به نام خدا

UML چیست ؟

نویسنده: نوید زراعتی

فهرست :

۱- طراحی برنامه

در سالیان اخیر ، متدولوژی های متفاوتی بمنظور طراحی برنامه ، پیاده سازی و در اختیار طراحان برنامه های کامپیوتری قرار گرفته شده است . برخی از این متدلوژی ها ی طراحی، پیچیده و برخی دیگر ساده می باشند . در تمامی حالات ، هدف یکی است : کمک به برنامه نویسان در جهت نوشتن برنامه ها ئی که بسادگی نوشته ، اشکال زدائی و در نهایت نگهداری گردند .

در این مقاله به بررسی مسائل مرتبط با طراحی برنامه پرداخته و در این راستا در ابتدا با یک متدولوژی ساده آشنا خواهیم شد. متدلوژی استفاده شده با اینکه بسیار مقدماتی می باشد ولی اهداف ما را در جهت نحوه طراحی یک برنامه بخوبی تامین خواهد کرد .در این مقاله ، به بررسی الگوریتم و مراحل پنج گانه برنامه نویسی خواهیم پرداخت ..

اهمیت طراحی برنامه

فرض کنید یک مسئله را برای تعدادی دانشجو تعریف و از آنان خواسته شده است که برنامه

ای بمنظور حل مسئله مورد نظر ، طراحی و پیاد ه سازی نمایند. پس از صرف چند ثانیه ، بلافاصله دانشجویان شروع به تایپ کد مورد نظر خود بمنظور حل مسئله می نمایند . در بین دانشجویان ، دانشجوئی وجود دارد که کاغذی را بر می دارد و شروع به نوشتن موضوع می نماید. دقایقی سپری می گردد ، اما همچنان دانشجویان مشغول تایپ برنامه خود ویا احتمالا" اشکال زدائی! آن هستند .تقریبا" بدون استثناء ، دانشجوئی که دیرتر از دیگران آغاز نموده است ، با سرعت بیشتری تکلیف خود را به پایان رسانده و حتی راه حل ارائه شده توسط وی ، نیز از سایر دانشجویان بمراتب بهتر است .چرا؟ در صورتیکه به کاغذی که در اختیار دانشجو قرار داده شده ، دقت نمائید ، یک طرح مناسب بمنظور طراحی برنامه را برای مسئله، مشاهده خواهید کرد . برخی از دانشجویان نیز ممکن است چندین کاغذ را تکمیل و یک طراحی پیچیده را انجام داده باشند. نکته مهم در این رابطه این است که این دانشجویان (چه آنانی که یک طراحی ساده را انجام داده اند و چه آنانی که یک طراحی پیچیده را دنبال نموده اند) دارای یک الگو(طرح) برای برنامه خود ، می باشند .

الگوریتم

هر برنامه، می بایست دارای یک طرح و یا الگو بوده تا برنامه نویس بر اساس آن عملیات خود را دنبال نماید.از دیدگاه برنامه نویسان ، هر برنامه نیازمند یک الگوریتم است . بعبارت ساده ، الگوریتم ، بیانه ای روشمند بمنظور حل یک مسئله بخصوص است . از منظر برنامه نویسان ،الگوریتم بمنزله یک طرح کلی و یا مجموعه دستورالعمل هائی است که با دنبال نمودن آنان ، برنامه ای تولید می گردد.

الگوریتم های میکرو در مقابل ماکرو

الگوریتم ها دارای ویژگی های متفاوتی می باشند . ما می توانیم در رابطه با الگوریتم استفاده شده به منظور نوشتن یک برنامه مشخص صحبت نمائیم . از این زاویه ، ما صرفا" در رابطه با الگوریتم در سطح ماکرو(**macro level**) ، صحبت نموده ایم . در چنین مواردی ، الگوریتم ارائه شده ، سعی در بدست آوردن جنبه های عمومی برنامه از طریق یک مرور کلی

به برنامه در مقابل درگیر شدن در جزئیات را دارد.ما می توانیم در رابطه با الگـوریتم هـا ، از سطح "میکرو" صحبت نمائیم . از این زاویه ، به سطوح پایین تر رفته و بـه عوامـل اساسـی ونگهدارنده ای که یک جنبه خاص از برنامه را با یکدیگر مرتبط می نماید، صـحبت کـرد. مثلا" در صورتیکه شما دارای داده هائی هستید که می بایست قبل از استفاده مرتـب گردنـد ،الگوریتم های مرتب سازی متعددی در این زمینه وجود داشته و مـی تـوان یکـی از آنهـا را بمنظور تامین اهداف مورد نظر خود انتخاب نمود. انتخـاب یـک الگـوریتم مرتـب سـازی ، صرفا" باعث حل شدن یکی از جنبه های متفاوت برنامه می گردد . پس از مرتب سـازی داده ها ،می بایست از یک الگوریتم میکرو دیگر بمنظور نمایش داده ها ی مرتب شـده اسـتفاده گردد .

همانگونه که احتمالا" حدس زده اید ، ما می توانیم تمام الگـوریتم هـای میکـرو را بمنظـور ایجاد یک الگوریتم ماکرو ، جمع آوری نمائیم . اگر ما با الگوریتم های میکرو ، آغاز نمـائیم ، و حرکت خود را بسمت نمایش ماکروی یک برنامه ، پیش ببریم ، کاری را انجام داده ایم کـه موسوم به طراحی " پایین به بالا" (**buttom-up**) ، است . اگر ما فعالیت خود را بـا یـک الگوریتم ماکرو آغاز و حرکت خود را بسمت پائین و الگوریتم هـای میکـرو ، ادامـه دهـیم ، طراحـــی از نـــوع " بـــالا بـــه پـــایین " (**top-down**) را انجـــام داده ایـــم . شاید این سوال مطرح گردد که کدام روش بهتر است ؟ اگر شما تمام مقالاتی را که تـاکنون در این زمینه نوشته شده اند را دنبال نمائید ، هرگز به یک نتیجه قابل قبـول دسـت نخواهیـد یافت . هر رویکرد، دارای نکات مثبت و منفی مربوط بـه خـود اسـت . صـرفنظر از رویکـرد طراحی استفاده شده ، می بایست دارای الگوئی (طرحی) مناسب برای برنامه باشـیم .حـداقل، نیازمند یک اعلامیه از مسئله برنامه نویسی و یک طرح (الگو) برای برخورد با مسئله ، خواهیم بود . پس از شناخت مسئله ، می توان نحوه حل مسئله را ترسـیم کـرد. شـناخت عمیـق و مناسب نسبت به مسئله ای که قصد حل آن را داریم ، شرط اساسی و ضروری برای طراحـی یک برنامه است .

با توجه به اینکه این اعتقاد وجود دارد که شناخت جامع و کلی از مسـئله ای کـه حـل آن را

داریم ، بخشی ضروری در اولین مرحله برنامه نویسی است ، ما در ادامه از رویکـرد "بـالا – پایین "، تبعیـت می نمائیم . فراموش نکنیم که رویکرد فوق ، امکان مشاهده مجـازی از هـر مسئله برنامه نویسی را فراهم خواهد نمود.

مراحل پنج گانه

هر برنامه را صرفنظر از میزان پیچیدگی آن ، می توان به پنج مرحله اساسی تجزیه کرد :

- مقدار دهی اولیه
- ورودی
- پردازش
- خروجی
- پاکسازی

در ادامه به بررسی هریک از مراحل فوق ، خواهیم پرداخت .

مرحله مقداردهی اولیه

مرحله مقداردهی اولیه ، اولین مرحله ای است که می بایست در زمان طراحی یک برنامـه در رابطه با آن فکر کرد . مرحله فوق ، شامل تمامی عملیات مورد نیازی اسـت کـه برنامـه مـی بایست قبل ازبرقراری ارتباط با کاربر ، انجام دهد . در ابتدا ممکن اسـت ایـن موضـوع کـه عملیاتی را قبل از برقراری ارتباط با کاربر می بایست انجام داد ، تا انـدازه ای عجیـب بنظـر رسد ولی احتمالا" برنامه های زیادی را مشاهده نموده اید که در این راستا عملیات مشـابهی را انجام می دهند. مثلا" ، در زمان استفاده از برنامه هائی نظیر **Word** ، **Excel** و یا برنامـه های مشابه دیگر ، با چنین مواردی برخورد نموده ایم . مثلا" با انتخاب گزینه منو **File** ، می توان لیستی از فایل هائی را که با آنها کار کرده ایم در بخش انتهائی منوفوق ، مشاهده کرد. (مشاهده آخرین فایل های استفاده شده در یک برنامه خاص ، با استفاده از جادو! میسر نشـده است) . برنامه مورد نظر شاید ، لیست فایل های اخیر را از دیسک خوانده و آنها را به لیست

مربوطه در منوی File ، اضافه کرده باشد . با توجه به اینکه لیست فایل های فوق ، می بایست قبل از اینکه برنامه هر چیز دیگر را برای کاربر نمایش دهد ، خوانده و نمایش داده شوند ، می توان انجام عملیات فوق را نمونه ای از مرحله مقداردهی اولیه، در نظر گرفت. یکی دیگر از عملیات متداول که به این مرحله مرتبط می باشد ، خواندن فایل های Setup است . چنین فایل هائی ممکن است حاوی اطلاعاتی در رابطه با نام مسیرهائی باشند که بانک ها ی اطلاعاتی خاصی و یا فایل های ذخیره شده دیگری را بر روی دیسک را مشخص می نمایند . با توجه به نوع برنامه ای که اجراء می گردد ، فایل های Setup می توانند شامل اطلاعاتی در رابطه با فونت های نمایش ، نام و محل چاپگر ، رنگ های زمینه و رویه ، وضوح تصویر صفحه نمایشگر و اطلاعات مشابهی دیگر باشند . سایر برنامه ها ممکن است مستلزم خواندن اطلاعاتی در رابطه با اتصالات شبکه ، مجوزهای امنیتی و دستیابی به اینترنت ، رمزهای عبور و سایر اطلاعات حساس دیگر باشند . در چنین مواردی فایل های Setup دارای نقشی مهم خواهند بود.

در زمان طراحی یک برنامه ، همواره می بایست در رابطه با اطلاعاتی که یک برنامه قبل آغاز خدمات و عملیات خود به آنها نیازمند است ، اندیشید و برای آنان در مرحله مقداردهی اولیه راهکار مناسب را انتخاب کرد . مرحله مقداردهی اولیه احتمالا" جائی است که می بایست از طریق آن اقدام به ارائه راهکار مناسب در جهت پاسخ به نیازهای فوق ، کرد.

مرحله ورودی

مرحله ورودی ، در حقیقت چیزی است که انتظار دارید باشد! مرحله فوق ، شامل اخذ (جمع آوری) هر آنچیزی است که یک برنامه برای انجام فعالیت های خود به آنها نیاز خواهد داشت . دراکثر موارد، اگر استنباط مناسبی از عملیاتی را که یک برنامه قصد انجام آنان را دارد ، حاصل گردد، مشخص نمودن لیستی از ورودی ها ، کاری ساده خواهد بود. مثلا" اگر شما قصد نوشتن یک برنامه وام را دارید ، می دانید که می بایست از کاربر میزان وام درخواستی ، بهره موردنظر و مدت زمان وام ، درخواست گردد. در حالات دیگر، لازم است در رابطه با نوع ورودی هائی که می بایست از کاربر اخذ گردد،

بررسی لازم و مبتنی بر اندیشه را دنبال نمود. مـثلا" در صـورتیکه قصدنوشـتن یـک برنامـه دفترچه آدرس را دارید ، آیا می خواهید نام فایل حاوی دفترچه تلفن و محـل ذخیـره فایـل مربوطه را در هر مرتبه که برنامه اجراء می گردد ، از کاربر درخواست نمائید ؟ بعبارت دیگـر برخی از مراحل ورودی می توانند و شاید می بایست ، توسط مرحله مقدار دهی انجام شـوند. ماهیت واقعی میزان اطلاعاتی که می توان آنها را در مرحله مقداردهی خواند ، بسـتگی بـه رفتار برنامه دارد. بعنوان یک قانون عمومی می توان به این مورد اشاره داشت که اکثر کاربران تمایل دارند که اطلاعات تکراری در یک فایل **Setup** و یا مقداردهی اولیه ذخیره گردد (در مقابل اینکه هر مرتبه که برنامه اجراء می گردد ، مجبور به ورود اطلاعـات تکـرای باشـند) . فایل های **Setup** بسیار مناسب بوده و در هرموردی که امکان بخدمت گـرفتن آنـان منطقـی بنظر می آید ، می بایست از آنان استفاده گردد . برخی دیگر از اطلاعات اولیـه دارای ماهیـت خاص خود بوده و تا زمانیکه کاربر آنها را تایپ ننماید ، شناخته نمی گردنـد . در مثـال وام اشاره شده ، می توان از **TextBox** های متعددی بمنظور احذ اطلاعات از کاربر و استفاده از آنان در برنامه ، کمک گرفت . با توجه به اینکه کاربر می بایسـت بـا ایـن **TextBox** هـا مرتبط تا اطلاعات موردنیاز برنامه را وارد نماید ، روشی را که شما بمنظور ارائه **Textbox** **Menus, ,Labels** و سایر عناصر برنامه ، استفاده می نمائید ، یکی از بخش های مهم یـک برنامه یعنی رابط کاربر (**user interface**) را مشخص خواهد کرد . فراموش نکنیم یکی از عوامل موفقیت هر نرم افزار ، بخش رابط کاربر آن است . طراحی مناسب بخش فوق ، امروزه بعنوان تخصصی خاص در طراحی و پیاد ه سازی نرم افزار مطرح و دارای جایگاه خاص خود است .

مرحله پردازش

مرحله پردازش ، شامل انجام عملیات بر روی ورودی (ورودی ها) ، بمنظـور تولیـد نتـایج مورد نظر برای برنامه است . در مثال وام ، برنامه پس از دریافـت ورودی هـای مـورد نظـر (میزان وام ، درصد بهره و زمان وام) آنها را از طریق یک معادله مالی بیکدیگر مرتبط و پس از حل معادله ، نتیجه مورد نظر حاصل خواهد شد(میزان پرداخت ماهانـه) . بعبـارت دیگـر ، مرحله پردازش قادر به دریافت ورودی ، برخورد با آنها و تولید پاسخ مناسب به مسئله است

. توجه داشته باشید که مرحله پردازش همواره باعث نمایش چیزی بر روی نمایشگر نخواهد شد. هدف ، عمل (عملیات) برروی داده (داده ها) بمنظور تولید یک نتیجه (نتایج) است . در این رابطه هیچگونه استثنائی وجود ندارد . در صورتیکه در برنامه ای از قبل می دانیم که مرحله پردازش زمان زیادی طول خواهد کشید ، منطقی است که فیدبک های لازم بمنظور آگاهی کاربر از میزان و درصد انجام پردازش (پردازش ها) در اختیار وی گذاشته شود (در زمانیکه برنامه در حال اجراء است) . در این رابطه می توان از روش های متعددی استفاده کرد . (ارائه یک میله پیشرفت ، برآورد زمان تقریبی بمنظور اتمام عملیات) .

مرحله خروجی

مرحله فوق ، پاسخ (پاسخ ها ی) مناسب و مورد انتظار را به کاربران مبنی بر حل مسئله مورد نظر ، ارائه می نماید. تعداد زیادی ازبرنامه ها ، پاسخ نهائی (نتیجه) خود را از طریق یک **Textbox** ، نمایش و در اختیار کاربر قرار می دهند . ، مثلا" اگر برنامه ای نوشته شده است که قصد محاسبه و نمایش میزان پرداخت ماهیانه یک وام دریافتی را داشته باشد ، می توان نتیجه بدست آمده (پرداخت ماهانه) را از طریق یک **textbox** ، ارائه تا پاسخی مناسب در ارتباط با مرحله خروجی یک برنامه، داده شده باشد . سایر برنامه ها ممکن است دارای وضعیتی بمراتب پیچیده تر باشند .مثلا" می توان برنامه ای را در نظر گرفت که نام ، آدرس ، شماره تلفن و سایر اقلام اطلاعاتی را از بانک اطلاعاتی خوانده و در ادامه آنها را بر روی صفحه نمایشگر ، نشان دهد. برنامه هائی اینچنین ، نیازمند شکل مناسبتری از نمایش خروجی بوده و نمی توان با استفاده از چند **textbox** به خواسته خود دست یافت (ارائه یک خروجی مطلوب و انعطاف پذیر) در اینگونه موارد می بایست از راهکارهای مناسبتری استفاده گردد . مثلا" می توان از جداول خاصی بمنظور نمایش اطلاعات مورد نظر استفاده کرد .(استفاده از **grid** و یا **List box** که برنامه در صورت ضرورت آنان را تکمیل نماید) . نکته مهمی که می بایست در رابطه با مرحله خروجی رعایت گردد ، آگاهی از این موضوع است که با توجه به نمایش نتایج خروجی برای کاربر، بخش فوق را می توان جزئی از بخش رابط کاربر یک نرم افزار در نظر گرفت . در زمان ورود اطلاعات (مرحله ورودی) از عناصر متفاوتی بمنظور اخذ اطلاعات توسط کاربر در بخش رابط استفاده می گردد ، در مرحله خروجی ،

بخش رابط کاربر با کاربر بگونه ای دیگر مرتبط خواهد شد (ارتباطی بمراتب غیر فعالتر نسبت به مرحله ورود اطلاعات) .

مرحله پاکسازی (Cleanup)

مرحله پاکسازی ، بمنظور خاتمه بخشیدن مودبانه یک برنامه، پس از تکمیل عملیات مربوطه است. می توان این مرحله را بعنوان مکمل مرحله مقداردهی اولیه در نظر گرفت .با اینکه تعداد زیادی از برنامه های ساده قادرند بسادگی و بدون انجام عملیات تکمیلی توسط برنامه نویس ، خاتمه یابند ولی برنامه های پیچیده زیادی نیازمند برخی کمک ها در این زمینه می باشند. مثلا" اگر برنامه ای یک فایل **Setup** را بمنظور مقداردهی برخی از متغیرها در زمان مرحله مقداردهی اولیه ، خوانده باشد ، مرحله پاکسازی می تواند شامل بهنگام سازی آندسته از متغیرهای موجود در فایل **Setup** باشد که نشاندهنده آخرین اطلاعات کاربر است . مرحله پاکسازی ، اغلب شامل بستن فایل ها (فایل های **Setup** و بانک اطلاعاتی) است . برخی برنامه ها میزان استفاده از برنامه توسط کاربران را ثبت و اطلاعات مربوطه را در مکانهائی که **Log file** نامیده می شوند ، ذخیره می نمایند(ثبت مشخصات افرادیکه برنامه را اجراء نموده بهمراه سایر اطلاعات مرتبط نظیر تاریخ و زمان آغاز و توقف برنامه ، در خیلی از برنامه ها به امری ضروری تبدیل شده است) .

یکی دیگر از انواع فایل های **Log** به فایل های ثبت خطاء برمی گردد(**error log file**) . هدف این نوع از فایل ها ، ثبت اطلاعاتی در رابطه با هر نوع خطائی است که ممکن است در مدت زمان اجرای یک برنامه ، محقق گردد. برنامه نویسان با استفاده از محتویات این نوع فایل ها ، قادر به اشکال زدائی برنامه خواهند بود . عملیات واقعی و مورد نظری که می بایست در مرحله پاکسازی ، انجام گردد ، به نیازهای یک برنامه بستگی خواهد داشت . معمولا" اگر در برخی برنامه ها عملیات خاصی را در مرحله مقدار دهی اولیه انجام می هیم ، می بایست برخی از عملیات متناظر با آنان را در مرحله پاکسازی انجام داد . باز نمودن و بستن فایل های مورد نیاز در یک برنامه ، نمونه ای متداول از دو مرحله فوق می باشد .

آیا هر برنامه شامل پنج مرحله گفته شده است؟

در پاسخ به سوال فوق می بایست با صراحت پاسخ منفی داده شود. در این راستا ، برنامه های متعددی وجود دارد که مثلا" به مراحل مقداردهی اولیه و یا پاکسازی ، نیاز نخواهند داشت . مراحل مقداردهی اولیه و پاکسازی در مرحله طراحی برنامه های پیچیده مورد توجه جدی قرار خواهند گرفت. بموازات افزایش تجربه در نوشتن برنامه ، شناخت مناسبی در این رابطه بوجود می آید(کدام برنامه به تمام مراحل پنج گانه نیاز و کدامیک نیاز ندارند).طراحان می بایست همواره یک مسئله برنامه نویسی را با فرض وجود پنج مرحله یاد شده ،دنبال نمائید . قطعا" حذف یک مرحله در زمان طراحی بمراتب ساده تر از نادیده گرفتن ! اولیه آن خواهد بود.

پالایش یک طرفه (Sideways Refinement)

همانگونه که قبلا" اشاره گردید ، ما علاقه مند به طراحی بالا به پایین می باشیم .(الگوریتم ماکرو بعنوان یک نقطه شروع در فرآیند طراحی برنامه) . پس از انتخاب رویکرد فوق ، می بایست شناخت مناسبی نسبت به مسئله ای که قصد حل آن وجود دارد ، ایجاد گردد. تا رسیدن به سطح میکرو(ارائه الگوریتم های میکرو) بمنظور حل مسئله مورد نظر راه زیادی را در پیش خواهیم داشت. بموازات حرکت از سطح مرور کلی برنامه به خصوصیات و ویژگی های یک برنامه ، می بایست دانش خود را نسبت به جزئیات مربوطه افزایش داد . از پنج مرحله گفته شده ، می توان بمنظور نقطه شروع دید ماکرو خود در زمان فرآیند طراحی استفاده کرد. درادامه ، می توان هر یک از مراحل را بدقت بررسی تا جزئیات بیشتری در رابطه با مرحله مورد نظر ، مشخص گردد (استخراج جزئیات لازم در رابطه با تحقق هر مرحله) . فرآیند فوق ، " پالایش یک طرفه " ، نامیده می شود . در ادامه ، بمنظور شناخت مناسب فرآیند پالایش یک طرفه ، به بررسی یک نمونه می پردازیم . فرض کنید ، کاربری دارای یک فایل بانک اطلاعاتی است که در آن تمام قرار ملاقات های وی ، ذخیره شده اند . قرار ملاقات ها در بانک اطلاعاتی با نظم و ترتیب خاص (تاریخ قرار

ملاقات) ذخیره شده اند . کاربر ، می خواهد قادر به مشاهده قرار ملاقات های خود بر اساس حروف الفبائی و بر اساس نام خانوادگی اشخاص مورد نظری که قصد ملاقات با وی را دارند ، باشد. چگونه می توان از پالایش یک طرفه ، بمنظور طراحی یک را ه حل استفاده کرد؟

پالایش یک طرفه مرحله مقدار دهی اولیه

می دانیم که کاربر دارای یک بانک اطلاعاتی شامل قرار ملاقات ها ، می باشد. ما همچنین می دانیم که کاربر می خواهد لیستی از قرار ملاقات های خود را بصورت مرتب شده و بر اساس نام خانوادگی مشاهده نماید . موارد فوق ، دید ماکروی ما از الگوریتم است . بنابراین ، در مرحله مقداردهی اولیه چه عملیاتی می بایست انجام داد ؟ واضح است که ما نیازمند باز نمودن بانک اطلاعاتی قرار ملاقات ها می باشیم . ما همچنین نیازمند یک فرم (مثلا" یک فرم مبتنی بر **VB.NET** و یا فرم وب) بمنظور نمایش نتایج پس از مرتب سازی قرار ملاقات ها ، خواهیم بود. (فرض می شود از مکان بانک اطلاعاتی بر روی شبکه آگاهی داریم ، و می توان نام و رمز عبور کاربر را از بانک اطلاعاتی مربوطه بمحض آغاز اجرای برنامه توسط کاربر ، مشخص کرد) . با استفاده از اطلاعات فوق، اولین "پالایش یک طرفه " ، بصورت زیر خواهد بود :

همانگونه که در شکل فوق ، مشاهده می گردد بموازات حرکت از سمت چپ بسمت راست ، جزئیات مربوطه افزایش خواهد یافت . شکل فوق ، پالایش یک طرفه ، لیستی از برنامه های جانبی و توابع مورد نیاز بمنظور انجام فعالیت های مربوطه در مرحله مقداردهی اولیه را نشان می دهد . هر روتین کوچک، مسئول انجام عملیاتی خاص خواهد بود .

شبه کد (Pseudo Code)

عملیات پالایش را می توان در رابطه با هر مرحله با استفاده از "شبه – کد " ، دنبال کرد. شبه کد ،الگوریتمی برای بیان عملیاتی است که می بایست توسط یک روتین محقق گردد .

در این راستا از یک گرامر مشابه انگلیسی ، استفاده مـی گـردد . مـثلا" شـبه کـد ، روتـین IsValidUser بصورت زیر خواهد بود:

شبه کد روتین IsvalidUser
Is ValidUser() If CurrentUserName Not in ValidUserList Display Invalid User Error Message Terminate Program Else Return ValidUserIDNumber End

شبه کد ، عملیاتی را که یک روتین می بایست انجام دهد ، بدون اتکاء به گرامـر یـک زبـان برنامه نویسی خاص ، تشریح می نماید. شبه کد ، زبانی مبتنی بـر گرامـری خـاص نبـوده و الگوریتمی از عملیات مورد نظر که می بایست توسط یک روتین انجام شود را مشـخص مـی نماید. مزیت شبه کد، شباهت زیاد آن به زبان انگلیسی است و مـی تـوان آن را بـا افرادیکـه برنامه نویس نبوده و بنوعی در فاز طراحی صاحبنظر مـی باشـند ، بـه اشـتراک تـا صـحت استنباطات حاصل شده تائید و یا اصلاح گردد.(در فاز طراحی می بایست یک ارتباط مسـتمر با کاربران صاحبنظر برقرارگردد، ما قرار است مسئله آنان را حل نمائیم نه مسئله خود را و یـا نمی خواهیم مسئله ای دیگر را بر حجم مسائل آنان اضـافه نمـائیم!) بـدین ترتیـب ، امکـان تشخیص خطاء و اعمال تعییرات لازم در خصوص برخورد با خطاهای احتمالی در ابتدا فراهم می گردد (یکی از اصول مهندسی نرم افزار در این رابطه به این موضوع اشاره می نماید که به هر میزان که زمان کشف یک خطاء در چرخه حیات یک برنامه سریعتر باشد ، هزینه برخـورد با خطاء کاهش خواهد یافت) .

پس از آگاهی از اهداف ارائه شده در شبه کد ، می توان بسادگی اقدام بـه ترجمـه شـبه کـد مربوطه به کدهای برنامه نویسی با استفاده از زبان مورد نظر نمود. فراموش نکنیم که طراحـی خوب ، همواره پیاده سازی ساده تر برنامه ها را بدنبال خواهد شد.

در بخش بعدی این مقاله ، با متدولوژی Unified Modeling Language(UML) (آشنا خواهیم شد. UML ، یک متدولوژی طراحی متداول خصوصا" در زمینه برنامه نویسی شی گراء است

۲- مدل‌سازی (Modelling) چیست؟

مدل‌سازی یکی از تکنیک‌های ذهنی بشر می‌باشد که نه تنها برای اهداف علمی، بلکه برای انجام امور روزمره بشر به دفعات مورد استفاده قرار می‌گیرد. مدل‌سازی به طور کلی یعنی شبیه‌سازی یک محیط با اندازه‌های متفاوت و از محیط واقعی و احتمالا مواد و مصالحی متمایز از جنس مواد و مصالح محیط مدل شده. در مدل‌سازی ابتدا اجزای محیط واقعی انتخاب شده و متناسب با هدف مورد نظر از مدل‌سازی خصوصیاتی از هریک از اجزای واقعی انتزاع می‌شود، یعنی به ازای هزیک از اجزای محیط واقعی یک موجودیت تجریدی ساخته می‌شود و با برقراری ارتباطی مشابه با ارتباط اجزای واقعی، در میان موجودیت‌های تجریدی، محیط واقعی مدل می‌شود. برای روشن شدن مثالی می‌زنیم:

فرض کنیم قصد داشته باشیم در فاز طراحی یک اتومبیل میزان موفقیت هوا در مقابل اتومبیل در حال حرکت را بسنجیم یکی از راه‌ها برای انجام این آزمایش، ساخت یک اتومبیل واقعی، راندن و سپس اندازه‌گیری مقاومت هوا می‌باشد که انجام اینکار اگرچه ما را به هدف می‌رساند، ولی دارای هزینه بالاییست چرا که بایستی ابتدا ماشین ساخته شود، سپس مورد آزمایش قرار گیرد.در این صورت اگر در آزمایش به نتیجه مورد نظر نرسیم، بایستی دوباره طراحی را تغییر داد، و پس از ساخت یک نمونه واقعی دیگر آزمایش را تکرار کنیم و این روند آنقدر ادامه پیدا کند تا طراحی مناسب برای اتومبیلی با خصوصیات مورد نظر شکل گیرد. می‌بینیم که چنین روشی بسیار پرهزینه است و این هزینه هم شامل هزینه‌های اقتصادی است و هم هزینه‌های زمانی، چون علاوه بر این که در هر مرحله آزمایش بایستی اتومبیل با صرف هزینه بالا ساخته شود، زمان ساخت آن نیز طول خواهد کشید.

ولی متخصصان برای انجام چنین آزمایشی به مدل روی می‌آورند. یعنی یک جسم فیزیکی کوچک با خصوصیات آئرودینامیکی لحاظ شده در طراحی اتومبیل، ساخته می‌شود و با قرار دادن آن در یک تونل باد، حرکت اتومبیل در فضای واقعی را شبیه سازی می‌کنند و بدین طریق میزان مقاومت هوا را می‌سنجند.

نکات مورد توجه در این مدل‌سازی، یکی اندازه مدل و دیگری خصوصیات آن می‌باشد. مدل بسیار ساده و کوچک می‌باشد و از طرفی تنها خصوصیت آئرودینامیکی اتومبیل در مدل لحاظ می‌شود. چرا که هدف ما از مدل‌سازی تنها بررسی خصوصیات آئرودینامیکی اتومبیل است و مدل الزاماً نبایستی از جنبه‌های دیگر، شباهتی به اتومبیل واقعی داشته باشد. مثلا در ساخت چنین مدلی به هیچ‌وجه به استحکام اجزا و یا زیبایی مدل توجه نمی‌شود چون بررسی چنین خصوصیاتی خارج از هدف این مدلسازی خاص است.

مثال بالاتنها یک جنبه از مدل‌سازی را بیان می‌کند و آن جنبه شناخت **Exploration** می‌باشد. یعنی در مدلسازی‌های مشابه مدل‌سازی فوق‌الذکر، هدف از مدل‌سازی تنها شناخت محیط مورد مدل می‌باشد. یک جنبه دیگر از مدل‌سازی تبیین (**specitication**) می‌باشد. یعنی گاه برای معرفی و ارائه خصوصیات یک موجودیت واقعی یک مدل از آن ارائه می‌شود. نقشه جغرافیایی مثال خوبی است که این جنبه از مدل‌سازی را مورد نظر دارد.

پس می‌توان گفت که هدف از مدل‌سازی دو چیز می‌باشد:
الف)شناخت(**exploration**)

ب)تبیین(**specification**)

که بر اساس تعریف مسئله، مدل‌سازی یکی یا هردو هدف را در نظر می‌گیرد.

نکته دیگری که بایستی در مدل‌سازی توجه کرد، روش (**methodo logy**) ساخت یک مدل می‌باشد. در بعضی موارد مدل چیز بسیار ساده‌ای است و به راحتی ساخته می‌شود. ولی در بعضی از موارد مدل خود بسیار پیچیده می‌باشد هر چند از نظر منطقی غیرممکن می‌نماید ولی می‌توان ادعا کرد که در بعضی موارد مدل پیچیده‌تر از موجودیت واقعی است. زمینه‌ای

که این ادعا را در آن مصداق فراوان دارد، نرم‌افزار می‌باشد. بنابراین در شاخه‌ای از مهندسی که مدل‌سازی حائز اهمیت فراوان می‌باشد قطعا روش‌های استانداردی برای ساختن مدل وجود دارد. در نرم‌افزار، روش‌های تولید نرم‌افزار مانند **SSAPM ,RUP,USDP**, در واقع روش‌های مدل‌سازی می‌باشند. هر روش مدل‌سازی طبیعتا نیازمند مصالحی برای ساخت مدل می‌باشد که در روش‌های مدل‌سازی نرم‌افزاری مصالح لازم برای تولید مدل، زبان‌های مدل‌سازی می‌باشند

۳- تاریخچهٔ UML :

دیدگاه شی‌گرایی (**Object Oriented**) از اواسط دهه ۱۹۷۰ تا اواخر دهه ۱۹۸۰ در حال مطرح شدن بود. در این دوران تلاشهای زیادی برای ایجاد روشهای تحلیل و طراحی شی‌گرا صورت پذیرفت. در نتیجهٔ این تلاشها بود که در طول ۵ سال یعنی ۱۹۸۹ تا ۱۹۹٤، تعداد متدولوژیهای شی‌گرا از کمتر از ۱۰ متدولوژی به بیش از ۵۰ متدولوژی رسید. تکثر متدولوژیها و زبانهای شی‌گرایی و رقابت بین اینها به حدی بود که این دوران به عنوان "جنگ متدولوژیها" لقب گرفت. از جمله متدولوژیهای پرکاربرد آن زمان می‌توان از **Booch**، **OOSE**، **OMT**، **Fusion**، **Coad-Yourdan**، **Shlayer-Mellor** وغیره نام برد. فراوانی و اشباع متدولوژیها و روشهای شی‌گرایی و نیز نبودن یک زبان مدلسازی استاندارد، باعث مشکلات فراوانی شده بود. از یک طرف کاربران از متدولوژیهای موجود خسته شده بودند، زیرا مجبور بودند از میان روشهای مختلف شبیه به هم که تفاوت کمی در قدرت و قابلیت داشتند یکی را انتخاب کنند. بسیاری از این روشها، مفاهیم مشترک

شی‌گرایی را در قالبهای مختلف بیان می‌کردند که این واگرایی و نبودن توافق میان این زبانها، کاربران تازه‌کار را از دنیای شی‌گرایی زده می‌کرد و آنها را از این حیطه دور می‌ساخت. عدم وجود یک زبان استاندارد، برای فروشندگان محصولات نرم‌افزاری نیز مشکلات زیادی ایجاد کرده بود.

اولین تلاشهای استانداردسازی از اکتبر ۱۹۹٤ آغاز شد، زمانی که آقای **Rumbaurgh** صاحب متدولوژی **OMT** به آقای **Booch** در شرکت **Rational** پیوست و این دو با ترکیب متدولوژیهای خود، اولین محصول ترکیبی خود به نام "روش یکنواخت" را ارائه دادند. در سال ۱۹۹۵ بود که با اضافه شدن آقای **Jacobson** به این دو، روش یکنواخت ارائه شده با روش **OOSE** نیز ترکیب شد واین خود سبب ارائهٔ **UML** نسخهٔ ۰.۹ در سال ۱۹۹٦ گردید. سپس این محصول به شرکتهای مختلفی در سراسر جهان به صورت رایگان ارائه شد و استقبال شدید شرکت‌ها از این محصول و تبلیغات گسترده شرکت **Rational**، سبب آن شد که گروه **OMG**، نسخهٔ ۱.۰ **UML** را به عنوان زبان مدلسازی استاندارد خود بپذیرد. تلاشهای تکمیلی **UML** استاندارد ادامه پیدا کرد و نسخهٔ ۱.۱ آن در سال ۱۹۹۷ و نسخه ۱.۳ آن در سال ۱۹۹۹ ارائه گردید.

٤- UML چیست ؟

مراحل پنج گانه برنامه نویسی ، نقطه شروع مناسبی برای طراحی یک برنامه است (اولین

فاز). در ادامه با استفاده از پالایش (بهسازی) یکطرفه مراحل پنج گانه برنامه نویسی ، فاز دوم طراحی یک برنامه انجام خواهد شد . استفاده از شبه کد بمنظور ارائه جزئیات پالایش ، کمک قابل توجه و مفیدی در ارتباط با طراحی برنامه را بدنبال خواهد داشت . رویکرد فوق (مراحل پنج گانه برنامه نویسی) ، روشی مفید بمنظور طراحی یک برنامه است . در این راستا برخی از طراحان برنامه های کامپیوتری ترجیح می دهند که از یک روش دقیق تر و موشکافانه تر استفاده نمایند . UML(Unified Modeling Language) مبتنی بر چنین رویکردی است .

UML ،زبانی استاندارد بمنظور مشخص نمودن ، پیش بینی ، ایجاد و مستند سازی تولیدات نرم افزاری است . UML ، مجموعه ای از بهترین امکانات مهندسی را بمنظور استفاده در مدل سازی سیستم های بزرگ و پیچیده ارائه که کارآئی آنان به اثبات رسیده است . UML یک متدولوژی رسمی برای پیاده سازی نرم افزار است .

روند شکل گیری UML

برنامه نویسی شی گراء (OOP) ، از اوایل سال ۱۹٦۰ مطرح گردید . برنامه نویسی شی گراء با اینکه بعنوان یک ایده جدید مطرح شده بود ولی بسرعت زبان های مدل سازی شی گراء برای پوشش ایده فوق ، مطرح و پیاده سازی گردیدند. در فاصله سال های ۱۹۷۰ تا اواخر ۱۹۸۰ چندین زبان مدل سازی شی گراء پیاده سازی گردید . تعداد زبان ها ی مدل سازی شی گراء در سال ۱۹۹۵ به بیش از پنجاه نمونه رسیده بود . از افراد فعال و پیشرو در این زمینه می توان به Rumbaugh Jim (شرکت جنرال الکتریک)، Grady Booch (شرکت software Rational) و Ivar Jacobson (شرکت Objectory) اشاره نمود. هر یک از افراد فوق ، تلاش گسترده ای را در جهت مدل سازی زبان برنامه نویسی انجام داده بودند . در سال ۱۹۹٤ ، Rumbaugh شرکت جنرال الکتریک را ترک و به Booch در شرکت Rational Software ملحق گردید. یک سال بعد ، شرکت Rational Software ، شرکت Objectory را خریداری و افراد یاد شده همکاری خود را با یکدیگر و در یک شرکت مشترک آغاز نمودند. ماحصل همکاری

فوق ، ارائه اولین نسخه UML 0.9 توسط شرکت Rational software در سال ۱۹۹۶ بود .

در سالیان بعد ، Object Management Group(OMG) ، تلاش های گسترده ای را بمنظور ارتقاء و بهسازی UML آغاز نمود. در اواسط سال ۲۰۰۱ ، اعضاء OMG ، کار خود را بمنظور ارتقاء به UML 2.0 آغاز نمودند. در حال حاضر ، UML شامل مدل سازی ویژوال ، شبیه سازی و امکانات پیاده سازی است . تعداد زیادی از ابزارهای UML طراحی و در اختیار علاقه مندان قرار گرفتند . Rational Rose 2002 از شرکت Rational Software ، نرم افزار Enterprise Describe از شرکت Embarcadero Technologies و Visio 2002 از شرکت مایکروسافت . نمونه هائی از ابزارهای UML می باشند .

نمودار های UML

UML یک ابزار ویژوال بوده و از انواع متفاوتی نمودار استفاده می نماید . هر یک از نمودار های UML ، امکان مشاهده یک سیستم نرم افزاری را از دیدگاههای متفاوت و با توجه به درجات متفاوت Abstraction در اختیار پیاده کنندگان قرار می دهد. برخی از نمودار های UML عبارتند از :

نمودار کلاس (Diagram Class):

این نمودار،کلاسها، واسطها و همکاری و روابط بین آنها را نمایش می‌دهد. و نمودار اصلی و مرکزی UML می‌باشد. که بیان‌کننده ساختار ایستای سیستم نرم‌افزاری می‌باشد.

نمودار اشیاء (Diagram Object):

این نمودار، اشیاء سیستم و روابط بین آنها را نمایش می‌دهد. در واقع یک تصویر لحظه‌ای از نمودار کلاس می‌باشد.

نمودار موردکاربرد (Usercase Diagram):

این نمودار، تعامل کاربران خارجی و سیستم را مدل می‌کند و از جهاتی شبیه نمودار سطح صفر **DFD** می‌باشد که جنبه‌های رفتاری سیستم را نمایش می‌دهد. این نمودار نقطه ورودی برای تمامی نمودارهای دیگری است که به تشریح نیازمندیها و معماری و پیاده‌سازی سیستم می‌پردازند.

نمودارهای تعامل (Interaction Diagram):

این نمودارها، بیان کننده تعامل هستند که شامل اشیاء مختلف و روابط بین آنها و همچنین پیغامهایی که بینشان رد و بدل می‌شود می‌باشند. این نمودارها جنبه‌های پویای یک سیستم را مدل می‌کنند و خود بر دو نوعند: نمودار توالی(**Sequence Diagram**) که ترتیب زمانی تعامل‌ها را نشان می‌دهد و نمودار همکاری(**Collaboration Diagram**) که تاکید بر نمایش ساختاری تعامل‌ها دارد.

نمودارحالت (Diagram Statechart):

این نمودار، بیان‌کننده جنبه‌های رفتاری سیستم می‌باشد و در واقع توصیف رسمی یک کلاس بوده که شامل حالات، انتقال بین حالات، رخدادها و فعالیتها می‌باشد. از این نمودارها برای نمایش دادن چرخه حیات اشیاء یک کلاس خاص نیز می‌توان استفاده کرد.

نمودار فعالیت(Diagram Activity):

این نمودار، نوع خاصی است از نمودار حالت، که انتقال جریان از یک فعالیت به فعالیت دیگر را نمایش می‌دهد. این نمودار جنبه‌های پویای یک سیستم را نمایش می‌دهد. در واقع حالات این نمودار، گامهای ترتیبی انجام یک عمل را نمایش می‌دهند.

نمودار اجزاء(Diagram Component):

از جمله نمودارهای پیاده‌سازی می‌باشد و سازماندهی و روابط بین مجموعه‌ای از اجزاء را نمایش می‌دهد. این نمودار، جنبه های ایستای پیاده‌سازی یک سیستم را مدل می‌کند.

نمودار به‌کارگماری(Deployment Diagram):

پیکربندی گره‌های پردازشی زمان اجرا را نمایش می‌دهد. که برای مدل کردن جنبه‌های ایستای به‌کارگماری یک معماری بکار می‌رود. همچنین نمایش‌دهندهٔ اجزای استفاده‌شده زمان اجرا مثل کتابخانه‌های DLL، فایل‌های اجرایی، کدهای مبدا و روابط بین آنها می‌باشد.

البته این نمودارها تمام نمودارهای UML نیستند بلکه بسته به نیاز و با کمک ابزارهای Case میتوان نمودارهای دیگری نیز تعریف و استفاده کرد.

آنالیز شی گراء (OOA)

آنالیز شی گراء و یا OOA ، یک متدولوژی قدرتمند برای تجزیه و تحلیل فرآیند پیاده سازی نرم افزار است . در زمان استفاده از OOA ، هر چیز در فرآیند پیاده سازی نرم افزار بمنزله کلاس در نظر گرفته خواهد شد (این طرز تفکر می بایست محور آنالیز سیستم قرار گیرد) . مثلا" در یک بیمارستان هر یک از عناصر موجود نظیر : دکتر ، پرستار ، بیمار و ملاقات کننده ، بمنزله یک کلاس در نظر گرفته می شوند . هر نسخه جدیدی که از یک کلاس ایجاد می گردد ، بمنزله یک نمونه (Instance) از کلاس در نظر گرفته خواهد شد . محوریت فرآیند آنالیز شی گراء ، تاکید بر ایجاد کلاس های مورد نیاز سیستم است . مهمترین و اصلی ترین رویکرد OOA ،یافتن پاسخ مناسب برای سوالاتی است که با What شروع و در فرآیند پیاده سازی نرم افزار حضوری موثر دارند . نمونه سوالات OOA در این زمینه عبارتند از : " چه کلاس هائی در برنامه وجود دارد؟" . " چه چیزی را برنامه انجام خواهد داد ؟" " هر یک از کلاس ها در برنامه چه عملیاتی را بمنظور حل مسئله انجام خواهند داد ؟" " مسئولیت هر کلاس در برنامه چیست ؟" در OOA ، تاکید بر آنالیز اشیاء ، فعالیت ها و مسئولیت های سیستم نرم افزاری است .

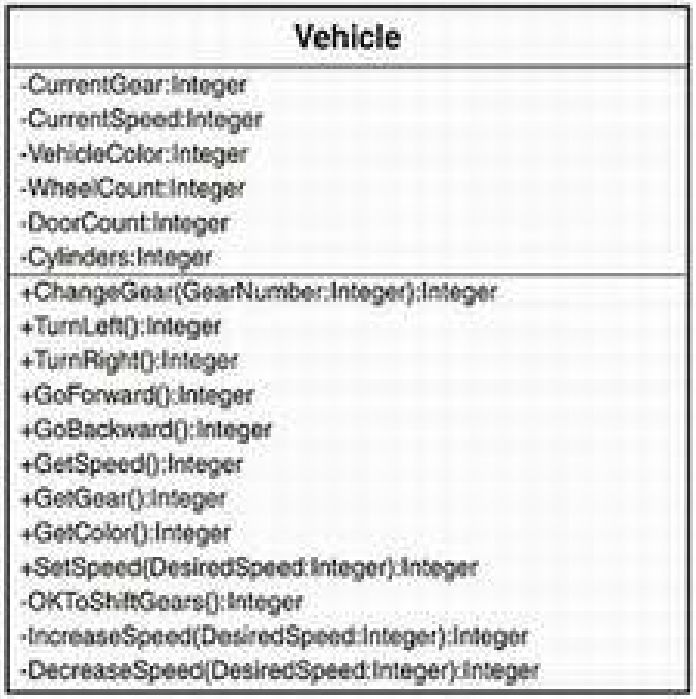
طراحی شی گراء (OOD)

نکته اساسی در طراحی شی گراء ، تاکید و سرو کار داشتن با سوالاتی است که با How شروع و در فرآیند پیاده سازی نرم افزار حضوری فعال و موثر خواهند داشت . " چگونه این کلاس داده را جمع آوری می کند ؟" . " چگونه این کلاس گزارش را چاپ می نماید ؟" ، نمونه سوالاتی در این زمینه می باشند .در نمونه مثال بیمارستان، وضعیت فوق به خصلت ها ، صفات و متدهای یک کلاس مرتبط می گردد . بنابراین OOA ، کلاس های مورد نظر و ضروری بمنظور نیل به اهداف نرم افزار را مشخص می نماید و محور عملیات بر جستجو و تبین جایگاه یک کلاس در برنامه متمرکز است . در OOD ، تاکید بر پیاده سازی کلاس ها ، صفات و خصایصی است که بمنزله هسته یک کلاس مطرح می گردند . ترکیب هر یک از فعالیت های فوق (آنالیز شی گراء و طراحی شی

گراء) بهمراه پیاده سازی لینک هائی که با کلاس ها سروکار دارند جملگی بعنوان بخشی از فرآیند **OOP** (برنامه نویسی شی گراء) محسوب می گردند.

نمودار های کلاس UML

نمودار کلاس در **UML** یکی از مهمترین نمودار ها تلقی می گردد . نمودار فوق ، مسئولیت مدل سازی ساختار کلاس و محتویات را با استفاده از عناصری نظیر کلاس ها ، اشیاء و پکیج ها برعهده دارد . این نمودار همچنین ، ارتباطاتی نظیر : توارث و پیوستگی را نمایش خواهد داد. نمودار فوق ، شکل خلاصه و استانداردی بمنظور نمایش یک کلاس را ارائه می نماید. در این راستا از یک مستطیل که به سه بخش متفاوت تقسیم می گردد ، استفاده می شود. در اولین بخش مستطیل ، نام کلاس قرار می گیرد . در دومین بخش مستطیل ، خصلت های یک کلاس قرار خواهند گرفت (ممکن است از واژه صفات و یا متغیر نیز استفاده گردد) و در بخش سوم ، متدهای یک کلاس قرار می گیرند.متدهای هر کلاس ، عملیاتی را که یک کلاس می تواند انجام دهد ، مشخص می نمایند. شکل زیر ، یک نمودار کلاس نمونه را نشان می دهد. در اولین بخش ، نام کلاس **Vehicle** مشخص شده است .نام هر کلاس با یک حرف بزرگ شروع و در مواردیکه نام کلاس شامل بیش از یک کلمه باشد ، هر کلمه در نام کلاس با یک حرف بزرگ آغاز می گردد . **Vehicle** ، **PassengerCar** و **IncomeStatement** نمونه هائی در این زمینه می باشند .استفاده از از فضای خالی بین کلمات تشکیل دهنده نام یک کلاس ، مجاز نمی باشد .



خصایص کلاس (*Properties , Attributes*)

در نمودار کلاس **Vehicle** و در بخش دوم از شش خصلت **Integer** استفاده شده است . در نمونه کلاس های دیگر ، یک کلاس ممکن است دارای دهها خصلت باشد .در برخی زبانهای برنامه نویسی نظیر ویژوال بیسیک دات نت ،از خصلت با نام **Prtoperty** نیز یاد می گردد. هر خصلت می تواند دارای مقادیر متفاوتی باشد . مقادیر جاری خصلت ها ، وضعیت یک کلاس را تشریح می نمایند . در مقام مقایسه می توان خصایص یک شی را نظیر نقش اسامی در یک جمله در نظر گرفت (مقایسه فوق صرفا" جنبه آموزشی دارد) .

علایم + و –

همانگونه که مشاهده می گردد ، هر **entry** دربخش دوم نمودار کلاس **Vehicle** ، دارای یک علامت – در جلوی نام خود است .در بخش سوم ، برخی از **Entry** ها ، دارای علامت + و برخی دیگر دارای علامت – می باشند . وجود علامت + در ابتدای یک آیتم (خصلت ، متد) ، نشاندهنده در دسترس بودن آن از طریق خارج از کلاس است . بعبارت دیگر ، علامت

+، امکان استفاده از آیتم مورد نظر و تاثیرگذاری بر وضعیت یک کلاس را نشان می دهد . علامت + ، عمومی بودن (**Public**) عناصر کلاس مربوطه را نشان می دهد . اگر یک **Entry** با یک علامت - شروع گردد ، بدین معنی خواهد بود که آیتم مورد نظر صرفا" برای استفاده خود کلاس در دسترس بوده و امکان استفاده از آن برای خارج از کلاس میسر نخواهد بود. بنابراین علامت - ، نشاندهنده خصوصی (**Private**) بودن عناصر مربوط به یک کلاس است .

استفاده از علائم + و - ، نشاندهنده نوع دستیابی به هر یک از عناصر مربوط به یک کلاس است . در حقیقت علامت + ، روشی بمنظور ارتباط با کلاس را مشخص نموده و علامت - نشاندهنده عناصری است که صرفا" برای خود کلاس قابل استفاده خواهند بود . ایجاد یک آیتم بصورت خصوصی همواره مورد توجه طراحان شی گراء بوده و تامین کننده اهداف کپسوله سازی در برنامه نویسی شی گراء است . با کپسوله سازی داده ، امکان بروز تغییرغیرعمد داده در برخی بخش ها ی برنامه و از طریق خارج از کلاس به حداقل مقدار خود خواهد رسید . بدین ترتیب، تشخیص و برطرف نمودن خطاهای احتمالی ، بسرعت و بسادگی میسر خواهد شد .

متدهای کلاس (عملیات)

عنصر سوم در نمودار کلاس ، نشاندهنده نوع عملیات مرتبط با کلاس است . در **UML** آیتم های موجود در این بخش را "عملیات " (**Operations**) ، ودر برخی از زبان های برنامه نویسی نظیر ویژوال بیسیک دات نت ، به آنان "متد" گفته می شود . متدها ، نحوه ارتباط برنامه نویسان با یک کلاس را مشخص می نمایند. هر متد عملیات خاصی را در ارتباط با یک کلاس انجام خواهد داد.اگر خصلت ها را بمنزله اسامی در یک جمله در نظر بگیریم ، می توان متدها را بمنزله افعال موجود در یک جمله در نظر گرفت .

متدهای کلاس و آرگومان ها

در برخی موارد یک متد نیازمند اطلاعات خارجی بمنظور انجام وظایف محوله است . مـثلا" در نمودار کلاس **Vehicle** از متد زیر استفاده شده است :

+SetSpeed(DesiredSpeed:Integer):Integer

علامت + نشاندهنده این موضوع است که ( ) **SetSpeed** یک متد **Public** است . بنـابراین امکان استفاده از آن توسط یک برنامه نویس وجود خواهد داشت .بمنظور ارسال داده بـه متـد مورد نظر از آرگومان استفاده شده که بین علامت پرانتز قرار می گیرند.در مثال فوق ، پـارامتر مورد نظر **DesiredSpeed** بوده و از نوع **Integer** است . در انتهای علامت پرانتز بسـته ، ازیک کالون ":" ، که بدنبال آن کلمه **Integer** آمده است ، استفاده شده است . ایـن بـدان معنی است که متد ( ) **SetSpeed** یک مقدار صحیح را به برنامه صدازننده ، بر می گرداند. در نمونه کلاس **Vehicle** از دومتد بمنظور افزایش و یا کاهش سرعت استفاده شده است :

-IncreaseSpeed(DesiredSpeed:Integer):Integer
-DecreaseSpeed(DesiredSpeed:Integer):Integer

هر یک از متدهای فوق ، عملیات مورد نظر در رابطه با افزایش و یا کاهش سـرعت را انجـام خواهند داد . برای نیل به خواسته فوق (افزایش و یا کاهش سرعت) می توان دو متد فوق را با یکدیگر تلفیق و در یک متد واحد دیگر جایگزین نمود:

-ChangeSpeed(DesiredSpeed:Integer):Integer

در صورتیکه پارامتر **DesiredSpeed** مثبت باشد ، سرعت افـزایش و در غیـر اینصـورت (پارامتر منفی باشد) ، سرعت کاهش خواهد یافت.

```
Dim MyVehicle as New Vehicle
Dim ObjectSpeed as integer
' Some code that does something...
ObjectSpeed = MyVehicle.GetSpeed()
ObjectSpeed =
MyVehicle.ChangeSpeed(-ObjectSpeed)
```

درنمونه مثال فوق ، در ابتدا یک شی **Vehicle** با نام **MyVehicle** تعریف شده است . در ادامه ، متد **GetSpeed** در ارتباط با شی **MyVehicle** فرا خوانده شده است .بمنظور جداسازی نام شی از متد مربوطه از علامت نقطه استفاده شده است . فرض کنید که **Vehicle** با سرعت ۵۵مایل در ساعت در حال حرکت است . مقدار **ObjectSpeed** ، پنجاه و پنج در نظر گرفته می شود .در صورتیکه در ادامه مقداری منفی را به متد فوق پاس دهیم سرعت کاهش پیدا خواهد کرد. اگر مقدار ۵۵ - را به متد **ChangeSpeed**() پاس دهیم ، توقف اتومبیل را بدنبال خواهد داشت . همانگونه که مشاهده می شود ، برخی از متدها با علامت - شروع شده اند . این بدان معنی است که آنان متدهای اختصاصی (**Private**) بوده و خارج از کلاس قابل دستیابی نخواهند بود. چنین متدهائی به سایر متدها ی موجود در کلاس ، سرویس و خدمات لازم را ارائه و استفاده از آنان برای برنامه نویس مجاز نخواهد بود.بعبارت دیگر متدهای فوق بعنوان اینترفیس کلاس مطرح نبوده و از خدمات آنان در داخل کلاس استفاده خواهد شد . در چنین مواردی ممکن است یک متد که بصورت **Public** تعریف و امکان استفاده از آن در خارج از کلاس و توسط برنامه نویسان وجود دارد ، خود از خدمات چندین متد خصوصی استفاده نماید . یک نمودار کلاس **UML** ، امکان مرور سریع و فشرده پتانسیل ها ی یک کلاس را فراهم و نحوه ارتباط یک برنامه نویس با کلاس مورد نظر را نیز مشخص خواهد شد. اگر یک کلاس را بعنوان یک جعبه سیاه در نظر بگیریم ، علامت "-" ، نشاندهنده آیتم هائی درون جعبه سیاه است که امکان استفاده از آنان توسط برنامه نویسان وجود نخواهد داشت . علامت "+" ، نشاندهنده امکاناتی است که می توان از آنان بمنظور ارتباط با متدها و خصایص یک کلاس استفاده کرد . آیتم های **Public** یک کلاس ، اینترفیس لازم برای یک کلاس را تعریف و نحوه ارتباط با آن را مشخص می نمایند. در حقیقت متدهای **Public** ، نحوه استفاده از یک کلاس را به برنامه نویسان دیکته خواهند کرد .

۵- مهندسی نرم افزار و UML

یکی از مباحث مهم در علم کامپیوتر بحث مهندسی نرم افزار می باشد امروزه شرکت ها بدون داشتن اصول مشخص مهندسی نرم افزار هیچگاه تصمیم به ایجاد سیستم های نرم افزاری نمی گیرند .

همانگونه که می دانید طراحی و تولید سیستم های نرم افزاری دارای یک چرخه حیات می باشد که در علم مهندسی نرم افزار به بررسی این چرخه حیات و عوامل مرتبط با آن پرداخته می شود . به طور کلی مراحل این چرخه به شرح زیر می باشد :

- فعالیت جمع آوری نیازمندی های و مشخص کردن آنها . این نیازمندی ها کاری را که سیستم می بایست انجام دهد را مشخص می کنند .
- فعالیت تحلیل نیازمندی ها برای درک بهتر آنها .
- فعالیت طراحی برای اینکه مشخص شود که سیستم چگونه نیازمندی ها را برآورده می کند .
- فعالیت ساخت سیستم .
- آزمایش سیستم برای تایید اینکه آیا سیستم نیازمندی ها را برآورده کرده است یانه ؟
- و در نهایت فعالیت تحویل سیستم .

حال متدلوژی های مختلفی برای انجام این فعالیت ها وجود دارد و هر کدام به نحوی به انجام این کار ها می پردازند .

متدولوژی

در ابتدا باید به تعریف متدلوژی و اینکه یک متدلوژی چه کاری انجام می دهد پرداخت .

تعریف : متدلوژی یا فراروش مجموعه ای است همگرا و هدف مدار از مفاهیم ، عقاید ، ارزش ها و اصولی که بوسیله منابعی در جهت حل مسایل گروهی بکار گرفته می شود و می خواهد تغییرات مطلوبی را در وضع موجود یک سیستم بطور *غیر تصادفی* ایجاد نماید .

یک متدلوژی در حقیقت سه وظیفه دارد .

۱. فرموله کردن مسئله .
۲. بیان نحوه حل مسئله
۳. پیاده سازی مسئله .

در اینجا هدف ما بررسی متدلوژی های شی گرا می باشد . دیدگاه شی گرایی از اواسط دهه ۷۰ میلادی در مباحث برنامه نویسی کامپیوتر متولد شد . پس از گذشت چند سال و در اوایل دهه ۹۰ به جهت ناکارآمدی روش های سنتی در مباحث تحلیل و طراحی سیستم های

اطلاعاتی و کامپیوتری و نیز ظهور سیستم هایی که مدل سازی آنها به روش های سنتی بسیار ناقص بود ، تحلیل گران و طراحان سیستم را به این فکر انداخت تا از دیدگاه شی گرا علاوه بر برنامه نویسی در زمینه تحلیل و طراحی سیستم نیز استفاده کنند .

از جمله این متدلوژی های شی گرا که در طول سال ها ایجاد گردید می توان متدولوژی های **Fusion** و **Booch** و **rumbaugh** و متدواوژی **Yourdon** را نام برد . تا در نهایت در سال ۱۹۹٦ آقایان **Boch,Rumbaugh** و **Jacobson** در کنار هم گرد آمدهند و پایه های " زبان مدل سازی یکپارچه "(**Unified Modeling Language**) یا همان **UML** را ایجاد کردند . این زبان مدل سازی در سال ۹۷ توسط (**OMG**) **Object Management Group** در آمریکا به عنوان یک استاندارد پذیرفته شد و شرکت های مشهوری نظیر **Oracle** و **Microsoft** از آن پشتیبانی کردند .

بعضی مفاهیم :

Object : به هر شیئ یا مفهوم یا هر چیز قابل درکی که بشود با خصوصیات و رفتار مستقل آنرا از یک محیط بازشناسی کرد **Object** گفته می شود .

هر شی یکسری خصوصیات دارد که به آنها **Attribute** گفته می شود که در واقع یک مقدار یا ارزش مشخصی برای آن به ازای هر شیئ می تواند وجود داشته باشد . از طرف دیگر هر شیئ یکسری رفتار دارد که به آنها **Method** گفته می شود که در واقع

پاسخ هایی است که آن شیئ که تحریکات یا پیغامها یا رخـدادهایی کـه از محـیطش صـادر میشود می دهد .

Class : برای دسته بندی اشیا مفهومی به نام کلاس تعریف می گردد . کـلاس نشـان دهنـده خصوصیات و رفتار گروه خاصی از اشیا می باشد که در آن خصوصیات و رفتار ها مشـترک هستند . مانند وبلاگ .

در واقع کلاس یک مفهوم انتزاعی است و در محیط عملیاتی کـلاس واقعـا وجـود نـدارد و **object** ی از آن کلاس وجود دارد . به عنوان یک مثـال دیگـر کـلاس دانشـجو - **object** دانشجو به شماره ۱۲۳٤٥٦۷۸ . پـس کـلاس یـک مفهـوم اسـت و **object** یـک واقعیـت . هر کلاس هم مانند شیئ یک نام دارد - یکسری **Attribute** و یکسـری **Method** . کـه در واقع **Attribute** ها رفتار **object** هـای آن کـلاس هسـتند و **Method** هـا پاسـخ هـای **object** های آن کلاس .

Instantiation : به معنای ایجاد یک شیئ از یک کلاس . یعنی نمونه ای از آن کـلاس کـه همان شیئ می باشد تولید می گردد .

مسئولیت (**Responsibility**) : چیزی است که به شی اختصاص داده میشود و سه جنبه دارد .

- آنچه که شی راجع به خودش می داند .
- کسانی که شی را می شناسند و با هم ارتباط دارند .
- کارهایی که شی انجام می دهد .

سناریو : یک مجموعه پشت سر هم یا متوالی می باشد که منجر به انجام کار خاصی می گردد .

کلاس : مجموعه ای از اشیا که دارای صفات ، اعمال و ارتباطات یکسان هستند در یک کلاس قرار می گیرند و هر یک از این اشیا نمونه ای (**Instance**) از آن کلاس به حساب می آیند .

در ادامه و مقالات آینده برای اینکه مفاهیم **UML** و عناصر آن را بهتر متوجه شوید کار را با بررسی یک مورد آموزشی ادامه خواهیم داد . این مورد یک سیستم مدیریت پروژه می باشد .

توجه : در اینجا هدف پیاده سازی و انجام یک پروژه کامل نمی باشد بلکه سعی می شود در خلال این مثال به بررسی **UML** بپردازیم .

یک سیستم مدیریت پروژه می بایست توانایی مدیریت پروژه ها و منابع و راهبری سیستم را داشته باشد .بنابراین نقش های زیر در آن تعریف می شود .

- مدیر پروژه (**Project Manager**) : این نقش مسئول تضمین تحویل محصولی با کیفیت در زمان و هزینه مشخص ، با توجه به محدودیت منابع می باشد .

- مدیر منابع (**Resource manager**) : این نقش مسئول تضمین تـامین منـابع انسانی آموزش دیده و ماهر برای انجام پروژه ها می باشد .
- مدیریت افراد (**Human Manager**) : این نقش مسئول تضمین بـروز بـودن و کافی بودن مهارت های افراد و کامل بودن کیفیت انجام کار برای یک پروژه می باشد .
- مدیر و راهبر سیستم (**System Administrator**) : مسـئول تضـمین فـراهم نمودن سیستم مدیریت پروژه برای یک پروژه می باشد .

٦- روند حرکت به سمت UML در جهان:

قبل از ارائه **UML**، زبان مدلسازی استانداردی وجود نداشت و استفاده‌کنندگان مجبور بودند از میان زبانهای مختلف موجود که هیچیک تقریباً کامل نبودند و تفاوتهایی با هم داشتند، یکی را انتخاب کنند. تفاوتهای زبانهای مدلسازی، چندان قدرت مدلسازی را افزایش نداده بود، اما در عوض باعث افول صنعت شی‌گرایی و سردرگمی کاربران شـده بـود. در چنـین شـرایطی طبیعی بود که استقبال زیادی از یک زبان مدلسازی استاندارد که ویژگیهای بارز زیادی داشت، بشود. بسیاری از شرکتها در همان اوایل کار به **UML** روی آوردند و تعداد دیگری نیز پـس از تثبیت **UML**، آن را به عنوان استراتژی تولید ومستندسازی خود پذیرفتند.

OMG که کنسرسیومی است متشکل از ۷۰۰ شرکت معتبر آمریکا، از **UML** حمایت کـرد و آن را به عنوان زبان مدلسازی استاندارد خود اعلام کـرد. البتـه عـلاوه بـر اسـتاندارد شـدن، حمایـت جداگانـه شـرکت‌هـای بـزرگ دنیـا مثـل **Hewlett-Packard**، **I-Logix**، **Microsoft**، **IBM**، **Oracle** و بسیاری دیگر، خود سبب افزایش کـاربرد آن در محافـل صنعتی و نرم‌افزاری دنیا گردید. امروزه نیز با ارائه نسخه ۱.۳ و رفع مشکلات گذشته، روز بـه روز بر کاربران آن افزوده می‌شود.

خلاصه

- الگوریتم ،اعلامیه ای سازماندهی شده بمنظور حل یک مسئله خاص و یـا سـوالاتی است که می بایست پاسخ آنان مشخص گردد . یک الگوریتم مناسب ، تمامی مراحل لازم بمنظور انجام فرآیندهای مورد نیاز و حل یک مسئله را ارائه می نماید .
- مقداردهی اولیه ، ورودی ، پردازش ، خروجی و پاکسازی ، پنج مرحله متفاوت برنامه نویسی می باشند .
- مراحل پنج گانه برنامه نویسی ، نگرشی ماکرو از یک برنامه را ارائه می نمایند . مثلا" مرحله ورودی ممکن است نیازمند اخذ داده از صفحه کلیـد ، خوانـدن یـک جـدول تنظیمات از یک بانک اطلاعاتی و نهایتا" خواندن اطلاعات بیشتر از بانـک اطلاعـاتی دیگر باشد . بهسازی (پالایش) یکطرفه ، فرآیندی است که بـر اسـاس آن یکـی از مراحل برنامه نویسی (نظیر مرحله ورودی) بررسی و به آن جزئیات بیشتری اضـافه اضافه خواهد شد . عملیات فوق تا استخراج و مشخص شدن تمامی جزئیات لازم در رابطه با یک مرحله خاص ادامه خواهد یافت . عملیات بهسازی (پالایش) یکطرفـه ، زمانی متوقف می گردد که کد واقعی یک تابع نوشته گردد .محوریت فرآینـد فـوق ، تبدیل الگوریتم های ماکرو به میکرو است :

```
Input Step->ReadKeyboard( )
ReadSetupTable( ) -
>ReadTable1( ) ->(Code)
```

ReadTable2( )
ReadTable3( )

- **UML** ، از کلمات **Unified Modeling Language** اقتباس شده است . مزیت استفاده از **UML** ، تفکر مبتنی بر برنامه نویسی شی گراء است .بلاک های اولیه ایجاد **UML** کلاس ، خصلت و متد نامیده می شوند. نمودار های کلاس **UML** تمام سه عنصر **OOP** را در یک نمودار مناسب نمایش می دهند .
- در **OOP** ، واژه های **Private** و **Public** به نحوه دستیابی به خصلت ها بر می گردد . اگرخصلتی از نوع **Private** باشد ،امکان تغییر آن صرفا" برای کسانی که به کلاس فوق تعلق دارند، وجود خواهد داشت .اگر خصلتی از نوع **Public** باشد ، سایر اشیاء امکان دستیابی کامل به خصلت را (اعمال تغییرات مورد نظر) خواهند داشت.