موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

درس و کنکور ارشد

مهندسی نرم‌افزار

(مدل تحلیل و مدل طراحی شیءگرا)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی IT

براساس کتاب مرجع

راجر اس. پرسمن هشتم ۲۰۱۴

ارسطو خلیلی‌فر

۷ مدل تحلیل و مدل طراحی شیءگرا

مقدمه

فرآیند تولید نرم‌افزار در متدولوژی شیءگرا نیز همانند متدولوژی ساخت‌یافته با فعالیت ارتباط آغاز می‌گردد. با این تفاوت که در فعالیت مدل تحلیل شیءگرا به جای استفاده از مفاهیم ساخت‌یافته و نمودارهایی همچون نهاد و رابطه (ERD) و جریان داده (DFD)، از مفاهیم شیءگرا و نمودارهای UML استفاده می‌گردد.

در فعالیت مدل‌سازی شیءگرا برخلاف شیوه ساخت‌یافته، سیستم به عناصر داده و عملکرد تفکیک نمی‌شود بلکه واحدهایی ایجاد می‌گردد که هر یک از آنها شامل عناصر داده و عملکرد (متد) مرتبط با آن می‌باشد (جعبه سیاه). این واحدها را کلاس می‌نامند. از دیدگاه شیءگرا **دامنه‌ی مسئله** از تعدادی کلاس و روابط میان آنها تشکیل شده است.

فعالیت‌های چارچوبی فرآیند تولید نرم‌افزارها بر اساس متدولوژی شیءگرا

۱- فعالیت ارتباط (مهندسی نیازمندی‌ها)

در این مرحله لیست نیازمندی‌های مشتری از طریق ارتباط با مشتری و ابزارهایی همچون گفتگو، مشاهده مصاحبه، پرسش‌نامه، بازدید، نمونه‌سازی دورانداختنی و نمونه‌سازی تکاملی، جمع‌آوری و آماده می‌گردد.

توجه: مهم‌ترین کار، در فعالیت ارتباطات، مدیریت شناسایی نیازمندی‌ها و پیگیری تغییرات نیازمندی‌های مشتری است.

توجه: لیست نیازمندی‌های مشتری در این مرحله به صورت متن نوشته می‌شود.

مثال: نمایش لیست نیازمندی‌های مشتری برای سیستم ATM.

۱- واریز وجه
۲- انتقال وجه
۳- برداشت وجه
۴- تغییر رمز
۵- پرداخت
۶- نمایش موجودی

۲- فعالیت برنامه‌ریزی

برنامه‌ریزی یعنی هنر حرکت از مبدأ موجود به مقصد مطلوب برای رسیدن به نتیجه‌ای مطلوب بر اساس خواسته‌های مورد نیاز در یک زمان مشخص.

«هر تلاشی منجر به نتیجه‌ای مطلوب نمی‌گردد، بلکه این تلاشی مطلوب است که منجر به نتیجه‌ای مطلوب می‌گردد.»

لازمه‌ی تلاش مطلوب، برنامه‌ریزی است. برنامه‌ریزی می‌تواند اجرای هر کار پیچیده‌ای را ساده‌تر سازد. هر کار مهندسی مستلزم برنامه‌ریزی می‌باشد. مهندسی نرم‌افزار نیز مانند هر فعالیت مهندسی دیگری، نیازمند برنامه‌ریزی است. فعالیت برنامه‌ریزی، برنامه‌ای را برای فعالیت‌های مختلف بخش‌های مختلف فرآیند تولید نرم‌افزار پایه‌ریزی می‌کند. این فعالیت، وظیفه‌های فنی که باید هدایت شوند، ریسک‌هایی که محتمل می‌شوند (مانند عدم شناسایی برخی نیازمندی‌ها، از دست دادن داده‌ها و مدیران)، منابع مورد نیاز، واحدهای کاری که باید ایجاد شوند و برنامه زمان‌بندی برای کارها را تشریح می‌کند. مدیریت، برنامه‌ریزی را به مرحله‌ی اجرا و نظارت می‌برد.

۳- فعالیت مدل‌سازی (تحلیل و طراحی)

یک مدل، ساده شده یک واقعیت است. ایجاد یک مدل برای سیستم‌های نرم‌افزاری قبل از ساخت یا بازساخت آن، به اندازه داشتن نقشه برای ساختن یک ساختمان ضروری وحیاتی است. بسیاری از شاخه‌های مهندسی، توصیف چگونگی محصولاتی که باید ساخته شوند را ترسیم می‌کنند و همچنین دقت زیادی می‌کنند که محصولاتشان طبق این مدل‌ها و توصیف‌ها ساخته شوند. مدل‌های خوب و دقیق در برقراری یک ارتباط کامل بین افراد پروژه، نقش زیادی می‌توانند داشته باشند. علت اصلی مدل کردن سیستم‌های پیچیده این است که نمی‌توان به یکباره کل سیستم را تجسم کرد و ممکن است سیستم دارای ابهامات بسیاری باشد. لذا برای رفع این ابهامات و نیز برای فهم کامل سیستم، یافتن و نمایش ارتباط بین قسمت‌های مختلف آن، از مدل‌سازی استفاده می‌شود. فعالیت مدل‌سازی خود شامل دو مرحله‌ی مدل تحلیل و مدل طراحی می‌باشد. مدل تحلیل پس از فعالیت ارتباطات (جمع‌آوری نیازمندی‌ها) و قبل از مدل طراحی انجام می‌شود، در

واقع خروجی مدل تحلیل، ورودی مدل طراحی می‌باشد.

زبان مدل‌سازی یکپارچه (UML)

UML که سرواژه عبارت Unified Modeling Language و به معنی زبان مدل‌سازی یکپارچه است، نمودارهایی را برای مدل‌سازی سیستم‌های شیءگرا ارایه می‌دهد (فعالیت تحلیل تا استقرار).

توجه: از آنجا که UML از ترکیب زبان‌های مدل‌سازی مختلفی همچون Booch، OMT، Rumbough، Jacobson و غیره ایجاد شده است. این زبان، «**زبان مدل‌سازی یکپارچه**» نام‌گذاری شده است.

ادغام مدل‌ها به منظور ایجاد UML از سال ۱۹۹۳ آغاز گردید و تا سال ۱۹۹۵ ادامه داشت. یعنی تا زمانی که نسخه Unified Method معرفی شد.

Unified Method اصلاح شد و در سال ۱۹۹۶ به Unified Modeling Language تغییر نام یافت، نسخه UML 1.0 تأیید شد و در سال ۱۹۹۷ به گروه تکنولوژی آبجکت (Object Technology Group) داده شد و شرکت‌های تولیدکننده نرم‌افزاری شروع به سازگار شدن با آن نمودند. سرانجام در ۱۴ نوامبر ۱۹۹۷، OMG نسخه UML 1.1 را به عنوان یک استاندارد صنعتی معرفی نمود.

توجه: هر نظام مهندسی یک روش استاندارد برای مستندسازی دارد. مهندسی الکترونیک schematic diagrams، مهندسی معماری blueprints diagrams، و مهندسین مکانیک mechanical diagrams را دارند. در حال حاضر صنعت نرم‌افزار UML را در اختیار دارد.

مدل تحلیل: OOA (Object-Oriented Analysis)

پس از جمع‌آوری نیازمندی‌ها در فعالیت ارتباطات، نوبت به مدل تحلیل (مدل‌سازی نیازمندی‌ها) می‌رسد. مدل‌سازی که فعالیتی فنی به شمار می‌رود، نیازمندی‌ها را باید به گونه‌ای مدل نماید که برای سازنده و مشتری قابل فهم باشد.

مدل تحلیل، شامل مراحل زیر می‌باشد:

الف) مدل‌سازی محیط عملیاتی یک کسب و کار

توجه: Business use case یا مورد کاربرد کسب و کار، جهت مدل‌سازی محیط عملیاتی یک کسب و کار مورد استفاده قرار می‌گیرد.

ب) مدل‌سازی لیست نیازمندی‌های مشتری

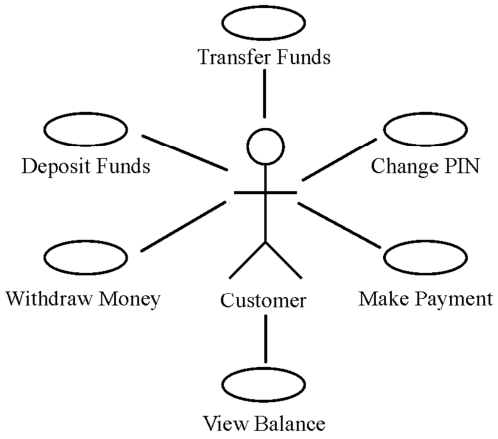
توجه: Use Case Diagram یا نمودار مورد کاربرد، Requirement Diagram یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد.

توجه: از آنجا که Use Case Diagram یا نمودار مورد کاربرد تعامل بین کاربر نهایی و برنامه

کامپیوتری را مدل‌سازی می‌کند، به این مدل، مدل مبتنی بر سناریوی میان کاربر نهایی و برنامه کامپیوتری نیز گفته می‌شود.

توجه: Use Case Diagram یا نمودار مورد کاربرد، می‌تواند به عنوان مدل لیست نیازمندی‌ها، به شکل چک لیست برای فعالیت تست مورد استفاده قرار بگیرد.

مثال: مدل‌سازی لیست نیازمندی‌های مشتری برای سیستم ATM.



توجه: در نمودار فوق، لیست نیازمندی‌های مشتری، مدل‌سازی شده است.

توجه: به هر یک از نیازهای فوق یک use case یا مورد کاربر گفته می‌شود و به اجتماع این use caseها، Use Case Diagram یا نمودار مورد کاربرد گفته می‌شود.

توجه: به یک use case یا مورد کاربرد، نیاز (requirement) یا زیرسیستم (subsystem) نیز گفته می‌شود.

توجه: از آنجا که Use Case Diagram یا نمودار مورد کاربرد یا نمودار نیاز مستقل از مفاهیم شیءگرایی (کلاس، وراثت و چندریختی) است، بنابراین می‌توان مدل‌سازی لیست نیازمندی‌های مشتری در مرحله **مدل تحلیل متدولوژی ساخت‌یافته** (مهندسی‌نرم‌افزار ساخت‌یافته) را توسط این نمودار انجام داد.

توجه: Use Case Diagram یا نمودار مورد کاربرد، هیچگاه به طور مستقیم، پیاده‌سازی نمی‌شود، بلکه مبنایی برای استخراج دیگر نمودارها قرار می‌گیرد، بنابراین ساختار نمودار کاربرد به پیاده‌سازی نرم‌افزار یا نسخه فیزیکی تبدیل نمی‌گردد.

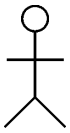
توجه: Use Caseها و Actorها محدوده سیستم در حال ساخت را مشخص می‌کنند. Use Case شامل تمام آن چیزهایی است که درون سیستم قرار دارد و Actor شامل تمام آن چیزهایی است که خارج از سیستم قرار دارد. هر فرد یا هر چیزی که با سیستم تعامل دارد، Actor

یا بازیگر نامیده می‌شود. Use Caseها هر چیز موجود در داخل و محدوده سیستم را توصیف می کنند، در حالی که Actorها هر چیز موجود در خارج از محدوده سیستم را توصیف می‌کنند.

توجه: بازیگران، افراد و یا گاهی نرم‌افزار و یا سخت‌افزارهایی هستند که از سیستم استفاده می‌کنند و یا اطلاعاتی را برای سیستم فراهم می‌کنند. با اینکه همه بازیگران، عناصر خارجی سیستم هستند، اما با این حال هر عنصر خارج سیستم، بازیگر نیست. بازیگران استفاده کنندگان نهایی و یا تولیدکنندگان ابتدایی اطلاعات هستند، بنابراین عناصر خارجی سیستم که تنها وظیفه انتقال اطلاعات را دارند و نقش **رسانه انتقال** را ایفا می‌کنند، نمی‌توانند به عنوان بازیگر در نظر گرفته شوند.

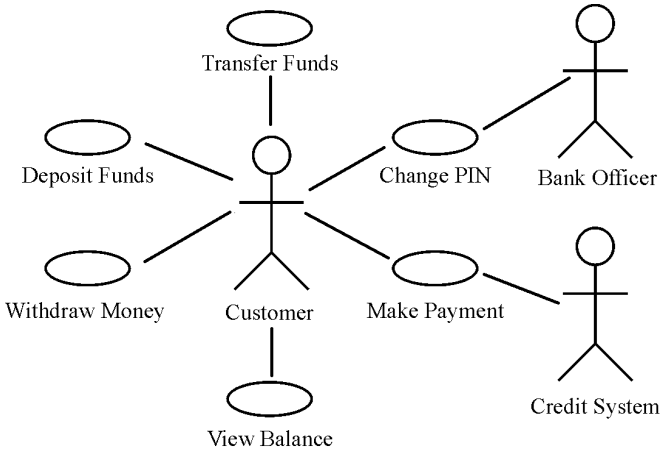
برای‌مثال اگر یک سنسور از طریق رسانه بی‌سیم، اطلاعاتی را به سیستم مرکزی ارسال می‌کند، رسانه بی‌سیم نمی‌تواند بازیگر این سیستم باشد، بلکه سنسور بازیگر آن خواهد بود. بنابراین هر بازیگر اگر استفاده‌کننده باشد نقطه انتها و اگر تولیدکننده اطلاعات باشد نقطه ابتدای آن ارتباط است.

در UML، بازیگران با آدمک‌هایی به شکل زیر نشان داده می‌شوند:



Actor

نمونه‌ای از استفاده نمودار Use Case در شکل زیر نشان داده شده است: (سیستم ATM)



رابطه انجمنی بین Use Case و Actor

توجه: در نمودار مورد کاربرد، رابطه بازیگران با موارد کاربرد از نوع انجمنی است. خط

ممتدی که مابین یک بازیگر و یک مورد کاربرد کشیده می‌شود، نشان‌دهنده رابطه انجمنی میان آن‌ها است و بدین معنا است که بازیگر و مورد کاربرد مذکور با یکدیگر در ارتباط هستند. جهت خط نیز، جهت ارتباط را نشان می‌دهد. بعضی از روابط انجمنی در نمودار مورد کاربرد، به صورت یک طرفه هستند، مانند زمانی که یک بازیگر بدون انتظار برای پاسخ، اطلاعاتی را در اختیار یک مورد کاربرد قرار می‌دهد. اما اغلب روابط انجمنی به صورت دو طرفه هستند، مانند زمانی که یک بازیگر به یک مورد کاربرد، متصل شده و از مورد کاربرد درخواست‌هایی دارد و مورد کاربرد نیز عملیات مورد نیاز را برای بازیگر به انجام رسانده و نتایج را به او بر می‌گرداند. رابطه انجمنی دو طرفه می‌تواند با فلش دو جهته نمایش داده شود، اما از آن جایی که اغلب رابطه‌ها دو طرفه هستند، غالبا از خط بدون فلش استفاده می‌گردد. شکل فوق گویای این مطلب می‌باشد.

توجه: هر بازیگر ممکن است چندین مورد کاربرد را اجرا کند و یک مورد کاربرد نیز می‌تواند توسط چندین بازیگر اجرا یا استفاده شود.

انواع روابط بین Use Caseها

علاوه بر روابطی که بین بازیگرها با موردهای کاربرد وجود دارد ممکن است بین موارد کاربرد با یکدیگر نیز روابطی وجود داشته باشد که عبارتند از:

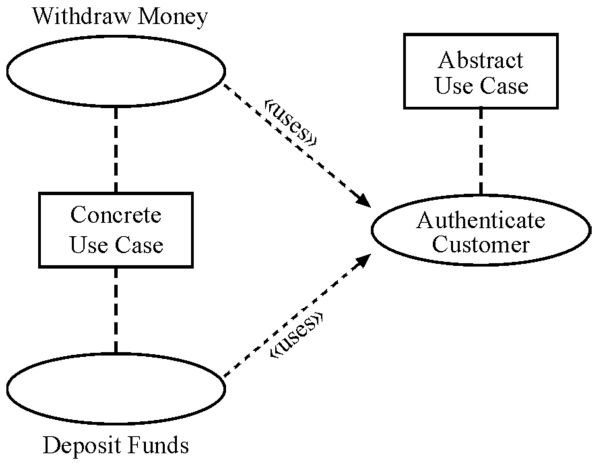
رابطه «شامل بودن» (Include) یا (Uses)

رابطه Include یا Uses، به یک Use Case، اجازه استفاده از عملیات مهیا شده توسط یک Use Case دیگر را می‌دهد. این نوع رابطه، برای مدل‌سازی برخی عملیاتی که بین دو یا تعداد بیشتری Use Case به طور مشترک استفاده می‌شوند، به کار می‌روند. به بیان دیگر اگر بدنه اصلی چند مورد کاربرد دارای بخش مشترکی باشند که به اندازه کافی بزرگ بوده، به طوری که بتوان آن را به عنوان یک مورد کاربرد مجزا در نظر گرفت، آنگاه می‌توان بخش مشترک مذکور را از درون موارد کاربرد متناظر خارج کرد و آن را به عنوان یک مورد کاربرد مجزا تعریف نمود. در این حالت موارد کاربرد اولیه و جدید به کمک رابطه «Include» به یکدیگر مرتبط شده و موارد کاربرد اولیه شامل مورد کاربرد جدید خواهند شد. بدین ترتیب به جای آن که یک عملیات، چندین بار در موارد مختلف تکرار شود، می‌تواند در قالب یک مورد کاربرد جداگانه، توسط بقیه موارد کاربرد مورد استفاده قرار بگیرد. بدیهی است مورد کاربرد جدید نمی‌تواند به طور مجزا دارای بازیگر باشد. زیرا این مورد کاربرد نمی‌تواند به طور مستقیم توسط یک بازیگر درخواست شود. بلکه یک بازیگر می‌تواند یکی از موارد کاربرد اولیه را درخواست نماید، در اینصورت حتما این مورد کاربرد نیز در حین اجرای آنها اجرا خواهد شد. در مثال ATM، در دو Use Case مربوط به برداشت وجه (Withdraw Money) از حساب و واریز وجه (Deposit Funds) به حساب، نیاز به شناسایی مشتری و PIN آنها، قبل از پیگیری عملیات است. بنابراین به جای تعریف پردازش شناسایی در هردوی این Use Caseها، می‌توان در یک Use Case مجزا به نام Authenticate Customer، این

عملیات را تعریف کرد. هر زمان دیگری که یک Use Case نیاز به شناسایی یک کاربر دارد، می‌تواند از عملیات موجود در Use Case موجود به نام Authenticate Customer استفاده کند.

توجه: رابطه «شامل بودن» در UML به صورت فلش نقطه چینی است که از سمت موارد کاربرد اولیه به سمت مورد کاربرد جدید رسم می‌شود و با برچسب «Include» یا «Uses» نشان داده می‌شود.

مثال:



یک رابطه‌ی Uses

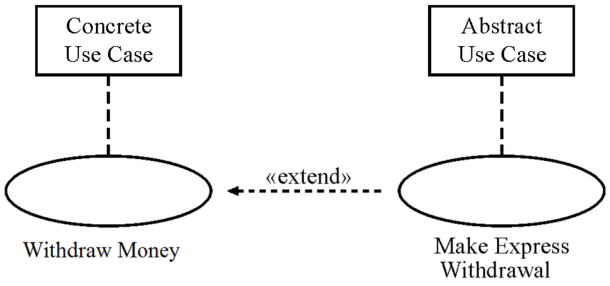
توجه: رابطه Include یا Uses در یک کلیشه «stereotype» نمایش داده می‌شود. کلیشه‌ها در نمودارهای UML توضیحات تکمیلی را ارئه می‌دهند و با نماد «» نمایش داده می‌شوند.

رابطه «توسعه دادن» (Extend)

یک رابطه‌ی extend به یک Use Case اجازه می‌دهد که به طور دلخواه عملیات مهیا شده توسط Use Caseهای دیگر را بسط دهد.

توجه: رابطه «توسعه دادن» در UML به صورت فلش نقطه چینی است که از سمت مورد کاربرد جدید به سمت مورد کاربرد اولیه رسم می‌شود و با برچسب «Extend» نشان داده می‌شود. در واقع، مورد کاربرد جدید، در موارد خاص به موارد کاربرد اولیه اضافه می‌گردد و باعث توسعه و گسترش آنها می‌شود. مورد کاربرد جدید در رابطه «Include» همواره در حین اجرای موارد کاربرد اولیه اجرا می‌شود، اما در رابطه «Extend»، مورد کاربرد جدید ممکن است در اثر برقراری شرایط خاصی اجرا گردد.

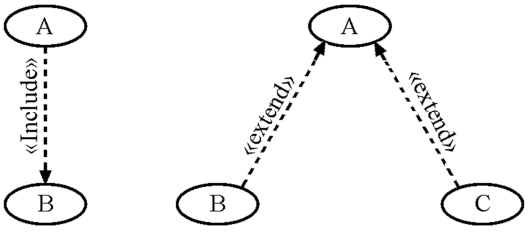
مثال:



یک رابطه‌ی extend

در مثال فوق، use case مربوط به برداشت پول (Withdraw Money) گاهی اوقات از عملیات موجود در use case برداشت سریع از حساب (Make Express Withdrawal) استفاده می‌نماید، اگر و فقط اگر کاربر یک میلیون ریال برداشت سریع را از use case برداشت پول (Withraw Money) درخواست کند.

توجه: در رابطه «Include» وقتی "A" اجرا شود، "B" هم **حتماً** اجرا می‌شود ولی در رابطه «extend» وقتی "A" اجرا شود، **ممکن** است "B" یا "C" یا هیچ‌کدام از Use Caseها اجرا نشوند.



توجه: در رابطه «extend» "B" یا "C" زمانی مورد استفاده "A" قرار می‌گیرد که "A" اجازه‌ی توسعه، در نقاط قابل توسعه (Extension Point) را فراهم نماید.

ج) سناریونویسی

در این مرحله برای هر use case (مورد کاربرد یا نیاز) سناریو یا شرح حال نوشته می‌شود.
سناریو بر دو طبقه اصلی و فرعی می‌باشد:

سناریوی اصلی: بیان روال حرکت قدم به قدم کارها داخل یک use case از نقطه شروع تا رسیدن به یک نتیجه مطلوب (موفق). (حرکت از خود موجود و رسیدن به خود مطلوب).

مثال: بیان روال حرکت قدم به قدم، برای برداشت وجه موفق از یک حساب، مربوط به use case برداشت وجه در یک سیستم ATM .

سناریوی فرعی: بیان روال حرکت قدم به قدم کارها داخل یک use case از نقطه شروع تا رسیدن به یک نتیجه نامطلوب (ناموفق). (حرکت از خود موجود و رسیدن به خود نامطلوب).

مثال: بیان روال حرکت قدم به قدم، برای برداشت وجه ناموفق از یک حساب، مربوط به use case برداشت وجه در یک سیستم ATM .

توجه: برداشت وجه ناموفق، دلایل مختلفی دارد، همچون رمز عبور نادرست، عدم موجودی کافی در حساب، عدم موجودی اسکناس کافی در صندوق فیزیکی و ... .

توجه: هر use case، فقط و فقط یک سناریوی اصلی دارد و می‌تواند چندین سناریوی فرعی داشته باشد.

نحوه نمایش سناریوی اصلی و فرعی به دو روش زیر می‌باشد:

نوشتاری(متنی): در این روش سناریوی اصلی و فرعی به صورت متنی نوشته می‌شود.

گرافیکی(نموداری): در این روش سناریوی اصلی و فرعی به صورت نمودار، مدل‌سازی می‌شود.

توجه: Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط شنا، جهت مدل‌سازی سناریوی اصلی و فرعی داخل یک use case (نیاز یا مورد کاربرد یا زیرسیستم) مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار فعالیت یا نمودار خط شنا، جهت مدل‌سازی روال انجام کارها داخل یک use case مورد استفاده قرار می‌گیرد.

نمودار فعالیت (Activity Diagram)

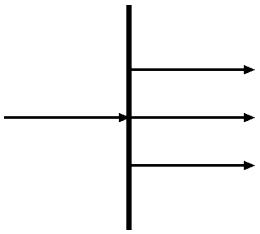
همانطور که پیش از این نیز گفتیم، فلوچارت ابزاری است که نحوه اجرای یک برنامه را به صورت گرافیکی در اختیار برنامه‌نویس قرار می‌دهد. نمودار فعالیت، نسخه توسعه‌یافته فلوچارت است که در UML مورد استفاده قرار می‌گیرد. بنابراین فلوچارت و نمودار فعالیت شباهت زیادی به یکدیگر دارند.

به هر سازمانی که مراجعه کنید با مفهوم روال انجام کار، سر و کار خواهید داشت. روال انجام کار، مراحل و نحوه انجام کارها را در سیستم مشخص می‌کند و به جزئیات آن نمی‌پردازد، به طوری که مشتری یا استفاده‌کننده سیستم، بدون نیاز به پرسش، از مراحلی که باید برای به انجام رساندن کار خود طی کند، آگاهی می‌یابد. برای مثال، روال انجام کار مربوط به یک سناریوی موفق برای «برداشت وجه»، به این صورت است که: «ابتدا کارت در دستگاه کارت خوان وارد می‌شود، سپس رمز عبور وارد می‌شود، مبلغ درخواستی مشخص می‌شود، مبلغ درخواستی از حساب مورد نظر کسر می‌شود، مبلغ مورد نظر توسط صندوق آماده می‌گردد و در انتها پس از دریافت وجه، رسید تحویل می‌گردد». بنابراین روال انجام کار، یک سری فعالیت متوالی است که در هر سازمان، برای رسیدن به هدف مورد نظر طی می‌شود. از این رو در هر سازمانی می‌توان این سوال را مطرح

کرد که «روال انجام کار یک عملیات خاص چگونه است؟». همانطور که گفتیم این روال انجام کار یا سناریو به صورت نوشتاری یا گرافیکی بیان می‌شود. نمودار فعالیت ابزار مناسبی برای نمایش گرافیکی گردش کار است. در نمودار فعالیت هر یک از فعالیت‌های مربوط به روال انجام کار، توسط نماد مستطیل گوشه گرد، جریان کار توسط نماد پیکان، نقاط تصمیم گیری توسط نماد لوزی، نقطه شروع روال انجام کار توسط نماد دایره توپر و نقاط پایان روال انجام کار توسط نماد دایره توپر مضاعف نشان داده می‌شود. نمادهای هم‌روندی و هم‌زمانی نیز در ادامه بررسی می‌گردد.

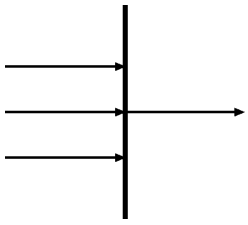
نماد هم‌روندی (Concurrency)

در نمودار فعالیت، هم‌روندی نیز قابل مدل‌سازی است. هم‌روندی در فلوچارت وجود ندارد، اما در نرم‌افزارهای امروزی زیاد مورد استفاده قرار می‌گیرد. برای مثال وقتی شما چند کار را با هم انجام می‌دهید در آنجا هم‌روندی رخ داده است. برای مدل‌سازی هم‌روندی از شکل یک چنگال، مطابق شکل زیر استفاده می‌شود. به طوری که یک مسیر به میله هم‌روندی وارد شده و سپس مسیرهایی که باید به صورت موازی با یکدیگر به اجرا درآیند، از این میله خارج می‌شوند.

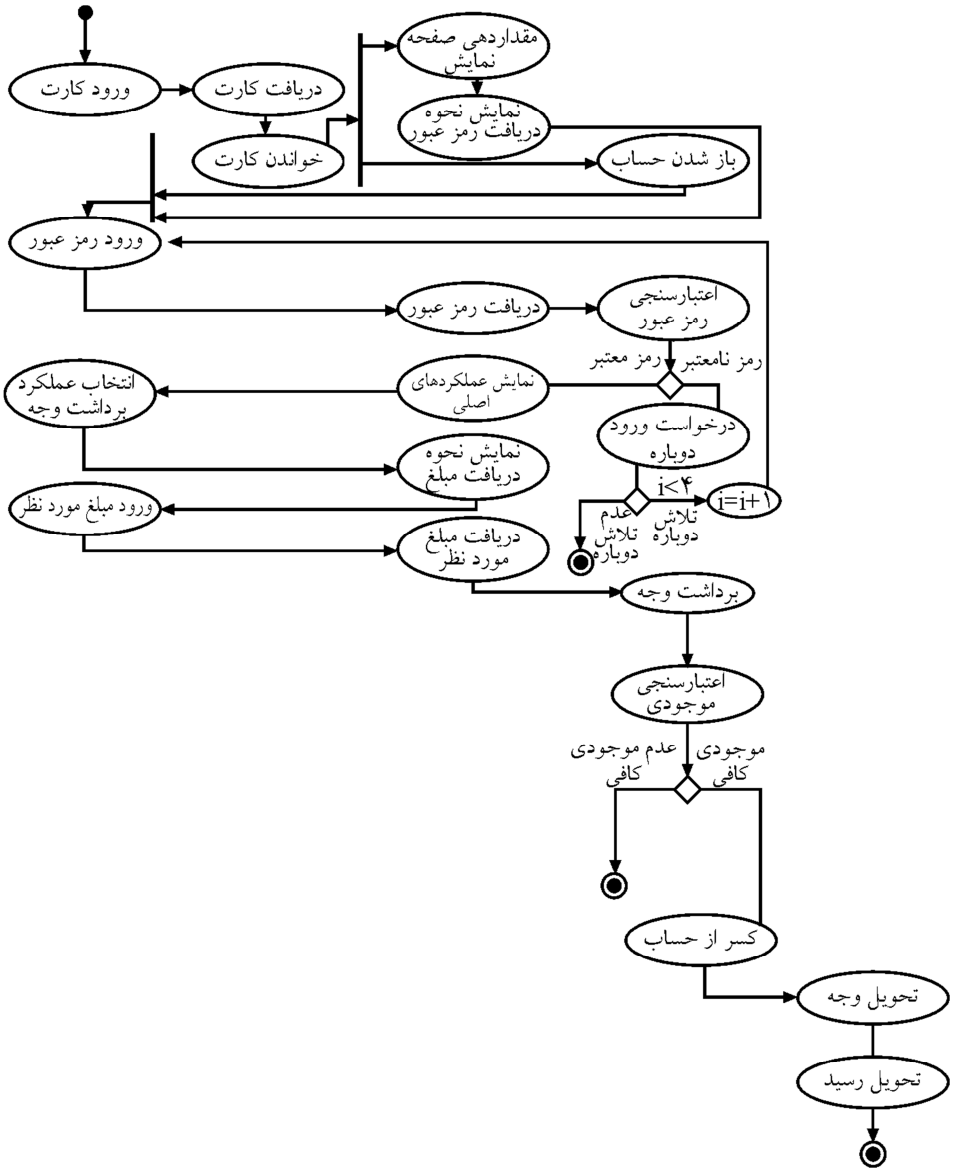


نماد هم‌زمانی (Synchronization)

مفهوم هم‌زمانی با هم‌روندی متفاوت است و برای ادغام مسیرهای اجرایی هم‌روند در نقطه انتهایی استفاده می‌شود. در شکل زیر این نماد نشان داده شده است.

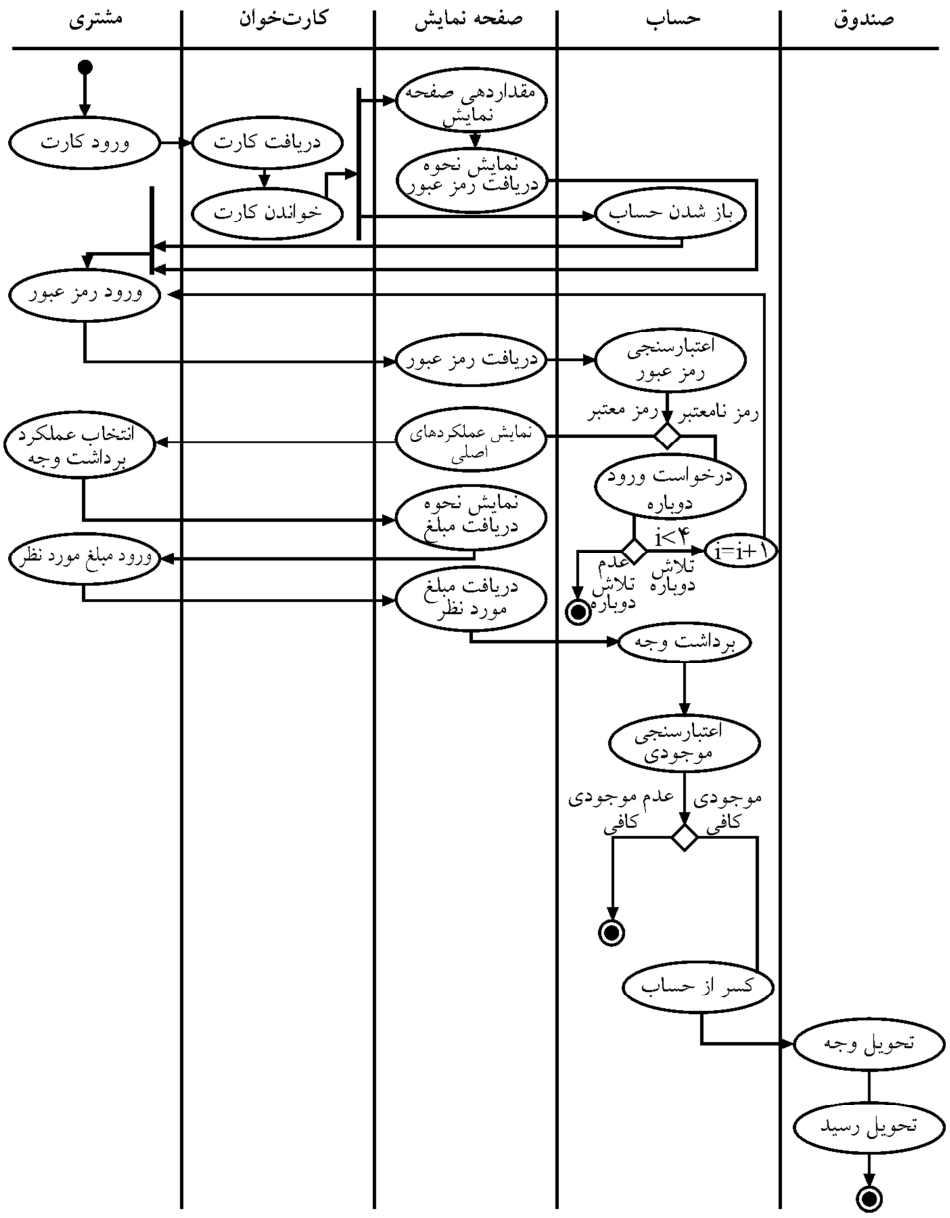


مثال: مدل‌سازی گردش کار مورد کاربرد «برداشت وجه» توسط نمودار فعالیت به صورت زیر است:



نمودار بخش‌بندی یا خط شنا (Swimlane Diagram)

نمودار بخش‌بندی، شکل توسعه یافته نمودار فعالیت است که در UML مورد استفاده قرار می‌گیرد. بنابراین نمودار فعالیت و نمودار بخش‌بندی شباهت زیادی به یکدیگر دارند. در نمودار بخش‌بندی علاوه بر مواردی که نمودار فعالیت نشان می‌دهد، عامل هر فعالیت نیز نشان داده می‌شود.



توجه: در شکل فوق هم‌روندی فعالیت‌های مقداردهی صفحه نمایش، نمایش نحوه دریافت رمز عبور و بازشدن حساب توسط **نماد هم‌روندی** و همچنین هم‌زمانی نقاط انتهایی آنها نیز توسط **نماد هم‌زمانی** نشان داده شده است.

د) کشف کلاس‌های همکار داخل هر use case یا مورد کاربرد

پس از شناسایی موارد کاربرد و سناریونویسی برای هر یک از موارد کاربرد، زمان تعریف کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌های همکار برای هر یک از موارد کاربرد می‌رسد. برای شناسایی کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌ها برای هر یک از موارد کاربرد، از تکنیکی موسوم به CRC که سرواژه‌ی عبارت Class – Responsibility Collaborator و به معنی «مدل همکاری مسئولیت‌های کلاس‌ها» می‌باشد، استفاده می‌شود. مدل‌سازی CRC روشی ساده جهت تعیین و سازماندهی کلاس‌های داخل هر مورد کاربرد است. در این روش به هر کلاس یک کارت CRC اختصاص داده می‌شود که شامل سه بخش کلی زیر است:

نام کلاس

مسئولیت‌های کلاس(Responsibilities)

- صفات کلاس
- متدهای کلاس

همکاران کلاس(Collaborators)

توجه: کلاس‌ها یا به تنهایی از عهده انجام مسئولیت‌های خود بر می‌آیند و یا از طریق همکاری با کلاس‌های همکار خود از عهده انجام مسئولیت‌های خود بر می‌آیند. بنابراین اگر کلاسی همکارانی دارد، باید در بخش مربوط به همکاران نوشته شود.

توجه: برای کشف کلاس‌های همکار داخل هر use case، از سناریوی اصلی (نوشتاری یا نموداری) هر use case استفاده می‌گردد. برای این منظور، **اسامی** موجود داخل هر use case مورد جستجو قرار می‌گیرند. از آنجا که نیازها یا موارد کاربرد مشتری به تدریج و در طی تکرار مشخص می‌شوند و همچنین از آنجا که کلاس‌های همکار داخل هر مورد کاربرد یا نیاز هستند، بنابراین با تکامل و کامل شدن نیازها یا موارد کاربرد، به تبع کلاس‌های همکار هر مورد کاربرد نیز کامل می‌شود. بنابراین تشخیص تمام کلاس‌های برنامه، در همان ابتدای کار تقریبا غیرممکن است، در واقع در طول پروژه و با گذشت زمان تحلیل‌گر متوجه نیازها و به تبع کلاس‌های جدیدی می‌شود که در ابتدای کار نیاز به آن‌ها چندان محسوس نبوده است. بنابراین می‌توان گفت روند تشخیص نیازها یا موارد کاربرد و به تبع کلاس‌های همکار به عنوان مرتفع‌کننده نیازها یا موارد کاربرد، یک فرآیند تکرارشونده است و در طول فرآیند تولید نرم‌افزار کامل و کامل‌تر می‌شوند.

انواع روابط بین کلاس‌های همکار در مدل CRC

سه نوع رابطه مختلف بین کلاس‌های همکار در مدل CRC وجود دارد:

رابطه آگاه است از (has knowledge of)

دو انسانی که همدیگر را می‌شناسند و امکان گفتگو و تبادل پیام با هم دارند، رابطه انجمنی با هم دارند. در رابطه انجمنی یک شیء بطور ساده درباره شیء دیگر می‌داند به همان طریقی که یک فرد ممکن است فرد دیگری را بشناسد. یک برنامه کامپیوتری شیءگرا از اجتماع تعدادی کلاس ایجاد شده است، کلاس‌های همکار **بدون** رابطه جزء و کل و هم‌هدف در یک بخش مشترک از برنامه، کلاس‌های همکار **با** رابطه جزء و کل و هم‌هدف در یک بخش مشترک از برنامه و کلاس‌های غیرهمکار در دو بخش مختلف از برنامه.

توجه: رابطه‌ای که میان کلاس‌های همکار بدون رابطه جزء و کل جهت گفتگوی میان اشیاء آنها وجود دارد، رابطه انجمنی است.

توجه: کلاس‌های همکار **با** رابطه جزء و کل و هم‌هدف در یک بخش مشترک از برنامه، جلوتر شرح داده می‌شود.

کلاس‌های همکار بدون رابطه جزء و کل، جهت انجام وظایف خود از طریق مکانیزم پیام به گفتگو با یکدیگر می‌پردازند. در یک بیان ساده، هرگاه میان دو شیء بدون رابطه جزء و کل گفتگو باشد، کلاس‌های این دو شیء هم باهم همکار هستند و هم رابطه انجمنی میان آنها برقرار است.

هرگاه مابین دو کلاس را**بطه انجمنی** مطرح باشد، یعنی تبادل پیام مابین برخی متدهای اشیای دو کلاس صورت گیرد، آنگاه در این شرایط **رابطه آگاهی** خواهیم داشت. مانند ارسال پیام از شیء بازیکن به شیء توپ پس از انجام متد شوت زدن مربوط به شیء بازیکن، برای تغییر مختصات توپ توسط صدا زدن متد تغییر مختصات توپ مربوط به شیء توپ. یعنی شیء بازیکن، باید در هنگام شوت‌زدن، مختصات توپ را توسط صدازدن متد تغییر مختصات توپ، تغییر دهد.

توجه: نحوه مدل‌سازی رابطه آگاهی یا رابطه انجمنی جلوتر شرح داده خواهد شد.

رابطه بخشی است از (is part of)

هرگاه رابطه‌ای مابین یک واحد کل و تعدادی واحد جزء مطرح باشد، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشد، آنگاه در این شرایط رابطه بخشی خواهیم داشت. این رابطه خود بر دو نوع **تجمع** و **ترکیب** می‌باشد. نحوه مدل‌سازی رابطه تجمع و رابطه ترکیب جلوتر شرح داده خواهد شد.

رابطه وابسته است به (depends upon)

هر گاه مابین دو کلاس رابطه وابستگی مطرح باشد، رابطه وابستگی خواهیم داشت. در واقع هرگاه تغییرات مقادیر صفات یک شیء، روی مقادیر صفات شیء دیگری تأثیر بگذارد، این دو شیء به هم وابسته هستند. در واقع این رابطه زمانی رخ می‌دهد که مقادیر صفاتی از یک کلاس به

مقادیر صفاتی از یک کلاس دیگر وابسته باشد. برای مثال در برنامه کامپیوتری فوتبال، مختصات شیء سر بازیکن، دست بازیکن و پای بازیکن، همواره باید به مختصات شیء بدنه بازیکن وابسته باشد (مگر خشونت در بازی زیاد باشد!). در واقع مختصات سر بازیکن، دست بازیکن و پای بازیکن، همواره باید با تغییرات مختصات بدنه بازیکن تغییر کند. وگرنه در هنگام حرکت بازیکن، سر بازیکن، دست بازیکن و پای بازیکن از بدنه بازیکن جلو یا عقب می‌افتند!

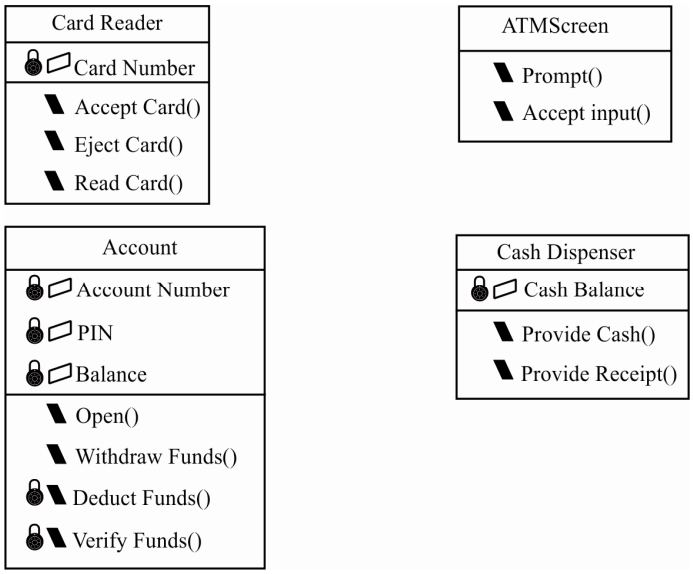
توجه: نحوه مدل‌سازی رابطه وابستگی جلوتر شرح داده خواهد شد.

مثال: عمل برداشت وجه در یک سیستم ATM یک نیاز یا مورد کاربرد (use case) است.

«برداشت وجه» یک نیاز یا مورد کاربرد است و این نیاز باید توسط نرم‌افزار برآورده شود. فرض کنید قرار است کمک کنید تا مراسم عروسی دوستتان به شکلی مطلوب و باشکوه برگزار گردد. بنابراین یک نفر مسئولیت تدارک شام را بر عهده می‌گیرد، شخص دیگری مسئولیت هماهنگی فضای مهمانی و شخص دیگری مسئولیت سفارش کیک را بر عهده می‌گیرد، در واقع این افراد با هم در یک کار گروهی همکاری و همیاری می‌کنند، تا این مراسم به عالی‌ترین شکل ممکن برگزار گردد. حال به وادی نرم‌افزار برگردید، یک نیاز به نام «برداشت وجه» برای مشتری وجود دارد.

در اینجا هم برای مرتفع کردن این نیاز باید کلاس‌هایی مسولیت‌هایی را بر عهده بگیرند و با یکدیگر همکاری کنند تا نیاز «برداشت وجه» بر طرف گردد.

شکل زیر کلاس‌های همکار داخل مورد کاربرد «برداشت وجه» را نمایش می‌دهد:



کلاس‌های همکار داخل مورد کاربرد برداشت وجه

توجه: به کلاس‌های همکار، **کلاس‌های مشارکت‌کننده** نیز گفته می‌شود.

توجه: پس از اتمام مدل‌سازی CRC، کارت‌های CRC با سناریوی اصلی هر use case یا مورد کاربرد تطابق داده می‌شود. تا مشخص شود که آیا تمام کلاس‌های مربوط به use case درست شناسایی شده‌اند یا خیر. که در صورت نیاز، می‌بایست تغییرات لازم بر روی کارت‌های CRC اعمال گردد.

در گام بعدی ارتباطات ایستای میان کلاس‌های همکار توسط نمودار کلاس مدل‌سازی می‌شود.

هـ) مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار

پس از شناسایی کلاس‌ها و کلاس‌های همکار برای هر یک از موارد کاربرد در مدل CRC، زمان مدل‌سازی نموداری و ساختاری توسط نمودار کلاس می‌رسد. در واقع یکی از اهداف کارت‌های CRC کشف زودهنگام و کم‌هزینه کاستی‌هاست، پاره‌کردن چند تکه کاغذ و کارت CRC به مراتب کم‌هزینه‌تر از سازماندهی مجدد ساختار نمودارهای کلاس است، بنابراین مدل CRC شروع کم‌هزینه مدل‌سازی نموداری و ساختاری توسط نمودار کلاس است.

توجه: Class Diagram یا نمودار کلاس جهت مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل یک case use مورد استفاده قرار می‌گیرد.

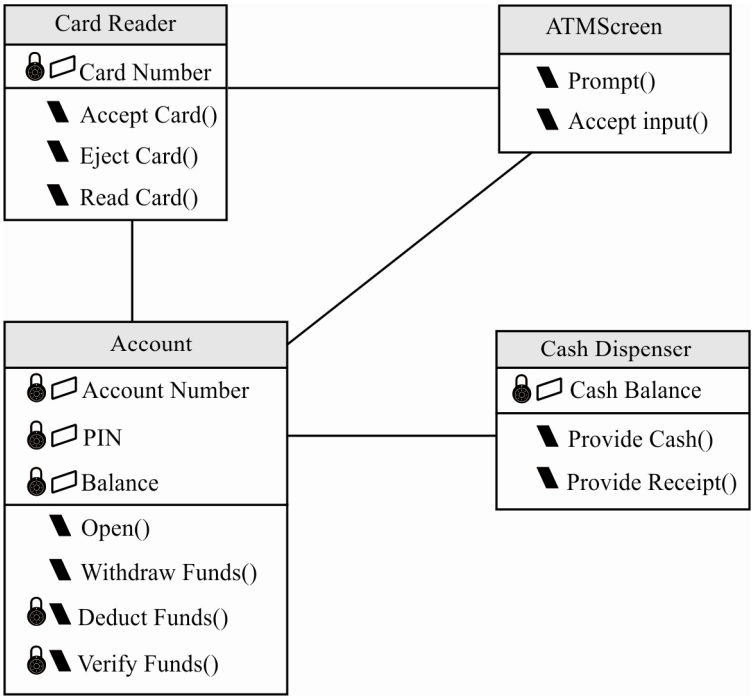
توجه: نمودار کلاس یک مدل‌سازی ساختاری محسوب می‌گردد.

توجه: در فعالیت مدل تحلیل، جزئیات مربوط به کلاس‌ها، شامل جزئیات دقیق صفات و متدها، مشخص نمی‌گردد، بیان این جزئیات تا بخش طراحی مولفه از مدل طراحی به تأخیر می‌افتد.

نمودار کلاس (Class Diagram)

نمودار کلاس به عنوان منبع اصلی تولید کد محسوب می‌گردد. هر کلاس دارای حداقل یک مسئولیت خواهد بود که در سلسله مراحل پالایش کلاس، این مسئولیت‌ها به مجموعه‌ای از صفات و متدها بدل می‌گردند. به نحوی که صفات و متدها بتوانند از عهده مسئولیت‌های کلاس برآیند.

مثال: شکل زیر مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل مورد کاربرد «برداشت وجه» را نمایش می‌دهد:



مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل مورد کاربرد برداشت وجه

توجه: نمودار کلاس فوق، مدل‌سازی ارتباطات ایستای مابین کلاس‌هایی را نشان می‌دهد که مورد کاربرد «برداشت وجه» را به انجام می‌رسانند. این کار با چهار کلاس انجام شده است.

- کارت‌خوان (Card Reader)
- حساب (Account)
- صفحه نمایش ATM (ATM Screen)
- صندوق (Cash Dispenser)

اجزای نمودار کلاس

نمودار کلاس، برای نمایش کلاس‌ها، صفات کلاس‌ها، متدهای کلاس‌ها و نیز روابط مابین کلاس‌ها مورد استفاده قرار می‌گیرد. یک نمودار کلاس از دو جزء اصلی تشکیل می‌شود که عبارتند از: کلاس‌ها و روابط. در ادامه به تشریح جزئیات هر یک از دو جزء مذکور می‌پردازیم.

کلاس‌ها

در یک نمودار کلاس، هر کلاس با مستطیلی نشان داده می‌شود که شامل سه بخش زیر است:

بخش اول: نام کلاس

نام یک کلاس، عنوان منحصر به فردی است که به آن نسبت داده می‌شود و تا حدودی نوع و عملکرد کلاس را توصیف می‌کند. بخش نام کلاس شامل نام کلاس و در صورت نیاز ذکر Stereotype است، برای معرفی بیشتر ماهیت کلاس.

بخش دوم: صفات کلاس (Attributes)

صفات یک کلاس در برگیرنده مشخصات ساختاری کلاس است. به بیان دیگر یک صفت قطعه‌ای از اطلاعاتی است که با یک کلاس مرتبط می‌باشد.

برای تعریف یک صفت لازم است نام و نوع آن مشخص شود. قابلیت رویت یک صفت نیز باید در زمان تعریف آن مشخص گردد. به عبارت دیگر، باید مشخص شود که اشیای کدام کلاس‌ها مجاز به رویت صفت مذکور هستند.

UML چهار نوع قابلیت رویت برای یک صفت متصور است که عبارتند از:

عمومی (Public) با نماد «+»: اگر قابلیت رویت یک صفت از نوع «عمومی» باشد، آنگاه تمام اشیاء(حتی اشیاء کلاس‌های متفاوت) حق دسترسی به صفت مذکور را دارند.

خصوصی (Private) با نماد «-»: اگر قابلیت رویت یک صفت از نوع «خصوصی» باشد، آنگاه حق دسترسی، فقط به صاحب آن صفت، یعنی خود شیء، محدود بوده و اشیای دیگر حق دسترسی به آن را ندارند.

محافظت شده (Protected) با نماد «#»: اگر قابلیت رویت یک صفت از نوع «محافظت شده» باشد، آنگاه علاوه بر شیء صاحب آن صفت، سایر اشیای هم کلاس با شیء مذکور به همراه کلیه اشیای کلاس‌های فرزند نیز اجازه دسترسی به آن صفت را دارند.

بسته (Package) با نماد «~»: اگر قابلیت رویت یک صفت از نوع «بسته» باشد، آنگاه کلیه اشیای کلاس‌های هم بسته با کلاس شیء صاحب آن صفت، اجازه دسترسی به آن ویژگی را دارند.

نکته قابل توجه دیگر این است که تعدادی از صفت‌ها و متدها، علاوه بر نمادهای فوق می‌توانند از قفل‌های کوچکی در سمت چپشان استفاده کنند. نماد قفل، یک صفت یا متد خصوصی (Private) را نشان می‌دهد. صفات و متدهای خصوصی فقط می‌توانند ار طریق شیء‌ای که شامل آنها است قابل دستیابی باشند.

مثال: کلاس حساب (Account) شامل سه صفت زیر است:

- شماره حساب (Account Number)
- پین (PIN)
- موجودی (Balance)

بخش سوم: متدهای کلاس (Operations)

متدهای یک کلاس نیز شامل قابلیت‌های عملکردی کلاس مذکور است. یک کلاس می‌تواند شامل تعدادی متد باشد، که اشیای آن کلاس می‌توانند آن‌ها را انجام دهند. هر متد مانند یک صفت با یک نام، مشخص می‌شود. علاوه بر آن، یک متد می‌تواند دارای یک سری آرگومان ورودی باشد. هر متد می‌تواند دارای خروجی نیز باشد که در این صورت لازم است نوع آن مشخص گردد. علاوه بر این، متدها، همانند صفات دارای قابلیت رویت هستند. قوانین و تعاریف انواع قابلیت رویت در صفات برای متدها نیز صادق است.

مثال: کلاس حساب (Account) شامل چهار متد زیر است:

- باز کردن (Open)
- برداشت وجه (Withdraw Funds)
- کسر از موجودی (Deduct Funds)
- تأیید موجودی (Verify Funds)

برای مدل‌سازی یک کلاس لازم است اجزای آن در یک قالب نمادین قرار گیرند. نماد یک کلاس از سه بخش مستطیل شکل تشکیل می‌شود. هر یک از این بخش‌ها به یکی از اجزای سه گانه کلاس اختصاص می‌یابد. همانطور که در شکل زیر مشاهده می‌نمایید، نام کلاس در اولین بخش مستطیلی شکل نماد کلاس قرار می‌گیرد. Stereotype نیز، در صورت نیاز در همین بخش و در بالای نام کلاس نشان داده می‌شود. مستطیل بخش میانی نماد کلاس، جهت درج و تعریف صفات کلاس، مورد استفاده قرار می‌گیرد. ترتیب قرار گرفتن صفات مهم نیست. در بخش تحتانی نماد کلاس، متدهای کلاس تعریف می‌شوند. ترتیب قرار گرفتن متدها نیز مهم نیست. در فرآیند تعریف یک کلاس، ذکر بخش نام، ضروری است، اما دو بخش دیگر در صورت لزوم و وجود اطلاعات مربوطه ذکر می‌شوند.

Account
- AccountNumber : long - PIN : int - Balance : long
+Open() + WithdrawFunds() + DeductFunds() + VerifyFunds()

روابط نمودار کلاس

همانطور که پیش از این نیز بیان شد، یکی از اجزای اصلی نمودار کلاس، روابط موجود بین کلاس‌ها است. روابط بین کلاس‌ها، مانند روابط بین انسان‌هاست. به عنوان مثال، اگر من و شما بخواهیم با یکدیگر کار کنیم لازم است با یکدیگر ارتباط برقرار نماییم. این ارتباط ممکن است از طرق مختلفی نظیر تلفن و پست الکترونیک انجام پذیرد که هرکدام در شرایط خاصی کاربرد دارد. بنابراین، در هر ارتباط، لازم است نوع و شرایط ارتباط به درستی مشخص گردد. از این رو UML، نمادهای خاصی را برای مشخص کردن نوع و نحوه این روابط ارائه می‌دهد. در این بخش با انواع روابط مابین کلاس‌ها، تعاریف و نمادهای آن آشنا می‌شوید.

مدل‌سازی انواع روابط بین کلاس‌های همکار در نمودار کلاس

۱- مدل‌سازی رابطه انجمنی (Association)

دو انسانی که همدیگر را می‌شناسند و امکان گفتگو و تبادل پیام با هم دارند، رابطه انجمنی با هم دارند. در رابطه انجمنی یک شیء بطور ساده درباره شیء دیگر می‌داند به همان طریقی که یک فرد ممکن است فرد دیگری را بشناسد. یک برنامه کامپیوتری شیءگرا از اجتماع تعدادی کلاس ایجاد شده است، کلاس‌های همکار بدون رابطه جزء و کل و هم‌هدف در یک بخش مشترک از برنامه، کلاس‌های همکار با رابطه جزء و کل و هم‌هدف در یک بخش مشترک از برنامه (جلوتر شرح داده می شود) و کلاس‌های غیرهمکار در دو بخش مختلف از برنامه، رابطه‌ای که میان کلاس‌های همکار بدون رابطه جزء و کل وجود دارد، رابطه انجمنی است.

کلاس‌های همکار بدون رابطه جزء و کل، جهت انجام وظایف خود از طریق مکانیزم پیام به گفتگو با یکدیگر می‌پردازند. در یک بیان ساده، هرگاه میان دو شیء بدون رابطه جزء و کل گفتگو باشد، کلاس‌های این دو شیء هم باهم همکار هستند و هم رابطه انجمنی میان آنها برقرار است.

هرگاه مابین دو کلاس را**بطه انجمنی** مطرح باشد، یعنی تبادل پیام مابین برخی متدهای دو کلاس صورت گیرد، آنگاه در این شرایط **رابطه آگاهی** خواهیم داشت. مانند ارسال پیام از شیء بازیکن به شیء توپ پس از انجام متد شوت زدن مربوط به شیء بازیکن، برای تغییر مختصات توپ توسط صدا زدن متد تغییر مختصات توپ مربوط به شیء توپ. یعنی شیء بازیکن، باید در هنگام شوت‌زدن، مختصات توپ را توسط صدازدن متد تغییر مختصات توپ، تغییر دهد. به عنوان مثالی دیگر، کلاس Account به کلاس ATM Screen توسط یک خط ممتد وصل شده است زیرا هر دو مستقیماً باهم **رابطه انجمنی و تبادل پیام** دارند. Card Reader به Cash Dispenser وصل نشده است زیرا این دو با هم ارتباطی ندارند. اشیاء کلاس‌های همکار بدون رابطه جزء و کل از طریق صدازدن متدهای عمومی یکدیگر اقدام به گفتگو و تبادل پیام می‌کنند و این یعنی شناخت و آگاهی از یکدیگر.

توجه: رابطه انجمنی به عنوان رابطه «has knowledge of» نیز شناخته می‌شود.

هر رابطه انجمنی از سه قسمت اصلی تشکیل می‌شود که عبارتند از:

۱) کلاس‌های شرکت‌کننده در رابطه

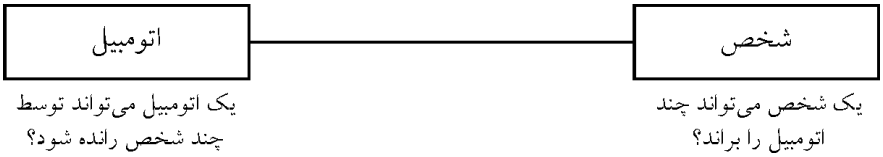
۲) خط ممتد رابطه که بین دو کلاس ترسیم می‌شود.

۳) نام رابطه که در وسط خط رابطه نوشته می‌شود و بیان‌کننده هدف رابطه است.

در مورد روابط انجمنی، بین کلاس‌ها، سوالات مهمی مطرح می‌شود. برای مثال، در مورد رابطه انجمنی شخص و اتومبیل، سوالات زیر مطرح است:

«یک شخص می‌تواند چند اتومبیل را براند؟»، «پاسخ: *..۱»

«یک اتومبیل می‌تواند توسط چند شخص رانده شود؟»، «پاسخ: *..۱»



UML برای پاسخگویی به چنین سوالاتی، مشخصه‌ای با نام **تعدد** (Multiplicity) را برای هر یک از کلاس‌های طرفین رابطه ارائه نموده است. تعدد اصطلاحی است که تعداد اشیای شرکت‌کننده در رابطه انجمنی را مشخص می‌کند. اعدادی که در پاسخ به این دو سوال داده می‌شود، مبین تعدد طرفین است. شیوه‌های مختلفی جهت تعیین تعدد وجود دارد، اما متداول‌ترین شیوه، بیان محدوده تعداد اشیای شرکت‌کننده در هر طرف رابطه است که به صورت «حداقل .. حداکثر» نشان داده می‌شود. در تعیین تعدد نمی‌توان از اعداد اعشاری استفاده نمود، بلکه فقط از اعداد صحیح مثبت استفاده می‌شود. در مواقعی که حداکثر تعداد اشیای رابطه مشخص نباشد و یا اینکه هیچ حد بالایی برای آن وجود نداشته باشد، از نماد «*» برای این منظور استفاده می‌شود. به عبارت دیگر، نماد «*» بیانگر عدم وجود یک حد بالا و پایین مشخص است و لذا در این حالت، تعداد صفر یا بیشتر شیء می‌توانند در رابطه مشارکت نمایند. در صورتی که حد بالا و پایین یکی باشد، می‌توان به اختصار فقط یکی از حدود را نمایش داد. برای مثال حد «۱..۱» را می‌توان به اختصار به صورت «۱» نمایش داد.

با توجه به مطالبی که ارائه شد، می‌توان انواع تعدد را در چهار دسته زیر طبقه‌بندی نمود:

۱- «یک به یک»

۲- «یک به چند»

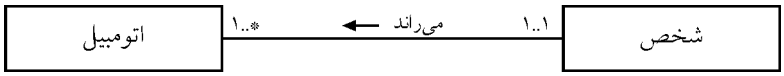
۳- «چند به یک»

۴- «چند به چند»

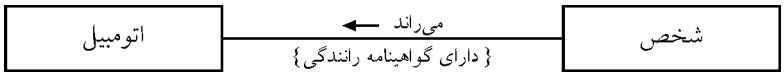
توجه: رابطه «یک به چند» معادل رابطه «چند به یک» است، با این تفاوت که جهت رابطه در

آنها عکس یکدیگر است.

مثال: تعدد رابطه انجمنی بین شخص و اتومبیل به صورت زیر است:

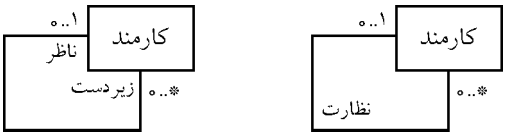


یک رابطه انجمنی ساده می‌تواند دارای شروط و محدودیت‌های خاص خود باشد تا صحت عملکرد رابطه تضمین گردد. برای روشن شدن این مطلب، در شکل زیر، رابطه شخص با اتومبیل نشان داده شده است. سوالی که اینجا مطرح است این است که آیا هر شخصی مجاز به راندن اتومبیل است؟ طبق قوانین راهنمایی و رانندگی فقط افراد دارای گواهینامه می‌توانند رانندگی کنند. بنابراین، محدودیتی باید در رابطه بین شخص و اتومبیل اعمال گردد. برای نمایش محدودیت‌ها نماد «{}» مورد استفاده قرار می‌گیرد. در شکل زیر محدودیت «{دارای گواهینامه رانندگی}» در نظر گرفته شده است. لازم به ذکر است که این محدودیت باید در طرف شخص قرار گیرد تا قبل از آنکه ارتباطی برقرار شود، محدودیت‌ها بررسی شده و نسبت به برقراری ارتباط تصمیم گیری شود.

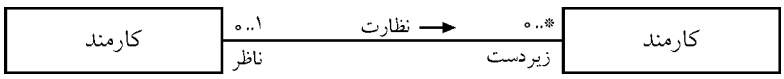


مدل‌سازی رابطه خودانجمنی (Self-Association)

در UML، یک رابطه‌ی خودانجمنی، رابطه‌ای است که یک کلاس با خودش دارد. لذا اگر اشیای یک کلاس با یکدیگر در ارتباط باشند، ارتباط مذکور از نوع خودانجمنی است. به عبارت دیگر، اگر ابتدا و انتهای یک رابطه انجمنی به یک کلاس اشاره داشته باشد، در این صورت رابطه‌ی مذکور از نوع خودانجمنی خواهد بود. در شکل زیر، دو مثال برای رابطه بازتابی ارائه شده است. توجه داشته باشید که در صورت مشخص کردن نقش دو طرف رابطه، ذکر نام رابطه، ضروری نیست.



برای سادگی در نحوه خواندن رابطه خودانجمنی توصیه می‌کنیم، رابطه را به شکل خطی ایجاد کنید، سپس رابطه را بخوانید:



شکل فوق گویای رابطه خودانجمنی (Self Association) است. به این معنی که یک کارمند هیچ زیردستی ندارد (چون مدیر نیست) یا چندین زیردست دارد (چون مدیر است) همچنین یک کارمند هیچ ناظری ندارد (چون مدیرکل است) یا یک ناظر دارد (چون مدیر کل نیست).

به کمک رابطه انجمنی ساده که در این قسمت به معرفی آن پرداختیم، امکان مدل‌سازی مشخصات خاص برخی روابط وجود ندارد. برای‌مثال رابطه بین «بازیکن» و «تیم» با رابطه بین «شخص» و «اتومبیل» تفاوت‌هایی دارد، در رابطه بین «بازیکن» و «تیم» مابین بازیکن و تیم رابطه جزء و کل مطرح است، به عبارت دیگر مابین کلاس‌های همکار یعنی بازیکن و تیم رابطه جزء و کل برقرار است. یعنی بازیکن بخشی یا جزئی از تیم محسوب می‌شود. در حالی در رابطه بین «شخص» و «اتومبیل» مابین شخص و اتومبیل رابطه جزء و کل مطرح نیست، به عبارت دیگر مابین کلاس‌های همکار یعنی شخص و اتومبیل رابطه جزء و کل برقرار نیست. یعنی شخص بخشی از اتومبیل و یا اتومبیل بخشی از شخص نیست، یعنی همانطور که گفتیم مابین شخص و اتومبیل رابطه انجمنی ساده برقرار است. بنابر مطلب بیان شده امکان نمایش رابطه جزء و کل مانند رابطه بین «بازیکن» و «تیم» توسط رابطه انجمنی ساده وجود ندارد.

برای پوشش چنین مشخصه‌ای از یک رابطه انجمنی، UML دو نوع رابطه به نام‌های تجمع و ترکیب ارائه نموده است. رابطه‌ی تجمع، نوع خاصی از رابطه انجمنی است که علاوه بر دارا بودن تمام ویژگی‌های رابطه انجمنی، یک سری ویژگی‌های مخصوص به خود نیز دارد. رابطه ترکیب نیز، نوع خاصی از رابطه تجمع است، لذا علاوه بر دارا بودن تمام ویژگی‌های رابطه تجمع، یک سری ویژگی‌های خاص خود را نیز دارد. در ادامه به معرفی رابطه انجمنی پیشرفته یا رابطه بخشی می‌پردازیم.

۲- مدل‌سازی رابطه انجمنی پیشرفته یا رابطه بخشی

همانطور که پیش‌تر از نیز گفتیم، هرگاه رابطه‌ای مابین یک واحد کل و تعدادی واحد جزء مطرح باشد، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشد، آنگاه در این شرایط رابطه انجمنی پیشرفته یا رابطه بخشی خواهیم داشت. رابطه انجمنی پیشرفته یا رابطه بخشی به دو رابطه تجمع و ترکیب تقسیم می‌گردد. در ادامه به تشریح دو رابطه مذکور می‌پردازیم.

الف) مدل‌سازی رابطه تجمع (Aggregation)

همانطور که پیش‌تر از این نیز گفتیم، رابطه تجمع رابطه‌ای مابین یک واحد کل و تعدادی واحد جزء است، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشد. برای مثال یک شیء کل «تیم» را در نظر بگیرید، که خود از چندین شیء جزء دیگر مانند اشیاء بازیکنان تشکیل شده است. به عنوان مثالی دیگر، یک شیء کل «اتومبیل» را در نظر بگیرید، که خود از چندین شیء جزء دیگر مانند شیء فرمان، موتور، چرخ و غیره تشکیل شده است. به چنین روابطی تجمع گفته می‌شود. در رابطه تجمع، به طرف جزء، عضو (Member) و به طرف کل (Whole)،

تجمع گفته می‌شود.

در رابطه **انجمنی‌ساده**، مابین طرفین همکار رابطه، رابطه جزء و کل مطرح نیست، در حالی که در رابطه **تجمع**، مابین طرفین همکار رابطه، رابطه جزء و کل مطرح است و از **تجمع** و سازماندهی اجزاء، یک موجودیت کامل‌تر ساخته می‌شود. برای مثال، برای ساخته شدن یک شیء کل «تیم» لازم است اشیاء جزء مختلفی با یکدیگر **مجتمع** شوند. یا برای مثالی دیگر، برای ساخته شدن یک شیء کل «اتومبیل» لازم است اشیاء جزء مختلفی با یکدیگر **مجتمع** شوند.

توجه: رابطه تجمع به عنوان رابطه «is part of» نیز شناخته می‌شود.

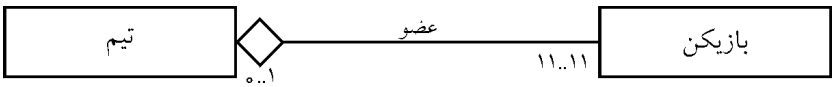
مراحل لازم جهت مدل‌سازی یک رابطه تجمع عبارتند از:

۱- ترسیم خط ممتد از سمت جزء به سمت کل

۲- ترسیم لوزی توخالی در انتهایی از خط رابطه که کلاس کل قرار دارد

۳- تعیین تعدد، نقش و محدودیت‌های طرفین رابط در صورت نیاز

شکل زیر نمونه‌ای از یک رابطه تجمع میان شیء کل «تیم» و اشیاء جزء «بازیکن» را نشان می‌دهد.



توجه: یک بازیکن، ممکن است عضو یک تیم باشد و یا نباشد، بدین معنی که یک بازیکن می‌تواند مستقل از یک تیم وجود داشته باشد، بنابراین از نماد ۱ .. ۰ استفاده شده است.

در رابطه تجمع، حیات شیء جزء به حیات شیء کل وابسته نیست. به بیان دیگر حیات شیء جزء و شیء کل به هم تقدم و تاخر دارند. یعنی ممکن است شیء جزء موجود باشد، بدون اینکه شیء کل ایجاد گردد و موجود باشد. در رابطه تجمع اگر شیء کل از بین برود، اشیاء جزء، همچنان به حیات خود ادامه می‌دهند. زیرا از ابتدای کار، اشیا جزء، توسط برنامه کامپیوتری ایجاد شده‌اند و فقط با پایان برنامه کامپیوتری اشیاء جزء از بین می‌روند و نه به هنگام از بین رفتن شیء کل. در بازی کامپوتری فوتبال، اشیا جزء یا همه بازیکن‌های تیم ملی توسط برنامه کامپیوتری ایجاد می‌شوند، اما شما بر حسب سلیقه ۱۱ نفر از آن‌ها را انتخاب می‌کنید و تیم خود یا شیء کل را می‌سازید. اگر هم تیم را منحل کنید، اشیاء بازیکن که توسط برنامه کامپیوتری از همان ابتدای اجرای برنامه ایجاد شده‌اند، همچنان هستند، و کافی است در یک دسته‌بندی دیگر تیم جدیدی را ایجاد نمایید. یا مانند بازی کامپیوتری اتومبیلرانی، اشیا جزء یا همه قطعات اتومبیل مانند انواع موتور و انوع فرمان توسط برنامه کامپیوتری ایجاد می‌شوند، اما شما بر حسب سلیقه قطعات مختلف ساخت اتومبیل را انتخاب می‌کنید و اتومبیل خود یا شیء کل را می‌سازید. اگر هم اتومبیل را از بین ببرید، اشیاء و قطعات اتومبیل همچون موتور و فرمان که توسط برنامه کامپیوتری از همان ابتدای اجرای برنامه ایجاد شده‌اند، همچنان هستند، و کافی است در یک دسته‌بندی دیگر

اتومبیل جدیدی را ایجاد نمایید.

ب) مدل‌سازی رابطه ترکیب (Composition)

همانطور که پیش‌تر از این نیز گفتیم، رابطه ترکیب رابطه‌ای مابین یک واحد کل و تعدادی واحد جزء است، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشد به نحوی که شیء کل و جزء با هم ایجاد شوند و با هم از بین بروند. برای مثال یک شیء کل «بازیکن» را در نظر بگیرید، که خود از چندین شیء جزء دیگر مانند سر، دست، پا و بدنه تشکیل شده است. به نحوی که شیء کل و جزء با هم ایجاد شوند و با هم از بین بروند. به عنوان مثالی دیگر، یک شیء کل «موتور اتومبیل» را در نظر بگیرید، که خود از چندین شیء جزء دیگر تشکیل شده است. به نحوی که شیء کل و جزء با هم ایجاد شوند و با هم از بین بروند. به چنین روابطی ترکیب گفته می‌شود. در رابطه ترکیب، به طرف جزء، عضو (Member) و به طرف کل (Whole)، ترکیب گفته می‌شود.

در رابطه **انجمنی‌ساده**، مابین طرفین همکار رابطه، رابطه جزء و کل مطرح نیست، در حالی که در رابطه **ترکیب**، مابین طرفین همکار رابطه، رابطه جزء و کل مطرح است و از **ترکیب** و سازماندهی اجزاء، یک موجودیت کامل‌تر ساخته می‌شود. برای مثال، برای ساخته شدن یک شیء کل «بازیکن» لازم است اشیاء جزء مختلفی با یکدیگر **ترکیب** شوند. یا برای مثالی دیگر، برای ساخته شدن یک شیء کل «موتور اتومبیل» لازم است اشیاء جزء مختلفی با یکدیگر **ترکیب** شوند.

توجه: رابطه تجمع به عنوان رابطه «is part of» نیز شناخته می‌شود.

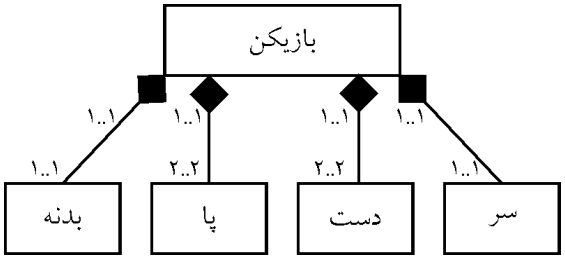
مراحل لازم جهت مدل‌سازی یک رابطه ترکیب عبارتند از:

۱- ترسیم خط ممتد از سمت جزء به سمت کل

۲- ترسیم لوزی توپر در انتهایی از خط رابطه که کلاس کل قرار دارد

۳- تعیین تعدد، نقش و محدودیت‌های طرفین رابط در صورت نیاز

شکل زیر نمونه‌ای از یک رابطه ترکیب میان شیء کل «بازیکن» و اشیاء جزء «سر، دست، پا و بدنه» را نشان می‌دهد.



توجه: حیات اجزای «سر، دست، پا و بدنه» کاملا به بازیکن، وابسته است، بدین معنی که اجزای «سر، دست، پا و بدنه» هرگز نمی‌توانند مستقل از یک بازیکن وجود داشته باشد، بنابراین از

نماد ۱..۱ در سمت بازیکن استفاده شده است.

در رابطه ترکیب، حیات شیء جزء به حیات شیء کل وابسته است. به بیان دیگر حیات شیء جزء و شیء کل به هم تقدم و تاخر ندارند. یعنی امکان ندارد شیء جزء موجود باشد، بدون اینکه شیء کل ایجاد گردد و موجود باشد. در رابطه ترکیب اگر شیء کل از بین برود، اشیاء جزء نیز به طور همزمان با شیء کل از بین می‌روند. زیرا از ابتدای کار، شیء کل، به طور همزمان از ایجاد و کنار هم قرار دادن اشیاء جزء توسط برنامه کامپیوتری ایجاد شده است و با پایان یافتن فعالیت شیء کل، شیء کل و جزء باهم و همزمان از بین می‌روند. در بازی کامپوتری فوتبال، شیء کل بازیکن از اشیاء جزء سر، دست، پا و بدنه تشکیل شده است. و پس از پایان فعالیت، شیء کل و جزء به طور همزمان باهم از بین می‌روند، در یک بیان ساده اشیاء کل و جزء در هنگام نیاز، باهم ایجاد می‌گردند و در صورت عدم نیاز، اشیاء کل و جزء با هم از بین می‌روند. در رابطه ترکیب، اشیاء جزء در شیء کل معنا و مفهوم دارند و به طور همزمان برای ساخت شیء کل ایجاد می‌گردند. اشیاء جزء در ترکیب به تنهایی معنا و مفهومی ندارند و برای خدمت به شیء کل ایجاد می‌گردند، و پس از، از بین رفتن شیء کل، اشیاء جزء هم به خدمت خود همزمان با پایان حیات شیء کل، پایان می‌دهند. اشیاء جزء در رابطه ترکیب فقط برای خدمت به شیء کل ایجاد می‌گردند و پس از پایان یافتن شیء کل، معنا و وظیفه دیگری ندارند و حیاتشان به پایان می‌رسد. برای مثال توسط برنامه کامپیوتری فوتبال، اشیاء جزء سر، دست، پا و بدنه به طور همزمان با ساخت بازیکن ایجاد می‌گردند و با پایان یافتن فعالیت بازیکن، اشیاء جزء سر، دست، پا و بدنه نیز به پایان فعالیت خود می‌رسند، در یک بیان ساده، در ابتدای اجرای بازی کامپیوتری فوتبال، شیء کل بازیکن از اشیاء جزء سر، دست، پا و بدنه به طور همزمان ایجاد می‌شود و تا پایان اجرای برنامه کامپیوتری هست که این ترکیب است، البته تک تک بازیکنان به نوبت به همین ترتیب توسط برنامه کامپیوتری ایجاد می‌شوند، اما حالا همه بازیکنان تیم ملی هستند، حال بر حسب سلیقه ۱۱ نفر از آن‌ها را انتخاب می‌کنید و تیم خود یا شیء کل را می‌سازید، که این تجمع است. که اگر تیم هم منحل شود، اشیاء بازیکن که توسط برنامه کامپیوتری ایجاد شده‌اند، همچنان هستند، و کافی است در یک دسته‌بندی دیگر تیم جدیدی ایجاد شود. برای مثالی دیگر توسط برنامه کامپیوتری اتومبیلرانی اشیاء جزء مربوطه به طور همزمان برای ساخت قطعات موتور، فرمان، چرخ و غیره ایجاد می‌گردند و با پایان یافتن فعالیت قطعات موتور، فرمان، چرخ و غیره، اشیاء جزء مربوطه نیز به پایان فعالیت خود می‌رسند. در یک بیان ساده، در ابتدای اجرای بازی کامپیوتری اتومبیلرانی، شیء کل موتور، فرمان، چرخ و غیره از اشیاء جزء مربوطه به طور همزمان ایجاد می‌شوند و تا پایان اجرای برنامه اتومبیلرانی هستند که این ترکیب است، البته تک تک قطعات موتور، فرمان، چرخ و غیره به نوبت به همین ترتیب توسط برنامه کامپیوتری ایجاد می‌شوند، اما حالا همه قطعات

ساخت اتومبیل همچون موتور، فرمان، چرخ و غیره هستند، حال بر حسب سلیقه قطعات را انتخاب می‌کنید و اتومبیل خود یا شیء کل را می‌سازید. که این تجمع است. که اگر اتومبیل هم از بین برود، اشیاء و قطعات اتومبیل همچون موتور، فرمان، چرخ و غیره که توسط برنامه کامپیوتری از همان ابتدای اجرای برنامه ایجاد شده‌اند، همچنان هستند، و کافی است در یک دسته‌بندی دیگر اتومبیل جدیدی ایجاد شود. برای مثالی دیگر، یک پنجره سیستم عامل ویندوز را در نظر بگیرید، یک پنجره از چندین شیء جزء تشکیل شده است، برای مثال اشیاء جزء دکمه کمینه، بیشینه، بستن پنجره و منوها، هنگامی که یک شیء کل پنجره بنا به درخواست اجرا می‌گردد، همزمان با آن تمام دکمه‌ها و منوها ایجاد می‌شوند. و با پایان دادن به فعالیت پنجره، شیء کل پنجره و اشیاء جزء به همراه هم و به طور همزمان به فعالیت خود پایان می‌دهند. در وادی زندگی نیز انسان کل با اشیاء جزء خود به طور همزمان متولد می‌شود و سپس در انتهای آن به حیات مادی خود به همراه اشیاء جزء خود به طور همزمان پایان می‌دهد، که این **ترکیب** است. اما این انسان در طول حیات خود در گروه‌ها، تیم‌ها، و کلاس‌های مختلفی شرکت می‌کند، که این **تجمع** است.

۳- مدل‌سازی رابطه وابستگی (Dependency)

همانطور که پیش‌تر از این در مورد رابطه وابستگی گفتیم، هرگاه مابین دو کلاس رابطه وابستگی مطرح باشد، رابطه وابستگی خواهیم داشت. در واقع هرگاه تغییرات مقادیر صفات یک شیء، روی مقادیر صفات شیء دیگری تأثیر بگذارد، این دو شیء به هم وابسته هستند. در واقع این رابطه زمانی رخ می‌دهد که مقادیر صفاتی از یک کلاس به مقادیر صفاتی از یک کلاس دیگر وابسته باشد. برای مثال در برنامه کامپیوتری فوتبال، مختصات شیء سربازیکن، دست بازیکن و پای بازیکن، همواره باید به مختصات شیء بدنه بازیکن وابسته باشد (مگر خشونت در بازی زیاد باشد!). در واقع مختصات سربازیکن، دست بازیکن و پای بازیکن، همواره باید با تغییرات مختصات بدنه بازیکن تغییر کند. وگرنه در هنگام حرکت بازیکن، سر بازیکن، دست بازیکن و پای بازیکن از بدنه بازیکن جلو یا عقب می‌افتند!

توجه: رابطه وابستگی به عنوان رابطه «depend upon» نیز شناخته می‌شود.

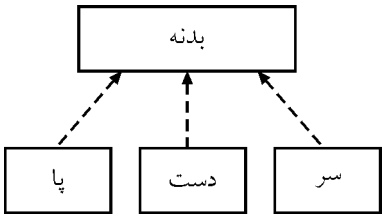
مراحل لازم جهت مدل‌سازی یک رابطه وابستگی عبارتند از:

۱- ترسیم یک خط جهت دار مقطع از سمت کلاس وابسته به سمت کلاس غیروابسته

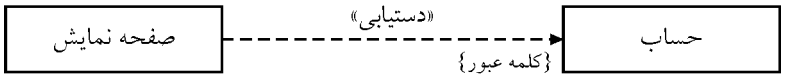
۲- بیان نوع وابستگی بر روی خط جهت‌دار مقطع (Stereotype یا کلیشه)

توجه: در UML کلیشه‌ها توسط نماد <<Stereotype>> مشخص می‌شوند.

شکل زیر نمونه‌ای از یک رابطه وابستگی میان «بدنه بازیکن» و «سر بازیکن، دست بازیکن، پای بازیکن»، را نشان می‌دهد.



توجه: رابطه سرویس گیرنده (client) و سرویس دهنده (server) میان دو کلاس، نوع دیگری از وابستگی میان دو کلاس است. در این گونه موارد کلاس سرویس گیرنده به نحوی به کلاس سرویس‌دهنده وابسته است و یک رابطه وابستگی برقرار می‌شود. در این نوع رابطه کلاس سرویس گیرنده برای انجام کارهای خود به داده‌های موجود در کلاس سرویس‌دهنده وابسته است. مانند وابستگی کلاس صفحه نمایش، به کلاس حساب، جهت مقدار دهی صفحه نمایش برای یک مشتری خاص در سیستم ATM. برای نمایش محدودیت‌های دسترسی نماد «{}» مورد استفاده قرار می‌گیرد. در شکل زیر محدودیت «{کلمه عبور}» در نظر گرفته شده است.



مدل‌سازی رابطه وراثت بین کلاس‌ها در نمودار کلاس

به طورکلی هرگاه یک کلاس (ب)، **از نوع** یک کلاس (الف) باشد، گوییم بین دو کلاس مذکور رابطه **وراثت** برقرار است. در این صورت کلاس (ب) فرزند و کلاس (الف) پدر یا والد نامیده می‌شود. به عبارت دیگر همانطور که پیش از این نیز گفتیم، وراثت فرآیندی است که به وسیله آن یک کلاس (فرزند) می‌تواند صفات و متدهای کلاس دیگری (پدر) را کسب کند. به عبارت کلی‌تر یک کلاس فرزند ضمن به ارث بردن مجموعه‌ای از صفات و متدهای عمومی کلاس پدر، می‌تواند ویژگی‌های خاص و مختص خود را نیز به آنها اضافه کند. در رابطه وراثت هر تغییر در کلاس پدر بر کلاس فرزند نیز اثر می‌گذارد اما عکس این مطلب برقرار نیست. برای مثال، بین سیب و میوه، رابطه ارث‌بری برقرار است، سیب (فرزند) **یک نوع** میوه (پدر) است. بنابراین یک سیب دارای تمام ویژگی‌های یک میوه است. بنابراین از دیدگاه وراثت، سیب **نوعی** میوه است، زیرا تمام ویژگی‌های میوه را به ارث می‌برد و البته دارای یک سری ویژگی‌های مختص به خودش نیز هست. از دیدگاهی دیگر، میوه تعمیم یافته سیب است، زیرا ویژگی‌های عمومی سیب که در میوه‌های دیگر نیز مشترک است در میوه گردآوری شده است. ابرکلاس، همان کلاس پدر است که ویژگی‌ها و عملیات خود را در اختیار کلاس‌های دیگر (فرزندان) قرار می‌دهد. برای مثال، «سیب» یک ابرکلاس است، زیرا ویژگی‌های خود را در اختیار کلاس «سیب قرمز» قرار می‌دهد. زیرکلاس نیز همان کلاس فرزند است که برخی از صفات و عملیاتش متعلق به یک ابرکلاس است. کلاس «سیب قرمز» نمونه‌ای از یک زیرکلاس است.

دو رویکرد برای ایجاد سلسله مراتب وراثت وجود دارد:

۱- رویکرد «بالا به پایین» یا تخصیص (Specialization) که در آن، ابتدا کلاس‌های پدر، تعریف و سپس کلاس‌های فرزند ایجاد می‌شوند.

۲- رویکرد «پایین به بالا» یا تعمیم (Generalization) که در آن، از طریق تعمیم کلاس‌های فرزند به کلاس‌های پدر می‌رسیم.

توجه: رابطه وراثت به عنوان رابطه «is a» یا «kind of» یا «type of» نیز شناخته می‌شود. زیرا در این رابطه می‌گوییم: «A red apple is an apple» و یا «An apple is a fruit».

مراحل لازم جهت مدل‌سازی یک رابطه ارث‌بری عبارتند از:

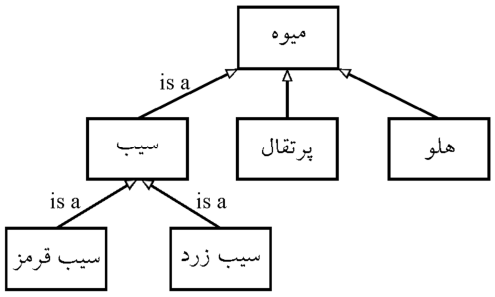
۱- ترسیم یک خط ممتد بین کلاس فرزند و کلاس پدر

۲- درج یک مثلث تو خالی در انتهایی از خط ممتد که کلاس پدر قرار دارد.

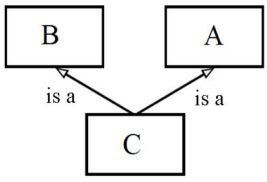
توجه: ذکر «is a» بر روی خط ممتد رابطه وراثت اختیاری است.

توجه: در تعریف رابطه وراثت، مشخصاتی نظیر تعدد و محدودیت وجود ندارد.

شکل زیر نمونه‌ای از یک سلسله مراتب ارث‌بری میان «میوه»، «سیب» و «سیب قرمز»، را نشان می‌دهد.



توجه: همانطورکه در فصل قبل نیز گفتیم، هرگاه یک کلاس برخی از صفات و عملیات را از یک کلاس و برخی دیگر را از یک کلاس دیگر به ارث ببرد، در این حالت وراثت چندگانه رخ داده‌است. شکل زیر نمونه‌ای از مدل‌سازی «**ارث‌بری چندگانه**» (Multiple Inheritance) را نشان می‌دهد.



در شکل فوق کلاس C به صورت وراثت چندگانه از کلاس A و کلاس B ارث‌بری کرده است.

پس از مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار نوبت به مدل‌سازی تعاملات پویای میان اشیاء همکار می‌رسد.

ی) مدل‌سازی تعاملات پویای میان اشیاء همکار

مدل‌سازی تعاملات پویای میان اشیاء همکار یا مدل‌سازی رفتاری، رفتار سیستم را در زمان اجرا نمایش می‌دهد. به بیان دیگر در این دیدگاه رفتارهای پویای سیستم مدل می‌شود.

مدل‌سازی رفتاری سیستم با استفاده از نمودارهای UML قابل انجام است:

الف) نمودار توالی (Sequence Diagram)

همانطورکه گفتیم نمودار کلاس ساختار ایستای داخل یک use case را نمایش می‌دهد. در یک سیستم در حال کار، به هر حال اشیاء با یکدیگر در تعامل هستند و این تعامل در طی زمان رخ می‌دهد. به بیان دیگر برای آنکه یک use case یا مورد کاربرد یا نیاز مرتع گردد، باید مجموعه‌ای از اشیاء با یکدیگر ارتباط برقرار کرده و پیام‌هایی را با یکدیگر رد و بدل نمایند. نمودار توالی نشان می‌دهد، برای مرتفع شدن یک use case یا مورد کاربرد یا نیاز، چه اشیایی باید چه پیام‌هایی را با چه ترتیبی ارسال کنند تا آن نیاز برآورده گردد. نمودار توالی، تعاملات پویا مابین اشیاء همکار را براساس زمان نشان می‌دهد.

توجه: Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

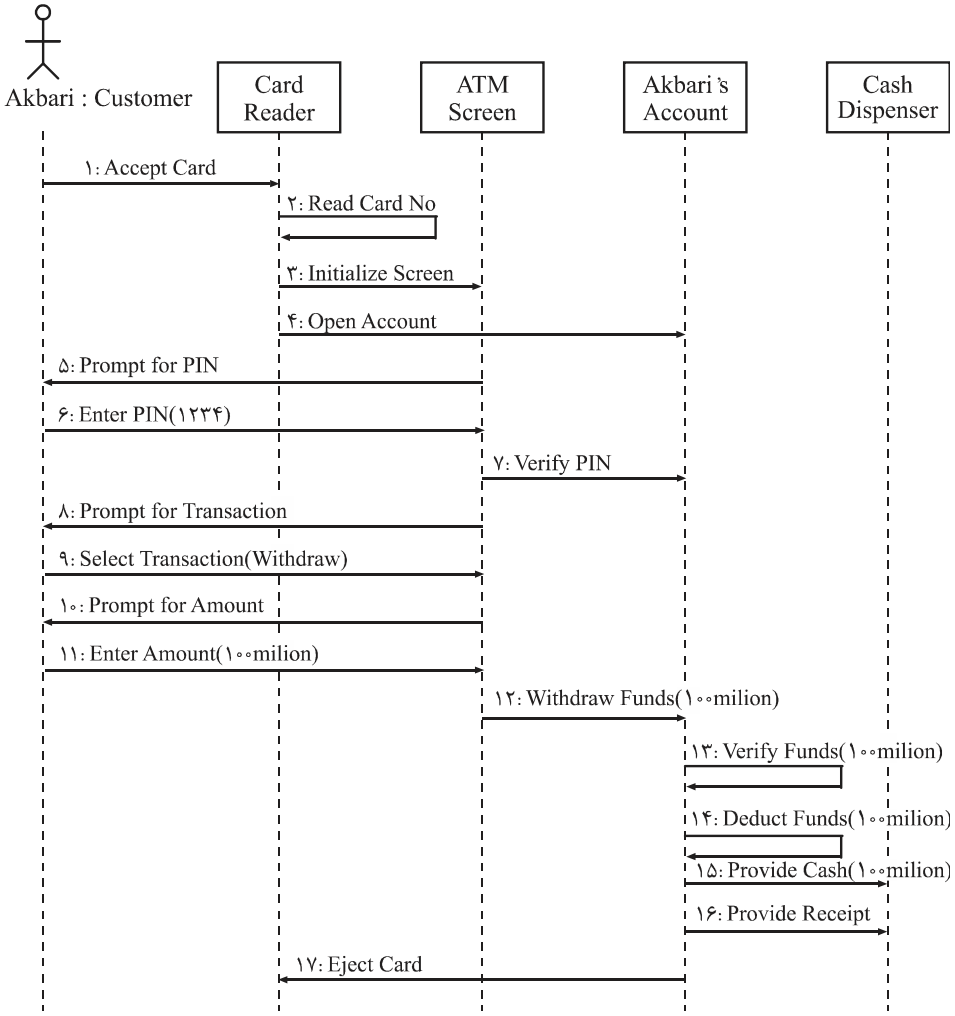
به طور کلی Sequence Diagram یا نمودارتوالی بر دو طبقه اصلی و فرعی می‌باشد:

نمودار توالی اصلی: نمایش روال تعاملات پویای میان اشیاء همکار داخل یک use case از نقطه شروع تا رسیدن به یک نتیجه مطلوب (موفق)

توجه: برای رسم نمودار توالی اصلی از سناریوی اصلی استفاده می‌گردد.

مثال: نمایش روال تعاملات پویای میان اشیاء همکار، برای برداشت وجه موفق از یک حساب، مربوط به use case برداشت وجه در یک سیستم ATM.

در شکل زیر طرح معمولی برداشت صد میلیون ریال پول بدون هیچ مسئله‌ای مانند وارد کردن رمز عبور (PIN) اشتباه نشان داده شده است:



نمودار توالی فوق جریان پردازش را در use case برداشت وجه (Withdraw Money) نشان می‌دهد.

همچنین اشیایی که سیستم نیاز دارد تا use case برداشت وجه را به نتیجه برساند در بالاترین نقطه نمودار نشان داده شده است. هر فلش یک پیغام ارسالی بین «بازیگر با شیء» یا «شیء با شیء» را نمایش می‌دهد. تا عملیات موردنیاز را به انجام برساند.

با توجه به نمودار، روال کار در use case برداشت وجه، بدین ترتیب شروع می‌شود که مشتری کارتش را وارد کارت‌خوان می‌کند، سپس کارت‌خوان شماره کارت را می‌خواند، شیء حساب آقای اکبری را باز می‌کند و صفحه نمایش ATM را مقداردهی می‌نماید. صفحه نمایش از آقای اکبری می‌خواهد که PIN را وارد نماید. او ۱۲۳۴ را وارد می‌کند. PIN وارد شده تأیید

می‌شود. صفحه نمایش انتخاب‌هایش را برای آقای اکبری آماده می‌کند و او برداشت صد میلیون ریال را انتخاب می‌کند. حساب آقای اکبری تأیید می‌کند که موجودی، حداقل شامل صد میلیون ریال است. سپس وجه از حساب کسر می‌شود و به صندوق اطلاع می‌دهد که صد میلیون ریال را آماده کند. همچنین یک رسید آماده شود. سرانجام به کارت‌خوان اطلاع داده می‌شود تا کارت را پس دهد. بنابراین این نمودار توالی، تمام جریان پردازشی use case مربوط به برداشت وجه موفق مربوط به توالی اصلی را با یک مثال نشان داد.

توجه: پس از اعتبارسنجی مشتری در مراحل ۱ تا ۷ و ارائه دریافت فرمان بعدی از سوی سیستم در مرحله ۸، در مرحله ۹، یعنی select transaction، از بین تراکنش‌های مختلف سیستم، مانند «برداشت وجه»، «انتقال وجه»، «تغییر رمز»، «نمایش موجودی» و غیره، مشتری تراکنش «برداشت وجه» را انتخاب کرده است. البته واضح است که مشتری در حال کار با مورد کاربرد، برداشت وجه است.

توجه: هر یک از چهار شیء فوق، به عنوان نماینده‌های یک ستون، در بالای شکل ظاهر می‌شوند. در زیر هر شیء خط چین‌های عمودی نشان داده شده است که به خط زندگی (Lifeline) موسوم است. محور عمودی بیانگر گذر زمان است. به بیان دیگر هر خط چین عمودی، خط زندگی یک شیء را نشان می‌دهد که تمام تعامل‌های آن را با اشیای دیگر از لحظه ایجاد تا زمان نابودی، مدل‌سازی می‌نماید. پیام‌هایی که در بین اشیاء گذر می‌کنند به شکل پیکان‌های افقی ظاهر می‌شوند. نام هر پیکان، متد و پارامترهای سرویس درخواستی را نشان می‌دهد. همچنین ترتیب پیام‌ها، با حرکت عمودی در امتداد محور اشیاء و رو به پایین مشخص می‌شود و به همین دلیل نیازی به شماره گذاری پیام‌ها نیست.

نمودار توالی فرعی: نمایش روال تعاملات پویای میان اشیاء همکار داخل یک use case از نقطه شروع تا رسیدن به یک نتیجه نامطلوب (ناموفق).

توجه: برای رسم نمودار توالی فرعی از سناریوهای فرعی استفاده می‌گردد.

مثال: نمایش روال تعاملات پویای میان اشیاء همکار، برای برداشت وجه ناموفق از یک حساب، مربوط به use case برداشت وجه در یک سیستم ATM. مواردی همچون تلاش برای برداشت پول از حساب بدون موجودی، تلاش برای برداشت پول با PIN اشتباه و غیره.

توجه: یک use case یک نمودار توالی اصلی و چندین نمودار توالی فرعی دارد.

ب) نمودار همکاری (Collaboration Diagram)

توجه: Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

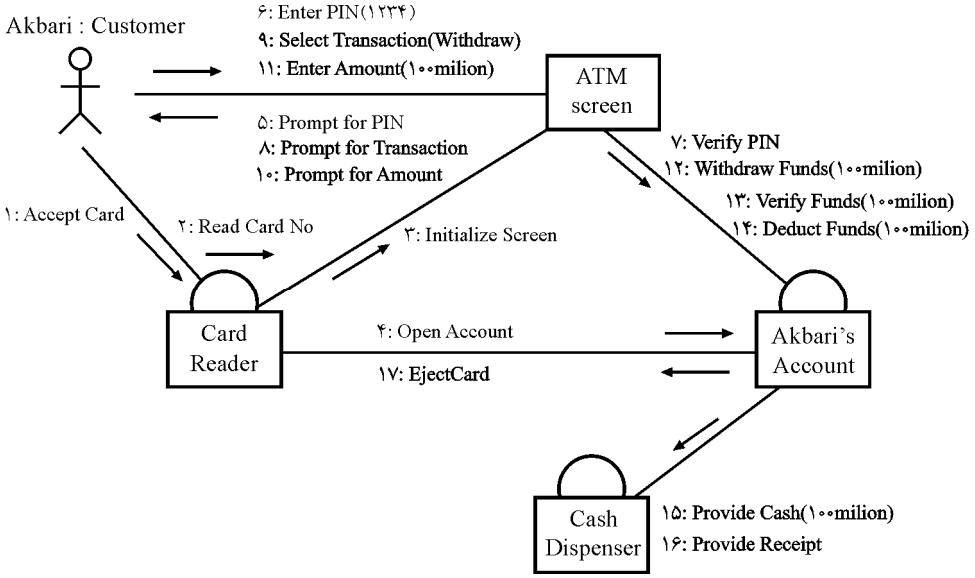
توجه: Collaboration Diagram یا نمودار همکاری، برای یک کاربرد خاص، جهت

مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

این نمودار نیز، مانند نمودار توالی، تعاملات بین اشیاء یک مورد کاربرد را نشان می‌دهد. در این نمودار نمی‌توان ترتیب زمانی را نشان داد. بنابراین، ترتیب ارسال پیام‌ها با شماره‌گذاری مشخص می‌شود. نمودار همکاری دقیقاً همان اطلاعات نمودار توالی را نشان می‌دهد. اگرچه، نمودار همکاری اطلاعات را به روش متفاوت و با یک هدف متفاوت نشان می‌دهد.

نمودار همکاری مثال قبل (نمودار توالی) به صورت زیر نشان داده شده است:

در نمودار همکاری زیر، مانند قبل، شیءها به شکل مستطیل و بازیگرها به شکل آدمک نشان داده شده است:



در حالی که در نمودار توالی، شیءها و ارتباطات بازیگرها به ترتیب زمان توضیح داده شده‌اند، نمودار همکاری شیءها و فعل و انفعالات بازیگرها را بدون توجه به زمان نشان می‌دهد.

مثلاً در این نمودار می‌بینید که کارت‌خوان به حساب آقای اکبری اطلاع می‌دهد تا باز شود و حساب آقای اکبری به کارت‌خوان اطلاع می‌دهد تا کارت را پس دهد. همچنین شیءهایی که مستقیماً با دیگری ارتباط برقرار می‌کنند. با خطوطی که بین آن‌ها کشیده شده، نشان داده شده‌اند. اگر صفحه نمایش ATM و کارت‌خوان مستقیماً با یکدیگر رابطه داشته باشند، باید یک خط بین آنها کشیده شده باشد. نبودن این خط به این معنی است که هیچ ارتباط مستقیمی بین این دو شیء وجود ندارد.

بنابراین نمودارهای همکاری همان اطلاعات نمودارهای توالی را نشان می‌دهد اما افراد به دلایل متفاوتی به نمودارهای همکاری مراجعه می‌کنند. مهندسین تضمین کیفیت و معماران سیستم

به این نمودارها نگاه می‌کنند تا توزیع شدن پردازش‌های بین شهرها را ببینند. فرض کنید که نمودار همکاری به شکل یک ستاره که در آن چند شیء که با یک شیء مرکزی ارتباط دارند، باشد. یک معمار سیستم ممکن است نتیجه بگیرد که سیستم خیلی به شیء مرکزی وابسته است و شیءها را دوباره طراحی نماید تا نیروی پردازش کردن را به طور یکنواخت توزیع کند. دیدن این نوع محاورات در یک نمودار توالی بسیار مشکل است.

توجه: همانند نمودار کلاس، نمودار همکاری نیز رابطه انجمنی میان اشیاء را نشان می‌دهد. به بیان دیگر در نمودار همکاری نیز باید مابین تمام کلاس‌هایی که در نمودار کلاس با یکدیگر رابطه انجمنی داشته‌اند، یک خط ارتباط رسم شود. پیام‌هایی که در بین اشیاء گذر می‌کنند به شکل پیکان‌هایی ظاهر می‌شوند. نام هر پیکان، متد و پارامترهای سرویس درخواستی را نشان می‌دهد. برای مثال، شیء حساب آقای اکبری یا Akbari Account به شیء ATM Screen توسط یک خط ممتد وصل شده است زیرا هر دو مستقیماً با دیگری **رابطه انجمنی و تبادل پیام** دارند. Card Reader به Cash Dispenser وصل نشده است زیرا این دو با هم ارتباطی ندارند. اشیاء کلاس‌های همکار از طریق صدا زدن متدهای عمومی یکدیگر اقدام به گفتگو و تبادل پیام می‌کنند. همچنین ترتیب پیام‌ها، به طور ذاتی و صریح مشخص نمی‌شود و به همین دلیل نیاز به شماره‌گذاری پیام‌ها می‌باشد.

ج) نمودار حالت (State Transition Diagram)

توجه: State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار انتقال حالت، چرخه حیات یک شیء را در حالت های مختلف (از زمانی که شیء ایجاد می‌شود تا زمانی که شیء از بین می‌رود) نمایش می‌دهد. در واقع این نمودار تمامی حالت‌های مختلفی را که یک شیء در طول حیات خود می‌تواند داشته باشد و همچنین نحوه انتقال بین حالت‌ها و وقایع باعث‌شونده این انتقال را نشان می‌دهد. به بیان دیگر نمودارهای حالت، راهی را آماده می‌کنند تا حالت‌های مختلف یک شیء را مدل کنند. در حالی که نمودارهای کلاس یک تصویر ثابت از کلاس‌ها و وابستگی آنها را نشان می‌دهد، نمودارهای حالت استفاده می‌شوند تا بیشتر، **رفتارهای پویای** یک سیستم را نمایش دهند.

توجه: برای تشخیص اینکه کلاسی دارای حالت‌های مختلفی است یا خیر، کافی است صفت‌های آن کلاس مورد بررسی قرار گیرد، اینکه یک شیء از یک کلاس چگونه با مقدارهای متفاوت در برخی از صفات خود می‌تواند در حالت‌های مختلفی قرار گیرد. اگر این‌چنین صفتی برای یک کلاس موجود باشد، این نشانه خوبی برای تشخیص ایجاد نمودار حالت برای کلاس مورد نظر است.

برای مثال یک کلاس حساب بانکی می‌تواند به چندین حالت متفاوت وجود داشته باشد.

می‌تواند «حساب باز دارای حداقل موجودی»، «حساب بسته» و «حساب باز بدون حداقل موجودی» باشد. یک حساب ممکن است در هر یک از این حالت‌ها، به طور متفاوتی رفتار کند. از نمودارهای حالت برای نشان دادن این اطلاعات استفاده می‌شود.

فرض کنید که برای انجام یک عملیات بانکی، اعم از ایجاد حساب، برداشت وجه و یا واریز وجه به بانک مراجعه می‌کنید. در هر کدام از این عملیات، بین «کارمند بانک»، «مشتری» **و** «حساب مشتری» اتفاقاتی می‌افتد که سناریوی هر کدام توسط نمودار توالی به خوبی قابل توصیف است. اما آیا می‌دانید که پس از انجام این عملیات، حساب بانکی مشتری در چه وضعیتی قرار می‌گیرد؟ آیا حساب بانکی مشتری همیشه در حالت ««حساب باز دارای حداقل موجودی» باقی می‌ماند؟ و یا این عملیات بانکی، بر روی حالت و وضعیت حساب تاثیر می‌گذارد؟ آیا نمودارهای توالی می‌توانند حالت‌های مختلف یک حساب بانکی را نشان داده و عوامل تغییردهنده آن را نیز مشخص نمایند؟

اجرای بعضی عملیات بر روی اشیاء ممکن است حالت و وضعیت آن‌ها را تغییر دهد. به عنوان مثال، شیء «حساب بانکی مشتری»، بر اثر اجرای عمل «برداشت وجه»، ممکن است به حالت «حساب باز بدون حداقل موجودی» وارد شود و یا حالت آن تغییر نکرده و همچنان در حالت «حساب باز دارای حداقل موجودی» باقی بماند.

همانطور که گفتیم حالت‌های یک شیء بر اساس مقادیر برخی از صفات شیء مشخص می‌شود. در شکل زیر، شیء حساب بانکی نشان داده شده است. در این شیء وقتی که مقدار صفت «موجودی»، به کمتر از ۱۰۰ هزار ریال برسد، شیء حساب بانکی وارد حالت «حساب باز بدون حداقل موجودی»، می‌شود. همچنین وقتی که مقدار موجودی به کمتر از ۱۰۰ هزار ریال برسد و در مدت ۳۰ روز به این حساب مراجعه نشود، بدین معنی که صفت «تاریخ آخرین عملیات»، به ۳۰ روز قبل اشاره کند، در این صورت حساب بانکی، وارد حالت «حساب بسته»، می‌شود و باید از سیستم حذف گردد.

حساب بانکی
محمد = نام صاحب حساب
۲۹۵۰۰۰ = موجودی
۹۱/۱/۷ = تاریخ آخرین عملیات

بنابراین، با توجه به اهمیت حالت‌های مختلف یک شیء و تاثیر آن‌ها بر عملکرد و رفتار سیستم، شناسایی حالت‌های مختف یک شیء به همراه عوامل تغییردهنده آن‌ها بسیار ضروری است. نمودار حالت به مدل‌سازی حالت‌های مختلف یک شیء در طول حیات آن، یعنی از زمان ایجاد تا زمان نابودی آن، می‌پردازد. علاوه بر آن، در نمودارحالت، عوامل تغییردهنده حالت‌های

یک شیء نیز مدل می‌شوند. همچنین، در هنگام تغییر حالت یک شیء ممکن است لازم باشد فعالیت‌هایی نیز انجام شود. این فعالیت‌ها نیز توسط نمودار حالت مدل‌سازی می‌شوند. برای مثال، به محض اینکه حسابی به حالت «حساب باز بدون حداقل موجودی»، وارد شود، موضوع باید به اطلاع دارنده حساب برسد.

همانطور که بیان شد، نمودار حالت، دوران حیات یک شیء را، در قالب اتفاقات و رخدادهایی که موجب تغییر حالت آن می‌شود، مدل‌سازی می‌کند. حالت یک شی مبین مقدار فعلی برخی صفات است، بدین معنی که مقدار فعلی برخی صفات یک شیء، حالت فعلی آن را مشخص می‌کند و تغییر مقادیر برخی صفات توسط اجرای عملیات مختلف، می‌تواند موجب تغییر حالت شیء گردد. زمانی که حالت یک شیء تغییر می‌نماید، نحوه پاسخگویی آن نیز تغییر می‌کند. برای مثال، وقتی که یک حساب بانکی در حالت «حساب باز دارای حداقل موجودی» است، مبلغ چک ممکن است پرداخت شود. در صورتی که چک، در حالت «حساب بسته»، برگشت داده می‌شود. نمودار توالی مبنای مناسبی برای ساخت نمودار حالت است. نمودار حالت، کل دوره حیات یک شیء را، از زمان ایجاد تا زمان نابودی آنرا، مورد بررسی قرار می‌دهد. در حالی که نمودار توالی، فقط برهه‌ای از زندگی شیء را، در طول یک سناریو نشان می‌دهد. بنابراین، برای به دست آوردن مجموعه حالت‌های مختلف یک شیء و مدل‌سازی نمودار حالت مربوطه، لازم است کلیه نمودارهای توالی که این شیء در آنها استفاده شده است، مورد بررسی قرار گیرند.

در ادامه به معرفی برخی از اصطلاحات مورد استفاده در نمودار حالت می‌پردازیم:

حالت (state): مشخص‌کننده وضعیت یک شیء است که بر اساس مقادیر فعلی برخی صفات آن تعیین می‌شود.

رخداد (event): به آنچه که موجب تغییر حالت یک شیء می‌گردد، رخداد گفته می‌شود. برای مثال، اگر عمل «برداشت وجه» به میزان زیاد انجام شود، رخداد «مانده حساب کمتر از ۱۰۰ هزار ریال»، واقع می‌شود و موجب تغییر حالت حساب از «حساب باز دارای حداقل موجودی» به «حساب باز بدون حداقل موجودی» خواهد شد. توجه داشته باشید که عمل «برداشت وجه»، همیشه موجب تغییر حالت حساب نمی‌شود.

اقدام ورودی (entry action): کارهایی که برای ورود به یک حالت انجام می‌شود. یک اقدام ورودی درون نماد حالت ظاهر می‌شود و یک کلمه entry و یک نماد : (colon) بعد از آن قرار می‌گیرد.

اقدام خروجی (exit action): کارهایی که برای خروج از یک حالت انجام می‌شود. یک اقدام خروجی درون نماد حالت ظاهر می‌شود و یک کلمه exit و یک نماد : (colon) بعد از آن قرار می‌گیرد.

فعالیت (activity): به عملیاتی که در یک حالت، اجرا می‌شوند، اما موجب تغییر حالت شیء

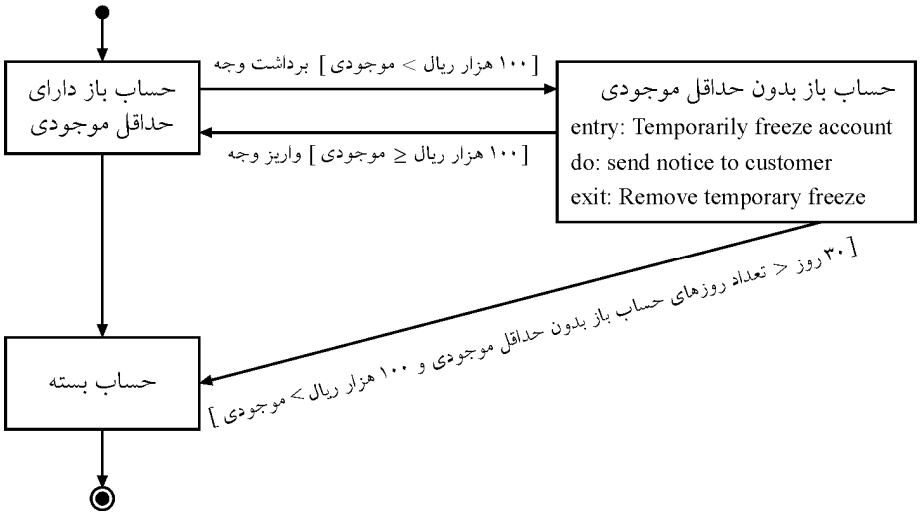
نمی‌گردند، فعالیت گفته می‌شود. مانند عمل «برداشت وجه» با حفظ حداقل موجودی که در حالت «حساب باز دارای موجودی حداقل» شیء حساب اجرا می‌شود، اما حالت آن را تغییر نمی‌دهد.

فعالیت داخلی: فعالیت داخلی، در شرایطی خاص، درون خود حالت برای اطلاع رسانی انجام می‌گردد. برای مثال وقتی که از یک حساب باز دارای حداقل موجودی، بیش از موجودی برداشت می‌شود، یک اخطار به مشتری فرستاده می‌شود. به عبارت دیگر وقتی شیء حساب بانکی به حالت «حساب باز بدون حداقل موجودی» وارد می‌شود، بهتر است این موضوع به اطلاع مشتری برسد. یک فعالیت داخلی درون نماد حالت ظاهر می‌شود و یک کلمه DO و یک نماد : (colon) بعد از آن قرار می‌گیرد.

پیکان انتقال: به پیکانی که بین دو حالت یک شیء کشیده می‌شود تا تغییر حالت آن را نشان دهد، پیکان انتقال گفته می‌شود.

توجه: هدف نمودار حالت، مدل‌سازی حالت‌ها، رخدادها و مشخص کردن ارتباط بین آنها است.

مثال: شکل زیر نمودار حالت را برای یک حساب بانکی نشان می‌دهد.



نمودار تغییر حالت متعلق به کلاس حساب (Account)

در این نمودار می‌توانیم حالت‌های مختلف یک حساب را ببینیم. همچنین می‌توانیم ببینیم که چگونه یک حساب از یک حالت به حالت دیگر منتقل می‌شود. برای مثال وقتی یک حساب در حالت «حساب باز دارای حداقل موجودی» است و مشتری درخواست بستن حساب را صادر می‌کند، حساب به حالت «حساب بسته» منتقل می‌شود. همانطور که گفتیم درخواستی از مشتری

که منجر به تغییر حالت می‌شود «رخداد» نامیده می‌شود و رخداد چیزی است که موجب می‌شود یک انتقال از حالتی به حالت دیگر صورت گیرد. برای مثال اگر حساب در حالت «حساب باز دارای حداقل موجودی» باشد و مشتری عمل «برداشت وجه» از حساب را با عدم حفظ حداقل موجودی انجام دهد، حساب از حالت «حساب باز دارای حداقل موجودی» به حالت حالت «حساب باز بدون حداقل موجودی» تغییر حالت می‌دهد.

در شکل فوق entry: Temporarily freeze account «اقدام ورودی» به معنی حساب به طور موقت مسدود شده، exit: Remove Temporary freeze «اقدام خروجی» به معنی حذف انسداد و do: Send notice to customer «فعالیت داخلی» است.

یک شرط که در براکت محصور شده است «شرط حفاظتی» (Guard Condition) نامیده می‌شود و وقوع یک انتقال (اینکه بتواند یا نتواند اتفاق بیفتد) را کنترل می‌کند.

در حالت ویژه، حالت شروع (Start State) و حالت پایان (Stop State) وجود دارد. «حالت شروع» یک شیء که مشخص‌کننده حالت اولیه و شروع زندگی آن است، باید به طور خاصی مدل‌سازی شود. برای این منظور، علاوه بر نماد حالت، باید یک دایره توپر به همراه یک پیکان که به حالت شروع اشاره دارد، ترسیم شود. توجه داشته باشید که حالت شروع، در برگیرنده همه اجزای مطرح شده است، یعنی نقطه توپر، پیکان و نماد حالت. بنابراین نقطه توپر و پیکان نیز جزء حالت شروع محسوب می‌شوند. نام رخدادی که باعث تغییر یک شیء از یک حالت به حالت دیگر می‌شود، بر روی پیکان انتقال بین دو حالت مذکور درج می‌گردد. جهت پیکان انتقال نیز، جهت تغییر حالت شیء را مشخص می‌کند. همانطور که هر شیء زمانی چرخه زندگی خود را از حالت شروع آغاز می‌کند، زمانی نیز به انتهای زندگی خود می‌رسد که به آن «حالت پایانی» گفته می‌شود، به طوری که پس از ورود به این حالت، به هیچ حالت دیگری نمی‌تواند وارد شود. نام حالت پایانی به وسیله یک دایره توپر مضاعف نمایش داده شده است و نشان می‌دهد که شیء درست قبل از اینکه از بین برود، در چه حالتی می‌باشد، شایان ذکر است که یک شیء فقط یک حالت شروع دارد، اما می‌تواند چندین حالت پایان داشته باشد. نمودارهای حالت برای هر کلاس ایجاد نمی‌شوند. آنها فقط برای کلاس‌های دارای حالات مختلف استفاده می‌شوند. اگر یک شیء از یک کلاس می‌تواند در چند حالت وجود داشته باشد و در هر حالت، متفاوت رفتار نماید، ممکن است بخواهد یک نمودار حالت برای آن ایجاد کنید. بسیاری از پروژه‌ها اصلاً به این نمودار نیازی ندارند. اگر آنها ایجاد شده‌اند، برنامه‌نویسان از آنها در زمان تولید کلاس‌ها استفاده می‌کنند، نمودار حالت خروجی کد ندارد بلکه به عنوان یک راهنما جهت تولید کد مورد استفاده قرار می‌گیرد. نمودارهای حالت فقط برای مستندسازی ایجاد شده‌اند.

توجه: همانطور که پیش از این نیز بیان شد، آنچه که باعث تغییر حالت یک شیء می‌گردد، تغییر مقادیر برخی صفات آن است و آنچه که باعث تغییر مقادیر صفات می‌شود، اجرای یکی از متدهای شیء صاحب آن صفات است. بنابراین، با توجه به اینکه اولا اجرای متدهای یک شیء

می‌تواند عاملی برای تغییر حالت آن باشد، ثانیا نمودار حالت، توصیفگر حالات مختلف یک شیء و علل و عوامل تغییر حالت آن است و ثالثا نمودارهای توالی، نمایانگر ترتیب و تقدم اجرای متدهای مختلف اشیای سیستم هستند، در نتیجه می‌توان از نمودارهای توالی به نمودارهای حالت رسید.

توجه: اطلاعات به دست آمده از نمودار حالت در مدل تحلیل مربوط به کلاس‌های دارای حالات مختلف، در مدل طراحی، بخش طراحی مولفه، جهت تکمیل نمودن الگوریتم‌های مربوط به متدهای کلاس‌های دارای حالات مختلف مورد استفاده قرار می‌گیرد. اینکه متدهای یک کلاس دارای حالات مختلف در سطح طراحی مولفه در مواجه با حالت‌های مختلف یک شیء چگونه رفتار کنند از نمودار حالت استنباط می‌گردد. طراحی مولفه جلوتر در بخش مدل طراحی شرح داده می شود.

توجه: همه نمودارهای فوق (توالی، همکاری و حالت)، مدل‌سازی رفتاری یا تعاملی محسوب می‌گردند.

توجه: از آنجا که وقوع رخداد در نمودارهای توالی و همکاری (ارتباط) نقشی ندارد، به نمودارهای رفتاری توالی و همکاری، مدل‌سازی رفتاری ایستا و به نمودار حالت، مدل‌سازی رفتاری پویا نیز گفته می‌شود.

مدل طراحی: OOD (Object – Oriented Design)

پس از مدل تحلیل، نوبت به مدل طراحی می‌رسد. مدل طراحی به روش شیگراء شامل چهار بخش **طراحی داده**، **طراحی معماری**، **طراحی مؤلفه** و **طراحی واسط** می‌باشد.

طراحی داده

طراحی داده، شامل طراحی ساختمان داده‌ها و طراحی پرس‌وجوها می‌باشد. هنگامی که قابلیت‌های پایگاه داده، با قابلیت‌های زبان برنامه‌نویسی شیگراء، ترکیب شوند، نتیجه، پایگاه داده شیگراء یا Object – Oriented DBMS خواهد شد. پایگاه داده شیگراء، به برنامه‌های شیگراء اجازه می‌دهند که اطلاعات خود را به همان صورت اشیاء در بانک اطلاعاتی ذخیره، بازیابی و بروزرسانی کنند، بخاطر سازگاری که بین پایگاه داده شیگراء و زبان برنامه‌نویسی شیگراء وجود دارد، برنامه‌نویس می‌تواند هر دو ابزار را در یک محیط مجتمع داشته باشد.

توجه: امروزه در تولید نرم‌افزارها، ترکیبی از سادگی پایگاه داده رابطه‌ای و مفاهیم پایگاه داده شیگراء، مورد استفاده قرار می‌گیرد.

طراحی معماری

طراحی معماری، نمودار کلاس و نمودار توالی مرتبط با هر یک از موارد کاربرد را از مدل تحلیل، به عنوان ورودی دریافت کرده و توسط سبک شیءگرا (مبتنی بر ارسال پیام مابین اشیاء)،

طراحی معماری را انجام می‌دهد.

در معنای عام، معماری به معنی نحوه ارتباط بخش‌های مختلف یک سازه است. در حیطه مهندسی نرم‌افزار نیز معماری به معنی نحوه ارتباط بخش‌های مختلف سازه‌ای به نام برنامه کامپیوتری است. طراحی معماری یا معماری نرم‌افزار، ساختار کلی نرم‌افزار و شیوه‌های یکپارچگی یک سیستم را بیان می‌کند. به عبارت دیگر، ساختار سلسله مراتبی **مولفه‌های برنامه (کلاس‌ها یا پیمانه‌ها)**، شیوه تعامل مولفه‌ها با یکدیگر و **ساختمان داده‌های** مورد نیاز مولفه‌ها را نشان می‌دهد. معماری نرم‌افزار یک مدل قابل درک از چگونگی سازمان‌دهی سیستم است. در واقع نشان‌گر ساختمان داده‌ها و مؤلفه‌های برنامه‌ای است که برای ساختن یک سیستم کامپیوتری لازم است. به طور دقیق‌تر معماری نرم‌افزار شامل دو سطح از طراحی می‌باشد یعنی طراحی داده و طراحی معماری. در واقع این ساختار مانند یک نقشه ساختمان، مبنای ساخت نرم‌افزار قرار می‌گیرد.

توجه: در طراحی معماری، اسکلت، ساختار و چیدمان کلی مؤلفه‌های برنامه به این معنی که چه مولفه‌ای (کلاسی) با چه مولفه‌ای (کلاسی) دیگر در ارتباط است، بدون ذکر جزئیات مربوط به شرح متدهای کلاس‌های همکار داخل یک مؤلفه فرعی (یک بخش از نرم‌افزار). مانند اسکلت یک ساختمان که گویای جایگاه مؤلفه‌های ساختمان است اما هنوز آجر چینی نشده است. (اسکلت یک ساختمان بدون آجر چینی).

توجه: واحد مؤلفه (پیمانه) در شی‌گرایی، کلاس است که از کنار هم قرار گرفتن کلاس‌های همکار داخل هر use case یا مورد کاربرد یا نیاز، یک مؤلفه فرعی (مؤلفه بخشی) ایجاد می‌گردد، که از کنار هم قرار گرفتن مؤلفه‌های فرعی برنامه، مؤلفه اصلی (مؤلفه کلی یا برنامه اصلی) ایجاد می‌گردد.

توجه: در این مرحله هر use case یا مورد کاربرد یا نیاز با اطلاعات مربوط به نمودارهای کلاس و توالی به یک مؤلفه فرعی ولی بدون ذکر جزئیات تبدیل می‌گردد. همچنین در این مرحله نحوه چیدمان مولفه‌ها و ساختار کلی برنامه، کلاس‌های همکار، اشیاء همکار و تعریف ساختار پیام‌ها مابین فرستنده و گیرنده پیام‌ها اما بدون ذکر جزئیات مشخص می‌شود.

توجه: به طراحی معماری، طراحی کلی نیز گفته می‌شود.

طراحی مؤلفه

طراحی مؤلفه، طراحی معماری از همان فعالیت مدل طراحی را به عنوان ورودی دریافت کرده و طراحی مؤلفه را توسط ابزارهایی همچون شبه کد یا Activity Diagram یا Swimlane Diagram ایجاد می‌کند.

طراحی مولفه، فعالیت تبدیل طراحی معماری به نرم‌افزار است. در این مرحله، سطح انتزاع طراحی معماری به سطح انتزاع نرم‌افزار کاربردی نزدیک می‌گردد. طراحی در سطح مؤلفه‌ها، نرم‌افزار را در سطحی از انتزاع تصویر می‌کند که به کد نزدیک است. طراحی مولفه، به عنوان نقشه

راهی دقیق، و نزدیک به زبان پیاده‌سازی، در فعالیت پیاده‌سازی نرم‌افزار، منجر به صرفه جویی در زمان و هزینه‌های تولید می‌گردد. در طراحی مولفه، مهندس نرم‌افزار باید ساختمان داده‌ها، واسطها، ساختار پیام‌ها و متدها را با جزییات کافی به نمایش در آورد تا راهنمای تولید کد منبع زبان برنامه‌نویسی باشد.

توجه: در طراحی مؤلفه، اسکلت، ساختار و چیدمان کلی مؤلفه‌های برنامه به این معنی که چه مولفه‌ای (کلاسی) با چه مولفه‌ای (کلاسی) دیگر در ارتباط است، با ذکر جزئیات مربوط به شرح متدهای کلاس‌های همکار داخل یک مؤلفه فرعی (یک بخش از نرم‌افزار). مانند اسکلت یک ساختمان که گویای جایگاه مؤلفه‌های ساختمان است و آجرچینی هم شده است. (اسکلت یک ساختمان به همراه آجرچینی).

توجه: در این مرحله هر use case یا مورد کاربرد یا نیاز با اطلاعات مربوط به نمودارهای کلاس و توالی به یک مؤلفه فرعی اما با ذکر جزئیات تبدیل می‌گردد. همچنین در این مرحله نحوه چیدمان مولفه‌ها و ساختار کلی برنامه، کلاس‌های همکار، اشیاء همکار و تعریف ساختار پیام‌ها مابین فرستنده و گیرنده پیام‌ها نیز با ذکر جزئیات مشخص مشخص می‌شود.

توجه: همانطور که گفتیم، در طراحی مؤلفه، باید جزئیات الگوریتمی متدهای کلاس تشریح شود. برای این منظور Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط شنا مورد استفاده قرار می‌گیرد. نمودار فعالیت و نمودار خط‌شنا ساختاری مشابه فلوچارت دارد. نمودار فعالیت و نمودار خط‌شنا نه تنها شرح‌حال متدهای کلاس را تصویر می‌کند، بلکه مطابق آنچه پیش از این در بخش مدل تحلیل شی‌گراء یا OOA گفتیم، جهت مدل‌سازی سناریوهای اصلی و فرعی داخل یک use case نیز مورد استفاده قرار می‌گیرد. علاوه بر نمودار فعالیت و نمودار خط‌شنا، توسط شبه کد (PDL) نیز می‌توان جزئیات مربوط به شرح متدهای کلاس را نمایش داد.

توجه: اطلاعات به دست آمده از نمودار حالت در مدل تحلیل مربوط به کلاس‌های دارای حالات مختلف، در مدل طراحی، بخش طراحی مولفه، جهت تکمیل نمودن الگوریتم‌های مربوط به متدهای کلاس‌های دارای حالات مختلف مورد استفاده قرار می‌گیرد. اینکه متدهای یک کلاس دارای حالات مختلف در سطح طراحی مولفه در مواجه با حالت‌های مختلف یک شیء چگونه رفتار کنند از نمودار حالت استنباط می‌گردد.

توجه: از کنار هم قرار گرفتن کلاس‌های همکار درون یک use case به عنوان واحد مولفه، یک مولفه بزرگتر با عنوان مولفه فرعی (مولفه بخشی) ایجاد می‌گردد و از اجتماع مؤلفه‌های فرعی (مؤلفه بخشی)، مؤلفه اصلی (مؤلفه کلی یا برنامه اصلی) خلق می‌گردد. در سیستم ATM، اجتماع مولفه‌های فرعی «برداشت وجه»، «واریز وجه»، «انتقال وجه»، «تغیر کلمه عبور»، «پرداخت» و «نمایش موجودی»، منجر به ساخت مؤلفه اصلی (مؤلفه کلی یا برنامه اصلی) یا سیستم ATM،

می‌گردد.

توجه: به طراحی مولفه، طراحی جزئی، طراحی تفصیلی و طراحی رویه‌ای نیز گفته می‌شود.

طراحی واسط

طراحی واسط یا همان واسط کاربر، براساس ورودی‌ها و خروجی‌های مورد نیاز کاربران نهایی به شکل نقشی بر روی کاغذ یا طرحی بر روی کامپیوتر ایجاد می‌گردد. مانند نحوه چیدمان منوها و فرم‌ها.

۴- فعالیت ساخت (پیاده‌سازی و تست)

پس از فعالیت مدل‌سازی (تحلیل و طراحی) نوبت به فعالیت ساخت (پیاده‌سازی و تست) می‌رسد.

پیاده سازی: OOP (Object – Oriented Programming)

پس از مدل طراحی نوبت به پیاده‌سازی می‌رسد. این کار توسط زبان‌های شی‌گراء، مانند C++ و SQL Server انجام می‌گردد. در این مرحله، ساختار کلاس‌ها، ساختار و نحوه ارسال پیام‌ها مابین اشیاء، ساختمان داده‌ها و پرس‌وجوها برای ایجاد مؤلفه‌های فرعی (مؤلفه بخشی) و به تبع ایجاد مؤلفه اصلی (مؤلفه کلی یا برنامه اصلی) پیاده‌سازی می‌گردند.

توجه: مؤلفه یک قطعه‌ی آماده، کاربردی و پیاده‌سازی شده است که دارای واسط لازم جهت اتصال با سایر قطعات و استقرار در بخشی از یک سیستم عملیاتی می‌باشد. از آنجایی که در یک مولفه، فقط بخشی از کدها یا داده‌های نرم‌افزار قرار می‌گیرد، لذا هر مولفه ممکن است نیازمند ارتباط با مولفه‌های دیگر باشد تا بتواند از کدها یا داده‌های موجود در آنها استفاده نماید. برای انجام این ارتباط، در مولفه‌ها یک واسط در نظر گرفته می‌شود تا کدها یا داده‌های یک مولفه، از طریق واسط آن در اختیار مولفه‌های دیگر قرار گیرد، به این معنی که ارتباط بین مولفه‌ها، فقط از طریق واسط‌های آنها انجام می‌گیرد. در برنامه‌نویسی شیءگرا، متدهای عمومی کلاس‌ها، نقش واسط را ایفا می‌کنند. به عبارت دیگر کلاس‌های همکار، از طریق متدهای عمومی یکدیگر اقدام به گفتگو و تبادل پیام می‌کنند.

در دیدگاه شیءگراء به مؤلفه پیمانه نیز گفته می‌شود. در حیطه مهندسی نرم‌افزار شیءگرا، **واحد مؤلفه** یک قطعه‌ی عملیاتی مبتنی بر کلاس است که بر دو نوع می‌باشد:

۱) کلاس کاربردی: مانند کلاس دانشجو که برنامه‌نویس آن را می‌نویسد و به دلیل داشتن شرایط قابل حمل به شکل ذاتی، می‌تواند به عنوان یک قطعه‌ی آماده و قابل استفاده‌ی مجدد در پروژه‌های بعدی مورد استفاده مجدد قرار گیرد.

۲) کلاس سیستمی: مانند کلاس جعبه‌ی متن که کامپایلر تعاریف آن را فراهم می‌کند و می‌تواند به عنوان یک قطعه‌ی آماده و قابل استفاده‌ی مجدد، در پروژه‌ها مورد استفاده مجدد قرار

گیرد.

برای مدل‌سازی این مرحله نمودار مولفه مورد استفاده قرار می‌گیرد.

نمودار مؤلفه (Component Diagram)

این نمودار به نحوه پیاده‌سازی فیزیکی نرم‌افزار می‌پردازد. در واقع، نمودار مولفه شامل کدهای نرم‌افزار است که در آن هر قطعه کد در قالب یک مولفه، دسته‌بندی می‌شود. کدها نیز چیزی جز کلاس‌های سیستم، در زمانی که جزئیات پیاده‌سازی به آن‌ها اضافه شده باشد، نیست. برای تولید یا توسعه یک سیستم نرم‌افزاری، ابتدا لازم است که نیازمندی‌های مشتری توسط تیم توسعه در فعالیت ارتباطات شناسایی گردد و سپس نرم‌افزار پیشنهادی، در قالب یک طرح منطقی در اختیار مشتری قرار گیرد. این طرح منطقی، با استفاده از نمودارهایی که تاکنون آموخته‌اید، مدل می‌شود. اما آیا UML به نحوه پیاده‌سازی نرم‌افزار و کیفیت سخت‌افزارهای مورد نیاز نیز توجهی دارد؟ پاسخ، قطعا مثبت است، UML علاوه بر ارائه یک طرح منطقی از نرم‌افزار، به جنبه‌های پیاده‌سازی و سخت‌افزاری آن نیز می‌پردازد. نمودار مولفه چگونگی پیاده‌سازی نرم‌افزار را مورد بررسی قرار می‌دهد. نمودار استقرار نیز، معماری سخت‌افزارهای مورد نیاز را مدل می‌نماید. ترکیب این دو نمودار، نحوه اجرای نرم‌افزار را روی سخت‌افزار مشخص می‌کند. نمودار مولفه در ادامه و نمودار استقرار در بخش بعدی مورد بررسی قرار می‌گیرند.

توجه: Component Diagram یا نمودار مؤلفه جهت مدل‌سازی ساختار کلی پیاده‌سازی برنامه و ترتیب کامپایل مؤلفه‌های فرعی، برای ایجاد مؤلفه‌اصلی (مؤلفه کلی یا برنامه اصلی) مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار مؤلفه یک دید فیزیکی از مدل سیستم به همراه مؤلفه‌های فرعی نرم‌افزار و روابط بین آنها را نشان می‌دهد.

توجه: نمودار مؤلفه زمانی که تولید کد تمام شده است ایجاد می‌گردد. بنابراین در اینجا منظور از مؤلفه فرعی یک مولفه فیزیکی کد است. به بیان دیگر مؤلفه فرعی در این مرحله، کلاس‌های همکار پیاده‌سازی شده داخل یک use case یا مورد کاربرد یا نیاز پیاده‌سازی شده است. دقت کنید که قبل از استفاده از نمودار مولفه هریک از کلاس‌های همکار موجود در نمودار کلاس مربوط به هر use case یا مورد کاربرد یا نیاز باید به یک مؤلفه فرعی حاوی کد منبع تبدیل شود.

توجه: در این مرحله هر use case یا مورد کاربرد یا نیاز، مطابق روالی که از فعالیت‌های ارتباط تا ساخت (پیاده‌سازی) طی می‌کند، سرانجام توسط کلاس‌های همکار پیاده‌سازی شده خود به یک مؤلفه فرعی (مؤلفه‌بخشی) پیاده‌سازی شده و کامپایل شده تبدیل می‌گردد، که از اجتماع این مؤلفه‌های فرعی (مؤلفه‌بخشی)، مؤلفه اصلی (مؤلفه کلی یا برنامه اصلی) خلق می‌گردد. و محصول نهایی آماده می‌گردد.

توجه: به یک مولفه فرعی که از تعدادی کلاس همکار پیاده‌سازی شده، ایجاد شده است، **مولفه محقق شده یا وابستگی محقق شده** (realization dependency) نیز گفته می‌شود.

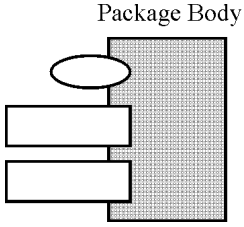
توجه: به یک مولفه فرعی، زیرسیستم (subsystem) نیز گفته می‌شود.

توجه: Component Diagram یا نمودار مؤلفه را می‌توان در زمره نمودارهای دید ایستا به شمار آورد. زیرا نمودارهای کلاس و مولفه از یک جنس هستند و شاید تنها سطح انتزاع آنها با یکدیگر متفاوت باشد.

در نمودار مؤلفه، دو نوع مؤلفه وجود دارد:

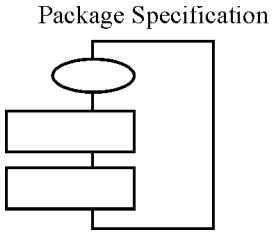
۱- مؤلفه قابل اجرا یا Package Body

حاوی کد عملیات (Operation) کلاس می‌باشد. در زبان C++، Package Bodyها فایل‌های CPP، هستند. این نوع مؤلفه با نماد زیر در UML نشان داده می‌شود:



۲- مؤلفه کتابخانه‌های کد یا Package Specification

حاوی تعاریف توابع سیستمی است. در زبان C++، Package Specificationها فایل‌های h. هستند. این نوع مؤلفه با نماد زیر در UML نشان داده می‌شود:



به طور عمومی، بسته(Package) مجموعه‌ای از کلاس‌های پیاده‌سازی‌شده و کامپایل‌شده به عنوان واحد مولفه و یا تعدادی مولفه فرعی پیاده‌سازی‌شده و کامپایل‌شده است. که از اجتماع بسته‌ها یک سیستم نرم‌افزاری ایجاد می‌گردد.

توجه: به یک بسته(Package) ، زیرسیستم (subsystem) نیز گفته می‌شود.

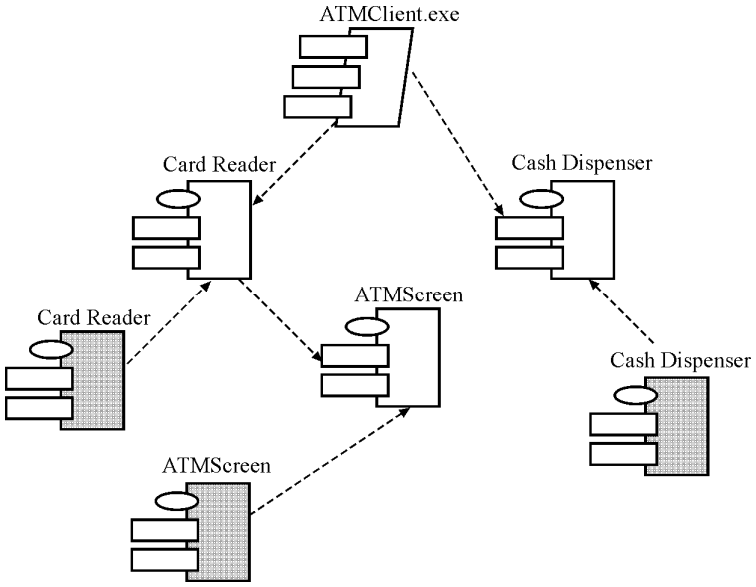
برای مثال سیستم ATM شامل دو بسته است:

۱- بسته ATMServer برای بخش سرویس‌دهنده

۲- بسته ATMClient برای بخش سرویس‌گیرنده

الف) بسته ATMClient

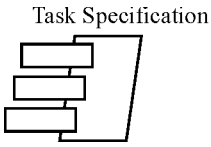
شکل زیر نمودار مؤلفه بسته سرویس‌گیرنده سیستم ATM یا ATMClient را نشان می‌دهد:



نمودار مولفه برای سرویس‌گیرنده ATMClient

در صورتی که زبان پیاده‌سازی C++ باشد، هر کلاس مؤلفه قابل اجرا (CPP.) و کتابخانه‌های کد (h.) خود را دارد. بنابراین هر کلاس در نمودار مؤلفه به یک کلاس پیاده‌سازی شده نگاشت می‌شود. برای مثال کلاس ATMScreen به یک کلاس پیاده‌سازی شده به نام ATMScreen نگاشت می‌شود.

توجه: یک برنامه قابل اجرا (برنامه اصلی یا برنامه کلی) که از مولفه‌های فرعی (مولفه‌های بخشی) تشکیل شده است در نمودار مؤلفه UML توسط نماد زیر نمایش داده می‌شود.



توجه: یک فایل اجرایی معمولاً به عنوان یک Task Specification با یک پسوند EXE. نمایش داده می‌شود.

توجه: تنها نوع رابطه‌ای که می‌تواند بین مؤلفه‌ها در نمودار مولفه وجود داشته باشد، یک رابطه وابستگی (Dependency) است و به این معنی است که یک مؤلفه باید قبل از کدام مولفه دیگر کامپایل شود.

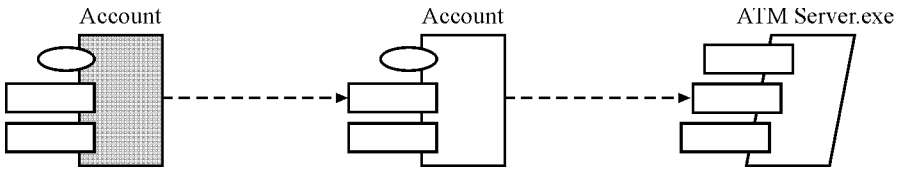
توجه: در نمودار مولفه، رابطه وابستگی بین مولفه‌ها توسط خطوط خط‌چین نشان‌داده می‌شود.

مثال: کلاس Card Reader به کلاس ATM Screen وابسته است. یعنی کلاس ATM Screen باید موجود باشد تا کلاس Card Reader کامپایل شود.

توجه: فایل اجرایی ATMClient.exe اولین باری که همه کلاس‌ها کامپایل شوند، می‌تواند ایجاد گردد.

ب) بسته ATMServer

شکل زیر نمودار مؤلفه بسته سرویس‌دهنده سیستم ATM یا ATMServer را نشان می‌دهد:



نمودار مولفه برای سرویس‌دهنده ATMServer

توجه: فایل اجرایی ATMServer.exe اولین باری که کلاس Account کامپایل شود، می‌تواند ایجاد گردد.

همان‌طور که در این مثال نشان داده شد، یک سیستم بسته به تعداد بسته‌ها می‌تواند چندین نمودار مؤلفه داشته باشد.

تست: OOT (Object – Oriented Testing)

پس از پیاده‌سازی نوبت به تست می‌رسد، در این مرحله کلیه موارد پیاده‌سازی شده از نظر خطاهای نحوی و خطاهای معنایی براساس لیست نیازمندی‌های مشتری (چک لیست) که در فعالیت ارتباطات تهیه شده بود، مورد وارسی قرار می‌گیرد تامشخص شود نرم‌افزار، براساس ورودی‌های موردنظر مشتری، خروجی‌های مورد انتظار مشتری را برآورده می‌سازد یا خیر.

توجه: این فعالیت در گام اول توسط برنامه‌نویسان و در صورت لزوم در گام بعدی توسط یک گروه تست مستقل (ITG) انجام می‌گردد.

توجه: ITG سرواژه عبارت Independent Testing Group و به معنی گروه تست مستقل است.

فعالیت استقرار

پس از تست، نوبت به فعالیت استقرار می‌رسد. هدف از فعالیت استقرار در گام اول، ارائه یک دید کلی از سخت‌افزارهای مورد نیاز سیستم است که مؤلفه‌های نرم‌افزاری باید بر روی آن‌ها قرار گیرند. و در گام دوم، نرم‌افزار به مشتری تحویل داده می‌شود و مشتری با بررسی محصول

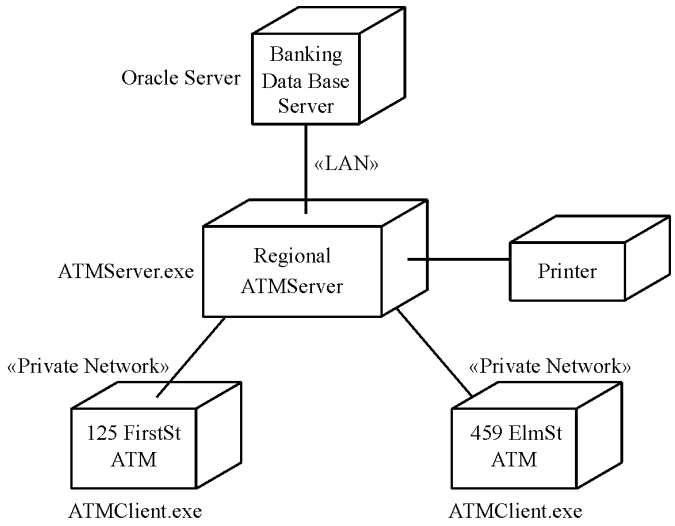
دریافتی، باز خوردهای به دست آمده براساس همین ارزیابی‌ها را به تیم نرم‌افزاری ارائه می‌دهد. این بازخوردها می‌توانند مبنایی برای ارتقاء و یا تصحیح نسخه‌ی بعدی نرم‌افزار باشد.

توجه: Deployment Diagram یا نمودار استقرار، جهت مدل‌سازی گره‌های سخت افزاری برای نصب نرم‌افزار مربوطه در فعالیت استقرار مورد استفاده قرار می‌گیرد.

نمودار استقرار (Deployment Diagram)

این نمودار معماری فیزیکی برای نصب یک سیستم مبتنی بر کامپیوتر را نشان می‌دهد. این نمودار می‌تواند کامپیوتر، دستگاه‌ها، اتصالات آنها با یکدیگر و نرم‌افزاری که روی هر دستگاه قرار می‌گیرد را نشان دهد. در اغلب نرم‌افزارهای امروزی کل ساختار یک نرم‌افزار بر روی یک گره سخت‌افزاری به شکل محلی قرار نمی‌گیرد. بلکه بخش‌های مختلف یک نرم‌افزار، بر روی گره‌های مختلف سخت‌افزاری به شکل غیر محلی توزیع می‌شوند. نمودار استقرار نشان می‌دهد که بخش‌های مختلف یک نرم‌افزار چگونه در گره‌های مختلف سخت‌افزاری توزیع می‌شوند.

سیستم ATM از سه بخش قابل اجرا (سرویس دهنده، سرویس گیرنده و پایگاه داده‌ها) بر روی گره‌های سخت‌افزاری مجزا تشکیل شده است. شکل زیر، نمودار استقرار سیستم ATM را نشان می‌دهد:



نمودار استقرار برای سیستم ATM

توجه: در بین گره‌ها، خطوطی رسم شده است، که نحوه ارتباط انجمنی مابین گره‌ها را بیان می‌کند، برای مثال «Regional ATMServer»، توسط شبکه محلی (LAN) به «Banking database server» متصل شده است. یک Stereotype برای توضیح ماهیت هر اتصال

استفاده می‌شود.

توجه: نمودار استقرار به خوبی می‌تواند ساختار شبکه یک سیستم را نشان دهد. بنابراین، نمودار شبکه می‌تواند از روی نمودار استقرار استخراج گردد.

سیستم ATM یک سبک معماری سه طبقه دارد:

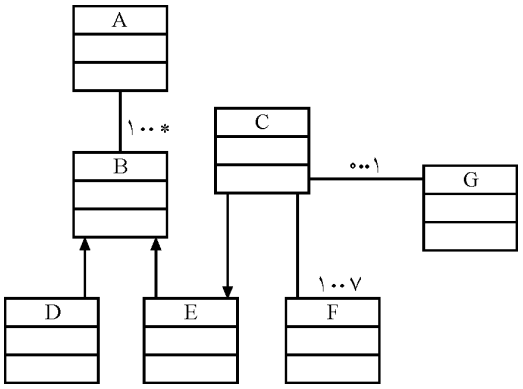
- سرویس‌دهنده
- سرویس‌گیرنده
- پایگاه داده‌ها

تست‌های فصل هفتم

۱- سیستم «مدیریت خانه» وجود وضعیت‌های نامطلوب مانند ورود غیرمجاز، آتش‌سوزی، ترکیدگی لوله آب و غیره را تشخیص می‌دهد و با انجام اقدامات لازم، خانه را در برابر این وضعیت‌ها محافظت می‌کند. این محصول، با استفاده از سنسورهای مناسب وضعیت‌های نامطلوب را کشف می‌کند و توسط صاحب‌خانه قابل برنامه‌ریزی است و در صورت کشف وضعیت نامطلوب به طور خودکار و از طریق خط تلفن، وضعیت را به اطلاع مرکز کنترل می‌رساند. در مدل مورد استفاده (use-case) این محصول، کدام‌یک از گزینه‌های زیر را به عنوان بازیگر (actor) در نظر گرفته نمی‌شود؟ (مهندسی IT – دولتی ۸۳)

۱) سنسور ۲) مرکز کنترل ۳) خط تلفن ۴) صاحب‌خانه

۲- با توجه به نمودار کلاس زیر گزینه‌ی صحیح را انتخاب کنید. (مهندسی IT – دولتی ۸۳)



۱) F بخشی از C می‌باشد. E نوعی از B می‌باشد. یک شیء از نوع G با چند شیء از نوع C رابطه دارد.

۲) C بخشی از E می‌باشد. D نوعی از B می‌باشد. یک شیء از نوع C با یک تا هفت شیء از نوع F رابطه دارد.

۳) E بخشی از C می‌باشد. E نوعی از B می‌باشد. یک شیء از نوع C با یک تا هفت شیء از نوع F رابطه دارد.

۴) C بخشی از E می‌باشد. B نوعی از D می‌باشد. یک شیء از نوع C با یک تا هفت شیء از نوع F رابطه دارد.

۳- کدام‌یک از نمودارهای UML در تجزیه و تحیل شیءگرا به شکل گرافیکی کارکردهای سیستم و ارتباط متقابل بین سیستم و موجودیت‌های خارج از سیستم را نشان می‌دهد؟ (مهندسی IT – دولتی ۸۴)

۱) use case diagram ۲) sequence diagram

۳) activity diagram ۴) collaboration diagram

۴- استخراج نیازمندی‌ها در شیءگرایی بر روی کدام‌یک از رویکردهای زیر استوار است؟
(مهندسی IT – دولتی ۸۴)

۱) مبتنی بر سرویس ۲) مبتنی بر هدف ۳) مبتنی بر فرآیند ۴) مبتنی بر وظیفه

۵- در مبحث شیءگرایی، زمانی که یک کلاس از تلفیق تعدادی کلاس دیگر تشکیل شود، دلالت بر چه نوع رابطه‌ای خواهد داشت؟ **(مهندسی IT – آزاد ۸۵)**

۱) رابطه وراثت (inheritance Relationship)
۲) رابطه انجمنی (Association Relationship)
۳) رابطه تجمعی (Aggregation Relationship)
۴) هیچ‌کدام

۶- در UML کدام‌یک از نمودارهای زیر، تعاملات درون مورد کاربردی (use-case) را نمایش می‌دهد؟ **(مهندسی IT – آزاد ۸۶)**

۱) نمودار کلاس (Class Diagram)
۲) نمودار تغییر حالت (State Transition Diagram)
۳) نمودار ترتیبی (Sequence Diagram)
۴) نمودار مؤلفه (Component Diagram)

۷- نمودارهای تعاملی (Interaction)، روابط میان کدام یک از گزینه‌های زیر را نمایش می‌دهد؟
(مهندسی IT – آزاد ۸۶)

۱) موارد کاربری (use cases) ۲) اشیاء (objects)
۳) کلاس‌ها (classes) ۴) بسته‌ها (packages)

۸- برای استخراج مشخصات رفتارهای ایستای (Static) موجودیت‌ها (Classes) کدام‌یک از ابزارهای زیر مناسب است؟ **(مهندسی IT – دولتی ۸۷)**

۱) Sequence Diagram, Use Case Model
۲) Collaboration Diagram, Use Case Model
۳) Sequence Diagram, Collaboration Diagram
۴) Use Case Model, State Transition Diagram

۹- یک انتقال حالت در نمودار حالت (State Chart Diagram) توسط کدام‌یک از عوامل زیر فعال می‌گردد؟ **(مهندسی IT – آزاد ۸۷)**

۱) عامل (actor) ۲) همکار (collaborator)
۳) رخداد (event) ۴) مؤلفه (component)

۱۰- کدام گزینه اولویت تعیین کلاس‌های سیستم و ترسیم نمودارهای ترتیبی (Sequence Diagrams) را به درستی بیان می‌نماید؟ **(مهندسی IT – آزاد ۸۷)**

۱) از آنجا که در نمودار ترتیبی به نمونه‌های کلاس نیاز داریم، لذا ابتدا می‌بایست کلیه کلاس‌های سیستم را مشخص نموده و سپس اقدام به ترسیم نمودار(های) ترتیبی نمود.

۲) همواره ابتدا نمودار(های) ترتیبی را رسم نموده و سپس اقدام به تعیین کلاس‌های سیستم می‌نماییم.

۳) نمودار ترتیبی و مشخص نمودن کلاس‌های سیستم دو فرآیند کاملاً متفاوت بوده و تقدم و تأخر در خصوص آنها مطرح نمی‌باشد.

۴) در ابتدا کلاس‌های اولیه را مشخص می‌نماییم و سپس نمودار(های) ترتیبی را رسم نموده و کلاس‌های سیستم را به روز می‌نماییم. به عبارت دیگر کلاس‌های سیستم به صورت تدریجی کامل می‌شود.

۱۱- کدام‌یک از نمودارهای UML ارایه‌کننده جریان کار در سطح سیستم، زیر سیستم، موارد کاربرد و کلاس‌ها می‌باشند؟ **(مهندسی IT – دولتی ۸۸)**

۱) Use Case Diagram ۲) Activity Diagram

۳) Class Diagram ۴) Collaboration Diagram

۱۲- تکمیل نمودن متدهای سیستم از وظایف کدام‌یک از نمودارهای زیر می‌باشد؟

(مهندسی IT – آزاد ۸۸)

۱) نمودار ترتیبی (Sequence Diagram) ۲) نمودار حالت (Statechart Diagram)

۳) نمودار مؤلفه (Component Diagram) ۴) نمودار استقرار (Deployment Diagram)

۱۳- در نمودار مؤلفه (Component Diagram)، رابطه وابستگی (Dependency) به چه منظور در میان مؤلفه‌ها قرار می‌گیرد؟ **(مهندسی IT – آزاد ۸۸)**

۱) مشخص نمودن ترتیب ایجاد مؤلفه‌ها ۲) مشخص نمودن ترتیب کامپایل

۳) نگاشت کلاس‌ها به مؤلفه‌ها ۴) نگاشت مؤلفه‌ها به بسته‌ها

۱۴- کدام عبارت صحیح است؟ **(مهندسی IT – دولتی ۸۹)**

۱) در تحلیل نیازها، افراز (Partitioning) مسأله منجر به تشریح دقیق‌تر داده‌ها، توابع و رفتار خواهد شد.

۲) Actorها در Use-Caseها افرادی هستند که کاربران نرم‌افزار خواهند بود.

۳) برای آنکه نمونه‌سازی نرم‌افزار عملی مؤثر باشد، نیاز به ابزاری برای توسعه‌ی سریع نمونه‌ها داریم تا طبق زمان‌بندی پیش برویم.

۴) وقتی که مستندات تعریف نیازهای نرم‌افزار به وسیله‌ی مشتری و توسعه‌دهنده، تأیید شد این اسناد تبدیل به مستنداتی غیرقابل تغییر خواهند شد.

۱۵- کدام عبارت نادرست است؟ **(مهندسی IT – دولتی ۸۹)**

۱) برای بازنگری یک مدل CRC کامل، بازنگری‌کننده، نیاز به توجه به تمام نمایش‌های موردکاربرد (Use Cases) دارد.

۲) مقادیری که به صفات یک شیء اختصاص می‌یابند، آن شیء را یکتا می‌نمایند.

۳) وراثت شامل مکانیزمی است که تغییرات در کلاس‌های سطح پایین به سرعت منجر به تعمیم روی تمام ابرکلاس‌ها می‌گردد.

۴) در حالتی که یک کلاس برخی از صفات و عملیات یک کلاس و برخی دیگر را از کلاس دیگر به ارث می‌برد، در این حالت وراثت چندگانه مطرح می‌گردد.

۱۶- کدام عبارت جزء ارتباطات عمومی بین کلاس‌های مشارکت‌کننده نیست و تحلیل‌گر نمی‌تواند براساس آن، کلاس‌های مشارکت را تشخیص دهد؟ **(مهندسی IT – دولتی ۸۹)**

۱) بخشی است از (...is-part-of)
۲) واقع شدن قبل از (...comes-before)
۳) آگاه است از (... has-knowledge-of)
۴) وابسته است به (...depends-upon)

۱۷- ارتباط میان اشیاء در شیءگرایی به چه صورت انجام می‌پذیرد؟ **(مهندسی IT – آزاد ۸۹)**

۱) شمول (Include)
۲) بسط (Extend)
۳) وابستگی (Dependency)
۴) تبادل پیام (Message Passing)

۱۸- کدام‌یک از موارد زیر برای گنجاندن در طراحی جزیی یک نرم‌افزار مناسب نیست؟
(مهندسی IT – دولتی ۹۰)

۱) حداکثر زمانی که استفاده‌کننده بایستی منتظر پاسخ سیستم بماند.

۲) نحوه ذخیره اطلاعات مربوط به عملیات انجام شده در هر روز

۳) اتفاقاتی که باید در صورت قطع اتصال از شبکه کامپیوتری در سیستم بیفتد.

۴) هیچ‌کدام

۱۹- در مدل‌سازی سیستم‌ها در جریان تحلیل و طراحی، کدام‌یک از نمودارهای زیر نوعاً نسخه‌ی فیزیکی ندارد و همواره منطقی باقی می‌ماند؟ **(مهندسی IT – دولتی ۹۰)**

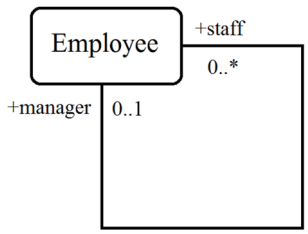
۱) نمودار جریان داده (DFD)

۲) نمودار تعامل (Interaction Diagram)

۳) نمودار مورد کاربرد (Use Case Diagram)

۴) نمودار کلاس (Class Diagram)

۲۰- کلاس Employee (عضو اداره) با نقش‌های Staff (کارمندان) و manager (مدیر) را در نظر بگیرید. شکل زیر چه رابطه‌ای را بیان می‌نماید؟ **(مهندسی IT – آزاد ۹۰)**



۱) Self Aggregation
۲) Message to Self
۳) Self Association
۴) تنها با ملاحظه شکل نمی‌توان نوع رابطه را بیان نمود.

۲۱- کدام‌یک از موارد زیر، کاربرد اصلی روش CRC (Class-Responsibility-Collaborator) است؟ **(مهندسی IT – دولتی ۹۲)**

۱) تشخیص کلاس‌ها
۲) تشخیص روابط توارث بین کلاس‌ها
۳) تشخیص روابط کل – جزء بین کلاس‌ها
۴) تشخیص ترتیب تبادل پیغام‌ها بین کلاس‌ها

۲۲- کدام‌یک از نمودارهای UML زیر، نوعاً شیءگرا (یعنی مبتنی بر مفاهیم خاص شیءگرایی) نیست؟ **(مهندسی IT – دولتی ۹۲)**

۱) نمودار مورد کاربرد (Use Case)
۲) نمودار ترتیب (Sequence)
۳) نمودار رده (Class)
۴) نمودار ارتباط (Communication)

۲۳- کدام‌یک از گروه Diagramهای زیر برای OOA صحیح نیست؟ **(مهندسی IT – دولتی ۹۳)**

۱) Class Diagram, Business Use Case
۲) Sequence Diagram, Use Case Diagram, CRC
۳) Flow-Oriented Diagram, Deployment Diagram
۴) Activity Diagram, Swimlane Diagram, Object Diagram

۲۴- بکارگیری مدل ساختاری (structural model) برای نمایش طراحی معماری دارای کدام مشخصات زیر است؟ **(مهندسی IT – دولتی ۹۴)**

۱) یک مدل ساختاری عناصر تکراری را در کاربردهای مشابه نمایش می‌دهد.
۲) یک مدل ساختاری نمایش ساختار واحدها و مولفه‌های برنامه است.
۳) یک مدل ساختاری رفتار سیستم را در قبال رخدادها نشان می‌دهد.
۴) مورد ۱ و ۲ صحیح است.

۲۵- کدام نمودار UML، برای مدل‌سازی تعامل اشیاء به کار می‌رود؟ **(مهندسی IT – دولتی ۹۶)**

۱) حالت (State Diagram)
۲) مولفه (Component Diagram)
۳) فعالیت (Activity Diagram)
۴) توالی (Sequence Diagram)

پاسخ تست‌های فصل هفتم

۱- گزینه (۳) صحیح است.

Use Case Diagram یا نمودار مورد کاربرد، یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد.

Use Caseها و Actorها محدوده سیستم در حال ساخت را مشخص می‌کنند. Use Case شامل تمام آن چیزهایی است که درون سیستم قرار دارد و Actor شامل تمام آن چیزهایی است که خارج از سیستم قرار دارد. هر فرد یا هر چیزی که با سیستم تعامل دارد، Actor یا بازیگر نامیده می‌شود. Use Caseها هر چیز موجود در داخل و محدوده سیستم را توصیف می کنند، در حالی که Actorها هر چیز موجود در خارج از محدوده سیستم را توصیف می‌کنند.

بازیگران، افراد و یا گاهی نرم‌افزار و یا سخت‌افزارهایی هستند که از سیستم استفاده می‌کنند و یا اطلاعاتی را برای سیستم فراهم می‌کنند. با اینکه همه بازیگران، عناصر خارجی سیستم هستند، اما با این حال هر عنصر خارج سیستم، بازیگر نیست. بازیگران استفاده کنندگان نهایی و یا تولیدکنندگان ابتدایی اطلاعات هستند، بنابراین عناصر خارجی سیستم که تنها وظیفه انتقال اطلاعات را دارند و نقش **رسانه‌انتقال** را ایفا می‌کنند، نمی‌توانند به عنوان بازیگر در نظر گرفته شوند.

برای مثال اگر یک سنسور از طریق رسانه بی‌سیم، اطلاعاتی را به سیستم مرکزی ارسال می‌کند، رسانه بی‌سیم نمی‌تواند بازیگر این سیستم باشد، بلکه سنسور بازیگر آن خواهد بود. بنابراین هر بازیگر اگر استفاده‌کننده باشد نقطه انتها و اگر تولیدکننده اطلاعات باشد نقطه ابتدای آن ارتباط است. بنابراین گزینه سوم نادرست است. زیرا خط تلفن، رسانه‌انتقال است. اما سایر گزینه‌ها به عنوان تولیدکننده اطلاعات یا مصرف‌کننده اطلاعات می‌توانند بازیگر در نظرگرفته شوند.

۲- گزینه (۲) صحیح است.

دو انسانی که همدیگر را می‌شناسند و امکان گفتگو و تبادل پیام با هم دارند، رابطه انجمنی با هم دارند. در رابطه انجمنی(Association Relationship) یک شیء بطور ساده درباره شیء دیگر می‌داند به همان طریقی که یک فرد ممکن است فرد دیگری را بشناسد. یک برنامه کامپیوتری شیءگرا از اجتماع تعدادی کلاس ایجاد شده است، کلاس‌های همکار بدون رابطه جزء و کل و هم‌هدف در یک بخش مشترک از برنامه، کلاس‌های همکار با رابطه جزء و کل و هم‌هدف در یک بخش مشترک از برنامه و کلاس‌های غیرهمکار در دو بخش مختلف از برنامه، رابطه‌ای که میان کلاس‌های همکار بدون رابطه جزء و کل وجود دارد، رابطه انجمنی است.

کلاس‌های همکار بدون رابطه جزء و کل، جهت انجام وظایف خود از طریق مکانیزم پیام به گفتگو با یکدیگر می‌پردازند. در یک بیان ساده، هرگاه میان دو شیء بدون رابطه جزء و کل گفتگو

باشد، کلاس‌های این دو شیء هم باهم همکار هستند و هم رابطه انجمنی میان آنها برقرار است.

رابطه انجمنی به عنوان رابطه «has knowledge of» نیز شناخته می‌شود.

هر رابطه انجمنی از سه قسمت اصلی تشکیل می‌شود که عبارتند از:

۱) کلاس‌های شرکت‌کننده در رابطه

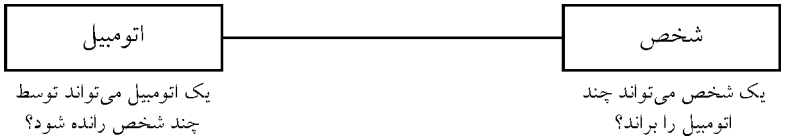
۲) خط ممتد رابطه که بین دو کلاس ترسیم می‌شود.

۳) نام رابطه که در وسط خط رابطه نوشته می‌شود و بیان‌کننده هدف رابطه است.

در مورد روابط انجمنی، بین کلاس‌ها، سوالات مهمی مطرح می‌شود. برای مثال، در مورد رابطه انجمنی شخص و اتومبیل، سوالات زیر مطرح است:

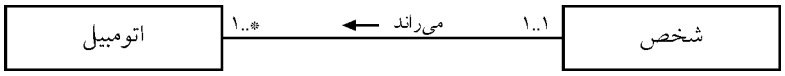
«یک شخص می‌تواند چند اتومبیل را براند؟»، «پاسخ: *..۱»

«یک اتومبیل می‌تواند توسط چند شخص رانده شود؟»، «پاسخ: *..۱»



UML برای پاسخگویی به چنین سوالاتی، مشخصه‌ای با نام **تعدد** (Multiplicity) را برای هر یک از کلاس‌های طرفین رابطه ارائه نموده است. تعدد اصطلاحی است که تعداد اشیای شرکت‌کننده در رابطه انجمنی را مشخص می‌کند. اعدادی که در پاسخ به این دو سوال داده می‌شود، مبین تعدد طرفین است. شیوه‌های مختلفی جهت تعیین تعدد وجود دارد، اما متداول‌ترین شیوه، بیان محدوده تعداد اشیای شرکت‌کننده در هر طرف رابطه است که به صورت «حداقل .. حداکثر» نشان داده می‌شود.

مثال: تعدد رابطه انجمنی بین شخص و اتومبیل به صورت زیر است:



مطابق نمودار مطرح شده در صورت سوال، رابطه کلاس A و B ، همچنین رابطه کلاس C و G و همچنین رابطه کلاس C و F از نوع رابطه انجمنی است.

همچنین یک شیء از نوع C حداقل با یک و حداکثر با هفت شیء ازنوع F رابطه دارد.

به طورکلی هرگاه یک کلاس (ب)، **از نوع** یک کلاس (الف) باشد، گوییم بین دو کلاس مذکور رابطه **وراثت** برقرار است. در این صورت کلاس (ب) فرزند و کلاس (الف) پدر یا والد نامیده می‌شود. به عبارت دیگر همانطور که پیش از این نیز گفتیم، وراثت فرآیندی است که به وسیله آن یک کلاس (فرزند) می‌تواند صفات و متدهای کلاس دیگری (پدر) را کسب کند. به

عبارت کلی‌تر یک کلاس فرزند ضمن بـه ارث بـردن مجموعـه‌ای از صـفات و متـدهای عمـومی کلاس پدر، می‌تواند ویژگی‌های خاص و مختص خود را نیز به آنها اضافه کنـد. در رابطـه وراثـت هر تغییر در کلاس پدر بر کلاس فرزند نیز اثر می‌گذارد اما عکس این مطلب برقرار نیست.

رابطه وراثت به عنوان رابطه «is a» یا «kind of» یا «type of» نیز شناخته می‌شود.

مراحل لازم جهت مدل‌سازی یک رابطه ارث بری عبارتند از:

۱- ترسیم یک خط ممتد بین کلاس فرزند و کلاس پدر

۲- درج یک مثلث تو خالی در انتهایی از خط ممتد که کلاس پدر قرار دارد.

ذکر «is a» بر روی خط ممتد رابطه وراثت اختیاری است.

در تعریف رابطه وراثت، مشخصاتی نظیر تعدد و محدودیت وجود ندارد.

رابطه کلاس D و B از نوع رابطه ارث‌بری است، بدین نحو که کلاس D فرزند کلاس B است. بنابراین کلاس D نوعی از کلاس B است.

رابطه کلاس E و B از نوع رابطه ارث‌بری است، بدین نحو که کلاس E فرزند کلاس B است. بنابراین کلاس E نوعی از کلاس B است.

رابطه کلاس C و E از نوع رابطه ارث‌بری است، بدین نحو که کلاس C فرزند کلاس E است. بنابراین کلاس C نوعی از کلاس E است.

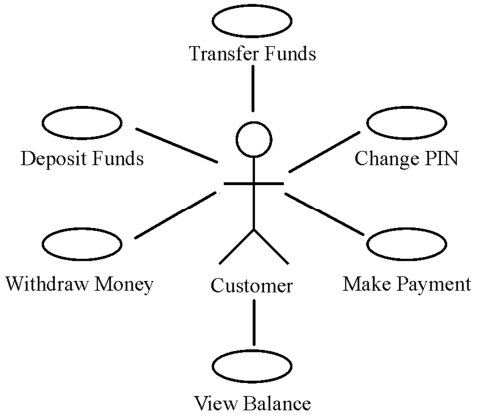
بنابر مطالب فوق واضح است که گزینه دوم درست اسـت و گزینـه‌هـای اول، سـوم و چهـارم نادرست هستند. سازمان سنجش آموزش کشور نیز گزینه دوم را به عنوان پاسخ درست اعلام کرده بود.

البته بهتر بود طراح محترم در گزینه دوم با توجه به استفاده از نماد ارث‌بری مابین کـلاس C و E در نمودار مطرح شده به جای عبارت «C بخشی از کلاس E است» عبارت «C نـوعی از کـلاس E است» را به کار می‌برد، زیرا رابطه بخشی یا انجمنی پیشرفته رابطـه‌ای مـابین یـک واحـد کـل و تعدادی واحد جزء است، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشـد. که خود بر دو نوع تجمع با نماد لوزی توخالی و ترکیب با نماد لوزی توپر است. البتـه خـود نمـاد ارث‌بری نشان‌داده شده در شکل صورت سوال نادرست است. زیرا در ارث‌بری نماد یک مثلث تو خالی در انتهایی از خط ممتد که کلاس پدر قرار دارد، درج می‌گردد. البته شاید هم طراح محترم و یا تایپیست محترم در نشانه‌گذاری و انتخاب نماد مناسب دچار خطا شده‌اند. یعنـی اگـر در رابطـه بین C و E از نماد لوزی توخالی یا توپر استفاده می‌شد، آنگاه گزینه دوم بدون اصلاح و بـه همـین شکل مطرح شده درست می‌بود. تستی که تا ابد راز نهفته در آن کشف نخواهد شد!

۳- گزینه (۱) صحیح است.

Use Case Diagram یـا نمـودار مـورد کـاربرد، یـا نمـودار نیـاز جهـت مـدل‌سـازی لیسـت نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد.

مثال: مدل‌سازی لیست نیازمندی‌های مشتری برای سیستم ATM.



در نمودار فوق، لیست نیازمندی‌های مشتری، مدل‌سازی شده است.

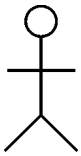
به هر یک از نیازهای فوق یک use case یا مورد کاربرد گفته می‌شود و به اجتماع این use caseها، Use Case Diagram یا نمودار مورد کاربرد گفته می‌شود.

Use Caseها و Actorها محدوده سیستم در حال ساخت را مشخص می‌کنند. Use Case شامل تمام آن چیزهایی است که درون سیستم قرار دارد و Actor شامل تمام آن چیزهایی است که خارج از سیستم قرار دارد. هر فرد یا هر چیزی که با سیستم تعامل دارد، Actor یا بازیگر نامیده می‌شود. Use Caseها هر چیز موجود در داخل و محدوده سیستم را توصیف می‌کنند، در حالی که Actorها هر چیز موجود در خارج از محدوده سیستم را توصیف می‌کنند.

بازیگران، افراد و یا گاهی نرم‌افزار و یا سخت‌افزارهایی هستند که از سیستم استفاده می‌کنند و یا اطلاعاتی را برای سیستم فراهم می‌کنند. با اینکه همه بازیگران، عناصر خارجی سیستم هستند، اما با این حال هر عنصر خارج سیستم، بازیگر نیست. بازیگران استفاده کنندگان نهایی و یا تولیدکنندگان ابتدایی اطلاعات هستند، بنابراین عناصر خارجی سیستم که تنها وظیفه انتقال اطلاعات را دارند و نقش **رسانه انتقال** را ایفا می‌کنند، نمی‌توانند به عنوان بازیگر در نظر گرفته شوند.

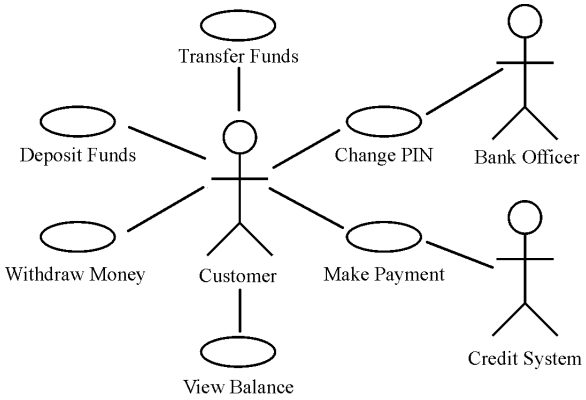
برای مثال اگر یک سنسور از طریق رسانه بی‌سیم، اطلاعاتی را به سیستم مرکزی ارسال می‌کند، رسانه بی‌سیم نمی‌تواند بازیگر این سیستم باشد، بلکه سنسور بازیگر آن خواهد بود. بنابراین هر بازیگر اگر استفاده‌کننده باشد نقطه انتها و اگر تولیدکننده اطلاعات باشد نقطه ابتدای آن ارتباط است.

در UML، بازیگران با آدمک‌هایی به شکل زیر نشان داده می‌شوند:



Actor

نمونه‌ای از استفاده نمودار Use Case در شکل زیر نشان داده شده است: (سیستم ATM)



Sequence Diagram یا نمودار توالی و Collaboration Diagram یا نمودار همکاری جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرند. بنابراین گزینه دوم و چهارم نادرست هستند.

Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط شنا، جهت مدل‌سازی سناریوی اصلی و فرعی داخل یک use case مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار فعالیت یا نمودار خط شنا، جهت مدل‌سازی روال انجام کارها داخل یک use case مورد استفاده قرار می‌گیرد. همچنین، در مدل طراحی در بخش طراحی مؤلفه، باید جزئیات الگوریتمی متدهای کلاس تشریح شود. برای این منظور Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط‌شنا مورد استفاده قرار می‌گیرد. بنابراین گزینه سوم نیز نادرست است.

۴- گزینه (۱) صحیح است.

استخراج نیازمندی‌های مشتری در شیءگرایی، بر اساس موارد کاربرد (use case) است، که سرویس‌های سیستم را از دید عوامل خارج سیستم منعکس می کنند.

۵- گزینه (۳) صحیح است.

دو انسانی که همدیگر را می‌شناسند و امکان گفتگو و تبادل پیام با هم دارند، رابطه انجمنی با

هم دارند. در رابطه انجمنی یک شیء بطور ساده درباره شیء دیگر می‌داند به همان طریقی که یک فرد ممکن است فرد دیگری را بشناسد. یک برنامه کامپیوتری شیءگرا از اجتماع تعدادی کلاس ایجاد شده است، کلاس‌های همکار بدون رابطه جزء و کل **(بدون تلفیق تعدادی کلاس)** و هم‌هدف در یک بخش مشترک از برنامه، کلاس‌های همکار با رابطه جزء و کل **(با تلفیق تعدادی کلاس)** و هم‌هدف در یک بخش مشترک از برنامه و کلاس‌های غیرهمکار در دو بخش مختلف از برنامه، رابطه‌ای که میان کلاس‌های همکار بدون رابطه جزء و کل وجود دارد، رابطه انجمنی است.

کلاس‌های همکار بدون رابطه جزء و کل، جهت انجام وظایف خود از طریق مکانیزم پیام به گفتگو با یکدیگر می‌پردازند. در یک بیان ساده، هرگاه میان دو شیء بدون رابطه جزء و کل **(بدون تلفیق تعدادی کلاس)** گفتگو باشد، کلاس‌های این دو شیء هم باهم همکار هستند و هم رابطه انجمنی میان آنها برقرار است. بنابراین گزینه دوم نادرست است.

رابطه انجمنی پیشرفته به دو رابطه تجمع و ترکیب تقسیم می‌گردد.

رابطه تجمع (Aggregation Relationship) رابطه‌ای مابین یک واحد کل و تعدادی واحد جزء است، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشد. **(با تلفیق تعدادی کلاس)**

در رابطه تجمع، حیات شیء جزء به حیات شیء کل وابسته نیست. به بیان دیگر حیات شیء جزء و شیء کل به هم تقدم و تاخر دارند. یعنی ممکن است شیء جزء موجود باشد، بدون اینکه شیء کل ایجاد گردد و موجود باشد. در رابطه تجمع اگر شیء کل از بین برود، اشیاء جزء، همچنان به حیات خود ادامه می‌دهند.

در رابطه **انجمنی‌ساده**، مابین طرفین همکار رابطه، رابطه جزء و کل مطرح نیست، در حالی که در رابطه **تجمع**، مابین طرفین همکار رابطه، رابطه جزء و کل مطرح است و از **تجمع** و سازماندهی اجزاء، یک موجودیت کامل‌تر ساخته می‌شود.

رابطه ترکیب (Composition Relationship) رابطه‌ای مابین یک واحد کل و تعدادی واحد جزء است، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشد. **(با تلفیق تعدادی کلاس)** به نحوی که شیء کل و جزء با هم ایجاد شوند و با هم از بین بروند.

در رابطه ترکیب، حیات شیء جزء به حیات شیء کل وابسته است. به بیان دیگر حیات شیء جزء و شیء کل به هم تقدم و تاخر ندارند. یعنی امکان ندارد شیء جزء موجود باشد، بدون اینکه شیء کل ایجاد گردد و موجود باشد. در رابطه ترکیب اگر شیء کل از بین برود، اشیاء جزء نیز به طور همزمان با شیء کل از بین می‌روند. زیرا از ابتدای کار، شیء کل، به طور همزمان از ایجاد و کنار هم قرار دادن اشیاء جزء توسط برنامه کامپیوتری ایجاد شده است و با پایان یافتن فعالیت شیء کل، شیء کل و جزء باهم و همزمان از بین می‌روند.

در رابطه **انجمنی‌ساده**، مابین طرفین همکار رابطه، رابطه جزء و کل مطرح نیست، در حالی که

در رابطه **ترکیب**، مابین طرفین همکار رابطه، رابطه جزء و کل مطرح است و از **ترکیب** و سازماندهی اجزاء، یک موجودیت کامل‌تر ساخته می‌شود.

از آنجا که در گزینه‌ها رابطه ترکیب مطرح نشده است، آنرا کنار می‌گذاریم. بنابراین گزینه سوم درست است.

به طورکلی هرگاه یک کلاس (ب)، **از نوع** یک کلاس (الف) باشد، گوییم بین دو کلاس مذکور رابطه **وراثت** برقرار است. در این صورت کلاس (ب) فرزند و کلاس (الف) پدر یا والد نامیده می‌شود. به عبارت دیگر، وراثت فرآیندی است که به وسیله آن یک کلاس (فرزند) می‌تواند صفات و متدهای کلاس دیگری (پدر) را کسب کند. به عبارت کلی‌تر یک کلاس فرزند ضمن به ارث بردن مجموعه‌ای از صفات و متدهای عمومی کلاس پدر، می‌تواند ویژگی‌های خاص و مختص خود را نیز به آنها اضافه کند. بنابراین گزینه اول نادرست است.

۶- گزینه (۳) صحیح است.

مدل‌سازی تعاملات پویای میان اشیاء همکار یا مدل‌سازی رفتاری، رفتار سیستم را در زمان اجرا نمایش می‌دهد. به بیان دیگر در این دیدگاه رفتارهای پویای سیستم مدل می‌شود.

مدل‌سازی رفتاری سیستم با استفاده از نمودارهای UML قابل انجام است:

الف) نمودار توالی (Sequence Diagram)

نمودار کلاس ساختار ایستای داخل یک use case را نمایش می‌دهد. در یک سیستم در حال کار، به هر حال اشیاء با یکدیگر در تعامل هستند و این تعامل در طی زمان رخ می‌دهد. به بیان دیگر برای آنکه یک use case یا مورد کاربرد یا نیاز مرتفع گردد، باید مجموعه‌ای از اشیاء با یکدیگر ارتباط برقرار کرده و پیام‌هایی را با یکدیگر رد و بدل نمایند. نمودار توالی نشان می‌دهد، برای مرتفع شدن یک use case یا مورد کاربرد یا نیاز، چه اشیایی باید چه پیام‌هایی را با چه ترتیبی ارسال کنند تا آن نیاز برآورده گردد. نمودار توالی، تعاملات پویا مابین اشیاء را براساس زمان نشان می‌دهد.

Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ب) نمودار همکاری (Collaboration Diagram)

Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

Collaboration Diagram یا نمودار همکاری، برای یک کاربرد خاص، جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

این نمودار نیز، مانند نمودار توالی، تعاملات بین اشیاء یک مورد کاربرد را نشان می‌دهد.

ج) نمودار حالت (State Transition Diagram)

State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد. بنابراین گزینه دوم نادرست است.

همه نمودارهای فوق(توالی، همکاری و حالت)، مدل‌سازی رفتاری محسوب می‌گردند.

بنابر مطالب فوق Sequence Diagram یا نمودار توالی و Collaboration Diagram یا نمودار همکاری جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرند. از آنجاکه Collaboration Diagram یا نمودار همکاری در گزینه‌ها وجود ندارد، پس آنرا کنار می‌گذاریم، بنابراین گزینه سوم درست است.

Class Diagram یا نمودار کلاس جهت مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل یک case use مورد استفاده قرار می‌گیرد. بنابراین گزینه اول نادرست است.

Component Diagram یا نمودار مؤلفه جهت مدل‌سازی ساختار کلی پیاده‌سازی برنامه و ترتیب کامپایل مؤلفه‌های فرعی، برای ایجاد مؤلفه‌اصلی (مؤلفه کلی یا برنامه اصلی) مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار مؤلفه یک دید فیزیکی از مدل سیستم به همراه مؤلفه‌های فرعی نرم‌افزار و روابط بین آنها را نشان می‌دهد. بنابراین گزینه چهارم نادرست است.

۷- گزینه (۲) صحیح است.

مدل‌سازی تعاملات پویای میان **اشیاء همکار** یا مدل‌سازی رفتاری، رفتار سیستم را در زمان اجرا نمایش می‌دهد. به بیان دیگر در این دیدگاه رفتارهای پویای سیستم مدل می‌شود.

مدل‌سازی رفتاری سیستم با استفاده از نمودارهای UML قابل انجام است:

الف) نمودار توالی (Sequence Diagram)

نمودار کلاس ساختار ایستای داخل یک use case را نمایش می‌دهد. در یک سیستم در حال کار، به هر حال اشیاء با یکدیگر در تعامل هستند و این تعامل در طی زمان رخ می‌دهد. به بیان دیگر برای آنکه یک use case یا مورد کاربرد یا نیاز مرتفع گردد، باید مجموعه‌ای از اشیاء با یکدیگر ارتباط برقرار کرده و پیام‌هایی را با یکدیگر رد و بدل نمایند. نمودار توالی نشان می‌دهد، برای مرتفع شدن یک use case یا مورد کاربرد یا نیاز، چه اشیایی باید چه پیام‌هایی را با چه ترتیبی ارسال کنند تا آن نیاز برآورده گردد. نمودار توالی، تعاملات پویا مابین اشیاء همکار را براساس زمان نشان می‌دهد.

Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ب) نمودار همکاری (Collaboration Diagram)

Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

Collaboration Diagram یا نمودار همکاری، برای یک کاربرد خاص، جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

این نمودار نیز، مانند نمودار توالی، تعاملات بین اشیاء یک مورد کاربرد را نشان می‌دهد.

ج) نمودار حالت (State Transition Diagram)

State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد.

همه نمودارهای فوق (توالی، همکاری و حالت)، مدل‌سازی رفتاری محسوب می‌گردند.

بنابر مطالب فوق Sequence Diagram یا نمودار توالی و Collaboration Diagram یا نمودار همکاری جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرند. بنابراین گزینه دوم درست است.

Use Case Diagram یا نمودار مورد کاربرد، یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد. بنابراین گزینه اول نادرست است.

Class Diagram یا نمودار کلاس جهت مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل یک case use مورد استفاده قرار می‌گیرد. بنابراین گزینه سوم نیز نادرست است.

بسته (Package) مجموعه‌ای از کلاس‌های پیاده‌سازی‌شده و کامپایل‌شده به عنوان واحد مولفه و یا تعدادی مولفه فرعی پیاده‌سازی‌شده و کامپایل‌شده است. که از اجتماع بسته‌ها یک سیستم نرم‌افزاری ایجاد می‌گردد. بنابراین گزینه چهارم نیز نادرست است.

۸- گزینه (۳) صحیح است.

مدل‌سازی تعاملات پویای میان **اشیاء همکار** یا مدل‌سازی رفتاری، رفتار سیستم را در زمان اجرا نمایش می‌دهد. به بیان دیگر در این دیدگاه رفتارهای پویای سیستم مدل می‌شود.

مدل‌سازی رفتاری سیستم با استفاده از نمودارهای UML قابل انجام است:

الف) نمودار توالی (Sequence Diagram)

نمودار کلاس ساختار ایستای داخل یک use case را نمایش می‌دهد. در یک سیستم در حال کار، به هر حال اشیاء با یکدیگر در تعامل هستند و این تعامل در طی زمان رخ می‌دهد. به بیان دیگر برای آنکه یک use case یا مورد کاربرد یا نیاز مرتفع گردد، باید مجموعه‌ای از اشیاء با یکدیگر ارتباط برقرار کرده و پیام‌هایی را با یکدیگر رد و بدل نمایند. نمودار توالی نشان می‌دهد،

برای مرتفع شدن یک use case یا مورد کاربرد یا نیاز، چه اشیایی باید چه پیام‌هایی را با چه ترتیبی ارسال کنند تا آن نیاز برآورده گردد. نمودار توالی، تعاملات پویا مابین اشیاء همکار را براساس زمان نشان می‌دهد.

Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ب) نمودار همکاری (Collaboration Diagram)

Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

Collaboration Diagram یا نمودار همکاری، برای یک کاربرد خاص، جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

این نمودار نیز، مانند نمودار توالی، تعاملات بین اشیاء یک مورد کاربرد را نشان می‌دهد.

ج) نمودار حالت (State Transition Diagram)

State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل Sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد.

همه نمودارهای فوق (توالی، همکاری و حالت)، مدل‌سازی رفتاری محسوب می‌گردند.

بنابر مطالب فوق Sequence Diagram یا نمودار توالی و Collaboration Diagram یا نمودار همکاری جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرند.

از آنجا که وقوع رخداد در نمودارهای توالی و همکاری (ارتباط) نقشی ندارد، به نمودارهای رفتاری توالی و همکاری، مدل‌سازی رفتاری ایستا و به نمودار حالت، مدل‌سازی رفتاری پویا نیز گفته می‌شود. بنابراین گزینه سوم درست است.

Use Case Diagram یا نمودار مورد کاربرد، یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد. بنابراین گزینه‌های اول، دوم و چهارم نادرست هستند.

۹- گزینه (۳) صحیح است.

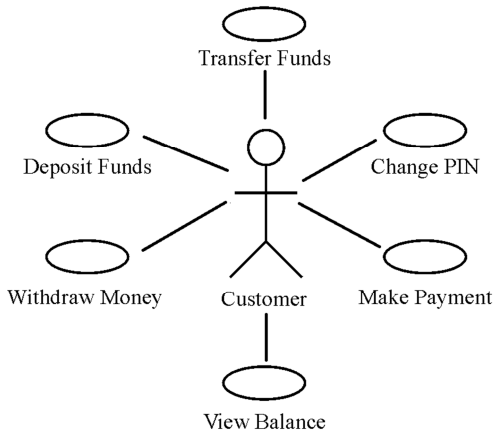
State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل Sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار انتقال حالت، چرخه حیات یک شیء را در حالت‌های مختلف (از زمانی که شیء ایجاد می‌شود تا زمانی که شیء از بین می‌رود) نمایش

می‌دهد. در واقع این نمودار تمامی حالت‌های مختلفی را که یک شیء در طول حیات خود می‌تواند داشته باشد و همچنین نحوه انتقال بین حالت‌ها و وقایع باعث‌شونده این انتقال را نشان می‌دهد. به بیان دیگر نمودارهای حالت، راهی را آماده می‌کنند تا حالت‌های مختلف یک شیء را مدل کنند. نمودارهای حالت استفاده می‌شوند تا بیشتر، **رفتارهای پویای** یک سیستم را نمایش دهند.

به آنچه که موجب تغییر حالت یک شیء می‌گردد، رخداد(event) گفته می‌شود. بنابراین گزینه سوم درست است.

Use Case Diagram یا نمودار مورد کاربرد، یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد.

مثال: مدل‌سازی لیست نیازمندی‌های مشتری برای سیستم ATM.



در نمودار فوق، لیست نیازمندی‌های مشتری، مدل‌سازی شده است.

به هر یک از نیازهای فوق یک use case یا مورد کاربرد گفته می‌شود و به اجتماع این use caseها، Use Case Diagram یا نمودار مورد کاربرد گفته می‌شود.

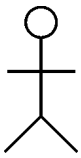
Use Caseها و Actorها محدوده سیستم در حال ساخت را مشخص می‌کنند. Use Case شامل تمام آن چیزهایی است که درون سیستم قرار دارد و Actor شامل تمام آن چیزهایی است که خارج از سیستم قرار دارد. هر فرد یا هر چیزی که با سیستم تعامل دارد، Actor یا بازیگر نامیده می‌شود. Use Caseها هر چیز موجود در داخل و محدوده سیستم را توصیف می‌کنند، در حالی که Actorها هر چیز موجود در خارج از محدوده سیستم را توصیف می‌کنند.

بازیگران، افراد و یا گاهی نرم‌افزار و یا سخت‌افزارهایی هستند که از سیستم استفاده می‌کنند و یا اطلاعاتی را برای سیستم فراهم می‌کنند. با اینکه همه بازیگران، عناصر خارجی سیستم هستند، اما با این حال هر عنصر خارج سیستم، بازیگر نیست. بازیگران استفاده کنندگان نهایی و یا

تولیدکنندگان ابتدایی اطلاعات هستند، بنابراین عناصر خارجی سیستم که تنها وظیفه انتقال اطلاعات را دارند و نقش **رسانه انتقال** را ایفا می‌کنند، نمی‌توانند به عنوان بازیگر در نظر گرفته شوند.

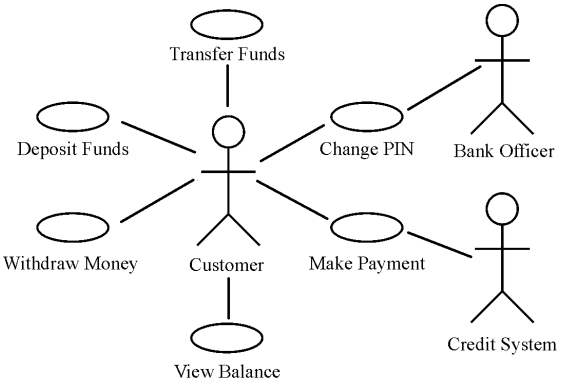
برای مثال اگر یک سنسور از طریق رسانه بی‌سیم، اطلاعاتی را به سیستم مرکزی ارسال می‌کند، رسانه بی‌سیم نمی‌تواند بازیگر این سیستم باشد، بلکه سنسور بازیگر آن خواهد بود. بنابراین هر بازیگر اگر استفاده‌کننده باشد نقطه انتها و اگر تولیدکننده اطلاعات باشد نقطه ابتدای آن ارتباط است.

در UML، بازیگران با آدمک‌هایی به شکل زیر نشان داده می‌شوند:



Actor

نمونه‌ای از استفاده نمودار Use Case در شکل زیر نشان داده شده است: (سیستم ATM)



بنابر مطالب فوق گزینه اول نادرست است.

کلاس‌ها یا به تنهایی از عهده انجام مسئولیت‌های خود بر می‌آیند و یا از طریق همکاری با کلاس‌های همکار(collaborator) خود از عهده انجام مسئولیت‌های خود بر می‌آیند. بنابراین اگر کلاسی همکارانی دارد، باید در بخش مربوط به همکاران نوشته شود. بنابراین گزینه دوم نادرست است.

مؤلفه یک قطعه‌ی آماده، کاربردی و پیاده‌سازی شده است که دارای واسط لازم جهت اتصال با سایر قطعات و استقرار در بخشی از یک سیستم عملیاتی می‌باشد. از آنجایی که در یک مولفه، فقط بخشی از کدها یا داده‌های نرم‌افزار قرار می‌گیرد، لذا هر مولفه ممکن است نیازمند ارتباط با

مولفه‌های دیگر باشد تا بتواند از کدها یا داده‌های موجود در آنها استفاده نماید. برای انجام این ارتباط، در مولفه‌ها یک واسط در نظر گرفته می‌شود تا کدها یا داده‌های یک مولفه، از طریق واسط آن در اختیار مولفه‌های دیگر قرار گیرد، به این معنی که ارتباط بین مولفه‌ها، فقط از طریق واسط‌های آنها انجام می‌گیرد. در برنامه‌نویسی شیءگرا، متدهای عمومی کلاس‌ها، نقش واسط را ایفا می‌کنند. به عبارت دیگر کلاس‌های همکار، از طریق متدهای عمومی یکدیگر اقدام به گفتگو و تبادل پیام می‌کنند.

در دیدگاه شیءگرا به مؤلفه پیمانه نیز گفته می‌شود. در حیطه مهندسی نرم‌افزار شیءگرا، **واحد مؤلفه** یک قطعه‌ی عملیاتی مبتنی بر کلاس است که بر دو نوع می‌باشد:

۱) کلاس کاربردی: مانند کلاس دانشجو که برنامه‌نویس آن را می‌نویسد و به دلیل داشتن شرایط قابل حمل به شکل ذاتی، می‌تواند به عنوان یک قطعه‌ی آماده و قابل استفاده‌ی مجدد در پروژه‌های بعدی مورد استفاده مجدد قرار گیرد.

۲) کلاس سیستمی: مانند کلاس جعبه‌ی متن که کامپایلر تعاریف آن را فراهم می‌کند و می‌تواند به عنوان یک قطعه‌ی آماده و قابل استفاده‌ی مجدد، در پروژه‌ها مورد استفاده مجدد قرار گیرد. بنابراین گزینه چهارم نادرست است.

۱۰- گزینه (۴) صحیح است.

Sequence Diagram یا نمودار توالی جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد. بنابراین برای رسم نمودار توالی ابتدا باید کلاس‌های سیستم شناسایی شوند.

Class Diagram یا نمودار کلاس جهت مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل یک case use مورد استفاده قرار می‌گیرد.

پس از شناسایی موارد کاربرد و سناریونویسی برای هر یک از موارد کاربرد، زمان تعریف کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌های همکار برای هر یک از موارد کاربرد می‌رسد. برای شناسایی کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌ها برای هر یک از موارد کاربرد، از تکنیکی موسوم به CRC که سرواژه‌ی عبارت Class – Responsibility Collaborator و به معنی «مدل همکاری مسئولیت‌های کلاس‌ها» می‌باشد، استفاده می‌شود. مدل‌سازی CRC روشی ساده جهت تعیین و سازماندهی کلاس‌های داخل هر مورد کاربرد است. در این روش به هر کلاس یک کارت CRC اختصاص داده می‌شود که شامل سه بخش کلی زیر است:

نام کلاس

مسئولیت‌های کلاس(Responsibilities)

- صفات کلاس
- متدهای کلاس

همکاران کلاس (Collaborators)

کلاس‌ها یا به تنهایی از عهده انجام مسئولیت‌های خود بر می‌آیند و یا از طریق همکاری با کلاس‌های همکار خود از عهده انجام مسئولیت‌های خود بر می‌آیند. بنابراین اگر کلاسی همکارانی دارد، باید در بخش مربوط به همکاران نوشته شود.

برای کشف کلاس‌های همکار داخل هر use case، از سناریوی اصلی (نوشتاری یا نموداری) هر use case استفاده می‌گردد. برای این منظور، **اسامی** موجود داخل هر use case مورد جستجو قرار می‌گیرند. از آنجا که نیازها یا موارد کاربرد مشتری به تدریج و در طی تکرار مشخص می‌شوند و همچنین از آنجا که کلاس‌های همکار داخل هر مورد کاربرد یا نیاز هستند، بنابراین با تکامل و کامل شدن نیازها یا موارد کاربرد، به تبع کلاس‌های همکار هر مورد کاربرد نیز کامل می‌شود. بنابراین تشخیص تمام کلاس‌های برنامه، در همان ابتدای کار تقریبا غیر ممکن است، در واقع در طول پروژه و با گذشت زمان تحلیل‌گر متوجه نیازها و به تبع کلاس‌های جدیدی می‌شود که در ابتدای کار نیاز به آن‌ها چندان محسوس نبوده است. بنابراین می‌توان گفت روند تشخیص نیازها یا موارد کاربرد و به تبع کلاس‌های همکار به عنوان مرتفع‌کننده نیازها یا موارد کاربرد، یک فرآیند تکرارشونده است و در طول فرآیند تولید نرم‌افزار کامل و کامل‌تر می‌شوند. بنابراین گزینه چهارم درست است و گزینه‌های اول، دوم و سوم نادرست هستند.

۱۱- گزینه (۲) صحیح است.

Use Case Diagram یا نمودار مورد کاربرد، یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی های مشتری مورد استفاده قرار می‌گیرد. بنابراین گزینه اول نادرست است.

Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط شنا، جهت مدل سازی سناریوی اصلی و فرعی داخل یک use case **(نیاز یا مورد کاربرد یا زیرسیستم)** مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار فعالیت یا نمودار خط شنا، جهت مدل‌سازی روال انجام کارها داخل یک use case مورد استفاده قرار می‌گیرد. همچنین، در مدل طراحی در بخش طراحی مؤلفه، باید جزئیات الگوریتمی **متدهای کلاس** تشریح شود. برای این منظور Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط‌شنا مورد استفاده قرار می‌گیرد. بنابراین گزینه دوم درست است.

Class Diagram یا نمودار کلاس جهت مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل یک use case مورد استفاده قرار می‌گیرد. بنابراین گزینه سوم نادرست است.

Sequence Diagram یا نمودار توالی و Collaboration Diagram یا نمودار همکاری جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرند. بنابراین گزینه چهارم نادرست است.

۱۲- گزینه (۲) صحیح است.

Sequence Diagram یا نمودار توالی و Collaboration Diagram یا نمودار همکاری جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرند. بنابراین گزینه اول نادرست است.

اطلاعات به دست آمده از نمودار حالت در مدل تحلیل مربوط به کلاس‌های دارای حالات مختلف، در مدل طراحی، بخش طراحی مولفه، جهت تکمیل نمودن الگوریتم‌های مربوط به متدهای کلاس‌های دارای حالات مختلف مورد استفاده قرار می‌گیرد. اینکه متدهای یک کلاس دارای حالات مختلف در سطح طراحی مولفه در مواجه با حالت‌های مختلف یک شیء چگونه رفتار کنند از نمودار حالت استنباط می‌گردد. بنابراین گزینه دوم درست است.

Component Diagram یا نمودار مؤلفه جهت مدل‌سازی ساختار کلی پیاده‌سازی برنامه و ترتیب کامپایل مؤلفه‌های فرعی، برای ایجاد مؤلفه‌اصلی (مؤلفه کلی یا برنامه اصلی) مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار مؤلفه یک دید فیزیکی از مدل سیستم به همراه مؤلفه‌های فرعی نرم‌افزار و روابط بین آنها را نشان می‌دهد. بنابراین گزینه سوم نادرست است.

Deployment Diagram یا نمودار استقرار، جهت مدل‌سازی گره‌های سخت افزاری برای نصب نرم‌افزار مربوطه در فعالیت استقرار مورد استفاده قرار می‌گیرد. بنابراین گزینه چهارم نادرست است.

۱۳- گزینه (۲) صحیح است.

ترتیب ایجاد مولفه‌ها براساس رویکرد تکرار و تکامل نرم‌افزار توسط برنامه‌نویسان مشخص می‌شود. بنابراین گزینه اول نادرست است.

Component Diagram یا نمودار مؤلفه به نحوه پیاده‌سازی فیزیکی نرم‌افزار می‌پردازد. در واقع، نمودار مولفه شامل کدهای نرم‌افزار است که در آن هر قطعه کد در قالب یک مولفه، دسته‌بندی می‌شود. کدها نیز چیزی جز کلاس‌های سیستم، در زمانی که جزئیات پیاده‌سازی به آن‌ها اضافه شده باشد، نیست.

Component Diagram یا نمودار مؤلفه جهت مدل‌سازی ساختار کلی پیاده‌سازی برنامه و ترتیب کامپایل مؤلفه‌های فرعی، برای ایجاد مؤلفه‌اصلی (مؤلفه کلی یا برنامه اصلی) مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار مؤلفه یک دید فیزیکی از مدل سیستم به همراه مؤلفه‌های فرعی نرم‌افزار و روابط بین آنها را نشان می‌دهد.

نمودار مؤلفه زمانی که تولید کد تمام شده است ایجاد می‌گردد. بنابراین در اینجا منظور از مؤلفه فرعی یک مولفه فیزیکی کد است. به بیان دیگر مؤلفه فرعی در این مرحله، کلاس‌های همکار پیاده‌سازی شده داخل یک use case یا مورد کاربرد یا نیاز پیاده‌سازی شده است. دقت کنید که قبل از استفاده از نمودار مولفه هریک از کلاس‌های همکار موجود در نمودار کلاس مربوط به

هر use case یا مورد کاربرد یا نیاز باید به یک مؤلفه فرعی حاوی کد منبع تبدیل شود.

در این مرحله هر use case یا مورد کاربرد یا نیاز، مطابق روالی که از فعالیت‌های ارتباط تا ساخت (پیاده‌سازی) طی می‌کند، سرانجام توسط کلاس‌های همکار پیاده‌سازی شده خود به یک مؤلفه فرعی (مؤلفه‌بخشی) پیاده‌سازی شده و کامپایل شده تبدیل می‌گردد، که از اجتماع این مؤلفه‌های فرعی (مؤلفه‌بخشی)، مؤلفه اصلی (مؤلفه کلی یا برنامه اصلی) خلق می‌گردد. و محصول نهایی آماده می‌گردد.

تنها نوع رابطه‌ای که می‌تواند بین مؤلفه‌ها در نمودار مولفه وجود داشته باشد، یک رابطه وابستگی (Dependency) است و به این معنی است که یک مؤلفه باید قبل از کدام مولفه دیگر کامپایل شود. بنابراین گزینه دوم درست است.

نگاشت کلاس‌ها به مولفه‌ها قبل از استفاده از نمودار مولفه توسط برنامه‌نویسان در فعالیت پیاده‌سازی انجام می‌گردد. بنابراین گزینه سوم نیز نادرست است.

بسته (Package) مجموعه‌ای از کلاس‌های پیاده‌سازی‌شده و کامپایل‌شده به عنوان واحد مولفه و یا تعدادی مولفه فرعی پیاده‌سازی‌شده و کامپایل‌شده است. که از اجتماع بسته‌ها یک سیستم نرم‌افزاری ایجاد می‌گردد. نگاشت مولفه به بسته توسط کامپایل کردن مولفه‌ها انجام می‌گردد، در حالی که رابطه وابستگی در نمودار مولفه ترتیب کامپایل مولفه‌ها را نشان می‌دهد، بنابراین گزینه چهارم نیز نادرست است.

۱۴- گزینه (۱) صحیح است.

مدل تحلیل، مدل‌سازی عالم خارج به زبانی شبیه انسان است، اما مدل طراحی مدل‌سازی عالم داخل ماشین به زبانی شبیه زبان ماشین است. بنابراین برای اینکه ساخت برنامه کامپیوتری که به زبان ماشین است، امکان‌پذیر باشد، باید مدل تحلیل به مدل طراحی نگاشت شود تا مدل طراحی بتواند به عنوان نقشه راهی شبیه به زبان ماشین، راهگشا باشد.

مدل‌های تحلیل، کلی‌تر و انتزاعی‌تر هستند، و برای مدل‌سازی عالم خارج مورد استفاده قرار می‌گیرند، بدون آنکه به جزئیات نحوه پیاده‌سازی بپردازند، در واقع مدل تحلیل به کلی گویی به زبان انسان و حذف جزئیات نحوه پیاده‌سازی می‌پردازد. اما مدل‌های طراحی، جزئی‌تر و غیرانتزاعی‌تر هستند، و برای مدل‌سازی عالم داخل ماشین مورد استفاده قرار می‌گیرند، و به بیان جزئیات نحوه پیاده‌سازی می‌پردازند، در واقع مدل طراحی به جزئی‌گویی به زبان شبیه ماشین و درج جزئیات نحوه پیاده‌سازی می‌پردازد.

با نگاهی سطح به سطح، به حل مساله، سطوح مختلفی از انتزاع را خواهیم داشت. در بالاترین سطح انتزاع، راه حل به صورت کلی به زبان محیط مساله بیان می‌گردد. در سطوح پایین‌تر، راه حل به جزئیات پیاده‌سازی نزدیک‌تر می‌شود و در پایین‌ترین سطح انتزاع راه حل به صورتی بیان می‌شود تا مستقیما قابل پیاده‌سازی باشد.

تجرید یا انتزاع، روش و نگرشی در ارائه و توضیح است که در آن فقط اطلاعات مهمی که برای حل یک مساله لازم است، گردآوری و نگهداری می‌شود و از بقیه اطلاعات صرف نظر می‌گردد.

درک یکجای مساله به یکباره، اغلب غیر ممکن است. بنابراین با استفاده از مفهوم انتزاع، ابتدا بر کلیت مساله تمرکز می‌شود، اما برای حل کامل مساله جزئیات آن نیز باید در نهایت بررسی شوند! فعالیت پالایش با تعیین ریز به ریز عملیات، جزئیات مساله را آشکار می‌کند. این فعالیت روندی بالا به پایین و سلسله مراتبی دارد، در واقع در سطوح پایینی پالایش، جزئیات واضح‌تر می‌شوند. هرچه به سطوح پایین‌تر سلسله مراتب نزدیک‌تر می‌شوید، جزئیات بیشتری از مساله آشکار می‌شود، تا جایی که در پایین‌ترین سطح پالایش، دستورات زبان برنامه‌نویسی ارائه می‌گردند.

انتزاع و پالایش دو مفهوم مکمل هستند. انتزاع مهندسان نرم‌افزار را قادر می‌سازد تا داده‌ها، عملکردها و رفتارها را باحذف جزئیات تعریف کنند. در حالی که پالایش بر محتوای داخلی داده‌ها، عملکردها و رفتارها تمرکز داشته و جزئیات آنها را تشریح می‌کند. این تشریح به صورت لایه به لایه و در یک ساختار سلسله مراتبی و بالا به پایین است که با پیشروی در عمق لایه‌ها، جزئیات بیشتری آشکار می‌شود. به پالایش، افراز یا بخش‌بندی(partitioning) نیز گفته می‌شود.

در متدولوژی ساخت‌یافته، در اولین مرحله مدل‌سازی (مدل تحلیل)، سیستم به دو وجه «داده» و «عملکرد» تفکیک می‌شود. سپس طی روندی سلسله مراتبی و مطابق با روش بالا به پایین، هر یک از این وجوه خود به مؤلفه‌های فرعی تجزیه می‌شوند. این روند تا به جایی ادامه می‌یابد که جزئیات توابع برنامه جهت پیاده‌سازی مشخص شوند.

همچنین در متدولوژی شیءگراء، در اولین مرحله‌ی مدل‌سازی (مدل تحلیل) سیستم در قالب کلاس‌ها (شامل داده (صفت) و عملکرد (متد)) نشان داده می‌شود. سپس طی روندی سلسله مراتبی و مطابق با روش بالا به پایین، کلاس‌ها با جزئیات بیشتری مشخص می‌شوند. این روند تا به جایی ادامه می‌یابد که جزئیات کلاس‌های برنامه جهت پیاده‌سازی مشخص شوند.

بنابر مطالب فوق واضح است که گزینه اول درست است.

Use Caseها و Actorها محدوده سیستم در حال ساخت را مشخص می‌کنند. Use Case شامل تمام آن چیزهایی است که درون سیستم قرار دارد و Actor شامل تمام آن چیزهایی است که خارج از سیستم قرار دارد. هر فرد یا هر چیزی که با سیستم تعامل دارد، Actor یا بازیگر نامیده می‌شود. Use Caseها هر چیز موجود در داخل و محدوده سیستم را توصیف می کنند، در حالی که Actorها هر چیز موجود در خارج از محدوده سیستم را توصیف می‌کنند.

بازیگران، افراد و یا گاهی نرم‌افزار و یا سخت‌افزارهایی هستند که از سیستم استفاده می‌کنند و یا اطلاعاتی را برای سیستم فراهم می‌کنند. لذا Actorها فقط افراد و کابران نرم‌افزار نیستند. بلکه می‌توانند نرم‌افزار و یا سخت‌افزارهایی باشند که از سیستم استفاده می‌کنند و یا اطلاعاتی را برای

سیستم فراهم می‌کنند. بنابراین گزینه دوم نادرست است.

نمونه‌سازی می‌تواند توسط ابزارهای کامپیوتری و غیرکامپیوتری انجام گردد، اما صرف استفاده از ابزارها برای توسعه سریع نمونه‌ها، منجر به این نمی‌شود که پروژه طبق زمان‌بندی پیش‌رود. بلکه مدیریت زمان‌بندی پروژه و مدیریت ریسک‌های پروژه نیز در زمان تحویل پروژه مطابق زمان‌بندی مشخص شده موثر است. بنابراین گزینه سوم نیز نادرست است.

هنگامی مستندات مربوط به تعریف نیازمندی‌های مشتری، به مستنداتی غیرقابل تغییر بدل می گردد که ماهیت پروژه غیرتکاملی باشد و همه نیازها در همان ابتدای کار مشخص، ثابت و بدون تغییر باشد. در این شرایط مدل‌های فرآیند تولید نرم‌افزار غیرتکاملی برای روند تولید نرم‌افزار مورد استفاده قرار می‌گیرند.

اما هنگامی که ماهیت پروژه تکاملی باشد و همه نیازها در همان ابتدای کار مشخص، ثابت و بدون تغییر نباشد. آنگاه در این شرایط مدل‌های فرآیند تولید نرم‌افزار تکاملی برای روند تولید نرم‌افزار مورد استفاده قرار می‌گیرند. که به تبع منجر به این می‌شود که مستندات مربوط به تعریف نیازمندی‌های مشتری، به مستنداتی قابل تغییر در طول روند حرکت پروژه بدل گردد. لذا در چنین شرایطی مهم‌ترین کار در فعالیت ارتباطات، مدیریت شناسایی نیازمندی‌ها و پیگیری تغییرات نیازمندی‌های مشتری است. بنابراین گزینه چهارم نیز نادرست است.

۱۵- گزینه (۳) صحیح است.

پس از شناسایی موارد کاربرد و سناریونویسی برای هر یک از موارد کاربرد، زمان تعریف کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌های همکار برای هر یک از موارد کاربرد می‌رسد. برای شناسایی کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌ها برای هر یک از موارد کاربرد، از تکنیکی موسوم به CRC که سرواژه‌ی عبارت Class – Responsibility Collaborator و به معنی «مدل همکاری مسئولیت‌های کلاس‌ها» می‌باشد، استفاده می‌شود. مدل‌سازی CRC روشی ساده جهت تعیین و سازماندهی کلاس‌های داخل هر مورد کاربرد است. در این روش به هر کلاس یک کارت CRC اختصاص داده می‌شود که شامل سه بخش کلی زیر است:

نام کلاس

مسئولیت‌های کلاس(Responsibilities)

- صفات کلاس
- متدهای کلاس

همکاران کلاس(Collaborators)

کلاس‌ها یا به تنهایی از عهده انجام مسئولیت‌های خود بر می‌آیند و یا از طریق همکاری با کلاس‌های همکار خود از عهده انجام مسئولیت‌های خود بر می‌آیند. بنابراین اگر کلاسی همکارانی

دارد، باید در بخش مربوط به همکاران نوشته شود.

برای کشف کلاس‌های همکار داخل هر use case، از سناریوی اصلی (نوشتاری یا نموداری) هر use case استفاده می‌گردد. برای این منظور، **اسامی** موجود داخل هر use case مورد جستجو قرار می‌گیرند. از آنجا که نیازها یا موارد کاربرد مشتری به تدریج و در طی تکرار مشخص می‌شوند و همچنین از آنجا که کلاس‌های همکار داخل هر مورد کاربرد یا نیاز هستند، بنابراین با تکامل و کامل شدن نیازها یا موارد کاربرد، به تبع کلاس‌های همکار هر مورد کاربرد نیز کامل می‌شود. بنابراین تشخیص تمام کلاس‌های برنامه، در همان ابتدای کار تقریبا غیر ممکن است، در واقع در طول پروژه و با گذشت زمان تحلیل‌گر متوجه نیازها و به تبع کلاس‌های جدیدی می‌شود که در ابتدای کار نیاز به آن‌ها چندان محسوس نبوده است. بنابراین می‌توان گفت روند تشخیص نیازها یا موارد کاربرد و به تبع کلاس‌های همکار به عنوان مرتفع‌کننده نیازها یا موارد کاربرد، یک فرآیند تکرارشونده است و در طول فرآیند تولید نرم‌افزار کامل و کامل‌تر می‌شوند.

پس از اتمام مدل‌سازی CRC، کارت‌های CRC با سناریوی اصلی هر use case یا مورد کاربرد تطابق داده می‌شود. تا مشخص شود که آیا تمام کلاس‌های مربوط به use case درست شناسایی شده‌اند یا خیر. که در صورت نیاز، می‌بایست تغییرات لازم بر روی کارت‌های CRC اعمال گردد.

منظور از عبارت «تمام نمایش‌های مورد کاربرد (use case)» در گزینه اول، شیوه‌های نمایش شرح حال هر use case است، این شرح حال use case همان سناریوهای اصلی و فرعی داخل use case است که به دو شیوه نوشتاری (متنی) و گرافیکی (نموداری) قابل نمایش است. به بیان روال حرکت قدم به قدم کارها داخل یک use case از نقطه شروع تا رسیدن به یک نتیجه مطلوب (موفق) سناریوی اصلی گفته می‌شود. و به بیان روال حرکت قدم به قدم کارها داخل یک use case از نقطه شروع تا رسیدن به یک نتیجه نامطلوب (ناموفق) سناریوی فرعی گفته می‌شود. بنابراین گزینه اول درست است.

کلاس مکانیزمی است که توسط آن، مفهوم کپسوله‌سازی پیاده‌سازی می‌شود.

بنابراین مطابق تعریف بسته‌بندی، کلاس نیز صفات و متدهای مرتبط به هم را در یک بسته، بسته‌بندی می‌کند. درون یک کلاس، صفات و متدها یا هردو می‌توانند به صورت اختصاصی (private) و یا عمومی (public) باشند

وجه تمایز کلاس‌ها در تفاوت در صفات آن‌ها است. مانند تفاوت در صفات کلاس استاد و کلاس دانشجو.

به نمونه‌ای از یک کلاس، شیء گفته می‌شود. مانند دانشجو اکبری از کلاس دانشجو.

معرفی یک کلاس یک انتزاع یا تجرید **منطقی** (logical abstraction) است که یک نوع داده جدید را معرفی می‌کند و نشان می‌دهد که یک شی از این نوع، چه شکل و شمایلی دارد. اما معرفی یک شیء از یک کلاس، یک موجودیت **فیزیکی** (physical entity) از نوع یک کلاس را ایجاد می‌کند، بدین معنی که با معرفی یک شیء، فضای حافظه‌ای به شیء اختصاص داده می‌شود

ولی در معرفی یک کلاس فضای حافظه‌ای تخصیص نمی‌یابد.

وجه تمایز اشیاء در مقادیر صفات اشیاء است. مانند تفاوت در مقادیر دانشجو اکبری و دانشجو احمدی. بنابراین گزینه دوم نیز درست است.

به طورکلی هرگاه یک کلاس (ب)، **از نوع** یک کلاس (الف) باشد، گوییم بین دو کلاس مذکور رابطه **وراثت** برقرار است. در این صورت کلاس (ب) فرزند و کلاس (الف) پدر یا والد نامیده می‌شود. به عبارت دیگر همانطور که پیش از این نیز گفتیم، وراثت فرآیندی است که به وسیله آن یک کلاس (فرزند) می‌تواند صفات و متدهای کلاس دیگری (پدر) را کسب کند. به عبارت کلی‌تر یک کلاس فرزند ضمن به ارث بردن مجموعه‌ای از صفات و متدهای عمومی کلاس پدر، می‌تواند ویژگی‌های خاص و مختص خود را نیز به آنها اضافه کند. **در رابطه وراثت هر تغییر در کلاس پدر بر کلاس فرزند نیز اثر می‌گذارد اما عکس این مطلب برقرار نیست.** ابرکلاس، همان کلاس پدر است که ویژگی‌ها و عملیات خود را در اختیار کلاس‌های دیگر (فرزندان) قرار می‌دهد. زیر کلاس نیز همان کلاس فرزند است که برخی از صفات و عملیاتش متعلق به یک ابرکلاس است.

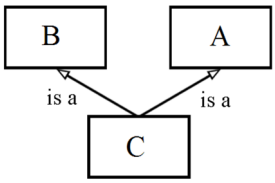
دو رویکرد برای ایجاد سلسله مراتب وراثت وجود دارد:

۱- رویکرد «بالا به پایین» یا تخصیص (Specialization) که در آن، ابتدا کلاس‌های پدر، تعریف و سپس کلاس‌های فرزند ایجاد می‌شوند.

۲- رویکرد «پایین به بالا» یا تعمیم (Generalization) که در آن، از طریق تعمیم کلاس‌های فرزند به کلاس‌های پدر می‌رسیم.

بنابر مطالب فوق، واضح است که گزینه سوم نادرست است.

هرگاه یک کلاس برخی از صفات و عملیات را از یک کلاس و برخی دیگر را از یک کلاس دیگر به ارث ببرد، در این حالت وراثت چندگانه رخ داده‌است. شکل زیر نمونه‌ای از مدل‌سازی **«ارث‌بری چندگانه»** (Multiple Inheritance) را نشان می‌دهد.



در شکل فوق کلاس C به صورت وراثت چندگانه از کلاس A و کلاس B ارث‌بری کرده است. بنابراین گزینه چهارم نیز درست است.

۱۶- گزینه (۲) صحیح است.

پس از شناسایی موارد کاربرد و سناریونویسی برای هر یک از موارد کاربرد، زمان تعریف

کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌های همکار برای هر یک از موارد کاربرد می‌رسد. برای شناسایی کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌ها برای هر یک از موارد کاربرد، از تکنیکی موسوم به CRC که سرواژه‌ی عبارت Class – Responsibility Collaborator و به معنی «مدل همکاری مسئولیت‌های کلاس‌ها» می‌باشد، استفاده می‌شود. مدل‌سازی CRC روشی ساده جهت تعیین و سازماندهی کلاس‌های داخل هر مورد کاربرد است. در این روش به هر کلاس یک کارت CRC اختصاص داده می‌شود که شامل سه بخش کلی زیر است:

نام کلاس

مسئولیت‌های کلاس (Responsibilities)

- صفات کلاس
- متدهای کلاس

همکاران کلاس (Collaborators)

کلاس‌ها یا به تنهایی از عهده انجام مسئولیت‌های خود بر می‌آیند و یا از طریق همکاری با کلاس‌های همکار خود از عهده انجام مسئولیت‌های خود بر می‌آیند. بنابراین اگر کلاسی همکارانی دارد، باید در بخش مربوط به همکاران نوشته شود.

برای کشف کلاس‌های همکار داخل هر use case، از سناریوی اصلی (نوشتاری یا نموداری) هر use case استفاده می‌گردد. برای این منظور، **اسامی** موجود داخل هر use case مورد جستجو قرار می‌گیرند. از آنجا که نیازها یا موارد کاربرد مشتری به تدریج و در طی تکرار مشخص می‌شوند و همچنین از آنجا که کلاس‌های همکار داخل هر مورد کاربرد یا نیاز هستند، بنابراین با تکامل و کامل شدن نیازها یا موارد کاربرد، به تبع کلاس‌های همکار هر مورد کاربرد نیز کامل می‌شود. بنابراین تشخیص تمام کلاس‌های برنامه، در همان ابتدای کار تقریبا غیر ممکن است، در واقع در طول پروژه و با گذشت زمان تحلیل‌گر متوجه نیازها و به تبع کلاس‌های جدیدی می‌شود که در ابتدای کار نیاز به آن‌ها چندان محسوس نبوده است. بنابراین می‌توان گفت روند تشخیص نیازها یا موارد کاربرد و به تبع کلاس‌های همکار به عنوان مرتفع‌کننده نیازها یا موارد کاربرد، یک فرآیند تکرارشونده است و در طول فرآیند تولید نرم‌افزار کامل و کامل‌تر می‌شوند.

سه نوع رابطه مختلف بین کلاس‌های همکار در مدل CRC وجود دارد:

رابطه آگاه است از (has knowledge of)

دو انسانی که همدیگر را می‌شناسند و امکان گفتگو و تبادل پیام با هم دارند، رابطه انجمنی با هم دارند. در رابطه انجمنی یک شیء بطور ساده درباره شیء دیگر می‌داند به همان طریقی که یک فرد ممکن است فرد دیگری را بشناسد. یک برنامه کامپیوتری شیءگرا از اجتماع تعدادی کلاس ایجاد شده است، کلاس‌های همکار **بدون** رابطه جزء و کل و هم‌هدف در یک بخش مشترک از

برنامه، کلاس‌های همکار **با** رابطه جزء و کل و هم‌هدف در یک بخش مشترک از برنامه و کلاس‌های غیرهمکار در دو بخش مختلف از برنامه.

رابطه‌ای که میان کلاس‌های همکار بدون رابطه جزء و کل جهت گفتگوی میان اشیاء آنها وجود دارد، رابطه انجمنی است.

کلاس‌های همکار بدون رابطه جزء و کل، جهت انجام وظایف خود از طریق مکانیزم پیام به گفتگو با یکدیگر می‌پردازند. در یک بیان ساده، هرگاه میان دو شیء بدون رابطه جزء و کل گفتگو باشد، کلاس‌های این دو شیء هم باهم همکار هستند و هم رابطه انجمنی میان آنها برقرار است.

هرگاه مابین دو کلاس رابطه **انجمنی** مطرح باشد، یعنی تبادل پیام مابین برخی متدهای اشیای دو کلاس صورت گیرد، آنگاه در این شرایط **رابطه آگاهی** خواهیم داشت. مانند ارسال پیام از شیء بازیکن به شیء توپ پس از انجام متد شوت زدن مربوط به شیء بازیکن، برای تغییر مختصات توپ توسط صدا زدن متد تغییر مختصات توپ مربوط به شیء توپ. یعنی شیء بازیکن، باید در هنگام شوت‌زدن، مختصات توپ را توسط صدازدن متد تغییر مختصات توپ، تغییر دهد.

رابطه بخشی است از (is part of)

هرگاه رابطه‌ای مابین یک واحد کل و تعدادی واحد جزء مطرح باشد، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشد، آنگاه در این شرایط رابطه بخشی خواهیم داشت. این رابطه خود بر دو نوع **تجمع** و **ترکیب** می‌باشد.

رابطه وابسته است به (depends upon)

هر گاه مابین دو کلاس رابطه وابستگی مطرح باشد، رابطه وابستگی خواهیم داشت. در واقع هرگاه تغییرات مقادیر صفات یک شیء، روی مقادیر صفات شیء دیگری تأثیر بگذارد، این دو شیء به هم وابسته هستند. در واقع این رابطه زمانی رخ می‌دهد که مقادیر صفاتی از یک کلاس به مقادیر صفاتی از یک کلاس دیگر وابسته باشد. برای مثال در برنامه کامپیوتری فوتبال، مختصات شیء سر بازیکن، دست بازیکن و پای بازیکن، همواره باید به مختصات شیء بدنه بازیکن وابسته باشد (مگر خشونت در بازی زیاد باشد!). در واقع مختصات سر بازیکن، دست بازیکن و پای بازیکن، همواره باید با تغییرات مختصات بدنه بازیکن تغییر کند. وگرنه در هنگام حرکت بازیکن، سر بازیکن، دست بازیکن و پای بازیکن از بدنه بازیکن جلو یا عقب می‌افتند!

به کلاس‌های همکار، **کلاس‌های مشارکت‌کننده** نیز گفته می‌شود.

۱۷- گزینه (۴) صحیح است.

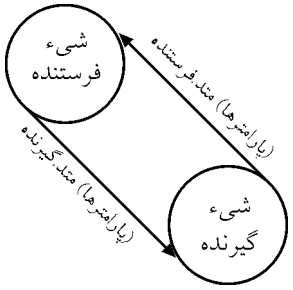
راه ایجاد ارتباط بین اشیاء، ارسال پیام است، در واقع پیام مکانیزمی است که اشیاء به وسیله آن

با هم ارتباط برقرار می‌کنند. یک پیام باعث می‌شود رفتاری در شیء گیرنده انجام شود. بنابراین ارسال یک پیام از یک شیء مبدا به شیء مقصد به معنی صدا زدن یکی از متدهای شیء مقصد است. یک پیام شامل شیء مقصد (شیء گیرنده). متد (متدی که در شیء مقصد پیام را دریافت می‌کند) و پارامترهای مربوطه می‌باشد.

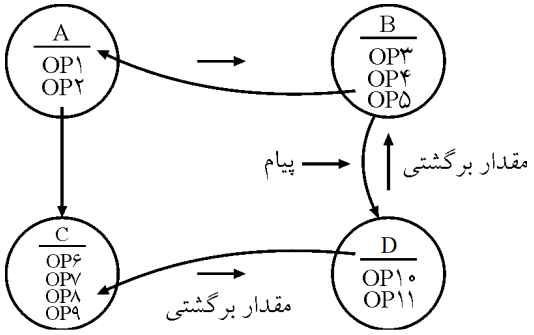
تعامل میان اشیاء در شکل زیر نشان داده شده است، یک عمل در داخل شیء فرستنده پیامی به شکل زیر تولید می‌کند:

(پارامترها) متد شیء مقصد. نام شیء مقصد

که در آن، مقصد، شیء گیرنده‌ای را تعریف می‌کند که توسط پیام صدا زده می‌شود.



مثال: چهار شیء A، B، C و D با مبادله پیام با یکدیگر ارتباط برقرار می‌کنند.



مبادله پیام‌های بین اشیاء

اگر شیء B بخواهد متد OP10 از شیء D را اجرا کند، پیامی به شکل زیر به D ارسال می‌کند:

D.OP10(data)

شیء D نیز به عنوان بخشی از اجرای OP10 ممکن است، پیامی به شکل زیر به C بفرستد:

C.OP8(data)

C عمل OP8 را می‌یابد، آن را اجرا می‌کند و سپس یک مقدار بازگشتی مناسب به D ارسال

می‌کند. عمل OP10 کامل می‌شود و مقداری را به B بازمی‌گرداند.

توجه: برای آن که از یک شیء درخواست شود تا یکی از متدهای خود را انجام دهد، باید پیامی به آن داده شود تا معلوم شود چه باید بکند. شیء گیرنده ابتدا با انجام متد درخواست شده و سپس با بازگرداندن کنترل به شیء مبدا، این پیام را پاسخ می‌دهد.

شمول یا رابطه شامل بودن (Include) و بسط یا رابطه توسعه‌دادن (Extend) مربوط به انواع روابط بین Use Caseها می‌باشد. رابطه‌ی وابستگی (Dependency) مربوط به روابط میان کلاس‌ها یا ترتیب کامپایل میان مولفه‌های برنامه می‌باشد.

۱۸- گزینه (۴) صحیح است.

قبل از پاسخ به سوال مطرح شده، به سوال زیر پاسخ دهید:

سوال:کدام‌یک از موارد زیر برای نوشتن مناسب نیست؟

۱) مداد ۲) خودکار ۳) اتود ۴) هیچ‌کدام

پاسخ - گزینه (۴) صحیح است. زیرا همه موارد فوق برای نوشتن **مناسب** هستند. مداد مناسب است. خودکار مناسب است و اتود هم مناسب است.

به طراحی مولفه، طراحی جزئی، طراحی تفصیلی و طراحی رویه‌ای نیز گفته می‌شود.

طراحی مولفه، فعالیت تبدیل طراحی معماری به نرم‌افزار است. در این مرحله، سطح انتزاع طراحی معماری به سطح انتزاع نرم‌افزار کاربردی نزدیک می‌گردد. طراحی در سطح مؤلفه‌ها، نرم افزار را در سطحی از انتزاع تصویر می‌کند که به کد نزدیک است. طراحی مولفه، به عنوان نقشه راهی دقیق، و نزدیک به زبان پیاده‌سازی، در فعالیت پیاده‌سازی نرم‌افزار، منجر به صرفه جویی در زمان و هزینه‌های تولید می‌گردد. در طراحی مولفه، مهندس نرم‌افزار باید ساختمان داده‌ها، واسطها و الگوریتم‌ها را با جزییات کافی به نمایش در آورد تا راهنمای تولید کد منبع زبان برنامه‌نویسی باشد.

طراحی مؤلفه ساخت‌یافته، طراحی معماری از همان فعالیت مدل طراحی را به عنوان ورودی دریافت کرده و طراحی مؤلفه را توسط ابزارهایی همچون شبه کد یا فلوچارت ایجاد می‌کند.

در طراحی مؤلفه ساخت‌یافته، اسکلت، ساختار و چیدمان کلی مولفه‌های (توابع) برنامه به این معنی که چه مولفه‌ای (تابعی) چه مولفه‌ای (تابعی) دیگر را صدا می‌زند، با ذکر جزئیات داخلی مولفه‌ها (توابع) مشخص می‌گردد (ساختار درختی برنامه با ذکر جزئیات مولفه‌ها (توابع)). مانند اسکلت یک ساختمان که گویای جایگاه مؤلفه‌های ساختمان است و آجرچینی هم شده است. (اسکلت یک ساختمان به همراه آجرچینی).

طراحی مؤلفه شیءگرا، طراحی معماری از همان فعالیت مدل طراحی را به عنوان ورودی دریافت کرده و طراحی مؤلفه را توسط ابزارهایی همچون شبه کد یا Activity Diagram یا Swimlane Diagram ایجاد می‌کند.

در طراحی مؤلفه شیءگرا، اسکلت، ساختار و چیدمان کلی مؤلفه‌های برنامه به این معنی که چه مولفه‌ای (کلاسی) با چه مولفه‌ای (کلاسی) دیگر در ارتباط است، با ذکر جزئیات مربوط به شرح متدهای کلاس‌های همکار داخل یک مؤلفه فرعی (یک بخش از نرم‌افزار). مانند اسکلت یک ساختمان که گویای جایگاه مؤلفه‌های ساختمان است و آجرچینی هم شده است. (اسکلت یک ساختمان به همراه آجرچینی).

مطابق آنچه گفتیم، در طراحی مولفه می‌بایست جزئیات مولفه‌ها توسط ابزارهای مربوطه نوشته شود.

از آنجاکه در عبارت مطرح شده در گزینه اول «حداکثر زمانی که استفاده‌کننده بایستی منتظر پاسخ سیستم بماند.» به جزئیات انجام کار پرداخته شده است. بنابراین برای گنجاندن در طراحی جزئی یا طراحی مولفه مناسب است.

از آنجاکه در عبارت مطرح شده در گزینه دوم «نحوه ذخیره اطلاعات مربوط به عملیات انجام شده در هر روز» به جزئیات انجام کار پرداخته شده است. بنابراین برای گنجاندن در طراحی جزئی یا طراحی مولفه مناسب است.

از آنجاکه در عبارت مطرح شده در گزینه سوم «اتفاقاتی که باید در صورت قطع اتصال از شبکه کامپیوتری در سیستم بیفتد.» به جزئیات انجام کار پرداخته شده است. بنابراین برای گنجاندن در طراحی جزئی یا طراحی مولفه مناسب است.

بنابراین گزینه چهارم یعنی «هیچکدام» انتخاب می‌گردد، زیرا گزینه‌های اول، دوم و سوم مناسب طراحی مولفه هستند و نه نامناسب.

۱۹- گزینه (۳) صحیح است.

مدل تحلیل به روش ساخت‌یافته از سه بخش مدل‌سازی داده‌ای، مدل‌سازی عملکردی و مدل‌سازی رفتاری تشکیل شده است، مدل‌سازی داده‌ای شامل تحلیل موجودیت‌ها و تحلیل پرس و جوها می‌باشد. تحلیل موجودیت‌ها توسط ابزار مدل ER و تحلیل پرس و جوها توسط ابزار حساب رابطه‌ای مدل می‌شوند، مدل‌سازی عملکردی توسط ابزار DFD و مدل‌سازی رفتاری توسط ابزار STD مدل می‌شود. نمودار جریان‌داده پس از نگاشت به طراحی معماری و گذر به طراحی مولفه در مدل طراحی سرانجام در فعالیت پیاده‌سازی به نسخه فیزیکی (کد) تبدیل می‌گردد. در واقع پیاده‌سازی عملکرد در فعالیت پیاده‌سازی، طراحی مؤلفه از مدل طراحی را به عنوان ورودی دریافت کرده و توسط یک زبان برنامه‌نویسی پیاده‌سازی عملکرد را انجام می‌دهد. بنابراین گزینه اول پاسخ سؤال نیست.

Use Case Diagram یا نمودار مورد کاربرد، Requirement Diagram یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد.

Use Case Diagram یا نمودار مورد کاربرد، هیچگاه به طور مستقیم، پیاده‌سازی نمی‌شود، بلکه مبنایی برای استخراج دیگر نمودارها قرار می‌گیرد، بنابراین ساختار نمودار کاربرد به

پیاده‌سازی نرم‌افزار یا نسخه فیزیکی تبدیل نمی‌گردد.

Use Caseها و Actorها محدوده سیستم در حال ساخت را مشخص می‌کنند. Use Case شامل تمام آن چیزهایی است که درون سیستم قرار دارد و Actor شامل تمام آن چیزهایی است که خارج از سیستم قرار دارد. هر فرد یا هر چیزی که با سیستم تعامل دارد، Actor یا بازیگر نامیده می‌شود. Use Caseها هر چیز موجود در داخل و محدوده سیستم را توصیف می‌کنند، در حالی که Actorها هر چیز موجود در خارج از محدوده سیستم را توصیف می‌کنند. بنابراین گزینه سوم پاسخ سؤال است.

پس از شناسایی کلاس‌ها و کلاس‌های همکار برای هر یک از موارد کاربرد در مدل CRC، زمان مدل‌سازی نموداری و ساختاری توسط نمودار کلاس می‌رسد.

Class Diagram یا نمودار کلاس جهت مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل یک case use مورد استفاده قرار می‌گیرد.

در فعالیت مدل تحلیل، جزئیات مربوط به کلاس‌ها، شامل جزئیات دقیق صفات و متدها، مشخص نمی‌گردد، بیان این جزئیات تا بخش طراحی مولفه از مدل طراحی به تأخیر می‌افتد.

نمودار کلاس به عنوان منبع اصلی تولید کد محسوب می‌گردد. هر کلاس دارای حداقل یک مسئولیت خواهد بود که در سلسله مراحل پالایش کلاس، این مسئولیت‌ها به مجموعه‌ای از صفات و متدها بدل می‌گردند. به نحوی که صفات و متدها بتوانند از عهده مسئولیت‌های کلاس برآیند.

نمودار کلاس پس از عبور از طراحی معماری و گذر به طراحی مولفه در مدل طراحی سرانجام در فعالیت پیاده‌سازی به نسخه فیزیکی (کد) تبدیل می‌گردد. در واقع فعالیت پیاده‌سازی، طراحی مؤلفه از مدل طراحی را به عنوان ورودی دریافت کرده و توسط یک زبان برنامه‌نویسی شیءگراء پیاده‌سازی عملکرد را انجام می‌دهد. در این مرحله، ساختار کلاس‌ها، ساختار و نحوه ارسال پیام‌ها مابین اشیاء، ساختمان داده‌ها و پرس‌وجوها برای ایجاد مؤلفه‌های فرعی (مؤلفه بخشی) و به تبع ایجاد مؤلفه اصلی (مؤلفه کلی یا برنامه اصلی) پیاده‌سازی می‌گردند. بنابراین گزینه چهارم پاسخ سؤال نیست.

پس از مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار نوبت به مدل‌سازی تعاملات پویای میان اشیاء همکار می‌رسد.

مدل‌سازی تعاملات پویای میان اشیاء همکار یا مدل‌سازی رفتاری، رفتار سیستم را در زمان اجرا نمایش می‌دهد. به بیان دیگر در این دیدگاه رفتارهای پویای سیستم مدل می‌شود.

مدل‌سازی رفتاری سیستم با استفاده از نمودارهای UML قابل انجام است:

الف) نمودار توالی (Sequence Diagram)

Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی

تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ب) نمودار همکاری (Collaboration Diagram)

Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

Collaboration Diagram یا نمودار همکاری، برای یک کاربرد خاص، جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

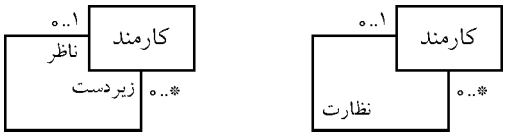
ج) نمودار حالت (State Transition Diagram)

State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل Sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد.

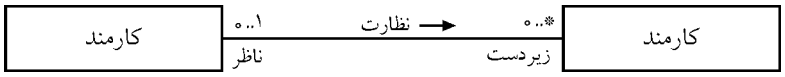
همه نمودارهای فوق (توالی، همکاری و حالت)، مدل‌سازی رفتاری یا تعاملی محسوب می‌گردند و در فعالیت پیاده‌سازی به نسخه فیزیکی بدل می‌گردند. بنابراین گزینه دوم پاسخ سؤال نیست.

۲۰- گزینه (۳) صحیح است.

در UML، یک رابطه‌ی خودانجمنی، رابطه‌ای است که یک کلاس با خودش دارد. لذا اگر اشیای یک کلاس با یکدیگر در ارتباط باشند، ارتباط مذکور از نوع خودانجمنی است. به عبارت دیگر، اگر ابتدا و انتهای یک رابطه انجمنی به یک کلاس اشاره داشته باشد، در این صورت رابطه‌ی مذکور از نوع خودانجمنی خواهد بود. در شکل زیر، دو مثال برای رابطه بازتابی ارائه شده است. توجه داشته باشید که در صورت مشخص کردن نقش دو طرف رابطه، ذکر نام رابطه، ضروری نیست.



برای سادگی در نحوه خواندن رابطه خودانجمنی توصیه می‌کنیم، رابطه را به شکل خطی ایجاد کنید، سپس رابطه را بخوانید:



شکل مطرح شده در صورت سوال گویای رابطه خودانجمنی (Self Association) می‌باشد. به این معنی که یک کارمند هیچ زیردستی ندارد (چون مدیر نیست) یا چندین زیردست دارد (چون

مدیر است) همچنین یک کارمند هیچ ناظری ندارد (چون مدیرکل است) یا یک ناظر دارد (چون مدیر کل نیست). بنابراین گزینه سوم درست است.

هرگاه رابطه‌ای مابین یک واحد کل و تعدادی واحد جزء مطرح باشد، یعنی یک کلاس کل از همکاری تعدادی کلاس جزء تشکیل شده باشد، آنگاه در این شرایط رابطه انجمنی پیشرفته یا رابطه بخشی خواهیم داشت. رابطه انجمنی پیشرفته یا رابطه بخشی به دو رابطه تجمع (Aggregation) و ترکیب (Composition) تقسیم می‌گردد. در رابطه انجمنی پیشرفته یا رابطه بخشی رابطه‌ای به شکل Self Aggregation وجود ندارد. بنابراین گزینه اول نادرست است.

راه ایجاد تعاملات پویا مابین اشیاء همکار، ارسال پیام (Message) است، در واقع پیام مکانیزمی است که اشیاء به وسیله آن با هم ارتباط برقرار می‌کنند. یک پیام باعث می‌شود رفتاری در شیء گیرنده انجام شود. بنابراین ارسال یک پیام از یک شیء مبدا به شیء مقصد به معنی صدا زدن یکی از متدهای شیء مقصد است. یک پیام شامل شیء مقصد (شیء گیرنده). متد (متدی که در شیء مقصد پیام را دریافت می‌کند) و پارامترهای مربوطه می‌باشد.

مدل‌سازی تعاملات پویای میان اشیاء همکار یا مدل‌سازی رفتاری، رفتار سیستم را در زمان اجرا نمایش می‌دهد. به بیان دیگر در این دیدگاه رفتارهای پویای سیستم مدل می‌شود.

مدل‌سازی رفتاری سیستم با استفاده از نمودارهای UML قابل انجام است:

الف) نمودار توالی (Sequence Diagram)

Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ب) نمودار همکاری (Collaboration Diagram)

Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

Collaboration Diagram یا نمودار همکاری، برای یک کاربرد خاص، جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ج) نمودار حالت (State Transition Diagram)

State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل Sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد.

در صورتی که یک شیء به خود پیامی ارسال کند به آن message to self گفته می‌شود مانند خواندن شماره کارت توسط دستگاه کارت‌خوان در یک سیستم ATM.

۲۱- گزینه (۱) صحیح است.

پس از شناسایی موارد کاربرد و سناریونویسی برای هر یک از موارد کاربرد، زمان تعریف کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌های همکار برای هر یک از موارد کاربرد می‌رسد. برای شناسایی کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌ها برای هر یک از موارد کاربرد، از تکنیکی موسوم به CRC که سرواژه‌ی عبارت Class – Responsibility Collaborator و به معنی «مدل همکاری مسئولیت‌های کلاس‌ها» می‌باشد، استفاده می‌شود. مدل‌سازی CRC روشی ساده جهت تعیین و سازماندهی کلاس‌های داخل هر مورد کاربرد است. در این روش به هر کلاس یک کارت CRC اختصاص داده می‌شود که شامل سه بخش کلی زیر است:

نام کلاس

مسئولیت‌های کلاس(Responsibilities)

- صفات کلاس
- متدهای کلاس

همکاران کلاس(Collaborators)

کلاس‌ها یا به تنهایی از عهده انجام مسئولیت‌های خود بر می‌آیند و یا از طریق همکاری با کلاس‌های همکار خود از عهده انجام مسئولیت‌های خود بر می‌آیند. بنابراین اگر کلاسی همکارانی دارد، باید در بخش مربوط به همکاران نوشته شود.

برای کشف کلاس‌های همکار داخل هر use case، از سناریوی اصلی (نوشتاری یا نموداری) هر use case استفاده می‌گردد. برای این منظور، **اسامی** موجود داخل هر use case مورد جستجو قرار می‌گیرند. بنابر آنچه گفتیم، گزینه اول درست است.

تشخیص روابط کل – جزء و توارث بین کلاس‌ها توسط نمودار کلاس انجام می‌گردد. بنابراین گزینه‌های دوم و سوم نادرست هستند.

کلاس‌ها با یکدیگر تعاملات پویا ندارند و با یکدیگر تبادل پیام نمی‌کنند در واقع کلاس‌ها همچون یک قاب عکس ثابت، فقط با یکدیگر ارتباطات ایستا دارند برای‌مثال فقط مشخص است که استاد و دانشجو با یکدیگر ارتباط دارند. اما اشیاء با یکدیگر تعاملات پویا در گذر زمان دارند و با یکدیگر اقدام به تبادل پیام می‌کنند، که این تبادلات پیام توسط نمودار توالی یا همکاری نشان داده می‌شود. مانند پیام‌هایی که میان مشتری و سیستم رد و بدل می‌شود تا مشتری مبلغی از حساب خود توسط یک دستگاه خودپرداز، برداشت کند. این تعاملات پویا توسط نمودار توالی و همکاری نشان داده می‌شود.

مدل‌سازی تعاملات پویای میان اشیاء همکار یا مدل‌سازی رفتاری، رفتار سیستم را در زمان اجرا نمایش می‌دهد. به بیان دیگر در این دیدگاه رفتارهای پویای سیستم مدل می‌شود.

مدل‌سازی رفتاری سیستم با استفاده از نمودارهای UML قابل انجام است:

الف) نمودار توالی (Sequence Diagram)

Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ب) نمودار همکاری (Collaboration Diagram)

Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

Collaboration Diagram یا نمودار همکاری، برای یک کاربرد خاص، جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ج) نمودار حالت (State Transition Diagram)

State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد.

بنابرآنچه گفتیم، گزینه چهارم نیز نادرست است.

۲۲- گزینه (۱) صحیح است.

در فعالیت ارتباطات(مهندسی نیازمندی‌ها) لیست نیازمندی‌های مشتری از طریق ارتباط با مشتری و ابزارهایی همچون گفتگو، مشاهده مصاحبه، پرسش‌نامه، بازدید، نمونه‌سازی دورانداختنی و نمونه‌سازی تکاملی، جمع‌آوری و آماده می‌گردد.

مهم‌ترین کار، در فعالیت ارتباطات، مدیریت شناسایی نیازمندی‌ها و پیگری تغییرات نیازمندی‌های مشتری است.

لیست نیازمندی‌های مشتری در این مرحله به صورت متن نوشته می‌شود.

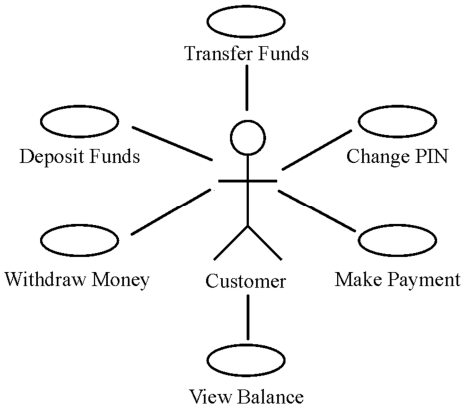
مثال: نمایش لیست نیازمندی‌های مشتری برای سیستم ATM.

۱- واریز وجه
۲- انتقال وجه
۳- برداشت وجه
۴- تغییر رمز
۵- پرداخت
۶- نمایش موجودی

پس از جمع‌آوری نیازمندی‌ها در فعالیت ارتباطات، نوبت به مدل تحلیل (مدل‌سازی نیازمندی‌ها) می‌رسد. مدل‌سازی که فعالیتی فنی به شمار می‌رود، نیازمندی‌ها را باید به گونه‌ای مدل نماید که برای سازنده و مشتری قابل فهم باشد.

Use Case Diagram یا نمودار مورد کاربرد، Requirement Diagram یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد.

مثال: مدل‌سازی لیست نیازمندی‌های مشتری برای سیستم ATM.



در نمودار فوق، لیست نیازمندی‌های مشتری، مدل‌سازی شده است.

به هر یک از نیازهای فوق یک use case یا مورد کاربر گفته می‌شود و به اجتماع این use caseها، Use Case Diagram یا نمودار مورد کاربرد گفته می‌شود.

از آنجا که Use Case Diagram یا نمودار مورد کاربرد یا نمودار نیاز مستقل از مفاهیم شیءگرایی (کلاس، وراثت و چندریختی) است، بنابراین می‌توان مدل‌سازی لیست نیازمندی‌های مشتری در مرحله **مدل تحلیل متدولوژی ساخت‌یافته** (مهندسی‌نرم‌افزار ساخت‌یافته) را توسط این نمودار انجام داد. بنابراین گزینه اول درست است.

سایر گزینه‌ها یعنی نمودارهای کلاس، ترتیبی (توالی) و ارتباط، مختص متدولوژی شیءگرا هستند.

Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

۲۳- گزینه (۳) صحیح است.

مدل تحلیل به روش ساخت‌یافته از سه بخش مدل‌سازی داده‌ای، مدل‌سازی عملکردی و مدل‌سازی رفتاری تشکیل شده است، مدل‌سازی داده‌ای شامل تحلیل موجودیت‌ها و تحلیل

پرس و جوها می‌باشد. تحلیل موجودیت‌ها توسط ابزار مدل ER و تحلیل پرس و جوها توسط ابزار حساب رابطه‌ای مدل می‌شوند، مدل‌سازی عملکردی توسط ابزار DFD و مدل‌سازی رفتاری توسط ابزار STD مدل می‌شود.

DFD یا Data Flow Diagram یا نمودار جریان داده جهت مدلسازی عملکردی مورد استفاده قرار می‌گیرد.

مدل‌سازی عملکردی یا مدل‌سازی جریان‌گرا (Flow-Oriented Models)، در زبان UML وجود نداشته و تحلیل شیءگرا (OOA) براساس آن انجام نمی‌شود. در عوض این مدل یکی از مهم‌ترین مدل‌های مدل تحلیل ساخت‌یافته می‌باشد.

هریک از نمودارهای UML در فعالیت‌های چارچوبی فرآیند تولید نرم‌افزار بر اساس متدولوژی شیءگرا مدل‌سازی خاص خود را بر عهده دارند.

۱- فعالیت ارتباط (مهندسی نیازمندی‌ها)

در این مرحله لیست نیازمندی‌های مشتری از طریق ارتباط با مشتری و ابزارهایی همچون گفتگو، مشاهده مصاحبه، پرسش‌نامه، بازدید، نمونه‌سازی دوراندااختنی و نمونه‌سازی تکاملی، جمع‌آوری و آماده می‌گردد.

لیست نیازمندی‌های مشتری در این مرحله به صورت متن نوشته می‌شود.

۲- فعالیت برنامه‌ریزی

برنامه‌ریزی یعنی هنر حرکت از مبدأ موجود به مقصد مطلوب برای رسیدن به نتیجه‌ای مطلوب بر اساس خواسته‌های مورد نیاز در یک زمان مشخص.

۳- فعالیت مدل‌سازی (تحلیل و طراحی)

مدل تحلیل: OOA (Object-Oriented Analysis)

پس از جمع‌آوری نیازمندی‌ها در فعالیت ارتباطات، نوبت به مدل تحلیل (مدل‌سازی نیازمندی‌ها) می‌رسد.

مدل تحلیل، شامل مراحل زیر می‌باشد:

الف) مدل‌سازی محیط عملیاتی یک کسب و کار

Business use case یا مورد کاربرد کسب و کار، جهت مدل‌سازی محیط عملیاتی یک کسب و کار مورد استفاده قرار می‌گیرد.

ب) مدل‌سازی لیست نیازمندی‌های مشتری

Use Case Diagram یا نمودار مورد کاربرد، Requirement Diagram یا نمودار نیاز جهت مدل‌سازی لیست نیازمندی‌های مشتری مورد استفاده قرار می‌گیرد.

ج) سناریونویسی

در این مرحله برای هر use case (مورد کاربرد یا نیاز) سناریو یا شرح حال نوشته می‌شود.
سناریو بر دو طبقه اصلی و فرعی می‌باشد.
نحوه نمایش سناریوی اصلی و فرعی به دو روش زیر می‌باشد:
نوشتاری (متنی): در این روش سناریوی اصلی و فرعی به صورت متنی نوشته می‌شود.
گرافیکی (نموداری): در این روش سناریوی اصلی و فرعی به صورت نمودار، مدل‌سازی می‌شود.

Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط شنا، جهت مدل‌سازی سناریوی اصلی و فرعی داخل یک use case (نیاز یا مورد کاربرد یا زیرسیستم) مورد استفاده قرار می‌گیرد.

د) کشف کلاس‌های همکار داخل هر use case یا مورد کاربرد

پس از شناسایی موارد کاربرد و سناریونویسی برای هر یک از موارد کاربرد، زمان تعریف کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌های همکار برای هر یک از موارد کاربرد می‌رسد. برای شناسایی کلاس‌ها، صفات، متدها و ارتباطات میان کلاس‌ها برای هر یک از موارد کاربرد، از تکنیکی موسوم به CRC که سرواژه‌ی عبارت Class – Responsibility Collaborator و به معنی «مدل همکاری مسئولیت‌های کلاس‌ها» می‌باشد، استفاده می‌شود.

هـ) مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار

پس از شناسایی کلاس‌ها و کلاس‌های همکار برای هر یک از موارد کاربرد در مدل CRC، زمان مدل‌سازی نموداری و ساختاری توسط نمودار کلاس می‌رسد. Class Diagram یا نمودار کلاس جهت مدل‌سازی ارتباطات ایستای میان کلاس‌های همکار داخل یک case use مورد استفاده قرار می‌گیرد.

در فعالیت مدل تحلیل، جزئیات مربوط به کلاس‌ها، شامل جزئیات دقیق صفات و متدها، مشخص نمی‌گردد، بیان این جزئیات تا بخش طراحی مولفه از مدل طراحی به تأخیر می‌افتد.

ی) مدل‌سازی تعاملات پویای میان اشیاء همکار

مدل‌سازی تعاملات پویای میان اشیاء همکار یا مدل‌سازی رفتاری، رفتار سیستم را در زمان اجرا نمایش می‌دهد. به بیان دیگر در این دیدگاه رفتارهای پویای سیستم مدل می‌شود.
مدل‌سازی رفتاری سیستم با استفاده از نمودارهای UML قابل انجام است:

الف) نمودار توالی (Sequence Diagram)

Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ب) نمودار همکاری (Collaboration Diagram)

Collaboration Diagram یا نمودار همکاری در نسخه‌های امروزی UML، به نمودار **Communication Diagram** یا **نمودار ارتباط**، تغییر نام یافته است.

Collaboration Diagram یا نمودار همکاری، برای یک کاربرد خاص، جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.

ج) نمودار حالت (State Transition Diagram)

State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد.

مدل طراحی: OOD (Object – Oriented Design)

پس از مدل تحلیل، نوبت به مدل طراحی می‌رسد. مدل طراحی به روش شی‌گراء شامل چهار بخش **طراحی داده**، **طراحی معماری**، **طراحی مؤلفه** و **طراحی واسط** می‌باشد.

طراحی داده

طراحی داده، شامل طراحی ساختمان داده‌ها و طراحی پرس‌وجوها می‌باشد.

طراحی معماری

طراحی معماری یا معماری نرم‌افزار، ساختار کلی نرم‌افزار و شیوه‌های یکپارچگی یک سیستم را بیان می‌کند.

طراحی مؤلفه

طراحی مؤلفه، طراحی معماری از همان فعالیت مدل طراحی را به عنوان ورودی دریافت کرده و طراحی مؤلفه را توسط ابزارهایی همچون شبه کد یا Activity Diagram یا Swimlane Diagram ایجاد می‌کند.

در این مرحله هر use case یا مورد کاربرد یا نیاز با اطلاعات مربوط به نمودارهای کلاس و توالی به یک مؤلفه فرعی اما با ذکر جزئیات تبدیل می‌گردد.

طراحی واسط

طراحی واسط یا همان واسط کاربر، براساس ورودی‌ها و خروجی‌های مورد نیاز کاربران نهایی به شکل نقشی بر روی کاغذ یا طرحی بر روی کامپیوتر ایجاد می‌گردد. مانند نحوه چیدمان منوها و فرم‌ها.

۴- فعالیت ساخت (پیاده‌سازی و تست)

پس از فعالیت مدل‌سازی (تحلیل و طراحی) نوبت به فعالیت ساخت (پیاده‌سازی و تست) می‌رسد.

پیاده سازی: OOP (Object – Oriented Programming)

پس از مدل طراحی نوبت به پیاده‌سازی می‌رسد. این کار توسط زبان‌های شی‌گراء، مانند C++ و SQL Server انجام می‌گردد.

Component Diagram یا نمودار مؤلفه جهت مدل‌سازی ساختار کلی پیاده‌سازی برنامه و ترتیب کامپایل مؤلفه‌های فرعی، برای ایجاد مؤلفه‌اصلی (مؤلفه کلی یا برنامه اصلی) مورد استفاده قرار می‌گیرد.

تست: OOT (Object – Oriented Testing)

پس از پیاده‌سازی نوبت به تست می‌رسد، در این مرحله کلیه موارد پیاده‌سازی شده از نظر خطاهای نحوی و خطاهای معنایی براساس لیست نیازمندی‌های مشتری (چک لیست) که در فعالیت ارتباطات تهیه شده بود، مورد وارسی قرار می‌گیرد.

فعالیت استقرار

پس از تست، نوبت به فعالیت استقرار می‌رسد. هدف از فعالیت استقرار در گام اول، ارائه یک دید کلی از سخت‌افزارهای مورد نیاز سیستم است که مؤلفه‌های نرم‌افزاری باید بر روی آن‌ها قرار گیرند. و در گام دوم، نرم‌افزار به مشتری تحویل داده می‌شود و مشتری با بررسی محصول دریافتی، باز خوردهای به دست آمده براساس همین ارزیابی‌ها را به تیم نرم‌افزاری ارائه می‌دهد.

Deployment Diagram یا نمودار استقرار، جهت مدل‌سازی گره‌های سخت افزاری برای نصب نرم‌افزار مربوطه در فعالیت استقرار مورد استفاده قرار می‌گیرد.

در ادامه به بررسی گزینه‌ها می‌پردازیم:

Business Use Case مربوط به مدل تحلیل شیءگرا است.

Class Diagram مربوط به مدل تحلیل و طراحی شیءگرا است.

CRC مربوط به مدل تحلیل شیءگرا است.

Use Case Diagram مربوط به مدل تحلیل شیءگرا است.

Sequence Diagram مربوط به مدل تحلیل و طراحی شیءگرا است.

Flow-Oriented Diagram مربوط به مدل تحلیل ساخت‌یافته است.

Deployment Diagram مربوط به فعالیت استقرار شیءگرا است. بنابراین گزینه سوم درست است.

Object Diagram مربوط به مدل تحلیل و طراحی شیءگرا است.
Activity Diagram, Swimlane Diagram مربوط به مدل تحلیل و طراحی شیءگرا است.

۲۴- گزینه (۲) صحیح است.

مدل طراحی به روش ساخت‌یافته شامل چهار بخش **طراحی داده**، **طراحی معماری**، **طراحی مؤلفه** و **طراحی واسط** می‌باشد.

طراحی معماری، تحلیل عملکرد (DFD) از مدل تحلیل را به عنوان ورودی دریافت کرده و توسط سبک ساخت‌یافته(فراخوانی و بازگشت)، طراحی معماری را انجام می‌دهد.

طراحی معماری یا معماری نرم‌افزار، ساختار کلی نرم‌افزار و شیوه‌های یکپارچگی یک سیستم را بیان می‌کند. به عبارت دیگر، ساختار سلسله مراتبی **مولفه‌های برنامه (توابع یا پیمانه‌ها)**، شیوه تعامل مولفه‌ها با یکدیگر و **ساختمان داده‌های** مورد نیاز مولفه‌ها را نشان می‌دهد. معماری نرم‌افزار یک مدل قابل درک از چگونگی سازمان‌دهی سیستم است. در واقع نشان‌گر ساختمان داده‌ها و مؤلفه‌های برنامه‌ای است که برای ساختن یک سیستم کامپیوتری لازم است. به طور دقیق‌تر معماری نرم‌افزار شامل دو سطح از طراحی می‌باشد یعنی طراحی داده و طراحی معماری. در واقع این ساختار مانند یک نقشه ساختمان، مبنای ساخت نرم‌افزار قرار می‌گیرد.

در طراحی معماری، اسکلت، ساختار و چیدمان کلی مولفه‌های (توابع) برنامه به این معنی که چه مولفه‌ای (تابعی) چه مولفه‌ای (تابعی) دیگر را صدا می‌زند، بدون ذکر جزئیات داخلی مولفه‌ها (توابع) مشخص می‌گردد (ساختار درختی برنامه بدون ذکر جزئیات مولفه‌ها (توابع)). مانند اسکلت یک ساختمان که گویای جایگاه مؤلفه‌های ساختمان است اما هنوز آجرچینی نشده است. (اسکلت یک ساختمان بدون آجرچینی).

در ساخت‌یافتگی یک مدل ساختاری ساختار واحدها و مولفه‌های برنامه را نشان می‌دهد و نه رفتار سیستم در قبال رخدادها را. در ساخت‌یافتگی رفتار سیستم در قبال رخدادها در مدل تحلیل به کمک نمودار انتقال حالت یا STD نشان داده می‌شود.

در ساخت‌یافتگی، STD برای مدل‌سازی رفتاری سیستم در قبال رخدادهای عالم خارج نرم‌افزار مورد استفاده قرار می‌گیرد.

مدل‌سازی رفتاری توسط STD، در سیستم‌های بی‌درنگ مورد استفاده قرار می‌گیرد.

هر نرم‌افزاری که توسط سنسور و حسگر، عالم خارج را شنود و مورد ارزیابی قرار می‌دهد، تا در موقع لزوم و در زمان واقعی، حقیقی و تا دیر نشده عکس‌العمل نشان دهد، یک نرم‌افزار بی‌درنگ است.

مدل طراحی به روش شیءگرا شامل چهار بخش **طراحی داده**، **طراحی معماری**، **طراحی مؤلفه** و **طراحی واسط** می‌باشد.

طراحی معماری، نمودار کلاس و نمودار توالی مرتبط با هر یک از موارد کاربرد را از مدل

تحلیل، به عنوان ورودی دریافت کرده و توسط سبک شیءگرا (مبتنی بر ارسال پیام مابین اشیاء)، طراحی معماری را انجام می‌دهد.

طراحی معماری یا معماری نرم‌افزار، ساختار کلی نرم‌افزار و شیوه‌های یکپارچگی یک سیستم را بیان می‌کند. به عبارت دیگر، ساختار سلسله مراتبی **مولفه‌های برنامه (کلاس‌ها یا پیمانه‌ها)**، شیوه تعامل مولفه‌ها با یکدیگر و **ساختمان داده‌های** مورد نیاز مولفه‌ها را نشان می‌دهد. معماری نرم‌افزار یک مدل قابل درک از چگونگی سازمان‌دهی سیستم است. در واقع نشان‌گر ساختمان داده‌ها و مؤلفه‌های برنامه‌ای است که برای ساختن یک سیستم کامپیوتری لازم است. به طور دقیق‌تر معماری نرم‌افزار شامل دو سطح از طراحی می‌باشد یعنی طراحی داده و طراحی معماری. در واقع این ساختار مانند یک نقشه ساختمان، مبنای ساخت نرم‌افزار قرار می‌گیرد.

در طراحی معماری، اسکلت، ساختار و چیدمان کلی مؤلفه‌های برنامه به این معنی که چه مولفه‌ای (کلاسی) با چه مولفه‌ای (کلاسی) دیگر در ارتباط است، بدون ذکر جزئیات مربوط به شرح متدهای کلاس‌های همکار داخل یک مؤلفه فرعی (یک بخش از نرم‌افزار). مانند اسکلت یک ساختمان که گویای جایگاه مؤلفه‌های ساختمان است اما هنوز آجر چینی نشده است. (اسکلت یک ساختمان بدون آجر چینی).

واحد مؤلفه (پیمانه) در شی‌گرایی، کلاس است که از کنار هم قرار گرفتن کلاس‌های همکار داخل هر use case یا مورد کاربرد یا نیاز، یک مؤلفه فرعی (مؤلفه بخشی) ایجاد می‌گردد، که از کنار هم قرار گرفتن مؤلفه‌های فرعی برنامه، مؤلفه اصلی (مؤلفه کلی یا برنامه اصلی) ایجاد می‌گردد.

در این مرحله هر use case یا مورد کاربرد یا نیاز با اطلاعات مربوط به نمودارهای کلاس و توالی به یک مؤلفه فرعی ولی بدون ذکر جزئیات تبدیل می‌گردد. همچنین در این مرحله نحوه چیدمان مولفه‌ها و ساختار کلی برنامه، کلاس‌های همکار، اشیاء همکار و تعریف ساختار پیام‌ها مابین فرستنده و گیرنده پیام‌ها اما بدون ذکر جزئیات مشخص می‌شود.

در شیءگرایی یک مدل ساختاری ساختار واحدها و مولفه‌های برنامه را نشان می‌دهد و نه رفتار سیستم در قبال رخدادها را. در شیءگرایی رفتار سیستم در قبال رخدادها در مدل تحلیل به کمک نمودار حالت نشان داده می‌شود.

اطلاعات به دست آمده از نمودار حالت در مدل تحلیل مربوط به کلاس‌های دارای حالات مختلف، در مدل طراحی، بخش طراحی مولفه، جهت تکمیل نمودن الگوریتم‌های مربوط به متدهای کلاس‌های دارای حالات مختلف مورد استفاده قرار می‌گیرد. اینکه متدهای یک کلاس دارای حالات مختلف در سطح طراحی مولفه در مواجه با حالت‌های مختلف یک شیء چگونه رفتار کنند از نمودار حالت استنباط می‌گردد.

بنابرآنچه گفتیم، واضح است که گزینه‌های اول، سوم و چهارم نادرست و گزینه دوم درست است.

۲۵- گزینه (۴) صحیح است.

صورت سوال به این شکل است:

کدام نمودار UML، برای مدل‌سازی تعامل اشیاء به کار می‌رود؟

۱) حالت (State Diagram)

گزینه اول پاسخ سوال نیست، زیرا State Transition Diagram یا نمودار انتقال حالت (وضعیت)، به عنوان یک نمودار مکمل sequence Diagram، برای مشخص‌کردن وضوح بیشتر برای شناخت حالت‌های مختلف یک شیء مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار انتقال حالت، چرخه حیات یک شیء را در حالت های مختلف (از زمانی که شیء ایجاد می‌شود تا زمانی که شیء از بین می‌رود) نمایش می‌دهد. در واقع این نمودار تمامی حالت‌های مختلفی را که یک شیء در طول حیات خود می‌تواند داشته باشد و همچنین نحوه انتقال بین حالت‌ها و وقایع باعث‌شونده این انتقال را نشان می‌دهد. به بیان دیگر نمودارهای حالت، راهی را آماده می‌کنند تا حالت‌های مختلف یک شیء را مدل کنند.

۲) مولفه (Component Diagram)

گزینه دوم پاسخ سوال نیست، زیرا Component Diagram یا نمودار مؤلفه جهت مدل‌سازی ساختار کلی پیاده‌سازی برنامه و ترتیب کامپایل مؤلفه‌های فرعی، برای ایجاد مؤلفه‌اصلی (مؤلفه کلی یا برنامه اصلی) مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار مؤلفه یک دید فیزیکی از مدل سیستم به همراه مؤلفه‌های فرعی نرم‌افزار و روابط بین آنها را نشان می‌دهد.

۳) فعالیت (Activity Diagram)

گزینه سوم پاسخ سوال نیست، زیرا Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط شنا، جهت مدل‌سازی سناریوی اصلی و فرعی داخل یک use case (نیاز یا مورد کاربرد یا زیرسیستم) مورد استفاده قرار می‌گیرد. به بیان دیگر نمودار فعالیت یا نمودار خط شنا، جهت مدل‌سازی روال انجام کارها داخل یک use case مورد استفاده قرار می گیرد. همچنین، در طراحی مؤلفه، باید جزئیات الگوریتمی متدهای کلاس تشریح شود. برای این منظور Activity Diagram یا نمودار فعالیت و Swimlane Diagram یا نمودار خط‌شنا مورد استفاده قرار می‌گیرد. نمودار فعالیت و نمودار خط‌شنا ساختاری مشابه فلوچارت دارد.

۴) توالی (Sequence Diagram)

گزینه چهارم پاسخ سوال است. زیرا Sequence Diagram یا نمودار توالی، Object Diagram یا نمودار شیء جهت مدل‌سازی تعاملات پویای میان اشیاء همکار داخل یک use case مورد استفاده قرار می‌گیرد.