عملگر های پایه ای زبان جاوا

هرجایی که متغیر ها وجود داشته باشند، نیاز به ابزاری داریم تا بتوانیم با آنها کار کنیم و عمل های مختلفی را برای تغییر دادن و کنترل آنها انجام بدهیم. به این ابزار ها Operator یا عملگر گفته میشود. در زبان برنامه نویسی جاوا هم متغیر های مختلفی وجود دارند که در جلسه های قبلی دوره آموزشی جاوا با آنها آشنا شدیم. در این جلسه میخواهیم درباره عملگر های جاوا صحبت کنیم. برای یادگیری عملگر های جاوا با **برنامه چی** همراه باشید.

در این جلسه چه چیزهایی یاد میگیرید؟

دسته بندی عملگر های جاوا
عملگر های ریاضی
عملگرهای رابطه ای جاوا
عملگر های بیتی
عملگر های منطقی
عملگر های وظیفه ای
عملگر های متفرقه
*عملگر شرطی(:?)
*عملگر instanceof
اولویت عملگر های جاوا

دیدن این مقاله در سایت

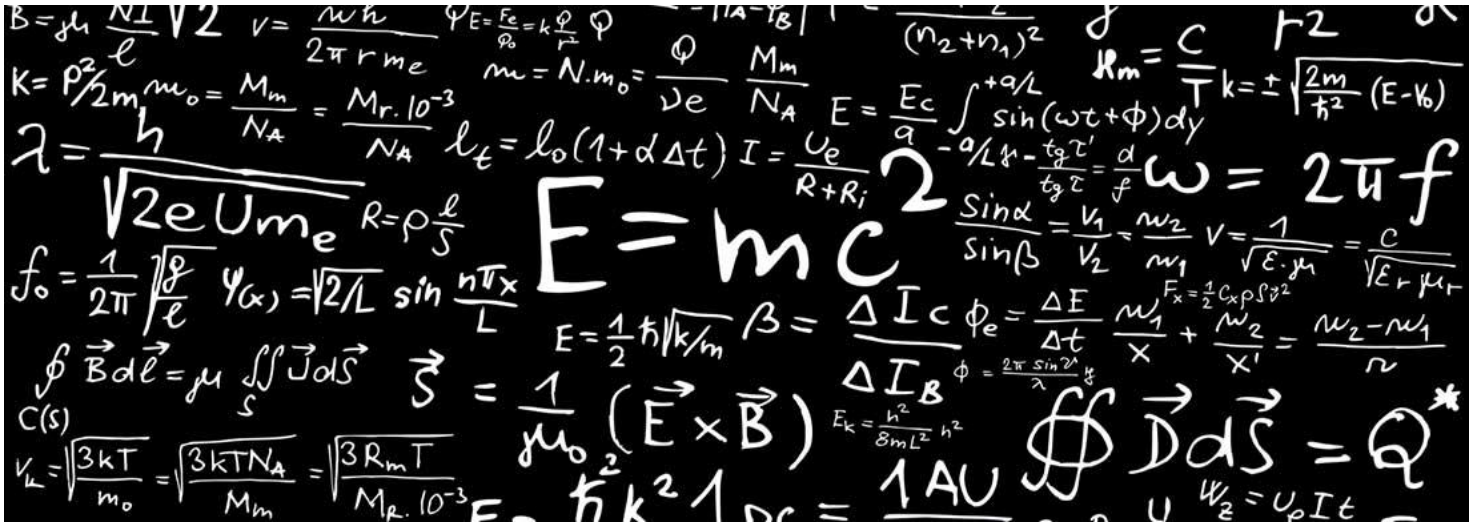
دسته بندی عملگر های جاوا

جاوا یک سری بسیار قدرتمند از عملگر ها دارد که میتوانید با استفاده از آنها متغیر ها را دستکاری کنید. میتوانیم همه عملگر های جاوا را در گروه های زیر دسته بندی کنیم:

- عملگر های ریاضی
- عملگر های ارتباطی
- عملگر های بیتی
- عملگر های منطقی
- عملگر های وظیفه ای
- عملگر های متفرقه

عملگر های ریاضی

عملگرهای ریاضی با نام انگلیسی Arithmetic Operators دقیقا همان عملگر هایی هستند که حداقل یکبار هم که شده در مدرسه با آنها کار کرده ایم. ضرب، تقسیم، جمع، تفریق و تعدادی دیگر، در میان این نوع این عملگر های جاوا قرار میگیرند. نوع عملکرد آنها هم دقیقا همان مدلی هست که در زیاضیات و جبر هر روزه خودمان از آنها استفاده میکنیم.



عملگر های ریاضی در عبارت هایی که به محاسبات ریاضی نیاز دارند استفاده میشوند، دقیقا به همان شکلی که در جبر به کار میروند. در جدول زیر همه عملگر های ریاضی که در جاوا وجود دارند را میتوانید ببینید. (فرض کنید که A مقدار 10 و B مقدار 20 را درون خود نگه میدارد).

عملگر	توضیحات	مثال
+ (جمع)	مقادیر دو طرف عملگر را با یکدیگر جمع میکند.	C = A + B حاصل جمع A + B را در C ذخیره میکند.
- (تفریق)	عملوند سمت راست را از عملوند سمت چپ کم میکند.	C += A برابر است با معادله C = C + A
*** (ضرب)**	مقادیر دو طرف عملگر را در یکدیگر ضرب میکند.	C -= A برابر است با معادله C = C - A
/ (تقسیم)	مقدار سمت چپ عملگر را به مقدار سمت راست آن تقسیم میکند.	C *= A برابر است با معادله C = C * A
% (باقیمانده)	مقدار سمت چپ عملگر را به مقدار سمت راست آن تقسیم میکند و باقیمانده را برمیگرداند.	C /= A برابر است با معادله C = C / A
++ (افزایش)	مقدار عملوند را 1 واحد افزایش میدهد.	C %= A برابر است با معادله C = C % A
-- (کاهش)	مقدار عملوند را 1 واحد کاهش میدهد.	C <<= 2 برابر است با معادله C = C << 2

عملگرهای رابطه ای جاوا

عملگر های جاوا که از نوع رابطه ای میباشند، برای بررسی ارتباط یا رابطه ی میان متغیر ها به کار میروند. رابطه بین دو متغیر میتواند مواردی مانند بزرگتر، کوچکتر، مساوی و ... بشوند. یعنی این operator ها بررسی میکنند که دو متغیر با یکدیگر چه رابطه ای دارند. عملگر های رابطه ای معمولا در شرط ها استفاده میشوند و توسط آنها میتوانید جریان اجرای برنامه را کنترل کنید. هنگامی که رابطه ای که بین دو متغیر مینویسیم برقرار باشد، نتیجه آن مقدار true (درست) برگردانده میشود.

عملگر	توضیحات	مثال
== (مساوی است با)	بررسی میکند که دو متغیر با یکدیگر مساوی هستند یا نه. اگر مساوی بودند مقدار true را برمیگرداند.	(A == B) مقدار true نمیدهد.
=! (مساوی نیست با)	بررسی میکند که دو متغیر با یکدیگر نامساوی هستند یا نه. اگر نامساوی بودند مقدار true را برمیگرداند.	(A != B) مقدار true نمیدهد.
< (بزرگتر است از)	بررسی میکند که متغیر سمت چپ از متغیر سمت راست بزرگتر است یا نه. اگر بزرگتر باشد مقدار true را برمیگرداند.	(A > B) مقدار true نمیدهد.
> (کوچکتر است از)	بررسی میکند که متغیر سمت چپ از متغیر سمت راست کوچکتر است یا نه. اگر کوچکتر باشد مقدار true را برمیگرداند.	(A < B) مقدار true نمیدهد.
=< (بزرگتر یا مساوی است با)	بررسی میکند که متغیر سمت چپ بزرگتر یا مساوی متغیر سمت راست میباشد یا نه. هرکدام از این دو شرط برقرار باشد، مقدار true را برمیگرداند.	(A >= B) مقدار true نمیدهد.
=> (کوچکتر یا مساوی است با)	بررسی میکند که متغیر سمت چپ کوچکتر یا مساوی متغیر سمت راست میباشد یا نه. هرکدام از این دو شرط برقرار باشد، مقدار true را برمیگرداند.	(A <= B) مقدار true نمیدهد.

عملگر های بیتی

عملگر های جاوا دارای چندین عملگر بیتی نیز میباشند. این عملگر ها میتوانند روی انواع داده ای integer، long، int، short، char و byte اعمال بشوند. عملگر های بیتی روی بیت ها کار میکنند و عملیات خودشان را بیت به بیت انجام میدهند. ساختار بیت ها در مبنای دو نمایش داده میشوند. یعنی اگر بخواهیم یک عدد را نشان بدهیم، این عدد با یک سری 0 و 1 نشان داده میشود.

سپس این عملگر ها بیت به بیت جلو میروند و اعمالی که برای آنها تعریف کرده ایم را یک به یک انجام میدهند. مثلا کار هایی مثل & (AND) یا | (OR) را اینگونه انجام میدهند که یکی یکی روی صفر و یک ها پیشمایش میکند و AND و OR را اعمال میکند. سپس خروجی تولید میشود.

توضیح کوتاه درباره AND و OR:

در دنیای کامپیوتر ها همه چیز به صفر و یک ترجمه میشوند. صفر یعنی false، نادرست یا خاموش. یک هم به معنی true، درست و یا روشن میباشد. دو عملگر به نام های And و Or وجود دارند که اینگونه عمل میکنند: And هر وقت که هر دو طرف عملگر 1 باشند، مقدار یک را بر میگرداند. اگر هر کدام از طرف ها، یا هردوی آنها 0 بشوند، نتیجه صفر خواهد شد. عملگر Or هم وقتی یکی از دو طرف، یا هردوی آنها 1 باشند، نتیجه 1 را برمیگرداند. یعنی فقط در صورتی صفر میشود که هر دو طرف عملگر صفر باشند.

فرض کنید a=60 و b=13 باشد. همین متغیر ها در زبان باینری (مبنای 2) به این صورت نمایش داده میشوند:

a = 0011 1100

b = 0000 1101

به عنوان مثال چندتا از عملگر های بیتی به صورت زیر عمل میکند:

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

جدول زیر عملگر های بیتی را بصورت لیست به شما نشان میدهد. (فرض کنید A مقدار 60 و B مقدار 13 را نگه میدارد).

عملگر	توضیحات	مثال
& (AND بیتی)	عملگر AND بیتی مقدار یک بیت را در نتیجه کپی میکند، اگر آن مدار در هر دو عملگر 1 باشد. (فقط وقتی هر دو 1 باشند، مقدارش true میشود).	(A & B) مقدار 12 را برمیگرداند که اینگونه آن را نمایش میدهیم: 0000 1100
| (OR بیتی)	عملگر OR بیتی مقدار یک بیت را در نتیجه کپی میکند، اگر آن مدار در هر دو عملگر 1 باشد. (فقط وقتی هر دو 1 باشند، مقدارش true میشود).	(A & B) مقدار 61 را برمیگرداند که اینگونه آن را نمایش میدهیم: 0011 1101
^ (XOR بیتی)	مقدارش true میشود فقط هنگامی که عملگر ها شبیه هم نباشند.	(A ^ B) مقدار 49 را برمیگرداند که اینگونه آن را نمایش میدهیم: 0011 0001
~ (نقیض بیتی)	این عملگر را هرجا که استفاده کنید مقدار آن بیت را برعکس میکند.	(A~) (A & B) مقدار 61- را برمیگرداند که اینگونه آن را نمایش میدهیم: 1100 0011
<< (شیفت به سمت چپ)	شیفت به سمت چپ بیتی عملوند سمت چپ را به تعداد عددی که سمت راست نوشته ایم، به طرف چپ منتقل میکند.	(A << 2) مقدار 240 را برمیگرداند که اینگونه آن را نمایش میدهیم: 1111 0000
>> (شیفت به سمت راست)	شیفت به سمت راست بیتی عملوند سمت چپ را به تعداد عددی که سمت راست نوشته ایم، به طرف راست منتقل میکند.	(A >> 2) مقدار 15 را برمیگرداند که اینگونه آن را نمایش میدهیم: 1111
>>> (شیفت به سمت چپ و پر کردن با صفر)	شیفت به سمت چپ بیتی عملوند سمت چپ را به تعداد عددی که سمت راست نوشته ایم، به طرف چپ منتقل میکند و بقیه جا ها را با صفر پر میکند.	(A >>> 2) مقدار 15 را برمیگرداند که اینگونه آن را نمایش میدهیم: 0000 1111

عملگر های منطقی

جدول زیر عملگر ها منطقی را به شما نشان میدهد. (فرض کنید A مقدار true و B مقدار false را درون خود نگه داری میکند.

عملگر	توضیحات	مثال
&& (AND منطقی)	عملگر AND منطقی اگر هر دو مقدار غیر صفر باشند، مقدار true برمیگرداند.	(A && B) مقدارش false است.
|| (OR منطقی)	عملگر OR منطقی اگر هر کدام از دو مقدار غیر صفر باشند، مقدار true برمیگرداند.	(A || B) مقدارش true است.
! (NOT منطقی)	هرجا استفاده بشود مقدار منطقی را برعکس میکند.	(!A) مقدارش true است.

عملگر های وظیفه ای

عملگر های جاوا انواع مختلف عملگر های وظیفه ای را شامل میشوند. این مدل ها میتوانند وظیفه هایی را در کد ها به عهده بگیرند و انجام بدهند. عملگر های وظیفه ای معمولا از ترکیب دیگر Operator هایی که تا اینجا درباره آنها خواندیم به وجود می آیند. کاری که انجام میدهند هم به این صورت است که یکبار آن را جلوی یک متغیر مینویسیم و عملی که تعریف کرده ایم را روی همان متغیر انجم میدهد و دوباره مقدار به دست آمده را روی همان متغیر ذخیره میکند.

پس یعنی در اینجا دیگر به دو متغیر نیاز نداریم. به عنوان مثال اگر بنویسیم a+=2، این operator باعث میشود که متغیر a یکبار با 2 جمع بشود، بعد دوباره روی a ذخیره شود. یعنی این عملگر دقیقا اینکار را میکند: a=a+2

در جدول زیر میتوانید عملگر های وظیفه ای را ببینید.

عملگر	توضیحات	مثال
=	عملگر وظیفه ای ساده به نام انتساب. مقدار سمت راست را درون متغیر سمت چپ عملگر میریزد.	C = A + B حاصل جمع A + B را در C ذخیره میکند.
+=	عملگر جمع و انتساب. عملوند راست را با عملوند چپ جمع میکند و مقدار را درون عملوند سمت چپ ذخیره میکند.	C += A برابر است با معادله C = C + A
-=	عملگر منها و انتساب. عملوند راست را از عملوند چپ کم میکند و مقدار را درون عملوند سمت چپ ذخیره میکند.	C -= A برابر است با معادله C = C - A
*=	عملگر ضرب و انتساب. عملوند راست را در عملوند چپ ضرب میکند و مقدار را درون عملوند سمت چپ ذخیره میکند.	C *= A برابر است با معادله C = C * A
/=	عملگر تقسیم و انتساب. عملوند راست را به عملوند چپ تقسیم میکند و نتیجه را درون عملوند سمت چپ ذخیره میکند.	C /= A برابر است با معادله C = C / A
%=	عملگر باقیمانده و انتساب. عملوند راست را به عملوند چپ تقسیم میکند و باقیمانده را درون عملوند سمت چپ ذخیره میکند.	C %= A برابر است با معادله C = C % A
<<=	عملگر شیفت به سمت چپ و انتساب.	C <<= 2 برابر است با معادله C = C << 2
>>=	عملگر شیفت به سمت راست و انتساب.	C >>= 2 برابر است با معادله C = C >> 2
&=	عملگر AND بیتی و انتساب.	C &= 2 برابر است با معادله C = C & 2
^=	عملگر OR بیتی و انتساب.	C ^= 2 برابر است با معادله C = C ^ 2
|=	عملگر نقیض بیتی و انتساب.	C |= 2 برابر است با معادله C = C | A

عملگر های متفرقه

در زیر میتوانید دیگر عملگر های جاوا را ببینید که در این زبان پشتیبانی میشوند.

عملگر شرطی (:?)

عملگر شرطی همچنین با نام عملگر های سه گانه هم شناخته میشوند. این عملگر برای کار کردن به سه متغیر نیاز دارد و برای این به کار میرود که یک عبارت را ارزیابی کند که مقادیر Boolean یا همان true و false دارند. هدف این عملگر تصمیم گیری میباشد. یعنی با استفاده از شرایط میتوانید تصمیم بگیرید چه اعمالی انجام بشود و چه مقادیری در متغیر ها ریخته بشوند. این عملگر را بصورت زیر در زبان جاوا مینویسیم:

```
variable x = (expression) ? value if true : value if false
```

در ادامه یک مثال را میبینید.

مثال

```
public class Test {
  public static void main(String args[]) {
    int a, b;
    a = 10;
    b = (a == 1) ? 20: 30;

    System.out.println( "Value of b is : " +  b );

    b = (a == 10) ? 20: 30;

    System.out.println( "Value of b is : " + b );
  }
}
```

خروجی این کدها باید مانند زیر باشد:

```
Value of b is : 30

Value of b is : 20
```

عملگر instanceof

این عملگر فقط برای متغیر های object reference استفاده میشوند. این متغیر بررسی میکند که آیا یک شئ از یک نوع خاص میباشد یا نه؟ (مثلا از نوع کلاس یا از نوع اینترفیس). عملگر instanceof بصورت زیر نوشته میشود:

```
( Object reference variable ) instanceof  (class/interface type)
```

اگر شئ سمت چپ عملگر نوعی از کلاس یا اینترفیسی باشد که در سمت راست عملگر نوشته شده است، این عملگر یک عبارت true را برمیگرداند. یعنی اگر آبجکت سمت چپ عملگر از آبجکت سمت راست عملگر مرجع گرفته باشد، (یعنی یک رابطه IS-A داشته باشند)، مقدار این عملگر برابر true خواهد شد. در زیر یک مثال از کاربرد آن را مشاهده میکنید:

```
public class Test {
  public static void main(String args[]) {
    String name = "James";
    // following will return true since name is type of String
    boolean result = name instanceof String;
    System.out.println( result );
  }
}
```

این کد ها نتیجه زیر را تولید خواهند کرد:

خروجی

```
true
```

اگر آبجکتی که در حال بررسی شدن میباشد، از لحاظ وظیفه سازگار با نوع سمت راست عملگر باشد، باز هم مقدار true را برمیگرداند. یک مثال دیگر را در زیر میتوانید ببینید.

```
class Vehicle {}
public class Car extends Vehicle {
  public static void main(String args[]) {
    Vehicle a = new Car();
    boolean result =  a instanceof Car;
    System.out.println( result );
  }
}
```

خروجی

```
true
```

اولویت عملگر های جاوا

اولویت ها در عملگر های جاوا، دسته بندی شرایط را در یک عبارت مشخص میکنند. این موضوع بر اینکه یک عبارت چگونه بررسی میشود تاثیر میگذارند. تعداد معینی از عملگر های جاوا اولویت بیشتری

نسبت به بقیه آنها دارند. برای مثال عملگر ضرب اولویت بیشتری نسبت به عملگر ضرب دارد. مثلا اگر عبارت x = 7+3*2 را داشته باشیم، مقدار x برابر 13 میباشد، نه 20. زیرا عملگر ضرب اولویت بیشتری نسبت به جمع دارد. بنابراین ابتدا ضرب دو عدد 3 و 2 انجام میشود و سپس حاصل آن، یعنی 6، با عدد 7 جمع میشود.

در جدول زیر عملگر های جاوا که اولویت بیشتری در زبان جاوا دارند بالاتر نشان داده شده اند. هرچه به سمت پایین جدول برویم اولویت ها کمتر خواهند شد. در یک عبارت، عملگر هایی که اولویت بیشتری داشته باشند، زودتر اجرا خواهند شد.

دسته بندی	عملگر	اولویت
پسوند	expression++ expression--	چپ به راست
تک واحدی	++expression --expression +expression -expression! ~	راست به چپ
ضربی	% / *	چپ به راست
افزودنی	- +	چپ به راست
شیفت کننده	<<< << >>	چپ به راست
رابطه ای	=< => < >instanceof	چپ به راست
تساوی	=! ==	چپ به راست
AND بیتی	&	چپ به راست
OR بیتی	^	چپ به راست
XOR بیتی	|	چپ به راست
AND منطقی	&&	چپ به راست
OR منطقی	||	چپ به راست
شرطی	?:	راست به چپ
انتسابی	=<<< =<< =>> =| =^ =% =/ =* =- =+ =	راست به چپ

در جلسه بعد چه چیزی یاد میگیریم؟

در جلسه بعد درباره کنترل حلقه ها در زبان جاوا بحث خواهیم کرد. در قسمت بعدی انواع مختلفی از حلقه های جاوا معرفی میشوند و سپس با نحوه کار کردن با آنها و هدفشان در زبان جاوا آشنا میشویم.

جلسه قبل: تعیین کننده سطح دسترسی جاوا

جلسه بعد: حلقه ها در جاوا