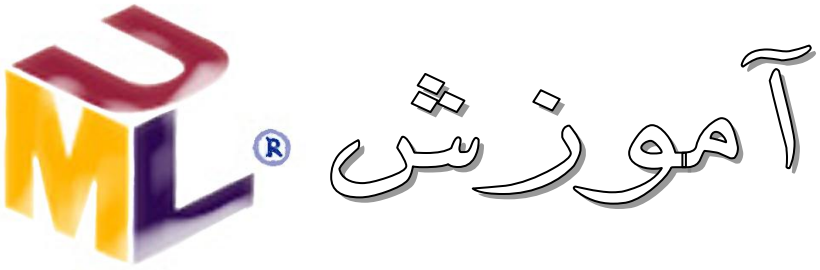


نویسنده : فرشاد حیدری

بر گرفته از سایت www. wanalyst.net

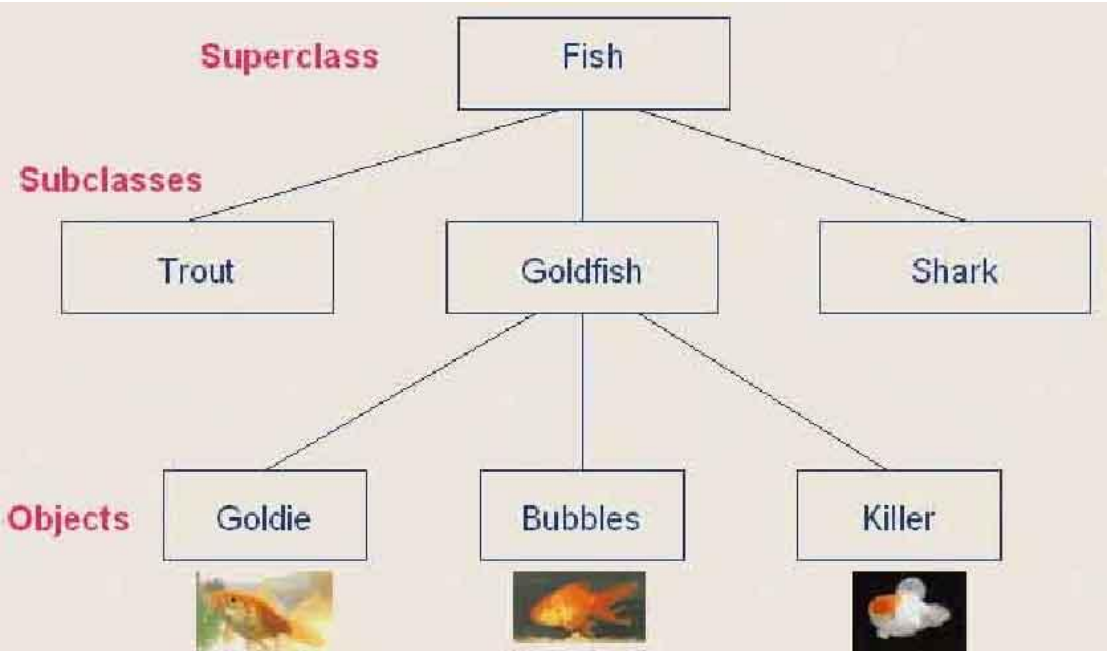
گردآوری : محمد حاجی عرب

به UML خوش آمدید

به UML خوش آمدید. UML مخفف (Modeling Language Unified) است که نشاندهنده یک زبان مدل سازی استاندارد برای طراحی سیستم می باشد. حالا معنی این حرف چیست؟ خوب، UML زبانی تصویری است که اجازه مدل سازی فرایند، نرم افزار و سیستم ها را به شما می دهد. UML چیست؟ UML از نمادها و دیاگرام ها ساخته شده است. نمادها و دیاگرام هایی که قصد داریم در این دوره با آنها کار کنیم. منظور از نماد، عناصری هستند که در یک دیاگرام با هم کار می کنند مانند المانهایی که برای دسته بندی داده ها، اشکالی برای اتصال داده ها، فلش ها، وابستگیها، نوشته هایی که شما ممکن است آنها را به نمودار خود اضافه و مقداردهی کنید و حتی چیزهایی مثل شبه کدها و کدها. همه این نمادها و دیاگرام های مطرح شده، عناصر تصویری یک فرایند، سیستم و یا بخش هایی از یک سیستم است. UML از کجا آمده است؟ UML مجموعه ای از ویژگی ها و استانداردهایی است که توسط شرکت OMG معرفی شده است. UML در دو بخش معرفی شده است، زیر ساخت و روساخت و نیز دو مشخصه مرتبط با آن که عبارتند از تبادل دیاگرام ها و زبان محدودیت شی . اگر مایل هستید نگاهی به این مشخصات بیاندازید می توانید آنها را از وب سایت شرکت OMG دانلود نمایید. UML توسعه پذیر است. یک زبان استاندارد صنعتی برای مدل سازی که همانطور که قبلاً اشاره شد شرکت OMG معیارهای رسمی را برای مشخص کردن آن فراهم کرده است. اما UML به گونه ای طراحی شده است که انعطاف پذیر باشد و این انعطاف پذیری یکی از نقاط قوت آن محسوب می شود. برای مثال شما می توانید با استفاده از قالب ها آن را گسترش داده و متناسب با نیازهای خود تفسیر کنید. تعریف رسمی UML بدین معناست که کاربران مختلف در یک صفحه یکسان قادر خواهند بود متناسب با نیازهای خود و موقعیتهای خاص نمادهای اصلی و معانی آن را گسترش دهند و همچنین این نکته نیز حائز اهمیت است که شما می توانید از UML همانقدر خوب برای مدل کردن یک موسسه بزرگ چند میلیون دلاری استفاده کنید که برای طراحی سریع فرایند یک کسب و کار کوچک استفاده می کردید. UML می تواند متناسب با نیاز شما هر قدر بزرگ یا کوچک باشد. این قابلیت انعطاف به شما اجازه می دهد که از UML برای چیزهای مختلف استفاده کنید. شما می توانید از UML برای مدل کردن فرایندهای کاری، نمایش ساختار برنامه ها، توصیف معماری و نمایش رفتار سیستم و مدل سازی ساختار داده ها استفاده کنید. و روشی که در آن شما می توانید با استفاده از UML در شرایط مختلف بهره ببرید، برای مثال شما می توانید به طرح ایده ها بپردازید و آنها را دستکاری کنید، روی تخته سفید (صفحه کاری برنامه) بکشید و پالایش کنید. شما می توانید از UML برای ساخت مشخصات یک سیستم و همچنین تولید کدهای برنامه نویسی استفاده کنید. شما می توانید از یک مدل UML به طور مستقیم به کد اجرایی بروید. در این دوره ما ده نمونه مختلف از دیاگرام های UML را پوشش خواهیم داد، اجازه دهید یک مرور سریع داشته باشیم. این دوره دیاگرام های Use Case را پوشش می دهد. دیاگرام های Use Case چشم اندازی از بازیگرانی که در تعامل با سیستم نقش های مختلفی را ایفا می کنند، نمایش می دهد. همچنین ما نگاهی به کلاس دیاگرام ها خواهیم انداخت که شامل تعریف کلاس ها و روابط ثابت بین آنهاست. کلاس دیاگرام ها انواع مختلفی از اشیا و ارتباط آنها با هم را در سیستم نمایش می دهند. ما دیاگرام های شی را هم پوشش خواهیم داد. این دیاگرام ها ساختار سیستم در یک بازه مشخص زمانی را نشان می دهند، در واقع دیاگرام های شی نمونه هایی از کلاس ها را در پیکربندی های مختلف مدل می کنند. به دیاگرام های بسته نگاهی خواهیم انداخت، دیاگرام هایی که عناصر مختلف را برای ایجاد یک دید سطح بالاتر از سیستم را در کنار هم متمرکز می کنند و همچنین به دیاگرام های حالت نگاهی خواهیم داشت که دیدی پویا از رفتار سیستم ها را به نمایش می گذارد. دیاگرام های فعالیت نیز یک دید پویا از سیستم را ارائه می دهند، این دیاگرام ها جریان کاری، فرایند کاری یا منطق رویه ای را مدل می کنند. به دیاگرام های ترتیب نگاهی خواهیم داشت که نوعی از دیاگرام های تعامل هستند. این دیاگرام ها چگونگی ارتباط اشیاء با تاکید بر زمان و ترتیب پیغام ها نمایش می دهند. نوع دیگری از دیاگرام های تعامل که نگاهی به آن خواهیم داشت نمودار ارتباط است. دیاگرام های ارتباط نیز روابط بین اشیا را نمایش می دهند. این دیاگرام ها بر چگونگی اتصال و ارتباط بین اشیا تاکید دارند. ما دیاگرام های ترکیب را هم پوشش خواهیم داد. این نوع دیاگرام ها یک دید کلی بر حسب عناصر تشکیل دهنده یک سیستم ارائه می دهند. یک دید پیمانه ای بلاک های ساختمان سیستم را نمایش میدهد و در نهایت نگاهی به دیاگرام های گسترش خواهیم انداخت، دیاگرام هایی که محصولات نرم افزاری به سخت افزاری را برای نمایش لایه فیزیکی سیستم ترسیم می کنند. انواع دیگری از دیاگرام های UML نیز وجود دارد اما به دلیل محدودیت های زمانی این ده دیاگرام رایج را در این دوره پوشش خواهیم داد. پس شروع می کنیم. از آنجا که UML از طراحی شی گرایی سیستم شروع شده است، ما با نگاهی سریع به مفاهیم شی گرایی شروع خواهیم کرد.

اشیا و شی گرایی :

ما قصد داریم در رابطه با برخی از عناصر اصلی شی گرایی یعنی شی (آبجکت) و کلاس صحبت کنیم. اجازه دهید با تعریف یک شی شروع کنیم. شی چیزی است که در زمینه یک سیستم وجود دارد و نمونه ای از یک کلاس مخصوص است. در عکس ما سیستمی به نام آکواریوم در نظر می گیریم که درون آن کلاسی به نام ماهی قرمز وجود دارد و درون این کلاس نمونه های منحصر به فردی از این ماهی خاص داریم.



همانطور که می بینید، کلاس به اشیایی تقسیم می شود که می توانند نمونه های منحصر به فردی از کلاس ماهی باشند و هر کدام از این نمونه ها خود نیز می توانند دارای نمونه هایی منحصر به فرد باشند. به علاوه اینکه یک کلاس می تواند قالبی باشد که اشیا از آن ساخته می شوند. می توانید اینگونه تصور کنید که کلاس همانند یک چاقوی برش کیک یا یک مُهر است. هنگامی که کلاسی را تعریف می کنید می توانید از آن برای ساخت اشیا کاملاً جدید استفاده کنید. کلاس ها و اشیای درون آنها عناصری دارند که آنها را تعریف می کنند. به این عناصر مشخصات یا خصوصیات گفته می شود. برای مثال در کلاس ماهی قرمز بعضی از مشخصات عبارتند از اندازه، رنگ یا جنسیت. کلاس ها همچنین دارای عملیات یا رفتارهایی می باشند که مشخص می کنند شی چه کارهایی را انجام می دهد یا چه کارهایی روی آنها انجام می شوند. برای مثال در کلاس ماهی قرمز بعضی از رفتارها عبارتند از شنا کردن، خوردن یا خورده شدن. مفهوم وراثت هنگامی که با کلاس ها کار می کنید بسیار مفید است زیرا هر زیر کلاس مشخصات کلاس بالاتر از خود را به ارث می برد. در اینجا ما سلسله مراتبی از وراثت را داریم. در بالا کلاس اصلی، کلاس ماهی است و تمامی خصوصیات و رفتارهای این کلاس در کلاس های زیرین یعنی کلاس های ماهی قزل آلا، ماهی قرمز و کوسه نیز وجود دارد (در کلاس های پایین به ارث برده شده است) ولو اینکه به طرق مختلفی با یکدیگر تفاوت داشته باشند. درون کلاس ماهی قرمز اشیا منحصر به فردی در حال شنا کردن در آکواریوم هستند که ما به آنها ماهی طلایی، ماهی حبابی و کیلر می گوییم. این ماهی ها تمام خصوصیات و رفتارهای که در ماهی قرمز وجود دارد را به اشتراک می گذارند. در واقع این ماهی خصوصیات و رفتارهای ماهی قرمز را به ارث برده اند.

شی گرایی ـ ارتباط ها

اشیا به تنهایی وجود ندارند بلکه در ارتباط با اشیا دیگرند. مثالی که شما می توانید این ارتباط را به طور واضح ببینید، این عکس از اتاق پذیرایی است.

اتاق پذیرایی شامل اثاث و مبلمان خود است که یک ارتباط را شامل می شود. اجزای تشکیل دهنده مبلمان و اثاث منزل در ارتباط با دیگر اجزای تشکیل دهنده آن در اتاق است که این نیز خود یک رابطه می باشد. همچنین هر بخش از مبلمان دارای ارتباطی معنادار با دیگر انواع مبلمان دارد. برای مثال میز قهوه یک نوع میز است. ای نوع از ارتباط مثالی از عمومیت دادن (تعمیم) است. در تعمیم، کلاس فرزند بر پایه کلاس والد قرار دارد. ما در اینجا یک کلاس والد به نام ماهی و دو کلاس فرزند به نام ماهی قرمز و کوسه داریم.

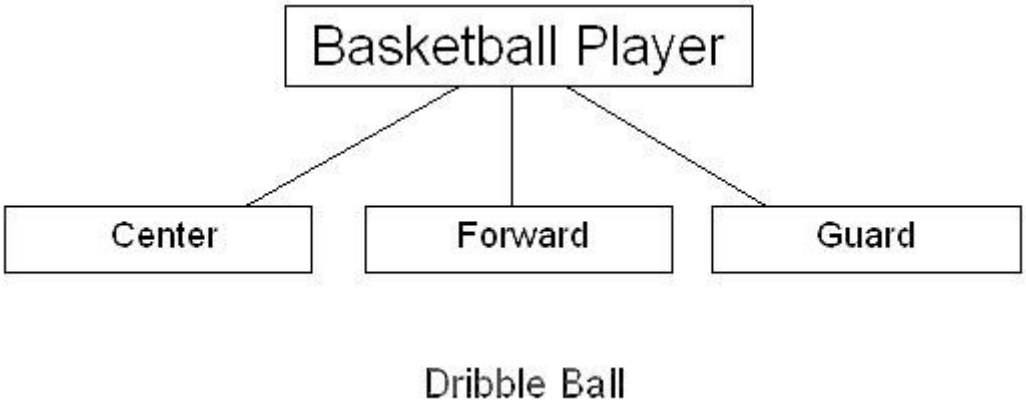
حال هر کدام از این کلاس های فرزند خصوصیات و رفتارهای کلاس والد را به ارث می برند. هر چیزی که برای خصوصیات ماهی ذکر شود در ماهی قرمز و کوسه نیز به طور مشترک وجود دارد. این در حالی است که ماهی قرمز و کوسه از نظر اندازه، اندام، روش های تغذیه با هم تفاوت دارند. لذا شما می توانید با استفاده از تعمیم ارتباط بین کلاس ها تشابه بین ماهی قرمز و کوسه را نشان دهید. وجه تشابه این دو این است که هر دو ماهی هستند حال آنکه دارای تفاوت هایی نسبت به هم نیز باشند. نوع دیگر رابطه، وابستگی نام دارد و بدین معناست که ارتباط بین دو کلاس به چند شکل می تواند باشد. برای مثال وقتی که بلندگوهای خود را به دستگاه پخش CD وصل می کنید، بین بلندگو و دستگاه یک وابستگی ایجاد کرده اید. وابستگی چند نوع است. نوع اول، پیوند و اتصال را نمایش می دهد. اطلاعات درباره یک کلاس به داده های کلاس دیگر متصل می شود. برای مثال اطلاعات شی صورتحساب به داده های مربوط به شی کارمندان متصل شده است. نوعی دیگری از وابستگی، همکاری را نشان می دهد. در این حالت یک کلاس با کلاس دیگر برای انجام برخی وظایف، کار می کند یا اینکه شاید فکر کنید چگونه یک کلاس روی کلاس دیگر کار می کند. برای مثال کوسه ، ماهی قرمز را می خورد. نوع دیگری از رابطه اجتماع (کل به جز) است. اجتماع رابطه بین یک کل و اجزایش را نشان می دهد. در شکل ما مجموعه ای از ماهی ها را می بینیم. این مجموعه از ماهی های (اجزای) منحصر به فردی ساخته شده است. این رابطه یک رابطه اجتماع است. ترکیب ، فرمی قوی از اجتماع است. در این نوع ارتباط هر جز ممکن است فقط به یک کل متعلق باشد. بر خلاف اجتماع که

در آن یک ماهی در یک مجموعه از ماهی ها که می تواند شنا کند و جزئی از یک مجموعه دیگر از ماهی ها شود، در ترکیب هر ماهی (جز) فقط متعلق به یک مجموعه (کل) در نظر گرفته می شود. مثلاً اتاق پذیرایی خود را تصور کنید. این اتاق جزئی از خانه شماست ولی جزئی از خانه همسایه نیست. جزئی از خانه هیچکس دیگر هم نیست. این اتاق نمی تواند با یک خانه دیگر به اشتراک گذاشته شود.

در رابطه ترکیب اگر کل از بین برود، اجزای آن نیز از بین خواهد رفت. همان طور که در شکل می بینید وقتی ساختمان خراب شود، اتاق ها هم که از بین می روند. (به طور کلی فرق بین رابطه Compostion و Aggregation در این می باشد که در رابطه Aggregation جز هم به تنهایی معنا دارد ولی در رابطه Compostion جز به طور مستقل نمی تواند ارائه شود.) یکی دیگر از مفاهیمی که در روابط باید به آن اشاره شود چند به چند بودن (چندگانگی) روابط است. چند به چند بودن تعداد اشایی که در یک رابطه می توانند سهیم باشند را نشان می دهد. در واقع تعداد اشیایی که می توانند در یک رابطه به خصوص به هم وابسته باشند را مشخص می کند. برای مثال هر درس یک کلاس را اشغال می کند که یک رابطه یک به یک را نشان می دهد. هر استاد می تواند چندین درس را ارائه دهد که یک رابطه یک به چند است. اما رابطه چند به چند چطور؟ این گونه تصور کنید که یک رابطه چند به چند ترکیبی از دو رابطه یک به چند است. برای مثال یک استاد می تواند به چندین دانشجو درس بدهد و یک دانشجو می تواند چندین استاد داشته باشد. لذا این دو رابطه ی یک به چند کمک می کند که به طور واضح تر رابطه بین استاد و دانشجو را یک رابطه چند به چند در نظر بگیریم.

چند ریختی:

چند ریختی ؛ مفهومی دیگر از شی گرایی. چند ریختی یعنی قابلیت استفاده به فرم های مختلف. چند ریختی می تواند روی اشیا و نیز روی رفتارهای آنها اعمال شود. بگذارید چند مثال بزنیم. یک شی چند ریخت، شی است که انواع آن در کلاس والد پنهان شده است. برای مثال کلاس بازیکن بستکبال را در نظر بگیرید. این کلاس شامل سه زیر نوع Center، Forward، Guard است که هر کدام از آنها خصوصیات، موقعیت، وظایف و استراتژی های بازی مخصوص به خود را دارند. اگرچه در حال صحبت کردن در رابطه با رفتار (عمل) دریبل کردن هستیم، این عمل برای هر سه زیر نوع یکی است. لذا هنگامی که درباره دریبل کردن صحبت می کنید، می خواهید منحصراً به کلاس بازیکن بسکتبال اشاره کنید. هنگامی که عمل دریبل کردن اجرا شد (به مرحله اجرا در آمد) شی بازیکن بستکبال به چندین فرم در خواهد آمد.

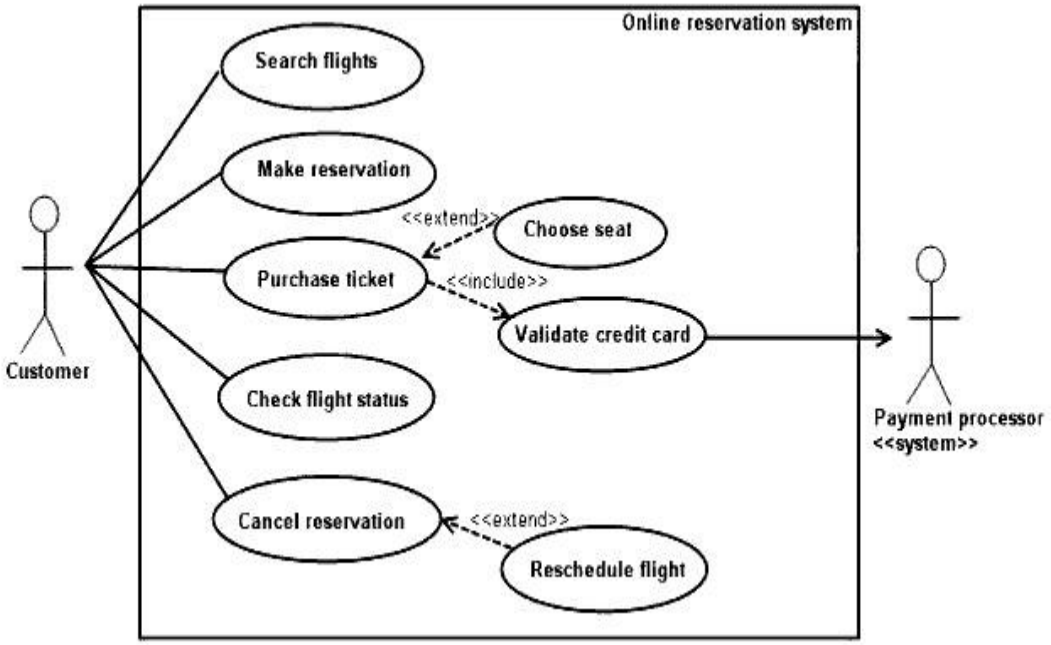


اما سوی دیگر چند ریختی رفتار چند ریخت است. در رفتار چند ریخت هر عمل می تواند بر مبنای رفتار تعریف شده در کلاس اصلی به روش های مختلفی انجام شود. به عبارت دیگر چند ریختی به شما این اجازه را می دهد که اعضای کلاس مشتق شده همانند کلاس والد رفتار کنند. این قابلیت به انواع مختلف اشیا این امکان را می دهد که به فراخوان های مختلف یک تابع پاسخ متناسب با آن فراخوان را بدهد. برای مثال ما در اینجا دو نوع از کلاس حیوانات را داریم. سگ و ماهی. برای مثال هر گاه از عمل نفس کشیدن صحبت می کنیم، عمل نفس کشیدن برای دو نوع کلاس متفاوت است. در واقع نفس کشیدن یک عمل چند ریخت است. نفس کشیدن در ماهی از طریق آبشش است و تولید حباب می کند اما نفس کشیدن در سگ از طریق شش به صورت تند نفس کشیدن است.

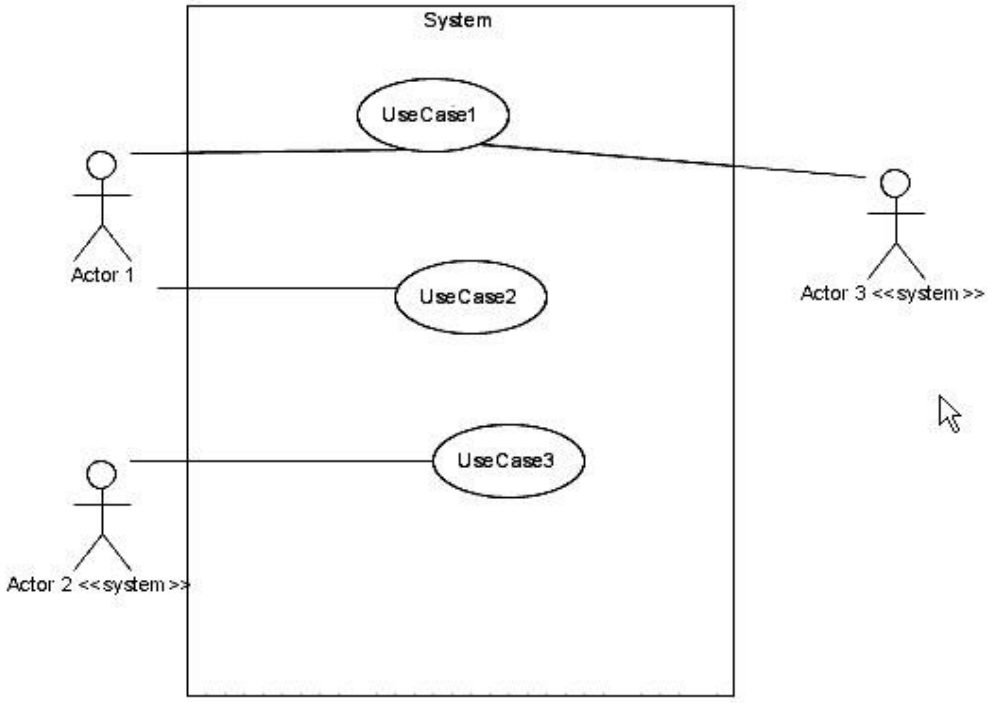
دیاگرام Use Case :

در این مقاله قصد داریم در رابطه با Use Case ها صحبت کنیم. نگاهی به مبانی Use Case و چگونگی ساخت دیاگرام های آن با UML خواهیم انداخت. سوال اول برای پرسش این است که Use Case چیست؟
Use Case کارکرد یک سیستم را نمایش می دهد. به عبارت دیگر Case Use به ما می گوید که یک سیستم باید چه کاری انجام دهد. Use Case تعامل بین بازیگران مختلف و سیستم را نشان می دهد. برای مثال ما از یک دستگاه خودپرداز استفاده می کنیم. برای رسم هر Use Case شامل دستگاه خودپرداز به یک بازیگر نیاز خواهید داشت و بازیگر کسی یا چیزی است که هدفی را در استفاده از سیستم دنبال می کند. برای مثال در سیستم خودپرداز بدیهی است که مشتری بازیگر اصلی است لذا یک بازیگر می تواند یک شخص یا یک سیستم یا سازمان دیگر باشد، به طور کلی هر کسی یا هر چیزی که در استفاده از سیستم هدفی را دنبال می کند. مفید خواهد بود که هر بازیگر را به عنوان یک نقش در نظر بگیرید نه یک شخص منحصر به فرد. ما گفتیم که بازیگر هدفی را دنبال می کند و منظور ما از هدف به طور ساده هر چیزی است که بازیگر می خواهد در تعامل با سیستم به دست آورد. برای مثال ما فرض می کنیم که مشتری قصد دریافت پول از خودپرداز را دارد. ما به این عمل، Use Case برداشت پول می گوییم. عنوان یک Case Use، بیان هدف بازیگر به فرم یک عبارت فعلی است. لذا برداشت پول عنوانی مناسب برای Use Case ماست. Use Case ها اهداف مختلفی را که بازیگران گوناگون در استفاده از سیستم دنبال می کنند، نشان می دهد. وقتی که به Use Case ها نگاه می کنید چیزی که باید انجام دهید این است که در رابطه با همه بازیگرانی که در تعامل با سیستم هستند فکر کنید و از خود بپرسید که این بازیگرها از سیستم چه استفاده ای می برند. حال Case Use ها متن های نوشته شده اند و شما آنها را در نیازمندیهای مشخص خواهید یافت. اما UML کجا وارد کار می شود؟ خوب، دیاگرام های Use Case در UML به صورت لیستی از Case Use های نوشته شده می باشند که تا لحظاتی دیگر به شما نشان خواهم داد. اما اجازه بدهید ابتدا راجع به Use Case های نوشته شده صحبت کنیم. Use Case های نوشته شده محدودیتی ندارند و در واقع شامل UML نمی شوند. به عبارت دیگر واقعاً از استانداردی پیروی نمی کنند. اما وقتی که Use Case خود را می کشید ایده ای دارید. ما می خواهیم یک نگاه سریع به آن داشته باشیم. همچنین برای هر Use Case نوشته شده چیزی که می خواهید انجام دهید، شامل گامهایی است که تعامل بین یک بازیگر و بازیگر اصلی در سیستم را شروع می کند. بازیگر اصلی بازیگری است که Use Case را شروع می کند. در مثال خودپرداز، بازیگر اصلی مشتری است، کسی که کارت خود را وارد دستگاه می کند. شما با یک سناریوی موفق شروع خواهید کرد چیزی که به طور غیر رسمی به آن سناریوی خوشحال (موفق) نیز گفته می شود و توضیحی است از مراحلی که بین بازیگر و سیستم اتفاق می افتد، به طوری که همه چیز از همان ابتدای Use Case یعنی هنگامی که مشتری کارت بانکی خود را وارد دستگاه می کند تا هنگامی که Use Case با موفقیت به پایان می رسد، همانطور که انتظار می رود، انجام شود و بازیگر پول را در دستانش داشته باشد. بنابراین مسیر موفق مشخص شد، شما قرار است که آن چیزی که در شرایط ایده آل و در دنیای ایده آل اتفال می افتد و بازیگر به هدف خود رسیده و با سیستم در تعامل است را توصیف کنید. اما از آنجا که همه ما می دانیم در دنیای ایده آل زندگی نمی کنیم لذا باید مسیرهای دیگری را نیز در نظر بگیرید. در اینجا دو نوع مسیر دیگر نیز وجود دارد که باید مورد توجه قرار گیرد. یکی استثنائات است. در هر مرحله در سناریوی موفق توقف می کنید و می پرسید که چه چیزی ممکن است در اینجا اشتباه شود. برای مثال هنگامی که مشتری بیشتر از مبلغ موجود در حساب بانکی اش در خواست کند چه اتفاقی می افتد؟ یکی دیگر از مسیرها، توسعه نامیده می شود. برای پیدا کردن توسعه ها باید از خود بپرسید چه اهداف دیگری ممکن است دنبال شود؟ برای مثال بعضی از دستگاههای خودپرداز این امکان را به شما می دهند که اگر مایل باشید پرینت رسید دریافت را به شما بدهند. شما مجبور نیستید هنگامی که پول خود را به صورت موفقت آمیز دریافت گردید، رسید هم بگیرید اما پرینت رسید انتخابی است که شما دارید. لذا Use Case پرینت رسید یک توسعه برای Use Case دریافت پول است. در اینجا مثالی از دیاگرام Use Case را مشاهده می کنید. این دیاگرام برای یک سیستم رزرو آنلاین برای مدیریت سفرهاست و من فقط می خواهم به المانهای اصلی اشاره کنم. ما در اینجا یک مستطیل داریم که سیستم را نشان می دهد و می بینید که به سیستم رزرو آنلاین نامگذاری شده است. بازیگران ما به شکل یک انسان نشان داده شده اند و همچنین نقششان در کنارشان نوشته شده است. ما مشتری و فرایند پرداخت را داریم که نحوه کار سیستم را نشان می دهد. ما همچنین Use Case هایی داریم که با بیضی نشان داده شده اند، لذا هر Use Case به همراه عنوانش درون یک بیضی که خود نیز درون یک مستطیل که نمایانگر سیستم است، قرار دارد. بنابراین ما بازیگرهایمان را داریم، Use Case هایمان را داریم و ما سیستم مان را داریم. ما همچنین ارتباط بین بازیگرها و Use Case ها و

نیز ارتباط بین Use Caseها را نیز نشان می دهیم. در ادامه دیاگرام های Use Case خود راترسیم خواهیم کرد.



در این مقاله ، ما به چگونگی ارائه عناصر پایه Use Case با استفاده از UML نگاهی خواهیم انداخت. حال وقتی به Use Case می پردازیم در واقع با سیستم سرو کار داریم، با بازیگرها، کسانی که از سیستم استفاده می کنند و با اهدافی که آن بازیگرها در تعامل با سیستم دنبال می کنند و ما این اهداف راUse Case می نامیم. برای نمایش سیستم ما یک مستطیل رسم می کنیم و به آن یک نام اختصاص می دهیم. در این مورد ما فقط آن را سیستم می نامیم. زیرا فقط به دنبال بیان مفاهیم هستیم.

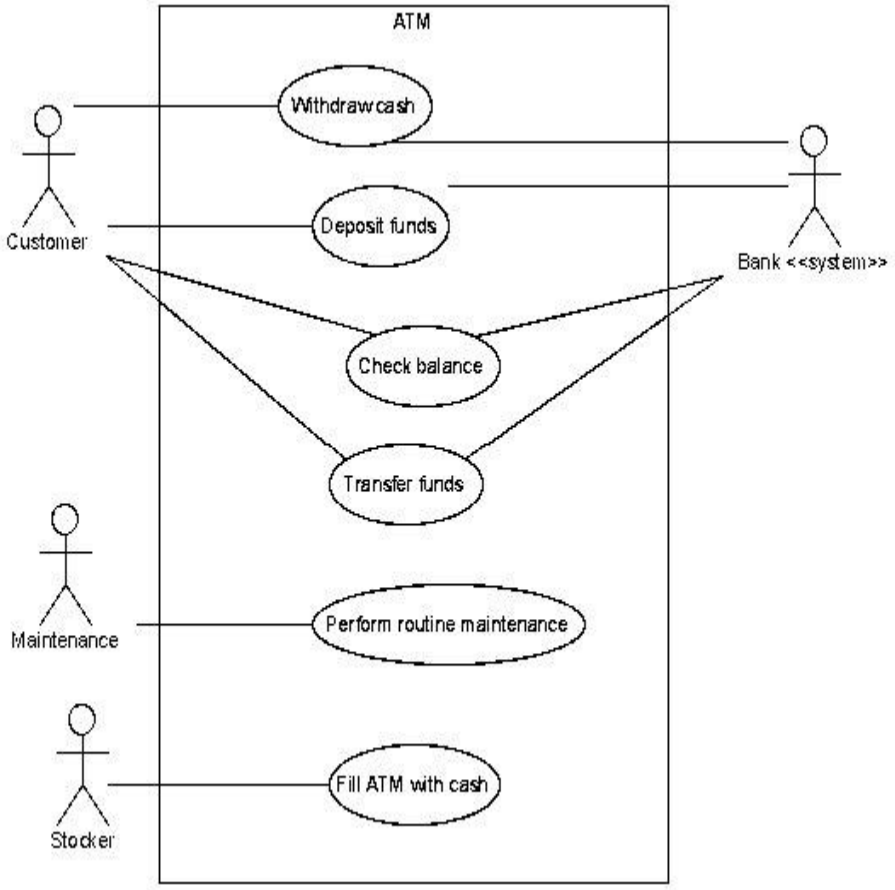


در حال حاضر در برخی از نمودارهای Use Case ممکن است شما افرادی را بیابید که مستطیل رسم نمی کنند، اما وقتی شما شروع به مشخص کردن بازیگرها و مشخص کردن Use Case های مختلف می کنید ترسیم مستطیل ساده مطمئنا یک ایده خوب محسوب می شود زیرا مشخص می کند که چه چیزی داخل ناحیه است و چه چیزی

خارج از ناحیه. اگر هدفی وجود داشته باشد که سیستم می تواند آن را تامین کند، به داخل مستطیل متعلق است و اگر این هدف، هدفی است که سیستم نمی تواند آنرا تامین کند به داخل مستطیل متعلق نیست. بنابراین استفاده از مستطیل کمک می کند تا شما ناحیه کاری را در ذهن داشته باشید. بعد باید بازیگرها را نشان دهیم. نمایش بازیگرها با استفاده از یک آدمک صورت می گیرد و شما هر جایی که از بازیگر انسانی یا سیستم دیگری صحبت شد از آدمک استفاده میکنید. اگر دوست دارید می توانید به آن برچسب سیستم به صورت <> بزنید تا مشخص کنید که بازیگر مورد نظری که در حال کار با سیستم شماست خود یک سیستم دیگر است. پس در حال حاضر دو بازیگر، دو آدمک و سیستم خود را داریم. بعد می خواهیم فکر کنیم این بازیگران در استفاده از سیستم چه اهدافی را دنبال می کنند. ما Use Case ها را با بیضی نمایش می دهیم. قرار دادن Use Case یک، Use Case دو و Use Case سه. بنابراین اینها اهداف مختلفی هستند که بازیگران ما ممکن است در استفاده از این سیستم داشته باشد. حالا این چیزها به صورت مجزا وجود ندارد، آنها با یکدیگر در ارتباط هستند و ما این ارتباط را با خط برای تعیین پیوند آنها نشان می دهیم. بنابراین فرض می کنیم که بازیگر 1، هدف شماره 1 (UseCase1) را دنبال می کند. ما بوسیله ترسیم یک خط نشان می دهیم که بازیگر یک از آن Use Case شروع به کار می کند و می توانیم بگوییم که بازیگر یک همچنین می تواند از Use Case دو آغاز کند و یا فرض کنیم بازیگر دو میتواند از Use Case سه آغاز کند. با این کار مشخص می شود که بازیگر شماره 1 بازیگر اصلی است، فردی که هدفش Use Case شماره یک است. هدف این بازیگر که بازیگر اصلی است،Use Case دو نیز می باشد و غیره. برای اینکه دیاگراممان تکمیل شود ممکن است یک Use Case را بیابید که با بازیگری دیگر در تعامل باشد. برای مثال ممکن است یک Use Case تایید کارت اعتباری به یک محاسبه کننده پرداخت برای تکمیل خود احتیاج داشته باشد. پس شما می توانید بازیگران ثانویه نیز داشته باشید، در این مثال آن را بازیگر سه می نامیم و وقتی که در مورد محاسبه گرها صحبت می کنیم، می گوییم این یکی سیستم دو است و اگر Use Case یک برای کامل شدن به مقداری مشارکت از سوی بازیگر سه احتیاج پیدا کرد ما بین آنها نیز یک پیوند برقرار می کنیم.

نمودار برای خود پرداز :

بگذارید نگاهی به مثال خودپرداز خودمان بیاندازیم تا ببینیم چگونه می توانیم Use Case های متفاتی را در سیستم داده شده در اختیار بگیریم. پس ما با رسم سیستم خود شروع می کنیم و آن را ATM می نامیم و یک بازیگر را که می خواهد با آن ATM در ارتباط باشد تعیین می کنیم و نام آن بازیگر را مشتری می گذاریم. پس یک مشتری چه اهدافی در ارتباط با سیستم ATM می تواند داشته باشد؟

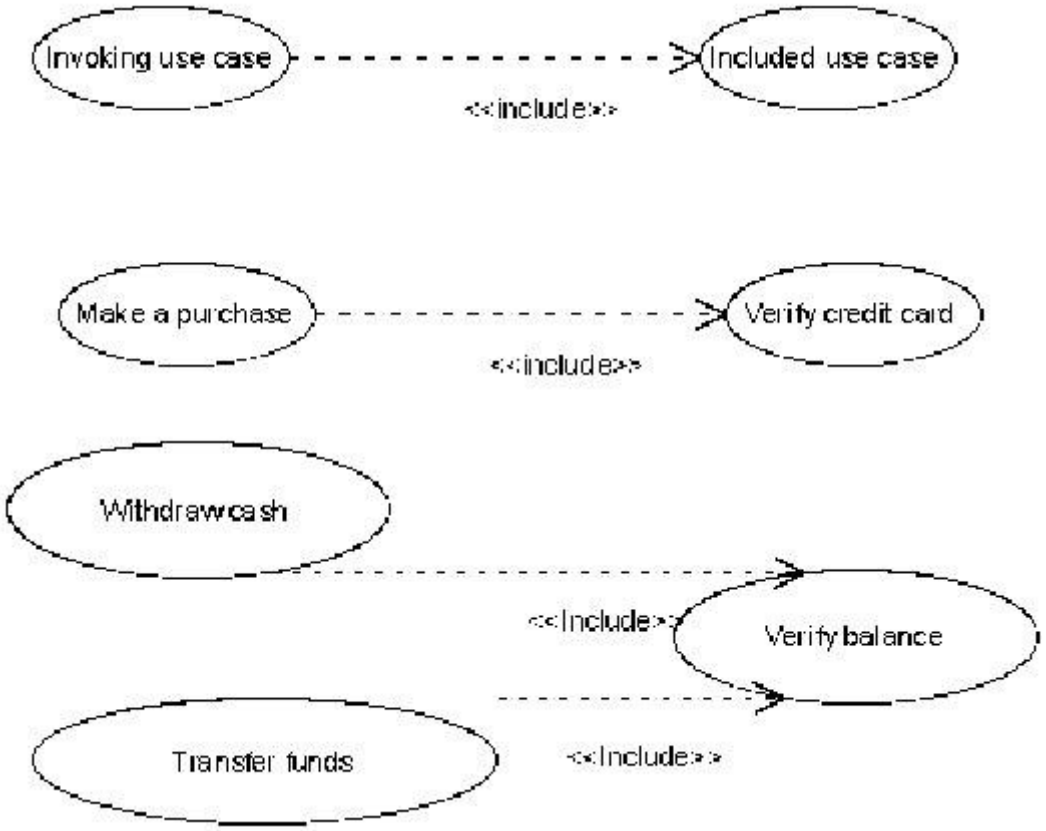


بگذارید ببینیم چه Use Case هایی را می توانیم تعیین کنیم، صندوق برداشت از حساب، مشتری ممکن است بخواهد پولی را به حساب بگذارد. مشتری ممکن است بخواهد صورت وضعیت حساب خود را چک کند و مشتری ممکن است بخواهد پول را جابجا کند. شما قادر خواهید بود تا کمی بیشتر فکر کنید ولی تا الان اینها برای منظور ما مناسب هستند. پس در هر یک از این موارد مشتری بازیگر اصلی است که این اهداف متفاوت در ارتباط با ATM را دنبال می کند. پس برای نشان دادن آن می خواهیم یک پیوند میان مشتری و هر یک از این Use Case های متفاوت را بر قرار کنیم. چه کس دیگری در ارتباط با ATM است؟ به عنوان مثال کسی هست که تعمیر و نگهداری بر روی ماشین را انجام می دهد. به عبارت دیگر هدفی که شخص نگهدار تاسیسات ممکن است در ارتباط با ماشین داشته باشد، نگهداری مستمر از آن است. حالا شخص نگه دار تاسیسات، مشتری نیست که آغاز کننده این Use Case باشد پس ما با ترسیم یک خط از نگهدار تاسیسات به Case Use نگهداری مستمر، پیوندی بین این بازیگر و Use Case مربوطش برقرار می کنیم. و همچنین وقتی شما در مورد این فکر می کنید که مشتری نمی تواند از حساب برداشت کند تا زمانی که شخصی پول را درون دستگاه قرار دهد، ما این شخص را نگهدار موجودی می نامیم و هدف نگهدار موجودی این است که صندوق دستگاه ATM را پر کند. و دوباره یک پیوندی وجود دارد که مشخص می کند این شخص برای این Use Case بازیگر اصلی است. حالا سیستم خود را داریم، ما سه بازیگر متفاوت و نیز اهداف مختلفی که در سیستم دنبال می کنند را مشخص کردیم. در قدم بعدی می خواهیم بپرسیم، که آیا بازیگر دیگری وجود دارد که نقش خاصی برای اتمام موفقیت آمیز یک Use Case داشته باشد؟ با نگاهی دقیق تر به Case Use های مشتری که عبارتند از برداشت از حساب، سپردن پس انداز، چک کردن موجودی حساب و غیره به این نتیجه می رسیم که برای هر یک از اینها نیاز به یک بازیگر ثانویه نیز داریم. ATM نیاز دارد که با سیستم بانک تعامل داشته باشد تا به هر یک از این اهداف دست پیدا کند، پس ما به ارتباطی بین این Use Case و این بازیگر ثانویه یعنی بانک خواهیم داشت. گاهی اوقات نیاز است تا بازیگرهای ثانویه را تعیین کنیم و گاهی اوقات

هم نیاز نیست. به نمودار خود نگاهی بیندازید و از خود بپرسید که چه کاری می خواهید انجام دهید، آیا این دیاگرام برای فهمیدن ساده است؟ آیا در هم ریخته نیست؟ آیا واضح است که مشتری آغاز کننده است؟ آیا واضح است که سیستم بانکی به همراه Use Case مربوطه اش در این سیستم نقش بازی می کنند؟ به نظر می رسد دیاگرام ساده ای که در این مثال رسم کردیم به اندازه کافی واضح است. یک ایده خوب برای خواناتر شدن دیاگرام، قرار دادن بازیگران اصلی در سمت چپ و قرار دادن بازیگران ثانویه در سمت راست است.

رابطه Include در نمودار UC :

در دیاگرام های Use Case شما ارتباط بین Use Case ها را از طریق وابستگی نشان خـواهید داد و قصد داریم درباره وابســتگی Include صحبت کنیم. بنابراین در دیاگـرام های Use Case، وابستگی Include یک ارتباط واجب بین دو Use Case را تعیین می کند. Use Case درخواست کننده بهInclude وابسته است تا کامل شود. اجازه بدهید موضوع را روشن تر کنیم، فرض کنید یک Use Case در خواست کننده دارید که جهت تکمیل شدن به یک Use Case ثانویه احتیاج دارد که آن را Included Use Case می نامیم. ارتباط بین این دو Use Case به وسیله یک فلش نقطه چین که Dependency Arrow نامیده می شود نشان داده می شود. بسته به نوع وابستگی شما می بایست فلش نقطه چین را نام گذاری کنید. پس ما آن را Include می نامیم. جهت این فلش همیشه جهت وابستگی را نشان می دهد.

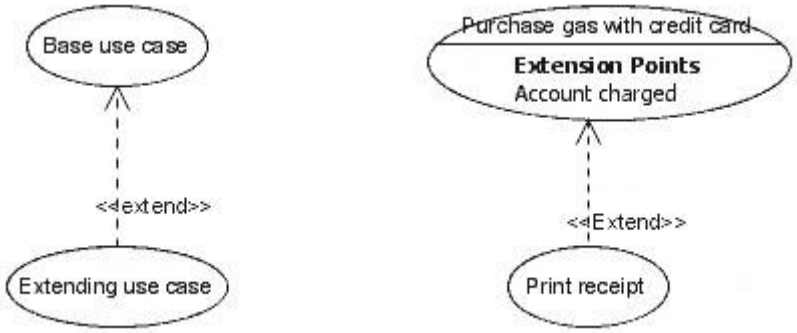


همانطور که در شکل می بینید Invoking Use Case به Included Use Case وابسته است. بگذارید برای این که واضح تر شود مثالی بزنیم. فرض کنید یک برنامه خرید online داریم و Use Case ما "انجام خرید" است، حالا برای این که Use Case انجام خرید تکمیل شود در یک جایی از سیستم باید کارت اعتباری خریدار چک شود . ما فلشی را بین آنها ترسیم می کنیم تا نشان دهیم Use Case انجام خرید به Use Case بررسی کارت اعتباری وابسته است، پس چیزی که این دیاگرام ها نشان می دهند این است که ، از Use Case انجام خرید تا دستیابی به هدفش ما باید Use Case بررسی کارت اعتباری را انجام دهیم. این موضوع این گونه در نظر داشته باشید که یک Use Case به Use Case دیگر احتیاج دارد، Use Case انجام خرید به Use Case بررسی کارت اعتباری احتیاج دارد. پس زمانی از وابستگی Include استفاده می کنید که مطمئن باشید Use Case درخواست کننده به Use Case درخواست شده احتیاج دارد. در هر جایی از عملیات می دانید که برای انجام خرید باید کارت اعتباری را بررسی کنید. در Use Case های نوشته شده بهتر است زمانی یک Use Case را Include کنید که مطمئن هستید توسط چندین Use Case دیگر

نیاز است. در مثال ATM شما باید صورت وضعیت را بررسی کنید قبل از این که مشتری بخواهد پولی را برداشت کند، پس Use Case برداشت پول Use Case بررسی صورت وضعیت را درخواست می کند. ولی Use Case های دیگر مثل جابجایی پول از یک حساب به یک حساب دیگر نیز نیازمند بررسی صورت وضعیت هستند. همان گونه که می توانید تصور کنید Include در نوشتن Use Case ها مراحل زیادی را اندوخته می کند. مثلا Use Case برداشت از حساب یا جابجایی پول از حساب، Use Case بررسی وضعیت را نیاز دارند. در این هنگام بررسی وضعیت را یک بار نوشته و سپس آن را به صورت Include به Use Case هایی که آن را درخواست میکنند وصل می کنیم. پس به عنوان نکته پایانی ، وقتی شما این نمودارها را ترسیم می کنید Use Case درخواست شده همیشه در سمت راست Use Case درخواست کننده ظاهر می شود. این قراردادی است وقتی که وابستگی Include را ترسیم می کنید در ذهن داشته باشید. در Use Case های نوشته شده اگر شما یک Use Case درخواست شده را به عنوان یک Use Case مجزا بنویسید و آن را توسط هر یک از Use Case های درخواست کننده Include کنید، در نوشتن گامها صرفه جویی کنید.

رابطه Extendدر نمودار UC :

نوع دیگری ارتباط بین Use Case ها وابستگی Extend (گسترش) نامیده می شود و وابستگی Extend یک ارتباط انتخابی بین دو Use Case را نشان میدهد. در این نوع وابستگی شما دو Use Case دارید ، ما یکی را Use Case پایه و دیگری را Use Case انتخابی می نامیم. و چیزی که این ارتباط نشان می دهد این است که Use Case انتخابی وضعیتی از Use Case پایه را بسط می دهد.



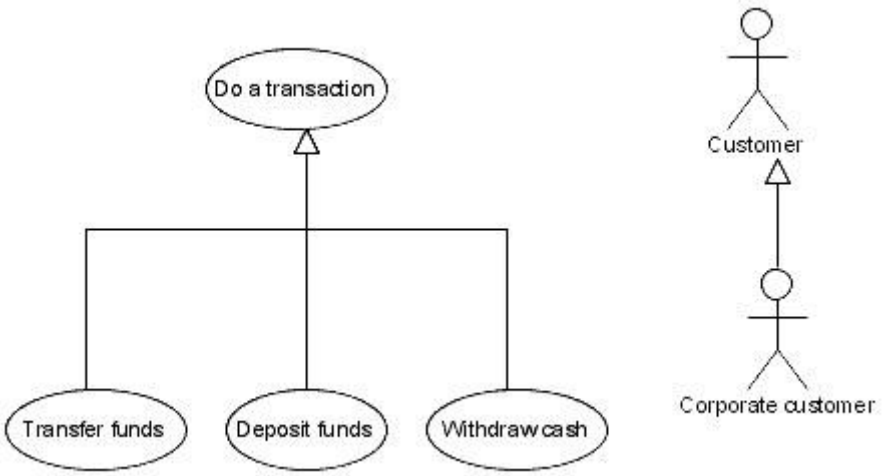
شما Use Case پایه خود و یک سری از مراحل که این Use Case را تشکیل می دهد را دارید. ولی گاهی اوقات در این مراحل، شما می توانید برای گسترش Use Case خود Use Case های دیگری را درخواست کنید. حالا به جهت فلش وابستگی در ارتباط Extend توجه کنید، بخاطر داشته باشید که جهت فلش جهت وابستگی را تعیین می کند. پس Use Case انتخابی به Use Case پایه وابسته است. همچنین وقتی در حال ترسیم وابستگی Extend هستید این قرارداد است که Use Case انتخابی در زیر Case Use پایه قرار بگیرد. پس همانطور که گفته شد Use Case انتخابی رفتاری از Use Case پایه را بوسیله اضافه کردن یک سری از مراحل به مراحل قبلی موجود در آن، گسترش می دهد. پس بگذارید به یک مثال نگاهی بیاندازیم ، تصور کنید در حال خرید گاز (بنزین) برای خودروی خود هستید و قصد دارید مبلغ آن را از طریق کارت اعتباری پرداخت کنید، ما آن را خرید گاز با کارت اعتباری می نامیم. در یک جایی از تراکنش به شما پیشنهاد داده می شود که رسید خود را چاپ کنید. برای تکمیل عملیات خرید شما نباید حتما رسید را چاپ کنید ولی شما این انتخاب را دارید. بنابراین خرید گاز با کارت اعتباری می تواند Use Case پایه باشد و چاپ رسید می تواند Use Case انتخابی باشد و همانگونه که گفته شد جهت فلش وابستگی از Use Case انتخابی به سمت Use Case پایه است. پس ما ارتباط Extend داریم. نگاهی به آنچه رخ داده بیاندازید، وقتی من فلش وابستگی را برای نشان دادن وابستگی Extend بین این دو Use Case اضافه کردم، فرصتی برای قرار دادن نقطه توسعه دارم و چیزی که این نقطه توسعه به شما می گوید این است که از کجا در Use Case پایه Use Case انتخابی نقش را بازی می کند. پس در این مورد این می تواند جایی باشد که حساب بار شده است. شما نباید حتما نقاط توسعه را وارد کنید، دیاگرام های Use Case فراوانی را می یابید که این را ندارند ولی مفید است که این را اضافه کنید. حالا Use Case انتخابی ممکن است به عنوان یکی از مراحل موجود در Use Case پایه رخ دهند، یا ممکن است به طور همزمان در مراحلی از Use Case پایه روی دهد یا ممکن است به صورت موازی روی دهد و یا حتی به صورت غیرهمزمان روی دهد. هر یک از آنها امکان پذیرند. وقتی در حال ترسیم دیاگرام های Use Case هستید استفاده از توسعه ها سودمند است اما ممکن است باعث در هم ریختگی دیاگرام شما هم شود، قرار دادن

اطلاعات زیاد باعث می شود خوانایی سخت تر شود. پس از خود بپرسید آیا دلیلی وجود دارد که ما بخواهیم توسعه ای را در اینجا قرار دهیم. در هنگام استفاده از گسترش ها مورد دیگری که باید مد نظر قرار گیرد این است که آیا این توسعه با توجه به تعریف ثانویه است یا خیر. در این مثال مهمترین چیزی که از سیستم می خواهید خرید شارژ گاز از طریق کارت اعتباری است، چاپ رسید چیزی است که بودنش عالی است، ولی این برای Use Case پایه ضروری نیست. پس هنگامی که زمانبندی پروژه فشرده است به Use Case های توسعه مثل یک کاندیدا نگاه کنید که می شود در پیاده سازی های آتی از آن بهره برد.

Generalizationدر نمودار UC :

در UML، تعمیم همانند مفهوم شی گرایی زیر کلاس است که ارتباط بین یک Case پایه و یک یا چند Case اختصاص داده شده به آن را نشان می دهد. لذا چیزی که به آن اشاره می کنیم در واقع نوعی از سلسله مراتب وراثتی است. ما یک Use Case اصلی پایه داریم و قصد داریم آن را والد بنامیم که شامل مجموعه مشخصی از رفتارها، محدودیت ها و فرضیات است. حال Use Caseهای تخصیص شده را فرزند 1 و فرزند 2 می نامیم.

شما می توانید بیش از یک فرزند نیز داشته باشید. هر کدام از Use Case های فرزند رفتارها، محدودیت ها و فرضیاتی که متعلق به Use Case والد است را به همراه بعضی که متعلق به خودشان است، دارند. بنابراین فرزند 1 نمونه ای از والد است. فرزند 2 نیز نمونه ای از والد است. اما تفاوت هایی نیز بین اینها وجود دارد که ایجاب می کند Use Case های خاص آنها را بنویسید. تعمیم روابط را با یک فلش نمایش دهید که از فرزند به والد کشیده شده است. حال این روشی خوب برای کشیدن دیاگرام هاست که آنها را تمیز و منظم نگه دارید و نیز راهی برای کم کردن تعداد فلش ها می باشد. حال شما می توانید با استفاده از ارتباطات تعمیم یافته نشان دهید که این دو Use Case فرزند Use Case والدند و از آن تعمیم یافته اند. اجازه دهید یک مثال ویژه دیگر بزنیم. یک دستگاه خودپرداز را به عنوان یک سیستم پایه فرض کنید و تراکنش را به عنوان Use Case والد در نظر بگیرید.

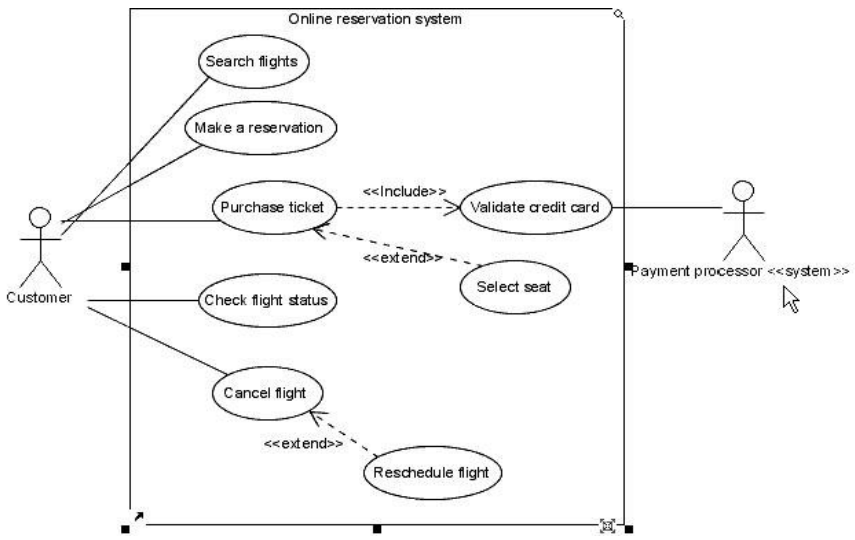


هرگاه که یک تراکنش را با دستگاه خودپرداز انجام می دهید، شما مجموعه از رفتارها، محدودیت ها و فرضیات خواهید داشت که وارد بازی می شوند. یک تراکنش خاص رفتارها، محدودیت ها و فرضیات خود را به اشتراک می گذارد، اما می تواند از این بیشتر مثل دریافت پول، به حساب گذاشتن پول، انتقال وجه و غیره نیز باشد. این برای نمایش اهداف کافی است بدانید که هر کدام از این اعمال یک تراکنش را انجام می دهند. برای مثال شما کارت خود را وارد دستگاه خودپرداز می کنید، شماه رمز را وارد می کنید اما برای هر کدام کارهای دیگری نیز انجام می دهید که Use Case مخصوص به خود را دارد. لذا ما قصد داریم که ارتباط تعمیم یافته شما را به نمایش گذاشته، آن را مرتب تر کرده و تعداد پیکانهایش را کاهش دهیم. شما می توانید رابطه تعمیم یافته را بین Use Case ها و بازیگرها نمایش دهید. برای مثال شاید شما یک بازیگر به نام مشتری داشته باشید که کارهای معینی انجام می دهد، محدودیت و مفروض های خاصی دارد که در حال ایفای نقش در این سناریو می باشند. همچنین ممکن است یک مثال ویژه تر داشته باشید که در آن بازیگری به نام مشتری حقوقی وجود داشته باشد. لذا هر چیزی که برای بازیگر مشتری حقیقی صدق می کند با اندکی تفاوت برای مشتری حقوقی نیز صدق می کند. برای مثال مشتری حقوقی ممکن است صدور صورتحساب ماهیانه یا قابلیت استفاده از سفارش خرید و غیره را داشته باشد و شما دوباره یک رابطه تعمیم یافته به همان شکل که برای Use Case ها نمایش داده اید را بین بازیگرها نیز نشان دهید. حال راهی

که با استفاده از آن می توانید یک رابطه تعمیم یافته را تشخیص دهید این است که از خود سوال کنید آیا فرزند نوعی از والد می باشد یا خیر، بدین معنی که آیا مشتری حقوقی آیا واقعا نوعی از مشتری است یا خیر؟ اگر جواب آری بود شما دارای یک رابطه تعمیم یافته خواهید بود. همین روال برای Use Case ها هم در نظر گرفته می شود. مثلاً آیا انتقال وجه یک نوع تراکنش است؟ بله. آیا دریافت وجه یک نوع تراکنش است؟ بله. و اگر جواب منفی است دیگر یک رابطه تعمیم یافته نخواهید داشت. ضمناً این مطلب نیز نشان داده شده است که چرا نمی توانید بین یک بازیگر و یک Case Use رابطه تعمیم یافته برقرار کرد. اگر فکر می کنید مشتری حقوقی یک نوع تراکنش است سخت در اشتباهید و ما می دانیم که این رابطه تعمیم یافته کار نخواهد کرد.

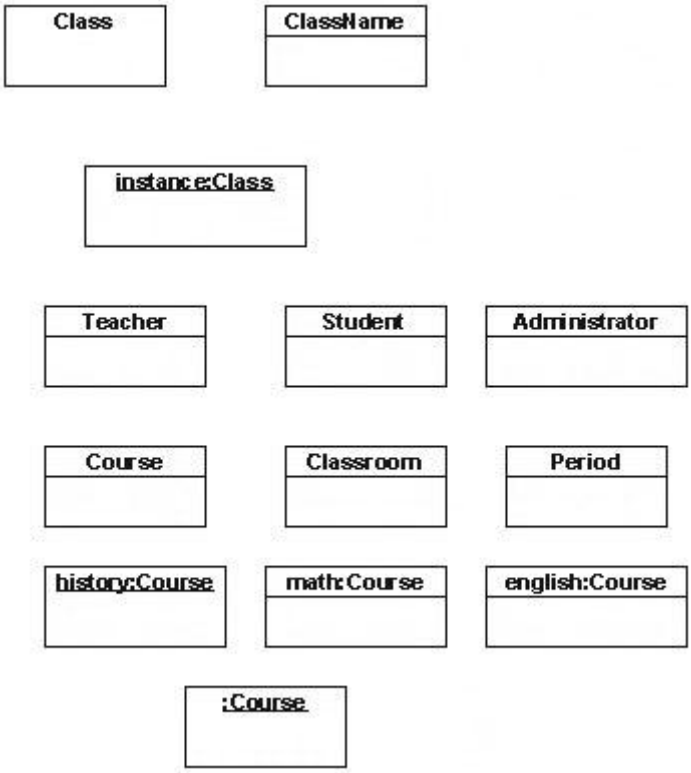
جمع بندی نمودار UC :

بیایید نگاهی دوباره به دیاگرام Use Case در مقاله قبلی بیاندازیم که از طریق آن می توانیم ببینیم چگونه این عناصر و روابط در کنار یکدیگر قرار می گیرند تا تصویر سریعی از اینکه سیستم چیست یا بعضی از قسمت های سیستم چگونه کار می کنند به ما می دهد. در شکل یک مستطیل داریم که سیستم ما درون آن وجود دارد و در بالا عنوان و نام را داریم، این یک سیستم رزرو آنلاین می باشد. در تعامل با سیستم در اینجا یک بازیگر که مشتری است داریم و مشتری با تمام این Case Use های مختلف در ارتباط است، هر یک از Use Case های داخل سیستم به شکل بیضی وجود دارند. پس مشتری بازیگر اصلی است که از طریق تمام این Use Case های مختلف، می تواند با سیستم در تعامل باشد، به عبارت دیگر اینکه این مشتری است که می خواهد برای پرواز جستجو کند، یک بلیط رزرو کند، وضعیت پرواز را چک کند و غیره، این مشتری است که شروع کننده است. در اینجا ما یک بازیگر ثانویه داریم، در این مثال بازیگر ثانویه یک سیستم پردازش دریافت و پرداخت است. حالا تفاوت بین بازیگر اصلی و بازیگر ثانویه این است که بازیگر اصلی کسی است که از Use Case شروع به کار می کند و بازیگر ثانویه از طریق Use Case نقشی را بازی می کند. پس برای مثال وقتی که Use Case "تایید اعتبار کارت اعتباری" اجرا می شود، "پردازشگر دریافت و پرداخت" به عنوان یک بازیگر نقش خود را شروع می کند. در کنار ارتباط بین بازیگرها و Use Case ها ما همچنین ارتباطاتی بین Use Case های حقیقی درون سیستم می بینیم و یک مثال خوب در اینجا Use Case خرید بلیط است. Use Case خرید بلیط شامل Use Case "تایید اعتبار کارت اعتباری" و همانگونه که شما وابستگی Include که به شکل پیکانی نقطه چین نمایش داده می شود را فراخوانی می کنید. پس اینکه Include در اینجا معنی ای دارد این است که برای خرید بلیط، برای اینکه Use Case کامل شود، ابتدا باید Use Case "تایید اعتبار کارت اعتباری" تکمیل شود. پس باید این اتفاق بیفتد قبل از اینکه Use Case خرید بلیط بتواند کامل شود. Use Case خرید بلیط همچنین یک ارتباط Extend با Use Case ای که درست در اینجا انتخاب صندلی است دارد. در این مثال نقاط پیکان وابستگی از Use Case انتخاب صندلی به سمت Use Case خرید بلیط است. ممکن است شما به Use Case های دیگر فکر کنید، ممکن است به راههایی فکر کنید که قادر خواهیم بود بعضی از این Use Case ها را توسعه دهیم ولی این را در ذهن داشته باشید که هدف شما این است که دیاگرام خود ساده را نگه دارید. دیاگرام همانگونه که قرار دارد ساده است، فهم آن ساده است و این شامل جزئیات خیلی زیاد که باعث در هم ریختگی دیاگرام بشود و باعث سخت تر شدن خوانایی آن بشود نیست.



دیاگرام Classدر UML :

دیاگرام های کلاس رایج ترین دیاگرام های UML می باشند. کلاس مقوله ای است که در آن اشیا می توانند طبقه بندی شده و یک قالب برای ساخت اشیا جدید نیز فراهم آورد. در واقع دیاگرام کلاس انواع کلاس و اشیا در یک سیستم و روابط بین آن اشیا را به شما نشان می دهد. کلاس ها دیاگرام های ایستا می باشند. آنها وجود و روابط را نشان می دهند نه یک عمل.

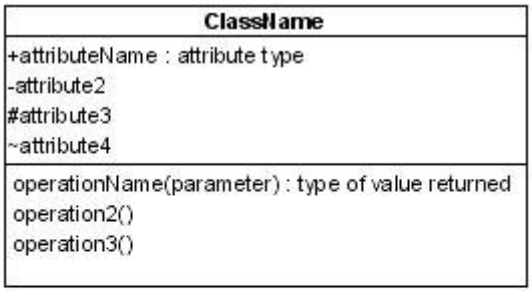


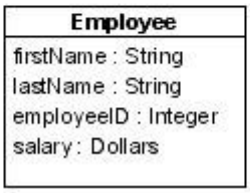
لذا چیزی که یک دیاگرام کلاس نشان می دهد، چگونگی کنارهم قرار گرفتن چیزهایی درون یک سیستم است. در این مقاله قصد داریم در رابطه با المان های پایه ای دیاگرام های کلاس یعنی کلاس ها و اشیا صحبت کنیم. برای نمایش یک کلاس از یک طبقه بندی کننده استفاده می شود که شما می توانید آن را به شکل یک مستطیل که از چندین قسمت مختلف تشکیل شده است ببینید. هر قسمت اطلاعاتی در رابطه با کلاس را نگهداری می کند. ما در قسمت بالایی نام کلاس، سپس خصوصیات و رفتارها که عبارت اند از ویژگی های کلاس، فیلدها، مشخصه ها، رفتار ها و غیره. لزومی ندارد که همیشه همه قسمت های یک کلاس را نشان داد لذا اگر تمایل داشتید که دیاگرامتان قدری خواناتر شد می توانید بعضی از این قسمت ها را مخفی کنید. چیزی که مهم است اینکه ما نام کلاس را می دانیم و برای معین کردن یک کلاس ما نام کلاس را در قسمت بالایی مستطیل می نویسیم. هر نام با حروف بزرگ شروع می شود. به صورت تو پر (Bold) نوشته شده و اگر نامی بیش از یک کلمه داشت همه کلمه ها را چسبیده به هم نوشته و ابتدای هر کلمه را با حرف بزرگ می نویسیم. حال کلاسها طبقه ای از اشیا هستند بدین معنا که هر شی نمونه ای از یک کلاس است، چیزی که متعلق به طبقه ای است که کلاس به نمایش گذاشته و یک شی در یک کلاس تمام حالات و رفتار را به اشتراک می گذارد. شما همچنین از یک طبقه بندی کننده برای نمایش شی استفاده می کنید که همان مستطیل است اما نام شی را به گونه ای متفاوت از نام کلاس می نویسیم. بدین صورت که شما ابتدا نام یک شی را نوشته سپس علامت : و در نهایت نام کلاس را می نویسیم. نام شی به حروف کوچک، سپس یک : نام کلاس با حروف بزرگ و زیر خط. اجازه بدهید مثالهایی از کلاسها و اشیا بزنیم. چه کلاس هایی را می توان در یک مدرسه نام برد، به عبارت دیگر مدرسه را به چه موضوعاتی می توان طبقه بندی کرد. یکی از کلاسها معلم، یکی دیگر دانش آموز و شاید کلاس مدیر. همچنین می توان کلاس درس و اتاق درس را هم مد نظر قرار دارد که البته این فقط یک شروع است. بدیهی است هنگامی که به این موضوع فکر می کنید ممکن است تعداد زیادی کلاس در سیستم وجود داشته باشد. برای مثال برای کلاس درس می توان اشیایی که متعلق به این کلاس اند مثل درس تاریخ، درس ریاضی و درس انگلیسی را تعریف کرد. می بینیم که این یک کلاس است زیرا به همان شکلی که طبقه بندی کننده ها تعریف می شوند تعریف شده است و از نام اشیا می

دانیم که این اشیا نمونه هایی از کلاس مذبور هستند. همچنین می توان اشیای بدون نام تعریف کرد بدین شکل که به جای نام در عنوان ابتدا یک : سپس نام کلاس را ذکر می کنیم.

خصوصیات و رفتار یک کلاس :

همان طور که دیدید یک کلاس، مجموعه ای از اشیا است که به آن تعلق دارند. هر کلاس از طریق خصوصیاتش مشخص (متمایز) می شود. اینها خصوصیات و رفتار هستند. این خصوصیات و رفتارها بوسیله اشیا درون کلاس به اشتراک گذاشته می شوند. در اینجا نیز از نمادهایی برای نمایش کلاس استفاده می کنیم. نماد کلاس یک مستطیل است که به چند قسمت تقسیم شده است. قسمت بالایی نام کلاس است که حتما باید نوشته شود. شما نیازی ندارید که هر 3 قسمت را نمایش دهید و می توانید فقط آنهایی که نیاز است را نشان دهید ولی می دانید که نام کلاس ضروری است مثل کلاس کارمند که در شکل می بینید. قسمت میانی خصوصیات کلاس رو نگهداری میکند. خصوصیات (Attributes) خواص (Properties) هستند. بویژه آنها خواصی هستند که یک سیستم برای تمام نمونه از کلاس خاص پیدا میکند. یک خصوصیت درست شبیه یک فیلد در طراحی پایگاه داده است. و در اینجا شما می توانید گرامر برای یک خصوصیت را ببینید. شما می خواهید نام خصوصیت را با حروف کوچک سروع کنید. اگر خصوصیت شما بیشتر از یک کلمه بود، شما باید کلمه ها را در کنار هم به کار ببرید و حروف اول هر کلمه را بزرگ بنویسد.در اینجا ما یک : داریم. که نوع داده ای خصوصیت را رشته، عدد، دقیقه و یا هر چیز دیگر مشخص می کند. شما نیازی به وارد کردن نوع داده ای خصوصیت ندارید. اما شما باید نام خصوصیت ها را وارد کنید. اطلاعات اضافه می تواند مفید باشد.



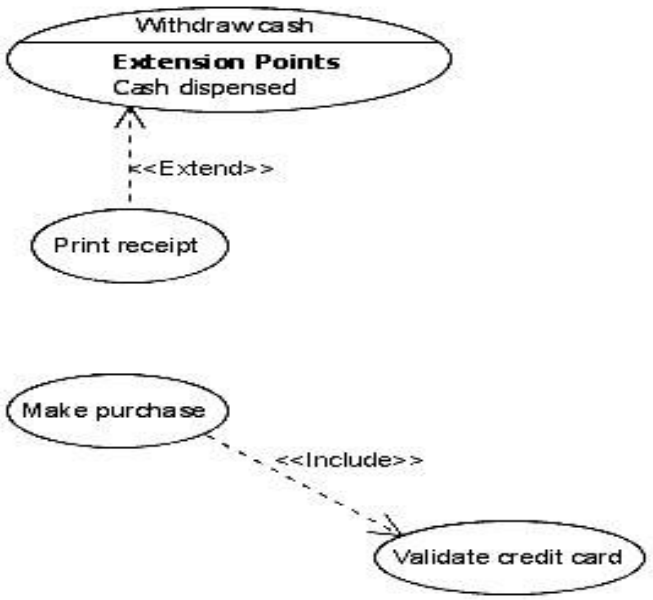


خصوصیت هایی که شما اینجا می بینید در هر خط به لیست شده اند و از کلاس کارمند ما استفاده شده است. بیاید نگاهی به چند مثال از خصوصیات بیندازیم. به ازای هر نمونه از کلاس کارمند، خصوصیات شامل نام که از نوع رشته، نام خانوادگی که باز هم از نوع رشته، شماره کارمندی از نوع عدد و دستمزد که از نوع داده ای پولی(دلار) می باشد. بنابراین در اینجا خصوصیت های مختلفی وجود دارد و برای طراحی پایگاه داده فیلد های مختلفی. کلاس ها همچنین شامل عملیات هستند. یک عملیات شامل چند تابع است که یک شیء میتواند انجام دهد یا می تواند توسط شیء پایان پذیرد.شما خواهید دید عملیات با توابع نشان داده می شوند. این میتواند برای فهمیدن آنها راهی مفید باشد. عملیات در قسمت پایینی کلاس می آیند و دارای گرامر زیر هستند : نام عملیات با حروف کوچک آغاز می شود، اگر نام آن بیشتر از یک کلمه داشته باشد، کلمه ها با هم ترکیب می شوند و حروف اول هر کلمه با حروف بزرگ نوشته می شود.بنابراین ما نام عملیات را داریم. درون پرانتز لیستی از پارامتر های ورودی (که در صورتی که بیشتر از یکی باشند) با ویرگول از هم جدا می شوند را داریم و در انتها نوع داده ای که برگردانده می شود. همانند خصوصیات، عملیات درون قسمت کلاس در هر خط لیست شده اند. درون کلاس کارمند چند عملیات ممکن آورده شده است. این عملیات عبارتند از استخدام شدن، تعیین مقدار حقوق که از نوع عددی ارسال می شود،گرفتن حقوق که مقداری عددی را باز می گرداند.بنابراین اینها توابعی هستند که کلاس می تواند انجام دهد یا می تواند توسط کلاس پایان پذیرد. یک چیز دیگر در مورد خصوصیات و عملیات این است که شما می توانید امکان دیدن (قابلیت دیدن) برای هر خصوصیت و عملیات تعیین کنید. شما ممکن بود خصوصیت و عملیات عمومی داشته باشید. المان عمومی به معنای آن است که المان می تواند توسط همه کلاس ها دیده شود. شما المان عمومی را با علامت مثبت (+) در روبروی نام المان مشخص می کنید. به طور مشابه یک المان می تواند خصوصی باشد.

المان های خصوصی فقط درون همان کلاس خودشان قابل دسترسی اند. المان های خصوصی با یک علامت (-) در روبروی نامشان مشخص می شوند. دو نوع دسترسی دیگر نیز وجود دارد: محافظت شده که با علامت (#) نمایش داده می شود این علامت نشان می دهد المان برای زیر کلاس های این کلاس قابل دیدن و استفاده می باشد و بسته بندی که با علامت (~) نمایش داده می شود و معنای آن است که تنها کلاس های موجود در یک پکیج می توانند المان را دیده و از آن استفاده کنند. برای پکیج های دیگر فقط عنوان آن نمایش داده می شود.

کلیشه کردن :

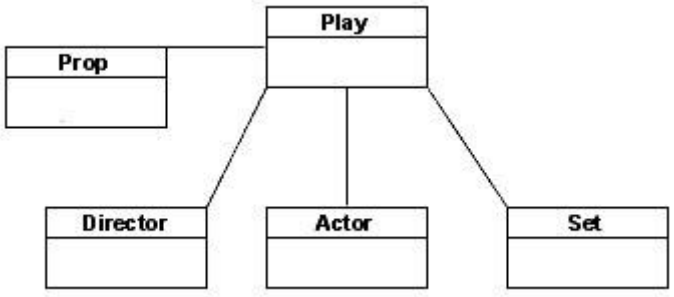
کلیشه کردن ، امکان گسترش UML را می دهد. گاهی اوقات زمانی که یک فرایند یا سیستم کسب و کار را مدلسازی می نمایید، ممکن است به عناصری احتیاج پیدا کنید که جزء UML به حساب نمی آیند اما شباهتهایی با عناصر اصلی آن دارند که با استفاده از آن (کلیشه کردن)، عناصر UML را برحسب نیاز گسترش می دهید. بنابراین کلیشه کردن، این امکان را می دهد که منظور خاص خود را بر UML اعمال کنید. زمانی که به کلاسها نگاه می کنیم، خواهیم گفت که کلیشه ی فوق از یک کلاس نتیجه میشود و منظور کلاس را می رساند و قواعد آن را با یکدیگر متمایز می سازد. مثلا همانطور که می بینید، 3 کلاس متفاوت داریم که هرکدام کلیشه و توضیحات خاص خود را دارند و چیزی که نشان می دهند این است که کلاسها با stereotype interface، کلاسها با enumeration stereotype و کلاسها با primitive stereotype قواعد متفاوت و خاص خود را در سیستم دارند. همه ی این مدلها، از عناصر کلاسها منتج می شوند اما هرکدام خصوصیات (properties) خاص خود را دارند که مناسب برای آن Stereotype می باشد. بنابراین کلیشه کردن اساسا اجازه میدهد که عناصر UML را اختصاصی یا سفارشی نمایید. همانطور که در شکل فوق می بینید، کلیشه ها با یک نام مانند نوع interface ارایه می شوند که داخل علامت << >> قرار می گیرند به آنها GMA گفته می شود. ممکن است آنها را به شکل علائم نقل قول (" ") در زبانهایی مثل فرانسوی یا ایتالیایی دیده باشید. خیلی شبیه به 2 براکت باز شده هستند و خیلی از مردم 2 تا براکت را با GMA عوض میکنند. پس اینها، نام کلیشه را دربر می گیرند و اطلاعاتی درباره کلاسی که اختصاصی شده، به شما میدهند. ممکن است کلیشه ها برای کلاسها در نمودارهای کلاس نتیجه گرفته شوند اما مفهوم کلیشه برای دیگر عناصرUML نیز بکار می رود. برای مثال، در Use Case ها، دیدیم که چگونه وابستگی ها می توانند برای نمایش نوع خاصی از وابستگی، کلیشه شوند. یک مثال این است که روابط Use Case،Extend ها را توسعه می دهند و یا نوع دیگری از وابستگی،Use Case دیگر را Include می کند.



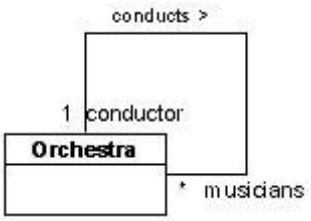
بنابراین کلیشه کردن می تواند برای انواع مختلفی از عناصر UML در هر زمان که نیاز دارید معنی یک المان خاص را فراتر از آنچه UML اجازه میدهد توضیح دهید، بکاربرده می شود. به خاطر داشته باشید که UML انعطاف پذیر طراحی شده است و طوری طراحی شده است که برای مدلهایی که طراحی می کنید، قابل گسترش باشد.

وابستگی و چندگانگی در کلاسها :

کلاسها فقط برای تعریف، وجود داشتن و قرار گرفتن در نمودارها نوشته نمی شوند. نوع های مختلفی از روابط را با یکدیگر دارند. خوب حال در مورد رابطه وابستگی صحبت خواهیم کرد. وابستگی رابطه ی بین دو کلاس را نشان می دهد. به بیان دیگر، اگر یک رابطه معنادار بین دو کلاس برقرار باشد می گوییم که این کلاسها وابسته هستند. معمولا وابستگی ها دلالت بر وجود یک رابطه دارند. یک مثال روشن می کند که مطالب گفته شده به چه معنی است. مثال را بدین گونه بیان می کنیم که یک کلاس داریم که آن را کلاس مدیر (Manager Class) و کلاس دوم را کلاس کارمند (Employee Class) می نامیم. اگر وابستگی این دو کلاس را با کشیدن خط بین آنها نشان دهیم، میتوانیم اینگونه برداشت کنیم که یک مدیر، یک کارمند دارد. همچنین می توانیم اینگونه هم بخوانیم که یک کارمند، یک مدیر دارد. خوب می توانیم این رابطه را در صورتی که نوع های بیشتری از یک حالت را بخواهیم، نامگذاری کنیم. پس می توانیم بگوییم "مدیریت" یا "manages". رابطه association را می توانید با فعل خاصی که نشان دهنده نوع Association هست، نامگذاری نمایید و مانند شکل زیر، جهت کارکرد آن را نمایش دهید. می توانید این کار را با روشهای مختلفی انجام دهید. می توانید از فلش در جهت نمایش فعل (manages>) به جای خط راست استفاده نمایید که معنی آن می شود، یک مدیر یک کارمند را مدیریت می کند. همچنین می توانید با یک نام واقعی آن را نشان دهید. و بهتر است که فعل مربوطه را در یک مثلث پر نمایش دهیم اما این ابزار اجازه همچین نمایشی را به من نمی دهد. پس فقط از علامت < برای نمایش جهت استفاده می شود که منظور را می رساند و این موضوع مهمی است. بنابراین همانطور که گفتم رابطه Association که داریم این است که یک مدیر، یک کارمند را مدیریت می کند. همچنین می توانید این رابطه را در جهت دیگر داشته باشید. یک کارمند، برای یک مدیر کار میکند. به شرطی که مشخص شود که این Association در چه جهتی کار می کند.

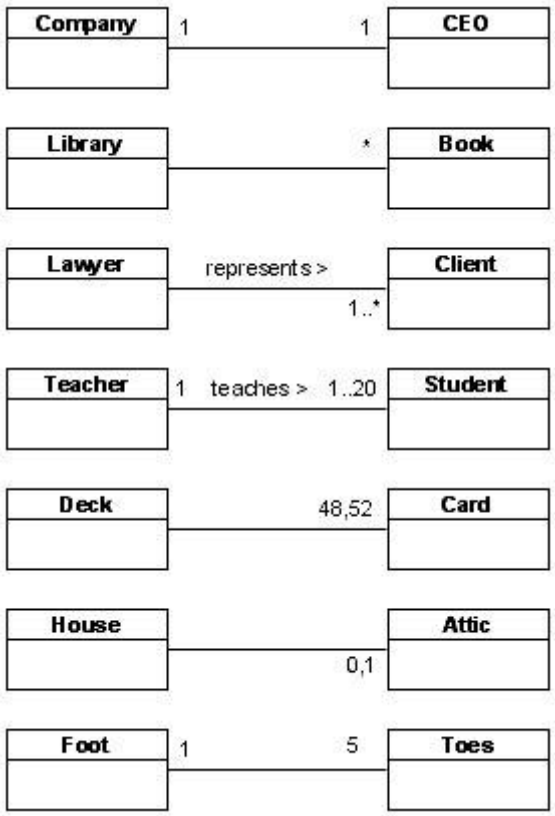


همچنین می توانید نمونه های پیچیده تری از یک کلاس داشته باشید. مثلا، کلاسی داریم به نام Play. در وابستگی با این کلاس، کلاس های کارگردان (Director)، بازیگر (Actor)، هماهنگ کننده (Set)، شاید کلاس پشتیبانی (prop) و ... هم داشته باشیم. می توانیم برای تمام این کلاسها رابطه association با کلاس Play داشته باشیم و اگر بخواهیم می توانیم به این Association ها نامی را اختصاص دهیم یا ندهیم. بازی، بازیگرانی دارد و بازیگران، بازی می کنند. بازی، هماهنگ کننده دارد و هماهنگ کننده وابستگی با بازی دارد و ....



پس این راه دیگری است که می توانید کلاسهای هم پیوند داشته باشید. می توانید چندین کلاس پیوسته با یک کلاس بخصوص داشته باشید. همچنین می توانید رابطه وابستگی بازتابی (Reflexive Association) یا رابطه وابستگی با همان کلاس داشته باشید. این زمانی اتفاق می افتد که اشیا در کلاس می توانند بیشتر از یک نقش را داشته باشند و مثالی که شکل آن را در زیر می بینید منظور من را توضیح می دهد. رابطه بازتابی را با کشیدن یک خط که از کلاس خارج شده و سپس مجدد به کلاس باز میگردد، نمایش می دهند و نقشها و عمل ها را جهت نمایش نحوه کارکرد رابطه پیوسته بازتابی نامگذاری می کنید. بنابراین کلاسی داریم به نام Orchestra که اعضایی دارد که ممکن است نقشهای متفاوتی داشته باشند. پس یک رهبر ارکستر داریم که تعدای نوازنده (musician) را در یک ارکستر رهبری می کند. حال مطمئنم متوجه شدید که عدد و ستاره ای که در شکل بالا مشخص است، نشان دهنده چندگانگی (Multiplicity) می باشد. چندگانگی نشان دهنده تعداد اشیایی در یک کلاس است که می

توانند با یک شی در کلاس هم پیوند، وابستگی داشته باشند. به عقب برمیگردیم و به مثال وابستگی کارمند و مدیر می رسیم. می توانید یک مدیر را به تعدادی کارمند پیوند دهیم که * نشان دهنده تعدادی (یا بیشتر از یک) است و می توانید در کنار کلاس مدیر 1 را قرار دهید که نشان دهنده "یک مدیر" است (الزامی نیست). بنابراین یک مدیر داریم که کارمندانی را مدیریت میکند. و نوع های مختلفی از چندگانگی داریم که می توانید در روابط وابستگی نشان دهید که برای شما ترتیب داده شده اند. می توانید رابطه یک به یک داشته باشید. یک کمپانی، یک مدیر دارد و همانطور که قبلا گفته شد، عدد 1 را بنویسید یا ننویسید مهم نیست. در مثال بعدی رابطه یک به چند نمایش داده شده است. یک کتابخانه، تعدادی کتاب دارد. بعلاوه می توانید اعداد مربوط به چندگانگی را بالا یا پایین خط قرار دهید، هرجا که حالت بهتری داشته باشد. در مثال بعد، یک وکیل داریم که یک یا تعدادی موکل دارد.

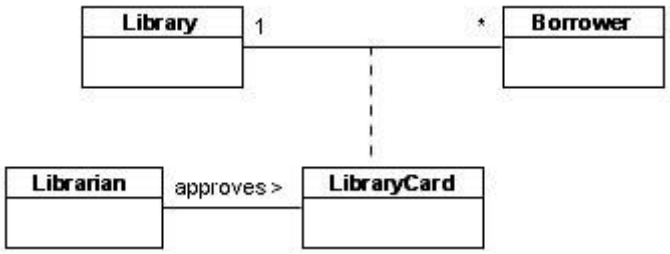


اگر بخواهید از یک تا تعداد بیشتری را نمایش دهید از علامت 1..* استفاده می نمایید. بنابراین یک وکیل، نماینده یک یا تعدادی موکل خواهد بود. حال اگر بخواهید بجای یک تا تعدادی خاص، بازه ای را نشان دهید می توانید این عمل را با علامت M..1 نمایش دهید. در مثال بعدی یک معلم می تواند از 1 تا 20 دانش آموز داشته باشد بنابراین اگر محدودیتی در کلاس باشد، یک معلم می تواند حداقل 1 و حداکثر 21 دانش آموز را تعلیم دهد. می توانید رابطه چندگانگی را نشان دهید که از یک شروع نشود. برای مثال یک دسته کارت می تواند از 48 تا 52 کارت داشته باشد که بستگی به نوع بازی دارد. مشابها می توانید 1 یا 0 را داشته باشید. مثلا یک خانه میتواند هیچ یا یک اتاق زیر شیروانی داشته باشد. و سرانجام، یک مثال از رابطه وابستگی؛ که در هر دو کلاس وابستگی وجود دارد. یک پا، 5 انگشت دارد. پس مثالهای مختلفی برای نمایش چندگانگی ارایه شد. می توانید رابطه 1 به 1 یا 1 به چند داشته باشید که رابطه 1 به چند، نوع بازه ای دارد و یا در هر طرف می تواند قرار بگیرد.

کلاسهای وابسطه :

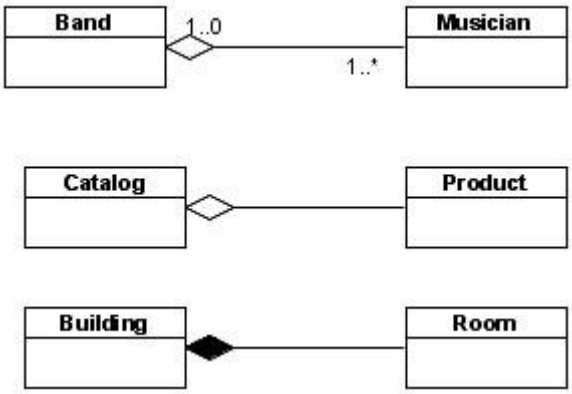
در تفکر در رابطه با وابستگی، شما ممکن است به این نکته توجه کنید که ارتباط بین کلاسها همواره روابط ساختارمندی نیست. در واقع یک وابستگی ممکن است ویژگی های خاص خود را داشته باشد. همانند کلاسها، وابستگی ها می توانند شامل خصوصیات و رفتارهایی باشند. ب رای نمایش وابستگی و همچنین ویژگی های مربوط به آن، شما می توانید یک کلاس وابستگی بسازید. یک کلاس وابستگی نمادی همانند نماد کلاس معمولی دارد اما با این تفاوت طبقه بندی کننده هایش با خط چین وصل می شود. لذا وابستگی که ما اینجا بین کتابخانه و عضو داریم، یک کلاس وابستگی است. یک عضو کتابخانه توسط خاصیت یک کلاس دیگر که همان کارت کتابخانه است با خود کتابخانه وابستگی دارد. لذا ما این کلاس وابستگی را با ارتباط دادن آن توسط نقطه چین نشان می

دهیم. یک کلاس وابستگی درست همانند دیگر کلاس ها می تواند به بقیه کلاس ها وابستگی داشته باشد. پس اجازه بدهید بگوییم که یک کلاس کتابدار داریم که به کلاس کارت کتابخانه توسط عمل (تصدیق صحت کارت) وابسته است. لذا ما کلاس کارت کتابخانه را داریم که یک کلاس وابستگی است و این کلاس خود خصوصیتی از وابستگی بین عضو و کتابخانه است. این کلاس وابستگی، با کلاس کتابدار نیز در وابستگی است. به گونه ای که کتابدار کارت کتابخانه را تصدیق می کند.



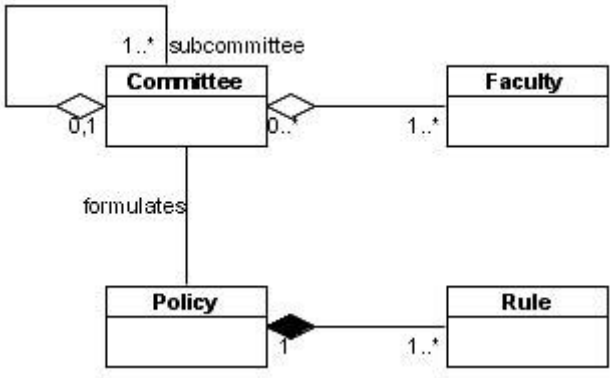
تجمیع و ترکیب در UML :

تجمیع و ترکیب هر دو یک نمونه بارزی از وابستگی هستند که هر دوی آنها ارتباطی را بین یک قسمت و جزئیات آن توصیف می کنند. پس بجای داشتن یک ارتباط به عنوان یک وابستگی ساده ما با یک ارتباطی سر و کار داریم که اجزای خود را توصیف می کند یا خواندن ارتباط در جهت دیگری که از آن ساخته شده است. پس مثال این نوع از وابستگی می تواند شامل یک گروه موسیقی ، یک فهرست از محصولات یا یک ساختمان که از اتاق هایش ساخته شده باشد. پس در هر صورت شما می توانید بگویید یک گروه از موسیقی دانانش یا اگر از جهت دیگر نگاه کنیم یک موسیقی دان جزئی از گروهش است. یک فهرست از محصولاتش تشکیل شده است یا یک محصول جزیی از فهرستش است. یک ساختمان از اتاق هایش تشکیل شده است یا یک اتاق جزیی از ساختمانش است. شما Aggregation را با یک متصل کننده خاص نشان می دهید وآن شبیه به یک لوزی تو خالی که به سرتاسر آن متصل است شبیه است.



ما می خواهیم که کل آن در سمت چپ و سایر قسمت هایش در سمت راست قرار گیرد ولی وقتی که ما می خواهیم جزئیات را بخوانیم به این شکل می خوانیم که یک موسیقی دان جزئی از گروهش است همچنین به طور مشابه برای فهرست و محصولات ، و ساختمان و اتاق. و اگر ما بخواهیم ما می توانیم چندگانگی خود را اضافه کنیم ، پس ما یک یا چندین موسیقی دان را به عنوان جزیی از یک گروه داریم ، یک یا چندین محصول اجزای یک فهرست هستند و غیره. حالا Composition یک ارتباط همواره قوی تری از aggregation است. برای تست آن چیزی که شما با Composition سرو کار دارید از قانون غیر مشترک استفاده کنید ، این قانون چگونگی این ارتباط را توصیف می کند که یک جزء فقط می تواند به یک کل تعلق داشته باشد پس بگذارید به مثال خود نگاهی بیاندازیم. آیا یک موسیقی دان فقط می تواند جزء یک گروه موسیقی باشد؟ نه این درست نیست. یک موسیقی دان می تواند به چندین گروه متفاوت تعلق داشته باشد. بگذارید چندگانگی ما این را نشان دهد. همچنین در مورد ارتباط فهرست محصولات ، هر محصول خاص می تواند در چندین فهرست ظاهر شود. پس بگذارید ببینیم وقتی که ما به آخرین ارتباط یعنی بین یک ساختمان و یک اتاق می رسیم چه اتفاقی می افتد. آیا یک اتاق می تواند به بیش از یک ساختمان تعلق داشته باشد یا آیا ما یک قانون غیر مشترک داریم؟ حالا پس از Aggregation این ارتباط بوسیله Composition بهتر توصیف

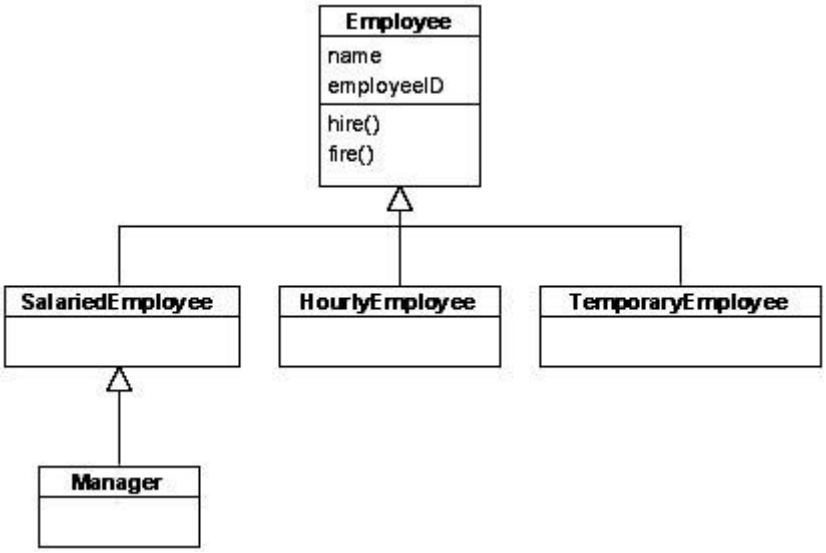
شده است. ما Composition رابوسیله متصل کننده ی مشابهی نشان می دهیم ولی به جای یک لوزی تو خالی ما یک لوزی پر داریم. پس یک ارتباط Composition در این گونه ارتباط می گوید که یک جزء فقط می تواند به یک کل غیر مشترک تعلق داشته باشد. به طور مرتبط به هم اگر کل از بین برود اجزایی که آن را تشکیل می دهند نیز از بین می روند. پس اگر شما یک ساختمان را از بین ببرید شما اتاق های واقع در آن را نیز از بین برده اید. در سمت دیگر در مورد aggregation این صحت ندارد. اگر یک گروه از بین برود موسیقی دانانی که اجزای آن هستند میتوانند به کاراهای دیگری بپردازند تا یک گروه دیگر را تشکیل دهند.



پس با دانستن aggregation و Composition شما قادر خواهید بود این دیاگرام کلاس را بخوانید. به خاطر داشته باشید که ما در مورد جزء و کل صحبت می کنیم پس ما یک یا چندین اعضای هیئت علمی را داریم که اجزای یک کمیسیون هستند پس این کمیسیون یک ارتباط انعکاسی با خودش داریم تا یک زیر مجموعه را شکل دهد ، پس یک یا چندین اعضای کمیسیون می توانند در یک یا چندین زیر مجموعه باشند. در زیر اینجا ما یک ارتباط Composition بین سیاست و قانون داریم. قوانین جزئی از یک سیاست هستند و آن را شکل می دهند ، این یک ارتباط قوی است. اگر سیاست از بین برود قانون نیز از بین خواهد رفت. توجه کنید که ما نمی توانیم وابستگی Composition یا aggregation برای مثال بین کمیسیون و سیاست داشته باشیم. این قابل درک نیست که بگوییم یک سیاست جزی از یک کمیسیون است یا یک کمیسیون از سیاست ها تشکیل شده است. به هر حال ما یک وابستگی بین این کلاسها داریم ، کمیسیون سیاست را تنظیم می کند. پس این دیاگرام aggregationو Composition نشان می دهد که چگونه مثل Association می تواند تنوع ارتباطات بین کلاسهای گوناگون بخصوص ، تا بیان واضح یک ارتباط جزء و کل را نشان دهد.

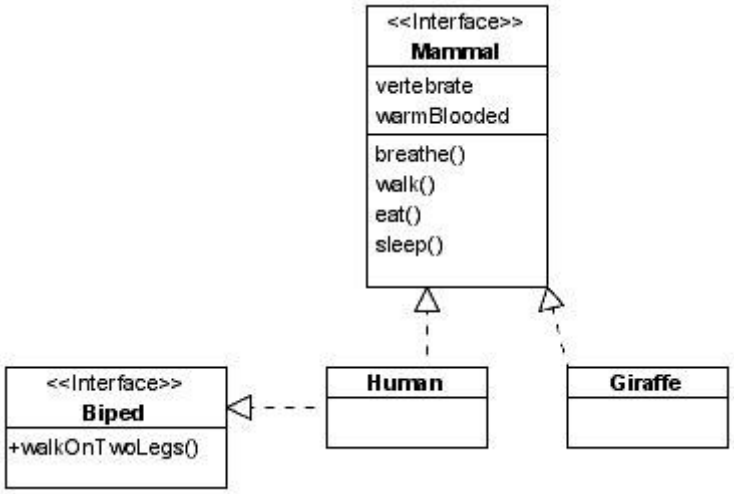
تعمیم Generalization:

تعمیم روشی است که UML به وسیله آن وراثت را توضیح می دهد. تعمیم در دیاگرام های کلاس بسیار شبیه به تعمیم در Use Case هاست. در تعمیم یک کلاس والد و حداقل یک کلاس فرزند وجود دارد و ممکن است به این کلاس ها به عنوان کلاس اصلی و زیر کلاس نیز اشاره شود. هر کلاس خصوصیات و رفتارهای والد خود را به ارث می برد و علاوه بر آن ویژگی های دیگری نیز دارد. برای مثال اگر یک کلاس والد به نام کارمند داشته باشیم، زیر کلاس های ما ممکن است کارمند رسمی، کارمند ساعتی و کارمند قراردادی باشد. بنابراین هر نوع از کارمندان شامل رسمی، ساعتی و قراردادی خصوصیات و رفتارهای کلاس والد را به ارث می برد در عین حال هر کدام از کلاس های فرزند با دیگری تفاوت دارد. برای مثال خصوصیات نام و شماره پرسنلی و همچنین رفتارهای مزد گرفتن یا اخراج شدن بین همه فرزندان مشترک است اما تفاوت بین آنها ممکن است چیزهایی از قبیل مزایایی که هر کدام دارند یا سیستم پرداخت حقوقشان یا غیره باشد. لذا هر کدام در خصوصیات و رفتارهایی که از والدشان به ارث برده اند مشترک اند در عین حال دارای تفاوت هایی با یکدیگرند. روشی که با آن تعمیم را نشان می دهید به شکل یک پیکان است که از فرزندان خارج شده و به کلاس والد وارد می شود. با استفاده از تعمیم می توان چند ریختی را نیز پیاده سازی کرد زیرا اگر یک رفتار برای کلاس والد تعریف شده باشد آنگاه هر کلاس فرزند می تواند با والد تعویض شود. نکته ی دیگری که می توان به آن اشاره کرد این است که شما محدودیتی در داشتن فقط یک کلاس والد و تعدادی فرزند نخواهید داشت. به عبارت دیگر هر کلاس فرزند خود نیز می تواند به عنوان کلاس والد حاوی تعدادی زیر کلاس باشد که از آن تعمیم یافته اند. برای مثال کارمند رسمی ممکن است دارای زیر کلاسی مانند مدیر باشد. بنابراین می توانید چندین سطح از وراثت را در دیاگرام خود داشته باشید.



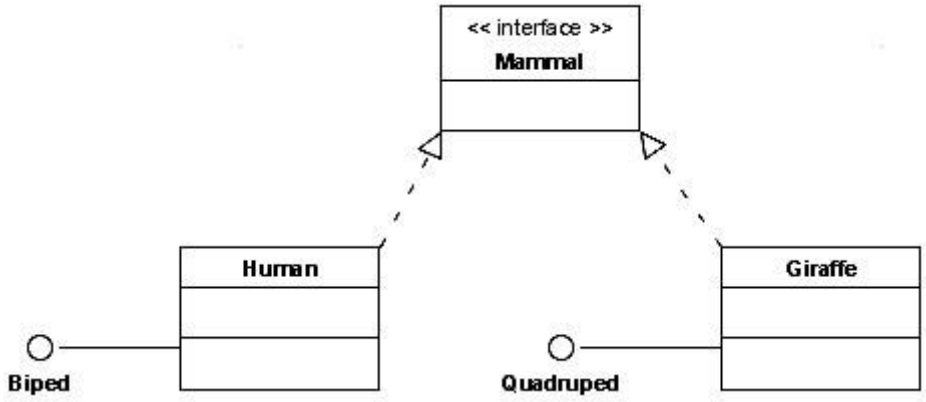
تحقق :

اگر تعمیم وراثت را نمایش می دهد، تحقق آن را پیاده سازی می کند. در UML تحقق ارتباط بین یک واسط و یک کلاس را توضیح می دهد. واسط یک طبقه بندی کننده است که خصوصیات، رفتارها و یا هر دو را که مجموعه ای ویژگی های به هم چسبیده را تشکیل می دهند، تعریف می کند.



واسط خصوصیات و رفتارها را نشان می دهد اما آنها را پیاده سازی نمی کند. در عوض خود واسط توسط کلاس و مشتقاتش پیاده سازی می شود. در UML پیاده سازی یک واسط، تحقق نامیده می شود. پس یک واسط قراردادی برای مجموعه ای از ویژگی ها تعریف می کند. واسط به عنوان یک قرارداد این ویژگی ها را مستقیماً تحقق نمی بخشد. یک کلاس ویژگی های یک واسط را با پیاده سازی آنها تحقق می بخشد. روشی که ما نشان می دهیم یا راهی که شما می توانید یک واسط را در UML نشان دهید، استفاده از قالب های واسطه ای برای توسعه دادن مفهوم کلاس است و لازم به یادآوری است که برای نمایش قالب، نام آن را داخل گیومه می گذاریم. اجازه بدهید نگاهی به چگونگی کارکرد این واسطه ها بیندازیم. ما اینجا واسطی به نام پستاندار داریم. این واسط مجموعه ای از ویژگی ها را که بین کلاس های پیاده سازی کننده این واسط مشترک است، تعریف می کند. ما خصوصیاتی مثل مهره دار بودن وخون گرم بودن و رفتارهایی مثل نفس کشیدن، راه رفتن، خوردن، خوابیدن وغیره را داریم. توجه داشته باشید که کلاس های مختلفی ویژگی های واسط پستاندار را پیاده سازی می کنند. برای مثال کلاس انسان و تمام نمونه های ساخته شده از آن، می توانند ویژگی های تعریف شده توسط واسط پستاندار را پیاده سازی کنند. به طور مشابه، تمام نمونه های ساخته شده از کلاس زرافه نیز می توانند ویژگی های تعریف شده توسط واسط پستاندار را پیاده سازی کنند. روشی که ما روابط تحقق یافته را بین کلاس و واسط را در آن نشان می دهیم، استفاده از یک پیکان نقطه چین است که از کلاس ها خارج و به واسط وارد شده اند. تصویر مقابل شکل این ارتباط را نشان می دهد. البته توجه داشته باشید که یک واسط می تواند بیش از یک کلاس پیاده سازی شده مرتبط با

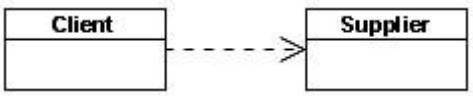
خود داشته باشد. همچنین یک کلاس می تواند بیش از یک واسط را پیاده سازی کند. برای مثال کلاس انسان ویژگی های تعریف شده توسط واسط پستاندار را پیاده سازی می کند اما ما یک واسط دیگر نیز داشتیم، که به آن موجود دوپا خواهیم گفت که شامل رفتاری به نام (راه رفتن روی دو پا) دارد. کلاس انسان می تواند ویژگی های واسط موجود دو پا را همانند ویژگی های واسط پستاندار به خوبی پیاده سازی کند. لذا می توانیم ارتباط را مثل این نمایش دهیم. کلاس انسان ویژگی های تعریف شده در هر دو این واسط ها را پیاده سازی می کند. این در حالی اس که کلاس زرافه فقط ویژگی های کلاس پستاندار را پیاده سازی می کند. حالی روشی برای نشان دادن رابطه تحقق یافته با استفاده از قالب واسطه ای با جعبه های طبقه بندی کننده و این پیکان های نقطه چین بیان می کنیم.



راه دیگری نیز وجود دارد که شما می توانید مشاهده کنید. اینجا چگونگی ساخت یکی دیگر در یک دیاگرام دیگر است. ما یک قالب واسطه ای برای پستانداران و همچنین کلاس های انسان و زرافه را داریم. اما این نمادهای دایره ای شکل یک واسط را هم نشان می دهند پس شما می توانید یک رابطه تحقیق یافته را نشان دهید که هر دو از پیکان های خط چین و جعبه های طبقه بندی کننده با قابل های واسطه ای استفاده می کنند یا شما می توانید از این نماد دایره ای واسط و یک خط ساده برای ارتباط با کلاس پیاده سازی شده استفاده کنید. پس دوباره ما اطلاعات مشابهی در این دیاگرام داریم، کلاس انسان ویژگی های واسط های پستاندار و موجود دو پا را پیاده سازی می کند. توجه داشته باشید که باید نماد واسط تان را برچسب گذاری کنید. کلاس زرافه ویژگی واسط های پستاندار و موجود چهار پا را پیاده سازی می کند. لذا این شکل اطلاعات مشابه با دیاگرام قبلی را نشان می دهد. نمادی که شما به کار می برید به چیزی بستگی داری که می خواهید در دیاگرامتان نشان دهید.اگر می خواهید جزئیات واسطتان را در دیاگرام نشان دهید، از قالب های واسطه ای در جعبه طبقه بندی کننده ها استفاده کنید. اگر واسطتان را در جای دیگری تعریف کرده اید و قصد دارید که دیاگرامتان ساده باشد آنگاه از نماد دایره ای شکل برای نشان داده واسط استفاده کنید.

وابستگی :

وابستگی ارتباطی را نشان می دهد که هر کلاسی از یک طریقی از کلاس دیگر استفاده می کند. بنابراین شما در اینجا می بینید که ما یک کلاس وابسته داریم ، که سرویس گیرنده نامیده شده است از کلاس ثانویه استفاده کرده است ، همچنین کلاسی داریم که کلاس دیگر به آن وابسته است ، که گاهی تهیه کننده نامیده می شود. و این دو کلاس از طریق پیکان وابستگی به یکدیگر متصل شده اند. نقطه شروع پیکان از سمت سرویس گیرنده (Client) تا تهیه کننده است (Supplier) یعنی از کلاس وابسته به کلاسی که از آن استفاده می کند.

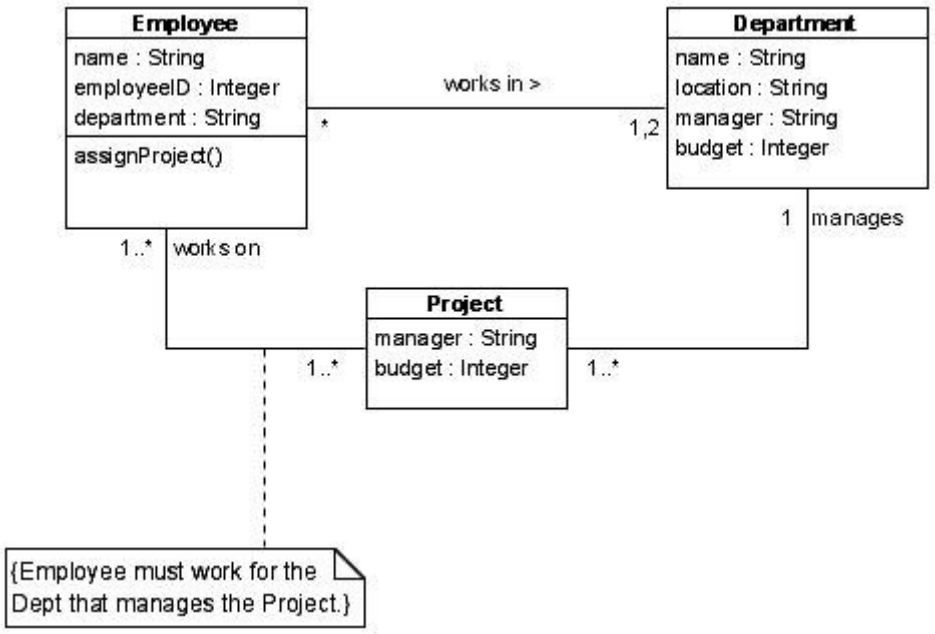


شما می توانید پیکان وابستگی را این طور بخوانید "استفاده می کند از یک" ، پس با نگاه به این دیاگرام ما می توانیم بگوییم که یک سرویس گیرنده از یک تهیه کننده استفاده می کند. یا شما می توانید با تکیه به روش بالا بخوانید یا اینکه از یک طریقی بفهمید ، تا نشان دهید که یک سرویس گیرنده از یک تهیه کننده استفاده می کند.پس برای مثال بگذارید بگوییم که ما یک کلاس داریم به نام رفتگر که این کلاس یک عملکردی به نام جارو زدن دارد. کلاس رفتگر میتواند به کلاس جارو وابسته باشد بنا بر این ما می توانیم بگوییم که یک رفتگر از یک جارو استفاده می کند. حالا به این نکته توجه کنید که اگر تغییراتی در کلاس تهیه کننده اعمال شود این می تواند بر

کلاس سرویس گیرنده نیز تاثیر بگذارد و این نکته ی با ارزشی است تا در ذهن بسپارید ، این مهم است که دنبالهء وابستگی را در به خاطر داشته باشید زیرا در یک سیستم پیچیده تغییرات بر یک کلاس ، مشابه این است که بر کلاس های دیگر تاثیر بگذاریم و شما می خواهید مطمئن شود که می توانید این وابستگی را از داخل سیستم دنبال کنید. پیکان وابستگی به طور پایه ای اکثرا "استفاده از یک" ارتباط را نشان می دهد.شما همان گونه که می توانید عناصر را UML قالب دهی کنید ، می توانید وابستگی ها را قالب دهی کنید. برای مثال شما می تونید در دیاگرام های Use Case وابستگی ها را برای Use Case های Include و Extend قالب دهی کنید. در دیاگرام های کلاس شما می توانید متناسب با احتیاجات خود قالب ها را برای وابستگی ها ایجاد کنید و در اینجا شما می توانید یک مثال از انواع وابستگی ها را ببینید که که شما ممکن است بخواهید به اعمال این وابستگی ها اعم از ، یافتن وابستگی سطح دسترسی ، وارد کردن ، ترکیب کردن ، نمونه سازی ، جانشینی و غیره رسیدگی کنید. پس بخاطر داشته باشید که حتی در وابستگی پایه ارتباط "استفاده می کند از" است. به طور کل شما مقداری انعطاف پذیری برای استفاده از قالب ها و اعمال آنها بر روی وابستگی ها دارید.

محدودیت ها و یادداشتها :

Constraint ها محدویت عناصر را در دیاگرام UML شما نشان می دهند. آنها بر روی انواع مختلف محدودیت می گذارند و جلوگیری می کنند. وقتی شما در مورد دیاگرام خود فکر می کنید تعداد زیادی از سئوال ها مثل این که چه کار باید در دیاگرام خود بکنید نمایش محدودیت ها به طریق گوناگون است. برای مثال Multiplicity می توند عملکردی مانند یک نوع محدودیت باشد. در اینجا برای مثال ما یک کارمندی داریم که در یک یا دو بخش کار می کند ، نه بیشتر ، نه کمتر. پس این مثال خوب است تا نشان دهیم چگونه Multiplicity همانند یک محدویت کار می کند. ولی از طریقConstraint ها می توانید محدویت ها به طور رسمی تری نشان دهد. شما می توانید محدودیت ها را مانند یک پیش شرط یا ما بعد شرط بیان کنید و شاید بهترین راه برای انجام این کار الصاق یک یادداشت در جایی که محدودیت اعمال شده است باشد.

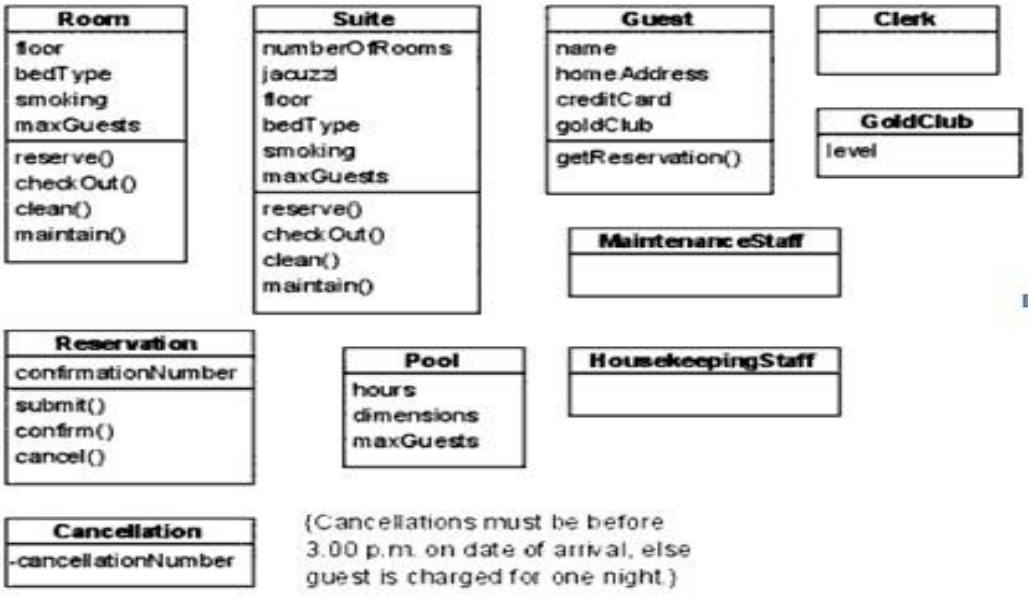


برای مثال در اینجا ما یک کلاس کارمند داریم. یک یا چند در یک یا چند پروژه کار می کنند. بگذارید یک محدودیت را به شکل یک یادداشت اضافه کنیم که تعیین می کند که یک کارمند باید در بخشی کار کند که پروژه را مدیریت می کند. یادداشت ها از علائمی که شما در اینجا می بینید استفاده می کنند ، نگاه کنید همانند یک تکه از کاغذ با گوشه بالایی شبیه Folder است. یک یادداشت محدودیت را از طریق براکت های مجعد تعیین می کند پس محدودیت در براکت های مجعد ضمیمه شده است.پس ما یک براکت باز داریم ، "کارمند باید برای بخشی کار کند که پروژه را مدیریت می کند" و براکت بسته می شود و یک پیوند داریم که محدودیت را به جایی که به آن تعلق دارد لنگر می کند ، در این مورد ، این پیوند بوسیله خط نقطه چنین نشان داده می شود. شما می توانید از یادداشت ها برای تعیین هر چیز دیگری در دیاگرام های UML خود استفاده کنید و همانگونه که گفته شد آنها مخصوصا برای محدودیت ها مناسب هستند. روش دیگر که می توانید محدودیت ها را نشان دهید این است که آنها را به ویژگی های جایی که به آن تعلق دارند اضافه کنیم. به طور مثال ما یک پروژه ای داریم که به وسیله یک دایره ای یا بخشی مدیریت می شود. یک بخش یک یا چند پروژه را مدیریت می کند. در هر دوی این کلاس ها ما ویژگیهایی به نام

بودجه داریم و در کلاس Project ما یک تابعی داریم به نام GetBudget() ، ما ممکن است بخواهیم یک محدودیتی را اضافه کنیم که یک پیش شرطی را تعیین کند ، ما در اینجا GetBudget() را داریم ، این یک Integer است و براکت نشان می دهد که ما یک پیش شرط داریم ، "بودجه پروژه کمتر یا مساوی بودجه دایره یا بخش است. بنابراین شما می توانید محدودیت را به این طریق نشان دهید یا شما می توانید محدودیت را به شکل یک یادداشت که به پیوند میان پروژه و سازمان ، به جایی که ترجیح می دهید اضافه کنید. وقتی شما با یادداشت ها سرو کار دارید شما همانگونه که من در اینجا استفاده کردم می توانید از زبان طبیعی خودتان استفاده کنید ، شما می توانید از یک زبان برنامه نویسی و یا یک زبان برنامه نویسی ساختگی استفاده کنید و شما همچنین می توانید از زبان محدودیت اشیاء استفاده کنید. وقتی شما تصمیم گرفتید که که چگونه محدودیت های خود را بیان کنید چیزی که باید در ذهن بسپارید این است که مخاطب شما چه کسی است ، افراد در سوی تجارت نمی توانند زبان محدودیت اشیاء را متوجه بشوند ولی در اغلب موارد استفاده از زبان طبیعی می تواند بهترین راه باشد. در سوی دیگر زبان طبیعی گاهی اوقات یه یک طریقی که شما نمی خواهید ابهام ایجاد کند. پس متناسب با این که چه کسی دیاگرام شما را می خواند ، از زبانی استفاده کنید برای آن مجموعه واضح ترین و بهترین باشد.

یافتن کلاسها :

شاید شگفت زده شوید از این که چگونه کلاس هایی که شما آن را در دیاگرام کلاس مدل می کنید را پیدا می کنیم. این ویدئو در این باره صحبت میکند پیدا کردن و تعریف کردن کلاس ها. یکی از راه های خوب برای شروع این است که Use Case هایی که تعریف کردید را بررسی کنید اگر آنها را داشته باشید و با فردی که در آن کار حرفه ای و با تجربه است صحبت کرده باشید. بوسیله کلاس موجودیت –منظور ما کلاس است که شیء را تعریف می کند - که توسط بیزینس پیدا می شوند. پس معمولا وقتی ما در مورد کلاسهای موجودیت صحبت می کنیم معمولا در مورد داده ها صحبت می کنیم. از آنجایی که شما مشخصات را مرور کرده اید و با فرد حرفه ای در آن زمینه کاری صحبت کرده اید، شما می خواهید به یک چیز خاص گوش دهید برای اسامی که استفاده کرده اید. اسامی بعدا به شما برای تعریف کلاس ها کمک می کنند. این ها شامل چیز های قابل لمس، چیز هایی که شما می بینید یا لمس می کنید، چیزهایی مثل سفارش، فاکتور، کارت ساعت کاری، حقوق ،ویا موجودی باشد.همچنین شما مخواهید که قوانین راببینید.اگر شما کار Use Case ها را تمام کرده باشد، شما تعدادی از قوانین مشخص و تایید شده را دارید.مشاهده برای روی دادن چیز ها که اتفاق می افتند مثل یک دادن سفارش و یا لغو سفارش،دریافت وجه و غیره ... .اگر شما Use Case ها را تمام کرده باشید بیشتر رویداد ها شبیه Use Case های شما هستند. نگاهی به خارج از نوع گوناگون تعامل بیندازید، فروش سفارش خرید را از مشتری دریافت می کند و یا در یک سیستم بیمارستان، دکتر مریض را درمان می کند. همچنین برای خصوصیت ها جستجو کنید. وقتی شما در مورد خصوصیت ها فکر می کنید، یک نواخت و استاندارد فکر می کنید یا استاندارد حکومتی در مورد ابعاد و حجم فکر می کنید. خصوصیات بعدا شما را در تعریف Attribute ها و همچنین در مورد محدودیت ها کمک می کنند. همچنین همان طور که گفته شد دیدن برای اسامی و دلیل این که اسامی شما را برای تعریف کلاس ها و خصوصیات و اشیاء کمک می کنند. شما همچنین باید گوش فرا دهید به فعل ها، به این مفهوم که اسامی کاری را انجام می دهند. فعل شامل عملیات و توابع و روابط می شوند. همچنین آنها به شما اطلاعات خوبی را پیشنهاد می دهند.

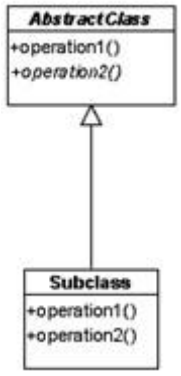


برای مثال مردم کارگر و کارمند را به جای هم به کار می برند. اسامی دیگر ممکن است وضع دیگری در کلاسی که شما می خواهید تعریف کنید و یا سیستمی که طراحی کرده اید، نداشته باشند. پس شما باید لیستی از اسامی

را در بیاورید و اطلاعات مورد استفاده از هر کدام را بنویسید (کدام اطلاعات را نیاز دارید و به شما کمک می کند و نیاز به ذخیره کدام اطلاعات دارید).کلاس ها را حفظ کنید که نمایش گر اشیای فیزیکی هستند.آنهایی قابل لمس هستند ما بعدا در موردشان صحبت خواهیم کرد.شما می خواهید که کلاس ها را نگه دارید که نمایش گر موجودیت های مفهومی هستند.شما همچنین می خواهید کلاسهای رو نگه دارید که که نمایشگر دسته های کلاس هستند.اینها می توانند به ابر کلاس تبدیل شوند که برای چیز هایی مانند عمومی سازی به ما کمک خواهد کرد.و در پایان کلاسی را نگه می داریم که نمایش گر واسط های شناخته شده جهان خارج از سیستم است.در اینجا یک مثال سریع در صحبت کردن با پرسنل درباره یک هتل می باشد و اینکه چگونه آنها کار می کنند. شما می توانید کلاسها را بر اساس یک سیستم طراحی شده هتل تعریف کنید. یکی از کلاس های اصلی Room می باشد که دارای خصوصیاتی همچون : طبقه و نوع و چند تخته ، یک طبقه یا دوبلکس، اجازه کشیدن سیگار دارند یا نه، حداکثر تعداد افرادی که می توانند در ان اقامت کنند، می باشد. عملیات آن شامل : رزرو شدن، تمیز شدن،و غیره... می باشد.یک کلاس فرزند که ارث برده شده از کلاس Room و اون Suite Room است.این کلاس فرزند تمام خصوصیات و عملیات های کلاس والد خود یعنی Room را دارد اما این کلاس تعدادی خصوصیت دیگر هم دارد.تعداد اتاق ها در سویت که می تواند بیشتر از یکی باشد و حتی ممکن است سویت جکوزی هم داشته باشد.همچنین اینجا تعدادی نقش در سیستم هتل وجود دارد. ما در اینجا مهمان را داریم که مهمان خصوصیت هایی مثل نام ، آدرس، کارت اعتباری ، و... را دارد. و همچنین عملیاتی همچون رزرو کردن. ما همچنین تعدادی کلاس کارمندان هتل را داریم. مثل منشی، کارمندان نگهداری از هتل،کارمندان خانه داری.علاوه بر اینها ما کلاس رزرو کردن را داریم خصوصیت هایی همچون : شماره تایید و عملیاتی همچون : ثبت ، تایید ، اغو.کلاس لغو ما خود شامل خصوصیت شماره لغو می باشد. از آنجایی که با متخصص این زمینه صحبت کرده اید شما شاید در مورد قوانین کار سیاست ها وچیزهای دیگر که به شما برای تعریف محدودیت ها کمک می کند. لغو اتاق باید قبل از ساعت 3 بعد از ظهر باشد و در غیر این صورت یک روز دیگر شارژ می شود. شما همچنین چیزهایی مثل محدودیت ها را جمع می کنید و سندهایی درست می کنید برای شروع تعریف کلاس.این فقط مرحله اول است که شاید شما به آن نگاه انداخته باشید. همچنین، همانطور که دارید در چرخه ساخت سیستم و پیاده سازی آن پیش می رود کلاس هایی را تعریف می کنید که از قبل آنها را ندیده اید و یا متوجه می شوید کلاس هایی برای سیستم شما ضروری نیستند.اما مرور خصوصیات و صحبت کردن با فرد متخصص در سیستم شما یک مرحله شروع خوب می تواند باشد که به شما برای تعریف کلاس هایی که به آن نیاز دارید کمک می کند.

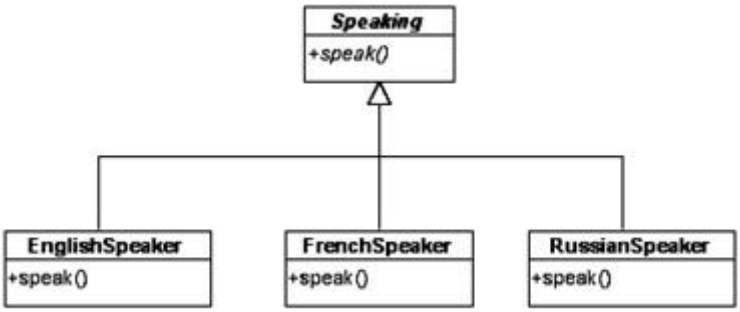
دیاگرامهای کلاس پیشرفته ؛ کلاسهای انتزاعی :

کلاس های انتزاعی به گونه ای تعریف می شوند که کلاس های دیگر می توانند از آنها به ارث ببرند. نمی توان به طور مستقیم از یک کلاس انتزاعی نمونه ساخت اما از نوادگان و در واقع از زیرکلاسهای آن می توان نمونه ساخت. برای نشان دادن کلاس های انتزاعی در UML نام کلاس را به صورت حروف کج (Italic) می نویسید. همان طور که در اینجا می بینید برای اینکه مطلب واضح تر شود نام کلاس را، Abstract Class (کلاس انتزاعی) می گذاریم. یک کلاس انتزاعی معمولاً یک یا چند رفتار انتزاعی دارد. یک رفتار انتزاعی پیاده سازی نمی شود.



اما هر زیرکلاسی که رفتارهای کلاس انتزاعی را به ارث می برد، آن رفتار را متناسب با نیاز خود پیاده سازی می کند. تا لحظاتی دیگر مثالی را خواهیم دید اما قبل از آن اجازه دهید چگونگی نمایش آن توسط ابزارهای (جعبه ابزار طبقه بندی کننده) نشان دهیم. ما دو رفتار داریم. رفتار شماره 2 را به عنوان یک رفتار انتزاعی در نظر می گیریم (نام آن را با حروف کج می نویسیم). توجه داشته باشید که که یک کلاس انتزاعی هم می تواند رفتارهای انتزاعی و هم رفتارهای واقعی داشته باشد و دلیل اینکه نام رفتارها را به صورت کج نوشته می شود این است که مشخص شود در کلاس انتزاعی، کدام رفتارها انتزاعیند. اینگونه تصور کنید که یک کلاس انتزاعی یک رفتار را تعریف می کند

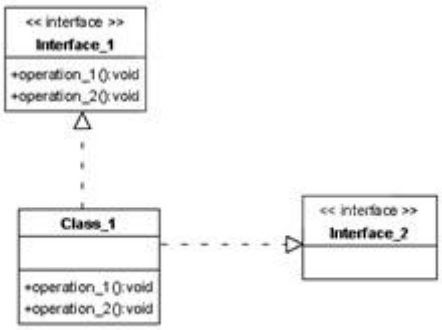
اما آن را پیاده سازی نمی کند. در این حالت کلاس انتزاعی یک رفتار انتزاعی را برای تمام زیرکلاسهای خود به گونه ای تعریف می کند که هر یک می تواند به شکلی مختلف آن را پیاده سازی کند. برای نمایش ارتباط بین یک کلاس انتزاعی و زیرکلاسهایش به این شکل عمل می کنیم، ما به این کلاس، کلاس شماره 2 خواهیم گفت که در اینجا یک زیرکلاس است و از آنجا که یک کلاس استاندارد است نام آن را به صورت کج ننوشتیم. این زیرکلاس همچنین دارای 2 رفتار است که از کلاس انتزاعی به ارث برده است و هر دوی این رفتارهای شماره 1 و 2 به صورت کج نوشته نشده اند (انتزاعی نیستند) زیرا رفتارهایی هستند که می توان آنها را پیاده سازی کرد. برای اینکه نمایش دهیم این زیرکلاس از کلاس انتزاعی ارث می برد، از فلش تعمیم (Generalization) به نحوی استفاده می کنیم که جهت پیکان از کلاس فرزند خارج و به کلاس انتزاعی وارد شده باشد.



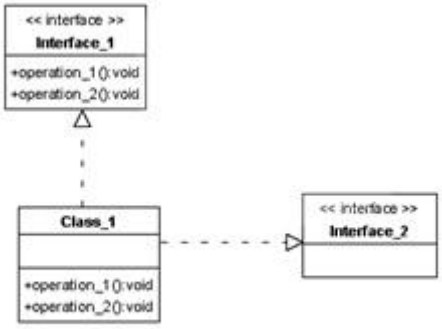
همانطور که می بینید در کلاس انتزاعی می توانیم به طور واضح نام کلاس و همچنین رفتار انتزاعی آن را مشاهده کنیم. همچنین رفتاری که از این کلاس به ارث برده شده را نیز می بینیم که می تواند پیاده سازی شود و از آن نمونه ای نیز ساخته شود. حال به مثالی که پیشتر اشاره کردم می رسیم، در اینجا یک کلاس انتزاعی به نام صحبت (Speaking) داریم که حاوی یک رفتار انتزاعی به نام صحبت کردن (Speak) می باشد. سه کلاس استاندارد به عنوان زیرکلاسهای این کلاس انتزاعی با نامهای «صحبت کننده به زبان انگلیسی (English Speaker)»، «صحبت کننده به زبان فرانسوی (French Speaker)» و «صحبت کننده به زبان روسی (Russian Speaker)» وجود دارد. هر کدام از این زیرکلاسها رفتار صحبت کردن را به ارث می برند اما هر کدام از آنها این رفتار را به شیوه خاص خود پیاده سازی می کنند. در واقع نمونه های کلاس صحبت کننده به زبان انگلیسی به یک شیوه سخن خواهند گفت، نمونه های کلاس صحبت کننده به زبان فرانسوی به شیوه ی دیگر صحبت خواهند کرد و در نهایت نمونه های کلاس صحبت کننده به زبان روسی نیز به شیوه ای متفاوت از دو شیوه قبل. لذا همانطور که می بینید کلاس انتزاعی اجازه پیاده سازی رفتار صحبت کردن را به تمام زیر کلاس های خود می دهد اما هر زیرکلاس به شیوه خاص خود آن را پیاده سازی می کند. و در نهایت توجه داشته باشید که هر کلاس والد لزومی ندارد حتماً انتزاعی باشد. یک کلاس استاندارد هم می تواند به عنوان کلاس والد در نظر گرفته شود برای مثال کلاس صحبت کننده به زبان انگلیسی می تواند به عنوان والد، زیر کلاس هایی با نام های آمریکایی، بریتانیایی، استرالیایی، کانادایی و غیره داشته باشد. لذا هم کلاس های استاندارد و هم کلاس های انتزاعی می توانند کلاس والد باشند و خصوصیات و رفتارهایشان توسط کلاسهای فرزند به ارث برده شوند.

واسطها (رابطها) :

چیزی که اینجا می بینید روشی برای نشان دادن یک واسط در UML است. یک واسط مجموعه ای از رفتارهاست که پیاده سازی نمی شوند اما سرویس خاصی را برای یک کلاس یا یک مولفه معین می کند.

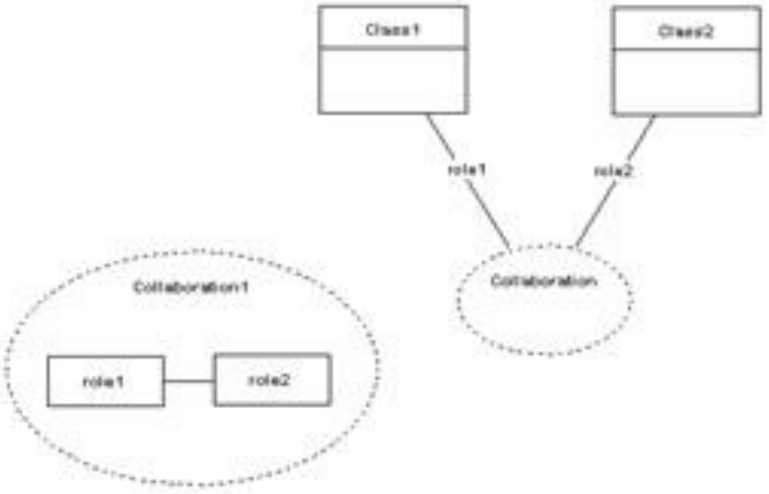


به عبارت دیگر واسط، یک یا چند سرویس پیاده سازی شده بوسیله بعضی از کلاس ها یا مولفه ها را نشان می دهد و از این به بعد تا هنگامی که در بخش دیاگرام های کلاس هستیم در رابطه با کلاس ها و واسط ها صحبت خواهیم کرد. در اینجا یک واسط داریم و اجازه بدهید به آن دو رفتار نیز بدهیم. این رفتارها توسط خود واسط نمی توانند پیاده سازی شوند و نیاز به کلاسی دارند که آنها را پیاده سازی کند. ما اینجا کلاسی داریم که به آن دو رفتار نیز می دهیم زیرا این کلاس قرار است رفتارهای تعریف شده در واسط شماره 1 را پیاده سازی کند لذا به آن رفتارهایی مشابه با رفتارهای تعریف شده در واسط شماره یک می دهیم. حال نشان می دهیم که این کلاس با استفاده از فلش تحقق (Realization) واسط را پیاده سازی می کند. شاید فکر کنید که یک واسط شبیه یک کلاس انتزاعی عمل می کند که درست هم هست اما در یک واسط تمام مشخصه های انتزاعی هستند. توجه داشته باشید که مستطیل واسط برخلاف مستطیل کلاس که از 3 بخش تشکیل شده است تنها دو بخش دارد. در واقع بخشی برای تعریف خصوصیات وجود ندارد و تنها رفتارها در واسط در تعریف می شوند. همچنین در بعضی از زبان های برنامه نویسی مثل جاوا (Java) یک کلاس ممکن است تنها از یک کلاس دیگر به ارث برده شود، یک کلاس استاندارد یا انتزاعی با واسط اگرچه یک کلاس ممکن است چندین واسط را پیاده سازی کند. لذا در اینجا کلاس شماره یک می تواند هر دو واسط 1 و 2 را پیاده سازی کند. از طرفی چندین کلاس می توانند یک واسط را پیاده سازی کنند. لذا واسط شماره 1 می تواند توسط کلاس های شماره 1 و 2 پیاده سازی شود. از هر دو صورت می توانید برای واسط هایتان استفاده کنید. از یک واسط نمی توان به طور مستقیم نمونه ساخت اما از هر کلاسی که آن را پیاده سازی می کند یا همان طور که گفتم در UML باید رفتارهای موجود در واسط را به نوعی پیاده سازی کرد. دو روش برای نمایش واسط در UML وجود دارد. یک روش، روشی است که با آن کار می کردیم، از یک مستطیل که از دو بخش تشکیل شده استفاده می کردیم که حاوی یک قالب با نام کلیدی واسط (Interface) است و دیگری استفاده از یک دایره است. برای واسط شما از این نماد استفاده می کنید. شما از فلش های Realization استفاده می کنید به نحوی که یک خط از سمت واسط به سمت کلاسی که آن را پیاده سازی می کند می کشید. به این واسط، واسط ملزوم گفته می شود. کلاس شماره 3 پیاده سازی واسط شماره 3 را فراهم می آورد. همچنین نوع دیگری از واسط ها وجود دارد که به آنها واسط های لازم گفته می شود. به جای نماد آب نبات یک واسط لازم شبیه فنجان یا یک گودی است. واسط ملزوم بدین معناست این کلاس پیاده سازی این واسط را برعهده دارد و یک واسط لازم بدین معناست که این کلاس به این واسط برای انجام عملیاتش نیازمند است. به عبارت دیگر این واسط به یک کلاس دیگر برای پیاده سازیش نیاز دارد و روشی که شما آن را نشان می دهید همان استفاده از نماد شکل توپ یا آب نبات است. لذا ما واسطی داریم که کلاس شماره 3 به آن احتیاج دارد و کلاس شماره 4 آن را فراهم می کند. بدین معنا که شما می توانید این دو را برای نمایش اینکه این دو واسط یکی اند، به هم وصل کنید. شما می توانید از یک فلش مستقل که از واسط لازم به ملزوم متصل است استفاده نمایید. البته اگز ابزار UML شما این اجازه را بدهد. شما می توانید نماد توپ شکل و فنجان شکل رو به هم متصل کنید برای اینکه نشان دهید کلاس شماره 3 به این واسط نیاز دارد و کلاس شماره 4 آن را فراهم می آورد. واسط ها می توانند برای تعریف رفتارهای مشترک میان چندین کلاس مفید باشند و انعطافی که برای نمایش واسط ها در دیاگرام های کلاس دارید به شما این امکان را می دهد که روی چیزهای خاصی در رابطه با واسطها تاکید کنید. ما در اینجا رفتارهای آن را تعریف می کنیم یا به شما اجازه می دهد که دیاگرامتان را فشرده تر کنید هنگامی که از اتصال های اسمبلی استفاده می کنید.

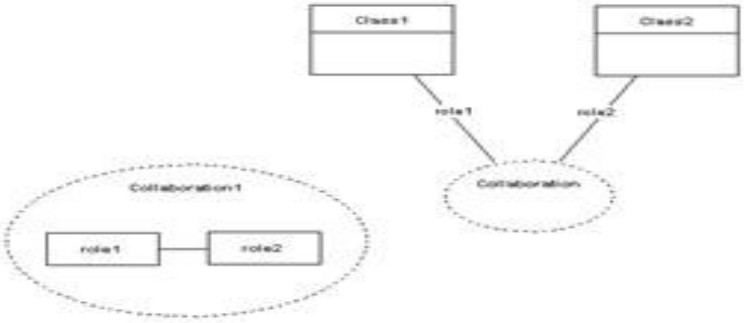


دیاگرامهای همکاری :

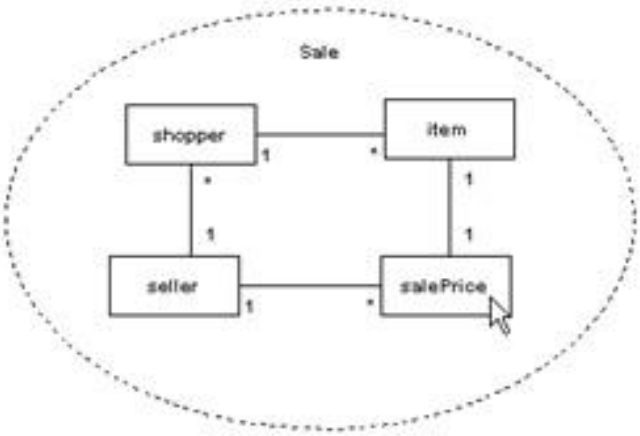
این مقاله آموزشی در مورد همکاری است. شما با Collaboration ها فقط در دیاگرام های کلاس روبرو نمی شوید بلکه در بسیاری از انواع دیاگرامهای UML با آن مواجه می شوید. معنی لغوی Collaboration همکاری با یکدیگر برای بدست آوردن یک نتیجه است.

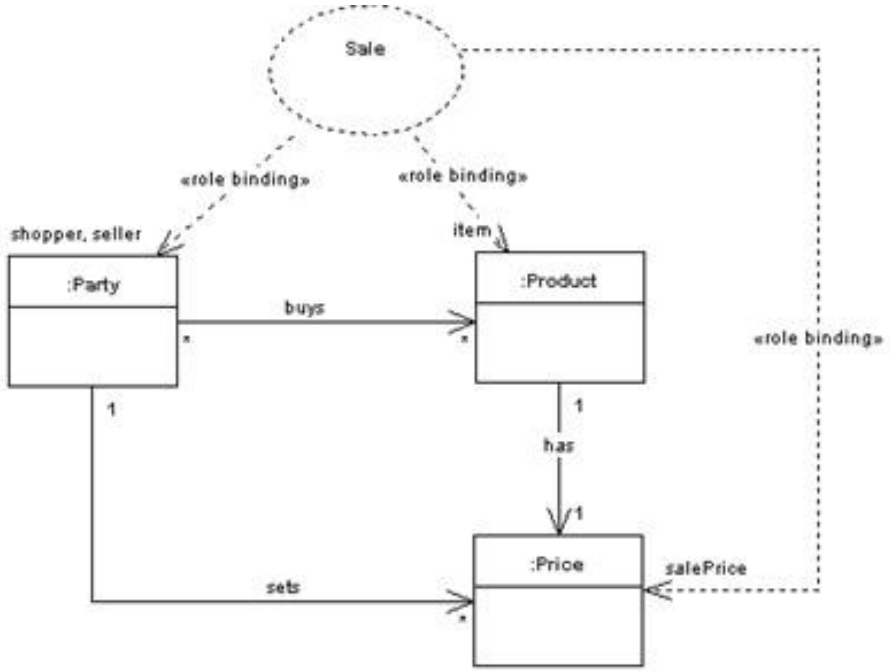


در یک دیاگرام UML، Collaboration یک گروه از عناصر است که با یکدیگر همکاری می کنند که یک نتیجه را بدست بیاورند. این همکاری ممکن است موقتی باشد یا برای همیشگی باشد. این یک الگویی از رفتار را نشان می دهد که شما می توانید به دیاگرام خود آن را اضافه کنید. چیزی که Collaboration را دسته بندی می کند یک بیضی ای است که محیط آن به صورت خطوط نقطه چین است و نام Collaboration داخل آن نوشته شده است.



چیزی که در Collaboration ایفای نقش می کند قوانین نامیده می شوند و شما می توانید آن را اینجا ببینید. قوانین با حروف کوچک نوشته می شوند ؛ عملکرد آنها مانند نگه دارنده ای برای اشیاء است زیرا در Runtime به صورت مرحله ای وارد می شوند و با قوانینی که در اینجا تعریف می شوند نقشی را بازی می کنند. متصل کننده ها در یک Collaboration به صورت لینک های موقتی نمایش داده می شوند . این ارتباط ها در طی یک همکاری وجود دارند ولی خارج Collaboration این طور نیست و با یکدیگر همکاری ندارند. پس بیاید ببینیم که یک Collaboration چگونه نشان داده می شود ، دو راه برای نشان دادن Collaboration در دیاگرام های UML وجود دارد. اولی آنی است که در اینجا می بینیم. شما دسته ی Collaboration را توسعه می دهید تا قوانینی که با یکدیگر همکاری می کنند و همچنین اتصال های بین آنها را درون دسته قرار دهید. روش دیگر برای ایجاد Collaboration استفاده از کلاس ها است که هر یک از قوانین به صورت نمونه (Instance) است ، پس ما از بیضی Collaboration فعلی خارج می شویم و ما این کار را با یک جفت کلاس انجام می دهیم ، ما آنها را کلاس یک و کلاس دو می نامیم. وقتی شما یک دسته ی Collaboration را که شبیه این است را می بینید ، این فقط یک بیضی را نشان می دهد و نه چیزی که داخل آن است ، این رویداد Collaboration نامیده می شود که شما می توانید رویداد Collaboration را ، که ما انجام می دهیم ، از طریق یک وابستگی به کلاسی که مشارکت می کند متصل کنید ، قوانین در Collaboration همان نمونه ها هستند. پس ما برای مثال می توانید این اتصال را "قانون یک" و این یکی را "قانون دو" بنامیم. پس این دیاگرام همکاری ، قوانین آن و همچنین کلاس هایی را نشان می دهد که هر یک از این قوانین به صورت نمونه هایی در Collaboration فعلی نمایش داده می شوند که شما می توانید آن را ببینید همچنین شما وقتی از اسن شیوه ی نماد گذاری استفاده می کنید می توانید اطلاعاتی را اضافه کنید زیرا شما عملیاتی را اضفه می کنید که برای مثال نقشی را در همکاری (Collaboration) بازی می کند. بنابراین بیاید به یک مثال فوری نگاهی بیاندازیم ، در اینجا ما یک Collaboration داریم که آن را فروش می نامیم که در اینجا چهار قانون پیچیده در این همکاری وجود دارد. ما دو قانون در طرفین فروش ، خریدار و فروشنده داریم . ما جنس را داریم و همچنین قیمت فروش را داریم . شما می توانید اتصال های گوناگونی را بین این قوانین ببینید همچنین شما می توانید Multiplicity را ببینید. در فروش یک خریدار می تواند تعداد زیادی جنس بخرد . پس این فقط یک مثالی از این شیوه ی نمایش Collaboration است ، شما نام Collaboration، اجزاء یا قوانین پیچیده و همچنین اتصال های بین آنها را دارید. حالا در اینجا وقتی که یک Collaboration را به Diagram Class خود اضافه می کنید می بینید که چگونه می تواند مشابه شود. این بسیار شبیه Class Diagram است. ما کلاس های گوناگونی داریم ، ما بخش ، محصول و قیمت و همچنین ارتباطات میان آنها را داریم. یک بخش محصول را خریداری می کند ، خریدار تعداد زیادی محصول می تواند بخرد ، یک محصول یک قیمت دارد و فروشنده می تواند قیمت ها را تنظیم کند .

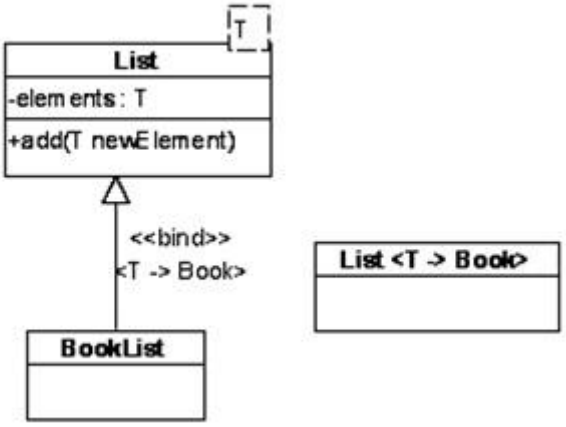




پس این Class Diagram ما است . ما می توانیم یک رویداد Collaboration را به این اضافه کنیم ، یک فروش ، که این فروش الگویی از رفتار این نمونه های این کلاس ها را در Class Diagram ما نشان می دهد . خطوط نقطه چین که رویداد فروش یا همان رویداد همکاری را به یک کلاس خاص متصل می کند که شما به شکل Stereotype می بینید یک قانون انقیاد نامیده می شود که آن یک قانون از Collaboration را به یک کلاس Bind می کند که در اینجا شما این قانون انقیاد را به دو شکل می بینید . این به شکل Stereotype است تا نوع وابستگی که ما با آن سر و کار را نشان دهد و همچنین این قانون انقیاد به صورت برچسبی از قوانین به کلاسی که متعلق به آن است نشان داده می شود . پس قوانینی که به کلاس Party ملزم شده اند خریدار و فروشنده هستند ، قانونی که به کلاس Product ملزم شده است ، جنس است و غیره. بنابرین به این روش می توانید اطلاعاتی را در مورد Collaboration به Class Diagram خود اضافه کنید. و درست به همین روش شما می توانید از Collaborationها با Sequence Diagramها استفاده کنید. در Sequence Diagram شما ممکن است بخواهید اجزایی که قوانین هستند را در یک Collaboration نشان دهید این کار را بوسیله قرار دادن یک Slash قبل از حروف کوچک انجام می دهید . پس ما جلوی Slash، فروشنده داریم ، همچنین ما "جنس یک" را که نمونه ای از قانون این نوع جنس است را داریم. ما فروشنده را جلوی Slash داریم وغیره. پس این آموزش فقط نشان می دهد که شما می توانید از Collaboration در تعداد از انواع مختلفی از دیاگرام ها استفاده کنید. Collaboration ها در Class Diagram ها می توانند مفید باشند مخصوصا وقتی شما می خواهید الگو ها را در ارتباط با ساختار Class Diagram نشان دهید.

Templates در UML :

در اینجا می خواهیم در مورد Template ها صحبت کنیم . OMG یک Template را اینگونه تعریف می کند: یک عنصر پارامتر ریزی شده که می تواند برای تولید مدلهای عناصر دیگر استفاده شود. پارامترهای Template علامت یا امضاء پارامترهای رسمی که قرار است با پارمترهای حقیقی جایگزین بشوند یا اتصال های پیش فرض دیگر را تعیین می کنند. اگر شما با C++ آشنا هستید احتمالا با کلاس های پارامتر ریزی شده آشنا هستید. کلاس هایی که پارامتر هایی را مشخص می کنند که کلاس هایی که به آن Bind می شود باید آنها را تعریف کنند.



در UML کلاس های پارامتر ریزی شده یک Template نامیده می شود. یک Template به شما اجازه می دهد که یک کلاسی را تعریف می کند که با کلاس های دیگر کار می کند مخصوصا وقتی که شما نمی دانید که آن کلاس های دیگر چه چیزی دارند. پس یک Template به شما اجازه می دهد که Abstraction یا همان انتزاعی بودن را برای نوع کلاسی که Template ممکن است با آن فعل و انفعال داشته باشد را فراهم کند ، همانطور که یک Interface با اشیاء این کار رامی کند ،ولی در اینجا ما در مورد کلاس ها صحبت می کنیم. برای نشان دادن یک کلاس به عنوان یک Template شما از این علامت که در اینجا می بینید استفاده می کنید ، یک جعبه طبقه بندی شده که در گوشه سمت راست بالای آن یک مستطیل با محیط خطوط نقطه چین قرار دارد. درون مستطیل یک جایی برای پارامتر نوع عنصر قرار دارد. در اینجا ما از "T" برای نوع استفاده می کنیم. وقتی شما از یک Template استفاده می کنید این یک اشتقاق نامیده می شود و بگذارید ببینیم این چگونه کار می کند . پس بگذارید کلاس دیگری ایجاد کنیم و ما آن را "Binding Class" می نامیم و برای نشان دادن این موضوع این کلاس یا این نمونه هایی از این کلاس که از Template مشتق شده اند ، ما از این الگو از پیکان برای نمایش استفاده می کنیم پیکان Generalization، خط مستقیم با مثلث نوک تیز. حالا وقتی یک نمونه مخصوص از کلاس Template شما احتیاج دارید تا نوع حقیقی جانشین مکان T را مشخص کنید. و این Binding یک نوع به Template نامیده می شود و در اینجا می بینیم که چگونه می توانید این را در Class Diagram خود نمایش دهید. شما یک Stereotype را در اینجا برای پیکان خود نشان می دهید پس از Stereotype با علامت Bind در اینجا استفاده می کنیم. در ادامه همانطور که قبلا گفتم شما برای تعیین نوع حقیقی احتیاج دارید که آن را جایگزین مکان T در مثال خود کنید. شما این را به شکل این گرامر انجام می دهید ، در داخل براکت زاویه مانند "<" شما نوع Template را قرار می دهید و از یک خط تیره "-" و یک براکت زاویه ای دیگر ">" برای ایجاد یک پیکان استفاده می کنید سپس نوع حقیقی را می نویسید. پس این شبیه این است و این یک ویژگی از نوع حقیقی را تعیین می کند که جایگزین مکان T می شود. پس بگذارید به یک مثال نگاهی بیاندازیم تا ببینیم این چگونه کار می کند. در اینجا ما یک Template داریم که List نامیده می شود و Template ها برای کار با لیست ها خیلی خوش دست هستند ، این یکی از پر کاربرد ترین موارد استفاده برای Template ها است که آن لیست پارامترهای خودش را در این بالا دارد و این ویژگی های خودش را در این پایین دارد ، لیست کتابها چیزی است که ما آن را به Template خود Bind می کنیم ، بنابراین شما می توانید ببینید که ما یک Stereotype برای Bind داریم و همچنین خصوصیات T را که در اینجا با Book جایگزین شده است که نشان می دهد که این اشتقاق چگونه کار می کند. حالا راه دیگری وجود دارد که می توان این را نشان داد و این راه کمی پیچیده تر است . در کلاس بندی شما می توانید از این گرامر استفاده کنید ، "List" و سپس براکت زاویه ای که برای T استفاده کردید و همان پیکانی که در بالا کشیدیم "->" ، پس این همان خصوصیات مشابه که ما در بالا دیدم را دارد ، ولی در اینجا قسمت نام کلاس بندی به این شکل است و در این مثال شما می بینید که دوباره از Template مشتق شده است. Template ها پیچیده هستند ولی وقتی با طرز کار با آن آشنا شوید شما می فهمید که چقدر برای استفاده در Class Diagram ها خوش دستند به خصوص وقتی با لیست ها یا نگاشت های آنها سر و کار دارید

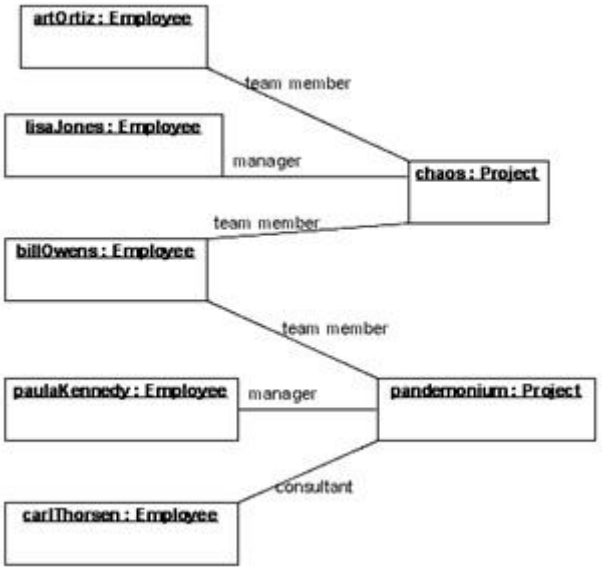
دیاگرامهای اشیاء - نمونه ها :

کلاس هایی که از اشیا تشکیل شده اند، نمونه نامیده می شوند. چیزهایی منحصربه فرد که نشاندهنده نماینده ای از کلاس است. دیاگرام های کلاس، کلاس ها را تعریف می کنند و چگونگی ارتباطشان با یکدیگر را نشان می دهند. دیاگرام های آبجکت (شی) امکان نمایش مثال های دنیای واقعی از اشیا و ارتباطشان با یکدیگر را می دهد. در نرم افزارهای Live در نهایت همه نمونه های یک کلاس برای اجرای منطق کاری استفاده می شود، نه خود کلاسها.

دیاگرام های شی که به آنها دیاگرام های نمونه هم گفته می شود، بسیار ساده تر از دیاگرام های کلاس اند. سودمندی آنها در نمایش چگونگی کارکرد یک سناریوی مشخص است. در ادامه خواهیم دید که چگونه می توان از یک دیاگرام کلاس به یک دیاگرام شی منتقل شد و روش های نمایش نمونه های یک کلاس توسط UML را به شما نشان خواهیم داد. نکته ی قابل توجه این است، برای مشخص کردن هم کلاس ها و هم اشیا از نماد کلاس (یک مستطیل) استفاده می کنیم. توجه داشته باشید که نام یک کلاس با حروف بزرگ و به صورت ضخیم (Bold) نوشته می شود. برای یک شی نام شی با حروف کوچک و با زیر خط نوشته می شود. پس هرگاه به یک نماد کلاس (مستطیل) نگاه می کنید و مطمئن نیستید که یک کلاس است یا یک شی به دنبال زیر خط بگردید. این نشان می دهد که شما با یک شی سر و کار دارید. یاد آور می شویم که کلاس ها از خصوصیات و رفتارها تشکیل شده بود. برای مثال فرض کنید کلاسی به نام اپرا داریم. همه اشیا در کلاس اپرا این خصوصیات را دارند: یک نام، یک سازنده ، یک زبان و تاریخ اولین اجرا. لذا همه اشیا درون کلاس اپرا این خصوصیات را خواهند داشت اما به جای اینکه صرفاً یک تعریف از عدد صحیح یا رشته باشند به آنها مقدار واقعی می دهیم. تا یک دقیقه دیگر روند کار را خواهیم دید. حال یکی از روش ها برای نشان دادن یک شی یا یک نمونه، این است که نام آن را قرار می دهیم برای مثال یکی از نمونه های کلاس اپرا می تواند کارمن باشد. پس شما می توانید یک شی را با نام آن شی تعریف کنید. همچنین می توانید یک شی را به همراه نام کلاس نیز تعریف کنید. این مورد بیان می کند که ما یک نمونه که با نام کارمن داریم متعلق به کلاس اپراست و همچنین می توانید یک شی بدون نام نیز داشته باشید به این شکل که نام شی را حذف کرده و صرفاً نام کلاسی که این شی به آن متعلق است بنویسید. و همچنین می توانید بگویید که این نماد به یک شی اشاره میکند نه به یک کلاس زیرا با یک : شروع شده و زیر خط نیز دارد. پس این یک شی بی نام بوده که متعلق به کلاس اپراست. همان طور که گفتم خصوصیات کلاس اپرا، دارای مقادیر واقعی منحصر به فرد در شی متعلق به آن کلاس هستند و ابزاری مدلسازی که من در اینجا از آن استفاده می کنم (Visual Paradigm) اجازه نمایش خصوصیات درون یک نمونه یا شی را نمی دهد، لذا من یک مثال را در یک برنامه دیگر ترتیب داده ام و صرفاً می خواهم آن را در اینجا کپی کنم. در اینجا یک شی مشخص متعلق به کلاس اپرا را می بینید، این شی کارمن نام دارد و توجه داشته باشید که این خصوصیات با مقادیر واقعی پر شده اند. ما نام اپرا را داریم، نام سازنده، زبان و همچنین سال اولین نمایش. به طور مشابه در اینجا می توانید رفتارهای متعلق به یک نمونه کلاس را نیز نشان دهید . برای مثال اگر یک واسط اپرا داشتیم که این متد را پیاده سازی می کرد، نام اپرا را می گرفت، می توانستید با استفاده از نمادی که یک نمونه بی نام و یک متد برای پیاده سازی آن دارد، نشانش دهید.

اتصال اشیاء :

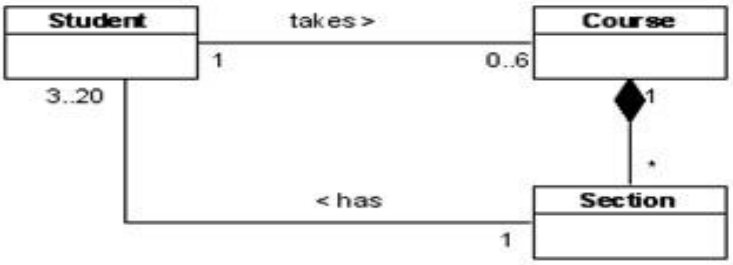
در دیاگرامهای شی تان برای نمایش چگونگی کارکرد اشیا با هم در یک سناریوی مشخص، آنها را توسط یک خط ساده به هم وصل کنید و روی این خطها برچسبی بگذارید که طبیعت ارتباط را مشخص سازد. همان طور که در مثال می بینید، 5 نمونه از کلاس کارمند و 2 نمونه از کلاس پروژه داریم و حقیقت امر این است که تعریف نمونه هایی از کلاس های مختلف اطلاعات زیادی در اختیار شما قرار نمی دهد. کاری که قرار است در دیاگرام شی مان انجام دهیم، پیدا کردن راههایی است که این کلاس ها یا اشیا می توانند از طریق آن درون این کلاس ها با یکدیگر ارتباط برقرار کنند. و همان طور که گفتم با کشیدن یک خط این کار را انجام می دهیم. ارتباط های مختلفی بین اشیا درون کلاس کارمند و اشیا درون کلاس پروژه وجود دارد. برای مثال یک کارمند (یک شی کارمند) در کلاس کارمند می تواند به بیش از یک نمونه از کلاس پروژه متصل شود، اما بهتر این است که خطوط ارتباط را برچسب گذاری کنیم. برای مثال Art Ortiz که یک نمونه از کلاس کارمند است می توانست یکی از اعضای تیم کاری در ارتباط با کلاس پروژه باشد، پس Art Ortiz یکی از اعضای تیم در پروژه Chaos است. به طور مشابه Bill Owens می تواند یکی از اعضای تیم در هر دو پروژه Chaos و Pandemonium باشد. Lisa Jones کسی که به پروژه Chaos نیز مرتبط است می توانست نقش متفاوتی نیز ایفا کند، شاید نقش یک مدیر. لذا اشیا مختلفی درون کلاس کارمند داریم که در ارتباط با کلاس پروژه نقش های مختلفی بازی می کنند.



همچنین Paula Kennedy که یک کارمند است می توانست درون پروژه Pandemonium یک مدیر باشد و یکی دیگر از نمونه های کلاس کارمند یعنی Carl Thorson نیز می توانست در نقش یک مشاور ظاهر شود. لذا با استفاده از این مثال مشاهده می کنید که چگونه با استفاده از برچسب گذاری خط ها می توانیم راههای مختلف برقراری ارتباط بین اشیا را انجام دهیم. حال دو توضیح در رابطه با ارتباط بین اشیا در دیاگرام شی تان. ارتباطات در یک دیاگرام شی همانند ارتباط بین کلاس هایی است که این اشیا از آنها نمونه گرفته شده اند. لذا اگر ارتباطی بین دو کلاس وجود نداشت، بین نمونه های آن ها هم می تواند ارتباطی وجود نداشته باشد. به طور مشابه همان طور که هیچ اجبار و محدودیتی در ارتباط بین کلاس ها وجود ندارد، در اشیا درون آن کلاس ها نیز وجود ندارد

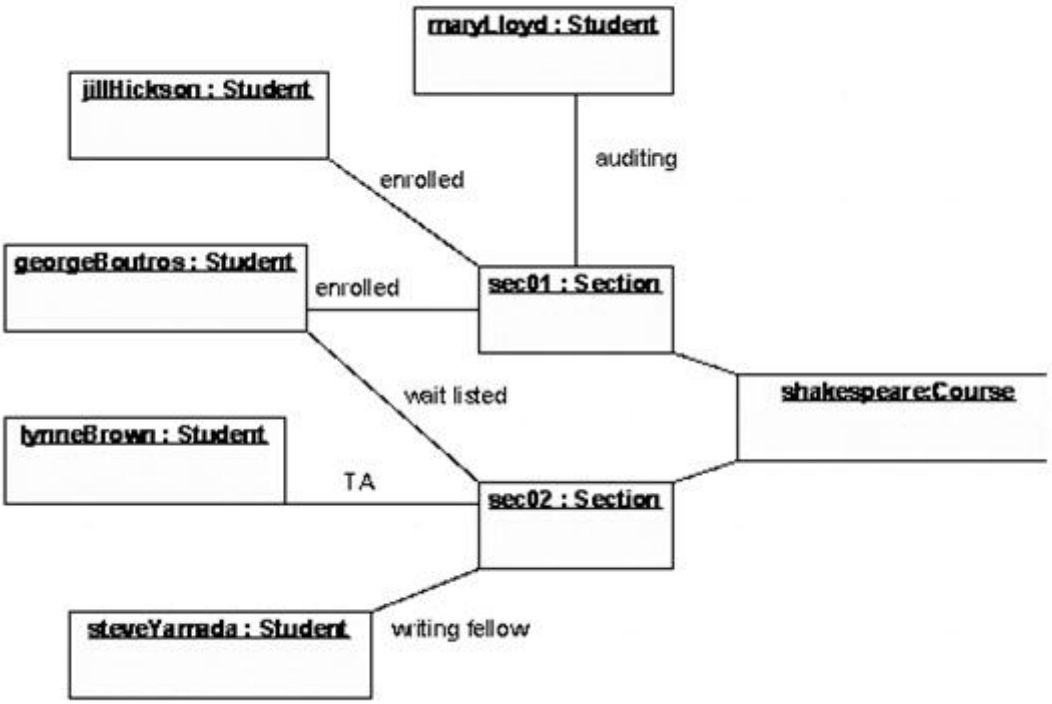
از دیاگرام کلاس به دیاگرام شئ :

مقاله شما را از دیاگرام های کلاس به دیاگرام شی می برد و نشان می دهد که چگونه دیاگرام های شی می توانند جهت تشریح کامل اطلاعات در یک کلاس کارساز باشند. دیاگرم های شی امکان تشریح ارتباطات پیچیده بین کلاس را می دهد.



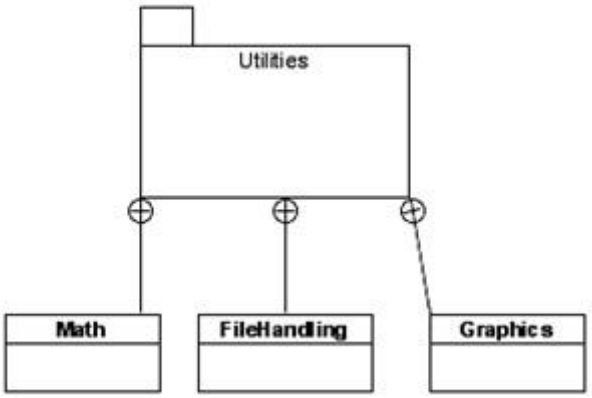
در این مثال از دیاگرام کلاس، 3 کلاس تعریف کردیم که به یک مدرسه یا یک سیستم آموزشی مرتبطند. یک درس داریم که از بخش هایی تشکیل شده است، کلاسی به نام دانشجو نیز داریم. همانطور که می بینید یک دانشجو می تواند یک درس را اخذ کند، یک دانشجو می تواند از صفر تا شش درس را اخذ کند. یک درس می تواند از گروه های مختلفی تشکیل شود و هر گروه می تواند از 3 تا 20 دانشجو داشته باشد. لذا در تعداد دانشجویان هر گروه محدودیتی وجود دارد. قبل از اینکه به چگونگی ارتباط بین این اشیا درون این کلاس ها فکر کنید، نگاهی به دیاگرام آبجکتی خواهیم انداخت که نمونه هایی از کلاس مختلف که در دیاگرام کلاس وجود دارد را نشان می دهد. ما کلاس دانشجو و اشیا مختلف از این کلاس را داریم، کلاس گروه را داریم که در این مثال دو نمونه از آن ساخته شده است و همچنین یک نمونه از کلاس درس داریم. در نگاه به این دیاگرام مشاهده می کنید که یک شی درس با نام شکسپیر چندین گروه (بخش) دارد. دو گروه مختلف و دانشجویان می توانند ارتباطات گوناگونی با این گروه ها

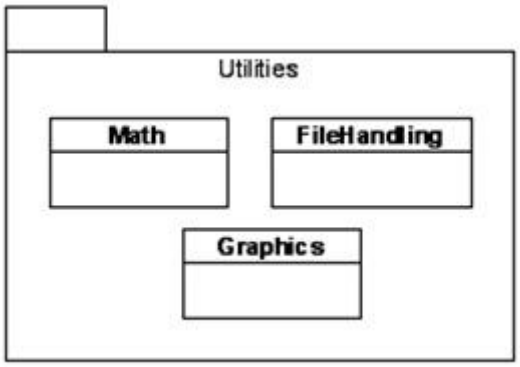
داشته باشند. برای مثل این دانش آموز، Jill Hickson در کلاس ثبت نام شده است. همچنین ممکن است یک دانشجو، George Butrose در یک گروه ثبت نام کند و در گروه دیگر در لیست انتظار قرار داشته باشد. شاید گروه 1 روزهای یکشنبه ساعت 8 صبح باشد و اما او (George Butrose) منتظر گروه 2 است که بتواند یکشنبه صبح ها بخوابد. همچنین ممکن است دانشجویی داشته باشید که مستمع آزاد باشد. نوع دیگر ارتباط بین درس و دانشجو این است که مثلا Lynn Brown دانشجو است اما در این درس دستیار استاد نیز می باشد. لذا ارتباطش با درس متفاوت از دانشجوی ثبت نامی است. همچنین به طور مشابه دانشگاه ممکن است شخصی را برای کمک به دانشجویان در نظر گرفته باشد که در کلاس ثبت نام شده تا جزء ات دانشجویان را بنویسد. لذا همانطور که می بینید با اینکه یک کلاس به نام دانشجو داریم و ابتدا به اشیا درون این کلاس نگاهی انداختیم، دیدیم که تعداد زیادی ارتباط مختلف بین اشیا درون این کلاس و کلاس درس وجود دارد. به دیاگرام کلاس خود باز می گردیم، دانشجویی را داریم که می تواند 0 تا 6 درس را اخذ کند، یک گروه از درس می تواند 3 تا 20 دانشجو داشته باشد، اما هنگامی که به نمونه های کلاس دانشجو نگاه می کنیم، متوجه می شویم که ارتباط بین دانشجو و گروه می تواند بسیار پیچیده باشد و معانی مختلفی بدهد. دیاگرام شی می تواند برای نشان دادن زوایای مختلف یک سیستم برای کسانی که احساس می کنند دیاگرام کلاس، یک دیاگرام انتزاعی است، مفید باشد و همانطور که مشاهده کردید، دیاگرام های شی می توانند در تعیین کلاسهای جدید یا روشن کردن انواع ارتباط بین کلاسها، کارساز باشند.



دیاگرامهای بسته : بسته ها :

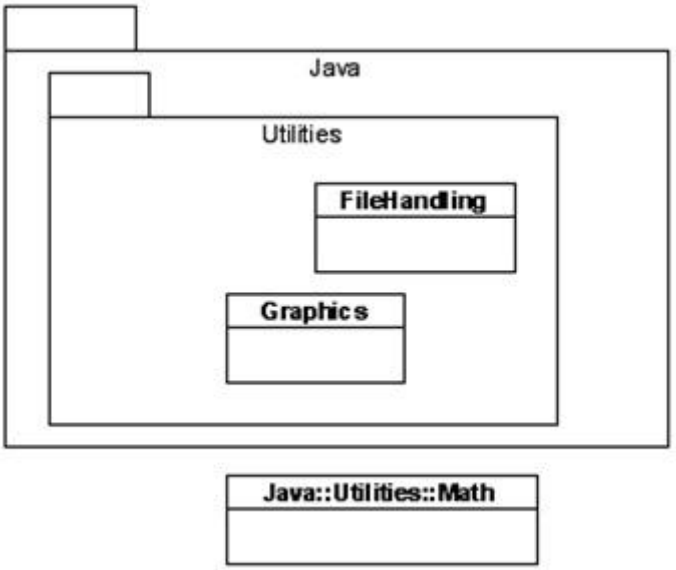
این مقاله تماما در مورد بسته ها است. ما تعریف می کنیم که آنها چه چیزی هستند ، نشان می دهیم شما چه چیزی را می توانید درون آنها قرار دهید و نشان می دهیم که آنها در UML چگونه نمایش داده می شوند. شما می توانید عناصر UML را با یکدیگر درون گروههای سطح بالایی قرار دهید و این گروهها با نام بسته شناخته می شوند. برای مثال شما می توانید کلاس های مرتبط با یکدیگر در یک Class Diagram را برای مختصر سازی و جلوگیری از در هم ریختگی گروه بندی کنید. یک بسته شبیه یک پوشه (Folder) برای فایل است و با یک مستطیل کوچک و یک Tab در بالای آن نمایش داده می شود. و این مفهوم بصری مفیدی است زیرا بسته ها عناصر را با یکدیگر گروه بندی می کند مانند یک پوشهء فایل که شامل اسناد مرتبط با یکدیگراست.



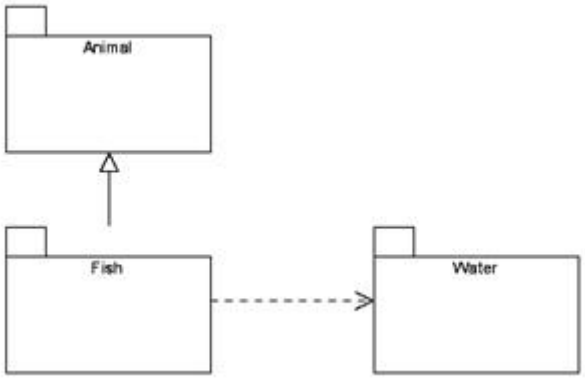


شما می توانید از بسته ها استفاده کنید تا شامل هر نوع ار عناصر در UML باشد شما احتمالا خواهید دید که آنها بیشتر در کلاس ها و Use Case ها استفاده می شوند. واین به این دلیل است که هر دوی Class Diagram ها و Use Case Diagram ها گرایش دارند تا سریع رشد کنند ، بنابراین بسته ها برای سادگی و مختصر سازی در این نوع از دیاگرام ها بسیار مفید هستند. در دیاگرام های خود شما می توانید از یک بسته که شبیه این یکی است استفاده کنید ، این یک نام دارد که داخل بسته نوشته شده است و این کاملا کافی است ، بگذارید بسته را "Utilities" بنامیم. شما می توانید کلاس هایی را نشان دهید که متعلق به یک بسته خاص است و دو راه وجود دارد که شما بتوانید این کار را انجام دهید. بگذارید کلاس هایی با نامهای "Math" ، " "File Handling، Graphics"" ایجاد کنیم. واین کلاس ها درون بسته ما که Utilities نام داشت قرار می گیرند ، یک راه که نشان دهد این کلاس ها متعلق به این بسته هستند این است که از یک متصل کننده به نام "Containment " استفاده کنیم و آن شبیه این چیزی است که در اینجا می بینید ، یک خط مستقیم با یک علامت جمع درون یک دایره در انتهای آن. علامتی که به صورت علامت جمع است به عنصری می چسبد که شامل عنصر دیگر است که به انتهای متصل کننده می چسبد. پس یک راه که شما می توانید نشان دهید این کلاس ها شامل این بسته هستند این است که از این متصل کننده استفاده کنید. یک راه ساده تر یا شاید راه شسته و رفته تر گرافیکی این است که کلاس ها را درون بسته قرار دهید وس ما می توانید این کار را انجام دهید پس بسته شما شبیه این میشود . و این به سور واضح محتویات داخل بسته را نشان می دهد نشان می دهد که کلاس های ریاضی و بررسی فایل و گرافیک هستند داخل بسته Utilities هستند . اگر شما وقتی از این روش استفاده می کنید بتوانید که نام بسته را بر روی قسمت Tab مانند آن قرار دهید این بهتر است زیرا نام بسته واضح تر می شود . ولی الگوی گرافیکی به من اجازه نمی دهد این کار را انجام بدهم . پس شما هنوز نظر مشابه قبل را می بینید. بسته Utilities این کلاس ها را دارد. شما می توانید موجودیت های داده ای را بسته بندی کنید و شما می توانید موجودیت های تجاری را منوط به این که هدف دیاگرام شما چه چیزی است بسته بندی کنید. شما همچنین می توانید بسته ها را درون یکدیگر قرار دهید برای مثال اگر ما یک بسته دیگر داشته باشیم که آن را Java می نامیم و که بسته جاوا حاوی بسته Utilities و هر انچه که داخل آن است باشد . پس ما می توانیم این ها درون یکدیگر قرار دهیم و در دیاگرام خود شما می خواهید که بسته Utilities به اندازه ای بزرگ باشد تا بتواند دیگری را در خود جای دهد . حالا خارج از نماد های این بسته اگر شما می خواهید که کلاس Math را خارج نشان دهید و نشان دهید که این متعلق به بسته Utilities است یک راه برای نمایش ان این است که از علامت :: استفاده کنید. بنابراین شما میوانید بنویسید Utilities::Math . پس به من اجازه دهید این را کمی بزرگتر از انچه می بینید انجام دهم . شما همچنین بسته های وابسه به هم را نمایش دهید که

ما این کار را جداگانه بوسیله تکیه بر نام کلاس از طریق :: انجام می دهیم . پس این می تواند در یک دیاگرام دیگر نشان دهید یا طور دیگر کلاس Math متعلق به بسته Utilities است که خود آن متعلق به بسته جاوا است.
Math::Utilities::Java وقتی شما بسته ها را از Diagram Class ایجاد می کنید به خاطر داشته باشید در کلاس همه منظوره سلسله مراتب ارث بری مشابه در یک گروه به صورت بسته بسته بندی می شود.

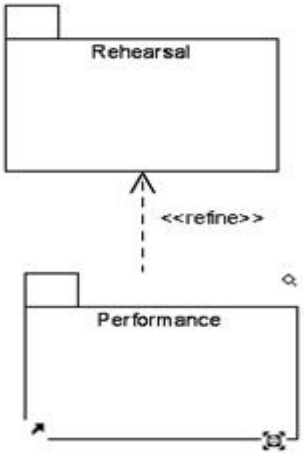


روابط بین بسته ها :
مانند دیگر عناصر UML، بسته ها هم بصورت جدا و تنها وجود ندارند. می توانید بسته ها را بهم دیگر به روشهای مختلفی ارتباط دهید. این روابط شامل وابستگی ، تعمیم یا ارث بری و بازشناسی یا پاک سازی باشد که این مقاله این باره صبحت می کند. به دنبال روشهایی خواهیم گشت که بتوانیم بسته ها را بهم دیگر در دیاگرام بسته مرتبط کنیم. خوب در گذشته استفاده از وابستگی و ارث بری را بین عناصر UML دیده اید اما نگاهی سریع با چند مثال ساده خواهیم داشت.



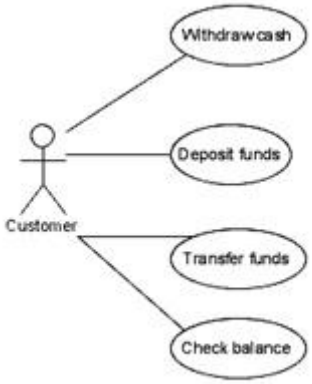
در یک وابستگی یک بسته، به بسته یا پکیجی دیگر وابسته است و یا تکیه می کند. پس در این مثال اگر بخواهیم این بسته را ماهی (Fish) بنامیم، عناصر آن ممکن است کلاسهایی متفاوتی از ماهی ها باشند؛ همچنین در اینصورت می توانیم یک بسته به نام آب (Water) داشته باشیم که کلاسهای آن، انواع مختلفی از آب باشند، مانند، رودخانه، دریاچه، اقیانوس و ... بنابراین می توانیم رابطه وابستگی را بین این بسته ها، با استفاده از فلش وابستگی برقرار کنیم. پس می توانیم اینگونه بخوانیم که بسته ماهی وابسته به بسته آب است و یا ماهی به آب احتیاج دارد. فقط جهت یادآوری، روشی که برای نمایش رابطه وابستگی استفاده می کردیم، استفاده از فلش نقطه چین با سر مثلثی باز بود که از عنصر وابسته به سمت عنصری که به آن واسته است، اشاره می کرد.
بنابراین شکل فوق، یک رابطه وابستگی است. رابطه تعمیم یافته ، بیانگر ارث بری است. پس اجازه دهید که نمودار فعلی را به پایین بفرستیم و یک بسته دیگر به نام حیوان (Animal) ایجاد کنیم. و رابطه ارث بری را با استفاده از فلشی که سر آن یک مثلث تو خالی است نشان می دهیم که از سمت فرزند به سمت کلاس پدر کشیده می

شود. پس در این مثال، اینطور می خوانیم که ماهی نوعی حیوان است. بنابراین می توانید رابطه وابستگی و ارث بری را بین بسته ها همانند دیگر عناصر UML استفاده نمایید. بازشناسی یا پاکسازی نوع دیگری از رابطه را بین بسته ها نشان می دهد. پاکسازی در جزییات مشخص می شود. یک بسته، بسته دیگر را پاک سازی می کند اگر این بسته شامل عناصری باشد که بسته دیگر هم دارا باشد اما جزییات بیشتر در مورد این عناصر را ارایه می دهد. پس می توانید رابطه پاک سازی مثل رابطه بین یک طرح و نقاشی روغن، تصور کنید. شما عناصر یکسانی را در طرح خود و نقاشی روغن دارید اما نقاشی، طرح را بوسیله رنگهای بیشتر، تعریف اشکال بیشتر، سایه و خطوط بیشتر و در کل جزییات بشتر پاکسازی کرده است و یا بهبود بخشیده است. بنابراین اجازه دهید مثالی در رابطه با رابطه Refinement بین بسته ها ارایه دهیم. به نقش بازی کردن فکر کنید. برای شروع به بازی کردن احتیاج به تمرین دارید که عناصر تمرین شامل بازیگران، کارگردان، تنظیمات، وسایل صحنه نمایش و ... می باشد. بسته دیگر که جزییات بیشتری را برای این عناصر ارایه می دهد، می تواند اجرا باشد. هنوز همان بازیگران را داریم، همان صحنه نمایش، همان کارگردان و ... اما اجرا، تمرین را بهبود می بخشد. بنابراین می توانیم فلش وابستگی را که از اجرا به سمت تمرین اشاره می کند، را رسم کنیم. پس اینطور می خوانیم؛ اجرا، تمرین را بهبود می بخشد. ممکن است متعجب شوید که چرا فلش وابستگی از بسته ای که جزییات بیشتری نسبت به بسته ای که جزییات کمتری دارد، کشیده شده است. به این دلیل است که باید اینگونه فکر کنید. اجرا وابسته به تمرین است. این سطح کاملتر جزییات، زمانی امکان پذیر است که سطح پایین تر، در لحظه اول وجود داشته است. از کلیشه کردن برای نمایش و درک بهتر استفاده کنید. شکل بالا بیانگر یک رابطه است که سطح بزرگتری از جزییات را نشان می دهد.

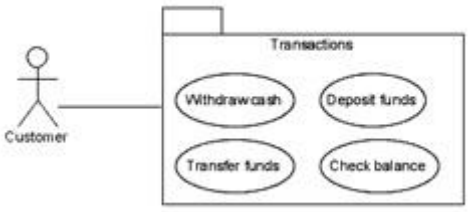


پیوند بسته ها :

در UML می توانید یک بسته را با بسته دیگر پیوند دهید. این رابطه پیچیده تر از چیزیست که به نظر می رسد. پس اجازه دهید زمانی را برای روشن شدن موضوع پیوند بسته ها درنظر بگیریم. زمانی می توانید بسته ها را با هم پیوند دهید که بخواهید بین عناصری در بسته مبدا و هر عنصری دیگری در بسته مقصد که نام یکسانی دارند، عمومی سازی (ارث بری) ضمنی تعریف کنید.



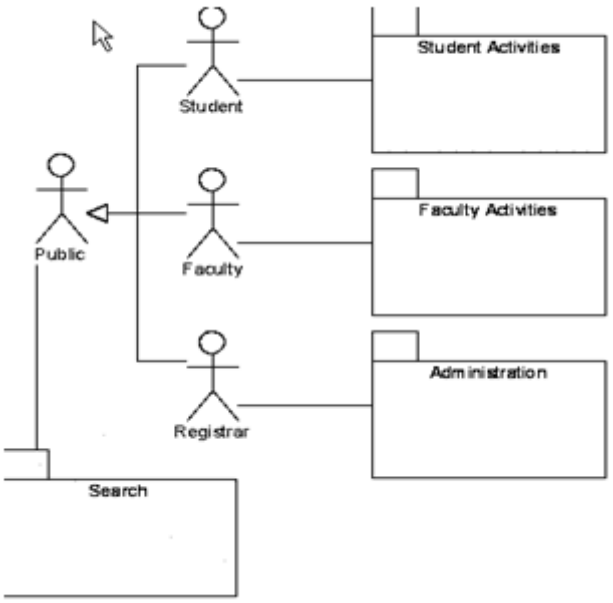
پس زمانی پیوند بین عناصر بسته مبدا با بسته مقصد تعریف می شود که عناصر نام های یکسانی داشته باشند. عناصر تعریف شده در بسته مقصد بدون تغییر باقی می مانند و هر عنصری در بسته مبدا که نام آن با نام عنصر بسته مقصد متفاوت است نیز بدون تغییر باقی می ماند. حال نگاهی به یک مثال می اندازیم تا متوجه شویم که مطالب گفته شده به چه معناست.



اما اول نگاهی به نمادها می اندازیم. زمانی که بسته ها را پیوند می دهید، 2 بسته دارید. بسته مبدا (Source) و بسته مقصد (Target). همچنین این دو بسته به نامهای بسته ی دریافت کننده (Receiving Package) و بسته پیوند خورده (Merged Package) نیز نامیده می شوند. بسته مبدا، بسته ای است که اطلاعاتی را از بسته مقصد دریافت می کند تا تعاریف و جزییات خود را کاملتر کند و این رابطه پیوند را با فلش وابستگی که با کلیشه <> نام گذاری شده است، نشان می دهیم. توجه داشته باشید که جهت فلش، از بسته مبدا به سمت بسته مقصد است. از بسته دریافت کننده به سمت بسته ی پیوند داده شده. مجددا این حالت یک نوع حالت وابستگی را نشان می دهد. مبدا وابسته به مقصد است تا بتواند جزییات و تعاریف عناصر مشترکش را گسترش دهد. پس اجازه دهید به مثالی در این رابطه بپردازیم. همانطور که در شکل زیر می بینید،

در این قسمت دو بسته داریم: مدرسان و محل هیئت علمی. بسته مدرسان شامل تعدادی کلاس است. استاد، شغل و همانطور که می بینید، یک یاچند استاد در یک شغل استخدام شده اند. کلاس استاد شامل خصوصیات و رفتارهای مختلفی است. خصوصیات شامل نام و شماره شناسایی (id) و همچنین رفتار دریافت حقوق و کلاس شغل لیستی از خواص را دارا می باشد؛ عنوان و دپارتمان. در قسمت دیگر، در بسته محل هیئت علمی نیز یک کلاس اساتید هیئت علمی داریم که از کلاس faculty در بسته Teaching staff خصوصیت id مشترک دارد و رفتار متفاوتی دارد. رفتار assign to برای محوطه دانشکده. پس کلاس Campus (محوطه دانشکده) را داریم که تعدادی از اعضای هیئت علمی در آن مشغول هستند. خوب اگر بخواهیم بسته حوزه هیئت عملی (Faculty Location) را با بسته مدرسان (Teaching Staff) پیوند دهیم، teaching staff بسته مبدا ما خواهد بود و faculty location بسته مقصد. چگونگی پیوند این دو بسته را در شکل می بینید. مجددا از فلش وابستگی استفاده می کنیم که از بسته مبدا به بسته مقصد کشیده شده است و شکل بسته نهایی را می بینید. حال یک کلاس faculty بجای دو کلاس

داریم اما می بینید که چگونه جزییات آن توسط پیوند (merge) بسط داده شده است. نام، شماره شناسایی و رفتار getSalary() را مثل قبل و همچنین رفتار assign to را داریم. و کلاس Campus را که اضافه شده است و تغییری در آن ایجاد نشده است و همینطور برای کلاس شغل یا job class. پس دیدید که اطلاعات چگونه از کلاس Faculty و مقصد اطلاعاتشان را درباره کلاس faculty در مبدا گسترش دادند این کل چیزیست که در merge اتفاق می افتد. قوانینی در مورد پیوندها وجود دارد که ارزش توجه را دارند. بنابراین نگاهی به آنها می اندازیم. 1. عناصر private بسته ها را نمی توانید با هم پیوند دهید. بنابراین اگر کلاسی دارید که سطح دسترسی private در بسته مقصد را دارد، این کلاس با کلاسی در بسته مبدا پیوند نخواهد خورد. 2. همچنین UML امکان ارث بری چندگانه را در پیوند بسته ها می دهد. 3. همچنین هر زیر بسته ای در بسته مقصد به بسته گیرنده اضافه خواهد شد و پیوند می خورد. بنابراین اگر بسته های تودرتو دارید، اگر زیر بسته هایی دارید که داخل بسته مقصد هستند و در بسته مبدا وجود ندارند، آنها به بسته گیرنده (Receiving بسته) اضافه می شوند. 4. اگر حتی در هر دو بسته مقصد و مبدا، زیربسته ها را با نام یکسان داشته باشید، دراینصورت بین آن زیر بسته ها، عمل پیوند نیز اتفاق می افتد. حال سوالی که پیش می آید این است که این قوانین کجا مفید خواهند بود. زمانی پیوند زدن مفید خواهد بود و یکی از مفیدترین روشها برای پیوند یک بسته آن است که معماری ای دارید که عملکردهایش، عناصر همنامی دارند که می توانند در معماری توسعه داده شده کنار هم قرار گیرند. پس زمانی که به پیشرفت محصول فکر می کنید می توانید از پیوند برای توسعه تعاریف و جزییات عملکردهای موجود استفاده کنید بدون اینکه تاثیری روی اشیای پیوند خورده داشته باشید.



<u>چه زمانی از بسته ها استفاده کنید</u>

حال می دانید که در نمودارهای بسته چه چیزهایی قرار می گیرند، سوالی که مطرح می شود آن است که چه زمانی از بسته ها استفاده نمایید. به این سوال اینگونه فکر کنید. یک پروژه بزرگ و پیچیده که شامل 100 ها کلاس است، بدون داشتن راهی که بتوان این کلاسها را سرو سامان داد غیر ممکن به نظر می رسد که بتوانیم آنها را از هم مجزا کنیم. به همین طریق، بخواهیم از بین این دسته کتاب یک داستان مشخص را بیابیم. بسته ها یک ساختار را برای کلاسها یا دیگر اجزای UML بوسیله گروه بندی اجزای مرتبط ایجاد می کنند. پس بجای این همه کتاب درهم، یک کتابخانه داریم جایی که کتابها برپایه موضوع یا همسانی عناوینشان به بخشهای مختلفی طبقه بندی شده اند و شما می خواهید همین عمل را با کتابخانه کلاسهایتان انجام دهید. می خواهید مطمئن شوید که اینها دسته بندی شده اند، کلاسهای مشابه با هم دسته بندی شده اند و این باعث می شود که با نگاهی اجمالی، براحتی بتوانید مطلب را درک نمایید. بنابراین جهت راهنمایی کلی، از نمودار بسته ها زمانی استفاده می کنید که بخواهید سطح بالاتری از نمایش سیستمان را داشته باشید. و ممکنه که در مورد سطح بالاتر بر حسب نیاز مرور اجمالی داشته باشیم و یا در مورد سطح بالاتری از طراحی صحبت کنیم. بسته همیشه عناصر UML را جمع و نمودارتان را پیشرفته تر می کنند. از نمودار بسته ها برای نگهداشتن رابطه وابستگی استفاده کنید. مجددا در یک سیستم بزرگ با 100 ها رابطه وابستگی برای پایداری سیستم نیاز دارید که تمام آنها را دسته بندی کرده و مدیریت نمایید

که بسته ها این کار را خیلی ساده تر انجام می دهند. همانطور که بیاد می آورید، عناصر داخل نمودار بسته ها، اساسا بسته ها و روابط وابستگیشان را نشان می دهند. پس نمودار بسته ها اجازه می دهد که روی وابستگی ها تمرکز داشته باشید. از نمودارهای بسته در یک سیستم بزرگ استفاده کنید تا بتوانید عناصر اصلی را نشان دهید و اینکه چگونه این عناصر با هم در ارتباط هستند. دوباره این کار سطحی بالاتر از سیستم را به شما ارایه می دهد. و می توانید از نمودارهای بسته برای تقسیم کردن یک سیستم پیچیده به چندین بخش (Module) استفاده نمایید. پس این روش ایده ای به شما می دهد برای زمانی که نمودارهای بسته مفید خواهند بود و همانطور که گفتم آنها مشخصا برای سیستم های بزرگ و پیچیده و دادن سطح بالاتر از کل و یا بخشی از نمودار مناسب هستند.

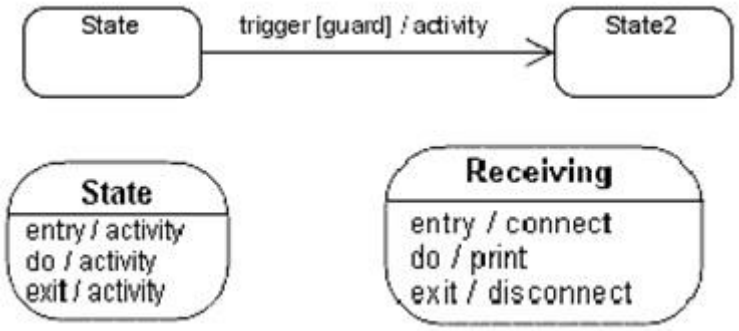
دیاگرامهای حالت- حالات و تغییر حالات :

ممکن است دیاگرام حالت هايي را دیده باشید که به چارت حالت یا نمودارهاي حالت ماشین اشاره میکنند اما هر آنچه که شما دیاگرام حالت مینامید رفتارهاي يك سیستم را نمایش میدهند. مثلا شما میتوانید يك دیاگرام حالت بسازید که با آن رفتار يك شي را در بین مجموعه اي از use case ها نمایش دهید. شما میتوانید با استفاده از دیاگرام حالت ها رفتار يك کلاس ، زیر سیستم یا کل يك سیستم را نمایش دهید. UML دو نوع دیاگرام حالت را میپذیرد ، نوع اول دیاگرامهای حالت رفتاري است و اینها پیاده سازي خاصي از عناصري مثل شي را نشان میدهند و بیشتر کلیپ هاي این بخش درباره همین دیاگرام ها صحبت خواهد کرد. نوع دوم، ماشین هاي حالت قراردادي است ؛ و اینها قرار نیست که پیاده سازي شوند.

در عوض قرارداد رویدادي را براي دنباله اي از رویدادهاي حالت و تغییر حالت ها که نشان میدهند تعریف میکنند بدون اینکه رفتار واقعي اشیا را نشان دهند و در ادامه کلیپي داریم که به طور ویژه به ماشین هاي حالت قراردادي اختصاص داده شده است. پس دیاگرام حالت ها يك تُن عنصر ندارند ، میتوانید يك حالت را همانند مثال اینجا با استفاده از يك مستطیل با گوشه هاي گرد شده نشان دهید و مستطیل باید نام حالت را که میتواند on، off،waiting، و active باشد نمایش بدهد. همچنین میتوانید نماد حالت را به تعدادي از بخشهاي مختلف تقسیم کنید و در آن بخش ها میتوانید متغیرها و یا فعالیت ها و آنهایي که انتخابي هستند نشان دهید. این فلش ها تغییر حالت ها را نشان میدهند و تغییر حالت به سادگي تغییر از يك حالت به حالت دیگر را نمایش میدهد یعني از حالت مبدا به حالت مقصد. همانطور که میبینید میتوانید انتقال حالت ها را با سه عنصر guard،trigger و effect نام گذاري کنید. به این عمل نامگذاری تغییر حالت می گویند و اختیاري است و میتوانید بخشي از آن، یا کل آن را قرار دهید و یا حتي میتوانید آن را رها کنید. عناصر مختلف نامگذاری تغییر حالت شامل trigger است که ممکن است ببینید که به يك رویداد اشاره میکند اما آنها همان معني را میدهند و trigger بطور ساده بعضي از رویدادهایي است که تغییر حالت را که باعث میشود شي از این حالت به آن حالت تغییر کند پایه گذاري میکند. بنابراین trigger ممکن است شامل يك رویداد زماني ، يك نامگذاری ، يك فراخواني ، يك تغییر باشد و بعضي رویدادها مثل اینها میتوانند يك trigger باشند. بعد guard مي آید و guard خیلي به guard condition در activity diagram ها شبیه است.
همانطوري که در اینجا میبینید يك guard را با محصور کردن آن در يك مربع نمایش دهید و guard يك شرایط بولي است که نشان میدهد transition چه اقداماتي میتواند انجام دهد. بنابراین اگر وضعیت true باشد transition میتواند اتفاق بیفتد و اگر false باشد نمیتواند انجام بپذیرد. بعد يك علامت / و به دنبال آن effect است و میتوانید ببینید که effect به يك عمل یا فعالیت اشاره میکند و دوباره همه آنها معني همان چیز را میدهند. effect رفتاري است که در طول تغییر حالت اتفاق مي افتد. چیزهایي براي شي اتفاق مي افتد که آن را تبدیل به يك تغییر از يك حالت به حالت دیگر میکند. نامگذاری trigger ممکن است شامل رویدادهاي چندگانه و پارامترها باشند و همانطور که گفتم تمام بخشهاي نامگذاری trigger اختیاري هستند. پس بیایید نگاه کوتاهي به يك مثال بسیار ساده بیندازیم ، این مثال براي يك لامپ یخچال است و این لامپ یخچال میتواند دو حالت داشته باشد ، خاموش و روشن. و میتواند به دو رویداد پاسخ بدهد ، باز شدن در و بسته شدن در و میتوانید از مسیر مستقیم transition ببینید که اگر لامپ در حالت خاموش باشد و رویداد باز شدن در رخ بدهد حالت لامپ به روشن تبدیل میشود. اگر لامپ در حالت روشن باشد و رویداد بسته شدن در رخ بدهد حالت لامپ به خاموش تبدیل میشود. بنابراین در دیاگرام حالت حالت هاي مختلف اساسا روشهاي گوناگوني که شي به رویدادها عکس العمل نشان میدهد را نمایش میدهد و به یاد داشته باشید که این موضوع ایجاد و مطالعه ي دیاگرام حالت ها را بسیار ساده تر میکند.

فعاليت هاى حالت :

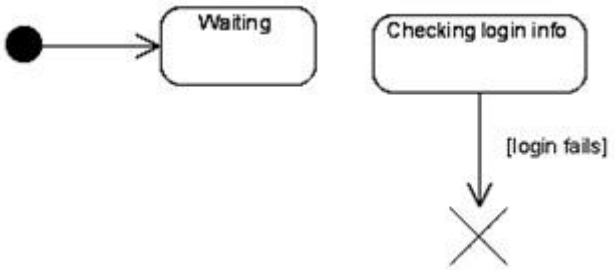
همانطور كه ديده ايد تغيير حالتها ميتوانند شامل فعل يا فعاليتى باشند و مثالي كه به كار برديم كه در آن فعلى كه موجب تغيير حالت براي لامپ يخچال مى شد باز كردن در بود كه لامپ را در يك وضعيت قرار ميداد. بنابراين تغيير حالتها ميتوانند شامل فعاليت هايي باشند و اين activity ها معمولا آنهايي هستند كه زود اتفاق مي افتند.



همچنين حالت ها هم ميتوانند داراي activity باشند. يك state ، يا فعال است يا غير فعال ، شرايطي وجود دارد كه بوسيله نام state نمايش داده ميشود كه در هر نقطه داده شده از زمان آيا اتفاق مي افتد يا خير. وقتي كه يك state فعال است ، activity هاي آن ميتوانند اتفاق بيفتند. State activity را مانند يك فرايند در نظر بگيريد ، بعضي ها كه بر حسب وظيفه بوسيله سيستم اجرا مي شوند زماني اتفاق مي افتند كه شي در بخش خاصي از state باشد. البته اين بدان معناست كه وقتي state غير فعال است activity هاي آن نميتوانند اتفاق بيفتند. اين activity اينگونه نوشته ميشود : ابتدا شما برچسب activity را مى نويسيد و بعد از آن نام آن يا عبارت آن را مينويسيد. آنها توسط يك علامت / از هم جدا شده اند.بنابراين ابتدا برچسب مي آيد ، سپس علامت / و بعد از آن نام يا عبارت activity مي آيد و از يك راه شما ميتوانيد عبارت activity را با استفاده از زبان طبيعي و يا شبه كد بنويسيد كه هر كدام حس را در متن شما ايجاد ميكند. UML سه نوع برچسب را براي activity هاي state ميپذيرد كه در اينجا نمايش داده شده اند.برچسب entry يا وروي به اين معناست كه اين activity وقتي اتفاق ميافتد كه شي وارد state ميشود و قبل از همه ي ديگر activity هاي state اتفاق مي افتد.همچنين برچسب do وجود دارد و اين برچسب نمايانگر اين است كه اين activity به مدت طولاني مثل زماني كه state فعال است اتفاق مي افتد. activity هاي do ممكن است توسط رويدادهاي ديگر دچار وقفه شوند و بالاخره برچسب exit وجود دارد كه بيانگر اين است كه اين activity وقتي اتفاق مي افتد كه شي از state خارج ميشود. يك activity exit در جايي اجرا ميشود كه activity do كاملا انجام شود يا متوقف شود و يا بي نتيجه بماند.پس آنها سه نوع از activity ها هستند كه شما ميتوانيد در state بيابيد و براي واضح تر شدن آن بياييد نگاهي به يك مثال بيندازيم.مثلا ما يك دستگاه فكس داريم.زماني كه دستگاه در وضعيت دريافت است سه state activity را دارد : activity ورودي آن connect يا اتصال است.activity نوع do آن print يا چاپ است و activity exit آن disconnect يا قطع ارتباط است. پس اگر اتفاقاتي بيفتد كه عمل تغيير حالت را متوقف كند ، در اينجا با متوقف شدن وضعيت دريافت activity نوع do ممكن است براي چاپ كردن متوقف شود و بطور كامل انجام نشود.حتي اگر اين اتفاق بيفتد activity نوع exit كه قطع ارتباط است انجام ميشود. بنابراين state activity ها به اين منظور استفاده ميشوند كه نشان دهند كه وقتي شي در يك state خاص قرار ميگيرد كدام activity ميتواند اتفاق بيفتد.

آغاز و پایان شبهه حالات :

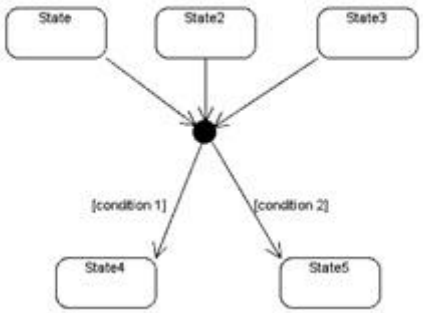
شبه حالت يك مفهوم در UML است كه به شما امكان ميدهد درباره اينكه در طول قسمت تغيير حالت چه اتفاقي مي افتد، به دياگرام حالت خود اضافه كنيد. فرق بين شبه حالت و حالت چيست؟



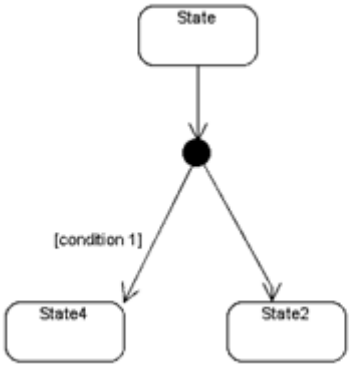
همانطور كه از نامش پيداست حالت شرايط يا وضعيتي از يك شي در دياگرام حالت است و يك شبه حالت چيزهايي را نشان ميدهد كه در مسير يك state به state ديگر اتفاق مي افتد. شما ممكن است به يك نوع نقطه تصميم گيري برسيد كه در آنجا دو تغيير حالت امكان پذير باشد و بستگي به اين دارد كه شرايط كدامشان true باشد.يا ممكن است به جايي برسيد كه چندين تغيير حالت با هم ادغام شوند يا يك تغيير حالت به چندين تغيير حالت تقسيم شود.بنابراين شبه حالت در طول زمان تغيير حالت ها اتفاق مي افتد. بياييد با نگاه به يك شبه حالت ابتدايي كه با دايره توپر نشان داده شده است شروع كنيم.اصولا وقتي درباره شبه حالت فكر ميكنيد يك state از ماشين نيست بلكه در واقع شروع نمودار را نشان ميدهد.بنابراين اگر يك دياگرام حالت داشته باشيم كه state ابتدايي آن waiting است آن را در نمودار اينگونه نشان ميدهيد . شما يك شبه حالت داريد كه نشان ميدهد چه چيزي كجا شروع ميشود.شما يك تغيير حالت از شبه حالت ابتدايي به state واقعي ماشين ميكشيد.شما ميتوانيد اين تغيير حالت را برچسب گذاري كنيد به همراه هر عمل و يا شرايطي كه لازم است به اين state ماشين ابتدايي داده شود.مشابه اين مفهوم شبه حالت نهايي است كه با علامت ضربدر X نشان داده شده است. شبه حالت پاياني ، پايان يافتن يك دياگرام حالت را نشان ميدهد مثل ورود ناموفق ، متوقف شدن power و چيزهايي كه ميتوانند منجر به پايان يافتن يك دياگرام شوند. پس اگر شما state ي داشته باشيد كه فراخواني شده است ميخواهيم ورودي اطلاعات را بررسي كنيم و شما شكست هايي در ورود داريد كه ممكن است منجر به پايان يافتن شبه حالت شود. پس شبه حالت ها روشي هستند كه نشان ميدهند در قسمت تغيير حالت در دياگرام حالت چه اتفاقي مي افتد.به شما اجازه ميدهيم كه دياگرامتان را پيچيده تر كنيد و در كليپ هاي بعدي انواع مختلف شبه حالت كه شما ميتوانيد استفاده كنيد نمايش داده ميشود.

شبهه حالت های اتصال و انتخاب :

مقاله در مورد شبه حالت اتصال و شبه حالت انتخاب است. شبه حالت اتصال يا همان Junction Pseudo state بوسيله يک دايره توپر که در اينجا می بينيد نمايش داده شده است.



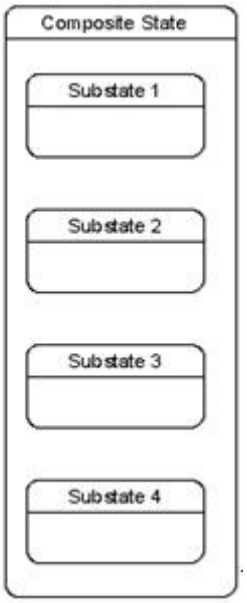
اين دقيقا شبيه نقطه آغازين شبه حالت (Pseudo State Initial) است ولی قطعا تفاوتی وجود دارد و آن اين است که "نقطه آغازين شبه حالت" در ابتدای دياگرام قرار دارد و هيچ نقطه انتقالی به آن وجود ندارد. در سوی ديگر "شبه حالت اتصال" در وسط دياگرام قرار دارد و هم به داخل و هم به بيرون از دايره توپر تعامل دارد.



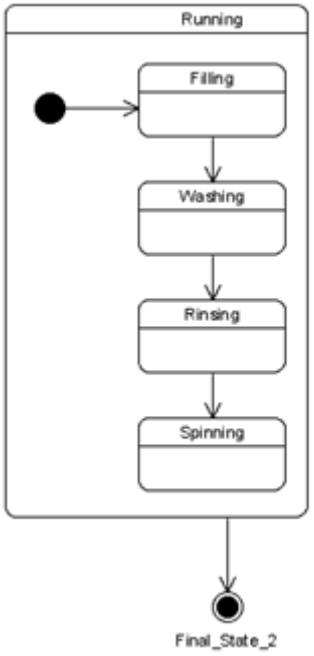
پس یک اتصال (Junction) چندین انتقال را به یکدیگر در یک شبه حالت مرتبط می کند ، بنابراین در این مثال چندین انتقال به داخل محل اتصال می توانید وارد کنید و به همین ترتیب می توانید چندین انتقال را از آن خارج کنید. شما از اتصال می توانید برای ترکیب تعدادی از انتقال های ورودی به یک یا چند انتقال خروجی بسته به انتخاب شما و مسیر دهی آن به مسیرهای اطراف از اتصال استفاده کنید. شما می توانید از این برای جداسازی یک انتقال ورودی به چندین انتقال خروجی استفاده کنید. و مطمئنا شما می توانید شرایطی را برای آن انتقال های خروجی اضافه کنید. پس تا اینجا در مورد شبه حالت اتصال بود. یک مفهوم مرتبط شبه حالت انتخاب است و در اصل یک نقطه تصمیم گیری است. "شبه حالت انتخاب" یک انتخاب از دو یا چندین حالت مختلف را مقدور می سازد ، بسته به شرایط حفاظتی از انتقال هایی که باعث می شوند از نقطه انتخاب به حالت های دیگری برسیم. پس شبه حات انتخاب بوسیله یک لوزی نمایش داده می شود و اگر شما دیاگرام های فعالیت (Activity Diagram) آشنا باشید ، این دقیقا شبیه یک نقطه تصمیم گیری در یک دیاگرام فعالیت است و لوزی یک انتقال ورودی به داخل و دو یا چند انتقال خروجی به بیرون خواهد داشت و انتقال هایی از محل شبه انتخاب خارج چندین نوع از شرایط حفاظتی را خواهند داشت. حالا مثال ما در اینجا در مورد یخچال است و ما از یک حالت تشخیص (Sense) شروع می کنیم و یخچال بسته به این شرایط حفاظتی که به دماسنج مرتبط است چه کاری را انجام می دهد. پس در شبه حالت انتخاب اگر دماسنج بالای 37 درجه باشد یخچال روشن میشود و اگر دماسنج روی 37 درجه یا پایین تر باشد جریان خاموش میشود و به حالت استراحت می رود و همانطور که شما می بینید هر دوی این حالت ها به حالت تشخیص برمی گردند. بنابراین انتخاب یک لوزی است که یک نقطه تصمیم گیری را بر پایه شرایط حفاظتی ممکن نشان می دهد.

حالت های مرکب :

پیش از این ما با حالت های ساده ای کار میکردیم. یک حالت ساده مثل این است که چه چیزی چه صدایی دارد ، این ها هیچ زیر حالتی را ندارند ، آنها به تنهایی وجود دارند. در سوی دیگر یک حالت مرکب شامل حالت های دیگر است ، شما می توانید فکر کنید یک حالت مرکب مانند یک حالت کلی است که از یک یا چندین حالت خاص ایجاد شده است و این چندین حالت خاص که درون حالت کلی قرار دارند زیر حالت (Sub State) نامیده می شوند.

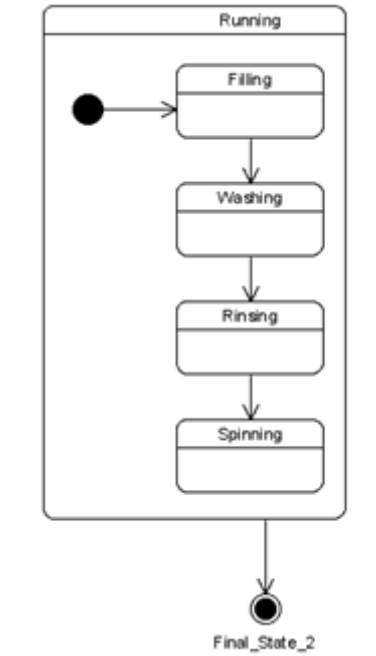


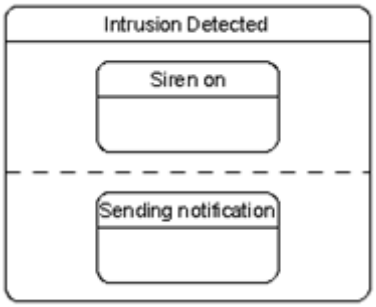
به عنوان مثال ، درمورد یک ماشین لباس شوئی ، ماشین لباس شوئی یک حالت دارد که "در حال کار"(Running) نامیده می شود ولی کار کردن فقط یک حالت ساده نیست ، این یک تعدادی از زیر حالت ها که آن را ایجاد می کنند را شامل می شود.



برای مثال زیر حالت ها برای یک ماشین لباس شوئی در حال کار می تواند شامل پر کردن با آب ، شستن ، آبکشی ، خشک کردن باشد. بنابراین حالت حالت کار کردن از این چهار زیر حالت تشکیل شده است و حالت کار ادامه پیدا می کند وقتی که هر یک از این چهار زیر حالت اتفاق بیافتد و حالت کار کردن تا زمانی که این چهار حالت تمام نشود پایان نمی یابد. وقتی شما از یک حالت مرکب در یک دیاگرام حالت استفاده می کنید بهتر است که در

ابتدا از یک شبه حالت آغازین استفاده کنید تا نشان دهید هر یک از این چهار زیر حالت می توانند حالت آغازین باشند و و سپس شما می توانید انتقال هایی میان این حالت ها یا زیر حالت ها ترسیم کنید. توجه داشته باشد روشی که با آن این دیاگرام تنظیم شده است به این شکل است این حالت ها به صورت پشت سر هم رخ می دهند ، شستشو تا زمانی که حالت پر کردن با آب کامل نشود شروع نمی شود. وقتی شستشو تمام می شود آبکشی می تواند شروع شود ، وقتی آبکشی انجام شد سپس خشک کردن می تواند شروع شود ، بنابراین این ها نمی توانند به صورت همزمان رخ دهند ، آنها به صورت پشت سر هم انجام می شوند ، پر کردن با آب آغاز می شود و وقتی تمام می شود ما می توانیم به حالت بعدی حرکت کنیم و غیره . وقتی تمام حالت ها انجام شدند ، حالت کار کردن یعنی همان حالت مرکب که آنها را در خود جای داده است نیز خاتمه می یابد بنابراین ما می توانیم علامت حالت پایانی را خارج از حالت مرکب قرار دهیم و یک پیکانی مثل این را رسم کنیم.

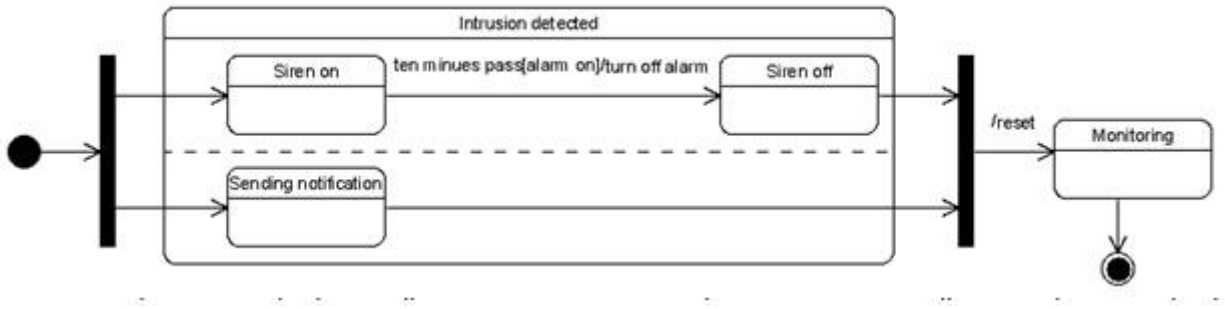




همانگونه که من گفتم وقتی تمام این زیر حالت ها تمام شوند حالت کار کردن نیز خاتمه می یابد. وقتی شما با یک حالت مرکب کار می کنید چه اتفاقی می افتد ، کدام زیر حالت ها نی توانند به صورت همزمان رخ دهند ، آیا می توانند با هم رخ دهند؟ این یک نوع خاصی از حالت مرکب است شود که یک"حالت قائم" (Orthogonal State) نامیده می شود و شما یک حالت قائم را مثل این نشان می دهید. ما دسته بندی حالت داریم ولی اسن به نواحی ای با خط نقطه چین تقسیم بندی شده است بنابراین شما می توانید زیر حالت ها را در نواحی قرار می دهید و غیره. پس وقتی شما این نقطه چین را در دسته بندی حالت مرکب خود دیدید این یک نشانه است که این زیر حالت ها می توانند به صورت همزمان رخ دهند. پس یک مثال از این مورد می تواند برای یک آژیر دزد باشد. اگر ما حالت شناسایی دزدی را دریافت کردیم ما باید صدای آژیر را به صدا در بیاوریم و ما باید اطلاعاتی را برای پلیس و مرکز آژانس امنیت و غیره ارسال کنیم. ما نمی خواهیم قبل از این که اطلاعات را ارسال کنیم صبر کنیم تا آژیر خاموش شود. ما می خواهیم اینها به صورت همزمان انجام شوند و این چیزی بود که در این دیاگرام حالت به شما نشان داده شد ، این یک حالت قائم (Orthogonal State) است که می توان دو زیر حالت را به صورت همزمان اجرا کرد.

شبهه حالت های انشعاب و پیوند :

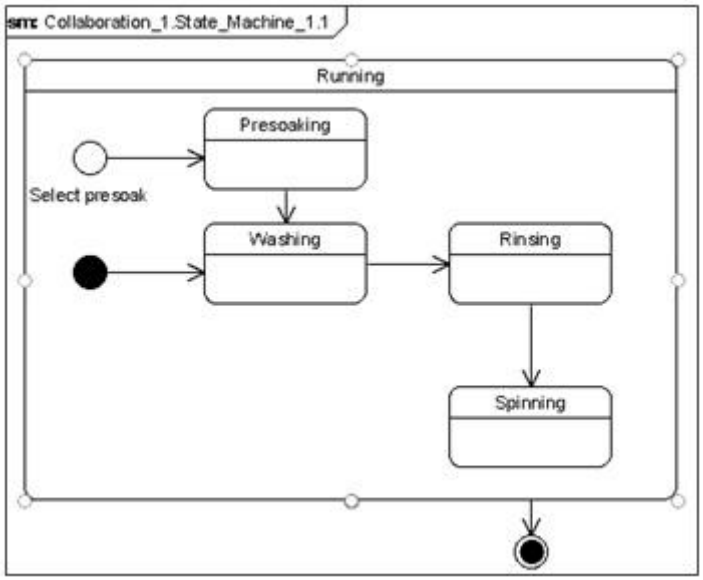
شما ممکن انشعاب یا پیوند را در هر دوی دیاگرام های حالت و فعالیت (Activity & State) بیابید. در دیاگرام های فعالیت انشعاب و پیوند "گره" (Node) نامیده می شود. در دیاگرام های حالت انشعاب و پیوند شبه حالت مطرح شده اند برای اینکه آنها یک حالت واقعی از یک شئ نیستند ، بلکه چیزی وجود دارد که باعث می شود در طی یک انتقال از یک شئ اتفاقی رخ دهد تا از یک حالت به حالت بعدی برسیم. انشعاب و پیوند بوسیله میله ای سیاه رنگ نمایش داده می شوند و آنها هم می توانند به صورت عمودی همانگونه که در اینجا می بینید و هم به صورت افقی قرار بگیرند که اگر با انشعاب سروکار داریم یک انتقال ورودی و یک یا چند انتقال خروجی داریم. اگر شما با یک پیوند سروکار دارید شما یک یا چند انتقال به صورت ورودی و یک انتقال به صورت خروجی دارید. و همانند دیاکرام های فعالیت در یک پیوند هر دوی این حالت ها باید قبل از اینکه بتوانید به حالت بعدی منتقل شوید عرضه شوند. این مشابه حالت مرکب قائم که ما قبلا در مورد آن برای سیستم آژیر دزد و حالت شناسایی دزدی صحبت کردیم ، هر دوی این حالت ها (انشعاب و پیوند) می توانند به صورت همزمان انجام بگیرند. آژیر روشن است و سیستم پیغامی را ارسال می کند که هم اکنون یک دزدی شناسایی شد ، بنابراین هر دوی اینها ، یعنی این دو حالت به صورت همزمان به صورت قسمتی حالت شناسایی دزدی با هم رخ می دهند. پس ما یک انشعاب و یک پیوند را برای نمایش این موضوع اضفه می کنیم.



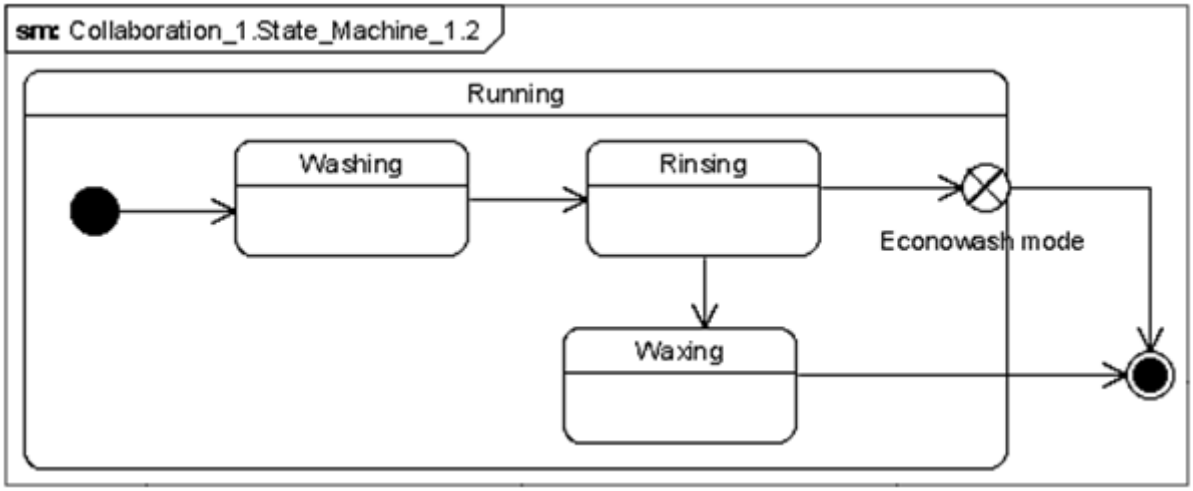
این معنی همزمانی قوی تری را می دهد ، ما شبه حالت آغازین را داریم ، ما این دو (آژیر روشن و ارسال اطلاعات) زیر حالت را برای حالت مرکب شناسایی دزدی را داریم و انشعاب در اینجا به ما نشان می دهد که در واقع اینها به صورت همزمان رخ می دهند. آژیر روشن است و بعد از 10 دقیقه اگر آژیر هنوز روشن است سیستم آژیر را خاموش می کند ، و به حالت آژیر خاموش می رود ، وقتی هر دو حالت های آژیر خاموش و ارسال اطلاعات به طور کامل رخ داد سپس سیستم می تواند به حالت راه اندازی مجدد (Reset) برگردد. این نمی تواند به حالت ابتدایی برود اگر هم اکنون اطلاعات ارسال شده باشد و آژیر هنوز روشن باشد ، ما باید به حالت آژیر خاموش برویم قبل از اینکه سیستم را راه اندازی مجدد کنیم و سپس به حالت دیده بانی برود که در این دیاگرام ساده می تواند حالت پایانی باشد. پس شما دیدید که شبه حالت های انشعاب و پیوند ما چگونه برخی اطلاعات اضافی را اضافه می کند ، اولین این بود که هر دوی اینها در یک زمان رخ می دهند و سپس هر دوی باید حاصل شوند قبل از این که شبه حالت بتواند به حالت راه اندازی مجدد برویم و به حالت بعدی برسیم. پس دوباره می گوئیم که شبه حالت ها ، حالت های واقعی از یک ماشین نیستند ولی چیزی که آنها از یک جهت انجام می دهند ، تکه تکه کردن انتقال ها میان حالت های واقعی است.

نقاط ورودی و خروجی شبهه حالات :

وقتی یک حالت ترکیبی را که زیر حالاتش به ترتیب اتفاق می افتند، مدل می کنید، همیشه از شبه حالت اولیه برای نمایش اینکه کدام حالت اول اتفاق می افتد استفاده می کنید، اگرچه گاهی اوقات امکان ورود یک حالت ترکیبی یا یک زیر حالت از چندین نقطه وجود دارد.



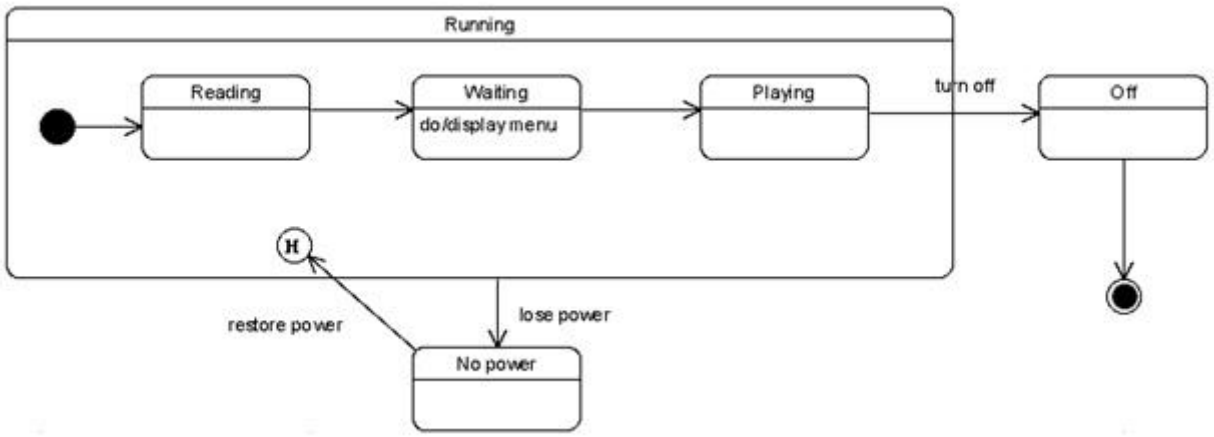
نقطه ورودی شبه حالت امکان نمایش دیگر نقاط ورودی و عبور را فراهم می آورد. در این مثال یک حالت ترکیبی برای ماشین لباسشویی داریم که به آن حالت کار کردن (Running) می گوییم. در اینجا شبه حالت ابتدایی و زیرحالت هایی اتفاق می افتند که قبلاً دیدیم مثل شستن، آبکشی، چلاندن و حالت نهایی. بعضی از ماشین های لباسشویی ابتدا عمل خیس کردن لباس را انجام می دهند لذا اگر بخواهیم یک زیر حالت با نام خیس کردن لباس بسازیم، می توانیم آن را به عنوان یک نقطه ورودی اختیاری برای حالت کار کردن ماشین لباسشویی در نظر گرفته و با استفاده از نماد نقطه ورودی که به شکل یک دایره تو خالی است نمایش دهیم. ما نام آن را انتخاب عمل خیس کردن می نامیم که البته دلیلی که شما نقطه ورودی خود را نام گذاری می کنید این است که مشخص کنید طبق چه شرایطی از این نقطه ورودی اختیاری به جای نقطه ورودی اصلی استفاده می کنید. در حالت عادی در هنگام شروع از نقطه ورودی اصلی (دایره مشکی تو پر) ما با حالت شستن شروع می کنیم و تحت این شرایط (یعنی عمل خیس کردن لباس) ما از حالت خیس کردن شروع خواهیم کرد و برای نمایش آن از یک خط استفاده می کنیم. یک خط از خیس کردن به زیر حالت بعد یعنی شستن می کشیم. لذا همانطور که می بینید دو راه برای ورود به حالت ترکیبی کارکردن ماشین لباسشویی وجود دارد. شما همچنین می توانید از نقطه ورودی اصلی (دایره مشکی تو پر) وارد شوید، که ما آن را به عنوان دایره عادی می شناسیم زیرا شبه حالت ابتدایی شستن، آبکشی و چلاندن را جز حالت پیش فرض در نظر می گیرد و ما می توانیم از طریق ورودی انتخاب عمل خیس کردن به زیر حالت خیس کردن رفته سپس به شستن، آبکشی و چلاندن برویم. بیاد داشته باشید که اگر از نقطه ورودی شبه حالت استفاده می کنید حتماً آن را نام گذاری کنید. از طرفی می توان بیش از یک نقطه خروجی را نشان داد. ممکن است بتوان یک حالت ترکیبی را از طریق چندین زیر حالت ترک کرد. در اینجا مثالی از یک کارواش داریم که حالت پیش فرض و عادی ترتیب زیر حالت ها شستن، آبکشی کردن و برق انداختن (Waxing) و سپس خروج از حالت کار کردن (Running) کارواش می باشد. البته یک نوع دیگر شستن ماشین وجود دارد که نام آن را شستن مقرون به صرفه (Economic Wash) می گذاریم و شما می توانید فقط حالت های شستن و آبکشی کردن را داشته باشید.



لذا ما آبکشی را یک نقطه خروجی به این شکل در نظر می گیریم، که نماد آن یک دایره می باشد که داخل آن یک علامت ضربدر (X) است و دقیقاً همانطور که نقطه ورودی خود را نام گذاری کردید، نیاز دارید که نقطه خروجی را هم نام گذاری کنید و نشان دهید که تحت چه شرایطی از این نقطه خروجی اختیاری به جای نقطه خروج اصلی و پیش فرض استفاده می کنید. لذا با استفاده از یک خط حالت آبکشی را نقطه خروج EconoWash وصل می کنیم. بنابراین حالت پیش فرض ما شامل شستن، آبکشی کردن و برق انداختن و در نهایت خارج شدن از حالت ترکیبی است و حالت شستن مقرون به صرفه ما فقط شامل شستن، آبکشی کردن است. نقاط ورودی و خروجی به شما انعطاف پذیری زیادی در دیاگرامهایتان می دهند. در نتیجه می توانید وضعیت ها و شرایطی به جز وضعیت پیش فرض که ممکن است در ورود به حالت ترکیبی یا خروج از آن اتفاق بیفتد، نشان دهید.

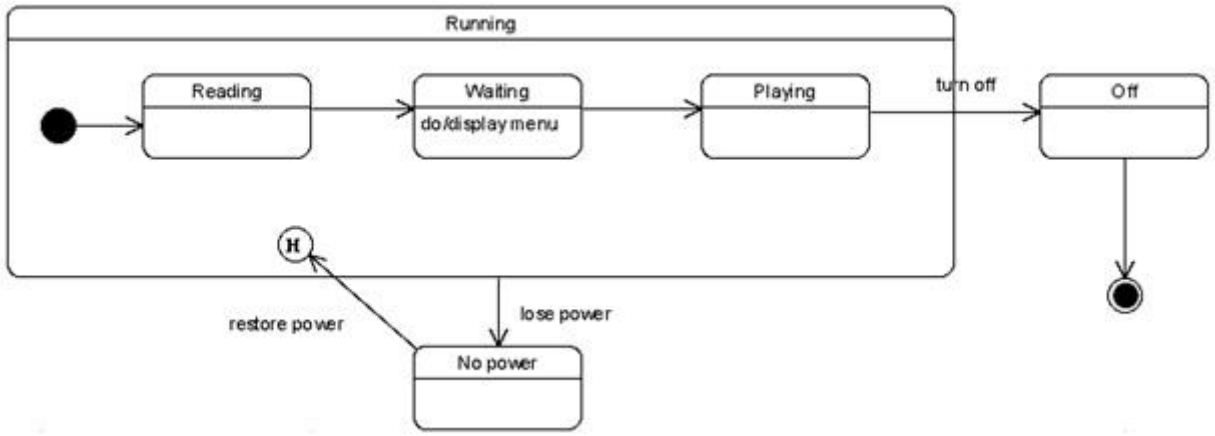
سابقه عمیق و سطحی شبهه حالات :

در این مقاله ، به نوع دیگری از شبه حالت یعنی شبه حالت سابقه ای نگاهی خواهیم انداخت. شبه حالات سابقه ای نوعی هستند که شبیه نشانه گذاری کتاب، حالات گذشته را بیاد می آورند. لذا اگر وقفه ای در یک حالت ترکیبی ایجاد شود (حالت ترکیبی منقطع شود)، شبه حالت سابقه ای امکان بازگشت به یک زیر حالت نشانه گذاری شده را می دهد. دو نوع شبه حالت سابقه ای وجود دارد: سطحی و عمیق. باز هم شبیه نشانه گذاری کتاب، شبه حالت سابقه ای عمیق، شما را به صفحه از کتاب که در آن بودید بر می گرداند در حالی که شبه حالت سابقه ای سطحی شما را به اول آن فصل بر می گرداند. به عبارت دیگر سابقه ای عمیق این شما را به نزدیکترین زیر حالت بر می گرداند و مهم نیست که در چه عمقی از حالت ترکیبی قرار دارد. سابقه ی سطحی شما را به دورترین حالت (یا زیر حالت ابتدایی) بر می گرداند.



برای مثال نگاهی به مدل کردن یک دستگاه پخش DVD می اندازیم. در حالت ترکیبی کار کردن (Running) برای دستگاه پخش DVD، زیر حالتهایی با عنوان خواندن، انتظار و پخش کردن داریم. رویداد خاموش کردن موجب خروج ما

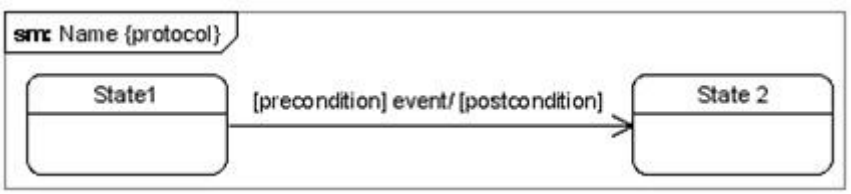
از حالت کارکردن به حالت خاموش که آخرین حالت ماست، می شود. اگر در حالت کارکردن به یکباره برق دستگاه قطع شود چه اتفاقی می افتد؟



رویداد خاموش شدن دستگاه نه، قطع برق. اجازه بدهید یک حالت با نام بدون برق ایجاد کنیم و یک خط از حالت کارکردن به حالت بدون برق وصل کرده و آن را قطع برق بنامیم. لذا در مقابل رویداد خاموش شدن (در تضاد با رویداد خاموش شدن)، رویداد قطع برق قرار دارد. اگر از سابقه ای سطحی استفاده می کنید با یک دایره و حرف H در داخل آن نمایش داده می شود. در هنگام استفاده از شبه حالت سابقه ای سطحی زمانی که برق دوباره وصل شد، همه چیزها به حالت اولیه یا اولین زیر حالت درون حالت ترکیبی بر می گردد. لذا دستگاه به حالت خواندن و سپس به حالت انتظار و هنگامی که یک رویداد را دریافت کرد، به حالت پخش خواهد رفت و همین طور الی آخر. سابقه ی سطحی همه چیز را به اولین زیر حالت و در واقع به دورترین زیر حالت در حالت ترکیبی بر می گرداند. در حالی که سابقه ای عمیق که با یک دایره و H* در داخل آن نشان داده می شود، هنگامی که برق دوباره وصل شد به جای اینکه شما را به دورترین زیر حالت برگرداند، به زیر حالتی که هنگام قطع برق فعال بود هدایت می کند. برای مثال هنگامی که در حال تماشای یک مقاله هستید، سابقه ای عمیق شما را به همان جایی از مقاله بر می گرداند که برق قطع شد. بنابراین سابقه ای عمیق می تواند شما را به هر زیر حالتی در حالت ترکیبی که رویداد قطع برق اتفاق افتاد برگرداند. در نهایت لازم به یادآوری است که شبه حالت سابقه ای همانند نشانه گذاری یک کتاب عمل می کند، سابقه ای سطحی شما را به زیر حالت ابتدایی و سابقه ای عمقی شما را به زیر حالتی که هنگام وقوع وقفه فعال بود بر می گرداند.

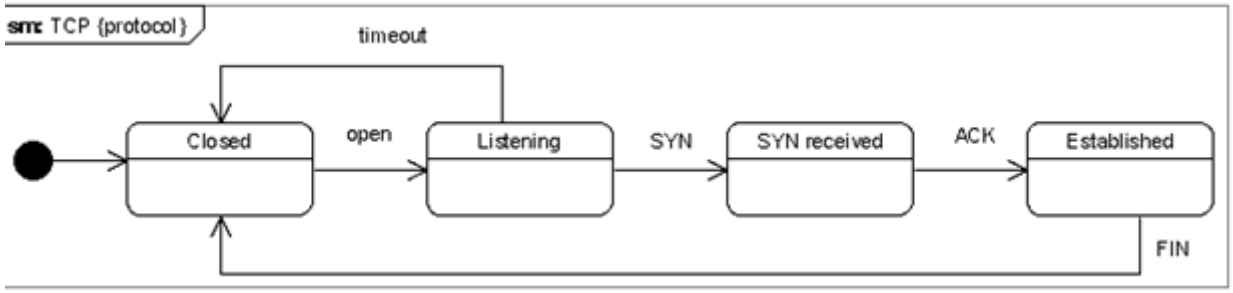
ماشین های حالت قراردادی :

همانطور که دیدید ماشین های حالت رفتاری]لازم به ذکر است که دو نوع ماشین های حالت مورد بررسی قرار خواهند گرفت: 1- رفتاری 2- قراردادی[، رفتار برخی از المانها مثل یک شی یا یک کلاس را تحت شرایطی خاص نشان می دهد. ماشین های حالت قراردادی را نیز می توانید به همین شکل مدل کنید. بجای مدل کردن رفتار یک المان، ماشین های حالت قراردادی ترتیبی از حالت ها و تغییر حالت ها (که با خط یا فلش نمایش داده می شوند) را نشان می دهند. لذا ترتیبی از رویدادها و حالتهایی که به علت آن رویدادها اتفاق می افتند را می بیند نه رفتار المان ها را.



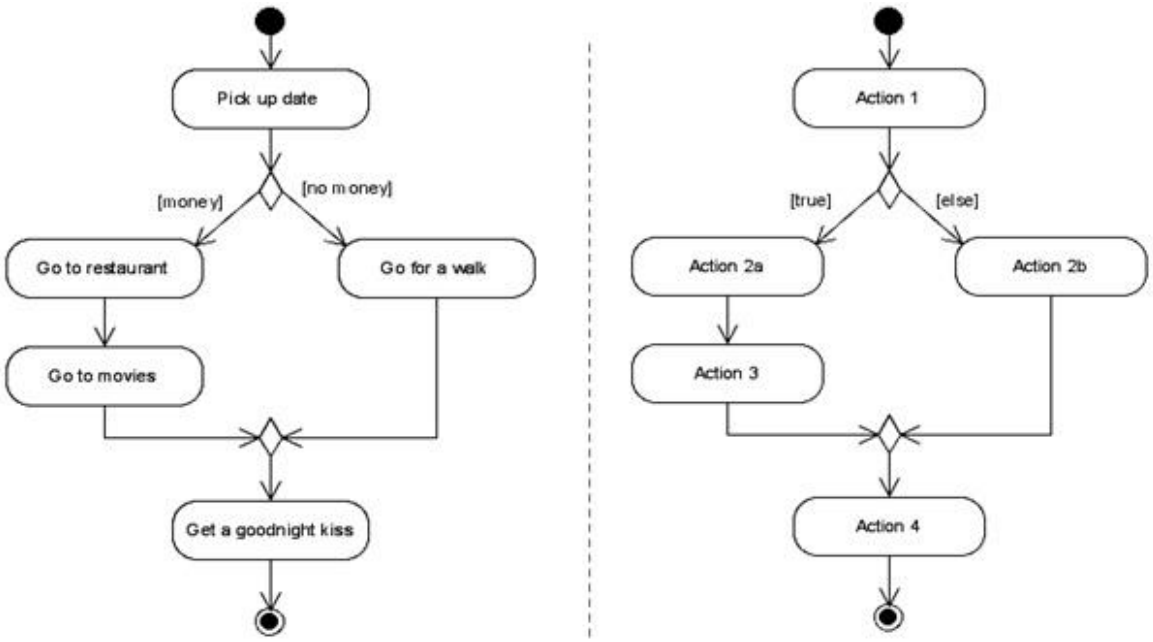
یعنی ماشین های حالت قراردادی ، فعالیت المان ها را روی خطوط تغییر حالت یا درون مستطیل های حالت (نماد

نمایش یک حالت) نشان نمی دهد. لذا در ماشین های حالت قراردادی فعالیتهای ورودی و خروجی پیدا نخواهید کرد. همانطور که در ماشین های حالت رفتاری روی خطوط تغییر حالت Trigger، Guard، و Activity می نوشتید، در اینجا به جای آنها این گونه عمل می کنیم که روی خطوط تغییر حالت به فرم زیر برچسب می نویسیم:
[Precondition] Event/ [PostCondition] که عبارات داخل کروشه اختیارند. هرگاه در حال نوشتن یک ماشین حالت قراردادی هستید، کل دیاگرامتان را درون یک مستطیل قرار دهید که در بالای سمت چپ آن، نام دیاگرام و در کنارش درون یک آکولاد کلمه کلیدی Protocol نوشته شده باشد. این عمل بیانگر آن است که ماشین حالتی که نمایش می دهید یک ماشین حالت از نوع قراردادی است. اجازه دهید نگاهی سریع به یک مثال داشته باشیم. این یک ماشین حالت قراردادی است که ساده شده یک پروتکل انتقال است. شما می توانید یک سری از حالت ها مثل بسته شدن (Closed)، گوش دادن، همسان سازی، دریافت و انتشار و همچنین خطوط انتقال حالت بین را نیز مشاهده کنید. توجه داشته باشید که در این مثال فقط نام حالت را نوشتیم و هیچ نوع فعالیت درونی نداریم. هیچ نوع فعالیت ورودی و خروجی نیز وجود ندارد و خطوط تغییر حالت هم بسیار ساده نام گذاری شده اند. لذا چیزی که این شکل نشان می دهد ترتیبی منظم از حالت ها و خطوط بین آنهاست. توجه کنید که قرار نیست ماشین حالت قراردادی پیاده سازی شوند، اما به سادگی ترتیب حالت و خطوط بین آنها را نمایش می دهند. لذا ماشین های حالت قراردادی مشابه واسط ها عمل می کنند و این دلیلی است که شما می خواهید از آنها (ماشین های حالت قراردادی) استفاده کنید.

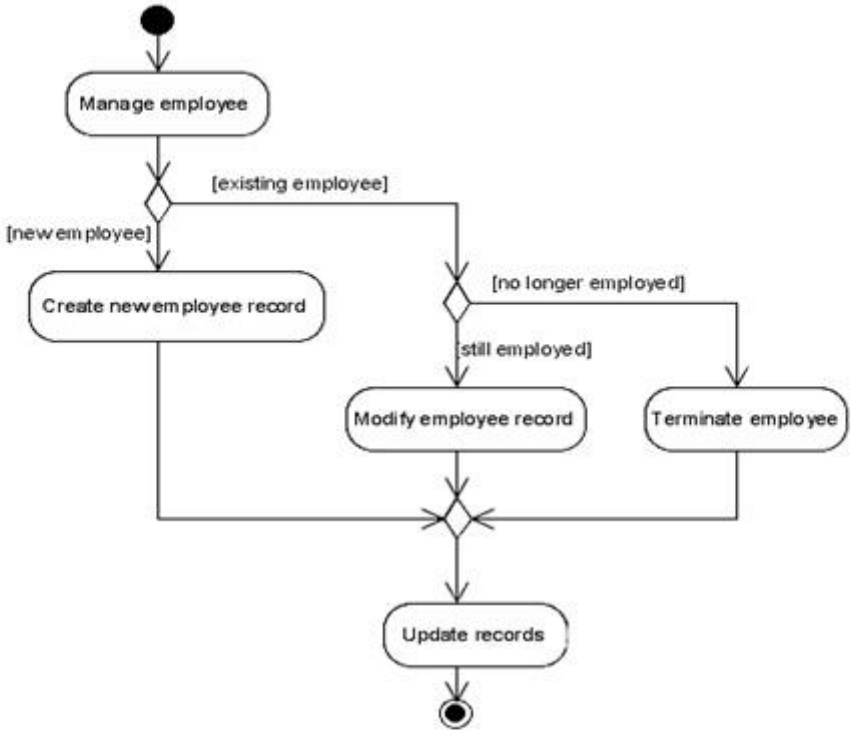


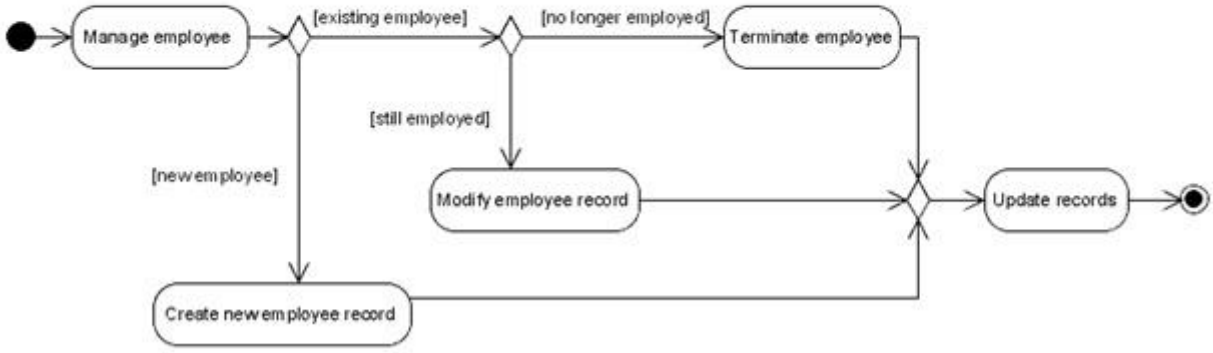
نمادهای پایه های دیاگرامهای فعالیت :

دیاگرام فعالیت اتفاقاتی را که در طول یک پروسه یا عملکرد رخ می دهد را نمایش می دهد. دیاگرام فعالیت شبیه نمودار جریان است. پس اگر شما با نمودار جریان کار کرده باشد، اطلاعات اولیه راجع به آن را دارید. شما ممکن است از دیاگرام فعالیت برای اکتشاف یک پروسه کاری، یک پروسه نرم افزاری، و اینکه چه اتفاقی در یک Use Case خاص یا در میان چند Use Case می افتد، استفاده کنید. در اینجا یک مثال بسیار ساده از نمودار فعالیت برای شما آماده کرده ایم، پس اجازه بدهید نگاهی به قسمت های مختلف آن بیندازیم. نقطه شروع را با یک دایره نمایش می دهیم که گره ابتدایی نامیده می شود.



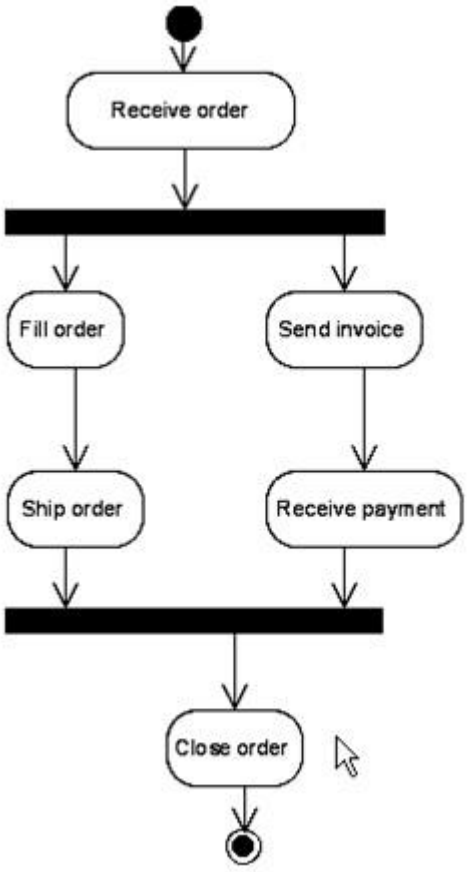
یک فعالیت از یک سری عمل تشکیل شده است. یک Activity یک پروسه است و هر مرحله در این پروسه یک عمل است ودر UML فعالیت های خاص در مستطیل با لبه های گرد نمایش داده می شوند. ارتباط بین گونه های مختلف عمل با فلش انجام می شود.توجه داشته باشید که فلش جهت حرکت را نمایش می دهد و هر مرحله در پروسه را به مرحله بعد می برد.و این فلش ها Control Flow نامیده می شوند. در هر پروسه امکان دارد به نقاط تصمیم برسید که به آنها گره های تصمیم گفته و با یک لوزی نمایش داده می شود. نقاط تصمیم نقاطی هستند که از آنجا دیاگرام چند شاخه (چند راهه) می شود. راه وابسته به تصمیمی است که در اینجا گرفته می شود. توجه داشته باشید که هر جریان که از گره تصمیم خارج می شود یک موقعیت محافظت شده است که با آن اضافه شده است و یک موقعیت محافظت شده اساسا یا درست است یا غلط . اگر درست باشد شما در طول مسیر حرکت می کنید و اگر درست نباشد شما باید مسیر دیگری را انتخاب کنید. وضعیت محافظت شده همان طور که دیدید معمولا در کروشه ([]) نمایش داده می شوند. پس ما یک نقطه تصمیم گیری و 2 راه مختلف که باید انتخاب کنید داریم. این دو راه مختلف در یک لوزی دیگر که به آن Merge گفته می شود ملحق می شوند. تعدادی از جریانات مختلف می توانند همه به یک فعالیت هدایت شوند و بهترین راه برای نمایش آنها در UML این است که این جرایانات در یک نقطه به هم ملحق شوند و این نقطه به سوی عمل بعدی حرکت کند. در پایان ما نقطه پایانی را داریم که به آن Final Node گفته می شود که نمایان گر پایان پروسه می باشد. خوب، با مفاهیم آشنا شدیم. اجازه بدهید به یک مثال نگاهی بیندازیم. پروسه ما خارج شدن از شب شنبه و عمل اول، انتخاب روز است. هر چند که کارهای شنبه شب که انجام می دهید نقطه آغازین است. حال به یک نقطه تصمیم می رسیم و تصمیم ما گرفتن پول یا نگرفتن پول است. اگر شما پول بگیرید شما می توانید چند عمل را انتخاب کنید. بروید به رستوران و ممکن است بعد از آن به سینما بروید. اگر پولی دریافت نکنید شما راه های دیگری مثل قدم زدن دارید. روز کم ارزش خوب! خوب شما می توانید راه جای گزین دیگری را ببینید اگر وضعیت داشتن پول درست باشد شما می توانید این کار ها را انجام دهید. اگر وضعیت نداشتن پول درست باشد شما می توانید این کار ها را انجام دهید. هر مسیری که انتخاب کنید در یک نقطه به هم ملحق شد و در نهایت به پایان روز می رسند. ما با آموزش نماد ها به شما راه خواندن دیاگرام فعالیت را بدون زحمت آموزش دادیم. چیزی که اینجا داریم یک پروسه برای منابع انسانی دپارتمان است و طی آن مدیریت کارمندان را داریم و با گره آغازین در بالا شروع می کنیم و کاری که می خواهیم انجام دهیم مدیریت کارمندان است. وقتی شما قصد مدیریت کارمندان را دارید، به سرعت به نقطه تصمیم می رسید. آیا در مورد کارمند موجود صحبت می کنیم؟ یا کارمند جدید است؟ اگر در مورد کارمند جدید صحبت می کنیم ما نیاز داریم تا یک رکورد کارمند جدید بسازیم. اگر در مورد کارمند موجود صحبت می کنیم به سرعت وارد نقطه تصمیم دیگری می شویم: آیا هنوز کارمند مشغول کار است یا از دپارتمان رفته است؟ اگر کارمند هنوز در دپارتمان کار می کند می توانیم اطلاعات کارمند را اصلاح کنیم و در غیر این صورت کارمند را پایان می دهیم. هر کدام از این عمل ها را که انتخاب کنید به یک نقطه الحاق می رسید. که میتواند شما را به مرحله بعدی هدایت کند. پس اگر ما یک کارمند جدید درست کنیم، به نقطه الحاق می رویم. به سوی مرحله بعد حرکت می کنیم. نموداری که در مورد آن صحبت می کردیم به صورت افقی بود.اگر دوست داشته باشید می توانید به صورت عمودی نیز آن را نمایش دهید. در شکل زیر دیاگرام فعالیتی است که قبلا به آن نگاه انداخته بودیم که فقط جهت آن افقی شده است.

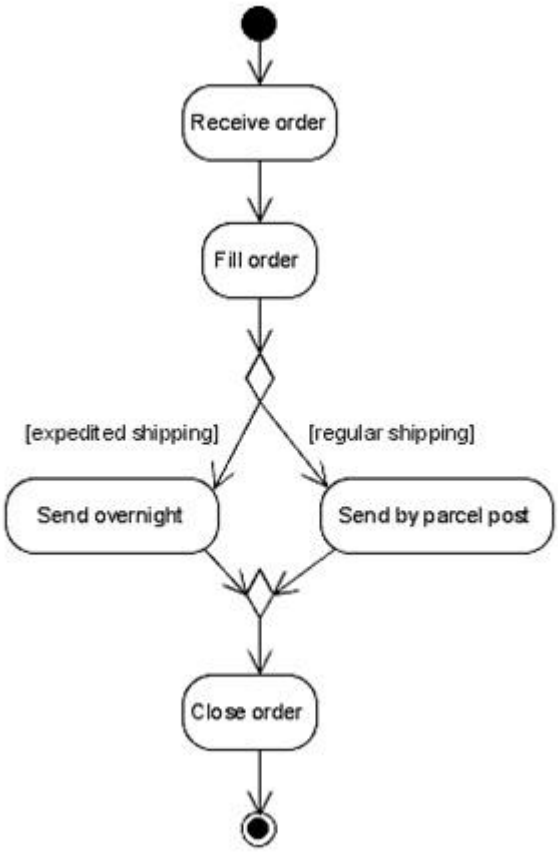




راههای جایگزین و موازی :

همان طور که در مقاله مربوط به نماد های پایه ای و نمودار فعالیت دیدید، نقاط تصمیم و نقاط الحاق راهی است برای نمایش اینکه چگونه یک جریان به چند راه تقسیم و دوباره در یک نقطه به هم ملحق می شود. در این مثال وقتی یک سفارش(Order) را پر می کنید، به نقطه تصمیم می رسیدید و آیا مشتری هزینه سرعت بخشیدن به ارسال کالا را پرداخت کرده است یا خیر. اگر پرداخت کرده بود همان شب برای آن ارسال میکنید و اگر پرداخت نکرده بود شما آن را با بسته پستی ارسال می کنید. وقتی به نقطه الحاق می رسید یک بار که بسته را ارسال کرده باشید می توانید به مرحله بعد بروید که بستن Order است. شما نمی توانید یک سفارش را با 2 متد ارسال کنید. شما هر کدام را به تنهایی انتخاب می کنید. حال چه اتفاقی می افتد وقتی به نقطه ای بروید که آنجا راه های موازی بتوانند خارج شوند؟ در این حالت از چیزی با نام چنگال (fork) استفاده می کنیم (همانند نمودار Fork and Join). گره چنگال نمایش می دهد که یک جریان وارد می شود و 2 یا چند جریان خارج می شود اما در این مورد 2 یا چند راه با هم(هم زمان) خارج می شوند که می توانند با هم اتفاق بافتند. اجازه بدید نگاهی به یک مثال بیندازیم. وقتی یک سفارش می آید ، سفارش در یک عمل ظاهر می شود.همان طور که در دیاگرام دیگر دیدید. آنها سفارش را پر می کنند و سفارش را ارسال می کند و آن را در جریانی قرار می دهند که نمایش می دهد چگونه یک مرحله به مرحله بعد هدایت می شود.



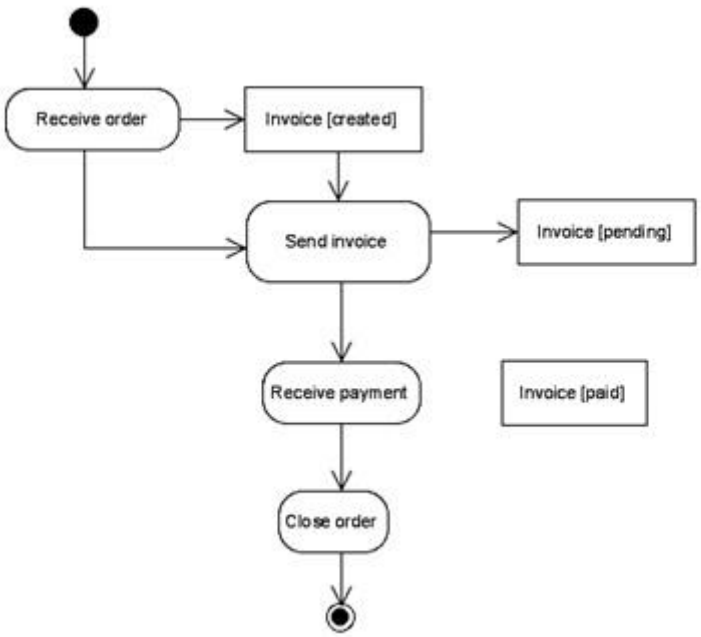


برای سفارش حمل ونقل میتوانیم یه نقطه انتخاب نمایش دهیم که آیا عادی است یا با سرعت. پس هنگام دریافت یک سفارش، تکمیل سفارش تنها دامنه ای نیست که وارد بازی می شود. همچنین ما مردم را داریم در حساب ها که کارهایی را انجام می دهند.ممکن است فاکتور ارسال کنند ، وجه دریافت کنند و دوباره آن را در جریانمان قرار می دهیم برای نمایش این که سفارش در کدام یک از مراحل است. به 2 را ه مختلف وابسته به هم که میتوانند هم زمان اتفاق بیافتند توجه داشته باشید، ما اطلاعاتی در مورد سفارش نداریم که چه اتفاقی افتاده است. ممکن است فردی وجود داشته باشد که یک روز سفارش را برای تکمیل، بسته بندی می کند و روز بعد فاکتور فرستاده می شود و بالعکس و یا شاید دقیقاً بسته بندی سفارش و ارسال فاکتور همزمان اتفاق بیافتند. وقتی همه این کارها انجام شد، یعنی سفارش تکمیل و فرستاده شد، فاکتور ارسال و مبلغ سفارش نیز دریافت شد، آنگاه می

توانیم به مرحله بعدی که بستن سفارش است برویم. آن را نیز با یک خط (میله) سیاه نمایش می دهیم که دقیقا شبیه گره Fork بوده و به آن گره Join گویند. تنها تفاوت آن با Forkدر این است که چندین جریان به آن وارد شده و تنها یک جریان خارج می شود. مرحله بعد بستن سفارش است. در نهایت قابل ذکر است که Fork و Join شبیه نقاط تصمیم گیری و الحاق هستند با این تفاوت که آنها راههای موازی که می توانند همزمان اتفاق بیافتند نمایش می دهند. یک جریان وارد گره Fork می شود و یک یا بیشتر خارج می شود. اما در Join بر عکس. یک یا چند جریان وارد شده و تنها یک جریان خارج می شود. وقتی که همه جریان ها در گره Join به هم ملحق شدند آنگاه می توانیم به مرحله بعدی برویم.

گره های شئ :

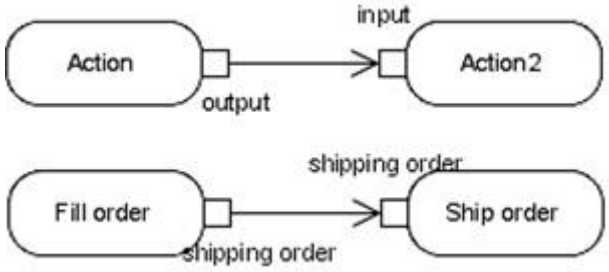
فعالیت ها نتیجه اشیاء هستند بدین معنی که اشیاء بارها وضعیت خود را در طول یک پروسه یا عملیات تغییر می دهند. وقتی به اشیاء ارجاع داده می شود و چگونه آنها بر اشیاء تاثیر می گذارند ما به وضعیت شیء اشاره می کنیم. State شبیه موضوع کسب و کار مهم که Object State نمایده می شود. برای مثال یک کتابخانه را تصور کنید، وضعیت کتاب ممکن است در قفسه باشد یا قرض داده شده ویا مورد نیاز باشد ویا گم شده یا در رزرو باشد. در دیاگرام فعالیت شما می توانید وضعیت و حالت یک شی را با نقاط در یک پروسه مختلف نمایش دهید. اشیاء تاثیر پذیرند با یک پروسه که شاید شامل فاکتور – چک – گزارش و حساب و... باشد. در اینجا ما یک دیاگرام فعالیت بسیار ساده داریم جریان رسیدن یک سفارش را نمایش می دهد. دریافت سفارش – ارسال فاکتور – دریافت وجه – بستن سفارش و پایان.هر مرحله از پروسه شی فاکتور به وسیله عملی که در وضعیت فاکتور رخ می دهد تاثیر می پذیرد. اجازه بدهید واضح تر توضیح بدهیم. شیء را توسط گره شی که یک مستطیل ساده با نام شی است نمایش می دهید. نام آن فاکتور است. و وضعیت آن درون کروشه نوشته می شود. وقتی سفارش دریافت می شود یک فاکتور ساخته می شود.



ارتباط بین عمل و شیء را توسط Object Flow نمایش می دهیم. همان طور که می بینید در این مثال شبیه Control Flow می باشد. سفارش دریافت شده و این بر وضعیت فاکتور تاثیر می گذارد . فاکتور ساخته می شود. مرحله بعدی ارسال فاکتور است،می توانیم جریانی داشته باشیم که وارد اشیاء می شوند . همچنین می توانیم جریانی داشته باشیم که از اشیاء خارج می شوند.در اینجا اول باید فاکتور ساخته شود قبل از اینکه فاکتور ارسال شود. پس از یک Object Flow برای نمایش اینکه ساخت فاکتور لازم است برای ارسال فاکتور استفاده می کنیم. و فاکتور ارسال می شود. در وضعیت شی فاکتور چه تاثیری می گذارد؟ در این مرحله وضعیت فاکتور از ساخته شده به در حالت انتظار و یا طلب می رود و ما برای نمایش آن از Object flow استفاده می کنیم. بعد از اینکه فاکتور ارسال شد و وجه دریافت شده وضعیت فاکتور به پرداخت شده تغییر پیدا می کند.

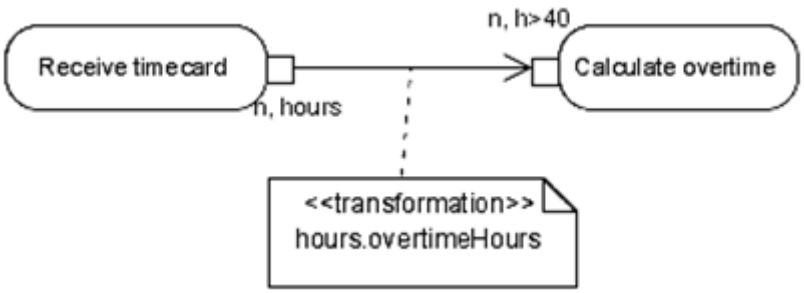
سنجاق ها :

همانند آن بخشی که درباره گره های اشیاء بود که UML را نشان می داد Activity Diagram می تواند اشیائی که در طی یک فرآیند بوسیله یک عمل ایجاد شده اند ، استفاده شده اند یا تغییر کرده اند را نشان دهد. این اشیاء می توانند اشیاء داده ای یا اشیاء فیزیکی باشند مانند یک فاکتور فیزیکی که فرآیند پرداخت وجه پول را نشان می دهد.



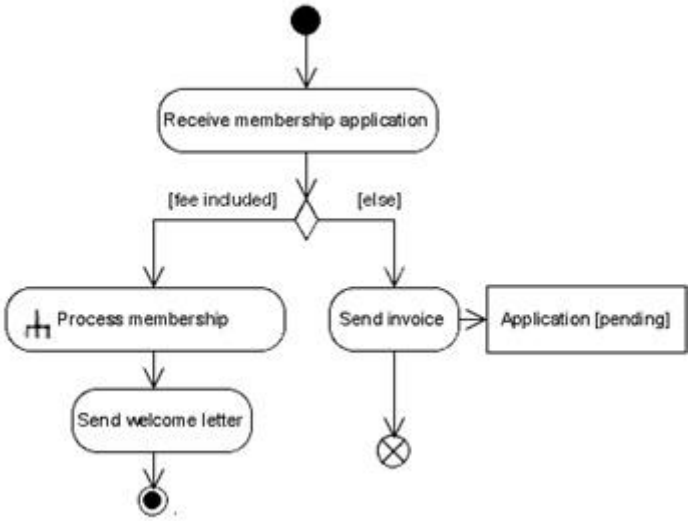
شما می توانید اشیاء را با استفاده از گره های اشیاء نشان دهید یا شما می توانید آنها را با استفاده از سنجاق ها (Pins) نشان دهید. سنجاق ها یک راه تند نویسی ارائه می دهند بوسیله نمایش اینکه یک شئ از یک نوعی ، یک ورودی برای یک عمل یا یک خروجی برای یک عمل است. اگر شما یک عملی را تجزیه می کنید Pin ها برابر جعبه های پارامتر در Sub Activity Diagram شما خواهد بود. برای اطلاعات بیشتر بخش مربوط به Sub Activity Diagram را مطالعه کنید. همانگونه که در اینجا مشاهده می کنید Pin ورودی تعیین می کند که شئ یک ورودی برای عملی است که Pin به آن متصل شده است. به همین طریق Pin خروجی نشان می دهد که این شئ یک خروجی از عملی است که که Pin به آن متصل شده است و شما می توانید بوسیله جریانی آنها را به یکدیگر متصل کره است بگوئید که که کدام Pin خروجی و کدام Pin ورودی است. اگر جه ت آن به سمت بیرون از Pin باشد آن Pin خروجی است و اگر جهت آن به سمت Pin باشد ، آن Pin ورودی است. بنابراین با استفاده از مثال "ترتیب انجام کار" خود ما می توانیم نشان دهیم که این چگونه کار می کند. ما یک عمل ترتیب پر کردن داریم که این عمل خروجی از یک ترتیب بارگیری است. که این ترتیب بارگیری به عنوان یک ورودی بوسیله عمل ترتیب حمل و نقل دریافت می شود پس ما Pin را برچسب گذاری می کنیم که این ورودی است و در اینجا دریافت شده است. پس شما می بینید که این یک مثال کاملا واضحی از نوعی است که خروجی بوسیله یک عمل ایجاد شده است و دیدید که آن چگونه به عنوان یک ورودی برای یک عمل دیگر دریافت شد. یک مثال ساده دیگر می تواند یک تابلوی امتیازات الکترونیکی باشد ، تابلوی امتیازات می تواند دو کار را انجام دهد ، محاسبه امتیاز (1) که آن می تواند بوسیله یک عملی از نمایش دادن امتیاز (2) پیروی کند و آن این کار را به شکل یک عدد انجام می دهد. پس ما برای رخ دادن این عملیات چه ورودی ها و خروجی هایی نیاز داریم؟ خوب محاسبه امتیاز به یک Pin ورودی نیاز دارد ، این خیال می کند که یک عددی را به یک شکلی دریافت کند ، بگذارید بگوییم در یک بازی فوتبال (آمریکایی) اگر یک TochDown اتفاق بیفتد (حالت فرود امدن توپ در منطقه خودی با بردن توپ به ان سوی خط دروازه حریف و کسب 6 امتیاز) این ورودی یک عدد 6 دریافت می کند ، آن (Action) از این عدد برای انجام عمل محاسبه نتیجه استفاده می کند و یک خروجی دیگر نیز دارد که می تواند نتیجه جدید باشد. حالا در گام بعدی ، عمل بعدی در فرآیند آن عدد را به عنوان ورودی دریافت خواهد کرد ، بنابراین ما به آن یک Pin ورودی اختصای می دهیم که آن مقدار محاسبه را به عنوان ورودی دریافت می کند و تابو را به عنوان خروجی ثبت می کند و ما آن را بوسیله یک روند شئ (object Flow) متصل می کنیم. بنابراین به طور مشابه ، "محاسبه امتیاز" یک عدد را به عنوان ورودی دریافت می کند ، نتیجه را محاسبه می کند و این یک عدد جدید به عنوان خروجی که نمایانگر نتیجه جدید است دارد ، که این عدد بوسیله عمل نمایش نتیجه به عنوان ورودی دریافت می شود. وسپس آن ما را به سمت پایان این فرآیند خاص می برد. حالا توجه داشته باشید که ورودی ها و خروجی های متصل شده گرایش دارند تا با یکدیگر مطابقت داشتته باشند ، ما ترتیب بارگیری را به عنوان خروجی ، ترتیب بارگیری به عنوان ورودی داریم. ما یک عدد نتیجه جدید خروجی ، یک عدد نتیجه جدید به عنوان ورودی داریم. اگر آنها با هم مطابقت نداشته باشند ، به عنوان مثال اگر یک عملی فقط به قسمتی از خروجی یک عمل دیگر برای ورودی خود احتیاج داشته باشد شما این را بوسیله یک تبدیل (Transformation) انجام می دهید که اینجا یک مثالی از آن است. ما یک فرآیند محاسبه لیست حقوق در اینجا داریم ، ما یک عمل "دریافت کارت محاسبه گر زمان ورود و خروج" (Receive Timecard) داریم ، خروجی این عمل یک عددی از ساعت ها است. برای عمل "محاسبه اضافه کاری" (Overtime Calculate) این فقط به قسمتی از خروجی نیاز دارد این نیاز دارد تا بداند آیا ساعات بالاتر از 40 بوده است تا بتواند طبق زمان محاسبه کند. پس ما این را با یک تبدیل نشان می دهیم و یک تبدیل به عنوان یک یادداشت به روند بین عمل ها متصل شده است. ما آن را یک Tarnsformation نشانه

گذاری می کنیم و شما می توانید تبدیل را ببینید ، این خروجی به این ورودی تبدیل شده است. ساعات به ساعات اضافه کاری تبدیل شده است. وجود Pin ها در Activity Diagram ها اختیاری است ، وقتی از آنها استفاده کنید که می خواهید نشان دهید داده بوسیله عملیاتی که Activity راتشکیل می دهد ایجاد شده است یا مورد درخواست قرار گرفته است ؛ یا زمانی که می خواهید فرایندهای تجاری را مدلسازی کنید تا منابع ایجاد شده و درخواست شده بوسیله گامهای مختلف در فرایند را نشان دهید.

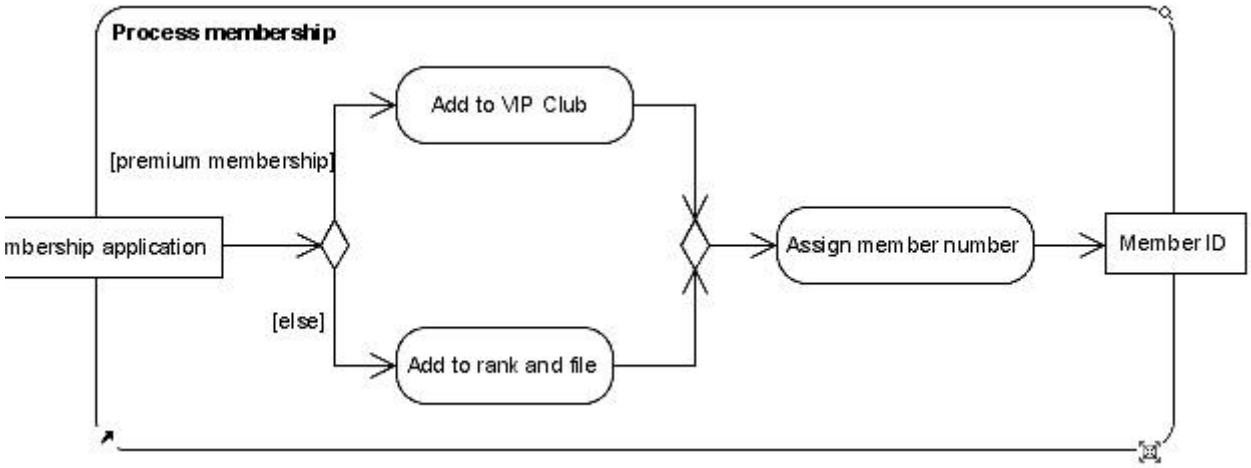


:Sub Activity Diagrams

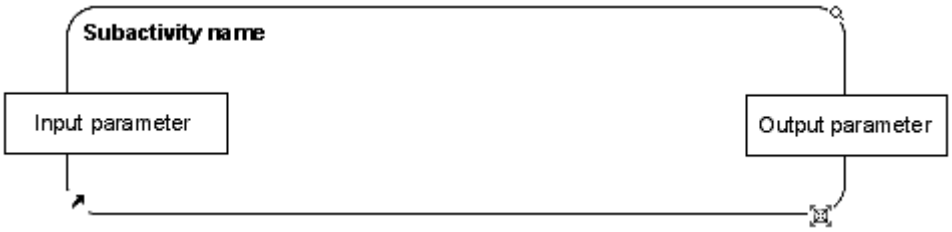
Activity Diagram Sub ها به شما اجازه میدهد تا روند اجرای عملیات را در یک Diagram Activity تجزیه کنید. به عبارت دیگر Sub Activity Digram به شما این امکان را می دهد که یک عمل (Action) را با جزئیان بیشتر نشان دهید ، یک عمل را به گامهای کوچکتر که آن را تشکیل می دهند بشکنید که کمک می کند یک Activity Diagram پیچیده و سخت برای فهمیدن نشود.



در یک Activity Diagram یک عملی که به یک Acitvity Diagram Sub شکسته شده است را بوسیله یک چنگال نشان داده می شود. بیائید به یک Activity Diagram بسیار ساده نگاهی بیاندازیم ، این برای یک نرم افزار دریافت و محاسبه عضویت است.

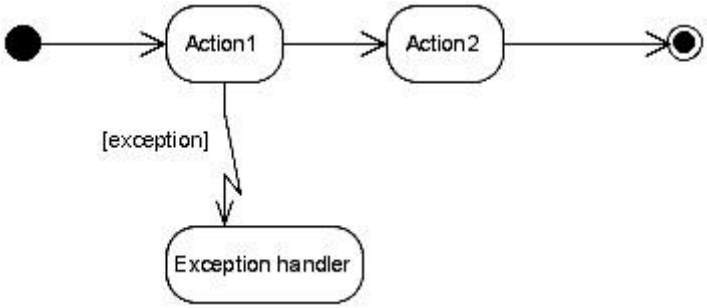


اینجا نقطه تصمصم ما است وقتی که وارد نرم افزار می شوید اگر ورودیه پرداخت شده باشد ما از سمت چپ ادامه می دهیم ، اگر پرداخت نشده باشد ما از سمت راست ادامه می دهیم ، یک فاکتور صورتحساب ارسال می کنیم و در آن نقطه شئ نرم افزار به حالت معلق می رود. این علامت کوچک در پایین در این مسیر یک گره پایانی است ولی این یک گره پایانی برای یک مسیر بخصوص است و با علامت پایانی (منظور علامت پایانی گوشه سمت چپ شکل) متضاد است که پایان یک Activity Diagram نشان می دهد. در اینجا ما یک علامت چنگال در عمل فرآیند عضویت داریم و همانگونه که گفته شد تعیین می کند که در آنجا یک Sub Activity Diagram وجود دارد که این عمل را به گامهای کوچکتری که آن را تشکیل می دهد شکسته است. حالا برخی از ابزار های UML این علامت چنگال را ندارند و اگر موردی با این ابزار پیش آمد شما می توانید کاملا راحت علامت خود را در Paint بکشید و آن را در دیاگرام خود Paste & Copy کنید. پس بیائید یک نگاهی بیاندازیم تا ببینیم چه اتفاقی در یک Sub Activity Diagram می افتد. ما یک نام Sub Activity داریم که باید با نام عملی که به عملیات خود تجزیه شده و به قسمتهای کوچک در این دیاگرام تقسیم شده یکسان باشد. پس برای آن عمل و برای این Sub Activity ما یک پارامتر ورودی داریم ، سپس در اینجا ما یک لیستی از گامها داریم ، عمل ها (Actions) ، مسیرها (Paths) ، تصمیمات و غیره که می توانند در این Sub Activity قرار گیرند و سپس وقتی شما به پایان فرآیند می رسید ، وقتی که عمل Activity Sub به پایان رسید ما پارامتر خروجی را داریم. پس در Activity Diagram خود در جایی که علامت چنگال وجود دارد Sub Activity وجود دارد که ما فرآیند عضویت را می شکنیم و به اینکه آن چه جزئیاتی دارد نگاه می کنیم. پس این Sub Activity ما است و این نام یکسانی دارد (Process membership) ، ورودی ما از نرم افزار عضویت است که پارامتر ورودی ما است. وقتی نرم افزار عضویت شروع به کار می کند ما به یک نقطه تصمیم گیری می رسیم ، آیا تقاضا کننده مبلغ را برای عضویت رسمی پرداخته است یا نه ، آیا ما تقاضا کننده را به انجمن VIP اضافه کرده ایم ، اگر نه ما از این مسیر پیگیری می کنیم و تقاضا کننده را به ردیف و فایا اضافه می کنیم. وقتی ما تقاضا کننده را به یک نوعی از عضویت اختصاص دادیم ، ما به یک نقطه ترکیب می رسیم و از هر راهی که رفته باشیم تا به اینجا برسیم به انجمن VIP اضافه کرده باشیم یا به ردیف و فایل اضافه کرده باشیم ما به عضو یک عدد اختصاص می دهیم پس پارامتر خروجی ما شناسه عضو (ID Member) خواهد بود.بنابراین این به شما گامهایی را که در عمل در دیاگرام اصلی فرآیند عضویت ما اتفاق می افتد را نشان می دهد. حالا خطر استفاده Sub Activity Diagram ها تمایل به عمیق شدن است ، شما می خواهید اگر ممکن باشد از تعداد زیادی از مجموعه لایه ها دوری کنید. هدف کلی از یک Activity Diagram این است که یک دید کلی واضح مانند نقشه جاده بدهد. واقعا دو چیز است که اهمیت دارد و آنها واضح بودن و متعادل بودن است. اگر Activity Diagram شما بوسیله Sub Activity ها زیاد پیچیده شود باعث می شود خوانایی آن سخت شود و همچنین تعقیب آن مشکل شود. اگر در سوی دیگر (برخلاف مثال ما) شما یک Activity Diagram را بیابید که شما جیی از جین ها (هر جین 12 عدد است) از Sub Activity Diagram ها را دارید پس شما نمی توانید Activity Diagram اصلی را بدون چرخش به عقب و پس و پیش کردن میان صفحات بیابید و باید به آن Activity Diagram Sub ها نگاه کنید سپس شما می خواهید یک Level بالاتر بیائید ، جزئیات بیشتری را در Activity Diagram اصلی خود قرار دهید و تعداد Sub Activity Diagram ها را به حداقل (بهینه باشد) برسانید.

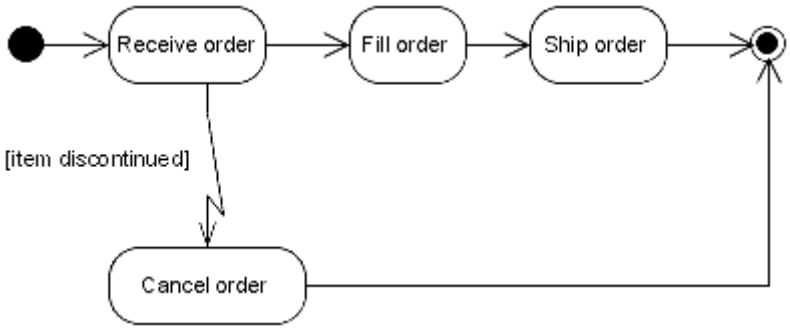


رسیدگی به خطاها:

گاهی اوقات یک دیاگرام فعالیت با موقعیتی غیرعادی روبرو می شود که به آن خطا یا استثناء گفته شده و باید به گونه ای با آن برخورد کرد که فعالیت مورد نظر ادامه پیدا کند یا متوقف شود. شما می توانید استثنائات و نحوه برخورد با آن را هم در دیاگرام های فعالیت خودتان قرار دهید.

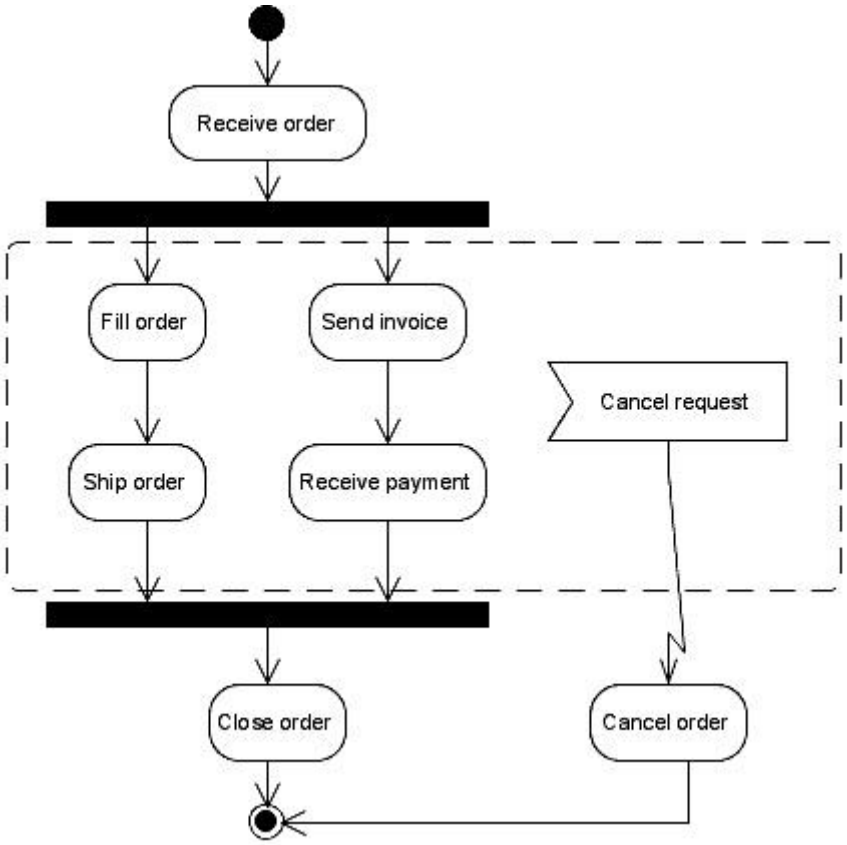


برای انجام این کار از یک کانکتور خاص که به شکل یک فلش زیگزاگ است استفاده می کنیم بدین نحو که آن را از فعالیتی که استثناء ممکن است در آن اتفاق بیفتد به رسیدگی کننده خطا وصل می کنیم. به عبارت دیگر آن را به فعالیتی وصل می کنیم که در پاسخ به خطا انجام می شود. این فعالیت یا ما را به فعالیت اصلی بر می گرداند یا ما را به پایان فعالیت می برد. در شکل چگونگی این اتصال را می بینید. روی فلش های زیگزاگی نام خطا نوشته می شود. این فلش زیگزاگی به رسیدگی کننده خطا متصل است. اگر رسیدگی کننده توانایی برگرداندن ما مثلاً به Action 2 را داشت، یک فلش از آن به Action 2 وصل می کنیم و یا اگر فقط می توانست روند کار را پایان دهد، از آن به گره پایانی یک فلش رسم می کنیم. برای مثال دیاگرامی که قبلاً مشاهده کردید را می آوریم. دیاگرامی که روند انجام کار در ساختمان اجرا را هنگامی که یک سفارش را دریافت می کرد، نشان می داد که عبارت بودند از دریافت سفارش، آماده کردن سفارش، فرستادن سفارش و تمام. اما هنگامی که سفارش (محصول) مورد نظر موجود نبود چه اتفاقی می افتد؟ ما قصد داریم چگونگی برخورد با این استثناء (خطا) را بیان کنیم. پس اجازه بدهید مسئله را این گونه مطرح کنیم که ما سفارشی را دریافت کردیم و خطا زمانی اتفاق می افتد که محصول مورد نظر موجود نباشد. در نتیجه نام این خطا را "موجود نبودن محصول" می گذاریم. شاید در این مورد تنها راه حل، لغو سفارش است. زیرا تا هنگامی که نتوانیم سفارش را آماده کنیم عملاً راهی به دیگر مراحل نخواهیم داشت. به عبارت دیگر هنگامی که فراهم کردن محصول و فرستادن آن امکان پذیر نباشد رسیدگی به این خطا (در این مثال) از طریق پایان دادن فعالیت صورت خواهد گرفت. لذا در این مثال یکی از خطاهایی که امکان دارد اتفاق بیفتد این است که ما سفارش را دریافت می کنیم که سفارش مربوط به محصولی است که موجود نیست. پس در این موقع کاری که انجام می دهیم لغو سفارش و بالطبع پایان دادن فعالیت است.



ناحیه وقفه :

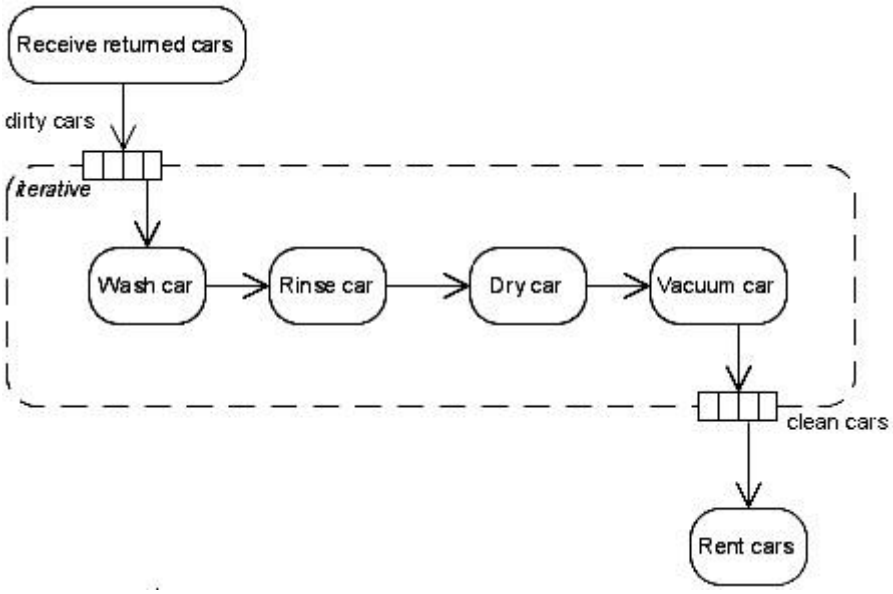
هنگامی که دیاگرام یک روند یا فعالیت را می کشید، ممکن است با فعالیت یا مجموعه ای از فعالیت ها روبرو شوید که می توانند توسط تعدادی رویداد دچار وقفه شوند. برای نمایش چنین حالتی، از یک ناحیه وقفه استفاده می کنیم. ناحیه وقفه با استفاده از یک مستطیل با کادر خط چین نمایش داده می شود. ما قصد داریم تا دقایقی دیگر یک ناحیه وقفه به دیاگرام خود اضافه کنیم. باز به سراغ همان مثال معروف یعنی فرایند سفارش می رویم. همان طور که در دیاگرام مشاهده می کنید هنگامی که یک سفارش دریافت می شود، دو بخش به طور موازی در حال انجام کار هستند. بخش انجام سفارش و بخش حسابداری. هنگامی که تمامی این فعالیت ها انجام شد، سفارش بسته شده و فرایند پایان می یابد. اما هنگامی که درخواستی مبنی بر لغو سفارش دریافت کردید، چه اتفاقی می افتد؟ درخواست لغو سفارش باید کل فرایند را متوقف کند. همانطور که قبلاً اشاره شد، ناحیه وقفه با استفاده یک مستطیل با کادر خط چین نمایش داده می شود. لذا فعالیت هایی که ممکن است دچار وقفه شوند را در این مستطیل قرار می دهیم. در عین حال رویدادی که موجب وقفه می شود را نیز در این مستطیل قرار می دهیم. وقفه را به عنوان یک سیگنال ورودی (سیگنال دریافت کننده) در نظر می گیریم که می تواند درخواست لغو سفارش را دریافت کند. لذا هنگامی که این ورودی اتفاق می افتد یا به عبارت دیگر این سیگنال ورودی دریافت می شود، تمامی فعالیت ها متوقف شده و به جای آنها یک فعالیت دیگر که می تواند لغو سفارش باشد انجام می شود. لذا سیگنال ورودی را با یک فلش زیگزاگ همانند کاری که در هنگام وقوع خطا (در ویدئوی قبلی) انجام می دادیم، به فعالیت لغو سفارش وصل می کنیم.



در اینجا به این فلش زیگزاگ، لبه وقفه می گوییم. پس هنگامی که وقفه اتفاق افتاد، تمام فعالیت ها را کنار گذاشته و فرایند را پایان می دهیم. بنابراین چیزی که در این ویدئو نمایش داده شد این بود که هنگامی که یک سیگنال وقفه دریافت شد، کل فعالیت های درون ناحیه وقفه، متوقف می شود. به جای آن یک فعالیت دیگر انجام می شود که در این مثال، این فعالیت فرایند ما را پایان می بخشد.

ناحیه توسعه :

گاهی اوقات خروجی یک فعالیت می تواند چندین درخواست یک فعالیت دیگر را آغاز کند. در این مواقع استفاده از ناحیه توسعه مفید به نظر می رسد. یک ناحیه توسعه دسته ای از فعالیت ها را نشان می دهد که برای هر قطعه در یک مجموعه، یکبار اتفاق می افتد. در واقع یک ناحیه توسعه شامل تعدادی فرایند است که هنگام دریافت داده چندین بار اجرا می شود اما هر کدام از فعالیت های درون این مجوعه یکبار اجرا می شوند. تصور اینکه ناحیه توسعه همانند یک حلقه For عمل می کند، می تواند کارساز باشد. یک آژانس کرایه ماشین را فرض کنید که وقتی ماشین ها برگشت داده می شوند آنها تعدادی از این ماشین ها را جمع کرده و یکی یکی به کارواش می فرستند.

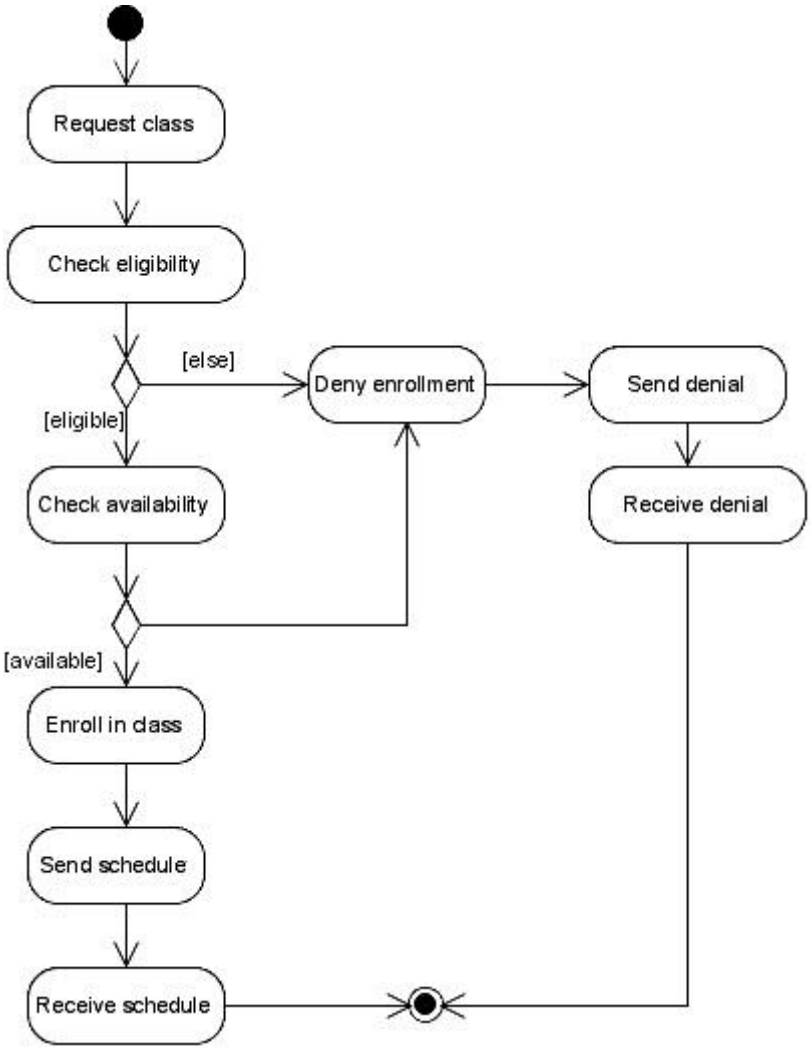


توجه داشته باشید که ناحیه توسعه توسط مستطیلی خط چین (ناحیه فعالیت) نمایش داده می شود. از آنجا که کارواش مورد نظر یک خط کاری بیشتر ندارد باید ماشین ها را یکی یکی به آن فرستاد و در نتیجه ناحیه توسعه ما به صورت تکرار شونده خواهد بود و ما به یک گره توسعه برای نمایش نقطه ورودی نیاز خواهیم داشت که در اینجا ورودی ما ماشین کثیف خواهد بود. بنابراین ما مجموعه ای از ماشین ها را خواهیم داشت که یکی یکی وارد این ناحیه توسعه خواهند شد و در این ناحیه اعمال شستن، خشک کردن و تمیز کردن روی ماشین ها اتفاق می افتد. همانطور که قبلا گفته شد از آنجا که این کارواش تنها یک خط کاری دارد هنگامی که یک ماشین کثیف وارد این خط کاری می شود تا زمانی که همه فعالیت های درون این ناحیه روی ماشین اتفاق نیفتد ماشین دوم نمی تواند وارد شود. در نتیجه باید این ناحیه (اعمال) به تعداد ماشین ها تکرار شوند. اجازه دهید با تعداد Flow (جریان/فلش) روند کار را مشخص کنیم. وقتی یک ماشین وارد این پروسه می شود، هنگامی که اعمال شستن، خشک کردن و تمیز کردن انجام شد، باید یک نقطه خروجی برای خارج شدن از ناحیه وجود داشته باشد که ما در اینجا این گره خروجی را ماشین تمیز می نامیم. بنابراین آنچه از این دیاگرام پیداست این است که دسته ای از ماشین های برگردانده شده وجود دارد که یکی یکی وارد کارواش شده و این دسته از اعمال درون ناحیه توسعه برای تک تک ماشین ها تکرار می شود. خروجی ما یک ماشین تمیز خواهد بود و هنگامی که مجموعه ای از ماشین های تمیز داشتیم می توانیم دوباره آنها را برای اجاره بفرستیم. در ناحیه توسعه 3 نوع تعامل بین اجراها وجود دارد. در این مثال ما از تعامل تکرار شونده استفاده کردیم. هنگامی که ناحیه توسعه از نوع تکرار شونده است تمامی اعمال به ترتیب اتفاق می افتند. می توانید از کلمات کلیدی دیگر برای نمایش انواع دیگر تعاملات در کنار آن استفاده کنید. تعاملات می توانند به صورت موازی انجام شوند بدین معنا که تعاملات مستقلند و می تواند یک جا انجام شوند که در این حالت می

توانید از کلمه کلیدی Stream استفاده کنید. Stream نشان می دهد که تمامی المانهای ورودی در یک لحظه وارد ناحیه توسعه می شوند و ناحیه توسعه آنها را به صورت یکجا اجرا کند.

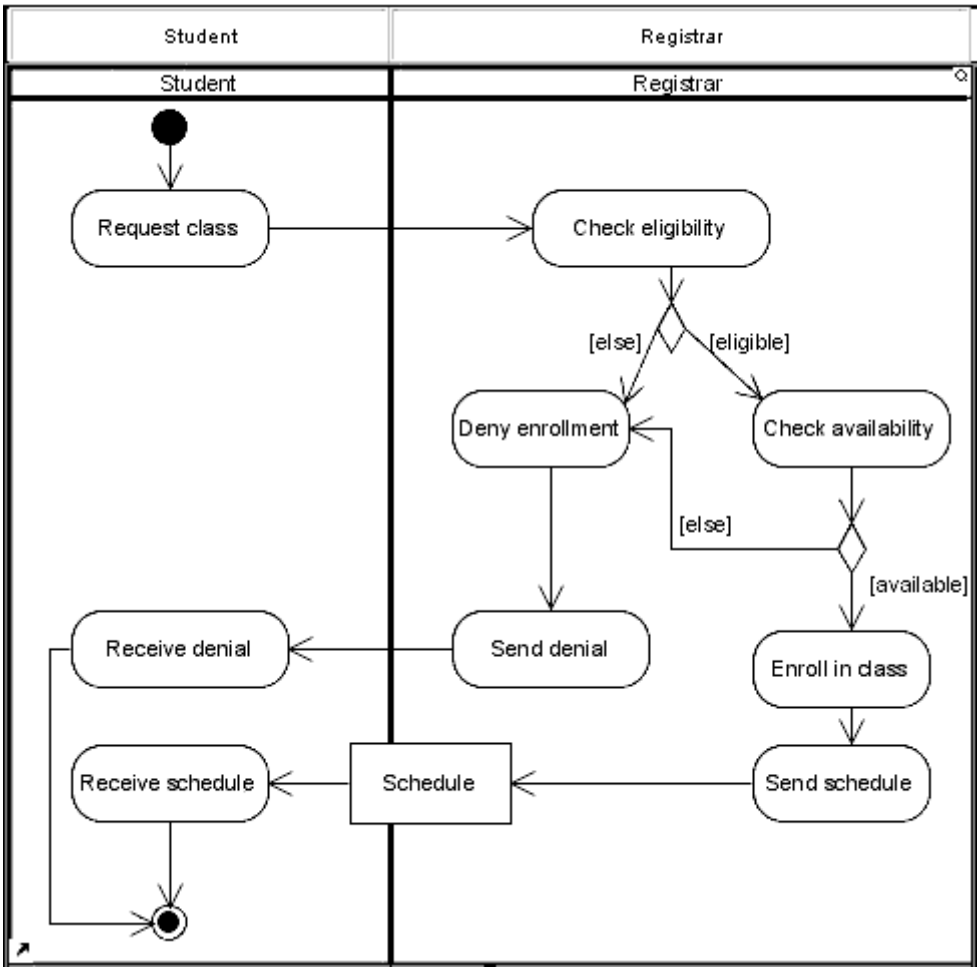
استفاده از swim Lanes:

دیاگرام های فعالیتی مثل این (که در شکل می بینید) یک فرایند را نشان می دهند اما مشخص نمی کند که هر فعالیت توسط چه کسی انجام می شود. همان طور که می بینید یک دیاگرام فعالیت داریم که فرایند ثبت نام برای یک کلاس را نمایش می دهد. ما در بالای دیاگرام فعالیت هایی شامل "درخواست کلاس" و "تایید صلاحیت" را داریم.



سپس یک نقطه تصمیم گیری وجود دارد که صلاحیت دانشجو را مورد بررسی قرار می دهد. اگر صلاحیت دانشجو تایید شد ما به فعالیت "چک کردن ظرفیت کلاس" منتقل می شویم اما اگر صلاحیت دانشجو تایید نشد یا ظرفیت کلاس تکمیل بود به فعالیت "رد ثبت نام (ثبت نام نکردن)" رفته و سپس فعالیت "ارسال پیغام عدم پذیرش" را اجرا کرده، دانشجو نیز فعالیت "دریافت پیغام عدم پذیرش" را اجرا کرده و فرایند پایان می یابد. هنگامی که کلاس ظرفیت داشت، فعالیت "ثبت نام دانشجو"، سپس فعالیت "تحویل برنامه درسی" و در نهایت فعالیت "دریافت برنامه درسی" اجرا شده و فرایند پایان می یابد. این دیاگرام به طور واضح روند ثبت نام در یک کلاس را مشخص می کند اما هیچ اطلاعاتی درباره اینکه چه کسی کدام فعالیت را انجام می دهد، در اختیار نمی گذارد. اگر بخواهید در دیاگرام فعالیت تان مشخص کنید که چه کسانی چه نقش هایی را بازی می کنند، باید قسمت هایی را به آن اضافه کنید که اصطلاحاً به آن Swim Lanes (مسیر شنا) گفته می شود. این نام گذاری به این دلیل است که ما کل دیاگرام را به یک استخر بزرگ و بازیگرها را به شناگر تشبیه کرده ایم. این استخر به مسیرهایی تقسیم شده

است که در آن هر شناگر (بازیگر) در مسیر خاص خود شنا می کند (فعالیت های خاص خود را انجام می دهد). در این دیاگرام نیز ما دو بخش داریم: دانشجو و ثبت نام کننده. هر Swim Lane شامل فعالیت هایی است که توسط آن بازیگر انجام می شود. لذا در اینجا دانشجو دیاگرام را با انجام فعالیت "درخواست کلاس" آغاز می کند، سپس همه فعالیت ها در حوزه ثبت نام کننده قرار می گیرد، همه فعالیت هایی که در دیاگرام قبلی نیز داشتیم.



شاید با دیدن این دو Swim Lane احساس کنید که نمودارمان اندکی تغییر کرده است اما همانطور که می بینید تمام اطلاعات مشابه دیاگرام قبلی اند، ما همان نقطه تصمیم را داریم که صلاحیت دانشجو را مورد بررسی قرار می دهد یا همان نقطه تصمیمی که ظرفیت کلاس را چک می کند و ... با این تفاوت که در اینجا به طور واضح مشخص است که چه کسی چه کاری انجام می دهد. توجه داشته باشید که Flow (جریان) ها می توانند از یک مسیر به مسیر دیگر نیز وارد شوند. برای مثال وقتی که ثبت نام کننده برنامه درسی را به دانشجو می دهد، برنامه درسی از یک ناحیه (ناحیه ثبت نام کننده) وارد ناحیه دیگر (ناحیه دانشجو) می شود. Swim Lane ها می توانند عمودی (همانطور که در این مثال دیدید) یا افقی باشند. این بستگی به شما دارد که چگونه دیاگرامتان را خواناتر کنید. Swim Lane ها بعد یا ابعاد دیگری از اطلاعات را به دیاگرام شما اضافه می کنند. اگر نقشی داشتید که مسئولیت آن بین چندین نقطه مختلف (چندین دپارتمان مختلف) تقسیم شده بود چه می کردید؟ ممکن است یک دفتر اصلی داشته باشید و شعبه های دیگر نیز نقش هایی مشابه دفتر اصلی اما با مسئولیت های مختلف داشته باشید و یا مثلاً در مثال دانشگاه ممکن است یک مجموعه اصلی (پردیس) برای تدریس دانشجویان اصلی و یک مجموعه ماهواره ای برای آموزش راه دور داشته باشید. در اینجا Swim Lane نقش های خود را خواهیم داشت که به صورت عمودی تنظیم شده اند و عبارتند از مدیریت و هیئت علمی. در Swim Lane های افقی نیز دو مجموعه اصلی و آموزش راه دور را خواهیم داشت. کل این سیستم باید یک سیاست کاری را تهیه و پیاده سازی کند لذا مدیریت مجموعه اصلی با تهیه پیش نویس این سیاست کاری، آغازگر این فرایند خواهد بود. سپس همتای آن یعنی مدیریت مجموعه آموزش راه دور این پیش نویس را مورد بررسی قرار داده و به مدیریت مجموعه اصلی بر می گرداند. در این هنگام مدیریت بخش اصلی با تصویب سیاست کاری اقدام به تولید سند آن کرده و سپس به هیئت علمی هر دو مجموعه تحویل می دهد. هیئت علمی مجموعه اصلی و هیئت علمی مجموعه آموزش راه دور

متناسب با وظایف خود سیاست های کاری را برای دانشجویان اصلی و دانشجویان آموزش از راه دور پیاده سازی می کنند. لذا این نوع شبکه چند بعدی دو مجموعه از بخش ها که در UML 2.0 جدید بود را به نمایش می گذارد و بعد تازه ای از اطلاعات را برای نمایش اینکه در یک دیاگرام فعالیت، چه کسی چه کاری انجام می دهد را فراهم می آورد.