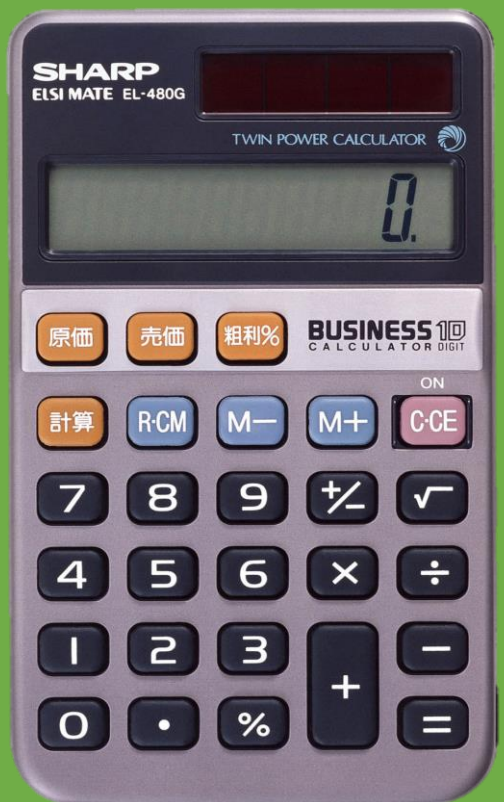
آموزش UML



میثم پاپی مطلق

به نام او

آموزش UML

میثم پاپی مطلق

نشر اینترنتی جامعه خردمندان پارسی

فهرست مطالب

مقدمه ای بر متد Obiect-Oriented (شیءگرایی)

شیءگرایی (Object-Oriented) لغتی است که امروزه در صنعت نرم افزار، باب شده است. شرکتها به سرعت حرکت می کنند تا خود را با این تکنولوژی سازگار کنند و آن را در برنامه های خود وارد نمایند.

متد شیءگرایی (O.O) یک راه متفاوت مشاهده برنامه هاست. با متد شیءگرایی، شما یک برنامه را به قطعات بسیار کوچک یا آبجکت هایی تقسیم می کنید، که تا اندازه ای مستقل از یکدیگر می باشند. مانند ساختمانی از بلوک ها نگاه کنید.

اولین قدم این است که آبجکت های اساسی (انواع مختلف بلوک ها) را بسازید یا بدست آورید. اولین باری که شما این بلوک های ساختمانی را دارید، می توانید آنها را کنار هم گذاشته و قصرتان را بسازید. به محض اینکه تعدادی آبجکت های اساسی را در دنیای کامپیوتر ساختید یا بدست آوردید، می توانید به سادگی آنها را کنار هم بگذارید تا برنامه‌های جدید ایجاد را کنید. یکی از امتیازات اساسی متد شیءگرایی این است که می

توانید یک بار Component (اجزاء) را ساخته و بارها و بارها از آنها استفاده کنید. درست مانند زمانی که می توانید یک بلاک ساختمانی را در یک قصر، یک خانه یا یک سفینه فضایی دوباره استفاده کنید، می توانید از یک قطعه طرح یا کد شیءگرایی در یک سیستم حسابداری، یک سیستم بازرگانی یا یک سیستم پردازش سفارش استفاده مجدد نمایید.

تفاوت متد شیءگرایی با روش سنتی توسعه، چیست؟ در روش سنتی، روش توسعه به همراه اطلاعاتی که سیستم نگهداری خواهد کرد به خودمان وابسته است.

در این روش، ما از کاربران می پرسیم که چه اطلاعاتی را نیاز دارند، پایگاه داده ای را طراحی می کنیم که اطلاعات را نگه دارد، صفحاتی را تهیه می کنیم تا اطلاعات را بگیرد، و گزارشاتی را چاپ می کنیم تا اطلاعاتی را برای کاربر نمایش دهد. به عبارت دیگر، ما بر روی اطلاعات متمرکز می شویم و کمتر توجه می کنیم که چه کاری با این اطلاعات انجام شده یا رفتار سیستم چگونه است. این روش data-centric (مبتنی بر داده)

نامیده شده است و برای ایجاد هزاران سیستم در سال، ایجاد شده است. مدلسازی data-centric مخصوص طراحی پایگاه داده و گرفتن اطلاعات خیلی مهم می باشـــد، اما انتخاب این روش در زمان طراحی برنامه های تجاری با مشکلاتی همراه است. یک چالش بزرگ این است که درخواستهای سیستم چندین بار تغییر خواهند کرد. سیستمی که از روش data-centric اســـتفاده می نماید، می تواند به آســـانی تغییر در پایگاه داده را مدیریت کند. اما اجرای تغییرات در قوانین تجاری یا رفتار(behavior) سیستم آن قدر آسان نمی باشد. متد شیءگرایی در پاسخ به این مشکل، ایجاد شده است. با متد شیءگرایی هم بر اطلاعات وهم بر رفتار متمرکز می شویم. در نتیجه اکنون می توانیم سیستم هایی را ایجاد کنیم که انعطاف پذیر شده اند تا اطلاعات و یا رفتار را تغییر دهند.

مزیت این انعطاف پذیری با طراحی یک سیستم شیءگرایی به خوبی شناخته شده است. این مطلب، به شناخت تعدادی اصول شیء گرایی نیاز

دارد. نهان سازی (Encapsulation) وراثت(Inheritance) و چند ریختی (Polymorphism).

Encapsulation (نهان سازی)

در سیستمهای شیءگرا، اینها (اطلاعات و رفتارها) را در یک آبجکت بسته بندی می کنیم. این مطلب در قالب اطلاعات Encapsulation (پنهان سازی) ارجاع داده شده است. راه دیگر برای نگاه کردن به توابع وابسته، این است که برنامه را به بخشهای کوچکی از توابع وابسته، تقسیم کنیم. مثلاً یک حساب بانکی شامل: شماره حساب، تراز جاری نام مشتری آدرس نوع حساب، نرخ بهره و تاریخ باز کردن حساب می باشد. همچنین رفتارهایی را برای یک حساب بانک داریم مانند: باز کردن یک حساب ، بستن حساب، به حساب گذاشتن، برداشت از حساب، تغییر نوع حساب، تغییر مشتری و تغییر آدرس. ما این اطلاعات و رفتارها را با هم در یک آبجکت

account پنهان می کنیم. در نتیجه همهٔ تغییرات سیستم بانکی مربوط به حسابها، می توانند به آسانی در آبجکت حساب انجام شوند.

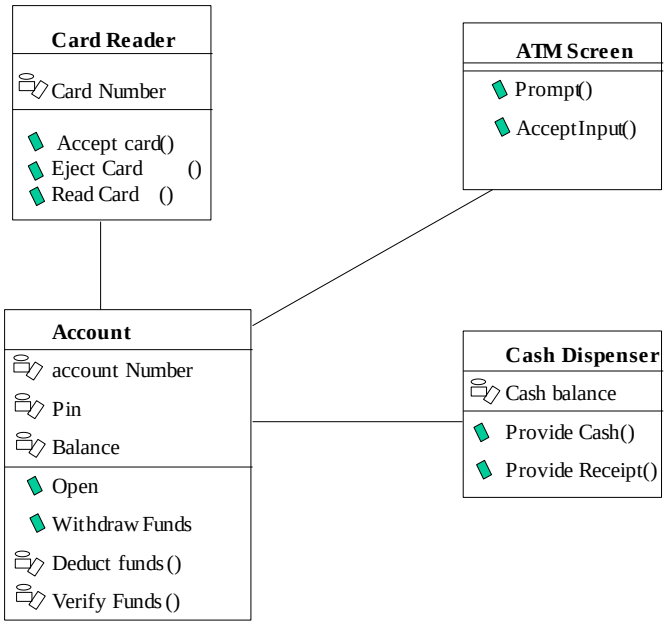
مـزیت دیگر پنهان سازی این است که تأثیرات اعمال شده به سیستم را محدود می کند. به یک سـیستم به عنوان بسـتری از آب و به تغییر درخواستها مانند یک صخره بزرگ نگاه کنیم. شما صخره را در آب می اندازید و امواج بزرگی در همه جهتها ایجاد می شـوند. آنها در سـرتاسـر دریاچه حرکت می کنند، به کرانه ضربه می زنند، طنین افکن می شوند و با امواج دیگر برخورد می کنند در حقیقت، حتی ممکن است مقدای آب بر روی سـاحل و خارج از دریاچه بریزد. بعبارت دیگر، برخورد صـخره با آب باعث ایجاد میزان زیادی موج های کوچک شده است. حال دریاچه خود را پنهان می کنیم. تدین ترتیب که آن را به تکه های کوچکتری از آب با موانعی میان آنها تقسیم می کنیم. سپس، ضربات سیستم را تغییر می دهد. قبل از این، امواج در همه جهتها ایجاد می شدند. اما اکنون، امواج فقط می

توانند از یکی از موانع عبور نماید. و سپس متوقف می گردند. بنابرین، با نهان سازی دریاچه، ما تاثیر موج کوچک حا صل از انداختن صخره در آب را محدود کرده ایم.

حال، بیایید ایدهٔ نهان سازی را درسیستم بانکی به کار ببریم. اخیراً مدیریت بانک تصمیم گرفته است که اگر مشتری در بانک یک حساب اعتباری دارد، بتوان از حساب اعتباری بعنوان یک مبلغ اضافه، برداشت کرده و برای حساب جاری آنها استفاده نمود. در یک سیستم غیر نهان سازی، کار را با یک روش اجباری شروع می کنیم تا تجزیه و تحلیل کاراتر شود. اساساً، ما محل تمام جاهایی که ازعملیات برداشت از حساب، در یک سیستم استفاده شده است را نمی دانیم،

بنابرین باید به هر جایی نگاه کنیم و وقتی که آن را پیدا کردیم، باید یک سری از تغییرات را ایجاد کنیم تا این درخوا ست جدید را یکپارچه کنیم. اگر کار به درستی انجام شده باشد، احتمالاً ۸۰ ٪ موارد برداشت از حساب را در سیستم پیدا کرده ایم. با یک سیستم نهان سازی، ما نیازی به استفاده

از روش اجباری برای تجزیه و تحلیل نداریم. ما به مدل سیستم خود نگاه می کنیم و به آسانی جایی که رویداد برداشت از حساب پنهان شده بود را پیدا می کنیم. بعد از اینکه عملیات را در حساب قرار دادیم، یک بار درخواستمان را فقط در آن آبجکت تغییر می دهیم، و کار ما تمام شده است. همان گونه که در شکل زیر می بینید، فقط کلاس Acount نیاز به تغییر دارد.



یک مفهوم مشابه نهان سازی، Information Hiding (پنهان سازی) می باشد. پنهان سازی اطلاعات، توانایی است که جزئیات مبهم یک آبجکت را از دنیای خارج پنهان می سازد. در یک آبجکت، دنیای خارج به معنی هر چیزی خارج از همان آبجکت اســـت حتی اگرچه دنیای خارج شـــامل بقیهٔ

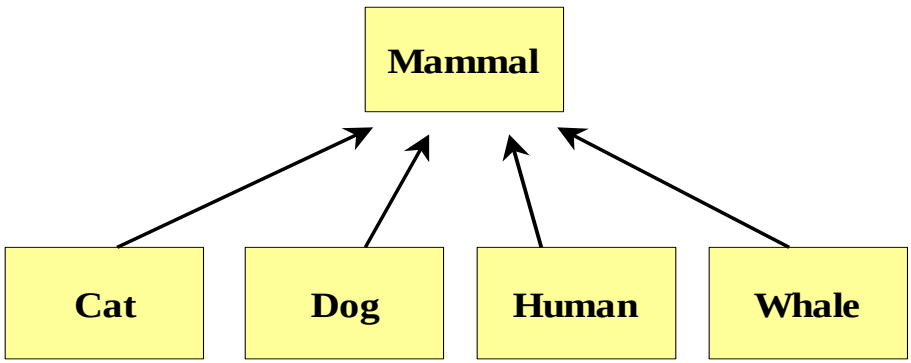
سیستم باشد. پنهان سازی اطلاعات (Information Hiding) همان مزیت نهان سازی (Encapsolation) را فراهم می کند.

Inheritance (وراثت)

Inheritance (وراثت) دومین مفهوم اساسی شیءگرایی می باشد. در سیستمهای شیءگرا به شما اجازه می دهد تا آبجکت های جدید را بر پایه آبجکت های قدیمی ایجاد کنید. آبجکت child (فرزند) ویژگیهای یک آبجکت parent(والد) را به ارث می برد. شما می توانید نمونه هایی از وراثت را در دنیای طبیعی ببینید. هزاران نوع مختلف از پستانداران مانند سگها، گربه ها، انسانها، نهنگها و غیره وجود دارند. هر یک از اینها ویژگیهای مشخصی دارند که منحصر به فرد بوده و این ویژگیهای مشخص مانند داشتن مو، خونگرم بودن و غذا دادن به بچه ها، در کل گروه مشترک می باشد. در اصطلاحات شیءگرایی یک آبجکت بنام mammal (پستاندار)

وجود دارد که ویژگیهای مشترک را نگه می دارد. این آبجکت والد آبجکت های فرزندی مانند گربه، سگ، انسان، نهنگ و غیره می باشد.

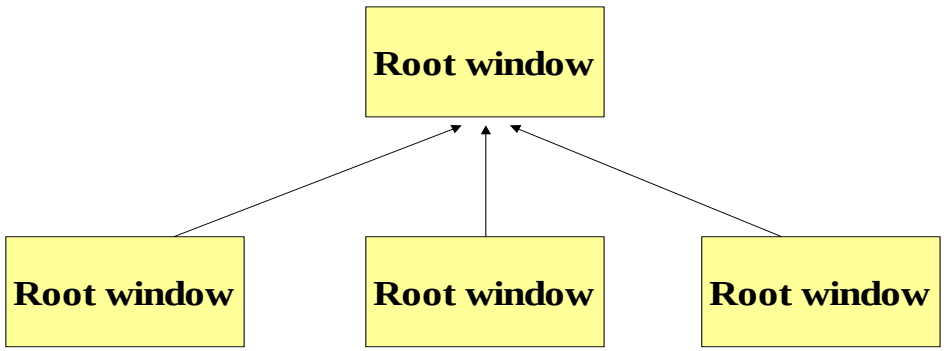
آبجکت سگ(dog) ویژگیهای آبجکت پستاندار (mammal) را به ارث می برد، مانند دویدن در دایره ها (حلقه ها) و ریختن آب از لب و لوچه. متد شیءگرایی ایدهٔ وراثت را از دنیای طبیعی گرفته است. شکل زیر نمونه ای از آن است.



بنابراین ما می توانیم همان مفهوم را در سیستم خود به کار ببریم. یکی از مزایای ا صلی وراثت، سهولت در نگهداری ا ست. وقتی چیزی تغییر می کند و برهمهٔ پستانداران اثر می گذارد، فقط آبجکت والد نتاز به تغییر دارد و آبجکت های فرزند به طور خودکار تغییرات را به ارث می برد. اگر پ ستانداران به طور ناگهانی خوذ سرد شوند، فقط پ ستاندار (mammal) باید تغییر نماید. گربه، سگ، ان سان، نهنگ و دیگر آبجکتهای فرزند به طور

خودکار ویژگی جدید خونسرد بودن پستانداران را به ارث می برند. در یک سیستم شیءگرا یک مثال می تواند پنجره ها باشند.

سیستمی بزرگ دارای ۱۲۵ تابلو داریم. روزی، یک مشتری درخواست یک پیغام انصراف بر روی همه تابلوها را می دهد. در یک سیستم بدون وراثت، ما کار خسته کننده ای داریم که باید به هر یک از این ۱۲۵ تابلو رفته و تغییر را در آن ایجاد کنیم. تا به آبجکت والد رفته ویکبار آن را تغییر دهیم. همهٔ تابلو ها به طور خود کار تغییرات را به ارث می برند. همانطور که در شکل زیر می بینید.



در یک سیستم بانکداری، ممکن است از وراثت برای انواع مختلفی از حسابهایی که داریم استفاده نماییم. بانک فرضی ما دارای چهار نوع مختلف حساب می باشد. جاری، پس انداز، کارت اعتباری، سپرده گذاری.این انواع مختلف حسابها شباهتهایی نیز دارند. هر کدام دارای یک شمارهٔ حساب، نرخ بهره و نام مالک می باشد. بنابراین می توانیم یک آبجکت والد بنام

account (حساب) را ایجاد کنیم تا ویژگیهای مشترک همهٔ این حسابها را نگهداری کنیم.

آبجکت های فرزند (child) می توانند علاوه بر ویژگیهایی که به ارث برده اند، ویژگیهای منحصر به فرد خودشان را داشته باشند. مثلاً، حساب اعتباری یک حد موجودی و حداقل میزان پرداخت را خواهد داشت. سپرده گذاری نیز دارای یک موعود پرداخت می باشد.

تغییرات آبجکت والد بر روی همهٔ فرزندان اثر خواهد گذاشت، اما بچه‌ها آزاد هستند که بدون به هم زدن آرامش فرزند دیگر یا والد شان تغییر نمایند.

Polymorphism(چند ریختی)

سومین اصل شیءگرایی Polymorphism (چند ریختی) است. در فرهنگ لغت بعنوان پیدایش شکلهای مختلف، نواحی یا انواع مختلف تعریف شده است. چند ریختی به این معنی است که شکلها یا پیامدهای زیادی از

یک تابع ویژه را داشـــته باشـــیم. همانند وراثت چند ریختی نیز در دنیای طبیعی دیده می شـود. در فرمان یا عمل صـحبت کردن ممکن اسـت یک انسـان جواب دهد «شـما چه طورید»، سـگ شـاید جواب دهد «واق واق» گربه ممکن است پاسخ دهد «میو».

چند ریختی در اصطلاحات یک سیستم شیءگرایی به این معنی است که ما می توانیم بسـیاری از رخدادها یا پیامدهای یک عمل ویژه را داشـته باشیم.

مثلاً، ممکن است یک سیستم رسم اشکال گرافیکی را بسازیم. وقتی کاربر می خواهد چیزی مانند یک خط، دایره یا یک مسـتطیل را بکشـیم، سیستم یک فرمان draw را می فرستد. سیستم با انواع مختلف شکلها سـازگار شـده اسـت، هرکدام شـامل رفتاری اسـت تا خودش را بکشـد. بنابراین وقتی که کاربر می خواهد یک دایره را بکشد، فرمان رسم آبخکت دایره (draw) درخواست خواهد شد. بااستفاده از چند ریختی، سیستم می

فهمد که در هنگام اجرا شدن کدام نوع شکل کشیده شده است. بدون چند ریختی، کد تابع draw ممکن است مانند زیر باشد.

```
Function Shape.drawMe()
{
        CASE Shape.Type
                Case “Circle“
                        Shape.drawCircle();
                Case
“Rectangle“
Shape.drawCircle();

                Case “Line“
Shape.drawCircle();
        END CASE

}
```

با چند ریختی، کد draw باید توسط تابع drawme() برای آبجکت طراحی شده فراخوانی شود. مانند این مثال:

```
Function draw ()
{
    Shape.drawme();
}
```

هر شکلی (دایره، خط وغیره) باید تابع drawMe را دا شته با شد تا شکل بخصوصی را بکشد. یکی از مزایای چند ریختی مانند دیگر اصول شیءگرایی، نگهداری است.

زمانی که لازم است تا برنامه یک مثلث را رسم نماید، چه اتفاقی می‌افتد؟ در موردی که از چند ریختی استفاده نشده است یک تابع darw triangle() جدید به آبجکت shape اضافه شده است. همچنین تابع

drawme() آبجکت shape تغییر کرده اســت تا با نوع جدید شــکل ســازگار شــود. با چند ریختی ما یک آبجکت triangle (مثلث) جدید را درون تابع drawMe() ایجاد می کنیم تا خودش را رســم نماید. تابع Draw() که عملگرهای طراحی را اجرا می نماید اصلاً نباید تغییر کند.

مدلسازی بصری (Visual Modeling) چیست؟

اگر چیز جدیدی را برای خانهٔتان می ســازید، احتمالاً فقط با خریدن یک تکه چوب و بستن آن به هم تا که در ست به نظر آید،این کار را انجام نمی دهید. شــما تعدادی طرح کلی می خواهید، تا آنها را دنبال نمایید. بنابرین می توانید قبل از شــروع به کار، آن چیز را طراحی و ســاختاربندی کنید. مدل ها در دنیای نرم افزار همان کار را برای ما انجام می دهند. آنها طرحهای کلی سیستم می باشند. یک طرح کلی به شما کمک می کند تا یک چیز اضافی را قبل از اینکه بسازید، طراحی کنید. یک مدل به شما کمک می کند تا قبل از اینکه یک سیستم را بسازید، آن را طراحی کنید. به شما

کمک می کند تا مطمئن شـــوید که طرح بی نقص می باشـــد،درخواســـتها دیده شـــده اند و ســـیســـتم می تواند حتی در مقابل کوهی از تغییرات درخوا ست، مقاومت نماید. هنگامی که درخوا ستهایی را برای سیستم خود جمع می کنید، نیازهای تجاری کاربران را گرفته و آنها را به درخواســتهایی که تیم شما می تواند از آنها استفاده کند وبفهمد، تبدیل می کنید. سرانجام شما می توانید این درخواستها را گرفته و از آنها کدی تولید کنید. با تبدیل ر سمی درخوا ستها به کد، می توانید مطمئن شوید که درخوا ستها بو سیله کد مطرح شـــده اند، و آن کد می تواند به آســـانی راه برگشـــت به درخوا ستها را طی کند. این پردازش modeling (مدلسازی) نامیده شده ا ست. نتیجه پردازش مدل سازی این توانایی ا ست که نیازهای تجاری را به درخواســتهایی تبدیل کند تا درکد به صــورت مدل درآید و دوباره آن را برگرداند بدون اینکه در طول راه چیزی گم شود.

مدلسازی بصری (Visual Modeling) پردازش گرفتن اطلاعات از مدل است و آنرا با استفاده از مجموعه ای از عناصر گرافیکی استاندارد به

صـــورت گرافیکی نمایش می دهد. یک اســتاندارد ضــروری برای فهمیدن یکی از مزایای مدل سازی بـ صری، ارتباطات ا ست. هدف ا صلی مدل سازی بـ صری ارتباط میان کاربران، برنامه نویـ سان، تحلیلگران، آزمایش کننده ها، مدیران و هر شخص دیگری که با یک پروژه درگیر شده است، می باشد.

به نظر می رسد که ما قادر هستیم پیچیدگی را بهتر درک کنیم البته وقتی که برخلاف متن نوشته شده، به صورت بصری به نمایش داده شود. با تولید مدلهای بصـــری یک ســـیســـتم، می توانیم نشـــان دهیم که چگونه سیستم روی چند سطح کار می کند. ما می توانیم فعل و انفعالات بین کاربر ویک سیستم را مدل نماییم.

ما می توانیم فعل و انفعالات آبجکت ها را در محدودهٔ یک ســیســـتم مدل کنیم. همچنین اگر برای ما مطلوب باشـــد می توانیم حتی فعل وانفعالات میان سیستم ها را مدل نماییم.

بعد از ایجاد این مدلها می توانیم آنها را به همهٔ بخشـــهای وابســـته نشان دهیم و آن بخشها می توانند اطلاعات را از مدل بدست آورند. مثلاً

کاربران می توانند فعل و انفعالاتی که با یک سیستم خواهند داشت را از طریق نگاه کردن به یک مدل مجسم نمایند، تحلیلگران می توانند فعل و انفعالات میان آبجکت ها را از طریق مدلها پیش بینی کنند.

برنامه نویسان می توانند آبجکت هایی که نیاز است برنامه نویسی شوند و آنهایی که باید به نتیجه برسند را پیش بینی کنند. آزمایش کنندگان می توانند فعل و انفعالات میان آبجکت ها را پیش بینی کنند و نمونه های آزمایش را بر اساس این فعل و انفعالات آماده کنند. مدیران پروژه می توانند کل سیستم را ببینند و اینکه چگونه بخشها را بر روی هم اثر می گذارند. و کارمندان ارشد اطلاعات (Chief Information Officer) می توانند به مدل های سطح بالا نگاه کنند وببینند که چگونه سیستم ها در سازمانهایشان بر روی یکدیگر اثر می گذارند. روی هم رفته، مدلهای بصری ابزار قدرتمندی را فراهم می کنند تا سیستم پیشنهاد شده را به همهٔ بخشهای وابسته نشان دهند.

Booch, OMT, and UML

یک موضـــوع مهم در مدلســـازی بصـــری این اســـت که از چه نماد گرافیکی برای دادن چهرهای مختلف یک سیستم استفاده شود. لازم است که این نماد به همهٔ بخشهای وابسته رسانده شود، در غیر اینصورت مدل خیلی مفید نخواهد بود. بســـیاری از افراد نمادهایی را برای مدلســـازی بصری پیشنهاد نموده اند. بعضی از این نمادهای محبوب که پشتیبانی قوی دارند، Object Modeling Technology (OMT), Booch (تکنولوژی مدلســـازی شـــیء) و unified Modeling Language (UML) (زبان مدلسازی یکپارچه) میباشند. برنامه Rational Rose 98 از این این سه نماد پشتیبانی می کند.

نمودارهای UML

UML به افراد اجازه می دهد تا چندین نوع مختلف از نمودارهای بصـــری را به وجود آورند ککه جنبه های مختلف ســـیســـتم را نمایش می

د هد. Rational Rose از ایجاد اکثر این مدل‌ها، همانطور که در زیر آمده، پشتیبانی می کند.

نمودار Use Case

نمودار Sequence (توالی)

نمودار Collabration (همکاری)

نمودار Class (کلاس)

نمودار State Transition (حالت)

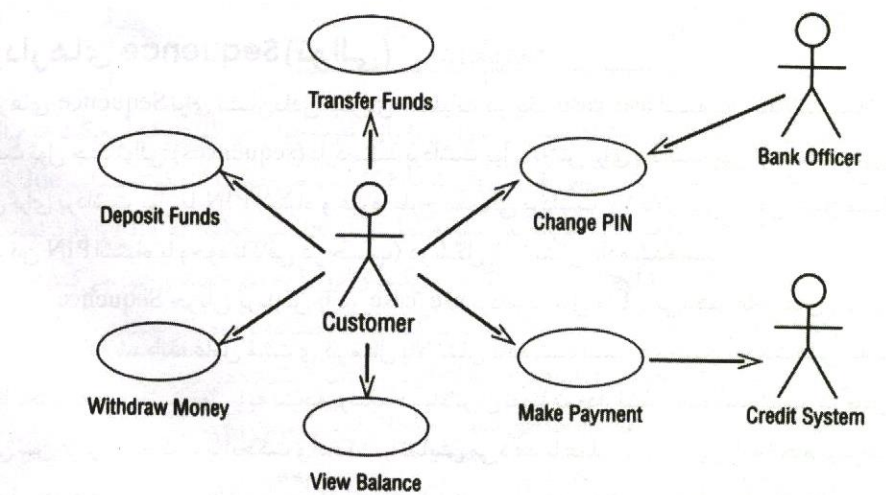
نمودار Component

نمودار Deployment

این نمودار های مدل، جنبه های مختلف سیستم را نشان می دهند. مثلاً نمودار Collaboration محاورات ضروری میان آبجکت ها را نشان می دهد، به این منظور که تعدادی از توابع سیستم را به انجام برساند. هر نمودار یک هدف و یک شنوندهٔ در نظر گرفته شده دارد.

نمودارهای Use Case

نمودارهای Use Case محاورات میان Use Case ها را نشان می دهند، که عملیات سیستمی و عامل ها (Actor) که نشاندهندهٔ افراد یا سیستم هایی است که اطلاعات را برای سیستم فراهم کرده و یا از آن دریافت می کنند را نمایش می دهند. مثلاً نمودار Use Case سیستم (ATM) در زیر نشان داده شده است.



نمودارهایUse Case محاورات میان Use Case و عامل ها را نشـان می دهند، Use Case ها درخواسـتهای سـیسـتم را از دید کاربر نشان می دهند، بنابراین Use Case ها عملیاتی هستند که سیستم فراهم می کند. عامل ها در واقع نگهدارنده پول (بانکدار) یک سیستم هستند. این نمودارها نشان می دهند که چه عامل هایی به Use Case ها مقدار اولیه می دهند.

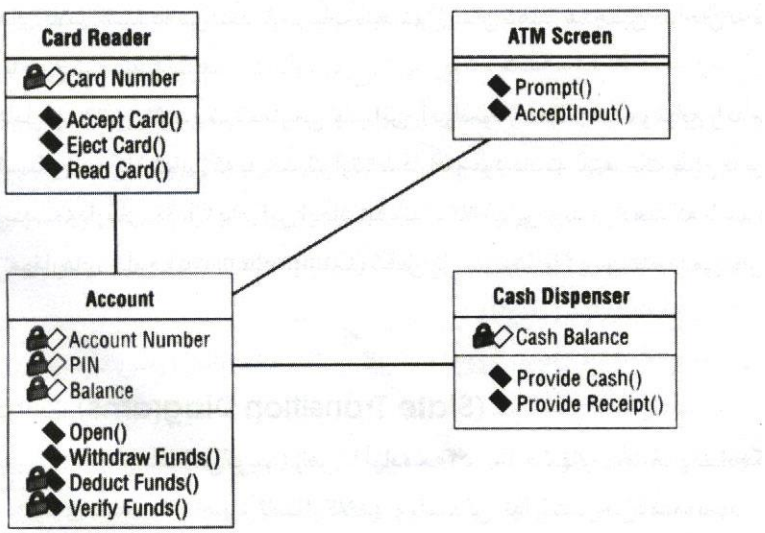
نمودارهای CLASS (کلاس)

نمودارهای CLASS (کلاس) ارتباطات بین کلاسهارادر سیستم نشان می دهد. کلا سها می‌توانند بعنوان طرحی کلی برای ابجکت ها دیده شوند که در ف صل ۵ درباره آنها بحث خواهیم کرد.مثلا د ساب JOE یک کلاس است.

کلاس ها شامل اطلاعات و رفتاری ه ستند که بر روی اطلاعات عمل می‌نمایند. کلاس حساب (account) شامل PIN را کنترل می‌کند میباشد.

در نمودار class برای هر نوع آبجکتی در نمودار Sequence و Collabration یک کلاس ایجاد شـــده اســـت. نمودارClass در use case برداشت پول در شکل ۱۱-۱ توضیح داده شده است.

شکل ۱۱-۱ : نمودار Class در use case برداشت پول از حساب متعلق به سیستم ATM



نمودار Class بالا ارتباطات بین کلاســـهایی را نشـــان می دهد که use case بردا شت پول را به انجام می ر سانند. این کار با چهار کلاس انجام شده است.

Cash dispenser (صـــندوق). دریک نمودارClass هر کلاس با مستطیلی نشان داده شده که به سه بخش تقسیم شده است. بخش اول نام کلاس را نشان می دهد.

بخش دوم صـــفات کلاس (Attrieutes) نشـــان می دهد.یک صـــفت قطعه ای از اطلاعاتی اســـت که با یک کلاس مرتبط می باشـــد.مثلا کلاس حساب (account) شامل سه صفت است. Account Number (شماره حساب)

PIN وBalance (تراز). اخرین بخش شـــامل عملگرد های کلاس (Operations) می باشد. یک عملگر تعدادی رفتار است که توسط کلاس آماده خواهد شـود. کلاس حسـاب (account) شـامل چهار عملگر اسـت Open. (باز کردن) Withdraw(برداشـــت وجوه) Deput Funds (واریز وجوه) و Verify funds (تایید موجودی).

خطوط بین کلاسها وابستگی ارتباطات بین کلاسها را نشان می دهد. برای نمونه کلاس حسـاب (Account) به کلاس ATM Screen وصـل

شـده اسـت . زیراهر د.و مسـتقئما با دیگری ارتباط دارند.Card reader (کارت خوان) یه CASH DISPENSER وصل نشده است زیرا این دو با هم ارتباطی ندارند. نقطه قابل توجه دیگر این ا ست که تعدادی از صفتها و عملکردها قفلهای کوچکی د سـمت چپشـان دارند و قفل یک صـفت یا عملگرد PRIVATE (خصوصی) فقط می توانند از طریق کلاسی که شامل انهاست قابل دستیابی باشند.

ACCONT NUMBER (شماره حسا ب) PIN و balance همه صفات private (خ صو صی) کلاس د ساب (account) ه ستند. به علاوه عملگرد deluct funds (واریز پول) وVerify funds (تایید موجودی) برای کلاس حسـاب (Account) هسـتند. به علاوه عملگر های Deluct funds (واریز پول) و Verify Funds (تـاییـد موجودی) برای کلاس حساب (Accunt) Prevate (خصوصی) هستند.

برنامه نویسان از نمودارهای CLASS استفاده می کنند تا که کلاسها را به طور وا قعی تولید نمایند. ابزارهایی مانند Rose چارچوب کلاسـها را

تولید می کنند سپس برنامه نویسان جزئیات را در زبان انتخابی خود نشان می دهند. تحلیل گران از نمودارهای کلاس استفاده می کنند سپس برنامه نویسان جزئیات را در زبان انتخابی خود نشان می دهند. تحلیل گران از نمودارهای کلاس استفاده می کنند تا جزئیات سیستم را نشان دهند. همچنین طراحان به نمودارهای Class نگاه می کنند تا جزئیات سیستم را نشان دهند. همچنین طراحان به نمودارهای Class نگاه میکنند تا طرح سیستم رابینند.

اگر یک کلاس شامل چند تابع باشند یک معمار می تواند این را در نمودارClass دیده وتوابع را به چند کلاس بشکند. بناید هیچ وابستگی بین کلاسهایی که با یکدیگر ارتباط دارند وجود داشته باشند. یک طراح یا برنامه نویس نیز می تواند این را ببیند.

نمودارهای Class برای این ایجاد شده اند تا کلاسهایی را نشان دهند که با هم در هر use case کار کنند و نمودار های جامع (

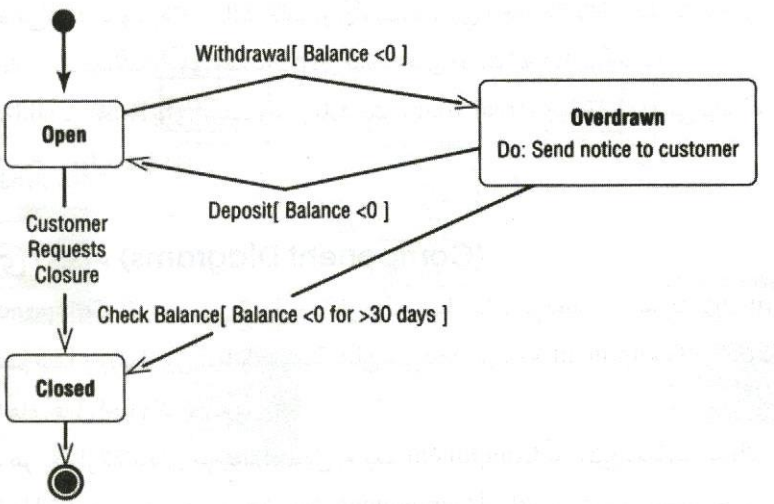
Comprehensive) شامل کل سیستم یا زیر سیستم را می توان به همین ترتیب ایجاد نمود.

نمودارهای حالت (State Transition Diagrams)

نمودارهای حالت (ما به آن حالت می گوییم) راهی را آماده می‌کنند تا حالتهای مختلف یک آبجکت را مدل کنند . در حالی که نمودار های Class یک تصویر ثابت از کلاسهاووابستگی آنها را نشان می‌دهند نمودارهای حالت استفاده می شوند تا بیشتر رفتارهای پویای یک سیستم را نمایش می دهند. یک نمودار حالت رفتار یک آبجکت را نشان می دهند. یک نمودار حالت زفتار یک آبجکت را نشان می دهد. مثلا یک حساب بانکی می توا ند به چندین حالت متفاوت وجود دا شته با شد. می تواند باز شود بسته شودیابه طوراضافی (بیشتر از موجودی) از حساب برداشته شود. یک

حساب ممکن است درهر یک از این حالت ها به طور متفاوتی رفتار کند. از نمودارهای حالت برای نشان دادن این اطلاعات استفاده می شود.

شکل ۱۲-۱ : نمودار تغییر حالت متعلق به کلاس حساب (Accourt)



در این نمودار میتوانیم حالت های مختلف یک حساب را ببینیم. همچنین می توانیم ببینیم که چگونه یک حساب از یک حالت به حالتی دیگر منتقل می شود. مثلا وقتی یک حساب باز است مشتری درخواست بستن حساب را می‌کند حساب به حالت بسته منتقل می شود. درخواست مشتری

Event (رخداد) نامیده می شـــود و رخداد چیزی ا ســـت که موجب می شود یک انتقال از حالتی به حالت دیگر صورت گیرد. اگر حساب باز است و مشتری برداشت از حساب را انتخاب می‌کند، حساب ممکن است به حالت برداشـــت برود. این فقط زمانی افتاق خواهد افتاد که تراز (موجودی) حساب کمتر از صفر باشد. ما این را با قراردادن [۰>تراز] در نمودار نشان می‌دهیم. یک شـــرط که در براکت محصـــور شـــده اســـت، Guard Condition (شرط حفاظتی) نیمده می‌شود و وقوع یک انتقال (اینکه بتواند یا نتواند اتفاق بیافتد) را کنترل می‌کند.

درحالت ویژه، Start State (حالت شـــروع) و Stop State (حالت پایان) وجود دارد. حالت شـــروع با یک دایره توپر ســـیاه در روی نمودار نشان داده شده است و نشان می‌دهد چه حالتی از آبجکت در ابتدا ایجاد شده است. حالت پایانی بو سیله یک خال هدف نمایش داده شده است و نشـــان می‌دهد که آبجکت درســـت قبل از اینکه از بین برود، در چه حالتی

می‌باشـــد. بر روی یک نمودار حالت، فقط و فقط یک حالت شـــروع وجود دارد، در حالی که شما می‌توانید حالت پایانی نداشته باشید یا اینکه هرچند حالت پایانی که نیاز دارید را داشته باشید.

ممکن اســـت زمانی که آبجکت داخل یک حالت ویژه اســـت، چیزهای مشخصی اتفاق بیفتد. در مثال ما وقتی که از یک حساب، زیادی برداشت می‌شود، یک اخطار به مشتری فرستاده می‌شود. پردازشهایی که در حالت مشخصی از آبجکت اتفاق می‌افتند، Action نامیده می‌شوند.

نمودارهای حالت برای هر کلاســـی ایجاد نمی‌شـــوند. آنها فقط برای کلاس‌های پیچیده استفاده می‌شوند. اگر آبجکتی از یک کلاس می‌تواند در چند حالت وجود داشـــته باشـــد و در هر حالت خیلی متفاوت رفتار نماید، ممکن است بخواهد یک نمودار حالت برای آن ایجاد کنید.

بسیاری از پروژه‌ها اصلاً به این نمودارها نیازی ندارند. اگر آنها ایجاد شـــده‌اند، برنامه‌نویســـان از آنها در زمان تولید کلاس‌ها اســـتفاده می‌کنند. نمودارهای حالت فقط برای مستندسازی ایجاد شده‌اند.

وقتی شـــما از روی مدل Rose خود کدی را ایجاد می‌کنید، کد از روی اطلاعات روی نمودارهای حالت ایجاد نخواهد شد. اگر چه add-ins در Rose برای سیستم‌های بلادرنگ (real time) وجود دارد، که می‌تواند کد قابل اجرا را بر پایه نمودارهای حالت تولید نماید.

مدلسازی بصری و پردازش تولید و توسعه نرم‌افزار

تولید نرم‌افزار می‌تواند به چندین روش انجام شـــود. چندین نوع متفاوت از پردازش‌های تولید شـــامل هر چیزی از پردازش‌های آبشـــاری گرفته تا شی‌گرایی وجود دارد که پروژه‌ها آنها را دنبال می‌کنند.

هر یک مزایا و امتیازات خودش را دارد. در این بخش، ما قصـــد نداریم که به شـــما بگوییم تا از کدام یک اســـتفاده کنید. اما مروری بر یک پرازش را آماده کرده‌ایم که بر روی مدلسازی بصری متمرکز می شود. مجدداً می‌گوییم که این فقط یک نظر اجمالی (مرور) است.

مدت طولانی برنامه‌نویسان نرم‌افزار از مدل آبشاری استفاده می‌کردند. در این مدل ما درخواست‌ها را تجزیه و تحلیل می‌کردیم. یک سیستم را تولید می‌کردیم، سیستم را آزمایش می‌کردیم و سیستم را نصب می‌نمودیم.

همان‌طور که از نامش پیداست، ما نمی‌توانیم برخلاف این جریان حرکت کنیم. این متد یک متدلوژی مستند شده می‌باشد که در هزاران پروژه استفاده شده است.

یکی از کمبودهای مدل آبشاری این است که باید از وسط مراحل به خارج از مجموعه یک پروژه که مدل آبشاری را دنبال می‌کند، بازگرد (Backtrack) نمایید. ما با تعیین همه درخواست‌های سیستم، کار هراس‌انگیزی را هنگام شروع پروژه انجام می‌دادیم. ما این کار را از طریق گفتگوهای مفصل با کاربران و رسیدگی‌های مفصل پردازش‌های تجاری انجام می‌دادیم. اما اگر واقعاً خوش‌شانس باشیم، ممکن است در حدود

٪۸۰ درخواست‌های سیستم را در طی مرحله تجزیه و تحلیل بدست آوریم.

سپس زمان طراحی ا ست. می‌ن شینیم و سبک معماری خود را تعیین می‌کنیم. ما موضوعاتی را نشان می‌دهیم، مانند جایی که برنامه‌ها مقیم خواهند شد و اینکه چه سخت‌افزاری برای عملکرد قابل قبول، ضروری است. زمانی که این کار را انجام می‌دهید، ممکن است تعداد موضوعات جد ید که بوجود آ مده‌ا ند، را پ یدا کنیم. سپس به س مت کاربران برمی‌گردیم و درباره موضوعات صحبت می‌کنیم. این نتیجه در درخواست‌های جدید قرار دارد. بنابراین ما به تجزیه و تحلیل برمی‌گردیم. بعد از برگشتن و از آن زمان به بعد، ما به فاز برنامه‌نویسی رفته و کدنویسی سیستم را شروع می‌کنیم.

در زمان کدنویسی، می‌فهمیدیم که به انجام رساندن یک تصمیم طراحی شده مشخص غیرممکن است. بنابراین به طراحی برگشته و

دوباره با موضـــوع روبه‌رو می‌شـــویم. بعد از اینکه کدنویســـی انجام شـــد، آزمایش‌کردن شروع می‌شود.

در زمان آزمایش، ما یاد می‌گیریم که یک درخوا ست به اندازه کافی تشریح نشده و تفسیر اشتباه بوده است. حالا باید به مرحله تجزیه و تحلیل برگردیم و دوباره با درخواست روبه‌رو شویم.

بعد از مدت زمانی، ما ســـرانجام ســـیســـتم انجام شـــده را بدســـت می‌آوریم و به کاربران تحویل می‌دهیم. از آنجایی که این کار مدت زمانی طول می‌کشـد و در زمانی که ما در حال سـاخت سـیسـتم هسـتیم، احتمالاً تجارب تغییر کرده اسـت، کاربران با اشـتیاق کمتری نسـبت به جمله «این فقط آن چیزی ا ست که من تقا ضا کردم، نه آنچه که واقعاً دو ست دا شتم، واکنش می‌دهد. این جمله کاربران، جادوی قدرتمندی ا ست که تیم پروژه را ناامید می‌نماید.

بنابراین، بعد از نگاه به این ســـناریوی غم‌انگیز و شـــگفت‌زده، اگر در صـــنعتی به ســـامان و منظم قرار دارید، چه کاری برای بهبود آن انجام

می‌دهید؟ با مشکل تغییرات سریع تجارت چه می‌کنید؟ آیا کاربران با آنچه که می‌خواهند ارتباط برقرار نمی‌کنند؟ آیا کاربران گروه پروژه را درک نمی‌کنند؟ آیا گرده یک پردازش را دنبال نکرد؟ جواب‌ها بله، بله، بله، نه می‌باشد.

تجارت به سرعت تغییر می‌کند و همانند نرم‌افزار حرفه‌ای باید با آن ارتقاء یابیم. همیشه کاربران با آنچه می‌خواهند ارتباط برقرار نمی‌کنند، زیرا آنچه آنها انجام می‌دهند، عادت همیشگی آنهاست. پرسیدن از یک کارمند حسابداری که حدود ۳۰ سال سر این کار بوده است، مانند این است که از شخصی بپرسیم که چگونه نفس می‌کشد. این عادت همیشگی می‌شود و توضیح دادن در مورد آن مشکل است.

مشکل دیگر این است که کاربران همیشه گروه پروژه را درک نمی‌کنند. گروه به آنها فلوچارت را می‌دهد و نسخه هایی از متن درخواست‌ها را تولید می‌کند، اما همیشه کاربران نمی‌فهمند که چه چیزی به آنها داده شده است. آیا می‌توانید به راه حل این مشکل فکر کنید؟ در

این راه مدلسازی بصری می‌تواند به ما کمک کند. اخیراً گروه، یک پردازش را دنبال نمود: متد آبشاری (waterfall). متاسفانه، طرح و اجرای متد دو چیزی متفاوت بودند.

بنابراین یکی از مشکلات این است که تیم برنامه‌ییزی کرده تا از متد آبشاری با پیغام منظم و مرتب در نواحی پروژه استفاده نماید. اما آنها باید در سراسر پروژه به عقب برگردند.

آیا مناسب است که برنامه‌ریزی ناچیزی صورت گیرد؟ احتمالاً نه. تولید نرم‌افزار یک پردازش پیچیده است و تلاش‌کردن برای انجام هر چیزی در نواحی منظم همیشه کار نمی‌کند. اگر نیاز به backtacking (دوباره برگشتن) نادیده گرفته شود.

سیستم دارای خطاهای طراحی خواهد بود و درخواست‌ها را از دست می‌دهد و احتمالاً بدتر می باشد. اما ما سال ها آموخته‌ایم که backtacking را طراحی نماییم. با این بینش به تولید مکرر (Iterative Developmant) می‌رسیم. تولید مکرر فقط بدین معنی است که ما قصد

داریم کارها را بارها و بارها انجام دهیم. در پردازش شـــی‌گراءِ، ما در سـراسـر مراحل تجزیه و تحلیل، طراحی، تولید، تسـت و تولید در مقاطع کوچک، بارها حرکت خواهیم نمود.

این غیر ممکن ا ست که همه درخوا ستها را در طول بخش نخ ست پروژه بفهمیم. چیزهای جدیدی ظاهر می‌شـوند. بنابراین با طراحی پروژه به صـــورت تکراری آنها را برنامه‌ریزی می‌کنیم. با این مفهوم، یک پروژه می‌تواند به عنوان یک سری از آبشارهای کوچک دیده شود. هرکدام که طراحی شـــده‌اند، به اندازه کافی بزرگ هسـتند تا پایان یک بخش مهم پروژه را علامت گذاری نمایند، در حالی که به اندازه کافی کوچک نیز می‌باشند تا نیاز backtacking را به حداقل برسانند.

شناخت Inception

فاز Inception شـروع پروژه اسـت.Inception زمانی شـروع می شود که شخصی بگوید:«خدایا اگر سیتمی داشتیم که این کار را انجام دهد چقدر خوب می شد».

سـپس شـخصـی در مورد این ایده تحقیق می کند اینکه مدیریت تقاضـاها چه قدر طول می کشـدو چه میزان ارزش دارند یا عملی شـدن پروژه امکان پذیر اسـت رابررسـی می کند. فازInception مخصـوص پیداکردن جوابهای این پر س شها ست.ما ک شف می کنیم که ا شکالت سطح بالای سیستم چه هستند وآنها را مستند سازی می نماییم. ما کشف می‌نیم که عاملهای سیستم چه کسانی هستند و use case را تعیین می نماییم. ما دراینجا وارد جزئیاتuse caseها نمی شـویم اما فقط یک یا دوجمله را آماده می کنیم.

همچنین تخمینی را فراهم می کنیم تا مدیریت را پیش ببریم. بنابراین با اســـتفاده ازRose برای پشـــتیبانی از پروژه های خود، عامل ها و use case را ایجاد خواهیم کرد ونمودار هایuse case را تولید خواهیم نمود.زمانی پایان می یابد که تحقیقات انجام شـــده اند و مدیریت؛ منابع را اختصاص می دهد تا بر روی پروژه کار کنند.

فازInception پروژه به طور اساسی دنباله دار وغیر تکراری است. حالتهای دیگر چندین بار در طول پروژه تکرار می شـــوند. به دلیل اینکه در حقیقت پروژه فقط یک بار می تواند شـــروع شـــود؛Inception نیز فقط یکبار روی پروژه انجام شـــده اســـت .به همین دلیل، بیش از یک وظیفه در می‌ماند. تولید یک طرح تکراری. یک طرح تکراری طرحی است که توضیح می دهد چهuse case در طول تکرار انجام می شـــود.اگر ما در طول تکرار ده تا را پیدا کنیم، ممکن است یک نقشه تکراری مانند این را بکشیم.

Iteration One	Use Cases 1.5.6
Iteration Two	Use Cases 7.9

Interation Three	Use Cases 2.4.8
Interation Four	Use cases 3.10

طرح به ما می گوید که کدام use case اول انجام شـــده اســـت. مشـــخص کردن این طرح نیازمند این اســـت که به وابســـتگیهای بین use case نگاه کنیم و بر اســـاس آن برنامه ریزی نماییم. اگر برای کارکردن use case ۵ نیاز به use case3 دا شته با شد، پس طرح تو ضیح داده شده بالا شدنی و امکان پذیر نیست، زیرا use case3باید در دور٤ اجرا شـــده باشـــد، در حالی کهuse case5در اولین دوره قرار دارد. ما باید طرحمان را تنظیم کنیم تا با وابستگیها وفق داده شود.

مهارت Elaboration

فاز مهارتElaboration پروژه شـــامل مقداری طراحی، تجزیه و تحلیل و طرح معماری اســـت. همراه با طرح تکرار، فاز مهارت برای هر use case در تکرار جاری انجام می شود. فاز مهارت شامل چندین جنبه

از یک پروژه اســـت. مانند کد کردن، اثبات مفاهیم،تولید نمونه های آزمایشی و ایجاد تصمیمات طراحی. از کارهای اصلی فازElaboration تکمیلuse case است.

در خواستهای سطح پایین یکuse case شامل جریان پردازش در طولuse case می باشد. چه عامل هایی باuse case درخواست شده اند، و نمودارهای Interaction جریان پردازش را به صـــورت گرافیکی نشان می دهد و کلا هر حالتی که تغییر می کند، ممکن است در زمان اتفاق بیافتد.

در خواستها، به شکلuse case کامل و با جزئیات، در یک سند جمع شـده اند که یکSRS (مشـخصـات درخواسـت نرم افزار) نامیده شـده است. SRS شامل همه جزئیات درخواستهای سیستم می باشد.

کارهای دیگری در فاز مهارت انجام می شـودمانند اصـلاح تخمینهای اولیه، بررسی کیفیت، و مدل و بررسی کردن خطرها.

Rational Rose می تواند به مدلuse case کمک کند و نمودارهای Sequenceوcollaboration را ایجاد کند تا جریان گرافیکی پردازش رانشان دهد.

نمودارهای، آبجکت هایی که ساخته شده اند یا در طول فاز مهارت طراحی شده اند را نشان می دهد. فاز مهارت زمانی تمام شده است که کاملا وارد جزئیات شده اند وبوسیله کاربران پذیرفته شده اند، اثبات مفاهیم کامل شده اند تا شدت خطرها را کاهش دهند ونمودارهای کامل می باشند. به عبارت دیگر این فاز زمانی کامل است که سیستم طراحی شده، بازبینی شده وآماده است تا برنامه نویسان آن را تولید نمایند.

ساختار Construction

(ساختار) به روند تولید و تست نرم افزار بر می گردد. مانند این فاز برای هر مجموعه از،در یک بار تکرار کامل شده است. کارهای فاز شامل

مشـــخص کردن در خواســـتهای ثابت، تولید نرم افزار و تســـت نرم‌افزار می‌باشـــد. ازآنجایی که نرم افزار در طول فاز به طور کامل طراحی شـــده است، نباید درگیر تصمیمهای طراحی زیاد باشد. این به گروه پروژه کمک می‌کند تا تولید موازی را به انجام رسانند.

تولید موازی به این معنی است که چند برنامه نویس بتوانند بر روی آبجکتهای مختلف نرم افزاری کار کنند و بدانند که کل ســـیتم با هم جمع خواهند شـــد. در، ما آبجکتهایی را در ســـیســـتم طراحی کردیم و اینکه آنها چگونه بر روی هم اثر می گذارند. فقط موضـــوع گذاشـــتن آن طرح در اســـت، تا تصـــمیمات طراحی جدید که می توانند آن فعل وانفعالات را تغییر بدهند ایجاد شوند.

مزیت دیگر مدل کردن ســـیســـتم این اســـت که می تواند کد اولیه را برای سیستم تولید کند. به منظور استفاده از این شکل، شما نیاز دارید تا و یک نمودار را به عنوان یک بخش اولیه ایجاد کنید.

اولین باری که ایجاد می شوند و ارتباطات آنها رسم می گردد، تولید کردن کد می تواند شـــروع شـــود.تولید کد، کدی را برپایه طراحی فراهم خواهد کرد. این کد به این معنی نیست که شما هر کد مخصوص تجاری خارج از بدست خواهید آورد.

آنچه که شما به دست خواهید آورد خیلی وابسته به زبانی است که انتخاب شده است، اما عموما شامل تعریف کلاس، تعریف صفات، تعریف محدوده (خصوصی عمومی حفاظت شده)، نوع توابع ودستورات وراثتی می‌باشـــد. این کار در وقت صـــرفه جویی می کند زیرا این کدی خســـته کننده‌ای برای نوشتن است.بعد از تولید کد، برنامه نویسان می‌توانند بر روی جنبه‌های ویژه تجاری متمرکز شوند.

بعد ازاینکه کد تکمیل گردید، باید توسط یک گروه هم تراز از برنامه نویسان بازبینی شود تا مطمئن شوند که استانداردها و نشستهای طراحی وتوابع رعایت شـــده اســـت. بعد از اینکه کدبازبینی شـــد، آبجکت ها باید به بازبینی تضـــمین کیفیت داده شـــوند. اگر صـــفات جدید یا توابعی به اضـــافه

شـــده اند یا اگر هر فعل وانفعالی بین آبجکتها تغییر کرده اســـت، پس کد جدید باید در مدل از طریق مهندسی معکوس به روز رسانی شده باشد.ما این موضــوع را به طور مفصــل در فصــلهای ۱۹ تا ۲٤ شــرح خواهیم داد. زمانی به پایان می رسد که این نرم افزار کامل شده وتست گردیده است. این مهم است که مطمئن شوید مدل ونرم افزار همزمان (سنکرون) شده اند. این مدل زمانی که نرم افزار، مد پشــتیبانی را وارد می کند به شــدت ارزشمند خواهد بود.

انتقال Transition

فاز زمانی است که محصول نرم افزاری کامل شده، به سمت اجتماع کاربربرمی گردد. کارها در این فاز شامل کارکردن مح صول نرم افزاری نهایی، تکمیل ت ست تاییدد نهایی، کامل کردن م ستند سازی کاربر و فراهم کردن آموزش برای کاربر می باشـــند. باید مشـــخصـــات درخواســـتهای

نرم‌افزار، نمودارهای، نمودارهای، نمودارهای و نمودارهای بروز رسانی شده باشند تا تغییرات نهایی را منعکس کنند.

مهم است که این مدلها با محصول نرم افزاری همزمان شده باشند زیرا مدلهایی که یکبار در محصول نرم افزاری استفاده خواهند شد به مد پشتیبانی می‌روند. چند ماه بعد از اتمام پروژه، این مدلها درکمک به ارتقا نرم‌افزار، گرانبهاتر خواهند بود.

در فاز مانند دیگر فاز ها مفید نیست. در این نقطه، محصول نرم‌افزاری تولید شده است. طراحی شده است تا به مدل سازی و تولید نرم افزار کمک کند و حتی به طرح ارتقا نرم افزار کمک می کند. اگر چه به عنوان یک ابزار آزمایشی یا کمک به طرح های آزمایشی یا توابع گسترشی طراحی نشده بود.

اما ابزار دیگری وجود دارد که مخصوصا برای این اهداف طراحی شده اند. بنابراین اصولا، در فاز استفاده خواهد شد تا مدلها را بعنوان محصول نرم افزاری تکمیل شده بروز رسانی نماید.

Rational Rose چیست؟

Rational Rose یک ابزار قدرتمند اســـت که به تجزیه و تحلیل و طراحی ســیســـتم‌های نرم‌افزاری شـــی‌گرا کمک می‌کند. این برنامه کمک می‌کند تا قبل از اینکه کدی را بنویسیم، سیستم خود را مدل نماییم. بنابراین می توان مطمئن شد که سیستم از ابتدا معماری معتبری دارد. با استفاده از مدل، می‌توان به زودی جریان‌های طراحی را بدســـت گرفت. در حالی که آنها هنوز برای آماده کردن ارزان و کم‌هزینه می‌باشند.

Rational Rose با کمک تجزیه و تحلیل و طراحی ســیســـتم‌ها ما را قادر می سازد تا use caseها و نمودارهای use case را برای نشان دادن اینکه آبجکت‌ها چگونه با هم کار می‌کنند، تا عملیات مورد نیاز را فراهم کنند، طراحی نمایید. کلاس ها و نمودار های Class ایجاد شـــده‌اند تا آبجکت‌های یک سیستم را نشان دهند و اینکه چگونه آنها با یکدیگر رابطه دارند. نمودارهای Component می‌توانند ایجاد شـوند تا نشـان دهند که

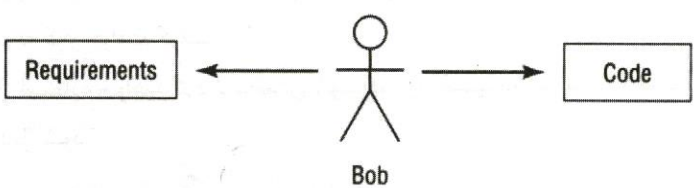
چگونه کلاس‌ها به اجزای ا سبابی و ابزاری نگا شت می شوند. در نهایت، یک نمودار Deployment می‌تواند تولید شـود تا طرح شـبکه‌ای سـیسـتم را نشان دهد.

مدل Rose تصـویری از یک سـیسـتم اسـت. این مدل شـامل همه نمودارهـای UML، عـامـل‌هـا، use caseهـا، آ بجکـت‌هـا، کلاس‌هـا، componentها و گره‌های deployment یک سیستم می‌باشد.

این مدل با جزئیات بی شتری تو ضیح می‌دهد که چه سی ستمی ایجاد خواهد شـد و چگونه کار خواهد کرد. بنابراین برنامه‌نویسـان می‌توانند از مدل به عنوان یک طرح کلی برای سـاخت سـیسـتم اسـتفاده کنند. این به کاهش یک مشکل قدیمی کمک می‌کند. گروه با مشتریان صحبت کرده‌اند و درخواست‌ها را مستندسازی نموده‌اند. حال برنامه‌نویسان آماده هستند تا کدنویسی را شروع کنند.

یکی از برنامه‌نوی سان تعدادی از درخوا ست‌ها را می‌گیرد، ت صمیمات طراحی کاملاً متفاوتی را ایجاد می‌کند و تعداد کد بیشتری را می‌نویسد.

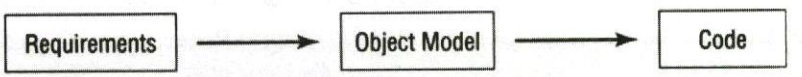
این تفاوت در سبک برنامه‌نویسی کاملاً طبیعی است. به ۲۰ برنامه‌نویس درخواست‌های یکسانی داده شده که ممکن است ۲۰ سیستم مختلف را کد کنند. مشکل زمانی پیش می‌آید که شخصی نیاز دارد تا سیستمی را بفهمد یا نگهداری کند. بدون اجرای مصاحبه‌های مفصل با هر یک از برنامه‌نویسان، برای هر شخص مشکل است تا ببیند که چه تصمیمات طراحی اتخاذ شده، قطعات سیستم چه هستند و یا ساختار کلی سیستم چیست؟ بدون یک طرح مستند شده، مشکل است مطمئن شوید که آیا سیستمی که ساختید، واقعاً همان سیستمی است که کاربران در ذهن داشتند. به طور قراردادی، ما پردازشی که شبیه زیر است را دنبال می‌کنیم:



درخواست‌ها مستندسازی شده‌اند، اما طرح در سر Bob است، بنابراین کس دیگری به غیر از Bob یک دید خوب نسبت به ساختمان

سیستم ندارد. اگر Bob برود، اطلاعات نیز با او می‌رود، اگر شما تا به حال یکی از کارهای Bob را گرفته با شید، می‌توانید بفهمید که چقدر درک یک سیستم با مستندات کم مشکل می‌باشد.

یک مدل Rose به ما پردازشی می‌دهد که شبیه زیر است:



حال طرح مستندسازی شده است. همه برنامه‌نویسان می‌توانند دور هم جمع شوند تا قبل از آنکه کد نوشته شود، درباره تصمیمات طرح بحث کنند. نباید نگران بود که هر کسی با طرح سیستم به یک جهت جدا می‌رود، اما فقط برنامه‌نویسان از مدل استفاده نمی‌کنند.

- مشتریان و مدیران پروژه از نمودارهای use case استفاده خواهند کرد تا دید سطح بالایی را نسبت به سیستم بدست آورند و بر روی محدوده پروژه موافقت کنند.

- مدیران پروژه از نمودارهای use case و م ستندات ا ستفاده خواهند کرد تا پروژه را به تکه‌های قابل مدیریت بشکنند.
- تحلیلگران و م شتریان به م ستندات use case نگاه خواهند کد تا ببینند که سیستم چه عملیاتی را فراهم خواهد نمود.
- نویسندگان تکنیکی به مستندات use case نگاه خواهند کرد تا شروع به نوشتن طرح‌های آموزشی و دستی کاربران کنند.
- تحلیلگران و برنامه‌نویسان به نمودارهای Collaboration, Sequence نگاه خواهند کرد تا ببینند که چگونه منطق سیستم آبجکت‌های سیستم، پیغام‌های بین آبجکت‌ها جریان خواهند داشت.
- کارمندان تضمین کیفیت از اسناد use case و نمودارهای Sequence, Collaboration استفاده خواهند کرد تا اطلاعاتی را بدست آورند که برای تست Script نیاز دارد.

- برنامه‌نویسان از نمودارهای Class و نمودارهای State Transition (حالت) استفاده خواهند کرد تا دید جزئی نسبت به قطعات سیستم و چگونگی ارتباط آنها را بدست آورند.
- کارمندان Deployment از نمودارهای Component, Deployment استفاده خواهند کرد تا ببینند که چه فایل‌های اجرایی، فایل‌های DLL یا Componentهای دیگری ایجاد خواهد شد و این Component در کجا بر روی شبکه قرار خواهد گرفت.
- کل تیم از مدل استفاده خواهند کرد تا مطمئن شوند که درخواست‌ها به کد تبدیل شده‌اند و آن کد می‌تواند به درخواست‌ها تبدیل شود.

بنابراین، Rose ابزاری است که به وسیله کل تیم پروژه استفاده می‌شود. منبع فرصت و آزادی عمل و اطلاعات طراحی است که هر عضو تیم می‌تواند استفاده کنند تا اطلاعاتی که نیاز دارد را بدست آورد.

علاوه بر موارد بالا، Rational Rose با تولید کد اولیه به برنامه‌نویسان کمک خواهد کرد و می‌تواند این کار را برای تعدادی از

زبان های متفاوت موجود در بازار شامل PowerBuilder, Visual Basic, Java, C++ انجام دهد. علاوه با آن، Rose می‌تواند مهندسی معکوس را بر روی کد اعمال کرده و مدلی بر پایه یک سیستم موجود ایجاد کند. داشتن یک مدل در Rose برای یک برنامه کاربردی موجود بسیار سودمند است.

وقتی که تغییری در کد ایجاد می‌شود، می‌توان آن تغییر را در مدل به صورت خودکار ثبت نمود این ویژگی‌ها کمک می‌نماید تا مدل را حفظ کرده و کد را هماهنگ نمود و خطر داشتن یک مدل قدیمی را کاهش داد.

همچنین Rose می‌تواند با استفاده از Rose Script (یک زبان برنامه‌نویسی است که با Rose بسته‌بندی شده است) گسترش داده شود. می‌توان با استفاده از این زبان برنامه‌نویسی، کدی نوشت که به طور خودکار تغییراتی را در مدل ایجاد نماید، یک گزارش ایجاد کند یا کارهای دیگری را با مدل Rose خود اجرا کند.

در حال حاضر سه نسخه متفاوت از Rose در دسترس است:

- Rose Modeler که اجازه می‌دهد تا مدلی را برای سیستم ایجاد کرد، اما از تولید و مهندسی معکوس کد پشتیبانی نمی‌کند.
- Rose Professional ، که اجازه می‌دهد تا کدی را در یک زبان ایجاد کرد.
- Rose Enterprise که اجازه می‌دهد تا کدی را در Visual Basic, Java, C++ ایجاد کرد. مانند نمودار Oracle 8

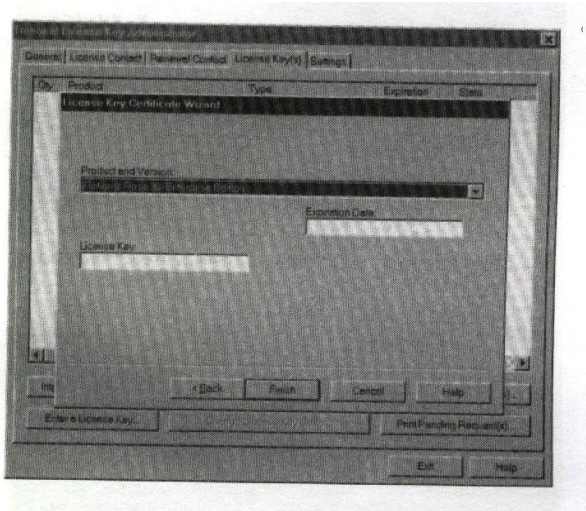
علاوه بر این، Rose 98i، ارتقاءِ (بهبودی) جدید Rose بر روی یکپارچه کردن Rose با دیگر ابزارها مانند Visual C++, Team Test, Rational Requisite Pro و غیره متمرکز کرد. همچنین Rose 98i اجازه می‌دهد تا مدل خود را روی وب منتشر کرد. Rose 98i مانند Rose 98 در نسخه‌های Enterprise, Pro, Modeler قابل دسترسی هستند.

پرداختن به Rational Rose

Rose عمدتاً یک برنامه منویی است و یا از نوارهای ابزاری که به شکل استفاده شده عمومی کمک می‌کند، استفاده می‌نماید. Rose از ۷ نوع

متفاوت از نمودارهای UML پشـــتیبانی می‌کند. نمودارهای Sequence, Use, Case, Deployment, Component, State Transitien, Class, Collaboration.

Rose برای هر یک از این نمودارها یک نوار ابزار متفاوت را نمایش خواهد داد. علاوه بر نوارهای ابزار و منوها، Rose شـــامل منوهای میان‌بر حساس به متن است که با راست کلیک بر روی یک آیتم ظاهر می شوند. مثلاً با راست کلیک بر روی یک کلاس از نمودار Class، منویی نمایش داده خواهد شد که شامل انتخاب‌هایی برای اضافه‌کردن صفات یا عملگرهایی به کلاس، مشاهده یا ویرایش کردن مشخصات کلاس،تولید کد برای کلاس، یا مشاهده کد تولید شده می‌باشد. یکی از آسان‌ترین راه‌ها برای پرداختن به Rose اســتفاده از مرورگر اســت. با مرورگر می‌توان به ســرعت و به آسانی نمودارهای و عناصر دیگری از مدل را به دست آورد.



بخش‌های صفحه نمایش

پنج قسمت مهم واسط کاربر Rose عبارتند از: مرورگر، پنجره مستندسازی، نوار ابزار، پنجره نمودار، و Log.

- مرورگر: برای این ا ستفاده می شود تا بتوان به سرعت در سرا سر مدل حرکت نمود.
- پنجره مستندسازی: برای دستیابی به مستندات عناصر مدل استفاده می‌شود.
- نوار ابزار: برای دستیابی سریع به فرمان‌هایی که عموماً استفاده می‌شوند.

- پنجره نمودار: برای نمایش و ویرایش یک یا چند نمودار UML اســتفاده می‌شود.
- Log: برای دیدن خطا و گزارش فرمان‌های مختلف استفاده می‌شود.

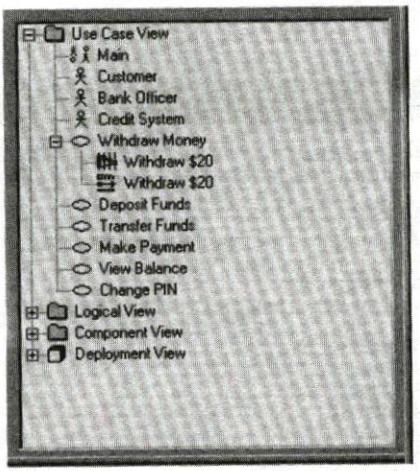
چهار نمای موجود در یک مدل Rose

چهارنما در یک مدل Rose وجود دارد. نمای Use Case, Logical, Component, Deployment. هر یک از این چهارنما یک هدف و مجوز متفاوت را عنوان می‌کند.

Use Case

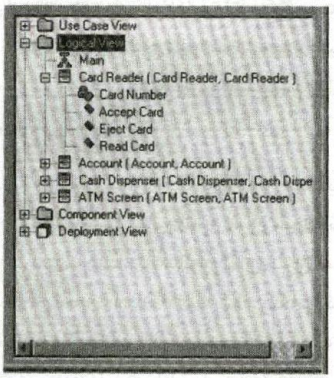
نمای Use Case شامل همه عامل‌ها، use caseها و نمودارهای use case ســیســتم می‌باشــد و ممک اســت شــامل تعدادی نمودارهای Collaboration, sequence باشــد. نمای Use Case یک نگاه وابســته به پیاده سازی در سیستم ا ست. این نما بر روی تصویری سطح بالا متمرکز می شود که سیستم چه کاری انجام خواهد داد، بدون اینکه نگران جزئیات چگونگی کار در سیستم باشد.

شکل زیر نمای Use Case را در مرورگر Rose نشان می‌دهد.



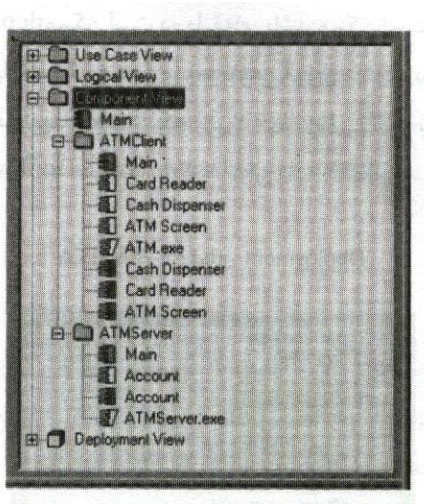
نمای منطقی

نمای منطقی بر روی این متمرکز می‌شود که چگونه سیستم، رفتار را در use caseها پیاده سازی کند. این نما تصویر مفصلی از قطعات سیستم را فراهم می‌کند و چگونگی ارتباط قطعات را شـــرح می‌دهد. نمای منطقی شامل، چیزهای میانی دیگر، کلاس‌های مشخصی که ضرورت پیدا خواهند کرد، نمودار های Class و نمودار های State Transition می باشـــد. برنامه‌نویسان می‌توانند با این عناصر مفصل، طرح مفصلی را برای سیستم بسازند.



نمای Component

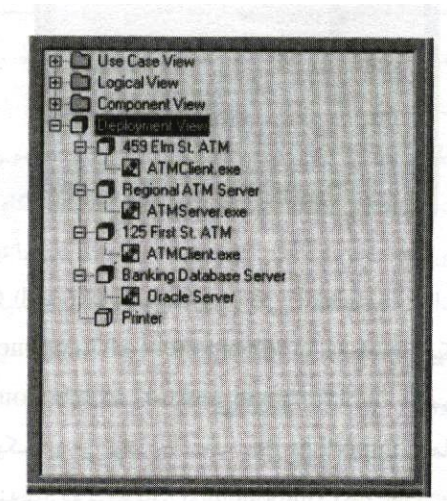
نمای فوق شـــامل اطلاعاتی درباره کتابخانه‌های کد، فایل‌های قابل اجرا، کتابخانه‌های زمان اجرا و Componentهای دیگر مدل شما می‌باشد. یک Component معیار فیزیکی از کد اســت. در Rose، Componentها و نمودارهای فوق در نمای Component نمایش داده شـــده‌اند. نمای فوق سیستم، اجازه می‌دهد تا ارتباطات بین ماژول‌های کد را دید.



نمای Deployment

نمای نهایی در Rose، نمای Deployment، مربوط به گسترش فیزیکی سیستم است که ممکن است با سبک معماری منطقی سیستم متفاوت باشد. مثلاً ممکن است سیستم یک سبک معماری سه طبقه‌ای منطقی داشته باشد. به عبارت دیگر، ممکن است واسط کاربر از منطق تجاری (موضوعی) جدا شده باشد، که از منطق پایگاه داده جدا شده است. اگرچه، Deployment ممکن ا ست دو طبقه‌ای با شد، ممکن ا ست وا سط کاربر بر روی یک ما شین قرار گرفته با شد، در حالی که منطق پایگاه داده و تجاری (موضوعی) روی ماشین دیگری واقع شده‌اند. همچنین موضوعات

دیگر همانند تحمل خطا، پهنای باند شبکه، بهبود خطا، زمان پاسخ، با استفاده از نمای فوق قابل دسترسی است.



کار با برنامه Rational Rose

هرکاری که در Rose انجام گردد، وابسته به یک مدل است.

ایجاد مدل‌ها

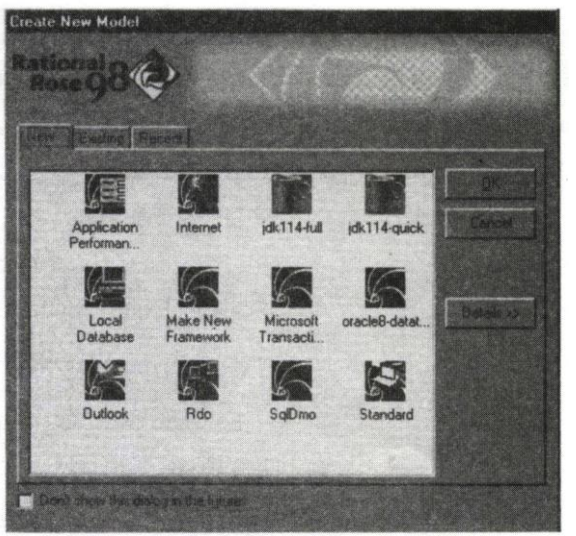
اولین مرحله در کار با Rose، ایجاد یک مدل است. مدل‌ها می‌توانند از طریق حافظه موقتی و یا اســتفاده از مدل چارچوب کاری موجود، ایجاد

شده با شد. یک مدل Rose، شامل همه نوارها، آبجکت‌ها و دیگر عنا صر مدل است که در یک فایل جداگانه با پسوند MDL، (مدل) ذخیره شده‌اند.

برای ایجاد یک مدل:

۱. از منوی فایل، گزینه NEW را انتخاب می‌کنیم.

۲. اگر Framework Wizard نصب شده باشد، لیستی از چارچوب‌های کاری قابل دسترس نمایش داده خواهد شد. چارچوب کاری را که می‌خواهید ا ستفاده کنید، انتخاب کرده و بر روی OK کلیک کرده یا برای استفاده نکردن از چارچوب کار، بر روی Cancel کلیک می‌نماییم.



واردکردن و ارسال مدل‌ها

یکی از امتیازات اصلی الگوی شی‌گرا، reuse (استفاده مجدد) است. Rose نه تنها در کدها بکار برده می‌شود، بلکه در مدل‌ها نیز به خوبی بکار برده می شود. برای گرفتن امتیاز بیشتر از Reuse، Rose از واردکردن و ارسال مدل‌ها و عناصر مدل پشتیبانی می‌کند. می‌توان یک مدل یا بخشی از یک مدل را ارسال کرده و آن را وارد مدل‌های دیگر نمود.

برای ارسال یک مدل:

۱. از منوی فایل گزینه Export Model را انتخاب می‌کنیم.

۲. نام فایل ارسال شده را وارد می‌کنیم.

برای ارسال یک بسته از کلاس‌ها

۱. بسته‌ای را انتخاب کرده تا آن را از یک نمودار Class ارسال می‌کنیم.

۲. از منوی فایل، گرینه Export>Package را انتخاب می‌کنیم.

۳. نام فایل خارج شده را وارد می‌کنیم.

برای ارسال یک کلاس

۱. کلاس را از نمودار Class برای ارسال انتخاب می‌نماییم.

۲. از منوی فایل، گزینه Export<Class را انتخاب می‌کنیم.

۳. نام فایل ارسالی را وارد می‌کنیم.

برای واردکردن یک مدل، بسته یا کلاس:

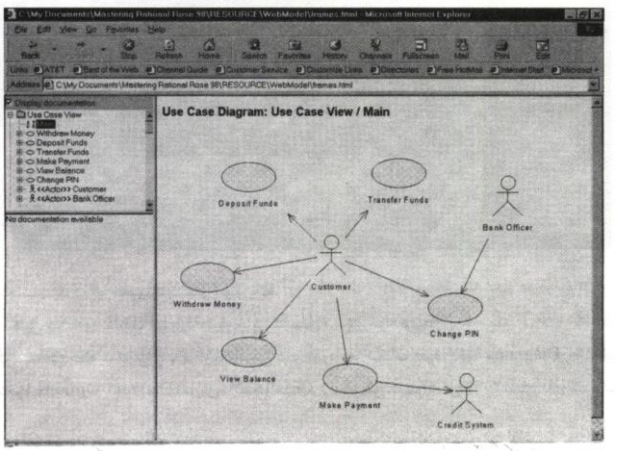
۱. از منوی فایل، گزینه Import Model را انتخاب می‌کنیم.

۲. برای واردکردن، فایل را انتخاب می‌کنیم. انواع فایل‌های قابل قبول categroy(CAT), petal(PIL), model(MDL), subsystem(SUB) هستند.

انتشار مدل‌ها بر روی وب

به آسانی می‌توان مدل Rose را روی وب منتشر نمود. بدین روش با استفاده Rose ممکن است بسیاری از افراد مدل شما را ببینند، بدون اینکه جزء کاربران Rose باشند و بدون اینکه بسیاری از مستندات مدل را چاپ

کنند. یک مدل منتشر شده بر روی وب در شکل زیر نمایش داده شده است.

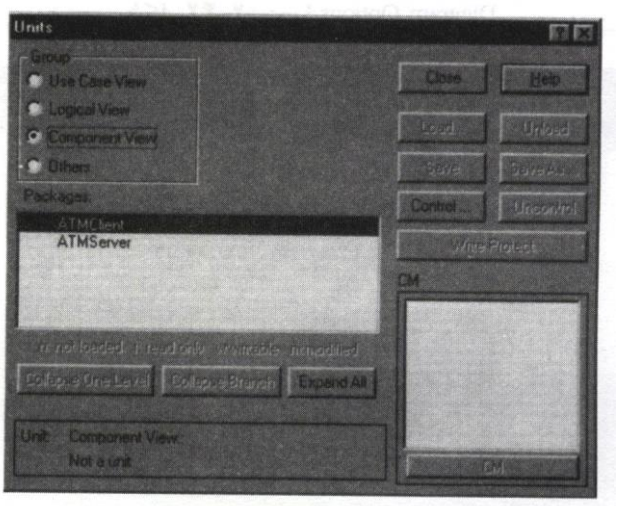


کار با واحدهای کنترل شده

Rose از چندکاربری پشتیبانی می‌کند. یک واحد کنترل شدهدر Rose می‌تواند هربسته‌ای در داخل نمای Use Case،نمای Logical یا نمای Component باشد. واحدهای نمای Model Properties, Deployment می‌توانند تحت کنترل قرار گیرند. وقتی که یک واحد کنترل شده است، در یک فایل جدا از بقیه مدل ذخیره می‌شود. بدین روش، فایل مجزا را می‌توان با استفاده از یک نسخه فرمانبردار SCC ابزاری مانند Rational

Clear Case Microsoft Source Safe با حداقل Rose به طور مستقیم کنترل کرد.

واحدهای کنترل شده می‌توانند از طریق مدل دیده شده بارگذاری یا حذف شوند، یا اگر از یک نسخه کنترل ابزار استفاده می‌شود. آنها را انتخاب نمود یا از انتخاب خارج کرد. در Rose واحد ها با استفاده از پنجره‌ای که در شکل زیر نشان داده شده است، مدیریت می شوند. در Rose واحدها از طریق منوی میانبری مدیریت می‌شوند که از طریق کلیک راست بر روی یک بسته مرورگر، قابل دسترسی هستند.



عامل شامل تمام آن چیزهایی ا ست که خارج از سی ستم قرار دارد. بحث را با آموزش ساخت نمودارهای شروع می کنیم. سپس را به نمودار می افزاییم و درباره تمام انتخابهای متفاوت وتفا سیری که می‌توانید به آن بیافزایید، صحبت خواهیم کرد.

ســـپس عامل را به نمودار افزوده و درباره گزینه های موجود برای عامل بحث خواهیم نمود. نهایتا رابطه بین و عامل، رابطه بین عامل های متفاوت و رابطه بین را بررسی خواهیم نمود.

در انتهای فصـــل، اولین ســـری تمرینات را ارائه خواهیم داد. در این تمرینها مدلی را برای یک سی ستم پرداز شی منظم خواهیم ساخت. دراین ف صل، م شکلاتی را که در این زمینه بوجود می آید را معرفی کرده و شما را به سمت ساخت مرحله به مرحله یک مدل هدایت خواهیم کرد.

نمای Use case

در این ف صل درباره برخی از آیتم های تولید شده در نمای موجود در صحبت خواهیم کرد.

- use caseها
- عامل(actor)
- ارتباطات وابسته بین use case وعامل
- به کارگیری و توسعه روابط بین use caseها
- رابطه های عمومی عامل
- نمودارهای use case
- نمودارهایSequence (توالی) و Collaboration (همکاری)

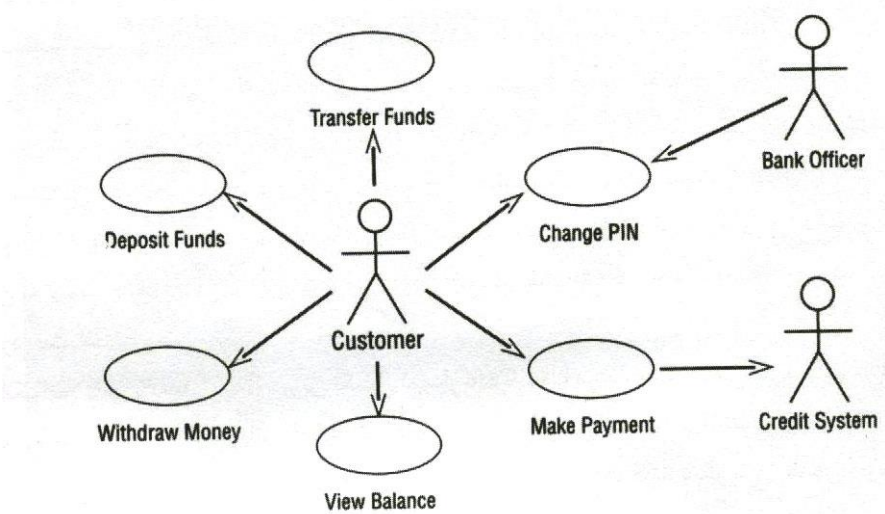
درباره تمام موضـــوعات بالا بجز نمودارهای که در فصـــل ٤ مورد بررسی قرار خواهند گرفت، صحبت خواهد شد.

نمای در حد بسیار بالایی به طور مستقل عمل می کند. و عامل بدون وارد شـــدن به جزئیات انجام کار، مانند زبان برنامه نویســـی که اســـتفاده خواهد شد، محدوده پروژه را تعیین می کنند.

نمودارهای Rational rose

نمودار، برخی از موجود در سیستم مورد نظر شما، برخی از عامل‌های موجود در سیستم شما و رابطه های بین تمامی اینها را مشخص می کند.عملیات سطح بالایی برخوردار ا ستکه سی ستم مهیا می‌کند. عامل هرچیز و یا هرکسی ا ست که بر سیستمی که درحال ساخت ا ست اثر می گذارد. چگونگی استفاده از و عامل را به طور مختصر برایتان شرح می دهیم.

شکل ۱-۳ : نمونه‌ای از نمودار Use Case



نمونه ای از استفاده نمودار در شکل بالا نشان داده شده است.در این نمودار سه عامل خواهیم داشت: مشتری، کارمند بانک و سیستم حسابداری. سیستم شش عملیات مهم را انجام می دهد که به این شرح‌اند:

واریز پول به حساب، انتقال پول، برداشت از حساب، تغییر،مشاهده مقدار موجودی و پرداخت پول.یکی از مزیت های بزرگ نمودار تبادل اطلاعات است.مراجعه کنندگان شما می توانند به این نمودارها نگاه کرده و اطلاعات وسیعی را بدست آورند.

با نگاه به نمودار، خواهند فهمید که چه عملیاتی در سیستم انجام می شود. با نگاه به عامل ها، خواهند فهمید که چه کسانی بر سیستم کنش دارند. با نگاه به مجموعه و عامل، می فهمند که چه محدوده ای از پروژه انجام خواهد شد. بنابراین کمکی به آنها خواهد بود تا از هر عملیات از قلم افتاده ای، یک ذهنیت اولیه داشته باشند. به طور مثال فردی ممکن است به این نمودارنگاه کرده و بگوید: «عالی است، ولی من نیاز به امکاناتی دارم که به من ده عمل آخر صورت گرفته برروی حساب شخصی را نشان دهد.»

شما مکررا نمودارهای متفاوتی را برای یک سیستم می سازید. یک نمودار سطح بالا، که در نامیده می شود، فقط بسته های نرم افزاری یا گروه بندی را نشان می دهد، نمودارهای دیگر مجموعه و عامل را نشان خواهند داد. اینکه چند نمودار را بسازید و چه نامی به آنها بدهید کاملا به خودتان بستگی دارد.

باید مطمئن شوید که نمودار شما دارای تمام اطلاعات مفید مورد نیاز می باشد، البته نباید نمودار را آن چنان شلوغ نمایید که باعث سردرگمی گردد.

نمودارهای کار مشخصی را- برای مستند سازی عامل ها (هر چیز خارج از محدوده سیستم)، use caseها (هر چیز درون محدوده سیستم) و ارتباط آنها، انجام دهد.

نکاتی را که باید به عنوان کسی که یک نمودار use caseها را ایجاد می نماید، به خاطر داشته باشید، بدین ترتیب می باشند:

* ارتباطات عامل با عامل را مدل سازی نکنید. بنا به تعریف، عامل‌ها خارج از محدوده پروژه جاری هستند. بنابراین ارتباط برقرار شده بین آنها نیز خارج از محدوده پروژه ای که می سازید خواهد بود. می توانید از یک نمودار جریان کاربرای مجسم کردن ارتباط بین عامل ها استفاده کنید.

* هیچ گاه مستقیما با فلش، را به هم وصل نکنید(بجز در ارتباطات که در مورد آنها بحث خواهد شد.) نمودار نشان می دهدکه چه وجود دارند، ولی روند و ترتیب اجرای نشان داده نخواهد شد. برای چنین منظوری می توان از یک نمودار فعالیت استفاده کرد.

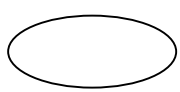
* هر باید توسط یک عامل آغازبه کار کند. به همین دلیل باید ابتدا فلش در سمت عامل و انتهای آن به سمت با شد. در این مورد هم برای ارتباطات یا که بعدا به بررسی آنها خواهیم پرداخت، استثنا وجود دارد.

* بانک اطلاعاتی را به عنوان لایه زیرین تکمیل نمودار در نظر بگیرید. می توانید با استفاده از یک درون یک بانک اطلاعاتی، اطلاعات را وارد کنید،

و به همین اطلاعات با اســتفاده از یک دیگر دســترســی پیدا نمایید. نباید فلشی را از یک به یک دیگر برای جریان حرکت اطلاعات بکشید.

کار با Use case

بخش سطح بالایی از عملیاتی است که سیستم مهیا می کند. به عبارت دیگر، اینکه شـخص چگونه از سـیسـتم اسـتفاده می کند را شـرح می‌دهد. بیایید با نگاه به یک مثال کار را شروع کنیم.یک ما شین، یک سری عملیات ا صلی را برای م شتری انجام می دهد. به م شتری اجازه می دهد تا پول به حساب بریزد، نقدا از حساب برداشت کند، پول را از یک حساب به حساب دیگر منتقل نماید، مقدار موجودی خود را مشاهده کند، تعویض نماید و یا توســط کارت اعتباری پول پرداخت نماید. هر کدام از این روش متفاوت اســتفاده مشــتری از ســیســتم می باشــد.به هر حال هر کدام از آنها یک متفاوت هستند. در یک با استفاده از علامت زیر نمایش داده می شود:



Use case

یک مزیت نگاه به سیستم با استفاده از Use case این است که می‌توان پیاده سازی سیستم را از دلیل ایجاد سیستم در ابتدا جدا نمود. ذهنتان را بر آنچه که مهم است متمرکز کنید- یعنی برطرف کردن نیازها و توقعات مشتری بدون نیاز به درگیر شدن با جزئیات پیاده سازی. با نگاه کردن به ها، مشتری خواهد فهمید که چه عملیاتی مهیا خواهد شد و قبل از اینکه پروژه به مراحل جلوتر برود، می تواند خودش را با سیستم وفق دهد.

Use caseها به صورت دیگری به متدهای سنتی نزدیک می‌شوند. شکستن پروژه به use caseها، یک روش نگاه کردن به پروژه به صورت پردازش گرا ست و نه به صورت عملگرا. البته با تجزیه عملیاتی که گاهی اوقات انجام می شود، تفاوت دارد. تجزیه عملیاتی بر اینکه چگونه باید مشکلات سیستم را برای حل شدن به قطعات کوچک و کوچکتر تبدیل کرد، تمرکز دارد، در حالی که use case تمرکز کار را بر روی آنچه مشتری

از سیستم توقع دارد، قرار می دهد. وقتی وقتی در حال شروع یک پروژه هستید، یک سوال طبیعی این است:

چگونه باید use caseها را پیدا کرد؟ یک راه خوب برای شروع این است که سندی که مشتری تهیه کرده است را درنظر بگیرید. اغلب اوقات، یک سند که دارای نسخه یا محدوده سطح بالایی است می تواند به شما در شناسایی use caseها کمک کند. هر کدام از بانکدارهای موجود در پروژه را درنظر بگیرید. از خودتان بپرسید که هر بانکداری چه توقعی از سیستم دارد. برای هر بانکدار، این سوال ها را مطرح کنید:

- بانکدار چه کاری را باید با سیستم انجام دهد؟
- آیا نیاز است که بانکدار اطلاعاتی را نگهداری کند(ساختن، خواندن، بروز رسانی، حذف)؟
- آیا بانکدار باید سیستم را درباره اتفاقاتی که در خارج از سیستم رخ می دهد، آگاه نماید؟

همانگونه که قبلا متذکر شدیم،use case ها مستقل از پیاده سازی هستند و یک دید سطح بالا از آنچه کاربر از سیستم انتظار دارد می‌باشند. بیایید هر بخش از این تعریف را جداگانه در نظر بگیریم.

اولاuse case ها به طور مستقل عمل می کنند. درحالی که use case را تعریف می کنید، فکر کنید که در حال ساخت یک سیستم دستی هستید که مکانیزه نشده است. Use case شما باید قابل ساخت در Java،C++،Visual Basic و یا حتی روی کاغذ باشد.use case بر آنچه سیستم باید انجام دهد متمرکز می شود، نه بر اینکه سیستم چگونه آن را انجام می دهد. بعدا در پردازش به چگونگی کار سیستم می رسیم.

دوما، use caseها یک دید سطح بالا از سیستم هستند. اگر سیستم شما ۳۰۰۰ مورد use case دارد، شما سادگی و روان بودن را از دست داده‌اید. وقتی که مجموعه ای از useها را می سازید، باید یک دید کلی سطح بالا از تمام سیستم را به صورت ساده وروان، برای مشتری ایجاد کند.

نباید آنقدر زیاد use case داشته باشید که مشتری به زحمت بتواند سند را بررسی کند، فقط در این حد باشد که بفهمدسیستم چه کاری را انجام می دهد. در همین حال باید به اندازه کافی use case داشته باشید تا آنچه را که سیستم انجام می دهد را دقیقا شرح نماید. یک سیستم در حد معقول باید بین ۲۰ تا۵۰،use case داشته باشد.

همانگونه که بعدا خواهید دید، می توانید رابطه های متفاوتی که رابطه‌های uses،extends نامیده می شوند رااستفاده کنید تا در صورت نیاز کمی use case را تجزیه نمایید. همچنین می توانید use case ها را به صورت یک بسته نرم افزاری جمع آوری کنید تا گروه‌هایی از use caseها را داشته باشید که در سازماندهی کردن آن راحت‌تر باشید. در فرصت دیگری در این مورد صحبت خواهیم کرد.

نهایتا تمرکز use case باید بر آنچه که کاربر از سیستم بدست می‌آورد، باشد. هر use case باید یک داد و ستد کاملی بین کاربر و سیستم ارائه دهد که نتیجه آن مقداری مختص کاربر خواهد بود. use case باید از دیدگاه تجاری نامگذاری شود و نه از دیدگاه تکنیکی و این

نام باید برای مشـــتری مفهوم باشـــد. در ATM، ما هیچ رابطه ای با use case ای که به این صورت تعریف شده ا ست نداریم: سی ستم بانکی که پول را ازیک کارت اعتباری به صورت چک منتقل می کند. در عوض ما یک use case خواهیم داشت که برای مشتری مفهوم تر است:

پرداخت پول توســـط کارت اعتباری. در نامگذاری use caseها معمولا از افعال و یا ا صطلاحات کوتاه دارای فعل ا ستفاده می شود که باید آنچه که م شتری به صورت نتیجه دریافت می کند را تو ضیح دهد. برای مشـــتری اهمیتی ندارد که شـــما با چند ســـیســـتم دیگرارتباط دارید، و چه مراحل م شخ صی باید اجرا شوند، و یا چند خط کد باید برای پرداخت پول گذرنامه نو شته شود. تمام آن چیزی که برای آنها اهمیت دارد این ا ست که پرداختی انجام شده است.مجددا به سمت آنچه کاربر از سیستم توقع دارد، (ونه مراحلی که باید طی شوند تانتیجه بدست آید)، متمرکز شوید.

وقتی که لی ست نهایی تمامuse case ها را بد ست آوردید، چگونه خواهید فهمید که آیا همه آنها را پیدا کرده اید یا نه؟ برخی پرسشهایی که باید مطرح کنید، بدین شرح اند:

- آیا هر نیاز عملکردی حداقل در یک use case وجود دارد یا نه؟ اگر یک نیاز در هیچ use case وجود نداشته باشد، اجرا نخواهد شد.
- آیا در نظر گرفته اید که هر بانکدار چگونه از سیستم استفاده می‌کند؟
- هر بانکدار چه اطلاعاتی از سیستم دریافت می کند؟
- ورود و خروج به سیستم را چگونه درنظر گرفته اید؟ کسی باید باشد که سیستم را راه اندازی کند و در انتها آن را متوقف نماید.
- آیا تمام سیستمهای خارجی که سیستم باید با آنها در تماس باشد را تعریف کرده اید؟
- چه اطلاعاتی از یک سیستم خارجی گرفته می شود و یا به آن داده می شود؟

مستند سازی جریان رخدادها (Flow of Event)

Use case شروع می کند به توضیح دادن اینکه سیستم شما چه کاری را انجام می دهد. برای اینکه سیستم را بصورت حقیقی بسازید، نیاز به جزئیات مشخص بیشتری دارید. این جزئیات در یک سند که جریان رخدادها (Flow of Event) نامیده می شود، نوشته می شوند.

جریان رخدادها این است که جریان منطقی در سرتاسر use case را مستند سازی می کند. این سند تمام آنچه کاربر سیستم انجام می‌دهد و آنچه که کاربر سیستم انجام می دهد و آنچه که خود سیستم انجام می‌دهد را تعریف می کند.

جریان رخدادها هنوز از نظر اجرایی مستقل عمل می کنند. می‌توانید فرض کنید که در حال نوشتن جریانی هستید که یک سیستم مکانیزه خواهد بود. به هر حال در این حالت هم نباید نگران این با شید که سیستم در C++ ساخته می شود یا در Power Builder یا در Jvav. هدف نهایی در اینجا هم آنچه که سیستم انجام می دهد خواهد بود، نه اینکه سیستم چگونه آن را انجام می دهد. جریان رخدادها دارای بخشهای زیراست:

- یک تعریف مختصر
- پیش شرایط
- جریانهای رخدادهای اصلی
- جریانهای رخدادهای فرعی
- شرایط پسین(Postcondition)

تعریف (description)

هر use case باید یک تعریف مختصـــر مرتبط با آنچه که use case انجام می دهد را داشـته باشـد.Use case مربوط به انتقال وجوه در ATM ممکن است تعریفی مانند زیر داشته باشد:

use case مربوط به انتقال وجوه، به مشـتری یا کارمند بانک اجازه می دهد تا وجوه را از یک حساب پس انداز یا حساب در حال بررسی به یک حساب پس انداز دیگر یا حساب در حال برررسی دیگر، منتقل کند.

تعریف باید کوتاه ومرتبط باشد، ولی باید تمام کاربرهای متفاوتی که use case را اجرا می کنند و نیز نتیجه نهایی که کاربر توقع دارد توسـط use case به آن برسد را دربرداشته باشد.

همچنان که پروژه جلو می رود(خصوصا در پروژه های طولانی)،

این تعاریف use case، به تمام تیم کمک می کند تا بخاطر آورند که دلیل قرار گرفتن هر use case در پروژه چه بوده است و قرار بوده

هر use case چه کاری را انجام دهد. همچنین با مستند سازی دقیق هدف، use case به کاستن سردرگمی بین اعضای تیم کمک می کند.

پیش شرایط (Precondition)

پیش شرایط یک use case، لیستی از شرایطی هستند که قبل از شروع کار use case باید برر سی شوند. به طور مثال، یک پیش شرط می تواند این با شد که یک use case دیگر اجرا شده ا ست و یا کاربر حق د ستیابی م ستقیم برای اجرای use case جاری را دارد.تمام use caseها دارای پیش شرط نیستند.

توضیحاتی که در بالا داده شدند در واقع این نکته را متذکر می‌شوند که نمودارهای use case، نشاندهنده ترتیبی که باید use caseها اجرا شوند، نیستند. پیش شرایط، می توانند برای م ستند سازی برخی اطلاعات این چنینی می باشند: به طور مثال، ممکن است پیش شرایط یک use case این باشد که یک use case دیگر اجرا شده باشد.

جریان رخداد اصلی وفرعی

مشخصات کامل use case در جریان رخداد اصلی وفرعی تعریف شده است.جریان رخدادها، آنچه را که در اجرای عملیات در use case پیش خواهد آمد را مرحله به مرحله توصیف می کند. جریان رخدادها بر آنچه سیستم انجام خواهد داد متمرکز می شود ولی به چگونگی انجام آن کاری ندارد زیرا جریان رخداد از دید کاربر نوشته می شود.جریان رخدادهای اصلی و فرعی شامل موارد زیر است:

- چگونگی شروع کارuse case
- مسیرهای متنوع منتهی بهuse case
- جریان رخداد معمولی یا اولیه منتهی بهuse case
- در طولuse case هر گونه انحرافی از جریان رخداد اصلی به عنوان روند فرعی شناخته می شوند.
- هر روند خطا
- چگونهuse case به پایان می رسد.

به طور مثال،جریان رخداد برای use case مربوط به برداشت پول از حساب ممکن است به این شکل باشد:

جریان اولیه

۱. use case وقتی شروع می شود که م شتری بانک کارتش را درون ATM قرار دهد.

۲. ATM یک پیغام خوش آمدگویی به مشـــتری می دهد و به او اجازه می دهد که شماره مشخصات فردی (PIN) خویش را وارد نماید.

۳. مشتری بانک PIN خود را وارد می کند.

٤. ATM صـحت و اعتبار PIN را بررسـی می کند. اگرPIN معتبر نباشد، روند فرعیA1 انجام خواهد شد .

۵. ATM گزینه های زیر را آماده می کند:

- وجوه واریز شده
- برداشت نقدی
- انتقال وجوه

۶. کاربر گزینه برداشت از حساب را انتخاب می کند.

۷. ATM اجازه برداشت می دهد.

۸. کاربر مقداری که می خواهد برداشت کند را وارد می نماید.

۹. ATM بررسی می کند که آیا در حساب به اندازه مورد نیاز پول موجود است یا خیر. اگر به اندازه کافی پول موجود بود، جریان رخداد فرعی A2انجام خواهد شد.

اگر در یافتن مقدار وجوه خواسته شده اشکالی ظاهر شود، روند خطایE1انجام خواهد شد.

۱۰. ATM مقدار وجوه برداشت شده ازحساب مشتری را کم می کند.

۱۱. ATMوجه خواسته شده را به مشتری پرداخت می نماید.

۱۲. ATM یک رسید برای مشتری چاپ می کند.

۱۳. ATM، کارت مشتری را از دستگاه بیرون می فرستد.

۱٤. use case به پایان می رسد.

جریان رخداد فرعی A1 :PIN وارد شده نا معتبر

۱. ATM به مشتری اخطار می دهد که به مقدار کافی پول در حساب او موجود نیست .

۲. ATMکارت مشتری را بیرون می فرستد .

۳. Use case به پایان می رسد.

روند خطای E1 : خطا در صحت وجوه درخواستی

۱. ATMیک پیغام خطا را به مشتری مبنی بر اینکه خطایی در شناسایی وجوه پیش آمده است ، می دهد و شماره تلفن مرکز خدمات بانکی را به مشتری ارائه می کند.

۲. ATM، خطای پیش آمده را در رکورد خطاها ثبت می کند.رکورد مربوطه دارای نام مشتری و شماره حساب، تاریخ و ساعت وقوع خطا و کد خطا، خواهد بود.

۳. ATM کارت مشتری را بیرون می فرستد.

٤. use case به پایان می رسد.

زمان مستند سازی جریان رخدادها، می توانید مانند مثال بالا از لیست‌های شماره دار، و یا متون پاراگراف بندی شده، لیست های bullet شده و یا حتی فلوچارت استفاده کنید.

روند جریان باید با نیازهای جمع آوری شده، هماهنگی داشته باشد. هنگامی که تصمیم می گیرید یک جریان را مستندسازی کنید، بازدیدکنندگان از سند را بخاطر بسپارید. مشتریان این سند را وارسی می‌کنند تا ببینند که آیا عملکرد آن، توقعات ونیازهای آنها را برطرف می‌سازد یا نه.

تحلیلگر(Analyst) برای اطمینان از اینکه آیا سند با نیازها هماهنگی دارد یا خیر، آن را بررسی می کند.مدیرپروژه آن را بررسی می کند تا بتواند تصویربهتری، از آنچه قرار است ساخته شود را بدست آورد.

چنانچه بخواهد، بتواند برآوردهای هزینه مربوط به پروژه را اصلاح کند.هنگامی که درحال نوشتن جریان هستید، از جزئیات چگونگی کار دوری کنید. به فکر نوشتن دستورالعمل باشید. یک دستورالعمل می‌واند به این شکل باشد« دوتخم مرغ اضافه کن» هرگز نباید بگویید «به سمت یخچال

برو. تخم مرغ را از آنجا بردار. اولین تخم مرغ را در د ست بگیر. آن را به کنار کاســه بزن...» در جریان رخداد ممکن اســت بگویید اعتبارIDکاربررا بررســی کن». ولی مشــخص نخواهید کرد که این عمل با نگاه کردن به کدام جدول مربوطه در جدول بانکی صورت

خواهد گرفت. توجهتان باید بر روی اطلاعات رد وبدل شــده بین کاربر و سیستم با شد و نه بر روی اینکه سیستم چگونه کار خود را انجام می دهد.

Post Conditions (شرایط پسین)

Postcoditionها شرطهایی هستند که باید بعداز پایان اجرای use case، دارای مقدار tnue باشند، به طور مثال، ممکن است بعد از کامل‌دن یک use case، یک flag را تنظیم کرده با شید. اینچنین اطلاعاتی شرایط پسین (Postconditions) نامیده می شوند. مانند پیش شرایط، شرایط های پســین هم می توانند برای افزودن اطلاعاتی درباره جریانی که use

caseها در آن اجرا می شوند، به کار گرفته شوند. به طور مثال، اگر یک use case همیشه باید بعد از یک use case دیگر اجرا شود، می توانید آن را در یک شرایط پسین مستند سازی کنید. همه use caseها دارای شرایط پسین نیستند.

کار کردن با عامل ها (Actor)

هر کس یا هر چیزی که با سیستم موجود برهم کنش دارد، عامل (actor) نامیده می شود.use case ها هر چیز موجود درمحدوده سیستم را توصیف می کنند، در حالی که عامل ها در خارج از محدوده سیستم قرار دارند. در UML، عامل ها با آدمک هایی به شکل زیر نشان داده می شوند:

Actor 1

سه نوع اصلی از عامل ها وجود دارند: کاربران سیستم، سیستم های دیگری که باسیستم موجود درارتباط هستند وزمان.

اولین نوع عامل یک انسان فیزیکی و یا به عبارت دیگر کاربر است. اینها بیشترین عامل مورد استفاده هستند و تقریبا در تمام سیستمها وجود دارند. درمثال ATM، برخی از کاربران ممکن است مشتریان ATM یا، فرد پشتیبان ATM باشند. برای نامگذاری این عامل ها ترجیحا به جای استفاده از نام موقعیت، از نام نقشها استفاده کنید.یک نام منحصر به فرد مزایای مهمی دارد. ممکن است یک روز صبح John Doe مسئول پشتیبانی ATM باشد، یعنی بازیگر نقش پشتیبان ATM. ممکن است از او بخواهد در اواسط روز مقداری پول ازحساب خود برای خرید غذای ظهر برداشت نماید.

پس در اینجا او نقش مشتری ATM را خواهد داشت.استفاده از نام نقشها به جای استفاده از نام موقعیت، تصویر پایدارتری از عامل ها را به شما ارائه خواهد داد. نام موقعیت برای یک عامل ممکن است تغییر کند و رابطه ها و مسئولیت ها ازیک موقعیت به یک موقعیت دیگر انتقال یابد.

بااستفاده از نقشها، با افزایش یا تغییر یک موقعیت، نیازی به بروز رسانی مدل نخواهید داشت.

دومین نوع عامل، سیستم دیگر است.به طور مثال، ممکن است بگویید که بانک ما دارای یک سیستم اعتباری است که برای پشتیبانی از اطلاعات اعتبار حساب هر مشتری استفاده می شود. سیستم ATM، ما باید قابلیت محاوره(ارتباط) با این سیستم اعتباری را داشته باشد.

در این مورد، سیستم اعتباری دارای نقش عامل خواهد بود. دقت کنید که با دادن نقش عامل به این سیستم، فرض ما براین است که سیستم اعتباری به هیچ وجه تغییر نمی کند. به خاطر دا شته با شید که عامل ها در بیرون از محدوده سیستمی که ما می سازیم قرار دارند و درنتیجه کاملا خارج از کنترل ما می باشند. اگر بخواهیم در سیستم اعتباری تغییرات مهمی ایجاد کنیم، این سیستم باید درون محدوده پروژه باشد و دارای نقش عامل نباشد.

سومین نوع عامل که زیاد استفاده می شود، زمان است. هنگامی که زمان تبدیل به یک عامل می شود که زمان در حال گذر باعث ایجاد رخدادی در سیستم گردد. به طور مثال ممکن است سیستم ATM هر

نیمه شـــب یک پردازش تطبیق ســیســـتم انجام دهد. چون زمان خارج از کنترل ما است، یک عامل می باشد.

ساخت یک عامل Abstract

یک عامل Abstract، عاملی اســت که هیچ مصــداق واقعی ندارد. به عبارت دیگر، کادرینالیتی عامل، دقیقا صفر است. به طور مثال، ممکن است چندین عامل داشته باشید: کارمند ساعتی، کارمند ثابت و کارمند موقتی . تمامی اینها نوعی از عامل چهارم هستند که عامل کارمند می باشد. باوجود این، هیچ کس در شـــرکت فقط یک کارمند نیســـت- هر کس یا کارمند ســـاعتی اســـت، یا کارمند ثابت اســـت یا موقتی. دلیل وجود عاملی با نام کارمند این است که رابطه معمول استخدام ساعتی، استخدام با حقوق ثابت و اســتخدام موقتی، نشــان داده شــود.هیچ مرحله و مصــداق واقعی برای عامل کارمند وجود ندارد، پس آن یک، عامل Abstract خواهد بود.

روش ایجاد یک عاملAbstract:

۱. در مرورگرو یا نمودار use case، عاملی را ایجاد نمایید.

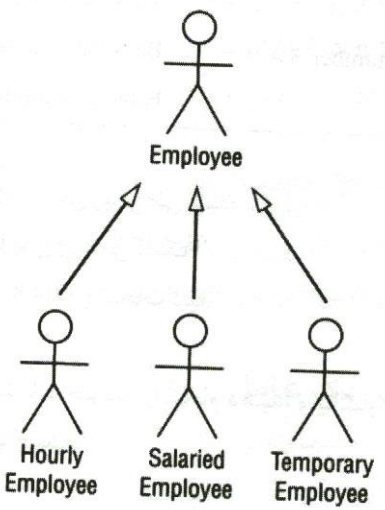
۲. در مرورگر و یا نمودارuse case، بر روی عامل کلیک راست کنید.

۳. از منوی باز شده Open Specification را انتخاب نمایید.

٤. برگه Detail را باز کنید.

۵. کادر انتخاب Abstract را علامت گذاری نمایید.

شکل ۱۴-۳ : عامل Abstract



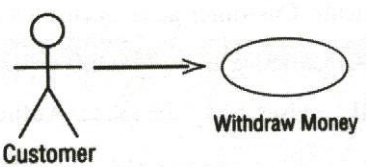
چگونگی کار با رابطه ها

UML از انواع متعددی از رابطه ها برای use caseها و عامل ها پشتیبانی می کند. این شامل رابطه های Communiccation، رابطه‌های extend و رابطه های generalization برای عامل می باشد.

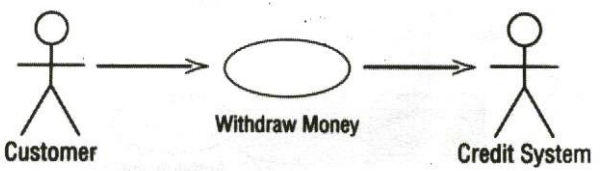
رابطه های uses و extend، رابطه های بین use case ها را تعریف می کنند. رابطه های Actor generalization، رابطه بین عامل‌ها را تعریف می کند.

رابطه Communiccation

به رابطه بین use case و عامل، رابطهCommuniccation می گویند. در UML، رابطه های اطلاعاتی با اســتفاده از فلش به حالت نمودار درمی آیند.



ابتدای فلش نشان دهنده کسی است که ارتباط از آنجا شروع می شود. در مثال زیر، عامل مشتری، آغازگر ارتباط با سیستم برای برداشت پول از حساب می باشد. یک use case هم می تواند آغازگر ارتباط با یک عامل باشد.



در این مثال، use case آغازگر ارتباط با عامل سیستم اعتباری است. در زمانی که use case مربوط به پرداخت پول دراجرا است، سیستم ATM ارتباط با سیستم اعتباری را برای کامل کردن محاوره شروع می کند.

تبادل اطلاعات در هر دو جهت انجام می شود. از سیستم ATM به سیستم اعتباری و به عکس، و فلش تنها نشان دهنده آغازگر این ارتباط خواهد بود.

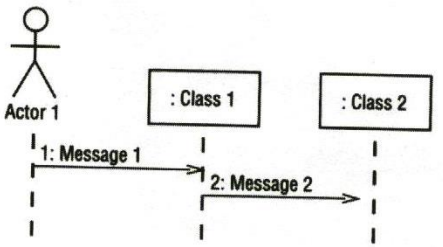
نمودارهای Interaction

یک نمودار Interaction روندی در یک use case را مرحله به مرحله نشان می دهد. در مثال ATM، ما چندین روند مناسب را در use case مربوط به برداشت پول از حساب داشتیم(Withdraw Money). چندین نمودار Interaction برای این use case خواهیم داشت. یکی از آنها نمودار Interaction با نام happy day است که نشان می دهد وقتی همه چیز به خوبی درحال انجام است چه اتفاقی می افتد.

نمودارهای اضافی دیگری را خواهیم داشت که نشان دهنده این است که چه اتفاقی برای رخدادهای متناوب می افتد. وقتی کسی PIN را اشتباه وارد می کند چه رخ خواهد داد، وقتی پول کافی هنگام برداشت از حساب وجود نداشته باشد چه رخ خواهد داد و غیره.

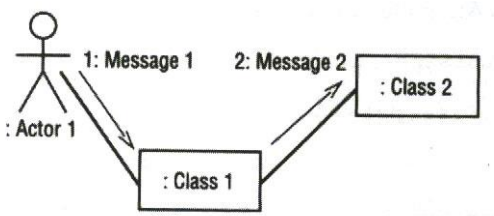
دو نوع نمودار Interaction وجود دارند که آنها را بررسی خواهیم نمود : نمودارهای Seqence ونمودارهای Collaboration. یک نمودار Seqence بر حسب زمان مرتب می شود.

شکل ۴-۱ : نمودارهای Sequence



یک نمودار Collaboration همان اطلاعات را نشان می دهد ولی به شیوه متفاوتی سازماندهی می شود.

شکل ۴-۲ : نمودارهای Collaboration



هر دونمودار seqence و Collaboration اطلاعات یکسانی را نشان خواهند داد؛ با وجود این، چند تفاوت کوچک بین نمودار های sqence و Collaboration وجود دارد. نمودارهای Seqence نشان دهنده مرکز کنترل هستند، نمودارهای Collaboration نشان دهنده یک روند داده ای هستند.

نمودارهایInteraction تعداد زیادی از همان جزئیات بیان شـــده در جریان رخدادها را در بر می گیرندو ولی در اینجا اطلاعات به گونه ای که برای برنامه نویس سودمندتر باشند، ارائه می شوند. تمرکز اصلی این نمودارها بر روی آبجکت هایی که برای پیاده سازی عملیات بیان شده در use case به کار می رو ند، می باشـــد. نمودار های Sequence و Collaboration آبجکت ها، کلاس ها یا هردوی آنها را نشان می دهند

یک Object چیست؟

آبجکت ها دراطراف ما قرار دارند. صندلی که روی آن می ن شینید، کتابی که آن را می خوانید و لامپی که به شـــما کمک می کند. به زبان نرم افزار، آبجکت معنای نزدیکی با این مفاهیم دارد.

آبجکت آن چیزی اســـت که اطلاعات و روش ها را در خود کپســـوله (نگهداری) می کند. روشی است که برخی از چیزهای عینی در دنیای واقعی را نشان می دهد. مثال هایی از آبجکت ها به صورت زیر می‌باشند:

- حساب Joe

- خانه ای در 7638 MAIN Street
- گل زردی که بیرون ازپنجره خانه من قرار دارد.

در مثال ATM، آبجکت های دیگر می توانند بدین صـــورت باشـــند: صفحه نمایشATM(ATM Screen)، کارت خوان(cart reader)، کارت ATM مربوطه بهJoe ، و رسـیدی که برای Joe پس از برداشـت ۲۰$، صـــادر می شـــود. هر آبجکت درون خودش مقداری اطلاعات و رفتار را قرار داده است. به طور مثال حساب Joe برخی اطلاعات را به این صورت دارد: شماره حساب ۱۲۳٤٥٦۷۸۹ می باشد، حساب متعلق به Joe Smit است ومقدار موجودی حساب درحال حاضر ۳۵۰$ می باشد.

همچنین حســـاب Joe دارای یک ســـری رفتار ها وعملکرد هایی می‌باشـــد، این حســـاب می داند هنگامی Joe پولی به حســـاب خودش می‌گـذارد، چگونـه مقـدار خودش را افزایش دهـد. هنگـامی کـه Joe می‌خواهد از حساب پول برداشت نماید، مقدار خودش را کاهش می‌دهد،

و هنگامی که Joe بخواهد اضافه تراز موجودیش از حساب برداشت نماید، چگونه رفتار کند.

بخش های اطلاعاتی که توسط آبجکت نگهداری می شوند، صفات (Attribute) آن می باشند. با وجود اینکه مقادیر صفات برای آبجکت ها تغییر خواهند کرد، خود صفات هرگز تغییر نخواهند کرد(ممکن است هفته آینده Joe دارای ٤٠٠$ پول درحساب خویش باشد). حساب Joe همیشه دارای یک شماره حساب، یک نام مالک حساب و یک مقدار موجودی می باشد.

رفتارهای یک آبجکت به عنوان عملیات آن شناخته می شوند. دراین مثال، عملیات برای حساب Joe شامل این موارد است: تنظیم موجودی حساب برای برداشت و یا واریز پول، وبررسی اینکه آیا از حساب قرار است بیشتر از موجودی پول برداشته شود یا خیر.

در Rose، آبجکت ها به نمودارهای Interaction افزوده می‌شوند.

هنگام کشیدن یک عامل(که Stereotype class است)

یا کشیدن دیگر کلاس ها به نمودار Interaction افزوده می شوند.

یک نمونه آبجکت از آن کلاس به طور خودکار ساخته می شود. در Rose، با حذف یک آبجکت از یک نمودار، کلاس را از کل مدل حذف خواهد نمود.

یک کلاس چیست؟

طرح کلی برای یک آبجکت را کلاس آن فراهم می کند. به عبارتی دیگر، یک کلاس تعیین کننده اطلاعاتی است که یک آبجکت می تواند نگهداری کند و نشان دهنده رفتارهایی است که می تواند داشته باشد.

به طور مثال،کلاس های مربوط به برر سی د ساب Joe، خانه ای در 7638 MAIN Street ممکن است ارتفاع ۲۰ فوت و عرض ۶۰ فوت باشد و دارای ۱۰ اتاق باشد ومساحت ۲۰۰۰ فوت مربع باشد.

یک کلاس شـــکل عمومی تری داردو یک الگو را برای آبجکت مهیا می‌کند.کلاس را به عنوان طرح کلی یک خانه در نظر بگیرید، وآبجکت‌ها را در۲۵ خانه که با همان طرح کلی ساخته شده اند، در نظر بگیرید.

یافتن آبجکت ها

یک راه برای یافتن برخی آبجکت ها این اســت که نام ها را درجریان رخدادها در نظر بگیرید. یک جای خوب دیگر برای بدســـت آوردن آنها، ســناریوی اســناد می باشــند. یک ســناریو، حالت خاصــی از جریان رخدادها می‌باشـــد. جریان رخدادها برای use case مربوط به برداشـــت پول از حساب، درباره فردی در ATM صحبت می کند که درحال برداشت پول از حساب است.یکی از سناریوها برای این مورد می تواند برداشت Joe از حساب، به مقدار ۲۰ دلار باشد. سناریوی دیگر، می تواند سعی Jane در برداشت ۲۰ دلار از حساب باشد، در حال که اوPIN را اشتباه وارد کرده است. سناریوی دیگر، می تواند سعی Bob در برداشت ۲۰ دلار از حساب

باشـــد، در حالی که او به مقدار کافی پول درحســـاب ندارد. معمولا سناریوهای زیادی برای جریان رخدادها وجود دارند.

یک نمودارSequence و Collaboration، یکی ازاین سـناریوها را شرح می دهد. هنگامی که در سناریوی خود به اسامی نگاه کنید، برخی از اسامی عامل خواهند بود، برخی از آنها آبجکت خواهند بود و برخی صفاتی برای یک آبجکت خواهند بود. وقتی درحال ایجاد نمودار Interaction هستید، اسامی به شما خواهند گفت که آبجکت ها چه خواهند بود.وقتی به یک اسم نگاه می کنید و مردد هستید که آیا آن یک آیجکت یا صفت است، از خودتان بپرسید که آیا این آبجکت رفتاری دارد. اگر صـرفا دارای اطلاعات اسـت، یک صـفت خواهد بود. اگر رفتارهایی نیز داشته باشد، می تواند یک آبجکت باشد.

همه آبجکت ها در جریان رخدادها وجود نخواهند داشـــت. به طور مثال، Formها ممکن اسـت در روند رویدادها ظاهر نشـوند، ولی باید بر نمودار ظاهر شـــوند تا به عامل اجازه دهد که اطلاعات را وارد کرده و یا ببیند. آبجکت های دیگری که در جریان رخدادها ظاهر نمی شـــوند،

آبجکت های کنترل هستند. اینها آبجکت هایی هستند که تناوب روند در use case را کنترل می کنند.

استفاده از نمودارهای Interaction

طراحان وبرنامه نویسان می توانند از طریق نمودارها، کلاس هایی را که نیاز دارند ساخته شود، رابطه های بین کلاس ها و عملیات و مسئولیت ها برای هر کلاس را تعریف کنند. نمودارهایInteraction اساس وپایه طرح در حال ساخت می شوند.

نمودارهای Sequence بر حسب زمان مرتب می شوند. برای مواقعی مفید است که کسی بخواهد روند منطقی یک سناریو را بازدید کند. با اینکه نمودارهای Collaboration اطلاعات متناوب و زنجیره ای را در بر می گیرند، نگاه به نمودار Sequence آسان تر خواهد بود.

نمودار هایCollaboration ز مانی مف ید واقع می شـــو ند که بخواهید به تاثیر تغییرات دســـت یابید. اینکه بفهمید چه آبجکت هایی با چه آبجکت های دیگری تبادل اطلاعاتی انجام می دهند، به راحتی با نگاه به نمودارهای Collaboration، قابل انجام است.

اگر نیاز دار ید که یک آبجکت را تغییر دهید، می توانید به راحتی ببینید که چه آبجکت های دیگری ممکن است درارتباط با آن باشند.

نمودارهای Sequence موارد زیر را دربر می گیرند:

Objects(آبجکت ها): یک نمودارInteraction می توا ند از نام آبجکت ها، نام کلاس ها و یا هر دوی آنهااستفاده کند.

Messages(پیغام ها): با اســـتفاده از یک پیغام، یک آبجکت یا کلاس می‌تواند از یک آبجکت یا کلاس دیگر، برخی عملیات خاص را درخواســـت کند. به طور مثال، ممکن ا ست یک فرم از یک آبجکت گزارش بخواهد که خودش را چاپ کند.

در هنگام ساختن نمودارهایSequence باید به این نکته توجه داشته باشید که درحال تخصیص مسئولیت به آبجکت ها می باشید. وقتی پیغامی را به یک نمودار Interaction می افزایید، درحقیقت به آبجکت درحال دریافت پیغام، یک مسئولیت را واگذار می کنید.

مطمئن شوید که یک رابطه مناسب را به آبجکت مناسب تخصیص داده اید. دربیشتر برنامه های کاربردی، صفحه نمایش ها وفرم ها نباید هیچ گونه پردازش تجاری انجام دهند. آنها صرفا به کاربر اجازه ورود و یا دیدن اطلاعات را می دهند. با جدا کردن پیش پرداخت از منطق تجاری، ساختاری را ایجاد کرده اید که اثرات کوچک تغییرات را کاهش می دهد. اگر منطق تجاری نیاز به تغییر داشته باشد، واسط گرافیکی نباید تحت تاثیر قرار بگیرد.

اگر شکل ظاهری صفحه نمایش را تغییر دهید، منطق تجاری نیازی به تغییرات نخواهد داشت. به آبجکت های دیگر نیز باید به همین ترتیب مسئولیت های مناسب خودشان را محول کرد. اگر نیاز به چاپ گزارش از

موجودی تمام حساب ها داشته باشیم، نباید حساب Joe مسئول این کارباشد. مسئولیت های حساب Joe به محدوده Joe و پول او ختم می‌شود. آبجکت دیگری می تواند مسئول همه حساب ها برای چاپ یک گزارش باشد.

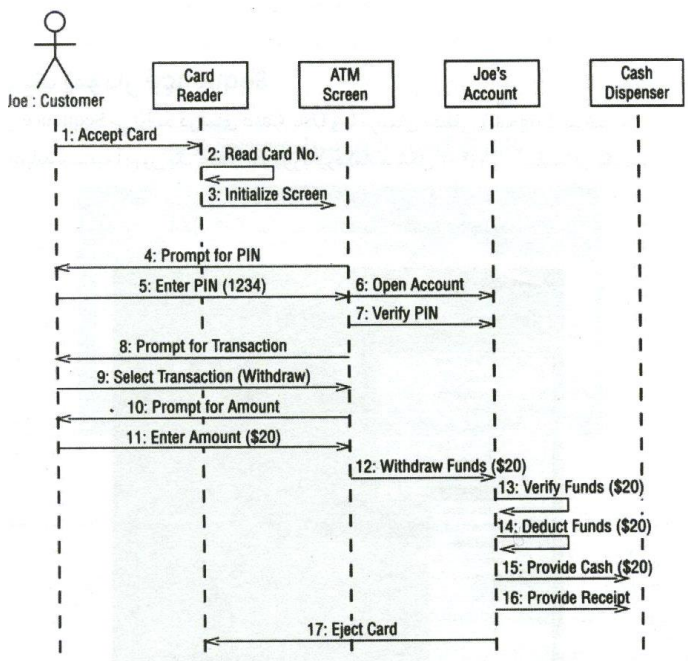
نمودارهای Sequence

بیایید نگاهی به نمودار های Sequence بیا ندازیم. نمودار های Sequence، نمودارهایInteraction هستند که بر مبنای زمان تنظیم می شوند؛ شما نمودار را از بالا به پایین مشاهده می کنید.هر use case تعدادی جریان متوالی خواهد داشت. هر نمودارSequence ، یک روند را در use case نمایش می دهد.

می‌توانیم این نمودار را با نگاه به آبجکت ها و پیغام ها، بخوانیم. آبجکت‌هایی که در روند شرکت می کنند، با مستطیل هایی در بالای نمودار نمایش داده می شوند. در این مثال، پنج آبجکت وجود دارد:

Joe، کارت خوان(دســـتگاه)، صـــفحه نمایشATM، حســـاب Joe و دســـتگاه پرداخت کننده پول.آبجکت با عاملی با نام Joe که آغازگر use case است دربالاترین قسمت نمودار و درسمت چپ نشان داده می‌شود.

شکل ۳-۴ : نمودار Sequence که Joe را در حال برداشت ۲۰ دلار از ATM نشان می‌دهد.



پردازش وقتی شروع می شود کهJoe کارت خویش را در د ستگاه کارت خوان قرار می دهد.کارت خوان، شـــماره قرار گرفته کارت Joe را می‌خواند و به صفحه نمایش ATM می گوید که کار خویش را آغاز نماید.

ATM به Joe ا جازه ورود PIN را می د هد.Joe ،PIN را وارد می‌کند (۱۲۳٤) و ATM حســـاب او را باز می کند. اعتبار PIN متعلق به Joe بررر سی می شود و ATM به او اجازه انجام عمل را خواهد داد. Joe برداشت پول از حساب(Withdraw) را انتخاب خواهد کرد. ATM به Joe اجازه بردا شت از حساب را می دهد. Joe مقدار ۲۰ دلار را وارد می کند. ATM حســاب Joe را برای بررســی مقدار موجودی، بررســی می کند و مقدار ۲۰ دلار را از حســـاب خارج می کند. ATM این ۲۰ دلار را پرداخت کرده و کارت Joe را از دستگاه خارج می کند.

هر آبجکت برای خودش یک خط عمر دارد که بصـــورت خطوط عمودی خط چین در زیر آبجکت کشــیده می شـــود. یک پیام بین دو خط عمر موجود بین دو آبجکت قرار داده می شود تا ارتباط بین آبجکت ها را نشـــان دهد. هر پیغامی نشـــان دهنده یک آبجکت اســـت که توســـط تابع آبجکت دیگر صدا زده می شود. در ق سمت های بعدی پردازش، هنگامی که برای کلا سها عملیاتی را تعریف می کنیم، هر پیغام تبدیل به یک عملیات

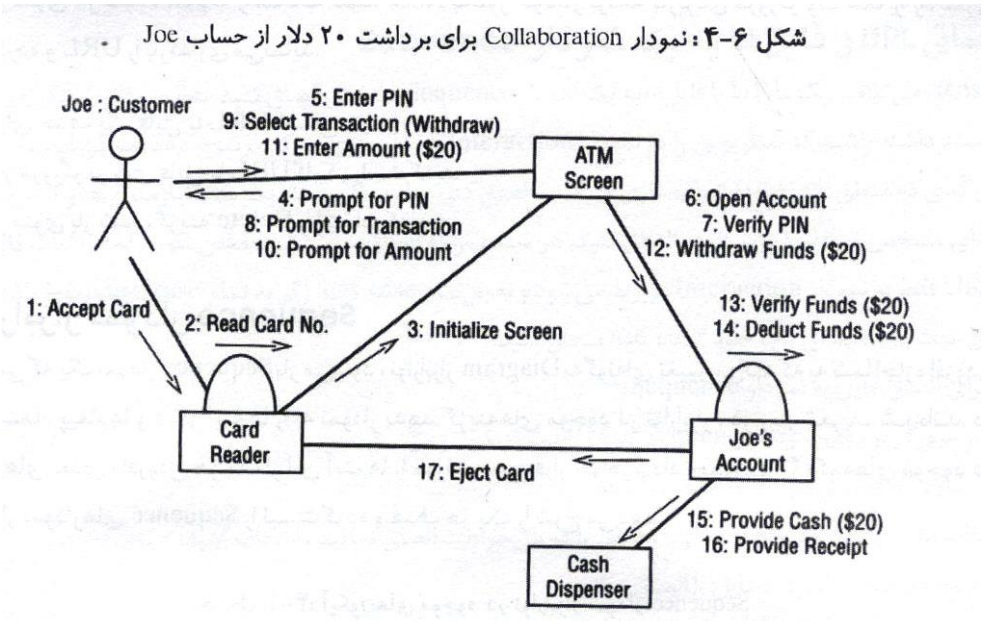
خواهد شد. پیغامها همچنین می توانند بازتابی با شند که نشان دهنده این خواهد بود که آبجکتی یکی از عملیات خویش را صدا می زند. در این مثال، پیغام شـــماره دو نشـــان دهنده این اســـت که کارت خوان از خودش درخواست خواندن شماره کارت را می کند.

نمودارهای Collaboration

مانند نمودارهای Sequence، نمودارهایCollaboration برای نشان دادن جریان در سناریوی مشخص یک use case، استفاده می‌وند. نمودارهای Sequence بر حســـب زمان منظم می شـــوند. نمودارهای Collaboration بیشتر بر روی رابطه بین آبجکت ها متمرکز می‌وند.

شکل۶-٤ یک نمودار Collaboration برداشت ۲۰ دلار توسط Joe از حساب را نشان می دهد.

شکل ۶-۴ : نمودار Collaboration برای برداشت ۲۰ دلار از حساب Joe



همانگونه که می بینید، اطلاعات موجود در نمودار Sequence بالا، در نمودار Collaboration نیز وجود دارد، ولی این نمودار، دید متفاوتی را از این روند ارائه می کند. دراین نمودار، مشاهده (رتباط بین آبجکت ها آسانتر است. با وجود این، مشاهده اطلاعات Sequence کمی مشکل است.

در بخش قبلی ، در این باره بحث کردیم که چگونه آبجکت ها بر روی یکدیگر اثر می گذارند تا توابع و عملیات یک سیستم را بدست بگیرند.

اکنون به خود کلاس ها نگاهی خواهیم انداخت و اینکه چگونه آنها را در بسته ها سازمان دهی کنیم. آبجکت ها درRose مطابق با کلاس‌ها در نمای Logical، مدل شده اند. کلاس ها، طرح ها کلی برای ساختن آبجکت ها می باشند. بنابراین حساب، کلاسی برای ساختن آبجکت حساب پس انداز Joe است.

اکنون به خود کلاس‌ها نگاهی خواهیم انداخت و اینکه چگونه آنها را در بسته‌ها سازماندهی می‌کنیم. آبجکت‌ها در Rose مطابق با کلاس‌ها در نمای Logical، مدل شده‌اند. کلاس‌ها، طرح‌های کلی برای ساخت آبجکت‌ها می‌با شند. بنابراین د ساب، کلا سی برای ساختن آبجکت د ساب پسا نداز Loe است. در این فصل در باره چگونگی ای جاد کلاس ها ونمودارهای Class در نمای Logical بحث خواهیم نمود.

نمای Logical(منطقی) یک مدل Rose

در این فصل، درباره تعدادی از آیتم ها بحث خواهیم کرد که در این نمای Logical یک مدل Rose ذخیره شـــده اند. می توان نمودارهای Sequence و Collaboration را در نمای Logical بسازید.

آیتم های دیگری را که می توانید به نمای Logical اضـــافه کنید شامل:

- کلاس ها
- نمودارهای Class
- نمودارهای use case
- صفات و عملگرها
- ارتباط ها
- نمودارهای State Transition

کار را باایجاد کلاس ها و نمودارهای class شروع می کنیم. در فصل بعدی جزئیاتی را به نمودارهای class اضـافه می کنیم که شـامل افزودن صفات و عملگرها، و رابطه های بین کلاس ها و بسته ها است.

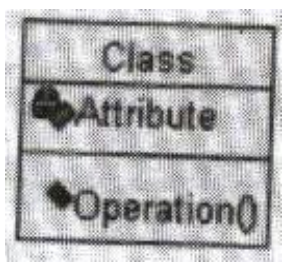
نمودارهای class

یک نمودار class برای نمایش تعدادی از کلاس ها و بســته های کلاس در سیستم شما استفاده شده است. این نمودار یک تصویر ایستا از قطعات سیستم و ارتباطات بین آنها را به شما می دهد. شما معمولا برای یک ســیســتم واحد چندین نمودار Class را ایجاد خواهید کرد. برخی از اینها زیرمجموعه ای از کلاس ها و روابط آنها را نمایش خواهند داد.

بقیه ممکن اســت زیر مجموعه ای از کلاس ها را نمایش دهند که شامل صفات و دیگر عملگرهای آنها می با شد. ممکن ا ست گروه دیگری فقط بســته های کلاس را نمایش دهند و ارتباطاتی که بین بســته ها وجود دارد. شما می‌توانید بسیاری از نمودارهای Class را که نیاز دارید، بسازید تا یک تصویر کاملی از سیستم خود را بدست آورید.

مثلا در یک سیستم شخصی، ممکن است کلاسی به نام Employee داشته باشیم. این کلاس شامل اطلاعاتی مانند یک شماره استخدام، نام، آدرس و شماره تلفن خواهد بود. کلاس Employee همچنین تعدادی رفتار خواهد داشت. یک کلاس Employee می داند که چگونه یک کارمند را استخدام یا اخراج نماید، یا یک ترفیع به کارمند بدهد.

در UML، یک کلاس با استفاده از نماد زیر نشان داده شده است:



اکنون کار با صفات و عملیات را مورد بررسی قرار خواهیم داد. بحث را با نشان دادن اینکه چگونه می توان صفات را پیدا کرد، آنها را به مدل Rose افزود و جزئیاتی را برای صفت قرار داد، شروع خواهیم کرد.

سپس به بررسی چگونگی یافتن عملیات، افزودن آنها به مدل و افزودن جزئیات به این عملیات، خواهیم پرداخت. پس از این مرحله، به نمایش صفات و عملیات بر روی نمودارهای کلاس می پردازیم. نهایتا، چگونگی نگاشت عملیات به یک پیغام بر روی نمودار Interaction را بررسی خواهیم کرد.

استفاده از صفات

یک صفت قطعه اطلاعاتی مرتبط با یک کلاس است. به طور مثال، کلاس company(شرکت) ممکن است صفاتی به این شکل داشته باشد: نام، آدرس، تعداد کارکنان.

یافتن صفات

چندین منبع برای صفات وجود دارد. برای شروع، می توانید به مستند سازی use case نگاهی بیاندازید و در جریان رخدادها، اسامی را

پیدا کنید. برخی از اســـامی، کلاس یا آبجکت خواهند بود، برخی عامل و برخی دیگر نیز صفات می با شند. به طور مثال ممکن ا ست جریان رخداد بیان کننده این مو ضوع با شد: کاربر، نام کارمند شرکت، آدرس، شماره تلفن و شماره امنیت گروهی را وارد می کند و شما را آگاه می سازد که کلاس کارمند دارای صـــفات، نام (Name)، آدرس(Address) وSSN و تلفن(Phone) است.

یک جای مناســـب دیگر برای یافتن صـــفات، نگاه کردن به ســـند نیازمندیها می باشـــد. ممکن اســـت نیازهایی را بیابید تا اطلاعاتی که باید تو سط سیستم جمع آوری شود را به طور خلاصه در اختیارتان قرار دهد. هر قطعه اطلاعاتی جمع آوری شده باید صفتی در یک کلاس باشد.

آخرین راه، بررسـی سـاختا ربانک اطلاعاتی اسـت. اگر بانک اطلاعاتی شما به طور کامل تعریف شده ا ست، فیلدهای موجود در جداول، درباره اینکه چه صفاتی خواهید داشت، ایده خوبی به شما می دهند.

همیشـــه یک نگاشـــت یک به یک (one to one) بین جداول بانک اطـلاعـاتـی و کـلاس‌هـای Etity وجـود دارد. در مـثـال قـبـل، جـدول

کارمند(Employee) ممکن اســـت دارای فیلدهای نام، آدرس، SSN و تلفن باشـــد. کلاس مربوطه با نام Employee نیز دارای صـــفات، نام، آدرس، SSN وتلفن خواهد بود. باید بدانید که همیشـــه بین جداول بانک اطلاعاتی و کلاس ها، ذهنیتها و ســـلیقه‌های متفاوتی وجود دارد. به طور مثال، بانک های رابطه ای مستقیما از وراثت پشتیبانی نمی کنند. به هر حال، وقتی صفات را تعریف می کنید، ازاینکه هر کدام از آنها راه برگشتی تبدیل به یک نیاز را دا شته با شند، مطمئن شوید. این مو ضوع می تواند در حل مشـــکلات مربوط به قدیمی بودن یک برنامه کاربردی که اطلاعات زیادی که هیچ کس مورد استفاده قرار نمی دهد را تعقیب می کند، کمک کند.

هر نیاز باید یک جریان رخداد متعلق به یکuse case ، به یک نیاز مشخص و یا یک جدول بانک اطلاعاتی موجود، برگردانده شود. اگر بتوانید یک نیاز را ردیابی کنید، از مورد اســتفاده بودن آن برای مشــتری مطمئن نخواهید شـــد. این موضـــوع ممکن اســـت باعث کمی انحراف از برخی از روشهای قدیمی تر شود. اینکه ابتدا ساختار بانک اطلاعاتی را ساخته و سپس ســیســتم را در اطراف آن قرار دهید. ســیســتم و بانک اطلاعاتی را در یک

زمان و در حالی که هر دوی آنها را با نیازهای یکســان وفق می دهید، می سازید.

در حالی که صــفات را تعریف می کنید، با احتیاط آنها را به کلاســهای مربوطه تخصــیص دهید. یک صــفت باید دارای اطلاعات مربوط به کلاس باشــد.به طور مثال، کلاس کارمند (Employee) ممکن اســت دارای نام و آدرس با شد، ولی نباید تولیداتی که شرکت مربوط به این کارمند، تولید می کند را دربرداشــته باشــد. کلاس تولیدات (Products) مکان بهتری برای ذخیره اطلاعات مربوط به تولیدات خواهد بود.

مواظب کلاس هایی که صــفات زیادی دارند، باشــید. اگر کلاســی را بیابید که دارای تعداد زیادی صفت با شد، نشان دهنده این خواهد بود که کلاس باید به دو کلاس کوچکتر شکسته شود. اگر کلاس با بیش از ۱۰ یا ۱۵ صفت دا شته با شید، نگاه دقیقتری به کلاس بیاندازید، ممکن ا ست کلاس کاملا منطقی و موجه به نظر رســد- فقط از اینکه تمامی صــفات مورد نیاز باشند و حقیقتا به همان کلاس تعلق داشته باشند، اطمینان حاصل کنید.

برای کلاسهایی که صفات بسیار کمی دارند نیز به طور مشابه و با احتیاط عمل کنید. ممکن است باز هم کلاس از نظر شما منطقی و موجه باشد. به طور مثال ممکن است کلاسهای Control گرایش بیشتری به صفات کم داشته باشند. همچنین ممکن است نشان دهنده این باشد که یک یا دو کلاس باید با هم ترکیب شوند. اگر کلاسی با یک یا دو صفت داشته باشید، ممکن است ارزش این را داشته باشد که یک نگاه دقیقتری به این مساله بیاندازید.

گاهی ممکن است به اطلاعات و داده هایی برسید که ندانید آیا صفت هستند و یا کلاس. به طور مثال، به صفت نام شرکت (company Name)، توجه کنید. پرسشی که ممکن است با آن روبرو شوید این است: آیا نام شرکت یک صفت برای کلاس Person است یا Company، برای خودش یک کلاس مجزا است؟ پاسخ این پرسش بستگی به برنامه ای دارد که آن را می نویسید. اگر نیاز به نگهداری اطلاعاتی در مورد شرکت دارید

و رفتارها و عملکردهای خاصی مرتبط با شرکت وجود دارند، در این صورت ممکن است کلاس شرکت، یک کلاس مخصوص به خود داشته باشد. به طور مثال، ممکن است سیستمی بسازید که ردوپا و نشانه های مشتری را در برگیرد، در این حالت، باید برخی اطلاعات درباره شرکتهایی که تولیدات را به آنها می فروشیم و یا به آنها خدمات دهی می کنیم را نگهداری نماییم.

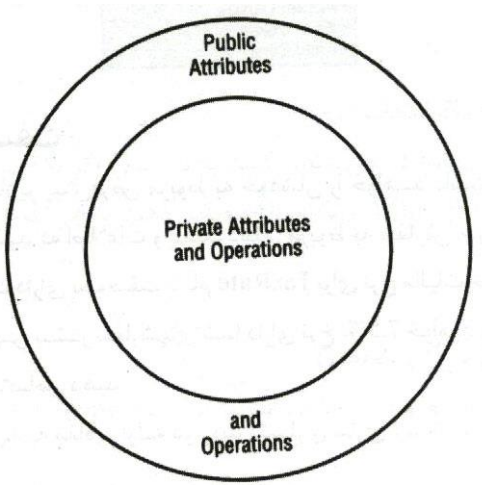
ممکن است بخواهید اطلاعاتی را درمورد تعداد کارمندان شرکت، نام و آدرس شرکت، نوع رابطه ای که با شرکت دارید،و غیره نگداری نمایید. از طرف دیگر، ممکن است به اطلاعات خاص و شاخص در مورد شرکت نیازی نداشته باشید. ممکن است برنامه ای بنویسید که برای تماسهای شما در دیگر سازمانها، نامه می نویسد. هنگام ساخته شدن نامه، حتما به نام شرکت نیاز خواهید داشت، در حالی که به اطلاعات دیگری در مورد آن شرکت نیاز نخواهید داشت. در این حالت، می توانید نام شرکت را به عنوان صفت برای کلاسcontant (تماس) در نظر بگیرید.

موضـوع دیگری که باید به آن توجه داشـته باشـید این اسـت که آیا اطلاعات به د ست آمده از پا سخ، هیچ گونه رفتار وعملکردی دارد یا خیر؟ اگر company(شـــرکت) تاثیرها و عملکردهایی را در برنامه کاربردی شما دا شته با شد، بهتر ا ست به عنوان یک کلاس مدل سازی شود. اگر شـرکت رفتار و عملکرد خاصـی نداشـته باشـد بهتر اسـت به صـورت یک صفت مدل سازی شود.

تنظیم Visibility صفت

یکی از مر کزی ترین و اصـلی ترین مفاهیم در برنامه نویسـی شـی گرا، کپسـوله سـازی اسـت. هر کلاس دارای صـفات و عملیات، مقداری از رفتارها و عملکردها را در خود کپسوله می کند. یکی از مزیتهای این روش، داشتن قطعات کد کوچک موجود در خود است.

به طور مثال، کلاس Employee ، تمام اطلاعات و رفتارهای یک کارمند را دربر خواهد داشت.



صـــفات، درون کلاس‌ها و به صـــورت مخفی از دیگر کلاس‌ها قرار دارند. به دلیل کپسوله بودن صفات درون یک کلاس، باید تعریف کنید که چه کلاس‌هایی برای م شاهده و تغییر صفتها، اجازه د ستر سی دارندکه به عنوان Visibility صفت شناخته می شوند.

چهر گزینه برای Visibility یک صفت وجود دارد. بیایید به هر کدام از آنها یک مثال نگاهی بیاندازیم. ما یک کلاس با نام Employee که دارای صفات Address و یک کلاس با نام company داشتیم.

Public (عمومی): نشان‌دهنده این است که صفت برای تمامی کلاس های دیگر، قابل رویت اســـت. هرکلاس دیگری می‌تواند مقدار صـــفت را

مشـــاهده کند و یا آن را تغییر دهد. در این حالت، کلاسcompany می‌توانند مقدار صفت Address مربوط به کلاس Employee را مشاهده کند و یا تغییر دهد.علامتی که UML برای یک صفت Public در نظر گرفته، علامت به علاوه می دهد.

Private (خصـوصـی): نشـاندهنده این اسـت که این صـفت برای کلاســهای دیگر قابل رویت نیســت. کلاس Employee، مقدار صــفت Address را خواهد دانست و حتی می تواند این مقدار را تغییر دهد، ولی کلاس company نمی تواند صفتAddress را مشاهده و یا آن را تغییر دهد. اگر کلاس company نیاز به مشـاهده و یا تغییر Address داشـته باشد، باید از کلاس Employee بخواهد این صفت را مشاهده کرده و یا تغییر دهد که معمولا توسط عملیاتهای عمومی انجام می شود. علامتی که UML برای صفتPrivate در نظر گرفته، علامت منها می‌باشد.

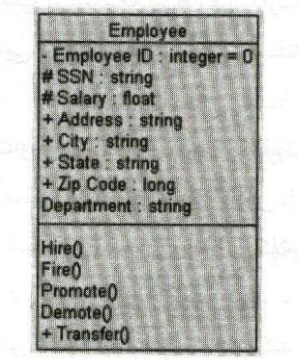
Protected (حفاظت شـده): نشـاندهنده این اسـت که کلاس و هر کدام از فرزندان آن به صفن می توانند دسترسی داشته باشند. فرض کنید در مثال ما دو نوع متفاوت کارمند وجود دارد، کارمندان با حقوق

ماهیانه و کارمندان با حقوق ساعتی. کلاسهای HourlyEmp(کارمند ساعتی) و SalariedEmp(کارمند ماهیانه) از کلاس Employee به وجود آ مده ا ند. اگر صـ فت Visibility متعلق به Address از نوع Protected باشد، می تواند توسط کلاس Employee، کلاس HourlyEmp و کلاس SalariedEmp، مشاهده و یا تغییر داده شود ولی تو سط کلاس company قابل م شاهده یا تغییر نخواهد بود. علامتی که UML برای صفتهای محافظت شده در نظر گرفته علامت # است.

Paccage or Implementation (بسته نرم افزاری یا پیاده سازی):

نشان می دهد که صفت حالت عمومی دارد، ولی فقط برای کلاسهای موجود در همان بسته. فرض کنید، در مثال ما صفتAddress برای کلاس Employee دارای Paccage Visibility باشد.Address می تواند فقط در صورتی که company وEmployee در یک بسته باشند، توسط company تغییر داده شود. در Visibility از نوع

Implementation(پیاده سازی) هیچ علامتی قبل از صفت وجود نخواهد داشت.



به طور کلی، Visibility از نوع Protected (محافظت شـــده) و از نوعprivate (اختصاصی) برای صفات توصیه می شوند. استفاده از این گزینه های دلخواه به شما کمک می کند که کنترل بیشتری بر روی کدها و صفات داشته باشید. با استفاده از صفتهای حفاظت شده یا اختصاصی، با حالتی که در آن یک صـفت توسـط همه کلاسـها در سـیسـتم تغییر یابد، برخورد نخواهید کرد. در عوض، منطق تغییر یک صـــفت به همراه آن صفت در یک کلاس منفرد، کپسوله شده است.

گزینه های مربوط بهVisibility که انتخاب خواهید کرد، کدهای تولید شـــده را تحت تاثیر قرار خواهند داد. به طور مثال، شـــکل ۳-۶ نشان‌هنده کدهای جاوایی است که برای کلاس بالا تولید شده است.

Rose از دو مجموعه نماد های مربوط به Visibility پشـــتیبانی می‌کند. اولین آن، نماد(# ، - ، +)UML به ترتیب برای صـــفحات عمومی، اختصـــاصـــی و محافظت شـــده می باشـــد. دومین نماد شـــامل چهار آیکون Rose مانند آنچه در جدول۱-۶ آمده، می باشد.

یافتن عملیتها

یافتن عملیتها حقیقتا بســـیار آســـان اســـت. در حالی که نمودارهای Sequence و Collaboration را می ســـازید، بیشـــترین کارهای مورد لزوم برای یافتن عملیاتها را انجام داده اید.

چهار نوع متفاوت از عملیاتها وجود دارند:

عملیاتهای مجری (Implementor)

عملیاتهای مجری، برخی وظایف تجاری را انجام می دهند.عملیاتهای مجری با تست نمودارهای Interaction یافت می شوند. نمودارهای Interaction بر روی وظایف تجاری تمرکز دارند، و هر پیغامی روی نمودار بیشتر باید به یک عملیات مجری نگاشت شود.

هر عملیات مجری باید دارای قابلیت بازگشت(تبدیل) به یک نیاز را داشته باشد. این کار توسط بخشهای متنوع مدل، میسر می شود. هر عملیاتی از یک پیغام بر روی نمودار Interaction به دست می آید

که این پیغام ابتدا ازنیازها به دست آمده و در use case قرار می‌گیرد و از use case در جزئیات جریان رخدادها قرار می گیرند و سپس روی نمودار Interaction قرار می گیرند. قابلیت بازگشت می‌تواند در مورد مطمئن شدن از اینکه،هر نیازی در کد پیاده سازی شده است و هر قطعه ای از کد بتواند توسط یک نیاز ردیابی شود، کمک می‌کند.

عملیاتهای مدیریتی (Manager)

عملیاتهای مدیریتی، ساخته شدن و خراب شدن آبجکت ها را کنترل می کنند. به طور مثال، عملیاتهای تولید کننده و خراب کننده یک کلاس، در این دسته قرار می گیرند.

عملیاتهای دسترسی (Access)

صفات معمولا یا اختصاصی و یا محافظت شده هستند. به هر حال، ممکن است کلاسهای دیگری وجود داشته باشند که بخواهند صفتهای یک کلاس خاص را تغییر بدهند. این کار توسط عملیاتهای دسترسی انجام پذیر است.

به طور مثال، اگر یک صفت در کلاس Employeeبا نام Salary داشته باشیم، نمی خواهیم که تمامی کلاسهای دیگر بتوانند Salary(حقوق) را تغییر دهند. در عوض، دو عملیات دسترسی را به کلاسEmployee می افزاییم، عملیات دسترسی GetSalary و عملیات دسترسی SetSalary. عملیاتGetSalary که عمومی است، می توانند توسط کلاس های دیگر صدا زده شود. این عملیات مقدار صفت Salary را به دست آورده و این مقدار را به کلاسی که آن را صدا زده، بر می گرداند.

عملیاتSetSalary هم که عمومی اســـت، به کلاس دیگر برای تنظیم صـــفت Salary(حقوق) کمک می کند.SetSalary می تواند هر قانون بررسی اعتبار برای حقوقی که باید قبل از تغییر مقدارحقوق، بررسی شود را در برگیرد.

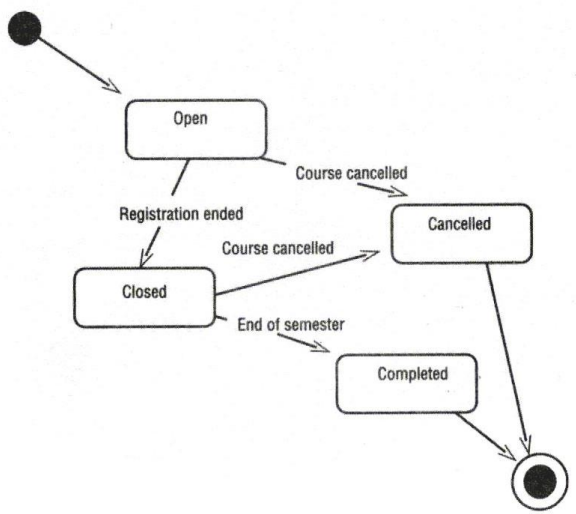
این روش باعث می شـــود که صـــفت با درجه امنیت بالایی در یک کلاس کپسوله شوند و از دسترسی کلاسهای دیگر در امان باشند، ولی باز هم اجازه دستر سی کنترل شده به صفات را می دهد. استاندارد در نظر گرفته شده به این صورت است که برای هر صفت در کلاس، یک عملیات Get و Set ساخته شود.

عملیاتهای امدادرسانی (Helper)

عملیاتهای امداد رســـانی، عملیاتهایی هســـتند که یک کلاس نیاز به نگهداری مسئولیتهای آن را دارد ولی کلاسهای دیگر نیازی به داشتن آن را ندارند. اینها عملیاتهای عمومی و محافظت شـــده یک کلاس هســـتند. عملیات امداد رســـانی نیز مانند عملیاتهای مجری با تســـت نمودارهای Sequence و Collaboration ظاهر می شوند.

نمودارهای تغییر حالت(State Transition)

یک نمودار تغییر حالت، چرخه زندگی یک آبجکت منفرد(از زمانی که آبجکت ایجاد می شـود تا زمانی که آبجکت از بین می رود) را نشـان می دهـد. این نمودارهـا روش خوبی برای مـدل کردن رفتـار و عملکرد دینامیکی(پویا) یک آبجکت می باشـند. در یک پروژه معمولی، برای هر کلاس، یک نمودار تغییر حالت ایجاد نمی کنید. در واقع خیلی از پروژه ها اسلا از آنها استفاده نمی کنند.



اگر یک کلاس دارید که چند رفتار و عملکرد مهم دینامیکی (پویا) دارد، ایجاد یک نمودار تغییر حالت برای آن مفید است. یک کلاس با رفتارها و عملکردهای دینامیکی قابل ملاحظه، کلاسی است که می تواند در حالات زیادی وجود داشته باشد. در مثال درس، یک آبجکت درس(Course) می تواند، باز، بسته، ملغی یا تمام شده باشد.

یک آبجکت سفارش(Order) می تواند در چندین حالت، درحال رسیدگی، انجام شده یا ملغی شده وجود داشته باشد. یک سفارش در هر کدام از این حالات به طور متفاوت رفتار می کند.

برای اینکه تعیین کنید که آیا یک کلاس رفتار ها و عملکرد های دینامیکی مهم دارد یا خیر، به صفتهای آن توجه کنید. بررسی کنید که چگونه یک نمونه از کلاس ممکن است با مقدارهای متفاوت در یک صفت، متفاوت عمل کند. اگر یک صفت به نام وضعیت(State) دارید، این می تواند نشانه خوبی مبنی بر حالات متنوع باشد. یک آبجکت چگونه با

مقدارهای متفاوت که به جای صـــفت قرار گرفته می شـــوند، رفتارهای متفاوتی ارائه می دهد؟

همچنین می توانید رابطه های یک کلاس را بررســـی کنید. به دنبال رابطه هایی با شید که multiplicity آنها صفر ا ست. صفرها ن شانگر این هستند که رابطه انتخابی است.

آیا یک نمو نه از کلاس وقتی رابطه وجود دارد یا ندارد، عملکرد و رفتار متفاوتی دارد؟ اگر تغییر می کند، ممکن اســـت چندین حالت داشـــته باشـــید. برای مثال، اجازه دهید به یک رابطه بین یک شـــخص(Person) و یک شرکت(Company) نگاهی دا شته با شیم. اگر یک رابطه وجود دا شته باشـــد، شـــخص در وضـــعیت اســـتخدام می باشـــد. اگر هیچ رابطه ای وجود نداشته باشد، شخص ممکن است اخراج یا بازنشته شده باشد.

فعالیت(Activity)

یک فعالیت، رفتارها یا عملکردهایی است که یک آبجکت وقتی در یک حالت خاص ا ست، اجرا می کند. برای مثال وقتی یک د ساب در حال ب سته شـــدن اســـت. کارت امضـــای صـــاحب حســـاب پاره می شـــود. یک فعالیت(activity) یک عملکرد وقفه ای اســـت. وقتی آبجکت در آن حالت اســـت،ممکن اســـت تا انتها اجرا شـــود یا ممکن اســـت به دلیل تغییر به حالت دیگر، دچار وقفه شـــود. یک فعالیت(activity) درون خود حالت ظاهر می شود و پیش از آن یک کلمه Do و یک دو نقطه(:) می آید.

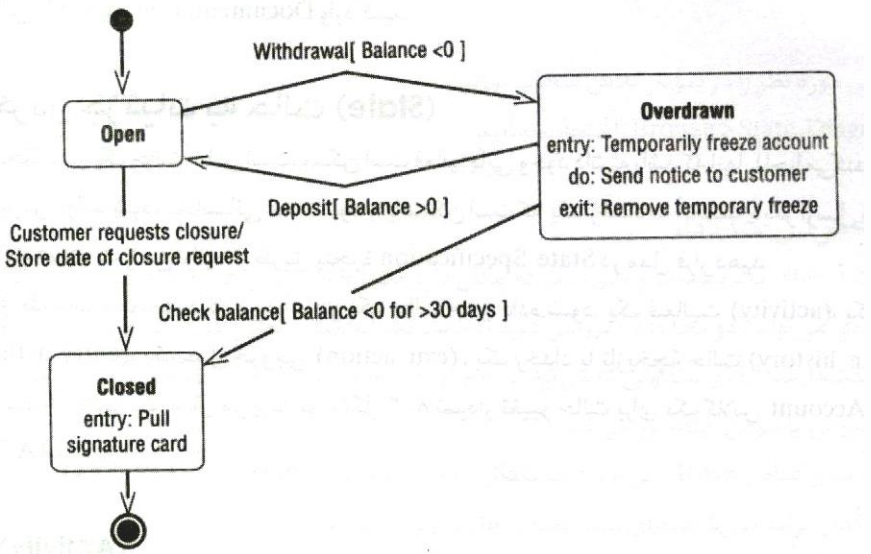
Action ورودی (Entry Action)

یک entry action یک رفتار و عملکرد اســـت که وقتی آبجکت در حال انتقال به حالت ا ست، اتفاق می افتد. به مثال د ساب (Account) توجه کنید.

وقتی یک حساب به حالت overdraw (برداشت اضافه) تغغیر می‌کند، بدون اهمیت به اینکه از کجا آمده است، عمل ورودی "Temporarily freeze account" اتفاق می افتد.

این عمل بعد از اینکه آبجکت به حالت overdrawn تغییر وضعیت داد، اتفاق نمی افتد بلکه به عنوان بخشی از انتقال حالت اتفاق می افتد. برخلاف یک فعالیت (activity)، یک عمل ورودی غیر وقفه ای در نظر گرفته شده است. یک عمل ورودی درون حالت ظاهر می شود و یک کلمهentry و یک دو نقطه (:) پیش از آن قرار می گیرد.

شکل ۲-۸: نمودار گذر برای یک کلاس Account در یک سیستم ATM



Action **خروج** (Exit Action)

یک exit action شبیه به یک entry action(عمل ورودی) می‌باشد.

اما یک action خروج هنگام خروج از یک حالت، به عنوان بخشـــی از انتقال اتفاق می افتد.

در مثال حساب(account)، وقتی یک حساب، وضعیت overdrawn را ترک می کند، بدون اهمیت به اینکه کجا می رود عمل خروج RemoveTemporary Freeze به عنوان بخشی از انتقال، اتفاق می‌افتد. مانند یک عمل ورودی(entry action) یک عمل خروجی نیز غیروقفه ای درنظر گرفته شده است.

یک عنل خروج درون حالت ظاهر می شود و قبل ازآن یک کلمه Exit و یک دو نقطه (:) قرار می گیرد. رفتار و عملکرد یک فعالیت (activity)، عمل ورودی(entry action) یا عمل خروج می تواند شامل فرستادن یک رخداد را به آبجکت های دیگر باشد.

برای مثال، آبجکت account ممکن است یک رخداد را به آبجکت cart reader(کارت خوان) بفرستد. درچنین موردی، یک^ قبل از فعالیت، action ورود یا action خروج قرار می گیرد. سپس نمودار به صورت زیر خوانده می شود:

Do:^Target.Event(Arguments)

رخداد(Event)

یک Event (رخداد) چیزی اســـت که اتفاق می افتد و باعث گذر از یک حالت به حالت دیگر می شود. در مثال account، رخداد Customer Reqests Closure باعث می شــود که حســاب از حالت باز (open) به حالت بســـته (close) برود. یک رخداد، در نمودار در طول فلش انتقال نشان داده می شود.

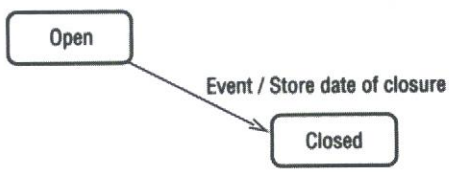
در نمودار یک رخداد می تواند با اســـتفاده از یک نام عملکرد یا به طور ســـاده با اســـتفاده از یک عبارت انگلیســـی کشـــیده شـــود. درمثال accountبه تمام رخدادها نامهای انگلیسی داده می شود. اگر به جای آن از عملیات(Operation) اســـتفاده کنید، رخداد Customer Reqests Closure ممکن است به صورت()Reqests Closure نوشته شود.

Action

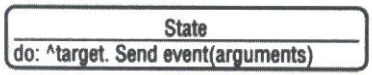
یک action، همانطور که در بالا ذکر شـــد، یک عملکرد غیر وقفه ای اســـت که به عنوان بخشـــی از یک انتقال رخ می دهد. Actionهای ورود و خروج در درون حالت نشان داده می شود، چرا که تعیین می کنند هنگامی

که یک آبجکت وارد یک حالت می شـــود یا از حالت خارج می‌شـــود، چه اتفاقی رخ می دهد. با این حال، اکثر actionها، در طول خط انتقال نمایش داده می شوند، چرا که هر بار یک آبجکت وارد یک حالت شده و یا از یک حالت خارج می شود اعمال نخواهند شد.

برای مثال، هنگام انتقال از حالت Open به حالت Closed، action به نام Reqest Store Date of Clouser رخ می دهد. یک action در طول خط انتقال، بعد از نام رخداد نشان داده می شود و قبل از آن نیز یک کاراکتر / قرار می گیرد.



یک رخداد یا یک action ممکن است یک رفتار(behavior) باشد که درون آبجکت رخ می دهد یا ممکن اســت فرســتادن یک پیغام به آبجکت دیگر باشــد. اگر یک رخداد یا action به آبجکت دیگر فرســتاده شــود، بر روی نمودار، قبل از آن یک ^ قرار می گیرد.



حالت آغازین(Start State)

حالت آغازین، حالتی اســت که وقتی آبجکت ایجاد می شــود در آن قرار دارد. در مثال accoun، یک حساب جدید درحالت Open شروع می شود. یک حالت آغازین در نمودار به وسیله یک دایره توپر نشان داده می شود. یک گذر از دایره به حالت آغازین کشیده می شود.

یک حالت آغازین اجباری اســت: کســی که نمودار را می خواند نیاز دارد تا بداند که یک آبجکت جدید درچه حالتی قرار دارد. فقط یک حالت آغازین می تواند در نمودار وجود داشته باشد.

حالت پایانی

حالت پایانی(Stop State) حالتی اســت که وقتی آبجکت خراب می‌شــود (از بین می رود) در آن قرار دارد. یک حالت پایانی در نمودار بوســیله یک دایره توپر با خطی به دور آن نشــان داده می شــود. حالات

پایانی انتخابی هستند و می توانید به هر تعداد که نیاز دارید، حالات پایانی به نمودار اضافه کنید.

برای اضافه کردن یک حالت آغازین:

۱. Start State را ازجعبه ابزار نوارابزار انتخاب نمایید.

۲. بر روی نمودار گذر حالت در جایی که حالت آغازین باید ظاهر شود، کلیک کنید.

برای اضافه کردن یک حالت پایانی :

۱. End State را از جعبه ابزار نوارابزار انتخاب نمایید.

۲. بر روی نمودار گذر حالت در جایی که حالت پایانی باید ظاهر شود ، کلیک کنید.

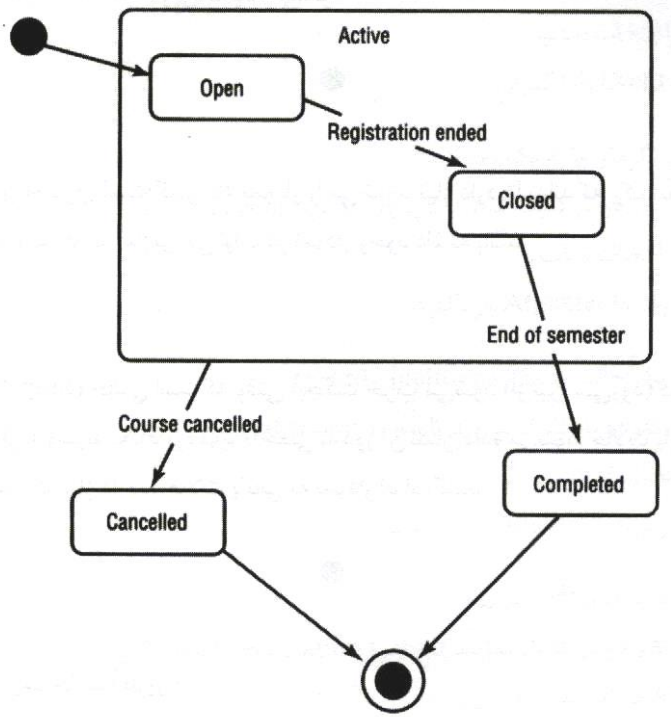
استفاده از حالات تو در تو (Nested State)

برای جلوگیری از بی نظمی در نمودار تان، می توانید یک یا چند حالت را درون یک حالت دیگر قرار دهید.حالات درونی به عنوان زیر حالت

(substate) شـــناخته می شـــوند و به حالت بزرگتر حالت مافوق (Superstate) گفته می شود.

اگر دو یا چند حالت یک گذر یکسان دارند، می توانند با هم یک گروه را تشکیل داده و یک حالت مافوق شوند. پس به جای به کارگیری دو گذر یکســـان (یکی برای هر کـدام)، گـذر می توانـد بـه حـالـت مـافوق (superstate) منتقل شود. حالت مافوق می تواند در کاهش شلوغی در نمودار تغییر حالت مفید باشد.

شکل ۴-۸ : نمودار انتقال حالت با حالات تودرتو



حال ممکن است نیاز داشته باشید تا سیستم، آخرین وضعیتی را که در آن بوده است به خاطر آورد. اگر سه حالت در یک حالت مافوق باشد، پس حالت مافوق را ترک کنید، ممکن است بخواهید سیستم جایی در درون حالت مافوق که در آنجا حالت مافوق را ترک کرده اید را به خاطرآورد.

دو راه وجود دارد که می توانید این مساله را حل کنید. اولی اضافه کردن یک حالت آغازین درون حالت مافوق است. حالت آغازین نشان می دهد که نقطه پیش فرض شروع در حالت مافوق کجاست و اینجایی است که اولین باری که آبجکت وارد آبجکت مافوق می شود در آن قرار دارد.

دومی استفاده از تاریخچه حالت(state history) است تا جایی را که وضعیت در آن بوده را به خاطر آورد. اگر گزینه تاریخچه در نظر گرفته شود، یک آبجکت می تواند یک حالت مافوق را ترک کرده و سپس درست به جایی که از آنجا وضعیت را ترک کرده است، بازگردد. امکان تاریخچه با یک "H" کوچک در دایره در گوشه نمودار نشان داده می شود.

برای تو در تو کردن یک حالت:

۱. state را از جعبه ابزار نوارابزار انتخاب نمایید.

۲. بر روی حالتی که قرار است در حالت جدید تودرتو شود، کلیک کنید

راه های ارتباطی با ما

www.meisampapi.blog.ir

meisampapi@gmail.com