بسمه تعالي

واحد آموزش

مرکز تحقیقاتی فناوری اطلاعات و ارتباطات پیشرفته

دانشگاه صنعتی شریف

آموزش هسته جاوا 1.7

استاد امیرسام بهادر

مقدمه اي در مورد جاوا

جاوا یک زبان برنامه نویسي قدرتمند است، و در سال ۱۹۹۵، توسط جیمز گاسلینگ به طور رسمي به برنامه نویسان و گسترش دهندگان برنامه هاي کاربردي معرفي گردید. ابتدا نام این زبان، اوآک (OAK) بوده، که به دلیل مشکلات ثبتي به جاوا، تغییر نام داده است. اوآک به معناي درخت بلوط، و جاوا یک اسم محلي براي قهوه است. این زبان برنامه نویسي با JavaScript هیچگونه ارتباطي ندارد.

در ابتدا پروژه اي با نام گرین (Green) در شرکت سان تعریف شد. مقصود از این پروژه، سکویي براي پیاده سازي برنامه هاي کاربردي بر روي تمامي ابزارهاي الکترونیکي همانند: مایکرویو، تلویزیون و ... بود. اما در آن زمان، این پروژه با شکست روبرو شد. این اتفاق باعث شد، تا جیمز (مدیر پروژه شکست خورده Green) به کمک تیم خود یک زبان برنامه نویسي جدید تولید نماید. یکي از دلایلي که بعضي از مواقع جاوا را به عنوان یک سکوي برنامه نویسي یاد مي کنند، همین موضوع است.

برتري هاي زبان برنامه نویسي جاوا نسبت به سایر زبان هاي برنامه نویسي

- مستقل از سکوي اجرا (Free Platform): این خاصیت به برنامه نویسان و گسترش دهندگان این امکان را مي دهد، تا بدون وابستگي به هیچ سیستم عاملي، برنامه هاي کاربردي خود را تولید و اجرا نمایند.
- پشتیباني از برنامه نویسي شي گرا (Object Oriented Programming): این خاصیت به برنامه نویسان و گسترش دهندگان این امکان را مي دهد، تا بتوانند برنامه هاي کاربردي خود را پیاده، و آن را به سهولت گسترش دهند.
- داراي تکنولوژي هاي متعدد کد باز و غیر کد باز: همان گونه که در پیشتر یادآور شدیم، جاوا متعلق به شرکت سان مي باشد، اما شرکت هاي متعددي براي این زبان، تکنولوژي ها و فریم ورک هاي مختلفي را ارایه کرده اند.

توضیحي در مورد مثال‌هاي این فصل

لازم به ذکر است که گروهي از مثال‌هاي ارائه شده در این فصل اجرا نمي شوند. دلیل این امر آن است که برنامه نویس ابتدا باید با مفاهیم بنیادي جاوا آشنا سپس درگیر قواعد آن شود، در غیر اینصورت برنامه نویس و فراگیرنده دچار سر درگمي مي‌شوند.

متد Main

پدر زبان جاوا زبان C است. قالب برنامه‌نویسي در زبان C بر پایه توابع است و براي اجراي برنامه، مترجم زبان C بدنبال تابع اصلي که همان تابع main است مي‌گردد و این تابع را اجرا مي‌کند. پس اجراي برنامه شما با اجراي این تابع آغاز مي‌گردد و هر کلاس شما که حاوي تابع یا همان متد main باشد به عنوان کلاس اجرایي شما مي‌تواند در نظر گرفته شود.

```
public static void main(String [] arg)
{
        System.out.print("Hello");
}
```

خروجي این برنامه فوق کلمه Hello است.

نوشتن توضیحات در برنامه

در صورتي که شما در زبان برنامه نویسي جاوا از علامت // استفاده کنید جاوا آن خط از برنامه را نادیده مي‌گیرد. از این روش براي نوشتن توضیحات در برنامه استفاده مي‌شود. به طور مثال :

```
public static void main (String [] arg)
{
        System.out.print("Hello"); // Tozihat
}
```

در صورتي که شما در زبان برنامه نویسي جاوا از علامت /** ... */ استفاده کنید جاوا آن قسمت از خطوط برنامه را نادیده مي گیرد. از این روش براي نوشتن توضیحات در برنامه استفاده مي‌شود. به طور مثال :

```
public static void main (String [] arg)
{
        System.out.print("Hello");
   /**
    * Tozihat
    */
}
```

متغيرها و نوع‌ها

همانطور كه مي‌دانيد در دنياي واقعي اطلاعات به دسته‌هاي مختلفي تقسيم مي‌شوند و هر دسته از نوع‌هاي مختلفي تشكيل شده است، مثلاً اطلاعات شخصي يک فرد شامل نام و نام خانوادگي وي كه از نوع رشته اي و شماره شناسنامه كه از نوع عددي و تاريخ تولد كه از نوع تاريخ است. ما براي نگهداري اين اطلاعات در كامپيوتر مجبوريم مقدار فضايي از حافظه كامپيوتر را به آن اختصاص دهيم. كه توانايي نگهداري اين اطلاعات را براي ما داشته باشد. به نوع اطلاعات در كامپيوتر Data Type گفته مي شود كه از ديد كلي به دسته‌هاي زير تقسيم مي‌شوند:

اطلاعات عددي: كه مقادير عددي را در خود ذخيره مي‌كند.

اطلاعات حرفي: كه براي نگهداري كاراكترها استفاده مي‌شود .

اطلاعات ارجاعي: كه براي دسترسي به يک محدوده خاص از فضاي حافظه كه داراي يک قالب خاص از اطلاعات مي‌باشد، مورد استفاده قرار مي‌گيرد.

اطلاعات منطقي: كه از اين نوع براي عمليات منطقي (صحيح يا غلط)، استفاده مي‌شود.

زبان برنامه‌نويسي جاوا نيز نوع‌هاي اطلاعاتي مختلفي را در اختيار برنامه نويسان قرار مي‌دهد، اين نوع ها عبارتند از:

int: كه براي نگهداري اعداد صحيح استفاده مي‌شود.

```
int i = 12;
```

long: براي نگهداري اعداد بسيار بزرگ مورد استفاده قرار مي گيرد. علامت l در انتهاي عدد مشخص كننده نوع long مي باشد.

```
long i = 14l;
```

float: كه براي نگهداري اعداد اعشاري كوتاه مورد استفاده قرار مي‌گيرد. علامت f در انتهاي عدد مشخص كننده نوع float مي باشد.

```
float i = 13.5f;
```

double: كه براي نگهداري اعداد اعشاري بلند مورد استفاده قرار مي‌گيرد. علامت d در انتهاي عدد مشخص كننده نوع double مي باشد.

```
double d = 14.0d;
```

char: كه براي نگهداري يک كاراكتر استاندارد مورد استفاده قرار مي‌گيرد. علامت ‘ در ابتدا و انتهاي حرف مشخص كننده نوع كاراكتري مي باشد.

```
char i = ‘a’;
```

boolean: كه براي نگهداري يک حالت منطقي مورد استفاده قرار مي‌گيرد.

```
boolean i = true;
```

String: كه براي نگهداري يک رشته مورد استفاده قرار مي‌گيرد. (توجه داشته باشيد كه String در حقيقت يک كلاس است) علامت " در ابتدا و انتهاي جمله مشخص كننده نوع رشته اي مي باشد.

```
String i = “salam donya”;
```

براي تعريف متغير در جاوا بدين صورت عمل مي‌كنيم.

```
DataType  Variablename [variablenames];
```

مثال :

```
int num ;
```

int n1,n2;

float p;

اين نكته را به يا داشته باشيد كه جاوا به حروف بزرگ و كوچک حساس است.

انواع عملگرها

وقتي ما اطلاعات را به منظور پردازش در كامپيوتر ذخيره مي‌كنيم پس بايد دستوراتي براي پردازش نيز در نظر بگيريم كه به اين دستورات عملگر گفته مي‌شود و برحسب نوع عملي كه انجام مي‌دهند به دسته‌هاي مختلفي تقسيم مي‌شوند. براي مثال از عملگر"=" براي مقداردهي استفاده مي‌شود و اصطلاحاً به آن عملگر Assignment گفته مي‌شود. مانند :

int number ;

number = 2 ;

با اين عمل مقدار ۲ به متغير number نسبت اعطا مي‌شود.

عملگرهاي محاسباتي

از اين عملگرها براي انجام اعمال رياضي بر روي مقادير يا متغيرها استفاده مي‌شود. اين عملگرها عبارتند از :

* : براي عمل ضرب مانند A * B

/ : براي عمل تقسيم مانند A / B

+ : براي عمل جمع مانند A + B

% : براي بدست آوردن حاصل باقيمانده يک تقسيم استفاده مي‌شود مانند 2 % 5 كه حاصل آن برابر ۱ مي‌باشد.

- : براي عمل تفريق و يا نشان دادن منهاي يكاني از اين عملگر استفاده مي‌شود مانند ۲ - ۳ و يا ۲-

++ : اين عملگر يک واحد بر مقدار متغير مورد نظر اضافه مي‌كند مانند:

A=2 ;

A++ ;

كه بعد از اجراي اين دستور مقدار A برابر ۳ خواهد شد.

-- : اين عملگر يک واحد از مقدار متغير مورد نظر كم مي‌كند مانند:

A = 2 ;

A--;

كه بعد از اجراي اين دستور مقدار A برابر ۱ خواهد شد.

عملگر هاي رابطه اي

از اين عملگرها براي مشخص كردن يا مقايسه رابطه بين دو يا چند حالت يا مقدار استفاده مي‌شود. نتيجه حاصل از اين عملگرها بر روي عملوندهاي خود يک مقدار منطقي مي‌باشد. البته اين نكته را فراموش نكنيد كه دو عملوند اين عملگرها بايد از يک جنس يا يک نوع باشند. اين عملگرها عبارتند از:

== :براي نشان دادن يا مقايسه دو چيز كه مقادير آنها باهم برابر باشد. مثلا A == B كه اگر مقدار A با مقدار B برابر باشد حاصل برابر با true (صحيح) و در غير اين صورت برابر با false (نادرست) مي‌باشد.

> :براي نشان دادن يا مقايسه دو چيز كه مقدار اولي از دومي كوچكتر باشد. مثلا A < B كه اگر مقدار A از مقدار B كوچكتر باشد حاصل برابر با true (صحيح) و در غير اين صورت برابر با false (نادرست) مي‌باشد.

< :براي نشان دادن يا مقايسه دو چيز كه مقدار اولي از دومي بزرگتر باشد. مثلا A > B كه اگر مقدار A از مقدار B بزرگتر باشد حاصل برابر با true (صحيح) و در غير اين صورت برابر با false (نادرست) مي‌باشد.

=! :زماني خروجي اين عملگر برابر با true مي‌باشد كه مقدار دو عملوند آن باهم برابر نباشد. اگر A =2 و B = 3 باشد در اين صورت حاصل A =! B برابر با true مي‌شود.

البته در جاوا عملگرهاي ديگري مانند => (كوچكتر مساوي) و =< (بزركتر مساوي) نيز وجود دارد.

عملگر هاي منطقي

اين عملگرها بر روي مقادير منطقي عمل مي‌كنند و نتيجه حاصل از آنها نيز يک مقدار منطقي است. اين عملگرها در شرطها

بسيار مورد استفاده قرار مي‌گيرند. و عبارتند از:

&& : اين عملگر جزء عملگرهاي باينري مي‌باشد يعني دو ورودي دارد. عملگر && معادل "و" مي‌باشد. اين عملگر بر پايه جدول زير عمل مي‌كند.

AND	T	F
T	T	F
F	F	F

همانطور كه مي‌بينيد زماني خروجي اين عملگر صحيح است كه حتماً دو ورودي آن صحيح باشد.

|| : اين عملگر نيز جزء عملگرهاي باينري مي‌باشد يعني دو ورودي دارد. عملگر || معادل "يا" مي‌باشد. اين عملگر بر پايه جدول زير عمل مي‌كند.

OR	T	F
T	T	T
F	T	F

همانطور كه مي‌بينيد زماني خروجي اين عملگر ناصحيح است كه فقط هر دو ورودي آن ناصحيح باشد.

! : اين عملگر جزء عملگرهاي تک مقدار مي‌باشد يعني فقط يک عملوند دارد. اين عملگر را "نقيض" مي‌نامند و بر پايه جدول زير عمل مي‌كند.

	T	F
NOT	F	T

همانطور كه مي‌بينيد اين عملگر ورودي خود را از لحاظ منطقي برعكس مي‌كند. يعني صحيح را ناصحيح و ناصحيح را به صحيح تغيير مي‌دهد.

عملگرهاي انتسابي

از اين عملگرها براي مقداردهي متغيرها و يا بدست آوردن حاصل يک عبارت استفاده مي‌شود. كه ما در اينجا چند عدد از پر كاربردترين آنها را معرفي مي‌كنيم.

= : همانطور كه گفتيم از اين عملگر براي مقداردهي مستقيم به يک متغير استفاده مي‌شود مانند:

```
Number = 2 ;
```

=+ : اين عملگر علاوه بر انتساب حاصل عبارت يا مقدار سمت راست خود را با مقدار فعلي متغير سمت چپ خود جمع مي‌كند.

مثال :

```
Num += 3 ;
```

معادل خط زير مي‌باشد.

```
Num = Num + 3 ;
```

همانند مثال بالا براي بقيه عملگرهاي انتسابي هم به روش ذيل مي‌توان عمل كرد.

-= : براي عمل تفريق — `Num -= 2 ; ==> Num = Num - 2 ;`

*= : براي عمل ضرب — `Num *= 2 ; ==> Num = Num * 2 ;`

/= : براي عمل تقسيم — `Num /= 2 ; ==> Num = Num / 2 ;`

=% : براي عمل باقيمانده Num %= 2 ; ==> Num = Num % 2 ;

البته عملگرهاي ديگري نيز براي عمليات‌هاي منطقي مانند =& وجود دارد كه در اينجا به آنها اشاره نمي‌كنيم.

تقدم عملگرها

يكي از اساسي‌ترين بحث‌ها در استفاده از عملگرها بحث تقدم عملياتي عملگرها مي‌باشد. زيرا رعايت تقدم باعث تغيير در مقدار حاصل عبارت مي‌شود براي مثال حاصل عبارت 5 * 2 + 3 برابر با ١٣ است در صورتي كه در نگاه اول بنظر مي‌رسد كه حاصل عبارت برابر با ٢٥ باشد. علت اينكه اين حاصل با فرض ما متفاوت است تقدم عملگر ضرب نسبت به جمع است در واقع اول عمل ضرب انجام مي‌شود كه حاصل 10 = (5 * 2) مي‌باشد و بعد عمل جمع و در نهايت جواب ١٣ مي‌شود.

تقدم عملگرها در جاوا به صورت زير است :

١ عبارت درون داخلي ترين پرانتز
٢ عملگرهاي بيتي (كه در اين كتاب توضيح نداديم)
٣ عملگرهاي محاسباتي
۴ عملگرهاي منطقي
۵ عملگرهاي رابطه‌اي
۶ عملگرهاي انتسابي

البته در درون هر كدام از اين دسته‌ها نيز بين عملگرها تقدم وجود.

ساختار هاي كنترلي

در زماني كه يک برنامه بر روي يک كامپيوتر اجرا مي‌شود از خط اول برنامه روند اجراي برنامه آغاز مي‌شود و تا پايان به ترتيب خط به خط اجرا مي‌شود به عبارت ديگر دستورات پشت سر هم اجرا مي‌شوند. اگر ما بخواهيم در طول اجراي برنامه اين روند را تغيير دهيم يعني بعد از دستور ٢ بجاي اينكه دستور ٣ اجرا شود دستور ۵ يا ١ اجرا شود بايد از دستوراتي استفاده كنيم كه اين روند ترتيبي را بشكنند. به اين دستورات، دستورات كنترلي مي‌گويند و به ساختارهاي برنامه‌نويسي كه از اين دستورات استفاده مي كنند ساختار كنترلي گفته مي‌شود. در زير چند ساختار كنترلي معروف كه در برنامه‌نويسي بسيار پركاربرد است را توضيح مي دهيم.

ساختار كنترلي شرطي

به اين دليل به آنها ساختار كنترلي شرطي مي‌گويند كه اجراي برنامه را وابسته به وجود مقدار يا شرط خاص مي‌كنند.

ساختار كنترلي شرطي if

يكي از پركاربردترين ساختارهاي كنترلي كه در تمام برنامه‌ها مورد استفاده قرار مي‌گيرد ساختار شرطي ifمي‌باشد. برنامه نويس با استفاده از اين ساختار مشخص مي‌كند كه در شرايط مورد نظرش چه دستوراتي بايد اجرا شود. ساختار كلي اين دستور به فرم زير است:

فرم اول : اگر شرط درست باشد دستور ١ اجرا مي‌شود و اگر نا درست باشد روند اجراي برنامه به دستور بعد از if خواهد رفت.
مثال :

```
public static void main (String [] arg)
{
        int Num ;
        Num = 2 ;

        if ( Num  == 2 )
        {
                System.out.print ("Number is two");
        }
}
```

اگر مقدار Num را به مقداري به جز مقدار ٢ تغيير دهيد در خروجي چيزي چاپ نمي‌شود.

فرم دوم : گاهي اوقات ما لازم داريم در صورتي كه شرط if صحيح نباشد عمل ديگري انجام دهيم براي اين موضوع از دستور else استفاده مي‌كنيم.

```
public static void main (String [] arg)
{
	int Num ;
	Num = 2 ;

	if ( Num  == 2 )
	{
		System.out.print ("Number is two");
	}
	else
	{
		System.out.print ("Number is not two");
	}
}
```

در اين حالت اگر مقداردهي به متغيير Num همان ۲ باشد در خروجي Number is two چاپ مي‌شود، در غير اين صورت چيزي كه در خروجي چاپ مي‌شود Number is not two است.

نكته: در زبان برنامه سازي جاوا در صورتي كه دستورات داخلي بيش از يک دستور باشند از بلاک استفاده مي‌كنيم. براي اين منظور دستورات را در بين دو علامت { } قرار مي‌دهيم. مثال :

```
public static void main (String [] arg)
{
	int num ;
	num = 2 ;

	if ( num  == 2 )
	{
		System.out.print ("your num is:"+ num);
	}
}
```

حتي ما در توابع و متدهاي داخل برنامه به دليل اينكه بيش از يک دستور در درون بلاک نوشته مي‌شود از بلاک استفاده مي كنيم. البته اين قالب در زبان C استاندارد و ويرايش‌هاي بعدي زبان C و جاوا و ساير زبان‌هايي كه از C به ارث گرفته شده‌اند وجود دارد.

ساختار كنترل شرطي switch

در طول برنامه بسيار اتفاق مي‌افتد كه براي مقادير مختلف يک متغير اعمال متفاوتي بايد انجام شود. همانطور كه ديديد براي دو حالت و حتي سه حالت از ساختار if استفاده مي‌كنيم ولي براي حالت‌هاي بيشتر از ساختار switch استفاده مي‌كنيم. فرم كلي اين ساختار بصورت زير است:

اگر مقدار جلوي case با مقدار متغير برابر باشد دستورات آن قسمت اجرا مي‌شود. قسمت default، يک قسمت دلخواه در اين ساختار مي‌باشد. يعني اگر برنامه نويس نياز داشت، در برنامه خود از اين قسمت استفاده مي‌كند در غير اينصورت آن را حذف مي‌كند.

البته اين نكته را هرگز فراموش نكنيد، اگر بعد از پايان دستورات هر قسمت case از دستور break، كه يک دستور براي شكستن كليه ساختارهاي كنترلي چه شرطي و چه حلقه‌هاي تكرار مي‌باشد استفاده كنيد از قسمت شرط خارج مي‌شويد. حال اگر از اين دستور استفاده نكنيد بعد از اجراي دستورات آن قسمت از case، دستورات case هاي پايين تر نيز مورد بررسي قرار مي گيرند. البته بعد از آخرين case و يا بعد از قسمت default، احتياجي به استفاده از دستور break نمي‌باشد. چون ساختار switch

بعد از اجراي اين دستورات تمام مي‌شود.

```
public static void main (String [] arg)
{
        int day =1;
        switch ( day )
        {
                case 1 :
                        System.out.print ("شنبه");
                        break;
                case 2 :
                        System.out.print ("يكشنبه");
                        break;
                case 3 :
                        System.out.print ("دوشنبه");
                        break;
                case 4 :
                        System.out.print ("سه شنبه");
                        break;
                case 5 :
                        System.out.print ("چهار شنبه");
                        break;
                case 6 :
                        System.out.print ("پنج شنبه");
                        break;
                case 7 :
                        System.out.print ("جمعه");
                        break;
                default :
                        System.out.print("عدد وارده براي تعيين روز اشتباه است !");
        }
}
```

در برنامه فوق شما مي‌توانيد مقدار متغير day را از ١ تا ٧ مقداردهي كنيد در غير اين صورت خروجي برنامه برابر با عبارت داخل " " در قسمت default است.

ساختارهاي حلقه‌اي

از اين ساختارها براي تكرار يک يا مجموعه‌اي از دستورات استفاده مي‌شود.

ساختار حلقه تكرار while

نحوه عملكرد اين ساختار به اين صورت است، كه اگر شرط حلقه درست باشد وارد حلقه شده و دستور درون حلقه اجرا مي‌شود و تا زماني كه شرط درون دستور while درست باشد حلقه، تكرار مي‌شود. اگر ما بخواهيم بيش از يک دستور را تكرار كنيم بايد آن را در درون يک بلاک قرار دهيم. مثال :

```
public static void main (String [] arg)
{
	int num ;
	num = 0 ;
	while ( num < 5 )
	{
		System.out.print( "*" );
		num ++ ;
	}
}
```

با اجراي اين برنامه مشاهده مي‌كنيد كه ۵ ستاره در خروجي چاپ مي‌شود كه نشان دهنده اين مطلب است كه حلقه تكرار while در درون اين برنامه ۵ بار تكرار مي‌شود.

ساختار حلقه تكرار do ... while

اين حلقه جزء حلقه‌هاي تكرار شرط در انتها مي‌باشد كه اين بدين معناست كه اول دستورات درون حلقه يک بار اجرا مي‌شوند و اگر شرط حلقه درست باشد دستورات درون حلقه تكرار مي‌شوند. مثال :

```
public static void main (String [] arg  )
{
	int num ;
	num = 0 ;
	do
	{
		System.out.print( "*" );
		num ++ ;
	}
	while ( num < 5 );
}
```

براي مشخص‌تر شدن فرق اين دو حلقه اگر در ابتدا به متغير num مقدار ۱۰ بدهيد مي‌بينيد كه يک '*' چاپ مي‌شود ولي در حلقه while اگر به num مقدار ۱۰ را بدهيد هيچ '*' چاپ نمي‌شود.

ساختار حلقه تكرار for

در اين نوع ساختار حلقه‌اي شما مي‌توانيد به تعداد مورد نياز دستورات را اجرا كنيد به طور مثال:

```
public static void main (String [] arg)
{
	for (int i=0;i<10;i++)
	{
		System.out.print(i);
	}
}
```

همانطور كه مشاهده مي‌كنيد در اين جا ما يک متغير در درون for تعريف كرديم كه مقدار آن برابر "صفر" مي‌باشد، در قسمت بعدي شرطي براي آن تعريف كرديم، آن شرط به معناي اين است كه آيا مقدار i كوچكتر از ۱۰ مي‌باشد؟ جواب در مرحله اول

صحيح است پس يک واحد به i اضافه مي‌شود و در خروجي مقدارi را نمايش مي‌دهد اين مراحل آنقدر اجرا مي‌شود تا مقدار i کوچکتر از ۱۰ نباشد سپس برنامه از داخل حلقه خارج مي‌شود.

آرايه‌ها

در دنياي واقعي ما يکسري اطلاعات داريم که اين اطلاعات بصورت ليست مي‌باشند مثلاً ليست اسامي دانشجويان، ليست شماره حسابها و . . . براي نگهداري اين اطلاعات در کامپيوتر ما مجبوريم تعداد زيادي متغير تعريف کنيم اگر بخواهيم اين متغيرها را نامگذاري کنيم براي مثال بايد به صورت x1,x2,x3,...,xn تعريف کنيم. اين عمل در برنامه باعث مي‌شود که براي دسترسي به اطلاعات دچار دردسر شويم. براي رفع اين مشکل در اکثر زبان هاي برنامه سازي اين امکان وجود دارد که شما متغيري از نوع ليست داشته باشيد. که به آن آرايه (Array) گفته مي‌شود. اين نوع متغير مي‌تواند به تعداد محدود به صورت ايستا يا پويا اعضايي را دارا باشد که ما مقادير خود را در اين اعضاء ذخيره و بازيابي کنيم. نحوه تعريف يک آرايه در زبان جاوا به صورت زير است.

Type variable name [];

Variable = new Type [Size];

براي دسترسي به اعضاي آرايه به منظور ذخيره و بازيابي اطلاعات درون آرايه فقط کافي است که شماره عضو يا خانه‌اي که در اصطلاح به آن انديس آرايه مي‌گويند را بدانيد. مثال :

int [] numbers = new int [7] ;

numbers [0] = 2 ;

numbers [1] = 27 ;

numbers [2] = 20 ;

System.out.print (numbers [2]);

در اين مثال خروجي برابر عدد ۲۰ است. البته ما در جاوا آرايه‌هاي چند بُعدي نيز داريم که ما در اينجا فقط به مثالي از آرايه‌هاي دو بُعدي کفايت مي‌کنيم.

int cordinate [][] = new int [10][12];

cordinate [2][3] = 7 ;

System.out.print (cordinate [2][3]) ;

در اين مثال خروجي برابر است با ۷.

ماهيت برنامه‌نويسي شي‌گرايي

در ابتدا برنامه‌نويسي، ساختار يافته بود. اين روش فقط براي پياده‌سازي رويدادها در نظر گرفته شده بود، که سرعت برنامه نويسي را کند مي‌کرد و پروژه‌هاي نرم‌افزاري را به طور کامل غير قابل توسعه مي‌نمود تا اين که ماهيت برنامه‌نويسي شي‌گرايي (OOP) شکل گرفت. شي‌گرايي به برنامه نويس ديدگاهي ارائه مي‌دهد تا بتواند به راحتي پروژه خود را پياده‌سازي کرده و آن را قابل توسعه کند.

مفهوم کلاس

کلاس مشخصه‌اي همانند انسان مي‌باشد که داراي خصوصيات و رفتارهاي مختلفي است.

اجزاي کلاس

اجزاي کلاس عبارتند از:

- خصوصيات: خصوصيات همانند اسم، فاميل، تاريخ تولد و ... يک انسان مي‌باشد.

- متدها: متدها نيز همانند رفتار يک انسان در مقابل رويدادها عکس العمل نشان مي‌دهند.

مثال:

```
class human
{
        String lastname = "elison";
        String address="newyork" ;
        public void hello()
```

```
	{
		System.out.print("hello human");
	}
}
```

در اين مثال lastname و address خصوصيات Human و ()hello رفتار Human مي‌باشند، زماني كه فردي به Human سلام مي كند Human در پاسخ جواب مي‌دهد hello human، Human مي‌تواند هزاران رفتار و خصوصيات داشته باشد، و ساير افراد مي توانند از خصوصيات او استفاده كرده و يا رفتار او را صدا بزنند.

متدها

در زبان‌هاي برنامه‌نويسي شي‌گرا متدها همانند رفتارهاي يک انسان عمل مي‌کنند به طور مثال زماني كه شما به يک شخص سلام مي‌دهيد در اصل متد سلام شخص مقابل را صدا مي‌کنيد، و شخص مقابل جواب سلام شما را مي‌دهد. در جاوا براي پياده سازي متدها كافيست تا آنها را داخل كلاس درج كنيد. به طور مثال:

```
class Amir
{
	public void hello()
	{
		System.out.print("salam");
	}
	public void bye()
	{
		System.out.print("khodahafez");
	}
}
```

همانطور كه مشاهده مي‌کنيد امير داراي دو متد سلام و خداحافظ است، حال زماني كه شخصي متد سلام امير را اجرا مي‌کند، امير در خروجي سلام را چاپ مي‌کند و زماني كه كسي متد خداحافظ امير را صدا مي‌کند، امير در خروجي كلمه خداحافظ را چاپ مي‌کند، شايد وجود كلمات void و public براي شما عجيب باشد ما در قسمت‌هاي بعدي به توضيح در مورد كلمه void مي پردازيم، و در قسمت‌هاي آينده در مورد كلمه public توضيح خواهيم داد. متدها نيز همانند متغيرها مي‌توانند داراي مقدار باشند. void به معناي هيچ مقدار است و متدي كه void باشد نمي‌تواند مقداري را در خود ذخيره كند حال به مثال زير توجه كنيد.

```
class Amir
{
	public int hello()
	{
		return 2;
	}
}
```

بوسيله دستور return مي‌توان به متد hello مقدار داد اما توجه داشته باشيد كه اين مقدار بايد از نوع عددي باشد زيرا متد hello از نوع int است. متدها به خودي خود نمي‌توانند اجرا شوند و حتماً بايد آنها را صدا كرد. براي صدا كردن متد hello كافيست اين دستورات را در متد main و يا ساير متدهاي امير بنويسيد.

```
hello();
```

و يا مي توانيد از دستور ذيل استفاده كنيد.

```
Amir.hello();
```

حال اگر متدي داراي مقدار باشد، مقدار آنرا مي توان به اين صورت چاپ كرد:

```
System.out.print(Amir.hello());
```

متدها مي‌توانند مقداري را نيز دريافت كنند به طور مثال:

```
class Amir
{
	public void hello(String fname , String lname)
	{
		System.out.print("salam "+fname+" "+lname);
	}
}
```

همانطور كه در مثال بالا مشاهده مي‌كنيد متد hello دو ورودي lname و fname را دريافت مي‌كند. اين دو ورودي مشخص كننده نام و فاميل است. حال اگر شما نام و فاميل خود را به متد hello دهيد به طور مثال:

```
hello("Amir","bahador");
```

متد hello براي شما اين متن را چاپ مي‌كند.

```
salam Amir bahador
```

متدها مي‌توانند داراي متغير باشند، اما فراموش نكنيد كه از اين متغيرها در متدهاي ديگر نمي‌توان استفاده كرد و فقط مي‌توان از اين متغيرها در درون خود متد استفاده كرد به طور مثال:

```
class Amir
{
	public void hello()
	{
		int i =3;
	}
}
```

همانطور كه مشاهده مي‌كنيد متد hello متغير i را دارا مي‌باشد اما از اين متغير در خارج از متد نمي‌توان استفاده كرد. در يک كلاس مي‌توان چندين متد همنام داشت اما بايد ورودي هاي آنها متفاوت باشد به طور مثال:

```
class Amir
{
	public void hello(String name)
	{

	}
	public void hello(int number)
	{

	}
}
```

همانطور كه در بالا مشاهده مي‌كنيد دو متد همنام در يک كلاس وجود دارد اما ورودي هاي آنها با هم متفاوت است.

خصوصيات

خصوصيات كلاس دقيقاً همان متغيرهاي كلاس هستند به طور مثال:

```
class Amir
{
	String famil = "bahador";
}
```

همانطور كه مشاهده مي‌كنيد كلاس امير داراي خصوصيت famil است و اين خصوصيت داراي مقدار بهادر است! خصوصيات كلاس، بر عكس متغيرهاي متد داراي سطح دسترسي محدود نيستند و از آنها در همه جا مي‌توان استفاده كرد مگر خود برنامه نويس اين سطح دسترسي را تغيير دهد كه در قسمت‌هاي آينده به آن مي‌پردازيم، جالب است بدانيد كه اين خصوصيات را مي توان از سطح متد تغيير داد به طور مثال:

```
class Amir
{
	String age="70";
	public void changeAge()
	{
		this.age = "20" ;
	}
}
```

بعد از صدا كردن متد changeAge مقدار age برابر عدد ۲۰ مي‌شود، كلمه this به معناي همين شي است.

Package ها

در توليد نرم‌افزار ما نياز به كلاس‌هاي زيادي داريم، گاهي اوقات اين كلاس‌ها باعث دردسر مي‌شوند، بهترين راه براي دوري از اين دردسر اينست كه ما آنها را دسته بندي كنيم به طور مثال اگر شما يک كلاس باشيد خانه شما package شما است ! مثال :

```
package tehran;
class Amir
{

}
```

در اين مثال همانطور كه مشاهده مي‌كنيد Amir در درون tehran مي‌باشد، Amir يک كلاس است و tehran يک package مي باشد. اين را به ياد داشته باشيد كه شما مي‌توانيد كلاس‌هاي ديگري در package مورد نظر قرار دهيد، پس نتيجه مي‌گيريم كه در هر package مي‌توان چندين كلاس داشت! براي اينكه بتوانيد از كلاس‌هاي خارج از package جاري استفاده كنيد مي‌توانيد از دستور

```
import packagename.ClassName;
```

استفاده كنيد به طور مثال:

```
package tehran;
import esfehan.Farhad;
class Amir
{
```

}

كلمه this

در قسمت هاي قبل ما از واژه اي بنام this استفاده كرديم اين كلمه به معناي شي جاري مي‌باشد و به برنامه نويس كمک مي كند تا سطح ديد خود را در برنامه مشخص كند به مثال زير توجه فرماييد:

```
class Amir
{
	int age = 3;
	public void showAge()
	{
		int age = 20;
		System.out.println(age);
		System.out.println(this.age);
	}
}
```

همانطور كه در اين مثال مشاهده مي‌كنيد ما در كلاس Amir دو متغير age را موجود داريم، age اول خصوصيت كلاس است و age دوم متغير متد، با استفاده از كلمه this در داخل متد showAge ما به JDK اعلام مي‌كنيم كه متغير مد نظر ما متغير مربوط به كلاس است. خروجي متد قبل به شكل زير است:

```
20
3
```

اين نكته را به خاطر داشته باشيد كه مي‌توانيم به جاي كلمه this از نام كلاس هم استفاده كنيم به طور مثال:

```
System.out.print(Amir.age);
```

Object سازي و متد سازنده

در دنياي واقعي شي‌سازي به معناي همانندسازي است، و اين كار بسيار پيچيده و سخت است اما در برنامه‌نويسي اين كار بسيار ساده است، فرض كنيد كلاسي به نام Amir ساخته‌ايد و حال مي‌خواهيد از اين كلاس يک كپي ايجاد كنيد، سوال اين است كه چرا ما اين كار را مي‌كنيم ؟ فرض كنيد شما مي‌خواهيد دست امير را قطع كنيد، اگر اين كار را بر روي Amir انجام دهيد Amir يک دست خود را از دست مي‌دهد، و كلاس Amir صدمه مي‌بيند، حال اگر از Amir همانند سازي كنيد و نام همانند او را Amir2 قرار دهيد مي‌توانيد دست Amir2 را قطع كنيد بدون آن كه به Amir صدمه‌اي وارد شود. به اين عمل Object سازي گفته مي‌شود. ما در مثال زير دو كلاس را تعريف مي‌كنيم يک كلاس همان كلاس امير است و كلاس دوم كلاس Fact نام دارد و كار اين كلاس Object سازي از امير است !

```
class Amir
{
	int age=20;
}


class Fact
{
	public static void main (String [] arg)
	{
		Amir amir2 = new Amir();
		Amir amir۴ = new Amir();
```

```
		amir2.age=30;
		System.out.print(amir2.age); // 30
		System.out.print(amir3.age); // 20
	}
}
```

همانطور که مشاهده کردید متغیر age در امیر واقعي تغییري نکرد و متغیر amir2 تغییر کرد. حال بیایید در مورد دستور Amir amir2 = new Amir(); بیشتر کنجکاوي کنیم. آیا این دستور براي شما آشنا نیست ؟

```
String name = "ali";
```

همانطور که در گذشته توضیح داده شد این دستور متغیري از نوع String ایجاد و مقدار آن را برابر ali قرار مي داد. حال مي‌توانید متغیر name را به این شکل نیز ایجاد کنید.

```
String name = new String("ali");
```

دلیل این امر آن است که بر خلاف آنچه که آموختید نوع String در واقع وجود ندارد. String یک کلاس است و شما در اصل با این روش از کلاس String، Object سازي مي‌کنید. در صورتي که مي‌خواهید براي سایر متغیرهاي جاوا این عمل را انجام دهید باید از کلاس‌هاي مورد نظر استفاده کنید به طور مثال به جاي نوع متغیر int مي‌توانید از کلاس Integer استفاده کنید مثال:

```
Integer num = new Integer(3);
```

در این روش یک مزیت وجود دارد و آن مدیریت حافظه بر روي اشیا مي‌باشد. یعني در صورتي که به جاي نوع int از کلاس Integer استفاده کنید، جاوا در صورت استفاده نکردن شما از شي مورد نظر آن را از سطح حافظه حذف خواهد کرد.

حال بیایید به شي‌سازي از کلاس Amir برگردیم، همانطور که مشاهده کردید ما با استفاده از دستور Amir amir2 = new Amir(); موجودي به نام amir2 خلق کردیم که از نوع کلاس Amir مي‌باشد. شاید استفاده از دستور Amir() برایتان عجیب باشد این دستور متدي بنام متد سازنده کلاس را صدا مي‌کند. به مثال زیر توجه کنید:

```
class Amir
{
	int age;
	public Amir(int age)
	{
		this.age = age;
	}
}

class Fact
{
	public static void main (String [] arg)
	{
		Amir amir2 = new Amir(34);
		System.out.print(amir2.age); // 34
	}
}
```

متد سازنده به صورت اختیاري از طرف برنامه نویس تعریف مي‌شود، این متد نوعي ندارد یعني حتي void هم نیست نام این متد همنام نام کلاس است. این متد به برنامه نویس این امکان را مي‌دهد تا زماني که از کلاس، شي‌سازي مي‌کند مقادیري را به آن

ارسال کند. همان طور که در مثال بالا مشاهده کردید ما متغير age را در هنگام شي‌سازي مقداردهي کرديم. متدهاي سازنده همانند متدهاي معمولي مي‌توانند تکراري باشند اما فراموش نکنيد که بايد امضاي متدهاي سازنده با هم تفاوت داشته باشد.
پس نتيجه مي‌گيريم زماني که از دستور new Amir() استفاده مي‌کنيم در اصل متد سازنده کلاس را صدا مي‌زنيم، اين نکته را فراموش نکنيد، در صورتي که کلاس مورد نظر متد سازنده را دارا نباشد مشکلي در برنامه‌نويسي پديدار نمي‌شود و خود جاوا يک متد سازنده نامرئي را براي شما پياده‌سازي مي‌کند. به ياد داشته باشيد متد سازنده باعث بارگذاري کلاس در سطح RAM مي شود، بنابراين بايد در اين متد سعي کنيد تا از دستورات حجيم استفاده نکنيد زيرا باعث کندي عمل شي‌سازي مي‌شود.

ارث‌بري و دستور super

ارث‌بري يکي از مفاهيم بنيادي شي‌گرايي است، آيا تا به حال به فکر پياده‌سازي نرم‌افزاري قدرتمندتر از Winamp افتاده‌ايد؟ اين کار بسيار وقت گير است زيرا شما ابتدا بايد برنامه‌اي همانند Winamp را پياده‌سازي کنيد سپس بايد به آن قابليت‌هاي جديدي اضافه کنيد، اما اگر شما با مفهوم ارث‌بري آشنا باشيد اين کار را بسيار ساده‌تر مي‌توانيد انجام دهيد. براي اين کار شما مي‌توانيد از کلاس هاي برنامه Winamp ارث برده و قابليت‌هاي خود را به Winamp اضافه کنيد.(البته به شرطي که Winamp کد باز و با جاوا پياده‌سازي شده باشد) ارث‌بري در زبان‌هاي شي‌گرا دقيقاً همانند ارث‌بري در انسان است، يعني تمام خصوصيات و رفتارهاي پدر به فرزند افزوده مي‌شود. مثال :

```
class Soheil
{
	public void hello()
	{
		System.out.print("Salam");
	}
	public void bye()
	{
		System.out.print("khoda hafez");
	}
}

class Amir extends Soheil
{

}
```

توجه داشته باشيد که کلاس Amir داراي ۲ متد bye و hello است دليل اين امر آن است که کلاس امير از کلاس سهيل ارث برده است براي اين کار ما از کلمه extends در کنار کلمه امير استفاده کرده‌ايم حال يک کلاس را توليد مي‌کنيم تا از امير شي‌سازي کند.

```
class Fact
{
	public static void main(String [] arg)
	{
		Amir amir2 = new Amir();
		amir2.hello(); //خروجي برابر با سلام است
	}
}
```

اين نكته را به ياد داشته باشيد كه در ارث‌بري خصوصيات نيز ارث برده مي‌شوند، نكته ديگري كه به آن مي‌توانيم اشاره كنيم اين است كه اگر فرزند متد پدر را از قبل دارا باشد ديگر از پدر خود ارث نمي‌برد. مثال:

```
class Soheil
{
	public void hello()
	{
		System.out.print("Soheil : Salam");
	}
	public void bye()
	{
		System.out.print("Soheil : khoda hafez");
	}
}


class Amir extends Soheil
{
	public void hello()
	{
		System.out.print("Amir : Salam !");
	}
}
```

حال اگر ما متد Amir.hello() را صدا بزنيم خروجي به شكل ذيل خواهد بود :

```
Amir : Salam !
```

اگر در متد سازنده فرزند ما از دستور super() استفاده كنيم متد سازنده پدر كلاس اجرا مي‌شود در بعضي از موارد نيز ما از دستور super براي دسترسي به خود كلاس پدر استفاده مي‌كنيم، استفاده از اين دستور كاملاً شبيه به دستور this است.

سطح‌هاي دسترسي

سطح‌هاي دسترسي به شما در پياده سازي پروژه‌هاي كلان كمك مي‌كنند. شايد شما نخواهيد تمام متدها و خصوصيات پدر به فرزند منتقل شود !

كلمه public نشان دهنده سطح دسترسي عمومي است اين كلمه نشان دهنده اين موضوع است كه محدوديتي براي آن خصوصيت يا كلاس و يا رفتار در نظر گرفته نشده است. حال بياييد به ساير سطح هاي دسترسي بپردازيم !

	خود كلاس	فرزند	داخل همان package	خارج از package
public	دسترسي دارد	دسترسي دارد	دسترسي دارد	دسترسي دارد
protected	دسترسي دارد	دسترسي دارد	دسترسي دارد	دسترسي ندارد
private	دسترسي دارد	دسترسي ندارد	دسترسي ندارد	دسترسي ندارد

حال به مثال زير توجه كنيد:

```
package tehran;
```

```
class Amir
{
	private int age;
	public String address;
	protected void hello()
	{
		System.out.print("Salam");
	}

}
```

اگر کلاسي از Amir در همان package ارث برد شرايط زير حاکم است :

- age را ارث نمي‌برد.
- address را ارث مي‌برد.
- متد hello را ارث مي‌برد.

اگر کلاس ديگري در داخل package، تهران بخواهد به امير دسترسي داشته باشد شرايط زير حاکم است:

- به age دسترسي ندارد.
- به address دسترسي دارد.
- به متد hello دسترسي دارد.

اگر کلاس ديگري در خارج از package، تهران بخواهد به امير دسترسي داشته باشد شرايط زير حاکم است:

- به age دسترسي ندارد.
- به address دسترسي دارد.
- به متد hello دسترسي ندارد.

کلاس انتزاعي و متد انتزاعي

گاهي اوقات در زمان پياده‌سازي کلاس‌ها ما نياز داريم تا کلاسي تعريف کنيم که ساير کلاس‌ها نتوانند از آن کلاس شي‌سازي کنند در آن صورت ما از کلمه abstract استفاده مي‌کنيم.

```
abstract class Amir
{
	public static void main (String [] arg)
	{
		System.out.print("This is Main Method");
	}
}
```

در اين حالت کلاس امير فقط مي‌تواند يا خود اجرا شود و يا از طريق کلاسي ديگر ارث برده شود. حال اگر متدي بصورت abstract تعريف شود، براي آن متد نمي توان بدنه اي تعريف کرد و اين متد بايد توسط فرزند پياده سازي شود. يک نکته مهم در مورد متد هاي abstract آن است که حتما بايد داخل کلاس abstract تعريف شوند.

```
abstract class Amir
{
	public abstract void hello ();
```

```
}
```

interface

نوعي ماهيت همانند كلاس است با اين تفاوت كه متدهاي آن داراي بدنه نيستند، كلاسي كه از interface ارث مي‌برد بايد تمامي متدهاي آن را به طور كامل پياده‌سازي كند، در غير اين صورت جاوا توليد خطا مي‌كند. مثال :

```
interface Soheil
{
        public void hello();
}

class Amir implements Soheil
{
        public void hello()
        {
                System.out.print("salam");
        }
}
```

نكته: به ياد داشته باشيد كه براي ارث‌بري از interface از كلمه implements بايد استفاده كنيد نه extends !

نكته: نكته جالب ديگري كه در interface ها وجود دارد اين است كه يک كلاس مي‌تواند از چندين interface ارث برد ! مثال :

```
class Amir implements Soheil , Javad
{

}
```

كلاس و متغير final

كلاسي كه به صورت final تعريف شود به ساير كلاس‌ها اجازه ارث بري نمي‌دهد كلاس final دقيقاً بر عكس كلاس abstract عمل مي‌كند.

```
final class Amir
{
        public void hello()
        {
                System.out.print("Salam");
        }
}
```

حال اگر متغيري به صورت final تعريف شود جاوا اجازه تغيير مقدار متغير را به برنامه نويس نمي‌دهد به اين حالت " متغير ثابت" گفته مي‌شود.

```
final int a=2 ;
```

متد static

متدهاي static نمي‌توانند متدهاي غير static را بصورت مستقيم صدا كنند و حتماً بايد در اين متدها براي دسترسي به ساير متدهاي غير static از روش Object سازي استفاده كرد.

```
class Amir
{
	public void hello()
	{
		System.out.print("Salam");
	}
	public static void main (String [] arg)
	{
		Amir amir2 = new Amir();
		amir2.hello();
	}
}
```

استثنائات

گاهي اوقات در برنامه‌نويسي خطاهايي در زمان اجرا رخ مي‌دهد كه شما در توليد آن نقش مستقيم نداشته ايد. به طور مثال شما قصد خواندن فايلي را در برنامه داريد كه آن فايل قبلاً پاک شده است در آن صورت شما با خطاي استثنا رو به رو مي شويد. زماني كه شما از دستوراتي كه احتمال بوجود آوردن خطا را دارند استفاده مي‌كنيد جاوا معمولا به شما اجازه نمي‌دهد كه بدون مديريت خطا آن دستور را بنويسيد، براي مديريت اين نوع خطاها مي‌توانيد به ۲ روش عمل كنيد:

```
class amir
{
	public static void main (String [] arg)
	{
		try
		{
			System.out.print("True");
		}
		catch (Exception e)
		{
			System.out.print("Exception !!!!!!!!!!!!");
			System.out.print(e);
		}
	}
}
```

در اين روش شما در قسمت try دستورات اصلي خود را مي نويسيد، در صورتي كه اين دستورات باعث ايجاد خطا شود قسمت catch اجرا مي‌شود در اين قسمت ابتدا پيغام !!!!!!!!!Exception چاپ مي‌شود سپس در قسمت بعد متغير e چاپ مي‌شود اين متغير نوع خطا را مشخص مي‌كند، لازم به ذكر است كه قسمت catch را مي‌توانيد به دلخواه خود پياده‌سازي كنيد. در روش ديگر مديريت خطا وجود ندارد و زماني كه خطايي بوجود مي‌آيد فقط پيغام خطا چاپ مي‌شود.

```
class amir
{
	public static void main (String [] arg) throws Exception
	{
```

```
            System.out.print("True");
        }
}
```

اصول تعريف نام

براي نامگذاري نياز است گروهي از قواعد را رعايت كنيم مثلاً نامگذاري كلاس با نامگذاري متد فرق دارد، اين قواعد اجباري نيست اما بهتر است از اين قواعد تابعيت كرد. براي نامگذاري كلاس بايد حرف اول كلاس بزرگ باشد، و اگر كلاس كلمه ديگري نيز داشته باشد بايد حرف اول آن كلمه نيز بزرگ باشد. مثال :

class **A**mir

و يا

class **A**mir**B**ahador

قواعد اجباري {به ياد داشته باشيد كه در جاوا نام فايل جاوا با نام كلاس تعريف شده در فايل بايد يكي باشد (حتي از لحاظ بزرگي و كوچكي حروف)، در جاوا تنها يک كلاس مي توان در يک فايل تعريف كرد، مگر كلاس ثانويه به صورت دروني در كلاس اصلي تعريف گردد. پسوند فايل هاي جاوا قبل از ترجمه java. است و بعد از ترجمه فايلي با پسوند class. از آن ايجاد مي شود، نام پكيج مشخص شده در سورس كد با نام پوشه كه فايل در آنجا موجود است بايد يكي باشد.} براي نامگذاري متد بايد حرف اول كوچک باشد، و اگر متد كلمه ديگري نيز داشته باشد بايد حرف اول آن كلمه بزرگ باشد. مثال :

void **s**end()

و يا

void **s**end**E**mail()

و يا

void **s**end**E**mail**T**o**A**mir()

براي نامگذاري متغير بايد تمامي حروف متغير كوچک باشد و جدا سازي كلمات بوسيله "_" انجام مي‌شود. به عنوان مثال :

int age

ويا

int age_of_amir

براي نام گذاري package نيز بايد تمامي حروف كوچک باشد.

تبديل انواع داده به يكديگر

در بسياري موارد لازم است تا داده اي به نوع ديگر داده تبديل شود، فرض كنيد لازم است تا مقدار يک رشته با يک عدد جمع رياضي شود در اين شرايط مي توان به شكل ذيل عمل كرد.

```
String a=”12”;
int b = Integer.parseInt(a);
int c = b+2;
```

در اين شرايط مقدار رشته اي ۱۲ به مقدار عددي ۱۲ تبديل سپس با عدد ۲ جمع و حاصل درون متغير c ريخته خواهد شد. در مثال قبل داده رشته اي تبديل به داده عددي شده است، براي تبديل داده عددي به رشته اي كافيست تا داده ي عددي را با "" جمع ببنديم.

```
int i = 12;
String j = i+””;
```

متد هاي كلاس String

كلاس String داراي متد هاي گوناگوني است، در اين قسمت به بعضي از اين متد ها اشاره شده است.
متد trim: اين متد فاصله هاي ابتدا و انتهاي رشته را حذف و مقدار رشته اي را باز مي گرداند.
متد equals: اين متد يک پارامتر دريافت و بررسي مي كند كه آيا مقدار رشته با مقدار پارامتر ورودي برابر است يا خير (خروجي

این متد یا صحیح و یا نا صحیح است)
متد indexOf: این متد یک پارامتر ورودي دریافت سپس درون رشته مورد نظر بدنبال مقدار پارامتر ورودي گشته و جایگاه اولین موجودي را به صورت عددي باز مي گرداند.
متد length: این متد طول رشته را به صورت عددي باز مي گرداند.
متد lastIndexOf: این متد یک پارامتر ورودي دریافت سپس درون رشته مورد نظر بدنبال مقدار پارامتر ورودي گشته و جایگاه آخرین موجودي را به صورت عددي باز مي گرداند.
متد replaceAll: این متد دو پارامتر ورودي دریافت، سپس در رشته مورد نظر تمامي مقادیر برابر با پارامتر ورودي اول را به مقدار پارمتر ورودي دوم تغییر مي دهد.
متد substring: این متد دو پارامتر ورودي به شکل عددي دریافت، سپس رشته را نسبت به نقاط مشخص شده جدا کرده و مقدار را به صورت رشته اي باز مي گرداند.
متد هاي toUpperCase و toLowerCase: تمامي حروف رشته مورد نظر را به حروف بزرگ و یا کوچک تغییر خواهند داد.

Generic ها

در بسیاري از مواقع لازم است تا نوع خصوصیات یک کلاس به صورت داینامیک تغیر نماید در این صورت باید از مبحث Generic استفاده کرد، به مثال ذیل توجه کنید.

```
class Amirsam <noe> {
   noe data;
   public noe getData() {
      return data;
   }

   public void setData(noe data) {
      this.data = data;
   }

   public static void main(String[] args) {
      Amirsam <String> amirsam = new Amirsam<String>();
      amirsam.setData("12");
      System.out.println(amirsam.getData()+2);
   }
}
```

همانطور که حدث زده اید خروجي برنامه فوق ۱۲۲ است دلیل این امر آن است که در زمان شي سازي مقدار noe را String مشخص کرده ایم در این حالت نوع متغیر data از نوع String تعریف مي شود، حال به مثال ذیل توجه کنید:

```
class Amirsam <noe> {
   noe data;
   public noe getData() {
      return data;
   }

   public void setData(noe data) {
      this.data = data;
   }

   public static void main(String[] args) {
      Amirsam <Integer> amirsam = new Amirsam<Integer>();
      amirsam.setData(12);
      System.out.println(amirsam.getData()+2);
   }
}
```

همانطور که حدث زده اید خروجي برنامه فوق ۱۲۲ است دلیل این امر آن است که در زمان شي سازي مقدار noe را Integer مشخص کرده ایم در این حالت نوع متغیر data از نوع Integer تعریف مي شود.

کلاس Object

گهگاه نیاز است تا اشیا خود را درون یک شي عمومي ذخیره کنیم براي اینکار میتوان از کلاس Object استفاده کرد به مثال ذیل دقت کنید:

```
class Amirsam {
   public static void main(String[] args) {
      Object amir = new Amirsam();
   }
```

```
}
```
همانطور که مشاهده مي کنيد شي جديد را درون amir که از نوع Object است ذخيره کرده ايم، تمامي اشيا را در درون شي Object مي توان ذخيره کرد، حال به مثال ذيل دقت کنيد:

```
class Amirsam {
   public static void main(String[] args) {
      Object amir = new Amirsam();
      Amirsam x = (Amirsam) amir;
   }
}
```

همانطور که مشاهده مي کنيد شي amir که از نوع Object است را نمي توان به صورت مستقيم در درون x که از نوع Amirsam است ذخيره کنيم، براي اينکار بايد اول شي amir تبديل به نوع Amirsam شود، براي اينکار از دستور (Amirsam) بايد استفاده کنيم. به اين عمل casting گويند!

Annotation ها

گهگاه نياز است تا توضيحاتي در سطح برنامه نوشته شود و از اين توضيحات در روند اجراي برنامه استفاده نمود. براي اينکار مي توان از Annotation ها استفاده نمود.

```
@Retention(RetentionPolicy.RUNTIME)
@interface Check {
   public String checkerName() default "Nadarad";
   public boolean status() default false;
}
```

در اين حالت نياز است تا ما توضيحات را قاعده مند کنيم، در مثال بالا يک توضيح به نام Check ايجاد کرده ايم که اين چک داراي دو خصوصيت checkerName و status مي باشد مقدار پيش فرض اين خصوصيات به ترتيب Nadarad و false مي باشد.

```
@Check(checkerName = "RezaAmini", status = true)
class Amirsam {
   public static void main(String[] args) {

      Annotation[] annotations = Amirsam.class.getAnnotations();

      if (annotations[0].toString().equals("@test.controller.Check(status=true, checkerName=RezaAmini)"))
{
         System.out.println("in class tavasote aghaye RezaAmini check shode ast");


      } else {

         System.out.println("in class check nashode ast!");

      }
   }
}
```

در اين حالت خروجي برنامه in class tavasote aghaye RezaAmini check shode ast خواهد بود.

```
@Check(checkerName = "RezaAmini", status = false)
class Amirsam {
   public static void main(String[] args) {

      Annotation[] annotations = Amirsam.class.getAnnotations();

      if (annotations[0].toString().equals("@test.controller.Check(status=true, checkerName=RezaAmini)"))
{
         System.out.println("in class tavasote aghaye RezaAmini check shode ast");


      } else {
```

```
            System.out.println("in class check nashode ast!");

        }
    }
}
```

حال اگر مقدار status را همانند مثال بالا به false تغير دهيم خروجي برنامه in class check nashode ast خواهد بود.

ويرايش هاي زبان برنامه نويسي جاوا
حال كه با هسته اصلي اوليه زبان جاوا آشنا شديد بايد بدانيد كه اين زبان داراي ويرايش هاي متعددي مي باشد:

-ويرايش J2SE براي پياده سازي برنامه هاي كاربردي كنسولي و دسكتاپي

-ويرايش J2EE براي پياده سازي برنامه هاي سازماني و تحت وب/وپ

-ويرايش J2ME براي پياده سازي برنامه هاي كاربردي موبايل و ابزار هاي الكترونيكي

-ويرايش Android براي پياده سازي برنامه هاي كاربردي بر روي سيستم عامل آندرويد (البته اين ويرايش متعلق به شركت Google مي باشد)

-ويرايش JavaCard براي پياده سازي برنامه هاي كاربردي كارت هاي هوشمند

در حال حاضر شركت Oracle شركت Sun را خريداري كرده است پس براي كسب اطلاعات بيشتر به سايت شركت Oracle مراجعه فرمائيد.

موفق و پيروز باشيد
اميرسام بهادر
پاييز ۹۰

بيوگرافي نويسنده

اميرسام بهادر يكي از برنامه نويسان و مديران حرفه اي پروژه هاي انجام شده توسط زبان برنامه نويسي جاوا و بانک اطلاعاتي اوراكل مي باشد. ايشان تمايل زيادي به كار با زبان هاي برنامه نويسي جديد همانند Ruby، Go و ActionScript دارد. وي تا به حال چندين تكنولوژي و چارچوب كاري براي زبان برنامه نويسي جاوا و رابي توليد كرده است. Shine Framework يكي از بهترين توليدات ايشان است، اين چارچوب كاري رتبه اول سايت SourceForge را كسب كرده است.

بهادر يكي از صاحبنظران در زمينه هوش مصنوعي است. ايشان تا به حال چندين بار در شبكه هاي تلويزيوني مورد تقدير قرار گرفته است. يكي از افتخارات وي اين است كه در هيچ كدام از پروژه هاي كلاني كه مديريت آنها را عهده دار بوده حتي يک روز تاخير در تحويل پروژه نداشته است. بهادر كتاب هاي زيادي در مورد زبان هاي برنامه نويسي و بانک هاي اطلاعاتي نوشته است. از جمله مهمترين اين كتابها:

- مرجع جامع بانک اطلاعاتي اوراكل (مرجع دانشگاهي)
- مفاهيم بنيادي كار با تكنولوژي هاي جاوا

از جمله ديگر محصولات وي:

- توليد اولين سيستم عامل تحت وب
- روش جديدي جهت رمزنگاري برگشت نا پذير
- ايجاد يک تكنولوژي جهت ارتباطات زبان هاي برنامه نويسي با يكديگر
- روش جديدي جهت ذخيره سازي و بازيابي اطلاعات
- انسان مجازي

بهادر در حال حاضر مشغول تدريس در مركز محاسبات دانشگاه صنعتي شريف (LPI) مي باشد . براي كسب اطلاعات بيشتر به سايت http://Amirsam.J2OS.orgمراجعه فرمائيد.