اصل و نسب جاوا

جاوا به زبان C++ نتیجه مستقیم زبان C وابسته است . بسیاری از خصلتهای جاوا بطور مستقیم از این دو زبان گرفته شده است . دستور زبان جاوا منتج از دستور زبان C است . بسیاری از جنبه های oop زبان جاوا از C++ بعاریت گرفته شده است . در حقیقت بسیاری از خصلتهای زبان جاوا از این دو زبان مشتق شده یا با آنها مرتبط است . علاوه بر این ، تولید جاوا بطور عمیقی متاثر از روال پالایش و تطبیقی است که طی سه دهه گذشته برای زبانهای برنامه نویسی موجود پیش آمده است . بهمین دلایل بهتر است سیر مراحل و نیروهایی که منجر به تولد جاوا شده را بررسی نماییم . هرنوع ابتکار و فکر جدید در طراحی زبانها براساس نیاز به پشت سر نهادن یک مشکل اصلی است که زبانهای قبلی از حل آن عاجز مانده اند . جاوا نیز بهمین ترتیب متولد شد .

جاوا از نظر ساختار بسیار شبیه زبان C/C++ و این به هیچ وجه تصادفی نیست c زبانی است ساخته یافته و c++ زبانی شی گرا و مهمتر از همه قسمت اعظم برنامه نویسان دنیا از C/C++ استفاده می کنند. و از سوی دیگر این حرکت به طرف جاوا را برای این قبیل افراد ساده خواهد کرد.

جاوا با دور انداختن نشانگرها (Pointers)و بر دوش کشیدن بار مدیریت حافظه برنامه نویسان C/C++ را برای همیشه از این کابوس رهایی بخشیده است جاوا همچون C/C++ به بزرگی و کوچکی حروف حساس است و برنامه نوشته شده باید دارای متد main باشد.

زمینه های پیدایش جاوا

تاریخچه زبانهای برنامه نویسی بشرح زیر است : زبان B منجر به ظهور زبان C وC زمینه پیدایش C++ شد و در نهایت زبان جاوا متولد شد . درک زبان جاوا مستلزم : درک زمینه های لازم برای ایجاد جاوا ، نیروهایی که این زبان را شکل داده اند و مشخصاتی است که این زبان از اسلاف خود به ارث برده است . نظیر سایر زبانهای برنامه نویسی موفق ، جاوا نیز عناصر بارث برده از اسلاف خود را با ایده های ابتکاری که ناشی از محیط منحصر بفرد این زبان بوده درهم آمیخته است . فصول بعدی جنبه های عملی زبان جاوا شامل دستور زبان (syntax) و کتابخانه ها (libraries)و کاربردهای جاوا را توصیف می کند . فعلا" چگونگی و علت ظهور جاوا و اهمیت آن را بررسی می کنیم . اگر چه جاوا تفکیک ناپذیری با محیط های همزمان اینترنت پیوستگی دارد ، اما بخاطر بسپارید که جاوا قبل از هر چیز یک زبان برنامه نویسی است . ابداعات و پیشرفت ها در زبانهای برنامه نویسی کامپیوتر بدو دلیل بروز می کنند : تطابق با تغییرات محیط ها و کاربردها . ایجاد پالایش و پیشرفت در هنر برنامه نویسی . همانطوریکه بعدا" مشاهده می کنید ، تولد جاوا از این دو دلیل بطور یکسان به ارث گرفته است.

جاوا هم مانند اکثر اختراعات مهم حاصل تلاش گروهی دانشمند پیشتاز است . مدیران سان به این فکر افتادند که کاری کنند که سیستم مزبور بتواند به سیستم سخت افزاری مختلف منتقل شود . برای این منظور ابتدا از کامپایلر c++ استفاده

کنند ولی به زودی نارسایی c++ در این زمینه خود را نشان داد .و مهندسان سان خیلی سریع دریافتند که برای ادامه کار باید چیزی جدید و قوی خلق کنند .

نسخه اولیه ی جاوا در سال 1991 با نام Oak توسط تیمی از برنامه نویسان شرکت سان به سرپرستی جیمز گاسلینگ طراحی شد و در سال 1992 به جاوا تغییر نام پیدا کرد و به بازار عرضه شد .

تولد زبان برنامه نویسی جدید C :

زبان C پس از تولد ، شوک بزرگی به دنیای کامپیوتر وارد کرد . این زبان بطور اساسی شیوه های تفکر و دستیابی به برنامه نویسی کامپیوتر را دگرگون ساخت . تولدC ناشی از نیاز به یک زبان ساخت یافته ، موثر و سطح بالا بعنوان جایگزینی برای کدهای اسمبلی و پیاده سازی برنامه نویسی سیستم بود . هنگامیکه یک زبان برنامه نویسی جدید متولد میشود ، مقایسه ها شروع خواهد شد . مقایسه ها براساس معیارهای زیر انجام می گیرند : ؤ راحتی کاربری در مقایسه با قدرتمندی زبان برنامه نویسی ؤ ایمنی در مقایسه با سطح کارآیی ؤ استحکام در مقایسه با توسعه پذیری قبل از ظهور زبان C برنامه نویسان با زبانهایی کار می کردند که قدرت بهینه سازی یک مجموعه خاص از خصایص را داشتند . بعنوان مثال هنگامیکه از فرترن برای نوشتن برنامه های موثر در کاربردهای علمی استفاده می کنیم ، برنامه های حاصله برای کدهای سیستم چندان مناسب نیست . زبان بیسیک با اینکه براحتی آموخته می شود ، اما قدرت زیادی نداشته و عدم ساخت یافتگی آن در برنامه های بزرگ مشکل آفرین خواهد شد . از زبان اسمبلی برای تولید برنامه های کاملا" موثر استفاده می شود ، اما آموزش و کار با این زبان بسیار مشکل است . بعلاوه اشکال زدایی کدهای اسمبلی بسیار طاقت فرساست . مشکل اصلی دیگر این بود که زبانهای اولیه برنامه نویسی نظیر بیسیک ، کوبول و فرترن براساس اصول ساخت یافته طراحی نشده بودند . این زبانها از Goto بعنوان ابزارهای اولیه کنترل برنامه استفاده می کردند . در نتیجه ، برنامه های نوشته شده با این زبانها تولید باصطلاح " کدهای اسپاگتی (spaghetti code)" می کردند منظور مجموعه ای در هم تنیده از پرشها و شاخه های شرطی است که درک یک برنامه طولانی را ناممکن می سازد . اگر چه زبانهایی نظیر پاسکال ، ساخت یافته هستند اما فاقد کارایی لازم بوده و جنبه های ضروری برای کاربرد آنها در طیف وسیعی از برنامه ها وجود ندارد . (بخصوص ویرایش پاسکال استاندارد فاقد ابزارهای کافی برای استفاده در سطح کدهای سیستم بود (. تا قبل از ابداع زبان C ، زبان دیگری قدرت نداشت تا خصلتهای متضادی که در زبانهای قبلی مشاهده میشد ، را یکجا گردآوری کند . نیاز به وجود یک چنین زبانی شدیدا" احساس میشد. در اوایل دهه 1970 میلادی ، انقلاب رایانه ای در حال شکل گیری بود و تقاضا برای انواع نرم افزارها فشار زیادی روی برنامه نویسان و تواناییهای ایشان اعمال میکرد. درمراکز آموزشی تلاش مضاعفی برای ایجاد یک زبان برنامه نویسی برتر انجام می گرفت . اما شاید از همه مهمتر تولید و عرضه انبوه سخت افزار کامپیوتری بود که بعنوان یک نیروی ثانویه روی زبانهای برنامه نویسی عمل میکرد .دیگر رایانه ها و اسرار درونی آنها پشت درهای بسته نگهداری نمی شد . برای اولین بار بود که برنامه نویسان واقعا" دسترسی نامحدودی به اسرار ماشینهای خود پیدا نمودند . این امر زمینه تجربیات آزادانه را بوجود آورد . همچنین برنامه نویسان توانستند ابزارهای مورد نیازشان را ایجاد نمایند . با ظهور زبان C ، زمینه جهش های بزرگ در زبانهای برنامه نویسی مهیا شد . زبان C نتیجه توسعه تحقیقاتی درباره یک زبان قدیمی تر بنام Bcpl بود . زبانC اولین بار توسط Dennis Ritchie ابداع و روی ماشینهای DEC PDP-11 دارای سیستم عامل یونیکس اجرا شد . زبان Bcpl توسط Martin Richards توسعه یافته بود . Bcpl منجر به تولد زبان B

شد که توسط Ken thompson ابداع شد و سرانجام به زبان C منتهی شد . برای سالیان متمادی ، نسخه روایت استاندارد زبان C همانی بود که روی سیستم عامل unix عرضه و توسط Briian Kernighan و Dennis Ritchieو در کتاب"The C programming Language" توصیف شده بود . بعدا" در سال 1989 میلادی زبان C مجددا "استاندارد شد وو استاندارد ANSI برای زبان C انتخاب شد . بسیاری معتقدند که ایجاد زبان C راهگشای دوران جدیدی در زبانهای برنامه نویسی بوده است . این زبان بطور موفقیت آمیزی تناقضهای موجود در زبان های برنامه نویسی قبلی را مرتفع نمود . نتیجه فرآیند ایجاد زبان C ، یک زبان قدرتمند ، کارا و ساخت یافته بود که براحتی قابل آموزش و فراگیری بود . این زبان یک ویژگی غیر محسوس اما مهم داشت : زبان C ، زبان برنامه نویسان بود . قبل از ابداع

زبان C

زبانهای برنامه نویسی یا جنبه های آموزشی داشته یا برای کارهای اداری طراحی میشد . اما زبان C چیز دیگری بود . این زبان توسط برنامه نویسان واقعی و درگیر با کارهای جدی ، طراحی و پیاده سازی شده و توسعه یافت . جنبه های مختلف این زبان توسط افرادی که با خود زبان سر و کار داشته و برنامه نویسی می کردند مورد بررسی ، آزمایش و تفکر و تفکر مجدد قرار گرفته بود . حاصل این فرآیند هم زبانی بود که برنامه نویسان آن را دوست داشتند . در حقیقت زبان C بسرعت مورد توجه برنامه نویسان قرار گرفت تا جایی که برنامه نویسان نسبت به C تعصب خاصی پیدا نمودند . این زبان مقبولیت و محبوبیت زیادی در بین برنامه نویسان یافت . بطور خلاصه زبان C توسط برنامه نویسان و برای برنامه نویسان طراحی شده است . بعدا "می بینید که جاوا نیز این ویژگی را از اجداد خود بارث برده است .

نیاز به C++

طی دهه 1970 و اوایل دهه 80 میلادی زبان C نگین انگشتری برنامه نویسان بود و هنوز هم در سطح وسیعی مورد استفده قرار می گیرد . از آنجاییکه C یک زبان موفق و سودمند بوده ، ممکن است بپرسید چه نیازی به زبانهای جدیدتر وجود داشته است . پاسخ شما یک کلمه یعنی پیچیدگی (Complexity) است . طی تاریخ کوتاه برنامه نویسی پیچیدگی فزاینده برنامه ها نیاز برای شیوه های بهتر مدیریت پیچیدگی را بوجود آورده است++C . پاسخی است به این نیاز مدیریت پیچیدگی برنامه ها که زمینه اصلی پیدایش C++ بوده است . شیوه های برنامه نویسی از زمان اختراع رایانه تاکنون بطور قابل توجهی تغییر نموده اند . بعنوان مثال ، هنگامیکه رایانه ها اختراع شدند ، برنامه نویسی با استفاده از دستور العملهای باینری (Binary) ماشین انجام می گرفت . مادامیکه برنامه ها شامل حدود چند دستور العمل بود ، این روش کارآیی داشت . بموازات رشد برنامه ها زبان اسمبلی ابداع شد تا برنامه نویسان بتوانند برنامه های بزرگتر و پیچیده تر را با استفاده از نشانه هایی که معرف دستورالعملهای ماشین بودند ، بنویسند . اما پیشرفت و رشد برنامه ها همچنان ادامه یافت و زبانهای سطح بالایی معرفی شدند که ابزارهای مناسب برای مدیریت پیچیدگی روزافزون برنامه ها را در اختیار برنامه نویسان قرار می دادند .

اولین زبان مطرح در این زمینه فرترن بود . اگر چه فرترن اولین گام در این مسیر بود، اما زبانی است که توسط آن برنامه

های تمیز و سهل الادراک نوشته میشود .در دهه 1960 میلادی برنامه نویسی ساخت یافته مطرح شد . با استفاده از زبانهای ساخت یافته ، برای اولین بار امکان راحت نوشتن برنامه های بسیار پیچیده بوجود آمد . اما حتی با وجود روشهای برنامه نویسی ساخت یافته ، هنگامیکه یک پروژه به اندازه معینی می رسید ، پیچیدگی آن از توان مدیریت برنامه نویس خارج می شد . در اوائل دهه 1980 میلادی بسیاری از پروژه های مدیریت برنامه نویسی از مرزهای برنامه نویسی ساخت یافته گذشتند . برای حل این قبیل مشکلات ، یک روش نوین برنامه نویسی ابداع شد . این روش را برنامه نویسی شی ئ گرا یا باختصار oop می نامند . با جزئیات بیشتری بعدا" در همین کتاب بررسی خواهد شد ، اما توصیف مختصر این روش عبارت است از oop : یک نوع روش شناسی برنامه نویسی است که امکان سازماندهی برنامه های پیچیده از طریق بهره گیری از سه روش : وراثت ، کپسول سازی و چند شکلی ، را ایجاد می کند . در تحلیل نهایی ، اگر چه C بزرگترین و مهمترین زبان برنامه نویسی جهان است

اما محدودیتهایی در مدیریت پیچیدگی برنامه ها دارد . هنگامیکه یک برنامه از محدوده 25000 تا 100000 خط از کدها تجاوز نماید، آنچنان پیچیده می شود که درک آن بعنوان یک برنامه کلی ناممکن خواهد شد C++ . این محدودیت را از بین برده و به برنامه نویس کمک می کند تا برنامه هایی از این بزرگتر را نیز درک و مدیریت نماید C++ . در سال 1979 میلادی توسط Bjarne stoustrup هنگامیکه در آزمایشگاه بل در Marry Hillایالت New jersy مشغول کار بود ، ابداع شد . او در ابتدا این زبان جدید را (C with classes) نامید . اما در سال 1983 میلادی نام این زبان جدید به C++ تغییر یافت C++ . تداوم زبان C بود که جنبه های oop نیز به آن اضافه می شد .از آنجایی که زبان C++ براساس زبان C شکل گرفته ، در بر گیرنده کلیه جنبه ها خصلتها (attributes)و مزایای زبان C می باشد . این عوامل دلایل قاطعی برای موفقیت حتمی C++ بعنوان یک زبان برنامه نویسی هستند . ابداع C++ در حقیقت تلاشی برای ایجاد یک زبان کاملا" جدید برنامه نویسی نبود. در حقیقت پروژه C++ منجر به افزایش تواناییهای زبان موفق C شد . چون C++ از ابتدای تولد تاکنون دارای روایتهای گوناگونی شده است ، در حال حاضر در تلاش استاندارد نمودن این زبان هستند . (اولین روایت پیشنهادی ANSIبرای استاندارد C++ در سال 1994 میلادی مطرح شد .)

برنامه نویسی شی ئ گرا object-oriented programming

برنامه نویسی شی ئ گرا هسته اصلی جاوا است . در حقیقت کلیه برنامه های جاوا شی ئ گرا هستند . بر خلاف C++که در آن امکان گزینش شی ئ گرایی وجود دارد . روشهای oop آنچنان با زبان جاوا پیوستگی دارند که حتی قبل از نوشتن یک برنامه ساده جاوا نیز باید اصول oop را فرا گیرید . بهمین دلیل این فصل را با بحث جنبه های نظری oop آغاز می کنیم . می دانید که کلیه برنامه های کامپیوتری دارای دو عضو هستند : کد و داده . علاوه بر این ، یک برنامه را میتوان بطور نظری حول محور کد یا داده اش سازماندهی نمود . یعنی بعضی برنامه ها حول محور " آنچه در حال اتفاق است " (کد (نوشته شده و سایر برنامه ها حول محور " آنچه تحت تاثیر قرار گرفته است

) "داده (نوشته می شوند . اینها دو الگوی مختلف ساخت یک برنامه هستند . روش اول را مدل پردازش گرا (process-oriented model) می نامند . در این روش یک برنامه بعنوان کدهای فعال روی داده ها در نظر گرفت . زبانهای رویه ای (procedural) نظیر C از این مدل بنحو موفقیت آمیزی استفاده می کنند . اما همانطوریکه در قسمتهای قبل عنوان شد ، بموازات رشد و گسترش برنامه ها ، این روش منجر به بروز مشکلات بیشتر و پیچیده تری خواهد شد . برای مدیریت پیچیدگی فزاینده ، دومین روش معروف به برنامه نویسی شی ئ گرا پیشنهاد شده است . برنامه نویسی شی ئ گرا یک برنامه را حول محور داده های آن یعنی اشیائ و یک مجموعه از رابطها (interfaces)خوش تعریف برای آن داده ها سازماندهی می کند . یک برنامه شی ئ گرا را می توان بعنوان داده های کنترل کننده دسترسی به کدها (data controlling access to code) تلقی نمود . بعدا" خواهید دید با شروع بکار واحد داده های کنترل کننده بسیاری ازمزایای این نوع سازماندهی نصیب شما خواهد شد .

چرا نام جاوا ؟

در سال 1991 میلادی در شرکت Sun Micro Systemsمتولد شد. این پروژه در ابتدا پروژه سبز نام داشت. سرپرستی پروژه را James Goslingبه عهده داشت. نتیجه کار بر این پروژه زبان oakبود که در سال 92 ایجاد شد oak.به معنای بلوط است و زمانی که جیمز از پنجره اتاق کارش به یک درخت بلوط نگاه می کرد، این نام را برگزید؛ اما پس از مدتی شرکت Sunتصمیم گرفت نامی بهتر برای محصول خود برگزیند. بنابراین افراد تیم پروژه سبز به یک کافی شاپ نزدیک شرکت رفتند، تا نامی دیگر برای این زبان انتخاب کنند. پس از نصف روز بحث و بررسی JAVA، که مخفف نامهای James Gosling، Arthur Van hoff و Andy bechtolsheim است به عنوان نام این زبان انتخاب شد. از آنجا که مراسم نامگذاری در کافی شاپ برگزار شده بود، یک فنجان قهوه داغ به عنوان نماد جاوا در نظر گرفته شد.

تجرید Abstraction

تجرید یک عنصر ضروری در برنامه نویسی شی ئ گرا است . افراد پیچیدگی ها را با استفاده از تجرید مدیریت می نمایند . بعنوان نمونه ، مردم درباره اتومبیل هرگز بعنوان مجموعه ای از هزاران قطعات منفک از هم تفکر نمیکنند. آنها اتومبیل را بعنوان یک شی ئ خوب تعریف شده دارای نوعی رفتار منحصر بفرد تلقی می کنند . این تجرید به مردم امکان می دهد تا از یک اتومبیل استفاده نموده و به خواربار فروشی بروند ، بدون اینکه نگران پیچیدگی اجزایی باشند که یک اتومبیل را تشکیل می دهند .

آنها قادرند براحتی جزئیات مربوط به نحوه کار موتور ، سیستم های انتقال و ترمز را نادیده بگیرند . در عوض آنها مختارند تا از اتومبیل بعنوان یک شی ئ کلی استفاده نمایند .

یکی از شیوه های قدرتمند مدیریت تجرید با استفاده از طبقه بندیهای سلسله مراتبی (hierarchical) انجام می گیرد . این امر به شما امکان می دهد تا معنی و مفهوم سیستم های پیچیده را کنار گذاشته و آنها را به اجزائ کوچک قابل مدیریت تقسیم نمایید. از دید بیرونی ، یک اتومبیل یک شی ئ منفرد است . از دید داخلی اتومبیل شامل چندین زیر سیستم است : فرمان ، ترمزها ، سیستم صوتی ، کمربندهای ایمنی ، سیستم حرارتی ، تلفن سلولی و غیره . هر یک از این زیر سیستم ها بنوبه خود از واحدهای تخصصی کوچکتری تشکیل شده اند. بعنوان نمونه ، سیستم صوتی اتومبیل شامل یک رادیو، یک پخش CD و یا یک پخش صوت است . نکته مهم این است که شما بدین ترتیب بر پیچیدگی اتومبیل (یا هر سیستم پیچیده دیگر) با استفاده از تجرید سلسله مراتبی ، فائق می آیید . تجریدهای سلسله مراتبی سیستم های پیچیده را می توان در مورد برنامه های کامپیوتری نیز پیاده سازی نمود. داده های یک برنامه پردازش گرای سنتی را می توان توسط تجرید اشیائ عضو آن ، منتقل نمود . یک ترتیب از مراحل پردازش را می توان به مجموعه ای از پیامها بین اشیائ تبدیل نمود. بدین ترتیب ، هر یک از این اشیائ رفتار منحصر بفرد خودش را تعریف خواهد کرد . می توانید با این اشیائ بعنوان موجودیتهای واقعی رفتار کنید که به پیامهایی که به آنها می گویند چکاری انجام دهند ، مرتبط و وابسته هستند . این هسته اصلی برنامه نویسی شی ئ گراست . مفاهیم شی ئ گرایی در قلب جاوا قرار گرفته اند ، همچنانکه پایه اصلی ادراکات بشری نیز هستند. مهم اینست که بفهمید این مفاهیم چگونه در برنامه های کامپیوتری پیاده سازی می شوند . خواهید دید که برنامه نویسی شی ئ گرا یک نمونه قدرتمند و طبیعی برای تولید برنامه هایی است که بر تغییرات غیر منتظره فائق آمده و چرخه حیات هر یک از پروژه های نرم افزاری اصلی

(شامل مفهوم سازی ، رشد و سالخوردگی (را همراهی مـی کننـد . بعنـوان نمونـه ، هـر گـاه اشـیائ خـوش تعریـف و رابطهای تمیز و قابل اطمینان به این اشیائ را در اختیار داشته باشید ، آنگاه بطـور دلپـذیری میتوانیـد قسـمتهای مختلـف یک سیستم قدیمی تر را بدون ترس جابجا نموده یا بکلی از سیستم خارج نمایید .

سه اصل oop

کلیه زبانهای برنامه نویسی شی ئگرا مکانیسمهایی را در اختیار شما قرار میدهند تا مدل شی ئگرا را پیـاده سـازی نماییـد. ایــن مــدل شــامل کپســول ســازی (encapsulation) وراثــت (inheritance) و چنــد شــکلی (polymorphism)می باشد . اکنون نگاه دقیقتری به این مفاهیم خواهیم داشت .

کپسول سازی encapsulation

کپسول سازی مکانیسمی است که یک کد و داده مربوط با آن کد را یکجا گرد آوری نموده (در یک کپسول فرضـی قرار داده) و کپسول بدست آمده را در مقابل دخالت یا سوئ استفاده های غیـر مجـاز محافظـت مـی نمایـد . مـی تـوان کپسول سازی را بعنوان یک لفافه (wrapper) درنظر گرفت که کد داده مربوطه را نسبت به دستیابیهای غیرمعمول و غیر منتظره سایر کدهای تعریف شده در خارج از لفافه محافظت می کند . دستیابی به کد و داده موجود داخـل لفافـه از طریق رابطهای خوب تعریف شده کنترل خواهد شد .در دنیای واقعی ، سیستم انتقـال اتومـاتیـک در یـک اتومبیـل را درنظر بگیرید . این سیستم صدها بیت از اطلاعات درباره موتور اتومبیل شما را کپسول سازی می کنـد : مثـل سـرعت حرکت شما ، شیب سطح در حال حرکت و موقعیت اهرم انتقال . شما بعنوان یک کاربر قفط یک راه برای تاثیر نهادن در این کپسول سازی پیچیده خواهید داشت : بوسیله تغییر اهرم انتقال دهنده ، اما با اسـتفاده از سـیگنالهای برگشـتی یـا برف پاک کن شیشه جلو ، نمی توانید روی سیستم انتقال قدرت اتومبیل تاثیری بگذارید .

بنابراین دست دنده (اهرم انتقال دنده) یک رابط خوب تعریف شده والبتـه منحصـر بفـرد بـرای سیسـتم انتقـال اسـت . مضاف بر اینکه آنچه درون سیستم انتقال اتفاق می افتد ، تاثیری بر اشیائ خارج از سیستم نخواهد داشت . بعنوان مثال ، دنده های انتقال ، چراغهای جلو اتومبیل را روشن نمی کنند . از آنجاییکه سیستم انتقال اتوموبیلها کپسول سازی شـده ،

دهها تولید کننده اتومبیل قادرند سیستم های انتقال دلخواه خود را طراحی و پیاده سازی نمایند . اما از نقطه نظر کاربر اتومبیل همه این سیستم ها یکسان کار می کنند. درست همین ایده را می توان در برنامه نویسی کامپیوتر نیز پیاده سازی نمود . قدرت کدهای کپسول شده در این است که هر کسی می داند چگونه به آنها دسترسی یافته و میتواند صرفنظر از جزئیات اجرا و بدون ترس از تاثیرات جانبی از آنها استفاده نماید .

در جاوا کپسول سازی بر اساس کلاس (class) انجام می گیرد . اگر چه کلاس با جزئیات بیشتری در این کتاب بررسی خواهد شد ، اما بحث مختصر بعدی در این مورد سودمند خواهد بود .کلاس توصیف کننده ساختار و رفتاری (داده و کد) است که توسط یک مجموعه از اشیائ اشاعه خواهد یافت . هر شی ئ در یک کلاس شامل ساختار و رفتار تعریف شده توسط همان کلاس است . بهمین دلیل ، به اشیائ گاهی " نمونه هایی از یک کلاس " نیز می گویند . بنابراین ، یک کلاس یک ساختار منطقی است ، یک شی ئ دارای واقعیت فیزیکی است . وقتی یک کلاس بوجود می آورید ، در حقیقت کد و داده ای که آن کلاس را تشکیل می دهند ، مشخص می نمایید . این عناصر را اعضائ (members) یک کلاس می نامند .

بطورمشخص ، داده تعریف شده توسط کلاس را بعنوان متغیرهای عضو (member variables)یا متغیرهای نمونه (instance variables)می نامند . کدی که روی آن داده ها عمل می کند را روشهای عضو member methodsیا فقط روشها (methods) می نامند . اگر باC و++C و آشنا باشید می دانید که روش در برنامه نویسی جاوا همان تابع (function) در زبانهایC و++C و می باشد . در برنامه های خوب نوشته شده جاوا روشها توصیف کننده چگونگی استفاده از متغیرهای عضو هستند . یعنی که رفتار و رابط یک کلاس توسط روشهایی تعریف می شوند که روی داده های نمونه مربوطه عمل می کنند .

چون هدف یک کلاس پیاده سازی کپسول سازی برای موارد پیچیده است ، روشهایی برای پنهان کردن پیچیدگی اجزائ در داخل یک کلاس وجود دارد. هر روش یا متغیر داخل یک کلاس ممکن است خصوصی privateیاعمومی public باشد . رابط عمومی یک کلاس ، معرفی کننده هر چیزی است که کاربران خارج از کلاس نیاز به دانستن آنها دارند . روشها و داده های خصوصی فقط توسط کدهای عضو یک کلاس قابل دسترسی

هستند . بنابراین هر کد دیگری که عضو یک کلاس نباشد، نمی تواند به یک روش خصوصی دسترسی داشته باشد . چون اعضائ خصوصی یک کلاس ممکن است فقط توسط سایر بخشهای برنامه شما از طریق روشهای عمومی کلاس قابل دسترسی باشند، می توانید مطمئن باشید که فعل و انفعالات غیر مناسب اتفاق نخواهد افتاد . البته ، این بدان معنی است که رابط عمومی باید با دقت طراحی شود تا کارکرد داخلی یک کلاس را چندان زیاد برملا نکند .

وراثت inheritance

وراثت رویه ای است که طی آن یک شی ئ ویژگیهای شی ئ دیگری را کسب می کند . این موضوع بسیار اهمیت دارد زیرا از مفهوم طبقه بندی سلسله مراتبی حمایت می کند . همانطوریکه قبلا" گفتیم ، بسیاری از دانشها توسط طبقه بندی سلسله مراتبی قابل فهم و مدیریت میشوند .بعنوان مثال سگهای شکاری طلایی یکی از انواع طبقه بندیهای سگها هستند که بنوبه خود جزئی از کلاس پستانداران خونگرم بوده که در کلاس بزرگتری تحت عنوان حیوانات قرار می گیرند . بدون استفاده از سلسله مراتب ، باید خصوصیات هر یک از اشیائ را جداگانه توصیف نمود . اما هنگام استفاده از سلسله مراتب ، برای توصیف یک شی ئ کافی است کیفیتهایی که آن شی ئ را در کلاس مربوطه منحصر بفرد و متمایز می سازد ، مشخص نمایید . آن شی ئ ممکن است خصوصیات عمومی را از والدین خود بارث برده باشد .

بدین ترتیب در مکانیسم وراثت ، یک شی ئ می تواند یک نمونه مشخص از یک حالت عمومی تر باشد . اجازه دهید تا با دقت بیشتری به این رویه نگاه کنیم . بسیاری از افراد، دنیا را بطور طبیعی بعنوان مجموعه ای از اشیائ می دانند که در یک روش سلسله مراتبی بیکدیگر مرتبط شده اند . نظیر حیوانات ، پستانداران و سگها . اگر بخواهید حیوانات را با یک روش تجریدی توصیف نمایید ، باید برخی خصوصیات نظیر اندازه ، هوش و نوع اسکلت آنها را مشخص نمایید . حیوانات همچنین ویژگیهای خاص رفتاری دارند ، آنها تغذیه نموده ، تنفس کرده و می خوابند . این توصیف از خصوصیات و رفتار را توصیف " کلاس " حیوانات می نامند . اگر بخواهید توصیف یک کلاس مشخصتر از حیوانات مثل پستانداران داشته باشید .

باید خصوصیات دقیقتری نظیر نوع دندانها و آلات پستانداری را مشخص نمایید . این را یک زیر کلاس (subclass)از حیوانات و خود حیوانات را کلاس بالای (super class) پستانداران گویند . چون پستانداران نوعی از حیوانات هستند ، بنابراین کلیه خصوصیات حیوانات را بارث برده اند. یک زیر کلاس ارث برنده درحقیقت کلیه خصوصیات اجداد خود در سلسله مراتب کلاس را به ارث می برد .

وراثت و کپسول سازی ارتباط دو جانبه و فعل و انفعالی دارند . اگر یک کلاس برخی از خصوصیات را کپسول سازی کند ، آنگاه هر یک از زیر کلاسها همان خصوصیات بعلاوه برخی ویژگیهای خاص خود را خواهند داشت . همین مفهوم ساده و کلیدی است که به برنامه نویسی شی ئ گرا امکان داده تا در پیچیدگیها بجای روش هندسی ، بروش خطی توسعه یابد . یک زیر کلاس جدید کلیه خصوصیات از کلیه اجداد خود را بارث می برد . این امر باعث شده تافعل و انفعالات غیر قابل پیش بینی صادره از کدهای دیگر موجود در سیستم وجود نداشته باشد .

چند شکلی polymorphism

چند شکلی مفهومی است که بوسیله آن یک رابط (interface) را می توان برای یک کلاس عمومی از فعالیتها بکار برد . فعالیت مشخص توسط طبیعت دقیق یک حالت تعریف می شود . یک پشته را در نظر بگیرید (که در آن هر چیزی که آخر آمده ، ابتدا خارج می شود) . ممکن است برنامه ای داشته باشید که مستلزم سه نوع پشته باشد . یک پشته برای مقادیر عدد صحیح ، یکی برای مقادیر اعشاری و یکی هم برای کاراکترها لازم دارید. الگوریتمی که این پشته ها را اجرا میکند، یکسان است ، اگرچه داده های ذخیره شده در هر یک از این پشته ها متفاوت خواهد بود. در یک زبان شی ئ گرا ، شما باید سه مجموعه متفاوت از روالهای (routines) پشته ایجاد نمایید ، و برای هر مجموعه اسامی متفاوت اختیار کنید .اما براساس مفهوم چند شکلی در جاوا میتوانید یک مجموعه کلی از روالهای پشته را مشخص نمایید که همگی از یک نام استفاده کنند .در حالت کلی مفهوم چند شکلی را می توان با عبارت " یک رابط و چندین روش " توصیف نمود . بدین ترتیب قادر هستید یک رابط ژنریک (generic) را برای گروهی از فعالیتهای بهم مرتبط طراحی نمایید . با طراحی یک رابط برای مشخص نمودن یک کلاس عمومی از فعالیتها، می توان پیچیدگی برنامه ها را کاهش داد. این وظیفه کامپایلر است تا عمل مشخصی (منظور همان روش است) را برای هر یک از حالات مختلف انتخاب نماید . شما بعنوان یک برنامه نویس لازم نیست این انتخاب را بصورت دستی انجام دهید . شما فقط کافی است رابط کلی را بیاد سپرده و از آن به بهترین وجه ممکن استفاده نمایید .

در مثال مربوط به سگها ، حس بویایی سگ نوعی چند شکلی است . اگر سگ بوی یک گربه را استشمام کند ، پارس کرده و بدنبال گربه خواهد دوید . اگر سگ بوی غذا را استشمام کند ، بزاق دهانش ترشح کرده و بطرف ظرف غذا حرکت خواهد کرد . در هر دو حالت این حس بویایی سگ است که فعالیت می کند . تفاوت در آن چیزی است که

استشمام می شود ، یعنی نوع داده ای که به سیستم بویایی سگ وارد می شود . همین مفهوم کلی را مـی تـوان در جـاوا پیاده سازی نمود و روشهای متفاوت درون برنامه های جاوا را ساماندهی کرد .

چند شکلی ، کپسول سازی و وراثت در تقابل با یکدیگر کار می کنند هنگامیکه مفـاهیم چنـد شـکلی ، کپسولسـازی و وراثت را بطور موثری تلفیق نموده و برای تولید یک محیط برنامـه نویسـی بکـار بـریم ، آنگـاه برنامـه هـایی تنومنـد و غیر قابل قیاس نسبت به مدلهای رویه گرا خواهیم داشت . یک سلسله مراتـب خـوب طراحـی شـده ازکلاسـها، پایـه ای است برای استفاده مکرر از کدهایی که برای توسعه و آزمایش آنها وقـت و تـلاش زیـادی صـرف نمـوده ایـد. کپسـول سازی به شما امکان می دهد تا کدهایی را که به رابط عمومی برای کلاسهای شما بستگی دارند ، بدون شکسـته شـدن برای پیاده سازیهای دیگر استفاده نمایید . چند شکلی به شما امکان می دهد تا کدهای تمیز قابل حس ، قابل خوانـدن و دارای قابلیت ارتجاعی ایجاد نمایید .

از دو مثالی که تاکنون استفاده شده ، مثال مربوط به اتومبیل کاملا" قدرت طراحی شی ئ گرا را توصـیف مـی کنـد . از نقطه نظر وراثت ، سگها دارای قدرت تفکر درباره رفتارها هستند ، اما اتوموبیلها شباهت بیشتری با برنامه های کامپیوتری دارند . کلیه رانندگان وسائط نقلیه با اتکائ به اصل وراثت انواع مختلفی (زیر کلاسـها) از وسـائط نقلیـه را مـی راننـد . خواه اتومبیل یک مینی بوس مدرسه ، یا یک مرسدس بنز ، یا یک پورشه ، یا یک استیشن خـانوادگی باشـد ، کمـابیش یکسان عمل کرده ، همگی دارای سیستم انتقال قدرت ، ترمز ، و پدال گاز هستند . بعد از کمی تمرین با دنده های یک اتومبیل ، اکثر افراد براحتی تفاوت بین یک اتومبیل معمولی با یک اتومبیل دنده اتوماتیک را فرا می گیرنـد ، زیـرا افـراد بطور اساسی کلاس بالا ، یعنی سیستم انتقال را درک می کنند .

افرادی که با اتومبیل سر و کار دارند همواره با جوانب کپسول شده ارتباط دارند . پدالهای گاز و ترمز رابطهایی هسـتند که پیچیدگی سیستم های مربوطه را از دید شما پنهان نموده تا بتوانید براحتی و سهولت با این سیستم های پیچیـده کـار کنید . پیاده سازی یک موتور ، شیوه های مختلف ترمز و اندازه تایرهای اتومبیل تاثیری بر ارتباط گیری شما با توصیف کلاس پدالها نخواهند گذاشت.

آخرین خصوصیت ، چند شکلی ، بوسیله توانایی کارخانه های اتومبیل سازی برای اجـرای طیـف وسـیعی از گزینـه هـا روی یک وسیله نقلیه منعکس می شود . بعنوان مثال کارخانه ممکن است از سیستم ترمز ضـد قفـل یـا همـان ترمزهـای معمولی ، فرمان هیدرولیک یا چرخ دنده ای ، نیز از موتورهای 6،4 ،یا 8 سیلندر استفاده نماید
. در هر حال شما روی پدال ترمز فشار می دهید تا اتومبیـل متوقـف شـود ، فرمـان را مـی چرخانیـد تـا جهـت حرکـت اتومبیل را تغییر دهید ، و برای شروع حرکت یا شتاب بخشیدن به حرکت روی پدال گاز فشار می دهید . در این موارد از یک رابط برای ایجاد کنترل روی تعداد متفاوتی از عملکردها استفاده شده است

کاملا" مشخص است که استفاده از مفاهیم کپسول سازی ، وراثت و چند شکلی باعث شده تا اجزائمنفـک بـا یکـدیگر ترکیب شده و تشکیل یک شی ئ واحد تحت عنوان اتومبیل را بدهند . همین وضعیت در برنامه های کامپیوتری مشاهده می شود . بوسیله استفاده از اصول شی ئ گرایی ، قطعات مختلف یک برنامه پیچیده را در کنار هم قرار مـی دهـیم تـا

یک برنامه منسجم ، تنومند و کلی حاصل شود در ابتدای این بخش گفتیم که کلیه برنامه نویسان جاوا خواه ناخواه شی ئ گرا کلیه برنامه نویسان جاوا با مفاهیم کپسول سازی ، وراثت و پلی مورفیسم آشنا خواهند شد.

مروری بر جاوا

نظیر سایر زبانهای برنامه نویسی کامپیوتر ، عناصر و اجزائ جاوا مجرد یا منفک از هم نیستند . این اجزائ در ارتباط تنگاتنگ با یکدیگر سبب بکار افتادن آن زبان می شوند . این پیوستگی اجزائ در عین حال توصیف یکی از وجوه خاص این زبان را مشکل می سازد . غالبا" بحث درباره یکی از جوانب این زبان مستلزم داشتن اطلاعات پیش زمینه در جوانب دیگر است . در قسمتهای بعدی به شما امکان داده برنامه های ساده ای توسط زبان جاوا نوشته و درک نمایید. جاوا یک زبان کاملا" نوع بندی شده است

در حقیقت بخشی از امنیت و قدرتمندی جاوا ناشی از همین موضوع است . اکنون بینیم که معنای دقیق این موضوع چیست . اول اینکه هر متغیری یک نوع دارد ، هر عبارتی یک نوع دارد و بالاخره اینکه هر نوع کاملا" و دقیقا" تعریف شده است . دوم اینکه ، کلیه انتسابها (assignments) خواه بطور مستقیم و صریح یا بوسیله پارامترهایی که در فراخوانی روشها عبور می کنند ، از نظر سازگاری انواع کنترل می شوند . بدین ترتیب اجبار خودکار یا تلاقی انواع در هم پیچیده نظیر سایر زبانهای برنامه نویسی پیش نخواهد آمد. کامپایلر جاوا کلیه عبارات و پارامترها را کنترل می کند تا مطمئن شود که انواع ، قابلیت سازگاری بهم را داشته باشند . هر گونه عدم تناسب انواع ، خطاهایی هستند که باید قبل از اینکه کامپایلر عمل کامپایل نمودن کلاس را پایان دهد ، تصحیح شوند .

نکته : اگر دارای تجربیاتی در زبانهایC و++C و هستید ، بیاد بسپارید که جاوا نسبت به هر زبان دیگری نوع بندی شده تر است . بعنوان مثال درC و++C و می توانید یک مقدار اعشاری را به یک عدد صحیح نسبت دهید . همچنین در زبان C کنترل شدید انواع بین یک پارامتر و یک آرگومان انجام نمی گیرد . اما در جاوا این کنترل انجام می گیرد . ممکن است کنترل شدید انواع در جاوا در وهله اول کمی کسل کننده بنظر آید . اما بیاد داشته باشید که اجرای این امر در بلند مدت سبب کاهش احتمال بروز خطا در کدهای شما

چرا جاوا برای اینترنت اهمیت دارد

اینترنت جاوا را پیشاپیش زبانهای برنامه نویسی قرار داد و در عوض جاوا تاثیرات پیش برنده ای روی اینترنت داشته است . دلیل این امر بسیار ساده است : جاوا سبب گسترش فضای حرکت اشیائ بطور آزادانه در فضای الکترونیکی می شود . در یک شبکه ، دو نوع طبقه بندی وسیع از اشیائ در حال انتقال بین سرویس دهنده و رایانه شخصی شما وجود دارد : اطلاعات غیر فعال (passive) و برنامه های فعال (active)و پویا . (dynamic) بعنوان نمونه هنگامیکه پست الکترونیکی e-mail خود را مرور می کنید ، در حال بررسی داده های غیر فعال هستید . حتی هنگامیکه یک برنامه را گرفته و بار گذاری می کنید ، مادامیکه از آن برنامه استفاده نکنید کدهای برنامه بعنوان داده های غیر فعال هستند . اما نوع دوم اشیائی که امکان انتقال به رایانه

شخصی شما را دارند ، برنامه های پویا و خود اجرا هستند . چنین برنامه ای اگر چه توسط سرویس دهنده ارائه و انتقال می یابد ، اما یک عامل فعال روی رایانه سرویس گیرنده است . بعنوان نمونه سرویس دهنده قادر است برنامه ای را بوجود آورد که اطلاعات و داده های ارسالی توسط سرویس دهنده را نمایش دهد . بهمان اندازه که برنامه های پویا و شبکه ای شده موردتوجه قرار گرفته اند بهمان نسبت نیز دچار مشکلاتی در زمینه امنیت و قابلیت حمل هستند . قبل از جاوا ، فضای الکترونیکی شامل فقط نیمی از ورودیهایی بود که اکنون وجود دارند . همانطوریکه خواهید دید ، جاوادرها رابرای یک شکل جدید از برنامه ها باز نموده است :

ریز برنامه ها (applets)

ریز برنامه ها و برنامه های کاربردی جاوا از جاوا برای تولید دو نوع برنامه می توان استفاده نمود: برنامه های کاربردی (applications)و ریز برنامه ها . (applets) یک برنامه کاربردی برنامه ای است که روی رایانه شما و تحت نظارت یک سیستم عامل اجرا می شود . بدین ترتیب یک برنامه کاربردی ایجاد شده توسط جاوا مشابه برنامه های ایجاد شده توسطC و++C و خواهد بود. هنگامیکه از جاوا برای تولید برنامه های کاربردی استفاده میکنیم ، تفاوتهای زیادی بین این زبان و سایر زبانهای برنامه نویسی مشاهده نمی کنیم . اما ویژگی جاوا برای تولید ریز برنامه ها دارای اهمیت زیادی است . یک ریز برنامه (applet) یک برنامه کاربردی است که برای انتقال و حرکت روی اینترنت و اجرا توسط یک مرورگر قابل انطباق با جاوا طراحی شده است . یک ریز برنامه در حقیقت یک برنامه ظریف جاوا است که بطور پویا در سراسر اینترنت قابل بارگذاری باشد ، درست مثل یک تصویر، یک فایل صوتی یا یک قطعه ویدئویی .تفاوت اصلی در اینست که ریزبرنامه یک برنامه کاربردی هوشمند است و شباهتی با یک تصویر متحرک یا فایل رسانه ای ندارد . بعبارت دیگر این برنامه قادر به عکس العمل در برابر ورودی کاربر و ایجاد تغییرات پویا است .

ریز برنامه های جاوا بسیار جالب و هیجان انگیزند و قادرند دو مشکل اصلی یعنی امنیت و قابلیت حمل را پشت سر بگذارند . قبل از ادامه بحث بهتر است مفهوم اصلی این دو مشکل را بیشتر مورد بررسی قرار دهیم .

امنیت

همانطوریکه خودتان هشیار هستید ، هرگاه که یک برنامه عادی (normal) را بار گذاری می کنید با خطر یک حمله ویروسی مواجه خواهید شد . قبل از جاوا اکثر کاربران ، برنامه های قابل اجرا را بتناوب گرفته و بارگذاری می کردند و قبل از اجرا برای ویروس زدایی اقدام به اسکن (Scanning) برنامه ها می کردند . با این حال بسیاری از این کاربران نسبت به حمله ویروسها به سیستم خود نگران بودند . علاوه بر ویروسها ، نوع دیگری از برنامه های مزاحم وجود دارند که باید در برابر آنها نیز ایمن ماند . این نوع برنامه ها قادرند اطلاعات خصوصی نظیر شماره کارتهای اعتباری ،ترازهای حساب بانکی و کلمات عبور برای جستجو درسیستم فایلهای محلی رایانه شما را کشف نموده و استفاده نمایند . جاوا توسط ایجاد یک دیواره آتش (firewall) بین رایانه شما و برنامه شبکه ای شده ، بر این مشکلات فائق آمده است .

هنگامیکه از یک مرورگر قابل انطباق با جاوا در وب استفاده میکنید، میتوانید با اطمینان ریزبرنامه های جاوا را بارگذاری نمایید، بدون اینکه از حمله ویروسها و برنامه های مزاحم واهمه ای داشته باشید . جاوا یک برنامه خاص جاوا را به محیط خاص اجرایی مربوطه اش منحصر کرده و اجازه دسترسی این برنامه به سایر بخشهای رایانه را نمی دهد و بدین ترتیب مشکل امنیت را حل کرده است . توانایی بارگذاری ریز برنامه ها بصورت مطمئن یکی از مهمترین جنبه های جاوا است .

قابلیت حمل

انواع مختلفی از رایانه ها و سیستم های عامل در سراسر دنیا مورد استفاده قرار می گیرند و بسیاری ازاین سیستم ها با اینترنت متصل میشوند. برای اینکه برنامه ها بتوانند بطور پویا روی انواع مختلف سیستم ها و محیط های عامل متصل به اینترنت بارگذاری شوند ، وسائلی برای تولید کدهای اجرایی و قابل حمل مورد نیاز است . همانطوریکه بزودی خواهید دید ، همان مکانیسمی که امنیت را بوجود می آورد سبب ایجاد قابلیت حمل نیز خواهد شد .

خصلتهای جاوا

هیچ بحثی درباره اصل و نسب جاوا بدون بررسی خصلتهای آن کامل نخواهد شد. اگر چه امنیت و قابلیت حمل ، نیروهای اصلی تسریع کننده روند شکل گیری جاوا بودند اما عوامل دیگری هم وجود دارند که در شکل نهایی این زبان تاثیر داشتند . این ملاحظات کلیدی توسط تیم جاوا در اصطلاحات زیر و بعنوان خصلتهای جاوا معرفی شده اند .

- simple ساده
- secure ایمن
- portable قابل حمل
- object-oriented شی ئ گرا
- Robust قدرتمند
- Multithreaded چند نخ کشی شده
- Architecture-neutral معماری خنثی
- Interpreted تفسیر شده
- High-performance عملکرد سطح بالا
- Distributed توزیع شده
- Dynamic پویا

قبلا" دو تا از این خصلتها را بررسی کرده ایم : ایمن و قابل حمل . اکنون سایر خصلتهای جاوا را یک به یک بررسی خواهیم نمود .

ساده

جاوا طوری شده که برنامه نویسان حرفه ای بسادگی آن را فراگرفته و بطور موثری بکار می برند . فرض کنیم که شما تجربیاتی در برنامه نویسی دارید ، آنگاه برای کار با جاوا مشکل زیادی نخواهید داشت . اگر قبلا" با مفاهیم شی ئ گرایی آشنا شده باشید ، آنگاه آموختن جاوا باز هم آسان تر خواهد شد . از همه بهتر اینکه اگر یک برنامه نویس مجرب C++ باشید ، حرکت بطرف جاوا بسیار راحت انجام می گیرد . چون جاوا دستور زبانC و++C و وهمچنین بسیاری از جوانب شی ئ گرایی C++ را بارث برده ، اکثر برنامه نویسان برای کار با جاوا دچار مشکل نخواهند شد . علاوه بر اینکه بسیاری از مفاهیم پیچیده C++ یا در جاوا داخل نشده و یا با روشی آسان تر و ساده تر مورد استفاده قرار گرفته اند . فراسوی شباهتهای جاوا باC و++C و خاصیت دیگری در جاوا وجود دارد که فراگیری آن را بسیار ساده تر می کند : جاوا تلاش کرده که جنبه های استثنایی و خارق العاده نداشته باشد . در جاوا ، تعداد اندکی از شیوه های کاملا" توصیف شده برای انجام یک وظیفه وجود دارد .

شی ئ گرا

اگر چه این زبان از اجداد خود تاثیر گرفته ، اما طوری طراحی نشده تا کد منبع آن قابل انطباق با سایر زبانهای برنامه نویسی باشد . این خاصیت به تیم جاوا اجازه داده تا آزادانه روی یک تخته سنگ حکاکی نمایند . یکی از نتایج این آزادی در طراحی ، یک روش تمیز، قابل استفاده و واقعیت گرا برای اشیائ (objects) است . جاوا از بسیاری از محیط های نرم افزاری اولیه براساس اشیائ مواردی را به عاریت گرفته و توازنی بین نظریه لفظ قلمی (purist) تحت عنوان " هر چیزی یک شی . است " و نظریه واقعیت گرایی " جلوی راه من قرار نگیر " بوجود آورده است . مدل شی ئ در جاوا بسیار ساده و براحتی قابل گسترش است در حالیکه انواع ساده آن نظیر اعداد صحیح (integers) بعنوان عملکردهای سطح بالای غیر شی ئ تلقی می شوند .

قدرتمند

محیط چندگانه روی وب ایجاب کننده تقاضاهای غیر عادی برای برنامه هاست ، زیرا این برنامه ها باید در طیف وسیعی از سیستم ها اجرا شوند . بدین ترتیب در طراحی جاوا اولویت اول توانایی ایجاد برنامه های قدرتمند بوده است . برای کسب اطمینان جاوا شما را به تعداد محدودی از نواحی کلیدی محدود می کند تا مجبور شوید اشتباهات خود را در توسعه برنامه خیلی زود پیدا کنید . در همین حال جاوا شما را از نگرانی درباره بسیاری از اشتباهات رایج ناشی از خطاهای برنامه نویسی می رهاند . از آنجاییکه جاوا یک زبان کاملا" نوع بندی شده است ، هنگام کامپایل کد شما را کنترل می کند . اما این زبان کدهای شما را هنگام اجرا نیز کنترل می نماید . در حقیقت بسیاری از اشکالات هارددیسک به شیار که اغلب در حالتهای حین اجرا ایجاد می شوند ، در جاوا ناممکن شده اند . آگاهی بر اینکه آنچه

شما نوشته اید بصورتی قابل پیش بینی در شرایط متغیر عمل می کند ، یکی از جنبه های اصلی جاوا است .

برای درک بهتر قدرتمندی جاوا، دو دلیل عمده شکست برنامه ها را درنظر بگیرید :اشتباهات در مدیریت حافظه و شرایط استثنایی پیش بینی نشده (یعنی خطاهای حین اجرا . (مدیریت حافظه در محیطهای برنامه نویسی سنتی یکی از وظایف دشوار و آزار دهنده است . بعنوان نمونه درC و++C و برنامه نویس باید بصورت دستی کل حافظه پویا را تخصیص داده و آزاد نماید . این امر گاه منجر به بروز مشکلاتی می شود . بعنوان نمونه گاهی برنامه نویسان فراموش می کنند حافظه ای راکه قبلا "تخصیص یافته ، آزاد نمایند . یا از این بدتر ، ممکن است تلاش کنند حافظه ای را که توسط بخشی از کد ایشان در حال استفاده است ، آزاد نمایند . جاوا بوسیله مدیریت تخصیص حافظه و تخصیص مجدد حافظه واقعا" این مشکلات را از میان برداشته است . (در حقیقت تخصیص مجدد کاملا" خودکار انجام می گیرد، زیرا جاوا یک مجموعه سطل آشغال برای اشیائ غیر قابل استفاده تهیه نموده است .) شرایط استثنایی در محیط های سنتی اغلب در حالتهایی نظیر " تقسیم بر صفر "یا"file not found ا اتفاق می افتند و باید توسط ساختارهای بد ترکیب و زمخت مدیریت شوند . جاوا در این زمینه بوسیله تدارک اداره استثنائات شی ئ گرایی object-oriented این مشکل را حل کرده است . در یک برنامه خوش ساخت جاوا، کلیه خطاهای هنگام اجرا توسط برنامه شما مدیریت خواهد شد .

چند نخ کشی شده

جاوا برای تامین نیازمندیهای دنیای واقعی بمنظور ایجاد برنامه های شبکه ای و فعل و انفعالی (interactive) طراحی شده است . برای تکمیل این هدف ، جاوا از برنامه نویسی چند نخ کشی حمایت می کند که امکان نوشتن برنامه هایی به شما میدهد که در آن واحد چندین کار را انجام می دهند . سیستم حین اجرای جاوا ، یک راه حل زیبا و بسیار ماهرانه برای همزمانی چندین پردازش (process) ارائه می دهد که امکان ساخت سیستم های فعل و انفعالی که بنرمی اجرا میشوند را بوجود آورده است . راه حل سهل الاستفاده جاوا برای چند نخ کشی شده به شما امکان تفکر درباره رفتار خاص برنامه تان (و نه یک زیر سیستم از چند وظیفه ای) را می دهد .

معماری خنثی Architecture-Neutral

یکی از مشغولیتهای اساسی طراحان جاوا موضوع طول و قابلیت حمل کدها بود . یکی از مشکلات اصلی سر راه برنامه نویسان این است که تضمینی وجود ندارد تا برنامه ای را که امروز می نویسید فردا حتی روی همان ماشین اجرا شود. ارتقائ سیستم های عامل و پردازنده ها و تغییرات در منابع هسته ای سیستم ممکن است دست بدست هم داده تا یک برنامه را از کار بیندازند . طراحان جاوا تصمیمات متعدد و دشواری در جاوا و حین اجرا اتخاذ نمودند تا بتوانند این موقعیت را دگرگون نمایند . هدف آنها عبارت بود از : یکبار بنویسید ، هر جایی ، هر زمان و برای همیشه اجرا کنید . این هدف تا حد زیادی توسط طراحان جاوا تامین شد .

تفسیر شده و عملکرد سطح بالا

همانطوریکه دیدیم ، جاوا قدرت ایجاد برنامه هایی قابل انطباق با چندین محیط را بوسیله کامپایل کردن یک نوع معرفی واسطه تحت عنوان کد بایتی پیدا کرده است . این کدها روی هر نوع سیستمی که یک حین اجرای جاوا را فراهم نماید ، قابل اجرا می باشند . بسیاری از راه حلهای قبلی در زمینه برنامه های چند محیطه سبب کاهش سطح عملکرد برنامه ها شده بود .سایر سیستم های تفسیر شده نظیرBASIC، Tcl،وPERL و از کمبزدها و نارساییهای عملکرد رنج می بردند . اما جاوا طوری طراحی شده تا روی انواع cpu نیز بخوبی اجرا شود . اگر چه صحت دارد که جاوا تفسیر شده است اما کدهای بایتی جاوا آنچنان دقیق طراحی شده که می توان آنها را بسادگی و بطور مستقیم به کدهای ماشین خاص شما برای عملکردهای سطح بالا ترجمه نمود . سیستم های حین اجرای جاوا که این بهینه سازی " درست در زمان " را اجرا می کنند ، سبب از دست رفتن هیچیک از مزایای کدهای مستقل از زیربنا نخواهد شد. اکنون دیگر عملگرد سطح بالا و زیربناهای متعدد در تناقض با یکدیگر نیستند .

توزیع شده

جاوا مختص محیط توزیع شده اینترنت طراحی شده زیرا پروتکل های Tcp/ip را تبعیت می کند . در حقیقت ، دستیابی به یک منبع توسط آدرس URL چندان تفاوتی با دستیابی به یک فایل ندارد . روایت اولیه جاوا یعنی oak دربرگیرنده جنبه هایی برای پیام رسانی آدرسهای داخلی فضای الکترونیکی بود . این امر امکان می داد تا اشیائ روی دو نوع رایانه متفاوت ، پردازشهای از راه دور را اجرا نمایند . جاوا اخیرا"این رابطها را در یک بسته نرم افزاری بنام Remote Method Invocation (RMI)احیائ نموده است . این جنبه یک سطح غیر موازی از تجرد برای برنامه نویس سرویس گیرنده / سرویس دهنده بوجود آورده است .

پویا

همراه برنامه های جاوا، مقادیر قابل توجهی از اطلاعات نوع حین اجرا وجود دارد که برای ممیزی و حل مجدد دستیابی به اشیائ در زمان اجرا مورد استفاده قرار می گیرند . این امر باعث پیوند پویای کد در یک شیوه مطمئن و متهورانه می شود . این مسئله برای قدرتمندی محیط ریز برنامه (applet) کاملا" قاطع است ، جایی که اجرا بصورت پویا ارتقائ

جادوی جاوا کدهای بایتی The Byte codes

کلید اصلی جادوی جاوا برای حل مشکل امنیت و قابلیت حمل در این است که خروجی یک کامپایلر جاوا، کدهای قابل اجرا نیستند، بلکه کدهای بایتی هستند . کد بایتی یک مجموعه کاملا" بهینه شده از دستورالعمل هایی است که برای اجرا توسط یک ماشین مجازی (Virtual Machine) طراحی شده که معادل سیستم حین اجرای-(run time)جاوا باشد . یعنی سیستم حین اجرای جاوا یک مفسر (interpreter) برای کد بایتی است . این امر ممکن است تا حدی سبب شگفتی شود. همانطوریکه اطلاع دارید C++ به کد قابل اجرا کامپایل می شود . در حقیقت اکثر زبانهای برنامه نویسی مدرن طوری طراحی شده اند که قابل کامپایل نه قابل تفسیر باشند و این امر بلحاظ مسائل اجرایی است .اما این واقعیت که برنامه های جاوا قابل تفسیر شدن است به حل مشکلات پیوسته با بارگذاری برنامه ها روی اینترنت کمک می کند . دلیل آن روشن است .

جاوا بگونه ای طراحی شده تا یک زبان قابل تفسیر باشد. از آنجاییکه برنامه های جاوا قبل از آنکه قابل کامپایل باشند قابل تفسیر هستند، امکان استفاده از آنها در طیف گسترده ای از محیط ها وجود دارد . دلیل آن هم بسیار روشن است . فقط کافی است تا سیستم حین اجرای جاوا برای هر یک از محیط ها اجرا گردد . هنگامیکه بسته نرم افزاری حین اجرا برای یک سیستم خاص موجود باشد ، برنامه جاوا روی آن سیستم قابل اجرا خواهد شد . بیاد داشته باشید که اگر چه جزئیات سیستم حین اجرای جاوا از یک محیط تا محیط دیگر متفاوت است ، اما همه آنها یک کد بایتی جاوا را تفسیر می کنند . اگر جاوا یک زبان قابل کامپایل می بود ، آنگاه برای هر یک از انواع cpu متصل به اینترنت ، باید روایتهای مختلفی از جاوا نوشته می شد . این راه حل چندان قابل انطباق نیست. بنابراین "تفسیر" راحتترین شیوه برای ایجاد برنامه های واقعا" قابل حمل است .

این واقعیت که جاوا یک زبان قابل تفسیراست ، به مسئله امنیت هم کمک میکند .از آنجایی که اجرای هر یک برنامه های جاوا تحت کنترل سیستم حین اجرا انجام شده سیستم فوق می تواند برنامه را دربرگرفته و مانع تولید اثرات جانبی خارج از سیستم گردد . همانطوریکه خواهید دید ، مسئله امنیت نیز توسط محدودیت های خاصی که در زبان جاوا وجود دارد اعمال خواهد شد .

هنگامیکه یک برنامه تفسیر میشود، معمولا" کندتر از زمانی که به کدهای اجرایی کامپایل شود ، اجرا خواهد شد . اما در مورد جاوا این تفاوت در زمان اجرا چندان زیاد نیست .استفاده از کد بایتی امکان اجرای سریعتر برنامه هارا بوجود می آورد .یک نکته دیگر : اگر چه جاوا طوری طراحی شده تا تفسیر شود ، اما محدودیتی برای کامپایل کدهای بایتی آن به کدهای معمولی وجود ندارد . حتی اگر کامپایل پویا به کدهای بایتی اعمال شود ، همچنان جنبه های امنیتی و قابلیت حمل آن محفوظ می ماند ، زیرا سیستم حین اجرا همچنان درگیر محیط اجرایی می ماند . بسیاری از محیط های حین اجرای جاوا این روش " درست در زمان (just in time)" کامپایل نمودن کدهای بایتی به کدهای معمولی را مورد استفاده قرار می دهند . چنین عملکردی قابل رقابت با C++ می باشند .

جاوا یک زبان کاملا" نوع بندی شده است

در حقیقت بخشی از امنیت و قدرتمندی جاوا ناشی از همین موضوع است . اکنون ببینیم که معنای دقیق این موضوع چیست . اول اینکه هر متغیری یک نوع دارد ، هر عبارتی یک نوع دارد و بالاخره اینکه هر نوع کاملا" و دقیقا" تعریف شده است . دوم اینکه ، کلیه انتسابها (assignments) خواه بطور مستقیم و صریح یا بوسیله پارامترهایی که در فراخوانی روشها عبور می کنند ، از نظر سازگاری انواع کنترل می شوند . بدین ترتیب اجبار خودکار یا تلاقی انواع در هم پیچیده نظیر سایر زبانهای برنامه نویسی پیش نخواهد آمد. کامپایلر جاوا کلیه عبارات و پارامترها را کنترل می کند تا مطمئن شود که انواع ، قابلیت سازگاری بهم را داشته باشند . هر گونه عدم تناسب انواع ، خطاهایی هستند که باید قبل از اینکه کامپایلر عمل کامپایل نمودن کلاس را پایان دهد ، تصحیح شوند .

نکته : اگر دارای تجربیاتی در زبانهایC و++C و هستید ، بیاد بسپارید که جاوا نسبت به هر زبان دیگری نوع بندی شده تر است . بعنوان مثال درC و++C و می توانید یک مقدار اعشاری را به یک عدد صحیح نسبت دهید . همچنین در زبان C کنترل شدید انواع بین یک پارامتر و یک آرگومان انجام نمی گیرد .

اما در جاوا این کنترل انجام می گیرد . ممکن است کنترل شدید انواع در جاوا در وهله اول کمی کسل کننده بنظر آید . اما بیاد داشته باشید که اجرای این امر در بلند مدت سبب کاهش احتمال بروز خطا در کدهای شما می شود.

و در آخر جاوا :

- ساده
- شیء گرا
- قابل انتقال (Portable)
- توزیع شده (Distributed)
- کارایی بالا
- ترجمه شده (Interpreted)
- Multithreaded
- پویا
- ایمن (Secure)
- جاوا مجانی (اما Open Source نیست)

منابع :

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

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

کتاب :

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

: