‌شود نیز بر عبارت باقاعدۀ منطبق است. مثالهای دیگری از انطباق رشته خالی را در جدول ۱۹-۸ مشاهده می‌کنید.

انطباق	رشته	عبارت
در شش نقطه، هر کاراکتر a یک نقطه انطباق، و رشته خالی در انتهای رشته یک انطباق	aaaaaa	a?
در دو نقطه، از ابتدا تا انتهای رشته (aaaaa) یک نقطه انطباق، و رشته خالی در انتهای رشته یک انطباق	aaaaaa	a*

aaaaa	در یک نقطه، تمام رشته (aaaaa) یک نقطه انباق	aaaaa	a+
-------	---	-------	----

جدول ۱۹-۱. مثالبایی از انباق رشته‌های خالی (۲)

توجه: همانطور که از بالا متوجه شده اید نشانه های a^* و a^+ به رشتة خالی انتهای رشتة نیز توجه می کنند بنابراین اگر رشتة خالی انتهای رشتة هم بر آنها منطبق شود آنرا به عنوان نقطه انباق معرفی می کنند اما رشتة خالی انتهای رشتة بر نشانه $+ \text{ منطبق نمی شود زیرا بر اساس این عبارت حداقل یک } a \text{ برای انباق باید وجود داشته باشد.}$

حال اجازه دهید انباق رشتة $ababaaaab$ را روی $a?$, a^* و a^+ بررسی کنیم. جدول ۱۹-۹ انباق این رشتة را روی عبارتهای باقاعده $a?$, a^* و a^+ نشان می دهد.

عبارت	رشته	انباق
$a?$	$ababaaaab$	<p>در ده نقطه،</p> <ul style="list-style-type: none"> • انطباق رشتة "a" از کاراکتر ۰ تا ۱ • انطباق رشتة خالی از کاراکتر ۱ تا ۱ • انطباق رشتة "a" از کاراکتر ۲ تا ۳ • انطباق رشتة خالی از کاراکتر ۳ تا ۴ • انطباق رشتة "a" از کاراکتر ۴ تا ۵ • انطباق رشتة "a" از کاراکتر ۵ تا ۶ • انطباق رشتة "a" از کاراکتر ۶ تا ۷ • انطباق رشتة "a" از کاراکتر ۷ تا ۸ • انطباق رشتة خالی از کاراکتر ۸ تا ۹ • انطباق رشتة خالی از کاراکتر ۹ تا ۹
a^*	$ababaaaab$	<p>در هفت نقطه،</p> <ul style="list-style-type: none"> • انطباق رشتة "a" از کاراکتر ۰ تا ۱ • انطباق رشتة خالی از کاراکتر ۱ تا ۱ • انطباق رشتة "a" از کاراکتر ۲ تا ۳ • انطباق رشتة خالی از کاراکتر ۳ تا ۳ • انطباق رشتة "aaaa" از کاراکتر ۴ تا ۸ • انطباق رشتة خالی از کاراکتر ۸ تا ۹ • انطباق رشتة خالی از کاراکتر ۹ تا ۹

در سه نقطه: <ul style="list-style-type: none"> • انطباق رشته "a" از کاراکتر ۰ تا ۱ • انطباق رشته "a" از کاراکتر ۲ تا ۳ • انطباق رشته "aaa" از کاراکتر ۴ تا ۸ 	bababaaaab	a⁺
--	------------	----------------------

جدول ۹-۱۹. مثالبایی از انطباق رشته های خالی (۳)

در مثال فوق، هیچ یک از عبارتهای باقاعده به دنبال کاراکتر **b** نیست، اما دقیقا در همان جایی که کاراکتر **b** قرار دارد یک انطباق رخ داده است. دلیل آن این است که عبارتهای باقاعده وقتی هیچ انطباقی پیدا نمی کنند انطباق رشته خالی را بررسی می کنند.

گروه بندی کاراکترها

عبارت های باقاعده ای که تا اینجا با آنها آشنا شدید همگی مبنی بر یک کاراکتر بودند، مثلا **a*** تمام رشته هایی را مشخص می کند که از کاراکتر **a** تشکیل شده اند. اما این امکان نیز وجود دارد تا با استفاده از پرانتز کاراکترها را گروه بندی کنیم و بدین ترتیب عبارتهای باقاعده ای تعریف کنیم که شامل مجموعه ای از کاراکترها می شود. مثلا **(abc)*** رشته هایی را مشخص می کند که حاوی رشته **abc** باشد یا عبارت **{3}** **(cat){3}** رشته هایی را مشخص می کند که از سه کلمه **cat** پشت سر هم تشکیل شده باشند. فراموش نکنید که از علامت **[]** برای مشخص کردن محدوده یک کاراکتر استفاده می شود. مثلا **[a-d]*** رشته هایی را مشخص می کند که از یکی از کاراکترهای **a, b, c, d** تشکیل شده باشند. یا مثلا عبارت **[abc]*** رشته هایی را مشخص می کند که از یکی از کاراکترهای **a, b, c** تشکیل شده باشد. برای درک بهتر گروه و محدوده کاراکترها به مثالهای زیر توجه کنید

عبارت	رشته	انطباق
(cat){3}	catcatcatcatcatcat	در دو نقطه, <ul style="list-style-type: none"> • انطباق رشته "catcatcat" از کاراکتر ۰ تا ۹ • انطباق رشته "catcatcat" از کاراکتر ۹ تا ۱۸
cat{3}	catcatcatcatcatcat	در هیچ نقطه ای انطباق رخ نمی دهد.
[abc]{3}	abccabaaaccbbbc	در پنج نقطه: <ul style="list-style-type: none"> • انطباق رشته "abc" از کاراکتر ۰ تا ۳ • انطباق رشته "cab" از کاراکتر ۳ تا ۶ • انطباق رشته "aaa" از کاراکتر ۶ تا ۹

<ul style="list-style-type: none"> • انطباق رشته "ccb" از کاراکتر ۹ تا ۱۲ • انطباق رشته "bbc" از کاراکتر ۱۲ تا ۱۵ 	در هیچ نقطه‌ای انطباق رخ نمی‌دهد.	abccabaaaccbbbc	$\{abc\}$
---	-----------------------------------	-----------------	-----------

جدول ۱۰-۱۹. گروه بندی کاراکترها

طبیعی است که عبارت باقاعده مبتنی بر بیش از یک کاراکتر باشد. با استفاده از نشانه پرانتز () می‌توان مجموعه‌ای از کاراکترها را به عنوان یک « واحد» مشخص نمود. دقت کنید که از () برای مشخص کردن یک رشته پیوسته استفاده می‌شود در حالیکه از [] برای مشخص کردن یکی از کاراکترها استفاده می‌شود. مثلاً عبارت + [cat] و + (cat) رشته‌هایی را مشخص می‌کنند که در آنها کلمه cat حداقل یکبار وجود دارد. در گروه بندی کاراکترها هیچ محدودیتی روی تعداد گروه‌ها وجود ندارد حتی گروه‌ها می‌توانند به صورت تودرتو تعریف شوند. مثلاً در عبارت ((A)(B(C))) چهار گروه زیر تعریف شده‌اند

- ((A)(B(C)))
- (A)
- (B(C))
- (C)

در انتهای این فصل، جایی که نحوه پردازش یک عبارت باقاعده را در زبان جاوا بررسی می‌کنیم با نحوه کشف تعداد گروه‌ها و پردازش هر گروه آشنا خواهید شد.

ارجاع به عقب

وقتی در یک عبارت باقاعده، یک گروه کاراکتر مشخص می‌کنیم این امکان وجود دارد تا بعد از آن گروه به آن گروه ارجاع دهیم به این عمل «ارجاع به عقب» گفته می‌شود. این کار با گذاشتن یک \ و یک عدد بعداز گروه انجام می‌شود. به عنوان مثال رشته‌هایی بر عبارت 1(a.) منطبق می‌شوند که در آنها گروه کاراکترهای a. یکبار پشت هم تکرار شده باشد از آنجاییکه . هر کاراکتری می‌تواند باشد، رشته‌های abab, amam, ... بیر این عبارت منطبق می‌شوند اما رشته‌های ab, abaa, abac, ... براین عبارت منطبق نمی‌شوند.

به عنوان مثالی دیگر اگر عبارت باقاعده 1(\d\d) را داشته باشیم رشته‌های 2121, 5454, ... بر آن منطبق می‌شوند و رشته‌های 54, 5455, ... منطبق نمی‌شوند. بر عبارت 2\1(a.\d\d) رشته 12ab12ab منطبق است.

توجه کنید که هر رشته‌ای که حاوی یک \ باشد در زبان جاوا باید به جای یک \ از دو \ استفاده شود.

انطباق در نقاط مرزی

تا این لحظه فقط انطباق یک رشته را بر یک عبارت باقاعده بررسی کردیم اما اینکه این انطباق در چه نقطه‌ای از رشته اتفاق بیفتند برای ما اهمیتی نداشت. اما این امکان وجود دارد تا در یک عبارت باقاعده به صورت

دقیقتر نقطه انتباط را مشخص کنیم. به عنوان مثال ممکن است در یک عبارت باقاعدۀ دنیال یک انتباط در ابتدای انتهای رشته باشد یا بخواهید انتباط دقیقاً بعد از انتباط قبلی یا در ابتدای انتهای یک کلمه باشد. برای این منظور از نشانه‌های مرزی استفاده می‌شود که در جدول زیر نشان داده شده‌اند.

نشانه‌های مرزی			
به عنوان مثال برای مشخص کردن کلمۀ Java که در شروع خط قرار دارد از عبارت <code>Java ^ Java</code> استفاده می‌شود.	شروع خط	۸	
برای یافتن کلمۀ Java که در انتهای خط قرار دارد از <code>Java \$</code> استفاده می‌شود. طبیعی است که اگر Java در انتهای جمله هم قرار داشته باشد (از . در انتهای جمله استفاده می‌شود) باید از عبارت <code>Java \\$. \$</code> استفاده کنید.	انتهای خط	\$	
منظور از کلمه، رشته‌ای است که ابتدا و انتهای آن فاصله (space) قرار داشته باشد. مثلاً برای یافتن کلماتی که با C شروع می‌شوند و به t ختم می‌شوند از <code>\bc.*t\bc</code> استفاده می‌شود بنابراین اگر در یک رشته کلمۀ cat وجود داشته باشد بر این عبارت منطبق می‌شود اما کلمات cati یا acat بر این عبارت منطبق نیستند.	محدودۀ یک کلمه	\b	
دقیقاً نقطۀ مقابل b است. مثلاً رشته <code>cati</code> بر <code>\bc.*t\B</code> منطبق می‌شود اما رشته <code>acat</code> و <code>cat</code> بر آن منطبق نمی‌شوند.	محدودۀ یک کلمه نباشد	\B	
انتباط را روی ابتدای رشته بررسی می‌کند. مثلاً اگر بخواهیم رشته‌هایی را مشخص کنیم که ابتدای آن کلمۀ The قرار دارد از عبارت <code>\AThe\b</code> استفاده می‌کنیم دقت کنید که نوشتن b لازم است زیرا اگر آن را ننویسیم کلمات <code>Then</code> و <code>There</code> نیز انتخاب می‌شوند.	ابتدای ورودی	\A	
انتباط را در انتهای رشته بررسی می‌کند. مثلاً اگر بخواهیم رشته‌هایی را مشخص کنیم که به کلمۀ country ختم می‌شوند می‌توانیم از عبارت <code>\bcountry\b\z</code> استفاده کنیم.	انتهای ورودی	\z	
مشابه z عمل می‌کند با این تفاوت که اگر انتباط در انتهای خط اتفاق بیفتد، کاراکتر «خط جدید» نیز بخشی از انتباط خواهد بود.	انتهای ورودی به همراه انتهای خط	\Z	
اگر عبارت باقاعدۀ cat را داشته باشیم رشته <code>cat</code> در یک نقطه و رشته <code>\Gcat</code> در دو نقطه برآن منطبق می‌شود اما اگر عبارت باقاعدۀ را به <code>\Gcat</code> تغییر دهیم رشته <code>cat</code> بر آن منطبق نمی‌شود و رشته <code>cat cat</code> فقط در یک نقطه (کاراکتر + تا ۳) بر آن منطبق می‌شود.	انتهای انتباط قبلی	\G	

جدول ۱۹-۱۱. انتباط در نقاط مرزی (۱)

عبارت‌های با قاعده

در جدول فوق، «انتهای خط» نیز به عنوان نقاط مرزی معرفی شده است. به بیشتر موارد رشته‌ای داریم و می‌خواهیم انطباق آن رشته روی عبارت باقاعده را بررسی کنیم، اما این امکان نیز وجود دارد تا انطباق محتویات یک فایل متنی بر یک عبارت باقاعده را بیابیم. نشانه‌های مرزی ابتداء و انتهای خط امکان می‌دهد تا کنترل بیشتری در انطباق محتویات فایل بر عبارت باقاعده داشته باشیم.

جدول ۱۹-۱۲ مثالهای بیشتری از استفاده از نشانه‌های مرزی را نشان می‌دهد.

انطباق	رشته	عبارة
یک انطباق از کاراکتر ۰ تا ۳	"cat"	^cat\$
بدون انطباق	"cat	
یک انطباق از کاراکتر از ۰ تا ۱۹	"cat	\s*cat\$
یک انطباق از کاراکتر از ۰ تا ۱۱	"catblahblah"	^cat\w*

جدول ۱۹-۱۲. انطباق در نقاط مرزی (۲)

پردازش عبارت باقاعده در جاوا

تا اینجا فقط نحوه نوشتن عبارات باقاعده و چگونگی انطباق رشته‌ها بر آنها بررسی کردیم. طبیعی است که بخواهیم با استفاده از زبان جاوا عبارتهای باقاعده را پردازش کنیم، نقاط انطباق رشته‌های متنی را بر عبارتهای باقاعدۀ بیابیم، یا رشته‌ها دستکاری کنیم. در این قسمت از فصل، قصد دارم چگونگی استفاده از زبان جاوا برای کار با عبارتهای باقاعدۀ، پردازش متن، کشف نقاط انطباق، و دستکاری متن به شما آموزش دهم. کلاس‌های مربوط به عبارتهای باقاعدۀ در پکیج `java.util.regex` قرار دارند. کلاس‌های `Pattern` و `Matcher` از مهمترین این کلاسها هستند. `Pattern` معرف یک عبارت باقاعدۀ است و `Matcher` شیئی است که نقاط انطباق یک رشته را بر عبارت باقاعدۀ پیدا می‌کند. تکه کد زیر نحوه استفاده از این دو کلاس برای پردازش یک عبارت باقاعدۀ و کشف نقاط انطباق را نشان می‌دهد.

```
Pattern pattern = Pattern.compile("\\Gcat");  
Matcher matcher = pattern.matcher("cat cat");  
  
if(matcher.matches()){  
    System.out.println("Entire Input String Matched.");  
}else{  
    System.out.println("No Match Point Found!");  
}
```

همانطور که ملاحظه می کنید برای ایجاد آبجکت Pattern متده استاتیک compile() از همین کلاس فراخوانی شده است در مثال فوق، متده compile() عبارت "Gcat" را دریافت نموده و آبجکتی از کلاس Pattern که معرف این عبارت باقاعدۀ \Gcat است را برگردانده است. برای کشف نقاط انطباق رشته "cat cat" بر عبارت باقاعدۀ فوق، متده matches() از آبجکت Matcher فراخوانی شده است. آبجکت Matcher دارای متده matches() است که انطباق تمام رشته را بر عبارت باقاعدۀ بررسی می کند اگر تمام رشته بر عبارت باقاعدۀ منطبق باشد true و در غیر اینصورت false برمه گرداند.

کد فوق، ساده ترین شکل استفاده از کلاسهای Pattern و Matcher را نشان می دهد این کلاسها همانطور که در ادامه خواهید دید کنترل کاملی روی پردازش یک عبارت و کشف نقاط انطباق یک رشته روی آن را فراهم می کنند.

ایجاد آبجکت Pattern

متده compile() در کلاس Pattern یک عبارت باقاعدۀ را دریافت می کنند و آبجکت Pattern که معرف آن عبارت باقاعدۀ باشد را برمه گرداند. متده compile() به دو شکل زیر در کلاس Pattern پیاده سازی شده است:

```
public static Pattern compile(String regex)
public static Pattern compile(String regex, int flags)
```

هر دو متده کارکرد مشابهی دارند تنها تفاوت آنها پارامتر flags است که در متده دوم وجود دارد. این پارامتر که ما به آن «شرط انطباق» می گوییم یک عدد int است و نحوه انطباق رشته بر عبارت باقاعدۀ را تعیین می کند. مثلا می توانیم با استفاده از آن مشخص کنیم که انطباق نباید به کوچک و بزرگ بودن کاراکترها حساس باشد یا به عنوان مثالی دیگر، وجود کاراکتر فاصله (space) در عبارت باقاعدۀ هیچ معنی ندارد و باید نادیده گرفته شود. هر کدام از این شرایط به صورت یک مقدار ثابت در کلاس Pattern تعریف شده است و در فراخوانی متده compile() می توانیم از آنها استفاده کنیم. این مقادیر در جدول زیر توضیح داده شده اند.

شرط انطباق	توضیح
CASE_INSENSITIVE	انطباق را بدون توجه به کوچک یا بزرگ بودن حروف رشته بررسی می کند.
MULTILINE	رشته ورودی را در قالب چند خط تفسیر می کند به صورت معمول که از این مقدار استفاده نمی شود، حتی اگر رشته ورودی چند خط باشد آنرا در به عنوان یک خط تفسیر می کند و اگر در عبارت باقاعدۀ نشانه های ^ و \$ که مشخص کننده ابتداء و انتهای یک خط هستند وجود داشته باشد، انطباق شروع و پایان خط را فقط در ابتداء و انتهای رشته جستجو می کند. اما اگر رشته

عبارت‌های با قاعده

وروی از چندین خط تشکیل شده باشد و از نشانه‌های ^ و \$ در عبارت باقاعدۀ استفاده شده باشد باید از علامت MULTILINE استفاده نمود.	
از این مقدار همزمان با مقدار CASE_INSENSITIVE استفاده می‌شود به صورت معمول اگر از CASE_INSENSITIVE استفاده شود، مقایسه کاراکترها در کدگذاری US-ASCII انجام می‌شود، استفاده از Unicode_ case امکان مقایس کاراکترها را در کدگذاری Unicode امکان‌پذیر می‌کند.	UNICODE_CASE
اگر از کاراکتر . در عبارت باقاعدۀ استفاده شود، آنرا به «هر کاراکتری» از جمله پایان خط تفسیر می‌کند. به صورت معمول که از این مقدار استفاده نمی‌شود، کاراکتر . در عبارت باقاعدۀ به هر کاراکتری به جز کاراکتر پایان خط تفسیر می‌شود.	DOTALL
وقتی از این مقدار استفاده شود، عبارت باقاعدۀ فقط به عنوان یک متن ساده تفسیر می‌شود و میانبرها و کاراکترهای کنترلی فقط نقش پیش فرض خود را از دست می‌دهند.	LITERAL
اگر در عبارت باقاعدۀ فاصله (space) وجود داشته باشد آنرا نادیده گرفته و لحاظ نمی‌کند. همچنین هر عبارتی که بعد از # قرار بگیرد را نیز به عنوان جملات توضیحی در آن عبارت در نظر می‌گیرد و تا آخر خطی که # قرار دارد را نادیده می‌گیرد.	COMMENTS
فقط کاراکتر \n به عنوان کاراکتر انتهای خط تفسیر می‌شود	UNIX_LINES

جدول ۱۴-۱۳. شرایط انطباق

مثال زیر استفاده از شرط CASE_INSENSITIVE را نشان می‌دهد.

```
Pattern pattern = Pattern.compile("cat",
Pattern.CASE_INSENSITIVE);
```

به این ترتیب، رشتۀ "DOGDOG" در دو نقطه (کاراکتر ۰ و کاراکتر ۳) بر عبارت فوق منطبق خواهد بود.

توجه کنید که با استفاده از عملگر OR بیتی می‌توان شرایط فوق را ترکیب نمود. به مثال زیر توجه کنید:

```
pattern = Pattern.compile("[az]$",
Pattern.MULTILINE |
Pattern.UNIX_LINES);
```

علاوه براینکه می‌توان شرایط انطباق را از طریق پارامتر flags متد compile() مشخص نمود، می‌توان در متن عبارت باقاعدۀ نیز شرایط انطباق را گنجاند. به عنوان مثال، cat(?i) عبارت باقاعدۀ ای است که

رشته های cat را بدون توجه به کوچک یا بزرگ بودن حروف مشخص می کند. جدول زیر، لیست تمام عبارتهايی که می توان از آنها در عبارتهاي باقاعدۀ استفاده نمود نشان داده شده اند

عبارت معادل (عبارت با قاعده)	شرط انطباق (در کلاس Pattern)
معادل ندارد	Pattern.CANON_EQ
(?i)	Pattern.CASE_INSENSITIVE
(?x)	Pattern.COMMENTS
(?m)	Pattern.MULTILINE
(?s)	Pattern.DOTALL
معادل ندارد	Pattern.LITERAL
(?u)	Pattern.UNICODE_CASE
(?d)	Pattern.UNIX_LINES

جدول ۱۴-۱۴. عبارتهاي متناظر با شرایط انطباق

کلاس Pattern

علاوه بر دو متند استاتیک compile() که در با آن آشنا شدید، کلاس Pattern دارای یک متند استاتیک دیگر matches() نیز می باشد که بدون ایجاد آبجکت Pattern می توانید انطباق یک رشته بر روی یک عبارت را بررسی کنید.

```
if(Pattern.matches("\\Gcat, "cat")){
    System.out.println("Entire Input String Matched.");
} else{
    System.out.println("No Match Point Found!");
}
```

کلاس Pattern دارای متند استاتیک quote نیز هست که یک رشته دریافت می کند و یک عبارت باقاعدۀ که آن رشته بر آن منطبق است را تولید می کند. علاوه بر متدهای استاتیک فوق، کلاس Pattern دارای تعدادی متند غیراستاتیک نیز هست که امکان می دهد پس از ایجاد شدن آبجکت Pattern رشته های مختلف را پردازش کرد، یا نقاط انطباق یک رشته را بر عبارت باقاعدۀ یافت. جدول زیر به صورت مختصر این متدها را نشان می دهد.

متند	توضیح
Matcher matcher(CharSequence input)	این متند یک رشته دریافت می کند و آبجکتی از کلاس Matcher بر می گرداند. آبجکت Matcher امکان یافتن نقاط انطباق رشته بر عبارت باقاعدۀ را فراهم می کند.

عبارت‌های با قاعده

این متدهای رشته را دریافت می‌کند و آنرا بر مبنای نقاط انتطاقی بر عبارت باقاعده به رشته های دیگر تجزیه می‌کند.	String[] split(CharSequence input)
این متدهای متناسب متد فوق، یک رشته را بر مبنای نقاط انتطاقی بر عبارت باقاعده به رشته های دیگر تجزیه می‌کند با این تفاوت که حداقل اندازه آرایه‌ای که برگردانده می‌شود را نیز به عنوان پارامتر دریافت می‌کند.	String[] split(CharSequence input, int limit)
رشته عبارت باقاعده‌ای که معادل آبجکت Pattern است را بر می‌گرداند. در واقع خلاف عمل متد compile می‌کند در حالیکه متد compile از روی یک رشته عبارت باقاعده، آبجکت Pattern معادل آنرا می‌سازد، متد toString از روی آبجکت Pattern رشته عبارت باقاعدۀ معادل آنرا بر می‌گرداند.	String toString()

جدول ۱۹-۱۵. متدهای کلاس Pattern

در ادامه هریک از متدهای فوق با یک مثال توضیح داده شده است.

متدهای split(String input)

متدهای split یک ابزار بسیار مناسب برای تجزیه یک رشته بر مبنای نقاط انتطاقی آن رشته روی عبارت باقاعده است. به عنوان مثال فرض کنید بخواهیم رشته های one, two, three, four را از رشته "one:two:three:four:five" استخراج کنیم برای این منظور کافیست با استفاده از متد split این رشته را بر مبنای عبارت ":" تجزیه کنیم. کد زیر نحوه انجام این کار را نشان می‌دهد.

```
1 import java.util.regex.Pattern;
2 import java.util.regex.Matcher;
3
4 public class SplitDemo {
5
6     private static final String REGEX = ":";
7     private static final String INPUT =
8         "one:two:three:four:five";
9 }
```

```

10     public static void main(String[] args) {
11         Pattern p = Pattern.compile(REGEX);
12         String[] items = p.split(INPUT);
13         for(String s : items) {
14             System.out.println(s);
15         }
16     }
17 }
```

کد ۱۹-۱

نتیجه اجرای کد فوق، جملات زیر در کنسول برنامه خواهد بود.

```

one
two
three
four
five
```

در مثال فوق، رشته "one:two:three:four:five" را به سادگی می‌توان بر مبنای عبارت ساده ":" تجزیه نمود. اما عبارت باقاعده می‌تواند پیچیده‌تر باشد، مثلاً اگر بخواهیم رشته "one9two4three7four1five" را به رشته‌های one, two, three, four, five بسازیم کافیست از عبارت باقاعدۀ "\d" استفاده نماییم. (\d معرف یک کاراکتر عددی است). کد زیر نحوه اینکار را نشان می‌دهد.

```

1 import java.util.regex.Pattern;
2 import java.util.regex.Matcher;
3
4 public class SplitDemo2 {
5
6     private static final String REGEX = "\\d";
7     private static final String INPUT =
8 "one9two4three7four1five";
9
10    public static void main(String[] args) {
11        Pattern p = Pattern.compile(REGEX);
12        String[] items = p.split(INPUT);
13        for(String s : items) {
14            System.out.println(s);
15        }
16    }
17 }
```

```

15     }
16 }
17 }
```

کد ۱۹-۲

خروجی اجرای کد فوق مشابه کد قبلی خواهد بود.

متدها

عملکرد متدها مشابه عملکرد متدهای split(String input, int limit) است با این تفاوت که شما می‌توانید روی اندازه آرایه‌ای که تولید می‌شود محدودیت بگذارید. رشته "one9two4three7four1five8siz3seven9eighth5nine" را تصور کنید واضح است که اگر مشابه قبل آنرا بر اساس کاراکترهای عددی تجزیه کنیم به رشته‌های one، two، three، two، one، three، two، one، five، four، six، seven، eight، nine و five، eight، seven، nine، eighth، five، nine خواهد شد. اگر بخواهیم اندازه آرایه حداقل ۵ باشد، در اینصورت رشته فوق به رشته‌های one، two، one، three، two، one، five، four، six، seven، eight، nine تجزیه خواهد شد. در نتیجه آرایه‌ای با تعداد ۹ تولید خواهد شد. اگر بخواهیم اندازه آرایه حداقل ۶ باشد، در اینصورت رشته فوق به رشته‌های one، two، one، three، two، one، five، four، six، seven، eight، nine تجزیه خواهد شد. کد زیر انجام اینکار را نشان می‌دهد.

```

1 import java.util.regex.Pattern;
2 import java.util.regex.Matcher;
3
4 public class SplitDemo2 {
5
6     private static final String REGEX = "\\d";
7     private static final String INPUT
8 ="one9two4three7four1five8siz3seven9eighth5nine";
9
10    public static void main(String[] args) {
11        Pattern p = Pattern.compile(REGEX);
12        String[] items = p.split(INPUT, 5);
13        for(String s : items) {
14            System.out.println(s);
15        }
16    }
17 }
```

کد ۱۹-۳

متد **matcher(String input)**

فراخوانی این متد آبجکتی از کلاس `Matcher` برمی گرداند که امکان پردازش رشته و کشف نقاط انطباق آنرا می دهد. در قسمت بعدی جزئیات کلاس `Matcher` شرح داده خواهد شد.

برخی متدهای کلاس **String**

برخی متدهای کلاس `String` زیر از عبارتهای باقاعدۀ پشتیبانی می کنند، که برخی از آنها در زیر توضیح داده شده اند.

```
public boolean matches(String regex)
```

متد `()` در کلاس `String` مشخص می کند که آیا آن رشته بر یک عبارت باقاعدۀ منطبق است یا خیر. فراخوانی این متد به صورت `; str.matches(regex)` دقیقاً معادل `Pattern.matches(regex, str);` خواهد بود.

```
public String[] split(String regex, int limit)
```

رشته را بر اساس عبارت باقاعدۀ `regex` را به آرایه ای از رشته ها با حداکثر اندازه `limit` تجزیه می کند. فراخوانی `str.split(regex, n)` دقیقاً معادل `Pattern.compile(regex).split(str, n)` است.

```
public String[] split(String regex)
```

عملکرد این متد مشابه متد فوق است با این تفاوت که محدودیتی روی اندازه آرایه ای که برگردانده می شود وجود ندارد.

```
public String replace(CharSequence target, CharSequence replacement)
```

در این متد `target` عبارت باقاعدۀ ای است که تمام نقاط انطباق رشته بر آن با رشته `replacement` جایگزین می شود.

کلاس **Matcher**

نتیجه فراخوانی متد `()` از آبجکت `Pattern` خود آبجکتی از کلاس `Matcher` است که همانطور که قبلاً گفته شد امکان پردازش رشته بر مبنای عبارت باقاعدۀ و کشف نقاط انطباق آن را فراهم می کند. متدهایی که در این کلاس وجود دارند از نظر عملکرد به سه دسته قابل تقسیم هستند که در ادامه هر دسته به صورت جداگانه بحث شده اند.

عبارت‌های با قاعده

متدهای موقعیت یاب

این متدها که موقعیت ابتدا و انتهای نقاط انطباق یک رشته بر عبارت باقاعده را برمی‌گردانند عبارتند از:

```
public int start()  
public int start(int group)  
public int end()  
public int end(int group)
```

نقطه شروع یک انطباق را برمی‌گرداند.

متدهای مطالعاتی

```
public boolean lookingAt()  
public boolean find()  
public boolean find(int start)  
public boolean matches()
```

متدهای جایگزینی

```
public Matcher appendReplacement(StringBuffer sb, String  
replacement)  
public StringBuffer appendTail(StringBuffer sb)  
public String replaceAll(String replacement)  
public String replaceFirst(String replacement)  
public static String quoteReplacement(String s)
```

در ادامه با برخی از متدهای با اهمیت فوق و نحوه استفاده از آنها آشنا خواهید شد.

استفاده از متدهای end و start

در زیر مثالی نشان داده شده است که تعداد کلمات dog را در یک رشته ورودی می‌شمارد.

```
1 import java.util.regex.Pattern;  
2 import java.util.regex.Matcher;  
3  
4 public class DogCounter {  
5  
6     private static final String REGEX = "\\\\bdog\\\\b";  
7     private static final String INPUT = "dog dog dog doggie  
8 dogg";
```

```

9
10    public static void main(String[] args) {
11        Pattern p = Pattern.compile(REGEX);
12        Matcher m = p.matcher(INPUT); // get a matcher object
13        int count = 0;
14        while(m.find()) {
15            count++;
16            System.out.println("Match number "+count);
17            System.out.println("start(): "+m.start());
18            System.out.println("end(): "+m.end());
19        }
20    }
21 }
```

۱۹-۴ کد**نتیجه اجرای کد فوق، خروجی زیر است**

```

Match number 1
start(): 0
end(): 3
Match number 2
start(): 4
end(): 7
Match number 3
start(): 8
end(): 11
```

متدهای `start()` و `end()` به ترتیب نقطه شروع و نقطه پایان dog بعدی را برمی گرداند.

استفاده از متدهای lookingAt و matches

هر دو متدهای `lookingAt()` و `matches()` انتطاق یک رشته بر یک عبارت باقاعده را بررسی می کنند. تفاوت آنها در این است که متدهای `matches()` انتطاق تمام رشته بر عبارت باقاعده را بررسی می کند در حالیکه متدهای `lookingAt()` انتطاق بخشی از رشته بر عبارت باقاعده را نیز بررسی می کند. مثال زیر استفاده از این دو متدهای را نشان می دهد.

```

1 import java.util.regex.Pattern;
2 import java.util.regex.Matcher;
3
4 public class MatchesLooking {
5
6     private static final String REGEX = "foo";
```

عبارت‌های با قاعده

```
7     private static final String INPUT = "oooooooooooooooooooo";
8     private static Pattern pattern;
9     private static Matcher matcher;
10
11    public static void main(String[] args) {
12
13        // Initialize
14        pattern = Pattern.compile(REGEX);
15        matcher = pattern.matcher(INPUT);
16
17        System.out.println("Current REGEX is: "+REGEX);
18        System.out.println("Current INPUT is: "+INPUT);
19
20        System.out.println("lookingAt():");
21        System.out.println("matches(): "+matcher.matches());
22
23    }
24}
25}
```

کد ۱۹-۵

نتیجه اجرای این متدهای زیر خواهد بود

```
Current REGEX is: foo
Current INPUT is: fooooooooooooooooooooo
lookingAt(): true
matches(): false
```

متدهای معادل در کلاس `java.lang.String`

در کلاس `String` متدهایی پیاده سازی شده اند که کارکردی مشابه برخی متدهای `Matcher` دارند دو متدهایی هستند که از آنها برای جستجوی متن استفاده می‌شوند:

```
public String replaceFirst(String regex, String replacement)
public String replaceAll(String regex, String replacement)
```

کلاس `PatternSyntaxException`

این کلاس معرف یک استثناء زمان اجراست که در پردازش عبارتهای باقاعده ممکن است رخ دهد. این کلاس حاوی چهار متدهای است:

```
public String getDescription()
```

توضیحی در مورد خطای پردازش عبارت باقاعده را برمی گرداند.

```
public int getIndex()
```

موقعیتی در رشته ورودی که خطای خطا رخ داده است را برمی گرداند.

```
public String getPattern()
```

عبارت باقاعده ای که در استفاده از آن خطای خطا رخ داده است را برمی گرداند.

```
public String getMessage()
```

یک پیغام چندخطی از خطای موقعیت خطا، عبارت باقاعده ای که استفاده شده است را برمیگرداند.

خلاصه

عبارت‌های با قاعده رشته‌هایی هستند که مجموعه‌ای از رشته‌های دیگر را توصیف می‌کنند. عبارت‌های با قاعده که از قوانین خاصی پیروی می‌کنند توسط آبجکت‌های `java.util.regex.Pattern` و `java.util.regex.Matcher` پردازش می‌شوند. با استفاده از متدهای مختلفی که در این کلاس وجود دارد می‌توانید نقاط انطباق یک رشته بر یک عبارت با قاعده را بررسی کنید.

با استفاده از علایم `*`، `+`، `?` می‌توانید تعداد تکرار یک کاراکتر یا یک مجموعه کاراکتر را در یک رشته مشخص کنید. کاراکترهای `..`، `\d`، `\w`، `\S`، `\s` به ترتیب برای اشاره به «هر کاراکتری»، «کاراکتر عددی»، «کاراکتر غیر عددی»، «کاراکتر فاصله و کاراکترهای غیر قابل مشاهده»، «کاراکترهای قابل مشاهده»، «اعداد و حروف الفبا» و «کاراکترهای غیر عددی و غیر الفبا» استفاده کنید.

تمرینات

- (۱) عبارت باقاعدۀ ای برای ایمیل، آدرس اینترنتی، کد ملی، کارت اعتباری بنویسید.
- (۲) کلاسی پیاده سازی کنید که لیستی از عبارتهای باقاعدۀ را از طریق سازنده خود دریافت می کند.
این کلاس یک متّد `consume()` دارد که رشته ای را دریافت می کند و انطباق آن رشته با هریک از عبارتهای باقاعدۀ ای که قبلا دریافت کرده را برسی می کند.
- (۳) کلاسی پیاده سازی کنید که یک عبارت با قاعده را از طریق سازنده خود دریافت می کند و دارای یک متّد `match()` می باشد که آرایه ای از رشته ها را دریافت می کند و نتیجهٔ انطباق هریک از رشته های آرایه را با عبارت باقاعدۀ را در قالب یک آرایه `boolean` برمی گرداند.

۳.

Annotation

در فصل اول با توضیحات برنامه آشنا شدید. به توضیحاتی که لایلای کدهای برنامه نوشته می‌شوند `comment` گفته می‌شود که البته جنبه اجرایی ندارند و توسط کامپایلر جاوا صرفنظر می‌شوند اما به برنامه نویسان دیگر (یا حتی خود شما) در آینده کمک می‌کنند تا کدهای شما را بفهمند و بتوانند آنها را تغییر دهند.

توضیحات می‌توانند «یک خطی» یا «چندخطی» باشند. توضیح یک خطی با استفاده از علامت `//` و توضیح چند خطی بین علائم `/*` و `*/` قرار داده می‌شوند. اگر یک توضیح چند خطی بین علائم `/*` و `*/` قرار گیرد به آن `JavaDoc` گفته می‌شود. تفاوت توضیحات چندخطی معمولی با `JavaDoc` در این است که توضیحات `JavaDoc` را می‌توان با استفاده از ابزار مخصوصی که در JDK قرار دارد از سورس برنامه استخراج نمود و در قالب فایل `HTML` جداگانه ارایه نمود. نام این ابزار نیز `javadoc` است. ممکن است بپرسید که در توضیحات `JavaDoc` چه چیزهایی نوشته می‌شوند. در جواب باید بگوییم هر توضیحی که در زمان استفاده از کد (و در زمان تغییر کد) به آن نیاز داریم. مثلاً `JavaDoc` مربوط به یک متدهای توسعه دهد که هدف آن متدهایی است که پارامترهایی دارد و چه استفاده ای می‌شوند، متدهای `Exception` که در زمان تولید می‌کند و مقدار برگشتی آن متدهایی است. بنابراین با مطالعه `JavaDoc` مربوط به یک متدهایی که طرز صحیحی آن متدهایی دارند می‌توان آن را فراخوانی کنید. در `JavaDoc` مربوط به کلاس معمولاً نام برنامه نویس (یا برنامه نویسان) آن کلاس، زمان نوشتن آن کلاس، و عملکرد کلاس شرح داده می‌شود.

اما annotation که موضوع این فصل است هیچ ارتباطی با توضیحات برنامه و JavaDoc ندارد! اما برای اینکه کاربرد و کارکرد annotation را بیاموزید می توانیم با یک مثال ساده در مورد JavaDoc شروع کنیم. برای این منظور به JavaDoc زیر که برای کلاس Simple نوشته شده است توجه کنید.

```

1 package ir.atlassoft.javase.chapter20;
2
3 /**
4 * Created:           Jan 31 2012
5 * @author:           Ahmad R. Seddighi
6 * Last Modified:    Nov 23 2012
7 * Last Modified By: Ahmad R. Seddighi
8 * @version:          3
9 */
10 public class Simple {
11     // ...
12 }
```

کد ۲۰-۱

در توضیحات `@deprecated`, `@version`, `@author` و `@since` استفاده نمود که هر یک توضیحی در مورد یک موضوع خاص ارایه می کنند مثل `@author` نام برنامه نویس یا برنامه نویسان آن کلاس را مشخص می کند.

همانطور که JavaDoc برای مستندسازی کدهای برنامه استفاده می شود، Annotation شیوه دیگری از مستندسازی است. Annotation‌ها مشابه JavaDoc‌ها جنبه اجرایی ندارند (یعنی فقط توضیح هستند و طبیعتاً اجرا نمی شوند) اما برخلاف JavaDoc‌ها به زبان جاوا نوشته می شوند و بنابراین توسط کامپایلر و اجرا کننده جاوا قابل فهم و تفسیر هستند. یک تفاوت مهم دیگر نیز بین JavaDoc و Annotation وجود دارد و آن این است که Annotation‌ها برخلاف JavaDoc‌ها توسط برنامه‌های جاوا تفسیر و استفاده می شوند در حالیکه JavaDoc‌ها توسط برنامه نویسان قابل فهم و تفسیر هستند. کلاسها و اینترفیس‌های مربوط به `annotation` در پکیج `java.lang.annotation` قرار دارند. معرفی Java 5 در تحول بزرگی را در شیوه برنامه نویسی جاوا خصوصاً برنامه نویسی سمت سورور یا همان Java EE ایجاد نمود.

یک مثال ساده از Annotation

Annotation چیزی شبیه کلاس است که در جایی از برنامه تعریف می شود و در جاهای دیگر از روی آن آجکت ساخته می شود. اما برخلاف کلاس که می تواند حاوی متدها و فیلد باشد، Annotation فقط حاوی داده است و هیچ رفتاری ندارد.

Annotation

برای اینکه بتوانیم توضیحاتی که قبل از کلاس Simple در مثال فوق آمده را توسط annotation ارایه کنیم ابتدا باید annotation مربوطه را تعریف کنیم که مشخص کننده نام و نوع توضیحاتی است که قرار است ارایه شود. این annotation در زیر آمده است.

```
1 package ir.atlassoft.javase.chapter20;  
2  
3 public @interface ClassInfo {  
4     String created();  
5     String createdBy();  
6     String lastModified();  
7     String lastModifiedBy();  
8     int revision();  
9 }
```

۲۰-۳

همانطور که ملاحظه می کنید برای تعریف annotation از کلمه @interface استفاده می شود که بلافصله بعد از آن، نام annotation آمده است. این نام، یک نام دلخواه است که از قواعد نامگذاری جاوا پیروی می کند بعد از نام، بدن annotation با { شروع و به } ختم می شود که حاوی متدهایی همانند متدهای اینترفیس است که بدن ندارند و به ; ختم می شوند. این متدها در واقع نوع داده هایی که annotation ارایه می کند را تعریف می کنند. در مثال فوق متدهای String created(); و String revision(); مشخص می کند که این annotation یک داده به نام created از جنس String دارد متدهای int revision(); مشخص می کند که داده ای با نام revision از جنس int توسط این annotation ارایه می شود.

حال که همراه نام و نوع توضیحات آن تعریف شد می توان آنرا قبل از هر عنصری از برنامه از قبیل کلاس، فیلد، متدها... قرار داد تا حاوی توضیحاتی برای آن عنصر از برنامه باشد. کد زیر نحوه انجام این کار را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter20;  
2  
3 @ClassInfo(  
4     created= "Jan 31 2012",  
5     createdBy= "Ahmad R. Seddighi",  
6     lastModified= "Nov 23 2012",  
7     lastModifiedBy= "Ahmad R. Seddighi",  
8     revision=3  
9 )
```

```

10 public class Simple {
11     // ...
12 }
```

۲۰-۳

توضیحاتی که توسط ClassInfo برای کلاس Simple ارایه شده است دیگر یک متن ساده که فقط توسط انسان قابل فهم و تفسیر باشد و توسط کامپایلر جاوا صرفنظر شود نیست بلکه کدی است که توسط کامپایلر جاوا کامپایل می شود و توسط زبان جاوا قابل درک و پردازش است.

همانطور که ملاحظه می کنید، برای استفاده از annotation، نام annotation که قبل از آن علامت @ قرار گرفته است بدون نیاز به هیچ کلمه دیگری می آید سپس مقادیر annotation بین () و () قرار می گیرند. مقادیر که قالب name=value دارند توسط کاما (,) از یکدیگر جدا می شوند.

Annotation انواع

Annotation ها را باید نوع خاصی از اینترفیس قلمداد کرد زیرا همانطور که دیدید با کلمه کلیدی interface که به ابتدای آن @ اضافه شده است تعریف می شوند. هرجایی که بتوان یک اینترفیس تعریف و پیاده سازی نمود می توان یک Annotation نیز تعریف و پیاده سازی کرد مثلاً جایی که می توان یک اینترفیس داخلی تعریف کرد می توان یک Annotation داخلی نیز تعریف نمود. مثلاً کلاس AnnoDemo را در زیر ملاحظه کنید.

```

1 package ir.atlassoft.javase.chapter20;
2
3 public class AnnoDemo{
4
5     interface Inner{
6         int process();
7     }
8
9     @interface InnerAnno{
10        String description();
11    }
12 }
```

۲۰-۴

داخل این کلاس یک اینترفیس داخلی به نام Inner و یک Annotation داخلی با نام InnerAnno تعریف شده است.

Annotation

مشابه اینترفیس، ممکن است با هر یک از کلمات `private`, `public`, `protected` یا `Annotation` تعریف شود.

متدهایی که در بدنه `annotation` تعریف می شوند به عنوان اجزا یا عناصر `Annotation` شناخته می شوند که هر کدام نام و نوع دارد، نام عنصر همان نام متده است و نوع عنصر مقدار برگشتی متده است. عناصر `annotation` باید طبق قواعد زیر نوشته شوند:

- نوع عنصر فقط می تواند داده های پایه از قبیل `int`, `long`, `boolean`, ...، `String`، یک `enum`، یک `annotation` دیگر، یک `Class` باشد، یا آرایه ای از این نوع داده ها.
- هیچ یک از متدهای `annotation` نمی توانند پارامتر داشته باشند.
- متدهای `annotation` نمی توانند `throws` داشته باشند.
- متدهای `annotation` نمی توانند فرم `generic` داشته باشند.

وقتی از یک `annotation` استفاده می شود باید مقدار تمام عناصر آن مشخص شود. مثلا در مثلا `Simple ClassInfo` شما نمی توانید وقتی از آن بالای کلاس `Simple` استفاده می کنید مقداری را برای `created` مشخص نکنید. اما اگر آن عنصر مقدار پیش فرض داشته باشد می توانید مقداری به آن ندهید و در عوض مقدار پیش فرض آن در نظر گرفته خواهد شد. برای مشخص کردن مقدار پیش فرض از کلمه `default` استفاده می شود. به عنوان مثال تصور کنید می خواهید نسخه یک کلاس را از طریق `annotation` مشخص کنید نسخه دارای دو مقدار اصلی و فرعی است مثلا در نسخه ۲،۱ عدد ۲ مقدار اصلی و عدد ۱ مقدار فرعی است. `annotation` ای که مشخص کننده نسخه یک کلاس است به شکل زیر تعریف می شود:

```
1 package ir.atlassoft.javase.chapter20;  
2  
3 public @interface Revision{  
4     int major() default 1;  
5     int minor() default 0;  
6 }
```

۲۰-۵ کد

طبعی است که مقداری که به عنوان پیش فرض مشخص می شود یک مقدار ثابت خواهد بود.

توجه: مقدار یک عنصر نمی تواند `null` باشد.

حال اگر بخواهیم annotation فوق را در جایی از برنامه بکار ببریم می توانیم آنرا به هریک از اشکال زیر استفاده کنیم.

```
@Revision
@Revision{
    major =2;
}
@Revision{
    minor = 1;
}
@Revision{
    major = 2;
    minor=1;
}
```

یک تواند یک عنصر از annotation دیگر تعریف شود مثلاً می توان عنصر `revision` که یک مقدار int دارد به صورت زیر با ClassInfo جایگزین کنیم.

```
1 package ir.atlassoft.javase.chapter20;
2
3 public @interface ClassInfo {
4     String created();
5     String createdBy();
6     String lastModified();
7     String lastModifiedBy();
8     Revision revision();
9 }
```

۲۰-۶

حال محلی که استفاده می شود می بایست عنصر `revision` نیز مشخص شود.

```
1 package ir.atlassoft.javase.chapter20;
2
3 @ClassInfo (
4     created = "Jan 31 2012",
5     createdBy = "Ahmad R. Seddighi",
6     lastModified = "Nov 23 2012",
7     lastModifiedBy = "Ahmad R. Seddighi",
8     revision = @Revision
9 )
```

Annotation

```
10 public class Simple {  
11     // ...  
12 }
```

کد ۲۰-۷

دقت کنید که چون هر دو عنصر `Revision` دارای مقادیر پیش فرض هستند آوردن جمله `@Revision` = کافیست اما عناصر آن مقدار اولیه نداشتند می بایست مقدار عناصر آن مشخص می شد. یک `annotation` ممکن است هیچ عنصری نداشته باشد، در اینصورت به آن «نشانگر» یا `marker` گفته می شود. به عنوان نمونه، `java.lang.Deprecated` یک نشانگر است. وقتی این `annotation` بالای یک کلاس، متدها یا فیلدها استفاده شود به این معنی خواهد بود که آن کلاس، متدها یا فیلد در نسخه های آینده از جاوا حذف خواهد شد. اگر در برنامه تان کلاس، فیلد، یا متدها که با `@Deprecated` نشانگذاری شده است را فراخوانی کنید، کامپایلر در زمان کامپایل برنامه یک پیغام اخطار به شما می دهد. اگر `annotation` فقط یک عنصر داشته باشد بهتر است نام آن عنصر `value` باشد، زیرا همانطور که در بخش بعدی این فصل خواهید دید در هنگام استفاده از آن `annotation` می توانید نام عنصر را مشخص نکنید.

چند نکته در تعریف Annotation

اگرچه `annotation`ها نوع خاصی از اینترفیس محسوب می شوند اما نمی توانید سلسه مراتبی از `annotation`ها را داشته باشید. به عبارت بهتر، یک `annotation` نمی تواند از یک `annotation` دیگر مشتق شود زیرا به صورت پیش فرض هر `annotation` از `java.lang.Annotation` مشتق شده است.

یک `Annotation` نمی تواند یک فرم `generic` داشته باشد. یک `annotation` نمی تواند به صورت مستقیم یا غیرمستقیم عنصری از جنس خودش داشته باشد. اگرچه تمام `annotation`ها به صورت تلویحی از `java.lang.Annotation` مشتق می شوند اما خود یک اینترفیس `annotation` نیست بلکه یک اینترفیس است. به همین دلیل، اگر یک اینترفیس از `java.lang.Annotation` مشتق شود، آن اینترفیس یک `annotation` نخواهد بود بلکه یک اینترفیس است. از آنجاییکه `annotation` نوعی اینترفیس است همانند اینترفیس می توان آنرا پیاده سازی نمود اگرچه اینکار بی معنی است!

عناصر Annotation

یک annotation را بر هر جزئی از برنامه شامل انواع داده ها (کلاس، اینترفیس، enum، annotation) فیلدها، متدها، سازنده کلاسها، متغیرهای محلی، و حتی پارامترهای متدها و حتی یک پکیج اعمال کنید. همانطور که قبلاً دیدید برای اعمال کردن یک annotation بر هر جزئی از برنامه، لازم است نام annotation که قبل از آن @ قرار داده شده است را بیاورید و داخل پرانتز مقدار عناصر annotation را مشخص کنید. اگر annotation عنصري نداشته باشد (نشانگر باشد)، یا تمام عناصر آن مقدار پیش فرض داشته باشند می توانید به شکل خیلی ساده فقط نام آنرا که قبل از آن @ قرار گرفته است بیاورید.

```
@Deprecated
public void badMethod() { /* ... */ }
```

یا آنرا با یک پرانتز خالی به صورت زیر بنویسید

```
@Deprecated()
public void badMethod() { /* ... */ }
```

در غیر اینصورت باید برای هر عنصری که مقدار پیش فرض ندارد، مقداری را با فرمت value=name مشخص کنید. ترتیب عناصر اهمیتی ندارد اما هر عنصر فقط می تواند یکبار ظاهر شود اگر یک عنصر مقدار پیش فرض دارد الزامی برای مشخص کردن مقدار برای آن ندارید ولی اگر می خواهید مقدار پیش فرض را با مقدار دیگری جایگزین کنید می توانید آنرا بیاورید.

اگر یک عنصر، یک آرایه باشد در اینصورت باید به شکل آرایه به آن مقدار بدهید. به عنوان مثال، فرض کنید یک annotation پیاده سازی کرده اید که مشخص کننده باگهای رفع شده کلاسها برای برنامه تان است. این annotation شکلی به صورت زیر خواهد داشت:

```
@interface BugsFixed {
    String[] bugIDs();
}
```

هدف این است که هرگاه باگ یک کلاس را رفع می کنید شناسه باگ رفع شده را به انتهای آرایه bugIDs اضافه می کنید. مثلاً برای یک کلاس نمونه این annotation به صورت زیر قابل استفاده است.

```
@BugsFixed(bugIDs = { "457605", "532456" })
class Foo { /* ... */ }
```

اگر یک آرایه فقط یک عنصر داشته باشد می توانید از نوشتن { } و } صرفنظر کنید و به صورت خلاصه زیر بنویسید.

Annotation

```
@BugsFixed(bugIDs = "457605")
```

اگر یک annotation همانند `BugsFixed` فقط یک عنصر آرایه ای داشته باشد می توانید نام آن عنصر `value` بنامید در اینصورت مقداردهی آن عنصر خیلی خلاصه تر خواهد شد.

```
@interface BugsFixed {  
    String[] value();  
}
```

اینک برای اعمال نمودن `BugsFixed` می توان آنرا به صورت خلاصه تر زیر نوشت

```
@BugsFixed({ "457605", "532456" })
```

اگر فقط یک باگ رفع شده باشد، شکل آن از این هم خلاصه تر خواهد شد

```
@BugsFixed("457605")
```

اگر حتی بیش از یک عنصر داشته باشید هنوز می توانید یک عنصر را `value` نامگذاری کنید اگر بقیه عناصر مقدار پیش فرض داشته باشند می توانید عنصر `value` را بدون اینکه نامی از `value` ببرید به صورت خلاصه فوق مقداردهی کنید اما اگر بخواهید علاوه بر `value` برای عنصر دیگری نیز مقداری را مشخص کنید باید برای مقداردهی `value` حتماً نام `value` را نیز مشخص کنید.
یک `annotation` را می توانید با خودش علامتگذاری کنید. مثلاً فرض کنید یک `annotation` به نام `Documented` داریم که یک نشانگر است و روی هر قسمتی از برنامه که اعمال شود به معنی این است که آن قسمت برنامه دارای مستنداتی است که باید پردازش شوند. طبیعی است که این نشانگر روی خودش نیز می توان اعمال نمود.

```
@Documented  
@interface Documented { }
```

دقت کنید که همانطور که قبل اگفته بود `annotation` نمی تواند عنصری از جنس خودش داشته باشد.

محدود کردن محل اعمال annotation

یک `Annotation` را روی هر قسمتی از برنامه شامل کلاسها، متدها، فیلددها، و ... می توان اعمال نمود طبیعی است که برخی مواقع بخواهیم یک `annotation` فقط روی یک قسمت، مثلاً فقط روی کلاس یا فقط روی متدها، اعمال شود. مثلاً `ClassInfo` که در قسمت ابتدایی این فصل مطرح شد فقط روی

کلاس اعمال شود. برای این منظور می‌توان از `@Target` برای محدود کردن محل اعمال یک annotation استفاده نمود.

خود یک annotation است که برای علامتگذاری روی annotationها طراحی شده است. این annotation روی یک annotation اعمال می‌شود و مشخص می‌کند آن annotation را می‌تواند روی چه قسمتی از برنامه اعمال نمود. فقط یک عنصر دارد که آن هم آرایه‌ای از ElementType است. Target که یک enum است حاوی مقادیری از قبیل CONSTRUCTOR (سازنده کلاس)، FIELD (متغیر محلی)، LOCAL_VARIABLE (متغیر محلی)، METHOD (پارامتر متده)، PACKAGE (پکیج) و TYPE (یک کلاس، اینترفیس یا enum) است. مثلاً برای اینکه بگویید فقط روی کلاس، اینترفیس یا enum اعمال شود باید آنرا به شکل ClassInfo زیر تعریف کنید

```
@Target(ElementType.TYPE)
@interface ClassInfo {
    String created();
    String createdBy();
    String lastModified();
    String lastModifiedBy();
    Revision revision();
}
```

حال اگر ClassInfo را روی هر قسمتی از برنامه به جز کلاس، اینترفیس، enum یا annotation اعمال کنید با خطای کامپایل مواجه خواهد شد. طبیعی است که شما می‌توانید بیش از یک محل را به عنوان محل هایی که یک annotation می‌تواند بر آنها اعمال شود را مشخص کنید مثلاً با استفاده از جمله زیر

```
@Target({ ElementType.FIELD, ElementType.LOCAL_VARIABLE })
```

مشخص می‌کنید که annotation فقط روی فیلدها و متغیرهای محلی در کلاس اعمال شود. نکته‌ای که باید به آن توجه کنید اینکه یک annotation همانند کلاس و اینترفیس ممکن است protected یا private یا حتی public تعريف شود این کلمات محدودیت دسترسی به آن annotation را تعريف می‌کنند اما محدودیتی روی اینکه روی چه قسمتی از برنامه اعمال شوند ایجاد نمی‌کنند. @Target مشخص می‌کند که جایی که دسترسی به یک annotation وجود دارد آن annotation را روی چه قسمتی از یک برنامه می‌توان اعمال نمود.

Annotation

نکته دیگری که باید به آن توجه کنید این است که اگر یک annotation خود عنصری از یک annotation دیگر باشد و شما محدودیتی را روی عنصر annotation اعمال نمودید این محدودیت تاثیری روی annotation اصلی ندارد. به عنوان نمونه فرض کنید که اعمال Revision را روی فیلد محدود کرده اید این موضوع تاثیری روی اعمال ClassInfo ClassInfo را کلاس ندارد. همچنان می تواند فیلدی از Revision داشته باشد در حالیکه اعمال Revision روی فیلد محدود شده، اعمال ClassInfo روی کلاس محدود گردیده است.

سیاست های ماندگاری

در هر پروژه ممکن است بسته به نیاز Annotation های مختلفی پیاده سازی کنید که هریک برای نیاز خاصی طراحی شده باشد. یک Annotation ممکن است همانند آنچه در ClassInfo دیدید فقط جنبه توضیح داشته باشد و فقط برای برنامه نویسان قابل استفاده باشد. اما annotation دیگر ممکن است علاوه بر برنامه نویس، برای کامپایلر نیز قابل فهم و استفاده باشد همانند نشانگر @Deprecated که کامپایلر با مشاهده آن پیغام اخطار می دهد. برخی ابزارها نیز می توانند بایت کدهای یک برنامه (همان فایلهای class) را پردازش کنند و annotation های برنامه را استخراج و سپس پردازش کنند این ابزارها عموماً برای درک اینکه چگونه برنامه را در محیط عملیاتی قرار دهنده و آنرا اجرا کنند این کار را انجام می دهند. در برخی موارد نیز، یک annotation در زمان اجرا توسط خود برنامه یا محیطی که برنامه در آن در حال اجراست پردازش می شود. نتیجه اینکه هر annotation ممکن است برای مرحله خاصی از تولید طراحی شده باشد این چیزی است که ما به آن «سیاست ماندگاری» می گوییم و شما می توانید از طریق Retention و RetentionPolicy آنرا برای یک annotation مشخص کنید.

مثلا اگر بخواهید مشخص کنید که ClassInfo فقط در سورس برنامه باید وجود داشته باشد می توانید آنرا به شکل زیر پیاده سازی کنید:

```
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.TYPE)
@interface ClassInfo {
    String created();
    String createdBy();
    String lastModified();
    String lastModifiedBy();
    Revision revision();
}
```

خود یک RetentionPolicy است و annotation با مقادیر زیر است:

- **SOURCE annotation**: فقط در سورس برنامه وجود دارد و هنگام کامپایل توسط کامپایلر در بایت کدهای تولید شده آورده نمی شود.
- **CLASS annotation**: در سورس و در بایت کدهای برنامه وجود دارد اما در زمان اجرا در دسترس نیست.
- **RUNTIME annotation**: هم در سورس، هم در بایت کدهای برنامه وجود دارد و در زمان اجرا نیز قابل دسترسی است.

مقدار پیش فرض CLASS است این بدین معناست که اگر برای یک annotation هیچ سیاست ماندگاری را مشخص نکنید آن annotation در سورس برنامه و در بایت کدهای برنامه وجود خواهد داشت اما در زمان اجرا قابل دسترس نخواهد بود.

توجه: بدون توجه به اینکه یک annotation چه سیاست ماندگاری داشته باشد، اگر آن annotation روی متغیرهای محلی اعمال شود فقط در سورس وجود خواهد داشت و در بایت کدهای برنامه نخواهد بود و طبیعتاً در زمان اجرا نیز در دسترس نخواهد بود.

کار کردن با annotation‌ها

Annotation‌ها بسیار مفید و قدرتمند هستند اما در مواردی ممکن است مفید نباشند. مثلاً اگر در برنامه تان خیلی از annotation استفاده کنید، و هر annotation نیز عناصر زیادی داشته باشد در اینصورت کد شما ناخوانا خواهد بود.

مسئله دیگری که در مورد annotation‌ها وجود دارد این است که هر کسی می تواند برای خود annotation تعريف کند این چیزی است که با هدف اصلی طراحی annotation‌ها مغایرت دارد. Annotation‌ها زمانی مفید هستند که مجموعه ای استاندارد و رایجی از آنها وجود داشته باشد تا در کل یک پروژه یا در مجموعه بزرگی از پروژه ها قابل استفاده باشد. بنابراین در اغلب موارد برنامه نویس ها نیازی به پیاده سازی یک annotation ندارند و در واقع نباید هم اینکار را انجام دهند. در صورت نیاز به تعريف یک annotation، برنامه نویس ارشد یک پروژه مسئولیت تعريف و پیاده سازی آنرا دارد. در حال حاضر در جاوا مجموعه کوچکی از annotation‌ها وجود دارد که در پایین به برخی از آنها اشاره شده است:

و @Retention که در این فصل با آنها آشنا شدید. به اینها meta-annotation گفته می شود زیرا فقط در تعريف annotation‌های دیگر کاربرد دارند. @Documented و @Deprecated که با آنها آشنا شدید.

Annotation

یک **meta-annotation** است یعنی روی **@Retention** و **@Target** همانند **@Inherited** های دیگر اعمال می شود. برای درک کاربرد آن فرض کنید دو کلاس **annotation** **Parent** و **Child** به صورت زیر تعریف شده اند:

```
@SomeAnnotation  
class Parent{  
}  
  
class Child extends Parent{  
}
```

در این مثال، روی کلاس **Parent** اعمال شده است موضوع جالب اینکه اگرچه از **Parent** مشتق شده است اما **@SomeAnnotation** را به ارث نمی برد. به صورت پیش فرض هیچ **annotation** ای به ارث برده نمی شود. مگر اینکه از **@Inherited** قبل از تعریف استفاده شده باشد. مثلاً اگر **@SomeAnnotation** به صورت زیر تعریف شده باشد در اینصورت به ارث برده می شود.

```
@Inherited  
@interface SomeAnnotation{  
}
```

روی یک متاداده اعمال می شود و مشخص می کند که آن متاداده، متاداده پدر را **override** کرده است. درصورتیکه در کلاس پدر چنین متاداده وجود نداشته باشد کامپایلر خطای کامپایل می دهد. این مطلب مانع از اشتباه رایج برنامه نویسان در **override** کردن متدهای یک کلاس می شود.

به کامپایلر می گوید که از دادن برخی اخطارهای کامپایل صرف نظر کند. این **@SuppressWarnings** یک عنصر دارد که آرایه ای از **String** است. این آرایه مشخص کننده نام اخطارهای کامپایل است که باید صرفنظر شوند.

خلاصه

روشی برای مستندسازی کدهای برنامه یا افزودن توضیحات به برنامه است. برخلاف توضیحات سنتی که با استفاده از علامتهای `//` و `/* */` لابلای کدهای برنامه نوشته می‌شوند، `Annotation`‌ها کدهای جاوا هستند که از قواعد برنامه نویسی جاوا پیروی می‌کنند و کامپایلر جاوا صحت استفاده از آنها را بررسی می‌کند. همه چیز با تعریف یک `Annotation` آغاز می‌شود که مشابه یک اینترفیس تعریف می‌شود ولی به جای کلمه کلیدی `interface` با کلمه کلیدی `@interface` معرفی می‌شود. مشابه اینترفیسها، متدهای `Annotation`‌ها بدنه ندارد و همگی به صورت `public` تعریف می‌شوند اما برخلاف اینترفیسها، متدهای `Annotation`‌ها پارامتر ورودی ندارند و حتماً مقدار برگشتی داده‌های اولیه، `String`, `Class`, `enum`, `Annotation` یا آرایه‌ای از آنها را دارد.

به متدهای `Annotation` فیلهای `Annotation` نیز گفته می‌شود. وقتی یک `Annotation` تعریف شد (مشابه یک کلاس) باید یک جایی از برنامه از آن استفاده شود (مشابه ایجاد یک آبجکت از روی کلاس). `Annotation`‌ها می‌توانند قبل از تعریف کلاس یا اینترفیس، قبل از تعریف فیلدها، متدها یا سازنده‌های یک کلاس، پارامترهای متدها، یا متغیرهای برنامه استفاده شوند تا توضیحاتی را به آن جزء کلاس یا اینترفیس اضافه کنند. جایی که یک `Annotation` استفاده می‌شود می‌بایست مقادیر تمام فیلدهای `Annotation` مشخص شود مگر اینکه در تعریف `Annotation` یک مقدار پیش فرض برای آن فیلد مشخص شده باشد.

ارث بری بین `Annotation`‌ها وجود ندارد، اما با آنها مشابه یک اینترفیس می‌توان برخورد کرد یعنی یک اینترفیس می‌تواند آنها را توسعه دهد یا یک کلاس آنها را پیاده سازی کند.

تمرینات

۱) هریک از کلاس‌های برنامه که نوشته می‌شوند کارکرد مشخص و متمایزی دارند و به یک گروه یا دسته خاص تعلق دارند. مثلاً یک کلاس ممکن است

- واسط کاربری باشد

مسئولیت انجام سرویس، انجام تراکنش و کنترل امنیت به عهده داشته باشد،
کار ذخیره و بازیابی داده‌های برنامه را به عهده داشته باشد

- بخشی از سیستم گزارشگیری باشد

یک کلاس عمومی باشد که توسط کلاس‌های دیگر استفاده می‌شود.

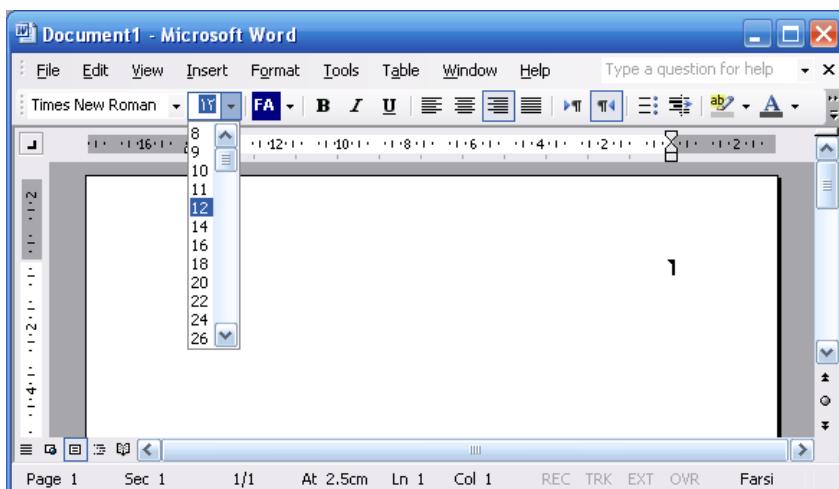
با این توضیح، یک Annotation پیاده سازی کنید که دو فیلد type و description داشته باشد و بتوانیم آنرا روی یک کلاس عمل کنیم.

۲) یک Annotation پیاده سازی کنید که جزئیات یک متده را توضیح دهد. مثلاً اینکه، چه نامی دارد، چه پارامترها یا مقدار برگشتی دارد، چه exception‌هایی ممکن است تولید کند.

۲۱

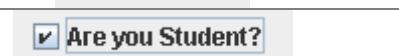
واسط کاربری

از فصلهای پیش به خاطر دارید که می توان با استفاده از کلاس `JOptionPane`، انواع دیالوگ ها را ایجاد کرد و نمایش داد در این فصل به دیگر کلاسهایی که برای ساخت رابط گرافیکی بکارمی روند خواهیم پرداخت. شکل زیر واسط کاربری MS-Word را نشان می دهد که از یک پنجره و مجموعه ای از کامپوننت ها از قبیل یک نوار منو (`File, Edit, View, ...`)، یک نوار ابزارها که زیر آن قرار دارد تشکیل شده است.



شکل ۱-۲۱. واسط کاربری MS-Word

برای پیاده سازی و نمایش هر یک از این اجزا، داخل پنجره، کلاس‌هایی در پکیج `javax.swing` پیاده سازی شده‌اند. جدول زیر برخی از این کلاس‌ها را نشان می‌دهد.

	Name
<code>javax.swing.JButton</code>	
<code>javax.swing.JRadioButton</code>	
<code>javax.swing.JTextField</code>	
<code>javax.swing.JTextArea</code>	
<code>javax.swing.JComboBox</code>	
<code>javax.swing.JCheckbox</code>	

جدول ۲۱-۱. کامپوننت های Swing

کلاس‌هایی که در جدول فوق مشاهده می‌کنید و تمام کلاس‌های دیگری که بخشی از واسط کاربری Swing محسوب می‌شوند مبتنی بر استانداردها جاوا و براساس شعار «یکبار بنویسید همه جا اجرا کنید» پیاده سازی شده‌اند و بنابراین مستقل از سخت افزار یا نرم افزار هستند. علاوه بر Swing، نوع دیگری از واسط کاربری توسط جاوا ارایه می‌شود که به آن AWT گفته می‌شود که به قابلیت‌های نمایشی platform وابسته هستند و بنابراین مستقل از سیستم عامل نیست. درصورتیکه از AWT برای ساخت UI استفاده شود، ممکن است UI مورد نظر در platform های مختلف به صورت متفاوتی نمایش داده شوند.

توجه: وقتی از واسط کاربری Swing استفاده می‌کنید، در مواردی ممکن است از کلاس‌های AWT ناقص عملکرد Swing نیستند نیز استفاده کنید. ولی در استفاده از AWT هیچگاه با کلاس‌های Swing مواجه نمی‌شوید.

واسط کاربری

در Swing، به نحوه نمایش واسط کاربری و نحوه ارتباط آن با کاربر، اصطلاحا گفته می شود. Look & Feel چیزی شبیه تم در واسط کاربری است که امکان می دهد یک واسط کاربری را هر بار با رنگ و لعاب متفاوتی نمایش داد. اما در واقع Look & Feel امکانات و قابلیتهايی فراتر از تم در بقیه واسطهای کاربری دارد. مثلاً نحوه نمایش هر کامپوننت (هر جزو واسط کاربری)، خطوط مرزی،... توسط Look & Feel مشخص می شود. در هر صورت Look & Feel از قدرتهای Swing محسوب می شود زیرا قسمت زیادی از کار طراحی واسط کاربری را که معمولاً تخصص برنامه نویسها نیست از دوش آنها برداشته تا برنامه نویسان صرفاً به برنامه نویسی و منطق برنامه خود فکر کند.

نکته آخر اینکه، از آنجاییکه کلاسهای swing تماماً به زبان جاوا نوشته شده اند به آنها lightweight می شود در مقابل به کلاسهای AWT که از قابلیتهاي پیچیده Platform استفاده می کنند گفته می شود.

JFrame

با استفاده از این کلاس می توانید یک پنجره بسازید

```
•  
•  
•  
JFrame frame = new JFrame("Frame's Title");  
frame.setSize(200, 300);  
frame.setVisible(true);  
•  
•  
•
```

جمله

```
JFrame frame = new JFrame("This is empty frame");
```

یک پنجره می سازد که عنوان آن "This is empty frame" است جمله

```
frame.setSize(200, 300);
```

عرض و این پنجره را با مقادیر ۲۰۰ و ۳۰۰ مشخص می کند از متدها setSize() برای تعیین ابعاد پنجره قابل نمایش استفاده می شود و دو عدد int که به عنوان پارامتر به این متدها ارسال می شوند به ترتیب مشخص

کننده عرض و ارتفاع این پنجره در مقیاس پیکسل می باشند (واحد نمایش هر شی در صفحه نمایش، پیکسل^۱ است). جمله

```
frame.setVisible(true);
```

باعث نمایش پنجره در صفحه نمایش می شود. نتیجه اجرای جملات فوق به صورت زیر خواهد بود:

متد **setLocation()**

همانطور که ملاحظه می کنید اجرای برنامه فوق پنجره ای را در گوشه بالا و سمت چپ صفحه نمایش، نمایش می دهد با استفاده از متدها **setLocation()** می توانید موقعیت نمایش پنجره در صفحه نمایش را مشخص کنید. شکل این متدها به صورت

```
setLocation(int x, int y)
```

X و Y مشخص کننده موقعیت گوشه سمت چپ و بالای پنجره است که در صفحه نمایش نشان داده می شود. هر پنجره به صورت پیش فرض در نقطه (0, 0) نمایش می یابد، به عبارت دیگر در صورتیکه از این متدها تعیین موقعیت پنجره در صفحه نمایش استفاده نکنید، به صورت پیش فرض، پنجره در گوشه بالای صفحه نمایش نشان داده خواهد شد.

متد **setBounds()**

کلاس **JFrame** همچنین دارای متدهای دیگر **setBounds()** است که از آن می توانید برای مشخص کردن همزمان اندازه پنجره و موقعیت نمایش آن استفاده کنید.

```
setBounds(int x, int y, int width, int height)
```

X و Y مشخص کننده موقعیتی از صفحه نمایش است که پنجره در آن موقعیت نمایش داده می شود (مقادیری که می توانید با استفاده از متدهای **setLocation()** آنها را تعیین کنید) و width و height نیز عرض و ارتفاع پنجره را در صفحه نمایش مشخص می کنند (مقادیری که می توانید از طریق متدهای **setSize()** آنها را مشخص کنید).

متد **setTitle()**

عنوان کلمه ای است که روی نوار بالایی پنجره نمایش داده می شود

```
JFrame frame = new JFrame();
frame.setTitle("Window Demo");
```



شکل ۲۱-۲. یک پنجره ساده در Swing

عنوان را می توان در هنگام ایجاد آبجکت از JFrame نیز مشخص نمود:

```
JFrame frame = new JFrame("Window Demo");
```

متده **getContainer()**

پنجره ای که در بالا نمایش داده شد یک پنجره خالی بود، به عبارت دیگر هیچ جزء نمایشی در پنجره وجود نداشت اما برای اینکه بتوانید اجزای نمایشی مختلف را درون پنجره نمایش دهید باید آنها را به JFrame اضافه کنید. در نسخه های قبلی JDK (نسخه ۱,۳ و ۱,۴) امکان اضافه کردن مستقیم اجزا به swing وجود نداشت طراحان swing از کلاسی به نام Container برای تسهیل نمایش اجزا استفاده می کردند. در واقع Container همانند ظرفی می ماند که حاوی تمام اجزای نمایشی یک پنجره است و هر پنجره برای نمایش اجزا و نحوه چیدمان اجزا از آن ظرف استفاده می کند. Container مربوط به هر پنجره باید از آبجکت همان پنجره بصورت زیر بدست آورد:

```
JFrame frame = new JFrame();
Container c = frame.getContentPane();
```

متده **getContentPane()** که در کلاس JFrame قرار دارد برای بدست آوردن Container مربوط به همان JFrame استفاده می شود.

Layout، چیدمان نمایش اجزا

برای مشخص کردن نحوه نمایش و چیدمان اجزای نمایشی در پنجره از اینترفیسی به نام LayoutManager استفاده می شود. در واقع چیدمان اجزا در پنجره (محل قرار گرفتن هر جزء و نحوه چیدمان آنها) توسط LayoutManager مشخص می شود. LayoutManager ساده ترین FlowLayout ترتیبی که به شمار می رود در این چیدمان تمام اجزا به همان Container اضافه می شوند با استفاده از این LayoutManager تمام اجزا به ترتیب از گوشه بالای سمت چپ در یک خط افقی چیده می شوند در صورتیکه برای نمایش یک جزء جای کافی وجود نداشته باشد، آن جزء در خط بعد نمایش داده خواهد شد. کدهای زیر نحوه مشخص کردن LayoutManager را در یک پنجره نشان می دهد.

```

JFrame frame = new JFrame();
Container c = frame.getContentPane();
FlowLayout layout = new FlowLayout();
c.setLayout(layout);
frame.setSize(200, 300);
frame.setVisible(true);
.
.
.
```

توجه: کلاس `JFrame` دارای یک متده به نام `setLayout()` است که نباید از آن استفاده کنید.

حال که نحوه چیدمان اجزای پنجره مشخص شد می توان اجزای دلخواه را به آن افزود.

JLabel

با استفاده از برچسب یا `label` می توان کلمه یا جمله ای را در UI نمایش داد (برای درک دقیق آن به شکلی که در ابتدای فصل آمده توجه کنید) برای ساختن برچسب باید آبجکتی از کلاس `JLabel` بسازید و آنرا به `Container` اضافه کنید.

```

JFrame frame = new JFrame();
Container c = frame.getContentPane();
FlowLayout layout = new FlowLayout();
c.setLayout(layout);
JLabel welcomeLbl = new JLabel("Welcome Label");
c.add(welcomeLbl);
frame.setSize(200, 300);
frame.setVisible(true);
```

از کلاس `JLabel` برای نمایش آیکن نیز استفاده می شود، برای نمایش آیکن، باید ابتدا از کلاس `ImageIcon` یک آبجکت بسازید و سپس از روی آن یک برچسب بسازید:

```

JFrame frame = new JFrame();
Container c = frame.getContentPane();
FlowLayout layout = new FlowLayout();
c.setLayout(layout);
ImageIcon welcomeIcon = new ImageIcon("welcome.gif");
JLabel welcomeLbl = new JLabel(welcomeIcon);
c.add(welcomeLbl);
frame.setSize(200, 300);
```

واسط کاربری

```
frame.setVisible(true);
```

همچنان که ملاحظه می کنید برای ساختن یک آبجکت از ImageIcon باید یک رشته که معرف آدرس یک عکس است را نیز به سازنده آن ارسال کنید در مثال فوق، رشتة welcome.gif معرف آدرس یک فایل است. به این شیوه آدرس دهی یک فایل آدرس دهی نسبت ۲ گفته می شود که مسیر فایل نسبت به محل قرار گرفتن فایلهای class. سنجیده می شود در مقابل روش دیگری برای مشخص کردن مسیر یک فایل وجود دارد که به آن آدرس دهی مطلق (absolute) گفته می شود، در این شیوه آدرس کامل یک فایل مشخص می شود مثلا در:

```
ImageIcon welcomeIcon =  
    new ImageIcon("C:/java-programs/images/welcome.gif");
```

آدرس فایل welcome.gif به صورت مطلق مشخص شده است.
کلاس JLabel دارای سازنده های دیگری نیز هست که عبارتند از:

```
JLabel(Icon icon)  
JLabel(String lbl)  
JLabel(String lbl, Icon icon, int align)
```

JTextField

با استفاده از این کلاس می توانید یک رشته را در یک خط نوشته یا ویرایش کنید. جدول ۲۱-۲ سازنده های JTextField را نشان می دهد.

JTextField()	یک Text Field خالی می سازد
JTextField(int cols)	یک Text Field خالی می سازد، cols مشخص کننده تعداد کارکترهای پیش فرض است.
JTextField(String s, int cols)	یک Text Field با طول پیش فرض نمایش می دهد که در آن رشته s به صورت پیش فرض نوشته شده است
JTextField(String s)	یک Text Field که به صورت پیش فرض رشتة s در آن نوشته شده است می سازد

جدول ۲۱-۲. سازنده های کلاس JTextField

به مثال زیر توجه کنید.

```
JFrame frame = new JFrame();  
Container c = frame.getContentPane();  
FlowLayout layout = new FlowLayout();
```

```
c.setLayout(layout);
JTextField textField = new JTextField(20);
c.add(textField);
frame.setSize(200, 300);
frame.setVisible(true);
```

دو متد مهم این کلاس عبارتند از

```
String getText()
void setText(String text)
```

با استفاده از متد () `getText()` می توان متنی که توسط کاربر درون `TextField` تایپ شده است را بدست آورد و با استفاده از متد () `setText()` می توان متن خاصی را درون `TextField` وارد نمود.

JButton

از این کلاس برای نمایش یک Push Button (دکمه) در یک پنجره استفاده می شود سازنده های کلاس `JButton` در جدول ۲۱-۳ نشان داده شده اند.

<code>JButton()</code>	یک Push Button خالی می سازد روی این دکمه هیچ نوشته ای وجود نخواهد داشت
<code>JButton(String s)</code>	یک Push Button که روی آن رشتہ <code>s</code> نوشته شده است می سازد
<code>JButton(Icon icon)</code>	یک Push Button که روی آن یک آیکن قرار دارد می سازد برای ساختن این آیکن می توان به همان روشی که برای <code>JLabel</code> عمل شد، عمل نمود.
<code>JButton(Action action)</code>	کلاسی است که توسط آن می توانید مشخص کنید در صورت کلیک شدن Push Button چه عملی انجام شود.

جدول ۲۱-۳. سازنده های کلاس JButton

به مثال زیر توجه کنید.

```
JFrame frame = new JFrame();
Container c = frame.getContentPane();
FlowLayout layout = new FlowLayout();
c.setLayout(layout);
JButton pressButton = new JButton("Press");
c.add(pressButton);
frame.setSize(200, 300);
frame.setVisible(true);
```

نتیجه اجرای تکه کد فوق به صورت زیر خواهد بود.



شکل ۲۱-۳. نمایش JButton

در صورتیکه بخواهید با فشرده شدن Push Button عمل خاصی انجام شود لازم است برای آن Listener مشخص کنید، Listener آبجکتی است که در ابتدا به Push Button معرفی می شود، سپس هرگاه برای Push Button حادثه ای رخ دهد(مثلا کلیک کردن بر روی Push Button)، Push Button ای که قبلا به او مطلع می کند، Listener نیز عمل خاصی را انجام می دهد. برای تعریف کردن یک Listener برای JButton باید کلاسی داشته باشد که اینترفیس ActionListener را پیاده سازی کرده باشد این اینترفیس دارای یک متده است:

```
public void actionPerformed(ActionEvent e);
```

که باید آنرا پیاده سازی کنید با رخداد حادثه برای Push Button این متده فراخوانی خواهد شد بنابراین عملیاتی که مایلید با کلیک شدن Button انجام شود را در این متده پیاده سازی کنید.

```

1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ActionEvent;
7
8 public class ButtonDemo {
9
10    public ButtonDemo() {
11        JFrame frame = new JFrame();
12        Container c = frame.getContentPane();
13        FlowLayout layout = new FlowLayout();
14        c.setLayout(layout);
15        JButton pressButton = new JButton("Press");
16        ActionListener listener =

```

```

17         new ButtonListener();
18     pressButton.addActionListener(listener);
19     c.add(pressButton);
20     frame.setSize(200, 300);
21     frame.setVisible(true);
22 }
23
24 public static void main(String[] args) {
25     new ButtonDemo();
26 }
27
28 class ButtonListener implements ActionListener {
29     public void actionPerformed(ActionEvent e) {
30         System.out.println(
31             "Press Button Clicked!!!!");
32     }
33 }
34 }
```

کد ۱-۱

همچنان که ملاحظه می کنید کلاسی به نام `ButtonListener` نوشته شده است که اینترفیس `ActionListener` را پیاده سازی نموده است بالطبع متده است `actionPerformed()` را نیز که در این اینترفیس تعریف شده است پیاده سازی کرده است در این مثال تنها جمله زیر در بدنه این متده نوشته شده است

```
System.out.println("Press Button Clicked!!!!");
```

که فقط جمله `Press Button Clicked!!!!` را در خروجی استاندارد چاپ می کند.
در هنگام ساختن آبجکت از `JButton` به صورت زیر:

```
JButton pressButton = new JButton("Press");
ActionListener listener = new ButtonListener();
pressButton.addActionListener(listener);
```

ابتدا آبجکتی از `JButton` ساخته شده، سپس آبجکتی از `ButtonListener` ساخته شده و با استفاده از متده است `addActionListener()` به آبجکت `JButton` معرفی شده است. به این ترتیب با کلیک کردن بر روی `Push Button` تعریف شده است فراخوانی شده و جمله `Press Button Clicked!!!!` در خروجی استاندارد چاپ می شود.

واسط کاربری

حال می خواهیم برنامه فوق را طوری تغییر دهیم که به هر بار کلیک شدن Push Button، نوشته روی آن تغییر کند و مشخص کند که Push Button چند بار کلید شده است. در اینصورت برنامه فوق به صورت زیر تغییر خواهد کرد

```
1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ActionEvent;
7
8 public class ButtonDemo1 {
9
10    JButton pressButton = new JButton("Press");
11    int clickNumber = 0;
12
13    public ButtonDemo1() {
14        JFrame frame = new JFrame();
15        Container c = frame.getContentPane();
16        FlowLayout layout = new FlowLayout();
17        c.setLayout(layout);
18        ActionListener listener =
19            new ButtonListener();
20        pressButton.addActionListener(listener);
21        c.add(pressButton);
22        frame.setSize(200, 300);
23        frame.setVisible(true);
24    }
25
26    public static void main(String[] args) {
27        new ButtonDemo1();
28    }
29
30    class ButtonListener implements ActionListener {
31        public void actionPerformed(ActionEvent e) {
32            clickNumber++;
33            pressButton.setText("I've been "+
34                "clicked "+ clickNumber+"times!!!!");
35        }
36    }
37}
```

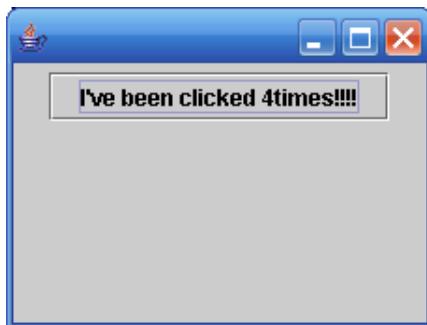
```

36      }
37
38  }

```

۲۱-۳

همانطور که ملاحظه می کنید آبجکت JButton بیرون از سازنده کلاس تعریف شده است تا کلاس ButtonListener به آن دسترسی داشته باشد یک متغیر int، به نام clickNumber هم در بالای کلاس تعریف شده که مشخص کننده تعداد دفعاتی است که Push Button کلیک شده است. در متده است `actionPerformed()`، که با هر بار کلیک شدن Push Button فراخوانی می شود، یک واحد افزوده شده است و سپس برچسب روی Push Button با فراخوانی متده است `setText()` با مقدار جدید جایگزین شده است. اجرای برنامه فوق به صورت زیر خواهد بود:



شکل ۲۱-۴. نتیجه اجرای ButtonDemo

تمرین: برنامه فوق را بگونه ای تغییر دهید که با هر بار کلیک شدن Push Button، یک دیالوگ باز شود (این کار را می توانید با بکارگیری کلاس JOptionPane انجام دهید)

به عنوان مثال می خواهیم برنامه ای بنویسیم که کار تبدیل درجه حرارت سلسیوس به فارنهایت را انجام دهد. این کار را با استفاده از سه کلاس JTextField، JButton و JLabel و JTextField انجام خواهیم داد. لازم است در یک پنجره، یک Text Field داشته باشیم تا با استفاده از آن عدد درجه حرارت سلسیوس از کاربر دریافت شود، سپس به یک Push Button نیاز داریم تا با کلیک شدن روی آن کار تبدیل درجه حرارت از سلسیوس به فارنهایت انجام شود کار محاسبه و تبدیل درجه حرارت باید در متده است `actionPerformed()` انجام شود و در نهایت می توان با استفاده از یک JLabel نتیجه محاسبه شده را نمایش داد این برنامه به صورت زیر خواهد بود.

```
1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ActionEvent;
7
8 public class CelsiusConverter {
9
10    JButton pressButton = new JButton("Convert");
11    JTextField input = new JTextField(5);
12    JLabel label = new JLabel();
13    int clickNumber = 0;
14
15    public CelsiusConverter() {
16        JFrame frame = new JFrame();
17        Container c = frame.getContentPane();
18        FlowLayout layout = new FlowLayout();
19        c.setLayout(layout);
20        ActionListener listener =
21            new ButtonListener();
22        pressButton.addActionListener(listener);
23        c.add(input);
24        c.add(pressButton);
25        c.add(label);
26        frame.setSize(300, 75);
27        frame.setVisible(true);
28    }
29
30    public static void main(String[] args) {
31        new CelsiusConverter();
32    }
33
34    class ButtonListener implements ActionListener {
35        public void actionPerformed(ActionEvent e) {
36            String str = input.getText();
37            double celcius =
38                Double.parseDouble(str);
39            int tempFahr =
40                (int) (celcius * 1.8 + 32);
```

```

41         label.setText(tempFahr +
42                         " Fahrenheit");
43     }
44 }
45
46 }
```

کد ۲۱-۳

نتیجه اجرای برنامه فوق به صورت زیر خواهد بود.



شکل ۲۱-۵. نتیجه اجرای CelsiusConverter

JCheckBox

از این کلاس برای ساختن یک Check Box استفاده می شود سازنده های این کلاس عبارتند از:

JCheckBox(String lbl)

یک Check Box با رشتة lbl نمایش می دهد

JCheckBox(String lbl,
boolean selected)

یک Check Box با رشتة lbl نمایش می دهد در صورتیکه
مقدار selected برابر true باشد این Check Box به
صورت انتخاب شده نمایش می یابد.

JCheckBox(String lbl,
Icon icon)

یک Check Box با رشتة lbl نمایش می دهد icon آبجکتی
از اینترفیس Icon است که در کنار این Check Box نشان
داده خواهد شد

JCheckBox(String lbl,
Icon icon,
boolean selected)

lbl مشخص کننده یک رشتة و icon مشخص کننده یک
آیکن است که توسط این Check Box نمایش داده خواهد شد
همچنین در صورتیکه مقدار true ، selected باشد این
به صورت انتخاب شده نمایش می یابد.

جدول ۲۱-۴. سازنده های کلاس JCheckBox**JRadioButton**

با استفاده از این کلاس می توان دکمه های رادیویی ساخته و نمایش داد برخی سازنده های این کلاس
 عبارتند از:

واسط کاربری

JRadioButton(String lbl)	یک Radio Button با رشتہ مورد نظر کہ در کنار آن نوشته شده است می سازد.
JRadioButton(String lbl, boolean selected)	یک Radio Button با رشتہ مورد نظر می سازد در صورتیکہ مقدار پارامتر selected true باشد به صورت انتخاب شده نمایش می یابد.
JRadioButton(Icon icon)	یک Radio Button با یک آیکن در کنار آن نمایش داده می شود.
JRadioButton(String lbl, Icon icon, boolean selected)	یک Radio Button با یک رشتہ به همراه یک آیکن نمایش داده می شود در صورتیکہ مقدار پارامتر selected true باشد به صورت انتخاب شده نمایش می یابد.

جدول ۲۱-۵. سازنده های کلاس JRadioButton

با استفاده از دکمه های رادیویی می توان از بین چندین گزینه، تنها یک گزینه را انتخاب نمود بدین منظور باید با استفاده از کلاس ButtonGroup، دکمه های مرتبط با یکدیگر را مشخص کرد در غیراینصورت هریک از دکمه ها را می توان به صورت جداگانه انتخاب نمود یا از حالت انتخاب خارج کرد برنامه زیر چندین دکمه رادیویی مرتبط با یکدیگر را ایجاد می کند.

```
1 package ir.atlassoft.javase.chapter21;  
2  
3 import javax.swing.*;  
4 import java.awt.*;  
5  
6 public class RadioButtonDemo {  
7  
8     public RadioButtonDemo() {  
9         JFrame frame = new JFrame();  
10        Container c = frame.getContentPane();  
11        c.setLayout(new FlowLayout());  
12  
13        //instantiate RadioButtons  
14        JRadioButton btn1 = new JRadioButton("First");  
15        JRadioButton btn2 = new JRadioButton("Second");  
16        JRadioButton btn3 = new JRadioButton("Third");  
17        JRadioButton btn4 = new JRadioButton("Others");  
18    }
```

```

19         //add RadioButtons to ButtonGroup
20         ButtonGroup group = new ButtonGroup();
21         group.add(btn1);
22         group.add(btn2);
23         group.add(btn3);
24         group.add(btn4);
25
26         //add RadioButtons to container
27         c.add(btn1);
28         c.add(btn2);
29         c.add(btn3);
30         c.add(btn4);
31
32         frame.pack();
33         frame.setVisible(true);
34     }
35
36     public static void main(String[] args) {
37         new RadioButtonDemo();
38     }
39 }
```

۲۱-۴ کد

اجرای برنامه فوق نتیجه‌ای بصورت زیر خواهد داشت.



شکل ۶. ۲۱-۶. نتیجه اجرای RadioButtonDemo

دو متد مهم این کلاس عبارتند از:

```

void setSelected(boolean selected)
boolean isSelected()
```

با استفاده از متد `setSelected()` می‌توان وضعیت انتخاب Radio Button را تغییر داد در صورتیکه مقدار پارامتر `true`، `selected` باشد Radio Button به صورت انتخاب شده در می‌آید و در غیر

واسط کاربری

اینصورت از حالت انتخاب خارج می شود. با استفاده از متده `isSelected()` نیز که یک مقدار boolean بر می گرداند می توان تعیین کرد که Radio Button انتخاب شده است یا انتخاب نشده است.

یک مثال کاربردی

می خواهیم برنامه ای بنویسیم که در آن تعدادی دکمه رادیویی نمایش داده می شود هر دکمه رادیویی مشخص کننده یک شی است و با انتخاب شدن هر کدام از دکمه ها شی متناظر با آن دکمه روی یک برچسب نمایش یابد. بدون هیچ توضیحی مشخص است که به یک Listener نیاز است تا از طریق آن به محض انتخاب یک گزینه، شی مورد نظر روی برچسب نمایش یابد کلاس Listener، کلاسی است که منتظر تغییر وضعیت یا رخداد یک حادثه (کلیک کردن موس,...) می ماند و به محض رخداد حادثه، عملیات مورد نظر شما را انجام می دهد. برای JRadioButton می توانید از دو Listener استفاده کنید `ActionListener` و `ChangeListener`. برنامه زیر، کلاس پیاده سازی شده این مثال را نشان می دهد:

```
1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import javax.swing.event.ChangeListener;
5 import javax.swing.event.ChangeEvent;
6 import java.awt.*;
7 import java.awt.event.ActionListener;
8 import java.awt.event.ActionEvent;
9 import java.beans.PropertyChangeListener;
10 import java.beans.PropertyChangeEvent;
11
12 public class RadioButtonExample implements ActionListener {
13     ImageIcon clockIcon = new ImageIcon("clock.gif");
14     ImageIcon calculatorIcon = new
15             ImageIcon("calculator.gif");
16     ImageIcon flopyIcon = new ImageIcon("flopy.gif");
17     ImageIcon notebookIcon = new
18             ImageIcon("notebook.gif");
19
20     //instantiate RadioButtons
21     JRadioButton clockBtn = new JRadioButton("Clock");
22     JRadioButton calculatorBtn = new
23             JRadioButton("Calculator");
```

```

24     JRadioButton flopyBtn = new JRadioButton("Flopy");
25     JRadioButton notebookBtn = new
26             JRadioButton("Notebook");
27
28     JLabel showLbl = new JLabel("Select One To Show!");
29
30     public RadioButtonExample() {
31         JFrame frame = new JFrame();
32         Container c = frame.getContentPane();
33         c.setLayout(new FlowLayout());
34
35         //add listener to all RadioButtons
36         clockBtn.addActionListener(this);
37         calculatorBtn.addActionListener(this);
38         flopyBtn.addActionListener(this);
39         notebookBtn.addActionListener(this);
40
41         //add RadioButtons to ButtonGroup
42         ButtonGroup group = new ButtonGroup();
43         group.add(clockBtn);
44         group.add(calculatorBtn);
45         group.add(flopyBtn);
46         group.add(notebookBtn);
47
48         //add RadioButtons to container
49         c.add(clockBtn);
50         c.add(calculatorBtn);
51         c.add(flopyBtn);
52         c.add(notebookBtn);
53         c.add(showLbl);
54
55         frame.pack();
56         frame.setVisible(true);
57     }
58
59     public static void main(String[] args) {
60         new RadioButtonExample();
61     }
62
63     public void actionPerformed(ActionEvent e) {

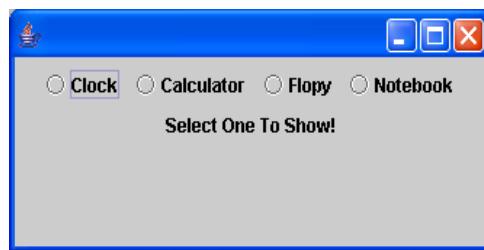
```

```

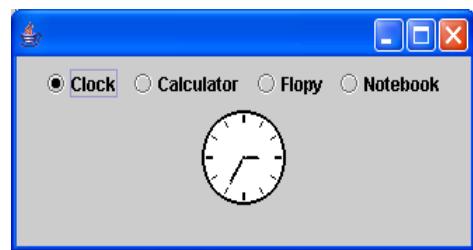
64
65     //remove initial text of label
66     showLbl.setText("");
67
68     JRadioButton source =
69         (JRadioButton) e.getSource();
70     //if clockBtn selected, show
71     //clock icon by label
72     if (source == clockBtn) {
73         showLbl.setIcon(clockIcon);
74
75         //if calculatorBtn selected, show
76         //calculator icon by label
77     } else if (source == calculatorBtn) {
78         showLbl.setIcon(calculatorIcon);
79
80         //if flopyBtn selected, show
81         //flopy icon by label
82     } else if (source == flopyBtn) {
83         showLbl.setIcon(flopyIcon);
84
85         //if notebookBtn selected, show
86         //notebook icon by label
87     } else if (source == notebookBtn) {
88         showLbl.setIcon(notebookIcon);
89
90     }
91 }
92 }
```

۲۱-۵ کد

همانطور که ملاحظه می کنید کلاس فوق اینترفیس ActionListener را پیاده سازی کرده است تا بتوان از همین کلاس به عنوان Listener برای تمام Radio Button ها استفاده نمود، همچنین با استفاده از متد () add ActionListener همین کلاس به عنوان Listener به تمام Radio Button معرفی شده است. در متد () actionPerformed که با رخداد حادثه به صورت خودکار فراخوانی می شود عملیات مورد نظر خود را نوشه ایم. در این متد ابتدا با استفاده از متد () getSource آبجکتی که روی آن حادثه رخداده است را مشخص می کنیم با بررسی اینکه برای کدام Radio Button حادثه رخداده است، آیکن برچسب را تغییر می دهیم. شکل زیر دو قسمت از اجرای این برنامه را نشان می دهد:



شروع برنامه



انتخاب Clock

شکل ۲۱-۷. نتیجه اجرای RadioButtonExample**JList**

از این کلاس برای نمایش لیستی از گزینه‌ها استفاده می‌شود که کاربر از طریق آن می‌تواند یک یا چند گزینه را انتخاب کند. این کلاس دارای سازنده‌های زیر است:

`JList()`

یک آبجکت `JList` خالی ساخته می‌شود.

`JList(Object[] objs)`

از طریق های آرایه `objs`، یک لیست ساخته می‌شود هر کدام از عناصر این آرایه، یک عنصر لیست محسوب می‌شوند.

`JList(Vector v)`

از طریق عناصر موجود در `v`، یک لیست ساخته می‌شود که هر کدام از عناصر `Vector`، یکی از عناصر لیست خواهد بود.

`JList(ListModel model)`

آبجکتی از جنس `ListModel` است که اطلاعات لیست را نگهداری می‌کند. `JList` از روی این آبجکت عناصر خود را نمایش می‌دهد.

جدول ۲۱-۶. سازنده‌های کلاس JList

اشکال زیر دو نمایش مختلف از لیست را نشان می‌دهد:

واسط کاربری



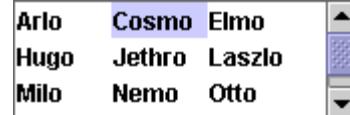
شکل ۲۱-۸. دو نمایش از JList

همانطور که ملاحظه می شود عناصر یکی از لیستها به صورت افقی و دیگری به صورت عمودی نمایش یافته است. همچنین به علت زیاد بودن عناصر یکی از لیست ها و عدم امکان نمایش تمام عناصر در یک پنجره، در کنار آن یک Scroll نمایش یافته است.

برای تعیین نحوه نمایش عناصر یک JList (به صورت عمودی یا افقی) باید یکی از مقادیر JList.HORIZONTAL_WRAP یا JList.VERTICAL_WRAP را انتقاده نمود

این مقادیر را باید با استفاده از متد () setLayoutOrientation JList برای مشخص نمود مثلا تکه کدهای زیر هر کدام لیستهای مختلفی که در مقابل آنها نشان داده شده است را ایجاد می کنند:

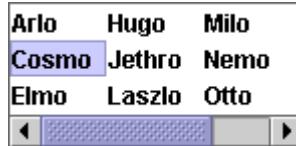
```
JList list = new JList(data)
list.setLayoutOrientation(JList.HORIZONTAL_WRAP);
```



```
JList list = new JList(data)
list.setLayoutOrientation(JList.VERTICAL);
```



```
JList list = new JList(data)
list.setLayoutOrientation(JList.VERTICAL_WRAP);
```



همچنین لیست را می توان به گونه ای ایجاد کرد که کاربر فقط امکان انتخاب یک گزینه را داشته باشد یا اینکه بتواند چندین عنصر را از داخل لیست انتخاب کند.

برای مشخص کردن این مطلب از یکی از مقدادیر ثابت در کلاس `ListSelectionModel` استفاده می شود جدول زیر مقدادیر ثابت این کلاس را که به این منظور بکار می روند به و نحوه نمایش لیست را با استفاده از این مقدادیر نشان می دهد.

SINGLE SELECTION



در این وضعیت فقط یک عنصر را می توان از لیست انتخاب نمود، با انتخاب یک عنصر جدید، عنصر قبلی از حالت انتخاب خارج می شود.

SINGLE_INTERVAL_SELECTION



در این وضعیت، چندین عنصر را می توان انتخاب نمود اما تنها عناصر که کنار یکدیگر هستند را می توان انتخاب کرد. برای انتخاب چندین گزینه باید دکمه `Ctrl` را در زمان انتخاب پایین نگه داشت.

MULTIPLE_INTERVAL_SELECTION



در این وضعیت می توان چندین گزینه را به دلخواه در هر موقعیت انتخاب نمود. همانند قبل، برای انتخاب چندین گزینه باید دکمه `Ctrl` را در زمان انتخاب پایین نگه داشت.

تکه کد زیر نحوه مشخص کردن یکی از مقدادیر فوق را برای لیست نشان میدهد:

```
list = new JList(data);
list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
list.setLayoutOrientation(JList.HORIZONTAL_WRAP);
```

از آنجاییکه ممکن است اندازه قابل نمایش یک `JList` بیشتر از مکانی باشد که در `JFrame` به آن اختصاص داده شده است، لازم است از `ScrollPane` استفاده شود. بدین منظور به جای اینکه یک `JList` را مستقیماً به `Container` اضافه کنند، ابتدا آنرا درون یک `JScrollPane` قرار می دهند و سپس `JScrollPane` را به `Container` اضافه می کنند. تکه برنامه زیر ایجاد یک لیست نوعی را به صورت کامل نشان می دهد.

واسط کاربری

```
1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class JListDemo {
7
8     public JListDemo() {
9         JFrame frame = new JFrame();
10        Container c = frame.getContentPane();
11        String[] elements = {"First", "Second",
12                            "Third", "Others"};
13        JList list = new JList(elements);
14        list.setSelectionMode(
15            ListSelectionModel.SINGLE_INTERVAL_SELECTION);
16        list.setLayoutOrientation(JList.HORIZONTAL_WRAP);
17        JScrollPane listPane = new JScrollPane(list);
18        c.add(listPane);
19
20        frame.setSize(300, 75);
21        frame.setVisible(true);
22    }
23
24    public static void main(String[] args) {
25        new JListDemo();
26    }
27}
28}
```

۲۱-۷

جدول زیر مهمترین متدهای کلاس `JList` را نشان می دهد.

<code>int getSelectedIndex()</code>	موقعیت (ایندکس) عنصری را که انتخاب شده است بر می گرداند در صورتیکه در لیست امکان انتخاب بیش از یک گزینه وجود داشته باشد، موقعیت اولین عنصر انتخاب شده را برگردانده می شود. در صورتیکه هیچ گزینه ای انتخاب نشده باشد -1- برگردانده خواهد شد.
<code>int[] getSelectedIndices()</code>	موقعیت (ایندکس) عناصر انتخاب شده را بر می گرداند. در

صورتیکه در لیست امکان انتخاب بیش از یک عنصر وجود نداشته باشد، تنها موقعیت عنصر انتخاب شده را برمی گرداند. در صورتیکه هیچ گزینه ای انتخاب نشده باشد -1 برگردانده خواهد شد.

`Object getSelectedValue()`

عنصری را که انتخاب شده است بر می گرداند در صورتیکه در لیست امکان انتخاب بیش از یک گزینه وجود داشته باشد، اولین عنصر انتخاب شده را برگردانده می شود. در صورتیکه هیچ گزینه ای انتخاب نشده باشد `null` برگردانده خواهد شد.

`Object[] getSelectedValues()`

عناصر انتخاب شده را برمی گرداند. در صورتیکه در لیست امکان انتخاب بیش از یک عنصر وجود نداشته باشد، تنها عنصر انتخاب شده را برمی گرداند. در صورتیکه هیچ گزینه ای انتخاب نشده باشد `null` برگردانده خواهد شد.

جدول ۱-۱. مهترین متدهای کلاس `List`

JComboBox

یک جزء نمایشی شبیه `List` است اما توسط آن تنها می توان یک گزینه را انتخاب نمود برخی سازنده های این کلاس عبارتند از:

`JComboBox()`

یک `Combo Box` خالی می سازد

`JComboBox(Object[] elements)`

یک `Combo Box` می سازد که عناصر آرایه `elements` گزینه های آن را تشکیل می دهند.

`JComboBox(Vector elements)`

یک `Combo Box` می سازد که عناصر `elements` گزینه های آن را تشکیل می دهند.

`JComboBox(ComboBoxModel model)`

از طریق عناصری که در `Combo Box` تعریف می شوند یک `Combo Box` ساخته می شود.

جدول ۱-۲. سازنده های کلاس `JComboBox`

کلاس `Combox`، ایجاد و نمایش `ComboBoxDemo` را نشان می دهد.

```

1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import java.awt.*;
```

```

5
6 public class ComboBoxDemo {
7
8     public ComboBoxDemo() {
9         JFrame frame = new JFrame();
10        Container c = frame.getContentPane();
11        c.setLayout(new FlowLayout());
12
13        String[] elements = {"First", "Second",
14                            "Third", "Others"};
15        JComboBox list = new JComboBox(elements);
16
17        c.add(list);
18        frame.setSize(300, 75);
19        frame.setVisible(true);
20    }
21
22    public static void main(String[] args) {
23        new ComboBoxDemo();
24    }
25 }
```

۲۱-۷

نتیجه اجرای برنامه فوق به صورت زیر است:



شکل ۲۱-۹. نتیجه اجرای ComboBoxDemo

مهمترین متد های کلاس JComboBox در جدول ۲۱-۱۰ توضیح داده شده اند.

int getSelectedIndex()	موقعیت عنصر انتخاب شده را برمی گرداند.
int getSelectedItem()	عنصر انتخاب شده را برمی گرداند.
void setSelectedIndex(int index)	عنصری را که در موقعیت index قرار دارد در Combo Box انتخاب می کند. در صورتیکه index خارج از محدوده تعداد عناصر Combo

باشد خطای Box

تولید IllegalArgumentException

خواهد شد.

عنصر `obj` را که از عناصر `Combo Box` را عنصر `obj` را که از عناصر `Combo Box` را انتخاب می کند.

جدول ۲۱-۱۰. مهمترین متدهای کلاس JComboBox

در صورتیکه بخواهید با انتخاب هر یک از گزینه های آن عمل خاصی صورت گیرد باید برای آن `Listener` تعریف کنید به این منظور باید از `ItemListener` استفاده کنید این اینترفیس باید توسط کلاسی که می خواهید نسبت به انتخاب گزینه های `Combo Box` حساس باشد باید پیاده سازی شود این اینترفیس دارای یک متد `itemStateChanged` است که با انتخاب گزینه های یک `Combo Box` اجرا خواهد شد. در مثال زیر با انتخاب هر یک از گزینه های `Combo Box` عکسی که توسط یک `Label` نمایش داده می شود تغییر می کند.

```

1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.ItemListener;
6 import java.awt.event.ItemEvent;
7
8 public class ComboBoxDemo1 implements ItemListener {
9
10    ImageIcon clockIcon = new ImageIcon("clock.gif");
11    ImageIcon calculatorIcon = new
12        ImageIcon("calculator.gif");
13    ImageIcon flopyIcon = new ImageIcon("flopy.gif");
14    ImageIcon notebookIcon = new
15        ImageIcon("notebook.gif");
16
17    String[] elements = {"Clock", "Calculator",
18        "Flopy", "Note Book"};
19    JComboBox list = new JComboBox(elements);
20
21    JLabel showLbl = new JLabel(clockIcon);
22
23    public ComboBoxDemo1() {

```

واسط کاربری

```
24         JFrame frame = new JFrame();
25         Container c = frame.getContentPane();
26         c.setLayout(new FlowLayout());
27         list.addItemListener(this);
28         c.add(list);
29         c.add(showLbl);
30
31         frame.pack();
32         frame.setVisible(true);
33     }
34
35     public static void main(String[] args) {
36         new ComboBoxDemo();
37     }
38
39     public void itemStateChanged(ItemEvent e) {
40
41         if (list.getSelectedIndex() == 0) {
42             showLbl.setIcon(clockIcon);
43         } else if (list.getSelectedIndex() == 1) {
44             showLbl.setIcon(calculatorIcon);
45         } else if (list.getSelectedIndex() == 2) {
46             showLbl.setIcon(flopyIcon);
47         } else if (list.getSelectedIndex() == 3) {
48             showLbl.setIcon(notebookIcon);
49         }
50     }
51 }
```

۲۱-۸ کد

همانطور که ملاحظه می کنید با فراخوانی متد `addItemListener()`، کلاس فعلی به عنوان `ComboBox` معرفی شده است این بدین معناست که با انتخاب شدن هریک از گزینه های `Combo Box` به `Listener` که در همین کلاس تعریف شده است فراخوانی خواهد شد.

در متد `itemStateChanged()` با مشخص کردن ایندکس گزینه انتخاب شده، آیکن متناظر آن توسط `Label` نمایش داده می شود. نتیجه اجرای این برنامه به صورت زیر خواهد بود:



شکل ۲۱-۷. نتیجه اجرای ComboBoxDemo

JTextArea

JTextArea شبیه JTextField است اما در آن می توان عبارتی بیش از یک خط نوشته سازنده های این کلاس به صورت زیر هستند:

JTextArea()	یک JTextArea خالی می سازد
JTextArea(String text)	یک JTextArea می سازد که عبارت text در آن نوشته شده است
JTextArea(int rows, int columns)	یک JTextArea با تعداد rows سطر و columns ستون می سازد.
JTextArea(String text, int rows, int columns)	یک JTextArea با تعداد rows سطر و columns ستون می سازد که عبارت text در آن نوشته شده است.

جدول ۲۱-۸. مهمترین متدهای کلاس JTextArea

برنامه زیر ساخت و نمایش یک JFrame را در یک JTextArea نشان می دهد.

```

1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class TextAreaDemo {
7
8     public TextAreaDemo() {
9
10         JFrame frame = new JFrame();
11         Container c = frame.getContentPane();
12
13         JTextArea area = new JTextArea(10, 20);
14         JScrollPane pane = new JScrollPane(area);

```

واسط کاربری

```
15         c.add(pane);  
16  
17         frame.setSize(200, 200);  
18         frame.setVisible(true);  
19  
20     }  
21  
22     public static void main(String[] args) {  
23         new TextAreaDemo();  
24     }  
25 }
```

۲۱-۹

همانطور که ملاحظه می کنید چون ممکن است متى که داخل `JTextArea` نوشته شود خارج از محدوده های نمایشی آن باشد از `JScrollPane` استفاده شده است. نتیجه اجرای برنامه فوق به صورت زیر خواهد بود:



شکل ۲۱-۸. نتیجه اجرای `TextAreaDemo`

JPanel

`JPanel` یکی از اجزای نمایشی است که برخلاف دیگر اجزای نمایشی که تاکنون گفته شد جنبه نمایشی ندارد و در حقیقت یک `Container` محسوب می شود. از آن می توانید برای مدیریت آسان تر اجزای نمایشی و نمایش بهتر آنها استفاده کنید. برای اینکه نحوه کار و استفاده آن را متوجه شوید به مثال زیر دقت کنید.

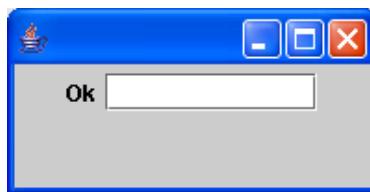
```
1 package ir.atlassoft.javase.chapter21;  
2  
3  
4 import javax.swing.*;  
5 import java.awt.*;
```

```

6
7 public class PanelDemo {
8     public PanelDemo () {
9         JFrame frame = new JFrame();
10        Container c = frame.getContentPane();
11
12        JPanel panel = new JPanel();
13        panel.setLayout(new FlowLayout());
14        panel.add(new JLabel("Ok"));
15        panel.add(new JTextField(10));
16        c.add(panel);
17
18        frame.setSize(200, 200);
19        frame.setVisible(true);
20    }
21
22    public static void main(String[] args) {
23        new PanelDemo();
24    }
25 }
```

۲۱-۱۰

در این برنامه ابتدا آبجکتی از JPanel ساخته شده است و سپس با فراخوانی متد () از این آبجکت، LayoutManager به عنوان FlowLayout برای آن تعیین شده است دو جزء نمایشی JTextField و JLabel به آن اضافه شده اند و در پایان، Container panel مورد نظر به اضافه شده است. نتیجه اجرای این برنامه به صورت زیر است:



شکل ۲۱-۹. نتیجه اجرای PanelDemo

در صورتیکه در یک پنجره اجزای نمایشی زیادی دارید و چیدن آنها در صفحه کار سختی است می توان با بکاربردن JPanel، پنجره مورد نظر را به چندین قسمت تقسیم کرد که هر کدام از Container های مختلفی برای چیدن اجزای خود استفاده می کنند سپس panel های تولید شده را به اضافه نمود.

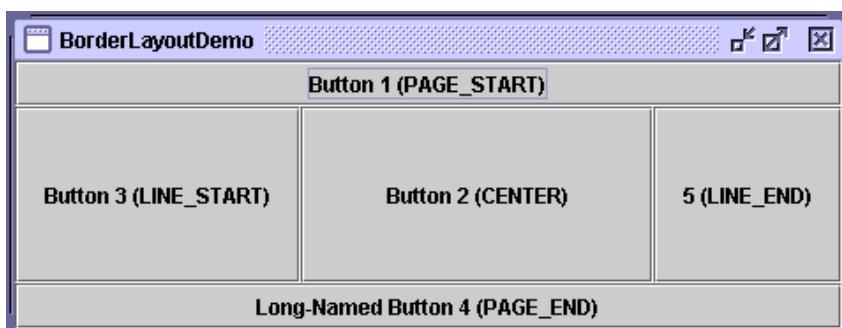
چندین LayoutManager دیگر

همانطور که در بخش های قبلی گفته شد، از **LayoutManager** برای مشخص کردن چیدمان اجزای نمایشی استفاده می شود همچنین با **FlowLayout** نیز آشنا شدید که با استفاده از آن تمام اجزای نمایشی به ترتیبی که به **Container** اضافه می شوند در طول یک خط افقی از چپ به راست چیده می شوند در صورتی که در آخر خط جای کافی برای یک جزء نمایشی وجود نداشته باشد بقیه اجزای نمایشی در خط بعدی از چپ به راست چیده می شوند.

فاقد بسیاری از خصوصیات لازم برای ساخت یک UI مناسب است به همین دلیل در عمل از **LayoutManager** های دیگر برای ساخت UI استفاده می شود در این قسمت به معرفی و استفاده از چندین **LayoutManager** دیگر خواهیم پرداخت.

BorderLayout

با استفاده از این **LayoutManager** به پنج ناحیه **Container** تقسیم می شود.



شکل ۲۱-۱۰ BorderLayout.

وقتی می خواهید یک جزء نمایشی را در یک ناحیه خاص از **Container** نمایش دهید لازم است در هنگام افزودن آن جزو به **Container**، با استفاده از مقادیر ثابتی که در کلاس **BorderLayout** تعریف شده است موقعیت آنرا مشخص کنید. تکه کدهای زیر نحوه ایجاد UI فوق را نشان می دهد:

```
//Container pane = aFrame.getContentPane()...
JButton button = new JButton("Button 1 (PAGE_START)");
pane.add(button, BorderLayout.PAGE_START);

//Make the center component big, since that's the
//typical usage of BorderLayout.
```

```

button = new JButton("Button 2 (CENTER)");
button.setPreferredSize(new Dimension(200, 100));
pane.add(button, BorderLayout.CENTER);

button = new JButton("Button 3 (LINE_START)");
pane.add(button, BorderLayout.LINE_START);

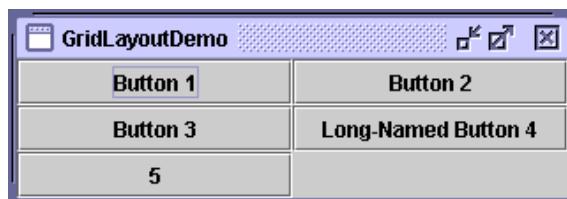
button = new JButton("Long-Named Button 4 (PAGE_END)");
pane.add(button, BorderLayout.PAGE_END);

button = new JButton("5 (LINE_END)");
pane.add(button, BorderLayout.LINE_END);

```

همانطور که مشاهده می کنید برای افزودن یک جزء نمایشی (در مثال فوق JButton) به Container استفاده شده است که پارامتر اول این متده مشخص کننده جزء نمایشی و پارامتر دوم مشخص کنند موقعیت نمایش آن جزء در Container است.

GridLayout



شکل ۱۱-۱۱. GridLayout

با استفاده از این LayoutManager می توانید Container را به یک ماتریس تقسیم کنید و هریک از اجزاء نمایشی را درون هر یک از خانه های ماتریس قرار دهید. در هنگام تعریف این LayoutManager باید طول و عرض ماتریس را مشخص کنید به عبارت دیگر لازم است تعداد سطرها و ستونهای ماتریس را مشخص کنید:

```
GridLayout layout = new GridLayout(3, 2);
```

(0, 0)	(0, 1)
(1, 0)	(1, 1)
(2, 0)	(2, 1)

فوق دارای ۳ سطر و ۲ ستون است.

واسط کاربری

پس از مشخص نمودن **LayoutManager**، می توانید با فراخوانی متده `add()` هر یک از اجزای نمایشی را به **Container** اضافه کنید، عناصر به ترتیب از خانه بالای سمت چپ (خانه `(0, 0)`) به اضافه می شوند. کدهای زیر نحوه ساختن UI فوق را نشان می دهد:

```
Container c = frame.getContentPane();
pane.setLayout(new GridLayout(3, 2));

pane.add(new JButton("Button 1"));
pane.add(new JButton("Button 2"));
pane.add(new JButton("Button 3"));
pane.add(new JButton("Long-Named Button 4"));
pane.add(new JButton("5"));
```

CardLayout

در برخی از UI ها، چندین جزء نمایشی در یک نقطه از UI نمایش می یابند و بسته به انتخاب کاربر یکی از اجزاء نمایش داده می شود. برای نمایش این اجزاء از **CardLayout** استفاده می شود به مثال زیر توجه کنید:

JComboBox ای را تصور کنید که با انتخاب یکی از گزینه های آن، محتوای یک ناحیه از پنجره نمایش تغییر می کند:



شکل .۲۱-۲۲

بسته به اینکه کاربر کدامیک از گزینه های **Combo Box** را انتخاب کند، **Text Field** یا **Label** را نمایش داده خواهد شد.

برای استفاده از این **LayoutManager**، باید **JPanel** را به صورت زیر ایجاد کنید

```
JPanel panel = new JPanel();
CardLayout layout = new CardLayout();
panel.setLayout(cardLayout);
```

سپس برای افزودن هر یک از اجزای نمایشی که قرار است روی یکدیگر (در یک ناحیه) نمایش داده شوند را به **panel** اضافه کنید

```
JLabel label = new JLabel("Label");
JTextField textField = new JTextField(10);
panel.add(label, "SHOW_LABEL");
panel.add(textField, "SHOW_TEXTFIELD");
```

توجه کنید که در هنگام افزودن هر یک از اجزای نمایشی به panel، از یک پارامتر String استفاده شده است. از این پارامتر بعدا برای نمایش آن جزء نمایشی استفاده می شود. مثلا در صورتیکه بخواهید label را نمایش دهید لازم است کد زیر را بنویسید:

```
layout.show(panel, "SHOW_LABEL");
و اگر بخواهید Text Field نمایش یابد باید کد زیر را بنویسید:
```

```
layout.show(textField, "SHOW_TEXTFIELD");
```

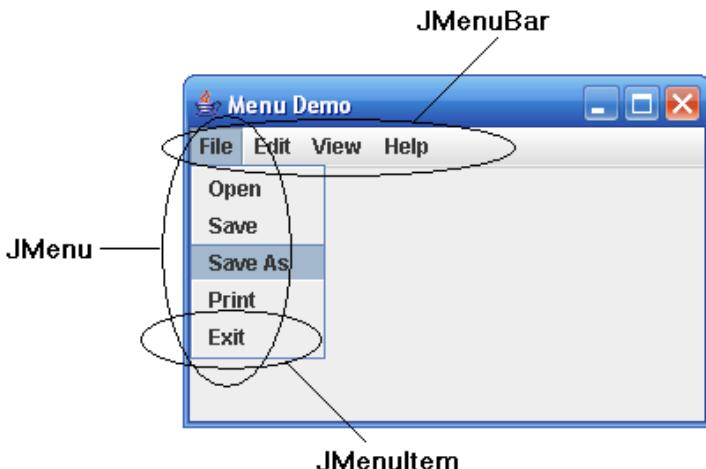
برنامه زیر ایجاد UI فوق را نشان می دهد:

```
1 package ir.atlassoft.javase.chapter21;
2
3 import javax.swing.*;
4 import java.awt.*;
5 import java.awt.event.ItemListener;
6 import java.awt.event.ItemEvent;
7
8 public class CardLayoutDemo implements ItemListener {
9
10    String SHOW_TEXTFILED = "TEXT_FIELD";
11    String SHOW_LABEL = "LABEL";
12
13    CardLayout cardLayout = new CardLayout();
14    JPanel panel3 = new JPanel();
15
16    String[] elements = {"Show TextField", "Show Label"};
17    JComboBox combo = new JComboBox(elements);
18
19    public CardLayoutDemo() {
20        JFrame frame = new JFrame();
21        Container c = frame.getContentPane();
22        c.setLayout(new BorderLayout());
```

```
23
24     JTextField field = new JTextField(10);
25     JLabel label = new JLabel("Label");
26
27     combo.addItemListener(this);
28
29     JPanel panel1 = new JPanel();
30     panel1.add(field);
31
32     JPanel panel2 = new JPanel();
33     panel2.add(label);
34
35     panel3.setLayout(cardLayout);
36
37     panel3.add(panel1, SHOW_TEXTFILED);
38     panel3.add(panel2, SHOW_LABEL);
39
40     c.add(combo, BorderLayout.PAGE_START);
41     c.add(panel3, BorderLayout.CENTER);
42
43     frame.setSize(300, 100);
44     frame.setVisible(true);
45
46 }
47
48 public static void main(String[] args) {
49     new CardLayoutDemo();
50 }
51
52 public void itemStateChanged(ItemEvent e) {
53     if(combo.getSelectedIndex() == 0){
54         cardLayout.show(panel3, SHOW_TEXTFILED);
55     }else{
56         cardLayout.show(panel3, SHOW_LABEL);
57     }
58 }
59 }
```

منوها

در بسیاری از برنامه ها، کاربر از طریق منوها امکان انجام تنظیمات، ذخیره کردن، چاپ کردن و... را دارد. در جاوا کلاس هایی برای ایجاد و نمایش منوها پیش بینی شده است برای ایجاد نوار منو از کلاس JMenuBar استفاده می شود، برای ایجاد هر یک از منوها از کلاس JMenu و برای ایجاد هر یک از گزینه های منو از کلاس JMenuItem استفاده می شود شکل زیر این مطلب را به خوبی نشان می دهد.



شکل ۳۱-۱۳. منوها و اجزای آنها

به این ترتیب باید برای ایجاد یک منو ابتدا یک آبجکت JMenuBar باسازید سپس به ازای هر یک از نوارها یک JMenu و به ازای هر یک از گزینه ها یک آبجکت از JMenuItem داشته باشد JMenu ها را باید به JMenuItem و JMenu را به JMenuItem اضافه کنید:

```
JMenuBar menuBar = new JMenuBar();
JMenu fileMenu = new JMenu("File");

JMenuItem menuItem = new JMenuItem("Open");
fileMenu.add(menuItem);
menuBar.add(fileMenu);
frame.setJMenuBar(menuBar); .
```

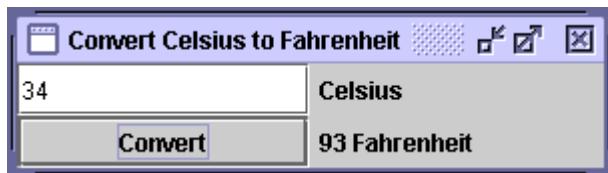
کلاس های JMenu و JMenuItem یک پارامتر String دریافت می کنند که همان رشته ای است که نمایش می دهند

خلاصه

در این فصل با API جاوا برای طراحی و ایجاد واسط گرافیکی کاربر آشنا شدید، در جاوا دو مجموعه کلاس و اینترفیس Swing و AWT برای این منظور وجود دارد AWT در نسخه های اولیه جاوا طراحی شده است اشکال اصلی AWT در این است که به سیستم عامل وابسته است و مثلا اگر برنامه ای در محیط ویندوز بنویسید ممکن است در محیط لینوکس به درستی کار نکند، از آنجاییکه AWT از امکانات سیستم عامل برای نمایش واسط گرافیکی استفاده می کند، قطعات نمایشی که توسط AWT تولید می شوند به قطعات نمایشی «سنگین وزن» (پاورقی: heavyweight) شهرت دارد. در مقابل Swing مدل پیشرفته تری نسبت به AWT است که در نسخه های اخیر توجه خاصی به آن شده است با استفاده از این مجموعه تقریبا هر جزء نمایشی را می توانید تولید کارایی آن به نسبت بالاتر از AWT است، باگ بسیار کمتری دارد، به platform وابستگی ندارد و به همین دلیل به آن «سبک وزن» (پاورقی: lightweight) گفته می شود. در این فصل با اجزای نمایشی Swing آشنا شدید. کلاسهای JLabel، JFrame، JDialoگ، JTable، JScrollPane، JTextArea، JTextField، JComboBox، JButton، JRadioButton و JCheckBox از مهمترین کلاسهای این مجموعه به شمار می روند. برای نمایش یک منو در یک پنجره، باید از آبجکت JMenuBar استفاده کنید هر منو با استفاده از کلاس JMenu و هر گزینه منو با استفاده از کلاس JMenuItem ساخته می شود. آبجکتهاي JMenu با آبجکتهاي JMenuBar قرار می گيرند و آبجکتهاي JFrame در JMenuBar قرار داده می شوند سپس با فراخوانی متده setJMenuBar() از آبجکت JFrame، منو فرييم قرار می گيرد.

تمرینات

۱- برنامه‌ای بنویسید که توسط یک Text Field یک عدد که معرف درجه حرارت سانتیگراد است را از کاربر دریافت کند، سپس کاربر با کلیک کردن روی یک دکمه، معادل فارنهایت آنرا مشاهده کند:



۲- برنامه‌ای بنویسید که کاربر از طریق یک Combo Box یک گزینه را انتخاب کند، سپس برنامه تصویر معادل گزینه انتخاب شده را به وی نشان دهد:



۳- برنامه‌ای بنویسید که کاربر از طریق واسط گرافیکی به مجموعه‌ای از سوالات پاسخ بگوید هر سوال به صورت چهار جوابی، و هر جواب آن از طریق یک Radio Button نمایش یابد:



۴- برنامه ای بنویسید که کاربر از طریق آن بتواند فایل یک عکس را روی سیستم خود انتخاب کند، سپس عکس انتخاب شده توسط برنامه نمایش داده شود. (توضیح: از کلاس JFileChooser استفاده کنید).
۵- فرض کنید برنامه ای برای یک رستوران نوشته اید، لیست تمام غذاهای رستوران را که در جدول FOOD از پایگاه داده ذخیره شده اند را در یک JTable نمایش دهید.

۵- برنامه ای بنویسید که مشخصات زیر را داشته باشد
وقتی اجرا می شود دیالوگی به کاربر نشان داده شود تا کاربر نام کاربری و رمز خود را وارد کند در صورتی که نام کاربری یا رمز کاربر نادرست باشد به وی اعلان کند و در غیر اینصورت پنجره مدیریت را نمایش دهد.
در پنجره مدیریت کاربر می تواند لیست تمام مشتریان بانک را در یک جدول مشاهده کند وقتی روی نام یک مشتری کلیک می کند در پنل پایینی اطلاعات جزئی تری از آن مشتری نمایش داده می شود.
کاربر می تواند با کلیک کردن روی گزینه "اضافه" یک مشتری جدید تعریف کند. همچنین با انتخاب یک مشتری که قبلاً تعریف شده است و انتخاب گزینه های "ویرایش" یا "حذف" آن مشتری را به ترتیب ویرایش یا حذف کند.

نمایش مشتریان در پنجره مدیریت باید دارای قابلیت "صفحه بندی" (pagination) باشد.
اطلاعات کاربران باید در پایگاه داده یا در فایل ذخیره شود.

۲۳

Logging

Logging از مفاهیم پایه و قدیمی نرم افزار محسوب می شود و هدف آن مشاهده و ردیابی روند اجرای برنامه هاست. اگر اجرای برنامه ها را به یک اتاق تاریک تشییه کنیم، Logging مثل یک چراغ روشن امکان می دهد حوادث و آنچه در اجرای برنامه اتفاق می افتد را مشاهده کنیم. در این فصل راجع به پیاده سازی Logging در جاوا صحبت خواهیم کرد و البته بحث خود را با این فرض شروع می کنیم که خواننده هیچگونه آشنایی با Logging و حتی مفاهیم اولیه آن ندارد.

Logging چیست؟

تا اینجای کتاب، از جمله `System.out.println()` برای نمایش متغیرها، ردیابی روند اجرای برنامه و به طور کلی دیباگ برنامه استفاده نمودیم. به این ترتیب از روی آنچه در زمان اجرای برنامه در کنسول برنامه نمایش می یابد می توان روند اجرای برنامه و وضعیت متغیرهای برنامه را مشاهده و برنامه را دیباگ کرد. اما به دلایل زیر استفاده از جملات `System.out.println()` مناسب نیست.

- حافظه کنسول برنامه موقتی است. این بدین معنی است که متنی که در کنسول برنامه نمایش می یابد پس از خاتمه یا توقف اجرای برنامه از حافظه پاک می شود و این موضوع در محیط عملیاتی که با داده های واقعی و با کاربران واقعی سروکار داریم به هیچ وجه مطلوب نیست. در محیط عملیاتی به مکانیسمی برای نگهداری پیامهای برنامه در یک حافظه ماندگار مثل فایل یا پایگاه نیاز داریم تا بتوان در آینده آنها را پردازش و تحلیل نمود.

- جملات `(System.out.println()` با تمام پیامها به یک شکل رفتار می کنند. در حالیکه ماهیت پیامها متفاوت است، مثلا پیامهای خطای، اطلاعاتی، هشدار، یا دیباگ وجود دارند، پیامهایی که در کنسول نمایش می یابند همگی از یک درجه اهمیت برخوردارند و راهی برای تفکیک آنها وجود ندارد.
- پیامهای کنسول یک قالب متحددالشکل ندارند، هر پیام به همان شکلی که به متدهای `System.out.println()` داده شود به همان شکل در کنسول برنامه نمایش می یابد. در یک محیط عملیاتی نیاز داریم تا بدون دستکاری اصل پیام که در متدهای `System.out.println()` فراخوانی شده است، قالب متحددالشکلی برای پیامها تعریف کنیم. مثلا زمان نمایش پیام به ابتدای پیام اضافه کنیم.

یک روش برای کنترل و مدیریت پیامهای برنامه است که ضعفهایی که در استفاده از `System.out.println()` وجود دارد را مرتفع کرده است. توجه داشته باشید که `Logging` جایگزین `System.out.println()` دائمی (`System.out.println()`) در مواردی است که از `System.out.println()` برای دیباگ و ردیابی اجرای برنامه استفاده میشود. همانطور که در ادامه این فصل خواهید دید `Logging` را می توان طوری تنظیم کرد که پیامها را در کنسول برنامه (با استفاده از `System.out.println()`) نیز نمایش دهد. بنابراین `Logging` را می توان یک API سطح بالاتر نسبت به `System.out.println()` تصور کرد که می تواند ضمن کنترل فرمت پیامها، آنها را دسته بندی و در جاهای مختلف ذخیره یا نمایش دهد. برخی از خصوصیات `Logging` عبارتند از:

- می توان بدون دستکاری پیامها، قالب پیامها را مشخص کرد یا تغییر داد.
- می توان پیامها را دسته بندی نمود (پیغام خطای، اخطار، اطلاعاتی، ...)، تا ضمن بامفهوم شدن هر پیام، کنترل بیشتری روی نمایش آن پیام داشته باشیم.
- می توان محل ذخیره پیامها (فایل، پایگاه داده، شبکه، کنسول، ...) را مشخص کرد.
- می توان نمایش پیغام ها را فعال یا غیرفعال نمود.

قبل از اینکه `Logging API` توسط جاوا ارایه شود فریم ورکهای منبع بازی توسط برنامه نویسان آزاد پیاده سازی شده بودند که کماکان توسعه داده و استفاده می شوند. یکی از این فریم ورکها `Log4j` است که از تمام رقیبان خود محبوب تر است و نسبت به `Logging API` که توسط جاوا ارایه می شود منعطف تر و قوی تر است.

Logging

JDK Logging

مهم ترین آبجکتی که در Logging وجود دارد همان پیام یا متنی است که قرار است ثبت شود(ذخیره شود یا نمایش داده شود). به این پیام اصطلاحا log گفته می شود و فایلهایی که این پیام ها در آنها ذخیره می شوند فایلهای لگ (Log File) نامیده می شوند. لاگها را علاوه بر فایل، می توان در پابگاه داده یا منابع دیگر ذخیره کرد، اما بیشتر اوقات استفاده از فایل به پایگاه داده و منابع دیگر ترجیح داده می شود. دلیلش هم این است که لاگها اولاً داده های بالهمیت و حیاتی برنامه ها محسوب نمی شوند تا در پایگاه داده نگهداری شوند و ثانیا سرعت کارکردن با فایل به مراتب بالاتر از پایگاه داده است که کمک می کند تا کارایی برنامه ها کمتر تحت تاثیر Logging قرار بگیرد. پایگاه داده ها سرویس‌های امنیت، تراکنش،... را فراهم می کنند که در ثبت پیامهای به آنها نیاز نیست و صرفا باعث سربار در ثبت پیامها می شوند. صرفظیر از اینکه لاگها در فایل یا پایگاه داده ثبت می شوند، کار ثبت لاگها را آبجکتی به نام Logger انجام می دهد که با فراهم کردن متدهایی از قبیل `log()`، `severe()`، `info()`، `warning()`، `finest()`، `finer()`، `config()` و `config()` ضمن ثبت یک پیام، اهمیت آن پیام را نیز مشخص می کند.

علاوه بر آبجکت Logger دو آبجکت مهم دیگر هم با نامهای Formatter و Handler وجود دارند که در پشت صحنه نقش بازی می کنند و برنامه نویسان کمتر با آنها سروکار دارند. آبجکت Formatter مسئولیت «قالبدھی پیام» و آبجکت Handler «ذخیره یا نمایش لگ» را به عهده دارد. هر برنامه نویس بدون اینکه درگیر این باشد که پیام با چه قالبی (آبجکت Formatter) و در چه منبعی ذخیره (آبجکت Handler) ثبت شود، صرفا لگ مورد نظر خود را با فراخوانی یکی از متدهای آبجکت Logger ثبت می کند. آبجکتهاFormatter و Handler در جای دیگری از برنامه و از طریق تنظیم و پیکربندی Logging تعیین می شوند. کد ۲۲-۱ نحوه ثبت پیام اطلاعاتی Hello را نشان می دهد.

```
1 package ir.atlassoft.framework.logging;
2
3 import java.util.logging.Logger;
4
5 public class LoggingUsage {
6
7     private final static Logger LOGGER =
8         Logger.getLogger(LoggingUsage.class.getName());
9
10    public void oneMethod() {
11        LOGGER.info("Hello!");
12    }
}
```

```

13     }
14 }
```

۲۲-۱ کد

در کلاس LoggingUsage یک متده فرضی oneMethod() وجود دارد که در آن پیام اطلاعاتی Hello به عنوان لگ ثبت شده است. اینکار با فراخوانی متده info() از آبجکت LOGGER انجام شده است. آبجکت LOGGER که به عنوان یک فیلد استاتیک تعریف شده، آبجکتی از کلاس java.util.logging.Logger است با فراخوانی متده getLogger() از همین کلاس ایجاد شده است (سازنده های کلاس Logger قابل فراخوانی نیستند). وقتی برنامه فوق را اجرا کنید، پیام Hello با فرمت زیر در کنسول برنامه نمایش می یابد!

```

Jul 31, 2018 3:03:47 PM ir.atlassoft.framework.log.TestLogging main
INFO: Hello!
```

فرمت فوق (SimpleFormatter)، فرمت پیش فرض لگ و کنسول برنامه (Handler) محل پیش فرض نمایش لگ است که در قسمتهای بعدی نحوه تغییر آنها را خواهید آموخت.

آبجکت Logger

ایجاد آبجکت Logger با فراخوانی متده getLogger() از کلاس java.util.logging.Logger انجام می شود. این متده یک پارامتر String دریافت می کند که نام آبجکت Logger نامیده می شود. به عبارت دیگر هر آبجکت Logger یک نام دارد که همان رشته ای است که به متده getLogger() ارسال می شود تا آبجکت Logger ساخته شود. نام هر آبجکت Logger منحصر بفرد است و با نام تمام آبجکتهای Logger برنامه متفاوت است. اگر دو بار متده getLogger() را با نام یکسان فراخوانی کنید، آبجکتهای یکسانی برگردانده می شود که معنی آن این است که به ازای هر نام فقط یک آبجکت Logger در برنامه ایجاد می شود. برای تسهیل نامگذاری آبجکتهای Logger، نام کامل کلاسی که تولید لگ در آن انجام می شود به عنوان نام Logger انتخاب می شود. مثلا در مثال فوق، از آنجاییکه آبجکت Logger در کلاس LoggingUsage تعریف شده است نام کامل کلاس از طریق جمله getLogger() بدست آمده و به متده LoggingUsage.getName() داده شده تا آبجکت LoggingUsage class.getName() ایجاد شود. دقت کنید که فراخوانی های زیر با یکدیگر مترادف هستند.

```

Logger.getLogger(LoggingUsage.class.getName())
Logger.getLogger("ir.atlassoft.framework.logging.LoggingUsage")
```

اما به علت ساده بودن و نگهداری آسان تر همیشه روش اول ترجیح داده می شود.

Logging

یکی از خصوصیات آبجکتهای Logger این است که بین یکدیگر رابطه پدر-فرزنده دارند. اینکه کدام آبجکت پدر و کدامیک فرزند آن است از طریق نام آنها مشخص می‌شود. آبجکتهای زیر را ملاحظه کنید.

```
Logger logger1= Logger.getLogger("ir");
Logger logger2= Logger.getLogger("ir.atlassoft");
Logger logger3= Logger.getLogger("ir.atlassoft.framework");
Logger logger4= Logger.getLogger("ir.atlassoft.framework.logging");
Logger logger5=
Logger.getLogger("ir.atlassoft.framework.logging.LoggingUsage");
```

نام آبجکت ir logger1 و logger2 نام آبجکت logger2 است. از آنجاییکه نام آبجکت logger2 با رشتہ ir شروع می‌شود که نام آبجکت logger1 است گفته می‌شود که logger2 فرزند logger1 است. با استدلال مشابه، آبجکت logger3 فرزند logger2 و پدر logger4 است و آبجکت logger4 فرزند logger3 و پدر logger5 است.

بنابراین می‌توان گفت که در مثال فوق، آبجکت logger1 پدر تمام آبجکتهای دیگر است. ممکن است بپرسید پدر آبجکت logger1 کیست؟ پدر این آبجکت، آبجکت global است که به صورت یک فیلد استاتیک در کلاس Logger تعریف شده است. آبجکت logger.global، پدر تمام آبجکتهای Logger است. حال که مشخص شد نسبت پدر و فرزندی برچه اساسی بین آبجکتهای Logger وجود دارد، حتماً می‌پرسید رابطه پدر و فرزندی چه کمکی به در مدیریت لاغهای می‌کند؟ جواب این سوال را پس از مطرح کردن مفهوم «سطح لاغ» خواهیم داد.

سطح لاغ

همانطور که قبلاً گفته شد ما نیاز داریم تا لاغهای را براساس اهمیتی که دارند گروه بندی کنیم. این خاصیت به ما امکان می‌دهد تا در شرایط مختلف (مثلاً در محیط عملیاتی) لاغهایی که اهمیت پایینی دارند را غیرفعال کنیم تا در خروجی ظاهر نشوند.

گروه بندی لاغهای با ارایه مفهومی تحت عنوان «سطح» (Level) امکان‌پذیر شده است. کلمه سطح در واقع سطح اولویت یا اهمیت یک لاغ را نشان می‌دهد. به همین منظور کلاس java.util.logging.Level برای نشان دادن سطح لاغ طراحی شده است. این بدین معنی است که برای نشان دادن یک سطح از آبجکتی از این کلاس استفاده می‌شود. در JDK Logging صورت آبجکتهایی از کلاس فوق و به صورت فیلد استاتیک در کلاس Level تعریف شده اند. در زیر، این سطوح براساس اهمیت از زیاد به کم لیست شده اند.

- SEVERE (highest)
- WARNING
- INFO

- CONFIG
- FINE
- FINER
- FINEST

در حالیکه SEVERE برای نشان دادن یک لاغ خطا استفاده می شود که از اهمیت بالایی برخوردار است، INFO و WARNING نشان دهنده لاغهای هشدار و اطلاعاتی هستند که اهمیت کمتری دارند. پایین ترین سطح هم FINEST است که برای پیامهایی استفاده می شود که صرفا در محیط توسعه و برای اطمینان از روال صحیح برنامه ثبت می شوند. بسته به نیاز و تصمیم خودتان می توانید از سطوح دیگر نیز در ثبت لاغها استفاده کنید.

برای ثبت یک لاغ کافیست متدهای log() از آبجکت Logger را فراخوانی کنیم و «متن لاغ» و «سطح لاغ» را به عنوان پارامترهای این متدها به صورت زیر مشخص کنیم.

```
LOGGER.log(Level.SEVERE, "Fatal Error Occurred!");
```

در اینجا پیام Fatal Error Occurred! به عنوان متن لاغ که از بالاترین سطح اهمیت (SEVERE) برخوردار است ثبت شده است. لاغ فوق را می توان با فراخوانی متدهای severe() نیز به شکل آسان تری ثبت کرد.

```
LOGGER.severe("Fatal Error Occurred!");
```

در اینجا سطح لاغ ذکر نشده، زیرا سطح لاغ در متدهای severe() مستتر است. حال وقت آن رسیده تا به سوالی که در قسمت قبل مطرح کردیم پاسخ دهیم. رابطه پدر و فرزندی بین آبجکتهای Logger چه فایده ای دارد؟!

در Logging این امکان وجود دارد که لاغهایی که پایین تر (کم اهمیت تر) از یک سطح هستند را غیرفعال کنیم تا در خروجی ظاهر نشوند. این کار با فراخوانی متدهای setLevel() از آبجکت Logger انجام می شود. مثلاً با فراخوانی جمله

```
logger.setLevel(Level.INFO);
```

آبجکت logger فقط لاغهایی که سطح اهمیت آنها INFO و بالاتر باشند را ثبت می کند و لاغهایی با سطح پایینتر را صرفنظر می کند. در این حالت گفته می شود که INFO «سطح موثر» آبجکت logger است. برای فعال کردن تمام سطوح از logger.setLevel(Level.ALL) و برای غیرفعال کردن تمام لاغها از logger.setLevel(Level.OFF) استفاده می شود. باید توجه کنید که هر آبجکت logger ممکن است سطح موثر خودش را داشته باشد، مثلاً آبجکت logger1 که در کلاس A استفاده می شود ممکن است

Logging

سطح موثر INFO داشته باشد، در حالیکه آبجکت logger2 که در کلاس B استفاده می شود سطح موثر WARNING داشته باشد.

نکته جالب در سطح موثر این است که این خاصیت به Logger های فرزند نیز سرایت می کند به عبارت دیگر اگر سطح موثر یک Logger را به سطح مشخص تنظیم کنیم تمام فرزندان آن Logger نیز از سطح موثر پدرشان پیروی می کنند.

همچنین یک Logger فرزند می تواند سطح موثری را که از پدرش به ارث برده تغییر دهد و سطح مورد نظر خودش را انتخاب کند، اما اگر هیچ سطحی را انتخاب نکند، سطح موثری پدرش برای وی نیز در نظر گرفته خواهد شد.

از طرف دیگر، از آنجاییکه Logger.global پدر تمام آبجکتهاي Logger محسوب می شود، هر سطحی INFO را برای این آبجکت تنظیم کنیم تمام Logger ها آنرا به ارث می بردند. به صورت پیش فرض، سطح برای Logger.global تنظیم شده که معنی آن این است که به صورت پیش فرض تمام پیامهایی که سطح آنها INFO و بالاتر باشد ثبت می شوند.

آبجکت Handler

همانطور که قبلاً گفته شد، بعد از اینکه آبجکت Logger متن لاغ و سطح آن را دریافت نمود، آنرا به آبجکت Handler تا در منبع داده ذخیره کند. کلاسهای Handler مختلفی در جاوا پیاده سازی شده است که پراستفاده ترین آنها FileHandler و ConsoleHandler هستند. این دو Handler به ترتیب برای نمایش لاغ در کنسول برنامه و ذخیره لاغ در فایل استفاده می شوند. برای فعال کردن هریک از آنها کافیست آبجکتی از روی این کلاسها ساخته شود و به صورت زیر به آبجکت Logger معرفی شود.

```
logger.addHandler(handler);
```

از نام متده addHandler() مشخص است که یک Handler می تواند بیش از یک آبجکت داشته باشد، مثلاً همزمان با نمایش لاغها در کنسول برنامه، آنها در یک فایل لاغ نیز ذخیره کند. در مبحث آبجکتهاي Logger توضیح دادیم که سطح موثر همانند یک فیلتر عمل می کند تا مشخص شود که یک لاغ توسط آبجکت Logger ثبت شود یا ثبت نشود. بعد از اینکه یک لاغ از فیلتر آبجکت Logger عبور کرد، به آبجکت Handler می رسد تا توسط آن، در فایل یا هر منبع داده دیگری ذخیره شود. نکته جالب اینکه، مشابه Logger ها مفهوم سطح موثر برای آبجکتهاي Handler نیز طراحی شده است. به عبارت دیگر، همانطور که برای آبجکت Logger یک سطح موثر وجود دارد، برای هر آبجکت Handler نیز یک سطح موثر وجود دارد که همانند فیلتری امکان ثبت شدن یا نشدن یک لاغ را در منبع نهایی می دهد. از آنجاییکه ممکن است چند آبجکت Handler مسئولیت ذخیره لاغها را در منابع مختلف به عهده داشته باشند، وجود سطح موثر در Handler ها امکان می دهد تا برخی لاغها در یک منبع داده ذخیره شود و دریک منبع دیگر ذخیره نشود. به صورت پیش فرض INFO به عنوان سطح موثر آبجکتهاي Handler طراحی شده

است. بنابراین اگر سطح آبجکت `Logger` را به پایین تر از `INFO` تغییر می دهید، لازم است تا سطح موثر آبجکتهای `Handler` را به شکل مشابه تغییر دهید زیرا در غیرانصورت، آبجکتهای `Handler` پیامهای پایین تر از سطح `INFO` را صرفنظر می کند.

آبجکتهای `Logger` به صورت پیش فرض از `ConsoleHandler` به عنوان `Handler` خود استفاده می کند. اما اگر بخواهیم لاغها علاوه بر کنسول در فایل هم ذخیره شوند، باید آبجکتی از `FileHandler` با استفاده از یکی از سازنده های این کلاس به صورت زیر ساخت.

```
FileHandler handler = new FileHandler();
FileHandler handler = new FileHandler(String pattern);
FileHandler handler = new FileHandler(String pattern, boolean append);
FileHandler handler = new FileHandler(String pattern, int limit, int count);
FileHandler handler = new FileHandler(String pattern, int limit, int count,
    boolean append);
```

در تمام سازنده های فوق، `pattern` الگوی نامگذاری فایلهای لاغ را مشخص می کند. `append` مشخص می کند که اگر فایل لاغ قبلا وجود داشته باشد آیا لاغهای تولید شده به محتویات قبلی آن اضافه شود یا فایل بازنویسی شود. `limit` حداکثر اندازه هر فایل و `count` حداکثر تعداد فایلهای لاغ را مشخص می کند. وقتی تعداد فایلهای لاغ به حداکثر تعداد ممکن بررسد لاغهای جدید در فایلهای موجود از ابتدا بازنویسی می شوند. مثالهای زیر نمونه هایی از ایجاد آبجکت `FileHandler` را نشان می دهد.

```
FileHandler handler = new FileHandler("myapp-log.%u.%g.txt");
FileHandler handler = new FileHandler("myapp-log.%u.%g.txt", true);
FileHandler handler = new FileHandler("myapp-log.%u.%g.txt", 1024 *
1024, 10);
FileHandler handler = new FileHandler("myapp-log.%u.%g.txt",
    1024 * 1024, 10, true);
```

در آبجکتهای فوق، تنها موضوعی که نیاز به توضیح بیشتر دارد الگوی نامگذاری فایلهای است که در قسمت بعدی به آن می پردازیم.

الگوی نامگذاری فایل لاغ

در مثالهای فوق، رشتہ ای که به عنوان `pattern` مشخص شده، رشتہ ای است که با `myapp-log` شروع می شود و به `.txt` ختم می شود و معنی آن این است که نام فایل های لاغ با کلمه `myapp-log` شروع

Logging

شود و با پسوند `.txt` باشد. در این رشتہ، `%u` و `%` نمادهایی هستند که در زمان تولید لاغ با کلمات دیگری جایگزین می شوند. جدول زیر لیست نمادهای قابل استفاده در تعریف الگو را نشان می دهد.

معنی	کد
جداگانده مسیر دایرکتوری و فایلها که در ویندوز \ و در بقیه سیستم عاملها / است. (مثلا مسیر یک دایرکتوری در ویندوز به صورت <code>C:\Projects\MyProject</code> و در لینوکس به صورت <code>/usr/Projects/MyProject</code> است)	/
دایرکتوری temp در سیستم را مشخص می کند.	<code>%t</code>
دایرکتوری کاربر را در سیستم نشان می دهد. (مثلا اگر در ویندوز نام کاربری شما <code>username</code> باشد دایرکتوری کاربر شما <code>\Users\username\</code> است)	<code>%h</code>
یک عدد شمارشی است که ترتیب فایلهای لاغ که تولید شده اند را مشخص می کند.	<code>%g</code>
یک عدد منحصر بفرد که مانع تداخل نام فایلها می شود.	<code>%u</code>
اگر بخواهید علامت % در نام فایل وجود داشته باشد.	<code>%%</code>

لیست ۱-۳۲. نمادهای قابل استفاده در تعریف الگو

آبجکت Formatter

قالب نمایش یا ذخیره پیامهای لاغ توسط آبجکت های `Formatter` مشخص می شود. دو کلاس `XMLFormatter` و `SimpleFormatter` به ترتیب برای قالبدهی پیامهای لاغ در قالب متن ساده و `XML` پیاده سازی شده اند. البته این امکان وجود دارد که با توسعه کلاس `Formatter` یک قالبدهی جدید برای متن پیامهای لاغ پیاده سازی کنید.

آبجکت LogManager

همانطور که از نام این آبجکت مشخص است، مسئولیت پیکربندی و مدیریت آبجکتهاي `Logger` را بر عهده دارد. این آبجکت فایل تنظیمات `logging.properties` که در مسیر `jre/lib/logging.properties` قرار دارد را بارگذاری کرده و پیکربندی می کند تا آبجکتهاي `Logger` بر اساس آن ایجاد شوند. فایل `logging.properties` که حاوی تنظیمات است را می توان ویرایش کرد و تنظیمات آنرا تغییر داد که در اینصورت تمام برنامه هایی که با آن جاوا اجرا می شوند از تنظیمات جدید تاثیر خواهد گرفت. اما یک راه بهتر ایجاد فایل `logging.properties` که جدیدی مخصوص پروژه خودتان است که در اینصورت باید با استفاده از دستور زیر در زمان اجرای برنامه آنرا به جای فایل پیش فرض بارگذاری کنید.

```
java -Djava.util.logging.config.file=configFile MainClass
```

در اینجا configfile مسیر فایل logging.properties است که قصد داریم به جای فایل پیش فرض بارگذاری شود. از آنجاییکه هر پروژه ممکن است تنظیمات Logging متفاوتی داشته باشد، استفاده از یک فایل logging.properties مخصوص پروژه راه حل منطقی تری به نظر می رسد.

در فایل تنظیمات، از طریق خط زیر می توان سطح پیش فرض آبجکتهای Logger را تعیین کرد.

```
.level=INFO
```

همچنین با استفاده از جمله زیر می توان سطح فعال یک Logger با نام com.mycompany.myapp را به سطح INFO تنظیم کرد.

```
com.mycompany.myapp.level=FINE
```

برای تعیین سطح موثر آبجکت handler نیز از خط زیر استفاده می کنیم.

```
java.util.logging.ConsoleHandler.level=FINE
```

در اینجا سطح موثر ConsoleHandler با FINE تنظیم شده است.

علاوه بر استفاده از فایل تنظیمات، از آبجکت LogManager هم می توان به روش برنامه نویسی سطح فعال آبجکتهای Logger را نیز مشخص کرد.

```
LogManager.setLevel(String name, Level level)
```

در اینجا name نام آبجکت Logger و level سطح مورد نظر آن است.

Logging

تمرینات

در تمام تمرینات فصل ۱۸، ارتباط با پایگاه داده، از Logging برای ثبت وقایع استفاده کنید.

۲۳

عبارت‌های لامبدا و کلاس‌های تو در تو

یک کلاس را می‌توان داخل یک کلاس دیگر تعریف کرد.

```
class OuterClass {  
    ...  
    class NestedClass {  
        ...  
    }  
}
```

در اینجا کلاس NestedClass که در محدوده داخل کلاس OuterClass تعریف شده است یک کلاس داخلی است. کلاس‌های داخلی همانند هر کلاس دیگری هستند، یعنی می‌توانند سازنده‌ها، متدها و فیلدهای خود را داشته باشند.

کلاس‌های داخلی می‌توانند به صورت استاتیک یا غیراستاتیک تعریف شوند.

```
class OuterClass {  
    ...  
    static class StaticNestedClass {  
        ...  
    }  
    class InnerClass {  
        ...  
    }  
}
```

به کلاس داخلی غیراستاتیک، کلاس `Inner` و به کلاس داخلی استاتیک کلاس `Nested` گفته می‌شود. کلاس «داخلی غیراستاتیک»، عضوی از کلاس بیرونی قلمداد می‌شود بنابراین به تمام اعضای کلاس بیرونی (متدها و فیلدها) حتی اگر به صورت `private` تعریف شده باشند دسترسی دارد. اما کلاس داخلی استاتیک به هیچ یک از اعضای کلاس بیرونی دسترسی ندارد.

موارد کاربرد کلاس داخلی

دسته بندی منطقی کلاسهای. اگر یک کلاس فرعی صرفاً توسط یک کلاس اصلی استفاده می‌شود می‌توان کلاس فرعی را به عنوان کلاس داخلی در کلاس اصلی تعریف کرد. این طراحی، نقش کمرنگ کلاس فرعی و ارتباط قوی آن با کلاس اصلی را بهتر نشان می‌دهد.

کپسوله سازی بهتر. با تعریف کلاس فرعی به عنوان کلاس داخلی، مفاهیم `encapsulation` که تاکید دارد محتویات کلاسهای باید از دسترسی خارجی محافظت شوند کاملاً رعایت می‌شوند. اگر کلاس فرعی به عنوان کلاس داخلی تعریف نشود، شاید مجبور شویم به برخی فیلدهای آن متدهای کلاس اصلی دسترسی غیراز `private` بدھیم تا کلاس فرعی بتواند به آنها دسترسی پیدا کند (که در اینصورت به دیگر کلاسهای هم ناخواسته دسترسی داده می‌شود)، اما با تعریف کلاس داخلی، کلاس فرعی می‌تواند به تمام اعضای کلاس اصلی دسترسی پیدا کند بدون آنکه لازم باشد آنها را از `private` دربیاوریم. کدخواناتر و قابل نگهداری. تعریف کلاس داخلی در موقعی که ایجاب می‌کند باعث تولید کدخواناتر و قابل نگهداری می‌شود.

کلاس داخلی استاتیک

اگر یک کلاس داخلی، استاتیک باشد همانند هر عضو استاتیک دیگر (متدهای استاتیک یا فیلدهای استاتیک) و بدون اینکه آبجکتی از کلاس بیرونی وجود داشته باشد می‌توان به آن دسترسی یافت.

`OuterClass.StaticNestedClass`

براساس همین قاعده و به شکل زیر می‌توان از روی کلاس استاتیک داخلی آبجکت ساخت

```
OuterClass.StaticNestedClass nestedClass = new
OuterClass.StaticNestedClass();
```

کلاس استاتیک داخلی نمی‌تواند به فیلدهای غیراستاتیک از کلاس بیرونی دسترسی یابد. این قاعده کاملاً منطقی است زیرا فیلدهای و متدهای غیراستاتیک کلاس بیرونی فقط زمانی قابل فراخوانی هستند که آبجکتی از کلاس بیرونی وجود داشته باشد این در حالیست که ما بدون ایجاد آبجکتی از کلاس بیرونی، آبجکتی از کلاس استاتیک داخلی ایجاد کردیم.

کلاس غیراستاتیک داخلی

کلاس غیراستاتیک داخلی همانند هر متدهای فیلد یا فیلد غیراستاتیک داخلی یک کلاس رفتار می‌کند، یعنی به تمام فیلدها و متدهای کلاس بیرونی دسترسی دارد و در عین حال فقط زمانی قابل دسترسی است که آبجکتی از کلاس بیرونی وجود داشته باشد. همچنین به دلیل اینکه بدون وجود یک آبجکت از کلاس بیرونی نمی‌توان به کلاس داخلی دسترسی یافت، کلاس داخلی نمی‌تواند عضو استاتیک داشته باشد.

```
class OuterClass {
    ...
    class InnerClass {
        ...
    }
}
```

برای اینکه آبجکتی از کلاس داخلی ایجاد کنیم می‌بایست ابتدا آبجکتی از کلاس بیرونی داشته باشیم.

```
OuterClass outer = new OuterClass();
OuterClass.InnerClass inner = outer.new InnerClass();
```

یک مثال

فرض کنید به یک کلاس ساختمان داده نیاز داشته باشیم که آرایه‌ای از اعداد را نگهداری می‌کند. این کلاس شکلی به صورت زیر خواهد داشت.

```
1 package ir.atlassoft.javase.chapter23;
2
3 public class ArrayStructure {
4
5     // Create an array
6     final static int SIZE = 15;
7     int[] arrayOfInts = new int[SIZE];
8
9     public ArrayStructure() {
10         // fill the array with ascending integer values
11         for (int i = 0; i < SIZE; i++) {
12             arrayOfInts[i] = i;
13         }
14     }
15 }
```

کد ۱۳-۱

در این کلاس یک آرایه از `int` یک متغیر استاتیک که اندازه آرایه را مشخص می‌کند تعریف شده است. سازنده این کلاس، آرایه را با اعداد ۱ تا ۱۵ پر می‌کند.

حال تصور کنید که از ما بخواهند تا یک کلاس کاربردی پیاده سازی کنیم که عناصر آرایه که در ایندکسهای زوج آرایه قرار دارند را برگرداند. این کلاس را با بهره گیری از اینترفیس `java.util.Iterator` به شکل زیر پیاده سازی می کنیم.

```

1 package ir.atlassoft.javase.chapter23;
2
3 import java.util.Iterator;
4
5 public class EvenIterator implements Iterator<Integer> {
6
7
8     ArrayStructure arrayStructure;
9
10    public EvenIterator(ArrayStructure arrayStructure) {
11        this.arrayStructure= arrayStructure;
12    }
13
14    // Start stepping through the array from the beginning
15    private int nextIndex = 0;
16
17    public boolean hasNext() {
18
19        // Check if the current element is the last in the
20        array
21        return (nextIndex <= arrayStructure.SIZE - 1);
22    }
23
24    public Integer next() {
25
26        // Record a value of an even index of the array
27        Integer retValue =
28        Integer.valueOf(arrayStructure.arrayOfInts[nextIndex]);
29
30        // Get the next even element
31        nextIndex += 2;
32        return retValue;
33    }
34}

```

کلاس EvenIterator یک کاندیدای مناسب برای پیاده سازی به عنوان یک کلاس داخلی است زیرا اولاً محدوده فعالیت این کلاس صرفاً محدود به کلاس ArrayStructure است ثانیاً با تبدیل شدن به کلاس داخلی، کلاس EvenIterator کپسوله سازی بهتری می‌تواند ارایه کند. اجازه دهید تا این کلاس را به صورت کلاس داخلی تعریف کنیم و تغییرات را مشاهده کنیم.

```

1 package ir.atlassoft.javase.chapter23;
2
3 import java.util.Iterator;
4
5 public class ArrayStructure2 {
6
7     // Create an array
8     private final static int SIZE = 15;
9     private int[] arrayOfInts = new int[SIZE];
10
11    public ArrayStructure2() {
12        // fill the array with ascending integer values
13        for (int i = 0; i < SIZE; i++) {
14            arrayOfInts[i] = i;
15        }
16    }
17
18    private class EvenIterator implements Iterator<Integer> {
19
20
21        // Start stepping through the array from the beginning
22        private int nextIndex = 0;
23
24        public boolean hasNext() {
25
26            // Check if the current element
27            // is the last in the array
28            return (nextIndex <= SIZE - 1);
29        }
30
31        public Integer next() {
32

```

```

33         // Record a value of an even index of the array
34         Integer retValue =
35     Integer.valueOf(arrayOfInts[nextIndex]);
36
37         // Get the next even element
38         nextIndex += 2;
39         return retValue;
40     }
41 }
42 }
```

کد ۲۳-۳

تغییراتی که اتفاق افتاده به شرح زیر است فیلدهای SIZE و arrayOfInts به صورت private تعریف شده اند تا کپسوله سازی بهتری ارایه شود. سازنده کلاس ArrayStructure که یک آبجکت از EveIterator دریافت می نمود حذف شده است زیرا کلاس داخلی EvenIterator فقط زمانی استفاده می شود که آبجکتی از کلاس بیرونی ArrayStructure وجود داشته باشد. از آنجاییکه کلاس داخلی EvenIterator به فیلدهای کلاس بیرونی ArrayStructure دسترسی دارد درون متدهای next() و hasNext() به فیلدهای SIZE و arrayOfInts دسترسی شده است.

با طراحی فوق، کلاس ArrayStructure می تواند دارای یک متدهای کاربردی printEven() باشد که از کلاس داخلی فوق برای نمایش ایندکس‌های زوج استفاده می کند.

```

public void printEven() {
    // Print out values of even indices of the array
    Iterator iterator = this.new EvenIterator();
    while (iterator.hasNext()) {
        System.out.print(iterator.next() + " ");
    }
    System.out.println();
}
```

کلاس داخلی می تواند به صورت package-visible یا protected، private، public تعریف شود.

کلاس‌های محلی

خیلی جالب است که بدانید داخل بدنه یک متدهم می توان یک کلاس تعریف کرد. به این کلاس‌ها، کلاس‌های محلی گفته می شود که دامنه استفاده از آنها محدود به بلوکی (نزدیکترین {} و {}) که در آن تعریف شده اند.

Lambda

به عنوان مثال متدها validatePhoneNumber() زیر را تصور کنید که دو شماره تلفن را دریافت کرده و اگر معتبر باشد آن شماره در فرمت صحیح و در صورت نامعتبر بودن، کلمه نامعتبر را نمایش می‌دهد.

```
1 package ir.atlassoft.javase.chapter23;
2
3 public class LocalClassExample {
4
5     static String regularExpression = "[^0-9]";
6
7     public static void validatePhoneNumber(
8         String phoneNumber1, String phoneNumber2) {
9
10        final int numberLength = 10;
11
12        // Valid in JDK 8 and later:
13
14        // int numberLength = 10;
15
16        class PhoneNumber {
17
18            String formattedPhoneNumber = null;
19
20            PhoneNumber(String phoneNumber) {
21                // numberLength = 7;
22                String currentNumber = phoneNumber.replaceAll(
23                    regularExpression, "");
24                if (currentNumber.length() == numberLength)
25                    formattedPhoneNumber = currentNumber;
26                else
27                    formattedPhoneNumber = null;
28            }
29
30            public String getNumber() {
31                return formattedPhoneNumber;
32            }
33
34            // Valid in JDK 8 and later:
35
36            // public void printOriginalNumbers() {
```

```

37 //           System.out.println("Original numbers are " +
38 //                               phoneNumber1 +
39 //                               " and " + phoneNumber2);
40 //               }
41 }

42

43 PhoneNumber myNumber1 = new PhoneNumber(phoneNumber1);
44 PhoneNumber myNumber2 = new PhoneNumber(phoneNumber2);
45
46 // Valid in JDK 8 and later:
47
48 //     myNumber1.printOriginalNumbers();
49
50 if (myNumber1.getNumber() == null)
51     System.out.println("First number is invalid");
52 else
53     System.out.println("First number is " +
54                     myNumber1.getNumber());
55 if (myNumber2.getNumber() == null)
56     System.out.println("Second number is invalid");
57 else
58     System.out.println("Second number is " +
59                     myNumber2.getNumber());
60
61 }
62
63 public static void main(String... args) {
64     validatePhoneNumber("123-456-7890", "456-7890");
65 }
66 }
```

۲۳-۴

در این مثال، یک کلاس LocalClassExample پیاده سازی شده که حاوی یک متده است. در بدنه متده validatePhoneNumber() یک کلاس محلی با نام PhoneNumber که حاوی یک متده getNumber() است پیاده سازی گردیده است. کلاس PhoneNumber یک شماره تلفن را در قالب یک رشته از طریق سازنده خود دریافت می کند و آنرا صحت سنجی می کند. برای این منظور تمام کاراکترهای غیر عددی را از رشته حذف می کند، و سپس در صورتیکه رشته باقیمانده از ۱۰ رقم تشکیل شده باشد آنرا به عنوان شماره صحیح در فیلد

نگهداری و درغیراینصورت آنرا به عنوان شماره نامعتبر قلمداد می کند. حال اگر متدها از این آبجکت را فراخوانی کنیم و مقدار `null` برگرداند شماره غیرمعتبر بوده و اگر مقدار غیراز `null` برگرداند شماره صحیح بوده است.

دسترسی به اعضای کلاس بیرونی

در یک کلاس محلی می توان به تمام اعضای کلاس بیرونی دسترسی پیدا کرد. به عنوان نمونه در مثال قبل، از فیلد `regularExpression` (عضوی از کلاس `LocalClassExample`) در کلاس `PhoneNumber` استفاده شده است. همچنین کلاس محلی می تواند به تمام متغیرهای محلی که در صورتیکه `final` تعریف شده باشند دسترسی پیدا کند. به عنوان نمونه، در مثال فوق متغیر محلی (`داخل متدها`) باشد `validatePhoneNumber()` تعریف شده است. همچنین اگر متغیری به صورت `final` تعریف شده باشد اما مقدار آن در طول متدها تغییر نماید می توان در کلاس محلی به آن دسترسی پیدا کرد. (به این متغیر `effectively final` گفته می شود). در Java 8 کلاس محلی می تواند به پارامترهای متدها که در آن تعریف شده است دسترسی یابد این موضوع در متدهای `printOriginalNumbers()` نشان داده شده است.

شباهت کلاس محلی به کلاس داخلی غیراستاتیک

کلاسهای محلی مشابه کلاسهای داخلی غیراستاتیک نمی توانند به اعضای استاتیک کلاس بیرونی دسترسی یابند. همچنین کلاسهای محلی که درون متدهای استاتیک تعریف می شوند فقط می توانند به اعضای استاتیک کلاس بیرونی دسترسی پیدا کنند.

اگر یک کلاس محلی درون یک متدهای استاتیک تعریف شود به تمام فیلدهای استاتیک کلاس بیرونی دسترسی دارد ولی به اعضای غیراستاتیک کلاس بیرونی دسترسی ندارد زیرا متدهای استاتیک که کلاس محلی درون آن تعریف شده بدون ایجاد آبجکت از کلاس بیرونی فراخوانی می شود و بنابراین فیلدهای غیراستاتیک کلاس بیرونی در حافظه وجود ندارند تا بتوان به آنها دسترسی پیدا کرد. اما اگر کلاس محلی داخل یک متدهای استاتیک تعریف شود به تمام اعضای استاتیک و غیراستاتیک کلاس بیرونی دسترسی دارد. کلاسهای محلی نمی توانند به صورت استاتیک تعریف شوند و همچنین نمی توانند حاوی یک عضو (فیلد یا متدهای استاتیک باشند ولی اگر فیلد فیلدی `final` تعریف شود می تواند استاتیک هم باشد.

توجه: درون یک بلوک (مثلاً یک متدها) نمی توان یک اینترفیس تعریف کرد زیرا اینترفیسها استاتیک فرض می شوند.

کلاس‌های بی نام (Anonymous)

براساس آنچه تاکنون دیده اید یک کلاس (اعم از کلاس عمومی، کلاس داخلی، یا کلاس محلی) در نقطه‌ای از برنامه تعریف می‌شود تا آبجکتها‌یی از روی آن کلاس در نقاط دیگر برنامه ایجاد شود. کلاس بی نام (Anonymous) نوع دیگری از تعریف کلاس است که بدون اینکه نامی به کلاس اختصاص داده شود یک کلاس تعریف می‌شود و همانجا آبجکتی از روی آن ساخته می‌شود. به عنوان نمونه، می‌توان یک کلاس Thread تعریف و آبجکتی از روی آن به صورت زیر ایجاد نمود.

```
Runnable threadObj = new Runnable() {
    public void run() {
        //do something
    }
};
```

از آنجاییکه `Runnable` یک اینترفیس است، نمی‌توان از روی آن آبجکتی ایجاد کرد ولی در اینجا یک کلاس بی نام تعریف شده که این اینترفیس را پیاده سازی کرده و در واقع از روی این کلاس بی نام آبجکت ایجاد شده است. به شکل مشابه، می‌توان یک کلاس را توسعه داد و در همانجا آبجکتی از روی آن ایجاد کرد. قواعد تعریف کلاس بی نام به شرح زیر است.

- استفاده از عملگر `new` که برای ایجاد آبجکت استفاده می‌شود.
- نام اینترفیس که قرار است پیاده سازی شود یا کلاسی که قرار است توسعه یابد بلافاصله بعد از `new` می‌آید.
- بعد از نام کلاس یا اینترفیس پرانتز باز و بسته `()` می‌آید. این پرانتزها به معنی فراخوانی سازنده کلاس است که اگر کلاسی که توسعه می‌یابد دارای سازنده است داخل پرانتز می‌باشد مقادیر پارامترهای سازنده پدر مشخص شود.
- بعد از برانتزها، داخل بلوکهای `{}` بدنه کلاس مثل هر کلاس دیگری تعریف می‌شود.
- در انتهای بلوک مثل انتهای هر جمله ای `;` گذاشته می‌شود.

کلاس بی نام یک قابلیت فوق العاده است، زیرا در برنامه‌های جاوایی موارد زیادی وجود دارد که بخواهیم به تعداد زیاد یک کلاس را توسعه دهیم و از روی آن آبجکت بسازیم. اگر امکان کلاس بی نام وجود نداشت مجبور بودیم کلاس‌های زیادی تعریف کنیم که می‌توانست پروژه را شلوغ و سرشار از کلاس‌های خسته کننده کند! که نام‌های آنها به اعداد `1, 2, 3, ...` ختم می‌شد.

رفتار کلاس‌های بینام بسیار شبیه کلاس‌های محلی است، بنابراین به تمام اعضای کلاس بیرونی دسترسی دارند و به متغیرهای محلی که `final` یا `final` effectively تعریف نشده باشند دسترسی ندارند. مشابه یک کلاس داخلی، تعریف یک متغیر داخلی در کلاس بی نام روی متغیر همنام آن در کلاس بیرونی تاثیر می‌گذارد که به آن خاصیت `Shadow` گفته می‌شود.

- همچنین کلاس بی نام می تواند فیلد استاتیک داشته باشد با شرط اینکه آن فیلد به صورت `final` تعریف شده باشد.

توجه: کلاسهای بی نام نمی توانند دارای سازنده باشند.

یکی از موارد کاربرد کلاس بی نام در پیاده سازی Listener هاست که در ادامه به آن می پردازیم.

Listener

تصویر کنید که در یک برنامه، منتظر رخداد یک حادثه باشیم تا پس از اینکه آن حادثه اتفاق افتاد عملیاتی انجام شود. به عنوان نمونه، یک بازی کامپیوتري را تصور کنید که سه گزینه به کاربر نشان میدهد و منتظر می ماند تا کاربر یک گزینه را انتخاب کند تا براساس انتخاب کاربر، بازی شروع شود یا ادامه یابد.

قسمت چالش برانگیز این برنامه جایی است که با کاربر تعامل می کند. برای پیاده سازی این قسمت دو راه حل داریم، راه حل اولی که در ابتدا به ذهن می رسد این است که کدی بنویسیم که گزینه هایی را به کاربر نشان دهد و سپس به صورت مداوم یا هر چند ثانیه (مثلا در یک حلقه for) انتخاب کاربر را بررسی کند تا بعد از اینکه کاربر گزینه خود را انتخاب می کند برنامه ادامه یابد. در این راه حل، ممکن است کاربر گزینه خود را در یک زمان نامعلوم انتخاب کند که به معنی این است که برنامه برای مدت نامعلوم در انتظار کاربر متوقف می ماند.

راه حل دوم برای پیاده سازی این نیازمندی، استفاده از مکانیسم Event/Listener است. براساس این طراحی، نیازمندی فوق را می توان در سه آبجکت مجازی Event و Listener و Source پیاده سازی کرد. آبجکت Event معرف «حادثه» ای است که اتفاق می افتد و در این مثال، گزینه ای است که توسط کاربر انتخاب می شود (مثلا گزینه ۱، گزینه ۲ یا گزینه ۳).

```
1 package ir.atlassoft.javase.chapter23;
2
3 public class Event {
4     int option;
5
6     public Event(int option) {
7         this.option = option;
8     }
9
10    public int getOption() {
11        return this.option;
```

```

12     }
13
14     public void setOption(int option){
15         this.option=option;
16     }
17 }
```

کد ۲۳-۵

آبجکت Listener همان آبجکتی است که بعد از رخداد حادثه (در این مثال انتخاب یک گزینه) عملیات متناسب را انجام دهد.

```

1 package ir.atlassoft.javase.chapter23;
2
3 public class SelectionListener{
4     public void process(Event event){
5         System.out.println("Optoin "+
6                             event.getOption()+
7                             " has selected");
8     }
9 }
```

کد ۲۳-۶

در اینجا کلاس SelectionListener دارای یک متده است که آبجکتی از Event دریافت می کند و حاوی عملیاتی است که باید پس از انتخاب گزینه انجام شوند. برای اینکه مثال خود را ساده نگهداشیم این متده صرفا گزینه انتخاب شده توسط کاربر در کنسول را نمایش می دهد. سومین آبجکت، آبجکت Source است که حادثه در آن اتفاق می افتد. در اینجا نیز بدون اینکه در گیر جزئیات این آبجکت شویم کلاس Interactive را که معرف آبجکت Source است به صورت زیر بیان سازی می کنیم.

```

1 package ir.atlassoft.javase.chapter23;
2
3 public class Interactive{
4
5     SelectionListener listener;
6
7     public void setListener(SelectionListener listener) {
8         this.listener=listener;
```

```

9      }
10
11     public void select(int option) {
12         Event event = new Event(option);
13         listener.process(event);
14     }
15
16     public void showOptions() {
17         //show options to user
18     }
19 }
```

۲۳-۷

همانطور که ملاحظه می کنید، در کلاس `Interactive` متد `showOptions()` پیاده سازی شده تا گزینه ها را به کاربر نشان دهد. در این کلاس، فیلدی از آبجکت `Listener` تعریف شده است که توسط متد `setListener()` مقداردهی می شود. وقتی کاربر گزینه ای را انتخاب می کند (انتخاب یک گزینه با `select()` انجام می شود) از آبجکت `process()` فراخوانی شود. به این ترتیب و با طراحی سه کلاس `Interactive`، `SelectionListener` و `Event`، فرایند اجرای برنامه از طریق کدهای زیر قابل انجام است.

```

Interactive interactive = new Interactive();
SelectionListener listener = new SelectionListener();
interactive.setListener(listener);
interactive.showOptions();
```

توجه: طراحی Event/Listener مبنای نوعی از فراخوانی متد است که به آن «فراخوانی غیرهمزمان» گفته می شود. همانطور که می دانید وقتی در نقطه ای از برنامه متدى فراخوانی می شود برنامه در آن نقطه متوقف می ماند تا اجرای متد فراخوانی شده به اتمام برسد. به این نوع فراخوانی «فراخوانی همزمان» گفته می شود. در مواردی ممکن است متدى که فراخوانی می شود خیلی طولانی باشد (بعنی اجرای آن زمانبر باشد) یا برای اجرا نیازمند حادثه ای باشد که هنوز اتفاق نیفتاده که باعث می شود برنامه برای مدتی طولانی یا نامعلوم متوقف بماند. اگر به نتیجه فراخوانی متد در همان لحظه نیازی باشد و برنامه بتواند با فرض اتمام اجرای آن متد ادامه یابد می توان با استفاده از **Event/Listener** برنامه را به گونه ای طراحی کرد که آن متد فراخوانی شود و بدون اینکه اجرای آن متد به اتمام برسد برنامه ادامه یابد، سپس زمانیکه متد اجرا شود و به اتمام برسد با استفاده از یک آبجکت `Listener` می توان عکس العمل برنامه نسبت به نتیجه اجرای متد را پیاده سازی کرد. به این نوع از فراخوانی یک متد، فراخوانی غیرهمزمان گفته می شود.

در عمل برای افزایش انعطاف پذیری طراحی Event/Listener. تغییراتی در آن داده می شود که در ادامه به آن می پردازیم.

اولین تغییر مربوط به کلاس Listener می شود. معمولا به جای کلاس Listener از اینترفیس استفاده می شود.

```

1 package ir.atlassoft.javase.chapter23;
2
3 public interface Listener{
4     //continue program with user's selected item
5     public void process(Event event);
6 }
```

کد ۱

با این تغییر کلاس Interactive که منبع حادثه است به جای اینکه فیلدی از جنس کلاس داشته باشد فیلدی از اینترفیس SelectionListener به صورت زیر دارد.

```

1 package ir.atlassoft.javase.chapter23;
2
3 public class Interactive2{
4
5     Listener listener;
6
7     public void setListener(Listener listener) {
8         this.listener=listener;
9     }
10
11    public void select(int option) {
12        Event event = new Event(option);
13        listener.process(event);
14    }
15
16    public void showOptions() {
17        //show options to user
18    }
19 }
```

کد ۲

Lambda

با این تغییر، کلاس Interactive می‌تواند با هر کلاسی که اینترفیس Listener را پیاده سازی کند کار کند و این موضوع وابستگی کلاس Interactive به یک پیاده سازی خاص از بین می‌برد.

با تغییرات فوق، می‌توان کلاس‌های Listener مختلفی پیاده سازی کرد که هر کدام عکس العمل متفاوتی نسبت به انتخاب گزینه‌ها از خود نشان می‌دهند. به عنوان مثال کلاس Listener زیر صرفا انتخاب کاربر را در کنسول برنامه نمایش می‌دهد.

```
1 package ir.atlassoft.javase.chapter23;
2
3 public class LogListener implements Listener{
4     public void process(Event event){
5         System.out.println("Optoin "+
6             event.getOption() +
7             " has selected");
8     }
9 }
```

۲۳-۱۰ کد

اما کلاس Listener زیر عملکرد متفاوتی از خود نشان می‌دهد.

```
1 package ir.atlassoft.javase.chapter23;
2
3 public class DifferentListener implements Listener{
4     public void process(Event event){
5         //do some different operations!
6     }
7 }
```

۲۳-۱۱ کد

از آنجاییکه کلاس Interactive فقط به اینترفیس Listener وابستگی دارد هر کدام از کلاس‌های زیر را می‌توان به عنوان Listener به آن معرفی کرد.

```
Interactive2 interactive = new Interactive2();
LogListener listener = new LogListener();
interactive.setListener(listener);
interactive.showOptions();
```

برای اینکه کلاس Interactive بیشتر انعطاف پذیر شود، فیلد listener در این کلاس را با یک لیست از Listener ها به صورت زیر تغییر می دهیم.

```

1 package ir.atlassoft.javase.chapter23;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Interactive3{
7
8     List<Listener> listeners = new ArrayList<Listener>();
9
10    public void addListener(Listener listener) {
11        this.listeners.add(listener);
12    }
13
14    public void select(int option) {
15        Event event = new Event(option);
16        for(Listener listener:listeners) {
17            listener.process(event);
18        }
19    }
20
21    public void showOptions() {
22        //show options to user
23    }
24 }
```

د ۲۳-۱۲

با این تغییر، این امکان بوجود می آید تا وقتی حدثی رخ می دهد (کاربر یک گزینه را انتخاب می کند) به جای یک آبجکت Listener مجموعه ای از آبجکتهای Listener که هر کدام ممکن است پیاده سازیهای مختلفی داشته باشند فراخوانی شوند.

```

Interactive3 interactive = new Interactive3();

Listener listener1 = new LogListener();
Listener listener2 = new DifferentListener();

interactive.addListener(listener1);
interactive.addListener(listener2);
```

Lambda

```
interactive.showOptions();
```

توجه کنید که کلاس جدید Interactive، نام متدهای `setListener()` و `addListener()` را تغییر داده شده است. این کاملاً منطقی است زیرا وقتی یک متدهای فیلد را کاملاً با یک مقدار جدید مقداردهی کند نام آن با `set` شروع می‌شود و این متدهای قبلاً فیلد `listener` را مقداردهی می‌کرد، اما در متدهای جدید `listener` به لیست اضافه می‌شود و کل لیست یکجا مقداردهی نمی‌شود به همین دلیل نام `addListener()` مناسب‌تر است.

دقت کنید که استفاده از کلاس‌ی `Listener` برای نام برای پیاده‌سازی کلاس‌های `Interactive` بسیار مناسب است، زیرا این کلاس‌ها بسیار متنوع هستند (در یک برنامه ممکن است به تعداد زیاد کلاس‌های `Listener` داشته باشید)، ثانیاً از هر کلاس صرفاً یک آجکت ساخته می‌شود. با بکارگیری امکان کلاس‌بی نام، کد فوق به صورت زیر درمی‌آید.

```
Interactive3 interactive = new Interactive3();

Listener listener1 = new Listener() {
    public void process(Event event) {
        System.out.println("Optoin "+event.getOption()+" has
selected");
    }
};

Listener listener2 = new Listener() {
    public void process(Event event) {
        //do some different operations!
    }
};

interactive.addListener(listener1);
interactive.addListener(listener2);

interactive.showOptions();
```

کد فوق را می‌توان به شکل ساده‌تر زیر در آورد.

```
Interactive3 interactive = new Interactive3();

interactive.addListener(new Listener() {
    public void process(Event event) {
        System.out.println("Optoin "+event.getOption()+" has
selected");
    }
});
interactive.addListener(new Listener() {
    public void process(Event event) {
        //do some different operations!
    }
});
```

```

});  

interactive.showOptions();

```

عبارت‌های لامبدا (Lambda)

مسئله‌ای که در پیاده سازی کلاس‌های بی نام وجود دارد این است که اگر کلاس بی نام بسیار ساده باشد (مشابه اینترفیس Listener که در مثال قبل ملاحظه کردید) پیاده سازی کلاس بی نام کمی عجیب و نامفهوم به نظر می‌رسد! تکه کد زیر، ایجاد یک کلاس بی نام از روی اینترفیس Listener را نشان می‌دهد.

```

interactive.addListener(new Listener() {
    public void process(Event event) {
        System.out.println("Optoin "+event.getOption()+" has
selected");
    }
});

```

در اینجا در پیاده سازی اینترفیس Listener چه چیزی اهمیت دارد؟ آیا ذکر نام اینترفیس Listener نام متدهای void process() یا مقدار برگشتی این متدهای اهمیتی دارند؟ واضح است که جواب این سوالها منفی است، آنچه اهمیت دارد نام پارامتر متدهای process() (حتی جنس پارامتر که کلاس Event است هم اهمیت ندارد) و جمله

```
System.out.println("Optoin "+event.getOption()+" has selected");
```

که اهمیت دارند. بر همین مبنای فوق را می‌توان به صورت زیر نیز نوشت

```

interactive.addListener(
    event -> System.out.println("Optoin "+event.getOption()+" has
selected"));
}

```

به عبارتی که جایگزین کلاس بی نام شده است عبارت لامبدا گفته می‌شود که در تعریف کلاس بی نام وجود دارد از آن حذف شده است. براساس این عبارت، با ورودی پارامتر event که تنها پارامتر از تنها متدهای کلاس است جمله مقابل <- باید اجرا شود.

در ادامه با ارایه یک مثال ساده، جزئیات بیشتری از عبارتهای لامبدا را تشریح خواهیم کرد. فرض کنید که نرم افزار یک شبکه اجتماعی را پیاده سازی کرده اید. در این برنامه کلاس User را که معرف کاربران شبکه اجتماعی است را به صورت زیر پیاده سازی کرده اید.

1	package ir.atlassoft.javase.chapter23;
2	
3	import java.util.Date;

```

4
5  public class User {
6
7      public enum Sex {
8          MALE, FEMALE
9      }
10
11     String name;
12     Date birthday;
13     Sex gender;
14     String emailAddress;
15     int age;
16
17     //getters/setters
18
19     public void printDetails() {
20         // ...
21     }
22 }
```

۲۳-۱۴

حال تصور کنید که از شما خواسته شده است که کدی بنویسید که بر اساس معیار یا معیارهای مشخصی، کاربران شبکه اجتماعی را انتخاب کند. ممکن است در جای دیگری از برنامه به کاربران انتخاب شده ایمیل ارسال شود، اعلان (notification) ارسال شود،... آنچه موضوع این مثال است، انتخاب کاربران براساس معیارهای مشخص است.

برای پیاده سازی این نیازمندی، تصور کنید که تمام کاربران شبکه اجتماعی در قالب یک آجکت وجود داشته باشد، برای انتخاب کاربران مثلاً براساس سن باید متده مشابه متده `printUser()` به صورت زیر پیاده سازی کنیم.

```

public static void printUsers(List<User> users, int age) {
    for (User u : users) {
        if (u.getAge() >= age) {
            u.printDetails();
        }
    }
}
```

این متده دو پارامتر که یکی لیست کاربران و دیگری حداقل سن آنان است را دریافت می کند و طی یک حلقه با مقایسه سن هر کاربر با حداقل سن مشخص شده، وی را انتخاب می کند. واضح است که کد باید به گونه ای نوشته شود که تحت تاثیر تغییر معیارهای انتخاب کاربر قرار نگیرد، به عبارت دیگر اگر معیارهای

انتخاب یک کاربر تغییر کند و مثلاً علاوه بر حداقل سن، حداکثر سن نیز لحاظ شود انتظار داریم که متده `printUsers()` دست نخورده باقی بماند زیرا مسئولیت متده `printUsers()` معيارهای انتخاب کاربران نیست بلکه صرفاً انتخاب کاربران براساس معيارهایی است که برای آن تعیی می کنیم. به این منظور اینترفیس `CheckUser` را که حاوی یک متده `test()` است به صورت زیر تعریف می کنیم.

```

1 package ir.atlassoft.javase.chapter23;
2
3 public interface CheckUser {
4     boolean test(User u);
5 }
```

۲۳-۱۴ کد

متده `test()` در اینترفیس فوق، یک آبجکت `Person` دریافت می کند و با برگرداندن مقدار `true` یا `false` انتطاق آن کاربر با معيارها را اعلان می کند. به این ترتیب می توان پیاده سازیهای مختلفی از این اینترفیس مشخص کرد که هریک براساس ضابطه یا معيارهای خاصی کاربران را انتخاب می کند. به عنوان مثال کلاس زیر، کاربران را براساس جنسیت (مرد بودن) و حداقل سن ۱۸ و حداکثر سن ۲۵ انتخاب می کند.

```

1 package ir.atlassoft.javase.chapter23;
2
3 public class CheckUserByAgeRange implements CheckUser {
4     public boolean test(User u) {
5         return u.gender == User.Sex.MALE &&
6             u.getAge() >= 18 &&
7             u.getAge() <= 25;
8     }
9 }
```

۲۳-۱۵ کد

با وجود اینترفیس `CheckUser`، متده `printUsers()` به صورت زیر تغییر می کند.

```

public static void printUsers(
    List<User> users, CheckUser checker) {
    for (User u : users) {
        if (checker.test(u)) {
            u.printDetails();
        }
    }
}
```

Lambda

در پیاده سازی جدید متدها printUsers و CheckUser یک آبجکت از اینترفیس نیز به عنوان پارامتر ارسال شده تا به عنوان معیاری در انتخاب کاربران استفاده شود. حال برای فراخوانی این متدها باید کد مشابه کد زیر نوشته شود.

```
printUsers(users, new CheckUserByAgeRange());
```

در اینجا با کد تمیزتر و قابل فهم تری مواجه هستیم که البته بی هزینه نیز نبوده است. هزینه آن تولید اینترفیس CheckUserByAgeRange و کلاس CheckUser است. با نگاهی دقیق تر به کد فوق، متوجه می شویم که می توان از پیاده سازی کلاس CheckUserByAgeRange صرفنظر کرد و به جای آن از کلاس بی نام بهره برد.

```
printUsers(  
    users,  
    new CheckUser() {  
        public boolean test(User u) {  
            return u.getGender() == User.Sex.MALE  
                && u.getAge() >= 18  
                && u.getAge() <= 25;  
        }  
    }  
);
```

که در نتیجه کد ساده تری خواهیم داشت.

اینترفیس functional

در مثال فوق، یک اینترفیس به نام CheckUser پیاده سازی کردیم. قبل از آن نیز در توضیح کلاسهای بی نام اینترفیس Listener را پیاده سازی کردیم. هر دوی این اینترفیسها فقط یک متدهای دارند که اساس کارکرد آنها را تشکیل می دهند، به این اینترفیسها «اینترفیس‌های functional» گفته می شود. از آنجاییکه این اینترفیسها فقط یک متدهای پیاده سازی دارند، می توان با بکار نوشتند نام آن متدهای سازی کلاسهای بی نام آنها خودداری کرد. به این شکل از پیاده سازی کلاسهای بی نام که در آن نام متدهای اینترفیس آورده نمی شود و منجر به کد قابل فهم تر و ساده تری می شود عبارتهای لامبدا گفته می شود. بر همین اساس متدهای printUsers به شکل زیر در می آید.

```
printPersons(  
    users,  
    (User u) -> u.getGender() == User.Sex.MALE  
        && u.getAge() >= 18  
        && u.getAge() <= 25  
);
```

معمولًا از علامت `@FunctionalInterface` برای علامتگذاری این اینترفیسها استفاده می شود، به این ترتیب کامپایلر جاوا در هنگام کامپایل این اینترفیس از وجود فقط یک متدهای این اینترفیس اطمینان حاصل می کند و در غیراینصورت خطای کامپایل می دهد.

```

1 package ir.atlassoft.javase.chapter23;
2
3 @FunctionalInterface
4 public interface CheckUser {
5     boolean test(User u);
6 }

```

کد ۲۳-۱۶

یکی از نکات جالب در مورد اینترفیس‌های functional این است که مجموعه‌ای از اینترفیس‌های functional که کاربرد عمومی دارند در پکیج `java.util.function` پیاده سازی شده اند که می‌توانید در برنامه‌های خود استفاده کنید. به عنوان نمونه، اینترفیس `Predicate<T>` یکی از اینترفیس‌های است که به صورت زیر تعریف شده است

```

package java.util.function;

import java.util.Objects;

@FunctionalInterface
public interface Predicate<T> {

    boolean test(T t);
}

```

که می‌توانید به جای اینترفیس `CheckUser` که قبلاً پیاده سازی کرده اید کنید.

```

public static void printUsersWithPredicate(
    List<User> users, Predicate<User> checker) {
    for (User u : users) {
        if (checker.test(u)) {
            u.printDetails();
        }
    }
}

```

فراخوانی متدهای فوق بر مبنای عبارتهای لامبدا به صورت زیر خواهد بود.

```

printUsersWithPredicate(
    users,
    u -> u.getGender() == User.Sex.MALE
        && u.getAge() >= 18
        && u.getAge() <= 25
);

```

حال که با اصول عبارتهای لامبда آشنا شدید اجازه دهید با معرفی چند اینترفیس functional دیگر مثال خود را تکمیل کنیم.

تا اینجا کار در تمام پیاده سازی های متدهای `printUsers()` و قیمتی کاربری حائز شرایط برای انتخاب شدن بود متدهای `printDetails()` آن کاربر را فراخوانی کردیم. این در حالیست که مایلیم مشابه معیارهای

Lambda

انتخاب کاربر که توسط یک آبجکت جداگانه (آبجکتی از `Predicate<User>`) مشخص گردید، عملیات پس از انتخاب کاربر نیز توسط آبجکت جداگانه ای ارایه گردد. خوشبختانه مشابه اینترفیس `Consumer` نیز در جاوا ارایه شده است.

```
package java.util.function;

import java.util.Objects;

@FunctionalInterface
public interface Consumer<T> {

    void accept(T t);

}

 به این ترتیب، متد printUsers() به صورت زیر تغییر داد.

public static void processUsers(
    List<User> users,
    Predicate<User> checker,
    Consumer<User> consumer) {
    for (User u:users) {
        if (checker.test(u)) {
            consumer.accept(u);
        }
    }
}
```

فراخوانی متد فوق با عبارتهای لامبدا به صورت زیر خواهد بود.

```
processUsers(
    users,
    u -> u.getGender() == User.Sex.MALE
        && u.getAge() >= 18
        && u.getAge() <= 25,
    u -> u.printDetails()
);
```

دقت کنید که در پیاده سازی اینترفیس `Consumer` از عبارت لامبدا استفاده شده است.

حال تصور کنید که بخواهیم تغییری در کد فوق بوجود آوریم بگونه ای که پس از اینکه یک کاربر حائز شرایط بود، اطلاعات خاصی (مثلا ایمیل، شماره تلفن...) از پروفایل وی استخراج شود تا در مرحله آخر (یعنی انجام عملیات پس از انتخاب) مورد استفاده قرار گیرد. خوشبختانه برای این منظور نیز اینترفیس `Function<T, R>` در جاوا وجود دارد.

```
package java.util.function;

import java.util.Objects;

@FunctionalInterface
public interface Function<T, R> {
```

```
R apply(T t);  
}
```

متدهای apply() در این اینترفیس، آبجکتی از یک جنس را دریافت میکند و آبجکتی از یک جنس دیگر برمی‌گرداند. به این ترتیب متدهای apply() می‌تواند در مثال ما با دریافت یک آبجکت User پروفایل وی را واکاوی کند و براساس آنچه ما برای آن تعیین می‌کنیم یکی از جزییات پروفایل آن کاربر (متلاعه ایمیل، شماره تلفن، ...) را برگرداند.

با اضافه کردن اینترفیس Function به کد برنامه مان، متدهای printUsers() به صورت زیر در خواهد آمد.

```
public static void processUsersWithFunction(  
    List<User> users,  
    Predicate<User> checker,  
    Function<User, String> mapper,  
    Consumer<String> consumer) {  
    for (User u:users) {  
        if (checker.test(u)) {  
            String data = mapper.apply(p);  
            consumer.accept(data);  
        }  
    }  
}
```

فراخوانی متدهای فوق با بکارگیری عبارت لامبда به شکل زیر خواهد بود.

```
processUsersWithFunction(  
    users,  
    u -> u.getGender() == User.Sex.MALE  
        && u.getAge() >= 18  
        && u.getAge() <= 25,  
    u -> u.getEmailAddress(),  
    email -> System.out.println(email)  
);
```

تصویر کنید که متدهای processUsersWithFunction() را بخواهیم به شکل عمومی درآوریم به گونه‌ای که بتواند با هر آبجکت دیگری علاوه بر User نیز کار کند. در اینصورت با استفاده از آنچه در فصل Generics با آن آشنا شدید این متدهای شکل زیر در خواهد آمد.

```
public static <X, Y> void processElements(  
    Iterable<X> source,  
    Predicate<X> tester,  
    Function <X, Y> mapper,  
    Consumer<Y> block) {  
    for (X p : source) {  
        if (tester.test(p)) {  
            Y data = mapper.apply(p);  
            block.accept(data);  
        }  
    }  
}
```

فراخوانی متدهای فوق برای آبجکتهای User به صورت زیر خواهد بود.

```

processElements(
    users,
    u -> u.getGender() == User.Sex.MALE
        && u.getAge() >= 18
        && u.getAge() <= 25,
    u -> u.getEmailAddress(),
    email -> System.out.println(email)
);

```

کلاس Stream

پکیج `java.util.stream` یک اینترفیس کاربردی به نام `Stream` دارد. برخی متدهای این اینترفیس، آبجکتی از جنس همین اینترفیس (یعنی `Stream`) برمی گرددند که این خاصیت امکان می دهد تا با داشتن یک آبجکت `Stream` به صورت زنجیره ای بتوان متدهای آن آبجکت را فراخوانی کرد. خصوصیت جالب دیگر اینترفیس `Stream` این است که برخی متدهای آن آبجکتها را از جنس اینترفیسهای `functional` دریافت می کنند. نتیجه اینکه اگر بتوانیم از روی یک آبجکت `Stream` تولید کنیم می توانیم به صورت زنجیره وار متدهای مختلف آنرا با عبارتهای لامبدا فراخوانی کنیم. کد زیر کاربران که در لیست `users` قرار دارند را با معیارهایی که تا اینجا مشخص کرده ایم انتخاب می کند.

```

users
    .stream()
    .filter(
        u -> u.getGender() == User.Sex.MALE
            && u.getAge() >= 18
            && u.getAge() <= 25)
    .map(u -> u.getEmailAddress())
    .forEach(email -> System.out.println(email));

```

واضح است که استفاده از اینترفیس `Stream` باعث ساده تر و خواناتر شدن کد شده است. دقت کنید که ایجاد آبجکت `Stream` از روی لیست کاربران با فراخوانی متدهای `stream()` از آبجکت `List` در `java.util` امکانپذیر است.

متدهای پیش فرض و متدهای استاتیک

براساس آنچه در معرفی مفهوم اینترفیس آموختید، وقتی یک کلاس یک اینترفیس را پیاده سازی می کند باید تمام متدهای آن اینترفیس را نیز پیاده سازی کند یا به صورت `abstract` تعریف شود. مشکلی که در این نوع طراحی وجود دارد این است که اگر متدهای این اینترفیس را نیز پیاده سازی کنیم که آن اینترفیس را پیاده سازی کرده است نیز باید تغییر کند و آن متدهای این اینترفیس را نیز تغییرات محدود شود و یک تغییر در اینترفیس تاثیری در کلاس یا کلاسها را پیاده سازی کننده آن اینترفیس نداشته باشد مفهومی تحت عنوان «متدهای پیش فرض» معرفی شده است. براساس این مفهوم، اینترفیسها نیز می توانند دارای

متدهایی باشند که بدنه دارند، این متدها در کلاس‌هایی که اینترفیس را پیاده سازی می‌کنند به ارث برده می‌شود و قابل فراخوانی هستند. به این ترتیب، اگر متدهایی به یک اینترفیس اضافه شود، می‌توان برای آن متدهای بدنۀ پیش فرض پیاده سازی نمود و بدین ترتیب کلاس‌هایی که آن اینترفیس را پیاده سازی می‌کنند بدون اینکه نیاز باشد تا تغییر کنند و آن متدهای نیز پیاده سازی کنند، حاوی متدهای جدید نیز خواهند بود. برای اینکه کامپایلر جاوا بتواند این متدها را از متدهای بدون بدنه در اینترفیسها تفکیک کند از کلمه `default` در تعریف این متدها استفاده می‌شود.

به عنوان نمونه در اینترفیس `java.util.Collection` یک متدهای `removeIf` به صورت زیر اضافه شده است بدون اینکه نیاز باشد تا کلاس‌هایی که این اینترفیس را پیاده سازی می‌کنند این متدهای را پیاده سازی کنند.

```
default boolean removeIf(Predicate<? super E> filter) {
    Objects.requireNonNull(filter);
    boolean removed = false;
    final Iterator<E> each = iterator();
    while (each.hasNext()) {
        if (filter.test(each.next())) {
            each.remove();
            removed = true;
        }
    }
    return removed;
}
```

در این متدهای `Predicate` به عنوان پارامتر دریافت می‌شود تا تمام عناصر `Collection` که حائز شرایط مشخص شده در `Predicate` هستند از لیست حذف شوند.

قواعد متدهای پیش فرض

اگر یک کلاس، اینترفیسی را که دارای متدهای پیش فرض است پیاده سازی کند آن متدهای پیش فرض از ارث می‌برد و مثل تمام متدهایی که در آن کلاس یا کلاس‌های پدر آن کلاس پیاده سازی شده اند قابل فراخوانی است. اما اگر این کلاس، یک کلاس `abstract` باشد این گزینه را دارد تا متدهای پیش فرضی که از اینترفیس بدست آمده را به صورت `abstract` تغییر دهد تا کلاس‌های فرزند خود مجبور به پیاده سازی آن شوند.

اگر یک اینترفیس، اینترفیس دیگری را که دارای متدهای پیش فرض است را توسعه دهد، متدهای پیش فرض را به ارث می‌برد و در تمام کلاس‌هایی که اینترفیس را دوم را پیاده سازی کنند قابل دسترس خواهد بود. اما اینترفیس دوم، این گزینه را دارد تا متدهای پیش فرضی را که به ارث برده، به صورت یک متدهای معمولی اینترفیس تغییر دهد که در اینصورت تمام کلاس‌هایی که اینترفیس دوم را پیاده سازی می‌کنند می‌باشد متدهای پیش فرض را نیز پیاده سازی کنند.

اگر دو اینترفیس که دارای متدهای پیش فرضی باشند که تعریف یکسانی داشته باشند، کلاسی که دو اینترفیس را پیاده سازی می کند می بایست الزاماً متدهای پیش فرض را پیاده سازی کند در غیراینصورت کامپایلر جاوا خطای کامپایل تولید می کند زیرا اینکه کدام یک از پیاده سازیهای متدهای کلاس قابل فراخوانی هستند مهم خواهد بود. اما اگر دو اینترفیس دارای متدهای تعریف شده باشند، کلاسی که دو اینترفیس به صورت پیش فرض و در دیگری به صورت معمول تعریف شده باشند، کلاسی که دو اینترفیس را پیاده سازی می کند می تواند متدهای را پیاده سازی نکند و دراینصورت متدهای پیش فرض را به ارث می برد یا اینکه آنرا پیاده سازی کند و دراینصورت آنرا `override` کند.

متدهای استاتیک

علاوه بر متدهای پیش فرض، اینترفیسها می توانند متدهای استاتیک هم داشته باشند. از آنجاییکه از روی اینترفیسها آبجکتی ساخته نمی شود، امکان پیاده سازی متدهای استاتیک که اساساً وابسته به یک آبجکت نیستند منطقی به نظر می رسد. وجود متدهای استاتیک در اینترفیس امکان می دهد تا متدهای کمکی برنامه را به جای یک کلاس در اینترفیس ها سازماندهی کنیم. متدهای استاتیک اینترفیس، مشابه متدهای استاتیک کلاس با پیشوند `static` تعریف می شوند.

توجه: اینترفیس های Functional می توانند علاوه بر تنها متدهای خود، متدهای پیش فرض یا متدهای استاتیک داشته باشند.

تمرینات

- (۱) یک کلاس Contact پیاده سازی کنید که از دو آدرس (منزل و محل کار)، سه شماره تلفن (شماره همراه، منزل و محل کار)، و یک (یا بیشتر) ایمیل تشکیل شده باشد. کلاسهای آدرس (Address)، تلفن (Phone) و ایمیل (Email) را به صورت کلاس داخلی پیاده سازی کنید.
- (۲) کلاس University را تعریف کنید که از کلاسهای استاتیک داخلی آزمایشگاه (Laboratory)، دانشکده (Department)، و کتابخانه (Library) تشکیل شده است.
- (۳) متدهای بنویسید که یک List از آبجکتهای Contact (که در تمرین اول پیاده سازی کردید) و یک آبجکت `Predicate<Contact>` را دریافت می کند و تمامی آبجکتهای Contact که منطبق بر `Predicate` هستند را بر می گرداند. سپس متدهای `shard` و `getShardCount` را با استفاده از یک عبارت لامبدا فراخوانی کنید به گونه ای که تمام Contact هایی که شماره تلفن آنها با یک پیش شماره (مثل "۰۲۱") شروع می شوند را برگرداند.
- (۴) اینترفیس Rangeable را پیاده سازی کنید که یک محدوده عددی (یک مقدار سقف و یک مقدار کف) را مشخص می کند. این اینترفیس می بایست یک متدهای `default` داشته باشد که فاصله سقف و کف را به `int` بر می گرداند. این اینترفیس می بایست به صورت `generic` طراحی شود به گونه ای که مقدار سقف و کف یک عدد باشد.
- (۵) به اینترفیس Rangeable که در تمرین ۴ پیاده سازی کردید یک متدهای `overlap` را اضافه کنید که محدوده مشترک پارامتر `r` را با آبجکت `Rangeable` فعلی برگرداند.
- (۶) به اینترفیس Rangeable که در تمرین ۴ پیاده سازی کردید یک متدهای `static overlap` را اضافه کنید که محدوده مشترک دو پارامتر `r1` و `r2` را با برگرداند.

۲۴

Reflection

در برخی برنامه ها ممکن است به «واکاوی کلاسها در زمان اجرا» یا «تغییر کلاسها یا آبجکتهاي برنامه در زمان اجرا» نياز داشته باشيد. اجازه دهيد قبل از ادامه بحث اين فصل، تكليف خود را با عبارت «در زمان اجرا» مشخص کنيم!

چرخه حيات يك نرم افزار از زمان شروع به توسعه (کدنويسى) شروع مى شود و با فعالitehای دیگری از قبيل کامپایل، دبیاگ، تست، بسته بندی (packaging)، استقرار (deployment) ادامه مى يابد و در انتها به «زمان اجرا» یعنی زمانی که برنامه اجرا مى شود ختم مى شود. در بين همه اين فعالitehها، زمان توسعه (کدنويسى) و زمان اجرا برای برنامه نويسان اهميت بيشتری دارد زира در اين دو مرحله است که نيازمنديهاي برنامه پياده سازی و اجرا مى شوند. با اين تعریف از «زمان اجرا» اکنون مى توانيم تعریف دقیق تری در مورد Reflection ارایه دهیم.

«با استفاده از Reflection می توانيم در زمان اجرا جزئيات کلاسهاي برنامه را واکاوی، تغییر، يا فراخوانی کنيم»

به عنوان مثال متدى را تصویر کنيد که نام يك کلاس (نام هر کلاسي که در برنامه وجود دارد) را دریافت مى کند و نام فیلد های آن کلاس را برمی گردد.

```
public String[] getFieldNames(String className) {  
    ...//returning fields of className  
}
```

بسته به اینکه در زمان اجرا نام چه کلاس یا کلاس‌هایی به این متده شود، انتظار داریم نام فیلدهای آنرا برگرداند. به عنوان نمونه ای دیگر متده زیر را تصور کنید که یک آبجکت و نام یکی از متدهای آن آبجکت را دریافت می‌کند و آن متده را فراخوانی می‌کند.

```
public void call(Object obj, String method) {
    ...//calling method of obj
}
```

در زمان اجرا، هر آبجکتی و نام هر متده از آن آبجکت ممکن است به متده `call()` داده شود تا فراخوانی شود.

پیاده سازی چنین نیازمندیهایی تنها با استفاده از **Reflection** امکانپذیر است که امکان می‌دهد تا در زمان اجرا به جزییات اجزای برنامه (از قبیل کلاسها، اینترفیسها، متدها,...) دسترسی پیدا کنیم، یا به مقدار یک فیلد از یک آبجکت دسترسی پیدا کنیم، آنرا تغییر دهیم یا متده را از یک آبجکت فراخوانی کنیم.

Reflection حتی امکانات بیشتری به ما می‌دهد، مثلاً می‌توانیم فیلد یا متده را در زمان اجرا به یک کلاس اضافه کنیم! در این فصل قصد ندارم که شما را در جزییات مفصل **Reflection** غرق کنم، اما شما را با مفاهیم پایه و چهارچوب **Reflection** آشنا می‌کنم تا در صورت نیاز بتوانید با مطالعه منابع اصلی، جزییات بیشتری را بیاموزید.

حال که کلیت مفهوم **Reflection** را متوجه شده اید اجازه دهید تا چند مثال عملی از کاربرد **Reflection** برایتان مطرح کنم.

کلاس

در جاوا کلاسی با نام `Class` در پکیج `java.lang` به صورت زیر تعریف شده است.

```
package java.lang;

public final class Class<T> implements java.io.Serializable,
    GenericDeclaration,
    Type,
    AnnotatedElement {
    ...
}
```

دقت کنید که نام این کلاس `Class` است که با حرف بزرگ انگلیسی شروع می‌شود و نباید آنرا با که با حرف کوچک انگلیسی شروع می‌شود و یک کلمه کلیدی است اشتباه نگیرید.

ممکن است کنجکاو شوید که این کلاس عجیب و غریب چه معنی و کاربردی دارد؟ همانطور که در طراحی شی گرایی آموختید، وقتی مجموعه ای از اشیای مشابه وجود دارند کلاسی می‌توان تعریف کرد و بالعکس هر کلاس مجموعه ای از آبجکتهای مشابه را توصیف می‌کند. نکته جالب این است که تمام کلاس‌های جاوا نیز در

Reflection

واقع موجوداتی مشابه اند! همگی یک نام دارند، در یک پکیج تعریف شده اند، ممکن است از یک کلاس پدر مشتق شده باشند و اینترفیس هایی را پیاده سازی کرده باشند، سازنده دارند، و مجموعه ای از متدها و فیلدها را دارند. بر همین مبنای جاوا کلاس Class در پکیج java.lang تعریف شده است.

باید توجه کنید که برخلاف معمول که از کلاسها آبجکت می سازیم، هیچگاه از کلاس Class آبجکتی ایجاد نمی کنیم. ماشین مجازی جاوا در «زمان اجرا» متناظر با هر کلاسی که در برنامه بارگذاری می شود، آبجکت آن کلاس را ایجاد می کند که با فراخوانی متدهای getClass() از هر آبجکت می توان به آبجکت کلاس آن آبجکت دسترسی پیدا نمود. علاوه بر این، روشهای دیگری نیز برای دسترسی به آبجکت Class وجود دارند که همگی در جدول زیر نشان داده شده اند.

مثال	روش دسترسی
Class c = obj.getClass();	با فراخوانی متدهای getClass() از یک کلاس
Class c = String.class;	با دسترسی به فیلد استاتیک class
Class c = Class.forName("java.lang.String");	با فراخوانی متدهای استاتیک Class.forName()

جدول ۱-۲۴. روشهای دسترسی به آبجکت Class

در ادامه هریک از این روشهای تشریح شده اند.

بالاستفاده از آبجکتی از یک کلاس

با داشتن یک آبجکت obj، متدهای getClass() و getName() به صورت زیر برمی گرداند:

```
Class c = obj.getClass();
```

آبجکت c امکان می دهد به جزئیات کلاس آبجکت obj (نام کلاس، نام پکیج، سازنده ها، متدها، فیلدها، کلاس پدر، اینترفیس‌های پیاده سازی شده و همه جزئیات دیگر کلاسی که آن آبجکت از آن ساخته شده است) دسترسی پیدا کنیم.

```
Object obj = "Iran is a strong country";
Class c = obj.getClass();
System.out.println("Class:" + obj.getName());
System.out.println("Package:" + obj.getPackage().getName());
```

انتظار داریم که با اجرای تکه کد فوق، جمله زیر را در کنسول برنامه مشاهده کنیم.

```
Class:java.lang.String
Package:java.lang
```

اگر آبجکت obj از کلاس دیگری (مثلاً از کلاس Date) ساخته شده باشد، انتظار داریم خروجی برنامه به صورت زیر باشد

```
Class:java.util.Date
Package:java.util
```

با دسترسی به فیلد استاتیک class

همانطور که گفته شد، متناظر با هر کلاسی که در برنامه بارگذاری می شود یک آبجکت Class ساخته می شود که با داشتن یک آبجکت از آن کلاس و با فراخوانی متد () از آن آبجکت می توان به آن دسترسی پیدا کرد. یک روش دیگر برای دسترسی به این آبجکت دسترسی به فیلد استاتیک class است که در تمام کلاسهای جاوا وجود دارند.

```
Class c1 = String.class;
Class c2=Date.class;
```

در اینجا بدون داشتن هیچ آبجکتی از کلاسهای Date و String به آبجکتهای کلاس آنها دسترسی پیدا کرده ایم. در این روش می توان به کلاس داده های اولیه (که با استفاده از () امکانپذیر نیست) نیز دسترسی پیدا نمود

```
boolean b;
Class c = b.getClass(); // compile-time error

Class c = boolean.class; // correct
```

با فراخوانی متد استاتیک Class.forName()

روش سوم برای دسترسی به آبجکت Class، استفاده از متد استاتیک () است که البته برای داده های اولیه امکانپذیر نیست.

```
Class c1 = Class.forName("java.lang.String");
Class c2 = Class.forName("java.util.Date");
```

واکاوی جزئیات کلاس با استفاده از آبجکت Class

حال که به آبجکت Class دسترسی یافته ایم می توانیم جزئیات کلاس را واکاوی کنیم و یا حتی آنها را دستکاری کنیم. در ادامه نحوه واکاوی کلاس و دستیابی «کلاس های مرتبط»، «متدهای کلاس»، «فیلدهای کلاس» و «سازنده های کلاس» خواهیم پرداخت.

کلاسهای مرتبط

منظور از کلاسهای مرتبط، کلاسهایی هستند که به هر ترتیبی با یک کلاس ارتباط دارند. این کلاسهای شامل کلاس یا کلاسهایی پدر، کلاسهای داخلی (Inner Classes)، و کلاسهای داخلی کلاسهای پدر می باشند.

نام متد	توضیح
Class.getSuperclass()	کلاس پدر یک کلاس را بر می گرداند.
Class.getClasses()	تمام کلاسها، اینترفیسها، و enum های داخلی تعریف شده

Reflection

در کلاس و یا در کلاس‌های پدر را برمی‌گرداند.	
تمام کلاس‌ها، اینترفیس‌ها، و enum های داخلی تعریف شده در کلاس را برمی‌گرداند.	Class.getDeclaredClasses()
کلاس بیرونی که در برگیرنده یک کلاس داخلی است را برمی‌گرداند.	Class.getEnclosingClass()

جدول ۲۴-۳. دسترسی به کلاس‌های مرتبط

۲۴-۴. Modifier

Modifier ها که در زیر لیست شده اند، پیشوندهایی هستند که در تعریف کلاس بکار بردہ می‌شوند و محدوده و رفتار کلاس در زمان اجرا را مشخص می‌کنند.

Modifier	گروه
public، private و protected	کنترل دسترسی
abstract	کلاس انتزاعی
static	یک نمونه
final	نهایی
annotations	علامتها

جدول ۲۴-۴. گروه بندی Modifier ها

هریک از سازنده‌ها، متدها و فیلدهای کلاس، Modifier های خودشان را دارند که به جز Annotation ها، به بقیه Modifier ها با می‌توان با فراخوانی متدهای getModifiers() دسترسی پیدا کرد. این متدهای آرایه ای از آبجکتهای java.lang.reflect.Modifier برمی‌گرداند.

```
Modifier[] modifiers = c.getModifiers();
```

کلاس Modifier مجموعه ای از متدهای static دارد که با استفاده از آنها می‌توان وضعیت یک متدهای کلاس، فیلد،... را به لحاظ وجود یا عدم وجود یک Modifier خاص بررسی کرد.

```
Modifier.isStatic(c.getModifiers())
Modifier.isPublic(c.getModifiers())
Modifier.isFinal(c.getModifiers())
Modifier.isTransient(c.getModifiers())
```

مثالهای فوق، مثالهایی از فراخوانی متدهای مختلف کلاس Modifier برای بررسی وضعیت Modifier های یک کلاس یا متدهای است.

متدهای هویتی

این متدها امکان می دهند که تا هویت آبجکت Class را شناسایی کنیم. به عنوان مثال، اینکه یک آبجکت Class معرف یک اینترفیس، annotation یا آرایه است. در زیر این متدها آورده شده اند.

```
public boolean isEnum()
public boolean isInterface()
public boolean isPrimitive()
public boolean isAnnotation()
public boolean isArray()
```

نام این متدها به وضوح کارکرد آنها را نشان می دهند.

دسترسی به اجزای داخلی کلاسها (فیلدها، متدها و سازنده ها)

در Reflection سه کلاس زیر تعریف شده اند

```
java.lang.reflect.Field
java.lang.reflect.Method
java.lang.reflect.Constructor
```

که به ترتیب معرف فیلد، متد و سازنده کلاس هستند. این کلاسها اینترفیس java.lang.reflect.Member را پیاده سازی کرده اند. برای دسترسی به تمام اجزای public کلاس، اعم از اینکه در همان کلاس یا کلاس‌های پدر تعریف شده باشند، متدهای زیر تعریف شده اند.

```
public Constructor[] getConstructors()
public Field[] getFields()
public Method[] getMethods()
public Class[] getClasses()
```

از آنجاییکه سازنده های کلاس به ارث برده نمی شوند، متد () getConstructors صرفا سازنده های public همان کلاس را برمی گرداند. دسترسی به تمام اجزای یک کلاس که به ارث برده نشده اند اما ممکن است private، public، protected باشد از متدهای زیر استفاده می شود.

```
public Constructor[] getDeclaredConstructors()
public Field[] getDeclaredFields()
public Method[] getDeclaredMethods()
public Class[] getDeclaredClasses()
```

همانطور که از تفاوت نام این مجموعه متدها و متدهای قبلی آنها مشخص است، کلمه Declared در نام آنها اضافه شده که دلالت بر اجزای تعریف شده در همان کلاس دارد. برای دسترسی به یک فیلد از یک کلاس که نام مشخصی دارد از متدهای زیر استفاده می شود.

Reflection

```
public Field getField(String name)
public Field getDeclaredField(String name)
```

متدهای `getField()` و `getDeclaredField()` فیلدهای با نام مشخص شده را که آن کلاس یا کلاسهای پدر تعریف شده است را برمی‌گردانند. در هر دو متدهای `get*` فیلدی با نام مشخص شده که مستقیماً در کلاس تعریف شده را برآورده می‌گرداند. در هر دو متدهای `get*` با آن نام وجود نداشته باشد `NoSuchFieldException` تولید می‌شود. به شکل مشابه برای دسترسی به متدهای تعریف شده در یک کلاس از متدهای زیر استفاده می‌شود.

```
public Method
    getMethod(String name, Class... parameterTypes)
public Method
    getDeclaredMethod(String name, Class... parameterTypes)
```

نام متدهای `name` و پارامترهای متدهای `parameterTypes` را مشخص می‌کنند. متدهای زیر، سازنده‌های کلاس را برمی‌گردانند.

```
public Constructor<T>
    getConstructor(Class... parameterTypes)
public Constructor<T>
    getDeclaredConstructor(Class... parameterTypes)
```

در اینجا `parameterTypes` جنس پارامترهای سازنده کلاس را مشخص می‌کنند. در حالیکه متدهای `getDeclaredConstructor()` کلاس را برمی‌گرداند. متدهای `getConstructor()` سازنده کلاس که ممکن است `protected`, `private`, `public` باشد را برمی‌گردانند. متدهای فوق، در صورتیکه متدهای `get*` با نام و پارامترهای مشخص شده وجود نداشته باشد، `NoSuchMethodException` تولید می‌کنند.

مثال کاربردی

تصور کنید که در یک برنامه کلاس `Book` به صورت زیر تعریف و پیاده‌سازی شده باشد.

```
1 package ir.atlassoft.javase.chapter24;
2
3 public class Book{
4     String title;
5     String subject;
6     String isbn;
7     String author;
8     //getters & setters
9 }
```

کد ۲۴-۱

حال تصور کنید که متن زیر که معرف داده های یک کتاب است از طریق شبکه برای برنامه ارسال شده است و برنامه باید آبجکت جاوای Book معادل آنرا در برنامه بازسازی کند. در کد ۲۴-۲ متدهای getBook() و main() پیاده سازی شده است که با دریافت رشته فوق، آبجکت Book معادل آنرا برمی گرداند.

```

1 package ir.atlassoft.javase.chapter24;
2
3 public class StringToBook {
4     public Book getBook(String input) {
5         Book book = null;
6         if(input!=null){
7             book = new Book();
8             String[] parts = input.split(",");
9             for(String part:parts){
10                 if(part.trim().startsWith("\"title\"")){
11                     book.setTitle(part.substring(9,
12                         part.length()-1));
13                 }else
14                 if(part.trim().startsWith("\"subject\"")){
15                     book.setSubject(part.substring(12,
16                         part.length()-1));
17                 }else if(part.trim().startsWith("\"isbn\"")){
18                     book.setIsbn(part.substring(9,
19                         part.length()-1));
20                 }else if(part.trim().startsWith("\"author\"")){
21                     book.setAuthor(part.substring(11,
22                         part.length()-1));
23                 }
24             }
25         }
26         return book;
27     }
28
29     public static void main(String[] args) {
30         StringToBook stb = new StringToBook();
31         String input = "\"title\":\"Java Programming\",
32         \"subject\":\"Computer\", \"isbn\":\"8131702219\",
33         \"author\":\"James Gosling\"";

```

Reflection

```
34     System.out.println(stb.getBook(input));  
35 }  
36 }
```

۲۴-۳ کد

اگر رشته ورودی همواره حاوی داده‌های یک کتاب باشد متده فوق به خوبی کار می‌کند، اما اگر این رشته در شرایطی آبجکتها بی‌از جنس کلاس‌های دیگر مثل Student یا City را ارایه کند و انتظار داشته باشیم بسته به رشته ورودی متده getObject() آبجکتها بی‌از جنس کلاس متناظر آنها برگرداند، متده فوق چه تغییر باید پذیرد؟

برای حل مساله، فرض کنید رشته‌های زیر که معرف داده‌های مختلفی هستند از شبکه دریافت شوند.

```
Book["title":" Java Programming", "subject":"Computer", "isbn":  
"8131702219", "author":"James Gosling"]
```

```
Student["firstName":"Ali", "lastName":"Kamranifar",  
"ssn":"8329282828"]
```

```
City["name":"Tehran", "Population":"12,000,000"]
```

در ابتدای هر رشته، نام کلاس آمده و بعد از آن بین علامت‌های [] نام فیلدها و مقادیر متناظر آنها ظاهر شده است، به فرمات فوق، JSON گفته می‌شود که در برنامه‌های وب رایج است و فریم ورکهای جاوا اسکرپت می‌توانند آنرا بخوانند و پردازش کنند. واضح است که متده getObject() باید به گونه‌ای تغییر کند تا به صورت پویا براساس رشته‌ای که دریافت می‌کند، آبجکتها بی‌از جنس کلاس‌های متفاوت تولید کند. اگرچه می‌توان این نیازمندی را بدون استفاده از Reflection نیز پیاده سازی نمود اما Reflection بهترین راه حل برای این مسایل است زیرا امکان می‌دهد تا به صورت کاملاً پویا و بدون اینکه به کلاس یا فیلدهای خاصی محدود شویم آبجکتها متناظر را بازسازی کنیم. اگر از Reflection استفاده کنیم، در آینده که رشته‌های جدیدی از جنس جدید اضافه شوند قسمتی از برنامه که کار تبدیل رشته به آبجکت را انجام می‌دهد تغییری نخواهد کرد.

پیاده سازی معکوس ستاربی فوق هم صرفاً با استفاده از Reflection امکان‌پذیر است. مثلاً تصور کنید به یک «سیستم لاغ» در برنامه نیاز داریم که می‌بایست آبجکتها بی‌از در اجرای فرایندهای مختلف برنامه ایجاد می‌شوند به فرمات متنی JSON در آورده و به عنوان یک رخداد در سیستم ثبت کند (در دیتابیس یا فایل ذخیره کند). از آنجاییکه آبجکتها بی‌از برای ثبت به سیستم لاغ ارسال می‌شوند، ممکن است از هر جنسی باشند، سیستم لاغ هیچ پیش ذهنیتی در مورد نام کلاس آنها و فیلدهای آنها ندارد و باید بتواند به صورت دینامیک نام فیلدها و مقادیر فیلدهای آبجکتها را یافته و از روی آنها یک رشته متنی JSON تهیی کند.

کاربردهای Reflection محدود به مثالهای فوق نیست، در واقع اغلب فریم ورکهای جاوای (که بعدا در مبحث Java Enterprise با آنها آشنا می شوید) براساس Reflection کار می کنند تا بتوانند با هر برنامه ای سازگار شوند. به جز فریم ورکهای دیباگرهای، ابزارهای تست، و محیط های توسعه (IDE) نیز همگی با بکارگیری خصوصیت Reflection کار می کنند تا مقدار متغیرها، فیلدها و هریک از اجزای آبجکتها و کلاسهای برنامه را در زمان اجرا و کاوی کنند.

اما با همه قدرتی که Reflection به جاوا آورده است، نقاط ضعفی هم در آن وجود دارد. مثلا کارایی Reflection در مقایسه با اجرای معمول پایین تراست زیرا زمانی که یک متده را به صورت معمول فراخوانی می کنیم ماشین مجازی جاوا بهینه سازی هایی انجام می دهد تا اجرای آن با کارایی بالاتری انجام شود در حالیکه در فراخوانی یک متده از طریق Reflection چنین بهینه سازی هایی وجود ندارد. همچنین باید توجه داشت که Reflection برخی قواعد معمول جاوا و شی گرایی را نقض می کند مثلا دسترسی به یک فیلد private و یا حتی تغییر مقدار آنرا اجازه می دهد. با همه اینها، همانطور که گفته شد پیاده سازی برخی نیازمندیها صرفا از طریق Reflection امکانپذیر است.

تمرینات

- (۱) با استفاده از Reflection لیست تمام متدها، فیلدها، سازنده های کلاس `java.lang.Integer` را در کنسول برنامه چاپ کنید
- (۲) متدهی بنویسید که نام یک کلاس را به صورت یک رشته دریافت می کند و تمام متدهای `public static` آن کلاس را برمی گرداند. (توضیح: مثلاً نام کلاس را به صورت رشته `“java.lang.Integer”` دریافت می کند و `Method[]` را برمی گرداند.)
- (۳) متدهی بنویسید که یک آبجکت (`Object obj`) و نام یک فیلد از آن آبجکت (`String field`) را دریافت می کند و مقدار آن فیلد از آن آبجکت را برمی گرداند.
- (۴) با استفاده از Reflection یک کلاس را در زمان اجرا `extend` کنید و تمام متدهای `getter` آنرا `override` کنید و در آنها متدهای `getter` پدرشان را فراخوانی کنید.

۲۸

چندزبانی (Internationalization)

شاید «چندزبانی» ترجمه مناسبی برای کلمه **internationalization** باشد اما مatasفانه معادل بهتری که بتواند در زبان فارسی به درستی و واضح مفهوم **internationalization** را نشان دهد نیافتد. در هر صورت، منظور از چندزبانی، طراحی یک برنامه به گونه ایست که به سادگی و با حداقل تغییرات بتواند با زبانهای مختلف سازگار شود. اولین و اصلی ترین تفاوتی که در زبانهای مختلف وجود دارد عبارات، متن ها و پیام هایی که در واسط کاربری برنامه وجود دارند. تفاوت دوم، راست چین یا چپ چین بودن آنهاست، مثلا انگلیسی، فرانسوی و زبانهای لاتین از چپ به راست نوشته می شوند و فارسی، عربی، و سیاری از زبانهای شرقی از راست به چپ. تفاوت های دیگری هم در زبانها وجود دارد، مثلا وجود تقویم های مختلف، واحد های پولی مختلف، و موضوعاتی از این قبیل. هدف چندزبانی این است که بخش های اجرایی برنامه که برای همه زبانها یکسان است را از بخش های غیر اجرایی که در زبانهای مختلف متفاوت هستند جدا سازد. حال با سازماندهی کردن بخش های تغییر پذیر، تولید برنامه های چندزبانه را آسان و ساده کند.

اگر بخواهید پشتیبانی از یک زبان را به یک برنامه چندزبانه اضافه کنید، به این فرایند که عمدتاً ترجمه متنها و پیامهای «بومی سازی» یا **localization** گفته می شود. هر چقدر که اصول طراحی چندزبانی رعایت شده باشد، فرایند بومی سازی آسان و ساده تر خواهد بود.

بعضی اوقات به جای واژه چندزبانی از عبارت `i18n` استفاده می شود که از کلمه انگلیسی `internationalization` گرفته شده، آ و `n` ابتداء و انتهای این کلمه هستد که بین آنها هم ۱۸ کاراکتر وجود دارد.

برای اینکه در ک سریعی از مفهوم چندزبانی بدست آورید، به کد زیر توجه کنید.

```

1 package ir.atlassoft.javase.chapter25;
2
3 public class NotI18N {
4     static public void main(String[] args) {
5         System.out.println("Hello.");
6         System.out.println("How are you?");
7         System.out.println("Goodbye.");
8     }
9 }
```

کد ۲۵-۱

در این برنامه، سه پیام در کنسول برنامه نمایش داده می شود. حال تصور کنید که بخواهید همین پیام ها را به زبانهای محلی مردمانی که در آلمان یا فرانسه زندگی می کنند نمایش دهید، برای این منظور لازم است برنامه نویس، پیامهایی که در کدهای برنامه نوشته شده اند (`hardcode` شده اند) را به زبانهای مقصد ترجمه و جایگزین کند. ولی اینکار برنامه نویس نیست، زیرا برنامه نویس الزاما زبانهای مختلف را نمی داند و چرا باید در گیر ترجمه متنها و جایگزینی آنها شود؟! به نظر می رسد این برنامه باید چند زبانه شود! کد زیر برنامه فوق را که به صورت چند زبانه طراحی شده است نشان می دهد.

```

1 package ir.atlassoft.javase.chapter25;
2
3 import java.util.*;
4
5 public class I18NSample {
6
7     static public void main(String[] args) {
8
9         String language;
10        String country;
11
12        if (args.length != 2) {
13            language = new String("en");
```

```

14         country = new String("US");
15     } else {
16         language = new String(args[0]);
17         country = new String(args[1]);
18     }
19
20     Locale currentLocale;
21     ResourceBundle messages;
22
23     currentLocale = new Locale(language, country);
24
25     messages = ResourceBundle.getBundle("MessagesBundle",
26                                         currentLocale);
27     System.out.println(messages.getString("greetings"));
28     System.out.println(messages.getString("inquiry"));
29     System.out.println(messages.getString("farewell"));
30 }
31 }
```

۲۵-۴

همینطور که از کد مشخص است، پیام هایی که به زبان انگلیسی در کد فوق وجود داشتند حذف شده اند تا برنامه انعطاف‌پذیر شده و بسته به پارامترهایی که در زمان اجرا (نام زبان و نام کشور) به متدهای main ارسال می‌شوند زبان پیامها مشخص شود. بنابراین برای مشاهده اجرای برنامه فوق به زبان فرانسوی باید در خط فرمان دستور زیر اجرا و خروجی بعد از آن مشاهده شود:

```
% java I18NSample fr FR
Bonjour.
Comment allez-vous?
Au revoir.
```

به شکل مشابه، برای اجرا به زبان انگلیسی باید دستور زیر اجرا و خروجی آن مشاهده شود.

```
% java I18NSample en US
Hello.
How are you?
Goodbye.
```

دقت کنید که در اولین اجرا، پارامترهای fr و FR در خط فرمان مشخص شده اند که زبان فرانسوی (fr) و کشور فرانسه (FR) مشخص می‌کنند. به شکل مشابه، برای اجرا به زبان انگلیسی آمریکایی را مشخص می‌کنند.

قطعاً هنوز هم در مورد کد فوق ابهاماتی دارید، اجازه دهید قدم به قدم نحوه چندزبانه کردن برنامه فوق را تشریح کنیم.

مرحله اول، تولید فایل‌های ResourceBundle

فایل‌های ResourceBundle در واقع فایل‌های properties هستند. فایل‌های properties که پیش از این در فصل java.util با آنها آشنا شدید فایل‌های متنی هستند که هر خط آنها متنی به صورت کلید/مقدار است. در برنامه‌های چندزبانه، به ازای هر زبان که برنامه از آن پشتیبانی می‌کند یک فایل ResourceBundle وجود دارد که معادل های کلمات آن زبان را مشخص می‌کند. در مثال فوق نیز فایل properties با محتوای زیر وجود دارد که معادل انگلیسی کلمات را نگهداری می‌کند.

```
greetings = Hello
farewell = Goodbye
inquiry = How are you?
```

به شکل مشابه، فایل MessagesBundle_fr_FR.properties وجود دارد که معادل های فرانسوی کلمات را تعریف می‌کند

```
greetings = Bonjour.
farewell = Au revoir.
inquiry = Comment allez-vous?
```

با اینکار، متنها و پیامهایی که برنامه آنها را نمایش می‌دهد را از کدهای برنامه خارج کرده ایم تا دیگر برای تغییر زبان نیازمند تغییر کدهای برنامه و کامپایل مجدد برنامه نباشیم. در ضمن اگر در آینده بخواهیم پشتیبانی از یک زبان جدید را نیز به برنامه اضافه کنیم، بدون نیاز به تغییر کدهای برنامه فقط لازم است یک فایل MessagesBundle_xx_XX.properties به برنامه اضافه کنیم. وقت کنید که نام این فایل‌ها از چه قالبی پیروی می‌کند، نام این فایل‌ها از یک بخش ثابت تشکیل شده است که در این مثال (MessagesBundle) است، بخش دوم نام فایل از ترکیب نام زبان و نام کشور که با _ از یکدیگر جدا شده اند تشکیل شده است. Properties هم که پسوند فایل است. با این قانون، اگر بخواهیم پشتیبانی از زبان فارسی را نیز به برنامه اضافه کنیم می‌بایست فایلی با نام MessagesBundle_fa_IR.properties در کنار بقیه فایل‌های چندزبانی برنامه اضافه کنیم. (fa متناظر با زبان فارسی و IR متناظر با کشور ایران است). همچنین لازم است بدانید که فایلی که در نام آن نام کشور و زبان وجود ندارد (در این مثال فایل MessagesBundle.properties) که معادل های کلمات در زبان انگلیسی را تعریف می‌کند) زبان پیش فرض برنامه است.

مرحله دوم، ایجاد آبجکت Locale

هر زبانی که برنامه آنرا پشتیبانی می کند توسط آبجکتی از کلاس `java.util.Locale` نشان داده می شود در کد زیر آبجکتی از `Locale` ایجاد شده است که معرف زبان انگلیسی آمریکایی است

```
Locale aLocale = new Locale("en", "US");
```

به پارامترهای سازنده کلاس `Locale` توجه کنید که زبان () و نام کشور () را دریافت می کنند. به شکل مشابه، آبجکتهای زیر به ترتیب معرف زبان فرانسوی کانادایی و فرانسوی هستند.

```
Locale caLocale = new Locale("fr", "CA");
Locale frLocale = new Locale("fr", "FR");
```

در مثالی که در این فصل مطرح شد، نام زبان و نام کشور توسط پارامترهایی که در زمان اجرای برنامه و از طریق خط فرمان به برنامه ارسال می شوند مشخص می شوند تا برنامه بدون اینکه حتی یک خط `hardcode` شود کاملاً انعطاف پذیر باشد.

```
String language = new String(args[0]);
String country = new String(args[1]);
currentLocale = new Locale(language, country);
```

تا اینجا برنامه ما با ایجاد آبجکتی از کلاس `Locale` صرفاً زبانی که قرار است برنامه با آن کار کند را مشخص کرده ایم. برای اینکه برنامه به زبان مورد نظر کار کند لازم است آبجکت `Locale` را که ایجاد کردیم به آبجکتهای دیگر بدهیم. یکی از این آبجکتهای `ResourceBundle` است که کار آن بازیابی پیامها و متنها از فایلهای `properties` است که پیش از این آنها را ایجاد کردیم.

مرحله سوم، ایجاد آبجکت ResourceBundle

کلاس `java.util.ResourceBundle` معرف متن و پیام های قابل ترجمه است که پیش از این در فایلهای `properties` تعریف شدند. این آبجکت به صورت زیر ساخته می شود

```
 ResourceBundle messages =
 ResourceBundle.getBundle("MessagesBundle",
                           currentLocale);
```

همانطور که مشخص است، متدهای استاتیک `getBundle` از کلاس `ResourceBundle` برای ایجاد آبجکت `ResourceBundle` استفاده شده است، این متدهای دو پارامتر دریافت می کند اولی نام پایه فایل های `properties` است و دومی هم آبجکت `Locale` ای است که معرف زبان مورد نظر است. به این ترتیب می توان انتظار داشت که فایل `properties` متناظر بارگذاری شود.

مرحله چهارم، واکشی متن از آبجکت ResourceBundle

هر خط از فایلهای properties که در مرحله اول ایجاد شدند در واقع زوجهای کلید/مقدار هستند. برای نمایش مقدار متناظر با زبان انتخابی کاربر کافیست متده است `getString()` از آبجکت `ResourceBundle` به صورت زیر فراخوانی کنیم.

```
String msg1 = messages.getString("greetings");
```

کلمه `greetings` کلیدی است که در فایلهای properties معادل های مختلفی در زبانهای مختلف برای آن وجود دارد.

با انجام این چهار مرحله اکنون باید شیوه برنامه نویسی برنامه های چند زبانی را درک کرده باشید. برای چند زبانه کردن یک برنامه، باید لیستی از مواردی که در زبانهای مختلف برنامه متفاوت هستند تهیه کنید به عنوان نمونه پیامها، راهنمای آنلاین، صداها، رنگها، عکسها و آیکن ها، تاریخ و زمان، اعداد، واحد پولی، شماره تلفن، عنوان و لقب افراد، آدرس، و حتی چهارچوب صفحات می توانند مواردی باشند که در زبانهای مختلفی که توسط برنامه پشتیبانی میشوند متفاوت باشند. در ادامه نحوه برخورد با مسایل فوق بحث می شوند.

کلاس Locale

همانطور که در قسمت قبل هم دیدید، هر زبانی که توسط برنامه پشتیبانی می شود توسط یک آبجکت Locale تعریف می شود. یک زبان ممکن است فقط گوییش آن زبان باشد (مثلًا زبان فارسی، زبان انگلیسی، زبان عربی...). یا ممکن است گوییش آن زبان در یک کشور خاص باشد (مثلًا زبان فارسی ایرانی، زبان فارسی افغانی، یا زبان انگلیسی بریتانیایی، زبان انگلیسی آمریکایی...). یا حتی ممکن است گوییش آن زبان در منطقه خاص گغرافیایی از یک کشور خاص باشد (مثلًا زبان انگلیسی آمریکایی نیویورکی، یا زبان انگلیسی بریتانیایی لندنی...). اگر برنامه قرار باشد زبانهایی را با این درجه از جزئیات پشتیبانی کند، آبجکت Locale نیز می تواند آنها را از یکدیگر تفکیک کند. اینکه برنامه شما بخواهد بین زبان انگلیسی آمریکایی و انگلیسی بریتانیایی تفکیک قابل شود یا خیر (وموارد مشابه دیگر) کاملا بستگی به نظر شما و نرم افزار شما دارد. ممکن است در یک برنامه بین المللی با کاربران فراوان از سراسر دنیا نیاز ببینید که بین دو لهجه از زبان انگلیسی تفاوت قابل شوید و آن را در برنامه خود انعکاس دهید یا ممکن است این موضوع برای شما اهمیتی نداشته باشد و صرفاً زبان انگلیسی را پشتیبانی کنید.

در تعریف آبجکت Locale از کد دو حرفی زبانها و کد دو حرفی کشورها که به صورت استاندارد ISO639 و ISO3166) درآمده اند استفاده می شود. مثلا `fr`, `ar`, `en`, `fa`, به ترتیب معرف زبان فارسی، انگلیسی، عربی و فرانسوی هستند و نامهای `IR`, `UK`, `FR`, `AF` به ترتیب معرف کشورهای ایران، بریتانیا، افغانستان و فرانسه هستند. در مورد منطقه گغرافیایی چنین استانداردسازی وجود ندارد و انتخاب نام منطقه کاملا اختیاری است. جداول ۱ و ۲۵-۲ چند نمونه از کد زبانها و کشورها را نشان می دهند.

چند زبانی

توضیح	زبان	کد زبان
فارسی	Farsi	fa
	Dutch	de
انگلیسی	English	en
فرانسوی	French	fr
ژاپنی	Japanese	ja
کره ای	Korean	ko
چینی	Chinese	zh

جدول ۱-۲۵. نمونه کدهای زبان ها

نام کشور	کد کشور
Iran	IR
China	CN
Germany	DE
France	FR
India	IN
United States	US

جدول ۲-۲۵. نمونه کدهای کشورها

در جدول زیر مثالهایی را از آبجکتها Locale مشاهده می کنید که هر کدام معرف یک زبان متفاوت هستند.

آبجکت Locale	ResourceBundle	زبان
new Locale("en");	XXX_en.properties	انگلیسی
new Locale("en", "UK");	XXX_en_UK.properties	انگلیسی بریتانیایی
new Locale("en", "UK", "London");	XXX_en_UK_London.properties	انگلیسی بریتانیایی لندنی
new Locale("fa");	XXX_fa.properties	فارسی
new Locale("fa", "IR");	XXX_fa_IR.properties	فارسی ایرانی

جدول ۳-۲۵. مثالهایی از آبجکتها

دقت کنید که نام فایلهای ResourceBundle متناظر با زبانی که برنامه پشتیبانی می کند باید باشد، مثلاً زمانیکه برنامه زبان انگلیسی بریتانیایی لندنی را پشتیبانی می کند لازم است فایل XXX_en_UK.properties وجود داشته باشد.

دقت کنید که یک برنامه ممکن است همزمان با چندزبان مختلف کار کند و در زمان اجرا بین زبانهای مختلف جابجا شود، حتی ممکن است در زمان اجرا داده های خود را به دو یا چند زبان نشان دهد. در برنامه های تحت وب یا توزیع شده این موضوع جالبتر است زیرا همانطور که در مبحث برنامه نویسی تحت وب خواهد دید، هر کاربر ممکن است ترجیح دهد با یک زبان متفاوت کار کند که در اینصورت برنامه باید بتواند از آبجکتهای Locale متفاوت برای هر کاربر استفاده کند.

متنها و پیامها

در ابتدای این فصل، نحوه برخورد با متنها و پیامها را دیدید. متنهایی که باید ترجمه شوند شامل پیامهای وضعیت، پیامهای خطأ، متنهای لاغ، و متنهای واسطه کاربری است. در تمام این مسایل نحوه برخورد بکسان است و در همگی با تفکیک متنهای زبانها در فایلهای مختلف از آبجکت ResourceBundle برای بارگذاری آنها استفاده می شود.

فرمت دهی

بخشی از کار چندزبانه کردن یک برنامه، نمایش داده هایی است که در اصل در زبانهای مختلف یکسان هستند اما در نمایش با یکدیگر فرق دارند. به عنوان مثال، عدد ۱۲۳۴۵۶.۷۸ را تصور کنید که در زبان فرانسوی با فرمت ۱۲۳.۴۵۶,۷۸ نمایش داده می شود یا تاریخ اول آپریل ۱۹۹۸ در انگلیسی آمریکایی به شکل ۰۹-Apr-۹۸ و در زبان دانمارکی به صورت ۹۸.۴.۹۰.۰۹ نمایش داده میشود. اجازه دهید بحث در این قسمت را به دو بخش تقسیم کنیم، ابتدا فرمتهای آمده ای که در زبان جاوا وجود دارند را معرفی کنیم و سپس نحوه شخصی سازی و بکاربردن فرمتهای شخصی را بحث کنیم.

استفاده از فرمتهای آماده

کلاسهای NumberFormat و DateFormat به ترتیب برای فرمت دهی اعداد، واحد پولی، درصد و تاریخ و زمان مبتنی زبان وجود دارد. برای این منظور لازم است ابتدا با استفاده از آبجکت Locale از طریق متد استاتیک getInstance به صورت زیر به آبجکتی از NumberFormat دست بیابیم.

```
NumberFormat numberFormatter =
NumberFormat.getInstance(currentLocale);
```

حال با فراخوانی متر () به شکل زیر عدد مورد نظر را بسته به زبان انتخاب شده فرمت کنیم. به عنوان مثال، کد زیر بسته به اینکه چه زبانی انتخاب شده باشد خروجی های مختلفی خواهد داشت

چند زبانی

```
Double amount = new Double(345987.246);
NumberFormat numberFormatter;
String amountOut;

numberFormatter =
NumberFormat.getNumberInstance(currentLocale);
amountOut = numberFormatter.format(amount);
System.out.println(amountOut + " " +
currentLocale.toString());
```

خروجی این تکه کد با زبانهای مختلف در زیر نشان داده شده است.

345 987,246	fr_FR
345.987,246	de_DE
345,987.246	en_US

برای نمایش مبالغ پول نیز از همین کلاس NumberFormat استفاده می شود با این تفاوت که به جای متند `getCurrencyInstance` از متند `getNumberInstance` استفاده می شود.

```
NumberFormat
currencyFormatter=NumberFormat.getCurrencyInstance(currentLocale);
```

به شکل مشابه، برای نمایش یک مبلغ پولی لازم است متند `format()` از آبجکت `NumberFormat` را فراخوانی کرد. تکه کد زیر اینجا اینکار را نشان می دهد.

```
Double currency = new Double(9876543.21);
NumberFormat currencyFormatter;
String currencyOut;

currencyFormatter =
NumberFormat.getCurrencyInstance(currentLocale);
currencyOut = currencyFormatter.format(currency);
System.out.println(currencyOut + " " +
currentLocale.toString());
```

نتیجه اجرای کد فوق با سه زبان متفاوت در زیر نشان داده شده است.

9 876 543,21 F	fr_FR
9.876.543,21 DM	de_DE
\$9,876,543.21	en_US

دقیق کنید که در مثال فوق، صرفا نمایش یک مبلغ از واحدهای متفاوت نمایش داده شده است و تبدیلی بین واحدهای پولی انجام نشده است! در مورد نمایش «درصد» نیز کمابیش همین قاعده برقرار است. کد زیر اینجا اینکار را نشان می دهد.

```
Double percent = new Double(0.75);
NumberFormat percentFormatter;
```

```

String percentOut;

percentFormatter =
NumberFormat.getPercentInstance(currentLocale);
percentOut = percentFormatter.format(percent);

```

برای فرمت دهی تاریخ و زمان، کلاس مجزای `DateFormat` پیش بینی و طراحی شده است که برای ایجاد آبجکتی از آن باید متده استاتیک `getInstance()` را فراخوانی کنید. سپس با فراخوانی متده `format()` می توانید تاریخ را در فرمت دلخواه نمایش دهید.

```

Date today;
String dateOut;
DateFormat dateFormatter;

dateFormatter = DateFormat.getDateInstance(DateFormat.DEFAULT,
                                             currentLocale);
today = new Date();
dateOut = dateFormatter.format(today);

System.out.println(dateOut + " " + currentLocale.toString());

```

در زیر خروجی این کد را با سه زبان مختلف مشاهده می کنید.

9 avr 98	fr_FR
9.4.1998	de_DE
09-Apr-98	en_US

حتما متوجه شده اید که در فراخوانی متده `getDateInstance()` از یک پارامتر `Locale` علاوه بر `DateFormat.DEFAULT` نیز استفاده شده است. از آنجاییکه تاریخ در زبانهای مختلف می تواند به صورت مختصر، متوسط، بلند یا کامل نوشته شود این پارامتر جزییات نمایش تاریخ در یک زبان را مشخص می کند. مقادیر مختلفی که این پارامتر می توانند بگیرد همگی به صورت مقادیر ثابت در کلاس `DateFormat` تعریف شده اند و در زیر لیست شده اند.

- **DEFAULT**
- **SHORT**
- **MEDIUM**
- **LONG**
- **FULL**

جدول زیر نمایش تاریخ را با جزییات متفاوت در دو زبان انگلیسی آمریکایی و فرانسوی نشان می دهد.

Style	U.S. Locale	French Locale
DEFAULT	10-Apr-98	10 avr 98
SHORT	4/10/98	10/04/98
MEDIUM	10-Apr-98	10 avr 98
LONG	April 10, 1998	10 avril 1998
FULL	Friday, April 10, 1998	Vendredi, 10 avril 1998

جدول ۴. نمونه ای از قالب‌های تاریخ در Locale‌های فرانسه و آمریکا

از کلاس `DateFormat` همچنین برای قالبدهی زمان نیز استفاده می‌شود با این تفاوت که از متد `getFormat` برای ایجاد آبجکت `DateFormat` به صورت زیر استفاده می‌شود.

```
DateFormat timeFormatter =
    DateFormat.getTimeInstance(DateFormat.DEFAULT,
        currentLocale);
```

جدول زیر نمایش زمان را در دو زبان انگلیسی آمریکایی و آلمانی با جزییات مختلف نشان می‌دهد.

Style	U.S. Locale	German Locale
DEFAULT	3:58:45 PM	15:58:45
SHORT	3:58 PM	15:58
MEDIUM	3:58:45 PM	15:58:45
LONG	3:58:45 PMPDT	15:58:45 GMT + 02:00
FULL	3:58:45 oclock PM PDT	15:58:45 Uhr GMT+02:00

جدول ۵. نمونه ای از قالب‌های زمان در Locale‌های فرانسه و آمریکا

در بسیاری از موارد زمان همراه با تاریخ نمایش داده می‌شود که در این صورت برای فرمتدھی لازم است متد `getDateTimeInstance()` را به صورت زیر فراخوانی کنید.

```
DateFormat formatter =
    DateFormat.getDateInstance(DateFormat.LONG,
        DateFormat.LONG,
        currentLocale);
```

جدول زیر تاریخ و زمان را با جزییات مختلف در دو زبان انگلیسی و فرانسوی نشان می‌دهند.

Style	U.S. Locale	French Locale
DEFAULT	25-Jun-98 1:32:19 PM	25 jun 98 22:32:20
SHORT	6/25/98 1:32 PM	25/06/98 22:32
MEDIUM	25-Jun-98 1:32:19 PM	25 jun 98 22:32:20
LONG	June 25, 1998 1:32:19 PM PDT	25 juin 1998 22:32:20

		GMT+02:00
FULL	Thursday, June 25, 1998 1:32:19 o'clock PM PDT	jeudi, 25 juin 1998 22 h 32 GMT+02:00

جدول ۲۵-۶. نمونه ای از قالب‌های زمان و تاریخ در Locale های فرانسه و آمریکا

فرمت‌های دلخواه برای نمایش اعداد

در موقع زیادی ممکن است بخواهید یک عدد را با فرمات دلخواه خود نمایش دهید. مثلاً وقتی می‌خواهید صفحه‌ای قبل یا بعد از یک عدد را نمایش دهید (مثلاً ۰۰۸۹۳،۰۰ یا ۸۹۳،۰۰ که هر دو در واقع عدد ۸۹۳ هستند). همچنین برای اضافه کردن پیشوند یا پسوند، دسته بندی ارقام (جداگانه سه رقمی)، یا جداگانه های دهدۀ میتوانید آنها را با استفاده از کلاس DecimalFormat فرمت دهید. اگر بخواهید سمبلهای فرمتدۀ (مثلاً استفاده از / به جای . یا استفاده از کاما به جای نقطه) را نیز تغییر دهید می‌باشد از کلاس DecimalFormat در کنار کلاس DecimalFormatSymbols استفاده کنید. در ادامه استفاده از این دو کلاس را تشریح می‌کنیم.

برای اینکه نحوه فرمتدۀ را برای DecimalFormat مشخص کنید لازم است از یک رشته استفاده کنید که در واقع الگوی نمایش اعداد مشخص می‌کند. تکه کد زیر نحوه ایجاد آبجکت DecimalFormat و فرمت کردن یک عدد را بر اساس آن قالب pattern نشان می‌دهد.

```
DecimalFormat myFormatter = new DecimalFormat(pattern);
String output = myFormatter.format(value);
```

جدول ۲۵-۷ خروجی اعداد مختلف را که براساس الگوهای مختلف تولید شده اند را نشان می‌دهد.

خروجی	الگو	مقدار
123,456.789	###,###.###	123456.789
123456.79	###.##	123456.789
000123.780	000000.000	123.78
\$12,345.67	\$###,###.###	12345.67
¥12,345.67	\u00A5###,###.###	12345.67

جدول ۲۵-۷. الگوهای مختلف برای نمایش اعداد

همانطور که دقت کرده اید تا اینجا در فرمتدۀ اعداد اثری از زبان وجود نداشت. در واقع فرمتدۀ تا اینجا با استفاده از زبان پیش فرض انجام شد. اگر بخواهیم زبان موردنظر خود را در این فرمتدۀ اثر دهیم می‌باشد بازدیگر از NumberFormat استفاده کنیم. کد زیر نحوه استفاده از NumberFormat برای تاثیرگذاری زبان در فرمت اعداد نشان می‌دهد.

```
NumberFormat nf = NumberFormat.getNumberInstance(loc);
```

چند زبانی

```
DecimalFormat df = (DecimalFormat)nf;
df.applyPattern(pattern);
String output = df.format(value);
System.out.println(pattern + " " + output + " " +
loc.toString());
```

با این حساب برای هر یک از زبانهای انگلیسی آمریکایی، دانمارکی و فرانسوی با قالب یکسان خروجی‌های زیر را خواهیم داشت.

###,###.###	123,456.789	en_US
###,###.###	123.456,789	de_DE
###,###.###	123 456,789	fr_FR

تغییر نمادها

در موقعی ممکن است بخواهید نمادهای استفاده شده در فرمتهایی به اعداد را تغییر دهید و از نمادهای دیگری مثل خط فاصله، درصد، ستاره،... استفاده کنید. برای این منظور همانطور که در تکه کد زیر قابل مشاهده است می‌توانید با فراخوانی متدهای `setDecimalSeparator` از `setGroupingSize` و `setGroupingSeparator` از هر نمادی برای فرمتهایی یک عدد استفاده کنید.

```
DecimalFormatSymbols unusualSymbols =
    new DecimalFormatSymbols(currentLocale);
unusualSymbols.setDecimalSeparator('।');
unusualSymbols.setGroupingSeparator('^');
```

```
String strange = "#,##0.###";
DecimalFormat weirdFormatter =
    new DecimalFormat(strange, unusualSymbols);
weirdFormatter.setGroupingSize(4);
```

```
String bizarre = weirdFormatter.format(12345.678);
System.out.println(bizarre);
```

خروجی اجرای تکه کد فوق، نمایش عجیب زیر خواهد بود
1^2345 | 678

فرمتهای دلخواه برای تاریخ و زمان

اگرچه در بیشتر مواقع فرمتهای پیش فرضی که برای تاریخ و زمان پیش بینی شده اند کافیست اما موقعی وجود دارد (خصوصاً زمانی که برنامه شما با زبان فارسی کار می‌کند) که نیاز به شخصی کردن فرمتهای تاریخ و زمان دارید. به این منظور کلاس `SimpleDateFormat` طراحی شده است. در استفاده از این کلاس

به آبجکت زبان (Locale) و الگو (pattern) نیاز داریم. تکه کد زیر نحوه ایجاد آبجکت SimpleDateFormat را نشان می‌دهد.

```
Date today;
String output;
SimpleDateFormat formatter;

formatter = new SimpleDateFormat(pattern, currentLocale);
today = new Date();
output = formatter.format(today);
System.out.println(pattern + " " + output);
```

جدول ۲۵-۸ خروجی کد فوق را با الگوهای مختلف برای زبان انگلیسی آمریکایی نشان می‌دهد.

الگو	خروجی
dd.MM.yy	09.04.98
yyyy.MM.dd G 'at' hh:mm:ss z	1998.04.09 AD at 06:15:55 PDT
EEE, MMM d, 'yy	Thu, Apr 9, '98
h:mm a	6:15 PM
H:mm	18:15
H:mm:ss:SSS	18:15:55:624
K:mm a,z	6:15 PM,PDT
yyyymmdd HHmmss aa	1998.April.09 AD 06:15 PM

جدول ۲۵-۹. الگوهای مختلف برای نمایش تاریخ

هر دو پارامتر زبان (Locale) و الگو (Pattern) در فرمتدی تاریخ و زمان تاثیرگذارند. این بدین معناست که با یک الگوی یکسان، تاریخ و زمان در زبانهای مختلف به شکل متفاوتی نمایش می‌یابند. شما می‌توانید فرمات دلخواه خود را با استفاده از نمادهای مختلفی که برای این منظور پیش بینی شده اند تعریف کنید.

سمبل	معنی	خروجی	مثال
y	سال	عدد	1996
M	ماه	عدد و متن	July & 07
d	روز از ماه	عدد	10
h	ساعت در (۱۲)	عدد	12
H	ساعت در روز (۲۴-۱)	عدد	0
m	دقیقه	عدد	30
s	ثانیه	عدد	55
S	میلی ثانیه	عدد	978

Tuesday	متن	روز هفته	E
189	عدد	روز سال	D
2 (2nd Wed in July)	عدد	روز از هفته از ماه	F
27	عدد	هفته در سال	w
2	عدد	هفته در ماه	W
PM	متن	اعلامت am/pm	a
Pacific Standard Time	متن	Time Zone	z
	جداگانده	escape	'
'	کاراکتر	کوتیشن	'

جدول ۹. نمادهای رایج در تعریف الگوها

اگر کاراکترهای غیرalfبایی در الگو استفاده شوند به همان شکل در خروجی ظاهر می‌شوند. همچنین تعداد کاراکترهای نماد هم در خروجی فرمات موثر هستند به عنوان مثال وجود z در قالب منجر به نمایش PDT و وجود zzzz در قالب منجر به نمایش Pacific Daylight Time خواهد شد.

تغییر نمادهای تاریخ و زمان

خروجی تولید شده توسط SimpleDateFormat شامل ترکیبی از اعداد و کلمات است. به عنوان نمونه رشته Friday, April 10, 1998 - حاوی کلمات Friday و April است که به عنوان نمادهای تاریخ شناخته می‌شوند. اگر نمادهایی که توسط SimpleDateFormat در خروجی تولید شده قرار می‌گیرند جوابگوی نیازهای شما نباشند می‌توانید با استفاده از کلاس DateFormatSymbols آنها را تغییر دهید. جدول زیر لیستی از متدهای DateFormatSymbols را نشان میدهد که امکان تغییر نمادهای تاریخ و زمان را به شما می‌دهد.

نام متدها	مثال از بخشی که توسط متدهای تغییر می‌کند
setAmPmStrings()	PM
setMonths()	December
setShortMonths()	Dec
setShortWeekdays()	Tue
setWeekdays()	Tuesday
setZoneStrings()	PST

جدول ۱۰. متدهای DateFormatSymbols

تمام این متدها، آرایه‌ای از String را به عنوان پارامتر دریافت می‌کنند که به این ترتیب امکان تغییر نمادهای تاریخ و زمان را می‌دهند. در مثال زیر از متدهای setShortWeekdays استفاده شده تا نام روزهای هفته را از حروف کوچک به حروف بزرگ تغییر دهیم.

```

Date today;
String result;
SimpleDateFormat formatter;
DateFormatSymbols symbols;
String[] defaultDays;
String[] modifiedDays;

symbols = new DateFormatSymbols(new Locale("en", "US"));
defaultDays = symbols.getShortWeekdays();

for (int i = 0; i < defaultDays.length; i++) {
    System.out.print(defaultDays[i] + " ");
}
System.out.println();

String[] capitalDays = {
    "", "SUN", "MON", "TUE", "WED", "THU",
    "FRI", "SAT"};
symbols.setShortWeekdays(capitalDays);

modifiedDays = symbols.getShortWeekdays();
for (int i = 0; i < modifiedDays.length; i++) {
    System.out.print(modifiedDays[i] + " ");
}
System.out.println();
System.out.println();

formatter = new SimpleDateFormat("E", symbols);
today = new Date();
result = formatter.format(today);
System.out.println(result);

```

دقت کنید که عنصر اول آرایه رشته خالی "" است که کاملاً منطقی است زیرا روز صفرم هفته وجود ندارد و روزهای هفته از اول تا هفتم هستند.

پیام‌ها

در تمام برنامه‌هایی نمایش داده می‌شوند که کاربر را از آنچه در برنامه اتفاق می‌افتد اعم از خطای غیرخطای مطلع می‌کنند. مسلم است که این پیامها باید به زبان مناسب ترجمه شوند تا برای کاربر قابل فهم باشند. انجام اینکار همانطور که قبله دیدید با بکارگیری فایلهای ResourceBundle امکان‌پذیر است. با این وجود مواردی هست که ممکن است یک پیام حاوی داده‌های متغیری باشد. به پیامهای زیر توجه کنید تا متوجه این نیازمندی بشوید.

چندزبانی

The disk named MyDisk contains 300 files.
The current balance of account #34-98-222 is \$2,745.72.
405,390 people have visited your website since January 1, 1998.

Delete all files older than 120 days.

در تمام این پیامها، کلمات یا عباراتی وجود دارد که در زمان اجرا مشخص می شوند. به عبارت دیگر، به جز بخشهایی که زیر آنها خط کشیده شده است و مقدار آنها در زمان اجرا مشخص می شود، بقیه قسمتهای این پیامها ثابت هستند. پردازش این پیامها که به آنها پیامهای مرکب گفته میشود با استفاده از کلاس `MessageFormat` انجام می شود.

بخشهای ثابت می توانند همانند متنهای معمولی چندزبانی در فایلهای `ResourceBundle` جد اگانه `ResourceBundle` تعریف شوند اما قسمتهای متغیر را می توان با استفاده از نشانه `{x}` مشخص نمود که در آن `x` یک عدد است که مقدار آن در زمان اجرا مشخص می شود. به مثال زیر توجه کنید

The number of {0} records has been updated.

تنها قسمت متغیر این پیام، بخش `{0}` است که در زمان اجرا مشخص می شود. اگر این پیام بخشهای دیگری هم داشته باشد طبیعی است که مقدار 0 افزایش می یابد.

The number of {0} records has been updated in {1} seconds.

این پیامها که در فایلهای `ResourceBundle` به صورت زیر تعریف می شوند

`records.updated = The number of {0} records has been updated.`

`records.updated.inseconds= The number of {0} records has been updated in {1} seconds.`

حال برای اینکه مقادیر متغیرها را در زمان اجرا مشخص کنیم کافیست که به شکل زیر از کلاس `MessageFormat` استفاده کنیم.

`ResourceBundle messages;`

```
...
String msg = messages.getString("records.updated.inseconds");
String result = MessageFormat.format(msg, new Double[]{47d, .12});
```

که در اینصورت مقدار `result` برابر زیر خواهد بود

The number of 47 records has been updated in .12 seconds.

در این مثال، فرض بر این بود که مقدار متغیر برای زبانهای مختلف یکسان است و تنها متن اصلی است که در فایلهای `ResourceBundle` برای زبانهای مختلف متفاوت است.

اگر متغیرها از جنس تاریخ، زمان، رشته، عدد، واحد پول، و یا درصد باشند که در زبانهای مختلف می بايست با فرمتهای مختلفی نمایش داده شوند به کمی تغییرات در کد فوق نیازمندیم. در اینصورت لازم است متغیرها را با استفاده از آنچه در قسمتهای قبلی این فصل آموختید پردازش و آماده کنیم.

تمرینات

تمام تمرینات فصل ۲۱، واسط کاربری، را به صورت دو زبانه (زبان انگلیسی و فارسی) پیاده سازی کنید. وقتی کاربر از طریق منو زبان برنامه را تغییر می‌دهد، برنامه را `exit` کنید تا در اجرای بعدی، برنامه به زبان انتخاب شده کاربر شروع به کار کند. طبیعی است که برای اینکه برنامه بتواند با چنین مکانیسمی کار کند می‌بایست زبان انتخاب شده برنامه در یک فایل تنظیمات ذخیره شود.

۲۴

ابزار JAR

توسعه برنامه های جاوا با تولید کلاس های جاوا (فایلهای `.java`) همراه است که کامپایل می شوند و از روی آنها فایلهای `.class` تولید می شود. در پایان توسعه، وقتی برنامه تکمیل شود مجموعه ای از فایلهای کامپایل شده جاوا و احتمالاً فایلهای غیرجاوا ای (عکس، XML، ...) وجود دارد که باید به محیط عملیاتی منتقل و اجرا شود. اما انتقال این تعداد فایل جدا که در یک برنامه نوعی ممکن است چندین هزار فایل باشد دلچسب نیست. به جای آن می توان با استفاده از ابزار JAR که یکی دیگر از ابزارهای JDK است، فایلهای یک برنامه را فشرده و بسته بندی کرد و در قالب یک فایل JAR ارایه نمود، در اینصورت ضمن کاهش حجم برنامه، حمل و نقل برنامه آسان خواهد شد و البته مزایای دیگری هم خواهد داشت که در ادامه این بخش توضیح داده خواهد شد.

فایل JAR

فایلهای JAR مشابه فایلهای ZIP هستند که مجموعه ای از فایلها را در خود بسته بندی می کنند. همچنین می توان با استفاده از نرم افزارهایی از قبیل WinZip که برای کار با فایلهای ZIP وجود دارند فایلهای JAR را باز کرد یا محتویات آنها را تغییر داد.

برای ایجاد یک فایل JAR و انجام عملیات دیگر روی فایلهای JAR یک ابزار در جاوا وجود دارد که به آن «ابزار JAR» گفته می شود. با استفاده از این ابزار می توان یک فایل JAR ایجاد کرد (مجموعه ای از فایلهای را در یک فایل JAR بسته بندی نمود) یا مجموعه ای از عملیات را روی یک فایل JAR موجود انجام داد.

ابزار JAR در کنار `javac` و بقیه ابزارهای JDK قرار دارد. برای استفاده از ابزار JAR کافیست از طریق پنجره دستور، دستور زیر را اجرا کنید.

```
jar options JARfileName File1 File2 . . .
```

در دستور فوق،

- کلمه `jar` فراخوانی ابزار JAR است.
- کلمه `options` «گزینه ها»ی دستور JAR است که مشخص می کند ابزار JAR قرار است چه عملی (ایجاد فایل، دستکاری فایل,...) را انجام دهد.
- کلمه `JAR` نام فایل JAR ای را مشخص می کند که میخواهیم ایجاد یا دستکاری کنیم.
- `JARfileName` فایلهایی را مشخص می کند که قرار است داخل فایل JAR بسته بندی شوند.
- `File1 File2 ...`

توجه: از آنجاییکه مسیر `%JAVA_HOME%/bin` در `PATH` سیستم قرار دارد (براساس تنظیماتی که در فصل اول انجام داده اید) و فایل `.jar.exe` در این مسیر قرار دارد، سیستم شما دستور `jar` را می شناسد.

مثلا برای ایجاد یک فایل JAR از طریق پنجره دستور به مسیری که میخواهید محتویات آنرا در یک فایل JAR بسته بندی کنید می روید و دستور فوق را به شکل زیر اجرا می کنید.

```
jar cvf JARfileName File1 File2 . . .
```

- کلمه `cvf` که در مثال فوق استفاده شده و شامل سه کلرکتر `c`, `v` و `f` است گزینه های دستور فوق است.
- **شروع کلمه** `(create)` مشخص می کند که میخواهیم فایل JAR جدیدی ایجاد کنیم. `v` (شروع کلمه `verbose`) مشخص می کند که در حین اجرای ابزار JAR میخواهیم گزارش کامل کارکرد ابزار JAR را در کنسول برنامه مشاهده کنیم و `f` (شروع کلمه `file`) به ابزار JAR می گوید که کلمه بعدی که در دستور ظاهر شده (کلمه `JARfileName`) نام همان فایلی است که قصد ایجاد آنرا داریم. جدول ۲۶-۱ گزینه های قابل استفاده در دستور JAR را نشان می دهد.

گزینه	معنی
<code>c</code>	یک فایل JAR جدید ایجاد می کند و فایلهای مشخص شده را در آن قرار می دهد. اگر به جای

ابزار JAR

فایل، مسیر یک دایرکتوری مشخص شود، فایلهای آن دایرکتوری به آن اضافه می شود.	e
نام فایل JAR را مشخص می کند. اگر از این پارامتر استفاده نشود، در هنگام باز کردن یک فایل JAR محتويات آن در کنسول برنامه نمایش داده می شود و در هنگام ایجاد فایل JAR یک فایل ايندکس به فایل JAR اضافه می کند که امكان می دهد بتوان جستجوی سريع روی محتويات فایل JAR داشت.	f
فایل Manifest را به فایل JAR اضافه می کند. فایل Manifest شناسنامه فایل JAR و محتويات آن است. هر فایل JAR به صورت پيش فرض حاوی یک فایل Manifest است اما با استفاده از اين پارامتر اين امكان وجود دارد تا يك فایل Manifest شخصی برای معرفی محتويات فایل JAR جایگزین کنيد.	m
فایل Manifest را ایجاد نمی کند.	M
شروع کلمه «table of contents» است و محتويات فایل JAR را نشان ميدهد.	t
محتويات یک فایل JAR را بروزرسانی (update) می کند.	u
جزييات اجرای دستور JAR را در کنسول نمایش می دهد.	v
فایل JAR را از حال فشرده باز می کند.	x
فایل JAR را بدون هیچ گونه فشردگی، بسته بندی می کند.	0

جدول ۱-۲۶. گزینه های دستور JAR

Manifest فایل

علاوه بر فایلهایی که به صورت معمول در یک فایل JAR بسته بندی می شوند، یک فایل Manifest هم داخل فایل JAR گنجانده می شود که در واقع شناسنامه فایل JAR است و خصوصیات فایل JAR را توضیح می دهد. نام این فایل MANIFEST.MF است و در زیرشاخه META-INF از فایل JAR قرار دارد. این فایل به صورت پيش فرض در تمام فایلهای JAR وجود دارد (مگر در مواردی که با استفاده از گزینه M مانع از ایجاد آن شده باشد) و به صورت پيش فرض حاوی متن زير است

Manifest-Version: 1.0

این فایل می تواند حاوی جزئيات بیشتری باشد که در بخشهاي جداگانه اى قرار داده می شوند. اولین بخش که به آن «بخش اصلی» گفته می شود، جزئياتی کلی در مورد فایل JAR را مشخص می کند. بخشهاي بعدی که همگی با عبارت Name: شروع می شوند، خصوصیات یک فایل و یا يك پکیج خاص موجود در فایل JAR را توضیح می دهند. به عنوان مثال:

Manifest-Version: 1.0

lines describing this archive

Name: Woozle.class

```
lines describing this file
Name: com/mycompany/mypkg/
lines describing this package
```

برای ویرایش فایل Manifest کافیست خطوطی را که میخواهید به فایل Manifest اضافه شود داخل یک فایل متنی قرار دهید و سپس دستور زیر را اجرا کنید.

```
jar cfm JARfileName ManifestfileName . . .
```

به عنوان مثال در زمان ایجاد یک فایل JAR می توانید فایل manifest.mf را که خودتان ایجاد کرده اید به صورت زیر در فایل JAR بگنجانید

```
jar cfm MyArchive.jar manifest.mf com/mycompany/*.class
```

برای ویرایش یک فایل Manifest موجود در یک فایل JAR، خطوطی را که میخواهید اضافه شود در یک فایل متنی قرار دهید و دستور زیر را اجرا کنید.

```
jar ufm MyArchive.jar manifest-additions.mf
```

اجرایی کردن فایل JAR

اگر فایل JAR قابل اجرا باشد (یعنی حاوی کلاسی باشد که حاوی متدهای main(String[] args) باشد) می توان این کلاس را در فایل Manifest معرفی کرد.

```
jar cvfe MyProgram.jar com.mycompany.mypkg.MainAppClass files to add
```

که باعث اضافه شدن خط زیر در فایل Manifest خواهد شد.
Main-Class: com.mycompany.mypkg.MainAppClass

حال که کلاس اجرایی برنامه به صورت فوق در فایل Manifest معرفی گردیده است ، تنها با دستور زیر و بدون مشخص کردن نام کلاس، فایل JAR اجرا می شود.

```
java -jar MyProgram.jar
```

JAR باز

اگر برنامه‌ای که اجرا می‌شود واسطه کاربری دارد و نمی‌خواهد کنسول برنامه باز شود به جای `java` از `javaw` به صورت زیر استفاده کنید.

```
javaw -jar MyProgram.jar
```

توجه داشته باشید که اگرچه تا اینجا یک فایل `jar` را قابل اجرا کرده اید، اما ممکن است تصور شما از اجرایی شدن یک فایل `jar` ایجاد یک فایل `exe`. باشد ☺ به این فایلهای محلی (`native`) گفته می‌شود که مخصوص یک سیستم عامل هستند و روی سیستم عاملهای دیگر مثلاً مکینتاش یا لینوکس اجرا نمی‌شوند. به همین دلیل که با شعار جاوا (یکبار بنویسید و همه جا اجرا کنید) سازگار نیستند و توسط جاوا پشتیبانی نمی‌شود، با این وجود بعضی محصولات منبع-باز یا تجاری وجود دارند که می‌توانند از روی یک فایل JAR اجرایی فایلهایی محلی ایجاد کنند. لینکهای زیر برخی از این محصولات را نشان می‌دهد.

JSmooth <http://jsmooth.sourceforge.net>

Launch4J <http://launch4j.sourceforge.net>

IzPack <http://izpack.org>

تمرینات

از روی چند نمونه از برنامه هایی که تاکنون نوشته اید فایل JAR بسازید و آنها را اجرا کنید.

PV

javadoc

در فصل اول، با نوع خاصی از توضیحات آشنا شدید که به آن **javadoc** گفته می‌شود. **javadoc** که با علامت `*/ *` شروع و با `/*` خاتمه می‌یابند توضیحاتی را در مورد کلاس، فیلد و متدهای کلاس ارایه می‌کنند که مستندات رسمی آنها محسوب می‌شوند و می‌توانند توسط برنامه نویسان دیگر که قصد استفاده از آن کلاسها، فیلدها و متدها را دارند مطالعه و درک شوند. باید دقت داشته باشید که کامپایلر جاوا با **javadoc** مشابه توضیحات دیگر برنامه رفتار می‌کند و آنها را صرفنظر می‌کند. بنابراین وقتی کدهای برنامه کامپایل شود اثری از آنها در بایت کدهای برنامه (فایلهای `Class`) وجود ندارد، اما تفاوتی که **javadoc** با توضیحات معمول برنامه دارد در این است که ابزار **javadoc** (یکی دیگر از ابزارهای `JDK`) می‌تواند سورس‌های برنامه را پردازش کند و توضیحات **javadoc** را در قالب فایلهای `HTML` ارایه کند. در اینصورت حتی زمانیکه سورس‌های برنامه موجود نباشد می‌توان فایلهای `HTML` تولید شده را به عنوان مستندات رسمی سورس ارایه کرد.

در توضیحاتی که به عنوان **javadoc** در سورس‌های برنامه نوشته می‌شوند می‌توان از برخی عالیم استفاده کرد تا توضیحات با معنی تری ارایه نمود. برخی از این عالیم پرکاربرد در جدول ۲۷-۱ نشان داده شده‌اند.

علامت	معنی
<code>@author</code>	نویسنده کد را مشخص می‌کند.
<code>@param</code>	پارامتر یک متر یا سازنده کلاس را مشخص می‌کند.

نوع برگشتی یک متند را مشخص می کند.	@return
یک exception که یک متند ممکن است تولید کند را مشخص می کند.	@throws
یک exception را توصیف می کند.	@exception

جدول ۱-۲۷. برخی خلاصه های رایج Javadoc

javadoc هایی که در سورس‌های برنامه نوشته می شوند را می توان به دو دسته تقسیم کرد:

- در سطح کلاس، که توضیحاتی در مورد کلاس ارایه می کنند.
 - در سطح اعضای کلاس، که توضیحاتی در مورد یک فیلد یا یک متند از کلاس ارایه می کنند.
- توضیحاتی که در سطح کلاس ارایه می شوند بالای تعریف کلاس قرار می گیرند و عموماً حاوی علامت `@author` هستند که برنامه نویس (یا برنامه نویسان) کلاس را مشخص می کند. کد زیر، این نوع `javadoc` را نشان می دهد.

```

1 package ir.atlassoft.javase.chapter27;
2 /**
3 * @author Reza Yazdanmehr.
4 * The Employee class contains data about one employee.
5 * Fields include an ID number and an hourly pay rate.
6 */
7 public class Employee {
8     private int id;
9     private double hourlyPay;
10    public Employee(int id, double hourlyPay) {
11        this.id = id;
12        this.hourlyPay = hourlyPay;
13    }
14
15
16    int getId () {
17        return id;
18    }
19
20    void setId (int id) {
21        this.id = id;
22    }
23 }
```

javadoc

javاهایی که در سطح اعضای کلاس ظاهر می شوند بالای فیلدها، متدها یا سازنده های کلاس قرار می گیرند و ممکن است حاوی علامتهای `@throws`، `@param` و `@return` باشند. کد زیر، مثالهایی از javahای سطح اعضای کلاس را نشان می دهد.

```
1 package ir.atlassoft.javase.chapter27;
2 /**
3 * @author Reza Yazdanmehr.
4 * The Employee2 class contains data about one employee.
5 * Fields include an ID number and an hourly pay rate.
6 */
7 public class Employee2 {
8
9     /**
10      * Employee ID number
11      */
12     private int id;
13
14     /**
15      * Employee hourly pay
16      */
17     private double hourlyPay;
18
19     /**
20      * Sole constructor for Employee2
21      */
22     public Employee2(int id, double hourlyPay) {
23         this.id = id;
24         this.hourlyPay = hourlyPay;
25     }
26
27     /**
28      * Returns the Employee2 ID number
29      * @return int
30      */
31     int getId() {
32         return id;
33     }
34
35     /**
```

```

36      * Sets the Employee2 ID number
37      * @param id employee ID number
38      */
39      void setId (int id) {
40          this.id = id;
41      }
42  }
```

کد ۲۷-۲

تولید اسناد javadoc

بالاخره وقتی نسخه ای از یک نرم افزار منتشر می شود، باید بتوانیم مستندات سورس آن نرم افزار را نیز تولید کنیم. این مستندات که در قالب فایلهای HTML هستند توسط ابزار **javadoc** که یکی از ابزارهای JDK است از روی توضیحات **javadoc** که برنامه نویسان در سورس های برنامه قرار داده اند تولید می شود. برای این منظور کافیست به مسیری که سورسهاي برنامه قرار دارند بروید و دستور زیر را از پنجره دستور اجرا کنید.

```
javadoc -d Documents *.java
```

پارامتر **-d** مسیری که فایلهای HTML باید در آن تولید شوند را مشخص می کند که در این مثال دایرکتوری **Documents** در کنار سورسهاي برنامه است. در اینصورت به ازای هر کلاس، اینترفیس یا نوعی که در پروژه وجود داشته باشد یک فایل HTML تولید و در دایرکتوری **Documents** ذخیره می شود. در دایرکتوری **Documents** یک فایل **index.html** وجود دارد که صفحه اصلی مستندات است و از طریق آن می توانید به تمام پکیج های برنامه و مستندات کلاسهاي داخل آنها دسترسی پیدا کنید.

چند نکته

در نوشتن توضیحات **javadoc** معمولاً قواعدی خاص پروژه تدوین می شود تا توضیحات برنامه نویسان منسجم و متحددالشكل و قاعده مند شود. به عنوان نمونه، در یک پروژه ممکن است قوانین زیر برای توضیحات **javadoc** تصویب شود.

توضیحات فقط به زبان انگلیسی نوشته می شود

شروع توضیحات با حرف بزرگ الفبای انگلیسی باشد

اگر توضیح مربوط به یک متده است با کلمه **Executes** یا **Sets** یا **Returns** و امثالهم شروع شود.

javadoc

وقتی کلاسی توسط فردی ایجاد می شود از علامت `@author` برای مشخص کردن ایجاد کننده آن کلاس استفاده شود. برنامه نویسان دیگری که آن کلاس را تغییر می دهنده، به ترتیب بعد از نام ایجاد کننده و هر کدام با استفاده از علامت `@author` نام خود را مشخص کنند.

توجه: متدهای `getter/setter` JavaDoc نیاز به دارند.

شما ممکن است در پروژه خود، قواعدی را به قواعد فوق اضافه کنید یا آنها را تغییر دهید که اینکار معمولاً توسط مدیرفNI پروژه انجام می شود.

نکته دیگری که لازم به ذکر است این است که ابزار `javadoc` به صورت پیش فرض نام نویسنده کد را به مستدات اضافه نمی کند ولی اگر جز این مد نظر شماست می بایست از طریق پارامتر `-author` به صورت زیر آنرا مشخص کنید.

```
javadoc -d Documents -author *.java
```

همچنین به صورت پیش فرض ابزار `javadoc` فیلدها و متدهای و دیگر اعضای `private` کلاسها حتی اگر توضیحات `javadoc` داشته باشند، در مستندات تولید شده لحاظ نمی کند. بنابراین اگر می خواهید برای اعضای `private` کلاسها نیز مستندات `javadoc` تولید شود می بایست پارامتر `-private` را به صورت زیر اضافه کنید.

```
javadoc -d Documents -private *.java
```

با دستور فوق برای تمام اجزای کلاسها مستندات `javadoc` تولید می شوند. لازم به ذکر است که بجای کلمه `private` می توانید از `public`, `protected` یا `private` استفاده کنید که در اینصورت به ترتیب فقط اعضای `public`, `protected` و `private` ظاهر می شوند. در مستندات `javadoc` `modifier`

