آموزش برنامه نویسی جاوا

جلد 2

ترجمه و تألیف: انور پوراحمد

انتشارات پندار پارس

سرشناسه : پوراحمد، انور، 1369 -
عنوان و نام پدیدآور : آموزش برنامه‌نویسی جاوا/ انور پوراحمد.
مشخصات نشر : تهران: پندار پارس: مانلی،1390 -
مشخصات ظاهری : ج.: مصور، جدول. (ج1: 400 ص، ج2: 472ص)
شابک : 120000ریال با لوح فشرده: جلد دوم: 978-964-2989-67-6 دوره: 978-964-2989-68-3
وضعیت فهرست نویسی : فیپا
موضوع : جاوا (زبان برنامه‌نویسی کامپیوتر)
رده بندی کنگره : 76/73QA /ج2پ9 1390
رده بندی دیویی : 005/133
شماره کتابشناسی ملی : 8425032

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوی رشتچی، شماره 14، واحد 16 **www.pendarepars.com**
تلفن: 66572335 - تلفکس: 66926578همراه: 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 : **ترام‌سنج، صالحان، خیام**
قیمت : **12000 تومان به همراه CD** **شابک** : 978-964-2989-67-6
شابک دوره : 978-964-2989-68-3

پیشگفتار

کتاب پیش‌روی، جلد دوم کتاب آموزش برنامه‌نویسی جاوا می‌باشد که ترجمه فصل‌های 7 تا 14 کتاب دکتر کای هورستمن[1] 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 کامل-ترین 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

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 صورتی‌که 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]
```

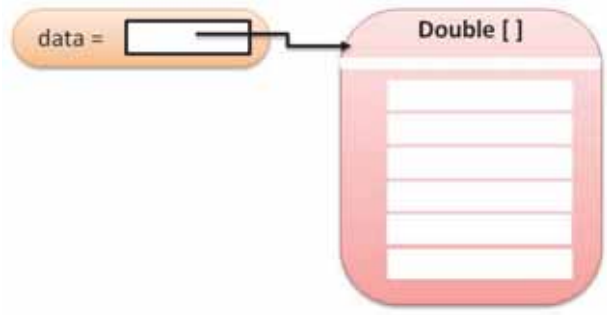
تعداد عناصر آرایه (که در مثال بالا، 10 است) *طول آرایه* نامیده می‌شود.

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 نشان-دهنده این است که نوع داده‌ای[1] 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 به‌صورت 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];
```

هنگامی‌که 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

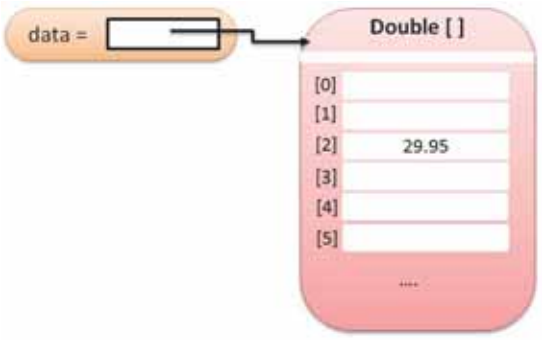
هریک از عناصر آرایه با استفاده از اندیس[1] که داخل براکت‌ها قرار می‌گیرد، مشخص می‌شود ([index]). برای مثال عبارت زیر نشان دهنده داده با اندیس چهارم آرایه می‌باشد:

```
data [4]
```

می‌توان مقداری را به عنصری دلخواه به‌صورت زیر تخصیص داد:

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

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



شکل 2-7 نگهداری مقدار در آرایه

به‌وسیله یک اندیس صحیح به‌صورت a[i] می‌توان به عناصر آرایه دسترسی داشت.

برای بازخوانی داده اندیس دوم به‌وسیله عبارت data [2] به‌صورت زیر می‌توان عمل نمود:

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

در صورتی که با دقت به شکل 2-7 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.

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

در مثال اول که آرایه‌ای با 10 عنصر ایجاد کردیم، آخرین عنصر آرایه دارای اندیس9 است(data [9]).

در صورتی‌که بخواهید به عنصری که وجود ندارد، دسترسی داشته باشید، برنامه وارد یکی از استثناها[1] می‌شود. برای مثال دستور زیر *خطای محدوده*[2] 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؛ به‌عبارت 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 نمی-شود، 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 به‌صورت 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 به‌کار برید، می‌توانید به‌صورت 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 Generic Class[1] نامیده می‌شود. در حال حاضر برای شروع کار می‌توانید از دستوری مانند ArrayList <T> 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: ArrayList< T > 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 هربار 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);
```

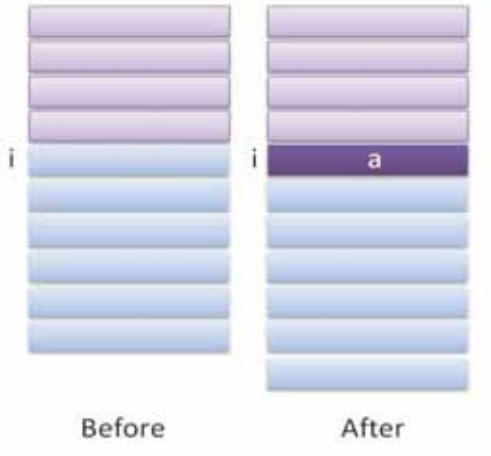
با فراخوانی متد set در دستور بالا مکان دوم لیست آرایه‌ای accounts را معادل anAccount قرار می‌دهد؛ به‌وسیله این متد، مقدار جدید جایگزین مقدار قبلی می‌شود.

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.

---

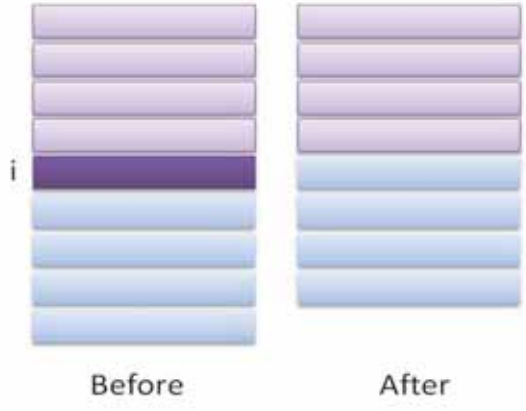
[1] در رابطه با کلاس ژنریک و انواع ژنریک به‌طور مفصل در فصل 17 بحث خواهد شد.

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 3-7 را مشاهده کنید).



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

همچنین بر عکس عمل بالا، به‌وسیله دستور accounts.remove(i) عنصر iام 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 4-7 را مشاهده نمایید).



شکل 4-7 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 }
```

خروجی Output

```
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

متأسفانه Syntax[1] جاوا در تشخیص تعداد عناصر آرایه‌ها، لیست‌های آرایه‌ای و رشته‌ها مشکلاتی دارد، به بیانی دیگر با همه انواع داده‌ای سازگار نیست. تنها کافیست 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] دستور یک زبان خاص را Syntax آن زبان می‌گویند. به عبارت دیگر Syntax قواعدی است که بر روی ساختار و محتویات عبارت‌ها و دستورات اعمال می گردد.

	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 ArrayList<BankAccount>- 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 آبجکت را نگهداری می‌کند.)

هرگاه عنصری را از لیست آرایه‌ای بازیابی 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، آبجکت‌هایی که در لیست آرایه‌ای نگهداری می‌شوند را بررسی نمی‌کند و نوع آبجکت به‌وسیله متد get ارجاع داده می‌شود.

لیست‌های آرایه غیر معمول هنوز هم قسمتی از زبان جاوا را تشکیل می‌دهند، به همین دلیل می-خواهیم 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 می-باشد. در صورتی که روش Cast را به‌کار 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 آبجکت 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 به‌وسیله کلاس‌های بسته‌بندی، به آبجکت‌ها تغییر دهید.

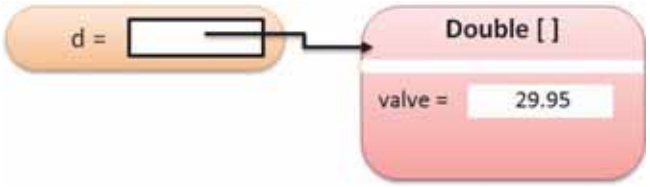
برای اینکه با مقادیر اولیه به‌عنوان آبجکت رفتار شود، باید از کلاس‌های بسته‌بندی استفاده کنید.

برای هرکدام 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 به‌صورت زیر استفاده کنید:

- ✓ ArrayList<Double>

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

برای مثال اگر عددی را به آبجکت Double تخصیص دهید، عدد به‌صورت 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 doubleValue, intValue 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 جدیدDouble، 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 سرراست‌تر و راحت-تر خواهند بود. به‌خاطر 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، اذیت کننده و مشکل می‌باشد زیرا باید تمام مقادیر را به‌صورت 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 مجبود 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 به‌صورت ساده‌تر و راحت‌تری صورت پذیرد.

فرض کنید می‌خواهید مجموع تمامی داده‌های یک آرایه را محاسبه کنید، در اینجا نحوه انجام این کار را به‌وسیله 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 هربار اجرای مجدد حلقه، مقدار عنصر بعدی به متغیر e تخصیص داده می‌شود. سپس بدنه حلقه اجرا می‌شود. این حلقه به‌صورت 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 به‌صورت "for each (e در داده‌ها)" بنویسید. مسلماً این فرم بسیار مرتب‌تر خواهد بود و طراحان جاوا به‌طور جدی آن را مد نظر قرارداده‌اند. به هر حال، فرم "برای هر" در جاوا چند سال پس از انتشار اولیه آن اضافه شد. کلمات کلیدی[1] 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 حلقهfor، 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 هم ارز 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 مینیمم[1]

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 مقداردهی 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 این‌صورت 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 متغیری 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    }
```