دستورات کنترلی

زبانهای برنامه نویسی از دستورات کنترلی استفاده می کنند تا جریان اجرای برنامه را پیشرفت داده و براساس تغییرات حالت یک برنامه شاخه هایی از آن برنامه منشعب نمایند . دستورات کنترلی برنامه در جاوا را می توان در طبقه بندی بعدی گنجاند : انتخاب (selection) ، تکرار (iteration)، و پرش . (jump) دستورات انتخاب به برنامه شما امکان میدهند تا مسیرهای متفاوت اجرای برنامه را براساس حاصل یک عبارت یا حالت خاص یک متغیر انتخاب نمایید . دستورات تکرار اجرای برنامه را قادر می سازد تا یک یا چند عبارت را تکرار نماید (یعنی دستورات تکرار حلقه ها را تشکیل می دهند). دستورات پرش به برنامه شما امکان می دهند تا یک روش اجرای غیر خطی داشته باشید . کلیه دستورات کنترلی جاوا را در اینجا بررسی نموده ایم . نکته : اگر C/ C++ را میدانید، دستورات کنترلی جاوا برای شما بسیار آشنا هستند . در حقیقت ، دستورات کنترلی جاوا برای برنامه نویسان C/ C ++ بسیار یکسان است . اما تفاوتهای محدودی وجود دارد بخصوص در دستورات break و.....

دستورات انتخاب در جاوا

جاوا از دو دستور انتخاب پشتیبانی می کنند: if و switch . با این دستورات شما اجرای برنامه را براساس شرایطی که فقط حین اجرای برنامه اتفاق می افتند کنترل می کنید. اگر سابقه برنامه نویسی با C/C++ را ندارید، از قدرت و انعطاف پذیری موجود در این دو دستور متعجب و شگفت زده خواهید شد .

if

دستور if دستور انشعاب شرطی در جاوا است . از این دستور می توان استفاده نمود و اجرای برنامه را طی دو مسیر متفاوت به جریان انداخت . شکل کلی این دستور بصورت زیر است :

```
if( condition )statement 1;
else statement 2;
```

دراینجا هر statement ممکن است یک دستور منفرد یا یک دستور مرکب قرار گرفته در ابروها (یعنی یک بلوک) باشد . condition (شرط) هر عبارتی است که یک مقدار boolean را برمی گرداند . جمله else اختیاری است . if بصورت زیر کار می کند : اگر شرایط محقق باشد ، آنگاه statement 1 اجرا می شود . در غیر اینصورت (statement 2 در صورت وجود) اجرا خواهد شد . تحت هیچ شرایطی هر دو دستور با هم اجرا نخواهند شد . بعنوان مثال ، در نظر بگیرید :

```
int a, b;
//...
```

```
if(a < b )a = 0;
else b = 0;
```

در اینجا اگر a کوچکتر از b باشد ، آنگاه a برابر صفر می شود . در غیر اینصورت b برابر صفر قرار می گیـرد . در هـیچ شـرایطی ایـن دو متغیر در آن واحد برابر صفر نمی شوند . غالب اوقات ، عبارتی که برای کنترل if استفاده میشود شامل عملگرهای رابطه ای است . اما از نظر تکنیکی ضرورتی وجود ندارد . می توان با استفاده از یک متغیرboolean تکی ، ifرا همانطوریکه در بخش زیر مشاهده می کنیـد ، کنتـرل نمود .

```
boolean dataAvailable;
//...
if( dataAvailable)
ProcessData();
else
waitForMoreData();
```

بیاد آورید که فقط یک دستور می تواند مستقیما" بعدازif یاelse ا قرار گیرد .اگر بخواهید دستورات بیشتری داخل نمایید ، نیازی به ایجاد یک بلوک ندارید نظیر این قطعه که در زیر آمده است :

```
int bytesAvailable;
//...
if( bytesAvailable > 0 ){
ProcessData();
bytesAvailable- = n;
} else
waitForMoreData();
```

در اینجا ، هر دو دستور داخل بلوک if اجرا خواهند شد اگر bytes Available بزرگتر از صفر باشـد . برخـی از برنامـه نویسـان راحـت ترند تا هنگام استفاده از if ، از ابروهای باز و بسته استفاده نمایند، حتی زمانیکه فقط یک دستور در هر جمله وجود داشته باشد .این امـر سـبب می شود تا بعدا "بتوان براحتی دستور دیگری را اضافه نمود و نگرانی از فراموش کردن ابروها نخواهید داشـت . در حقیقـت ، فرامـوش کـردن تعریف یک بلوک هنگامی که نیاز است ، یکی از دلایل رایج بروز خطاها می باشد . بعنوان مثال قطعه زیر از یک کد را در نظر بگیرید :

```
int bytesAvailable;
//...
if( bytesAvailable > 0 ){
ProcessData();
bytesAvailable- = n;
```

```
} else
waitForMoreData();
bytesAvailable = n;
```

بنظر خیلی روشن است که دستور bytes Available=n طوری طراحی شده تا داخل جمله else اجرا گردد ، و این بخاطر سطح طراحی آن است . اما حتما" بیاد دارید که فضای خالی برای جاوا اهمیتی ندارد و راهی وجود ندارد که کامپایلر بفهمد چه مقصودی وجود دارد . این کد بدون مشکل کامپایل خواهد شد ، اما هنگام اجرا بطور ناصحیح اجرا خواهد شد . مثال بعدی داخل کدی که مشاهده می کنید تثبیت شده است :

```
int bytesAvailable;
//...
if( bytesAvailable > 0 ){
ProcessData();
bytesAvailable- = n;
} else {
waitForMoreData();
bytesAvailable = n;
}
```

if های تودرتو شده Nested ifs

یک nested if یک دستور if است که هدف if یاelse ا دیگری باشد .if های تودرتو در برنامه نویسی بسیار رایج هستند. هنگامیکه if ها را تودرتو می کنید، مهمترین چیزی که باید بخاطر بسپارید این است که یک دستور elseهمیشه به نزدیکترین دستورif خود که داخل همان بلوک else است و قبلا" با یک else همراه نشده ، مراجعه خواهد نمود . مثالی را مشاهده نمایید :

```
if(i == 10 ){
if(j < 20 )a = b;
if(k > 100 )c = d; // this if is
else a = c; // associated with this else
}
else a = d; // this else refers to if(i == 10)
```

همانگونه که توضیحات نشان می دهند ، elseنهایی با (20چون داخل همان بلوک قرار ندارد (اگر چه نزدیکترین if بدون else است) . بجای آن ، elseنهایی با if(i==10) همراه می شود . else داخلی به (if 100<kارجاع می کند ، زیرا نزدیکترین if در داخل همان بلوک است .

نردبان if-else-if

یک ساختار برنامه نویسی رایج براساس یک ترتیب از if های تودرتو شده یا نردبان if-else-if است . این ساختار بصورت زیر است :

```
if(condition)
statement;
else if(condition)

statement;
else if(condition)
statement;
.
.
.
else
statement;
```

دستورات if از بالا به پایین اجرا می شوند . مادامیکه یکی از شرایط کنترل کننده if صحیح باشد(true) ، دستور همراه با آن if اجرا می شود ، و بقیه نردبان رد خواهد شد . اگر هیچکدام از شرایط صحیح نباشند، آنگاه دستور else نهایی اجرا خواهد شد . else نهایی بعنوان شرط پیش فرض عمل می کند ، یعنی اگر کلیه شرایط دیگر صحیح نباشند ، آنگاه آخرین دستور else انجام خواهد شد . اگر else نهایی وجود نداشته باشد و سایر شرایط ناصحیح باشند ، آنگاه هیچ عملی انجام نخواهد گرفت .

در زیر ، برنامه ای را مشاهده می کنید که از نردبان if-else-if استفاده کرده تا تعیین کند که یک ماه مشخص در کدام فصل واقع شده است .

```
// Demonstrate if-else-if statement.
class IfElse {
public static void main(String args[] ){
int month = 4; // April
String season;

if(month == 12 || month == 1 || month == 2)
season = "Winter";
else if(month == 3 || month == 4 || month == 5)
season = "Spring";
else if(month == 6 || month == 7 || month == 8)
```

```
season = "Summer";
else if(month == 9 || month == 10 || month == 11)
season = "Autumn";
else
season = "Bogus Month";

System.out.println("April is in the"   season   ".");
}
}
```

خروجی این برنامه بقرار زیر می باشد :

April is in the Spring.

ممکن است بخواهید این برنامه را تجربه نمایید . خواهید دید که هیچ فرقی ندارد که چه مقداری به month بدهید ، یک و فقط یک دستور انتساب داخل نردبان اجرا خواهد شد .

switch

دستور switch ، دستور انشعاب چند راهه در جاوا است . این دستور راه ساده ای است برای تغییر مسیر اجرای بخشهای مختلف یک کد براساس مقدار یک عبارت . اینروش یک جایگزین مناسب تر برای مجموعه های بزرگتر از دستورات if-else-if است شکل کلی دستور switchبقرار زیر می باشد :

```
switch(expression){
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
.
.
.
case valueN:
```

```
// statement sequence
break;
default:
// default statement sequence
}
expression
```

می تواند هر نوع ساده ای را برگرداند ، هر یک از مقادیر (values) در دستورات case باید از نوع سازگار با عبارت باشند . هر یک از مقادیر case باید یک مقدار لفظی منحصر بفرد باشد یعنی باید یک ثابت ، نه متغیر ، باشد دو برابر سازی مقادیر case مجاز نیست دستور switchبشرح فوق عمل می کند : مقدار عبارت با هر یک از مقادیر لفظیدر دستورات case مقایسه می شوند. اگر تطابق پیدا شود ، کد سلسله ای تعقیب کنندهآن دستور case اجرا خواهد شد . اگر هیچیک از ثابت ها با مقدار عبارت تطابق نیابند ، آنگاه دستور پیش فرض (default)اجرا خواهد شد ، اما دستور default اختیاری است . اگر هیچیک از case ها تطابق نیابد و default وجود نداشته باشد آنگاه عمل اضافی دیگری انجام نخواهد شد از دستور break داخل دستورswitch استفاده شده تا سلسله یک دستور را پایان دهد .هنگامیکه با یک دستور break مواجه می شویم ، اجرا به خط اول برنامه که بعد از کل دستور switch قرار گرفته ، منشعب خواهد شد . این حالت تاثیر پریدنswitch است . در زیر مثال ساده ای را مشاهده می کنید که از دستورswitch استفاده نموده است :

```
// A simple example of the switch.
class SampleSwitch {
public static void main(String args[] ){
for(int i=0; i<6; i++  )
switch(i ){
case 0:
System.out.println("i is zero.");
break;
case 1:
System.out.println("i is one.");
break;
case 2:
System.out.println("i is two.");
break;
case 3:
System.out.println("i is three.");
break;
default:
System.out.println("i is greater then 3.");
}
}
}
```

خروجی این برنامه بقرار زیر می باشد :

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

همانطوریکه مشاهده می کنید ، داخل حلقه ، دستوراتی که همراه ثابت case بوده و با i مطابقت داشته باشند ، اجرا خواهند شد . سایر دستورات پشت سر گذاشته می شوند . (bypassed) بعد از اینکه i بزرگتر از 3 بشود ، هیچ دستور همراه case مطابقت نداشته ، بنابراین دستور پیش فرض (default) اجرا خواهد شد . دستور break اختیاری است . اگر break را حذف کنید ، اجرای برنامه با caseبعدی ادامه خواهد یافت . گاهی بهتر است چندین case بدون دستورات break در بین آنها داشته باشیم . بعنوان مثال ، برنامه بعدی را در نظر بگیرید :

```
// In a switch/ break statements are optional.
  class MissingBreak {
  public static void main(String args[] ){
  for(int i=0; i<12; i++  )
  switch(i ){
  case 0:
  case 1:
  case 2:
  case 3:
  case 4:
  System.out.println("i is less than 5");
  break;
  case 5:
  case 6:
  case 7:
  case 8:
  case 9:
  System.out.println("i is less than 10");
  break;
  default:
  System.out.println("i is 10 or more");
  }
  }
  }
```

خروجی این برنامه بقرار زیر خواهد بود :

```
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 10
i is less than 10
i is less than 10
i is less than 10
i is less than 10
```

```
i is 10 or more
i is 10 or more
```

همانطوریکه مشاهده می کنید، اجرا طی هرcase ، بمحض رسیدن بـه یـک دسـتور break یـا انتهـای switch متوقـف مـی شـود . در حالیکه مثال قبلی برای توصیف نظر خاصی طراحی شده بود ، اما بهر حال حذف دستور break کاربردهای عملی زیادی در برنامه های واقعی دارد . برای نشان دادن کاربردهای واقعی تر این موضوع ، دوباره نویسی برنامه نمونه مربوط به فصول سال را مشاهده نمایید . این روایت جدیـد همان برنامه قبلی از switch استفاده می کند تا پیاده سازی موثرتری را ارائه دهد .

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

```
season = "Bogus Month";
}
System.out.println("April is in the"   season   ".");

}
}
```

تودرتو کردن دستورات switch

می توانید از یک switch بعنوان بخشی از ترتیب یک دستور switch خارجی تر استفاده نمایید. این حالت را switch تودرتو مینامند. از آنجاییکه دستور switch تعریف کننده بلوک مربوط به خودش می باشد، هیچ تلاقی بین ثابتهایcase درswitch ر داخلی و آنهایی که در switch خارجی قرار گرفته اند ، بوجود نخواهد آمد . بعنوان مثال ، قطعه بعدی کاملا" معتبر است .

```
switch(count ){
case 1:
switch(target ){ // nested switch
case 0:
System.out.println("target is zero");
break;
case 1 :// no conflicts with outer switch
System.out.println("target is one");
break;
}
break;
case 2 ://...
```

در اینجا دستورcase 1: درswitch ر داخلی با دستورcase 1: درswitch ر خارجی تلاقی نخواهد داشت . متغیر count فقط با فهرست case ها در سطح خارجی مقایسه می شود .اگر count برابر 1 باشد، آنگاه target با فهرست case های داخلی مقایسه خواهد شد .

بطور خلاصه ، سه جنبه مهم از دستور switch قابل توجه هستند : ؤswitch باif ا متفاوت است چون switch فقط آزمایش کیفیت انجام می دهد ، در حالیکه if هر نوع عبارت بولی را ارزیابی می کند . یعنی که switch فقط بدنبال یک تطابق بین مقدار عبارت و یکی از ثابت های case خودش می گردد . ؤ دو ثابتcase درswitch ر مشابه نمی توانند مقادیر یکسان داشته باشند . البته ، یک دستور

switchقرار گرفته داخل یک switch خارجی تر می تواند ثابتهایcase مشترک داشته باشد . ؤ یک دستور switch معمولا" بسیار کاراتر از یک مجموعه از if های تودرتو شده است . آخرین نکته بخصوص جالب توجه است زیرا روشنگر نحوه کار کامپایلر جاوا می باشد . کامپایلر جاوا هنگامیکه یک دستور switch را کامپایل می کند ، به هر یک از ثابتهای case سرکشی نموده و یک جدول jump tableمی سازد که برای انتخاب مسیر اجرا براساس مقدار موجود در عبارت استفاده می شود . بنابراین ، اگر باید از میان گروه بزرگی از مقادیر انتخاب نمایید ، یک دستور switch نسبت به یک ترتیب از if-else ها که بطور معادل و منطقی کد بندی شده باشد ، بسیار سریعتر اجرا خواهد شد. کامپایلر قادر است اینکار را انجام دهد چون می داند که ثابتهایcase همه از یک نوع بوده و باید خیلی ساده با عبارت switchبرای کیفیت مقایسه شوند . کامپایلر چنین شناسایی را نسبت به یک فهرست طولانی از عبارات if ندارد .

دستورات تکرار iteration statements

دستورات تکرار در جاوا عبارتند ازfor ، while،وdo-while و . این دستورات آن چه را ما " حلقه " می نامیم ، ایجاد می کنند . احتمالا" می دانید که حلقه یک مجموعه از دستورالعملها را بطور تکراری اجرا می کند . تا اینکه یک شرط پایانی را ملاقات نماید . همانطوریکه بعدا " خواهید دید، جاوا حلقه ای دارد که برای کلیه نیازهای برنامه نویسی مناسب است .

while

حلقه while اساسی ترین دستور حلقه سازی (looping) در جاوا است . این دستور مادامیکه عبارت کنترل کننده ، صحیح (true) باشد، یک دستور یا یک بلوک را تکرار می کند . شکل کلی این دستور بقرار زیر است :

```
while(condition ){
// body of loop
}
```

شرط یا condition ممکن است هر عبارت بولی باشد . مادامیکه عبارت شرطی صحت داشته باشد ، بدنه حلقه اجرا خواهد شد . هنگامیکه شرط صحت نداشته باشد ، کنترل بلافاصله به خط بعدی کدی که بلافاصله پس از حلقه جاری قرار دارد ، منتقل خواهد شد . اگر فقط یک دستور منفرد در حال تکرار باشد ، استفاده از ابروها غیر ضروری است . در اینجا یک حلقه while وجود دارد که تا 10 را محاسبه کرده و دقیقا" ده خط"tick" را چاپ می کند .

```
// Demonstrate the while loop.
class While {
public static void main(String args[] ){
int n = 10;

while(n > 0 ){
```

```
System.out.println("tick"   n);
n--;
}
}
}
```

هنگامیکه این برنامه را اجرا می کنید، ده مرتبه "tick" را انجام خواهد داد :

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

از آنجاییکه حلقه while عبارت شرطی خود را در بالای حلقه ارزیابی میکند، اگر شرط ابتدایی ناصحیح باشد ، بدنه حلقه اجرا نخواهد شد .

بعنوان مثال ، در قطعه زیر ، فراخوانی println() هرگز اجرا نخواهد شد .

```
int a = 10, b = 20;

while(a < b)
System.out.println("This will not be displayed");
```

بدنه) while یا هر حلقه دیگر در جاوا) ممکن است تهی باشد. زیرا دستور تهی) دستوری که فقط شامل ; باشد) از نظر قواعد ترکیبی در جاوا معتبراست . بعنوان مثال ، برنامه زیر را در نظر بگیرید :

```
// The target of a loop can be empty.
class NoBody {
public static void main(String args[] ){
int i, j;

i = 100;
j = 200;
```

```
// find midpoint between i and j
while(  i <-- j); // no body in this loop

System.out.println("Midpoint is"  + i);
}
}
```

کند و خروجی زیر را و را پیدا می j و i بین (midpoint) این برنامه نقطه میانی

: تولید خواهد کرد

Midpoint is 150

در اینجا چگونگی کار حلقه while را می بینید . مقدار i افزایش و مقدار j کاهش می یابد . سپس این دو مقدار با یکدیگر مقایسه می شوند . اگر مقدار جدید i همچنان کمتر از مقدار جدید j باشد ، آنگاه حلقه تکرار خواهد شد . اگر i مساوی با یا بزرگتر از j بشود ، حلقه متوقف خواهد شد . تا هنگام خروج از حلقه ، iمقداری را می گیرد که بین مقادیر اولیه i وj و می باشد . (بدیهی است که این رویه هنگامی کار می کند که i کوچکتر از مقدار اولیه j باشد .) همانطوریکه می بینید ، نیازی به بدنه حلقه نیست ، کلیه عملیات داخل خود عبارت شرطی اتفاق می افتد . در کدهای حرفه ای نوشته شده دیگر جاوا ، وقتی که عبارت کنترل کننده توانایی مدیریت کلیه جزئیات خود را داشته باشد ، حلقه های کوتاه غالبا" بدون بدنه کد بندی می شوند .

do-while

گفتیم اگر عبارت شرطی کنترل کننده یک حلقه while در ابتدا ناصحیح باشد آنگاه بدنه حلقه اصلا" اجرا نمی شود . اما گاهی مایلیم در چنین شرایطی ، بدنه حلقه حداقل یکبار اجرا شود . بعبارت دیگر، در حالات خاصی مایلید تا عبارت پایان دهنده در انتهای حلقه را آزمایش کنید .خوشبختانه ، جاوا حلقه ای را عرضه می کند که دقیقا" همین کار را انجام می دهد . do-while : حلقه do-while همواره حداقل یکبار بدنه خود را اجرا می کند، زیرا عبارت شرطی آن در انتهای حلقه قرار گرفته است . شکل کلی آن بصورت زیر است :

```
do{
// body of loop
} while(condition);
```

هر تکرار از حلقه do-while ابتدا بدنه حلقه را اجرا نموده ، سپس به ارزیابی عبارت شرطی خود می پردازد . اگر این عبارت صحیح (true)باشد ، حلقه اجرا خواهد شد . در غیر اینصورت حلقه پایان می گیرد . نظیر کلیه حلقه های جاوا ، شرط باید یک عبارت بولی باشد .

اینجا یک روایت دیگر از برنامه (tick) وجود دارد که حلقه do-while را نشان می دهد . خروجی این برنامه مشابه برنامه قبلی خواهد بود :

```
// Demonstrate the do-while loop.
class DoWhile {
public static void main(String args[] ){
int n = 10;

do {
System.out.println("tick"   n);
n--;
} while(n > 0);
}
}
```

حلقه موجود در برنامه قبلی ، اگر چه از نظر تکنیکی صحیح است ، اما می توان آن را به شکل کاراتری بصورت زیر دوباره نویسی نمود :

```
do {
System.out.println("tick "   n);
} while--(n > 0);
```

در این مثال ، عبارت (0 <n) عمل کاهش n و آزمایش برای صفر را در یک عبارت گنجانده است . عملکرد آن بقرار بعدی است . ابتدا دستور n اجرا می شود و n را کاهش داده و مقدار جدید را به n برمی گرداند . این مقدار سپس با صفر مقایسه می شود . اگر بزرگتر از صفر باشد ، حلقه ادامه می یابد . در غیر اینصورت حلقه پایان می گیرد. حلقه do-while بویژه هنگام پردازش انتخاب منو بسیار سودمند است ، زیرا معمولا" مایلید تا بدنه یک حلقه منو حداقل یکبار اجرا شود . برنامه بعدی را که یک سیستم Help ساده را برای دستورات تکرار و انتخاب در جاوا پیاده سازی می کند در نظر بگیرید :

```
// Using a do-while to process a menu selection -- a simple help system.
class Menu {
public static void main(String args[])
throws java.io.IOException {
char choice;

do {
System.out.prinln("Help on:");
System.out.prinln(" 1 .if");
System.out.prinln(" 2 .switch");
System.out.prinln(" 3 .while");
```

```
System.out.prinln(" 4 .do-while");
System.out.prinln(" 5 .for\n");
System.out.prinln("Choose one:");
choice =( char )System.in.read();
} while(choice < '1' || choice > '5');

System.out.println("\n");
switch(choice ){
case '1':
System.out.println("The if:\n");
System.out.println("if(condition )statement;");
System.out.println("else statement;");
break;
case '2':

System.out.println("The switch:\n");
System.out.println("switch(expression ){");
System.out.println(" case constant:");
System.out.println(" statement sequence");
System.out.println(" break;");
System.out.println(" //... ");
System.out.println("}");
break;
case '3':
System.out.println("The switch:\n");
System.out.println(while(condition )statement;");
break;
case '4':
System.out.println("The do-while:\n");
System.out.println("do {");
System.out.println(" statement;");
System.out.println("} while( condition);");
break;
case '5':
System.out.println("The for:\n");
System.out.print("for(init; condition; iteration)");
System.out.println(" statement;");
break;
```

```
  }
  }
  }
```

: مشاهده می کنید اکنون یک اجرای نمونه تولید شده توسط این برنامه را

```
Help on:
1 .if
2 .switch
3 .while
4 .do-while
5 .for
Choos one:
4
```

The do-while:

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

در برنامه ، از حلقه do-while برای تصدیق اینکه کاربر یک گزینه معتبر را وارد کرده باشد ، استفاده می شود . در غیر اینصورت ، به کاربر مجددا" اعلان خواهد شد . از آنجاییکه منو باید حداقل یکبار بنمایش درآید ، do-whileحلقه کاملی برای انجام این مقصود است .

چند نکته دیگر درباره این مثال : دقت کنید که کاراکترهااز صفحه کلید بوسیله فراخوانی system.in.read() خوانده می شوند . این یکی از توابع ورودی کنسول در جاوا است .

اگر چه بررسی تفصیلی روشهای I/o جاوا به بحثهای بعدی موکول شده ، اما از system.in.read() در اینجا برای بدست آوردن گزینه کاربر استفاده شده است . این تابع کاراکترها را از ورودی استاندارد می خواند (که بعنوان عدد صحیح برگردان شد ، این دلیلی است که چرا مقدار برگردان از طریق تبدیل (cast) به char تبدیل شده است). بصورت پیش فرض ، ورودی استاندارد، بافر شده خطی است (line buffered)بنابراین قبل از اینکه کاراکترهایی را که تایپ کرده اید به برنامه اتان ارسال کنید ، باید کلید ENTER را فشار دهید . (این حالت مشابه C /C است و احتمالا "از قبل با آن آشنایی دارید . (ورودی کنسول در جاوا کاملا" محدود شده و کار با آن بسیار مشکل است . بعلاوه اکثر برنامه و ریز برنامه های واقعی نوشته شده با جاوا گرافیکی و پنجره ای هستند. از سوی دیگر : چون از

system.in.read()استفاده شده ، برنامه باید جمله throwsjava.io.IoException را کاملا" توصیف نماید . این خط برای مدیریت خطاهای ورودی ضروری است . این بخشی از جنبه های مختلف اداره استثنائ در جاوا است که بعدا" بررسی خواهد شد .

for

خواهید دید که حلقه for یک ساختار قدرتمند و بسیار روان است .شکل کلی دستورfor بصورت زیر است :

```
for(initialization; condition; iteration; ){
// body
}
```

اگر فقط یک دستور باید تکرار شود ، نیازی به ابروها نیست . عملکرد حلقه for بشرح بعدی است . وقتی که حلقه برای اولین بار شروع می شود بخض مقدار دهی اولیه در حلقه اجرا می شود . معمولا" ، این بخش یک عبارت است که مقدار متغیر کنترل حلقه را تعیین می کند ، که بعنوان یک شمارشگر ، کنترل حلقه را انجام خواهد داد . مهم است بدانیم که عبارت مقدار دهی اولیه فقط یکبار اجرا می شود . سپس شرط مورد ارزیابی قرار می گیرد . این شرط باید یک عبارت بولی باشد . این بخش معمولا" مقدار متغیر کنترل حلقه را با مقدار هدف مقایسه می کند .اگر عبارت صحیح (true) باشد، آنگاه بدنه حلقه اجرا خواهد شد . اگر ناصحیح باشد حلقه پایان می گیرد . بعد، بخش تکرار (iteration)حلقه اجرا می شود . این بخش معمولا" عبارتی است که مقدار متغیر کنترل را افزایش یا کاهش می دهد. آنگاه حلقه تکرار خواهد شد ، ابتدا عبارت شرطی را ارزیابی می کند ، سپس بدنه حلقه را اجرا می کند و سرانجام عبارت تکرار را در هر گذر (pass) اجرا میکند. این روال آنقدر دادمه می یابد تا عبارت شرطی ناصحیح (false) گردد . در زیر روایت جدیدی از برنامه "tick" را می بینید که از یک حلقه for استفاده کرده است :

```
// Demonstrate the for loop.
 class ForTick {
 public static void main(String args[] ){
 int n;
 for(n=10; n>0; n--)
 System.out.println("tick" +  n);
 }
 }
```

اعلان متغیرهای کنترل حلقه داخل حلقه for

غالبا" متغیری که یک حلقه for را کنترل می کند ، فقط برای همان حلقه مورد نیاز بوده و کاربری دیگری ندارد . در چنین حالتی ، می توان آن متغیر را داخل بخش مقدار دهی اولیه حلقه for اعلان نمود . بعنوان مثال در اینجا همان برنامه قبلی را مشاهده می کنید که متغیر کنترل حلقه یعنی n بعنوان یک int در داخل حلقه for اعلان شده است .

```
// Declare a loop control variable inside the for.
class ForTick {
public static void main(String args[] ){

// here/ n is declared inside of the for loop
for(int n=10; n>0; n--)
System.out.println("tick"   n);
}
}
```

هنگامیکه یک متغیر را داخل یک حلقه for اعلان می کنید ، یک نکته مهم را باید بیاد داشته باشید : قلمرو آن متغیر هنگامیکه دستور for انجام می شود ، پایان می یابد . (یعنی قلمرو متغیر محدود به حلقه for است .) خارج از حلقه for حیات آن متغیر متوقف می شود . اگر بخواهید از این متغیر کنترل حلقه در جای دیگری از برنامه اتان استفاده کنید ، نباید آن متغیر را داخل حلقه for اعلان نمایید . درشرایطی که متغیر کنترل حلقه جای دیگری موردنیاز نباشد، اکثر برنامه نویسان جاوا آن متغیر را داخل for اعلان می کنند . بعنوان مثال ، در اینجا یک برنامه ساده را مشاهده می کنید که بدنبال اعداد اول می گردد. دقت کنید که متغیر کنترل حلقه ، چون جای دیگری مورد نیاز نیست ، داخل forاعلان شده است .

```
// Test for primes.
 class FindPrime {
 public static void main(String args[] ){
 int num;
 boolean isPrime = true;

 num = 14;
 for(int i=2; i < num/2; i  ++ ){
 if((num % i )== 0 ){
 isPrime = false;
 break;
 }
```

```
}
if(isPrime )System.out.println("Prime");
else System.out.println("Not Prime");
}
}
```

استفاده از کاما Comma

شرایطی پیش می آید که مایلید بیش از یک دستور در بخش مقدار دهی اولیه (initalization) و تکرار (iteration) بگنجانید . بعنوان مثال ، حلقه موجود در برنامه بعدی را در نظر بگیرید :

```
Class Sample {
public static void main(String args[] ){
int a, b;
 b = 4;
for(a=1; a+ System.out.println("a = " + a);
System.out.println("b = " + b);
b--;
}
}
}
```

همانطوریکه می بینید ، حلقه توسط ارتباط متقابل دو متغیر کنترل می شود . از آنجاییکه حلقه توسط دو متغیر اداره می شود ، بجای اینکه b را بصورت دستی اداره کنیم ، بهتر است تا هر دو را در دستور for بگنجانیم . خوشبختانه جاوا راهی برای اینکار دارد . برای اینکه دو یا چند متغیر بتوانند یک حلقه for را کنترل کنند ، جاوا به شما امکان می دهد تا چندین دستور را در بخشهای مقدار دهی اولیه و تکرار حلقه for قرار دهید . هر دستور را بوسیله یک کاما از دستور بعدی جدا می کنیم . حلقه for قبلی را با استفاده از کاما ، خیلی کاراتر از قبل می توان بصورت زیر کد بندی نمود :

```
// Using the comma.
class Comma {
public static void main(String args[] ){
int a, b;

for(a=1; b=4; a++)
System.out.println("a = "  a);
System.out.println("b = "  b);
```

```
}
}
}
```

در این مثال ، بخش مقدار دهی اولیه ، مقادیر a و b را تعیین می کند . هربار که حلقه تکرار می شود ، دو دستور جدا شده توسط کاما در بخش تکرار (iteration) اجرا خواهند شد . خروجی این برنامه بقرار زیر می باشد :

```
a=1
b=4
a=2
b=3
```

نکته : اگر با C/ C++ آشنایی دارید ، حتما" می دانید که در این زبانها ، علامت کاما یک عملگر است که در هر عبارت معتبری قابل استفاده است . اما در جاوا اینطور نیست . در جاوا ، علامت کاما یک جدا کننده است که فقط در حلقه for قابل اعمال می باشد .

برخی گوناگونیهای حلقه for

حلقه for از تعدادی گوناگونیها پشتیبانی می کند که قدرت و کاربری آن را افزایش می دهند . دلیل انعطاف پذیری آن است که لزومی ندارد که سه بخش مقداردهی اولیه ، آزمون شرط و تکرار ، فقط برای همان اهداف مورد استفاده قرار گیرند . در حقیقت ، سه بخش حلقه forبرای هر هدف مورد نظر شما قابل استفاده هستند . به چند مثال توجه فرمائید . یکی از رایجترین گوناگونیها مربوط به عبارت شرطی است . بطور مشخص ، لزومی ندارد این عبارت ، متغیر کنترل حلقه را با برخی مقادیر هدف آزمایش نماید . در حقیقت ، شرط کنترل کننده حلقه for ممکن است هر نوع عبارت بولی باشد . بعنوان مثال ، قطعه زیر را در نظر بگیرید :

```
boolean done = false;

for(int i=1; !done; i   ){
//...
if(intettupted )()done = true;
}
```

در این مثال ، حلقه for تا زمانیکه متغیر بولی done معادل true بشود ، اجرا را ادامه خواهد داد . این مثال مقدار i را بررسی نمی کند .

اکنون یکی دیگر از گوناگونیهای جالب حلقه for را مشاهده می کنید. ممکن است یکی یا هر دو عبارت مقدار دهی اولیه و تکرار غایت باشند ، نظیر برنامه بعدی :

```
// Parts of the for loop can be empty.
```

```
class ForVar {
public static void main(String args[] ){
int i;
boolean done = false;

i = 0;
for (; !done; ) {
System.out.println("i is"   i);
if(i == 10 )done = true;
i  ;
}
}
}
```

در اینجا عبارتهای مقدار دهی اولیه و تکرار به خارج از forانتقال یافته اند .برخی از بخشهای حلقه for تهی هستند . اگر چه در این مثال ساده چنین حالتی هیچ ارزشی ندارد ، اما در حقیقت شرایطی وجود دارد که این روش بسیار کارا و سودمند

خواهد بود. بعنوان مثال ، اگر شرط اولیه بصورت یک عبارت پیچیده و در جای دیگری از برنامه قرار گرفته باشد و یا تغییرات متغیر کنترل حلقه بصورت غیر ترتیبی و توسط اعمال اتفاق افتاده در داخل بدنه حلقه تعیین شود ، پس بهتر است که این بخشها را در حلقه for تهی بگذاریم . اکنون یکی دیگر از گوناگونیهای حلقه for را مشاهده می کنید. اگر هر سه بخش حلقه for را تهی بگذارید ، آنگاه بعمد یک حلقه نامحدود (حلقه ای که هرگز پایان نمی گیرد) ایجاد کرده اید . بعنوان مثال :

```
for (; ; ) {
//...
}
```

این حلقه تا ابد ادامه خواهد یافت ، زیرا هیچ شرطی برای پایان گرفتن آن تعبیه نشده است . اگر چه برخی برنامه ها نظیر پردازشهای فرمان سیستم عامل مستلزم یک حلقه نامحدود هستند ، اما اکثر حلقه های نامحدود در واقع حلقه هایی هستند که ملزومات پایان گیری ویژه ای دارند . بزودی خواهید دید ، راهی برای پایان دادن به یک حلقه حتی یک حلقه نامحدود نظیر مثال قبلی وجود دارد که از عبارت شرطی معمولی حلقه استفاده نمی کند .

حلقه های تودرتو

نظیر کلیه زبانهای برنامه نویسی ، جاوا نیز امکان تودرتو کردن حلقه ها را دارد . یعنی یک حلقه داخل حلقه دیگری قرار خواهد گرفت . بعنوان مثال ، در برنامه بعدی حلقه های for تودرتو نشده اند :

```
// Loops may be nested.
class Nested {
public static void main(String args[] ){
int i/ j;

for(i=0; i<10; i++   ){
for(j=i; j<10; j++  )
System.out.print(".");
System.out.println();
}
}
}
```

خروجی تولید شده توسط این برنامه بقرار زیر می باشد ..... :

.....

دو دستور کنترلی

این دو دستور یک جنبه بسیار مهم از جاوا یعنی بلوک های کد (block of code) را تشریح می کنند .

دستور if(if statement()

دستور if در جاوا نظیر دستور If درهر یک از زبانهای برنامه نویسی کار میکند .بعلاوه این دستور از نظر قواعد صرف و نحو با دستورif در Cرو C++ و شباهت دارد . ساده ترین شکل آن را در زیر مشاهده می کنید :

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

در اینجا شرط (condition) یک عبارت بولی (Boolean) است . اگر شرایط درست باشد ، آنگاه دستور اجرا خواهدشد . اگر شرایط صحیح نباشد ، آنگاه دستور پشت سر گذاشته می شود .(bypassed) بعنوان مثال در نظر بگیرید :

```
if(num < 100 )println("num is less then 100");
```

در این حالت ، اگر متغیر num شامل مقداری کوچکتر از 100 باشد ، عبارت شرطی درست بوده و println() اجرا خواهد شد . اگر numشامل مقداری بزرگتر یا مساوی 100باشد ، آنگاه روش println() پشت سر گذاشته می شود . بعدا" خواهید دید که جاوا یک ضمیمه کامل از عملگرهای رابطه ای (Relational) تعریف می کند که قابل استفاده در عبارات شرطی هستند . چند نمونه از آنها بشرح زیر است :

| عملگر | مفهوم آن |

| < | کوچکتر از |

| > | بزرگتر از |

| = = | مساوی با |

دقت داشته باشید که آزمایش تساوی با علامت تساوی انجام می گیرد . در زیر برنامه ای مشاهده می کنید که یک دستور if را توصیف کرده است :

```
/*
Demonstrate the if.
Call this file "IfSample.java".
*/
class IfSample {
public static void main( String args [] ){
int x/ y;

x = 10;
y = 20;
if(x < y )System.out.println("x is less than y");

x = x * 2;
if(x == y )System.out.println("x now equal to y");

x = x * 2;
if(x > y )System.out.println("x now greater than y");
```

```
// this won't display anything
if(x == y )System.out.println("you won't see this");
}
}
```

خروجی تولید شده توسط این برنامه بشرح زیر خواهد بود :

```
x is less than y
x now equal to y
x now greater than y
```

به یک نکته دیگر در این برنامه دقت نمایید . خط

```
int x, y;
```

دو متغیرx وy و را با استفاده از فهرست جدا شده با کاما اعلان می کند .

حلقه for

شاید از تجربیات قبلی در برنامه نویسی تا بحال فهمیده باشید که دستورات حلقه (loop statements) یک بخش بسیار مهم در کلیه زبانهای برنامه نویسی هستند .جاوا نیز از این قاعده مستثنی نیست . در حقیقت همانگونه که در فصل پنجم خواهید دید ، جاوا یک دسته بندی پر قدرت از ساختارهای حلقه ای عرضه می کند . شاید از همه این ساختارها سلیس تر حلقه for باشد . اگر باC و C++ و آشنایی داشته باشید خوشحال خواهید شد که بدانید نحوه کار حلقه های for در جاوا با این زبانها مشابه است . اگر باC و++ C و آشنایی ندارید ، باز هم فراگیری استفاده از حلقه for بسیار ساده خواهد بود . ساده ترین شکل این حلقه بقرار زیر می باشد :

```
for( initialization; condition )statement;
```

دستور اجرای مکرر شرایط مقداردهی اولیه بخش مقداردهی اولیه در یک حلقه در معمول ترین شکل خود یک مقدار اولیه را در یک متغیر کنترل اگر ماحصل این آزمایش صحیح باشد ، حلقه for به تکرار خود ادامه می دهد . اگر حاصل ناصحیح باشد ، حلقه متوقف خواهد شد . عبارت اجرای مکرر تعیین کننده این است که متغیر کنترل حلقه پس از هر بار تکرار حلقه چگونه تغییر خواهد کرد . برنامه کوتاه زیر توصیف کننده یک حلقه for می باشد :

```
/*
Demonstrate the for loop.
Call this file "ForTest.java".
*/
```

```
class ForTest {
public static void main(String args [] ){
int x;

for(x = 0; x<10; x = x 1)
System.out.println("This is x :"   x);
}
}
```

این برنامه ، خروجی زیر را تولید خواهد نمود :

```
this is x:0
this is x:1
this is x:2
this is x:3
this is x:4
this is x:5
this is x:6
this is x:7
this is x:8
this is x:9
```

در این مثال ، xمتغیر کنترل حلقه است . در بخش مقداردهی اولیه حلقه for به این متغیر مقدار صفر داده می شود. در شروع هر تکرار (شامل مرحله اول) آزمایش شرط 10اجرا خواهد شد ، و سپس بخش اجرای مکرر حلقه اجرا خواهد شد . این روال مادامیکه آزمایش شرایط صحیح باشد ، ادامه می یابد. بعنوان یک نکته قابل تامل ،در برنامه های حرفه ای نوشته شده توسط جاوا بندرت بخش اجرای مکرر حلقه بصورت برنامه قبلی نوشته می شود . یعنی شما بندرت دستوری مثل عبارت زیر خواهید دید :

```
x = x   1;
```

دلیل این است که جاوا دربرگیرنده یک عملگر افزایشی ویژه است که همین عملیات را بطور موثرتری انجام می دهد . این عملگر افزایشی است (دو علامت جمع در کنار هم) . عملگر افزایشی ، عملوند خود را یکی یکی افزایش خواهد داد . با استفاده از عملگر افزایشی دستور قبلی را می توان بصورت زیر نوشت :

```
x ;
```

بدین ترتیب for در برنامه قبلی را معمولا "بصورت زیر می نویسند :

```
for(x = 0; x<10; x++  )
```

ممکن است بخواهید این مورد را آزمایش کنید . همانطوریکه خواهید دید ، حلقه درست مثل قبل و بهمان ترتیب اجرا خواهد شد . جاوا همچنین یک عملگر کاهشی (decrement) فراهم نموده که باعلامت (--) مشخص یکی یکی کاهش خواهد داد.

آرایه ها

یک آرایه گروهی از متغیرهای یک نوع است که با یک نام مشترک به آنها ارجاع می شود . می توان آرایه ها را برای هر یک از انواع ایجاد نمود و ممکن است این آرایه ها دارای یک یا چندین بعد باشند . برای دسترسی به یک عضو آرایه از نمایه (index) آن آرایه استفاده می شود . آرایه ها یک وسیله مناسب برای گروه بندی اطلاعات مرتبط با هم هستند . نکته : اگر با C و C++ و آشنایی دارید ، آگاه باشید . آرایه ها در جاوا بطور متفاوتی نسبت به زبانهای دیگر کار می کنند .

آرایه های یک بعدی

آرایه یک بعدی بطور ضروری فهرستی از متغیرهای یکنوع است . برای ایجاد یک آرایه ، باید یک متغیر آرایه از نوع مورد نظرتان ایجاد کنید . فرم عمومی اعلان یک آرایه یک بعدی بقرار زیر است :

```
type var-name [];
```

نام متغیر نوع

در اینجا type اعلان کننده نوع اصلی آرایه است . نوع اصلی تعیین کننده نوع داده برای هر یک از اعضائ داخل در آرایه است . بنابراین ، نوع اصلی آرایه تعیین می کند که آرایه چه نوعی از داده را نگهداری می کند . بعنوان مثال ، در زیر یک آرایه با نام month-days با نوع آرایه ای از عدد صحیح اعلان شده است .

```
int month_days[];
```

اگر چه این اعلان تثبیت می کند که month-days یک متغیر آرایه است ، اما بطور واقعی آرایه ای وجود ندارد . در حقیقت ، مقدار month-days برابر تهی (null) می باشد که یک آرایه بدون مقدار را معرفی می کند . برای پیوند دادن month-days با یک آرایه واقعی و فیزیکی از اعداد صحیح ، باید از یک عملگر new استفاده نموده و به month-days منتسب کنید . new یک عملگراست که حافظه را اختصاص میدهد .بعدا " new را با دقت بیشتری بررسی می کنیم ، اما لازم است که هم اکنون از آن استفاده نموده و حافظه را برای آرایه ها تخصیص دهید . فرم عمومی new آنگونه که برای آرایه های یک بعدی بکار می رود بقرار زیر ظاهر خواهد شد :

```
array-var=new type [size];
```

اندازه نوع متغیر آرایه

در اینجا type مشخص کننده نوع داده ای است که تخصیص داده می شود، sizeمشخص کننده تعداد اعضائ آرایه است و array-varمتغیر آرایه است که به آرایه پیوند می یابد . یعنی برای استفاده از new در تخصیص یک آرایه ، باید نوع و تعداد اعضایی که تخصیص می یابند را مشخص نمایید . اعضائ آرایه که توسط new تخصیص می یابند بطور خودکار با مقدار صفر مقدار دهی اولیه می شوند . این مثال یک آرایه 12 عضوی از اعداد صحیح را تخصیص داده و آنها را به month-days پیوند می دهد .

```
month_days = new int[12];
```

بعد از اجرای این دستور ، month-daysبه یک آرایه 12 تایی از اعداد صحیح ارجاع خواهد نمود . بعلاوه کلیه اجزائ در آرایه با عدد صفر مقدار دهی اولیه خواهند شد . اجازه دهید مرور کنیم : بدست آوردن یک آرایه مستلزم پردازش دو مرحله ای است . اول باید یک متغیر با نوع آرایه مورد نظرتان اعلان کنید . دوم باید حافظه ای که آرایه را نگهداری می کند ، با استفاده از new تخصیص دهید و آن را به متغیر آرایه نسبت دهید . بنابراین در جاوا کلیه آرایه ها بطور پویا تخصیص می یابند . اگر مفهوم تخصیص پویا برای شما ناآشناست نگران نباشید . این مفهوم را بعدا "تشریح خواهیم کرد . هر بار که یک آرایه را تخصیص می دهید ، می توانید بوسیله مشخص نمودن نمایه آن داخل کروشه [] به یک عضو مشخص در آرایه دسترسی پیدا کنید . کلیه نمایه های آرایه ها با عدد صفر شروع می شوند. بعنوان مثال این دستور مقدار 28 را به دومین عضو month-days نسبت می دهد .

```
month_days[1] = 28;
```

خط بعدی مقدار ذخیره شده در نمایه 3 را نمایش می دهد .

```
System.out.println(month_days[3]);
```

با کنار هم قرار دادن کلیه قطعات ، در اینجا برنامه ای خواهیم داشت که یک

آرایه برای تعداد روزهای هر ماه ایجاد می کند .

```
// Demonstrate a one-dimensional array.
 class Array {
 public static void main(String args[] ){
 int month_days[];
 month_days = new int[12];
```

```
month_days [0] = 31;
month_days [1] = 28;
month_days [2] = 31;
month_days [3] = 30;
month_days [4] = 31;
month_days [5] = 30;
month_days [6] = 31;
month_days [7] = 31;
month_days [8] = 30;
month_days [9] = 31;
month_days [10] = 30;
month_days [11] = 31;
System.out.println("April has " + month_days[3] + " days .");
}
}
```

وقتی این برنامه را اجرا میکنید ، برنامه ، تعداد روزهای ماه آوریل را چاپ میکند. همانطوریکه ذکر شد، نمایه های آرایه جاوا با صفر شروع می شوند، بنابراین تعداد روزهای ماه آوریل در month-days[3] برابر 30 می باشد . این امکان وجود دارد که اعلان متغیر آرایه را با تخصیص خود آرایه بصورت زیر ترکیب نمود

```
int month_days[] = new int[12];
```

این همان روشی است که معمولا" در برنامه های حرفه ای نوشته شده با جاوا مشاهده می کنید . می توان آرایه ها را زمان اعلانشان ، مقدار دهی اولیه نمود . پردازش آن بسیار مشابه پردازشی است که برای مقدار دهی اولیه انواع ساده استفاده می شود . یک مقدار ده اولیه آرایه فهرستی از عبارات جدا شده بوسیله کاما و محصور شده بین ابروهای باز و بسته می باشد . کاماها مقادیر اجزائ آرایه را از یکدیگر جدا می کنند . آرایه بطور خودکار آنقدر بزرگ ایجاد می شود تا بتواند ارقام اجزایی را که در مقدار ده اولیه آرایه مشخص کرده اید ، دربرگیرد . نیازی به استفاده ازnew وجود ندارد . بعنوان مثال ، برای ذخیره نمودن تعداد روزهای هر ماه ، کد بعدی یک آرایه مقدار دهی اولیه شده از اعداد صحیح را بوجود می آورد :

```
// An improved version of the previous program.
class AutoArray {
public static void main(String args[] ){
int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,31 };
System.out.println("April has " + month_days[3] + " days .");
```

```
}
}
```

وقتی این برنامه را اجرا کنید ، همان خروجی برنامه قبلی را خواهید دید . جاوا بشدت کنترل می کند تا مطمئن شود که بطور تصادفی تلاشی برای ذخیره نمودن یا ارجاع مقادیری خارج از دامنه آرایه انجام ندهید . سیستم حین اجرای جاوا کنترل می کند که کلیه نمایه های آرایه ها در دامنه صحیح قرار داشته باشند . (از این نظر جاوا کاملا"باC او++C و متفاوت است که هیچ کنترل محدوده ای در حین اجرا انجام نمی دهند .) بعنوان مثال ، سیستم حین اجرا ، مقدار هر یک از نمایه ها بهmonth-days را کنترل می کند تا مطمئن شود که بین ارقام 0 و 11 داخل قرار داشته باشند . اگر تلاش کنید تا به اجزائ خارج از دامنه آرایه) اعداد منفی یا اعدادی بزرگتر از طول آرایه) دسترسی یابید، یک خطای حین اجرا (run-time error) تولید خواهد شد . در زیر یک مثال پیچیده تر مشاهده می کنید که از یک آرایه یک بعدی استفاده می کند . این برنامه میانگین یک مجموعه از ارقام را بدست می آورد .

```
// Average an array of values.
class Average {
public static void main(String args[] ){
double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};
double result = 0;
int i;
for(i=0; i<5; i++)
result = result + nums[i];
System.out.println("Average is " + result / 5);
}
}
```

آرایه های چند بعدی

در جاوا آرایه های چند بعدی در واقع آرایه ای از آرایه ها هستند . این قضیه همانطوریکه انتظار دارید ظاهر و عملکردی مشابه آرایه های چندبعدی منظم (regular) دارد . اما خواهید دید که تاوتهای ظریفی هم وجود دارند . برای اعلان یک متغیر آرایه چند بعدی ، با استفاده از مجموعه دیگری از کروشه ها هر یک از نمایه های اضافی را مشخص می کنید. بعنوان مثال ، عبارت زیرر یک متغیر آرایه دو بعدی بنامtwoD را اعلان می کند .

```
int twoD[][] = new int[4][5];
```

این عبارت یک آرایه 4در 5ر را تخصیص داده و آن را به twoD نسبت می دهد . از نظر داخلی این ماتریس بعنوان یک آرایه از آرایه نوع int پیاده سازی خواهد شد . بطور فرضی ، این آرایه را می توان بصورت شکل زیر نمایش داد .

Right index determines column.

|| || || || ||
\/ \/ \/ \/ \/

[0][4] | [0][3] | [0][2] | [0][1] | [0][0] >

[1][4] | [1][3] | [1][2] | [1][1] | [1][0] >
Left index
determines
[2][4] | [2][3] | [2][2] | [2][1] | [2][0] .> row

[3][4] | [3][3] | [3][2] | [3][1] | [3][0] >
Given :

int twoD[][] = new int [4][5];

برنامه بعدی هر عضو آرایه را از چپ به راست ، و از بالا به پایین شماره داده

و سپس مقادیر آنها را نمایش می دهد :

```
// Demonstrate a two-dimensional array.
 class TwoDArray {
 public static void main(String args[] ){
 int twoD[][] = new int[4][5];
 int i, j,k = 0;
 for(i=0; i<4; i++)
 for(j=0; j<5; j++ ){
 twoD[i][j] = k;
 k++;
 }

 for(i=0; i<4; i++ ){
 for(j=0; j<5; j++)
 System.out.print(twoD[i][j] + " ");
```

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

خروجی این برنامه بقرار زیر خواهد بود : 4 3 2 1 0

```
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

هنگام تخصیص حافظه به یک آرایه چند بعدی ، کافی است فقط حافظه برای اولین

بعد را مشخص نمایید . می توانید ابعاد دیگر را جداگانه تخصیص دهید . بعنوان

مثال ، کد زیر حافظه اولین بعد twoD را هنگام اعلان آن تخصیص می دهد . این کد

حافظه دومین بعد را بصورت دستی اختصاص می دهد .

```
int twoD[][] = new int[4][];
twoD[0] = new int[5];
twoD[1] = new int[5];
twoD[2] = new int[5];
twoD[3] = new int[5];
```

اگرچه در این حالت اختصاص انفرادی حافظه به دومین بعد هیچ مزیتی ندارد، اما احتمال چنین مزیتهایی وجود دارد . بعنوان مثال ، هنگامیکه ابعاد را بصورت دستی اختصاص می دهید ، نیازی نیست که همان ارقام برای اجزائ هر بعد را تخصیص دهید . همانطوریکه قبلا" گفتیم ، از آنجاییکه آرایه های چند بعدی واقعا" آرایه ای از آرایه ها هستند ، طول هر یک از آرایه ها تحت کنترل شما قرار می گیرند . بعنوان مثال ، برنامه بعدی یک آرایه دو بعدی ایجاد می کند که در آن اندازه های دومین بعد نامساوی هستند .

```
// Manually allocate differing size second dimension.
class TwoDAgain {
public static void main(String args[] ){


int twoD[][] = new int[4][];
twoD[0] = new int[1];
twoD[1] = new int[2];
twoD[2] = new int[3];
```

```
twoD[3] = new int[4];

int i, j, k = 0;

for(i=0; i<4; i++)
for(j=0; j+ towD[i][j] = k; k++)
}

for(i=0; i<4; i++ ){
for(j=0; j+ System.out.print(twoD[i][j] + " ");
System.out.println();
}
}
}
```

خروجی این برنامه بقرار زیر می باشد : 0

1 2
3 4 5
6 7 8 9

آرایه ای که توسط این برنامه ایجاد می شود ، بصورت زیر خواهد بود :

| [0][0] |

| [1][0] | [1][1] |

| [2][0] | [2][1] | [2][2] |

| [3][0] | [3][1] | [3][2] | [3][3] |

از آرایه های چند بعدی ناجور (یا نامنظم) در اکثر برنامه ها استفاده نمیشود زیرا برخلاف آنچه مردم هنگام مواجه شدن با یک آرایه چند بعدی انتظار دارند رفتار می کنند . اما این آرایه ها در برخی شرایط بسیار کارا هستند . بعنوان مثال ، اگر نیاز به یک آرایه دو بعدی خیلی بزرگ دارید که دارای تجمع پراکنده باشد (یعنی که یکی و نه همه اجزائ آن مورد استفاده قرار می گیرند) ، آنگاه آرایه بی قاعده احتمالا" یک راه حل کامل خواهد بود . این امکان وجود دارد که آرایه های چند بعدی را مقدار دهی اولیه نمود . برای

اینکار ، فقط کافی است هر یک از مقدار ده اولیه ابعاد را داخل مجموعه ابروهای ختص خودش قرار دهید . برنامه بعدی یک ماتریس ایجاد می کند که هر یک از اجزائ آن شامل حاصلضرب نمایه های سطرها و ستونها هستند. همچنین دقت نمایید که می توان از عبارات همچون مقادیر لفظی داخل مقدار ده اولیه آرایه استفاده نمود .

```
// Initialize a two-dimensional array.
class Matrix {
public static void main(String args[] ){
double m[][] = {
{ 0.0, 1.0, 2.0, 3.0 };
{ 0.1, 1.1, 2.1, 3.1 };
{ 0.2, 1.2, 2.2, 3.2 };
{ 0.3, 1.3, 2.3, 3.3 }};
int i, j;

for(i=0; i<4; i++ ){
for(j=0 j<4; j++)
System.out.print(m[i][j] + " ");
System.out.println)(;
}
}
}
```

پس از اجرای این برنامه ، خروجی آن بقرار زیر خواهد بود : 0 0 0 0

0 1 2 3
0 2 4 6
0 3 6 9

همانطوریکه مشاهده می کنید، هر سطر در آرایه همانگونه که در فهرستهای مقدار دهی اولیه مشخص شده ، مقدار دهی اولیه شده است . مثالهای بیشتری درباره استفاده از آرایه چند بعدی بررسی می کنیم . برنامه بعدی یک آرایه سه بعدی 3 x4x5ایجاد می کند . سپس حاصل نمایه های مربوطه را برای هر عضو بارگذاری می کند . در نهایت این حاصل ها را نمایش خواهد داد :

```
// Demonstrate a three-dimensional array.
class threeDDatrix {
```

```
public static void main(String args[] ){
int threeD[][][] = new int[3][4][5];
int i, j,k;
for(i=0; i<3; i++)
for(j=0; j<4; j++)
for(k=0; k<5; k++)
threeD[i][j][k] = i * j * k;

for(i=0; i<3; i++ ){
for(j=0; j<4; j++ ){
for(k=0; k<5; k++)
System.out.print(threeD[i][j][k] + " ");
System.out.println)(;
}
System.out.println)(;
}
}
}
```

خروجی این برنامه بقرار 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
```

دستور زبان جایگزین اعلان آرایه

یک شکل دوم برای اعلان یک آرایه بصورت زیر وجود دارد :

```
type [] var-name;
```

نام متغیر نوع

در اینجا کروشه ها بعداز مشخص کننده نوع می آیند نه بعداز نام متغیر آرایه . بعنوان مثال دو شکل اعلان زیر یکسان عمل می کنند :

```
int a1[] = new int[3];
int[] a2 = new int[3];
```

دو شکل اعلان زیر هم یکسان عمل می کنند :

```
char twod1[][] = n
```

آرایه های دوباره ملاقات شده Arrays Revisited

آرایه ها بعنوان اشیائ پیاده سازی می شوند . بهمین دلیل ، یک خصلت ویژه وجود دارد که می توانید از مزیت آن استفاده نمایید . بطور اخص ، اندازه یک آرایه یعنی تعداد اعضایی که یک آرایه میتواند نگهداری نماید را می توان در متغیر نمونه length پیدا نمود . کلیه آرایه ها این متغیر را دارند، و این متغیر همیشه اندازه آرایه را نگهداری می کند. در اینجا برنامه ای وجود دارد که این خاصیت را نشان می دهد :

```
// This program demonstrates the length array member.
 class Length {
 public static void main(String args[] ){
 int a1[] = new int [10];
 int a2[] = {3, 5, 7, 1, 8, 99, 44,- 10};
 int a3[] = {4, 3, 2, 1};
 System.out.println("length of a1 is " + a1.length);
 System.out.println("length of a2 is " + a2.length);
 System.out.println("length of a3 is " + a3.length);
```

خروجی این برنامه بقرار زیر می باشد :

length of a1 is 10
length of a2 is 8
length of a3 is 4

همانطوریکه می بینید ، اندازه هر یک از آرایه ها بنمایش درآمده است . بیاد بسپارید که مقدار length کاری با تعداد اعضایی که واقعا" مورد استفاده قرار گرفته اند، نخواهد داشت . این مقدار فقط منعکس کننده تعداد اعضایی است که آرایه برای نگهداری آن طراحی شده است .

در بسیاری از شرایط می توانید عضو length را در معرض کاربردهای مناسب قرار دهید . بعنوان مثال ، یک روایت توسعه یافته از کلاس stack را در اینجا مشاهده می کنید. احتمالا" بیاد دارید که روایتهای اولیه این کلاس همیشه یک پشته ده عضوی ایجاد میکرد . روایت بعدی همین برنامه ، به شما امکان ایجاد پشته هایی به هر اندازه دلخواه را می دهد . از مقدار stck.length برای ممانعت از وقوع سرریزی پشته استفاده شده است .

```
// Improved Stack class that uses the length array member.
class Stack {
private int stck[];
private int tos;

// allocate and initialize stack
Stack(int size ){
stck = new int[size];
tos =- 1;
}

// Push an item onto the stack
void push(int item ){
if(tos==stck.length-1 )// use length member
System.out.println("Stack is full.");
else
stck[++tos] = item;
}
```

```
// Pop an item from the stack
int pop )({
if(tos < 0 ){
System.out.println("Stack underflow.");
return 0;
}
else
return stck[tos--];
}
}

class TestStack2 {
public static void main(String args[] ){
Stack mystack1 = new Stack(5);
Stack mystack2 = new Stack(8);

// push some numbers onto the stack
for(int i=0; i<5; i++ )mystack1.push(i);
for(int i=0; i<5; i++ )mystack2.push(i);

// pop those numbers off the stack
System.out.println("Stack in mystack1:");
for(int i=0; i<5; i++)
System.out.println(mystack1.pop))(;

System.out.println("Stack in mystack2:");
for(int i=0; i<8; i++)
System.out.println(mystack2.pop))(;
}
```

}

دقت کنید که برنامه فوق دو پشته ایجاد می کند : یکی با عمق پنج عضو ، و دیگری با عمق هشت عضو . همانطوریکه می بینید ، این حقیقت که آرایه ها اطلاعات length خودشان را نگهداری می کنند .

منابع :

http://www.irandevelopers.com/
http://docs.sun.com

نویسنده :
محمد باقر معموری mamouri@ganjafzar.com
ویراستار و نویسنده قسمت های تکمیلی :
احسان شاه بختی zehs_sha@yahoo.com

کتاب :

اتشارات نص در 21 روز Java
برنامه نویسی شی گرا اتشارات نص