پروژه درس

طراحی و پیاده سازی زبان های برنامه سازی

استاد :
مهندس جعفرنژاد قمی

تهیه و تنظیم :
سعید عمرانی
81141162

1384

فهرست :

آشنائی با زبان C#

مایکروسافت در مصاف با جاوا، بدنبال ارائه یك زبان كامل بود كه سایه جاوا را در میادین برنامه نویسی كم رنگ تر نماید.

شاید بهمین دلیل باشد که C# را ایجاد کرد. شباهت های بین دو زبان بسیار چشمگیر است. مایکروسافت در رابطه با میزان استفاده و گسترش زبان فوق بسیار خوشبین بوده و امیدوار است بسرعت زبان فوق گستردگی و مقبولیتی به مراتب بیشتر از جاوا را نزد پیاده کنندگان نرم افزار پیدا کند.

با توجه به نقش محوری این زبان، از آن بعنوان مادر زبانهای برنامه نویسی در دات نت نام برده می شود. مورد فوق به تنهائی، می تواند دلیل قانع کننده ای برای یادگیری این زبان باشد، ولی دلایل متعدد دیگری نیز وجود دارد که در ادامه به برخی از آنها اشاره می گردد.

مطرح شدن بعنوان یك استاندارد صنعتی

انجمن تولیدکنندگان کامپیوتر اروپا (ECMA) زبان C# را در سوم اکتبر سال 2001 بعنوان یك استاندارد پذیرفته (ECMA-334) و بدنبال آن تلاش های وسیعی برای کسب گواهی ISO نیز انجام شده است. زبان فوق در ابتدا توسط شرکت مایکروسافت و بعنوان بخشی از دات نت پیاده سازی و بلافاصله پس از آن توسط شرکت های اینتل، هیولیت پاکارد و مایکروسافت مشترکا، جهت استاندارسازی پیشنهاد گردید.

زبان C# بگونه ای طراحی شده است که نه تنها وابستگی به یك Platform خاص را ندارد، بلکه در اغلب موارد وابستگی RunTime نیز ندارد. کامپایلر C# می تواند بر روی هر نوع معماری سخت افزاری طراحی و اجرا گردد. در برخی از نسخه های اولیه کامپایلر زبان فوق که توسط برخی از شرکت های جانبی ارائه شده است، کدهای C# را به بایت کدهای جاوا کمپایل می کنند. یکی از چنین کامپایلرهائی را می توان در سایت Halcyonsoft.com مشاهده نمود. بنابراین کدهای C# براحتی قابلیت حمل بر روی محیط های متفاوت را دارا خواهند بود.

مشخصات تعریف شده زبان C# با سایر استانداردهای تعریف شده ECMA نظیر CLI (Common Language Infrastructure) (ECMA-335) بخوبی مطابقت می نمایند. CLI قلب و روح دات نت و CLR(Common Language Runtime) است. اولین نسخه از کامپایلر زبان C# که از CLI استفاده می کند، NET Framwork. مایکروسافت است.

با توجه به موارد گفته شده، مشخص می گردد که این زبان بسرعت بسمت استاندارد شدن حرکت و با تایید استانداردهای مربوطه از طرف انجمن های معتبر بین المللی و حمایت فراگیر شرکت های معتبر کامپیوتری در دنیا مسیر خود را بسمت جهانی شدن بخوبی طی می نماید.

C# چیست ؟

طراحان زبان C# با تاکید و الگوبرداری مناسب از مزایای زبانهائی نظیر C++، C و جاوا و نادیده گرفتن برخی از امکانات تامل برانگیز و کم استفاده شده در هر یك از زبانهای فوق، یك

زبان برنامه نویسی مدرن شی گراء را طراحی کرده اند. در مواردی، برخی از ویژگی های استفاده نشده و درست درك نشده در هر یك از زبانهای گفته شده، حذف و یا با اعمال کنترل های لازم بر روی آنها، زمینه ایجاد یك زبان آسان و ایمن برای اغلب پیاده کنندگان نرم افزار بوجود آمده است. مثلا C و C++ می توانند مستقیما با استفاده از اشاره گرها عملیات دلخواه خود را در حافظه انجام دهند. وجود توانائی فوق برای نوشتن برنامه های کامپیوتری با کارائی بالا ضرورت اساسی دارد. اما در صورتیکه عملیاتی اینچنین بدرستی کنترل و هدایت نگردند، خود می تواند باعث بروز مسائل (Bugs) بیشماری گردد.

طراحان زبان C#، با درك اهمیت موضوع فوق، این ویژگی را کماکان در آن گنجانده ولی بمنظور ممانعت از استفاده نادرست و ایجاد اطمینان های لازم مسئله حفاظت نیز مورد توجه قرار گرفته است. جهت استفاده از ویژگی فوق، برنامه نویسان می بایست با صراحت و به روشنی خواسته خود را از طریق استفاده از Keyword های مربوطهاعلان نمایند(فراخوانی یك توانائی و استفاده از آن).

C# بعنوان یك زبان شی گراء عالی است. این زبان First-Class را برای مفهوم Property (Data Member) بهمراه سایر خصائص عمومی برنامه نویسی شی گراء حمایت می کند. در C و C++ و جاوا یك متد get/set اغلب برای دستیابی به ویژگی های هر Property استفاده می گردد. CLI همچنان تعریف Property را به متدهای get/ser ترجمه کرده تا بدین طریق بتواند دارای حداکثر ارتباط متقابل با سایر زبانهای برنامه نویسی باشد. C# بصورت فطری Events، Declared Value، Reference Type، Overloading Operator را نیز حمایت می کند.

کد مدیریت یافته

با استفاده از نسخه پیاده سازی شده C# توسط مایکروسافت، می توان همواره کد مدیریت یافته ای را تولید کرد. یك برنامه C# پس از کامپایل، بصورت برنامه ای در خواهد آمد که شامل دستورالعمل های تلفیق شده (CIL (Common Intermediate Language است (درست بر خلاف دستورالعمل های مختص یك ماشین خاص). CIL (گاها با نام MSIL(Microsoft Intermediate Language) با به اختصار IL نیز نامیده می شود) ، در مفهوم مشابه بایت کدهای جاوا بوده و شامل مجموعه ای از دستورالعمل های سطح پایین قابل فهم توسط تکنولوژی مبتنی بر CLI نظیر CLR مایکروسافت خواهد بود. این برنامه ها بدین دلیل کد مدیریت یافته، نامیده می شوند که CLR مسئولیت تبدیل این دستورالعمل ها به کدهای قابل اجرا برروی ماشین و ارائه اغلب سرویس های اساسی برای کدینگ نظیر : Garbage Collection، مدیریت Heap و عمر مفید یك Object و یا Type Verification را فراهم می کند.

روش یادگیری C#

یادگیری این زبان برای افرادیکه دارای سابقه آشنائی با یکی از زبانهای برنامه نویسی C++، C و یا جاوا باشند کار مشکلی نخواهد بود، حتی افرادیکه دارای آشنائی اولیه با جاوااسکریپت و یا دیگر زبانهای برنامه نویسی نظیر ویژوال بیسك می باشند، امکان پذیر و راحت خواهد بود. برخی از برنامه نویسان حرفه ای بر این باور هستند که C# نسبت به VB.NET با اقبال بیشتر و سریعتری مواجه خواهد شد، چراکه C# نسبت به ویژوال بیسك خلاصه تر است. حتی برنامه های بزرگ و پیچیده ای که توسط C# نوشته می گردند خواناتر، کوتاه و زیبا خواهند بود. برخی از ویژگی های ارائه شده در C# نظیر Unsigned Integer، Operator OverLoading و امنیت بیشتر Type ها، در VB.NET وجود نداشته و این امر می تواند دلیلی بر فراگیرتر شدن C# نسبت به VB.NET نزد برنامه نویسان با تجربه باشد.

برای یادگیری هر یك از زبانهای حمایت شده در دات نت، می بایست از BCL (Basic Class Library) مربوط به .NET Framework شروع کرد. C# خود صرفا دارای ۷۷ کلمه کلیدی یا Keyword بوده که برای اکثر برنامه نویسان غریب نخواهند بود. در مقابل BCL، دارای ۴۵۰۰ کلاس و تعداد بیشماری متد و Property است که برنامه نویسان C#، می توانند از آنها برای انجام عملیات دلخواه خود استفاده نمایند. شاید یکی از مسائل قابل توجه جهت یادگیری این زبان برای برخی از برنامه نویسان حرفه ای عدم وجود برخی از ویژگی ها و امکاناتی باشد که در گذشته و از طریق سایر زبانهای استفاده شده، بخدمت گرفته می شدند. مثلا عدم وجود امکاناتی جهت توارث چندگانه (MI) سلسله مراتبی یك شئ.

شئ گرایی در C#

دنیای برنامه نویسی امروزه بر مفهوم شئ گرایی استوار است. شئ گرایی به زبان ساده یعنی دنیا را آنطور ببنیم که هست! زبان C#یکی از بهترین زبانهایی است که دارای تمامی امکانات جهت پیاده سازی مفاهیم اصیل شئ گرایی است.

شئ گرایی (OOP) در C# بر چند پایه استوار است که به قرار زیرند:

- Inheritance
- Encapsulation
- Polymorphism
- Abstraction
- Interface

اکنون به توضیح مختصر هر یك می پردازیم.

Inheritance (ارث بری)

پدر و فرزندی را در نظر بگیرید. هر پدری مشخصات فردی به خصوصی دارد. فرزند وی می تواند همه خصوصیات او را به ارث برد و خصوصیتهای دیگری نیز داشته باشد که پدرش ندارد. این یعنی

ارث بری! برای مثال پدر وقتی عصبانی می شود، داد و فریاد می کند. پسر هم این خصوصیت را به ارث می برد با این تفاوت که وقتی عصبانی می شود، علاوه بر داد زدن، چند عدد بشقاب هم می شکند. در برنامه نویسی شئ گرا از مفهوم ارث بری استفاده های زیادی می شود. برای تفهیم راحت تر مسئله فرض کنید کلاسی به نام وسیله نقلیه داریم. از آنجا که هر وسیله نقلیه ای حرکت می کند، رنگ دارد، سرعت دارد، ترمز می گیرد و... می توانیم همه این متدها و فیلدها (کدام متدها و فیلدها!؟) را در کلاس وسیله نقلیه تعریف کنیم. حال یک وهله از این کلاس را در نظر بگیرید (مثلا دوچرخه!). یک دوچرخه یک وسیله نقلیه است که همه خصوصیات عمومی یک وسیله نقلیه را دارد و البته خصوصیاتی دارد که مختص خودش هستند و در انواع دیگر یافت نمی شوند. به این منظور این دوچرخه می تواند ویژگیها و متدهای مشترک را از کلاس وسیله نقلیه به ارث ببرد و در عین حال ویژگیهای منحصر به خود را نیز داشته باشد. قابلیت استفاده دوباره از کد (Reusability) یکی از مزیات اصلی ارث بری است.

Encapsulation

همانطور که از اسمش پیداست، به قرار دادن پیاده سازی در یک کپسول اشاره می کند، به طوری که کاربر بیرونی از نحوه پیاده سازی مطلع نباشد و فقط بداند که این کپسول کار خاصی را انجام می دهد. وقتی یك كپسول می خورید نمی دانید که در داخل آن چه چیزی هست و فقط به این فکر می کنید که این کپسول چه تاثیری در بدن شما می گذارد!

فرض کنید سوار ماشینی هستید که به سرعت در حرکت است! در مسیری که می روید ماشین پدر نامزدتان از روبرو به شما نزدیک می شود و سعی می کنید سریع ترمز بگیرید تا برخورد نکنید. اگر قرار باشد که بدانید بعد از فشار دادن پدال ترمز چه عملیاتی انجام می شود تا ترمز گرفته شود، دیگر باید از ازدواج قطع امید کنید. ولی اگر تنها بدانید که با فشار دادن پدال، ترمز گرفته می شود شما خوشبخت خواهید شد. در واقع ما در اینجا کار ترمز گرفتن ماشین را به صورت یک کپسول آماده در نظر می گیریم. هدف Encapsulation این است که ما را از پرداختن به ریز موضوعات رها کند و اشیاء را به صورت یک جعبه سیاهی بدانیم که به ازای یک ورودی خاص خروجی خاصی می دهند. اگر می خواهیم کدهای ما نیز این مورد را رعایت کنند باید سعی کنیم نگاه کپسولی به اشیاء و عملکرد آنها داشته باشیم. در C# برای کپسوله کردن از Access Modifierهای protected، private و public استفاده می شود.

Polymorphism

فرض کنید پدر شما کار خاصی را به طریق خاصی انجام می دهد. مثلا برای پختن غذا (حقیقتی است تلخ!) اول ظرفهای دیشب را شسته و بعد گاز را روشن می کند و بعد غذا می پزد! شما که خصوصیات پدر و کارهای او را به ارث می برید برای مثال برای پختن غذا

ابتدا گاز را روشن می کنید، بعد کبریت می کشید، غذا را می پزید و بعد ظرفهای دیشب را می شویید! (توصیه می کنم نگذارید ظرفهایتان نشسته بمانند!) برادر شما ممکن است همین کار را به طریق دیگری انجام دهد. پختن غذا کاری است که شما از پدر خود به ارث می برید!!! ولی آن را به طریق دیگری انجام می دهید. یعنی یک کار ثابت توسط فرزندان مختلف یک پدر به طرق مختلفی انجام می شود. این دقیقا همان چیزی است که به آن چند شکلی یا Polymorphism می گویند.

Abstraction

تجرید یا مجرد سازی! به کلاسی مجرد گفته می شود که پیاده سازی متدها در آن انجام نمی شود! بر خلاف انسانها که مجرد تعریف دیگری برایشان دارد! حال سئوالی پیش می آید که اگر کلاسی داشته باشیم که نخواهیم پیاده سازی متدها را در آن انجام بدیم، از آن کلاس چه استفاده ای می کنیم؟ برای پاسخ به این سئوال شرایط زیر را در نظر بگیرید:

فرض می کنیم که شما رییس یک شرکت بزرگ برنامه نویسی هستید و می خواهید پروژه بزرگی را انجام دهید. برای اجرای پروژه از برنامه نویسان مختلفی استفاده می کنید که ممکن است همه آنها هموطن نباشند! مثلا هندی، ایرانی یا آلمانی باشند! اگر قرار باشد هر برنامه نویسی در نامگذاری متدها و کلاسهایش آزاد باشد، در کد نویسی هرج و مرج به وجود می آید. شما به عنوان مدیر پروژه، کلاسی تعریف می کنید که در آن تمام متدها با ورودی و خروجی هایشان مشخص باشند. ولی این متدها را پیاده سازی نمی کنید و کار پیاده سازی را به برنامه نویسان می دهید و از آنها می خواهید که همه کلاسهایی را که می نویسند از این کلاس شما به ارث ببرند و متدها را به طور دلخواه پیاده سازی کنند. این باعث می شود که شما با داشتن یک کلاس، ورودی و خروجی های مد نظر خود را داشته باشید و دیگر نگران برنامه نویسان نباشید. کلاسی که شما تعریف می کنید یک کلاس مجرد نامیده می شود.

برای تعریف یک کلاس مجرد از کلمه کلیدی abstract استفاده می کنیم. فیلدهایی که می خواهیم در کلاسهای مشتق شده از این کلاس پیاده سازی شوند حتما باید با abstract تعریف شوند. یک کلاس مجرد می تواند فیلدها و متدهای ناجرد داشته باشد. اگر متد ناجردی در یک کلاس مجرد تعریف کردید، حتما باید آن را پیاده سازی کنید و نمی توانید پیاده سازی آن را به کلاسهای مشتق شده بسپارید.

Interface

اینترفیس در برنامه نویسی همانند همان کلاس است تنها با این تفاوت که هیچکدام از اعضای آن پیاده سازی نمی شوند. در واقع یک اینترفیس گروهی از متدها، خصوصیات، رویدادها و Indexer ها هستند که در کنار هم جمع شده اند. اینترفیس ها را نمی توان Instantiate (وهله سازی) کرد (یعنی نمی توان وهله ای از یک اینترفیس ایجاد کرد!). تنها چیزی که یک اینترفیس دارا می باشد

امضای (signature) تمامی اعضای آن می باشد. به این معنی که ورودی و خروجی متدها، نوع Property ها و... در آن تعریف می شوند ولی چیزی پیاده سازی نمی شود. اینترفیس ها سازنده و فیلد ندارند (امری است بدیهی! چرا؟). یک اینترفیس نمی تواند Operator Overload داشته باشد و دلیل آن این است که در صورت وجود این ویژگی، احتمال بروز مشکلاتی از قبیل ناسازگاری با دیگر زبانهای .NET مانند VB.NET که از این قابلیت پشتیبانی نمی کند وجود داشت. نحوه تعریف اینترفیس بسیار شبیه تعریف کلاس است تنها با این تفاوت که در اینترفیس پیاده سازی وجود ندارد.

حالا این اینترفیس در کجا به کار می آید؟ اگر با C++ کار کرده باشید (در آن صورت کارتان خیلی درست می باشد!!!) با واژه ارث بری چند گانه آشنا هستید. ولی احتمالا شنیدید که جاوا و C# از ارث بری چندگانه پشتیبانی نمی کنند. (یعنی یک کلاس از چند کلاس دیگر به ارث ببرد). گاهی لازم داریم از چند کلاس به ارث ببریم. راه حلش این است که از اینترفیس ها استفاده کنیم. ولی بدانید که اگر از اینترفیسی به ارث بردید باید تمام متدهای آن را پیاده سازی کنید. یک کلاس می تواند از n تا اینترفیس و تنها یک کلاس به ارث ببرد.

آشنایی با زبان PHP

PHP و برنامه نویسی شی گرا

PHPتا چه حد شی گراست؟! آیا تمام امکاناتی که در زبانهای برنامه نویسی شی گرای کاملی همچون Java وجود داره، در PHP هم پشتیبانی میشه؟! پاسخ به این سوال تا حدی بر میگرده به معیارها و شرایطی که شما برای شی گرا بودن یک زبان مدنظر دارید. در این پست من سعی میکنم امکاناتی که نوعا در یک زبان برنامه نویسی شی گرا وجود داره رو بررسی کنم و حرفایی که PHP در هر مورد برای گفتن داره رو تا حدی توضیح بدم. (این راهنما

بیشتر برای کسانی هست که از زبانهای OO دیگری میخوان به PHP روی بیارن

وراثت یگانه:(Single Inheritance)

PHPبه شما اجازه میده که با استفاده از عبارت extends یک کلاس فرزند تعریف کنید که تمام مشخصه ها و رفتارهای کلاس والد رو دارا هست.(تعریف یک کلاس، از کلاس دیگه به ارث می رسه.

وراثت چندگانه:(Multiple Inheritance)

PHPاز وراثت چندگانه پشتیبانی به عمل نمی یاره و هیچ نشانی از وراثت واسطه که در جاوا وجود داره دیده نمیشه. هر کلاس حداکثر یک کلاس والد خواهد داشت.

سازنده ها:(Constructors)

هر کلاس می تونه یک تابع سازنده داشته باشه که در نسخه فعلی PHPکه از Zend Engine 1 بهره می بره، باید همنام کلاس باشه. در نسخه بعدی PHP که مجهز به ZE2 می باشد، تابع سازنده هر کلاس __construct()نام خواهد گرفت. تابع سازنده کلاسهای والد به طور اتوماتیک فراخوانی نمی شن مگر اینکه صریحا احضار بشن!

تخریب کننده ها:(Destractors)

نسخه فعلی) PHP با (ZE 1 تابع تخریب کننده نداره، بیشتر به این خاطر هست که آزاد سازی حافظه و برگرداندن حافظه تخصیص یافته دست برنامه نویس نیست. اما نسخه های بعدی تابع تخریب کننده را دارا می باشد.

کپسوله سازی و کنترل دسترسی:(Encapsulation)

هیچ پشتیبانی از کپسوله سازی در نسخه جاری PHP به عمل نیومده و تمام مشخصه ها و رفتارها Public هستند، اما تو نسخه بعدی دو نوع Private و Protected افزوده شده است.

چند شکلی:(Polymorphism)

PHPچند شکلی رو به این صورت پشتیبانی میکنه که اجازه میده نمونه کلاس های فرزند به جای نمونه کلاسهای والد استفاده بشه.

اتصال دیر یا زود(Early vs. late binding)

دو پاسخ مناسب به این موضوع به قرار ذیل هست:

۱ (از اونجایی که PHP یک زبان Loosely Type هست، این سوال پیش نمی یاد.

۲ (تمام اتصالات Late هستند. در PHP مقادیر دارای نوع هستند ولی متغیرها بی نوع هستند، بنابراین این سوال که اگه نوع متغیر و مقدار متفات باشه کدوم متد فراخوانی بشه، به وجود نمی یاد.

توابع ایستا(Static Functions)

پشتیبانی صریحی ازClass Function ها وجود نداره اما میشه که با استفاده از سینتکس Classname::function() میشود تابع یک کلاس رو فراخوانی کرد. این تابع تا وقتیکه به یک متغیر داخلی ارجاع نکنه ، به عنوان یک Class Function در نظر گرفته میشه.

درون نگری:(Introspection)

در این مورد PHP با دست پر ظاهر شده و توابع متنوعی برای کسب اطلاعات در مورد یک کلاس وجود داره، مثلا بازیابی نام کلاس، نام یک تابع، نام متغیرهای(مشخصه های) یک نمونه کلاس و...

اغلب از مفهوم شئ گرايي در زبان مورد علاقه ما چشم پوشي مي شود يا به غلط تفسير مي شود ولي اگر به طور صحيح بكار گرفته مي شود خيلي هم قوي به نظر مي رسد .آينده PHP درباره شئ گرايي بسيار روشن است اين از خصوصيات جديدی که در PHP5 گنجانده شده کاملاً مشخص است. با ابزار مناسب تنها چيزی که لازم داريم دانش است. در اين مقاله سعی شده که مفهوم واقعی شئ را بررسی کنيم و اينکه چه جوری اونها را شناسايی کنيم همچنين سعی شده ما را با سه رکن اصلی OOP يعنی کپسوله کردن، ارث بری و پلی مورفيسم بيشتر آشنا کند.
برنامه نويسی شئ گرا همانطور که از اسمش هم پيداست برنامه نويسی با اشياء است. ولی خوب معنی دقيق شئ چيه؟

بذاريد اول بگم شئ چی نيست! شئ فقط يه کلاس نيست که يه مشت تابع داخلش ريخته باشن. شايد به نظر بديهی بياد ولی واقعيت اينه که وقتی آدم مقوله OOP رو در يه زبان رويه ای مثل PHP کشف می کنه هيجان زده ميشه که زودتر استفادش کنه بدون اينکه از تئوری اوليه اون خبر داشته باشه!

حالا شئ چيه؟

شئ موجوديتی است که خصوصيات (properties)و رفتارهايی را (behavior) در خودش کپسوله می کند که مخصوص همان موجوديت است.

شايد به نظر عجيب بياد ولی اشياء دور و ور ما هستند. بعضی خودشون از اشياء ديگری تشکيل شدن. روزنامه دستتون، پنجره اتاق خود اتاق در ديوار...

ولی درباره OOP بعنوان یک مفهوم چیزای زیادی هست که باید بدونیم. در طول این سالها (کدوم سالها؟!) متدلوژیهای مختلفی برای نزدیک شدن به شئ گرایی توسعه یافتند، حتی می شه به سه مرحله تقسیمش کرد :تحلیل شئ گرا(OOA)، طراحی شئ گرا (OOD)و برنامه نویسی شئ گرا (OOP)که برای تبحر در هر کدومشون وقت زیادی لازم است مثلا مدلهای طراحی هنوز بدجوری رویه ای هستند. می دونید فقط syntax نیست که مهم هست در واقع قسمت سخت ماجرا شئ گرایانه فکر کردن و تمرین کردن و در نهایت بکارگیری هست. نگذارید این واقعیات شما رو دلسرد کنه چون یادگیری مفاهیم قسمت اعظم موضوعه (که لابد تو این مقاله قراره یاد بگیریم(شئ گرایی مزایای زیادی داره از جمله: استفاده دوباره، توسعه پذیری و نگهداشت پذیری که مهمترین اونها هستن.

استفاده دوباره :اشیاء می توانند رو پای خوشون بایستند یعنی مجردند؛ و نشاندهنده یک چیز هستند. به این معنی که می توانند به گونه های مختلف ترکیب شوند، که این همون خاصیت استفاده دوباره را ایجاد میکند. استفاده دوباره از اشیاء کلی توی وقت ما صرفه جویی میکنه چون مجبور نیستیم همه چیز و از اول بسازیم .توسعه پذیری: بجای نوشتن یک شئ از اول ما می تونیم یک شئ را گسترش بدیم. یه شئ می تواند از یک شئ دیگر مشتق شود و فقط کارایی هایی را که لازم است به شئ جدید بیفزاییم.

نگهداشت پذیری: طبیعتاً اشیاء چون خوانایی بالایی دارند، خیلی راحتتر تحلیل میشوند و خیلی بهتر میتوان از برنامه های موجود توسعه اشون داد و اینکه طبیعت "pluggable" دارند کد کمتری برای ویرایش اونها لازم است.

تصور غلط درباره بکارگیری OOP درPHP

قبل از ادامه بحث دوست دارم یه مقدار درباره این موضوع که PHP 4شئ گرایی را پشتیبانی نمی کند صحبت کنم. بریم سر اصل مطلب:

فقدان وجود کنترلهای دستیابی: گرچه این در php5 تغییر میکنه (private, public و protected اضافه خواهند شد) به عقیده بنده این پارامتری نیست که جلوی ما را در بکارگیری OOP بگیره، کنترلهای دستیابی به هیچ وجه به منظور ایجاد امنیت بوجود نیامدند، بلکه برای کمک کردن به برنامه نویس درست شدند که برنامه نویس لازم نباشه نگران باشه که تصادفی به اعضایی دسترسی پیدا شود که نباید دست بخورند .فقط باید حواسمون رو بیشتر جمع کنیم. اینکه مفسر برنامه این اجازه را از شما نمی گیره به این معنا نیست که شما تئوری oop رو نمی تونید بکار بگیرید.

فقدان کلاسهای مجرد: بازهم این نمی تونه شما رو از کدنویسی شئ گرا باز نگه داره حتی در تئوری! اساساً یک کلاس مجرد به این معناست که شما باید در هنگام ایجاد زیرکلاسها متدهای مجرد پیاده سازی کنید، و قادر نباشید از آن instance بسازید (خدا وکیلی اینو خودم هم نفهمیدم چی گفت). بعضی از زبانها مثل جاوا بصورت built-in این موضوع را پشتیبانی می کنند که خیلی خوبه ولی بدون اون

هم زندگی غیر ممکن نیست! بسادگی میشه یک متد خالی در superclass رو در زیر کلاسها override کنید. (که در قسمت پلی مورفیسم نشان خواهم داد چگونه).

عدم پشتیبانی از ارث بری چندگانه: گرچه بعضی اعتقاد دارند که ارث بری چندگانه مفید هست ولی به عقیده من فقط ایجاد ابهام میکند. شما چندتا مثال سراغ دارین که در آن یک شئ از دو یا چند superclass ارث ببره و هنوز بتونه رابطه is-a را حفظ کنه (اگر معنی این رابطه را نمی دانید من پایینتر توضیح دادم.(گرچه عقاید متفاوتند ولی من فکر نمی کنم این محدودیتها کاملا قابل چشم پوشی هستند و اصلاً دلیل خوبی برای ترک شئ گرایی در php نیستند.

درک اشیاء وقتی داریم تکنیک های جدید برنامه نویسی را یاد می گیریم، معمولاً خوبه که از نمونه های دنیای واقعی کمک بگیریم. اینجا می تونیم یک ماشین را در نظر بگیریم. فکر کنید یک ماشین چه چیزهایی دارد (properties)و چه کارهایی می تواند انجام دهد. (behaviour)چیزهایی که ماشین دارد ((properties :پنجره در چرخ موتور در اینجا لازم به ذکر است که properties خودشون می توانند شئ باشند با properties و behaviour خودشان. در شئ گرایی این ترکیب نام دارد. شما می توانید بگویید که ماشین تشکیل شده از... و حتی پا را فراتر بگذارید و بگویید که هر شئ هم از اشیاء دیگر تشکیل شده و .... ولی در موقع برنامه نویسی سعی کنید فقط چیزهایی را تعریف کنید که بدرد میخورند و بیخودی وارد جزئیات نشوید.

خوب بریم ببینیم یک ماشین چه کار می تواند بکند:

شتاب بگیرد

ترمز کند

درها باز شوند

این رفتارها مخصوص خود ماشین هستند. شما باید قادر به تشخیص اشیاء در واقعیت باشید تا بتوانید خوب از معماری شئ گرا استفاده کنید واقعاً احمقانه نیست که در زندگی عادی شئ گرا فکر کنیم .کل عالم از اشیاء تشکیل شده، سعی کنید اشیاء را شناسایی کنید!

رکن اول: کپسوله کردن

از اینجا به بعد باید از یک زاویه دیگه به قضیه نگاه کنیم. آیا یک ماشین می تواند خودش براند؟ البته که نه، وقتی شئ گرا نگاه کنیم خود راننده یک موجودیت جداست.

خیلی مهم است که این تفاوت را حس کنیم: اشیاء باید مسوولیت های خودشان را داشته باشند .اونها فقط باید قادر باشند کارهای خودشان را انجام بدهند و نه چیزی بیشتردر واقع property های یک شئ فقط باید تحت تأثیر رفتارهای همان شئ تغییر کنند. اینکه یک شئ مستقیمأ بتواند روی property های شئ دیگر تأثیر بگذارد بکلی غلط است. یک شئ باید جزئیاتش را برای خودش حفظ کند و آنچه که بروز می دهد فقط interface اش باشد و یک شئ دیگر فقط به جزئیات آن شئ از طریق اینinterface می تواند دسترسی داشته باشد. وقتشه که این تئوری را ازطریق یک کد بازگو کنیم: یک کارخانه اتومبیل می خواهد اتومبیل بسازد ولی فقط در سه رنگ:

قرمز، آبی و سبز. برای اینکه بدونید مسوولیت یعنی چی من اعتراف می کنم مثال اولم مثال خوبی نبود! در ضمنsyntax برنامه ربطی به من نداره می تونید php manual را تحت عنوان Class/object functionsمطالعه کنید.

```
<?php
        class Car
        {
        var $color;
        }

        $specificCar = new Car();
        $specificCar->color = "yellow\";
        ?>
```

اشتباه! قانون ما این بود که هیچ رنگی غیر از قرمز و آبی و سبز ساخته نشود. خوب معلومه آخر عاقبت اداره کردن مستقیم colour propertyهمینه دیگه! اتومبیل موند رو دستمون .چون ما یک interfaceبرای مراقبت از مسوولیت ها ایجاد نکردیم. در لیست 1 مثال بهتری را خواهید یافت.

لیست 1

```
<?php
        class Car
        {
        var $color;
        var $possibleColors = array("red\", \"green\", \"blue\");

        function setColor($color)
        {
        if (in_array($color, $this->possibleColors)) {
        $this->color = $color;
        }
        }
        }

        $specificCar = new Car();
        $specificCar->setColor(\"yellow\"); //would not alter the
object

        $specificCar->setColor(\"red\"); //would alter the object
        ?>
```

سعی کنید که این کد را درک کنید و اینکه چرا این راه بهتر است، این بار اتومبیل ما یک interface دارد که ما از طریق این interface می توانیم مقادیر properties را تغییر دهیم، متد setColor مراقب هست که چه اتفاقاتی ممکن است برای properties اتومبیل بیفتد. اگر interface امان را خوب بنویسیم، می توان مطمئن بود که اگر به غلط هم مقداردهی شود اتفاق بدی برای properties نخواهد بود و interface مراقب اوضاع خواهد بود. متد setColor اصلاح کننده (modifier)نام دارد و propertyشئ اتومبیل را اصلاح می کند. در کنار اصلاح کننده، accessorها را داریم accessor .برای برگرداندن property بجای اصلاح آن بکار می رود. لیست 2 مثالی برای accessor را نشان می دهد.

لیست 2

```
<?php
class Car
{
var $color;

/*
* constructor, code inside this function is executed on object
* initialisation
* note that in PHP 5 the constructor will have to have the name
* __construct(), instead of the class name
* although this way of constructing will still work as long as no
*___construct() is found
*/
function Car()
{
$this->color = "red\";
}

// Accessor
function getColor()
{
return $this->color;
}
}
?>
```

در لیست 2 متد getColor همان accessor است accessor .ها همچنین می توانند داده را قبل از برگرداندن اداره کنند. همیشه بهتر است که بجای ارجاع مستقیم به property های داخلی یک شئ از accessor ها استفاده کنیم. حالا می دونم بعضی از شما می پرسید که "من واقعاً نمی تونم property های شئ را مستقیماً اداره کنم؟" درسته. وقتی این مقاله نوشته می شد php هنوز فاقد اصلاح کننده هایی مثل private بود. هرچند این موضوع در php5 تغییر خواهد کرد شما می توانید محدودیت های سر سختی برای قسمت های خصوصی خودتان بگذارید تا خیالتان راحت باشد که بهیچ وجه دستکاری نمی شوند .خیلی مهم است که رفتار شئ شما بخوبی از طریق interface یا متدها تعریف شده باشد. چون خیلی مهم است که یک interface خوب در موقع استفاده دوباره هم خوب باشد، باید واضح باشد که یک شئ چکار می کند. همیشه سعی کنید اسامی برای کلاسها انتخاب کنید که کار اون کلاس را بطور خلاصه بیان کند. از نوشتن اسامی بلند نترسید در مجموع کپسوله کردن یعنی:

اشیاء جزئیات را برای خودشون نگه دارند

یک شئ فقط باید چیزی را ارائه کند که لازم است

یک شئ مسوولیت های خودش را دارئ

یک شئ باید interface واضحی داشته باشد

کلاسها درPHP

یکی از مسائلی که در PHP فهمیدن آن خیلی راحت نیست "مزیت استفاده از کلاسها"
"در برنامه نویسی می باشد . من قبلا با هیچ بانک اطلاعاتی در PHPکار نکرده
بودم اما بعد از کمی تلاش دیدم که یادگیری دستورات مقدماتی آن چقدر راحت است .
اما در مقابل برنامه نویسی) OOP برنامه نویسی شیء گرا) نیز تا بحال انجام نداده
بودم ولی پی بردم که فهمیدن مفاهیم و چرائی استفاده از آن بسیار مشکل بود . من
فکر می کردم که برنامه نویسی شیء گرا حتما خیلی قدرتمند است و مزایای زیادی دارد
اما از اونجائی که سعی می کردم قبل از تجربه هر چیز خودم را قانع کنم که اون چیز
به دردم می خوره هیچ موقع دنبال اون نبودم.

تا اینکه چند روز پیش هنگامی که داشتم یک Function ساده می نوشتم و اون رو تغییرمی دادم به ذهنم رسید که ببینم می شود این کار را توسط یک) Object شیء) انجام داد یا نه . سعی می کنم نتایجی روکه در هنگام این کار بهش رسیدم رو براتون به زبان ساده شرح بدم.

) Classکلاس) در حقیقت تعدادی از) Variable متغیر) ها به همراه تعدادی) Functionعملگر) است که این عملگر ها بر روی آن متغیر ها کار می کنند . اونها در حقیقت مفهوم یک چیز در دنیای واقعی هستند. به عبارت دیگر کلاسها یک شیء رو معرفی می کنند . نشانه یک کلاس زندگی واقعی و یا نفس کشیدن است که زیرساختهای اون کلاس هستند .

فرض کنید که ما می خواهیم یک دوچرخه را شرح دهیم . کلاس مرتبط با یک دوچرخه می بایست متغیر های زیر را داشته باشد:

```
$pedals, $chain, $front_wheel, $rear_wheel, $brakes, $handle_bars
```

(بدنه نگهدارنده ، ترمزها ، چرخ عقب ،چرخ جلو ، زنجیر ، پدالها)

عملگر هایی که یک دوچرخه دارد از قرار زیر است:

```
Stop(), Accelerate(), Coast(), TurnLeft(), TurnRight()
```

)ایستادن ، شتاب دادن ، راندن ، پیچیدن به سمت راست ، پیچیدن به سمت چپ(

شما هم اکنون می توانید فرض کنید کنید که توسط این اجزاء این دوچرخه می تواند حرکت کند و در حقیقت معنای دوچرخه بدهد . مثلا عملگر Accelerate می تواند متغیری به عنوان $Braking_Force به عنوان ورودی دریافت کند و مثلا از متغیرهای $brakes و $wheels هم استفاده کند و مثلا نتیجه مطلب را به عنوان مقدار بازگشتی به برنامه برگرداند.

بسیار جالب به نظر می رسه . اما به نظر شما نمی شه که همه اینها را به صورت منظم توسط تعدادی متغیر و عملگر انجام داد ؟ مسلما این کار امکان پذیر هستش . اما در صورتیکه شما فقط یک دوچرخه را بخواهید دربرنامه خودتون استفاده کنید فرقی به حال شما ندارد که از کلاس استفاده کنید یا نه . اما اگر از تعدادی از دوچرخه ها بخواهید در برنامه استفاده کنید چطور ؟ در اون صورت تعقیب اینکه هر دوچرخه هم اکنون در چه مرحله ای است و چه متغیری از آن باید به کدام عملگرفرستاده شود توسط برنامه نویس بسیار سخت و پیچیده می شود . اما در کلاس دیگر این مشکلات وجود ندارد و هر متغیر برای هر دوچرخه در اختیار عملگر های آن است و هر دوچرخه به صورت جدا متغیر هایش برای عملگرها فرستاده می شود بدون اینکه شما نیازی به کنترل آنها داشته باشید

.همچنین استفاده از کلاس در برنامه های مختلف راحت است چرا که شما با یک کلاس آماده که قبلا در جای دیگر کار می کرده به راحتی می توانید کار کنید و نیازی به تغییر آن ندارید.

حالا اجازه بدید که یک مثال کاربردی و واقعی بزنیم که من در خیلی از صفحات وبی که می سازم از آن استفاده می کنم و ممکن است که بدرد شما هم بخورد.شما رو نمی دونم اما من موقعی که می خواهم صفحات وب پویا رو برنامه نویسی کنم خیلی بدم میاد که همش به فکر روند منطقی درست بودن خروجی HTML برنامه باشم . برای همین هیچ موقع صفحات زیبایی در خروجی برنامه های من نیست چون از رنگها و فونتهای مختلف استفاده نمی کنم.

راه حل موجود استفاده از یک کلاس PHP برای تنظیم خروجی های HTML است.من این کلاس رو Style نامگذاری می کنم . این کلاس شامل متغیرهای زیر است که مهمترین تنظیمات HTML خروجی از برنامه می باشند:

```
<?php
class Style {
var $text;
  var $alink;
  var $vlink;
  var $link;
  var $bgcol;
  var $face;
  var $size;
  var $align;
  var $valign;
}
?>
```

مطمئن هستم که شما با HTML آشنا هستید و این متغیرها برای شما آشنا هستند . هم اکنون سعی می کنم که عملگری به نام Style برای تعیین نوع خروجی درست کنم:

```
<?php
class Style {
function Style ($text="#000000\",$alink=\"#AA00AA\",
  $vlink=\"#AA00AA\",$link=\"#3333FF\",
  $bgcol=\"#999999\",$face=\"Book
Antiqua\",$size=3,$align=\"CENTER\",$valign=\"TOP\")
  {
$this->text=$text;
  $this->alink=$alink;
  $this->vlink=$vlink;
  $this->link=$link;
  $this->bgcol=$bgcol;
  $this->face=$face;
  $this->size=$size;
  $this->align=$align;
  $this->valign=$valign;
}
```

```
}
  ?>
```

هنگامی که شما عملگری همنام با کلاس خودتون تعریف می کنید ، این عملگر همزمان با تعریف شیء ای از نوع این کلاس اجرا خواهد شد . این عملگر ، عملگر constructor یا سازنده نام می گیرد . این عملگر به شما این امکان را می دهد که در هنگام ایجاد شیء ، برای همه متغیر ها یک مقدار اولیه داشته باشید.

```
<?php $Basic = new Style; ?>
```

در اینجا شما شیء ای با نام $basic توسط دستور new از نوع کلاس Style درست کرده اید.

همچنین این امکان وجود دارد در هنگامیکه می خواهید یک شیء جدید توسط یک کلاس ایجاد کنید مقدار اولیه متغیرهای آن را نیز در هنگام ایجاد خوتان انتخاب کنید . اما در صورتیکه مقدار یکی از آنها را شما بخواهید در هنگام ایجاد انتخاب کنید باید همه متغیرهای دیگر را نیز خودتان مقدار اولیه هایشان را تعیین کنید و این امکان وجود ندارد که فقط یکی یا چند تا رو شما تعیین کنید و بقیه رو مقدار اولیه مقدار دهی کنند . به همین خاطر راه بهتری را برای این مشکل پیشنهاد می کنیم و آن استفاده از یک عملگر به نام Set در کلاس است که مقدار یک متغیر را در کلاس تغییر می دهد :

```
<?php
Function Set($varname,$value) {
$this->$varname=$value;
}
?>
```

بنابراین برای تغییر مقدار هر یک از متغیر ها می توانیم از نمونه زیر استفاده کنیم:

```
<?php $Basic->Set('size','2'); ?>
```

شما می توانید از علامت -> برای اشاره به متغیر و یا عملگر یک شیء استفاده کنید . بنابراین خط بالا به اجرا کننده برنامه می گوید که عملگر Set مربوط به شیء $Basic را اجرا کن . از آنجایی که ما $Basic را قبلا از نوع Style تعریف کردیم بنابراین عملگر Set این کلاس اجرا می شود و پارامتر های ورودی شما را می گیرد و برای شیء مورد درخواست تغییرات لازم را اجرا می کند . همچنین شما می توانید این کار را برای متغیر های دیگر مانند $Basic->text نیز انجام دهید و مقدار اولیه آنرا تغییر دهید.

آشنایی با زبان C++

کلاس چیست ؟include

۱ -کلاسها به صورت فایلهایی با پسوند " h "به طور مثال " cat.h" ذخیره میشوند و برای استفاده از آن در برنامه نیاز است که آنرا به برنامه ضمیمه کنید به طورمثال(#include " cat.h"):

اما در داخل فایل کلاس چه چیزهایی نوشته می شود: در کلاس دو مسئاله تعیین می شود:

الف) متغییرهای لازم

ب) توابع مورد نیاز

در مورد تعریف متغییرها باید گفت که همانند تعریف متغییر در بر نامه نویسی غیر شئ گرا می باشد اما یک نکته مهم را باید

مد نظر داشت و آن این است که در برنامه نویسی شئ گرا در کلاس شما حق مقدار دهی به یک متغییر در حین تعریف آن را ندارید . به مثال زیر توجه کنید:

```
;int a=10
```

این در برنامه نویسی عادی مشکلی ندارد اما در یک کلاس حق چنین کاری نداریم و باید به شکل زیر بنویسیم:

```
;int a
```

چگونگی مقدار دهی به این متغییر در بحث شئ مورد بررسی قرار می گیرد. و اما توابع مورد نیاز نیز روش خاص خود را داراست . در یک کلاس شما فقط باید تابع را تعریف کنید و کد نویسی بدنه تابع در داخل کلاس انجام نمی شود . در ادامه یک نمونه از یک کلاس ساده آورده شده است:

```
}class cat

;int age
;int weight,length

;(void setage(int x
;( int getage(void
;(void Meow(void

{
```

همانطور که مشاهده کردید در یک کلاس تنها تعریف متغییرها و توابع صورت می گیرد.
این نوشته به صورت فایلی با پسوند " h" ذخیره می شود بطور مثال : cat.h

حالا باید بتونیم توابع تعریف شده رو بصورت کامل بنویسیم.
برای این کار یک فایل هم نام با نام کلاس می سازند با پسوند cppمثلا cat.cpp
در این فایل جدید ابتدا نوشته می شود" i #include "cat.h دیگر هدر فایلهای مورد نیاز نیز که جزء ملزومات برناه نویسی است نیز باید نوشته شود
بعد از آن نوشتن بدنه فایلها شروع می شود که همانند سی معمولی است اما با کمی تفاوت جزئی:

بعد از تعیین نوع تابع باید نام کلاس نوشته شده بعد ۲ تا علامت :: گذاشته و اسم تابع و بقیه مجرا نوشته شود . مثلا:

```
(void cat::setage (int a
}
;age=a

}
```

سطح دسترسی در کلاسها

در مورد سطح دسترسی به متغیرها و توابع در یک کلاس بحث خواهیم کرد
سطح دسترسی یعنی چی؟
یعنی یک کلاس برای محتویات خود یکسری دسته بندی را رعایت می کند که هر کسی به هر چیزی نتواند دسرسی داشته باشد که در بعضی موارد اگر رعایت نشود می‌تواند باعث بسیاری مشکلات شود.
در یک کلاس ۳ نوع دسترسی وجود دارد:
Public:در این نوع دسترسی هیچ محدودیتی اعمال نمی شود و هر چیزی چه داخلی و چه خارجی می تواند از ان استفاده کند (فعلا قصد بنده معرفی دسترسی ها می‌باشد و برای توضیحات عمیق تر لازم به دانستن یک سری مطالب دیگر است که در روزهای آینده ذکر خواهد شد)
۲ :Private - این نوع دسترسی بر عکس نوع قبل عمل می کند . یعنی غیر از توابع عضو این کلاس هیچ چیز دیگری نمی تواند به آنها دسترسی داشته باشد . مثلا وقتی یک شئ از این کلاس تعریف می کنیم از طریق شئ نمی توانیم مستقیما به ایم نوع متغیرها یا توابع دسترسی داشته باشیم اما خود توابع عضو این کلاس میتوانند در کد نویسی خود از این نوع استفاده کنند که در آینده بیشتر آشنا خواهیم شد.
۳:Protected - در این نوع نیز شئی که از کلاس تعریف می شود نمی تواند به این نوع دسترسی داشته باشد . این نوع تعریف خاص خود را دارد که بعد از بحث ارث بری قابل ذکر است و در اینجا تنها نامی از آن برای تکمیل بحث آورده شده است.
*نکته قابل ذکر این است که غالبا از دو نوع ۱و۲ استفاده می شود و از نوع ۳ خیلی کم استفاده خواهید کرد.
در کلاس این انواع دسترسی با ۳ کلمه کلیدی ذکر شده تعیین می شوند:
اگر در ابتدای کلاس باشیو و هیچ کدام را ننویسیم متغیرها و توابع تعریفی تا کلمه کلیدی دیگر همه private محسوب می شوند تا زمانی که از یکی از دو کلمه دیگر استفاده شود . بعد از آن نیز بقیه از این کلمه استفاده شده تبعیت می کنند تا کلمه کلیدی بعدی.

در ادامه یک نمونه مثال از سطح دسترسی آورده شده و در ادامه آن مثالهای اشتباه و درست نیز آورده شده است:

```
class Cat
}
;int a,b
;(void setage(int age
:public
;int c,d
;(void setlength(int length
:protected
;int e
;(void setwidth(int width
}
```

در بالا یک نمونه کلاس آورده شده حالا یک شئ از آن تعریف کرده و مثالهای درست و غلط را ذکر میکنم:

```
;Cat m
```

مثالهای درست:

```
;m.a=10
;m.b=80
;()m.setage
```

مثالهای غلط:

```
;m.b=20
;m.c=13
;()m.setlength
;m.e=90
;()m.setwidth
```

Polymorphism چند ریختی

چند ریختی یا Polymorphism یکی از خواص جالب در C++ محسوب میشه. شما نمی تونید توابعی با اسامی یکسان داشته باشید مگر در یک حالت استثناء اون هم مسئله چند ریختی هستش. یعنی چند تابع مختلف با یک اسم یکسان تعریف می کنید اما باید دقت داشته باشید که در مقادیر ورودی توابع متفاوت باشند.
مثال:

```
}(void set(int a
; int b
;b=a
}
*************
}(void set(float s
; float m
;m=s
}
*************
}(int set(int f,int g
; int j,k
;j=f
;k=g
; return k+j
}
*************
```

در مثالهای بالا دقت کنید که اسم توابع یکسان است و تنها در موقع صدا زدن آنها با توجه به نوع ورودی تابع ، تابع مورد نظر اجرا می شود.
مثال:
اگر بنویسیم set (1); تابع اولی اجرا میشود

اگر بنویسیم set (5.1); تابع دومی اجرا میشود

اگر بنویسیم set (90, 8); تابع سومی اجرا میشود و مقدار ۹۸ را بر می گرداند.

ارث بری کلاسها (مشتق کردن کلاسی از کلاس دیگر)

ارث بری همونطور که از اسمش کاملا پیداست همانند قانون ارث بردن در موجودات زنده عمل می کند ، یعنی همانند طبیعت یک کلاس به عنوان کلاس پدر (مادر) فرض می شود و یک کلاس به عنوان فرزند از این کلاس یک سری خصوصیات و قابلیتها را به ارث می برد (فرزند نیز دارای بعضی امکانات پدر می شود . البته میزان آن بستگی به خواست ما دارد که توضیح خواهم داد)
کلاس پدر را کلاس پایه گویند. Base Class
ارث بری را مشتق کردن نیز می گویند .
++++++++++++++++++++++++++++++++++
نحوه مشتق کردن یک کلاس از کلاس دیگر:
فرض می کنیم کلاسی با نام A وجود دارد:

```
} class B :(type) A
; int a,d
:public
;(void Rotate (void
}
```

نوشته بالا یعنی کلاس B از کلاس A با دسترسی نوع (type) که توضیح خواهم داد که نوع دسترسی ۳ دسته است مشتق شده یا خصوصیات و قابلیتهای آن را به ارث برده است.

۳ نوع دسترسی برای مشتق کردن وجود دارد که عبارتند از:
۱- public
۲- protected
۳- private
یعنی یکی از سه کلمه بالا به جای (type) نوشته خواهد شد.
هر کدام از این انواع دسترسی توضیحات خاص خود را دارد که عبارتند از:
۱ : public -

یعنی تمام خواص عمومی و خصوصی کلاس پایه را به همان شکل به ارث می برد . بدین شکل که موارد public در کلاس مبنا (پایه) برای

این کلاس جدید نیز وجود دارد و برای این نیز public خواهد بود . و تمام خواص protetted و private نیز به همین شکل می باشد که در کلاس جدید نیز هر کدام protected و private خواهند بود.

نکته مهم : در این نوع دسترسی توابع عضو کلاس جدید اجازه دسترسی به خواص protected کلاس مبنا را دارند اما به خواص private خیر !! دسترسی ندارند.

۲: protected -

در این نوع دسترسی ، کلاس جدید خواص کلاس مبنا را به این شکل به ارث می برد که تنها توابع عضو کلاس جدید به فقط خواص public و protected کلاس مبنا دسترسی دارند و به خواص private دسترسی ندارند.

۳: private -

این نوع دسترسی یعنی نه شئی از کلاس جدید و نه تابع عضو کلاس جدید به هیچ چیز از کلاس مبنا دسترسی ندارند !! (عملا یعنی این نوع دسترسی یعنی اصلا مشتق نکنیم سنگین تریم(!!!

نکته بسیار مهم:

ارث بری میتواند به گونه ای باشد که یک کلاس از تعداد بیش از ۱ کلاس ارث ببرد (مشتق شود) . مثلا:

```
}class B: public A , public C , protected D , public E , private F
.
.
.
.
}
```

آشنایی با زبان جاوا

بنیادهای کلاسjava

كلاسهاي توليد شده در بحثهاي گذشته فقط براي كپسول سازي روش main()استفاده مي شد ، كه براي نشان دادن اصول دستور زبان جاوا مناسب بودند . شايد بهترين چيزي كه بايد درباره يك كلاس بدانيد اين است كه كلاس يك نوع جديد داده را تعريف مي كند . هربار كه اين نوع تعريف شود ، مي توان از آن براي ايجاد اشيائي از
همان نوع استفاده نمود . بنابراين ، يك كلاس قالبي (template) براي يك شي ئ است و يك شي ئ نمونه اي (instance) از يك كلاس است . چون شي ئ يك نمونه از يك كلاس است غالبا" كلمات شي ئ (object) و نمونه (instance)را بصورت مترادف بكار مي بريم .

شكل عمومي يك كلاس

هنگاميكه يك كلاس را تعريف مي كنيد ، در حقيقت شكل و طبيعت دقيق آن كلاس را اعلان مي كنيد . ابتكار را با توصيف داده هاي موجود در آن كلاس و كدهايي كه روي آن داده ها عمل مي كنند ، انجام مي دهيد . در حاليكه كلاسها ممكن است خيلي ساده فقط شامل داده يا فقط كد باشند، اكثر كلاسهاي واقعي هردو موضوع را دربرميگيرند .بعدا" خواهيد ديد كه كد يك كلاس ، رابط آن به داده هاي همان كلاس را توصيف ميكند .
يك كلاس را با واژه كليدي class اعلان مي كنند . كلاسهايي كه تا بحال استفاده شده اند ، نوع بسيار محدود از شكل كامل كلاسها بوده اند . خواهيد ديد كه كلاسها مي توانند (و معمولا" هم) بسيار پيچيده تر باشند . شكل عمومي توصيف يك كلاس به
شرح زير است :

```
type methodname2(parameter-list ){
// body of method
}
//...
type methodnameN(parameter-list ){
// body of method
}
}
```

داده يا متغيرهايي كه داخل يك كلاس تعريف شده اند را متغيرهاي نمونه instance) (variables مي نامند . كدها ، داخل روشها (methods) قرار مي گيرند . روشها و متغيرهاي تعريف شده داخل يك كلاس را اعضائ (members) يك كلاس مي نامند . در اكثر كلاسها ، متغيرهاي نمونه يا روي روشهاي تعريف شده براي آن كلاس عمل كرده يا توسط اين روشها مورد دسترسي قرار مي گيرند . بنابراين ، روشها تعيين كننده چگونگي
استفاده از داده هاي يك كلاس هستند .متغيرهاي تعريف شده داخل يك كلاس ، متغيرهاي نمونه خوانده شده زيرا هر نمونه از كلاس (يعني هر شي ئ يك كلاس) شامل كپي خاص خودش از اين متغيرهاست . بنابراين داده مربوط به يك شي ئ ، جدا و منحصر بفرد از داده مربوط به شي ئ ديگري است . ما بزودي اين نكته را بررسي خواهيم نمود ، اما فعلا" بايد اين نكته بسيار مهم را بياد داشته باشيد .

کلیه روشها نظیر main()همان شکل عمومي را دارند که تاکنون استفاده کرده ایم .
اما ، اکثر روشها را بعنوانstatic یاpublic ا توصیف نمي کنند . توجه داشته باشید که شکل عمومي یك کلاس ، یك روش main() را توصیف نمي کند . کلاسهاي جاوا لزومي ندارد که یك روش main() داشته باشند . فقط اگر کلاس ، نقطه شروع برنامه شما باشد ، باید یك روش main() را توصیف نمایید . علاوه بر این ، ریز برنامه ها
(applets)اصولا" نیازي به روش main() ندارند .نکته : برنامه نویسان C++ آگاه باشند که اعلان کلاس و پیاده سازي روشها در یك مکان ذخیره شده و بصورت جداگانه تعریف نمي شوند. این حالت گاهي فایلهاي خیلي بزرگ java ایجاد مي کند ، زیرا هر کلاس باید کاملا" در یك فایل منبع تکي تعریف شود . این طرح در جاوا رعایت شد زیرا احساس مي شد که در بلند مدت ، در اختیار داشتن مشخصات ، اعلانها و پیاده سازي در یك مکان ، امکان دسترسي آسانتر کد را بوجود مي آورد .

یك کلاس ساده

بررسي خود را با یك نمونه ساده از کلاسها شروع مي کنیم . در اینجا کلاسي تحت غنوان Box وجود دارد که سه متغیر نمونه را تعریف مي کندwidth : ، height، وdepth و فعلا" ، کلاس Box دربرگیرنده روشها نیست .

```
+ class Box {
+ double width;
+ double height;
+ double depth;
+ }
```

قبلا" هم گفتیم که یك کلاس نوع جدیدي از داده را توصیف مي کند . در این مثال نوع جدید داده را Box نامیده ایم . از این نام براي اعلان اشیائ از نوع Box استفاده مي کنید . نکته مهم این است که اعلان یك کلاس فقط یك الگو یا قالب را ایجاد مي کند ، اما یك شي ئ واقعي بوجود نمي آورد . بنابراین ، کد قبلي ، شیئي از نوع Box را بوجود نمي آورد .براي اینکه واقعا" یك شي ئ Boxرا بوجود آورید ، باید از دستوري نظیر مورد زیر استفاده نمایید :

```
+ Box mybox = new Box)(; // create a Box object called mybox
```

پس از اجراي این دستور ، myboxنمونه اي از Box خواهد بود. و بدین ترتیب این شي ئ وجود فیزیکي و واقعي پیدا مي کند .مجددا" بیاد داشته باشید که هر بار یك نمونه از کلاسي ایجاد مي کنید ، شیئي ایجاد کرده اید که دربرگیرنده کپي (نسخه خاص) خود از هر متغیر نمونه تعریف شده توسط کلاس خواهد بود . بدین ترتیب ، هر شي ئ Box دربرگیرنده کپي هاي خود ازمتغیرهاي نمونهwidth ، heidht،و depthو مي باشد . براي دسترسي به این متغیرها از عملگر نقطه (.) استفاده مي کنید . عملگر نقطه اي ، نام یك شي ئ را با

نام يك متغير نمونه پيوند مي دهد . بعنوان مثال ، براي منتسب كردن مقدار 100 به متغيرwidth درmybox ر ، از دستور زير استفاده نماييد :

```
+ mybox.width = 100;
```

اين دستور به كامپايلر مي گويد كه كپي width كه داخل شي ئ mybox قرار گرفته را معادل عدد 100 قرار دهد . بطور كلي ، از عملگر نقطه اي براي دسترسي هم به متغيرهاي نمونه و هم به روشهاي موجود در يك شي ئ استفاده مي شود .

دراينجا يك برنامه كامل را مشاهده ميكنيد كه از كلاس Box استفاده كرده است :

```
+ /* A program that uses the Box class.
+
+ Call this file BoxDemo.java
+ /*
+ class Box {
+ double width;
+ double height;
+ double depth;
+ }
+
+ // This class declares an object of type Box.
+ class BoxDemo {
+ public static void main(String args[] ){
+ Box mybox = new Box)(;
+ double vol;
+
+ // assign values to mybox's instance variables
+ mybox.width = 10;
+ mybox.height = 20;
+ mybox.depth = 15;
+
+ // compute volume of box
+ vol = mybox.width * mybox.height * mybox.depth;
+
+ System.out.println("Volume is " + vol);
+ }
+ }
```

فايلي را كه دربرگيرنده اين برنامه است بايد با نام BoxDemo.java بخوانيد زيرا روش main() در كلاس BoxDemo و نه در كلاس Box قرار گرفته است . هنگاميكه اين برنامه را كامپايل مي كنيد ، مي بينيد كه دو فايل .class ايجاد شده اند ، يكي براي Box و ديگري براي BoxDemo . كامپايلر جاوا بطور خودكار هر كلاس را در فايل .class مربوط به خودش قرار مي دهد . ضرورتي ندارد كه كلاسBox و BoxDemo و هر دو در يك فايل منبع قرار گيرند . مي توانيد هر كلاس را در فايل خاص خودش گذاشته و آنها را بترتيبBox.java وBoxDemo.java و بناميد . براي اجراي اين برنامه بايد BoxDemo.class را اجرا كنيد . پس از اينكار حاصل زير را بدست مي آوريد :

```
Volume is 3000
```

قبلا" هم گفتيم كه هر شي ئ داراي كپي هاي خاص خودش از متغيرهاي نمونه است. يعني اگر دو شي ئ Box داشته باشيد، هركدام بتنهايي كپي (يا نسخه اي) ازlenght width وheight و خواهند داشت . مهم است بدانيد كه تغييرات در متغيرهاي نمونه يك شي ئ تاثيري روي متغيرهاي نمونه كلاس ديگر نخواهد داشت . بعنوان مثال ، برنامه بعدي دو شي ئ Box را اعلان مي كند :

```
+ // This program declares two Box objects.
+
+ class Box {
+ double width;
+ double height;
+ double depth;
+ }
+
+ class BoxDemo2 {
+ public static void main(String args[] ){
+
+ Box mybox1 = new Box)(;
+ Box mybox2 = new Box)(;
+ double vol;
+
+ // assign values to mybox1's instance variables
+ mybox1.width = 10;
+ mybox1.height = 20;
+ mybox1.depth = 15;
+
+ /* assign different values to mybox2's
+ instance variables */
+ mybox2.width = 3;
+ mybox2.height = 6;
+ mybox2.depth = 9;
+
+ // compute volume of first box
+ vol = mybox1.width * mybox1.height * mybox1.depth;
+ System.out.println("Volume is " + vol);
+
+ // compute volume of second box
+ vol = mybox2.width * mybox2.height * mybox2.depth;
+ System.out.println("Volume is " + vol);
+ }
+ }
```

خروجي توليد شده توسط اين برنامه بقرار زير مي باشد:

```
Volume is 3000
Volume is 162
```

آرايه ها در جاوا

يك آرايه گروهي از متغيرهاي يك نوع است كه با يك نام مشترك به آنها ارجاع مي شود . مي توان آرايه ها را براي هر يك از انواع ايجاد نمود و ممكن است اين آرايه ها داراي يك يا چندين بعد باشند . براي دسترسي به يك عضو آرايه از نمايه (index) آن آرايه

استفاده مي شود . آرايه ها يك وسيله مناسب براي گروه بندي اطلاعات مرتبط با هم هستند . نكته : اگر با C و C++ و آشنايي داريد ، آگاه باشيد . آرايه ها در جاوا بطور متفاوتي نسبت به زبانهاي ديگر كار مي كنند .

آرايه هاي يك بعدي
آرايه يك بعدي بطور ضروري فهرستي از متغيرهاي يكنوع است . براي ايجاد يك آرايه ، بايد يك متغير آرايه از نوع مورد نظرتان ايجاد كنيد . فرم عمومي اعلان يك آرايه يك بعدي بقرار زير است :

type var-name [];

نام متغير نوع

در اينجا type اعلان كننده نوع اصلي آرايه است . نوع اصلي تعيين كننده نوع داده براي هر يك از اعضائ داخل در آرايه است . بنابراين ، نوع اصلي آرايه تعيين مي كند كه آرايه چه نوعي از داده را نگهداري مي كند . بعنوان مثال ، در زير يك آرايه با نام month-days با نوع آرايه اي از عدد صحيح اعلان شده است .

+ int month_days[];

اگر چه اين اعلان تثبيت مي كند كه month-days يك متغير آرايه است ، اما بطور واقعي آرايه اي وجود ندارد . در حقيقت ، مقدار month-days برابر تهي (null) مي باشد كه يك آرايه بدون مقدار را معرفي مي كند . براي پيوند دادن month-days با يك آرايه واقعي و فيزيكي از اعداد صحيح ، بايد از يك عملگر new استفاده نموده و به month-days منتسب كنيد .يك عملگراست كه حافظه را اختصاص ميدهد .بعدا " newرا با دقت بيشتري بررسي مي كنيم ، اما لازم است كه هم اكنون از آن استفاده نموده و حافظه را براي آرايه ها تخصيص دهيد . فرم عمومي new آنگونه كه براي آرايه هاي يك بعدي بكار مي رود بقرار زير ظاهر خواهد شد :

array-var=new type [size];

اندازه نوع متغير آرايه

در اينجا type مشخص كننده نوع داده اي است كه تخصيص داده مي شود ، sizeمشخص كننده تعداد اعضائ آرايه است و array-var متغير آرايه است كه به آرايه پيوند مي يابد . يعني براي استفاده از newدر تخصيص يك آرايه ، بايد نوع و تعداد اعضايي كه تخصيص مي يابند را مشخص نماييد . اعضائ آرايه كه توسط new تخصيص مي يابند بطور خودكار با مقدار صفر مقدار دهي اوليه مي شوند . اين

مثال يك آرايه 12 عضوي از اعداد صحيح را تخصيص داده و آنها را به month-days پيوند مي دهد .

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

بعد از اجراي اين دستور ، month-daysبه يك آرايه 12 تايي از اعداد صحيح ارجاع خواهد نمود . بعلاوه كليه اجزائ در آرايه با عدد صفر مقدار دهي اوليه خواهند شد . اجازه دهيد مرور كنيم : بدست آوردن يك آرايه مستلزم پردازش دو مرحله اي است . اول بايد يك متغير با نوع آرايه مورد نظرتان اعلان كنيد . دوم بايد حافظه اي كه آرايه را نگهداري مي كند ، با استفاده از newتخصيص دهيد و آن را به متغير آرايه نسبت دهيد . بنابراين در جاوا كليه آرايه ها بطور پويا تخصيص مي يابند . اگر مفهوم تخصيص پويا براي شما ناآشناست نگران نباشيد . اين مفهوم را بعدا "تشريح خواهيم كرد . هر بار كه يك آرايه را تخصيص مي دهيد ، مي توانيد بوسيله مشخص نمودن نمايه آن داخل كروشه [] به يك عضو مشخص در آرايه دسترسي پيدا كنيد . كليه نمايه هاي آرايه ها با عدد صفر شروع مي شوند. بعنوان مثال اين دستور مقدار 28 را به دومين عضو month-days نسبت مي دهد .

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

خط بعدي مقدار ذخيره شده در نمايه 3 را نمايش مي دهد .

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

با كنار هم قرار دادن كليه قطعات ، در اينجا برنامه اي خواهيم داشت كه يك آرايه براي تعداد روزهاي هر ماه ايجاد مي كند .

```
+ // Demonstrate a one-dimensional array.
+ class Array {
+ public static void main(String args[] ){
+ int month_days[];
+ month_days = new int[12];
+ month_days [0] = 31;
+ month_days [1] = 28;
+ month_days [2] = 31;
+ month_days [3] = 30;
+ month_days [4] = 31;
+ month_days [5] = 30;
+ month_days [6] = 31;
+ month_days [7] = 31;
+ month_days [8] = 30;
+ month_days [9] = 31;
+ month_days [10] = 30;
+ month_days [11] = 31;
+ System.out.println("April has " + month_days[3] + " days .");
+ }
+ }
```

وقتي اين برنامه را اجرا ميكنيد ، برنامه ، تعداد روزهاي ماه آوريل را چاپ ميكند. همانطوريكه ذكر شد، نمايه هاي آرايه جاوا با صفر شروع مي شوند، بنابراين تعداد روزهاي ماه آوريل در برابر month-days[3] 30 مي باشد .
اين امكان وجود دارد كه اعلان متغير آرايه را با تخصيص خود آرايه بصورت زير تركيب نمود:

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

اين همان روشي است كه معمولا" در برنامه هاي حرفه اي نوشته شده با جاوا مشاهده مي كنيد . مي توان آرايه ها را زمان اعلانشان ، مقدار دهي اوليه نمود . پردازش آن بسيار مشابه پردازشي است كه براي مقدار دهي اوليه انواع ساده استفاده مي شود . يك مقدار ده اوليه آرايه فهرستي از عبارات جدا شده بوسيله كاما و محصور شده بين ابروهاي باز و بسته مي باشد . كاماها مقادير اجزائ آرايه را از يكديگر جدا مي كنند . آرايه بطور خودكار آنقدر بزرگ ايجاد مي شود تا بتواند ارقام اجزايي را كه در مقدار ده اوليه آرايه مشخص كرده ايد ، دربرگيرد . نيازي به استفاده ازnew وجود ندارد . بعنوان مثال ، براي ذخيره نمودن تعداد روزهاي هر ماه ، كد بعدي يك آرايه مقدار دهي اوليه شده از اعداد صحيح را بوجود مي آورد :

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

وقتي اين برنامه را اجرا كنيد ، همان خروجي برنامه قبلي را خواهيد ديد .جاوا بشدت كنترل مي كند تا مطمئن شود كه بطور تصادفي تلاشي براي ذخيره نمودن يا ارجاع مقاديري خارج از دامنه آرايه انجام ندهيد . سيستم حين اجراي جاوا كنترل مي كند كه كليه نمايه هاي آرايه ها در دامنه صحيح قرار داشته باشند . (از اين نظر جاوا كاملا"باC اوC++ و متفاوت است كه هيچ كنترل محدوده اي در حين اجرا انجام نمي دهند .) بعنوان مثال ، سيستم حين اجرا ، مقدار هر يك از نمايه ها بهmonth-days را كنترل مي كند تا مطمئن شود كه بين ارقام 0 و 11 داخل قرار داشته باشند . اگر تلاش كنيد تا به اجزائ خارج از دامنه آرايه (اعداد منفي يا اعدادي بزرگتر از طول آرايه) دسترسي يابيد، يك خطاي حين اجرا (run-time error)توليد خواهد شد .در زير يك مثال پيچيده تر مشاهده مي كنيد كه از يك آرايه يك بعدي استفاده مي كند . اين برنامه ميانگين يك مجموعه از ارقام را بدست مي آورد .

```
+ // Average an array of values.
+ class Average {
+ public static void main(String args[] ){
+ double nums[] = {10.1/ 11.2/ 12.3/ 13.4/ 14.5};
+ double result = 0;
+ int i;
+
```

```
+ for(i=0; i<5; i++)
+ result = result + nums[i];
+
+ System.out.println("Average is " + result / 5);
+ }
+ }
```

آرايه هاي چند بعدي

در جاوا آرايه هاي چند بعدي در واقع آرايه اي از آرايه ها هستند . اين قضيه همانطوريكه انتظار داريد ظاهر و عملكردي مشابه آرايه هاي چندبعدي منظم (regular) دارد . اما خواهيد ديد كه تاوتهاي ظريفي هم وجود دارند . براي اعلان يك متغير آرايه چند بعدي ، با استفاده از مجموعه ديگري از كروشه ها هر يك از نمايه هاي اضافي را مشخص مي كنيد. بعنوان مثال ، عبارت زيرر يك متغير آرايه دو بعدي بنام twoD را اعلان مي كند .

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

اين عبارت يك آرايه 4در 5ر را تخصيص داده و آن را به twoD نسبت مي دهد . از نظر داخلي اين ماتريس بعنوان يك آرايه از آرايه نوع int پياده سازي خواهد شد .بطور فرضي ، اين آرايه را مي توان بصورت شكل زير نمايش داد .

Right index determines column.

```
|| || || || ||
V V V V V

|
| [0][4] | [0][3] | [0][2] | [0][1] | [0][0] >
|
| |
| [1][4] | [1][3] | [1][2] | [1][1] | [1][0] >
Left index
determines |
| [2][4] | [2][3] | [2][2] | [2][1] | [2][0] .> row
|
| |
| [3][4] | [3][3] | [3][2] | [3][1] | [3][0] >
Given :int twoD[][] = new int [4][5];
```

برنامه بعدي هر عضو آرايه را از چپ به راست ، و از بالا به پايين شماره داده و سپس مقادير آنها را نمايش مي دهد :

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

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

خروجي اين برنامه بقرار زير خواهد بود

```
4 3 2 1 0 :

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

هنگام تخصيص حافظه به يك آرايه چند بعدي ، كافي است فقط حافظه براي اولين بعد را مشخص نماييد . مي توانيد ابعاد ديگر را جداگانه تخصيص دهيد . بعنوان مثال ، كد زير حافظه اولين بعد twoDرا هنگام اعلان آن تخصيص مي دهد . اين كد حافظه دومين بعد را بصورت دستي اختصاص مي دهد .

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

اگرچه در اين حالت اختصاص انفرادي حافظه به دومين بعد هيچ مزيتي ندارد، اما احتمال چنين مزيتهايي وجود دارد . بعنوان مثال ، هنگاميكه ابعاد را بصورت دستي اختصاص مي دهيد ، نيازي نيست كه همان ارقام براي اجزائ هر بعد را تخصيص دهيد . همانطوريكه قبلا" گفتيم ، از آنجاييكه آرايه هاي چند بعدي واقعا" آرايه اي از آرايه ها هستند ، طول هر يك از آرايه ها تحت كنترل شما قرار مي گيرند . بعنوان مثال ، برنامه بعدي يك آرايه دو بعدي ايجاد مي كند كه در آن اندازه هاي دومين بعد نامساوي هستند .

```
+ // Manually allocate differing size second dimension.
+ class TwoDAgain {
+ public static void main(String args[] ){
+
+
+ int twoD[][] = new int[4][];
+ twoD[0] = new int[1];
+ twoD[1] = new int[2];
+ twoD[2] = new int[3];
+ twoD[3] = new int[4];
+
+ int i/ j/ k = 0;
```

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

خروجي اين برنامه بقرار زير مي باشد

```
0 :

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

آرايه اي كه توسط اين برنامه ايجاد مي شود ، بصورت زير خواهد بود :

| [0][0] |

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

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

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

از آرايه هاي چند بعدي ناجور (يا نامنظم) در اكثر برنامه ها استفاده نميشود زيرا برخلاف آنچه مردم هنگام مواجه شدن با يك آرايه چند بعدي انتظار دارند رفتار مي كنند . اما اين آرايه ها در برخي شرايط بسيار كارا هستند . بعنوان مثال ، اگر نياز به يك آرايه دو بعدي خيلي بزرگ داريد كه داراي تجمع پراكنده باشد (يعني كه يكي و نه همه اجزائ آن مورد استفاده قرار مي گيرند) ، آنگاه آرايه بي قاعده احتمالا" يك راه حل كامل خواهد بود . اين امكان وجود دارد كه آرايه هاي چند بعدي را مقدار دهي اوليه نمود . براي اينكار ، فقط كافي است هر يك از مقدار ده اوليه ابعاد را داخل مجموعه ابروهاي خنص خودش قرار دهيد . برنامه بعدي يك ماتريس ايجاد مي كند كه هر يك از اجزائ آن شامل حاصلضرب نمايه هاي سطرها و ستونها هستند. همچنين دقت نماييد كه مي توان از عبارات همچون مقادير لفظي داخل مقدار ده اوليه آرايه استفاده نمود .

```
+ // Initialize a two-dimensional array.
+ class Matrix {
+ public static void main(String args[] ){
+ double m[][] = {
+ { 0*0/ 1*0/ 2*0/ 3*0 };
```

```
+ { 0*1/ 1*1/ 2*1/ 3*1 };
+ { 0*2/ 1*2/ 2*2/ 3*2 };
+ { 0*3/ 1*3/ 2*3/ 3*3 };
+ };
+ int i/ j;
+
+ for(i=0; i<4; i++ ){
+ for(j=0 j<4; j++)
+ System.out.print(m[i][j] + " ");
+ System.out.println)(;
+ }
+ }
+ }
```

پس از اجراي اين برنامه ، خروجي آن بقرار زير خواهد بود

```
0 0 0 0 :

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

همانطوريكه مشاهده مي كنيد، هر سطر در آرايه همانگونه كه در فهرستهاي مقدار دهي اوليه مشخص شده ، مقدار دهي اوليه شده است.مثالهاي بيشتري درباره استفاده از آرايه چند بعدي بررسي مي كنيم . برنامه بعدي يك آرايه سه بعدي 3 x4x5ايجاد مي كند . سپس حاصل نمايه هاي مربوطه را براي هر عضو بارگذاري مي كند . در نهايت اين حاصل ها را نمايش خواهد داد :

```
+ // Demonstrate a three-dimensional array.
+ class threeDDatrix {
+ public static void main(String args[] ){
+ int threeD[][][] = new int[3][4][5];
+ int i/ j/ k;
+ for(i=0; i<3; i++)
+ for(j=0; j<4; j++)
+ for(k=0; k<5; k++)
+ threeD[i][j][k] = i * j * k;
+
+ for(i=0; i<3; i++ ){
+ for(j=0; j<4; j++ ){
+ for(k=0; k<5; k++)
+ System.out.print(threeD[i][j][k] + " ");
+ System.out.println)(;
+ }
+ System.out.println)(;
+ }
+ }
+ }
```

خروجي اين برنامه بقرار زير خواهد بود

```
0 0 0 0 0 :

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12

0 0 0 0 0
0 2 4 6 8
0 4 8 12 16
0 6 12 18 24

دستور زبان جایگزین اعلان آرایه

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

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

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

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

دو شکل اعلان زیر هم یکسان عمل می کنند :

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