

آموزش برنامه نویسی جاوا

جلد 2

ترجمه و تألیف: انور پوراحمد

انتشارات پندار پارس

سرشناسه	: پورا احمد، انور، 1369 -
عنوان و نام پدیدآور	: آموزش برنامه‌نویسی جاوا/ انور پورا احمد.
مشخصات نشر	: تهران: پندار پارس: مانلی، 1390 -
مشخصات ظاهری	: ج.: مصور، جدول. (ج 1: 400 ص، ج 2: 472ص)
شابک	: 120000 ریال با لوح فشرده: جلد دوم: 6-67-2989-964-978-3-68-2989-964-978
وضعیت فهرست نویسی	: فیپا
موضوع	: جاوا (زبان برنامه‌نویسی کامپیوتر)
رده بندی کنگره	: 76/73QA / ج 2 پ 9 1390
رده بندی دیویی	: 005/133
شماره کتابشناسی ملی	: 8425032

انتشارات پندار پارس



دفتر فروش: انقلاب، ابتدای کارگر جنوبی، کوی رشتچی، شماره 14، واحد 16 www.pendarepars.com
 تلفن: 66572335 - تلفکس: 66926578 همراه: 09122452348
info@pendarepars.com



نام کتاب	: آموزش برنامه‌نویسی جاوا (جلد 2)
ناشر	: انتشارات پندار پارس ناشر همکار: مانلی
ترجمه و تالیف	: انور پور احمد
چاپ اول	: پاییز 90
شمارگان	: 1000 نسخه
طرح جلد	: محمد اسماعیلی هدی
لیتوگرافی، چاپ، صحافی	: ترام سنج، صالحان، خیام
قیمت	: 12000 تومان به همراه CD
شابک	: 6-67-2989-964-978-3-68-2989-964-978

شابک دوره : 3-68-2989-964-978



* هرگونه کپی برداری، تکثیر و چاپ کاغذی یا الکترونیکی از این کتاب بدون اجازه ناشر تخلف بوده و پیگرد قانونی دارد *

پیشگفتار

کتاب پیش‌روی، جلد دوم کتاب آموزش برنامه‌نویسی جاوا می‌باشد که ترجمه فصل‌های 7 تا 14 کتاب دکتر کای هورستمن¹ یعنی Big JAVA است که بنا به مصلحت، در 2 جلد ترجمه و تقدیم علاقه‌مندان شده است. ممکن است در برخی از فصل‌های این جلد، ارجاعی به فصل‌های جلد 1 داده شده باشد، به همین دلیل ملزم به استفاده از جلد اول نیز خواهید بود.

در صورتی که فردی مبتدی هستید یا با زبان جاوا آشنایی اندکی دارید توصیه می‌کنم ابتدا جلد اول را به‌طور کامل مطالعه و سپس، مطالب تکمیلی را در این جلد دنبال نمایید.

در رابطه با لوح فشرده و سورس برنامه‌ها

در لوح فشرده همراه این جلد، سورس کدهای تمامی فصل‌های هر دو جلد ارائه شده است. همچنین آخرین نسخه از بهترین محیط‌های برنامه‌نویسی یعنی NetBeans 7.0.1 و Eclipse به همراه JDK نسخه 7 (آخرین نسخه ارائه شده جاوا تاکنون) ارائه شده است. هرچند قادرید نسخه‌های جدیدتر و کامل‌ترین سورس برنامه‌ها را از آدرس‌های زیر (وبلاگ مترجم) نیز دانلود نمایید:

www.apcomputer.tk

www.apcomputer.blogfa.com

با توجه به نمودار فصل‌های کتاب که در ابتدای جلد اول آمده است، فصل‌های 1 تا 6 در جلد اول و فصل‌های 7 تا 14 در جلد دوم قرار گرفته است. لذا با توجه به زیاد شدن حجم کتاب از ارائه مباحث فصول 15 و 16 که مربوط به ساختمان داده‌هاست خودداری کرده‌ایم. لازم به ذکر است که با مبحث ساختمان داده‌ها به‌صورت کاملاً تخصصی در درسی با همین نام آشنا خواهید شد و شاید دلیل اصلی خودداری از ارائه این دو فصل در این کتاب همین بوده باشد. همچنین مترجم کتاب در نظر دارد به زودی کتابی تخصصی برای ساختمان داده‌ها در جاوا منتشر نماید. افرادی که علاقه‌مند به آموختن ساختمان داده‌ها در جاوا هستند به زودی می‌توانند کتابی تحت همین عنوان را تهیه نمایند.

سخن آخر

در انتها بر خود لازم می‌دانم از همکاری صمیمانه جناب آقای محمدمهدی ایزدی جهت ویرایش کتاب تشکر نمایم. همچنین از همکاری کارکنان زحمتکش انتشارات پندار پارس به‌ویژه جناب آقای مهندس حسین یعسوبی، مدیریت محترم انتشارات پندار پارس به دلیل زحمات شبانه‌روزی نهایت تشکر و قدردانی خود را اعلام می‌دارم.

¹ Cay's Horstmann

به دلیل اینکه هیچ مجموعه‌ای عاری از اشکال نمی‌باشد، لذا از شما اساتید محترم، دانشجویان گرامی و سایر خوانندگان عزیز خواهشمند است جهت بهبود کیفی کتاب، اشکالات موجود را به یکی از روش‌های ارتباطی زیر به اطلاع مترجم برسانند. همچنین آماده دریافت نظرات، پیشنهادات و انتقادات سازنده شما عزیزان هستیم:

E-mail: apco_pourahmad@yahoo.com

G-mail: apco.pourahmad@gmail.com

انور پوراحمد

پاییز 1390

فهرست

385.....	فصل 7 آرایه‌ها و لیست‌های آرایه‌ای
385.....	7,1 آرایه‌ها
389.....	خودآزمایی
389.....	دستورالعمل 7,1: ایجاد آرایه
389.....	دستورالعمل 7,2: دسترسی به عناصر آرایه
390.....	خطای رایج 7,1: خطاهای محدوده (Bound Errors)
390.....	خطای رایج 7,2: آرایه‌های مقداردهی نشده
390.....	مبحث پیشرفته 7,1: مقداردهی اولیه به آرایه‌ها
391.....	7,2 لیست‌های آرایه‌ای
396.....	خودآزمایی
396.....	خطای رایج 7,3: طول و اندازه
397.....	نکته مدیریتی 7,1: پارامتری نمودن لیست‌های آرایه‌ای
397.....	7,3 بسته بندی‌ها و Auto-Boxing
400.....	خودآزمایی
400.....	7,4 for بهبود یافته
402.....	دستورالعمل 7,3
402.....	7,5 الگوریتم‌های ساده در آرایه‌ها
402.....	الف) شمارش تطابق
403.....	ب) پیدا کردن یک مقدار
403.....	ج) پیدا کردن مقدار ماکزیمم یا مینیمم
407.....	7,6 آرایه‌های دو بعدی
411.....	7,1 نحوه کار با لیست‌های آرایه‌ای و آرایه‌ها
411.....	گام 1. انتخاب ساختمان داده مناسب
411.....	گام 2. ایجاد یک لیست آرایه‌ای یا آرایه و ارجاع مقداری به آن
411.....	گام 3. اضافه نمودن عناصر
411.....	گام 4. پردازش عناصر
412.....	مبحث پیشرفته 7,2: آرایه‌های دوبعدی با سطرهای متغیر
413.....	مبحث پیشرفته 7,3: آرایه‌های چندبعدی
413.....	7,7 نسخه برداری از آرایه‌ها
418.....	نکته مدیریتی 7,2: آرایه‌های موازی مقارن با آرایه‌های آبجکتی
421.....	مبحث پیشرفته 7,5: متدهای دارای تعدادی از پارامترها
422.....	وقایع تصادفی 7,1: اولین کرم اینترنتی
423.....	7,8 آزمون رگرسیون (آزمایش بازگشتی)
428.....	کلاس‌ها، آبجکت‌ها و متدهای معرفی شده در این فصل

428.....	تمرینات مروری
431.....	تمرینات برنامه‌نویسی
455.....	پروژه‌های برنامه‌نویسی
456.....	پاسخ خودآزمایی‌ها
459.....	فصل 8 طراحی کلاس‌ها
460.....	8,1 انتخاب کلاس‌ها
462.....	8,2 انسجام و تطابق
464.....	نکته مدیریتی 8,1: ثبات و سازگاری
465.....	8,3 متدها و کلاس‌های دسترسی، تغییر و تغییرناپذیر
470.....	نکته مدیریتی 8,2: به حداقل رسانی عوارض جانبی
470.....	میچت پیشرفته 8,1: فراخوانی با ارجاع و فراخوانی با مقدار
472.....	8,5 پیش‌نیازها
479.....	8,6 متدهای استاتیک
480.....	8,7 فیلدهای استاتیک
484.....	میچت پیشرفته 8,3: فرم‌های جایگزین برای مقداردهی اولیه فیلدها
485.....	8,8 حوزه
485.....	الف) حوزه متغیرهای محلی
487.....	ب) حوزه اعضای کلاس
488.....	ج) همپوشانی در حوزه‌ها
489.....	خطای رایج 8,2: یدک‌کشی فیلدها
490.....	اشاره سودمند 8,1: جستجو و جایگزینی سراسری
492.....	اشاره سودمند 8,2: عبارات باقاعده
492.....	میچت پیشرفته 8,4: import‌های استاتیک
493.....	8,9 پکیج‌ها
493.....	الف) سازماندهی کلاس‌های مرتبط در پکیج‌ها
494.....	ب) import نمودن پکیج‌ها
495.....	ج) نام پکیج‌ها
496.....	د) چگونگی تعیین مکان کلاس‌ها
498.....	دستورالعمل 8,2: تشخیص پکیج‌ها
498.....	خطای رایج 8,3: اشتباه در مکان نقطه‌ها
499.....	8,1 نحوه برنامه‌نویسی با پکیج‌ها
501.....	وقایع تصادفی 8,1: رشد سریع کامپیوترهای شخصی
504.....	8,10 واحد آزمون frameworkها
514.....	تمرینات برنامه‌نویسی
521.....	پروژه‌های برنامه‌نویسی
522.....	پاسخ خودآزمایی‌ها

525.....	فصل 9 اینترفیس‌ها و چندریختی
526.....	9,1 استفاده از اینترفیس‌ها جهت استفاده مجدد از کدها
532.....	دستورالعمل 9,2: اجرای اینترفیس
532.....	خطای رایج 9,1: فراموش کردن تعریف ایجاد متدها به صورت public
533.....	مبحث پیشرفته 9,1: ثوابت در اینترفیس‌ها
533.....	9,2 توانایی تبدیل اینترفیس‌ها و کلاس‌ها
535.....	خطای رایج 9,2: سعی برای نمونه‌سازی اینترفیس
536.....	9,3 چند ریختی
543.....	9,5 کلاس‌های داخلی
546.....	مبحث پیشرفته 9,2: کلاس‌های ناشناس
547.....	وقایع تصادفی 9,1: سیستم‌های عامل
550.....	9,6 رویدادها، منابع آن و رویدادهای شنیداری
554.....	خطای رایج 9,3: اصلاح متد ایجاد شده
555.....	9,7 استفاده از کلاس‌های داخلی برای شنودگرها
558.....	9,8 ایجاد برنامه‌های کاربردی با دکمه‌ها
562.....	خطای رایج 9,4: فراموش کردن اضافه نمودن شنودگر
562.....	اشاره سودمند 9,1: از ظرف نگهدارنده به‌عنوان شنودگر استفاده نکنید
563.....	9,9 پردازش رویدادهای زمانسنج
567.....	خطای رایج 9,5: فراموش کردن ترسیم مجدد
567.....	9,10 رویدادهای ماوس
571.....	مبحث پیشرفته 9,3: آداپتورهای رویداد (وفق دهنده‌ها)
572.....	وقایع تصادفی 9,2: زبان‌های برنامه‌نویسی
575.....	کلاس‌ها، آبجکت‌ها و متدهای معرفی شده در این فصل
576.....	تمرینات مروری
579.....	تمرینات برنامه‌نویسی
596.....	پروژه‌های برنامه‌نویسی
597.....	پاسخ خودآزمایی‌ها
599.....	فصل 10 وراثت
600.....	10,1 معرفی وراثت
603.....	دستورالعمل 10,1: وراثت
604.....	خطای رایج 10,1: شرایط گیج‌کننده کلاس‌های مافوق و کلاس‌های فرعی
605.....	10,2 سلسله‌مراتب وراثت
608.....	10,3 ارثیری از فیلدها و متدها
611.....	دستورالعمل 10,2: فراخوانی متد کلاس مافوق
612.....	خطای رایج 10,2: پنهان کردن فیلدها
613.....	خطای رایج 10,3: عدم فراخوانی متد کلاس مافوق

614	10,4 ساخت کلاس فرعی
615	دستورالعمل 10,3: فراخوانی سازنده کلاس مافوق
615	10,5 تبدیل میان انواع کلاس فرعی و کلاس مافوق
618	دستورالعمل 10,4: عملگر instanceof
619	10,6 چند ریختی
624	مبحث پیشرفته 10,1: کلاس‌های انتزاعی
626	مبحث پیشرفته 10,2: کلاس‌ها و متدهای Final
627	10,7 کنترل دسترسی
629	خطای رایج 10,4: دسترسی تصادفی به پکیج‌ها
630	خطای رایج 10,5: ایجاد متدهای ارثی با قابلیت دسترسی کمتر
630	مبحث پیشرفته 10,3: سطح دسترسی حفاظت شده
631	10,8 آبجکت: کلاس مافوق سطح بالا
632	10,8 (الف) override نمودن متد toString
634	10,8 (ب) override نمودن متد equals
636	10,8 (ج) متد clone
637	اشاره سودمند 10,1: اعمال متد toString در تمامی کلاس‌ها
637	مبحث پیشرفته 10,4: وراثت و متد toString
638	خطای رایج 10,6: تعریف متد equals همراه با انواع پارامترهای غلط
639	مبحث پیشرفته 10,5: وراثت و متد equals
640	خطای رایج 10,7: فراموشی clone نمودن
640	نکته مدیریتی 10,1: clone نمودن فیلدهای متغیر در متدهای دسترسی
641	مبحث پیشرفته 10,6: ایجاد متد clone
644	مبحث پیشرفته 10,7: بازنگری انواع شمارشی
645	وقایع تصادفی 10,1: زبان‌های اسکریپتی
647	10,9 استفاده از وراثت برای سفارشی نمودن Frameها
649	مبحث پیشرفته 10,8: اضافه نمودن متد main به کلاس Frame
649	10,10 پردازش ورودی متنی
653	10,11 نواحی متن
657	10,1 نحوه ایجاد رابط کاربری گرافیکی (GUI)
658	خطای رایج 10,8: به‌صورت پیش‌فرض اجزاء دارای طول و عرض صفر هستند
658	اشاره سودمند 10,2: استفاده مجدد از کدها
660	کلاس‌ها، آبجکت‌ها و متدهای استفاده شده در این فصل
661	تمرینات مروری
664	تمرینات برنامه‌نویسی
677	پروژه‌های برنامه‌نویسی
678	پاسخ خودآزمایی‌ها

681	فصل 11 ورودی‌ها، خروجی‌ها و مدیریت استثناها
682	11,1 نوشتن و خواندن فایل‌های متنی
684	خطای رایج 11,1: بکاسلش‌ها در نام فایل‌ها
685	مبحث پیشرفته 11,1: جعبه محاوره‌ای فایل
687	مبحث پیشرفته 11,2: آرگومان‌های خط فرمان
688	11,2 ایجاد استثناها
691	دستورالعمل 11,1: ایجاد استثناها
691	11,3 استثناهای بررسی شده و بررسی نشده
694	دستورالعمل 11,2: تشخیص استثناها
694	11,4 دریافت استثناها
696	دستورالعمل 11,3: بلوک عمومی Try
697	نکته مدیریتی 11,1: Catch Late و Throw Early
697	نکته مدیریتی 11,2: استثناها را نادیده نگیرید
697	11,5 عبارت Finally
699	دستورالعمل 11,4: عبارت finally
700	نکته مدیریتی 11,3: از finally و catch در دستور try یکسان استفاده نکنید
700	11,6 طراحی انواع استثناها
701	نکته مدیریتی 11,4: ایجاد استثناهای خاص
702	11,7 بررسی موردی: یک مثال کامل
708	وقایع تصادفی 11,1: حوادث موشک آریان
710	کلاس‌ها، متدها و آبجکت‌های معرفی شده در این فصل
711	تمرینات مروری
712	تمرینات برنامه‌نویسی
718	پروژه‌های برنامه‌نویسی
719	پاسخ خودآزمایی‌ها
721	فصل 12 طراحی شیء‌گرایی
722	12,1 چرخه دوام نرم‌افزار
727	وقایع تصادفی 12,1: بهره‌وری برنامه‌نویس
729	12,2 کشف کلاس‌های جدید
732	12,3 روابط میان کلاس‌ها
735	12,1 نحوه ایجاد و ترسیم کارت‌های CRC و نمودارهای UML
737	مبحث پیشرفته 12,1: صفت‌ها و متدها در نمودار UML
737	مبحث پیشرفته 12,2: تعدد، چندگانگی
738	مبحث پیشرفته 12,3: تراکم و وابستگی
757	تمرینات مروری
758	تمرینات برنامه‌نویسی

761.....	پاسخ خودآزمایی‌ها
763.....	فصل 13 بازگشت
764.....	13,1 اعداد مثلثی
768.....	خطای رایج 13,1: بازگشت نامتناهی
768.....	13,2 جایگشت
772.....	خطای رایج 13,2: ردیابی روش بازگشتی
772.....	13,3 متدهای بازگشتی کمکی
774.....	13,4 کارایی بازگشت
781.....	13,5 بازگشت‌های دوطرفه
788.....	تمرینات مروری
788.....	تمرینات برنامه‌نویسی
791.....	پروژه‌های برنامه‌نویسی
791.....	پاسخ خودآزمایی‌ها
793.....	فصل 14 مرتب‌سازی و جستجو
794.....	14,1 مرتب‌سازی انتخابی
798.....	14,2 نمایه‌ای از الگوریتم مرتب‌سازی انتخابی
802.....	14,3 تجزیه و تحلیل کارایی الگوریتم مرتب‌سازی انتخابی
803.....	مبحث پیشرفته 14,1: مرتب‌سازی درجی
806.....	14,4 مرتب‌سازی ادغامی
810.....	14,5 تجزیه و تحلیل الگوریتم مرتب‌سازی ادغامی
813.....	مبحث پیشرفته 14,3: الگوریتم مرتب‌سازی سریع
815.....	14,6 جستجو
818.....	14,7 جستجوی دودویی (باینری)
821.....	14,8 مرتب‌سازی داده‌های حقیقی
823.....	خطای رایج 14,1: متد compareTo می‌تواند هر عدد صحیحی را برگرداند
824.....	مبحث پیشرفته 14,4: اینترفیس Comparator (مقایسه‌گر)
826.....	متدها، کلاس‌ها، آبجکت‌ها و کلاس‌های معرفی شده در این فصل
826.....	تمرینات مروری
827.....	تمرینات برنامه‌نویسی
832.....	پروژه‌های برنامه‌نویسی
833.....	پاسخ خودآزمایی‌ها

فصل 7

آرایه‌ها و لیست‌های آرایه‌ای

اهداف فصل

- آشنایی با آرایه‌ها و لیست‌های آرایه‌ای
- آشنایی با ویژگی‌های آرایه‌ها
- بررسی الگوریتم‌های رایج در استفاده از آرایه‌ها
- نحوه استفاده از آرایه‌های دو بعدی
- درک چگونگی استفاده از آرایه‌ها و لیست‌های آرایه‌ای در برنامه
- درک نحوه بازگشت در آرایه‌ها

همانطور که می‌دانید جهت انجام محاسبات بر روی بعد وسیعی از کمیت‌های داده‌ای باید یک ساختمان داده‌ای ایجاد کنید. رایج‌ترین نوع ساختمان داده‌ای در جاوا، آرایه‌ها و لیست‌های آرایه‌ای هستند. در این فصل، نحوه ایجاد آرایه‌ها و لیست‌های آرایه‌ای را خواهید آموخت. این انواع داده‌ای با داده‌ها پر می‌شوند و داده‌ها را در خود نگهداری می‌کنند. سریع‌ترین و صحیح‌ترین راه ممکن برای انجام محاسبات بر روی تمامی اجزای آرایه‌ها، `for` بهبود یافته است که در ادامه، آن را نیز معرفی خواهیم کرد. همچنین نحوه استفاده از `for` بهبود یافته برای ایجاد الگوریتمی از آرایه‌ها را خواهید آموخت.

7,1 آرایه‌ها

در بسیاری از برنامه‌ها می‌بایست مجموعه وسیعی از داده‌ها را مدیریت کنید. در صورتی که بخواهید برای هر داده، متغیری مانند `data1`, `data2`, `data3`, ... تعریف کنید، قطعاً کاری طاقت فرسا یا غیر ممکن خواهد بود. آرایه، روش بهتری برای نگهداری داده‌ها ارائه داده است.

آرایه، ترتیبی از مقادیر با نوع یکسان می‌باشد. برای مثال در دستور زیر نحوه ایجاد آرایه‌ای که 10 عدد اعشاری را نگهداری می‌کند، مشاهده می‌کنید:

```
new double[10]
```

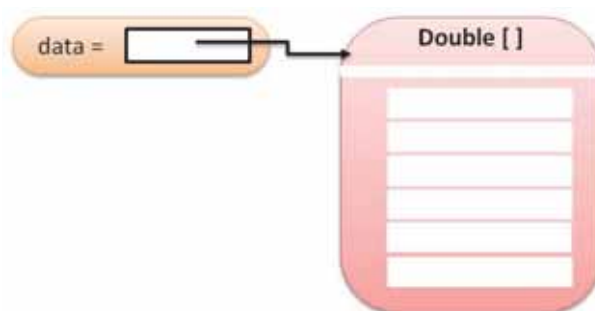
تعداد عناصر آرایه (که در مثال بالا، 10 است) طول آرایه نامیده می‌شود.

آرایه ترتیبی از مقادیر با نوع یکسان است.

عملگر new تنها یک آرایه ایجاد می‌کند. در صورتی که در جلوی نوع متغیر از علامت [] استفاده کنید، آرایه‌ای از نوع تعریف شده، ایجاد می‌شود. در مثال بالا علامت‌های [] پس از عبارت double نشان‌دهنده این است که نوع داده‌ای¹ تعریف شده، آرایه است و عناصر آرایه همگی اعداد اعشاری از نوع double هستند. دستور کامل برای تعریف و تخصیص حافظه به متغیر آرایه‌ای در جاوا به صورت زیر است:

```
double [ ] data = new double [10];
```

به عبارت دیگر data به آرایه‌ای از اعداد اعشاری اشاره می‌کند (شکل 1-7 را مشاهده نمایید).



شکل 1-7 آرایه‌ای از عناصر و ارجاع به آن

می‌توانید آرایه‌ای از آبجکت‌ها را قالب‌بندی کنید، به‌عنوان مثال:

```
BankAccount [ ] accounts = new BankAccount [10];
```

هنگامی که آرایه‌ای ایجاد می‌شود، تمامی عناصر آن با مقدار صفر (برای آرایه‌هایی از نوع int [] یا double []، false (برای آرایه از نوع boolean [] یا null (برای آرایه‌ای از آبجکت‌ها و رشته‌ها)، مقداردهی اولیه می‌شوند.

¹ Data Type

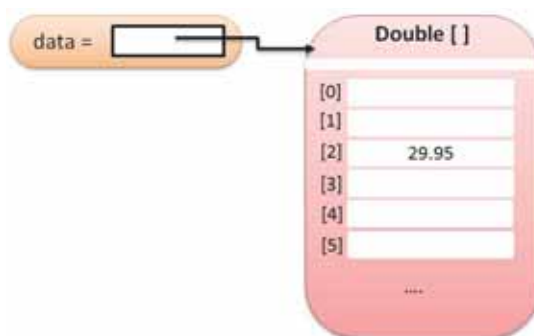
هریک از عناصر آرایه با استفاده از اندیس¹ که داخل براکت‌ها قرار می‌گیرد، مشخص می‌شود (index). برای مثال عبارت زیر نشان دهنده داده با اندیس چهارم آرایه می‌باشد:

```
data [4]
```

می‌توان مقداری را به عنصری دلخواه به صورت زیر تخصیص داد:

```
data [2] = 29.95;
```

اکنون مکان سوم آرایه (یعنی جایی که اندیس آن 2 است)² حاوی مقدار 29,95 است (شکل 7-2 را مشاهده نمایید).



شکل 7-2 نگهداری مقدار در آرایه

به وسیله یک اندیس صحیح به صورت `a[i]` می‌توان به عناصر آرایه دسترسی داشت.

برای بازخوانی داده اندیس دوم به وسیله عبارت `data [2]` می‌توان عمل نمود:

```
System.out.println ("The value of this data item is:" + data [2]);
```

در صورتی که با دقت به شکل 7-2 نگاه کنید، متوجه می‌شوید که اولین اندیس با صفر شروع می‌شود. به جملات زیر توجه کنید:

- `data [0]` اولین عنصر آرایه است.
- `data [1]` دومین عنصر آرایه است.

¹ عدد صحیحی که گاهی شمارنده نیز نامیده می‌شود.

² توجه کنید که چون آرایه‌ها با اندیس صفر شروع می‌شوند، مکانی که اندیس آن 2 است، عنصر سوم آرایه است.

- [2] data سومین عنصر آرایه است.
- ...

در مثال اول که آرایه‌ای با 10 عنصر ایجاد کردیم، آخرین عنصر آرایه دارای اندیس 9 است ([9] data). در صورتی که بخواهید به عنصری که وجود ندارد، دسترسی داشته باشید، برنامه وارد یکی از استثناها¹ می‌شود. برای مثال دستور زیر خطای محدوده² می‌باشد:

```
data [10] = 29.95; // ERROR
```

محدوده اندیس آرایه‌ها از 0 تا length-1 است.

برای جلوگیری از رخ دادن خطای محدوده باید بدانید که آرایه دارای چند عنصر است. دستور length تعداد عناصر آرایه را محاسبه می‌کند: data.length (این دستور تعداد عناصر آرایه data را محاسبه می‌کند).

توجه کنید که عدم وجود پرانتزها پس از length نشان‌دهنده این است که متغیری نمونه از نوع آبجکت آرایه می‌باشد و متد نمی‌باشد؛ به عبارت دیگر length متغیر نمونه از نوع final public است. البته این نوع استفاده کاملاً نامتعارف می‌باشد. برنامه‌نویسان جاوا معمولاً از متد برای بررسی ویژگی‌های آبجکت‌ها استفاده می‌کنند. ولی به یاد داشته باشید که در این حالت خاص باید از قراردادن پرانتز در مقابل دستور length پرهیز نمایید.

جهت پی بردن به تعداد عناصر آرایه از فیلد length استفاده کنید.

در دستور زیر، در صورتی که اندیس یعنی متغیر i در محدوده مجاز (legal bound) باشد، اجازه دسترسی به آرایه را خواهید داشت:

```
if (0 <= i && i < data.length) data [i] = value;
```

در صورتی که آرایه‌ای ایجاد کنید که دارای 10 عنصر است و در ادامه برنامه متوجه شوید که نیازمند آرایه‌ای با طول بیشتر (عناصر بیشتر)، هستید می‌توانید آرایه‌ای با طول بیشتر ایجاد کنید و مقادیر آرایه قبلی را در آن کپی نمایید. در مورد جزئیات این اعمال در بخش 7,7 بحث خواهیم نمود.

¹ Exception

² Bound Error

خودآزمایی

1. عناصر آرایه `data` پس از انجام دستورات زیر حاوی چه مقادیری خواهند بود؟

```
double [ ] data = new double [10];
for (int i = 0; i < data.length; i++) data[i] = i*i;
```

2. تکه برنامه‌های زیر چه چیزی را چاپ می‌کنند؟ یا در صورتی که خطایی رخ می‌دهد، علت خطا را

تشریح نمایید و تشخیص دهید که این خطا مربوط به زمان کامپایل است یا زمان اجرا؟

- A. `double[] a = new double[10];`
`System.out.println(a[0]);`
- B. `double[] b = new double[10];`
`System.out.println(b[10]);`
- C. `double[] c;`
`System.out.println(c[0]);`

دستورالعمل 7,1: ایجاد آرایه

نحوه ایجاد یک آرایه از عناصر به طول `length`:

```
New typeName [length]
```

مثال:

```
New double [10]
```

هدف:

ایجاد آرایه‌ای از عناصر به طول `length` جهت نگهداری انواع داده‌ای مختلف.

دستورالعمل 7,2: دسترسی به عناصر آرایه

دستور نحوه دسترسی به آرایه:

```
arrayReference [index]
```

مثال:

```
data [2]
```

هدف:

جهت دسترسی به عناصر آرایه.

خطای رایج 7,1: خطاهای محدوده (Bound Errors)

رایج‌ترین خطا در آرایه‌ها، درخواست دسترسی به موقعیت یا عنصر ناموجود است.

```
double[ ] data = new double[10];
data[10] = 29.95;
// Error-only have elements with index values 0 ... 9
```

در صورتی که محدوده اندیس رعایت نشود، هنگامی که برنامه اجرا می‌شود وارد یک استثناء شده و برنامه پایان می‌پذیرد.

این خطا در زبان‌هایی چون C و C++ بهبود یافته‌اند. در این زبان‌ها هیچ پیام خطایی مشاهده نمی‌شود، بلکه به‌طور کاملاً شگفت‌آوری تنها ابتدای آرایه تخریب می‌گردد (ممکن است ابتدای آرایه آزاد شود). چنین خطاهایی بسیار جدی بوده و اشکالیابی برنامه‌های C و C++ را بسیار مشکل می‌سازد.

خطای رایج 7,2: آرایه‌های مقداردهی نشده

خطای رایج در آرایه‌های مقداردهی نشده، مقداردهی اولیه به آن‌ها می‌باشد. به دستورات زیر توجه کنید:

```
double [ ] data;
data [0] = 29.95; //Error-data not initialized
```

متغیرهای آرایه دقیقاً مانند متغیرهای آبجکتی (آبجکت‌ها) عمل می‌کنند؛ آنها تنها ارجاعی به آرایه واقعی هستند. برای ایجاد آرایه واقعی باید از عملگر new استفاده کنید:

```
Double [ ] data = new double [10];
```

مبحث پیشرفته 7,1: مقداردهی اولیه به آرایه‌ها

همان‌گونه که ذکر شد برای ایجاد آرایه در ابتدا به‌وسیله عملگر new این کار را انجام می‌دهید، سپس می‌توانید مقادیر مختلف را مانند مثال زیر به عناصر اختصاص دهید:

```
int [ ] primes = new int[5];
primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
```



```
primes[4] = 11;
```

البته روش ساده‌تر و سریع‌تری نیز برای مقداردهی به عناصر آرایه وجود دارد که به صورت زیر می‌باشد:

```
int[] primes = { 2, 3, 5, 7, 11 };
```

در این روش کامپایلر جاوا تعداد عناصری که می‌خواهید وارد نمایید را شمارش نموده و در صورتی که تعداد آنها از تعداد کل عناصر آرایه بیشتر نباشد، به همان ترتیبی که وارد نموده‌اید، آنها را به خانه‌ها و عناصر آرایه اختصاص می‌دهد.

در صورتی که بخواهید آرایه‌ای را در هنگام تعریف نمودن مقداردهی نمایید و عناصر آن را در متدی به کار برید، می‌توانید به صورت زیر عمل نمایید:

```
new int[] { 2, 3, 5, 7, 11 }
```

7,2 لیست‌های آرایه‌ای

در این بخش، کلاس ArrayList را معرفی می‌کنیم. این کلاس به شما اجازه می‌دهد مجموعه‌ای از آبجکت‌ها را انتخاب نمایید. لیست‌های آرایه‌ای از دو لحاظ دارای اهمیت خاصی هستند:

- لیست‌های آرایه‌ای در صورت نیاز کوچک یا بزرگ می‌شوند
 - کلاس ArrayList متدهای متعددی را فراهم می‌کند، مانند الحاق نمودن یا حذف نمودن عناصر
- در اینجا می‌خواهیم یک لیست آرایه‌ای برای حساب‌های بانکی ایجاد کنیم و عناصر آن را با آبجکت‌ها پر کنیم. (کلاس BankAccount مربوط به فصل سوم – در جلد اول کتاب – را ارتقاء می‌دهیم. در این کلاس هر حساب کاربری دارای یک شماره کاربری است.)

```
ArrayList<BankAccount> accounts = new ArrayList<BankAccount> ();
accounts.add (new BankAccount (1001));
accounts.add (new BankAccount (1015));
accounts.add (new BankAccount (1022));
```

کلاس ArrayList مجموعه‌ای از آبجکت‌ها را مدیریت می‌کند.

نوع ArrayList<BankAccount> یک لیست آرایه‌ای به نام Bank Account تعریف می‌کند. براکت‌های گوشه‌دار در اطراف BankAccount مشخص می‌کنند که این نوع داده از نوع داده پارامتری است که اصطلاحاً به آن Type Parameter می‌گویند. می‌توانید نام BankAccount را با نام هر کلاس دیگری که

نیازمند لیست آرایه‌ای است جایگزین نمایید، به همین دلیل ArrayList کلاس ژنریک یا ¹Generic Class نامیده می‌شود. در حال حاضر برای شروع کار می‌توانید از دستوری مانند `<T>` ArrayList برای انتخاب آبجکت‌های از نوع T استفاده کنید. به خاطر داشته باشید که نمی‌توان از انواع اولیه به‌عنوان Type Parameter استفاده کرد؛ یعنی انواع به‌صورت زیر را نداریم:

- `ArrayList<double>`
- `ArrayList<int>`

کلاس ArrayList از نوع کلاس ژنریک است: `<T>` ArrayList آبجکت‌های از نوع T را برمی‌گزیند.

هنگامی که آبجکت ArrayList را ایجاد می‌کنید، اندازه آن صفر است. باید از متد `Add` که برای اضافه کردن آبجکت جدید استفاده می‌شود، استفاده کنید. پس از هر بار فراخوانی متد `Add` اندازه آبجکت تغییر کرده و بزرگتر می‌شود. متد `Size` مقدار اندازه لیست آرایه‌ای ایجاد شده را تعیین می‌کند.

برای استفاده از آبجکت‌های لیست‌های آرایه‌ای از متد `get` استفاده کنید، توجه کنید که استفاده از عملگر `[]` صحیح نمی‌باشد. لیست‌های آرایه‌ای نیز مانند آرایه‌ها با اندیس صفر شروع می‌شوند.

برای مثال `accounts.get(2)` اندیس دوم لیست آرایه‌ای را برمی‌گرداند، یعنی عنصر سوم آن:

```
BankAccount anAccount = accounts.get(2);
```

مشابه با آرایه‌ها در صورتی که به عنصری در لیست آرایه‌ای اشاره کنید که وجود نداشته باشد، خطا رخ می‌دهد. رایج‌ترین خطای محدوده به‌صورت زیر است:

```
int i = accounts.size();
anAccount = accounts.get(i); // Error
```

آخرین اندیس مجاز جهت استفاده `accounts.size() - 1` است.

جهت تنظیم نمودن مقدار عناصر لیست آرایه‌ای با مقداری جدید، از متد `set` استفاده کنید:

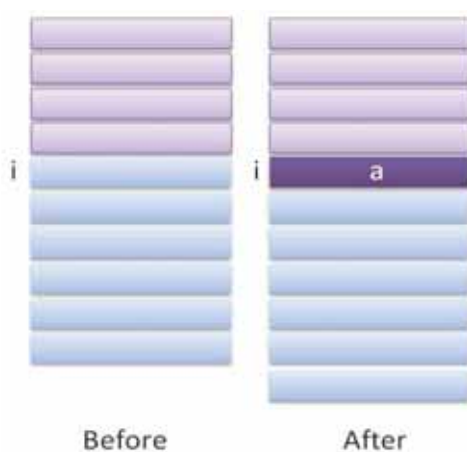
```
BankAccount anAccount = new BankAccount (1729);
accounts.set (2 , anAccount);
```

با فراخوانی متد `set` در دستور بالا مکان دوم لیست آرایه‌ای `accounts` را معادل `anAccount` قرار می‌دهد؛ به‌وسیله این متد، مقدار جدید جایگزین مقدار قبلی می‌شود.

متد `set` تنها مقدار جدید دریافتی را جایگزین مقدار قبلی می‌کند. این متد با متد `Add` متفاوت عمل می‌کند، زیرا در متد `Add` آبجکت جدیدی به انتهای لیست آرایه‌ای افزوده می‌شود.

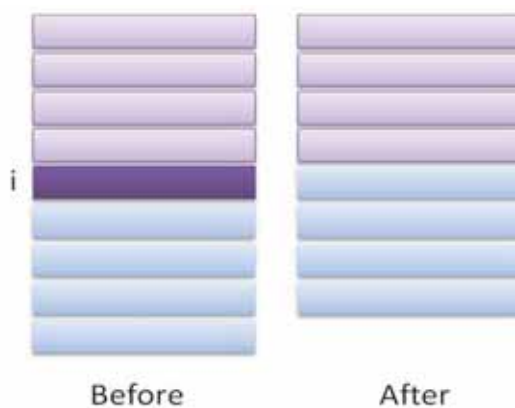
¹ در رابطه با کلاس ژنریک و انواع ژنریک به‌طور مفصل در فصل 17 بحث خواهد شد.

همچنین می‌توانید آبجکت جدیدی را به وسط لیست آرایه‌ای بیافزایید. با فراخوانی دستور `accounts.add(i, a)` آبجکت جدیدی در مکان `i` ام حافظه اضافه شده و تمامی عناصر پس از آن به اندازه یک واحد شیفت داده می‌شوند. در نهایت با استفاده از متد `Add`، اندازه لیست آرایه‌ای یک واحد افزایش می‌یابد (شکل 7-3 را مشاهده کنید).



شکل 7-3 اضافه نمودن عنصری جدید به مرکز لیست آرایه‌ای

همچنین بر عکس عمل بالا، به وسیله دستور `accounts.remove(i)` عنصر `i` ام از لیست حذف می‌شود و تمامی عناصر پس از آن به اندازه یک واحد شیفت می‌یابند و همچنین اندازه لیست آرایه‌ای به اندازه یک واحد کاهش می‌یابد (شکل 7-4 را مشاهده نمایید).



شکل 7-4 حذف یک عنصر از وسط لیست آرایه‌ای

برنامه زیر نمونه بارزی برای تشریح متدهای کلاس ArrayList ارائه می‌دهد. توجه کنید که باید کلاس ژنریک مقابل را به ابتدای برنامه بیافزایید: `java.util.ArrayList`

ch07/arraylist/ArrayListTester.java

```

1 import java.util.ArrayList;
2
3 /**
4  This program tests the ArrayList class. 5 */
6 public class ArrayListTester
7 {
8     public static void main(String[] args)
9     {
10        ArrayList<BankAccount> accounts
11        = new ArrayList<BankAccount>();
12        accounts.add(new BankAccount(1001));
13        accounts.add(new BankAccount(1015));
14        accounts.add(new BankAccount(1729));
15        accounts.add(1, new BankAccount(1008));
16        accounts.remove(0);
17
18        System.out.println("Size: " + accounts.size());
19        System.out.println("Expected: 3");
20        BankAccount first = accounts.get(0);
21        System.out.println("First account number: "
22        + first.getAccountNumber());
23        System.out.println("Expected: 1008");
24        BankAccount last = accounts.get(accounts.size() - 1);
25        System.out.println("Last account number: "
26        + last.getAccountNumber());
27        System.out.println("Expected: 1729");
28    }
29 }

```

ch07/arraylist/BankAccount.java

```

1 /**
2  A bank account has a balance that can be changed by
3  deposits and withdrawals.
4  */
5 public class BankAccount
6 {
7     /**
8     Constructs a bank account with a zero balance.
9     @param anAccountNumber the account number for this account
10    */
11    public BankAccount(int anAccountNumber)
12    {
13        accountNumber = anAccountNumber;
14        balance = 0;

```

```
15 }
16
17 /**
18  Constructs a bank account with a given balance.
19  @param anAccountNumber the account number for this account
20  @param initialBalance the initial balance
21  */
22 public BankAccount(int anAccountNumber, double initialBalance)
23 {
24     accountNumber = anAccountNumber;
25     balance = initialBalance;
26 }
27
28 /**
29  Gets the account number of this bank account.
30  @return the account number
31  */
32 public int getAccountNumber()
33 {
34     return accountNumber;
35 }
36
37 /**
38  Deposits money into the bank account.
39  @param amount the amount to deposit
40  */
41 public void deposit(double amount)
42 {
43     double newBalance = balance + amount; 44     balance = newBalance;
45 }
46
47 /**
48  Withdraws money from the bank account.
49  @param amount the amount to withdraw
50  */
51 public void withdraw(double amount)
52 {
53     double newBalance = balance - amount;
54     balance = newBalance;
55 }
56
57 /**
58  Gets the current balance of the bank account.
59  @return the current balance
60  */
61 public double getBalance()
62 {
63     return balance;
64 }
```

```

65
66 private int accountNumber;
67 private double balance;
68 }

```

Output

خروجی

```

Size: 3
Expected: 3
First account number: 1008
Expected: 1008
Last account number: 1729
Expected: 1729

```

خودآزمایی

3. چگونه آرایه‌ای حاوی 10 رشته ایجاد می‌کنید؟ لیست آرایه‌ای از رشته‌ها را چگونه ایجاد می‌کنید؟

4. پس از اجرای دستورات زیر محتوای عناصر لیست آرایه‌ای چه خواهد بود؟

```

ArrayList <String> names = new ArrayList <String> ();
names.add("A");
names.add(0, "B");
names.add("C");
names.remove(1);

```

خطای رایج 7,3: طول و اندازه

متأسفانه¹ Syntax جاوا در تشخیص تعداد عناصر آرایه‌ها، لیست‌های آرایه‌ای و رشته‌ها مشکلاتی دارد، به بیانی دیگر با همه انواع داده‌ای سازگار نیست. تنها کفایت به یاد داشته باشید که برای هر نوع داده‌ای Syntax آن را اصلاح کنید. در جدول زیر نحوه شمارش و به‌دست آوردن تعداد عناصر را در هر کدام از انواع آرایه‌ها، لیست‌های آرایه‌ای و رشته‌ها مشاهده می‌کنید:

¹ دستوری یک زبان خاص را Syntax آن زبان می‌گویند. به عبارت دیگر Syntax قواعدی است که بر روی ساختار و محتویات عبارت‌ها و دستورات اعمال می‌گردد.

	Data Type	Number of Elements
آرایه	Array	a.length
لیست آرایه‌ای	Array list	a.size()
رشته	String	a.length()

نکته مدیریتی 7,1: پارامتری نمودن لیست‌های آرایه‌ای

لیست‌های آرایه‌ای پارامتری شده – مانند `<BankAccount>ArrayList` – مختص جاوا در سال 2004 معرفی و مطرح شدند. نسخه‌های قبل از نسخه 5,0 جاوا تنها دارای کلاس غیر معمول `ArrayList` بودند. در لیست‌های آرایه‌ای غیر معمول می‌توان عناصر هر کلاسی را نگهداری نمود. (از لحاظ فنی می‌توان گفت عناصر از نوع آجکت را نگهداری می‌کند.)

هرگاه عنصری را از لیست آرایه‌ای بازایی می‌کنید، کامپایلر مستلزم استفاده از عمل `Cast` (عملی که در آن داده‌ای به قالب مطلوب مبدل می‌شود)، می‌باشد:

```
ArrayList accounts = new ArrayList(); // Untyped ArrayList
accounts.add(new BankAccount(1729)); // OK—can add any object
BankAccount a = (BankAccount) a.get(0); // Need cast
```

عمل `Cast` به این دلیل لازم است که کامپایلر، آجکت‌هایی که در لیست آرایه‌ای نگهداری می‌شوند را بررسی نمی‌کند و نوع آجکت به وسیله متد `get` ارجاع داده می‌شود.

لیست‌های آرایه غیر معمول هنوز هم قسمتی از زبان جاوا را تشکیل می‌دهند، به همین دلیل می‌خواهیم با برنامه‌هایی که قبل از 2004 نوشته شده‌اند، ادامه دهیم؛ اما شما در برنامه‌های جدیدی که می‌نویسید نباید از آن استفاده کنید. استفاده از `cast` کمی خسته کننده و همچنین متمایل به خطا می‌باشد. در صورتی که روش `Cast` را به کار برید، کامپایلر نمی‌تواند اشتباهات احتمالی شما را تشخیص دهد و برنامه در هنگام اجرا وارد یکی از استثناها می‌شود.

7,3 بسته بندی‌ها و Auto-Boxing

به دلیل اینکه اعداد در جاوا آجکت نیستند نمی‌توان مستقیماً آنها را در لیست‌های آرایه‌ای قرار داد. برای مثال ترکیب زیر را پیاده کنید:

- `ArrayList <double>`

برای نگهداری ترتیبی از اعداد در لیست‌های آرایه‌ای باید آن‌ها را به وسیله کلاس‌های بسته‌بندی، به آبجکت‌ها تغییر دهید.

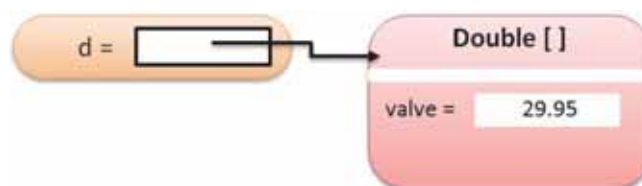
برای اینکه با مقادیر اولیه به عنوان آبجکت رفتار شود، باید از کلاس‌های بسته‌بندی استفاده کنید.

برای هر کدام از هشت نوع اولیه کلاس‌های بسته‌بندی وجود دارد:

نوع اولیه (Primitive Type)	کلاس بسته‌بندی (Wrapper Class)
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

توجه کنید که نام کلاس‌های بسته‌بندی با حروف بزرگ شروع می‌شوند و در دو مورد نام آنها با هم متفاوت می‌باشد: Integer , Character

هر کلاس بسته‌بندی حاوی مقدار معادل نوع اولیه آن می‌باشد. برای مثال آبجکت کلاس Double حاوی مقدار نوع double است (شکل 5-7 را مشاهده کنید).



شکل 5-7 آبجکتی از کلاس بسته‌بندی

کلاس‌های بسته‌بندی در کلیه مکان‌هایی که به جای انواع اولیه ملزم به استفاده از آبجکت هستید، قابل استفاده است. برای مثال می‌توانید مجموعه‌ای از اعداد اعشاری را در ساختاری به صورت زیر استفاده کنید:

✓ ArrayList<Double>

در نسخه Java 5.0 به بعد، تبدیل از انواع اولیه به نوع متناظر آن در کلاس بسته‌بندی به صورت خودکار انجام می‌شود؛ این فرآیند را Auto-Boxing می‌گویند.

برای مثال اگر عددی را به آبجکت Double تخصیص دهید، عدد به صورت خودکار داخل یک جعبه - Box - نگهداری می‌شود که آبجکت بسته‌بندی نامیده می‌شود.

```
Double d = 29.95; // auto-boxing; same as Double d = new Double(29.95);
```

اگر از نسخه قدیمی جاوا استفاده می‌کنید، باید خودتان برای آن سازنده نیز تعریف کنید.

در ویرایش‌های قدیمی‌تر باید متدهایی مانند intValue, doubleValue, یا booleanValue را نیز برای unbox نمودن فراخوانی کنید.

عمل Auto-Boxing را حتی برای عبارتهای منطقی نیز می‌توانید استفاده کنید. برای مثال دستور زیر را در نظر بگیرید:

```
Double e = d + 1;
```

این دستور کاملاً مجاز می‌باشد و معنای آن به شرح زیر است:

- متغیر d را به کلاس Double، Auto-unbox می‌کند
 - عدد 1 را به آن می‌افزاید
 - حاصل را به آبجکت جدید Double، Auto-box می‌نماید
 - آدرس مرجع آن را به آبجکت بسته‌بندی جدید به نام e تخصیص می‌دهد
- در صورتی که از نسخه 5,0 و بالاتر جاوا استفاده می‌کنید، لیست‌های آرایه‌ای سراسرتر و راحت‌تر خواهند بود. به خاطر داشته باشید که برای تعریف لیست آرایه‌ای و Auto-boxing از نوع بسته‌بندی استفاده کنید:

```
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);
double x = data.get(0);
```

در نسخه‌های قدیمی جاوا استفاده از کلاس‌های بسته‌بندی برای نگهداری اعداد در لیست آرایه‌ای، اذیت‌کننده و مشکل می‌باشد زیرا باید تمام مقادیر را به صورت دستی box یا unbox نمود.

مهم نیست که از کدام نسخه جاوا استفاده می‌کنید بلکه باید بدانید نگهداری اعداد در بسته‌بندی‌ها کاملاً ناکارا خواهد بود. استفاده از بسته‌بندی برای لیست‌های آرایه‌ای غیر قابل قبول است، اما برای مجموعه وسیعی از اعداد یا کاراکترها باید از آرایه‌ها استفاده کنید.

خودآزمایی

5. تفاوت میان انواع double و Double چیست؟
6. فرض کنید داده‌ی داخل ساختار ArrayList<Double> دارای اندازه بزرگتر از صفر است، چگونه عنصر با اندیس 0 را افزایش می‌دهید؟

7,4 for بهبود یافته

در نسخه 5,0 جاوا میانبرهای بسیار مناسبی برای حلقه‌ها معرفی شدند. اغلب اوقات مجبور می‌شوید که برای مقداردهی یا کار با عناصر آرایه و لیست آرایه‌ای، عملیاتی را تکرار کنید. استفاده از for بهبودیافته باعث می‌شود که این فرآیند در برنامه‌ها به صورت ساده‌تر و راحت‌تری صورت پذیرد.

فرض کنید می‌خواهید مجموع تمامی داده‌های یک آرایه را محاسبه کنید، در اینجا نحوه انجام این کار را به وسیله حلقه for بهبودیافته بیان می‌کنیم:

```
double[] data = ... ;
double sum = 0;
for (double e : data)
{
    sum = sum + e;
}
```

بدنه حلقه به تعداد تمامی عناصر در آرایه اجرا می‌شود. در هر بار اجرای مجدد حلقه، مقدار عنصر بعدی به متغیر e تخصیص داده می‌شود. سپس بدنه حلقه اجرا می‌شود. این حلقه به صورت زیر خوانده می‌شود: "for each e در داده‌ها"

For بهبودیافته معبری برای تمامی عناصر یک مجموعه است.

شاید تعجب کنید که چرا جاوا به شما اجازه نمی‌دهد به صورت "for each (e در داده‌ها)" بنویسید. مسلماً این فرم بسیار مرتب‌تر خواهد بود و طراحان جاوا به طور جدی آن را مد نظر قرار داده‌اند. به هر حال، فرم "برای هر" در جاوا چند سال پس از انتشار اولیه آن اضافه شد. کلمات کلیدی¹ جدید همواره به زبان‌ها اضافه می‌شوند، با این وجود برنامه‌های قدیمی که از شناسه‌ها و کلمات کلیدی قبلی استفاده کرده‌اند (مانند System.in)، دیگر به درستی کار نخواهند کرد.

¹ Keywords

شما ملزم به استفاده از ساختار "for each" برای تمامی عناصر آرایه نیستید. می‌توانید از همان حلقه ساده for سابق و یک متغیر برای مدیریت اندیس‌ها استفاده کنید:

```
double[] data = ... ;
double sum = 0;
for (int i = 0; i < data.length; i++)
{
    double e = data[i];
    sum = sum + e;
}
```

در حلقه بالا مقادیر عناصر data[0]، data[1]، ... به ترتیب به متغیر e تخصیص داده می‌شوند، همچنین در حلقه for، اندیس i، مقادیر صفر، یک، دو و ... را به ترتیب تخصیص می‌دهد.

همواره می‌توانید از for بهبودیافته برای بررسی تمامی عناصر لیست آرایه‌ای استفاده کنید. برای مثال حلقه زیر مقدار کل تمامی حساب‌ها را محاسبه می‌کند:

```
double sum = 0;
for (BankAccount a : accounts)
{
    sum = sum + a.getBalance();
}
```

این حلقه دقیقاً هم‌ارز و مشابه حلقه زیر عمل می‌کند:

```
double sum = 0;
for (int i = 0; i < accounts.size(); i++)
{
    BankAccount a = accounts.get(i);
    sum = sum + a.getBalance();
}
```

حلقه "for each" جهت انجام هدف بسیار خاصی استفاده می‌شود:

- پیمودن و بررسی عناصر یک مجموعه از ابتدا تا انتهای آن.

البته برخی اوقات نیازی به بررسی تمامی عناصر از ابتدا تا انتها نداریم و یا ممکن است نیاز باشد عناصر از انتها به ابتدا بررسی شوند، در این مواقع حتماً باید از حلقه for معمولی استفاده کنید.

خودآزمایی

7. حلقه‌ای با ساختار "for each" بنویسید که تمامی عناصر آرایه را چاپ نماید.

8. چرا حلقه "for each" میانبر مناسبی برای حلقه معمولی زیر نمی‌باشد؟

```
for (int i = 0; i < data.length; i++) data[i] = i * i ;
```

دستورالعمل 7,3

```
for (Type variable : collection)
    statement
```

مثال:

```
for (double e : data)
    sum = sum + e;
```

هدف:

اجرای (یا اداره نمودن) حلقه برای هر عنصر در یک مجموعه از داده‌ها. در هر بار تکرار، متغیر به عنصر بعدی مجموعه تخصیص می‌یابد، سپس اجرای حلقه خاتمه می‌یابد.

7,5 الگوریتم‌های ساده در آرایه‌ها

الف) شمارش تطابق

برای شمارش مقادیر در لیست آرایه‌ای، تمامی عناصر را بررسی کنید و مقادیر مطابق را شمارش نمایید تا زمانی که به انتهای لیست آرایه‌ای می‌رسید.

فرض کنید می‌خواهید بدانید که چه مقدار از حساب‌ها از نوع یکسانی هستند. برای این کار عناصر مجموعه را بررسی کرده و در صورت مشاهده تطابق دو نوع حساب، یک واحد به شمارنده اضافه می‌کنیم. در مثال زیر می‌خواهیم تعداد حساب‌هایی که مقدار سپرده آنها از مقدار آستانه‌ای بیشتر باشد را محاسبه کنیم:

```
public class Bank
{
    public int count(double atLeast) {
        int matches = 0;
        for (BankAccount a : accounts)
        {
            if (a.getBalance() >= atLeast) matches++;
            // Found a match
        }
        return matches;
    }
    ...
    private ArrayList<BankAccount> accounts;
}
```

ب) پیدا کردن یک مقدار

فرض کنید می‌خواهید بدانید چه تعداد از حساب‌های بانکی، مشابه حساب بانکی شماست. توجه کنید که ممکن است حلقه در پیدا کردن پاسخ ناموفق عمل کند. این نوع جستجو به جستجوی خطی معروف است:

```
public class Bank
{
    public BankAccount find(int accountNumber)
    {
        for (BankAccount a : accounts)
        {
            if (a.getAccountNumber() == accountNumber) // Found a match
                return a;
        }
        return null; // No match in the entire array list
    }
    ...
}
```

دقت کنید که متد در صورت عدم یافتن تطابق در حساب‌ها، مقدار null را برمی‌گرداند.

برای پیدا کردن یک مقدار در لیست آرایه‌ای، تمامی عناصر لیست را بررسی کنید.

ج) پیدا کردن مقدار ماکزیمم یا مینیمم¹

فرض کنید می‌خواهید حسابی که دارای بیشترین مقدار سپرده است را بیابید. یک متغیر به‌عنوان کاندیدا (Candidate) تعریف کنید. در صورتی که عنصر دارای بیشترین مقدار سپرده را یافتید، آن را در متغیر کاندیدا جایگزین نمایید. هنگامی که به انتهای لیست آرایه‌ای رسیدید، مقدار نهایی ماکزیمم، مشخص می‌شود.

برای محاسبه ماکزیمم یا مینیمم (بیشترین یا کمترین) مقدار در لیست آرایه‌ای، متغیری به‌عنوان کاندیدا (candidate) تعریف کنید و آن را با اولین عنصر لیست مقارنه‌ی اولیه نمایید. سپس کاندیدا را با دیگر عناصر لیست مقایسه نموده تا مقدار نهایی ماکزیمم یا مینیمم را بیابید.

```
BankAccount largestYet = accounts.get(0);
for (int i = 1; i < accounts.size(); i++)
{
    BankAccount a = accounts.get(i);
    if (a.getBalance() > largestYet.getBalance())
```

¹ Maximum or minimum

```

    largestYet = a;
}
return largestYet;

```

در اینجا مشاهده می‌کنید که برای عمل مقایسه و پیدا نمودن مقدار ماکزیمم، از حلقه for و یک if استفاده کرده‌ایم.

البته با این رویکرد باید حداقل یک عنصر در لیست موجود باشد، در غیر این صورت باید دستوری برای بازگردانی مقدار null گنجانده شود:

```

if (accounts.size() == 0) return null;
BankAccount largestYet = accounts.get(0);
...

```

در تمرینات 7,6 و 7,5 اصلاحات جزئی برای این الگوریتم مشاهده خواهید نمود.

برای محاسبه و یافتن مقدار مینیمم (کوچکترین) در میان مجموعه‌ای از داده‌ها، مانند مثال قبلی تغییری به‌عنوان کاندیدا تعریف کنید و هنگامی که مقداری کوچکتر از آن را در مجموعه یافتید، آن را در کاندیدا جایگزین نمایید.

برنامه زیر نمونه‌ای از کلاس Bank است که یک لیست آرایه‌ای از حساب‌های بانکی را نگهداری می‌کند. متدهای به‌کار رفته در کلاس Bank الگوریتم‌هایی هستند که در این بخش معرفی شدند.

```

ch07/bank/Bank.java
1 import java.util.ArrayList;
2
3 /**
4  This bank contains a collection of bank accounts.
5  */
6 public class Bank
7 {
8  /**
9   Constructs a bank with no bank accounts.
10  */
11  public Bank()
12  {
13   accounts = new ArrayList<BankAccount>();
14  }
15
16  /**
17   Adds an account to this bank.
18   @param a the account to add
19  */
20  public void addAccount(BankAccount a)
21  {
22   accounts.add(a);
23  }

```