www.bindownload.com

انواع داده ها ، متغیرها و عملگرها در جاوا

از این قسمت به بعد سه عنصر اساسی جاوا را مورد بررسی قرار خواهیم داد : انواع داده ها ، متغیرها و آرایه ها. نظیر کلیه زبانهای جدید برنامه نویسی ، جاوا از چندین نوع داده پشتیبانی می کند . با استفاده از انواع داده ، می توانید متغیرها را اعلان نموده و آرایه ها را ایجاد کنید. خواهید دید که شیوه جاوا برای ین مسئله ، کاملا" روشن ، کارا و منسجم است .

متغیرها در جاوا

در یک برنامه جاوا ، متغیر ، اساسی ترین واحد ذخیره سازی است . یک متغیر به وسیله ترکیبی از یک شناسه ، یک نوع و یک مقدار ده اولیه اختیاری تعریف خواهد شد . علاوه بر این ، کلیه متغیرها دارای یک قلمرو هستند که رویت پذیری آنها را تعریف می کند و یک زمان حیات نیز دارند. متعاقبا" این اجزائ را مورد بررسی قرار می دهیم .

اعلان یک متغیر Declaring a variable

در جاوا کلیه متغیرها قبل از استفاده باید اعلان شوند . شکل اصلی اعلان متغیر بقرار زیر می باشد : type identifier [=value] [,identifier[=value]...];

مقدار شناسه مقدار شناسه نوع

نوع (type) یکی از انواع اتمی جاوا یا نام یک کلاس یا رابط است . (انواع کلاس و رابط بعدا" بررسی خواهد شد .) شناسه نام متغیر است . می توانید با گذاشتن یک علامت تساوی و یک مقدار ، متغیر را مقدار دهی اولیه نمایید . در ذهن بسپارید که عبارت مقدار دهی اولیه باید منتج به یک مقدار از همان نوعی(یا سازگار با آن نوع) که برای متغیر مشخص شده ، گردد . برای اعلان بیش از یک نوع مشخص شده ، از فهرست کاماهای (') جدا کننده استفاده نمایید . در زیر مثالهایی از اعلان متغیر از انواع گوناگون را مشاهده می کنید . دقت کنید که برخی از آنها شامل یک مقدار دهی اولیه هستند .

نکته

برای نوشتن توضیحات در جاوا از // یا /* */ استفاده می کنیم .

```
1. int a, b, c; // declares three int a, b, and c.
2. int d = 3, e, f = 5; // declares three more ints/ initializing // d and f.
3. byte z = 22; // initializes z.
4. double pi = 3.14159; // declares an approximation of pi.
5. char x = 'x'; // the variable x has the value 'x'.
```

www.bindownload.com

شناسه هایی که انتخاب می کنید هیچ عامل ذاتی در نام خود ندارند که نوع آنها را مشخص نماید . بسیاری از خوانندگان بیاد می آورند زمانی را که FORTRAN کلیه شناسه های از I تا N را پیش تعریف نمود تا از نوع INTEGER باشند ، در حالیکه سایر شناسه ها از نوع REAL بودند . جاوا به هر یک از شناسه های متناسب شکل گرفته امکان اختیار هر نوع اعلان شده را داده است .

مقدار دهی اولیه پویا Dynamic initialization

اگر چه مثالهای قبلی از ثابت ها بعنوان مقدار ده اولیه استفاده کرده اند اما جاوا امکان مقداردهی اولیه بصورت پویا را نیز فراهم آورده است . این موضوع با استفاده از هر عبارتی که در زمان اعلان متغیر باشد ، انجام می گیرد. بعنوان مثال ، در زیر برنامه کوتاهی را مشاهده می کنید که طول ضلع یک مثلث قائم الزاویه را با داشتن طول دو ضلع مقابل محاسبه می کند :

```
class DynInit {

    public static void main(String args[] ){

        double a = 3.0, b = 4.0;

        // c is dynamically initialized

         double c = Math.sqrt(a * a + b * b);

         System.out.println("Hypotenuse is " + c);

    }

}
```

در اینجا سه متغیر محلی a ، b، c، اعلان شده اند . دو تای اولی توسط ثابت ها مقدار دهی اولیه شده اند . اما متغیر C بصورت پویا و بر حسب طول اضلاع مثلث قائم الزاویه (بنابر قانون فیثاغورث) مقدار دهی اولیه می شود . این برنامه از یکی از روشهای توکار جاوا یعنی sqrt () که عضوی از کلاس Math بوده و ریشه دوم آرگومانهای خود را محاسبه میکند استفاده کرده است . نکته کلیدی اینجا است که عبارت مقدار دهی اولیه ممکن است از هر یک از اجزائ معتبر در زمان مقدار دهی اولیه ، شامل فراخوانی روشها ، سایر متغیرها یا الفاظ استفاده نماید .

قلمرو زمان حیات متغیرها

www.bindownload.com

تابحال کلیه متغیرهای استفاده شده ، در زمان شروع روش main() اعلان می شدند .اما جاوا همچنین به متغیرها امکان می دهد تا درون یک بلوک نیز اعلام شوند . همانطوریکه قبلا" توضیح دادیم ، یک بلوک با یک ابرو باز و یک ابرو بسته محصور می شود : یک بلوک تعریف کننده یک قلمرو است . بدین ترتیب هر بار که یک بلوک جدید را شروع میکنید ، یک قلمرو جدید نیز بوجود می آورید . همانطوریکه احتمالا "از تجربیات برنامه نویسی قبلی بیاد دارید ، یک قلمرو (scope) تعیین کننده آن است که چه اشیائی برای سایر بخشهای برنامه قابل رویت هستند . این قلمرو همچنین زمان حیات (lifetime) آن اشیائ را تعیین می کند . اکثر زبانهای کامپیوتری دو طبقه بندی از قلمروها را تعریف می کنند : سراسری (global) و محلی . (local) اما این قلمروهای سنتی بخوبی با مدل موکد شی ئ گرایی جاوا مطابقت ندارند . اگر چه در جاوا هم می توان مقادیری را بعنوان قلمرو سراسری ایجاد نمود ، اما این فقط یک نوع استثنائ است و عمومیت ندارد . در جاوا قلمرو اصلی همانهایی هستند که توسط یک کلاس یا یک روش تعریف می شوند . حتی همین تمایز نیز تا حدی ساختگی و مصنوعی است . اما از آنجاییکه قلمرو کلاس دارای مشخصات و خصلتهای منحصر بفردی است که قابل استفاده در قلمرو تعریف شده توسط روش نیست ، این تمایز تا حدی محسوس خواهد بود . بخاطر تفاوتهای موجود ، بحث قلمرو کلاس (و متغیرهای اعلان شده داخل آن) این مبحث بتعوق افتاده است . در حال حاضر فقط قلمروهای تعریف شده توسط یک روش یا داخل یک روش را بررسی می کنیم . قلمرو تعریف شده توسط یک روش با یک ابروی باز شروع می شود. اما اگر آن روش دارای پارامترهایی باشد ، آنها نیز داخل قلمرو روش گنجانده خواهند شد . بعدا "نگاه دقیقتری به پارامترها خواهیم داشت و فعلا" کافی است بدانیم که پارامترها مشابه هر متغیر دیگری در یک روش کار می کنند . بعنوان یک قانون عمومی ، متغیرهای اعلان شده داخل یک قلمرو برای کدهایی که خارج از قلمرو تعریف می شوند ، قابل رویت نخواهند بود (قابل دسترسی نیستند .(بدین ترتیب ، هنگامیکه یک متغیر را درون یک قلمرو اعلان می کنید ، در حقیقت آن متغیر را محلی دانسته و آن را در مقابل دستیابیها و تغییرات غیر مجاز محافظت می کنید . در حقیقت ،قوانین قلمرو اساس کپسول سازی را فراهم می کنند . قلمروها را می توان بصورت تودرتو (nesting) محفوظ داشت . بعنوان مثال ، هر زمان یک بلوک کد ایجاد کنید ، یک قلمرو جدید تودرتو ایجاد نموده اید . هنگامیکه این واقعهروی می دهد ، قلمرو بیرونی ، قلمرو درونی را دربرمی گیرد . این بدان معنی است که اشیائ اعلان شده در قلمرو بیرونی برای کدهای داخل قلمرو درونی قابل رویت هستند اما عکس این قضیه صادق نیست . اشیائاعلان شده داخل قلمرو درونی برای بیرون قلمرو قابل رویت نخواهند بود . برای درک تاثیر قلمروهای تودرتو ، برناه ریز را در نظر بگیرید :

```
// Demonstrate block scope.
 class Scope {
 public static void main(String args[] ){

    int x; // known to all code within main

    x = 10;
   if(x == 10 ){ // start new scope
```

www.bindownload.com

```
    int y = 20; // known only to this bock
     // x and y both known here.
     System.out.println("x and y :" + x + " " + y);
     x = y * 2;
  }
  // y = 100 :// Error! y not known here

  // x is still known here.
    System.out.println("x is " + x);
  }
  }
```

همانطوریکه توضیحات نشان می دهند ، متغیر x در ابتدای قلمروی main() اعلان شده و برای کلیه کدهای متعاقب داخل main()قابل دسترسی می باشد . داخل بلوک if متغیر y اعلان شده است . از آنجاییکه یک بلوک معرف یک قلمرو است ، yفقط برای سایر کدهای داخل بلوک خود قابل رویت است . این دلیل آن است که خارج بلوک مربوطه ، خط y=100 در خارج توضیح داده شده است . اگر نشانه توضیح راهنمایی را تغییر مکان دهید ، یک خطای زمان کامپایل (compile-time error) اتفاق می افتد چون y برای بیرون از بلوک خود قابل رویت نیست . داخل بلوک if متغیر x قابل استفاده است زیرا کدهای داخل یک بلوک (منظور یک قلمرو تودرتو شده است) به متغیرهای اعلان شده در یک قلمرو دربرگیرنده دسترسی دارند . داخل یک بلوک ، در هر لحظه ای می توان متغیرها را اعلان نمود ، اما فقط زمانی معتبر می شوند که اعلان شده باشند . بدین ترتیب اگر یک متغیر را در ابتدای یک روش اعلان می کنید، برای کلیه کدهای داخل آن روش قابل دسترس خواهد بود. بالعکس اگر یک متغیر را در انتهای یک بلوک اعلان کنید ، هیچ فایده ای ندارد چون هیچیک از کدها به آن دسترسی ندارند . بعنوان مثال این قطعه از برنامه غیر معتبر است چون نمی توان از count قبل از اعلان آن استفاده نمود :

```
// This fragment is wrong!
count = 100; // oops! cannot use count before it is declared!
int count;
```

یک نکته مهم دیگر در اینجا وجود دارد که باید بخاطر بسپارید: متغیرها زمانی ایجاد می شوند که قلمرو آن ها وارد شده باشد ، و زمانی خراب می شوند که قلمرو آنها ترک شده باشد . یعنی یک متغیر هربار که خارج از قلمروش برود ، دیگر مقدار خود را نگهداری

www.bindownload.com

نخواهد کرد . بنابراین ، متغیرهای اعلان شده داخل یک روش مقادیر خود را بین فراخوانی های آن روش نگهداری نمی کنند . همچنین یک متغیر اعلان شده داخل یک بلوک ، وقتی که بلوک ترک شده باشد ، مقدار خود را از دست خواهد داد . بنابراین ، زمان حیات (lifetime)یک متغیر محدود به قلمرو آن می باشد . اگر اعلان یک متغیر شامل مقدار دهی اولیه آن باشد ، آنگاه هر زمان که به بلوک مربوطه واردشویم ، آن متغیر مجددا" مقدار دهی اولیه خواهد شد . بعنوان مثال برنامه زیر را در نظر بگیرید :

```
// Demonstrate lifetime of a variable.
class LifeTime {
public static void main(String args[] ){
int x;

for(x = 0; x < 3; x++ ){
int y =- 1; // y is initialized each time block is entered
System.out.println("y is :" + y); // this always prints- 1
y = 100;
System.out.println("y is now :" + y);
}
}
}
```

خروجی تولید شده توسط این برنامه بقرار زیر است :

```
y is- :1
y is now:100
y is- :1
y is now:100
y is- :1
y is now:100
```

همانطوریکه مشاهده می کنید ، هر بار که به حلقه for داخلی وارد می شویم ، y همواره بطور مکرر مقدار اولیه 1- را اختیار میکند . اگر چه بلافاصله به این متغیر مقدار 100 نسبت داده می شود، اما هر بار نیز مقدار خود را از دست میدهد .و بالاخره آخرین نکته : اگر چه میتوان بلوکها را تودرتو نمود، اما نمیتوانید متغیری را اعلان کنید که اسم آن مشابه اسم متغیری در قلمرو بیرونی باشد. از این نظر

www.bindownload.com

جاوا با زبانهایC و++C و متفاوت است . در زیر مثالی را مشاهده می کنید که در آن تلاش شده تا دو متغیر جدا از هم با اسم اعلان شوند . در جاوا اینکار مجاز نیست . درC و++C و این امر مجاز بوده و دو bar کاملا" جدا خواهند ماند .

```
// This program will not compile
class ScopeErr {
    public static void main(String args[] ){
    int bar = 1;
      { // creates a new scope
          int bar = 2; // Compile-time error -- bar already defined!
      }
    }
}
```

تبدیل خودکار و تبدیل غیر خودکار انواع اگر تجربه قبلی برنامه نویسی داشته اید ، پس می دانید که کاملا" طبیعی است که مقداری از یک نوع را به متغیری از نوع دیگر نسبت دهیم . اگر این دو نوع سازگار باشند ، آنگاه جاوا بطور خودکار این تبدیل (conversion)را انجام می دهد . بعنوان مثال ، همواره امکان دارد که مقدار int را به یک متغیر long نسبت داد . اما همه انواع با یکدیگر سازگاری ندارند ، بنابراین هر گونه تبدیل انواع مجاز نخواهد بود . بعنوان نمونه ، هیچ تبدیلی ازdouble به byte تعریف نشده است . خوشبختانه ، امکان انجام تبدیلات بین انواع غیر سازگار هم وجود دارد . برای انجام اینکار ، باید از تبدیل cast استفاده کنید که امکان یک تبدیل صریح بین انواع غیر سازگار را بوجود می آورد . اجازه دهید تا نگاه دقیقتری به تبدیل خودکار انواع و تبدیل cast داشته باشیم .

تبدیل خودکار در جاوا Java's Automatic conyersions

هنگامیکه یک نوع داده به یک متغیر از نوع دیگر نسبت داده می شود ، اگر دو شرط زیر فراهم باشد ، یک تبدیل خودکار نوع انجام خواهد شد : ؤ دو نوع با یکدیگر سازگار باشند . ؤ نوع مقصد بزرگتر از نوع منبع باشد . هنگامیکه این دو شرط برقرار باشد ، یک تبدیل پهن کننده (widening) اتفاق می افتد . برای مثال نوع int همواره باندازه کافی بزرگ است تا کلیه مقادیر معتبر byte را دربرگیرد، بنابراین نیازی به دستور صریح تبدیل cast وجود ندارد .در تبدیلات پهن کننده ، انواع رقمی شامل انواع عدد صحیح و عدد اعشاری با هر یک از انواع سازگاری دارند . اما انواع رقمی با انواعchar وboolean و سازگار نیستند . همچنین انواعchar و boolean و با یکدیگر سازگار نیستند . همانطوریکه قبلا" ذکر شد ، جاوا هنگام ذخیره سازی یک ثابت عدد صحیح لفظی (Literal integer constant)به متغیرهای از انواعbyte ، short،و long و ، یک تبدیل خودکار نوع را انجام می دهد .

www.bindownload.com

تبدیل غیر خودکار انواع ناسازگار

اگر چه تبدیلات خودکار انواع بسیار سودمند هستند ، اما جوابگوی همه نیازها نیستند . بعنوان مثال ، ممکن است بخواهید یک مقدار intرا به یک متغیر byte نسبت دهید. این تبدیل بطور خودکار انجام نمی گیرد، زیرا یکbyteازint ز کوچکتر است .این نوع خاص از تبدیلات را گاهی تبدیل باریک کننده (narrowing conversions) می نامند ، زیرا بطور صریح مقدار را آنقدر باریک تر و کم عرض تر می کنید تا با نوع هدف سازگاری یابد . برای ایجاد یک تبدیل بین دو نوع ناسازگار ، باید از cast استفاده نماییدcast . یک تبدیل نوع کاملا" صریح است . شکل عمومی آن بقرار زیر می باشدvalue(target - type) :)

نوع نوع مقصد یا هدف

در اینجا نوع هدف ، همان نوعی است که مایلیم مقدار مشخص شده را به آن تبدیل کنیم . بعنوان مثال ، قطعه زیر از یک برنامه تبدیل غیر خودکار ازint به byte را اجرا می کند . اگر مقدار integer بزرگتر از دامنه یک byte باشد ، این مقدار به مدول(باقیمانده تقسیم یک integer بر دامنه byte) کاهش خواهد یافت;int a .

```
byte b;
//...
b =( byte )a;
```

هر گاه که یک مقدار اعشاری به یک عدد صحیح نسبت داده شود ، شکل دیگری از تبدیل اتفاق می افتد : بریدن ، truncation . همانطوریکه می دانید ، اعداد صحیح دارای قسمت اعشاری نیستند . بنابراین هنگامیکه یک مقدار اعشاری به یک نوع عدد صحیح نسبت داده می شود ، جزئ اعشاری از بین خواهد رفت (بریده خواهد شد) . (بعنوان مثال ، اگر مقدار 1.23 را به یک عدد صحیح نسبت دهیم ، مقدار حاصله فقط عدد 1 می باشد . مقدار 0.23 بریده (truncated) خواهد شد . البته اگر اندازه اجزائ عدد کلی آنچنان بزرگ باشد که در نوع عدد صحیح مقصد نگنجد ، آنگاه مقدار فوق به مدول دامنه نوع هدف کاهش خواهد یافت . برنامه زیر نشان دهنده برخی از تبدیلات انواع است که مستلزم تبدیل cast می باشند :

```
// Demonstrate casts.
class Conversion {
public static void main(String args[] ){
bytt b;
int i = 257;
double d = 323.142;
System.out.println("\nConversion of int to byte.");
```

www.bindownload.com

```
b =( byte )i;
System.out.println("i and b " + i + " " + b);
System.out.println("\nConversion of double to int.");
i =( int )d;
System.out.println("d and i " + d + " " + i);

System.out.println("\nConversion of double to byte.");
b =( byte )d;
System.out/println("d and b " + d + " " + b);
}
}
```

خروجی این برنامه بقرار زیر می باشد :

```
Conversion of int to byte.
i and b 257 1

Conversion of double to int.
d and i 323.142 323

Conversion of double to byte.
d and b 323.142 67
```

اکنون به هر یک از این تبدیلات نگاه می کنیم . هنگامیکه مقدار 257 از طریق تبدیل cast به یک byte منتسب می شود ، نتیجه برابر باقیمانده تقسیم 257 بر 256 دامنه byte (یعنی عدد 1 است . هنگامیکه d به یک int تبدیل می شود ،) بخش خواهند رفت . هنگامیکه d به یک byte تبدیل می شود ،

نگاهی دقیقتر به متغیر

اکنون که انواع توکار را بطور رسمی توضیح داده ایم ، نگاه دقیقتری به این متغیر خواهیم داشت .

boolean	False,true		char	16.bits, one character
byte	one byte, integer		float	4bytes, single-precision.

www.bindownload.com

short	bytes, integer 2		double	8bytes, double-precision.
long	.bytes, integer 8		int	4bytes, integer.
void	Return type where no return value is .required		String	N byte

متغیر عدد صحیح integer literals

احتمالا" اعداد صحیح رایجترین نوع استفاده شده در برنامه های نوع بندی شده هستند. هر مقدار رقمی کلی یک لفظ عدد صحیح است . اعداد 1، 2،، 3، و 42 مثالهای روشنی هستند . این اعداد همگی مقادیر دهدهی می باشند ، بدین معنی که این اعداد در یک مبنای ده رقمی تعریف شده اند . دو مبنای دیگر نیز در متغیر عدد صحیح قابل استفاده هستند : مبنای هشت (octal) و مبنای 1 .(hexadecimal()مقادیر در مبنای هشت در جاوا با یک رقم 0 پیش آیند مشخص میشوند. ارقام دهدهی معمولی نمی توانند رقم 0 پیش آیند داشته باشند . بنابراین مقدار بظاهر معتبر 09 خطایی را در کامپایلر تولید می کند، زیرا رقم 9 خارج از دامنه 0تا 7ا مبنای هشت قرار دارد .یکی دیگر از مبناهای رایج برای ارقام مورد استفاده برنامه نویسان ، مبنای 16 می باشد که با مدول اندازه های کلمه 8 تایی نظیر 8 ، 16 ، 32 و 64 بیتی کاملا "سازگاری دارد . یک ثابت درمبنای 16 را توسط OX یاox ا مشخص می کنید . دامنه یک رقم در مبنای 16 از رقم 0 تا 15 و حروفA تاF ا (یاa تاf ا) بعنوان جایگزین ارقام 10 تا 15 می باشد . متغیر عدد صحیح یک مقدار int تولید می کنند که در جاوا یک مقدار عدد صحیح 32بیتی است . از آنجاییکه جاوا شدیدا" نوع بندی شده است ، ممکن است تعجب کنید که چگونه می توان یک لفظ عدد صحیح را به یکی دیگر از انواع عدد صحیح جاوا نظیر byte

یاlong ا نسبت داد ، بدون اینکه خطای عدم سازگاری انواع بوجود آید . خوشبختانه چنین حالتهایی بسادگی اداره می شوند . هنگامیکه یک لفظ عدد صحیح به یک متغیرbyte یاshort ا منتسب می شود، اگر مقدار لفظ داخل محدوده نوع هدف قرار داشته باشد ، خطایی تولید نخواهد شد . همچنین همواره می توان یک لفظ عدد صحیح را به یک متغیر long منتسب نمود . اما برای مشخص نمودن یک لفظ long باید بطور صریح به کامپایلر بگویید که مقدار لفظ از نوع long است . اینکار را با الحاق یک حرف L بزرگ یا کوچک به لفظ انجام می دهیم . بعنوان مثال ،

ox7fffffffffffffffL

9223372036854775807L

بزرگترین Long می باشد .

متغیر عدد اعشاری Floating-point literals

www.bindownload.com

ارقام اعشاری معرف مقادیر دهدهی با اجزائ کسری می باشند . آنها را می توان به شکل استاندارد یا به شکل علامتگذاری علمی بیان نمود . نشانه گذاری استاندارد شامل یک جزئ عدد صحیح است که بعد از آن یک نقطه و بعد از آن جزئ کسری عدد قرار می گیرد . بعنوان مثال 2.0 یا 3.14159 یا 0.6667 معرف نشانه گذاری استاندارد معتبر در ارقام اعشاری هستند . نشانه گذاری علمی از یک نشانه گذاری استاندارد نقطه مخصوص اعشاری بعلاوه یک پیوند که مشخص کننده توانی از عدد 10 است که باید در عدد ضرب شود استفاده می کند . توان (نما) را توسط علامتE یاe ا که یک رقم دهدهی بدنبال آن می آید و ممکن است مثبت یا منفی باشد ، نشان می دهیم.

مثل 6.022 E23یا 314159-05 Eو . 2e+100

متغیر عدد اعشاری در جاوا بصورت پیش فرض دارای دقت مضاعف (double) هستند . برای مشخص نمودن یک لفظ float باید یک حرفF یاf ا را به ثابت الحاق نمایید . همچنین میتوانید بطور صریح یک لفظ double را با الحاق یک حرفD یاd ا نیز انجام دهید . انجام اینکار البته اضافی است . نوع double پیش فرض 64 بیت حافظه را مصرف می کند در حالیکه نوع کم دقت تر float مستلزم 32 بیت حافظه است .

متغیر Boolean

متغیر boolean بسیار ساده هستند. یک مقدار boolean فقط دو مقدار منطقی شاملtrue وfalse و می تواند داشته باشد . مقادیرtrue وfalse و هرگز به رقم تبدیل نمی شوند . در جاوا لفظ true مساوی یک نبوده ، همچنانکه لفظ false معادل صفر نیست . در جاوا ، آنها را فقط می توان به متغیرهای اعلان شده بعنوان boolean منتسب نمود و یا در عباراتی با عملگرهای booleanاستفاده نمود .

متغیر کاراکترها Character literals

کاراکترهای جاوا در مجموعه کاراکتر کدهای جهانی نمایه سازی شده اند . آنها مقادیر 16 بیتی هستند که قابل تبدیل به اعداد صحیح بوده و با عملگرهای عدد صحیح نظیر عملگرهای اضافه و کسر نمودن اداره می شوند . یک کاراکتر لفظی همواره داخل یک علامت ' ' معرفی می شود . کلیه کاراکترهای ASCII قابل رویت می توانند بطور مستقیم به داخل این علامت وارد شوند ، مثل'a' یا'z' ایا '@'ا . برای کاراکترهایی که امکان ورود مستقیم را ندارند ، چندین پیش آیند وجود دارند که امکان ورود کاراکتر دلخواه را فراهم مینمایند، نظیر '\' برای ورود خود کاراکتر و '\n' برای کاراکتر خط جدید . همچنین مکانیسمی برای ورودی مستقیم مقدار یک کاراکتر در مبنای هشت یا شانزده وجود دارد . برای نشانه گذاری مبنای هشت از علامت \ که توسط یک عدد سه رقمی دنبال

www.bindownload.com

میشود،استفاده کنید. بعنوان مثال '\141' همان حرف 'a' است . برای مبنای شانزده از علامت (\u) استفاده کنید و بعد از آن دقیقا" چهار رقم مبنای شانزده . بعنوان مثال '\uoo61' که معادل حرف 'a' در استاندارد lso-latin-1 است چون بایت بالایی آن صفر است '\ua432' . یک کاراکتر Katakana ژاپنی است . جدول زیر پیش آیندهای کاراکترها را نشان می دهد .

توصیف آن پیش آیند

- کاراکتر مبنای هشت \ddd (ddd)
- کاراکتر کد جهانی مبنای شانزده \uxxxx (xxxx)
- علامت تکی نقل قول '\0'
- علامت جفتی نقل قول| Backslash '\\'
- کاراکتر برگشت به سر خط\r
- خط جدید \n
- تغذیه فرم \t \f
- \b Backspace

متغیر String

متغیر رشته ای در جاوا نظیر سایر زبانهای برنامه نویسی مشخص می شوند قرار دادن یک دنباله از کاراکترها بین یک جفت از علامات نقل قول ، در زیر نمونه هایی از متغیر رشته ای را مشاهده می کنید .

```
"Hello world"
"tow\nlines"
"\"This is in quotes\"
```

پیش آیندها و نشانه گذاریهای مبنای هشت / شانزده که برای متغیر کاراکترها توصیف شد ، بهمان روش در داخل متغیر رشته ای کار می کنند . یک نکته مهم درباره رشته های جاوا این است که آنها باید روی یک خط شروع شده و پایان یابد . برخلاف زبانهای دیگر در جاوا ادامه خط در خطهای دیگر مجاز نیست . نکته : حتما" می دانید که در اکثر زبانهای دیگر شامل C و C++ و ، رشته ها بعنوان آرایه های کاراکتری پیاده سازی می شوند . اما در جاوا این حالت وجود ندارد . رشته ها از نوع اشیائ هستند . بعدا" می بینید از آنجاییکه جاوا پیاده سازی می کند .

انواع ساده The simple Types

www.bindownload.com

جاوا هشت نوع ساده (یا ابتدایی) از داده را تعریف می کندint bbyte short : ، long، char، float، double، boolean، این انواع را می توان در چهار گروه بشرح زیر دسته بندی نمود :

integers اعداد صحیح:

این گروه دربرگیرندهbyte ، short، int،و long و میباشد که مختص ارقام علامتدار مقدار کل (whole-valued signed numbers)میباشد .

floating-point number اعداد اعشاری :

این گروه دربرگیرنده float وdouble است که معرف اعدادی است با دقت زیاد .

characters کاراکترها : (این گروه فقط شامل char بوده که نشانه هایی نظیر حروف و ارقام را در یک مجموعه خاص از کاراکترها معرفی می کند .

Boolean بولی : این گروه فقط شامل boolean است . که نوع خاصی ازمعرفی و بیان مقادیر صحیح / ناصحیح می باشد .

شما می توانید از این انواع همانطوریکه هستند استفاده کرده ، یا آرایه ها و انواع کلاسهای خود را بسازید . انواع اتمی معرف مقادیر تکی و نه اشیائ پیچیده هستند . اگر چه جاوا همواره شی ئ گرا است ، اما انواع ساده اینطور نیستند . این انواع ، مشابه انواع ساده ای هستند که در اکثر زبانهای غیر شی ئ گرا مشاهده می شود . دلیل این امر کارایی است . ساختن انواع ساده در اشیائ سبب افت بیش از حد کارایی و عملکرد می شود . انواع ساده بگونه ای تعریف شده اند تا یک دامنه روشن و رفتاری ریاضی داشته باشند . و زبانهایی نظیرC و++C و امکان می دهند تا اندازه یک عدد صحیح براساس ملاحظات مربوط به محیط اجرایی تغییر یابد . اما جاوا متفاوت عمل می کند . بدلیل نیازهای موجود برای قابلیت حمل جاوا ، کلیه انواع داده در این زبان دارای یک دامنه کاملا" تعریف شده هستند . بعنوان مثال یک int همیشه 32 بیتی است ، صرفنظر از اینکه زیر بنای خاص محیطی آن چگونه باشد . این حالت به برنامه های نوشته شده

اجازه می دهد تا با اطمینان و بدون در نظر گرفتن معماری خاص یک ماشین اجرا شوند. در حالیکه مشخص کردن دقیق اندازه یک عدد صحیح ممکن است در برخی محیط ها سبب افت عملکرد شود ، اما برای رسیدن به خاصیت قابلیت حمل پرداخت .

انواع اعداد اعشاری

اعداد اعشاری یا همان اعداد حقیقی برای ارزش گذاری عبارتهایی که نیازمند دقت بیشتری هستند ، استفاده می شوند . بعنوان نمونه ، محاسباتی نظیر ریشه دوم و محاسبات مثلثاتی نظیر سینوس و کسینوس منجربه جوابهایی می شوند که برای تعیین دقت آن نیاز به نوع عدد اعشاری می باشد . جاوا یک مجموعه استاندارد (IEEE-754) از انواع عدد اعشاری و عملگرها را پیاده سازی می کند. دو نوع عدد اعشاری تعریف شده یعنیfloat وdouble و هستند که بترتیب معرف دقت معمولی و مضاعف می باشند .

پهنا و دامنه آنها را در زیر نشان داده ایم :

دامنه پهنا بر حسب تعداد بیت نام

double 64 1.7e-308 to 1.7e+308

www.bindownload.com

float 32 3.4e-038 to 3.4e+038

هر یک از انواع اعشاری را متعاقبا" مورد بررسی قرار می دهیم .

float

این نوع مشخص کننده یک مقدار با دقت معمولی بوده که از 32 بایت حافظه استفاده می کند . دقت معمول روی بعضی پردازنده ها سریعتر بوده و نسبت به دقت مضاعف نیمی از فضا را اشغال می کند ، اما هنگامیکه مقادیر خیلی بزرگ یا خیلی کوچک باشند ، دقت خود را از دست میدهد . متغیرهای نوع float برای زمانی مناسب هستند که از یک عضو کسری استفاده می کنید اما نیازی به دقت خیلی زیاد ندارید . بعنوان مثال ، نوع float برای معرفی دلار و سنت بسیار مناسب است float hightemp/ lowtemp; .

double

دقت مضاعف که با واژه کلیدی double معین می شود برای ذخیره کردن یک مقدار 64بیت فضا را اشغال می کند . دقت مضاعف روی برخی پردازنده های جدید که برای محاسبات ریاضی با سرعت زیاد بهینه شده اند ، واقعا" سریعتر از دقت معمولی عمل می کند . کلیه توابع مثلثاتی نظیر sin() ، cos()و sqrt() مقادیر مضاعف را برمی گردانند . هنگام اجرای محاسبات مکرر که نیاز به حفظ دقت دارید و یا هنگام کار با ارقام خیلی بزرگ double بهترین انتخاب است . در زیر برنامه ای را مشاهده می کنید که از doubleاستفاده نمود تا محیط یک دایره را محاسبه کند :

```
// Compute the area of a circle.
class Area {
public static void main(String args[] ){
double pi/ r/ a;
r = 10.8; // radius of circle
pi = 3.1416; // pi/ approximately
a = pi * r * r; // compute area
System.out.println("Area of circle is " + a);
}
}
```

کاراکترها

www.bindownload.com

در جاوا از نوع داده char برای ذخیره کردن کاراکترها استفاده می شود . اما برنامه نویسانC و++C و آگاه باشند که char در جاوا مشابه char در زبانهای C و++C نیست . در زبانهایC و++C و ، نوع char یک نوع عدد صحیح با پهنای 8 بیت است . اما جاوا متفاوت عمل می کند . جاوا از کدهای جهانی (unicode) برای معرفی کاراکترها استفاده می کند . کدهای جهانی یک مجموعه کاملا" جهانی از کاراکترها هستند که می توانند همه کاراکترها را معرفی نمایند . این مجموعه شامل دهها مجموعه کوچک تر کاراکتری نظیر Latin ، Greek، Arabic، Cyrillic، Hebrew، Katakana ، Hangul، و امثال آن است .

برای این منظور ، 16 بیت مورد نیاز است . بنابراین char در جاوا یک نوع 16 بیتی است . دامنهchar از 0ز تا 536/65 می باشد . در نوع char مقدار منفی وجود ندارد . مجموعه استاندارد کاراکترها موسوم به ASCII همچون گذشته دارای دامنه از 0 تا 127 و مجموعه کاراکترهای 8 بیتی توسعه یافته موسوم به lso-Latin-1 دارای دامنه از 0 تا 255 می باشند .

چون در جاوا امکان نوشتن ریز برنامه ها برای کاربری جهانی وجود دارد ، بنظر می رسد که بهتر است جاوا از کدهای جهانی برای معرفی کاراکترها استفاده نماید .

البته بکار بردن کدهای جهانی درمورد زبانهایی نظیر انگلیسی ،آلمانی ،اسپانیایی یا فرانسوی که کاراکترهای آنها را می توان براحتی داخل 8 بیت جای داد ، تا حدی سبب نزول کارآیی خواهد شد . اما این بهایی است که برای رسیدن به قابلیت حمل جهانی در برنامه ها باید پرداخت . نکته : اطلاعات بیشتر درباره کدهای جهانی را در آدرسهای وب زیر پیدا خواهید نمود :

http://www .unicode .org

http://www .stonehand .com/unicode .html

در زیر برنامه ای را مشاهده می کنید که متغیرهای char را نشان می دهد :

```
// Demonstrate char data type.
class CharDemo {
public static void main(String args[] ){
char ch1/ ch2;

ch1 = 88; // code for X
ch2 = 'Y';

System.out.print("ch1 and ch2 :");
System.out.println(ch1 + " " + ch2);
}
}
```

www.bindownload.com

این برنامه خروجی زیر را نشان خواهد داد

```
: ch1 and ch2 :xy
```

دقت کنید که مقدار 88به ch1 نسبت داده شده ، که مقدار متناظر با حرف x در کد) ASCII و کد جهانی) است . قبلا" هم گفتیم که مجموعه کاراکتریASCII ´127 مقدار اولیه در مجموعه کاراکتری کدهای جهانی را اشغال کرده است . بهمین دلیل کلیه فوت و فنهای قدیمی که قبلا "با کاراکترها پیاده کرده اید ، در جاوا نیز به خوبی جواب می دهند .

اگر چه انواع char عدد صحیح محسوب نمی شوند ، اما در بسیاری از شرایط می توانید مشابه عدد صحیح با آنها رفتار کنید . بدین ترتیب قادرید دو کاراکتر را با هم جمع نموده و یا اینکه مقدار یک متغیر کارکتری را کاهش دهید . بعنوان مثال ، برنامه زیر را در نظر بگیرید :

```
// char variables behave like integers.
class CharDemo2 {
public static void main(String args[] ){
char ch1;
ch1 = 'X';
System.out.println("ch1 contains " + ch1);
ch1++; // increment ch1
System.out.println("ch1 is now " + ch1);
}
}
```

خروجی این برنامه بشرح زیر خواهد بود

```
: ch1 contains x
ch1 is now y
```

در برنامه ابتدا مقدارx به ch1 داده میشود . سپس ch1 افزایش می یابد . این روال باعث می شود تا ch1 حرف y را اختیار کند، که کاراکتر بعدی در ترتیب ASCII و کدهای جهانی)می باشد .

boolean

جاوا یک نوع ساده موسوم به boolean برای مقادیر منطقی دارد . این نوع فقط یکی از مقادیر ممکنtrue یاfalse ا را اختیار می

www.bindownload.com

کند . این نوعی است که توسط کلیه عملگرهای رابطه ای نظیر b شرطی که دستورهای کنترلی نظیر if و for و را مدیریت می کنند ، استفاده می شود .

در زیر برنامه ای مشاهده می کنید که نوع boolean را نشان می دهد :

```
// Demonstrate boolean values.
class BoolTest {
public static void main(String args[] ){
boolean b;
b = false;
System.out.println("b is " + b);
b = true;
System.out.println("b is " + b);
// a boolean value can control the if statement
if(b )System.out.println("This is executed.");
b = false;
if(b )System.out.println("This is not executed.");
// outcome of a relational operator is a boolean value
System.out.println("10 > 9 is " +( 10 > 9));
}
}
```

خروجی برنامه فوق بقرار زیر خواهد بود :

```
b is false
b is true
This is executed.
10>9 is true
```

درباره این برنامه سه نکته جالب توجه وجود دارد . اول اینکه وقتی که مقدار boolean توسط println() خارج می شود ، می بینید که "true" یا "false" ا بنمایش درمی آید . دوم اینکه یک متغیر boolean بتنهایی برای کنترل دستور if کفایت می کند . دیگر نیازی به نوشتن یک دستور if بقرار زیر نخواهد بود(if(b == true... :

یک مقدار > سوم اینکه ، پی آمد یک عملگر رابطه ای نظیر boolean است . بهمین دلیل است که عبارت 10>9 مقدار true را نمایش می دهد . علاوه بر این ، مجموعه ی از پرانتزهایی که عبارت 10>9 را محصور کرده اند ، ضروری است زیرا عملگر

www.bindownload.com

عملگرهای منطقی بولی boolean

عملگرهای منطقی بولی که در زیر نشان داده ایم فقط روی عملوندهای بولی عمل می کنند . کلیه عملگرهای منطقی باینری دو مقدار booleanرا ترکیب می کنند تا یک مقدار منتج boolean ایجاد نمایند .

نتیجه آن عملگر

AND

منطقیOR &

منطقیXOR |

منطقی (خارجOR ^)

مدار کوتاهAND ||

مدار کوتاهNOT &&

یکانی منطقی !

انتساب =& AND

انتساب =| OR

انتساب =^ XOR

مساوی با ==

نامساوی با =!

سه تایی ?: if-then-else

عملگرهای بولی منطقی& ، |،، ^، روی مقادیر Boolean همانطوری که روی بیت های یک عدد صحیح رفتار می کنند ، عمل خواهند کرد . عملگر منطقی ! حالت بولی را معکوس می کند :

!false=true t!true=false

جدول بعدی تاثیرات هر یک از عملیات منطقی را نشان می دهد :

www.bindownload.com

A B A|B A&B A^B !A

False False False False False True

True False True False True False

False True True False True True

True True True True False False

در زیر برنامه ای را مشاهده می کنید که تقریبا" با مثال Bitlogic قبلی برابر است ، اما در اینجا بجای بیت های باینری روی مقادیر منطقی بولی عمل می کند :

```
// Demonstrate the boolean logical operators.
class BoolLogic {
public static void main(String args[] ){
boolean a = true;
boolean b = false;
boolean c = a | b;
boolean d = a & b;
boolean e = a ^ b;
boolean f =( !a & b )|( a & !b);
boolean g = !a;
System.out.println(" a = " + a);
System.out.println(" b = " + b);
System.out.println(" a|b = " + c);
System.out.println(" a&b = " + d);
System.out.println(" a^b = " + e);
System.out.println("!a&b|a&!b = " + f);
System.out.println(" !a = " + g);
}
}
```

پس از اجرای این برنامه ، شما همان قوانین منطقی که برای بیت ها صادق بود در مورد مقادیر boolean مشاهده می کنید . در خروجی این برنامه مشاهده می کنید که معرفی رشته ای یک مقدار بولی درجاوا یکی از مقادیر لفظیtrue یاfalse است .

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

www.bindownload.com

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

عملگرهای منطقی مدار کوتاه

جاوا دو عملگر بولی بسیار جالب دارد که در اکثر زبانهای دیگر برنامه نویسی وجود ندارند . این ها روایت ثانویه عملگرهایAND و ORو بولی هستند و بعنوان عملگرهای منطقی مدار کوتاه معرفی شده اند. در جدول قبلی می بینید که عملگر OR هرگاه که A معادل trueباشد، منجر به true می شود، صرفنظر از اینکه B چه باشد .بطور مشابه ، عملگر AND هرگاه A معادل false باشد منجر به falseمی شود. صرفنظر از اینکه B چه باشد . اگر از اشکال ||و&& و بجای |و& و استفاده کنید، هنگامیکه حاصل یک عبارت می تواند توسط عملوند چپ بتنهایی تعیین شود ، جاوا دیگر به ارزیابی عملوند راست نخواهد پرداخت . این حالت بسیار سودمند است بخصوص وقتی که عملوند سمت راست بستگی به عملوند سمت چپ وtrue یاfalse ا بودن آن برای درست عمل کردن داشته باشد . بعنوان مثال ، کد قطعه ای زیر به شما نشان می دهد چگونه می توانید مزایای ارزیابی منطقی مدار کوتاه را استفاده نموده تا مطمئن شوید که عملیات تقسیم قبل از ارزیابی آن معتبر است .

```
if(denom != 0 && num / denom > 10)
```

از آنجاییکه شکل مدار کوتاه AND یعنی && استفاده شده است ، هنگامیکه denom صفر باشد ، خطر ایجاد یک استثنائ حین اجرا منتفی است . اگر همین خط از کد را با استفاده از رایت تکی AND یعنی & بنویسیم ، هر دو عملوند باید مورد ارزیابی قرار گیرند و هنگامیکه denom صفر باشد یک استثنائ حین اجرا بوجود می آید . در حالتهایی که شامل منطق بولی باشند : استفاده از ارزیابیهای مدار کوتاهAND وOR و یک روش استاندارد است که روایتهای تک کاراکتری عملگرها را منحصرا "برای عملیات رفتار بیتی قرار می دهد . اما استثنائاتی بر این قوانین وجود دارند . بعنوان مثال ، دستور زیر را در نظر بگیرید :

```
if(c==1 & e++ < 100 )   d = 100;
```

در اینجا استفاده از یک علامت & تکی اطمینان می دهد که عملیات افزایشی بهe

اعلان نمودن اشیائ

بدست آوردن اشیائ از یک کلاس ، نوعی پردازش دو مرحله ای است . اول ، باید یک متغیر از نوع همان کلاس اعلان نمایید . این متغیر یک شی ئ را تعریف نمی کند . در عوض ، متغیری است که می تواند به یک شی ئ ارجاع نماید .دوم ، باید یک کپی فیزیکی

www.bindownload.com

و واقعی از شی ئ بدست آورده و به آن متغیر منتسب کنید . می توانید اینکار را با استفاده از عملگر new انجام دهید . عملگر new بطور پویا (یعنی در حین اجرا) حافظه را برای یک شی ئ تخصیص داده و یک ارجاع به آن را برمی گرداند . این ارجاع (reference)کمابیش آدرس آن شی ئ در حافظه است که توسط new تخصیص یافته است . سپس این ارجاع در متغیر ذخیره می شود. بدین ترتیب ، در جاوا، کلیه اشیائ کلاس دار باید بصورت پویا تخصیص یابند. اجازه دهید که به جزئیات این روال دقت نماییم . در مثال قبلی ، یک خط مشابه خط زیر برای اعلان یک شی ئ ازنوع Box استفاده شده

```
Box mybox = new Box();
```

این دستور دو مرحله گفته شده را با یکدیگر ترکیب نموده است . برای اینکه هر یک از مراحل را روشن تر درک کنید، میتوان آن دستور را بصورت زیر بازنویسی نمود :

```
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```

خط اول ، myBoxرا بعنوان یک ارجاع به شیئی از نوع Box اعلان می کند . پس از اجرای این خط ، myboxمحتوی تهی (null)خواهد داشت که نشانگر آن است که هنوز شی ئ بطور واقعی بوجود نیامده است . هر تلاشی برای استفاده از myboxدر این مرحله سبب بروز خطای زمان کامپایل (compile-time error) خواهد شد . خط بعدی یک شی ئ واقعی را تخصیص داده و یک ارجاع از آن به mybox انجام می دهد . پس از اجرای خط دوم ، می توانید از mybox بعنوان یک شی ئ Box استفاده نمایید . اما در واقعیتmybox خیلی ساده آدرس حافظه شی ئ واقعی Box را نگهداری می کند . تاثیر این دو خط کد را در شکل زیر نشان داده ایم .

نکته : کسانی که با C++/C آشنایی دارند احتمالا" توجه نموده اند که ارجاعات شی ئ مشابه اشاره گرها هستند . این تشابه تا حدود زیادی صحیح است . یک ارجاع شی ئ (object reference) مشابه یک اشاره گر حافظه است . مهمترین تفاوت و کلید ایمنی در جاوا این است که نمی توانید از ارجاعات همچون اشاره گرهای واقعی استفاده نمایید . بدین ترتیب ، نمی توانید ارجاع شی ئ را بعنوان اشاره ای به موقعیت دلخواه حافظه یا بعنوان یک عدد صحیح بکار برید .

Statement Effect

Box mybox; //(null)

mybox

mybox Width = new Box ()

www.bindownload.com

mybox Height
Depth
Box object

نگاهی دقیقتر به new

شکل عمومی عملگر new بقرار زیر می باشد :

class-var = new classname();

در اینجا class-var یک متغیر از نوع کلاسی است که ایجاد کرده ایم class name . نام کلاسی است که می خواهیم معرفی کنیم . نام کلاس که بعد از آن پرانتزها قرار گرفته اند مشخص کننده سازنده (constructor) کلاس است . سازنده تعریف می کند که وقتی یک شی ئ از یک کلاس ایجاد شود ، چه اتفاقی خواهد افتاد . سازنده ها بخش مهمی از همه کلاسها بوده و خصلتهای بسیار قابل توجهی دارند . بسیاری از کلاسهای دنیای واقعی (real-world) بطور صریحی سازندگان خود را داخل تعریف کلاس ، معرفی می کنند . اما اگر سازنده صریحی مشخص نشده باشد ، جاوا بطور خودکار یک سازنده پیش فرض را عرضه می کند . درست مثل حالت Box . در این مرحله ، ممکن است تعجب کنید که چرا از new برای مواردی نظیر اعداد صحیح وکاراکترها استفاده نمی شود. جواب این است که انواع ساده در جاوا بعنوان اشیائ پیاده سازی نمی شوند . در عوض ، آنها بعنوان متغیرهای عادی پیاده سازی می شوند. اینکار برای افزایش کارایی انجام می گیرد. جاوا قادراست بدون استفاده از رفتارهای خاص نسبت به اشیائ، این انواع ساده را بطور موثری پیاده سازی کند .نکته مهم این است که new حافظه را برای یک شی ئ طی زمان اجرا تخصیص می دهد .

مزیت این روش آن است که برنامه شما میتواند اشیائ مورد نیازش را طی زمان اجرای برنامه ایجاد کند . اما از آنجاییکه محدودیت حافظه وجود دارد ، ممکن است new بعلت عدم کفایت حافظه نتواند حافظه را به یک شی ئ تخصیص دهد . اگر چنین حالتی پیش بیاید، یک استثنائ حین اجرا واقع خواهد شد .ولی در زبانهای C++/C در صورت عدم موفقیت ، مقدار تهی (null) برگردان می شود .

اجازه دهید یکبار دیگر تفاوت بین یک کلاس و یک شی ئ را مرور کنیم . یک کلاس یک نوع جدید داده را ایجاد می کند که می توان برای تولید اشیائ از آن نوع استفاده نمود. یعنی یک کلاس یک چهارچوب منطقی ایجاد می کند که ارتباط بین اعضائ را توصیف می نماید . هنگامیکه یک شی ئ از یک کلاس را اعلان می کنید ، در حقیقت نمونه ای از آن کلاس را بوجود آورده اید. بدین ترتیب ، کلاس یک ساختار منطقی است . یک شی دارای واقعیت فیزیکی است . (یعنی یک شی ئ فضایی از حافظه را اشغال در ذهن داشته باشید .

عملگرها

جاوا یک محیط عملگر غنی را فراهم کرده است . اکثر عملگرهای آن را می توان در چهار گروه طبقه بندی نمود :حسابی arithmetic رفتار بیتی bitwise رابطه ای ralational و منطقی logical جاوا همچنین برخی عملگرهای اضافی برای اداره حالتهای خاص و مشخص تعریف کرده است . نکته : اگر با C++/C آشنایی دارید ، حتما" خوشحال می شوید که بدانید کارکرد عملگرها در جاوا دقیقا"مشابه با C++/C است . اما همچنان تفاوتهای ظریفی وجود دارد .

www.bindownload.com

عملگرهای حسابی Arithmetic operators

عملگرهای حسابی در عبارات ریاضی استفاده می شوند و طریقه استفاده از آنها بهمان روش جبری است . جدول بعدی فهرست عملگرهای حسابی را نشان می دهد :

نتیجه آن عملگر

اضافه نمودن +

تفریق نمودن : همچنین منهای یکانی

ضرب *

تقسیم /

تعیین باقیمانده %

افزایش ++

انتساب اضافه نمودن +=

انتساب تفرق نمودن -=

انتساب ضرب نمودن *=

انتساب تقسیم نمودن /=

انتساب تعیین باقیمانده %=

کاهش - -

عملوندهای مربوط به عملگرهای حسابی باید از نوع عددی باشند . نمی توانید از این عملگرها روی نوع boolean استفاده کنید ، اما روی انواع char قابل استفاده هستند ، زیرا نوع char در جاوا بطور ضروری زیر مجموعه ای از int است .

عملگرهای اصلی حسابی

عملیات اصلی حسابی جمع ، تفریق ، ضرب و تقسیم همانطوریکه انتظار دارید برای انواع عددی رفتار می کنند . عملگر تفرق نمودن همچنین یک شکل یکانی دارد که عملوند تکی خود را منفی (یا خنثی) می کند . بیاد آورید هنگامیکه عملگر تقسیم به یک نوع عدد صحیح اعمال می شود ، هیچ عنصری کسری یا خرده به جواب ملحق نمی شود . برنامه ساده بعدی نشاندهنده عملگرهای حسابی است . این برنامه همچنین تفاوت بین تقسیم اعشاری و تقسیم عدد صحیح را توضیح می دهد .

```
// Demonstrate the basic arithmetic operators.
```

www.bindownload.com

```
class BasicMath {
public static void main(String args[] ){
// arithmetic using integers
System.out.println("Integer Arithmetic");
int a = 1 + 1;
int a = a * 3;
int a = b / 4;
int a = c - a;
int a =- d;
System.out.println("a = " + a);
System.out.println("a = " + b);
System.out.println("a = " + c);
System.out.println("a = " + d);
System.out.println("a = " + e);
// arithmetic using doubles
System.out.println("\nFloating Point Arithmetic");
double da = 1 + 1;
double db = da * 3;
double dc = db / 4;
double dd = dc - a;
double de =- dd;
System.out.println("da = " + da);
System.out.println("db = " + db);
System.out.println("dc = " + dc);
System.out.println("dd = " + dd);
System.out.println("de = " + de);
}
}
```

خروجی این برنامه بقرار زیر می باشد :

```
integer Arithmetic
a=2
b=6
c=1
d=-1
e=1
```

www.bindownload.com

```
floating point arithmetic
da=2
db=6
dc=1.5
dd=-0.5
de=0.5
```

عملگر تعیین باقیمانده The Modulus operator

عملگر تعیین باقیمانده یعنی % ، باقیمانده یک عملیات تقسیم را برمی گرداند .این عملگر برای انواع عدد اعشاری و انواع عدد صحیح قابل استفاده است . (اما در C++/C این عملگر فقط در مورد انواع عدد صحیح کاربرد دارد .) برنامه بعدی نشان دهنده عملگر ٪ می باشد :

```
// Demonstrate the % operator.
class Modulus {
public static void main(String args[] ){
int x = 42;
double y = 42.3;
System.out.println("x mod 10 = " + x % 10);
System.out.println("y mod 10 = " + y % 10);
}
}
```

هنگامیکه این برنامه را اجرا می کنید ، خروجی زیر حاصل می شود :

```
x mod 10=2
y mod 10=2.3
```

عملگرهای انتساب حسابی Arithmetic Assignment operators

جاوا عملگرهای ویژه ای را تدارک دیده که با استفاده از آنها می توان یک عملیات حسابی را با یک انتساب ترکیب نمود . احتمالا" می دانید که دستوراتی نظیر مورد زیر در برنامه نویسی کاملا" رایج هستند :

```
a = a + 4;
```

www.bindownload.com

در جاوا ، می توانید این دستور را بصورت دیگری دوباره نویسی نمایید :

```
a += 4;
```

این روایت جدید از عملگر انتساب += استفاده می کند هر دو دستورات یک عمل واحد را انجام می دهند : آنها مقدارa را4 ا واحد افزایش می دهند . اکنون مثال دیگری را مشاهده نمایید :

```
a = a % 2;
```

که می توان آن را بصورت زیر نوشت :

```
a %= 2;
```

در این حالت ٪= باقیمانده a/2 را گرفته و حاصل را مجددا"درa ر قرار می دهد .عملگرهای انتسابی برای کلیه عملگرهای حسابی و دودوئی (باینری) وجود دارند . بنابراین هر دستور با شکل :

```
Var = var op expression;
```

عبارت عملگر متغیر متغیر را می توان بصورت زیر دوباره نویسی نمود :

```
var op = expression;
```

عبارت عملگر متغیر عملگرهای انتساب دو مزیت را بوجود می آورند . اول اینکه آنها یک بیت از نوع بندی را برای شما صرفه جویی می کنند ، زیر آنها کوتاه شده شکل قبلی هستند .دوم اینکه آنها توسط سیستم حین اجرای جاوا بسیار کاراتر از اشکال طولانی خود پیاده سازی می شوند. بهمین دلایل ، در اکثر برنامه های حرفه ای نوشته شده با جاوا این عملگرهای انتساب را مشاهده می کنید .

در زیر برنامه ای وجود دارد که چندین عملگر انتساب op را نشان می دهد :

```
// Demonstrate several assignment operators.
class OpEquals {
public static void main(String args[] ){
int a = 1;
int b = 2;
int c = 3;
a += 5;
b *= 4;
```

www.bindownload.com

```
c += a * b;
c %= 6;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
}
}
```

خروجی این برنامه بقرار زیر می باشد :

```
a=6
b=8
c=3
```

افزایش و کاهش Increment and Decrement

علامات ++ و -- عملگرهای افزایشی و کاهشی جاوا هستند . این عملگرها را قبلا "معرفی کرده ایم . در اینجا آنها را با دقت بیشتری بررسی می کنیم . همانگونه که خواهید دید ، این عملگرها خصلتهای ویژه ای دارند که بسیار جالب توجه است . بحث درباره این عملگرها را از نحوه کار آنها شروع می کنیم . عملگر افزایشی ، عملوند خود را یک واحد افزایش می دهد . عملگر کاهشی نیز عملوند خود را یک واحد کاهش می دهد . بعنوان مثال ، دستور زیر را

```
x = x + 1;
```

می توان با استفاده از عملگر افزایشی بصورت زیر دوباره نویسی نمود :

```
x++;
```

بطور مشابهی ، دستور زیر را

```
x = x - 1;
```

می توان بصورت زیر باز نویسی نمود :

```
x--;
```

این عملگرها از آن جهت که هم بشکل پسوند جایی که بعد از عملوند قرار می گیرند و هم بشکل پیشوند جایی که قبل از عملوند قرار می گیرند ظاهر می شوند کاملا" منحصر بفرد هستند. در مثالهای بعدی هیچ تفاوتی بین اشکال پسوندی و پیشوندی وجود ندارد . اما

www.bindownload.com

هنگامیکه عملگرهای افزایشی و کاهشی بخشی از یک عبارت بزرگتر هستند ، آنگاه یک تفاوت ظریف و در عین حال پرقدرت بین دو شکل وجود خواهد داشت . در شکل پیشوندی ، عملوند قبل از اینکه مقدار مورد استفاده در عبارت بدست آید ، افزایش یا کاهش می یابد . در شکل پسوندی ، ابتدا مقدار استفاده در عبارت بدست می آید ، و سپس عملوند تغییر می یابد . بعنوان مثال:

```
x = 42;
y = ++x;
```

در این حالت ، همانطوریکه انتظار دارید y معادل 43 می شود ، چون افزایش قبل از اینکهx به y منتسب شود ، اتفاق می افتد . بدین ترتیب خط y=++ معادل دو دستور زیر است :

```
x = x + 1;
y = x;
```

اما وقتی که بصورت زیر نوشته می شوند :

```
x = 42;
y = x++;
```

مقدار x قبل از اینکه عملگر افزایشی اجرا شود ، بدست می آید ، بنابراین مقدار y معادل 42 می شود . البته در هر دو حالت x معادل 43 قرار می گیرد . در در اینجا ، خط y=x++ معادل دو دستور زیر است :

```
y = x;
x = x + 1;
```

برنامه بعدی نشان دهنده عملگر افزایشی است .

```
// Demonstrate ++.
class IncDec {
public static void main(String args[] ){
int a = 1;
int b = 2;
int c;
int d;
c = ++b;
d = a++;
```

www.bindownload.com

```
c++;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d);
}
}
```

خروجی این برنامه بقرار زیر می باشد :

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

عملگرهای رفتار بیتی The Bitwise operators جاوا چندین عملگر رفتار بیتی تعریف نموده که قابل اعمال روی انواع عدد صحیح شامل long ، int، short، char،وbyte و می باشند . این عملگرها روی بیت های تکی عملوندهای خود عمل می کنند .

این عملگرها را در جدول زیر خلاصه نموده ایم :

نتیجه آن عملگر

Bitwise unary Not

Notیکانی رفتار بیتی Bitwise AND ~

ANDرفتار بیتی Bitwise OR &

ORرفتار بیتی Bitwise exclusive OR |

ORخارج رفتار بیتی shift right ^

حرکت براست shift right zero fill >>

حرکت براست پر شده با صفر shift left >>>

حرکت به چپ Bitwise AND assignment <<

انتساب AND رفتار بیتی Bitwise OR assignment &=

انتساب OR رفتار بیتی Bitwise exclusive OR assignment |=

انتساب OR خارج رفتار بیتی shift right assignment ^=

www.bindownload.com

انتساب حرکت راست>>shift right zero fill assignment =

انتساب حرکت براست پر شده با صفر>>>shift left assignment =

انتساب حرکت به چپ<<=

از آنجاییکه عملگرهای رفتار بیتی با بیت های داخل یک عدد صحیح سر و کار دارند ، بسیار مهم است بدانیم که این سر و کار داشتن چه تاثیری ممکن است روی یک مقدار داشته باشد . بخصوص بسیار سودمند است بدانیم که جاوا چگونه مقادیر عدد صحیح را ذخیره نموده و چگونه اعداد منفی را معرفی می کند . بنابراین ، قبل از ادامه بحث ، بهتر است این دو موضوع را باختصار بررسی نماییم . کلیه انواع صحیح بوسیله ارقام دودوئی (باینری) دارای پهنای بیتی گوناگون معرفی میشوند. بعنوان مثال ، مقدار byte عدد 42 در سیستم باینری معادل 00101010 است ، که هر یک از این نشانه ها یک توان دو را نشان می دهند که با 2 به توان 0 در بیت سمت راست شروع شده است . یا موقعیت بعدی بیت بطرف چپ 2'یا 2ا است و به طرف چپ بیت بعدی 2 به توان 2یا4 ا است ، بعدی 8 ، 16 ، 32 و همینطور الی آخر هستند . بنابراین عدد 42 بیت 1 را در موقعیتهای اول ، سوم و پنجم (از سمت راست درنظربگیرید) دارد. بدین ترتیب 42 معادل جمع 5 بتوان 3+2 بتوان 1+2 بتوان 2 یعنی 2+8+32 می باشد . کلیه انواع عدد صحیح (باستثنائ char) اعداد صحیح علامت دار هستند . یعنی که این انواع مقادیر منفی را همچون مقادیر مثبت می توانند معرفی کنند . جاوا از یک روش رمزبندی موسوم به مکمل دو (two's complement) استفاده می کند که در آن ارقام منفی با تبدیل (تغییر 1به 0 و بالعکس) کلیه بیت های یک مقدار و سپس اضافه نمودن 1 به آن معرفی می شوند . بعنوان مثال برای معرفی 42 ، ابتدا کلیه بیت های عدد 42 (00101010) را تبدیل می نماییم که 11010101 حاصل می شود آنگاه 1 را به آن اضافه می کنیم . که حاصل نهایی یعنی 11010110 معرف عدد 42 خواهد بود . برای رمز گشایی یک عدد منفی ، کافی است ابتدا کلیه بیت های آن را تبدیل نموده ، آنگاه 1 را به آن اضافه نماییم . 42- یعنی 11010110 پس از تبدیل برابر 00101001 یا 41 شده و پس از اضافه نمودن 1 به آن برابر 42 خواهد شد . دلیل اینکه جاوا)واکثر زبانهای برنامه نویسی)از روش مکمل دو (two's complement)استفاده می کنند ، مسئله تقاطع صفرها (Zero crossing) است . فرض کنید یک مقدارbyte برای صفر با 00000000 معرفی شده باشد. در روش مکمل یک (one's complement) تبدیل ساده کلیه بیت ها منجر به 11111111 شده که صفر منفی را تولید می کند

.

اما مشکل این است که صفر منفی در ریاضیات عدد صحیح غیر معتبر است . این مشکل بااستفاده از روش مکمل دو (two's complement)برای معرفی مقادیر منفی حل خواهد شد . هنگام استفاده از روش مکمل دو ، 1 به مکمل اضافه شده و عدد 100000000 تولید می شود . این روش بیت 1 را در منتهی الیه سمت چپ مقدار byte قرار داده تا رفتار مورد نظر انجام گیرد، جایی که 0با 0ا یکسان بوده و 11111111 رمزبندی شده 1 است . اگر چه در این مثال از یک مقدار byte استفاده کردیم ، اما

www.bindownload.com

همین اصول برای کلیه انواع عدد صحیح جاوا صدق می کنند . از آنجاییکه جاوا از روش مکمل دو برای ذخیره سازی ارقام منفی استفاده میکند و چون کلیه اعداد صحیح در جاوا مقادیر علامت دار هستند بکار بردن عملگرهای رفتار بیتی براحتی نتایج غیر منتظره ای تولید می کند . بعنوان مثال برگرداندن بیت بالاتر از حد مجاز (high-order) سبب می شود تا مقدار حاصله بعنوان یک رقم منفی تفسیر شود ، خواه چنین قصدی داشته باشید یا نداشته باشید . برای جلوگیری از موارد ناخواسته ، فقط بیاد آورید که بیت بالاتر از حد مجاز (high-order)

علامت یک عدد صحیح را تعیین می کند، صرفنظر از اینکه بیت فوق چگونه مقدار گرفته باشد .

عملگرهای منطقی رفتار بیتی

عملگرهای منطقی رفتار بیتی شامل& ، |،، ^،، ~، هستند. جدول زیر حاصل هر یک از این عملیات را نشان می دهد . در بحث بعدی بیاد داشته باشید که عملگرهای رفتار بیتی به بیت های منفرد داخل هر عملوند اعمال می شوند .

A B A|B A&B A^B ~A
0 0 0 0 0 1
1 0 1 0 1 0
0 1 1 0 1 1
1 1 1 1 0 0

NOT

رفتار بیتی

عملگر NOT یکانی یعنی ~ که آن را مکمل رفتار بیتی (bitwise complement) هم می نامند ، کلیه بیت های عملوند خود را تبدیل می کند . بعنوان مثال ، عدد 42 که مطابق الگوی بیتی زیر است : 00101010

پس از اعمال عملگر NOT بصورت زیر تبدیل می شود11010101 :

رفتار بیتیAND

عملگر AND یعنی & اگر هر دو عملوند 1 باشند ، یک بیت 1 تولید می کند . در کلیه شرایط دیگر یک صفر تولید می شود . مثال زیر را نگاه کنید : 42 00101010

& 00001111 15
00001010 10

www.bindownload.com

رفتار بیتیOR

عملگر OR یعنی | بیت ها را بگونه ای ترکیب می کند که اگر هر یک از بیت های عملوندها 1 باشد ، آنگاه بیت حاصله نیز 1 خواهد بود . به مثال زیر نگاه کنید : 42 00102010

00001111 15

00101111 47

رفتار بیتیXOR

عملگر XOR یعنی ^ بیت ها را بگونه ای ترکیب می کند که اگر دقیقا" یک عملوند 1باشد ، حاصل برابر 1 خواهد شد . در غیر اینصورت ، حاصل 0 می شود . مثال بعدی چگونگی کار این عملگر را نشان می دهد . این مثال همچنین یکی از خصلتهای سودمند عملگر XOR را نمایش می دهد . دقت کنید که هر جا عملوند دوم یک بیت 1 داشته باشد ، چگونه الگوی بیتی عدد 42 تبدیل می شود . هر جا که عملوند دوم بیت 0 داشته باشد ، عملوند اول بدون تغییر می ماند . هنگام انجام برخی از انواع عملکردهای بیتی ، این خصلت بسیار سودمند است . 4200101010

^ 00001111 15

00100101 37

استفاده از عملگرهای منطقی رفتار بیتی

برنامه بعدی نشان دهنده عملگرهای منطقی رفتار بیتی است :

```
// Demonstrate the bitwise logical operators.
class BitLogic {
public static void main(String args[] ){
String binary[] = {
"0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111",
"1000", "1001", "1010", "1011", "1100", "1101", "1110", "1111"
};
int a = 3; // 0 + 2 + 1 or 0011 in binary
int b = 6; // 4 + 2 + 0 or 0110 in binary
int c = a | b;
int d = a & b;
int e = a ^ b;
int f =( ~a & b )|( a & ~b);
```

www.bindownload.com

```
int g = ~a & 0x0f;

System.out.println(" a = " + binary[a]);
System.out.println(" b = " + binary[b]);
System.out.println(" a|b = " + binary[c]);
System.out.println(" a&b = " + binary[d]);
System.out.println(" a^b = " + binary[e]);
System.out.println("~a&b|a&~b = " + binary[f]);
System.out.println(" ~a = " + binary[g]);
}
}
```

در این مثال ، aوb و الگوهای بیتی دارند که کلیه چهار احتمال برای ارقام دو تایی باینری را معرفی می کنند . 0-0، 0-1، 1-0،و 1-1و . می توانید مشاهده کنید چگونه |و& و روی هر یک از بیت ها با توجه به نتایج درc وd و عمل می کنند .مقادیر نسبت داده شده بهe وf و مشابه بوده و نشان دهنده چگونگی کار عملگر ^ می باشند . آرایه رشته ای با نام binary معرفی ارقام 0 تا 15 را بصورت باینری و قابل خواندن برای انسان نگهداری می کند . در این مثال ، آرایه فوق طوری نمایه سازی شده تا معرفی باینری هر یک از نتایج را نشان دهد . آرایه طوری ساخته شده که معرفی رشته ای صحیح یک مقدار باینری n را در binary[n] ذخیره می کند . مقدارa~ بوسیله عملگرAND باoxof ا(00001111 باینری) عمل شده تا مقدار آن را به کمتر از 16 کاهش دهد تا بتوان با استفاده از آرایه binary از آن چاپ گرفت . اکنون خروجی این برنامه بصورت زیر می باشد :

```
a=1011
b=0110
a^Eb=0111
a&b=0010
a&b=0101
~a&b^Ea&~b=0101
~a=1100
```

حرکت به چپ

کلیه بیت های موجود در یک مقدار را به تعداد<< عملگر حرکت به چپ یعنی دفعات مشخص بطرف چپ منتقل می کند . شکل کلی آن بقرار زیر است :

```
Value << num
```

www.bindownload.com

تعداد دفعات مقدار

در ایجا num مشخص کننده تعداد مکانهایی است که بیت های موجود در value باید کلیه بیت های موجود در یک مقدار مشخص را<< به چپ انتقال یابند . بدین ترتیب بتعداد مکانهایی که در num مشخص شده بطرف چپ حرکت می دهد . برای هر بار حرکت به چپ ، بیت (high-order بیش از حد مجاز) منتقل شده و از دست خواهد رفت و یک صفر در طرف راست مقدار ، جایگزین می شود . بدین ترتیب هنگامیکه یک حرکت به چپ روی یک عملوند int عمل می کند ، بیت های گذشته از مکان 31 از دست خواهند رفت . اگر عملوند یک long باشد، بیت ها پس از گذشتن از مکان 63 از دست میروند .هنگامیکه مقادیر byte و short و را انتقال می دهید ، ارتقائ خودکار انواع در جاوا نتایج غیر منتظره ای ایجاد می کند . حتما" می دانید که هنگام ارزشیابی عبارات ، مقادیر byte و short وبه int ارتقائ می یابند. بعلاوه جواب چنین عبارتی از نوع int خواهد بود .بنابراین حاصل یک حرکت به چپ روی مقادیر byte و short و یک int خواهد بود و بیت های انتقال یافته به چپ تا زمانیکه از مکان بیت 31 نگذرند ، از دست نمی روند . علاوه براین ، یک مقدار منفی byte و short و هنگامیکه به int ارتقائ می یابد ، بسط علامت پیدا می کند . بنابراین بیت های بیش از حد مجاز با بیت 1 پر می شوند . بخاطر این دلایل ، انجام یک حرکت به چپ روی byte و short مستلزم آن است که از بایت های بیش از حد مجاز در جواب int دست بکشید . بعنوان مثال ، اگر یک مقدار byte را حرکت به چپ بدهید ، آن مقدار ابتدا به نوع int تبدیل شده و سپس انتقال خواهد یافت . باید سه بایت بالایی حاصل را از دست بدهید . اگر بخواهید حاصل یک مقدار byte انتقال یافته را بدست آورید . باید سه بایت بالایی حاصل را از دست بدهید . آسان ترین روش برای انجام اینکار استفاده از تبدیل cast و تبدیل جواب به نوع byte است .

مثال بعدی همین مفهوم را برای شما آشکار می سازد :

```
// Left shifting a byte value.
class ByteShift {
public static void main(String args[] ){
byte a = 64/ b;
int i;

i = a << 2;
b =( byte( )a << 2);

System.out.println("Original value of a :" + a);
System.out.println("i and b :" + i + " " + b);
}
}
```

www.bindownload.com

خروجی تولید شده توسط این برنامه بقرار زیر می باشد :

```
original value of a:64
i and b :256 0
```

چون برای اهداف ارزشیابی ، a به نوع int ارتقائ یافته ، دوبار حرکت به چپ مقدار 64 (0100´0000) منجر به i می گردد که شامل مقدار 256 (0000 1´0000) می باشد . اما مقدار b دربرگیرنده صفر است زیرا پس از انتقال ، بایت کمتر از حد مجاز (loworder) اکنون شامل صفر است . تنها بیت دربرگیرنده 1 به بیرون انتقال یافته است .

از آنجاییکه هر بار حرکت به چپ تاثیر دو برابر سازی مقدار اصلی را دارد برنامه نویسان اغلب از این خاصیت بجای دو برابر کردن استفاده می کنند . اما باید مراقب باشید . اگر یک بیت 1 را به مکان بیت بیش از حد مجاز (31 یا 63) منتقل کنید ، مقدار فوق منفی خواهد شد . برنامه بعدی همین نکته را نشان میدهد .

```
// Left shifting as a quick way to multiply by 2.
class MultByTwo {
public static void main(String args[] ){
int i;
int num = 0xFFFFFFE;

for(i=0; i<4; i++ ){
num = num << 1;
System.out.println(num);
}
}
}
```

خروجی این برنامه بقرار زیر خواهد بود :

```
536870908

1073741816
2147483632
- 32
```

مقدار آغازین را با دقت انتخاب کرده ایم بطوریکه بیت بعد از چهار مکان حرکت بطرف چپ ، مقدار 32 -را تولید نماید .

همانطوریکه می بینید ، هنگامیکه بیت 1 به بیت 31 منتقل می شود ، رقم بعنوان منفی تفسیر خواهد شد .

www.bindownload.com

حرکت به راست

کلیه بیت های موجود در یک مقدار را به تعداد<< عملگر حرکت به راست یعنی دفعات مشخص بطرف راست انتقال می دهد . شکل کلی آن بقرار زیر می باشد :

```
value >> num
```

تعداد دفعات مقدار

در اینجا ، numمشخص کننده تعداد مکانهایی است که بیت های value باید بطرف کلیه بیت های یک مقدار مشخص شده را به تعداد<< راست انتقال یابند . یعنی مکانهای بیتی مشخص شده توسط num بطرف راست انتقال می دهد . کد قطعه ای زیر مقدار 32 را دو مکان بطرف راست منتقل می کند و آنگاه جواب آن در a معادل 8 قرار می گیرد :

```
int a = 32;
a = a >> 2; // a now contains 8
```

اگر بیت هایی از یک مقدار به بیرون منتقل شوند ، آن بیت ها از دست خواهند رفت . بعنوان مثال کد قطعه ای بعدی مقدار 35 را دو مکان بطرف راست منتقل نموده و باعث می شود تا دو بیت کمتر از حد مجاز از دست رفته و مجددا "جواب آن در a معادل 8 قرار گیرد .

```
int a = 35;
a = a >> 2; // a still contains 8
```

همین عملیات را در شکل باینری نگاه می کنیم تا اتفاقی که می افتد ، روشن تر شود : 35 00100011

>> 2

00001000 8

هر بار که یک مقدار را به طرف راست منتقل می کنید ، آن مقدار تقسیم بر دو می شود و باقیمانده آن از دست خواهد رفت . می توانید از مزایای این روش در تقسیم بر دو اعداد صحیح با عملکرد سطح بالا استفاده نمایید . البته ، باید مطمئن شوید که بیت های انتهایی سمت راست را به بیرون منتقل نکنید . هنگامیکه حرکت بطرف راست را انجام می دهید ، بیت های بالایی (از سمت چپ) در معرض حرکت بطرف راست قرار گرفته ، با محتوی قبلی بیت بالایی پر می شوند . این حالت را بسط علامت (sign extension) نامیده و برای محفوظ نگهداشتن علامت ارقام منفی هنگام حرکت بطرف راست استفاده می شوند. بعنوان مثال 1 -8<<معادل 4 -است که به شکل باینری زیر می باشد : 8 -11111000

>> 111111100- 4

www.bindownload.com

جالب است بدانید که اگر 1- را بطرف راست حرکت دهید، حاصل آن همواره 1- باقی می ماند ، چون بسط علامت ، مراقب آوردن یک بیت دیگر در بیت های بیش از حد مجاز خواهد بود . گاهی هنگام حرکت بطرف راست مقادیر ، مایل نیستیم تا بسط علامت اجرا شود . بعنوان مثال ، برنامه بعدی یک مقدار نوع byte را به معرفی رشته ای در مبنای 16 تبدیل می کند . دقت کنید که مقدار منتقل شده با استفاده از عملگر AND یا oxof ا پوشانده شده تا هر گونه بیت های بسط یافته علامت را بدور اندازد بطوریکه مقدار فوق را بتوان بعنوان یک نمایه به آرایه ای از کاراکترهای در مبنای 16 استفاده نمود .

```
// Masking sign extension.
class HexByte {
static public void main(String args[] ){
char hex[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };
byte b =( byte )oxf1
System.out.println("b = ox" + hex[(b >> 4 )& oxof] + hex[b & oxof]);
}
}
```

خروجی این برنامه بقرار زیر می باشد :

```
b=oxf1
```

حرکت به راست فاقد علامت بطور خودکار << اکنون می دانید که هر بار یک انتقال اتفاق می افتد ، عملگر جای خالی بیت بیش از حد مجاز را با محتوی قبلی اش پر می کند . این عمل سبب حفظ علامت آن مقدار می گردد . اما گاهی تمایلی برای اینکار نداریم .

بعنوان مثال

می خواهید چیزی را منتقل کنید که معرف یک مقدار عددی نیست . بالطبع نمی خواهید عمل بسط علامت انجام گیرد . این حالت هنگام کار با مقادیر براساس پیکسل (pixel) و گرافیک اغلب وجود دارد . در چنین شرایطی لازم است تا مقدار صفر در بیت بیش از حد مجاز قرار گیرد ، صرفنظر از اینکه مقدار قبلی در آن بیت چه بوده است . این حالت را انتقال فاقد علامت (usigned shift) می گویند. برای این منظور، از عملگر استفاده کنید که صفرها را در بیت بیش <<< حرکت به راست فاقد علامت در جاوا یعنی از حد مجاز منتقل می کند .

می باشد . در اینجا <<< کد قطعه ای زیر نشان دهنده عملگر a معادل 1- است که کلیه 32 بیت را در باینری روی 1 تنظیم می

www.bindownload.com

کند . این مقدار سپس 24 بیت بطرف راست انتقال می یابد، و 24 بیت بالایی را با صفرها پر می کند و بسط علامت معمولی را نادیده می گیرد . بدین ترتیب a معادل 255 می باشد int a =- 1; .

a = a >>> 24;

اینجا همان عملیات را در شکل باینری مشاهده می کنید تا بهتر بفهمید چه تفاقی افتاده است : 1

در باینری بعنوان یک int 11111111 11111111 11111111 11111111 >>>24

255

در باینری بعنوان یک int´11111111 00000000 00000000 00000000 اغلب اوقات آنچنان سودمند که بنظر می رسد ، نبوده چون فقط برای >>> عملگر مقادیر 32 بیتی و 64 بیتی معنی دارد . بیاد آورید که مقادیر کوچکتر در عبارات بطور خودکار به int ارتقائ می یابند . بدین ترتیب بسط علامت اتفاق افتاده و حرکت بجای مقادیر 8 بیتی و 16 بیتی روی مقادیر 32 بیتی انجام می شود . یعنی باید انتظار یک حرکت به راست فاقد علامت روی یک مقدار byte داشته باشیم که در بیت 7 ، صفر را قرار می دهد . اما واقعا" اینطور نیست ، چون در واقع مقدار 32 بیتی است که منتقل می شود . برنامه بعدی این تاثیری را نشان می دهد .

```
// Unsigned shifting a byte value.
class ByteUShift {
static public void main(String args[] ){
char hex[] = {
'0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', 'a', 'b', 'c', 'd', 'e', 'f'
};
byte b =( byte )oxf1
byte c =( byte( )b >> 4);
byte d =( byte( )b >>> 4);
byte e =( byte( )b & oxff )>> 4);

System.out.println(" b = ox"+ hex[(b >> 4 )& oxof] + hex[b & oxof]);
System.out.println(" b >> 4 = ox" + hex[(c >> 4 )& oxof] + hex[c & oxof]);
System.out.println(" b >>> 4 = ox" + hex[(d >> 4 )& oxof] + hex[d & oxof]);
System.out.println("(b & oxof )>> 4 = ox" + hex[(e >> 4 )& oxof] + hex[e & oxof]);
}
}
```

چگونه هنگام کار با بایت ها عملی >>> خروجی این برنامه نشان میدهد که عملگر انجام نمی دهد . متغیر b بعنوان یک مقدار byteمنفی قراردادی در این نمایش تنظیم شده است . سپس مقدار byte در b ر که چهار مکان بطرف راست انتقال بافته به C

www.bindownload.com

منتسب می شود که بخاطر بسط علامت مورد انتظار oxff است . سپس مقدارbyte درb ر که چهار مکان بطرف راست و فاقد علامت منتقل شده به d منتسب می شود که انتظار دارید oxof باشد ، اما در حقیقت oxff است چون بسط علامت هنگامیکه b به نوع int قبل از انتقال ارتقائ یافته اتفاق افتاده است . آخرین عبارت ، eرا در مقدارbyte متغیر b که با استفاده از عملگرAND با 8ا بیت پوشانده شده تنظیم نموده و سپس چهار مکان بطرف راست منتقل می کند که مقدار مورد انتظار oxof را تولید می کند . دقت کنید که عملگر حرکت به راست فاقد علامت برای d استفاده نشد ، چون حالت بیت علامت بعد از AND شناخته شده است .

```
b=oxf1
b>>4=oxff
b>>>4=oxff
( b&oxff)>>4=oxof
```

انتسابهای عملگر رفتار بیتی

کلیه عملگرهای رفتار بیتی باینری یک شکل مختصر مشابه باعملگرهای جبری دارند که عمل انتساب را با عملیات رفتار بیتی ترکیب می کنند. بعنوان مثال ، دو دستور بعدی که مقدار a را چهار بیت به راست حرکت می دهند ، معادل یکدیگرند :

```
a = a >> 4;
a >>= 4;
```

بطور مشابه ، دو دستور زیر که a را به عبارت روش بیتی aoRb منتسب می کنند معادل یکدیگرند :

```
a = a | b;
a |= b;
```

برنامه بعدی تعدادی از متغیرهای عدد صحیح را بوجود آورده آنگاه از شکل مختصر انتسابهای عملگر رفتار بیتی برای کار کردن بااین متغیرها استفاده میکند :

```
class OpBitEquals {
 public static void main(String args[] ){
 int a = 1;
 int b = 2;
```

www.bindownload.com

عملگر انتساب The Assignment Operator

عملگر انتساب علامت تکی تساوی = می باشد . عملگر انتساب در جاوا مشابه سایر زبانهای برنامه نویسی کار می کند . شکل کلی آن بصورت زیر است :

```
Var = expression;
```

عبارت متغیر

در اینجا نوع (var متغیر) باید با نوع (experssion عبارت) سازگار باشد .عملگر انتساب یک خصلت جالب دارد که ممکن است با آن آشنایی نداشته باشید : به شما امکان می دهد تا زنجیره ای از انتسابها بوجود آورید . بعنوان مثال ، این قطعه از یک برنامه را در نظر بگیرید :

```
int x, y, z;
x = y = z = 100; // set x, y, and z to 100
```

این قطعه از برنامه مقدار 100 را با استفاده از یک دستور در متغیرهای x ، y، و z قرار می دهد . زیرا = عملگری است که مقدار عبارت سمت راست را جذب می کند . بنابراین مقدار Z=100 برابر 100 است که این مقدار به y منتسب شده و نیز به x منتسب خواهد شد . استفاده از " زنجیره ای از انتسابها " یک راه آسان برای قرار ادن یک مقدار مشترک در گروهی از متغیرهاست .

ارتقائ خودکار انواع در عبارات Automatic Type promotion in Expressions

علاوه بر انتسابها ، در شرایط دیگری هم تبدیلات خاص انواع ممکن است اتفاق بیفتد : در عبارات . حالتی را در نظر بگیرید که در یک عبارت ، میزان دقت لازم برای یک مقدار واسطه گاهی از دامنه هر یک از عملوندهایخود تجاوز می نماید .

بعنوان مثال ، عبارت زیر را در نظر بگیرید :

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;
```

نتیجه قلم واسطه a*b از دامنه هر یک از عملوندهای byte خود تجاوز می نماید .برای اداره این نوع مشکلات ، جاوا بطور خودکار هر یک از عملوندهای byte و short و را هنگام ارزشیابی یک عبارت به int ارتقائ می دهد . این بدان معنی است که زیر عبارت

www.bindownload.com

a*bبا استفاده از اعداد صحیح و نه byte اجرا می شود. بنابراین عدد 2000 نتیجه عبارت واسطه 50*40 مجاز است ، اگر چهa و bو هر دو بعنوان نوع byte مشخص شده اند .

همانقدر که ارتقائ خودکار مفیداست ، می تواند سبب بروز خطاهای زمان کامپایل (compile-time) گردد . بعنوان مثال ، این کد بظاهر صحیح یک مشکل را بوجود می آورد .

```
byte b = 50;
b = b * 2; // Error! Cannot assign an int to a byte!
```

این کد تلاش می کند تا 50*2 را که یک مقدار کاملا" معتبر byte است به یک متغیر byte ذخیره کند . اما چون عملوندها بطور خودکار هنگام ارزشیابی عبارت بهint ارتقائ یافته اند ، جواب حاصله نیز به int ارتقائ یافته است . بنابراین جواب عبارت اکنون از نوع int است که بدون استفاده از تبدیل cast امکان نسبت دادن آن به یک byte وجود ندارد. این قضیه صادق است ، درست مثل همین حالت ، حتی اگر مقدار نسبت داده شده همچنان با نوع هدف سازگاری داشته باشد . در شرایطی که پیامدهای سر ریز (overflow) را درک می کنید، باید از یک تبدیل صریح cast نظیر مورد زیر استفاده نمایید .

```
byte b = 50;
b =( byte( )b * 2);
```

که مقدار صحیح عدد 100 را بدست می آورد .

قوانین ارتقائ انواع

علاوه بر ارتقائbyteوshort وبه int جاوا چندین قانون ارتقائانواع را تعریف کرده که قابل استفاده در عبارات می باشند . این قوانین بصورت زیر هستند . اول اینکه کلیه مقادیرbyte وshort وبه int ارتقائ می یابند ، همانگونه که قبلا "توضیح داده ایم . آنگاه اگر یک عملوند ، longباشد ، کل عبارت به long ارتقائ می یابد . اگر یک عملوند float باشد ، کل عبارت به float ارتقائ می یابد . اگر هر یک از عملوندها یک double باشند ، حاصل آنها double خواهد شد . برنامه بعدی نشان می دهد که چگونه هر یک از مقادیر در عبارت ارتقائ می یابد تا با آرگومان دوم به هر یک از عملگرهای دودویی ، مطابقت یابد .

```
class Promote {
public static void main(String args[] ){
byte b = 42;
char c = 'a';
short s = 1024;
int i = 50000;
```

www.bindownload.com

```
float f = 5.67f;
double d =. 1234;
double result =( f * b )+( i / c( - )d * s);
System.out.println((f * b )+ " + " +( i / c )+ " - " +( d * s));
System.out.println("result = " + result);
}
}
```

اجازه دهید به ارتقائ انواع که در این خط از برنامه اتفاق افتاده ، دقیقتر

نگاه کنیم :

```
double result =( f * b )+( i / c(  )d * s);
```

در اولین زیر عبارت یعنیf*b ، b، به یک نوع float ارتقائ یافته و جواب زیر عبارت نیز از نوع float خواهد بود . در زیر عبارت بعدی یعنیi/c ، c، به یک نوعint ارتقائ یافته و جواب آن زیر عبارت نیز از نوع int خواهد بود . سپس در زیر عبارت d*s ، مقدار Sبه نوع double ارتقائ یافته و نوع زیر عبارت نیز double خواهد بود . در نهایت این سه مقدار واسطه ، float، int، double، در نظر گرفته می شوند . خروجی float بعلاوه int از نوع float خواهد شد . آنگاه این نتیجه منهای آخرین double به نوع double ارتقائ یافته ، که نوع مربوط به جواب نهایی

استفاده از بلوکهای کد Blocks of code

جاوا این امکان را فراهم نموده تا دو یا چند دستور در بلوکهای کد گرد آوری شوند که آنها را معمولا " code blocksمی نامند . اینکار با محصور کردن دستورات بین ابروهای باز و بسته انجام می گیرد .یکبار که یک بلوک کد ایجاد می شود، این بلوک که تبدیل به یک واحد منطقی شده و هر جایی که یک دستور ساده بتوان استفاده نمود، مورد استفاده قرار می گیرد .بعنوان مثال ، یک بلوک ممکن است هدف دستوراتif و یا for جاوا باشد . دستور if زیر را در نظر بگیرید :

```
if(x < y ){ // begin a block
x = y;
y = 0;
} // end of block
```

در اینجا اگر x کوچکتر از y باشد ، آنگاه هر دو دستور موجود در داخل بلوک اجرا خواهند شد . بنابراین دو دستور داخل بلوک تشکیل یک واحد منطقی داده اند و آنگاه اجرای یک دستور منوط به اجرای دستور دیگر خواهد بود . نکته کلیدی در اینجا این است

www.bindownload.com

که هر گاه لازم باشد دو یا چند دستور را بطور منطقی پیوند دهید توسط ایجاد یک بلوک اینکار را انجام می دهید . به یک مثال دیگر نگاه کنید. برنامه بعدی از یک بلوک کد بعنوان هدف (target) یک حلقه for استفاده می کند .

```
/*
Demonstrate a block of code.
Call this file "BlockTest.java"
*/
class BlockTest {
public static void main(String args[] ){
int x/ y;
y = 20;
// the target of this loop is a block
for(x = 0; x<10; x++ ){
System.out.println("This is x :" + x);
System.out.println("This is y :" + y);
y = y - 2;
}
}
}
```

خروجی این برنامه بقرار زیر می باشد :

```
This is x:0
This is y:20
This is x:1
This is y:18
This is x:2
This is y:16
This is x:3
This is y:14
This is x:4
This is y:12
This is x:5
This is y:10
This is x:6
This is y:8
This is x:7
```

www.bindownload.com

```
This is y:6
This is x:8
This is y:4
This is x:9
This is y:2
```

در این حالت ، هدف حلقه for یک بلوک کد است نه یک دستور منفرد. بدین ترتیب هر بار که حلقه تکرار می شود ، سه دستور داخل بلوک اجرا خواهد شد . این حقیقت در خروجی تولید شده توسط برنامه کاملا" هویداست . همانگونه که بعدا" خواهید دید، بلوکهای کد دارای ویژگیها و کاربردهای دیگری هم هستند . اما دلیل اصلی حضور آنها ایجاد واحدهای منطقی و تفکیک ناپذیر از باشد.

استفاده از پرانتزها

پرانتزها حق تقدم عملیاتی را که دربرگرفته اند ، افزایش می دهند . اینکار اغلب برای نگهداری نتیجه دلخواهتان ضروری است . بعنوان مثال ، عبارت زیر را در نظر بگیرید :

```
a >> b + 3
```

این عبارت ابتدا 3 را به b اضافه نموده و سپس a را مطابق آن نتیجه بطرف راست حرکت می دهد. این عبارت را می توان با استفاده از پرانتزهای اضافی بصورت زیر دوباره نویسی نمود :

```
a >>( b + 3)
```

اما ، اگر بخواهید ابتدا a را با مکانهای b بطرف راست حرکت داده و سپس 3 را به نتیجه آن اضافه کنید ، باید عبارت را بصورت زیر در پرانتز قرار دهید3 +(a >> b) :

علاوه بر تغییر حق تقدم عادی یک عملگر ، پرانتزها را می توان گاهی برای روشن نمودن مفهوم یک عبارت نیز بکار برد . برای هر کسی که کد شما را می خواند، درک یک عبارت پیچیده بسیار مشکل است . اضافه نمودن پرانتزهای اضافی و روشنگر به عبارات پیچیده می تواند از ابهامات بعدی جلوگیری نماید. بعنوان مثال ، کدامیک از عبارات زیر راحت تر خوانده و درک می شوند ?

```
a | 4 + c >> b & 7 || b > a % 3
( a |((( 4 + c )>> b )& 7 ))||( b >( a % 3))
```

یک نکته دیگر : پرانتزها (بطور کلی خواه اضافی باشند یا نه) سطح عملکرد برنامه شما را کاهش نمی دهند. بنابراین ، اضافه کردن پرانتزها برای کاهش ابهام نفی روی برنامه شما نخواهد داشت.

عملگر ?

www.bindownload.com

جاوا شامل یک عملگر سه تایی ویژه است که می تواند جایگزین انواع مشخصی از دستورات if-then-else باشد. این عملگر علامت ? است و نحوه کار آن در جاوا مشابه باC و++C و است . ابتدا کمی گیج کننده است ، اما می توان از ? براحتی و با کارایی استفاده نمود شکل کلی این عملگر بصورت زیر است :

experssion 1? experssion2 :experssion3

در اینجا experssion1 می تواند هر عبارتی باشد که با یک مقدار بولی سنجیده می شود . اگر experssion1 صحیح true باشد ، آنگاه experssion2 سنجیده می شود در غیر اینصورت experssion3 ارزیابی خواهد شد . نتیجه عملیات ? همان عبارت ارزیابی شده است . هر دو عبارتexperssion2 وexperssion3 و باید از یک نوع باشند که البته void نمی تواند باشد . در اینجا مثالی برای استفاده از عملگر ? مشاهده می کنید :

ratio = denom == 0 ? 0 : num / denom;

هنگامیکه جاوا این عبارت انتساب را ارزیابی می کند ، ابتدا به عبارتی که سمت چپ علامت سئوال قرار دارد ، نگاه می کند. اگر denomمساوی صفر باشد ، آنگاه عبارت بین علامت سئوال و علامت (colon) ارزیابی شده و بعنوان مقدار کل عبارت ? استفاده می شود . اگر denom مساوی صفر نباشد ، آنگاه عبارت بعد از (colon) ارزیابی شده و برای مقدار کل عبارت ? استفاده می شود . نتیجه تولید شده توسط عملگر ? سپس به ratio نسبت داده می شود .

در زیر برنامه ای مشاهده می کنید که عملگر ? را نشان می دهد . این برنامه از عملگر فوق برای نگهداری مقدار مطلق یک متغیر استفاده می کند .

```
// Demonstrate ?.
class Ternary {
public static void main(String args[] ){
int i/ k;

i = 10;
k = i < 0 ?- i : i; // get absolute value of i
System.out.print("Absolute value of ");
System.out.println(i + " is " + k);
}
}
```

خروجی این برنامه بصورت زیر می باشد :

```
Absolute value of 10 is 10
```

www.bindownload.com

```
Absolute value of- 10 is 10
```

حق تقدم عملگر

جدول زیر ترتیب حق تقدم عملگرهای جاوا را از بالاترین اولویت تا پایین ترین نشان می دهد . دقت کنید که در سطر اول اقلامی وجود دارد که معمولا" بعنوان عملگر درباره آنها فکر نمی کنید : پرانتزها ، کروشه ها و عملگر نقطه .

1. Highest
2. () [].
3. ++ -- ~ !
4. / %
5. +-
6. >> >>> <<
7. >= < <=
8. == !=
9. &
10. ^
11. |
12. &&
13. ||
14. ?:
15. = op=
16. Lowest

www.bindownload.com

از پرانتزها برای تغییر حق تقدم یک عملیات استفاده می شود . قبلا" خوانده اید که کروشه های دوتایی نمایه سازی آرایه ها را فراهم می سازند . عملگرهای نقطه یاناستفاده شده که بعدا" مورد بررسی قرار خواهیم داد .

منابع :

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

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

کتاب :

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