پیوست ۱ آشنایی با UML[1]

زبان مدل‌سازی یکپارچه (UML) "زبانی استاندارد برای نوشتن نقشه‌های نرم‌افزار است. از UML می‌توان برای مصورسازی، مشخص‌کردن، ساخت و مستندسازی محصولات یک سیستم نرم‌افزاری استفاده کرد". [Boo05]. به بیان دیگر، درست همان‌طور که معماران ساختمانی، نقشه‌هایی تهیه می‌کنند که شرکت‌های ساخت‌وساز از آن‌ها استفاده می‌کنند، معماران نرم‌افزار نیز نمودارهای UML را برای کمک به سازندگان نرم‌افزار در ساخت نرم‌افزار تهیه می‌کنند. اگر واژگان UML (یعنی عناصر تصویری نمودار و معانی آن‌ها) را فرا بگیرید، بسیار آسان‌تر می‌توانید سیستمی را بفهمید و مشخص کنید و طراحی آن سیستم را برای دیگران توضیح دهید. گرادی بوچ، جیم رومباف و ایوار جیکابسون، UML را در میانه‌ی دهه ۱۹۹۰ با گرفتن بازخوردهای فراوان از جامعه‌ی نرم‌افزاری توسعه دادند. UML، چند نمادگذاری مدل‌سازی رقیب را که در صنعت نرم‌افزار آن زمان مورد استفاده قرار می‌گرفتند، در هم ادغام کرد. در سال ۱۹۹۷ UML 1.0 تسلیم گروه مدیریت اشیا (یک کنسرسیوم غیرانتفاعی دخیل در حفظ و نگهداری مشخصات مورد استفاده در صنعت نرم‌افزار) شد. UML 1.0 بازبینی و به UML 1.1 ارتقا داده شد و سال بعد پذیرفته شد. استاندارد فعلی، UML 2.3 است اکنون یکی از استانداردهای ISO است. از آن‌جا که این استاندارد بسیار جدید است، بسیاری از مراجع قدیمی نظیر [Gam95] از نمادگذاری UML استفاده نمی‌کنند.

UML 2.3 سیزده نمودار متفاوت برای استفاده در مدل‌سازی نرم‌افزار فراهم می‌آورد. در این پیوست، تنها به بحث درباره‌ی نمودارهای کلاس‌ها، استقرار، مورد کاربری[2]، توالی[3]، ارتباطات، فعالیت و حالت[4] خواهیم پرداخت. در این ویرایش از کتاب مهندسی نرم‌افزار از همین نمودارها استفاده شده است.

باید توجه داشته باشید که در نمودارهای UML، ویژگی‌های اختیاری فراوانی موجود است. زبان UML این گزینه‌ها را (که گاهی اسرارآمیز هستند) فراهم می‌آورد تا بتوانید همه‌ی جنبه‌های مهم

---

۱. این پیوست با کسب اجازه از دیل سرکین و از کتاب ایشان با عنوان *An Introduction to Object-Oriented Design Patterns in Java* آورده شده است.

2. Use case 3. Sequence 4. State

یک سیستم را بیان کنید. در عین حال، این انعطاف را دارید که بخش‌هایی از نمودار را که به جنبه‌ی مدل‌سازی‌شده مربوط نمی‌شوند، بپوشانید به طوری که از ازدحام نمودار با جزییات نامربوط جلوگیری شود. بنابراین، حذف یک ویژگی خاص به این معنی نیست که آن ویژگی وجود ندارد. بلکه ممکن است به این معنی باشد که آن ویژگی پوشانده شده است. در این پیوست، همه‌ی ویژگی‌های UML به طور کامل ارائه نشده است. در عوض، گزینه‌های استاندارد کانون توجه قرار گرفته‌اند به ویژه آن دسته از گزینه‌هایی که در این کتاب استفاده شده‌اند.

نمودار کلاس‌ها

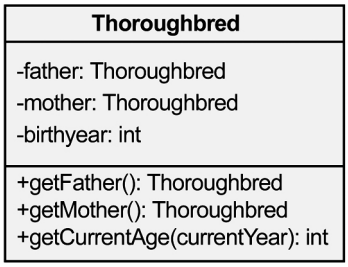
UML برای مدل‌سازی کلاس‌ها که شامل خصیصه‌ها، اَعمال و روابط آن‌ها با سایر کلاس‌ها می‌شود، یک نمودار کلاس‌ها ایجاد می‌کند[1]. نمودار کلاس‌ها، دیدی ساختاری یا ایستا از سیستم به دست می‌دهد ولی ماهیت پویایی برقراری ارتباط میان اشیای کلاس‌های موجود در نمودار را نشان نمی‌دهد.

عناصر اصلی نمودار کلاس‌ها عبارتند از: کادرهای چهارگوش که آیکن‌های مورد استفاده در نمایش کلاس‌ها و واسط‌ها هستند. هر کادر به بخش‌های افقی تقسیم می‌شود. بخش بالایی حاوی نام کلاس است. در بخش میانی، خصیصه‌های کلاس قرار می‌گیرد. هر خصیصه به چیزی اشاره دارد که شیءای از آن کلاس می‌داند یا می‌تواند همیشه آن را فراهم بیاورد. خصیصه‌ها معمولاً به عنوان فیلدهایی از کلاس پیاده‌سازی می‌شوند، ولی این ضرورتی ندارد. این خصیصه‌ها می‌توانند مقادیری باشند که کلاس قادر به محاسبه‌ی آن‌ها از متغیرهای نمونه است یا مقادیری باشند که کلاس می‌تواند از اشیای دیگری که از آن‌ها تشکیل شده است، به دست آورد. برای مثال ممکن است یک شیء همواره زمان کنونی را بداند و هر گاه که آن را درخواست کردید، به شما بازگرداند. بنابراین، فهرست کردن زمان کنونی به عنوان خصیصه‌ای از آن کلاس اشیا، بی‌مناسبت نیست. ولی، به احتمال زیاد، شیء آن زمان را در یکی از متغیرهای نمونه‌ای خود ذخیره نکرده است، زیرا باید پیوسته آن فیلد را به‌هنگام کند. در عوض، شیء احتمالاً زمان کنونی را (مثلاً از طریق مشورت با اشیای سایر کلاس‌ها) و در لحظه‌ای که زمان درخواست می‌شود، محاسبه می‌کند. بخش سوم نمودار کلاس‌ها حاوی اَعمال یا رفتارهای کلاس‌هاست. منظور از عمل، آن چیزی است که اشیای کلاس قادر به انجام آن هستند. این عمل معمولاً به عنوان متدی از کلاس پیاده‌سازی می‌شود.

در شکل پ ۱-۱، مثال ساده‌ای از کلاس **Thoroughbred** را نشان می‌دهد که اسب‌های مسابقه‌ای را مدل‌سازی می‌کند. این کلاس سه خصیصه را نشان می‌دهد – mother، father و birthyear. نمودار همچنین سه عمل را نشان می‌دهد: ()getCurrentAge، ()getFather و ()getMother. ممکن است خصیصه‌ها و اَعمال دیگری نیز وجود داشته باشد که پوشیده شده‌اند و در این نمودار نشان داده نمی‌شوند.

---

۱. درصورت ناآشنایی با مفاهیم شیءگرا، آشنایی مختصری در پیوست ۲ ارائه شده است.

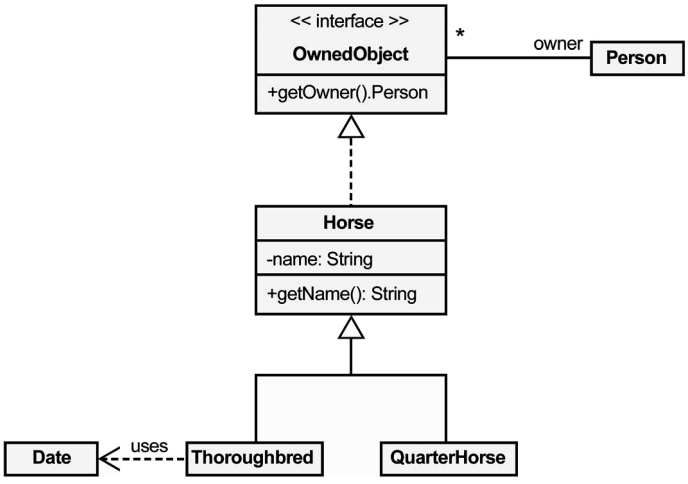


شکل پ۱-۱ نمودار کلاس‌ها براس کلاس Thoroughbred

هر خصیصه می‌تواند دارای یک نام، یک نوع و سطحی از قابلیت مشاهده باشد. نوع و قابلیت مشاهده، اختیاری‌اند. نوع بعد از نام ذکر می‌شود و توسط دو نقطه (:) از نام جدا می‌شود. قابلیت مشاهده با آوردن - ، # ، ~ یا + مشخص می‌شود که به‌ترتیب نشانگر خصوصی، محافظت‌شده، پکیج یا عمومی هستند. در شکل پ۱-۱، همه‌ی خصیصه‌ها دارای قابلیت مشاهده‌ی خصوصی‌اند که با علامت منها (-) نشان داده شده است. همچنین با خط کشیدن زیر یک خصیصه می‌توانید مشخص کنید که آن خصیصه، ایستاست یا خصیصه‌ی کلاس است. هر عمل را نیز می‌توان با سطحی از قابلیت مشاهده، پارامترهایی با نام و نوع و نوع بازگشت نمایش داد.

کلاس انتزاعی یا متد انتزاعی با استفاده از شکل ایتالیک نام در نمودار کلاس‌ها نشان داده می‌شود. برای مثال، کلاس Horse در شکل پ ۲-۱ را ببینید. واسط با افزودن عبارت << interface >> (که یک نمونه‌ی اولیه نامیده می‌شود) به بالای نام نشان داده می‌شود. واسط OwnedObject در شکل پ ۱-۲ را ببینید. واسط‌ها را به صورت گرافیکی با یک دایره‌ی توخالی نیز می‌توان نشان داد.

ذکر این نکته نیز خالی از ارزش نیست که آیکن نمایشگر کلاس می‌تواند بخش‌های اختیاری دیگری نیز داشته باشد. برای مثال، بخش چهار در پایین کادر چهارگوش کلاس را می‌توان در فهرست‌کردن



شکل پ ۱-۲ نمودار کلاس‌ها برای اسب‌ها

مسئولیت‌های کلاس به کار برد. این بخش، به ویژه هنگامی مفید واقع می‌شود که گذار از کارت‌های CRC (فصل ۱۰) به نمودارهای کلاس رخ می‌دهد از این روی که مسئولیت‌های فهرست‌شده در کارت‌های CRC را می‌توان قبل از ایجاد خصیصه‌ها و اَعمالی که این مسئولیت‌ها را اجرا می‌کنند به این بخش چهارگوش کلاس در نمودار UML افزود. این بخش چهارم در هیچ یک از نمودارهای این پیوست اضافه نشده است.

کلاسی که زیرکلاس یک کلاس دیگر باشد، توسط پیکانی نشان داده شده می‌شود که از یک خط پیوسته با نوک مثلث‌شکل تشکیل می‌شود. جهت پیکان از زیرکلاس به کلاس پایه است. در UML، چنین رابطه‌ای را **تعمیم** می‌نامند. برای مثال، در شکل پ ۱-۲، کلاس‌های **Thoroughbred** و **QuarterHorse**، به عنوان زیرکلاس‌های کلاس انتزاعی **Horse** نشان داده می‌شوند. پیکانی با خط مقطع و نوک زاویه‌ای، نشانگر پیاده‌سازی واسط است. در UML، چنین رابطه‌ای را **تحقق**[1] می‌نامند. برای مثال، در شکل پ ۱-۲، کلاس **Horse** واسط **OwnedObject** را پیاده‌سازی می‌کند یا تحقق می‌بخشد.

رابطه‌ی انجمنی میان دو کلاس به معنای وجود یک واسط ساختاری میان این دو است. رابطه‌های انجمنی با خطوط پیوسته نشان داده می‌شوند. رابطه‌های انجمنی بخش‌های اختیاری بسیار دارد. می‌توان نماد رابطه‌ی انجمنی و همچنین دو انتهای آن را برچسب‌گذاری کرد و نقش هر کدام از کلاس‌ها را در رابطه‌ی انجمنی مشخص کرد. برای مثال، در شکل پ ۱-۲، بین **OwnedObject** و **Person** یک رابطه‌ی انجمنی وجود دارد که در آن، **Person** نقش مالک را دارد. پیکان‌های روی یکی از دو طرف یا هر دو طرف خط انجمن نشانگر قابلیت گشت‌وگذارند. همچنین هر انتهای خط انجمن می‌تواند یک مقدار چندگانگی را نشان دهد. قابلیت گشت‌وگذار و چندگانگی، را بعداً در همین بخش با تفصیل بیشتری توضیح خواهیم داد. یک چنین انجمنی نشانگر اتصال شیءای از کلاس با سایر اشیای همان کلاس است.

رابطه‌ی انجمنی با یک پیکان در یکی از دو انتها، نشانگر گشت‌وگذار یک‌سویه است. پیکان بدان معناست که از یک کلاس می‌توان به سهولت به دومین کلاسی که انجمن به آن اشاره دارد، دست پیدا کرد، ولی از کلاس دوم الزاماً نمی‌توانید به سهولت به کلاس اول دست پیدا کنید. یک راه دیگر برای فکر کردن به این رابطه‌ی انجمنی این است که کلاس اول از کلاس دوم آگاه است، ولی شیء کلاس دوم الزاماً به طور مستقیم از کلاس اول اطلاع ندارد. رابطه‌ی انجمنی بدون پیکان معمولاً نشانگر رابطه‌ی انجمنی دوسویه است که در شکل پ ۱-۲ نشان داده شده است، ولی این تنها می‌تواند بدان معنا باشد که قابلیت گشت‌وگذار اهمیتی ندارد و از این رو به آن پرداخته نشده است.

شایان ذکر است که خصیصه‌ی یک کلاس، شباهت بسیار به انجمن یک کلاس با نوع کلاسِ آن خصیصه دارد. یعنی، برای این‌که نشان دهیم کلاسی دارای خاصیت موسوم به "نام" برای نوع String

1. Realization

(رشته‌ای) است، می‌توانیم آن خاصیت[1] را به عنوان یک خصیصه نشان دهیم، همانند کلاس **Horse** در شکل پ ۱-۲. به طریق دیگر، می‌توانیم رابطه‌ی انجمنی یک سویه‌ای از کلاس **Horse** به کلاس **String** ایجاد کنیم که در آن، نقش کلاس **String**، "نام" بودن آن است. رویکرد خصیصه‌ای برای انواع داده‌های اولیه بهتر است، در حالی که رویکرد رابطه‌ی انجمنی غالباً در صورتی بهتر است که کلاس خاصیت، نقش عمده در طراحی داشته باشد که در آن مورد، داشتن یک کادر کلاس برای آن نوع می‌تواند ارزشمند باشد.

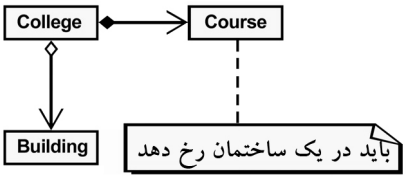
رابطه‌ی وابستگی[2]، نشانگر نوع دیگری از اتصال میان کلاس‌هاست و با یک خط مقطع (با پیکان‌های اختیاری در دو انتها و با نشانه‌های اختیاری) مشخص می‌شود. یک کلاس در صورتی به کلاس دیگر وابسته است که تغییرات اِعمال شده روی کلاس دوم، ممکن است به تغییرات کلاس اول نیاز داشته باشد. رابطه‌ی انجمنی کلاس با کلاس دیگر به طور خودکار نشانگر وابستگی است. اگر از قبل بین دو کلاس، رابطه‌ی انجمنی وجود داشته باشد، دیگر نیازی به رسم خط مقطع میان آن‌ها نیست. ولی، برای یک رابطه‌ی گذاری[3] (یعنی کلاسی که حاوی هیچ اتصال درازمدتی با کلاس دیگر نیست، ولی گاهی از آن کلاس استفاده می‌کند) باید خط مقطعی از کلاس نخست به کلاس دوم رسم کنید. برای مثال، در شکل پ ۱-۲، کلاس **Thoroughbred** هر گاه متد ()getCurrentAge فرا خوانده شود، از کلاس **Date** استفاده می‌کند و لذا وابستگی به صورت "استفاده می‌کند" برچسب‌گذاری می‌شود.

چندگانگی[4] در یک انتهای رابطه‌ی انجمنی، به معنای تعداد اشیایی از آن کلاس است که با کلاس دیگر رابطه‌ی انجمنی دارند. چندگانگی توسط یک عدد صحیح یا محدوده‌ای از اعداد صحیح مشخص می‌شود. چندگانگی که با "0..1" مشخص می‌شود، بدان معناست که در آن انتهای رابطه‌ی انجمنی، صفر یا یک شیء وجود دارد. برای مثال، هر شخصی در دنیا یک شماره‌ی ملی دارد یا ممکن است چنین شماره‌ای نداشته باشد و لذا در یک نمودار کلاس‌ها می‌توان از چندگانگی 0..1 برای رابطه‌ی انجمنی میان کلاس **Person** و کلاس **SocialSecurityNumber** استفاده کرد. چندگانگی‌ای که با "1..*" مشخص می‌شود، به معنای یک یا چند است و چندگانگی "0..*" یا صرفاً "*" به معنای صفر یا چند است. در شکل پ ۱-۲، در انتهای **OwnedObject** کلاس **Person** از * به عنوان چندگانگی استفاده شده است زیرا **Person** می‌تواند صفر یا چند شیء را مالک باشد.

اگر یک انتهای رابطه‌ی انجمنی دارای چندگانگی بزرگ‌تر از 1 باشد، اشیایی از کلاس که در آن انتها به آن‌ها اشاره شده است، احتمالاً در یک مجموعه یا فهرستی مرتب نگهداری می‌شود. حتی می‌توان خود آن کلاس مجموعه را در نمودار UML لحاظ کرد، ولی چنین کلاسی معمولاً آورده نمی‌شود و وجود آن به خاطر چندگانگی رابطه‌ی انجمنی، تلویحاً پذیرفته می‌شود.

---

1. Property 2. Dependency 3. Transient relationship 4. Multiplicity



شکل پ ۱-۳ رابطه میان کالج‌ها، دوره‌های درسی و ساختمان‌ها

تجمیع[1]، نوع خاصی از وابستگی است که با یک لوزی توخالی در یک انتهای آیکن نمایش داده می‌شود. این نماد نشانگر یک رابطه‌ی "کل/جزء" است، از این لحاظ که کلاسی که پیکان به آن اشاره دارد، به عنوان "جزیی" از کلاس مشخص‌شده با لوزی در نظر گرفته می‌شود. *ترکیب*[2]، نوعی تجمیع است که نشانگر مالکیت قوی اجزاست. در یک ترکیب، مرگ و زندگی اجزا به مالک آن‌ها وابسته است، چون نقشی مستقل از مالک خود در سیستم نرم‌افزار ندارند. برای مثال‌های تجمیع و ترکیب، شکل پ ۱-۳ را ببینید.

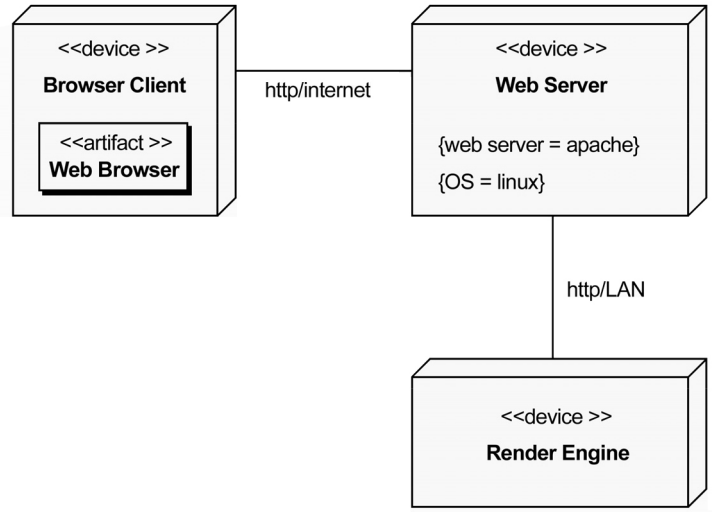
یک **College** دارای تجمیعی از اشیای **Building** است که نشانگر ساختمان‌های تشکیل‌دهنده‌ی محوطه‌ی دانشگاه هستند. کالج همچنین حاوی مجموعه‌ای از دوره‌های درسی است. اگر قرار باشد این کالج تعطیل شود، ساختمان‌های آن همچنان باقی خواهند ماند (با این فرض که کالج از نظر فیزیکی تخریب نشده باشد) و می‌توان از آن‌ها برای مقاصد دیگر استفاده کرد، ولی یک شیء **Course** در خارج از کالجی که دوره‌های درسی در آن ارائه می‌شدند، دیگر کاربردی نخواهد داشت. اگر کالج به عنوان یک نهاد تجاری دیگر به کار خود ادامه ندهد، شیء **Course** نیز دیگر بی‌فایده خواهد بود و وجود نخواهد داشت.

یک عنصر رایج دیگر در نمودار کلاس‌ها، یادداشت[3] است که توسط کادری با یک گوشه‌ی تاخورده نشان داده می‌شود و توسط خطی مقطع به سایر آیکن‌ها متصل می‌شود. این کادر می‌تواند دارای محتوای دلخواه (متون و تصاویر) باشد و مشابه خطوط توضیح در زبان‌های برنامه‌نویسی است. ممکن است این کادر حاوی توضیحاتی درباره‌ی نقش یک کلاس یا قیدوبندهایی باشد که همه‌ی اشیای آن کلاس باید آن‌ها را رعایت کنند. اگر این محتوا قید باشد، دو طرف محتوا با آکولاد {} محصور می‌شود. به قیدوبند متصل به کلاس **Course** در شکل پ ۱-۳ توجه کنید.

نمودارهای استقرار[4]

نمودار استقرار در UML، ساختار سیستم نرم‌افزار را کانون توجه قرار می‌دهد و برای نشان دادن توزیع فیزیکی یک سیستم نرم‌افزار در میان سکوهای سخت‌افزاری و محیط‌های اجرایی مفید واقع می‌شود. برای مثال، فرض کنید در حال ساخت یک پکیج رندر گرافیکی[5] تحت وب هستید. کاربران پکیج شما با استفاده از مرورگرهای خودشان به وب‌سایت شما می‌روند و اطلاعات رندر را وارد می‌کنند. وب‌سایت

1. Aggregation 2. Composition 3. Note 4. Deployment diagrams 5. Graphic render package



شکل پ ۱-۴ یک نمودار استقرار

شما باید تصویرگرافیکی را مطابق با آن‌چه کاربر مشخص کرده است، رندر کند و آن را برای کاربر برگرداند. از آن‌جا که رندرهای گرافیکی از نظر محاسباتی هزینه‌بر هستند، تصمیم می‌گیرید خودِ عملِ رندر را به وب‌سرور روی سکویی جداگانه انتقال دهید. بنابراین، در سیستم شما سه دستگاه سخت‌افزاری وجود خواهد داشت: کلاینت وب (کامپیوتر کاربر که یک مرورگر روی آن در حال اجراست)، کامپیوتر میزبان وب‌سرور و کامپیوتری که موتور رندر را میزبانی کند.

شکل پ ۱-۴ نمودار استقرار را برای یک چنین پکیجی نشان می‌دهد. در این نمودارها، مولفه‌های سخت‌افزاری درون کادرهایی رسم می‌شوند که با عبارت <<device>> مشخص می‌شود. مسیرهای ارتباطی میان مولفه‌های سخت افزاری با خطوطی رسم می‌شوند که دارای برچسب‌های اختیاری‌اند. در شکل پ ۱-۴، مسیرها با پروتکل ارتباطی و نوع شبکه‌ی به کار رفته در متصل کردن دستگاه‌ها نشان داده شده‌اند.

هر گره‌ی موجود در یک نمودار استقرار را همچنین می‌توان با جزییات مربوط به دستگاه، حاشیه‌نویسی کرد. برای مثال، در شکل پ ۱-۴، کلاینت مرورگر تصویر شده است تا نشان داده شود که حاوی شیءای متشکل از نرم‌افزار مرور وب است. این شیء معمولاً فایلی حاوی یک نرم‌افزار است که روی دستگاه اجرا می‌شود. می‌توانید مقادیر برچسب‌داری را نیز مشخص کنید که نمونه‌ای از آن در شکل پ ۱-۴ برای گره‌ی وب‌سرور نشان داده شده است. این مقادیر، فروشنده‌ی وب‌سرور و سیستم‌عامل مورد استفاده‌ی سرور را تعیین می‌کنند.

نمودارهای استقرار همچنین گره‌های محیط اجرا را نمایش می‌دهند که به صورت کادرهای حاوی نشانه‌ی <<execution environment>> رسم می‌شوند. این گره‌ها نشانگر سیستم‌هایی، نظیر سیستم‌های عامل، هستند که می‌توانند میزبان نرم‌افزارهای دیگر باشند.

نمودارهای مورد کاربری

موارد کاربری (فصل‌های ۸ و ۹) و نمودار مورد کاربری برای UML، شما را در تعیین قابلیت‌های عمکردی و ویژگی‌های نرم‌افزار از دیدگاه کاربر یاری می‌دهد. برای این‌که دیدی از چگونگی عملکرد نمودارهای مورد کاربری به دست آورید، یک نمودار برای برنامه‌ای ایجاد می‌کنیم که فروشگاه موسیقی دیجیتال آنلاین را مدیریت می‌کند، برخی از کارهایی که این نرم‌افزار ممکن است انجام دهد، عبارتند از:

- دانلود یک فایل موسیقی MP3 و ذخیره‌سازی آن در کتابخانه‌ی برنامه.
- گرفتن موسیقی از اینترنت و ذخیره‌سازی آن در کتابخانه‌ی برنامه.
- مدیریت کتابخانه‌ی برنامه (مثلاً حذف آهنگ‌ها یا سازمان دهی آن‌ها در فهرست‌های اجرا).
- تهیه‌ی CD از یک مجموعه آهنگ در کتابخانه.
- بارکردن فهرستی از آهنگ‌های موجود در کتابخانه روی یک iPod یا MP3 Player.
- تبدیل آهنگی از قالب MP3 به قالب AAC و بالعکس.

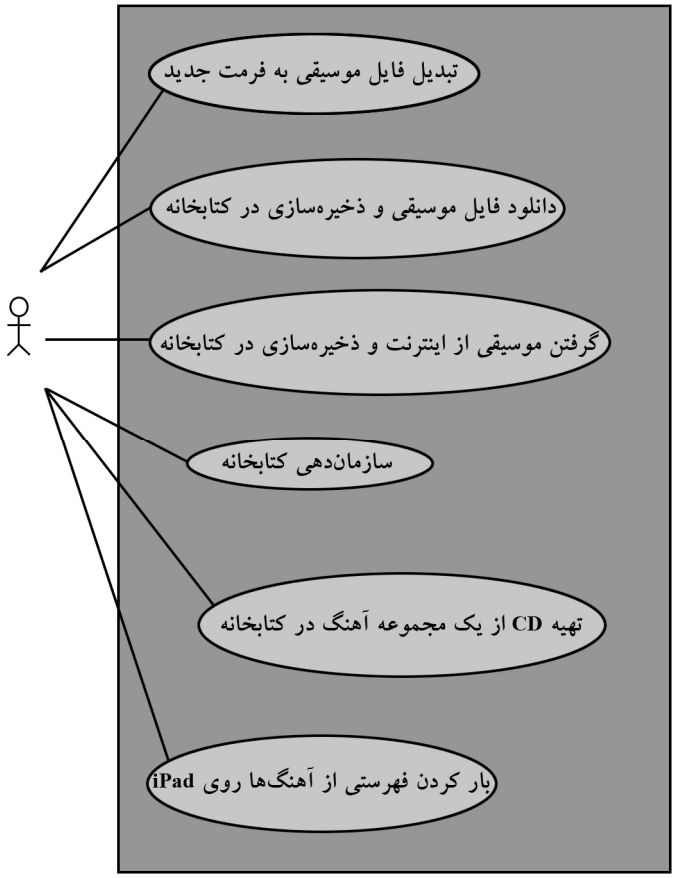
این فهرست، بلندبالا نیست، ولی برای درک نقش مورد کاربری و نمودارهای مورد کاربری کفایت می‌کند.

مورد کاربری، چگونگی تعامل کاربر با سیستم را با تعریف مراحل لازم برای دستیابی به هدفی خاص (مثلاً تهیه‌ی CD از آهنگ‌ها) توصیف می‌کند. تغییرات در مراحل، سناریوهای گوناگون را توصیف می‌کنند (مثلاً این‌که اگر همه‌ی آهنگ‌های موجود در فهرست روی CD جا نشود، چه باید کرد؟).

نمودار مورد کاربری در UML، دیدی اجمالی است بر کلیه موارد کاربری و چگونگی ارتباط میان آن‌ها. این نمودار، تصویر بزرگی از قابلیت عملکردی سیستم در اختیار قرار می‌دهد. یک نمودار مورد کاربری برای برنامه‌ی موسیقی دیجیتالی ما در شکل پ ۱–۵ نشان داده شده است.

در این نمودار مورد کاربری، تصویر آدمک نشانگر یک کنش‌گر (فصل ۸) است که با گروهی از کاربران (یا سایر عناصر تعاملی) در ارتباط هستند. سیستم‌های پیچیده معمولاً بیش از یک کنش‌گر دارند. برای مثال، اپلیکیشن ماشین‌های فروش ممکن است سه کنش‌گر داشته باشند که عبارتند از مشتریان، پرسنل تعمیرات و کسانی که ماشین‌ها را دوباره از کالا پر می‌کنند.

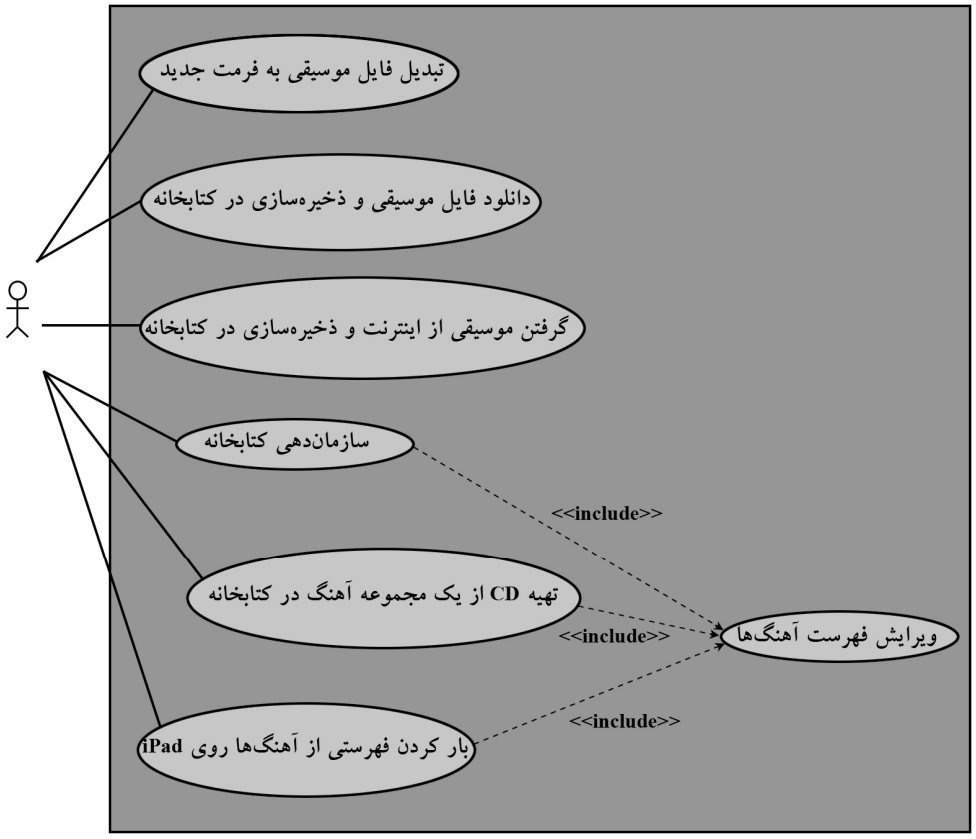
در نمودار مورد کاربری، موارد کاربری به صورت بیضی نشان داده می‌شوند. کنش‌گران توسط خطوط به موارد کاربری‌ای که اجرا می‌کنند، متصل می‌شوند. توجه دارید که هیچ کدام از جزییات مورد کاربری در نمودار ذکر نمی‌شود و در عوض باید آن‌ها را جداگانه ارائه داد. همچنین توجه دارید که موارد کاربری در یک چهارگوش قرار داده می‌شوند، ولی کنش‌گران خیر. این چهارگوش یادآور مرزهای سیستمی است که کنش‌گران خارج از آن قرار دارند.



شکل پ ۱-۵ نمودار مورد کاربری برای سیستم موسیقی

ممکن است برخی موارد کاربری در یک سیستم با یکدیگر در ارتباط باشند. برای مثال، در تهیه‌ی CD از فهرست آهنگ‌ها و بار کردن فهرست آهنگ‌ها روی iPod چند مرحله‌ی مشابه وجود دارد. در هر دو مورد، کاربر ابتدا فهرستی خالی ایجاد می‌کند و سپس آهنگ‌ها را از کتابخانه به فهرست اضافه می‌کند. برای پرهیز از دوباره‌کاری در موارد کاربری، معمولاً بهتر است یک مورد کاربری جدید ایجاد شود که فعالیت دوباره‌کاری شده را نمایش می‌دهد و سایر موارد کاربری می‌توانند این مورد کاربری جدید را به عنوان یکی از مراحل خود لحاظ کنند. این لحاظ کردن در نمودارهای مورد کاربری به صورتی که در شکل پ ۱-۶ ملاحظه می‌شود، نمایش داده شده است؛ یعنی توسط یک پیکان مقطع با برچسب <<inclule>> که مورد کاربری را به یک مورد کاربری لحاظ شده متصل می‌کند.

نمودار موارد کاربری، از آن‌جا که همه‌ی موارد کاربری را نمایش می‌دهد، در حصول اطمینان از این‌که همه‌ی قابلیت‌های عملکردی سیستم را پوشش داده‌اید، مفید واقع می‌شود. در برنامه‌ی موسیقی دیجیتال، حتماً می‌خواهید موارد کاربری بیشتری، نظیر مورد کاربری مربوط به اجرای یکی از آهنگ‌های موجود در کتابخانه، داشته باشید. ولی به خاطر بسپارید که ارزشمندترین سهم را در موارد کاربری، برای



شکل پ ۱-۶ نمودار مورد کاربری با موارد کاربری گنجانده شده در آن

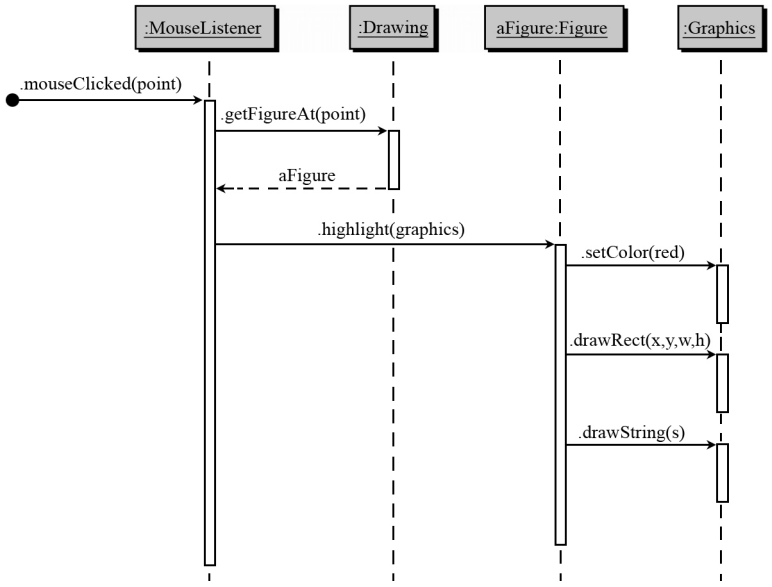
توسعه‌ی نرم‌افزار، توصیف‌های متنی هر مورد کاربری دارد نه نمودار کل موارد کاربری [Fow04b]. از طریق این توصیفات است که می‌توانید به درک درستی از اهداف سیستمِ در حال توسعه برسید.

نمودارهای توالی[1]

برخلاف نمودار کلاس‌ها و نمودار استقرار که ساختار ایستای یک مولفه‌ی نرم‌افزار را نشان می‌دهند، نمودار توالی برای نشان دادن ارتباطات میان اشیا در حین اجرای یک وظیفه به کار می‌رود. این نمودار، توالی زمانی ارسال پیام‌ها میان اشیا برای انجام یک وظیفه را نشان می‌دهد. برای نشان دادن تعامل‌های موجود در یک مورد کاربری یا در یک سناریو از سیستم نرم‌افزار می‌توان از نمودار توالی استفاده کرد.

در شکل پ ۱-۷ نمودار توالی را برای یک برنامه‌ی ترسیمی مشاهده می‌کنید. این نمودار نشانگر مراحل موجود در انتخاب و برجسته‌سازی یک شکل در ترسیمی است که با کلیک کردن روی آن رخ می‌دهد. هر کدام از کادرهای ردیف بالایی نمودار معمولاً متناظر با یک شیء هستند هر چند که

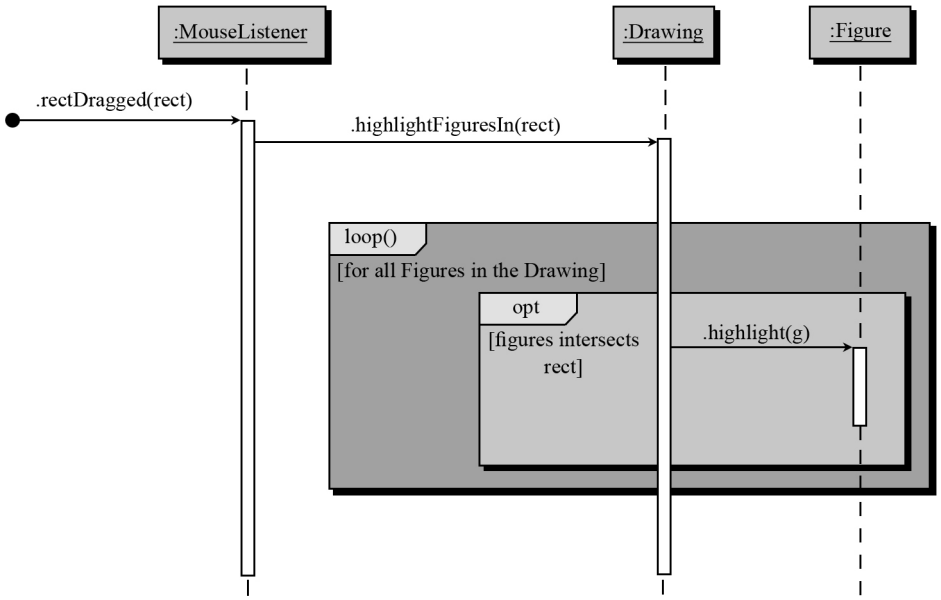
1. Sequence Diagrams



شکل پ ۱–۷ نمونه‌ای از نمودارهای توالی

می‌توان مدل‌سازی چیزهای دیگری مثل کلاس‌ها را نیز به این کادرها محول کرد. اگر کادر نشانگر یک شیء باشد (در همه‌ی مثال‌های ما چنین است)، در آن صورت، در داخل کادر می‌توانید به اختیار خود نوع شیء را ذکر کنید و قبل آن یک علامت دو نقطه (:) بگذارید. همچنین می‌توانید پیش از دو نقطه، نامی برای نوع شیء بگذارید که در کادر سوم شکل پ ۱–۷ مشاهده می‌شود. زیر هر کادر یک خط مقطع به نام خط حیات شیء وجود دارد. محور عمودی در نمودار توالی متناظر با زمان است به طوری که زمان از بالا به پایین افزایش می‌یابد.

نمودار توالی، فراخوانی متدها را با استفاده از پیکان‌های افقی از فراخواننده به فراخوانده‌شده نشان می‌دهد که با نام متد و لحاظ کردن اختیاری پارامترهای آن، نوع آن‌ها و نوع بازگشت برچسب‌گذاری می‌شود. برای مثال، در شکل پ ۱–۷، **MouseListener** متد getFigurAt() را شیء **Drawing** فرا می‌خواند. هنگامی که شیءای در حال اجرای یک متد است (یعنی هنگامی که روی پشته دارای قاب فعال‌سازی است)، می‌توانید به اختیار، یک نوار سفید به نام نوار فعال‌سازی در زیر خط طول عمر شیء نمایش دهید. در شکل پ ۱–۷، نوارهای فعال‌سازی برای همه‌ی فراخوان‌های متدها رسم شده‌اند. نمودار همچنین می‌تواند به دلخواه شما بازگشت از یک فراخوان متد را با پیکان مقطع و نشانه‌های اختیاری مشخص سازد. در شکل پ ۱–۷، برگشت فراخوان متد getFigurAt() با نشانه‌ی نام شیءای که برگشت داده می‌شود، نمایش داده شده است. یک روش سوم، همان‌سور که ما در شکل پ ۱–۷ انجام داده‌ایم، آن است که پیکان را در صورت فراخوانی یک متد تهی، به حال خود باقی بگذاریم، زیرا اگر بخواهیم اطلاعات کم اهمیت را روی نمودار بیاوریم، آن را بیهوده شلوغ می‌کنیم. یک دایره‌ی سیاه با پیکانی که از آن بیرون زده است، نشانگر پیامی است که منشأ آن ناشناس یا بی‌ربط است. اکنون باید

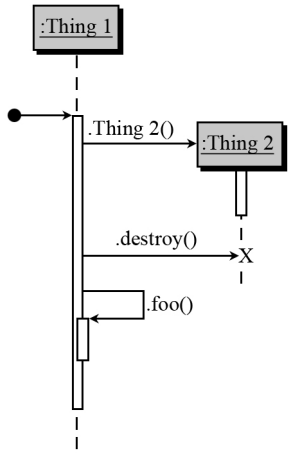


شکل پ ۱–۸ نمودار توالی با دو قاب تعامل

بتوانید وظیفه‌ای را که شکل پ ۱–۷ نشان می‌دهد، درک کنید. یک منبع ناشناس، متد mouseClicked() را از کلاس **MouseListener** فراخوانی می‌کند و نقطه‌ی کلیک را به عنوان شناسه تحویل می‌دهد. **MouseListener** به نوبه‌ی خود متد *getFigureAt()* را از **Drawing** فراخوانی می‌کند که آن هم **Figure** را باز می‌گرداند. سپس **MouseListener** متد *highlight*() را فراخوانی می‌کند و یک شیء **Graphics** را به عنوان شناسه تحویل می‌دهد. **Figure** نیز در پاسخ، سه متد از شیء **Graphics** را فراخوانی می‌کند تا شکل را به رنگ قرمز رسم کند.

نمودار شکل پ ۱–۷ بسیار ساده است و هیچ گونه حلقه یا ساختار شرطی ندارد. اگر به ساختارهای کنترلی نیاز باشد، احتمالاً بهترین راه، رسم یک نمودار توالی جداگانه برای هر مورد است. یعنی اگر جریان پیام‌ها بسته به یک شرط بتواند دو مسیر متفاوت را پیش بگیرد، در آن صورت دو نمودار توالی جداگانه، برای هر کدام از حالت‌های ممکن یک نمودار، رسم کنید.

اگر بر گنجاندن حلقه‌ها، ساختارهای شرطی و سایر ساختارهای کنترلی در نمودار خود اصرار دارید، می‌توانید از قاب‌های تعامل استفاده کنید که مستطیل‌هایی هستند که بخش‌هایی از نمودار را دربرگرفته و با نوع ساختار کنترلی مربوطه مشخص می‌شوند. در شکل پ ۱–۸ این را مشاهده می‌کنید؛ در این شکل فرآیند مربوط به برجسته‌سازی همه‌ی شکل‌های درون یک مستطیل نشان داده شده است. به **MouseListener** پیام rectDragged ارسال می‌شود. سپس **MouseListener** به بخش ترسیم می‌گوید که همه‌ی شکل‌های داخل مستطیل را با فراخوانی متد *highlightFigures()* برجسته کند و مستطیل را به عنوان شناسه تحویل بگیرد. این متد در میان همه‌ی اشیای **Figure** در شیء **Drawing** دور می‌زند و اگر **Figure** با مستطیل اشتراک داشت، از **Figure** خواسته می‌شود تا خودش را برجسته کند.



شکل پ ۱-۹ ایجاد، تخریب و حلقه‌ها در نمودارهای توالی

عبارت‌های داخل کروشه‌ی [] را نگهبان می‌نامند که شرط‌هایی بولی هستند و باید درست باشند تا کنش داخل قاب تعامل ادامه پیدا کند.

ویژگی‌های خاص فراوان دیگری وجود دارد که می‌توان در نمودار توالی گنجاند. برای مثال:

۱. میان پیام‌های همگام و ناهمگام می‌توانید تمایز قائل شوید. پیام‌های همگام با سر پیکان‌های توپر نشان داده می‌شوند در حالی که پیام‌های ناهمگام با سرپیکان‌های میله‌ای مشخص می‌شوند.

۲. با پیکانی که از یک شیء خارج می‌شود، به طرف پایین می‌پیچد و دوباره به همان شیء باز می‌شود، می‌توانید نشان دهید که آن شیء به خودش پیام ارسال می‌کند.

۳. می‌توانید ایجاد شیء را با رسم پیکانی با برچسب مناسب (مثلاً با برچسب <<create>>) نشان دهید که به چهارگوش شیء اشاره دارد. در این مورد، چهارگوش در داخل نمودار و پایین‌تر از چهارگوش‌های مربوط به اشیایی قرار می‌گیرد که هنگام شروع کنش از قبل وجود داشته‌اند.

۴. می توانید تخریب یک شیء را با قرار دادن X در انتهای خط حیات شیء نشان دهید. اشیای دیگر نیز قادر به تخریب کردن یک شیء هستند که در آن صورت، یک پیکان از شیء دیگر به X رسم می‌شود. همچنین X نشان می‌دهد که شیء دیگر قابل استفاده نیست و جمع‌کننده‌ی حافظه‌ی مازاد[1] باید آن را جمع‌آوری کند.

سه ویژگی آخر در شکل پ ۹-۱ نشان داده شده است.

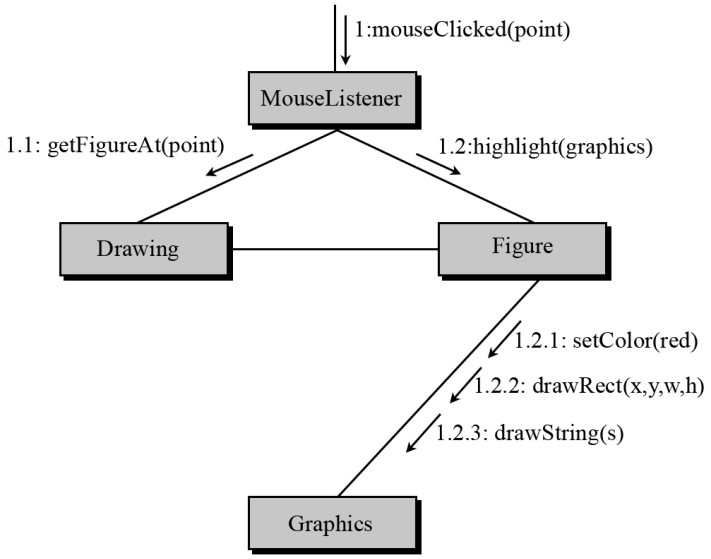
نمودارهای ارتباطات[2]

نمودارهای ارتباطات در UML (که در UML 1.X، "نمودار همکاری" خوانده می‌شدند) شاخص دیگری از ترتیب زمانی ارتباطات را در اختیار می‌گذارند، ولی به جای ترتیب زمانی بر روابط میان اشیا و کلاس‌ها تأکید دارند.

یک نمودار ارتباطات که در شکل پ ۱-۱۰ نشان داده شده است، همان کنش‌های نمودار ترتیب شکل پ ۱-۷ را به تصویر می‌کشد.

در یک نمودار ارتباطات، اشیای در حال تعامل توسط مستطیل نشان داده می‌شوند. رابطه‌های انجمنی میان اشیا توسط خطوط اتصال‌دهنده‌ی مستطیل‌ها نشان داده می‌شوند. معمولاً یک پیکان وارد شیء‌ای از نمودار می‌شود که دنباله‌ی تبادل پیام را آغاز می‌کند. آن پیکان با یک عدد و نام پیام برچسب‌گذاری می‌شود. اگر پیام وارد شده با عدد 1 برچسب‌گذاری شده باشد و اگر باعث شود شیء دریافت کننده‌ی

1. Garbage collection 2. Communication diagrams



شکل پ ۱-۱۰ یک نمودار ارتباطات

آن، پیام‌های روی سایر اشیا را فراخوانی کند، در آن صورت آن پیام‌ها توسط پیکان‌های وارد شده از فرستنده به گیرنده در راستای یک خط تجمع نشان داده می‌شوند و به ترتیب فراخوانی، اعداد 1.1، 1.2 و غیره به آن‌ها داده می‌شود. اگر آن پیام‌ها به نوبه‌ی خود پیام‌های دیگری را فراخوانی کنند، یک نقطه‌ی اعشاری دیگر و یک عدد دیگر به نشان آن‌ها اضافه می‌شود تا تبادل پیام‌ها بهتر مشخص شود.

در شکل پ ۱-۱۰، می‌بینید که پیام mouseClicked، متدهای ()getFigureAt و ()highlight را فراخوانی می‌کند. پیام ()highlight را فراخوانی می‌کند. پیام ()highlight سه پیام دیگر را فراخوانی می‌کند: ()setColor، ()drawRect و ()drawstring. شماره‌گذاری در هر برچسب، تودرتو بودن و نیز ماهیت ترتیبی هر پیام را نشان می‌دهد.

ویژگی‌های اختیاری فراوانی وجود دارند که می‌توان به برچسب پیکان‌ها افزود. برای مثال، می‌توانید پیش از عدد، یک حرف اضافه کنید. پیکان واردشونده را می‌توان به صورت **A1:**mouseClicked(point) برچسب‌گذاری کرد که یک نخ اجرایی A را مشخص می‌سازد. اگر پیام‌های دیگر در نخ‌های دیگر اجرا شوند، برچسب آن‌ها با حروف متفاوت شروع خواهد شد. برای مثال، اگر متد ()mouseClicked در نخ A اجرا شود، ولی نخ جدید B را ایجاد کند و ()highlight را در آن بند فراخوانی کند، در آن صورت، پیکان وارد شده از **MouseListener** به **Figure** به صورت **1.B2:**highlight(graphics) برچسب‌گذاری خواهد شد.

اگر مایلید روابط میان اشیا را علاوه بر پیام‌های ارسال شده میان آن‌ها نشان دهید، نمودار ارتباطات احتمالاً گزینه‌ای بهتر از نمودار توالی است. اگر توالی زمانی تبادل پیام‌ها برای شما اهمیت دارد، احتمالاً نمودار توالی بهتر است.

نمودارهای فعالیت[1]

نمودار فعالیت UML رفتار پویای یک سیستم یا بخشی از سیستم را از طریق جریان کنترل میان کنش‌های انجام شده توسط سیستم، تصویر می‌کند. این نمودار مشابه نمودارهای گردشی است با این تفاوت که نمودار فعالیت می‌تواند جریان‌های همروند را هم نمایش دهد.

مولفه‌ی اصلی یک نمودار فعالیت، گره‌ی کنش است که توسط مستطیل لبه‌گرد نشان داده می‌شود و متناظر با وظیفه‌ای است که توسط سیستم نرم‌افزاری اجرا می‌شود. پیکان‌هایی که یک گره‌ی کنش را به دیگری متصل می‌کنند، جریان کنترل را نشان می‌دهند. یعنی، پیکان میان دو گره‌ی کنشی بدان معناست که پس از کامل شدن کنش اول، کنش دوم آغاز می‌شود. یک نقطه‌ی سیاه توپر، گره‌ی اولیه‌ای را تشکیل می‌دهد که نقطه‌ی شروع فعالیت را نشان می‌دهد. یک نقطه‌ی سیاه که توسط دایره‌ای سیاه احاطه می‌شود، گره‌ی نهایی است که انتهای فعالیت را نشان می‌دهد.

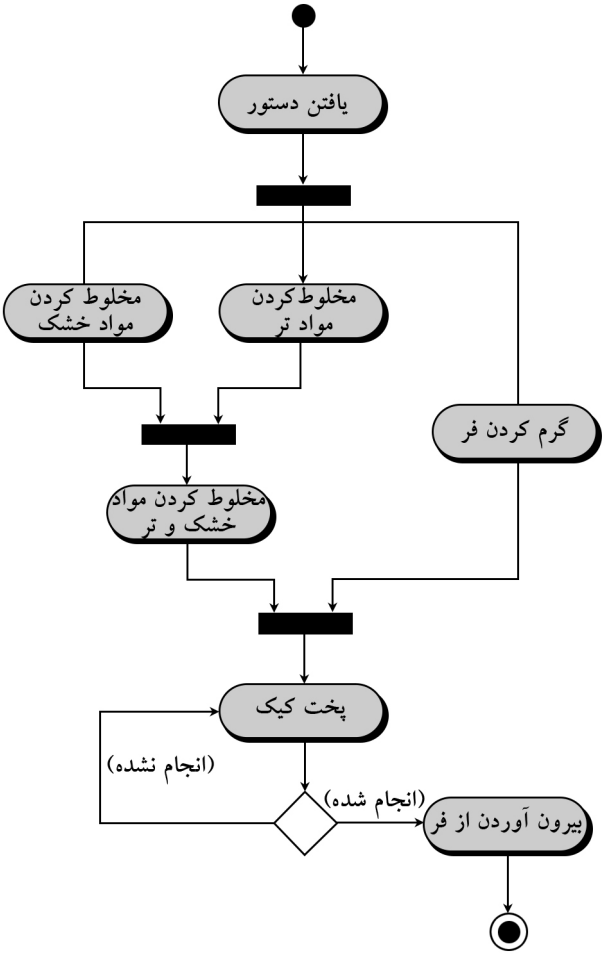
انشعاب[2]، نشانگر جدا شدن فعالیت‌ها به دو یا چند فعالیت همروند است و به صورت یک نوار سیاه افقی رسم می‌شود که یک پیکان وارد آن می‌شود و دو یا چند پیکان از آن خارج می‌شود. هر پیکانی که خارج می‌شود، نشانگر جریان کنترلی است که همروند با جریان‌های متناظر با سایر پیکان‌های خارج‌شونده، قابل اجراست. این فعالیت‌های همروند روی یک کامپیوتر و با استفاده از نخ‌های متفاوت یا حتی با استفاده از کامپیوترهای متفاوت قابل اجرا هستند.

شکل پ ۱-۱۱ یک نمودار فعالیت برای پختن کیک را نشان می‌دهد. نخستین مرحله، یافتن دستور تهیه‌ی کیک است. پس از این‌که دستور تهیه یافته شد، مواد خشک و موادِ تر را می‌توان اندازه‌گیری و مخلوط کرد و حتی فر را هم گرم کرد. مخلوط کردن مواد خشک را می‌توان به موازات مخلوط کردن موادِ تر انجام داد و فر را همزمان گرم کرد.

پیوند[3] راهی برای همگام‌سازی جریان‌های کنترل همروند است و با یک نوار سیاه افقی با دو یا چند پیکان وارد شونده به آن و تنها یک پیکان خارج‌شونده نشان داده می‌شود. اجرای جریان کنترلی که توسط پیکان خارج‌شونده نمایش داده می‌شود، نمی‌تواند شروع شود تا این‌که همه‌ی جریان‌های نشان داده شده توسط پیکان‌های وارد شونده کامل شده باشند. در شکل پ ۱-۱۱، پیش از کنش مخلوط کردن مواد خشک و تر با هم، یک پیوند داریم. این پیوند نشان می‌دهد که همه‌ی مواد خشک و همه‌ی موادِ تر باید قبل از ترکیب شدن، با هم مخلوط شده باشند. تلاقی دوم در این مشکل نشان می‌دهد که پیش از آن‌که بتوان پخت کیک را آغاز کرد، همه‌ی مواد باید با هم مخلوط شوند و فر باید به دمای مناسب رسیده باشد.

گره‌ی تصمیم‌گیری[4] متناظر با شاخه‌ای از جریان کنترل بر اساس یک شرط معین است. چنین گره‌ای به صورت یک لوزی نشان داده می‌شود که پیکانی وارد آن شده دو یا چند پیکان از آن خارج می‌شود. هر پیکانی که خارج می‌شود با یک نگهبان برچسب‌گذاری می‌شود (این نگهبان، شرطی است

1. Activity diagrams 2. Fork 3. Join 4. Decision



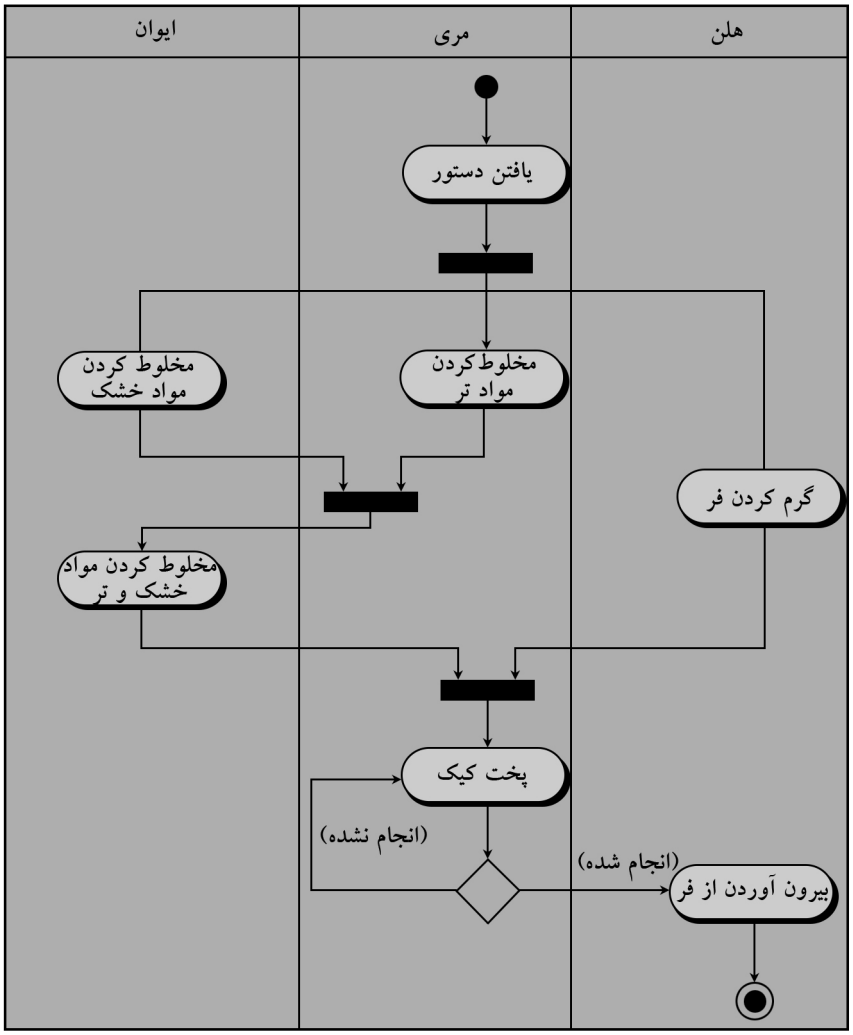
شکل پ ۱-۱۱ یک نمودار فعالیت UML که چگونگی پخت کیک را نشان می‌دهد

که داخل کروشه ذکر می‌شود). جریان کنترل، پیکانی را دنبال می‌کند که شرط نگهبان آن درست باشد. توصیه می‌شود که این شرط‌ها همه‌ی حالت‌های ممکن را پوشش بدهند، به طوری که دقیقاً یکی از آن‌ها با هر بار رسیدن به گره‌ی تصمیم‌گیری، درست باشد. شکل پ ۱-۱۱ یک گره‌ی تصمیم‌گیری را نشان می‌دهد که پس از پختن کیک آورده شده است. اگر کیک آماده بود، از فر برداشته می‌شود. در غیر این صورت، قدری بیشتر پخته می‌شود.

یکی از چیزهایی که نمودار فعالیت در شکل پ ۱۱-۱ نمی‌گوید، این است که هر کنش را چه کسی یا چه چیزی انجام می‌دهد. غالباً، تقسیم‌بندی دقیق کارها اهمیتی ندارد. ولی اگر می‌خواهید خاطرنشان کنید کنش‌ها چگونه در میان مشارکت‌کنندگان[1] تقسیم می‌شوند، می‌تواند نمودار فعالیت را

1. Participants

با نمودارهای بخش‌بندی[1] غنی کنید (شکل پ ۱-۱۲). نمودارهای بخش‌بندی، همان‌طور که از نام آن‌ها برمی‌آید، با تقسیم نمودار به چند نوار یا "بخش" تشکیل می‌شوند که هر کدام از آن‌ها به یک مشارکت‌کننده مربوط می‌شود. همه‌ی کنش‌های موجود در یک بخش یا نوار توسط مشارکت‌کننده‌ی مربوط انجام می‌شوند. در شکل پ ۱-۱۲، ایوان مسئول مخلوط کردن مواد خشک و سپس مخلوط کردن موادتر است، هلن مسئولیت گرم کردن فر و بیرون آوردن کیک را برعهده دارد و مری مسئول کارهای دیگر است.



شکل پ ۱-۱۲ نمودار فعالیت پخت کیک که خطوط بخش‌بندی نیز به آن افزوده شده است

1. Swimlane

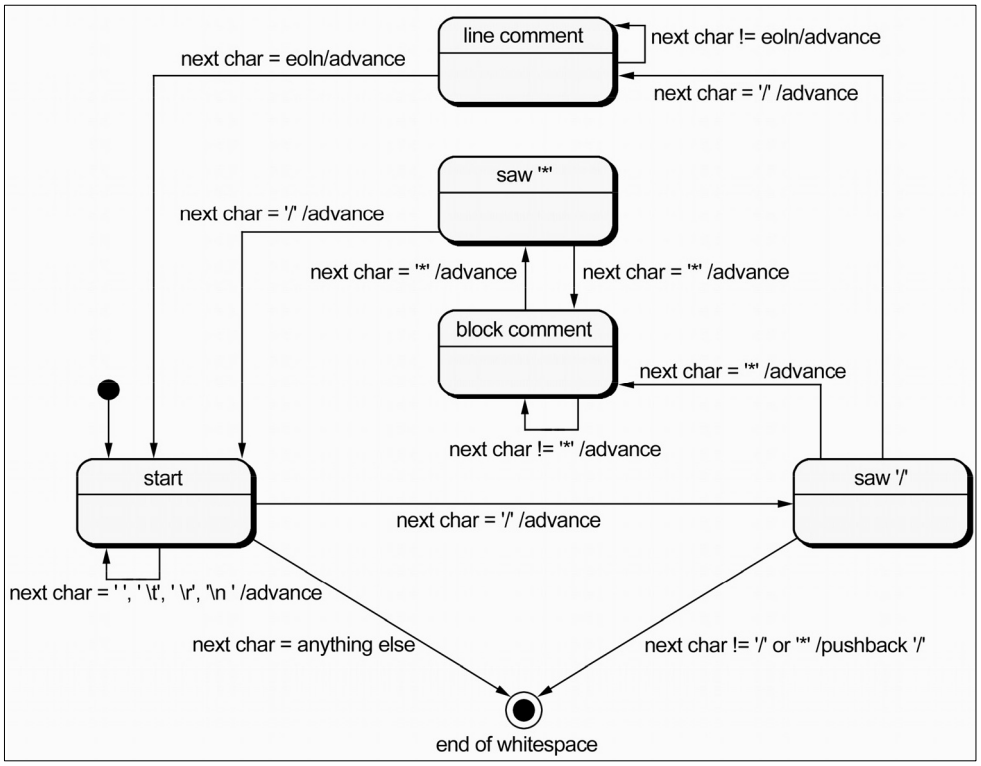
نمودارهای حالت[1]

رفتار یک شیء در نقطه‌ی خاصی از زمان غالباً به حالت شیء یعنی مقادیر متغیرهایش در آن زمان بستگی دارد. به عنوان مثالی ساده، شیءای با یک متغیر بولی را در نظر بگیرید. هنگامی که از شیء خواسته می‌شود تا عملی را انجام دهد، ممکن است در صورت درست بودن آن متغیر یک کار انجام دهد و در صورت نادرست بودن، کاری دیگر.

نمودار حالت در UML، حالت‌های شیء، کنش‌هایی که بسته به این حالت انجام می‌شوند و گذارهای میان حالت‌های شیء را مدل‌سازی می‌کند. برای مثال، نمودار حالت را برای بخشی از یک کامپایلر جاوا در نظر بگیرید. ورودی کامپایلر، یک فایل متنی است که می‌توان آن را یک رشته‌ی متنی طولانی در نظر گرفت. کامپایلر، کاراکترها را یک به یک می‌خواند و از آن‌ها ساختار برنامه را تعیین می‌کند. یک بخش کوچک از این فرآیند خواندن کاراکترها، شامل چشم‌پوشی از کاراکترهای "فضای سفید" (مانند کاراکترهای space، tab، newline و return) می‌شود.

فرض کنید که کامپایلر، وظیفه‌ی عبور از فضاهای سفید و کاراکترهای موجود در جملات توضیحی را به **WhiteSpaceAndCommentEliminator** می‌سپارد. یعنی وظیفه‌ی این شیء، خواندن کاراکترهای ورودی است تا این‌که همه‌ی کاراکترهای فضای سفید و کاراکترهای جملات توضیحی خوانده شوند و در این نقطه، کنترل را به کامپایلر باز می‌گرداند تا کاراکترهای غیرفضای سفید و غیرتوضیحی را بخواند و پردازش کند. به این بیندیشید که شیء WhiteSpaceAndCommentEliminator چگونه کاراکترها را یک به یک می‌خواند و تعیین می‌کند که آیا کاراکتر بعدی فضای سفید است یا بخشی از یک جمله‌ی توضیحی. این شیء می‌تواند فضاهای سفید را با جستجو به دنبال کاراکتر بعدی " "، "\t"، "\n"، و "\r" چک کند. ولی چگونه می‌تواند تعیین کند که آیا کاراکتر بعدی بخشی از یک جمله‌ی توضیحی است. برای مثال، هنگامی که "/" را برای نخستین بار می‌بیند، هنوز نمی‌داند که آیا آن کاراکتر یک عملگر تقسیم است، بخشی از عملگر "=/" است یا شروع یک جمله‌ی توضیحی است. برای انجام این تعیین، **WhiteSpaceAndCommentEliminator** باید به این واقعیت توجه داشته باشد که یک "/" دیده است و سپس به طرف کاراکتر دیگر حرکت کند. اگر کاراکتری که از پس "/" می‌آید، یک "/" دیگر یا یک "*" باشد، **WhiteSpaceAndCommentEliminator** می‌فهمد که اکنون در حال خواندن یک جمله‌ی توضیحی است و می‌تواند تا انتهای جمله را پیش برود بدون این‌که کاراکترها را پردازش یا ذخیره کند. اگر کاراکتری که پس از "/" می‌آید، چیزی غیر از "/" یا "*" باشد، در آن صورت، **WhiteSpaceAndCommentEliminator** می‌فهمد که "/" نشانگر عملگر تقسیم یا بخشی از عملگر =/ است و بنابراین، از پیشروی بیشتر در کاراکترها خودداری می‌کند.

1. State diagrams



شکل پ ۱-۱۳ نمودار حالت برای پیشروی کامپایلر پس از رسیدن به فضای خالی یا جملات توضیحی در زبان جاوا

به طور خلاصه، همچنان که **WhiteSpaceAndCommentEliminator**، کاراکترها را می‌خواند، باید چند چیز را مدنظر داشته باشد، از جمله این‌که آیا کاراکتر فعلی فضای سفید است، آیا کاراکتر قبلی که می‌خواند، "/" بوده است، آیا در حال حاضر کاراکترهای موجود در یک جمله‌ی توضیحی را می‌خواند، آیا به پایان جمله‌ی توضیحی رسیده است و غیره. این‌ها همگی متناظر با حالت‌های متفاوت شیء **WhiteSpaceAndCommentEliminator** هستند. در هر کدام از این حالت‌ها، **WhiteSpaceAndCommentEliminator** بسته به کاراکتر بعدی متفاوت رفتار می‌کند.

برای کمک به مصورسازی همه‌ی حالت‌های این شیء و چگونگی تغییر حالت‌ها می‌توانید از نمودار حالت UML شکل پ ۱-۱۳ استفاده کنید. نمودار حالت نشانگر حالت‌ها با استفاده از مستطیل‌های لبه‌گرد است که هر کدام در نیمه‌ی بالایی خودش یک نام دارد. همچنین یک دایره‌ی سیاه به نام "شبه‌حالت اولیه" وجود دارد که در واقع یک حالت نیست بلکه تنها به حالت اولیه اشاره دارد. در شکل پ ۱-۱۳، حالت Start حالت اولیه است. پیکان‌هایی که یک حالت را به حالت دیگر متصل می‌کنند، گذارها یا تغییر حالت‌ها را نشان می‌دهند. هر گذار با یک رویداد آغازگر، یک خط مایل (/) و یک فعالیت آغاز می‌شود. همه‌ی بخش‌های برچسب‌های گذار در نمودارهای حالت، اختیاری‌اند. اگر شیء در یک حالت باشد و رویداد آغازگر مربوط به یکی از گذارهای آن رخ دهد، در آن صورت،

فعالیت آن گذار، اجرا می‌شود و شیء حالت جدیدی را به خود می‌گیرد که توسط گذار مشخص می‌شود. برای مثال، در شکل پ ۱-۱۳، اگر شیء **WhiteSpaceAndCommentEliminator** از آن کاراکتر عبور می‌کند به **saw'/'** تغییر حالت می‌دهد. اگر کاراکتر مابعد "/" یک "/" باشد، در آن صورت شیء به **line comment** تغییر حالت می‌دهد و آن‌جا باقی می‌ماند تا این‌که به کاراکتر end of line (پایان خط) برسد. در عوض، اگر کاراکتر بعد از "/" یک "*" باشد، در آن صورت شیء به حالت **block comment** می‌رود و در همان حالت باقی می‌ماند تا این‌که یک "*" و سپس یک "/" دیگر ببیند که این نشانگر انتهای جمله‌ی توضیحی است. نمودار را بررسی کنید تا اطمینان پیدا کنید که آن را فهمیده‌اید.

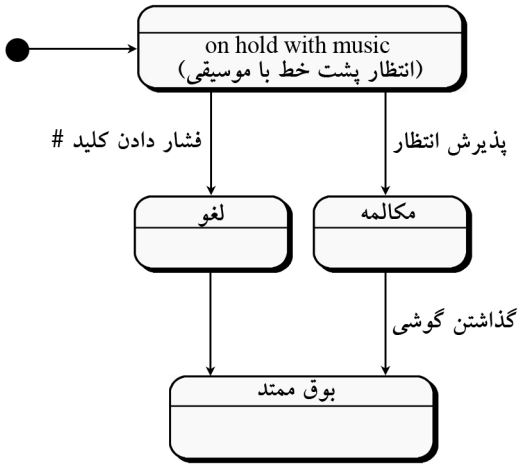
توجه دارید که همه‌ی گذارها به جز دو گذاری که به حالت نهایی منجر می‌شوند، دارای فعالیت‌هایی هستند که شامل رفتن به کاراکتر بعدی می‌شوند.

ممکن است یک شیء به حالتی نهایی گذار کند که توسط دایره‌ای سیاه با یک دایره‌ی دور آن نمایش داده می‌شود که نشان می‌دهد گذار دیگری وجود نخواهد داشت. در شکل پ ۱-۱۳، شیء **WhiteSpaceAndCommentEliminator** هنگامی پایان می‌گیرد که کاراکتر بعدی، فضای خالی یا بخشی از یک جمله‌ی توضیحی نباشد. توجه دارید که همه‌ی گذارها به جز دو گذاری که به حالت نهایی ختم می‌شوند، دارای فعالیت‌هایی هستند که از پیشروی به کاراکتر بعدی تشکیل می‌شوند. دو گذار به حالت نهایی به کاراکتر بعدی نمی‌روند چون کاراکتر بعدی بخشی از یک واژه یا نماد توجه به کامپایلر است. توجه دارید که اگر شیء در حالت **saw'/'** باشد، ولی کاراکتر بعدی "/" یا "*" نباشد، در آن صورت، "/" یک عملگر تقسیم یا بخشی از عمل‌گر =/ است و بنابراین، نمی‌خواهیم که ادامه دهیم. در واقع، می‌خواهیم یک کاراکتر به عقب باز گردیم تا "/" کاراکتر بعدی شود به طوری که "/" توسط کامپایلر قابل استفاده شود. در شکل پ ۱-۱۳، این فعالیت بازگشت به عقب، به صورت عقبگرد "/" برچسب‌گذاری می‌شود.

نمودار حالت، شما را در کشف وضعیت‌های غیرمنتظره یا از قلم افتاده یاری می‌دهد. یعنی، به کمک نمودار حالت، نسبتاً به آسانی می‌توانید اطمینان حاصل کنید که همه‌ی رویدادهای تریگر[1] ممکن، برای همه‌ی حالت‌های ممکن مدنظر بوده‌اند. برای مثال، در شکل پ ۱-۱۳، می‌توانید به آسانی اطمینان حاصل کنید که در هر حالت، گذارهای مربوط به همه‌ی کاراکترهای ممکن لحاظ شده است.

نمودارهای حالت UML می‌توانند حاوی ویژگی‌های بسیار دیگری باشند که در شکل پ ۱-۱۳ لحاظ نشده‌اند. برای مثال، هنگامی که شیءای در یک حالت باشد، معمولاً کاری انجام نمی‌دهد جز این‌که منتظر بماند تا یک رویداد تریگر رخ دهد. ولی یک نوع خاص از حالت به نام *حالت فعالیت* وجود دارد که در آن، شیء یک فعالیت به نام do-activity را انجام می‌دهد در حالی که در آن حالت قرار دارد. برای این‌که در نمودار حالت، نشان دهیم حالت، یک حالت فعالیت است، در نیمه‌ی پایینی مستطیل لبه‌گرد حالت، عبارت "do/" و پس از آن نام فعالیتی آورده می‌شود که باید در آن حالت ذکر شود.

1. Trigger



شکل پ ۱-۱۴ نمودار حالت با حالت فعالیت و یک گذار بدون رویداد آغازگر

ممکن است do-activity قبل از رخ دادن هر گذر حالتی پایان یابد و پس از آن، حالت فعالیت همانند یک حالت انتظار عادی عمل می‌کند. اگر یک گذار خارج از حالت فعالیت و پیش از پایان یافتن do-activity رخ دهد، در آن صورت، do-activity متوقف می‌شود.

از آن‌جا که رویداد تریگر به هنگام رخ دادن گذار، اختیاری است، این امکان وجود دارد که هیچ رویداد تریگری به عنوان بخشی از برچسبِ گذار تعیین نشود. در چنین مواردی، برای حالت‌های انتظار عادی، شیء بلافاصله از آن حالت به حالتِ جدید گذار می‌کند. برای حالت‌های فعالیت، چنین گذاری به محض پایان یافتن do-activity رخ می‌دهد.

شکل پ ۱-۱۴ این وضعیت را با استفاده از حالت‌های مربوط به یک تلفن تجاری نشان می‌دهد. هنگامی که تماس‌گیرنده پشت خط منتظر گذاشته می‌شود، تماس به حالت **on hold with music** می‌رود (به مدت ده ثانیه موسیقی به گوش می‌رسد). پس از ده ثانیه، do-activity برای این حالت کامل می‌شود و حالت همانند یک حالت عادی و بدون فعالیت رفتار می‌کند. اگر تماس‌گیرنده در حالت **on hold with music**، کلید # را فشار دهد، تماس به حالت **canceled** و سپس بلافاصله به حالت **dial tone** گذار می‌کند. اگر کلید # قبل از به پایان رسیدن ده ثانیه موسیقی فشرده شود، do-activity متوقف شده و موسیقی بلافاصله پایان می‌یابد.

زبان قید و بند اشیا – نگاهی اجمالی

گستره‌ی وسیعی از نمودارها که به عنوان بخشی از UML در دسترس است، مجموعه‌ای غنی از شکل‌های نمایشی را برای مدل طراحی در اختیار شما می‌گذارد. ولی، نمایش‌های گرافیکی غالباً کافی نیستند. ممکن است برای وضوح بیشتر و نمایش رسمی اطلاعاتی که عنصری از مدل طراحی را مقید می‌کنند، راهکاری نیاز داشته باشید. البته، توصیف قیدوبندها به زبان طبیعی نظیر فارسی امکان‌پذیر است،

ولی این رویکرد ناگزیر به ناسازگاری و ابهام خواهد انجامید. به این دلیل، یک زبان رسمی‌تر، زبانی که از نظریه‌ی مجموعه‌ها و زبان‌های مشخص‌سازی رسمی بهره گیرد (فصل ۲۸ و پیوست ۳)، ولی قالب نحوی آن به اندازه‌ی زبان‌های برنامه‌نویسی ریاضی نباشد – مناسب به نظر می‌رسد.

زبان قیدوبند اشیا[1] (OCL) از این رومکمل UML است که به کمک آن می‌توانید از یک گرامر و قالب نحوی رسمی برای ایجاد گزاره‌های عاری از ابهام درباره‌ی انواع عناصر مدل طراحی (از قبیل کلاس‌ها و اشیا، رویدادها، پیام‌ها و واسط‌ها) استفاده کنید. ساده‌ترین گزاره‌های OCL در چهار بخش تنظیم می‌شوند: (۱) حیطه‌ای که وضعیت محدودی را تعریف می‌کند که گزاره در آن معتبر است، (۲) خاصیتی که خصوصیات حیطه را نشان می‌دهد (مثلاً اگر حیطه، یک کلاس باشد، این خاصیت ممکن است یک خصیصه باشد)، (۳) یک عمل (مثلاً محاسباتی یا مجموعه‌ای) که خاصیتی را دستکاری یا آماده می‌کند و (۴) واژه‌های کلیدی (مثلاً **اگر**، **آن‌گاه**، **دیگر**، **و**، **یا**، **خیر**، نتیجه می‌شود که) که برای مشخص‌سازی عبارت‌های شرطی استفاده می‌شوند.

به عنوان مثالی از عبارت‌های OCL، سیستم چاپ بحث شده در فصل ۱۴ را در نظر بگیرید. شرط نگهبان قرار داده شده بر رویداد **jobCostAccepted** را در نظر بگیرید که باعث گذار میان حالت‌های **computingJobCost** و **formingJob** در داخل نمودار حالت برای شیء **PrintJob** می‌شود (شکل ۱۴–۹). در این نمودار (شکل ۱۴–۹ در فصل ۱۴)، شرط نگهبان، به زبان طبیعی بیان می‌شود و از آن چنین برمی‌آید که صدور اجازه تنها در صورتی می‌تواند رخ دهد که مشتری مجاز به تصویب هزینه‌ی کار باشد. در OCL، این عبارت ممکن است به شکل زیر بیان شود:

```
customer
     self.authorizationAuthority = 'yes'
```

که در آن یک خصیصه‌ی بولی، authorizationAuthority، از کلاس (در واقع نمونه‌ی مشخصی از کلاس) به نام **Customer** باید برابر با "yes" قرار داده شود تا شرط نگهبان برقرار شود.

به موازاتی که مدل طراحی ایجاد می‌شود، غالباً نمونه‌هایی وجود دارد که در آن‌ها پیش‌شرط و پس‌شرط باید قبل از تکمیل یک کنش مشخص‌شده توسط طراحی برقرار باشند. OCL، ابزاری پرقدرت برای مشخص کردن پیش‌شرط‌ها و پس‌شرط‌ها به شیوه‌ای رسمی فراهم می‌آورد. به عنوان مثال، فرض کنید سیستم چاپخانه (فصل ۱۴) را بسط می‌دهیم به طوری که مشتری برای هزینه‌ی کار چاپی را مشخص می‌کند، یک تاریخ تحویل نیز تعیین می‌کند. اگر برآورد هزینه و تحویل از این دو مقدار مرزی فراتر برود، کار تسلیم نمی‌شود و مشتری باید مطلع شود. در OCL، مجموعه‌ای از پیش‌شرط‌ها و پس‌شرط‌ها را می‌توان به شیوه زیر مشخص کرد:

```
context PrintJob::validate(upperCostBound : Integer,
custDeliveryReq :Integer)
```

1. Object Constraint Language

```
pre: upperCostBound > 0
     and custDeliveryReq > 0
     and self.jobAuthorization = 'no'
post: if self.totalJobCost <= upperCostBound
and self.deliveryDate <= custDeliveryReq
    then
      self.jobAuthorization = 'yes'
    endif
```

این عبارت OCL، یک ثابت (inv)- شرط‌هایی تعیین می‌کند که باید قبل و بعد از یک رفتار وجود داشته باشند. در آغاز، این پیش‌شرط برقرار می‌شود که هزینه و تاریخ تحویل باید توسط مشتری تعیین شود و صدور مجور باید "no" باشد. پس از تعیین هزینه‌ها و تاریخ تحویل، پس‌شرط اِعمال می‌شود. همچنین باید متذکر شد که عبارت:

```
self.jobAuthorization = 'yes'
```

مقدار "yes" را نسبت نمی‌دهد بلکه اعلان می‌کند که jobAuthorization باید در پایان عمل برابر با "yes" شده باشد. شرح کاملی از OCL از حوصله‌ی این پیوست خارج است. مشخصات کامل OCL را می‌توانید در www.omg.org/technology/documents/formal/ocl.htm بیابید.