مقدمه

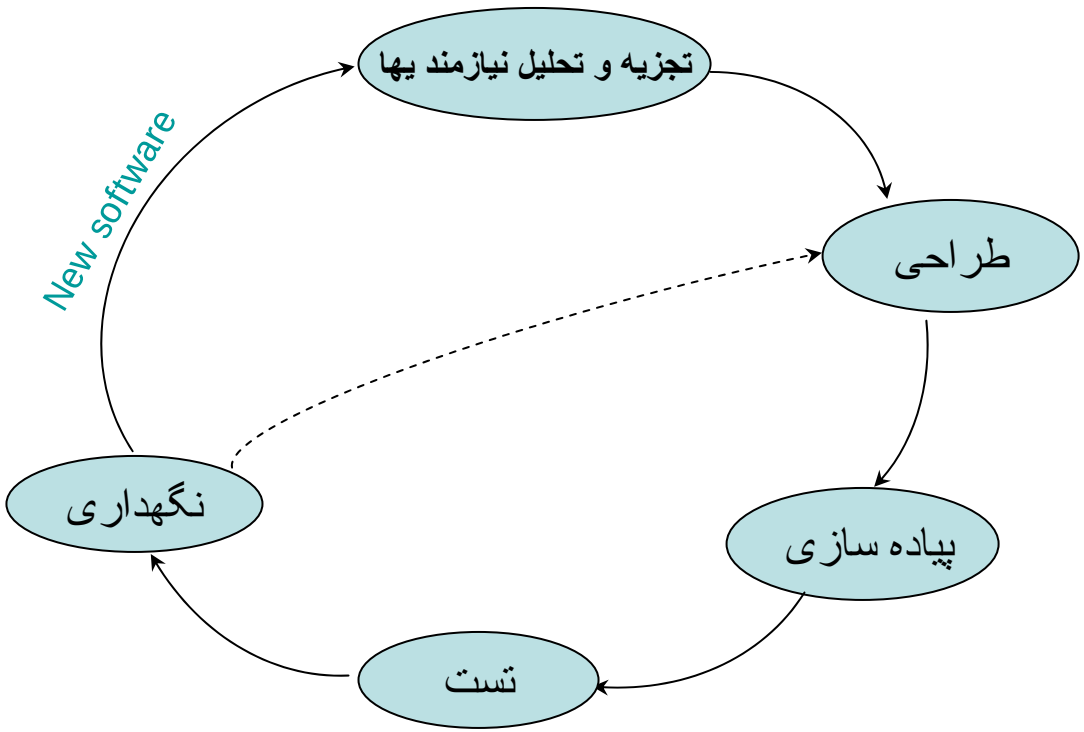
Systems Analysis

طراحی تجزیه و تحلیل سیستم ها

هدف از این درس :

نحوه ای شروع پروژه از از صورت مسئله تا پیاده سازی با تجزیه وتحلیل کلیه جوانب

چرخه حیات یک نرم افزار



نکته : برای این که از صورت مسئله تا اتمام پروژه مراحل را به خوبی طی کنیم می بایست طبق **حیات نرم افزار** عمل کنیم .

متدهای تجزیه و تحلیل وطراحی نر م افزار

ساختیافته : بر ایده ی ورودی و خروجی استوار می باشد و دارای نمودار DFD (Data Flow Diagram)

شی گرا

از دو متد فوق ما از **شی گرا** استفاده خواهیم کرد .

اکنون که می واهیم از شی گرا استفاده کنیم به یکسری نیازمندیها ازقبیل یک زبان برنامه نویسی و یک ابزار مدلسازی Modeling نیاز داریم .

انواع ابزار مدلسازی

- OMT
- Booch
- UML

که از سه ابزار ذکر شده ما UML که به یک استاندارد تبدیل شده برای مدلسازی پروژه انتخاب می کنیم.

نکته : در مدلسازی ما بیشتر روی دو قسمت اول چرخه حیات نرم افزار کار خواهیم کرد .

دو روش استفاده از UML : **۱- ابزار کاغذ و قلم**
۲- نرم افزاری

ابزار نرم افزاری که برای UML وجود دارد : CASE
(Computer Adder Software Engeeniring)

CASE : قادر به طراحی دیاگرامها تبدیل به زبان برنامه نویسی و یا حتی از یک ابزار به ابزار دیگر می باشد و اشکالات موجود را به ما نشان خواهد داد.

انواع نرم افزار CASE

Rational Rose : نرم افزاری که با هر سه ابزار مدلسازی سازگاری دارد.

Together : نرم افزاری که با مدل UML و زبان Java سازگاری دارد .

UML

Unified Modeling Language

یک زبان گرافیکی برای طراحان نرم افزار به جهت طراحی و یا توسعه سریع نرم افزار می باشد .

شرکت OMG سازنده UML برای جزوات و مقالات آموزشی دارای سایت :
WWW.OMG.ORG

ویژگی های UML :

۱- نمای گرافیکی
۲- قابل بسط و توسعه
۳- امکان مدل کردن سیستم های شی گرا

Design Patterns : یکسری الگو ها است که از قبل ساخته شده که برای طراحی بهتر است بر حسب نیاز استفاده نمائیم مانند نقشه ی خانه برای معمار و دوم اینکه مستقل از زبان های برنامه نویسی طراحی را انجام می دهد .

ویژگی الگو ها :

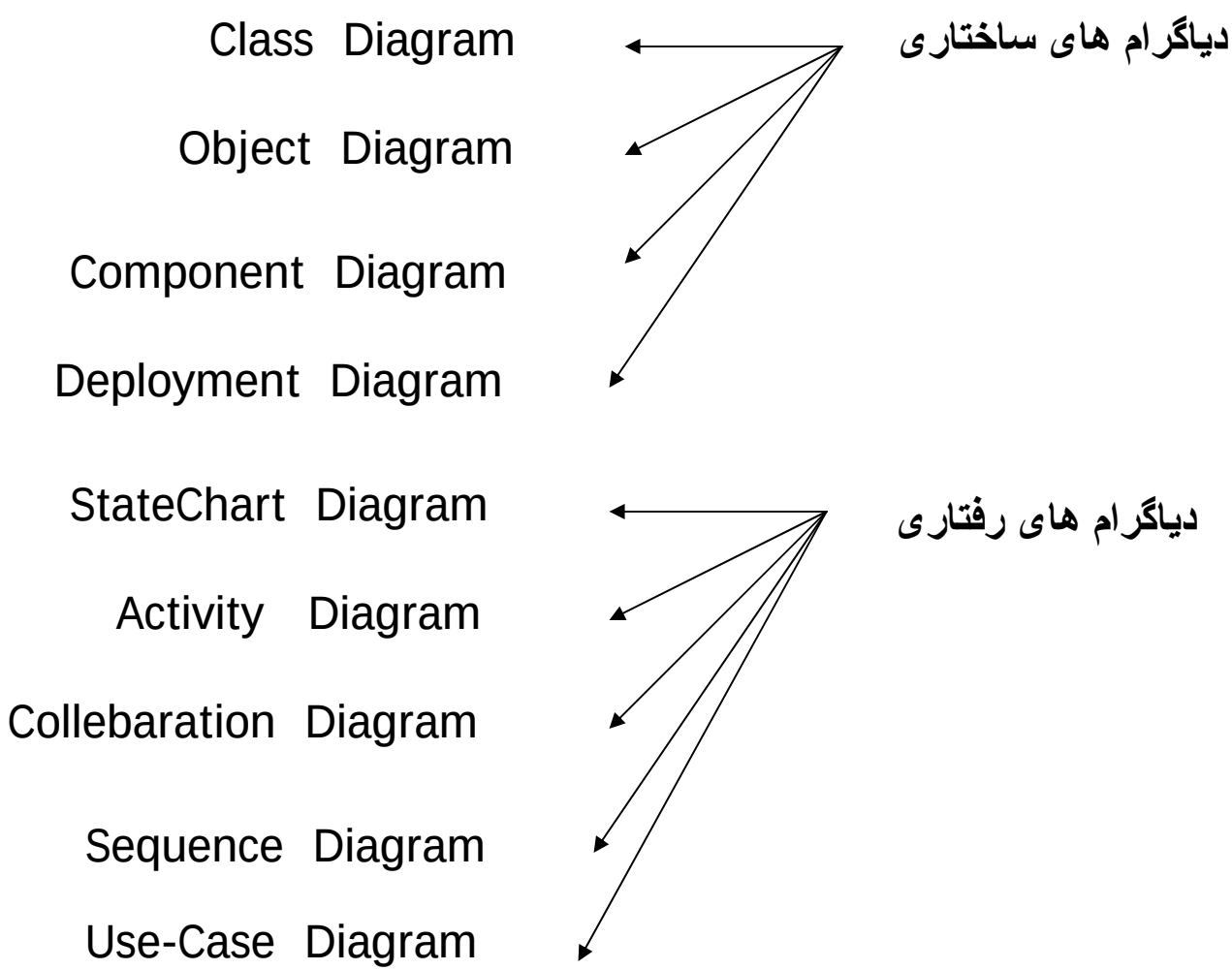
- **معماریهای اثبات شده برای طراحی نرم افزار**
- **کاهش فرآیند پیچیدگی**
- **نشان دادن اشکالات موجود در طرح**
- **لغت نامه مشترک در طراحی**
- **...**

انواع نمودار (دیاگرام ها) موجود در UML :

به دو دسته کلی تقسیم بندی می شوند

۱- ساختار سیستم (ساختاری)
۲- رفتار سیستم (رفتاری)

Diagrams


Class Diagram
Object Diagram
Component Diagram
Deployment Diagram
دیاگرام های ساختاری
StateChart Diagram
Activity Diagram
Collebaration Diagram
Sequence Diagram
Use-Case Diagram
دیاگرام های رفتاری

کلیه دیاگرام های ذکر شده در فصول آتی بصورت مفصل شرح خواهیم داد. اما خلاصه ای از برخی دیاگرام های عمده را برای ایجاد ذهنیت ذکر خواهیم نمود .

Class Diagram

در این دیاگرام ما کلاس ها و روابط بین آنها خواهیم داشت .

Object Diagram

این ساختار به مانند عکس گرفتن از اشیا می باشد که برای نمایش دادن اشیا موجود در یک لحظه از سیستم می باشد .

Component Diagram

نمایش بسته ها و اجزایی به مانند کلاس ها در آن می باشد و مزیت آن کاهش پیچیدگی سیستم می باشد.

Deployment Diagram

نمایش دادن وضعیت نهایی عملیاتی نرم افزار و نمایش دادن زیر ساختار هایی که در آینده قرار است این نرم افزار روی آن نصب و اجرا شود و سخت افزار و حافظه در کجا از این منبع اجرا خواهند شد برای ما مدل می کند .

StateChart Diagram

دیاگرام وضعیت برای نمایش دادن وضعیت اشیا موجود در سیستم

Activity Diagram

نمودار فعالیت که رفتار یک شی را مدل می کند و معادل همان فلوچارت است ولی با یکسیری تفاوت جزیی در ساختار و هدف

Use-Case Diagram

لیست تمامی کارهایی که یک کاربر قرار است انجام دهد با این سیستم نمایش می دهد .

Collaboration Diagram

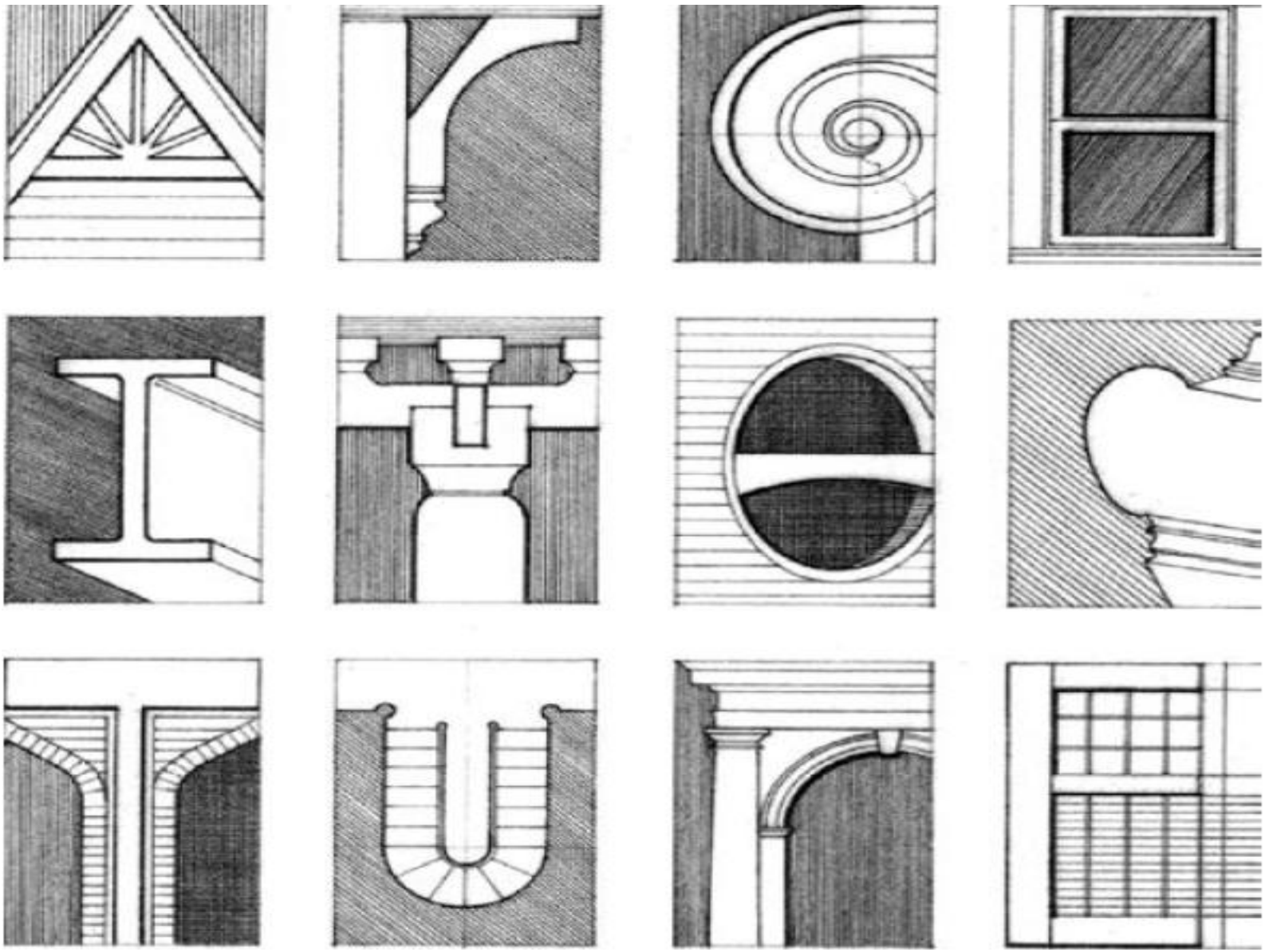
نمودار همکاری برای شفاف کردن ارتباط اشیا و تاثیر آنها روی هم می باشد .

Sequence Diagram

نمودار مرحله یا فاز برای نمایش دادن زمانبندی نمودار ها که در یک موقعیت با توجه به زمانبندی باید چه کارهایی انجام شود .

بغیر از ۹ دیاگرام ذکر شده سه دیگرام دیگر نیز وجود دارد که از مهمترین این دیاگرام ها Class دیاگرام می باشد .

Part I: Getting Started



فصل اول

مدل

Why We Model ?

تاریخچه UML

از دهه ی ۷۰ تا ۸۰ متدلوژی بوجود آمد و در ۸۰ تا ۹۴ حدود ۴۰ متدلوژی مختلف در دنیا ایجاد شد که این متدلوژیها متناسب با نیاز ها می بود سه تا از معروف ترین این متدلوژی ها ، متدلوژی Booch و OOSE و OMT که هر کدام در یک قسمت قوی بودند که در نهایت با پیوستن هر سه بهم متدلوژی UML ارائه شد و چیزی که مد نظر ماست درمتدلوژی UML نقطه قوت Booch می باشد .

هر سه نفر برای ایجاد UML اهدافی از قبیل زیر دنبال می کردند :

۱) ایجاد ابزار مدلسازی برنامه نویسی شی گرا

۲) قابل توسعه بودن

۳) ایجاد مدلی ساده به نحوه ای که قابل فهم انسانها و ماشین ها باشد .

اولین نسخه ی UML که به نام UM بود در سال ۱۹۹۵ بوجود آمد که خود UML در نسخه های بعدی ایجاد شد .

شرکت OMG با دادن پیشنهادی به این گروه سه نفره و با پذیرفتن آنهادر سال ۱۹۹۷ نسخه ی UML 1.0 ایجاد شد و این کتاب در مورد UML نسخه ۱,۳ می باشد .

چرا ما مدلسازی نیاز داریم ؟

با یک مثال ساده این مسئله را توضیح خواهیم داد به فرض اگر بخواهیم برای سگ مان یک کلبه بسازیم و در نهایت روز به روز بدون طرح و برنامه ی اساسی بخواهیم این کلبه را توسعه دهیم تا اینکه به یک آپارتمان تبدیل شود مسلما در نهایت این بنا ساخته شده خراب خواهد شد .

در مورد نرم افزار نیز به همین شکل است اگر بدون مدل و طرح اولیه بخواهیم نرم افزار ساده ی تولید شده را توسعه دهیم و در نهایت این نرم افزار به جایی خواهید رسید که درون خود خراب خواهد شد .

بنابراین باید یک مدل از پیش تعیین شده ای برای آن وجود داشته باشد که بتوانیم باصرف کمتر اتلاف وقت مدل خود را توسعه داده که در آینده برای آن مشکلی پیش نیاید .

مدل چيست؟

A model is a simplification of reality.

مدل يك ساده سازي از واقعيت است.
جدا سازي يك جزييات بي تاثير مسئله و تاكيد زياد روي مسايلي كه تاثير زيادي روي برنامه ما دارند.

✓ هدف از مدل سازي :
مدل ها را انجام مي دهيم تا به يك درك و فهم بهتر ازسيستمي كه با ان كار مي كنيم برسيم.

اهداف مدل سازي:

۱- به ما كمك مي كند تا ان سيستمي را كه مي خواهيم را به تصوير در اوريم.
۲- به ما اجازه مي دهد تا سيستم را به لحاظ رفتاري و ساختاري مشخص كنيم.
۳- يك الگوي است براي طراحي ساختمان سيستممان.
۴- تصميماتي كه در زمان طراحي و تجزيه و ساخت سيستم انجام مي گيرد را مستند مي كند.

اصول مناسب در مدل سازي:

اصل ۱- مدلي كه براي سيستم انتخاب مي شود بايد تاثير مستقيمي روي نتيجه جوابي كه بدست مي ايد داشته باشد.

اصل ۲-هر مدل مي تواند در سطوح مختلفي براي بيان جزييات باشد.

اصل ۳-بهترين مدل ان مدلي است كه منطبق و متصل با واقعيت باشد.

اصل ۴-معمولا با يك مدل به جواب نمي رسيم و بايد با همكاري و كمك چندين مدل به نتيجه رسيد .

✓ **در سيستم هاي شي گرا به ۵ ديد نياز داريم:**

۱- ديدي است مبتني بر نيازمندي هاي سيستم .

۲- فضاي طراحي و قابل حل را براي ما توصيف مي كند .

۳- فرايند ها را مدل سازي مي كند .

۴- در پياده سازي قطعات فيزيكي به ما كمك و انها را مدل سازي مي كند .

۵- محيط ساخت و عملياتي را مدل سازي مي كند.

در تجزيه و تحليل دو ديدگاه وجود دارد :

١ - شي گرايي

٢ - ساخت يافته

فصل دوم

مقدمه UML

Introdaction the UML

✓ UML زباني است براي:

۱ - به تصویر در آوردن Visualizing

۲ - مشخص کردن Specifying

۳ - ساختن Constructing

۴ - مستند کردن Documenting

UML زبان مدلسازی نرم افزاری برای نمایش این لغت نامه و قواعد ی که برای در کنارهم قرار دادن این اجزا می باشد .

۱- به تصویر در آوردن (visualizing)

ايراد هاي موجود در اين بحث در رابطه با عدم عدم استفاده از UML شامل:

۱ - در ذهن خود مفاهيم و كد ها را در نظر گرفته ايم.پس انتقال مفاهيم با استفاده از كد به ذهن ديگران نكته منفي و ايراد بدي است.

۲ - مفاهيم را بايد غير مستقيم استنتاج كرد و اين ايراد ديگري است.

۳ - راه حلي كه در ذهن مي باشد به صورت كد تبديل شده كه بعد از مدتي همه داده ها به خاطر خلاصه شدن در قالب كد از ذهن فراموش شده و از ياد مي رود.

در نتيجه ما به زبانی نياز داريم كه برای قابل درک و فهم كردن كد و به تصوير كشيدن آن كارما را راحتتر كند .

۲ - مشخص کردن:

سيستمي كه با UML مشخص مي كنيم بايد :

۱- **دقيق** ۲- **غير مبهم** ۳- **كامل** باشد.

اين قسمت به دو دسته تقسيم مي شود:

۱ - اشيا ساختاري

۲ - اشيا رفتاري

۳ – ساختن :

اگر UML ما در ساخت سيستم به ما كمك نكند ديگر UML به حساب نمي اید .

پس در اينجا ما يك Maping داريم كه به ما دو امكان را مي دهد :

۱) مهندسي رو به جلو

forward engineering : يعنی توليد كد از مدل ايجاد شده UML

۲) مهندسي معكوس

Reverse engineering : يعنی از مدل به كد برسيم .

✓UML آنقدر سريح و غير مبهم است كه سه امكان را به ما مي دهد :

۱) اجراي سيستم در مراحل طراحي وRUN
۲) شبيه سازي سيستم
۳) سيستم ساخته شده را از نظر كيفي اندازه گيري مي كند.

۴ - مستند كردن :

بايد دستور العمل ها وكد نويس هاي خود را به صورت مكتوب نوشت و اين كار UML مي باشد.

✓ مسائل مکتوب(ویژگی مستند سازی) در UML شامل ۸ قسمت مي باشد :

۱ – **Requirment :** نیاز ها را در اورده و در برنامه ضمیمه مي کند .

۲ – **Architecture : باید دیدي از معماري و طرح کلي سیستم را داشته باشیم .**

۳ – **Design : روابط و نیازها را در بعد طراحي مکتوب مي کند .**

۴ - **Source Code : در کنار هر کد دستور العمل ها و Help ها را باید داشته باشیم .**

۵ - **Project Planes : دسته بندي کارها و اجرایي پروژه را انجام مي دهیم .**

۶ - **Tests : سیستم از این تست ها کمك گرفته و انجام شده .**

۷ – **Prototyles : یك مقدمه و امکانات نیمه اماده روي سیستم و برنامه خود گذاشته و مکتوب می کند .**

۸ – **Released : نسخه هایي که از نرم افزار بیرون مي اید را مکتوب مي کند .**

برای شناخت UML نیاز به شناخت سه مورد زیر داریم :

1. شناخت اجزا و مولفه های تشکیل دهنده Building Blocks
2. قواعد بین اجزا
3. مکانیزم کنترل زمان

توجه : تمامی زیر ساختار UML به صورت درختی سطح بندی شده توضیح داده شده است که به ترتیب سطح رنگ شماره گذاری تغییر می کند به این شکل که :

1. سطح اول
2. سطح دوم
3. سطح سوم
4. سطح چهارم
5. سطح پنجم

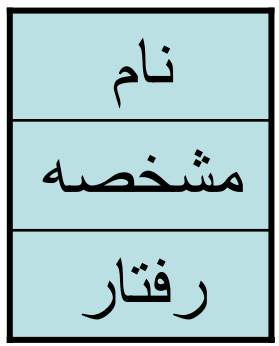
1. **شناخت اجزا و مولفه های تشکیل دهنده**
   - 1. **اشیا**
   - 2. **روابط**
   - 3. **نمودار**

چهار دسته

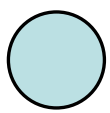
- 1. **رفتاری**
- 2. **ساختاری**
- 3. **دسته بندی**
- 4. **نمادهای مستند سازی**

اشیا ساختاری معادل اسم کلاس در UML

1. **کلاس Class:**
توصیف یك مجموعه اشیا



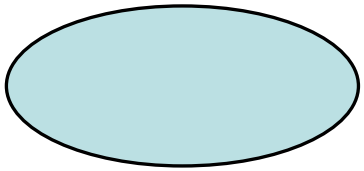
2. **رابط ها Inteface :** مجموعه ای از عملیاتی است که یک کلاس یا یک سرویس ویا یک مولفه و متد ها را مشخص می کند .



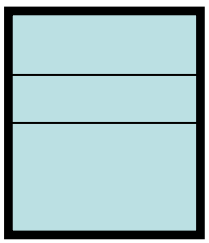
3. **همکاری Collaboration :** تعریف کننده تعامل است که این تعامل شامل **مجموعه ای از قواعد دیگر مولفه هاست که با هم تشکیل دهنده رفتاری است که رفتار بزرگتر از رفتار همه مولفه هاست.**

نمایش رفتار و مدل کردن آن

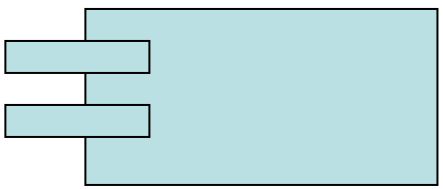
...



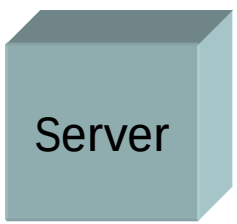
4 **Use-Case** : توصیف مجموعه ای از عملیات که سیستم انجام می دهد برای اینکه نتیجه قابل مشاهده به یک عامل خاص **Actor** برساند .



5 **Active Classes** : کلاس هایی هستند که دارای یک یا چند فرآیند یا رشته های اجرایی می باشند واز این رو می توانند شروع کننده یک فعالیت کنترلی باشند .



6 **Component** : مولفه شی زمان اجرا است (**Package** مفهوم زمان طراحی ست ولی مولفه مفهوم زمان اجراست و قابل تغییر می باشد) مولفه ، یعنی یک جز فیزیکی و قابل تعویض از سیستم که تبعیت می کند از یکسری رابط ها و پیاده سازی می کند .



7 **Node** : یک جز فیزیکی از زمان اجرا وجود دارد و نشان دهنده یک منبع محاسباتی و توان پردازش داراست چنین سیستمی مقداری حافظه خواهد داشت .

...

❶ اشیا

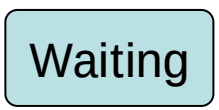
❷ ساختاری

❶ رفتاری: **(معادل فعل ها در UML)** نشان دهنده قسمت های دینامیک (پویا) یعنی حرکت ها مدلها و پیامهایی که ردوبدل می شود را مدل می کند .

❶ **Interaction : (تعاملات و پیامهایی که بین اشیا ردوبدل می شود)**

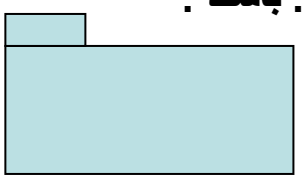
یک تعامل یک رفتار که تشکیل شده از مجموعه ای از عوامل پیغام میان مجموعه ای از اشیایی که داخل یک مجموعه مشخص قرار دارند و برای رسیدن به هدف معین می باشد .

❷ **State Machine : یک ماشین وضعیت یک رفتار که مشخص کننده یک توالی از وضعیت از یک شی یا Interaction طی خواهد کرد و در طی حیاتش برای پاسخ به رویدادها .**



❸ **دسته بندی : مهمترین شی دسته بندی package می باشد.**

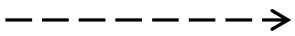
Package : یک مکانیزم همه منظوره است برای سازمان دهی عناصر در داخل یکسری گروهها این اشیا می توانند اشیا رفتاری یا ساختاری یا ... باشند .



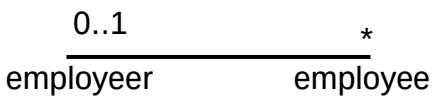
❹ **نمادهای مستند سازی: قسمت توضیحی UML**

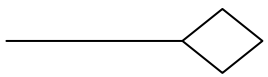
❷ روابط

❶ وابستگی Dependency : یک رابطه ی مفهومی بین دو شی می باشد که در حقیقت تغییر در یک شی تاثیر می گذارد روی تعریف شی دوم

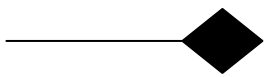


❷ همکاری Association : یک رابطه ساختاری که توصیف کننده بین پیوندهاست.



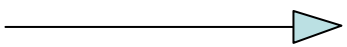


❶ تجمع aggregation

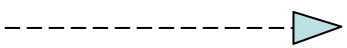


❷ ترکیب

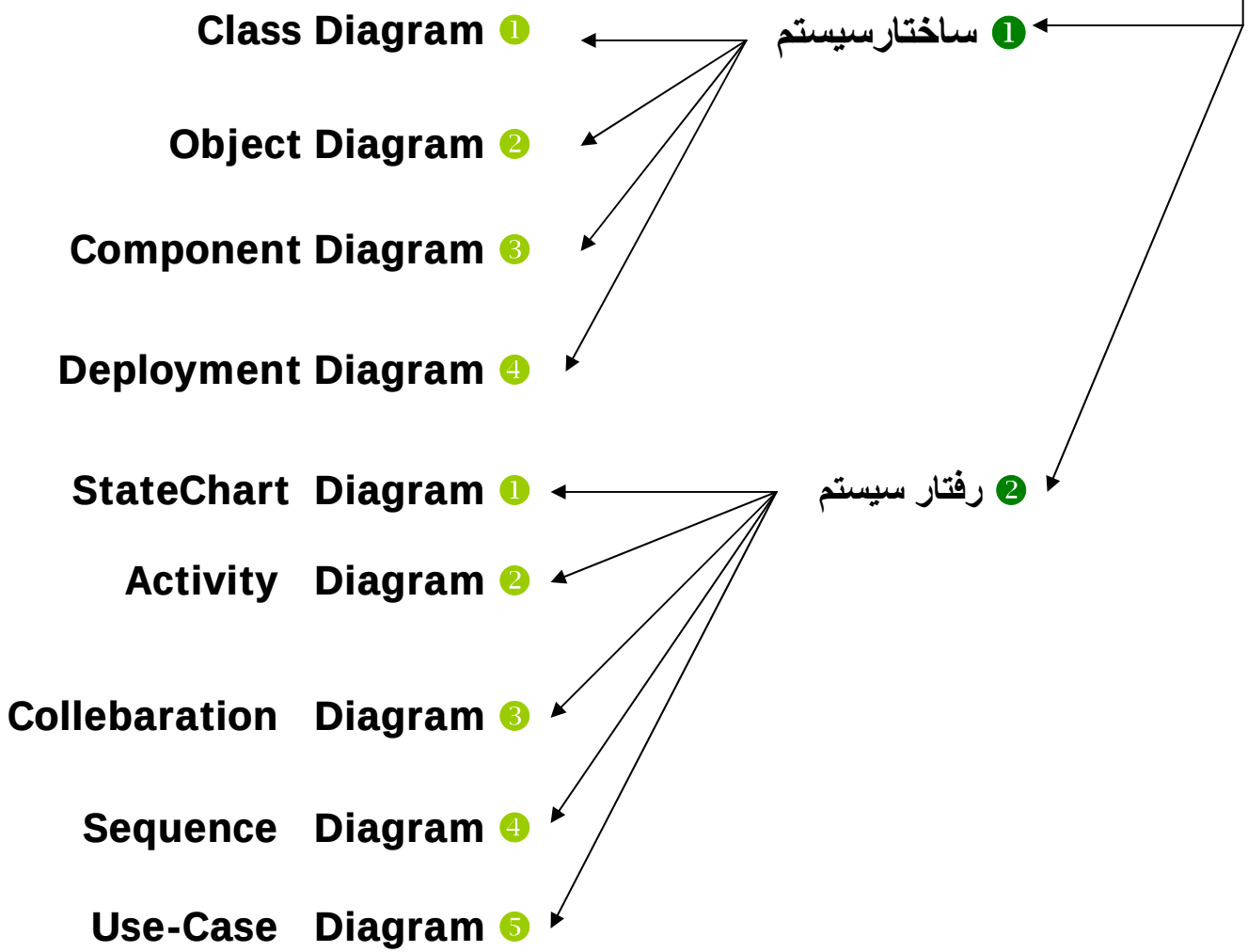
❸ عمومیت دادن Genralization : رابطه بین Sub کلاس و Super کلاس ، Sub کلاس ها نوع خاصی از Super کلاس ها و Super کلاس ها نوع عامی از Sub کلاس هستند . رابطه تعمیم مثل حالت وراثت میباشد بین شي پدر و شي فرزند



❹ به واقعیت در آوردن Realization : یک رابطه مفهومی میان تعریف کننده های کلاس یک کلاس یک Interface را پیاده سازی و به واقعیت در می آورد .



3 نمودار : منظور از نمودار نمایش تصویر مجموعه ای از اجزا معمولا بصورت یک گراف نمایش داده می شود لبه یا Node آن اشیا می باشند و خطوط ارتباطی روابط هستند .



❷ قواعد بین مولفه ها در UML

ما در مرحله قبل اجزا و مولفه ها را معرفی کردیم در این بین ما نیاز به قواعدی بین این اجزا داریم .

این قواعد در واقع مانند هر زبان دیگر UML نیز دارای یکسری قواعد بوده که اگر رعایت شوند مدل ما خوش ترکیب خواهد بود .

مدل خوش ترکیب : مدلی است که به لحاظ مفهومی خود سازگار (بین اجزا داخلی با هم سازگارند)

یعنی دارای تناقض داخلی نباشد و هماهنگ است با بقیه مدلهای مرتبط

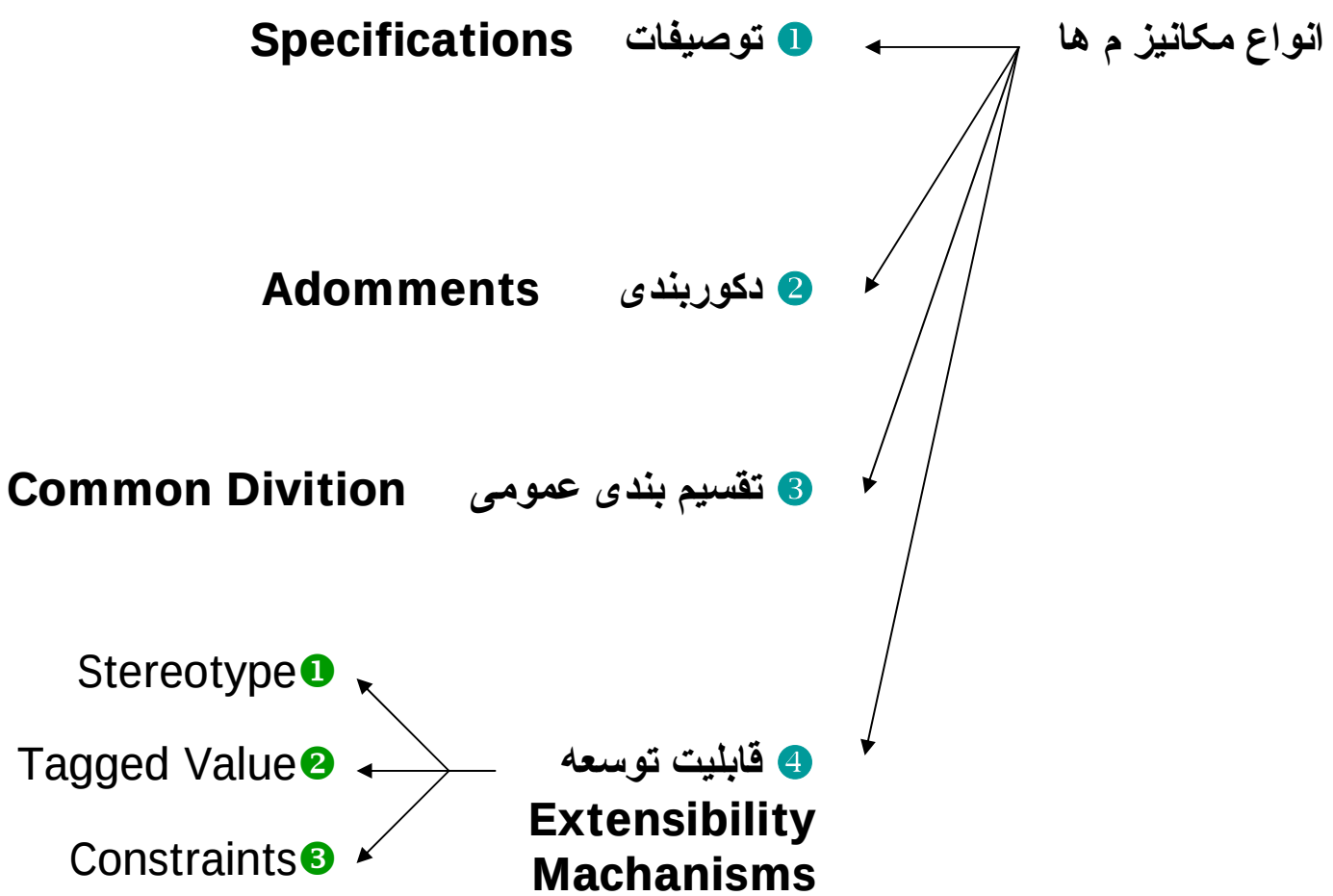
Name: آنهايي است كه شما مي توانيد براي اشيا روابط وديا‌گرام ها بناميد.
Visibility: قابليت دسترسي يك سري قاعده مي باشد.
Integery: اگر يك شي از كلاس ديگر مي سازيم بايد آن كلاس وجود داشته باشد.
Execution: ناظر بر اين كه يك مدل با چه نحوي ما تواند باشد تا اجرا شود.

آيا هميشه نمودارهايمان يک مدل خوش ترکيب اند يا خير ؟

سه حالت امکان دارد که اين مدل خوش ترکيب نباشد :

۱- Elided (مخفی)
۲- Imcomplete (ناقص)
۳- Inconsistent (اشکال)

❸ مکانیزم عمومی در UML



توصیفات : ممکن است در یک مدل رسم شده بعضی از اجزای مدل از دید ما مخفی باشند ولی در پس زمینه وجود دارند پس نیاز به جای داریم که این توصیفات و توضیحات از اینگونه قسمت های مخفی و غیر قابل دید ذکر شود .

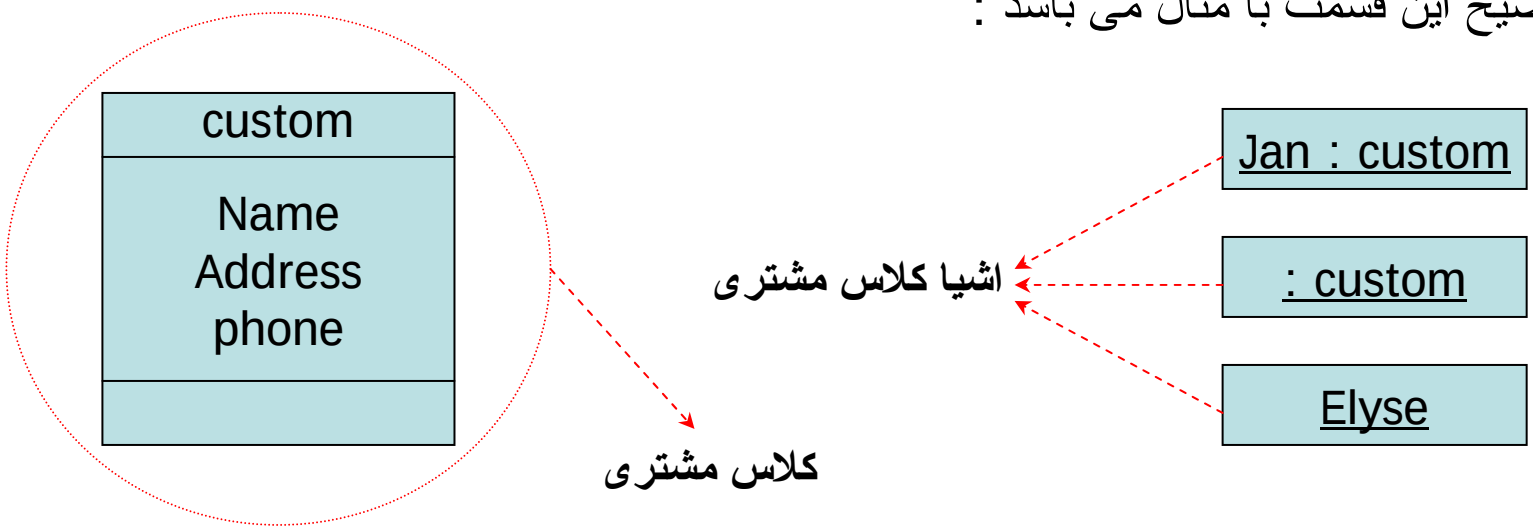
موارد استفاده توصیفات :

- تست خوش ترکیب بودن مدل
- تولید خود کار کد

دکور بندی : نمایش جزییات در مدل و اضافه کردن توصیفات گرافیکی و مستند سازی گرافیکی بر عهده دارد .

تقسیم بندی عمومی : ما مي توانيم به وسيله ان فضاي خودمان را مدل سازي كنيم و اشيا را با توجه به اينكه جزچه كلاسي هستند دسته بندي كرد.

توضیح این قسمت با مثال می باشد :

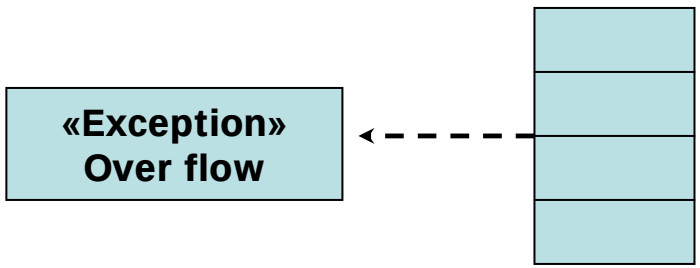


قابلیت توسعه (مکانیزم توسعه) : مثال : خود UML قابل توسعه و بسط می باشد .

انواع مکانیزم توسعه

❶ Stereotype : **توسعه دهنده ی لغت نامه** UML

مثال شی Exception **در جاوا تعریف نشده می توانیم از این طریق به مدل اضافه کنیم .**



❷ Tagged Value : **می توان خاصیت اجزا** UML **را توسعه داد . مثال : ما در مولفه کلاس سه قسمت نام مشخصه و رفتار داریم از این طریق می توانیم خاصیت ورژن (version) به آن اضافه کنیم .**

❸ Constraints : **محدودیت برای توسعه قواعد** UML

معماری نرم افزار :

معماري جمع كردن مجموعه اي از ديدها براي يك طرح كلي از سيستم مي باشد.

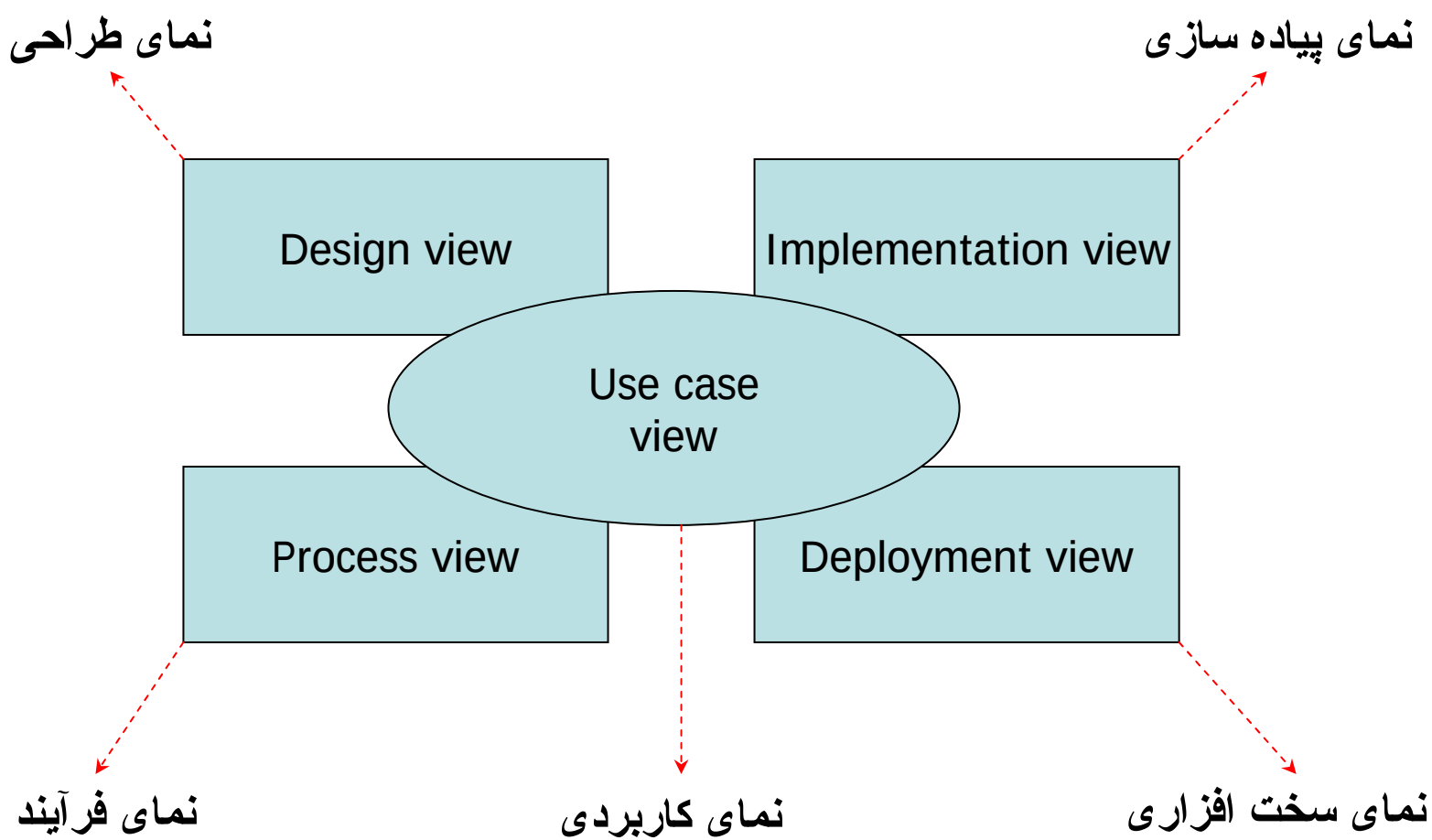
یک مجموعه ی تقسیمات موثر در این اقلام است :

- سازماندهی سیستم نرم افزار (چیدن اجزا کنار هم)
- انتخاب اجزا ساختاری و روابطی که اینها بهم متصل میکند
- رفتار اجزا که مشخص کننده ی همکاری بین اجزا است
- ترکیب این عناصر رفتاری و ساختاری در زیر سیستم های بزرگتر
- (دسته بندی و سازماندهی زیر سیستم ها در یک سیستم)

الگویی که برای سازماندهی انتخاب شده است .

انواع دیدگاه نسبت به نرم افزار :

✓ ما پنج دید نسبت به سیستم داریم :



1. **Use Case view :** این دید کاربردی از سیستم است که این کاربردها توصیف کننده ی رفتارهای سیستم از تظر کاربر نهایی می باشد علاوه ر کابر نهایی تحلیل گر سیستم خود را باید بجای کاربران دیگر بگذارد .

2. **Design view :** در این دید که شامل مولفه های تشکیل دهنده ی لغت نامه ی صورت مسئله پروژه از قبیل کلاس ها رابط ها و همکاری ها و... و راه حلهای مسئله

3. **Process view :** این دید تشکیل دهنده ی فرآیندها و روش های سیستم می باشد و همزمان کردن Thread در این دید نمایش داده می شود (بحث هماهنگی یا Synchronization)

4. **Implementation view :** تشکیل دهنده ای اجزا و فایل های است که از آنها برای ترکیب استفاده می شود .

5. **Deployment view :** دید سخت افزاری از سیستم می باشد . **که مشخص مي كند ساختار سخت افزاري كه اين سيستم روي ان قرار مي گيرد به چه صورت مي باشد و از چه قطعاتي تشكيل شده است.**

چرخه حیات توسعه نرم افزار

فرآیند طراحی از UML جداست .

پرکاربرد ترین نرم افزاری که برای توسعه وجود دارد :

RUP (Retional Unified Process)

UML مستقل از تولید نرم افزار بیشترین فر آیند هایی که می توانند از UML استفاده کنند :

- Use Case گرایش به کاربرد
- Architecture-Centvic معماری محور
- Interative And ineremertal چرخشی و افزایشی

منظور از گرایش به کاربرد : استفاده از Use Case بعنوان ابزار اساسی از بنا کردن رفتار سیستم خواسته شده برای تست و اعتبار سنجی سیستم و ایجاد ارتباط بین اجزای (اعضا) پروژه

منظور از معماری محور بودن : یعنی از معماری سیستم استفاده شود برای ایجاد مفاهیم ساخت و مدیریت و توسعه سیستم در دست ساخت .

منظور از فرآیند تکراری و چرخشی و افزایشی : اینکه بتوان استفاده دوباره از سیستم و توسعه آن .

يعني فرايند بتواند يك جرياني از نسخه هاي قابل اجراي برنامه را توليد كند كه هر نسخه نسبت به نسخه قبلي يك سري امكانات افزوده داشته باشد و نيازمندي هاي بيشتري را پوشش دهد.

مزیت تکراری و چرخشی و افزایشی ریسک گرا بودن نبست آبشاری .

چنین فرآیندی یک فرآیند **فازبندی** شده است .

فاز : محدوده ی زمانی میان دو قسمت اصلی از فرآیند می باشد وقتی یک مجموعه خوش تعریف از اهداف بدست می آید و مجموعه ای از کار ها تکمیل می شود و یکسری تصمیمات اتخاذ می شود که آیا امکان انتقال به فاز بعدی هستیم یا خیر .یا یک مجموعه خوش ترکیب می باشد .

در RUP فرآیند را به ۴ فاز تقسیم بندی می کنند :

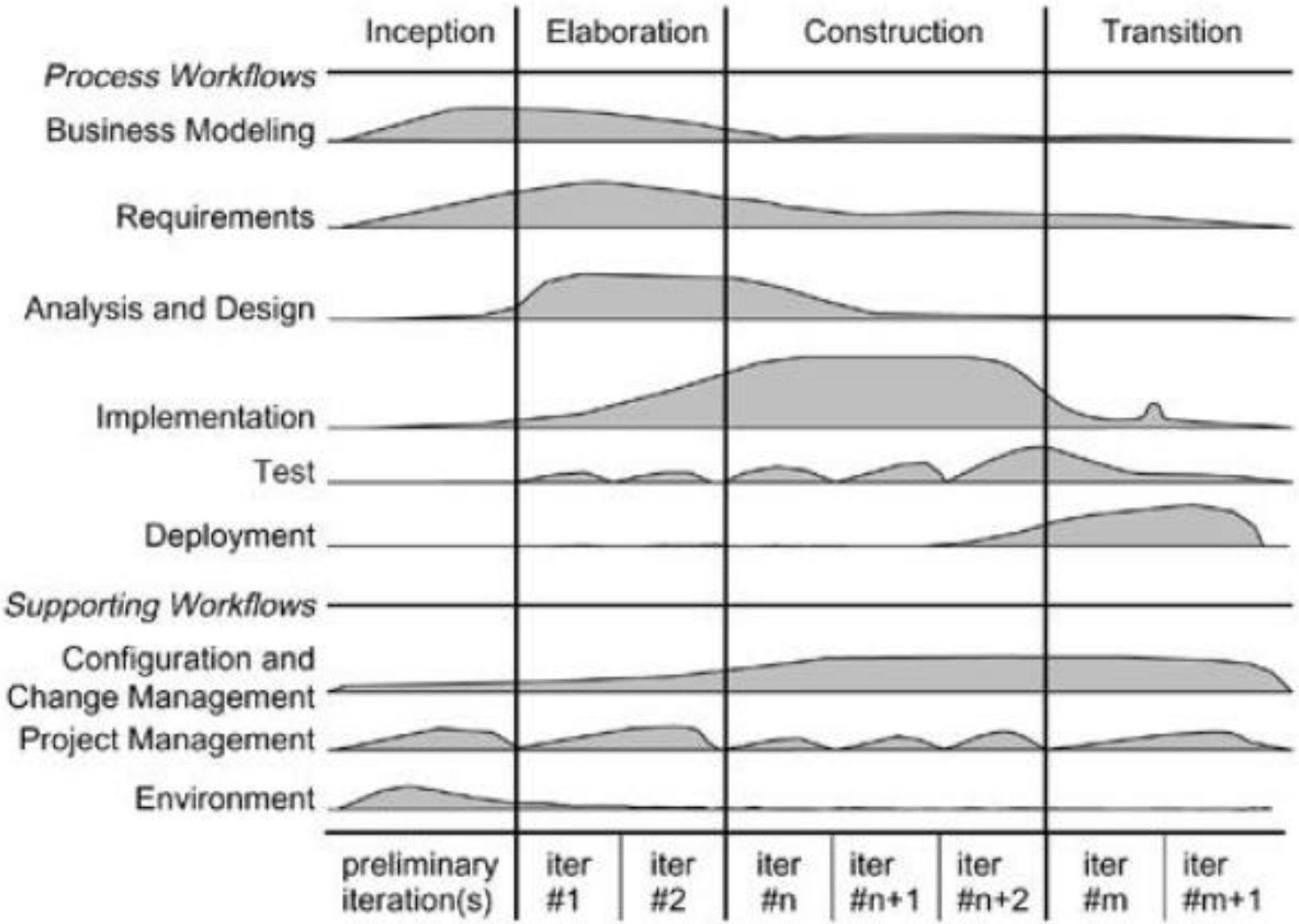
۱- آغازی Inception

2- شناخت Elaboration

۳- ساخت Constraction

۴- انتقال Transition

آغازی شناخت ساخت انتقال



- **اولین فاز فرآیند (آغازین) Inception**

  اولین جرقه ی پروژه زده می شود و پروژه می رود که شکل بگیرد بررسی می شود که این پروژه شدنی است یا امکان آن وجود ندارد .

- **دومین فاز فرآیند (شناخت) Elaboration**

  در این فاز دید از محصول و دید از نرم افزار کامل شده و بر اساس آن معماری شکل می گیرید ولیست نیازمندیها در می آید و اولویت بندی و برنامه ریزی می شود (نیازمندی کلی یا جزی) هر کدام از این نیازمندیها مشخص کننده یک رفتار کار کردی یا یر کار کردی سیستم می باشد .

- **سومین فرآیند (ساخت) Constraction**

  در این مرحله معماری بوجود آمده به اجرا مرحله ساخت برسد . در این قسمت مدیریت پروژه بسیار مهم است .

- **چهارمین فاز فرآیند (انتقال) Transition**

  فازی که نرم افزار به دست کاربر نهایی می رسد . در این قسمت تست انجام می شود و همچنین Deployment نیز صورت می گیرد .

یک چرخشی : یک مجموعه مجزا از فعالیتهاست که با یک برنامه از پیش تعیین شده و یکسری معماری های ارزش یابی است که نتیجه آن نسخه ی جدید از نرم افزار می باشد .

Interation : مجموعه اي از فعاليت ها در يك برنامه مدون و يك سري معيار هاي ارزيابي كه نتيجه اش به صورت يك سري نسخه هاي نرم افزاري نشان داده مي شودكه اين نسخه ها مي توانند براي يك شركت كامپيوتري و يا براي يك كاربر كامپيوتري باشد.

فصل سوم

مثال

Hello, World!

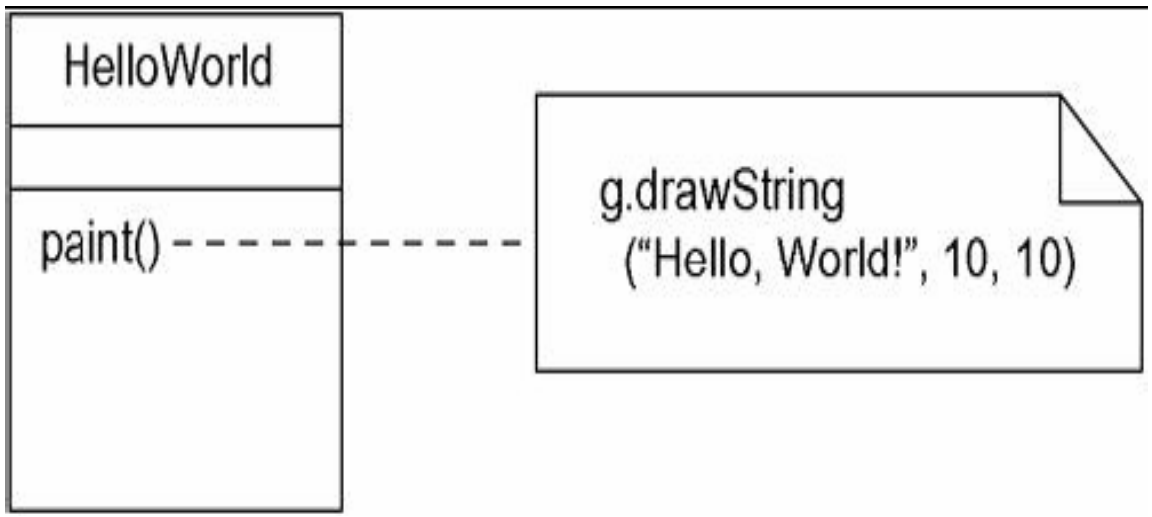
چکیده های کلیدی

در این مثال می خواهیم قطعه کدی به جاوا بنویسیم که:

کد اپلتی بنویسید که در مرورگر وب ای جمله را چاپ کند :

```
import java.awt.Graphics;
class HelloWorld extends java.applet.Applet {
  public void paint (Graphics g) {
    g.drawString("Hello, World!", 10, 10);
  }
}
```

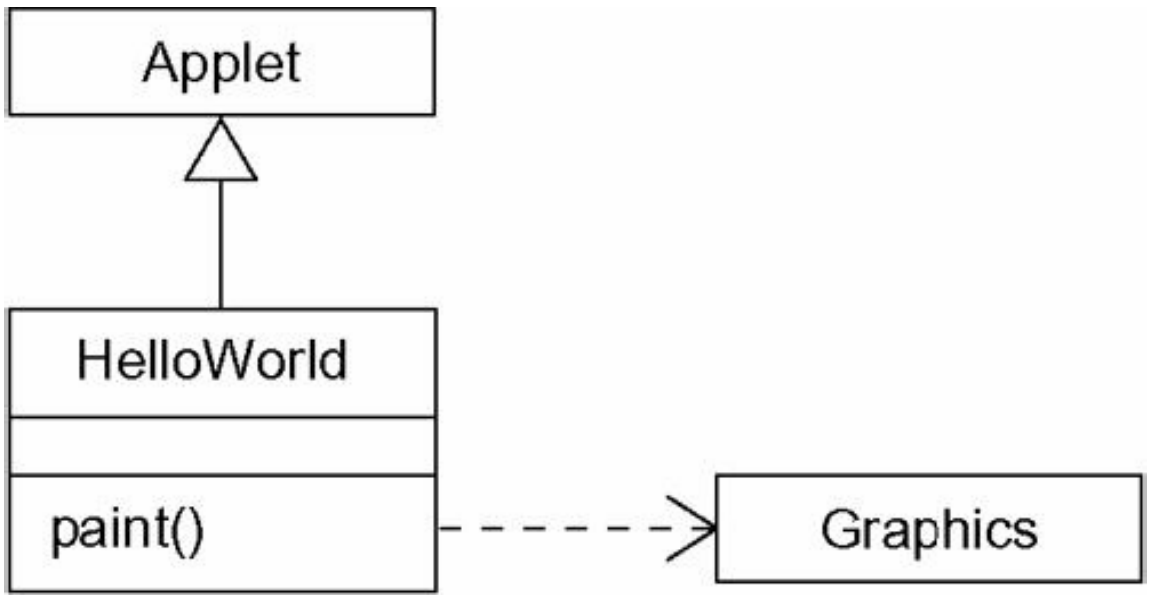
اکنون برنامه ای که نوشته ایم می خواهیم در UML مدل کنیم :



در این مدل ما یک کلاس داریم به نام HelloWorld و دارای یک متد به نام Paint()

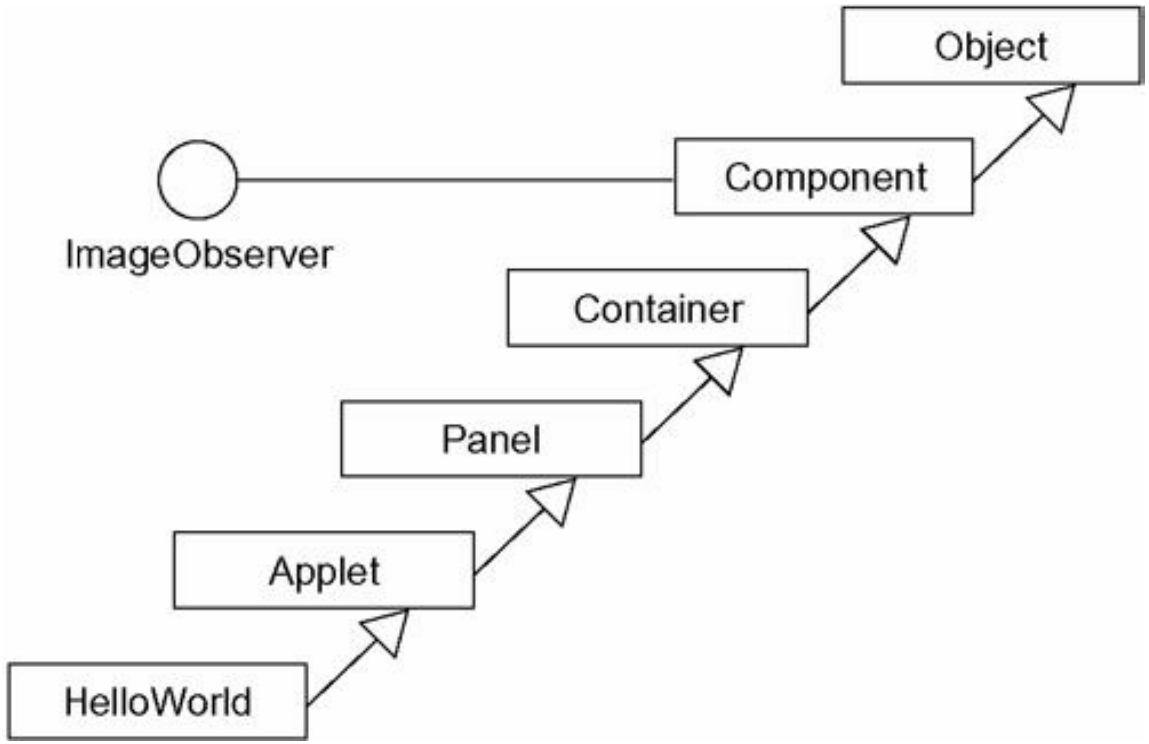
و یک مولفه توضیح داریم که به متد کلاس مرتبط است و نشان می دهد که این متد متن مورد نظر را چاپ می کند .

اکنون ارتباط ها را مدل می کنیم :



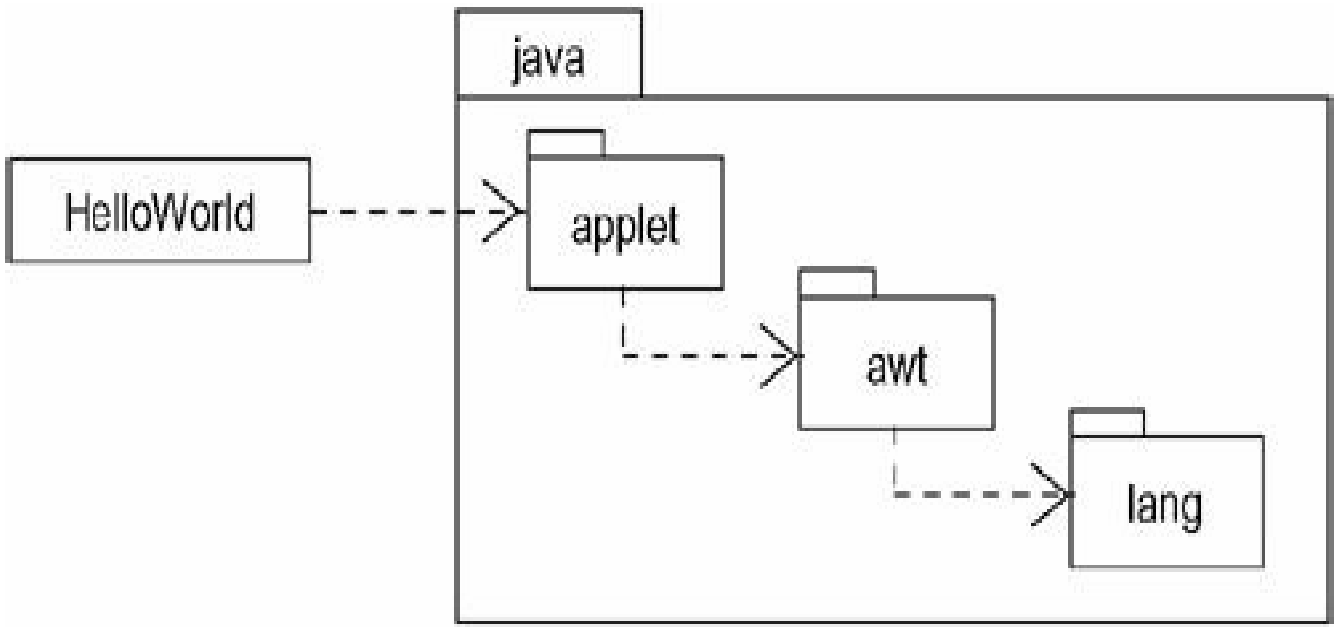
همان طور که در شکل فوق می بینید که کلاس HelloWorld به دو کلاس Applet و Graphics ارتباط دارد ولی نوع ارتباط آنها فرق دارد از یک طرف کلاس HelloWorld فرزند کلاسApplet می باشد و از طرفی دیگر کلاس ما با کلاس Graphics رابطه یوابستگی وجود دارد .به این معنی که کلاس HelloWorld از کلاس Graphics استفاده می کند .

همان طور که در جاوا خوانده ایم در رابطه کتابخانه کلاسها و وراثت بین آنها می خواهیم این وراثت را مدل کنیم در حینی که ببینیم کلاس HelloWorld چه جایگاهی دارد به شکل توجه کنید :



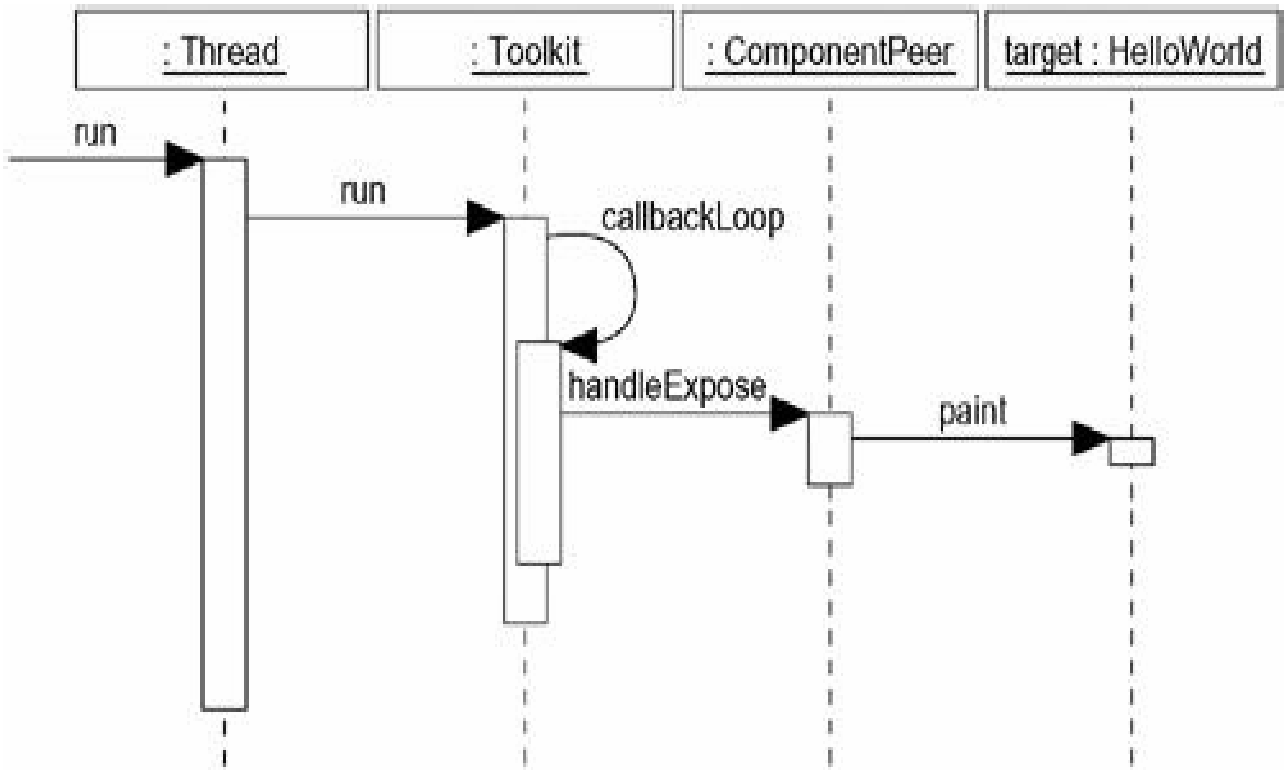
همانطور که مشاهده می کنیم کلاس helloWorld فرزند Applet و Applet فرزند panelوpanel فرزند Container و Container فرزند کلاس Componentکه خود پیاده سازی Interface ،ImageObserver می باشد و از طرفی Component فرزند کلاس پدر Object می باشد .

شما قادر هستید بسته بندی Packaging به تصویر بکشید در کلاس دیاگرام :



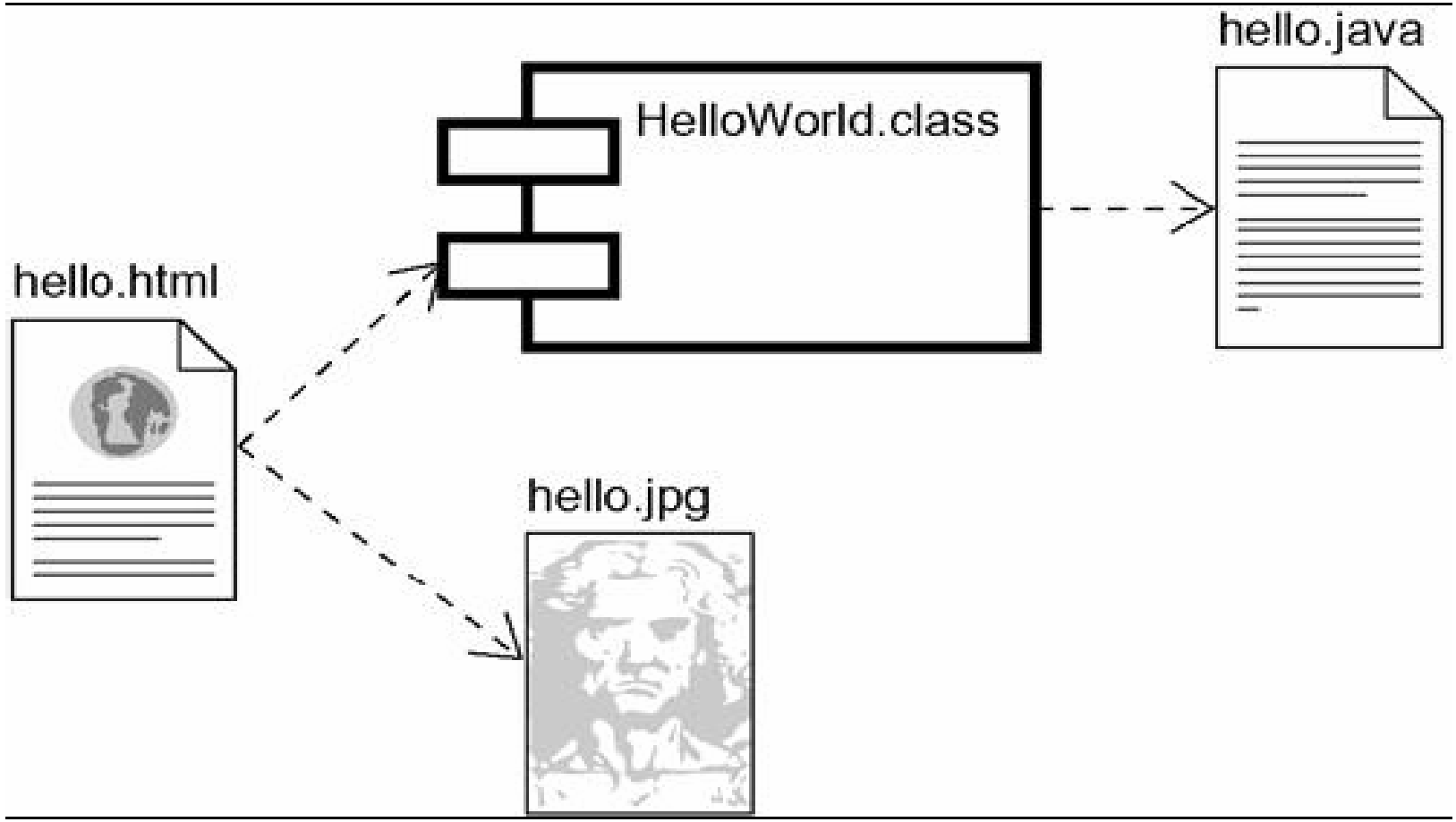
آنچه را که می بینیم کلاس HelloWorld به package Applet وابستگی دارد و از طرفی Package Applet خود نیز بعنوان زیر مجموعه ای از Package Java می باشد و از طرفی دیگر خود Package Appletبه package های awt و lang وابسته است.

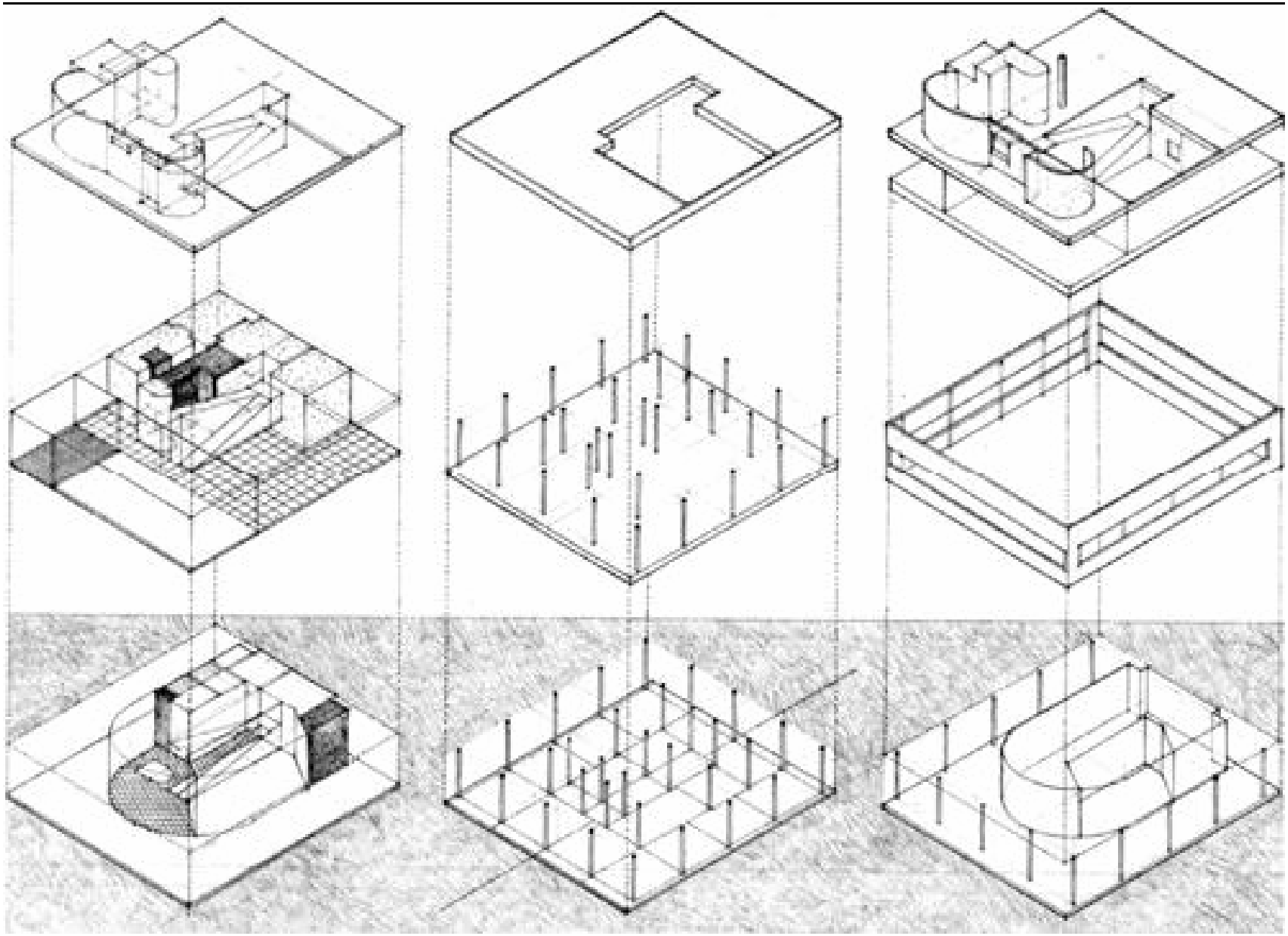
مکانیزم ها



این یک Sequence دیاگرام را نشان می دهد وارد جزئیات نمی شویم فقط بطورکلی یک مکانیزم چاپ راشرح خواهیم داد در این دیگرام ما چهار شی داریم که یکی پس از دیگری اجراشده و این مستطیل های ایستاده نشان دهنده بازه زمانی که یک شی فعالیت می کند .

مولفه ها





فصل چهارم

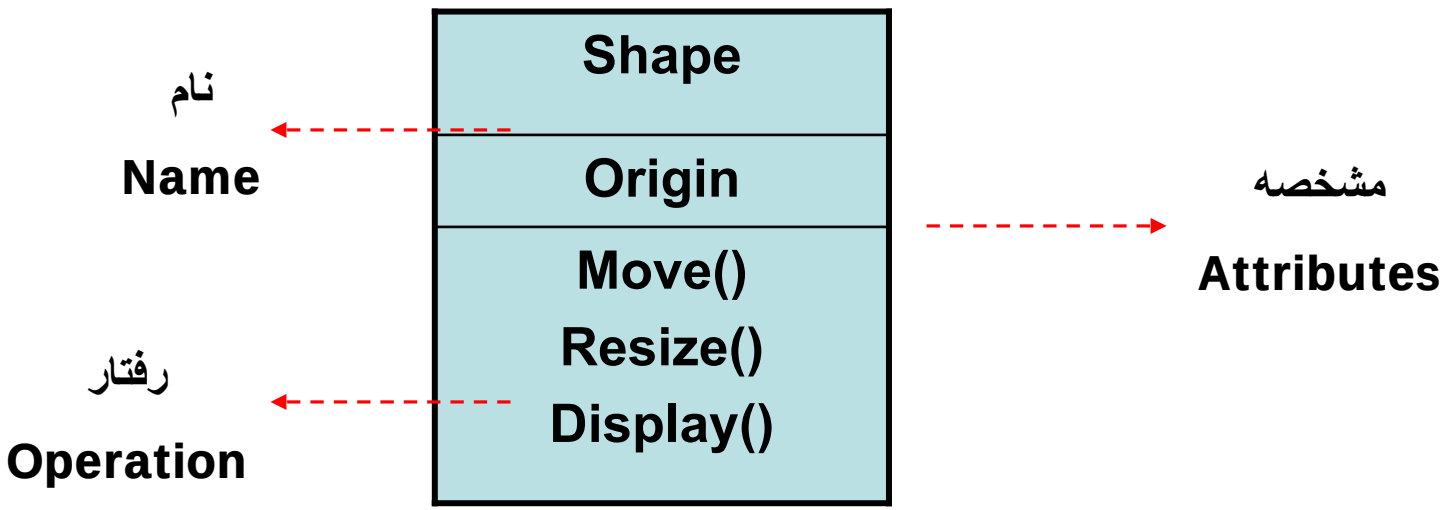
کلاس ها

Classes

کلاس چیست؟

مولفه اصلي سازنده سيستم هاست.نشان دهنده مجموعه اي از objectها كه از نظر Operation رفتار مفهوم و غيره مشابه هستند كه به جاي آن يك كلاس توليد مي شود.

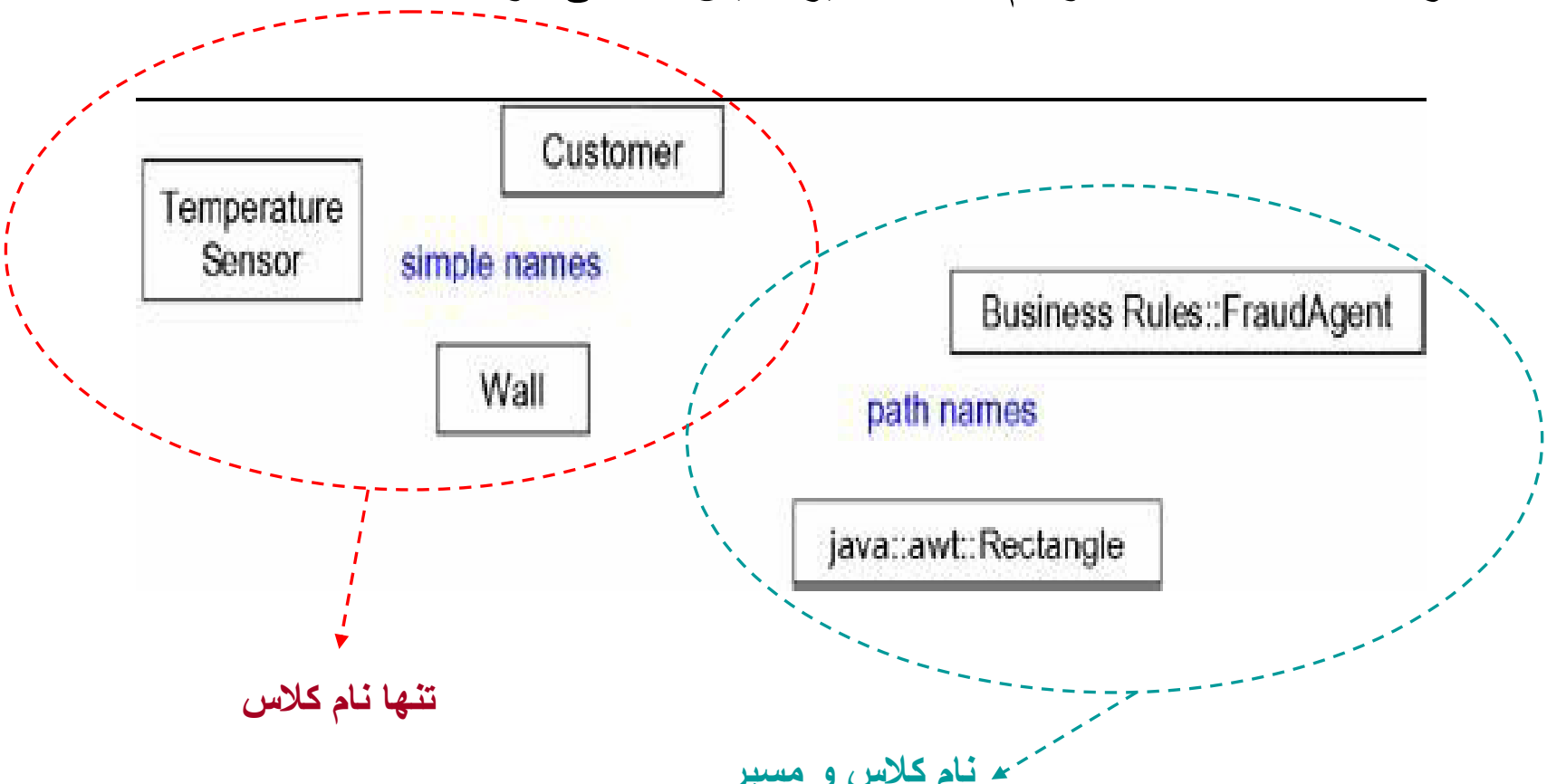
مهمترين اجزاي سيستم هاي شي گراست.اين كلاس توصيف مجموعه اي از اشيا است كه داراي نام ، انجام عمليات و مشخصه ها ارتباطات و مفهوم مي با شد.



کلاس شامل اجزا زیر می باشد :

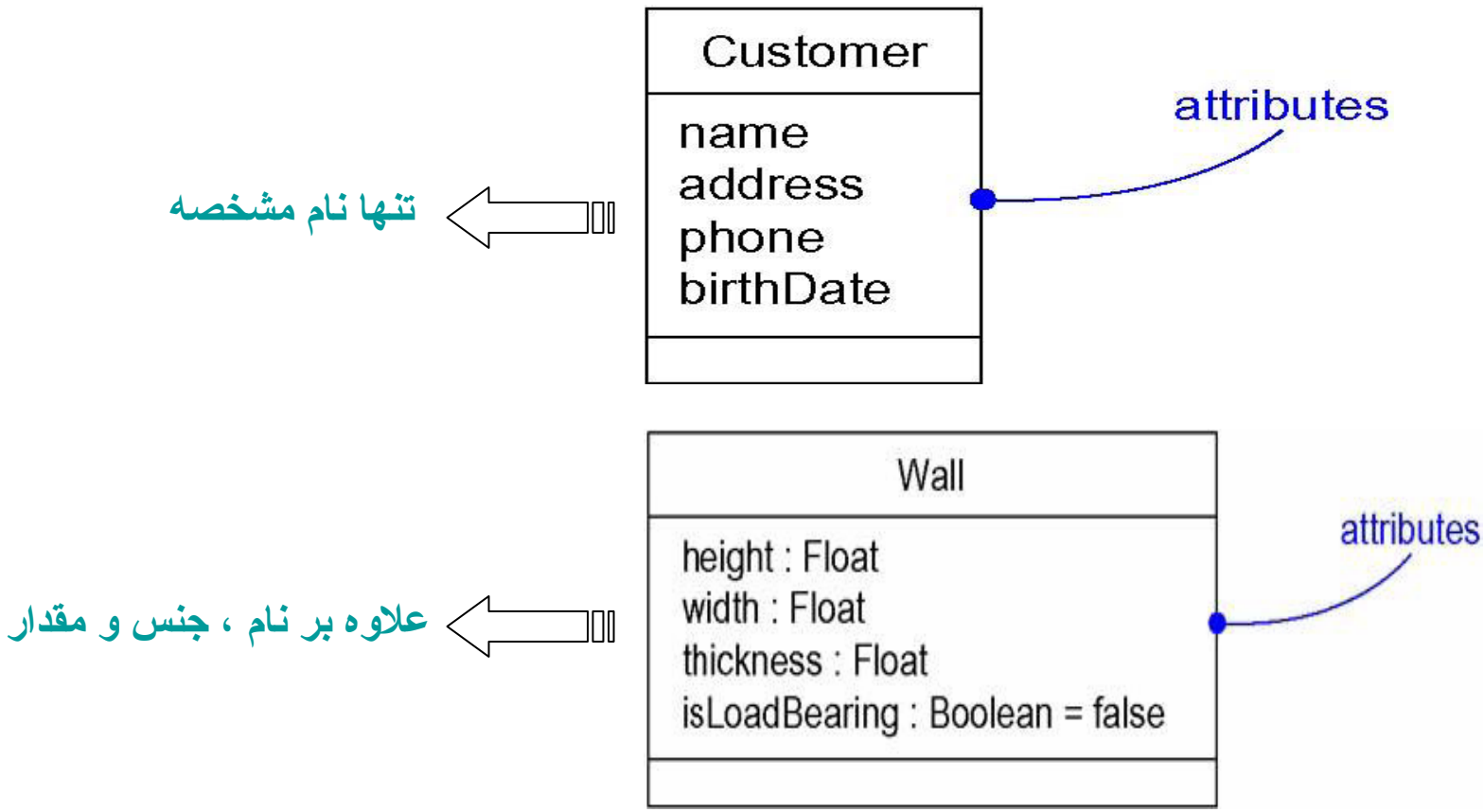
1. Name نام : نام کلاس از یکسری حروف و ارقام وعلائم تشکیل شده که به حروفشروع شده و این حروف قاعدتا با حف بزرگ و بقیه کوچک می باشد .
   نام کلاس که در Package تعریف می شود باید منحصر به فرد باشد .

که در UML به شکل ساده و نام های با مسیر نمایش داده می شوند :



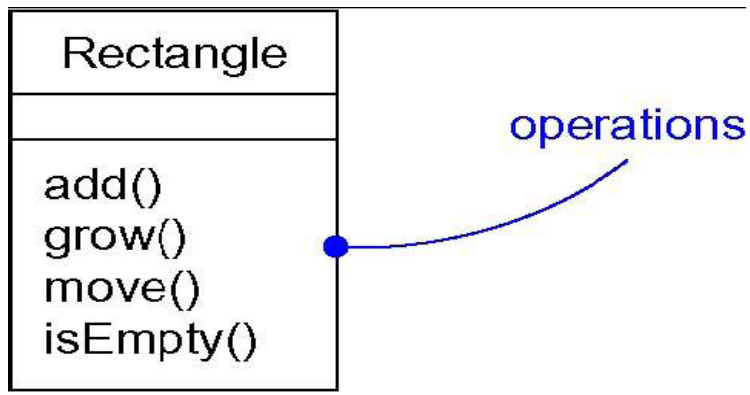
❷Attribute مشخصه ها

مشخصه های یک کلاس می تواند علاوه بر خود مشخصه جنس یا نوع و مقدار نیز در آن مشخص شود.

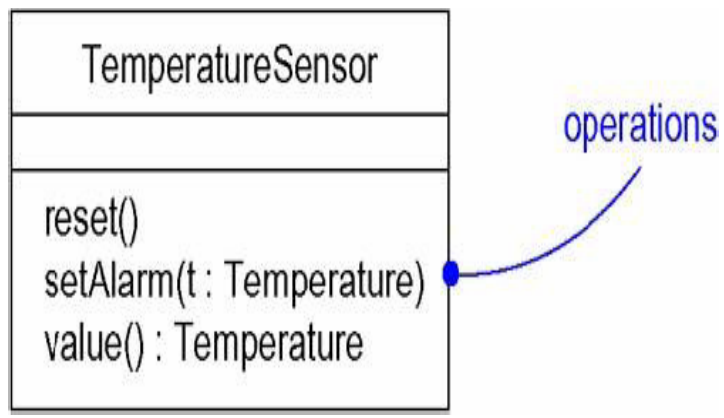


❸Operationعملیات

(رفتارهای کلاس) در اين جا مي توان از اسم متد به اضافه ليست پارامتر ها يا مقدار برگشتي استفاده كنيم.

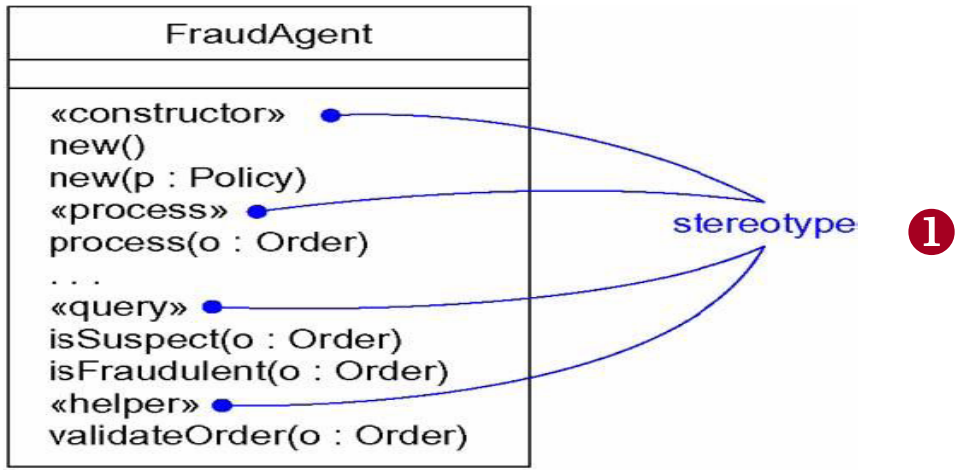


بدون پارامتر

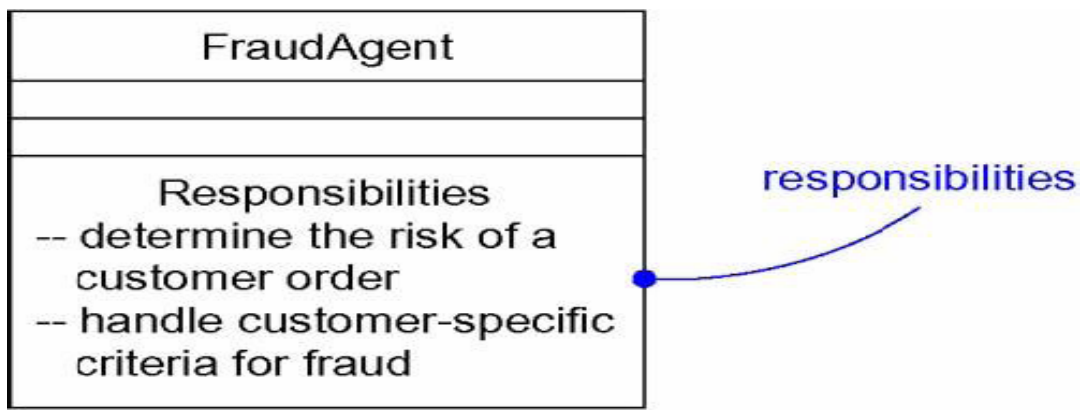


با پارامتر دريافتی و مقدار برگشتی

دو روشی که می توان یک کلاس را از لحاظ دسته بندی توسعه داد :



❷ مي توان يكل كلاس را به چهار قسمت تقسيم كرد كه مسئوليت در قسمت چهارم است.
مسئوليت يا Responsibility هيچ تاثير مفهومي در خود كلاس ندارد و مثل يك Comment مي باشد و فقط يك يادداشت و Node در كلاس مي نويسد.



نكته: يك كلاس حداقل يك مسئوليت و حداكثر ده مسئوليت دارد.وقتي مسئوليت مشخص شود بعد خصوصيات و عمليات مشخص مي شود.

قدم هاي مدل كردن سيستم :

براي مدل كردن لغت نامه سيستم مي بايست قدم هاي زير را اعمال كرد :

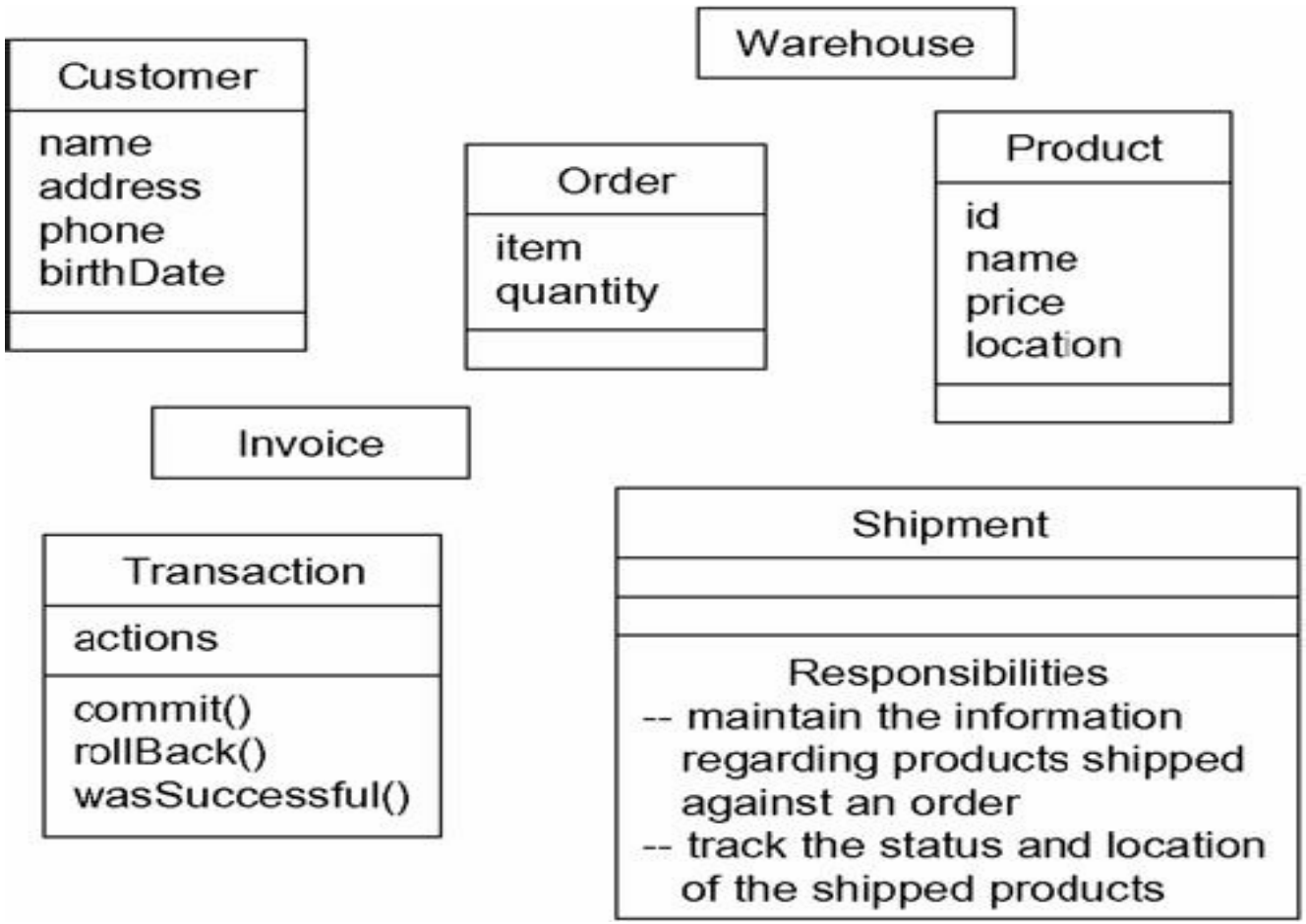
۱- در صورت مسئله به دنبال كلماتي بگرديم كه توصيف كننده خود مشكل و مسئله و راه حل هاي آن باشد.ان ها را پيدا كرده و بعد جدا مي كنيم.با استفاده از Abstraction آنها يي كه لازم است را جدا مي كنيم.(استخراج اسامی کلاس ها و اشیا)

۲- براي هر ديد و Abstraction يك مسئوليت تعريف مي كنيم.در اين جا بايد براي براي يك ديد و كار مجموعه اي از مسئوليت ها را برايش تعريف كرد.بايد مشخص شود مسئوليت هر كلاس چه مي باشد؟

۳- Operation ها و Attribute ها را كه كلاس براي انجام آن مسئوليت به ان نياز دارد را به ان اضاف كرد.

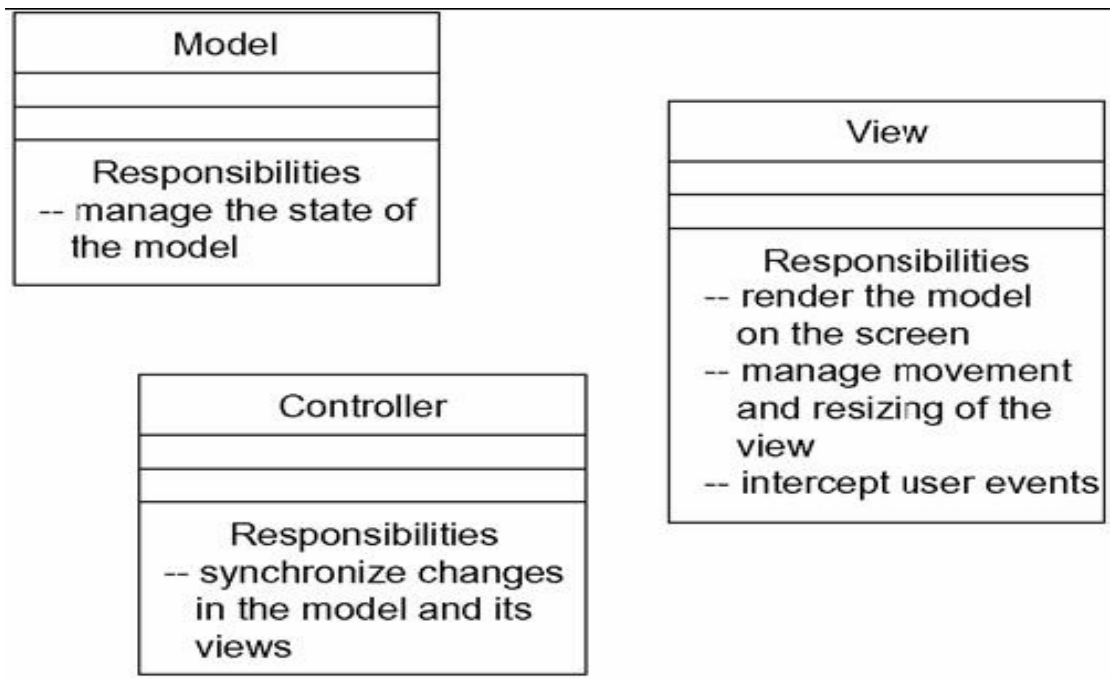
۴- در اخر كلاس هايي كه ارتباط معنايي و مفهومي زيادي با هم دارند را از هم جدا كرده و در يك Packageو دسته قرار دهيم.

مثال : سیستم انبارداری و فروش



توزيع مسئوليت هاي مدل سازي يك كلاس در سيستم :

۱- يك مجموعه اي از كلاس ها را كه با هم كار مي كنند تا يك رفتار مشخص نشان دهند را مشخص كنيم.
۲- مشخص كنيم مسيوليت هر كلاس در ان رفتار چه چيز مي باشد؟
۳- اگر مسئوليت روي يك كلاس شديد باشد به نحوي كه يك مسئوليت مركزي دارد بايد آن كلاس شكسته شود تا مسئوليت تقسيم شود.اگر مسئوليت كن باشد بايد تقسيم را به اعتدال رساند تا به هر كلاس مسئوليت متعادل و دقيقي برسد.



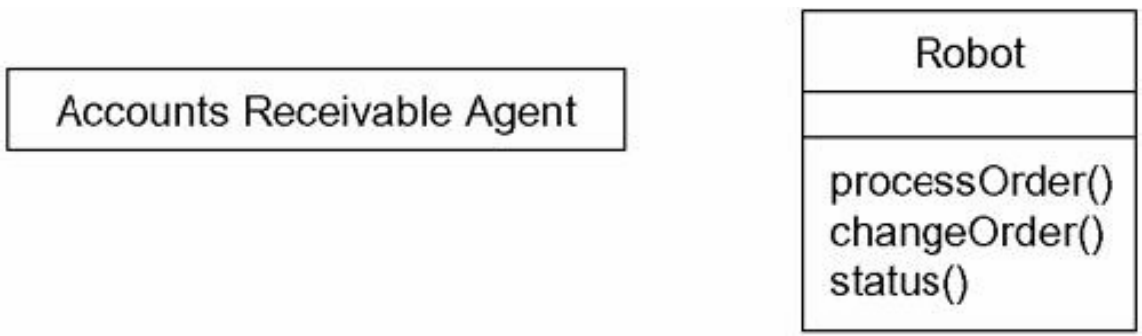
Modeling Nonesoftware things

تعريف اشيا غير نرم افزاری : اشيايی كه در آينده مدل نرم افزاری نخواهند داشت .

براي مدل كردن اشيا غير نرم افزاري چه اعمالي بايد انجام داد؟

۱- مي توان ان ها را به صورت يك كلاس استفاده كرد.
۲- مي توان با استفاده از Stereotype يك بلاك يا مولفه جديد براي نمايش ان شي استفاده كرد(به وسيله دادن icon تصويري ان را در برنامه مشخص كرد).
۳- اگر اشيا ي سخت افزاري مي باشد مي توان به وسيله Node آن ها را مدل كرد.

مثال : شخصی که مسئول بر چسب زدن آدرس پرینت شده از پرینتر بر روی اجناس سفارش شده از طریق اینترنت می باشد .

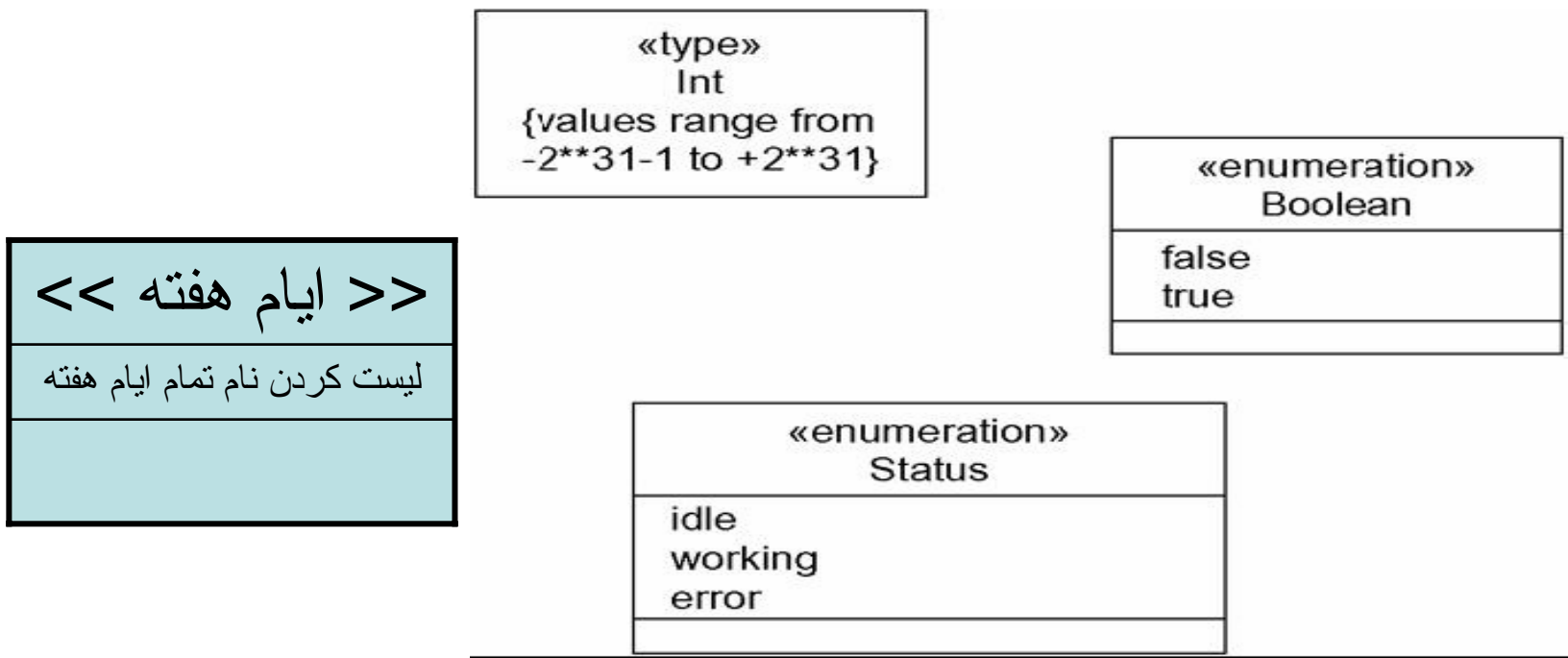


Modeling Primitive types

در مدل كردن نوع هاي اوليه بوسيله دو روش كار را انجام مي دهيم:

۱- با استفاده از Enumerat types كه به سوي Stereotype تغيير مي كند.

۲- با استفاده از Stereotype كه به سوي Enumerat types تغيير مي كند.



- مدل کردن برای انواع شمارشی در جاوا ۱,۵ به نام enum داریم :

(در منبع CookCook)
به شکل زیر داریم :

```
Public  enum  Media {
      book, music,cd,music-viny1, movie
}
```

نتايج و نكات كلي:

*يك كلاس خوي ساخت:كلاسي است كه بايد يك Abstraction يا تجريد باشد از يك متن دقيق يا در يك سوال يا در يك جواب.

*يك كلاس خوش ساخت بايد يك مجموعه كوچك و خوش تعريف از مسيوليت ها را داشته باشد كه بتواند ان ها را به خوبي جواب دهد.

*در يك كلاس خوش ساخت يك خط مشخص براي كلاس بين مشخصات و پياده سازي وجود دارد.

*يك كلاس خوش ساخت بايد قابل فهم و ساده و قابل توسعه و همچنين قابل تلفيق باشد.

براي رسم كلاس در يك نمودار: UML

۱- آن Property ها Operation, ها Attribute ,ها را نمايش دهد.آن Propertyهايي كه لازم و ضروري مي باشد را ذكر نموده جوري كه ان هايي كه لازم است نياز برنامه باشد را نگه دارد.

۲- ليست طولاني ازAttribute ها Operation ,ها را بايد به نوع مديريت در آورد.يا اينكه همه را ذكر نكنيم يا اينكه بياييم ان ها را دسته بندي كنيم.

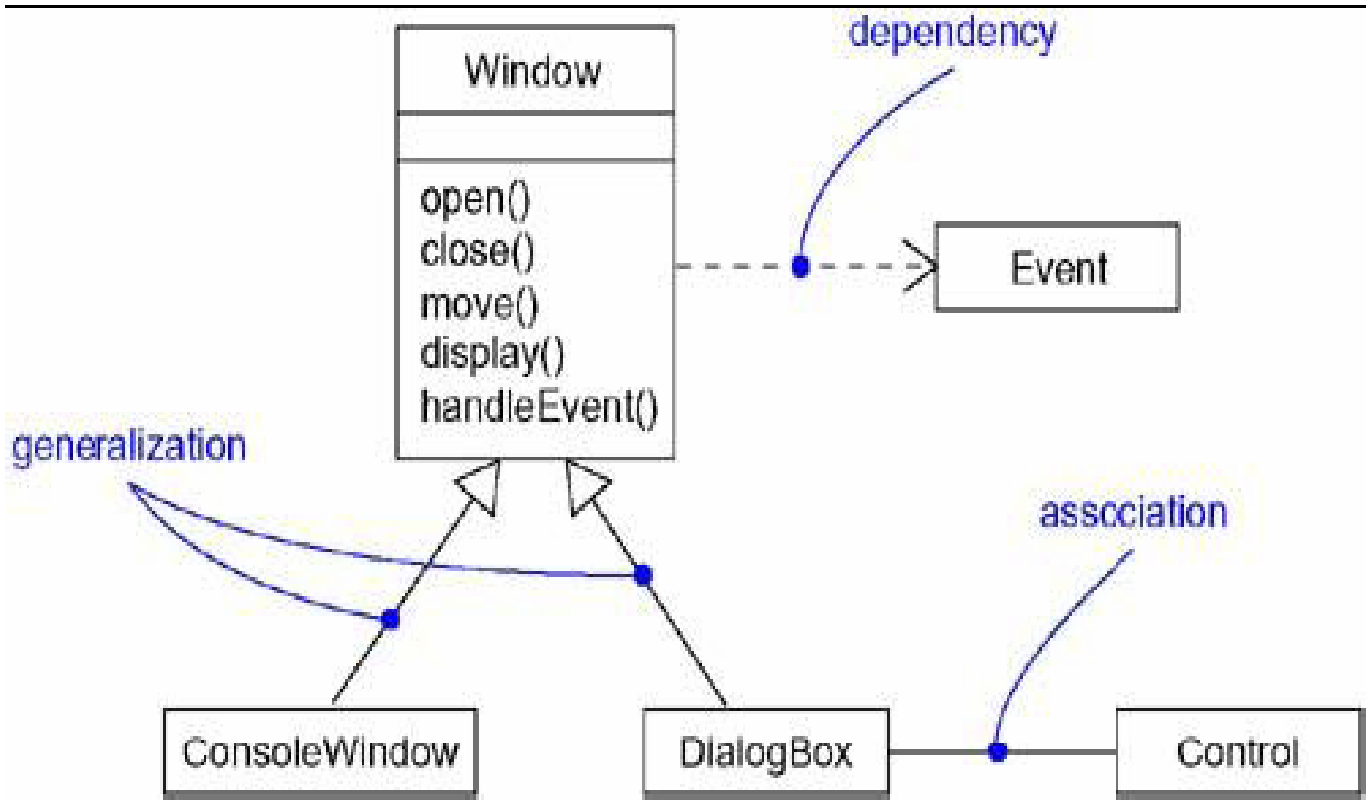
۳- كلاس هايي كه به هم مربوط هستند را دريك Diagram نمايش دهيم.

فصل پنجم

ارتباطات

Relationship

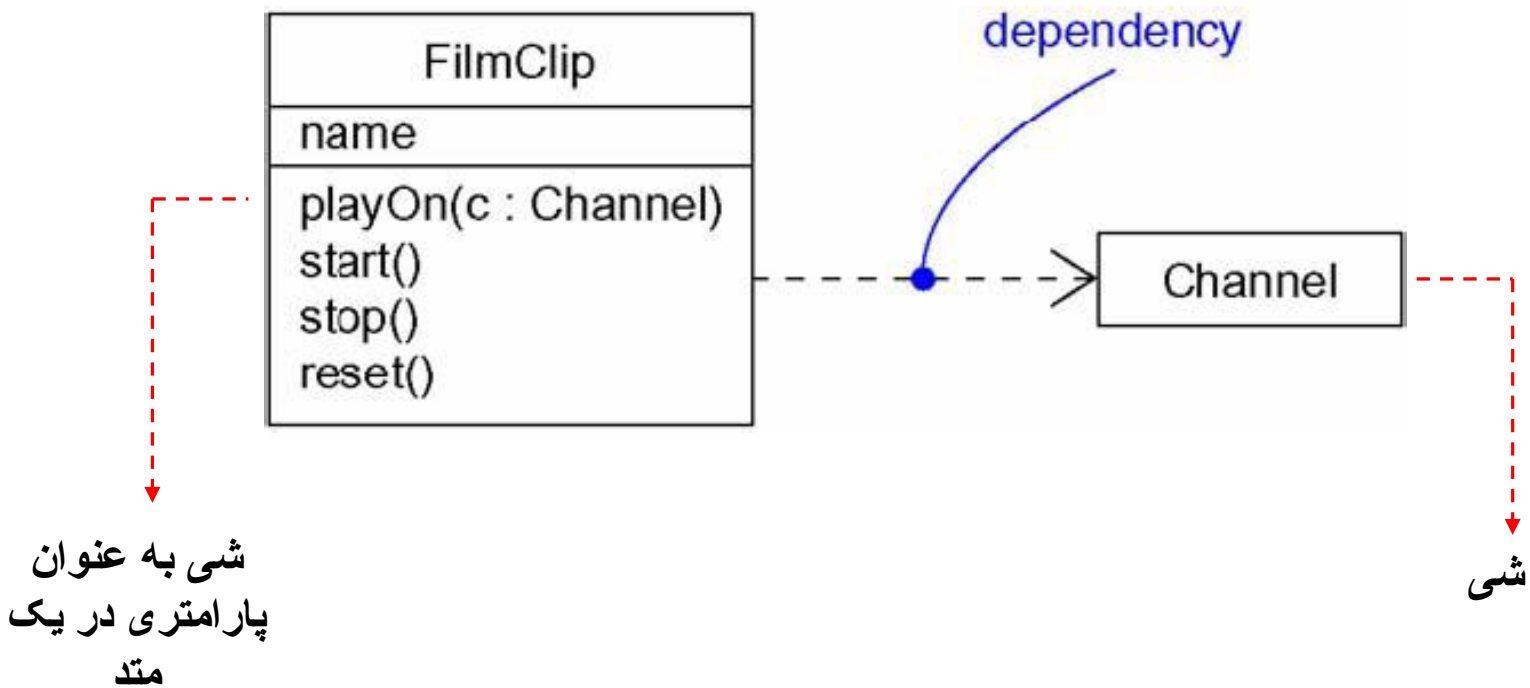
Relationship ارتباط بین اشیا است . در مدلسازی شی گرا سه ارتباط بسیار مهم داریم که شامل ارتباطهای وابستگی **dependency** ، همکاری **association** و تعمیم **generalization** می باشد . که بصورت گرافیکی یکسری خطوط ارتباط هستند که از مسیرهای مختلفی می گذرند و شکل های متفاوتی دارند .



وابستگي (Dependency)چيست؟

وقتي كه يك شي از لحاظ معنايي وابستگي به شي ديگري داشته باشد . يعني تغيير در يك شي مستقل باعث تغيير در شي مرتبط به همان شي در شي وابسته ديگر گردد.

اسم شی (وابسته) در پارامتری از متد شی (مستقل) آمده که این وابستگی شی را بیان می کند .

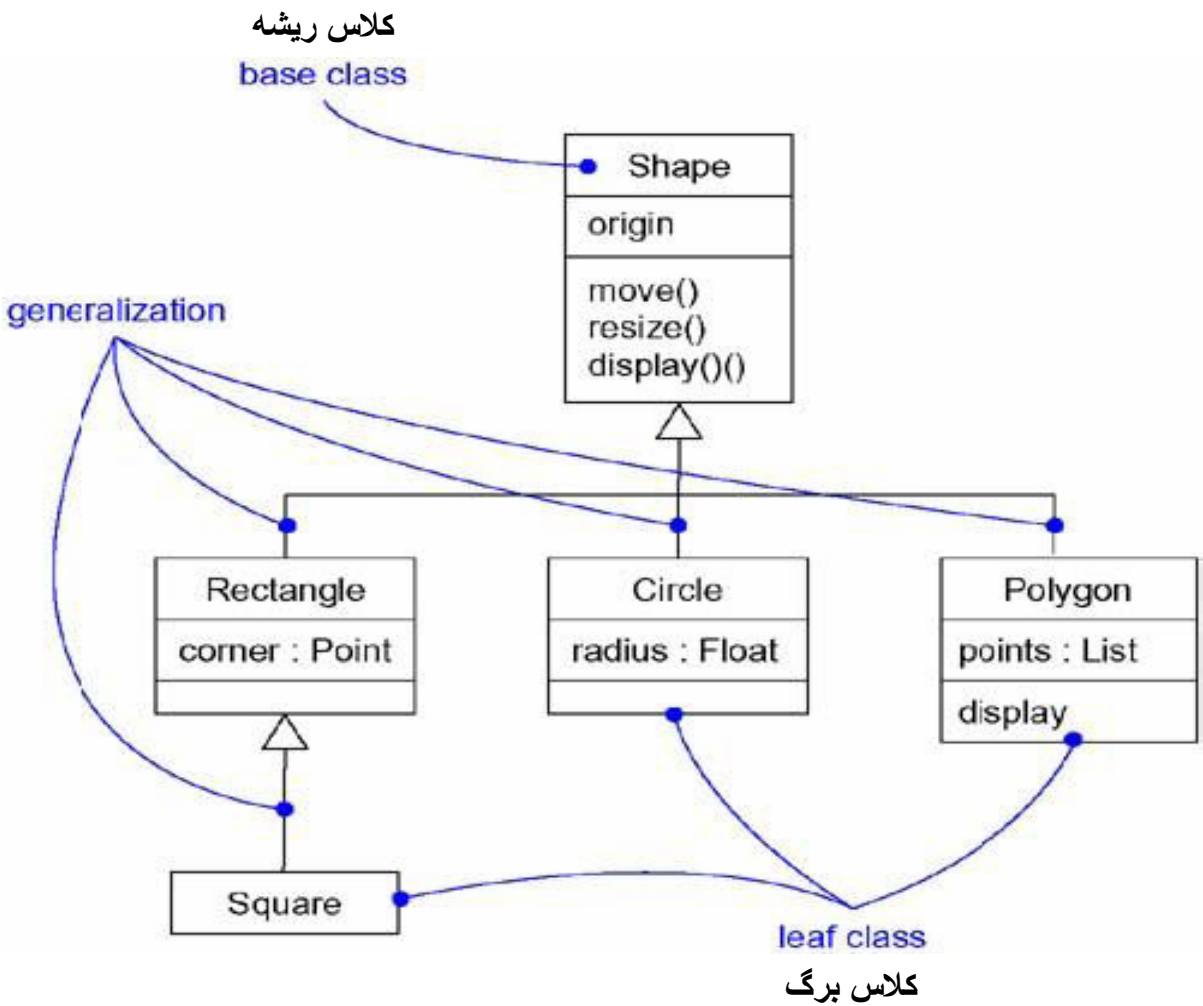


- بیشترین حالت وابستگي مربوط به حالتي است که شي اي از يك کلاس به صورت پارامتر در متد کلاس ديگر باشد.

- معمولا اسمي براي وابستگي ها ذکر نمي شود . تنها حالت اسم گذاري موقعي است که بين دو کلاس چندين وابستگي وجود داشته باشد.

تعميم(Generalization) چيست؟

رابطه اي كه بين يك Sub class و Super class مي باشد.
آن را اصطلاحا با Is-a مشخص مي كنند.



- این رابطه نیز می توان علاوه بر بین کلاس ها بین Packageها نیز اعمال کرد .

نکته : Sub class از جنس Super class است ولي Super class از جنس Sub classنمي باشد.

يك كلاس مي تواند صفر يا يك حالت يا والد داشته باشد كلاس Object فقط صفر حالت يا والد دارد.

كلاس پايه كلاسي است كه هيچ والدي ندارد. در جاوا كلاس برگ را با كلمه Finall نشان مي دهند.

Final classكلاس هايي هستند كه در جاوا نشان دهنده والدهايي است كه هيچ فرزندي ندارند.

همكاري(Association) چيست؟

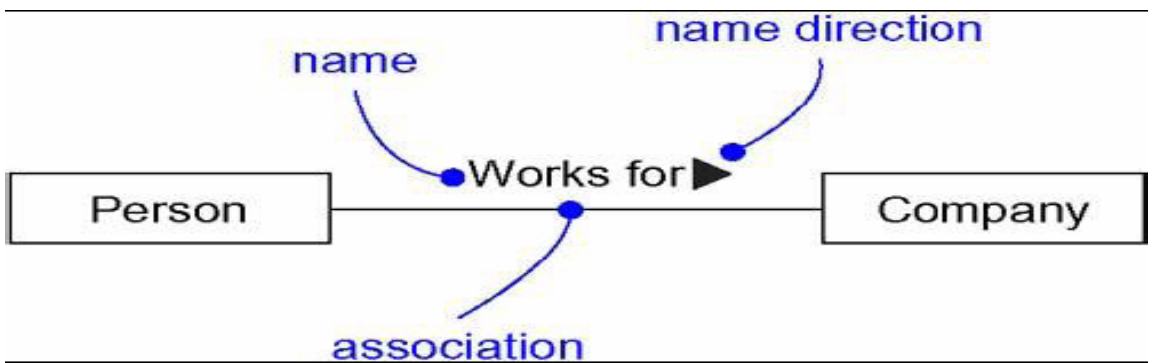
همكاري و وابستگي مي توانند انعكاسي باشند.
يك شي مي تواند همكار ي و وابستگي به خودش داشته باشد.
رابطه همكاري و وابستگي Reflective نيست يعني نمي تواند از خودش به ارث ببرد.

تعريف همكاري:

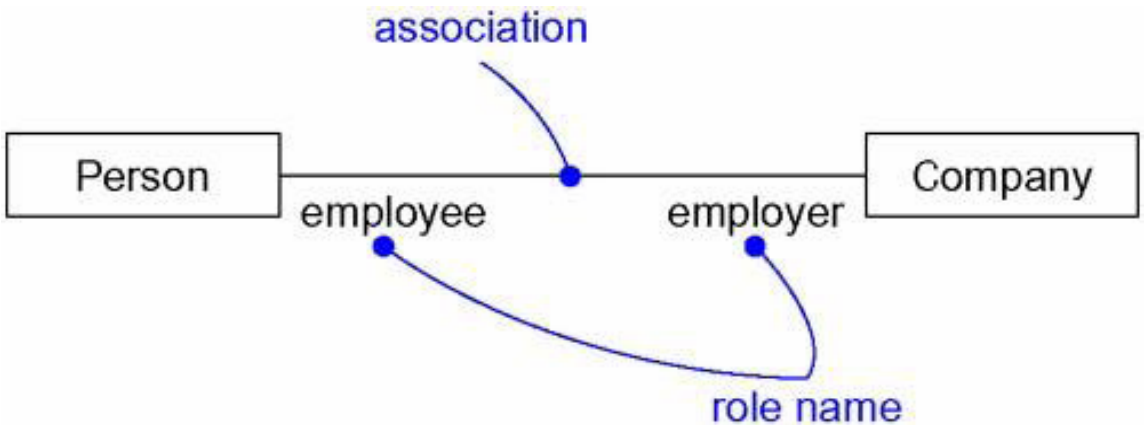
يك رابطه ساختاري است كه مشخص مي كند با داشتن اشيا يك كلاس ما مي توانيم به شي يك كلاس ديگر برسيم (كه رابطه اشيا موجود كه نشان دهنده تعداد اشيا درگير در دو طرف با كليه مشخصات عمومی) .

رابطه همکاری به چهار شکل زیر نشان داده می شود :

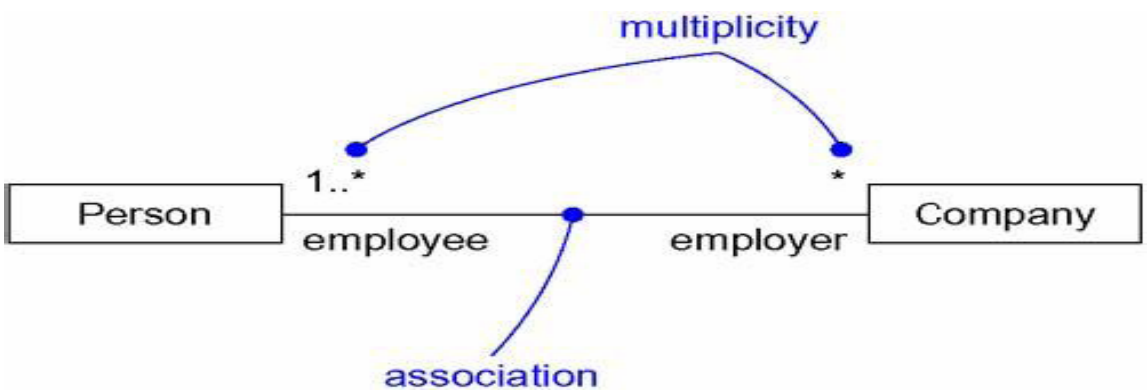
نمونه ۱: رابطه همکاری با نام و جهت ارتباط



نمونه ۲ : رابطه همکاری با نقش دو طرف



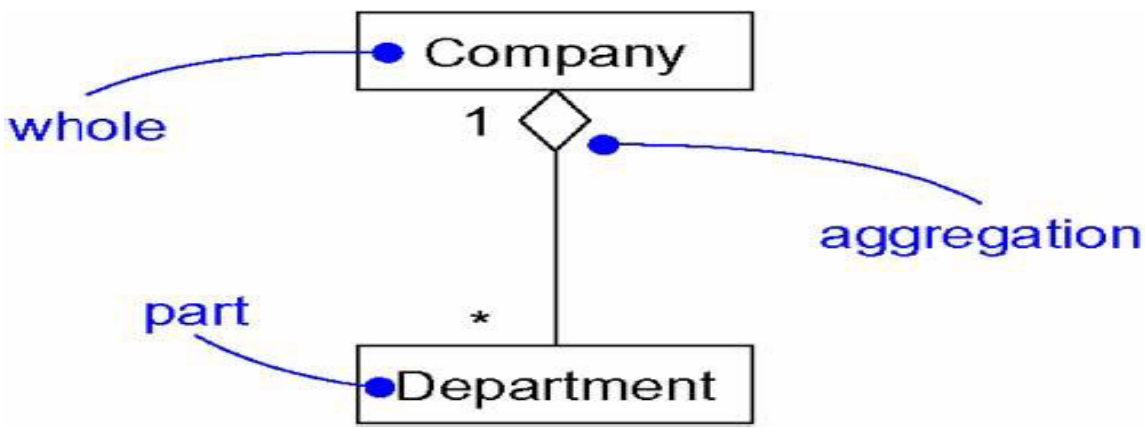
نمونه ۲ : رابطه همکاری با نقش دو طرف و تعدد در همکاری



- Super class از Sub class خبري ندارد ولي Sub class از Super class خبر دارد.
- حركت كردن از يك شي شروع مي شود و به يك شي ديگر ختم مي شود كه اين گفته را Navigation مي نامند . پس بايد تفاوتي بين navigation و association وجود داشته باشد.

تجمع (Aggregation) چيست؟

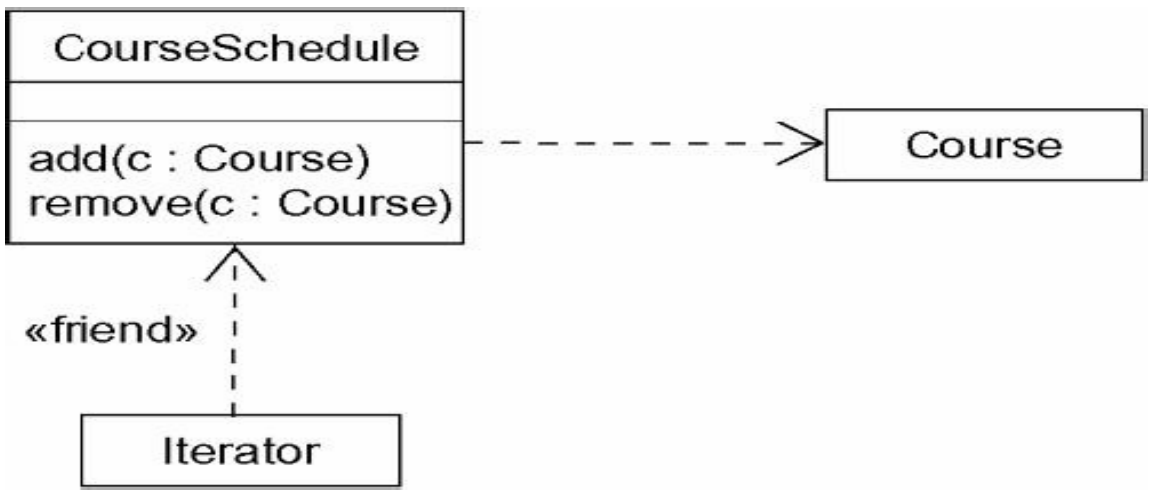
نوع خاصي از همكاري است.
Associationمي تواند حالت ساده باشد يعني در يك سطح باشد.
اما حالت ديگر مي تواند در سطح بالاتري باشد يعني: حالت جز به كل.



- براي نمايش ان از يك لوزي در طرف كل استفاده شود.
  اگر اين لوزي تو پر باشد Composition aggregation مي باشد . به اين معني كه اين رابطه جز در رابطه كل ما ضروري مي باشد.

نکات و تکنیک هایی که در روابط باید رعایت شوند

مدل کردن وابستگی Dependency:



در اين قسمت موضوع وراثت يگانه نيز مطرح مي باشد.

براي مدل كردن وراثت يگانه بايد سه قدم را انجام دهيم:

❶ بايد به كلاس ها نگاه كنيم . به اشتراك در Attribute و Operation و Responsibility نگاه مي كنيم . آيا اشتراكي بين آن ها مي باشد؟
اين كلاس را مشخص مي كنيم . اگر كلاسي وجود دارد مطالب مهم را از دل ان ها بيرون مي كشيم و ان را به عنوان پايه و مركز مي گيريم.

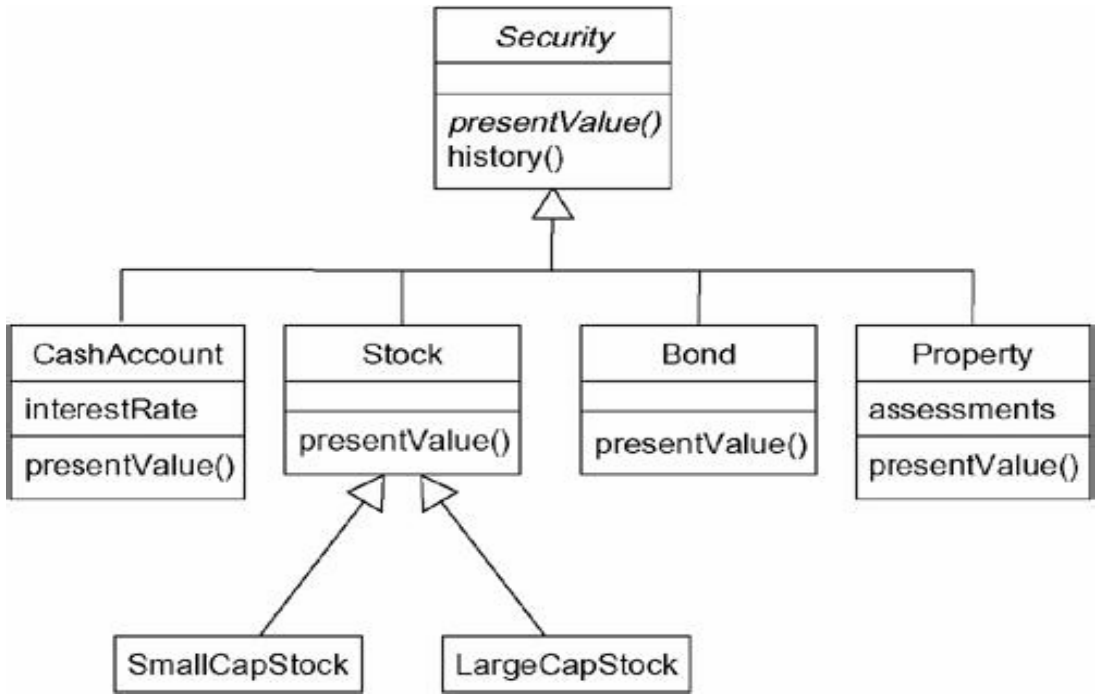
❷يك كلاس را بيرون مي كشيم و ان را به عنوان Parent انتخاب مي كنيم.آن گاه ديگر كلاس هاي موجود را به عنوان فرزند هاي ان والد انتخاب مي كنيم.

- ✓ نبايد خيلی عميق وگسترده باشد (هر كلاس نبايد مسئوليت های زيادی انجام دهد)
- ✓ درخت وراثت نبايد ارتفاع آن بيشتر از ۵ Level باشد .

❸ارتباط Sub class و Super class را مشخص مي كنيم و آنها را به كلاس جديدي كه ما ساخته ايم مدل مي كنيم .

مدل كردن ارتباطات ساختاري:

Generealization و Relationship يك ارتباط يك طرفه است اما Dependencyيك رابطه دو طرفه است.



- تفاوت عمده ی بین وابستگی و عمومیت دادن با همکاری یکطرفه بودن این ارتباط است که در همکاری این ارتباط دوطرفه ولی در وابستگی و عمومیت دادن یکطرفه می باشد .

براي مدل كردن ارتباطات ساختاري مراحل چهاگانه ريز را بايد انجام داد :

1. دو كلاس را در ابتدا انتخاب مي كنيم.بررسي مي كنيم كه ايا لازم است از يك شي از يك كلاس به يك شي از كلاس ديگرحركت كنيم يا نه؟
   ☑اگر نياز وجود داشت يك Association بين ان ها رسم ميكنيم.يك ديد **داده گرا** بين ان ها توليد مي كنيم.

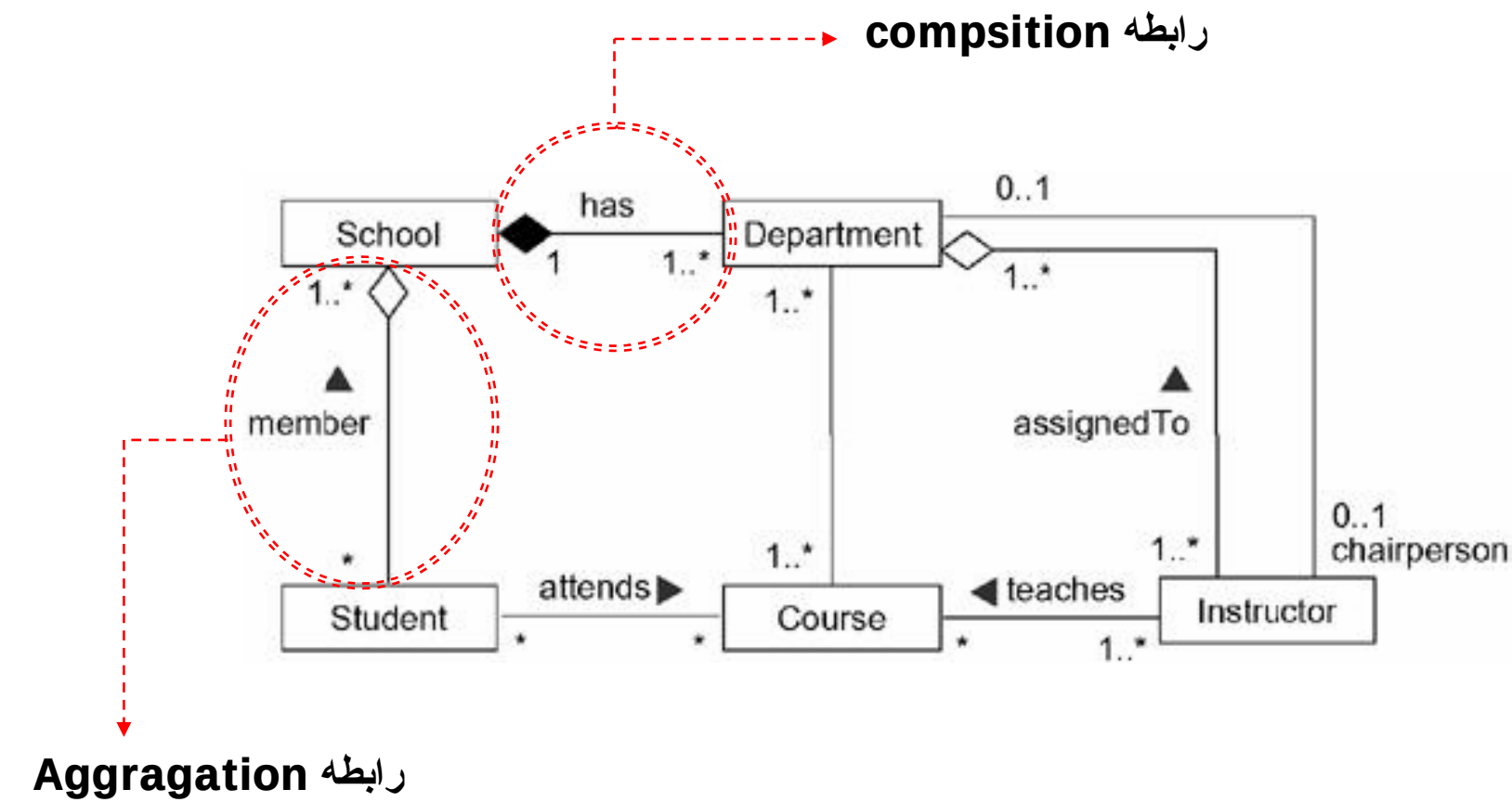
2. براي هر جفت از اشيا ي سيستم بررسي مي كنيم كه ايا بين اشيا دو كلاس ارتباط يا تعاملي وجود دارد غير از ان تعاملي كه در Dependency است يا نه؟
   ☑اگر غير از اين تعامل ارتباط ديگري باشد ان مي شود Association و كلا ديد **رفتار گرا** را در ان تعريف مي شود.

3. بايد مشخصه هر Association را بيان كنيم .

4. در قدم اخر بايد بياييم نگاه كنيم ايا اينAssociation نشان دهنده Aggregation است يا نه؟
   ☑ اگر بود Association را با دادن يك لوزي شكل و ساختارAggregation در مي اوريم.

و همچنين در گام چهاربايد تعدد را مشخص كرد كه در ارتباط همكاری اين رابطه ی Aggragation كه با لوزی تو خالی ◇ نمايش داده و در كنار كل قرار می دهيم نوع ديگر ارتباط همكاری كه شديدتر می باشد رابطه ی تركيب Compsition كه با لوزی تو پر ◆ نمايش داده می شود .

مثال : یک مدرسه حتما باید کلاس ، دفتر داشته باشد به این رابطه compsition گفته می شود ولی هر مدرسه میتواند مدرسه دانش آموز نداشته باشد که به این رابطه Aggragation گفته می شود .



☑ نكات :

- از وابستگي زماني استفاده مي كنيم كه ارتباطي كه ما داريم مدل مي كنيم ساختاري باشد.
- زماني از وراثت استفاده مي كنيم كه يك رابطه is-a درسيستممان باشد.
- وراثت چند گانه را مي توان از رابطه Aggregation استفاده كرد.
- از وراثت چرخشي بپرهيزيد.
- متعادل نگه داشتن درخت وراثت مي با شد.

☑درخت وراثت نبايد خيلي عميق و نيز خيلي مسطح باشد.مسطح بودن درخت يعني اينكه همه فرزند ها فرزند object باشند .

☑ درخت پر عمق درختي است كه اختلاف جزيي بين پدر و فرزند وجود دارد . اگر اختلاف بين ان ها بيشتر شود عمق درخت كم مي شود . مقدار سطح در اين درخت ۵ مي باشد .

☑ از همكاري براي نشان دادن ارتباطات ساختاري استفاده كنيم .

براي رسم اين كلاس در UML :

Oblique lineدر اينجا داريم كه:

✓مي توان اين خطوط را مايل وافقي و يا عمودي رسم كرد.

✓منسجم و به صورت منظم از اين دو ساختار استفاده كرد.

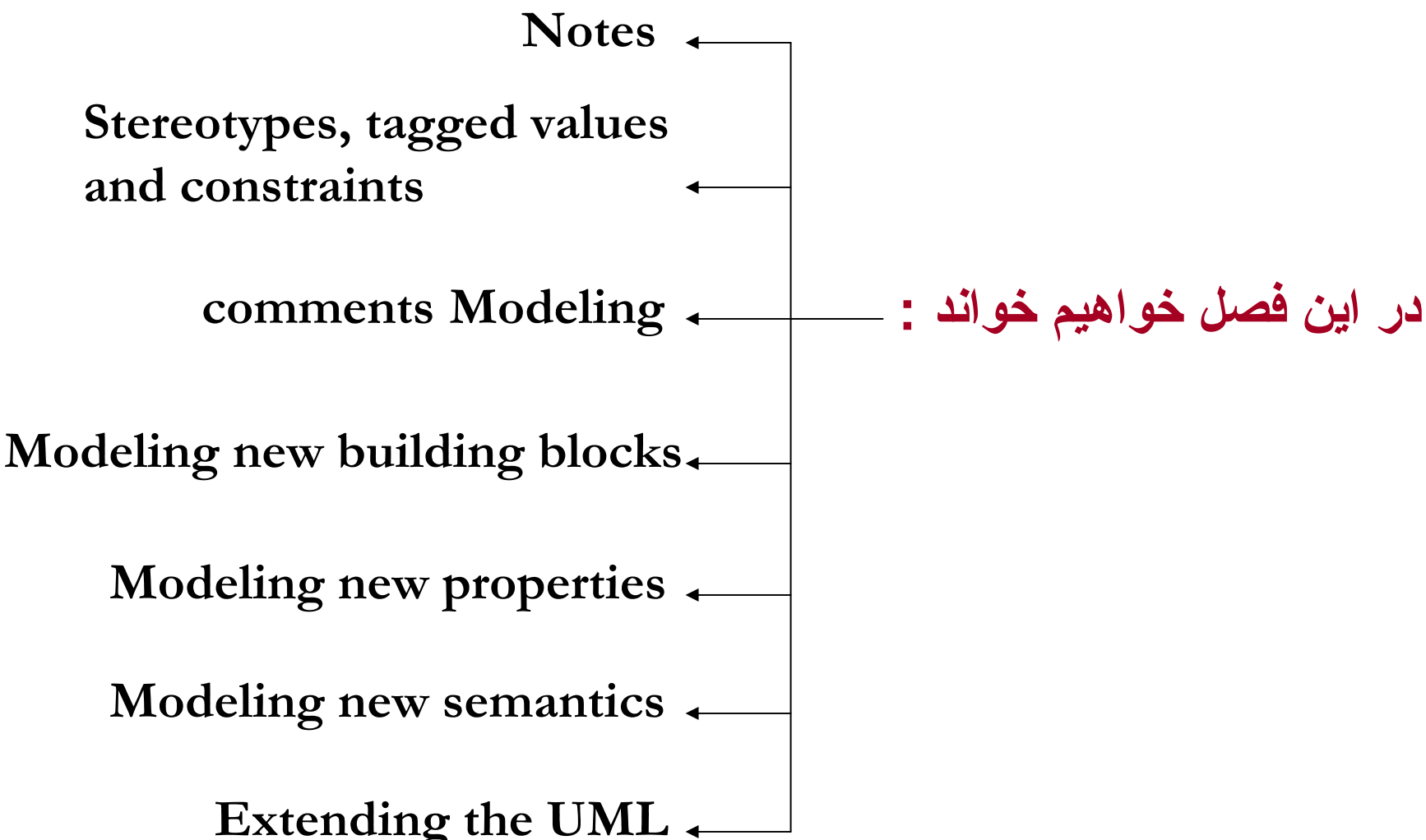
✓مزيت خطوط افقي و عمودي در زيبايي ان است.به وسيله خطوط عمودي و افقي مي توان ساختار را به صورت زيبا و خوبي نمايش داد.

- مزيت مايل در اين است كه جاي كمتري مي گيرد و در فضا صرفه جويي مي شود .
- معمولا از هر دو خط مي توان استفاده كرد به شرط اين كه به صورت منسجم و منظم باشد.
- بايد كاري كرد كه در ان خطوط همديگر را قطع نكند.
- در رسم خطوط باز Abstraction را در نظر بگيريم.در ان خطوطي را رسم كنيم كه از نظر Abstraction ضروري است و از رسم خطوطي كه Abstraction نيازي به ان ندارد بپرهيزيم.

فصل ششم

مکانیزم های عمومی

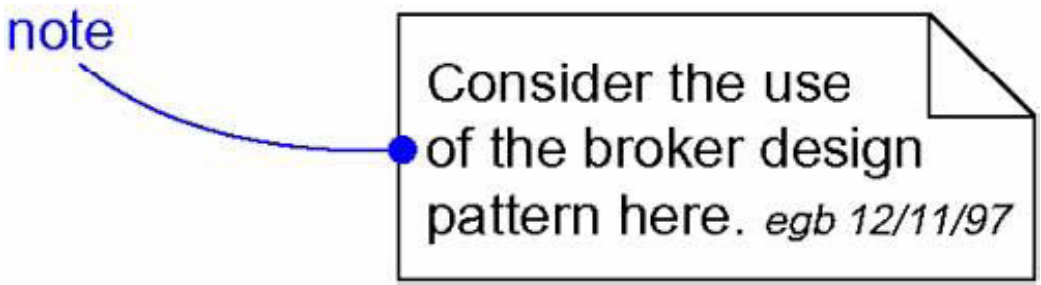
Common Mechanisms


Notes
Stereotypes, tagged values and constraints
comments Modeling
Modeling new building blocks
Modeling new properties
Modeling new semantics
Extending the UML
در این فصل خواهیم خواند :

☑ **Note نت (يادداشت ها) :**

يك نمايش گرافيكي براي نمايش محدوديت ها يا توضيحات منحصر به اجزاي عناصر ديگر است .

معمولا نت ها را با استفاده از يك مستطيل كه گوشه هايش خم شده است نشان مي دهند .



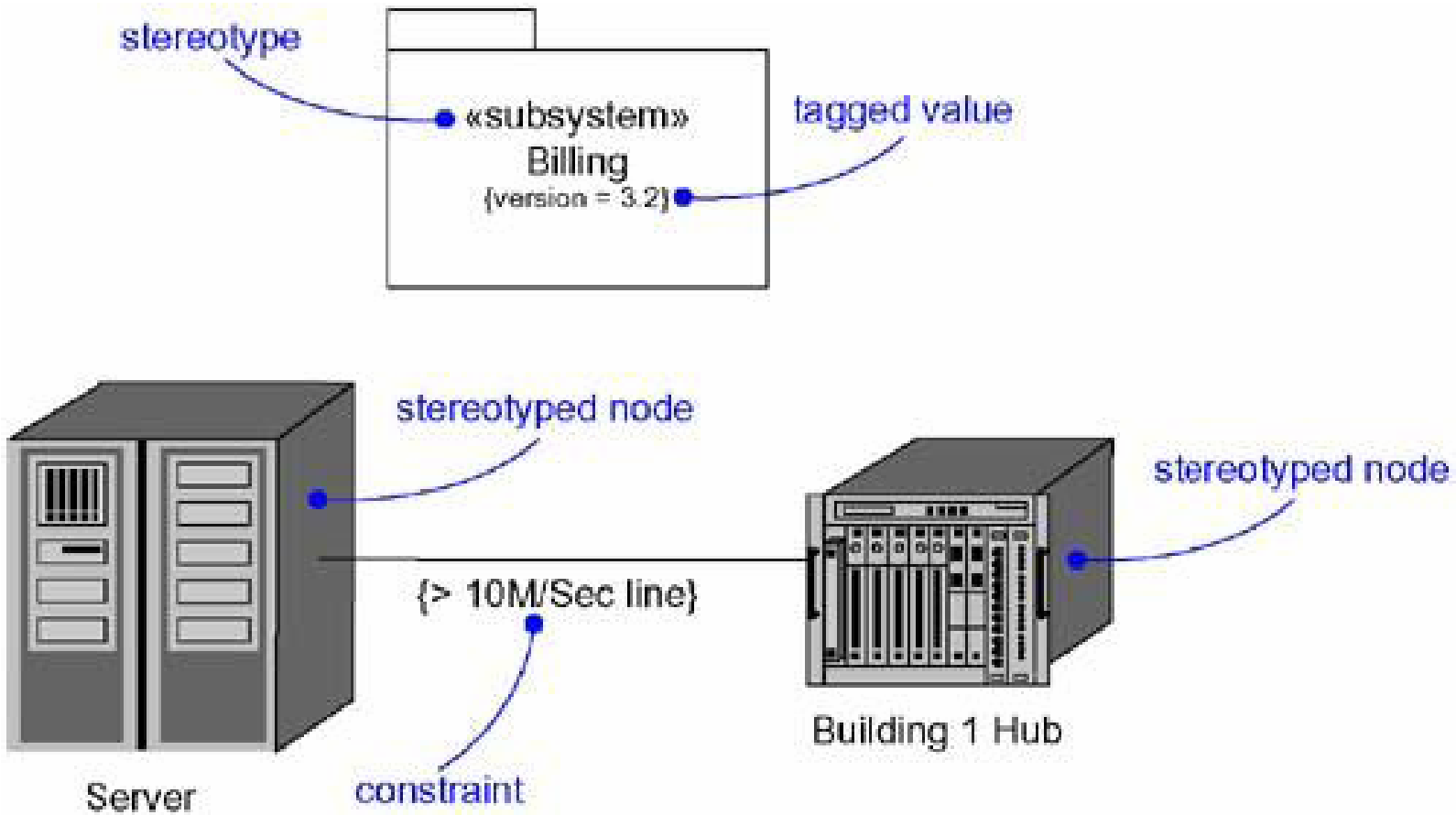
☑ كاربرد هاي Tagged value :

١ - افزودن يك سري مشخصات جهت توليد كد

٢ - مشخص كردن يك سري تنظيمات پيكر بندي

- ما با استفاده از Stereotypes لغت نامه UML توسعه دهیم می توانیم یک بلاک سازنده جدید به UML اضافه کنیم ساده تر ین حالت Stereotypes آن یک text که در گیومه قرار میگیرد .

در شکل زیر : **Stereotypes و Tagged Values و Constraints**

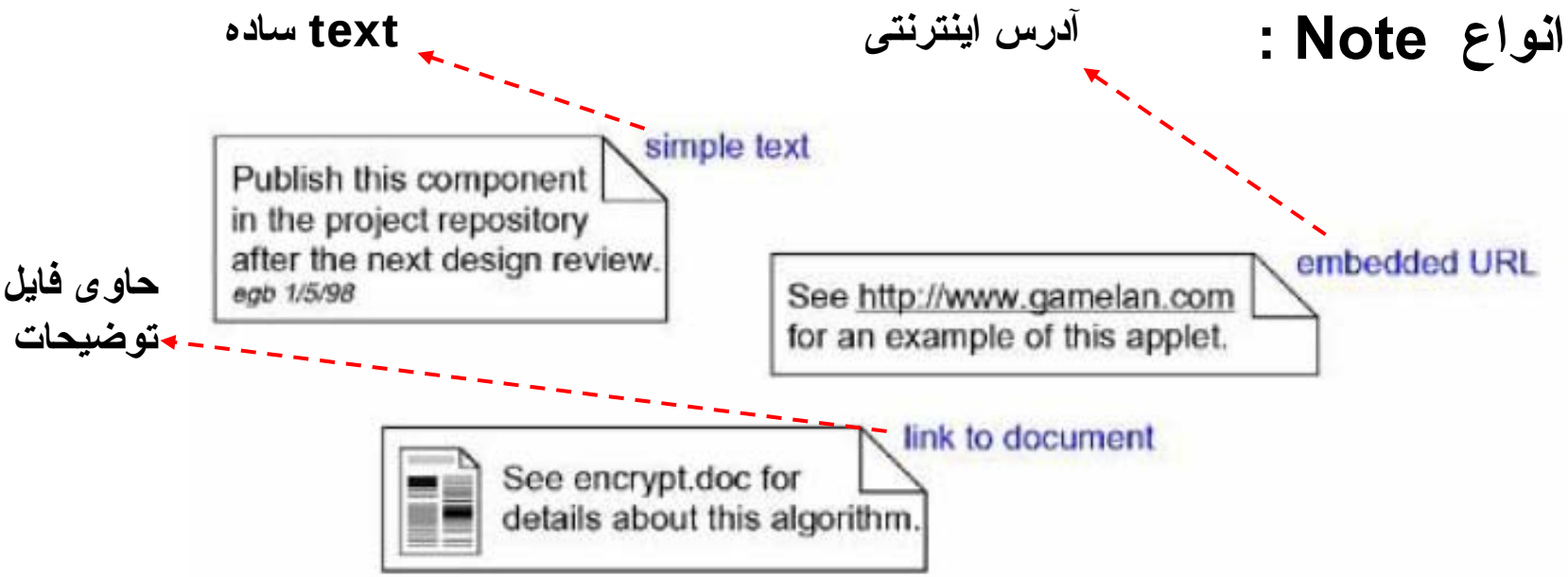


روش اول : اگر Stereotypes نمی آید یک شکل package در نظر گرفته می شود .

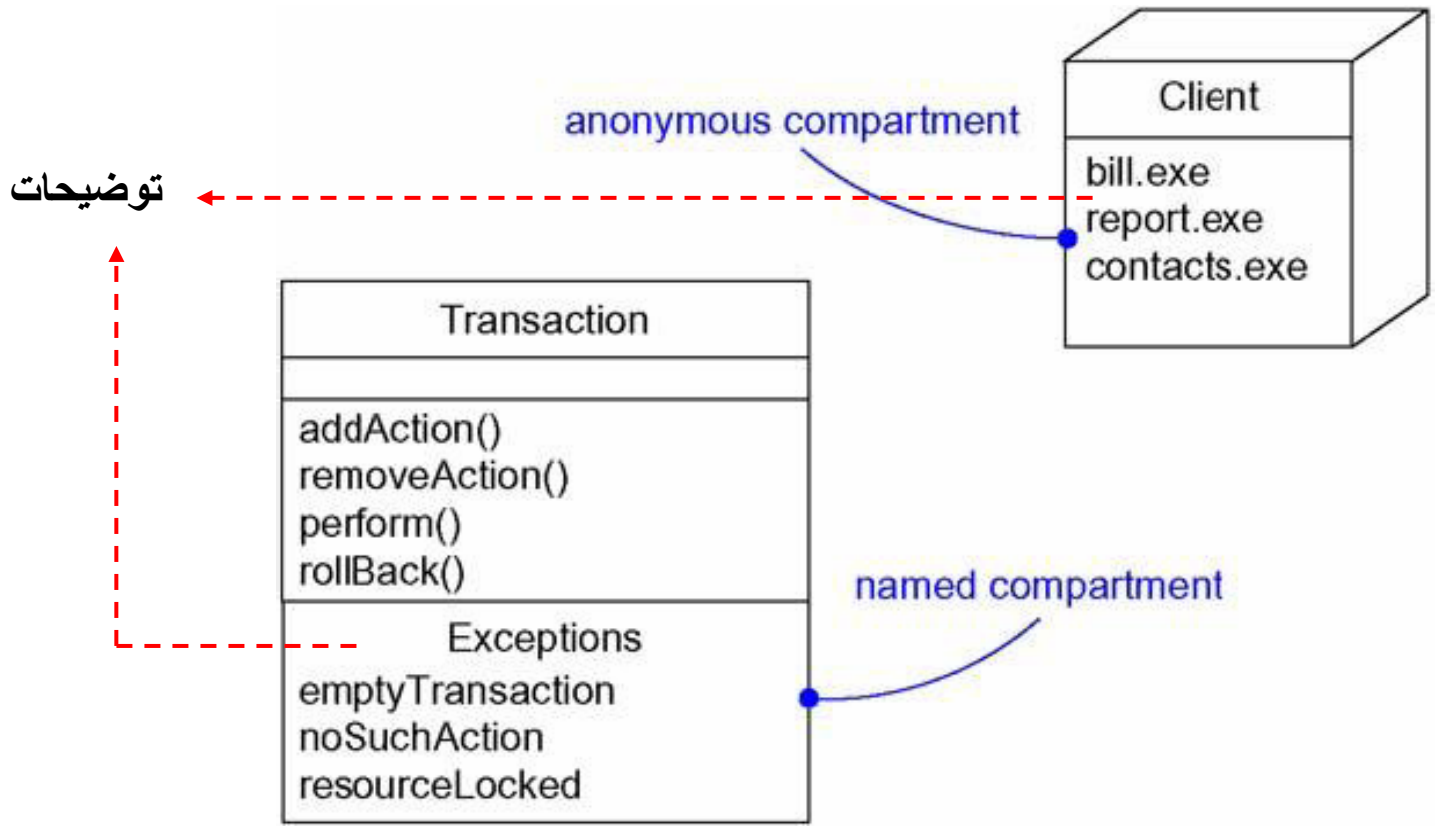
روش دوم : برای نمایش آن استفاده از آیکن توسعه مشخصه های داخل UML خواهیم داشت و از نظر گرافیکی به صورت یک رشته که داخل { } قرار می گیرد نوشته میشود زیر نام عنصر ما می توانیم constraint یک مجموعه قاعده جدید به UML اضافه کرد که به صورت { } که اگر روی کلاس اعمال شود در کنار آن وارد می کنیم مانند مثال قبل که روی یک رابطه همکاری اعمال شده است .

در مورد Note ها اگر بخواهیم از آن کد تولید شود هیچ تاثیری در source که ندارد .

انواع Note :



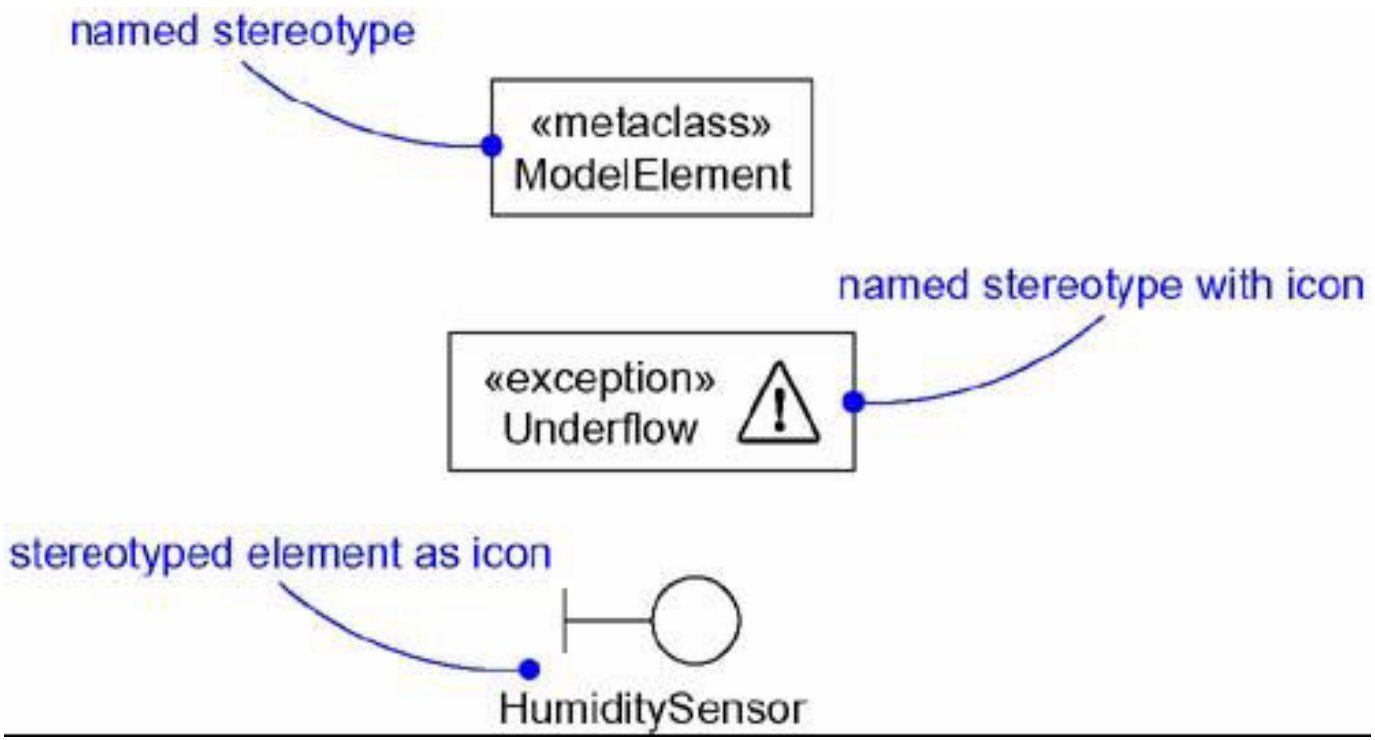
علاوه بر سه نوع قبلی یک نوع دیگر Node وجود دارد که به این صورت است که ما می توانیم توضیحات زیر خود عناصر اضافه کنیم :



Stereotypes ها :

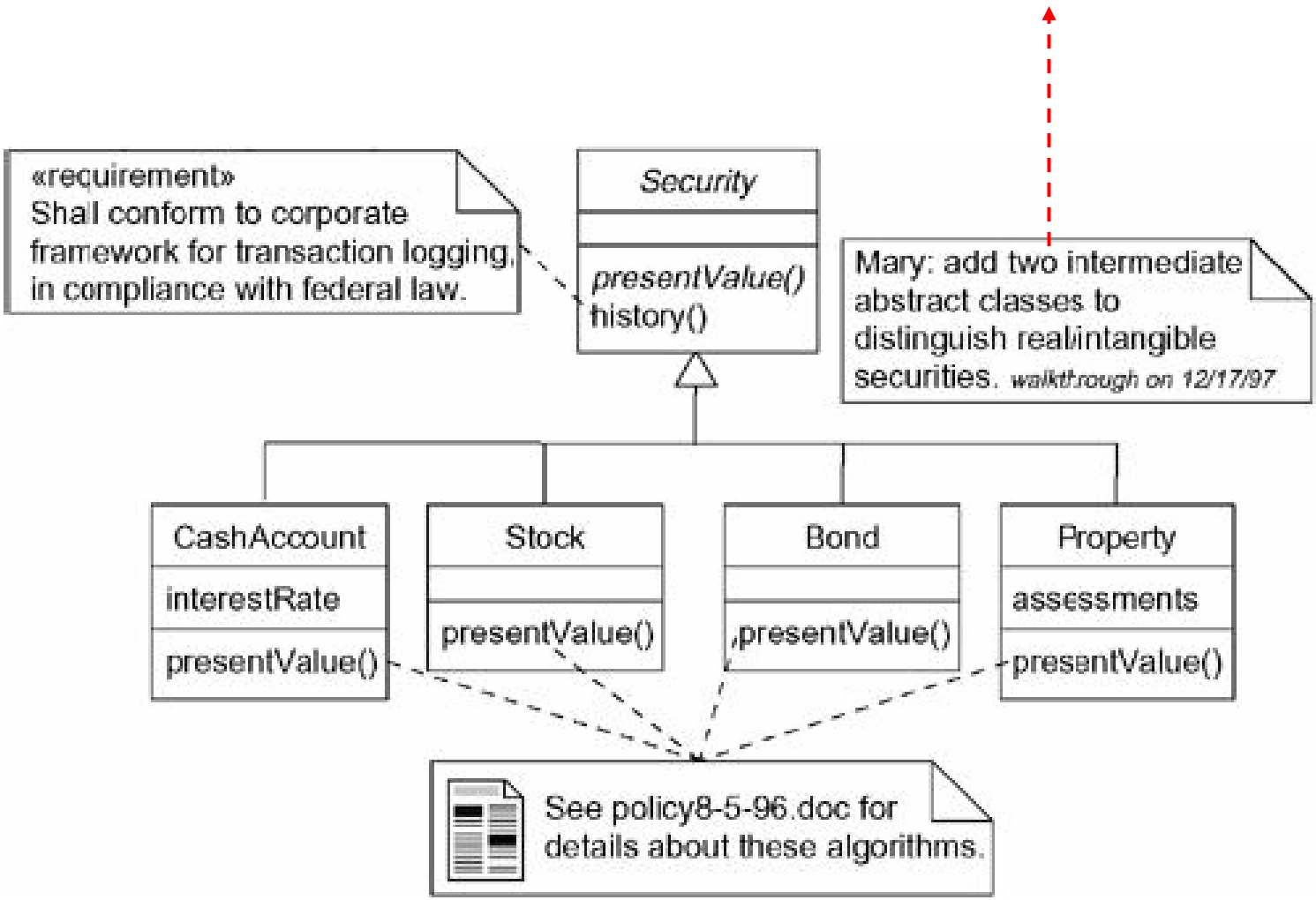
با آنها UML را گشترش میدهیم گرفتن یک بلاک از UML و به یک بلاک جدید اضافه کردن – خواص جدید – اضافه کردن محدودیت ها – تغییر شکل که با استفاده از یک آیکن صورت میگیرد .

انواع Stereotypes : (سه نوع)



مثال :

توضیح میانه طی مدل سازی بوجود آمده و ممکن است بعدا از بین برود .



براي مدل كردن مولفه ها و بلاك هاي جديد سه قدم را بايد انجام داد:

۱- بايد مطمين باشيم كه اين مولفه در مدل وجود ندارد و اگر در انجا نبود و در stereotypeهم نبود بايد يك stereotype جديد اضافه كنيم.براي اين كار يك مولفه يا يك stereotype كه به ان مولفه نزديك است را انتخاب كنيم و به ان چيزهايي كه لازم است را بدهيم.يعني از وراثت براي ايجاد مولفه stereotype جديد استفاده كنيم.

۲- tagged value كه بسيار در پروژه استفاده مي شود را به stereotypeها تبديل كنيم .

۳- براي يك stereotype از يك ايكن تصويري استفاده كنيم .

۴- اضافه كردن

✓ جهت استفاده از stereotype دقت نياز دارد .
✓ توجه شود زياد از stereotype استفاده نشود .
مثال : از stereotype از تيم و مربی ايجاد كرده .

☑ براي مدل كردن پروپرتي ها يا خواص بايد اقدام به اعمال زير كرد :

۱- براي پروپرتي هاي جديد اول دقت مي كنيم ان مطالبي كه مي خواهيم بيان كنيم جزاستاندارد هاي UML نباشد اگر نبود به Tagged value هاي خود UML رفته اگر ان جا هم نبود بايد ان Tagged value را ايجاد كنيم كه مي بايست مشخص كنيم كه ان را مستقيم به مولفه اضاف كنيم.

۲- يك Streotype جديد تعريف مي كنيم و ان Tagged value را به ان اضافه مي كنيم كه به صورت غير مستقيم است.

۳- اگر مدلي داريم كه يك سري Tagged value مربوط به خود ان مدل است نبايد در ان Tagged value دستكاري كنيم .

مثال :

يكی از موارد Tagged value قرار دادن version می باشد . حتی المكان از حالت پيش فرض استفاده كنيم .

☑ براي مدل كردن توضيحات چهارقدم را بايد انجام داد :

۱- توضيح مورد نظر را در محل نزديك به همان محل در مستطيل ذكر كنيم.اگر هم بخواهيم با دقت بيشتري ان را معلوم كنيم با خطي ان نقطه را به توضيح مورد نظر وصل مي كنيم .

۲- سعي كنيم كه توضيحات اصلي كه براي برنامه نويس جالب است را قرار دهيم و بقيه را مخفي نگه داريم .

۳- مي توانيم براي مشخص كردن روند اجراي پروژه از توضيحات استفاده كنيم .

۴- از توضيحات براي مشخص كردن تصميماتي كه مستقيما از خود مدل قابل فهم نيست استفاده كرد .

مدل کردن مفهوم و قاعده جدید :

۱- کنترل کنیم که آیا این محدودیت جز محدودیت های عمومی UMLهستند یا نه ؟

۲- اگر این محدودیت نبود در اینجا باید خودمان قاعده را تعریف کنیم .
آن محدودیت کنار جایی که می خواهیم محدودیت ایجاد کنیم در یک { }
قرار می دهیم .

مثال : اگر بخواهیم دقیق تر بنویسیم این قاعده و محدودیت از OCL استفاده می کنیم .

☑ ما سه نوع محدوديت داريم كه به ترتيب شامل:

۱- ساده: روي يك مولفه كار مي كند.
۲- تركيبي: روي دو مولفه و تركيبي از چند عنصر كار مي كند .
۳- دقيق: كه با OCL نوشته مي شود .

☑ در رابطه با مدل کردن محدوديت ها:

۱- در مرحله اول: ببينيم كه ايا ان قواعد مورد نظر در خود قواعد UML وجود دارد يا نه؟
اگر نبود به محدوديت هاي استاندارد UML برويم.
اگر ان جا هم نبود بايد يك محدوديت جديد اضاف كنيم.

۲- در مرحله دوم: اگر بخواهيم محل ان محدوديت را دقيق ترمشخص كنيم مي توانيم از نقطه چين ..... استفاده كنيم .

۳- در مرحله سوم : در صورتي كه بخواهيم به صورت دقيق ترمفهوم محدوديت را بيان كنيم از OCL استفاده مي كنيم.

✓نکته : چهار مدل یادداشت داریم :

۱- یادداشت ساده
۲- زماني كه مي خواهيم توضيحات را link دهيم
۳- زماني كه حجم زيادي را از توضيحات نياز داريم.
۴- نت هايي كه در پايين يك بلاك اضافه مي شوند كه هم ميتوانند اسم باشند و هم بدون اسم.

✓نکته : **زماني از توضيحات استفاده مي كنيم كه مطلب را نتوانيم با خود UML بيان كنيم . يا توضيحاتي را بنويسيم كه با خود UML نتوان نوشت.**

✓نکته : **مي توانيم توضيحات را در يك محل بياوريم و بعد قسمت هاي مختلف را به وسيله link به يادداست ها وصل كنيم.**

☑ وقتي مي خواهيم يك مدل را با tagged value و stereotype و constraint گسترش داد بايد شش نكته را انجام داد :

1. توافق براي استفاده از stereotype هاي مشخص براي يك گروه كه مي خواهيم با هم يك مدل ايجاد كنيم .

2. از اسم هاي كوتاه براي نامگذاري tagged value و stereotype و constraint استفاده شود.

3. براي رسم سه مولفه از ايكن هاي گرافيكي كمتر استفاده شود.ضعف استفاده از انها در قابل انتقال نبودن ان هاست.

4. براي بيان دقيق محدوديت ها از OCL استفاده كنيم .

5. براي استفاده از رنگ ها از رنگ هاي منسجم و معنا داراستفاده شود.رنگ ها باشد سازگاري داشته باشند.

6. از ايكن هاي گرافيكي ساده استفاده شود.

فصل هفتم

دیاگرام ها

Diagrams

دیاگرام چیست؟

وسیله اي است که از طریق آن مي توان این مولفه هاي سازنده سیستم را نمایش داد.

مولفه هاي عمده در uml دو گونه هستند:

* اشیا
*ارتباطات

دیاگرام یك نمایش گرافیکي از مجموعه اي از عناصر است که به صورت گراف است و اشیا نود ها و ارتباطات لبه هاي ان مي باشند.
دیاگرام خوب دیاگرامی است که برای توسعه سیستم قابل فهم و نزدیک شدنی باشد . انتخاب مجموعه ای درست از دیاگرام ها برای مدل کردن سیستم و برای پرسیدن سوالات صحیح در رابطه با سیستم و راهنمایی برای روشن کردن یک دلیل تضمینی و محکم باشد .

سیستم چیست؟
یك سیستم مجموعه اي است از سیستم هاست که براي پاسخ دادن به یك نیاز سازماندهي شده و با مجموعه اي از مدل ها نمایش داده مي شود.

زير سيستم چيست؟

يك زيرسيستم يك دسته بندي از اشيا است كه تعيين كننده مشخصه اي از رفتارهاست كه توسط عناصر دروني ارايه مي شود.

مدل چيست؟

يك مدل يك تجريد (Abstraction) است كه از لحاظ مفهوم بسته (جدا كننده با دنیای پیرامون) از لحاظ سيستم است . و ساده سازی از واقعيت است .
به لحاظ معنايي تجريدي مرز سيستم را مشخص مي كند . و حذف موارد غير ضروری و در يک ديد خاص نمايش مستقلی از سيستم به ما می دهد و هدف آن كمک كردن به فهم سيستم می باشد .

دید چیست؟

يك view نمايشي است به سازمان و ساختار مدل يك سيستم كه روي يك نقطه خاص ازسيستم تمركز دارد.

پنج دید اصلي در سيستم وجود دارند كه عبارتند از:

1. use case view
2. Design view
3. Process view
4. Implemention view
5. Deployment view

در هر سيستمي چهار مدل ايستا(دياگرام های ساختاری) وجود دارد:

1. Class diagram
2. Object diagram
3. Component diagram
4. Deployment diagram

حال مي خواهيم هر كدام از مدل ها را توضيح دهيم:(دياگرام های ساختاری)

Class diagram :

محتوي كلاس ها رابط ها و همكاري و رابطشان است . نموداري است كه بيشترين استفاده را در مدل هاي uml دارد . معمولا استفاده اصلي ان در ديد طراحي و بعضا در process viewهاست .

Object diagram :

ابجکت ها و ارتباط بين ان ها را داريم.براي مشخص کردن که لحظات مختلف به اشيا يي براي سيستم ايجاد شده و هر لحظه عکس از سيستم مي باشد انجام مي گيرد.استفاده عمده ان در design view و البته بعضا در process view استفاده مي شود.

Component diagram :

شامل مولفه ها و ارتباط بين ان هاست.و بيشتردر ديد پياده سازي استفاده مي شود.

Deployment diagram:

نرم افزار اماده را در محيط عملياتي نصب مي کنيم.در نمودار deployment يك سري مولفه ها و نودهايي وجود دارد که براي نمايش ديد سيستم استفاده مي شود.

پنج دید دیاگرام رفتاري(پویا) در سیستم وجود دارد که عبارتند از:

1. Use case diagram : سازماندهی رفتارهای سیستم
2. Sequence diagram: قرار گرفتن فوکوس یا مرکزتوجه بر ترتیب زمانی پیام ها
3. Collaberation diagram: قرار دادن فوکوس بر ساندهی ساختاری اشیا که در ارسال و دریافت پیام ها
4. Statechart diagram: قرار دادن فوکوس بر تغییر وضعیت گردنده سیستم توسط رو یداد ها
5. Activity diagram: قرار دادن فوکوس بر جریانی از کنترل فعالیت برای فعالیت

حال مي خواهيم هر كدام از مدل ها را توضيح دهيم:

Use case diagram

شامل مجموعه اي از Actor ها و use caseها و ارتباط بين آن ها مي باشد و براي ديد use case سيستم استفاده مي شود و سيستم را در بالا ترين سطح توصيف مي كند.

2&3.equence diagram& Collaberation diagram

كه با هم Interaction diagram را تشكيل مي دهند.
تعاملات بين اشيا و پيغام هاي بين اشيا مي باشد.
در Sequence diagram بر روي زمان بندي در پيغام ها دلالت مي كند.
در Collaberaton diagram بر روي ساختار انتقال پيغام ها كار مي كند و نه بر روي زمان بندي.

Statechart diagram

حاوي يك state machine مي باشد.
ماشين وضعيت براي يك شي وضعيت هاي مختلفي دارد كه شي در ان قرار دارد.با طريقه انتقال و پاسخي كه نسبت به ان از ديد طراحي مي دهد به ان شي نگاه مي كند.

Activity diagram

جريان انتقال سيستم از يك فعاليت به فعاليت ديگر را نشان مي دهد.خيلي شبيه به فلوچارت است ولي كاملا فلوچارت نيست.از اين خصيصه بيشتر در ديد طراحي و هم در ديد use case استفاده مي شود.

براي مدل كردن سيستم از ديد هاي مختلف چهار قدم را بايد انجام دهيم:

1. مشخص كنيم كدام ديد ضروري است.ان ديدي را انتخاب كنيم كه ريسك هاي سيستم را بهتر نشان دهد. ممكن است در يك مدل به هر پنج ديد نياز داشته باشيم.

2. مشخص مي كنيم براي يك ديد خاص چه چيز هايي بايد ايجاد شود تا بتوانيم جزييات ان ديد را بهتر نمايش دهيم.

3. زمانبندي كنيم تا بر چه اساسي مدل رسم و به چه شكل رسم شود.

4. دياگرام هايي كه موقت رسم مي شوند ان دياگرام ها را دور نياندازيم.ممكن است در اينده به ان ها نياز داشته باشيم.

مدلسازی سیستم در سطوح Abstraction

ما نياز داريم از Abstraction هاي مختلف به مدل نگاه كنيم .يعني بسته به نيازمندي هاي يك شخص مدل سازي را براي ان انجام مي دهيم . يعني اشيايي كه نياز ندارند بايد ان را مخفي كند . مثلا در مدل هاي زير اين اقدام براي هر كاربر خاص استفاده مي شود:

روش اول:

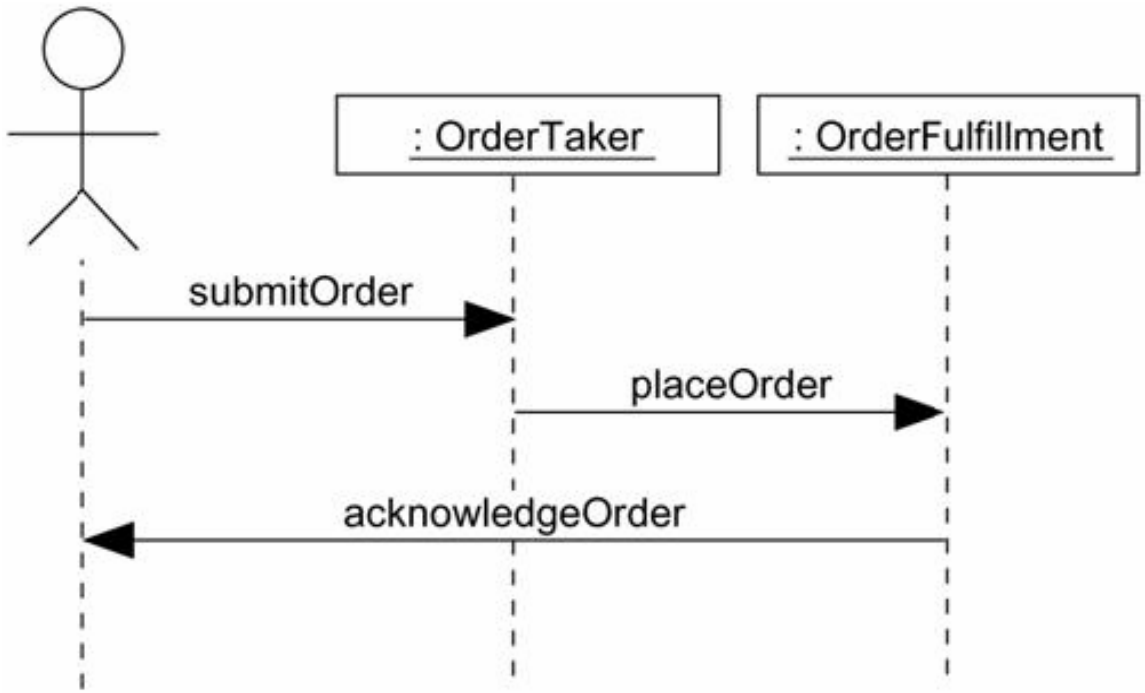
قدم اول: هم شخص و هم مدل را براي رسم دياگرام مشخص كنيم.
قدم دوم: مشخص شود كه كاربر پياده ساز سيستم است يا شخص است كه مي خواهد از نظر مفهومي اطلاعاتي از سيستم داشته باشد.
قدم سوم: شخص در چه محلي قرار دارد؟
قدم چهارم: در مورد نشان دادن و يا نشان ندادن ان مولفه خاص تصميم گيري كنيم.

روش دوم:

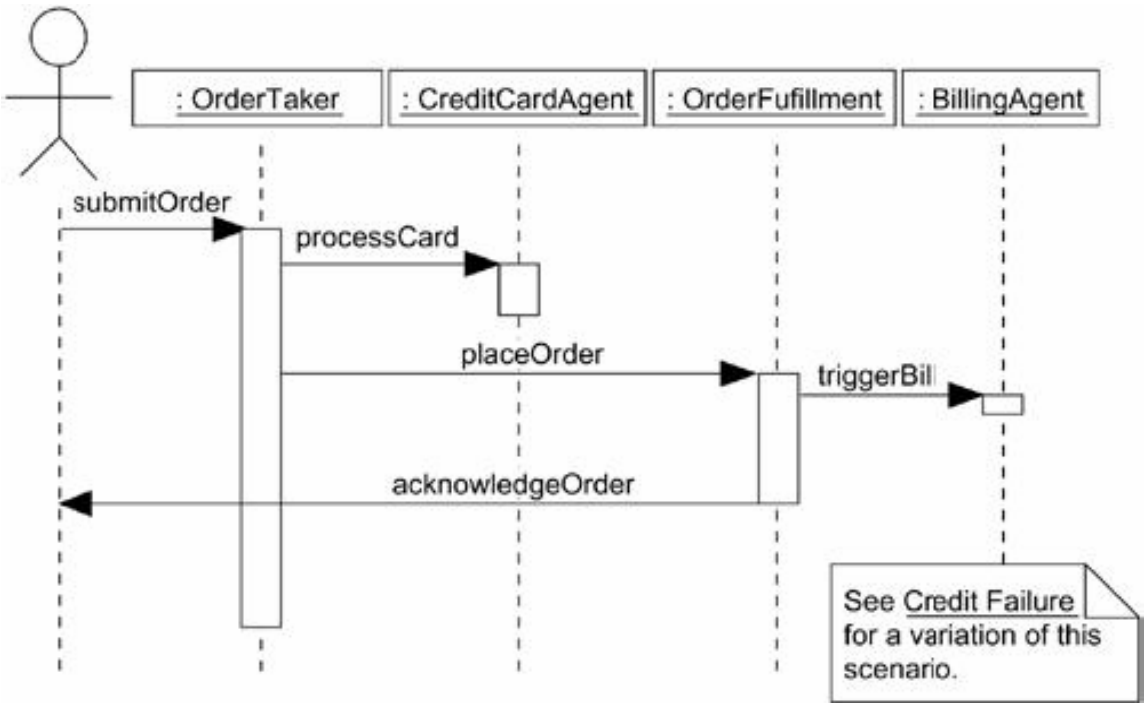
براي هر مدل با توجه به Abstraction كه دارد دياگرام هاي مختلفي را كه ايجاد كنيم. يعني با توجه به ديد و نيازي كه شخص دارد دياگرام ايجاد كنيم. كه نياز به چهار قسمت زير داريم :

1. ساختن بلاک ها و ارتباط ها
2. توضيحات
3. جريانها
4. Streotype

مثال ۱: یک نمودارتوالی برای یک تحلیل گر سیستم در سطح بالا ست .



مثال ۲: یک نمودارتوالی برای یک برنامه نویس سیستم که سطح آن نسبت به تحلیل گر کمتر است .



سيستم را دوباره بازسازي مي كنيم .

چگونه با سيستم هاي پيچيده كار كنيم؟

①مدل را تقسيم بندي مي كنيم . سيستم را به دياگرام هاي مختلف بشكنيم . اگر اين كار را كرديم و هنوز پيچيده بود مي توان از بسته بندي كردن قسمت ها در لايه بالا تر استفاده كرد .

②اگر باز هم پيچيده است مخفي كردن يكسری از اجزانمودار و از يادداشت ها و رنگ ها براي جلب توجه كابر ايجاد كنيم .

③اگر باز هم پيچيده بود يك پرينت كامل از مدل گرفته و در محل كار و يا اتاق آويزان كنيم.

نكته:

-هدف از مدل نقاشي نيست.

- همه دياگرام ها لازم نيست نگه داري شوند و بايد از مدل حذف شوند ولي ان ها را خارج از مدل نگه داري كنيم شايد در اينده به ان ها نياز پيدا كرديم.

- بين نمودار هاي رفتاري و ساختاري توازن برقرار باشد.

- دياگرام ها نبايد خيلي كوچك و نبايد خيلي بزرگ باشد.

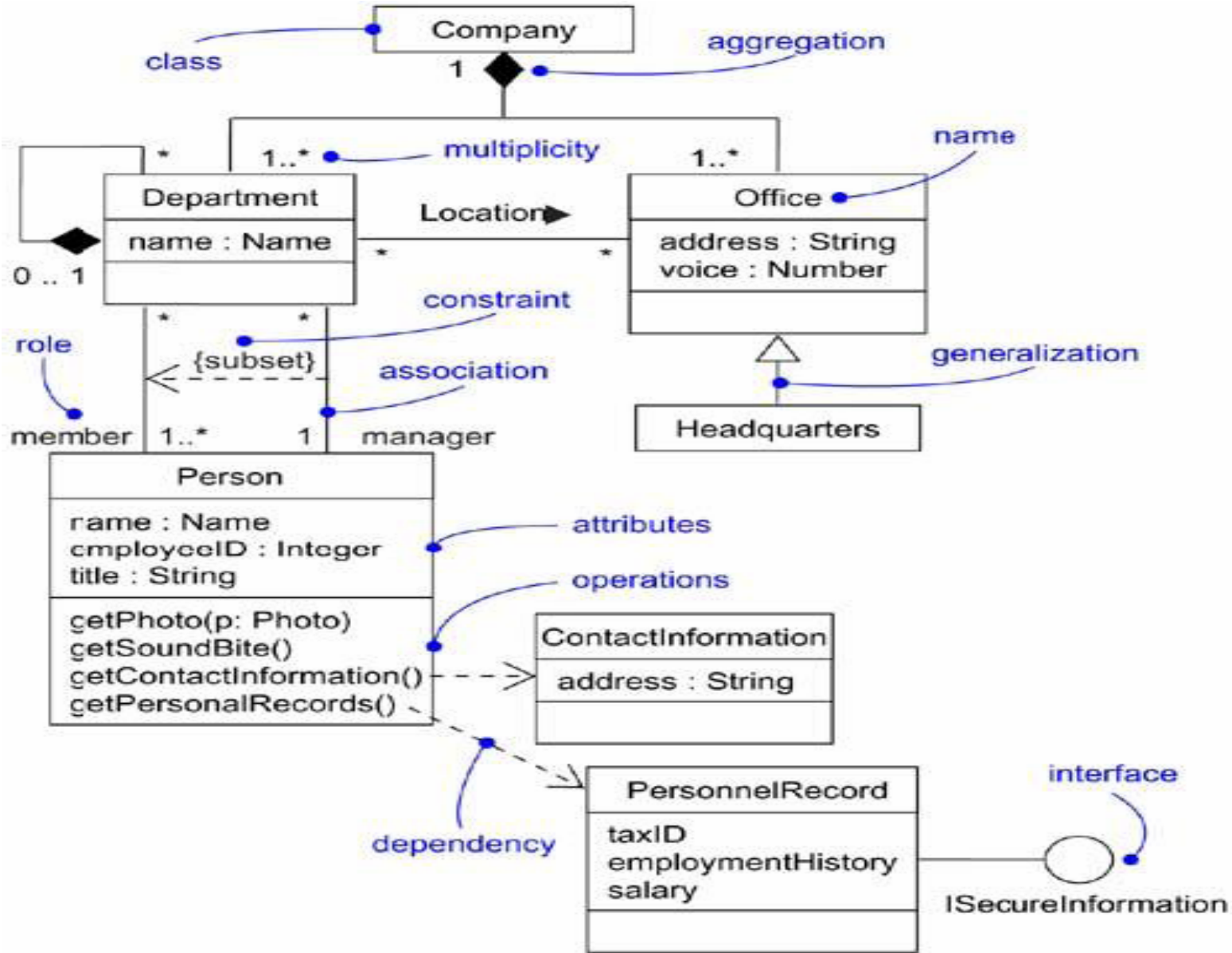
- در رسم دياگرام سعي كنيم كمترين خطوط متقاطع را داشته باشيم.يعني حداقل خطوط را داشته باشيم.ضمنا عناصر مرتبط معنايي را در يك قسمت از دياگرام قرار دهيم.

فصل هشتم

کلاس دیاگرام

Class Diagram

کلاس دیاگرام



مفهوم کلاس دیاگرام :

کلاس دیاگرام از مجموعه ای از کلاس ها ، رابط ها و همکاری و ارتباط ها بصورت گرافیکی می باشد .

کلاس دیاگرام معمولا شامل اشیا زیر می باشد :

- کلاس ها
- رابط ها
- همکاریها
- ارتباط های وابستگی ، همکاری و تعمیم

کلاس دیاگرام مانند دیگر دیاگرام ها ممکن است شامل یادداشت های توضیحی و محدودیت باشد .

کلاس دیاگرام ممکن است که شامل package یا subSystem یا هر دو باشد.

عمده ترین جایی که از کلاس دیاگرام استفاده می شود دید طراحی است .

وقتی که یک دید طراحی استاتیک از سیستم را مدل می کنید شما از کلاس دیاگرام ممکن است بعنوان یکی از سه مورد زیر استفاده کنید .

از کلاس دیاگرام در چه مواردي استفاده مي کنند؟

①در مدل کردن لغت نامه سيستم
②براي مدل کردن همکاري هاي ساده
③مدل کردن شماي منطقي پايگاه داده ها

مدل کردن لغت نامه سيستم :
To model the vocabulary of a system

منظور فصل چهارم که در رابطه با کلاس ها می باشد برای توضیحات بیشتر به این فصل مراجعه کنید .

براي مدل كردن همكاري ساده:

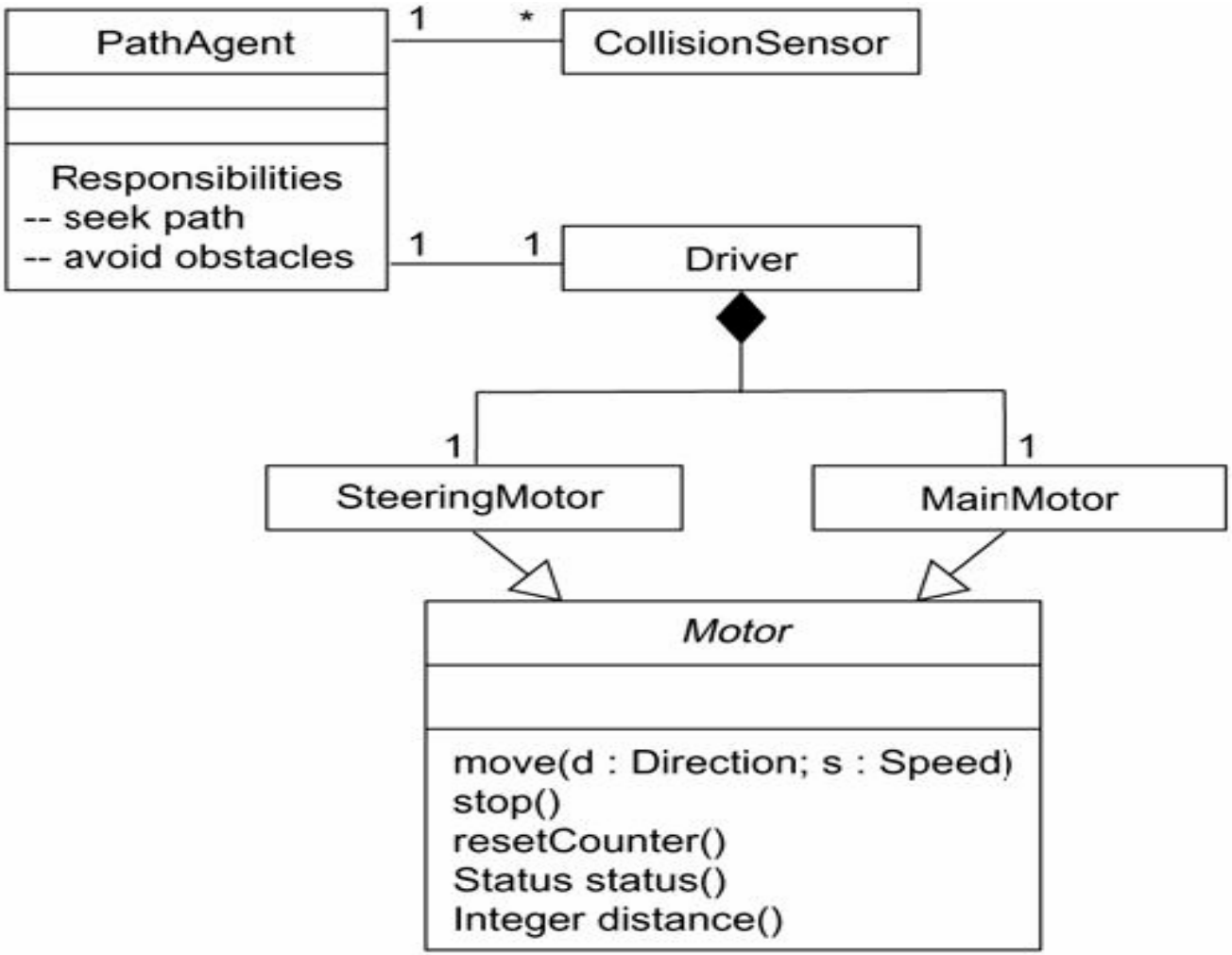
1. يكي از رفتار هاي سيستم را بر مي داريم.

2. براي هر رفتاري كه بر مي داريم كليه اينترفيس ها و رفتار ها و Collaberation ها را بر مي داريم و ان كلاس را جدا مي كنيم.

3. بررسي بيشتري روي ان رفتار انجام مي دهيم و بررسي مي كنيم كه آيا براي ان رفتار ها اين كلاس ها كافي است يا نه؟ كه اگر كافي نبود به آن رفتار ها كلاس اضاف كنيم.

4. مجموعه ان كلاس ها را با يك علامت Collaberation يا همكاري مشخص كنيم.در اين مرحله رسم صورت می گيرد .

مثال : مدل کردن یک همکاری ساده



براي مدل كردن بانك اطلاعاتي:

①دانه به دانه كلاس ها را بررسي مي كنيم.از اين جهت كه كدام كلاس عمر داده هايي كه در درون ان است بيش از عمر برنامه است .

②يك كلاس دياگرام را رسم مي كنيم كه فقط طول عمر داده هايش بيش از طول عمر برنامه هايش باشد.هر كدام از اين كلاس ها را با يك Tagged value مشخص مي كنيم.
Tagged valueرا به عنوان يك Persistent بررسي مي كنيم.

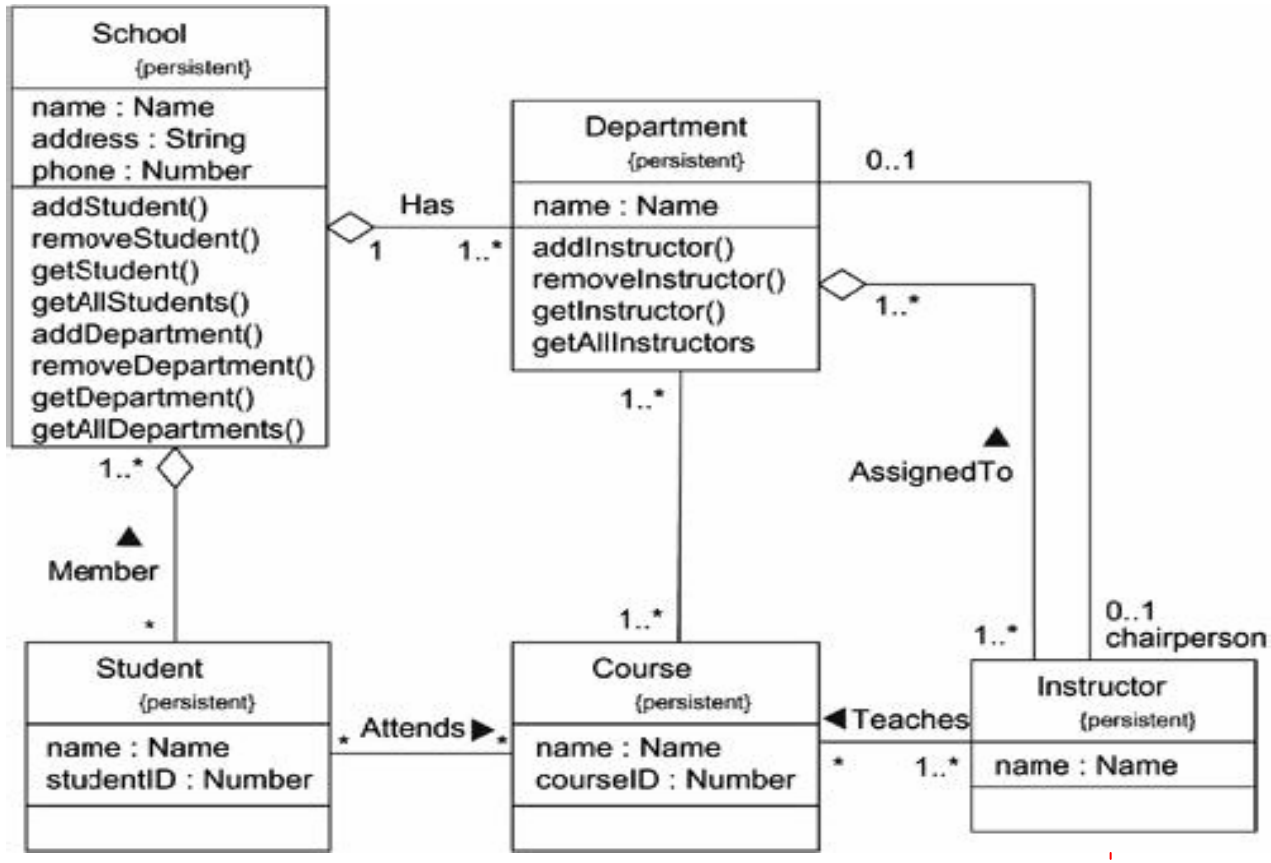
③ Attribute ها را دقيقا مشخص مي كنيم Type.و اسم ان ها را مشخص مي كنيم و مي گوييم ايا اين مشخصات ذخيره مي شوند يا نه؟

④به وضعيت رابطه هاي يك به يك و يك به چند و چند به چند رسيدگي مي كنيم.اگر لازم باشد يك سري عناصر را مثلا در رابطه چند به چند اضاف مي كنيم.

⑤روي هر كلاس دقت شود كه چه كارهاي اطلاعاتي انجام شود.عمليات لازم براي هر كلاس را در صورت لازم توسعه مي دهيم.

⑥در اين مرحله از يك ابزار براي توليد خودكار شماي بانك استفاده مي كنيم.

مثال : مدلسازی یک شما



persistent

مهندسي مستقيم و مهندسي معكوس دياگرام:

نكته: دياگرامي كه بيشترين توانايي مهندسي مستقيم رادارد كلاس دياگرام مي باشد.
نمي توان از روي يك Source code كلاس مربوط را از آن بيرون كشيد. اما

براي مهندسي مستقيم كارهاي زير را انجام مي دهيم:

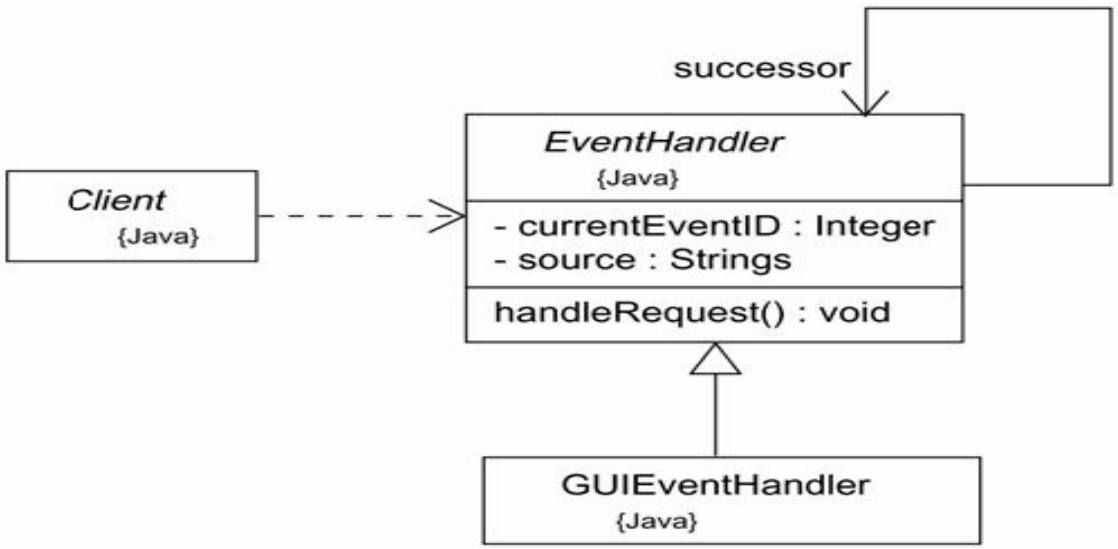
1. بايد مشخص كنيم كه اين mapping كه مي خواهيم تحويل دهيم به چه زباني است؟ يعني كد هاي ما به چه زباني است؟

2. بسته به زباني كه انتخاب مي كنيم محدوديت هايي اعمال مي شود كه بايد محدوديت ها را روي مدل بررسي كنيم.مي توان از تجمع براي راه حل در وراثت چند گانه استفاده كرد.

3. با استفاده از Tagged value ها مي توان زبان هر كلاس را علامت گذاري كنيم. مي توان در كلاس در Package و يا در Collaberation گذاشت.

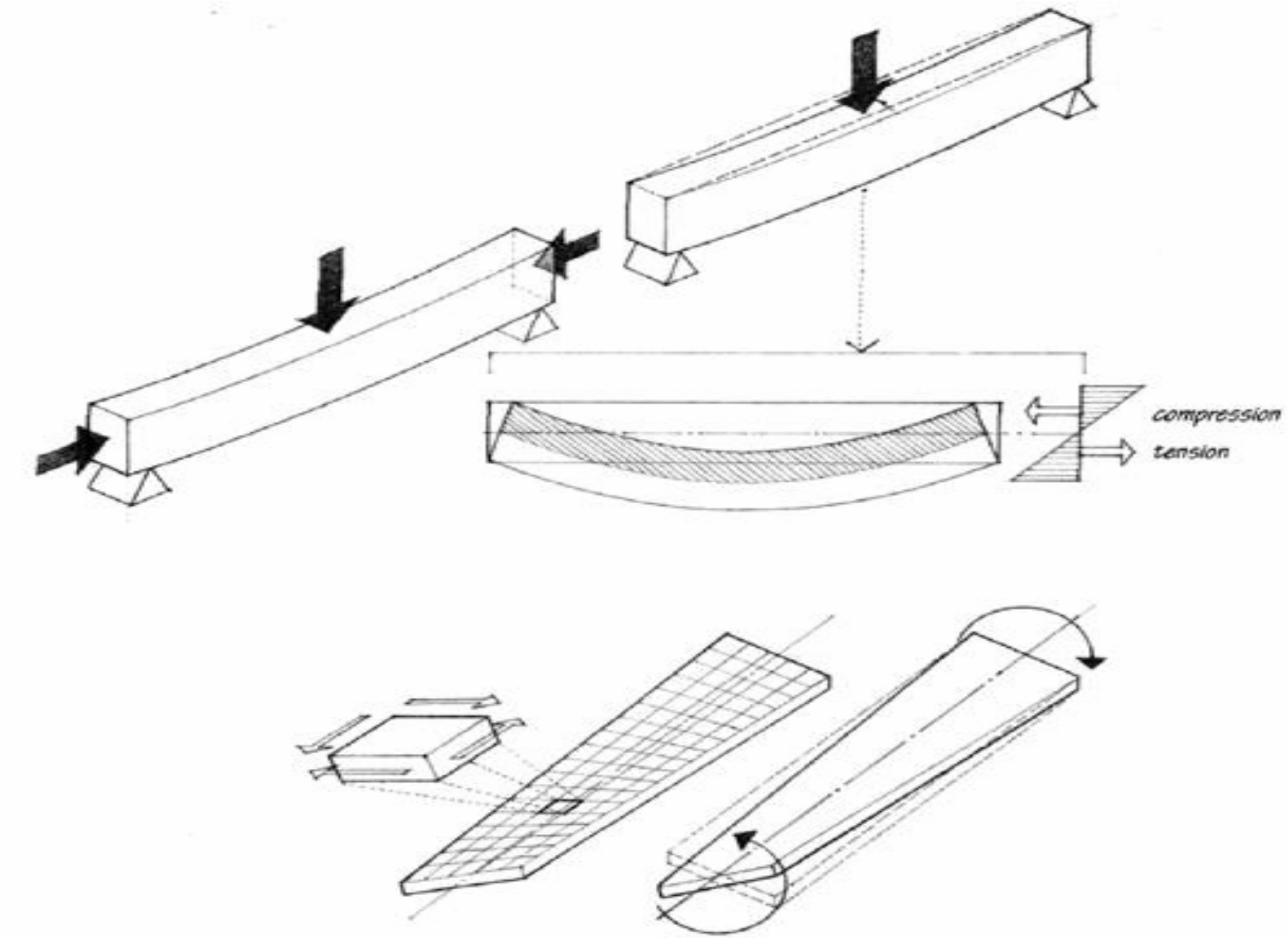
4. مي توان از ابزار كمك گرفت.(تا حد امكان از ابزار ها براي مدل هايمان استفاده كني).

مثال : مهندسی مستقیم




```
public abstract class EventHandler {
        EventHandler successor;
        private Integer currentEventID;
        private String source;
        EventHandler() {}
        public void handleRequest() {}
}
```

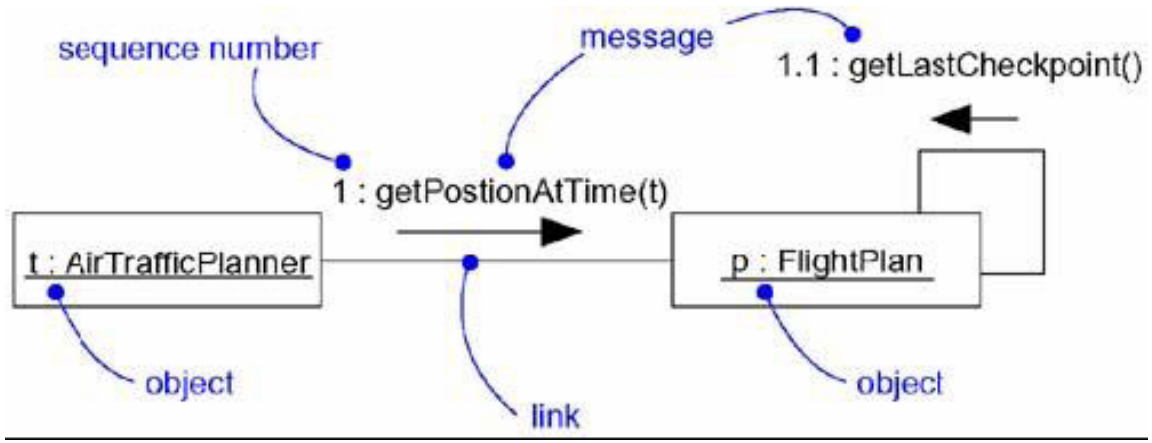
Basic Behavioral Modeling



فصل نهم

Interactions

مثال : پیغام ها ، لینک و توالی



Interactionچیست؟

نمايش دهنده يك تعامل است كه تشكيل شده از مجموعه اي از اشيا و ارتباط بين آن ها و همين طور پيغام هايي كه بين اشيا رد و بدل مي شود .

Interaction

از دو دیاگرام Sequence و Collobration تشکیل شده است .

تعریف Interaction : یک رفتار که تشکیل شده از مجموعه ایی از تبادل پیام ها میان مجموعه ای اشیا در یک زمینه خاص برای انجام یک هدف می باشد .

استفاده از interaction برای مدل کردن جنبه دینامیک از همکاریها ، به نمایندگی مجمعی از اشیایی که نقش ایفا می کنند و همه با هم کار می کنند برای اجرای برخی از رفتارها .

Sequence diagram : يك interaction دياگرامي است كه بر توالي زمانبندي پيغام ها تاكيد دارد.

Collaberation diagram : يك interaction دياگرامي است كه بر ساختار و سازماندهي انتقال پيغام ها بين اشيا تاكيد دارد.

اين دياگرام براي دو هدف استفاده مي شود:

❶براي مشخص كردن سيستم .

❷براي اين استفاده مي شود كه مي خواهيم از روي ان كد بنويسيم.

: Sequence diagram

آن را به صورت يك جدول نمايش مي دهيم.

- در محور X ها اشيايي كه در ان تعامل شركت دارند رسم مي شوند.
- معمولا ترتيب قرار گرفتن اشيا از چپ به راست بر حسب تعامل هايي كه در آن شركت مي كنند مي باشد.

: Collaboration diagram

در اين قسمت فقط بين اشيا يي كه با هم تعامل پيغامي دارند مشخص مي شود.

در حالت كلي سه چيز در تعامل وجود دارد:

① اشيا Object
② پيوند ها Link
③ پيغام ها Message

Sequence diagramنسبت به collaboration diagram داراي دو چيز اضافه تري هستند:

❶ Life line : نقطه ايجاد شي تا نقطه از بين رفتن شي رامشخص مي كند.

❷ Focus of control : نقطه اي كه در ان كنترل وجود دارد.ان را با يك مستطيل ضخيم نمايش مي دهند.

در sequence diagram مي توان دو چيز را اضاف كرد:

①امكان شرط گذاشتن : در صورتي كه ان پيغام اجرا شود ان شرط برقرار مي شود.

② actorها : كه شروع كننده يك فعاليت و يا تعامل هستند.

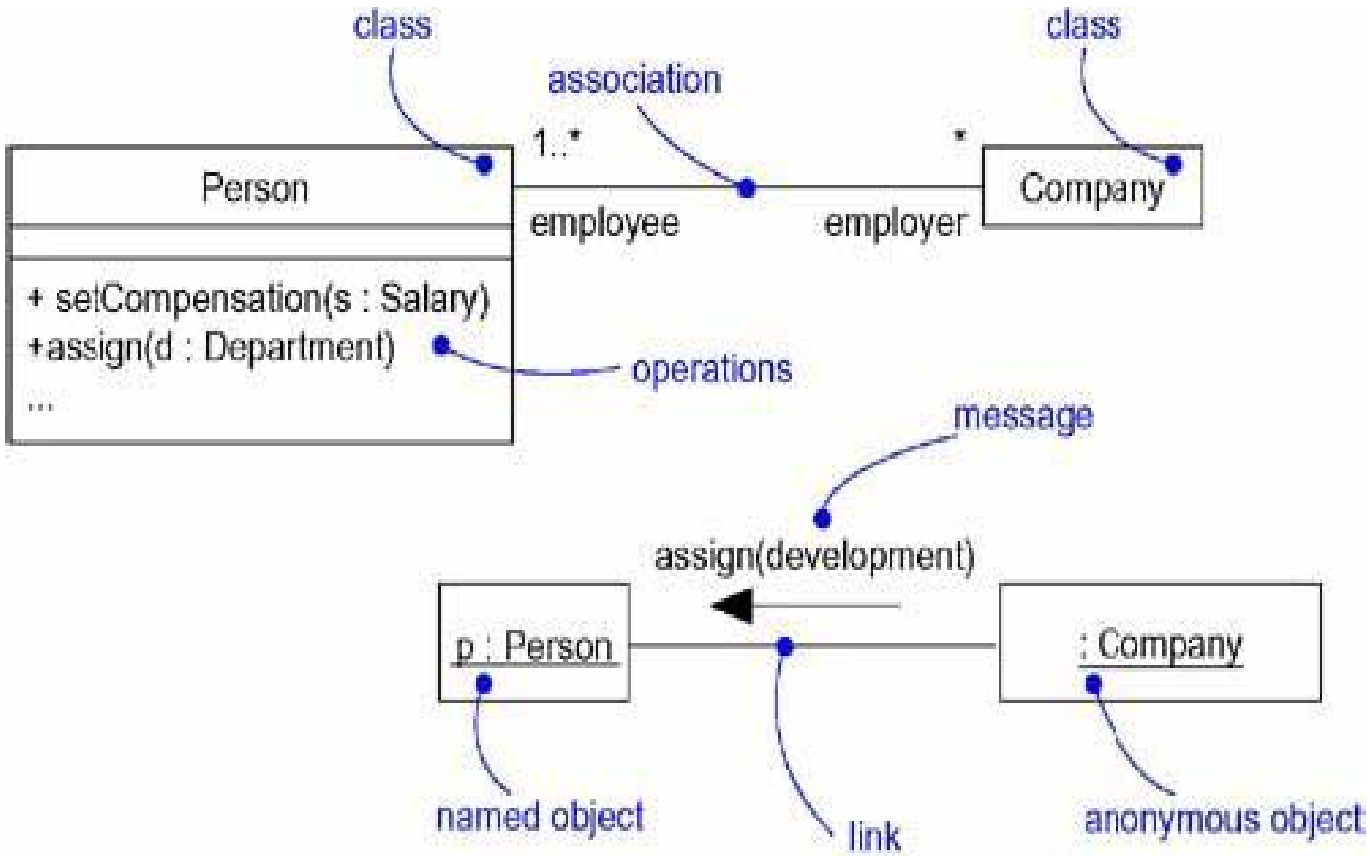
در collaberation diagram مي توان دو چيز را اضافه كرد:

1. بحث شماره گذاري
2. بحث شرط گذاري روي پيغام ها

از sequence diagram در موارد زير استفاده مي شود:

1. براي كنترل جريان برنامه
2. براي زمان بندي

مثال : Link و همکاری



پیغام Message : پیغام هایی که بین اشیا وجود دارد به یکی اشکال زیر می باشد :

- Call
- Return
- Send
- Destory
- Create

Call : صدا زدن یک متد ازیک شی

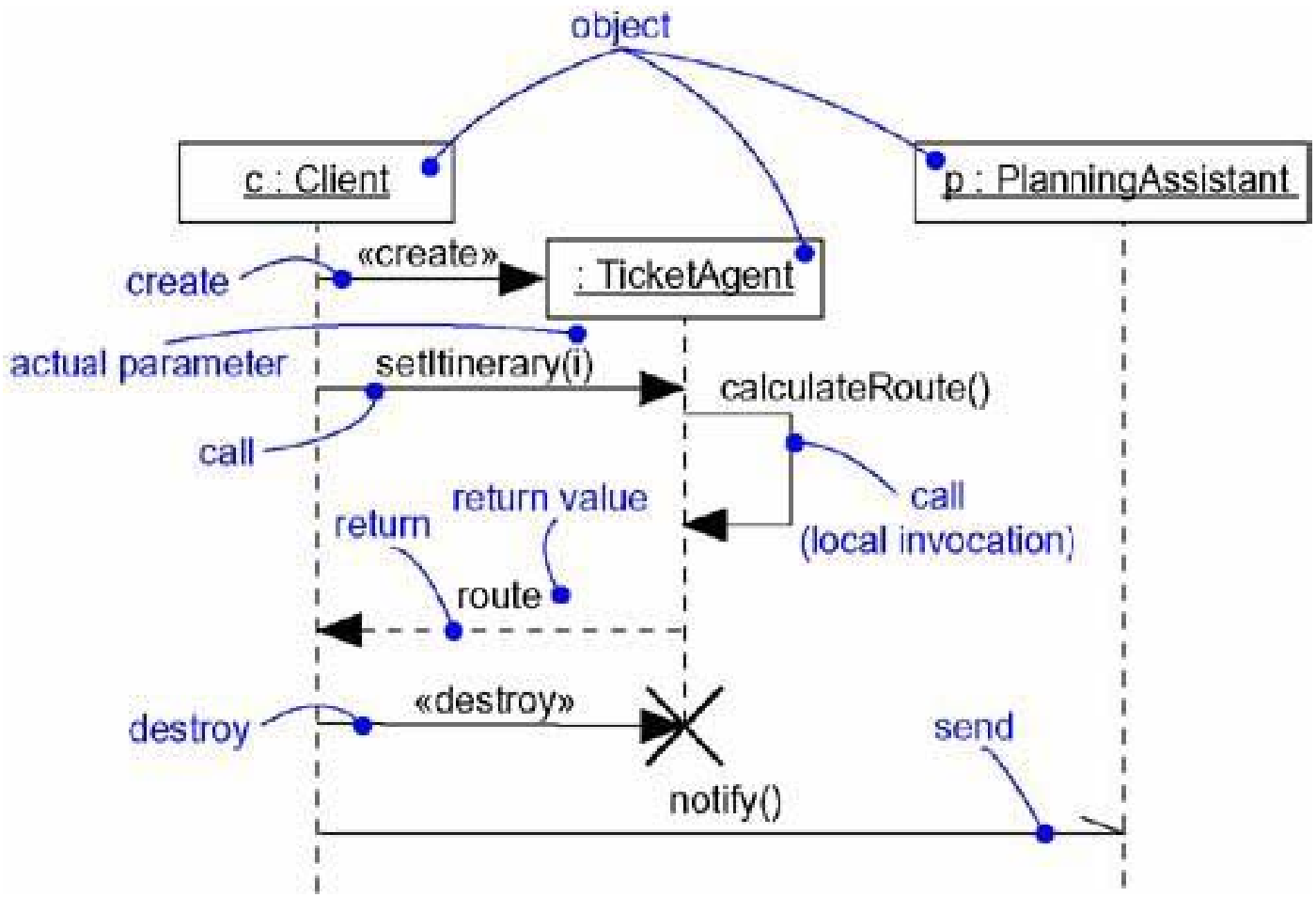
Return : تمام شدن کار یک متد

Send : سیگنال فرستادن بین اشیا مانند : wait و notify

Create : بوجود آوردن یک شی

Destory : از بین بردن اشیا

Message : مثال



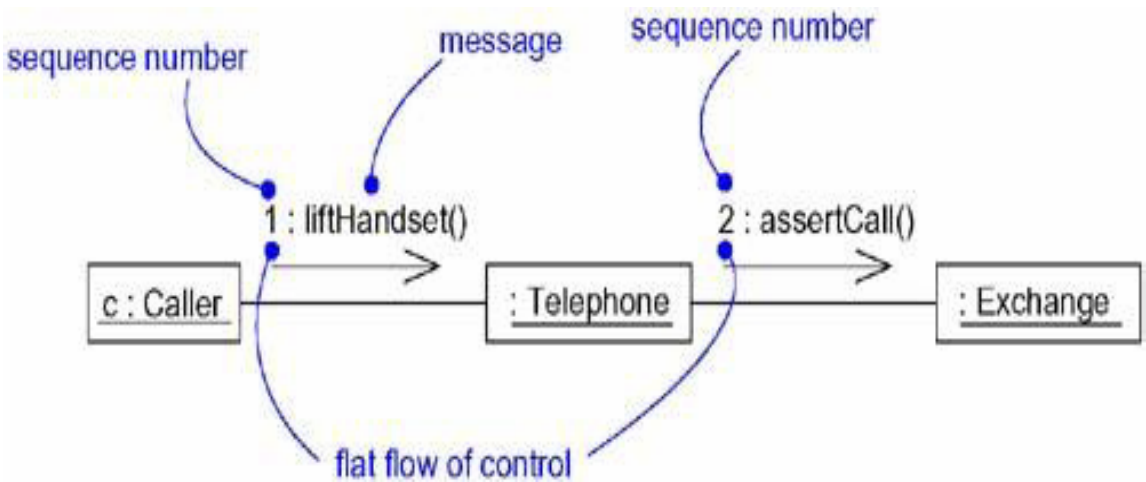
Sequencing : توالی یا عملیات پشت سر هم

دو نوع توالی وجود دارد :

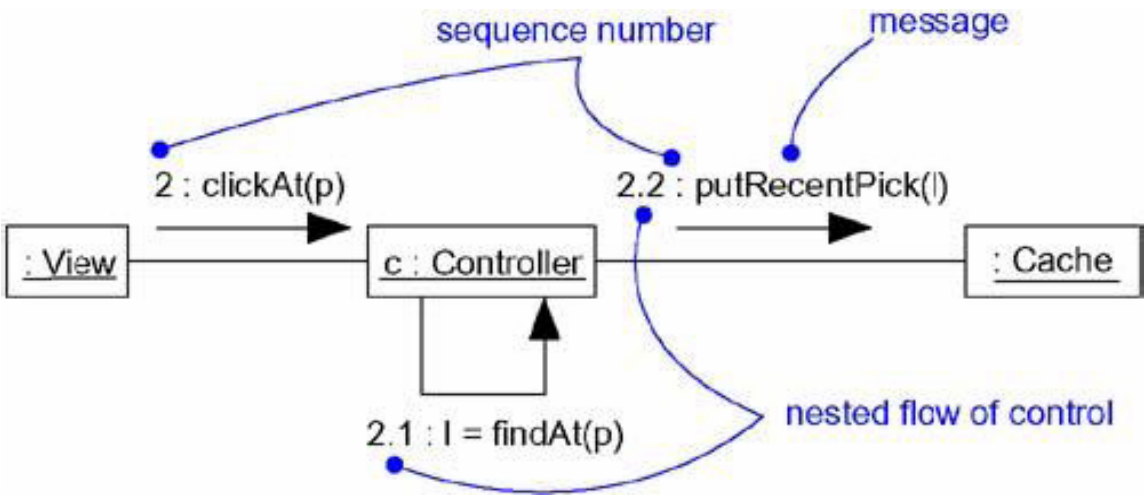
❶توالی همسان Flat : شماره گذاری به ترتیب می باشد از ابتدا تا انتها

❷توالی رویه ای Procedaral : شماره گذاری رویه ای می باشد به یک متد شماره ای می دهیم اگر همین متد ، متد بعدی را صدا زد شماره بالاتری به متد بالایی می دهیم به همین ترتیب تا انتها

Flat Sequence : مثال



Procedural Sequence : مثال



D5 : ejectHatch(3)

از D که نقطه شروع می باشد ۵ مرحله به جلو حرکت می کنیم .

1.3.2 : p := find("Rachelle")

از نقطه شروع که ۱ باشد به قسمت ۳ آن و سپس به قسمت ۲ آن حرکت می کنیم .

براي مدل سازي كنترل جريان برنامه :

1. براي چه Interaction مي خواهيم Sequence رسم كنيم . مثلا در سطح سيستم يا در سطح زير سيستم و غيره.

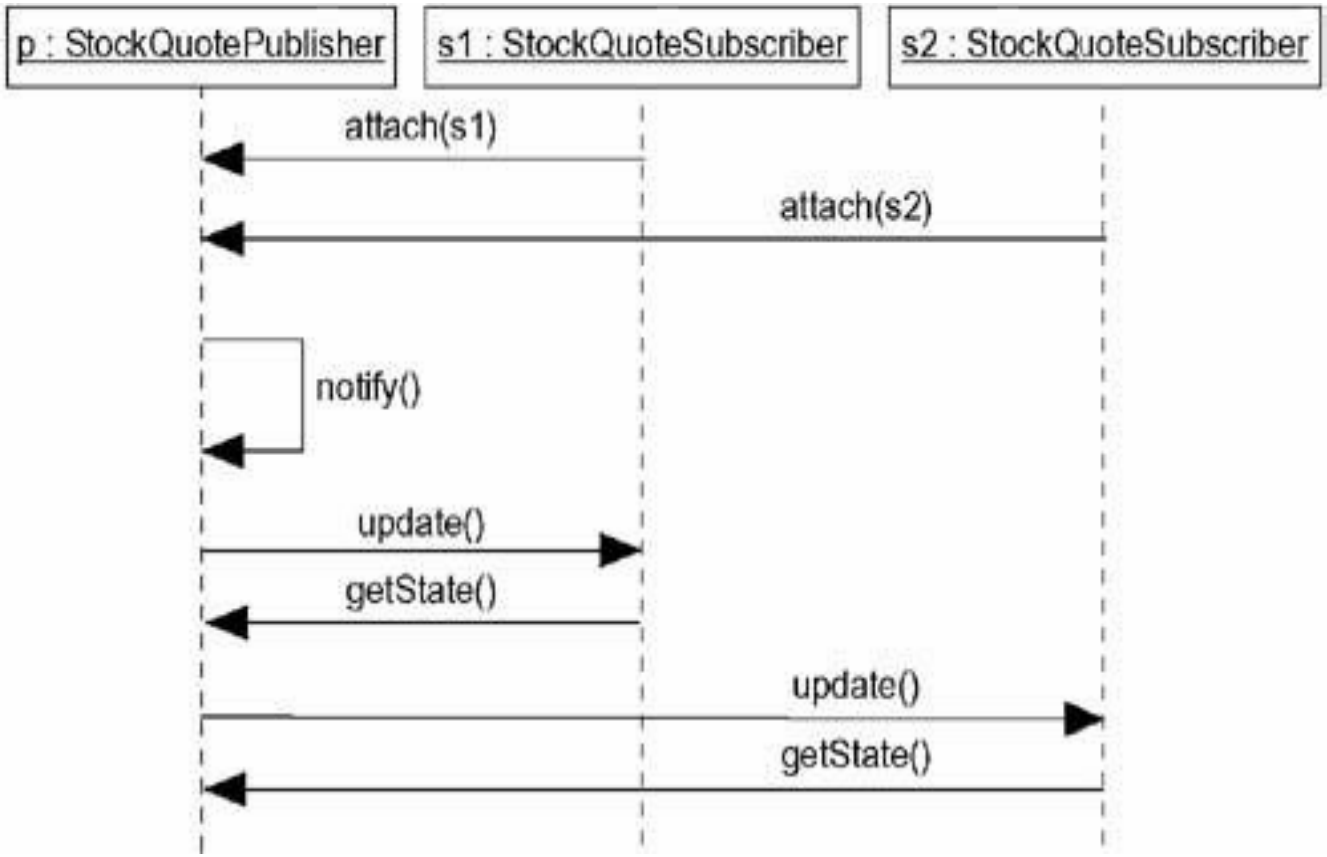
2. اشيايي كه در تعامل نقش دارند را مشخص مي كنيم.اشيا را از بالا به پايين و از چپ به راست بر حسب نقش و زمان اجرا مشخص مي كنيم.

3. براي همه اشيا خط زندگي ان ها را مشخص مي كنيم.

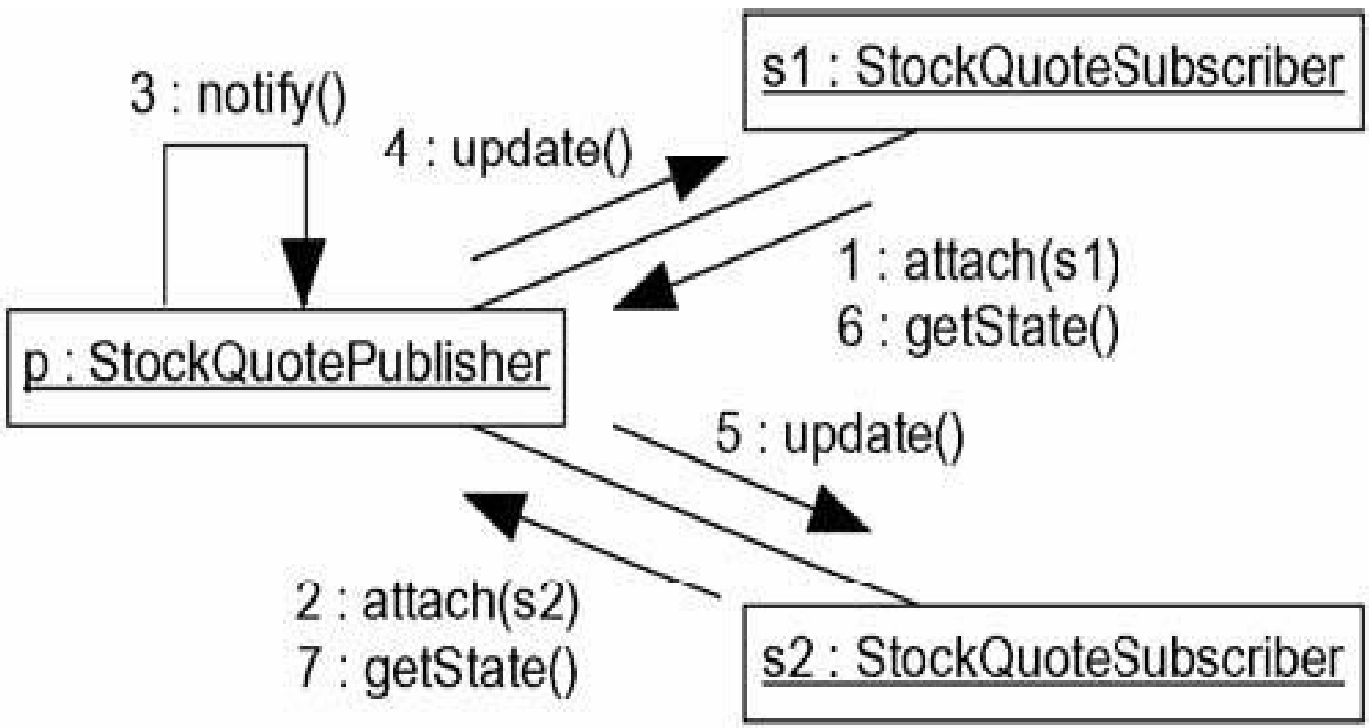
4. با پيغام شروع كننده Interaction شروع مي كنيم.الگوريتم و برنامه را Trace كرده و قدم به قدم پيش مي رويم و پيغام ها را به دياگراممان اضافه مي كنيم . در اين قسمت اگر لازم باشد براي برنامه شرط يا پيغام مي گذاريم.

5. پيغام ها را شماره گذاري مي كنيم.

Flow of Control by Time : مثال



Flow of Control by Organization : مثال



براي مدل كردن جريان كنترل با استفاده از ساختار يا سازماندهي پيغام ها:

① objectهايي كه در اين تعامل نقش دارند را مشخص مي كنيم.اشيايي كه انتقال پيغام زيادي به ان ها انجام مي شود را در وسط صفحه قرار داده.اگر attribute هاي ان شي به نحوي تغيير كرده باشد كه در يك شي ديگر قرار بگيرد هر دو شي را ذكر مي كنيم و با رابطه هاي زير ارتباط بين ان ها را ذكر مي كنيم :

- ✓ رابطه Become تبديل شدن
- ✓ رابطه Copy از شي مورد نظر اصلي مان كپي مي گيريم تا شي ديگر توليد شود .

②ارتباط بين اشيا را مشخص مي كنيم.

نكته : براي چهار دياگرامclass diagram و sequence diagram و collaboration diagram و state chart diagram مي توان مهندسي مستقيم و معكوس استفاده كرد.

از شرط و چرخش زياد در collaboration diagram و sequence diagramاستفاده نكنيم .بلكه بهتر است درactivity diagram قرار دهيم زيرا activity دياگرام حالت الگوريتم را دارد.

فصل دهم

Use case
&
use case diagram

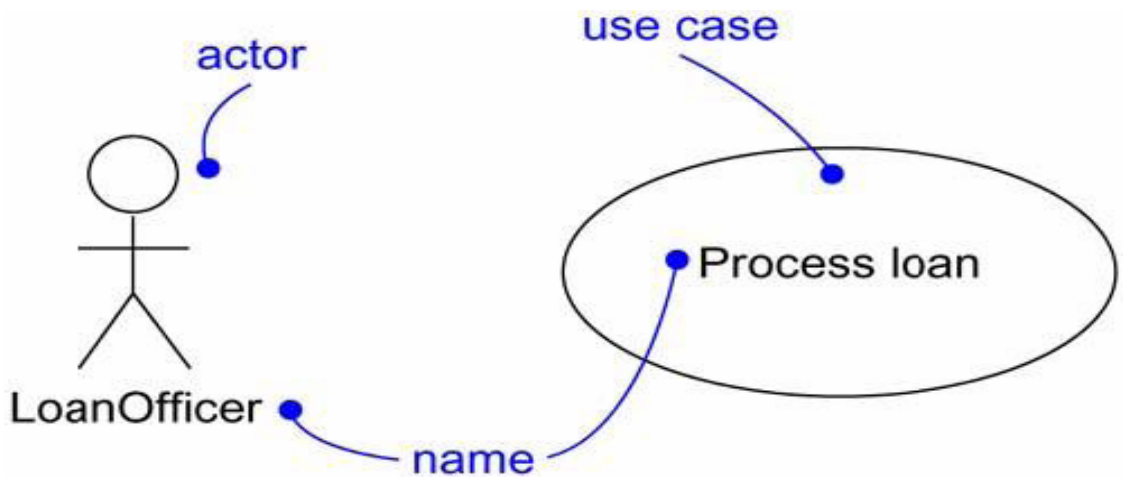
Use Case : مشخص کننده رفتار یک سیستم یا قسمتی از یک سیستم می باشد در حقیقت با توالی از عملیات تشکیل شده کهسیستم انجام می دهد تا یک نتیجه را به Actor یا عامل برساند .

و مجموعه ای از interaction هاست که در یک طرف اشیا هستند که خارج از سیستم هستند .

سیستمی موفق است که که کلیه انتظارات Actor بر آورده کند .

Actor (عامل) : یک عامل نمایش دهنده یک مجموعه همسان از نقش هاست که کاربر use case در تعامل با use case بازی خواهد کرد .

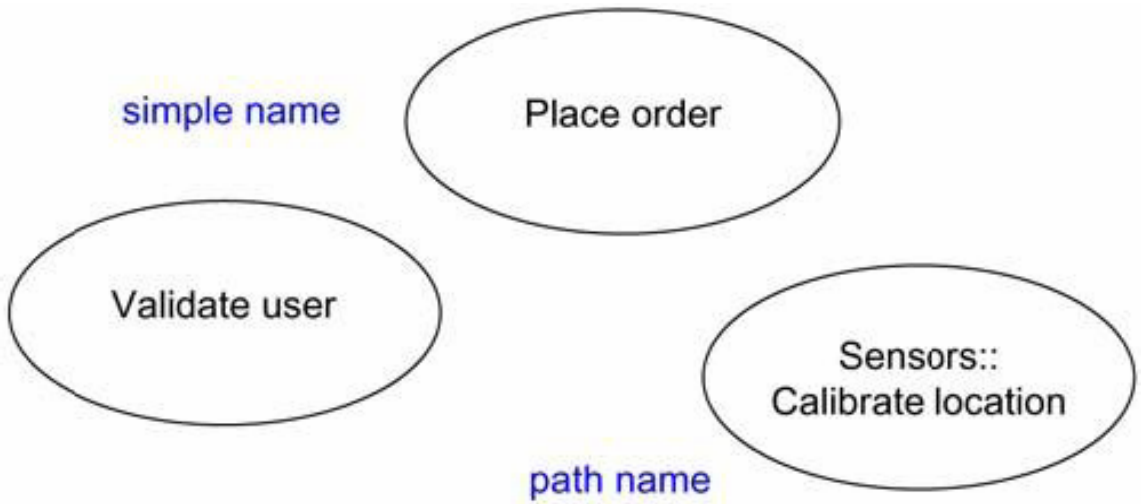
انواع Actor : انسان ها ، سیستم های خود کار دیگر بیرون از سیستم و دستگاه سخت افزاری



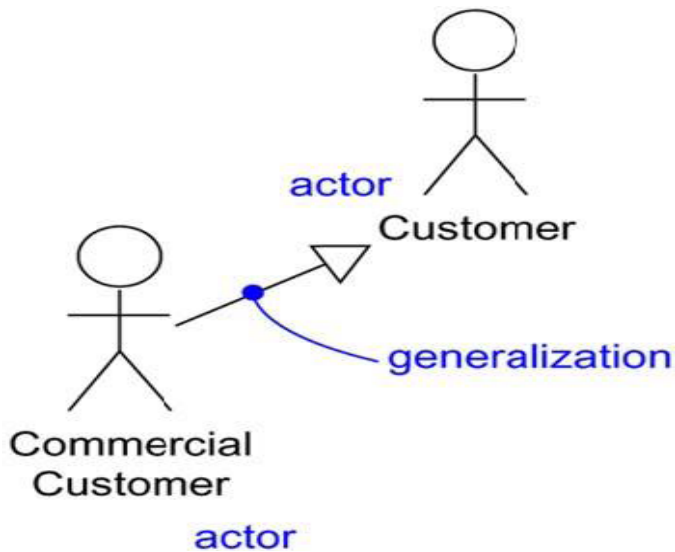
Names (نام) :

نام یک use case در یک package می بایست منحصر به فرد باشد .

می توان نام آن را ساده یا با مسیر مشخص نمود مانند زیر :



- رابطه بین Actor ها generalization می باشد .



برای نمایش عامل ها به شکل های مختلف از streotype استفاده می شود .

کارهایی که در use caseانجام می شود :

جریان رویداد ها در use case

روی کاری که درون سیستم انجام می شود در مورد چگونگی انجام کار در این قسمت بحث نمی شود بلکه کارهایی use case باید انجام دهد مورد بررسی قرار می گیرد .

انواع روش برای نمایش انجام کار use case :

- ✓استفاده از متن ساده
- ✓ Activity Diagram

نوشتن جریان رویدادها در متن ساده :

❶ جریان اصلی رویداد روی سیستم (رفتار های نرمال عامل)
مانند زمانیکه شخص پسورد درست وارد کند .

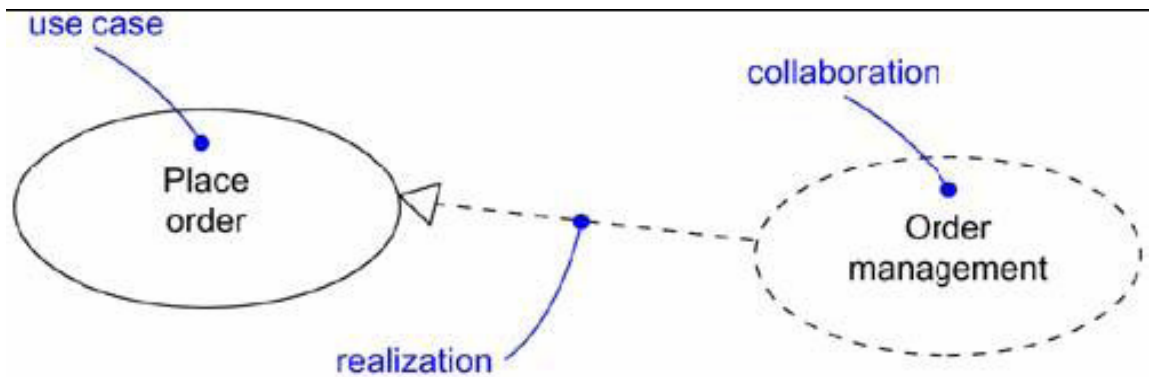
❷ جریان استثنا در سیستم
مثلا زمانیکه فرضا شخص سه مرتبه پسورد درست وارد نکند).

سناریو و use case

رابطه سناریو با use case مانند رابطه ی شی با کلاس می باشد
پس برای use case ممکن است سناریو های مختلفی ایجاد شود .

همکاری ها و use case

رابطه بین collaboration و use case ، realization
می باشد .



سازماندهی use caseها :

ما می توانیم use case های سیستم را در گرو ههایی درون بسته تقسیم بندی و سازماندهی کنیم با سه مکانیزم سازماندهی که عبارتند از :

- Generalization (تعمیم)
- include (شمولیت)
- Extend (توسعه)

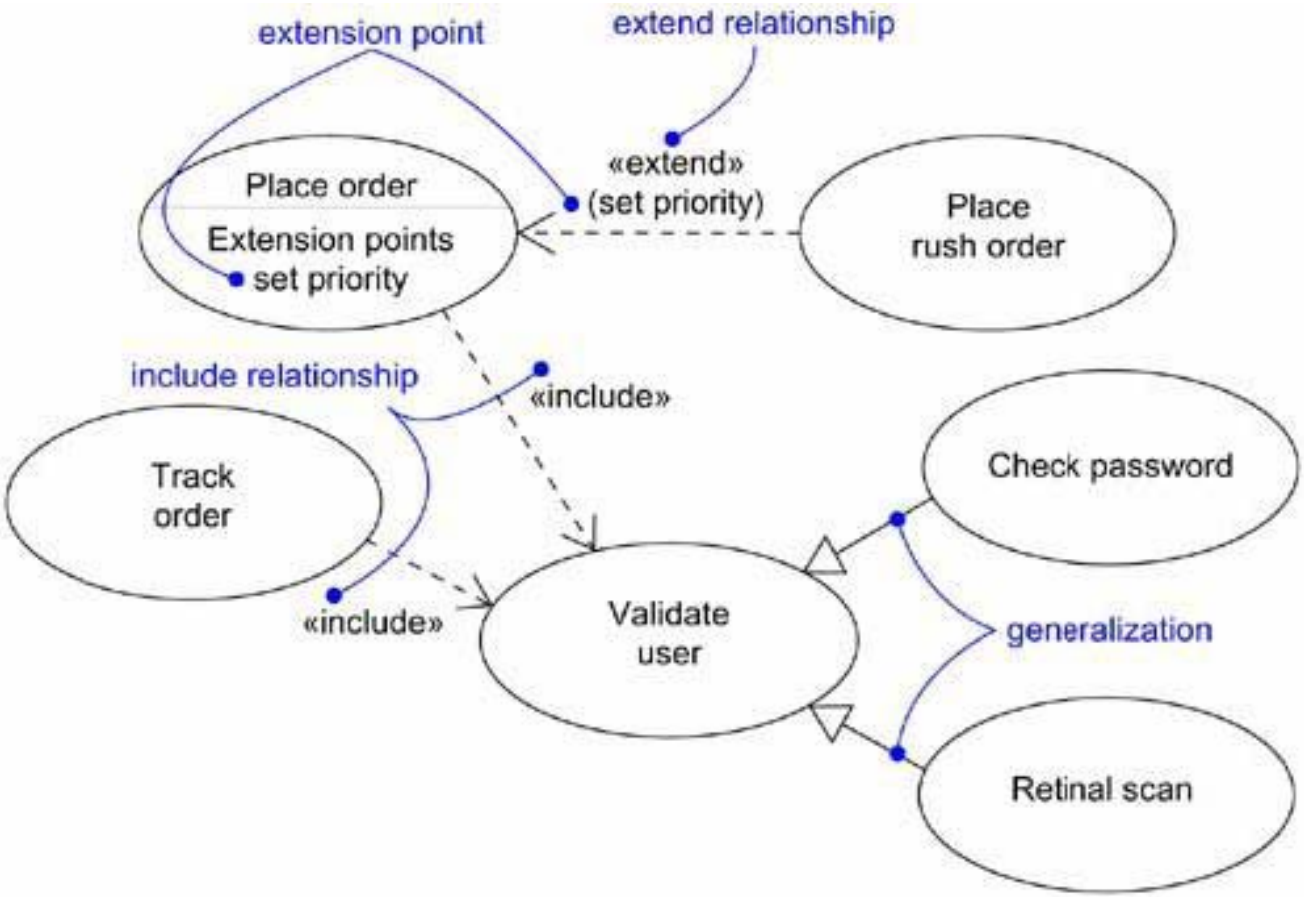
رابطه تعمیم بین use case ها خیلی شبیه رابطه تعمیم بین کلاس هاست .

در use case دیاگرام زمانیکه می خواهیم use caseای را که حتما انجام شود تا نیاز Actor برآورده شود آن را با include نشان می دهیم .

Extend : use case ای که حالت خاصی از use case ای که include شده را توسعه داده و Actor می تواند آن use case را انجام داده یا از آن صرف نظر کند .

برای توضیح بیشتر و ایجاد یک use case سازماندهی شده در طی یک مثال تحت عنوان سفارش کتاب شرح خواهیم داد .

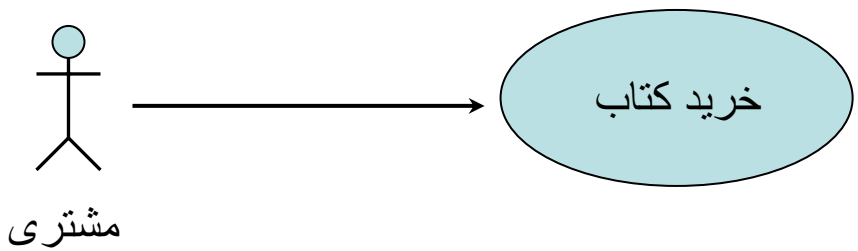
Generalization, Include, and Extend



مثال : رسم use case خرید یا سفارش کتاب .

برای رسم این نمودار ابتدا نیاز عامل را بررسی کرده سپس تحت عنوان یک use case در آورده حال اگر این کاربرد حجیم و پیچیده باشد همانطور که قبلا ذکر کردیم می بایست به کاربرد های کوچکتری تجزیه کنیم سپس اعمال مکانیزم های سازماندهی به این دیاگرام انجام خواهیم داد .

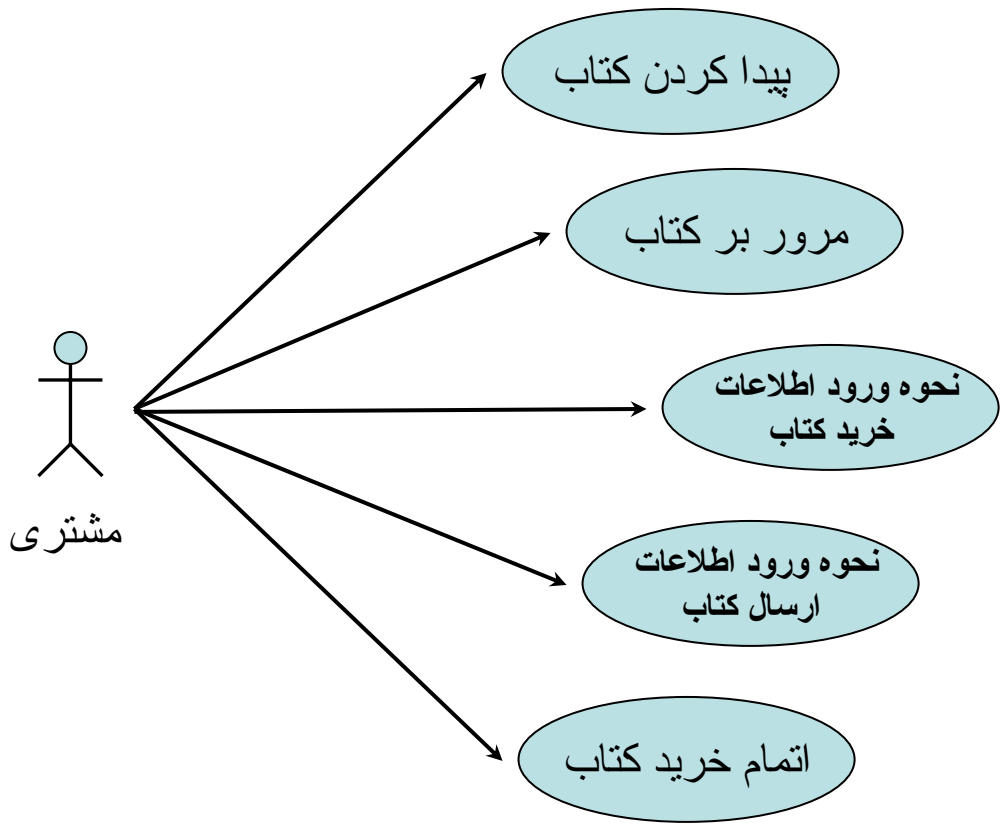
داریم :



همان طور که مشاهده می کنید این کاربرد چندان چیزی را بیان نمی کند و مبهم خواهد بود پس نیاز داریم این کاربرد را تفکیک کنیم .

تفکیک use case خرید کتاب

داریم :



اعمال روابط include و extend

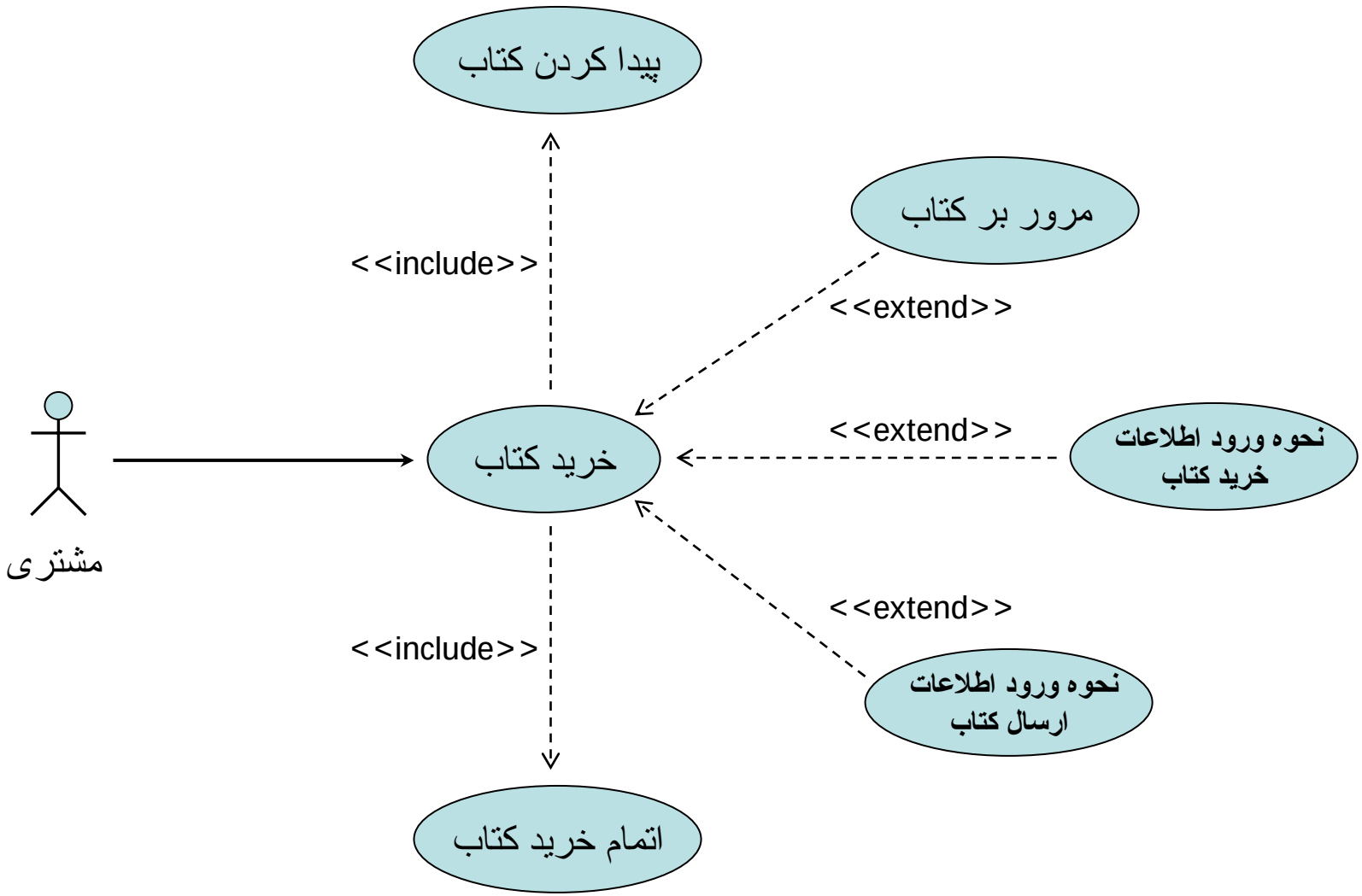
رابط include این امکان را به ما می دهد که یک کاربرد را همواره درون یک کاربرد دیگر قرار داد به این شکل که کاربرد اصلی باید به کاربرد درج شده وابسته گردد چرا که مجبور به استفاده از عملیات اوست . این رابطه دو مزیت دارد :

- Use case اصلی ساده شده
- Use case درج شده را می توان در بیش از یک use case اصلی استفاده کرد .

رابط extend این امکان را فراهم می کند تا یک کاربرد را به صورت یک گزینه داخل کاربرد دیگر قرار داد به این شکل که کاربرد اصلی همواره کاربد توسعه را در بر نمی گیرد بلکه بسته به وجود شرایطی که کاربرد اصلی تعیین می کند ممکن است جریان رویدادهای کاربرد توسعه به جریان رویداد های کاربرد اصلی اضافه گردند . وابستگی از سمت کاربرد توسعه به سمت کاربرد اصلی است .

اکنون این مکانیزم ها را اعمال خواهیم کرد .

کاربرد خرید کتاب به همراه رابط های include و extend

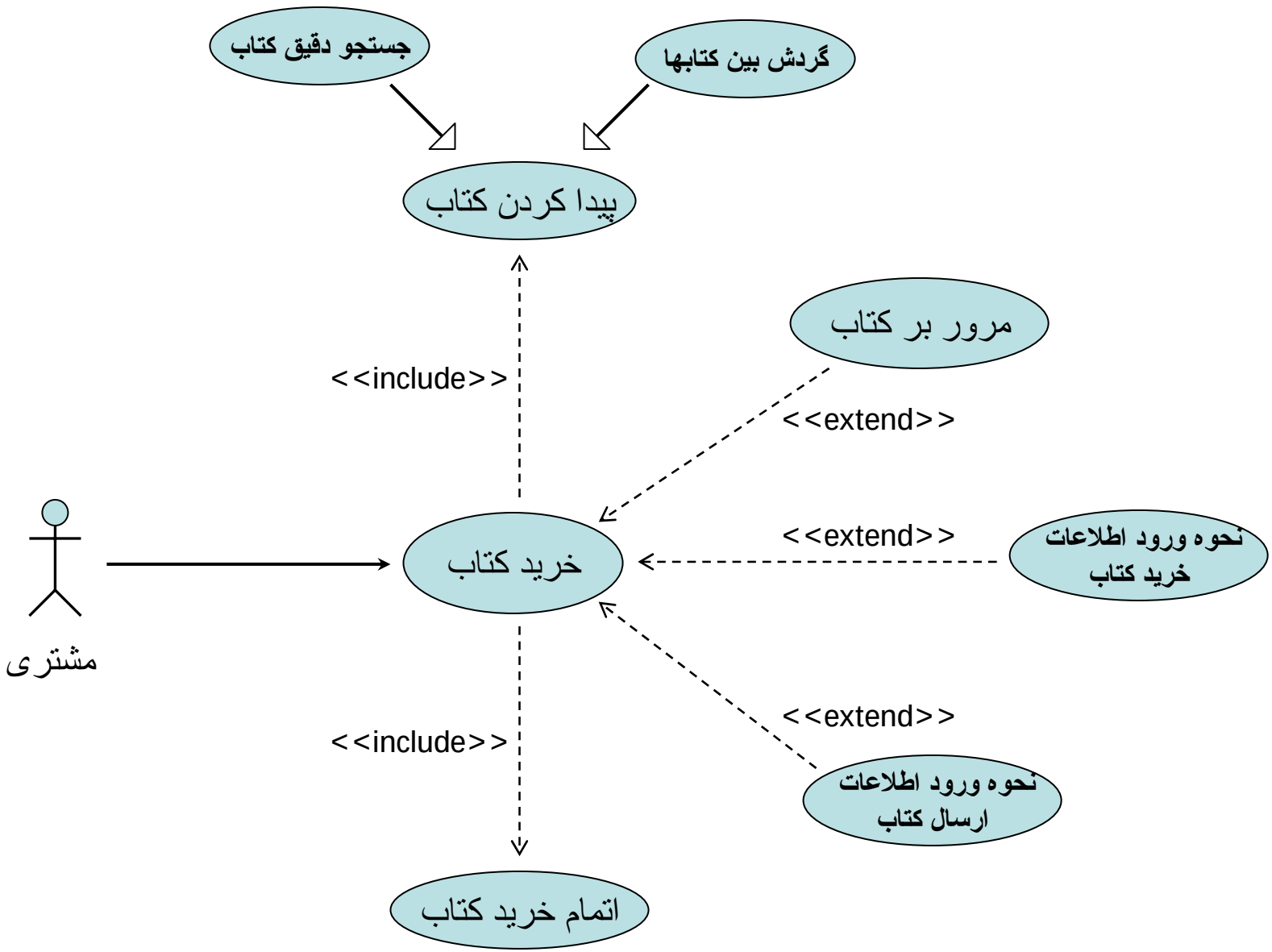


تعمیم use case

در بعضی موارد ممکن است که در یک کاربرد چندین مسیر جداگانه داشته باشیم . مثلا ممکن است برای پیدا کردن کتاب دو راه وجود داشته باشد . مشتری که درک روشنی از کتاب دارد و مشتری که در رابطه با کاتب مورد نظر خود اطلاعات چندان ندارد .

در این قبیل موارد از رابطه تعمیم استفاده می کنیم البته بیاد داشته باشید که صرفا براساس میزان پیچیدگی یک کاربرد ، به تعمیم آن مبادرت نورزید .

کاربرد خرید کتاب به همراه تعمیم



روشهای مدل کردن use case

برای مدل کردن use case می توانیم آن را در سطوح مختلف اعمال نمود ، داریم :

❶مدل کردن یک سیستم
❷مدل کردن یک subSystem
❸مدل کردن یک رفتار سیستم

مزایای استفاده از use case

①مدل کردن رفتار یک عنصر با استفاده از دید کاربرد باعث انتقال جزئیات و مفاهیم سیستم به کسانی که بیرون از سیستم هستند .
②با این وسیله می توان آن عناصر را سند کرد و برای کاربران آن مولفه
③تست کردن

مزایا در حالت کلی :

- راحتتر ساخته می شود
- راحتتر مستند می شود
- راحتتر تست می شود .

مدل کردن رفتار یک عنصر (قدمها)

1. مشخص کردن Actor های موجود در سیستم

2. عامل های پیشنهادی شامل آن مجموعه ای می شوند که نیازمند رفتار خاصی از آن عنصر هستند برای انجام کارهایشان یا آن عامل هایی که به صورت مستقیم یا غیر مستقیم از کارکردهای سیستم استفاده می کنند .

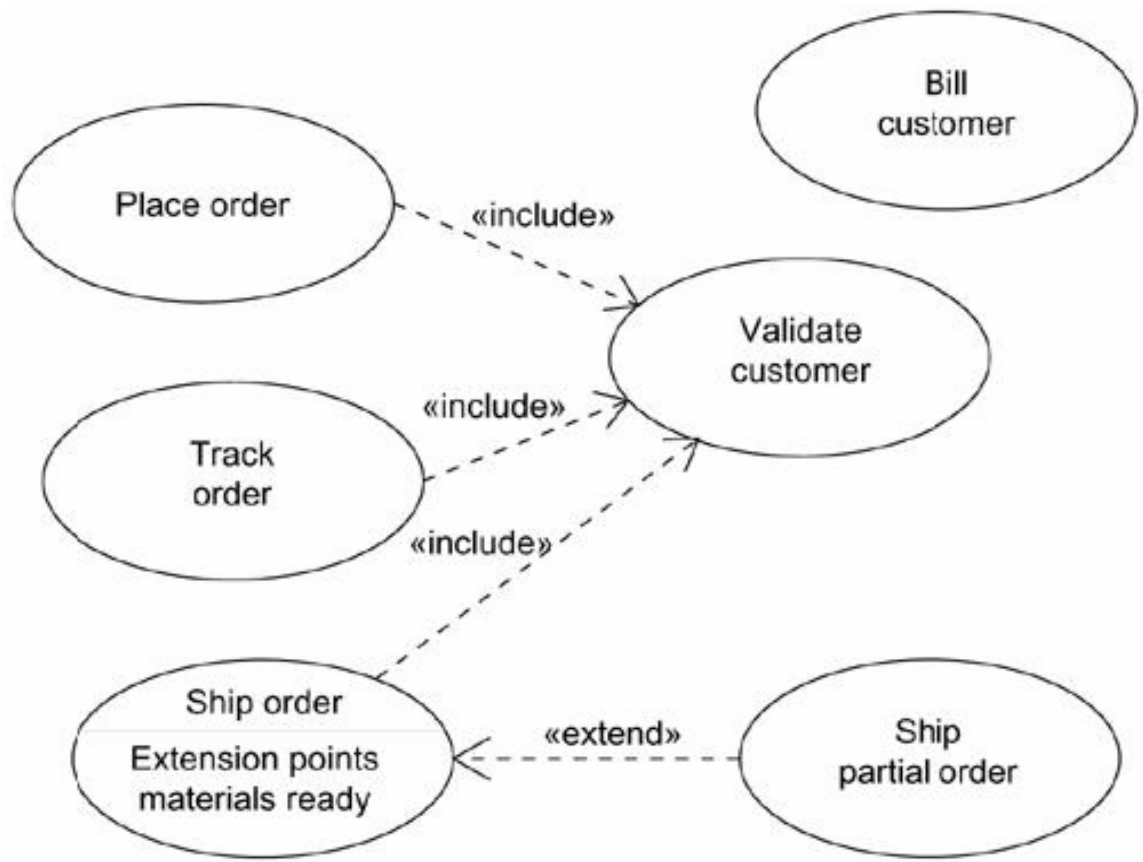
3. سازمان دهی Actor ها در یک ساختار سلسله مراتبی (تعمیم)

4. برای هر عامل سیستم آن و راه های اساسی که Actor با سیستم در ارتباط و تعامل است مشخص می کنیم .
   باید به تعاملاتی که موجب تغییر در عنصر می شود دقت خاصی شود .

5. حالات استثنا هر Actor با سیستم تعامل می کند مشخص شود .

6. دسته بندی یا به عبارتی سازماندهی use case ها توسط مکانیزم های ذکر شده .

Modeling the Behavior of an Element



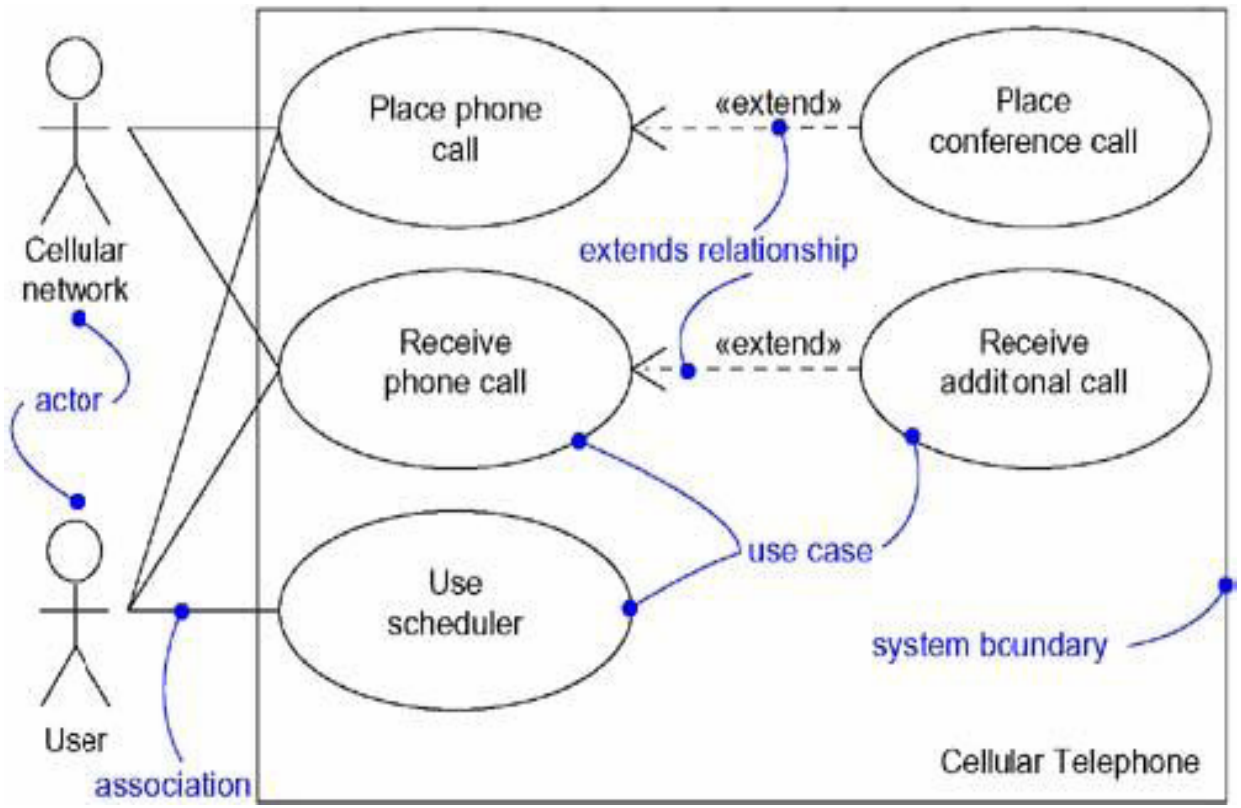
هر چه تا کنون خواندیم در رابطه با

Use case

بود و اما

Use case Diagram

A Use Case Diagram : مثال



Use case Diagram

نموداری است برای نمایش مجموعه ای از کاربرد ها و عامل ها و روابط بین آنها

انتظارات از use case Diagram

1. مرزبندی سیستم
2. مدل کردن نیازمندیهای یک سیستم

در use case view دو بحث را مطرح مي كنيم:

1. use case diagram
2. activity diagram

در use case دیاگرام سه قسمت وجود دارد که شامل:

1. actorها
2. use caseها
3. روابط بین آن ها

در اين قسمت دو چيز قابل اضافه كردن است:

❶مرز بندي سيستم
❷ packageبراي دسته بندي use case مي توان به ان اضافه كرد.

براي مدل كردن (زمينه) Context سيستم:

① actorهاي پيرامون سيستم را بايد مشخص كنيم.گروههايي را بايد در نظر بگيريم كه به اين سيستم نياز دارند.

②چه گروههايي نياز داردن كه كاركردهاي سيستم را اجرا كنند.

③چه كساني با سخت افزار و نرم افزار ها ارتباط داردند

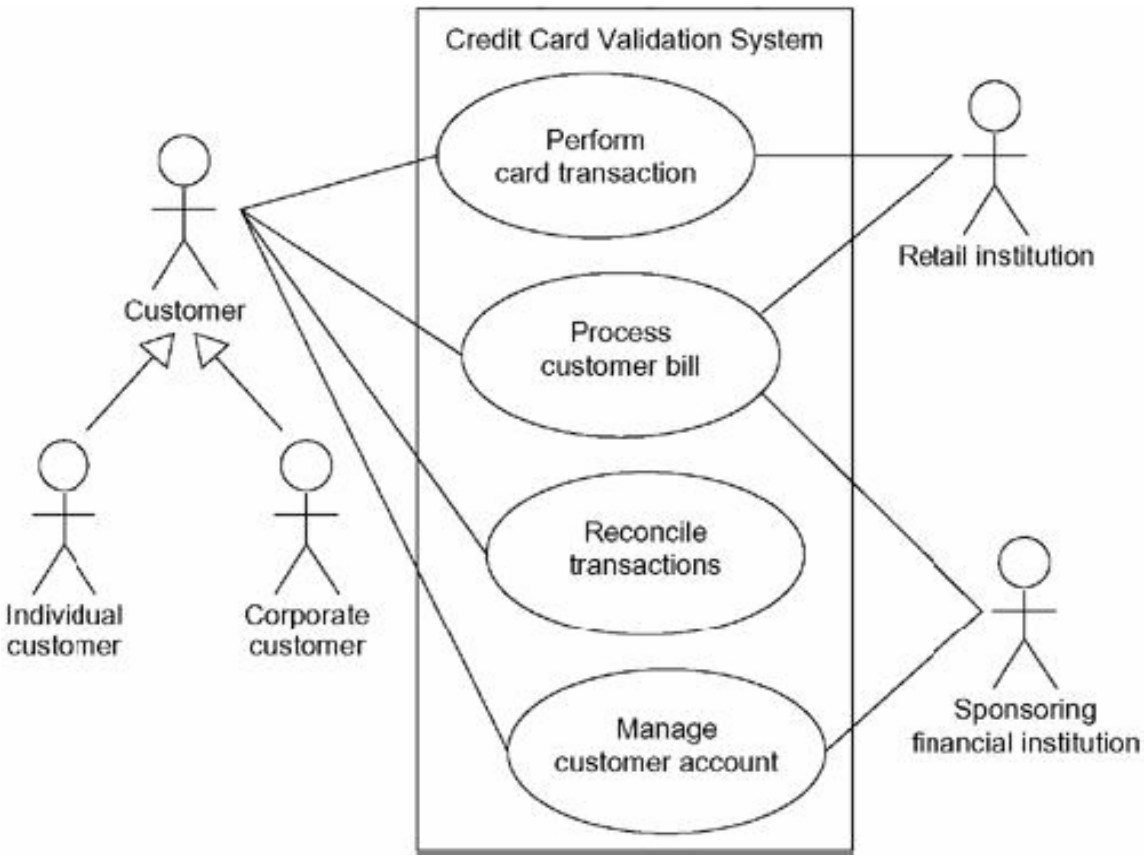
④چه كساني مديريت را روي سيستم ها انجام خواهند داد.

⑤ acotrهاي سيستم را دسته بندي و سازماندهي كرده.

⑥اگر به فهم مطالب كمك مي كند از streotype استفاده كنيم.

⑦ use caseهاي كلي كه actor ةا از ان ها استفاده مي كنند را اعمال كنيم .

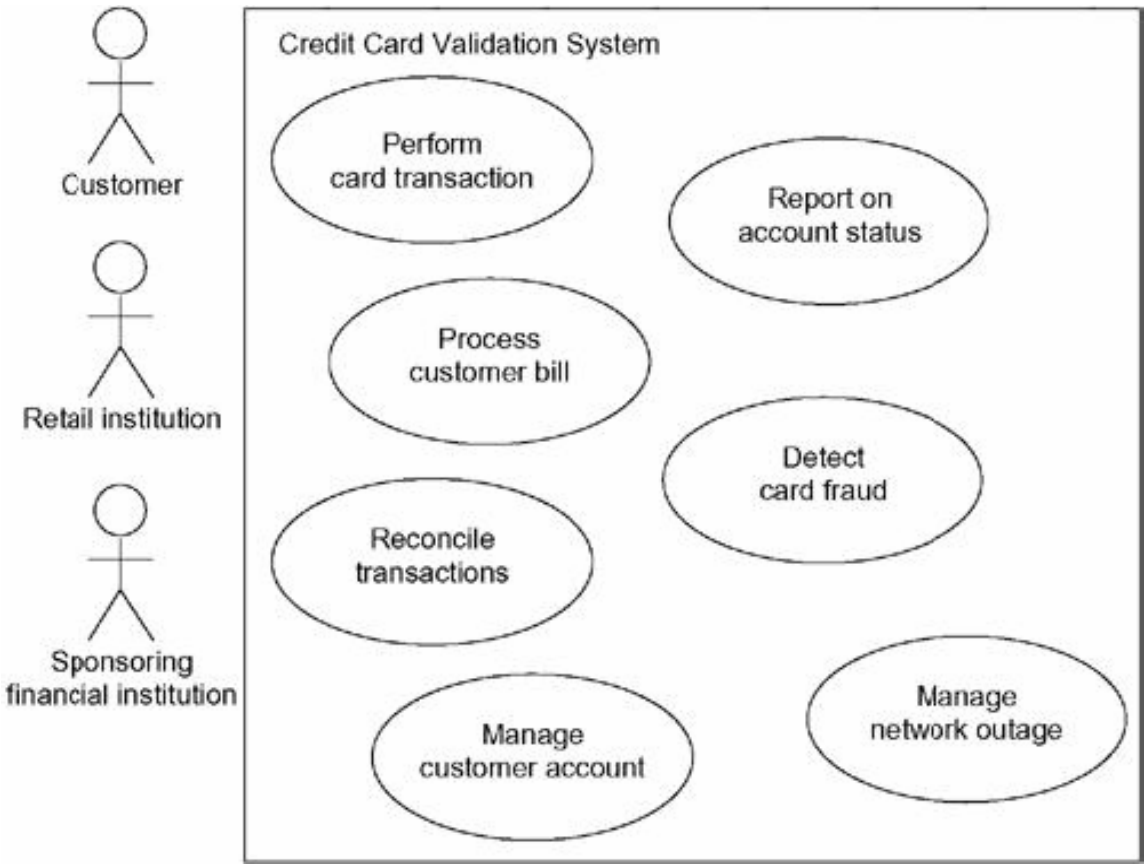
Modeling the Context of a System



براي مدل كردن نيازمندي ها:

1. contextسيستم يا الگوريتم سيستم را درآوريم.
2. براي هر actor رفتارهايي را كه انتظار داريم سيستم براي ان انجام دهد را مشخص كنيم.
3. اين رفتار هاي مشترك را تحت نام use case دسته بندي كنيم.
4. با استفاده از سه رابطه Extend و Generalization و Include ان use case را سازماندهي مي كنيم.
5. در صورت لزوم براي هر كدام از use case ها جريان رويدادها را مشخص مي كنيم.

Modeling the Requirements of a System



مهندسي مستقيم و معكوس در مورد: use case

در این دیاگرام نمی توان این مهندسی پیشرو و معکوس استفاده کرد چون یک دیاگرام مفهومی است و پس به ترتیب زیر عمل می کنیم تا بتوانیم این دو مهندسی را اعمال کنیم .

مهندسی پیشرو توسط ابزار امکان پذیر نیست .

قدم هاي لازم در اين قسمت شامل:

1. از use case مي توان Test case ان را درست كرد.
2. براي هر use case در سيستم جريان اجرايي اصلي را در نظر مي گيريم.
3. براي هر flow يا جريان يك سناريو Test بنويسيد.

Reverse Engineering Use case:

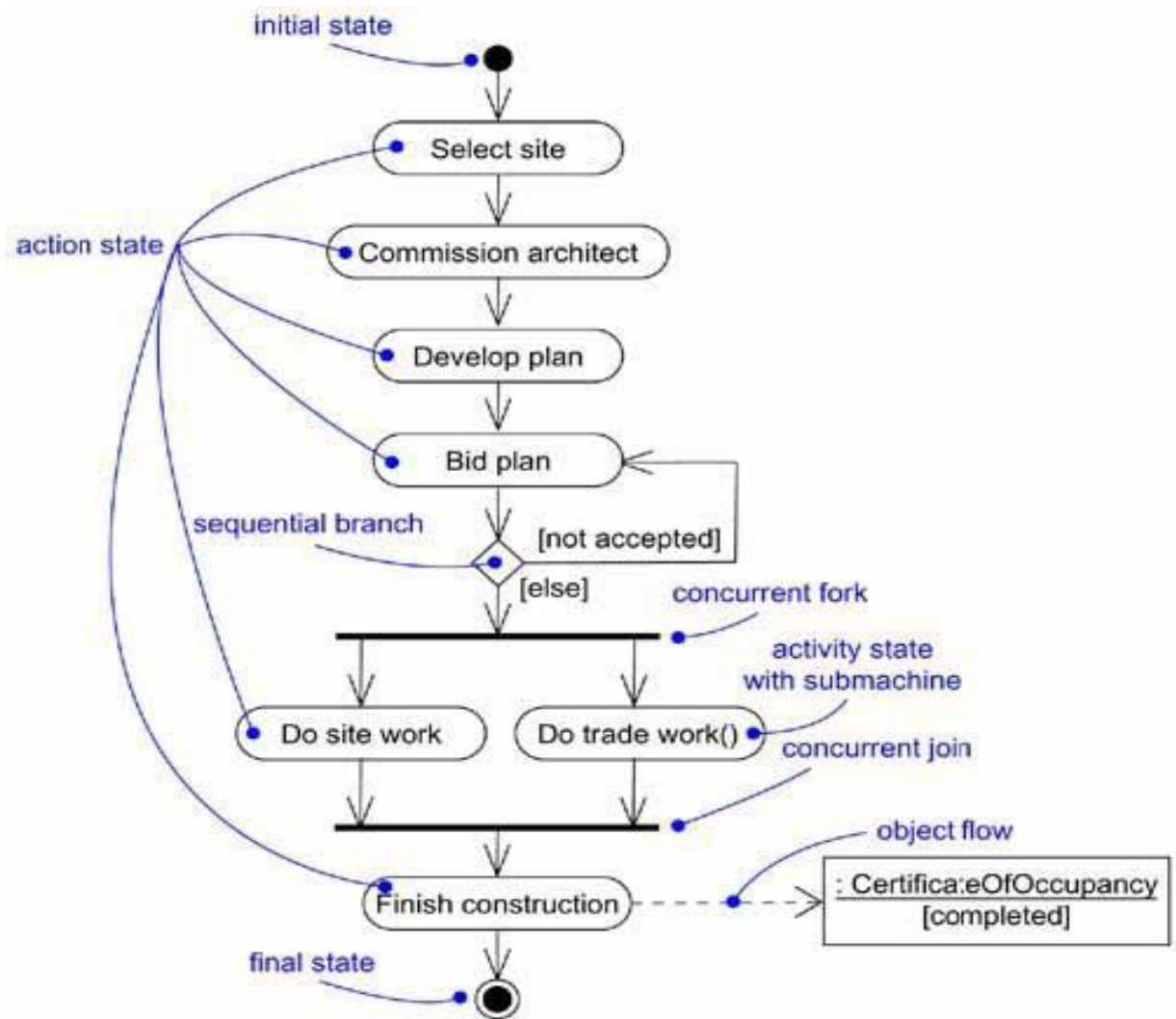
در اين قسمت source را به ما مي دهند و use case مربوطه را بايد تحويل دهيم:

1. مشخص كردن actor هايي كه با سيستم ارتباط دارند.
2. براي هر actor طريقه تعامل ان را با سيستم مشخص مي كنيم البته از روي كد.
3. در صورت امكان از اين تعاملات مي توان use case را به دست اورد

فصل یازدهم

Activity diagram

Activity Diagrams

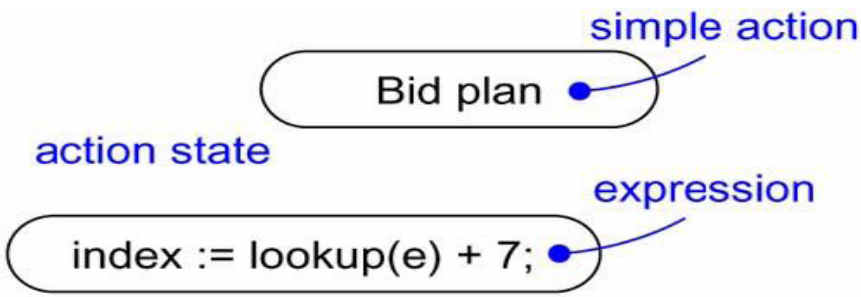


اين دياگرام نمايش دهنده جريان فعاليت ها از يك فعاليت به يك فعاليت ديگري است.
يك فعاليت در حالت انجام غير اتميك است.

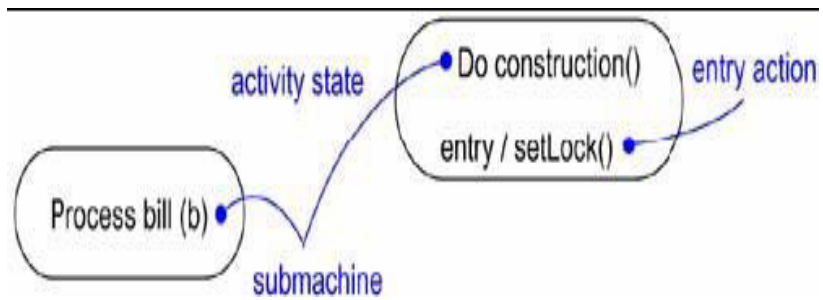
فرق activity با action در چيست؟

در activity يك كاري داريم كه قابل تجزيه است يعني كارهايمان مشخص است.
اما در action يك كار غير قابل تجزيه است.بعضي از عمليات ها قابل تجزيه نيستند يعني ان قدر كوچكند كه ديگر تجزيه نمي شوند.

Action States



Activity States



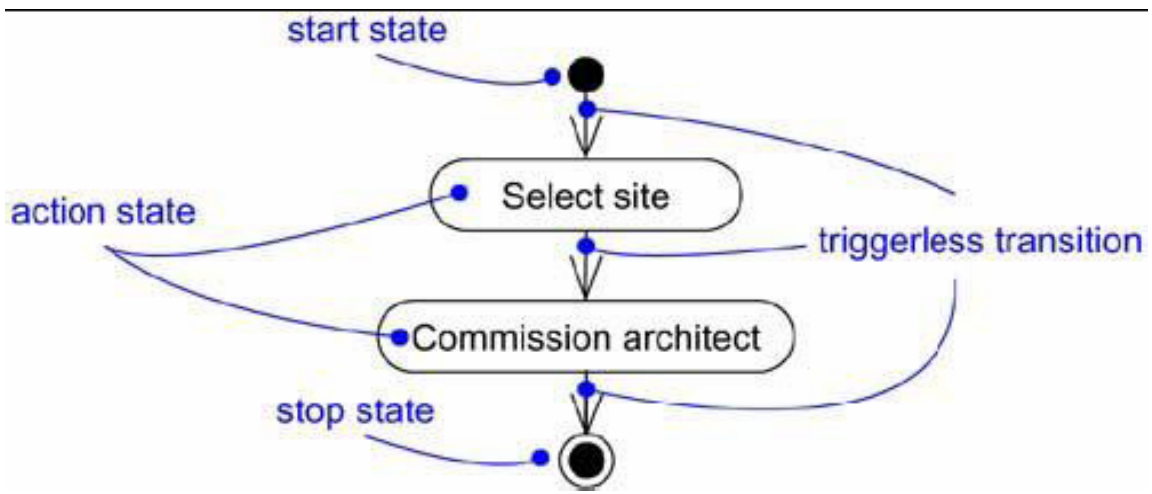
سه قطعه اصلي در **activity diagram** عبارتند از:

❶ activity state and action state
❷ Transition
❸ Object

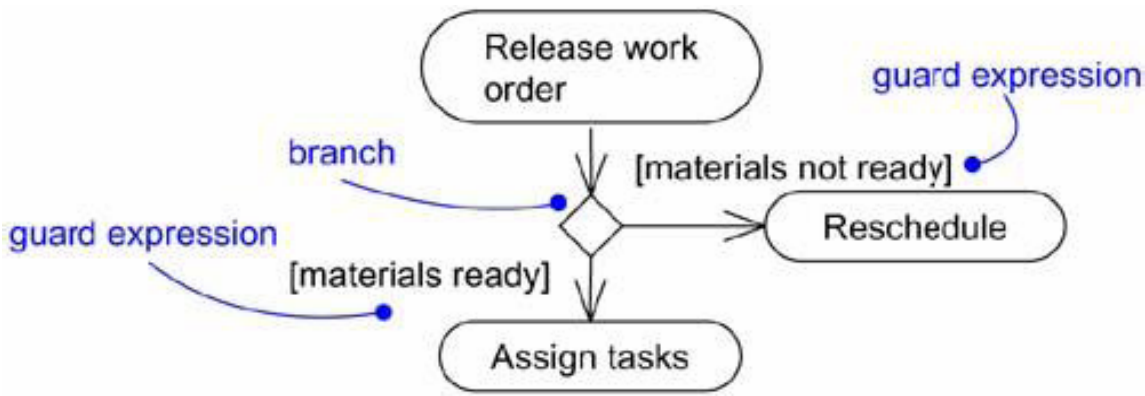
: Transition

مشخص كننده پايان يك Activity و شروع يك Action است.
مشخص مي كند يك Activity يا يك Action تمام شده و چه Activity و يا Action هايي بايد شروع شود.

Triggerless Transitions



Branching


Release work order
guard expression
branch
[materials not ready]
Reschedule
guard expression
[materials ready]
Assign tasks

در عمليات join اين قضيه را داريم كه در ان چند انشعاب و شاخه دريك جا به هم متصل مي شوند و يك شاخه را ايجاد مي كند كه اين بحث fork & join را مي سازند.

شي اي اجازه fork & join را دارد كه:

❶استفاده از Therad

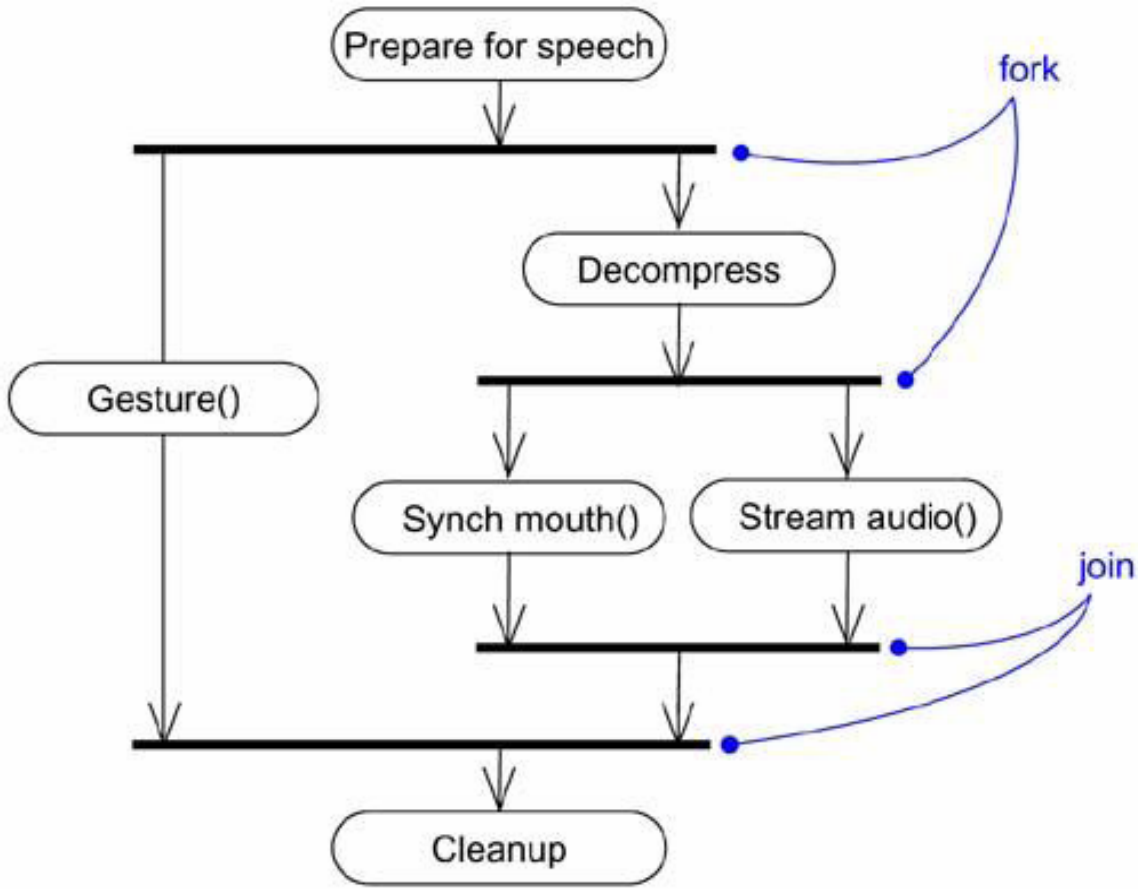
❷ fork & joinبايد balance يا تعادل داشته باشد.يعني اگر چهار تا join داشتيم بايد چهار تا هم fork داشته باشيم.

اگر fork & join با هم برابر نباشد يك شاخه اي ازاد باقيمانده كه اين از ايراد كار ماست.

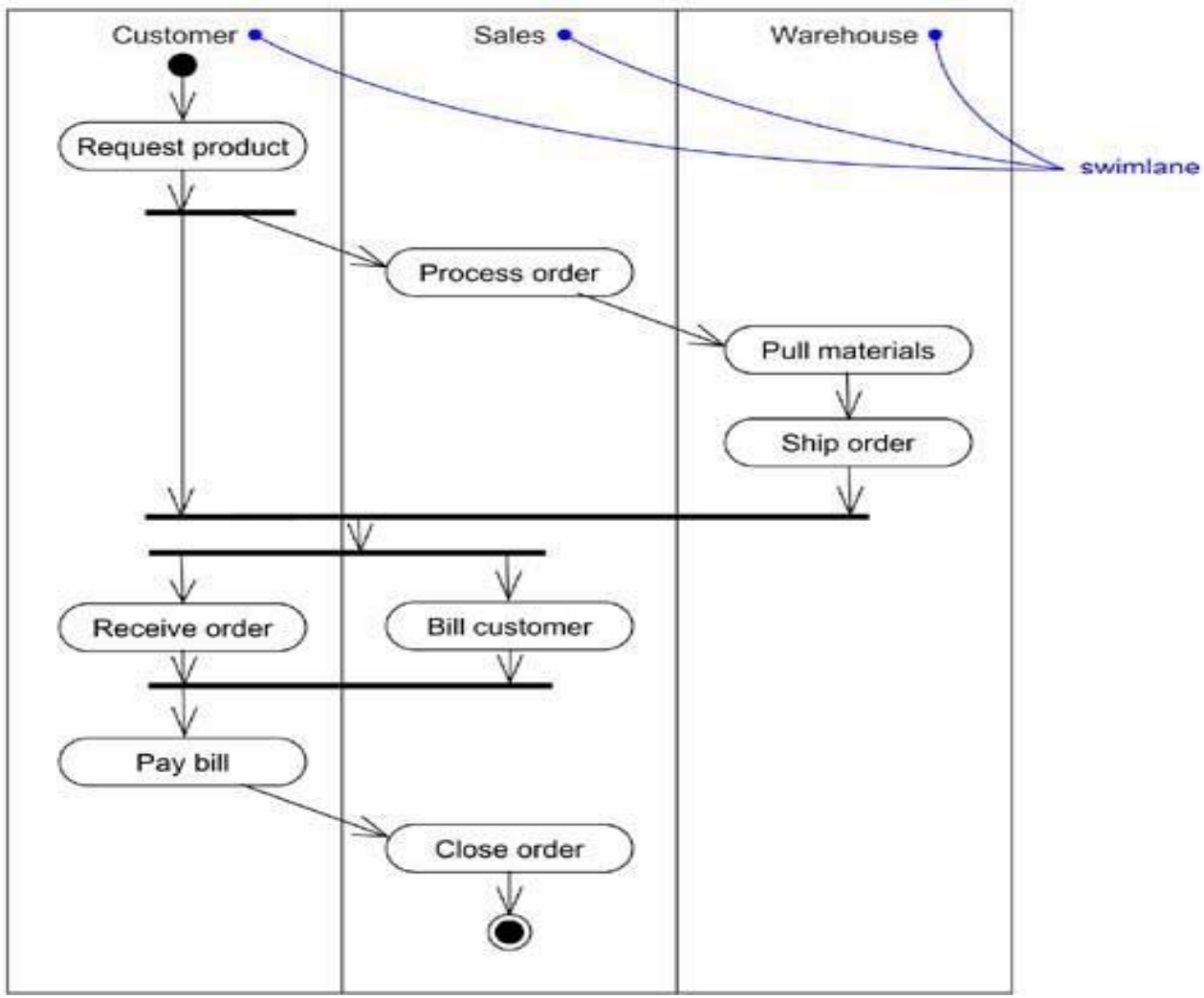
نكته:از swimlane براي گروه بندي activity ها در يك ACTIVITY DIAGRAMاستفاده مي شود.

از bissnes logic براي تعيين قواعد يا قوانين حاكم در ان سازمان يا مجموعه مورد استفاده قرار مي گيرد.
bissnes objectاشيايي است كه در ان منطق سيستم و سازمان نقش دارد.

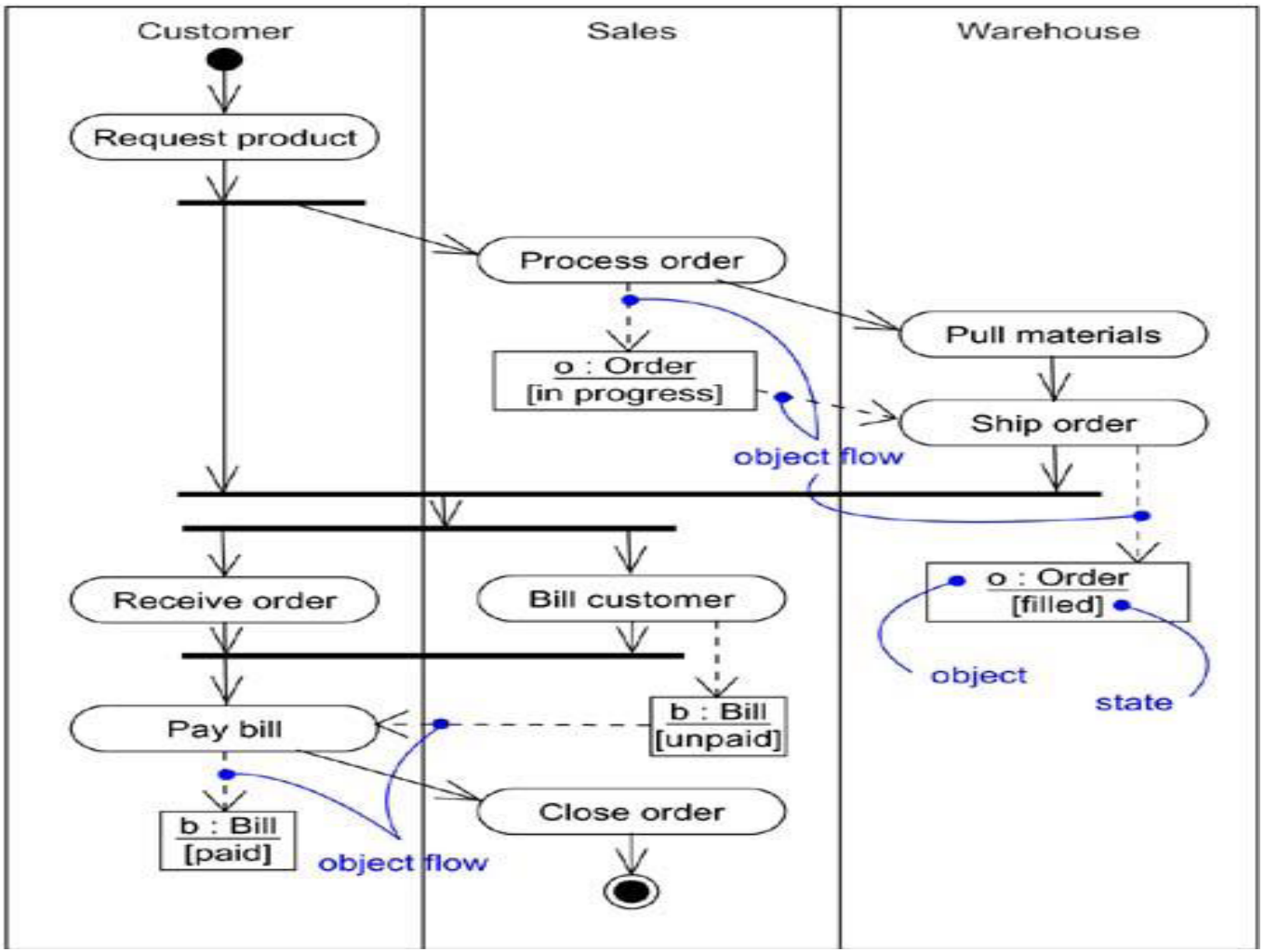
Forking and Joining



Swimlanes


Customer
Sales
Warehouse
swimlane
Request product
Process order
Pull materials
Ship order
Receive order
Bill customer
Pay bill
Close order

Object Flow


Customer
Sales
Warehouse
Request product
Process order
o : Order
[in progress]
Pull materials
Ship order
object flow
Receive order
Bill customer
o : Order
[filled]
object
state
Pay bill
b : Bill
[unpaid]
Close order
b : Bill
[paid]
object flow

براي مدل كردن يك جريان كاري چه اعمالي را بايد انجام داد:

1. مشخص كنيم Work flow چه چيز مي باشد.بهترين انتخاب ان است كه براي هر use case يك activityدياگرام استفاده كرد.

2. ان bissnes object هايي كه در بالاترين سطح در اين جريان كاري دخالت دارند و مسيوليت دارند را مشخص كنيم.براي هر كدام از اشيا يك swimlane رسم كنيم.

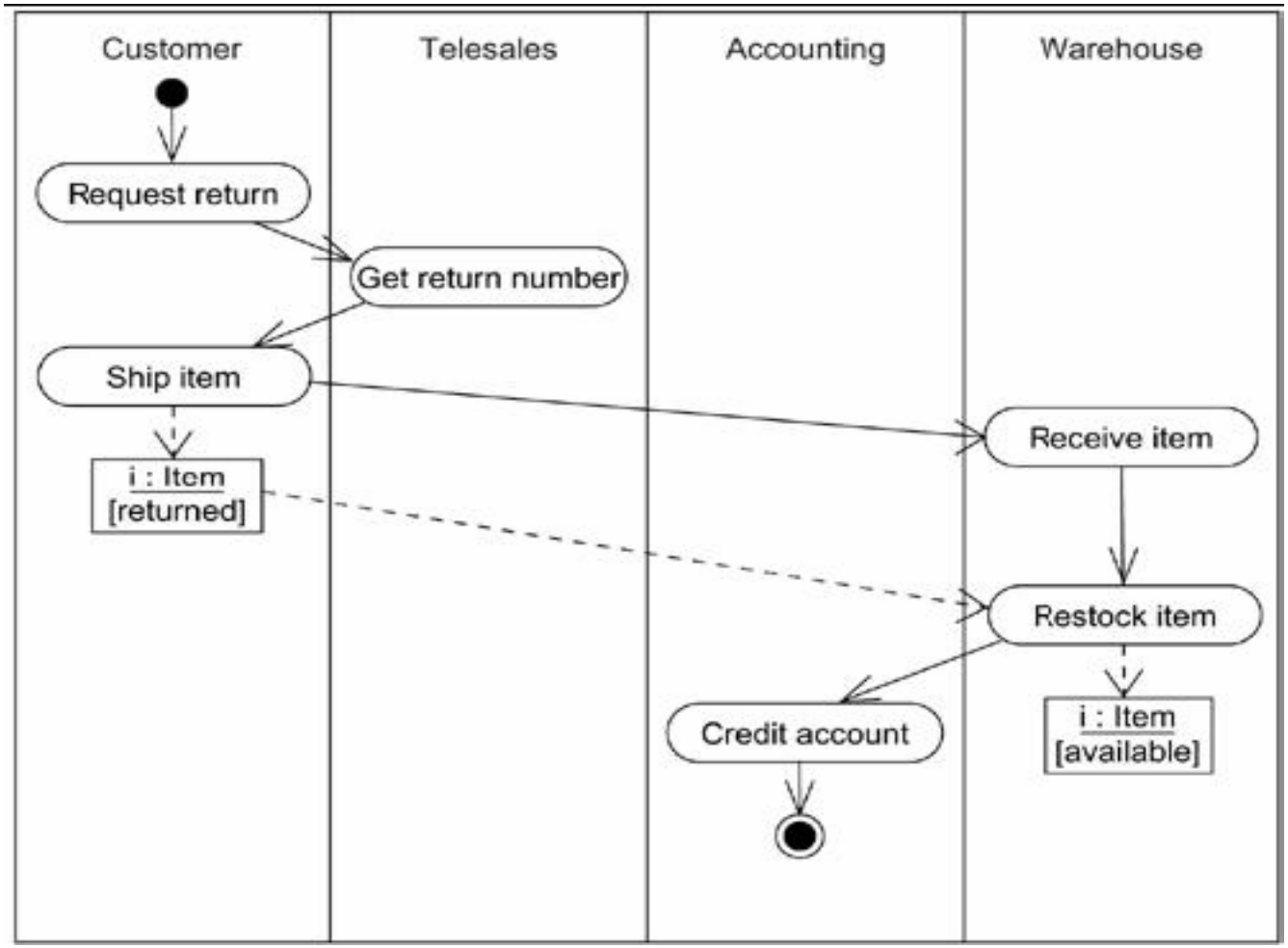
3. بايد مشخص كنيم قبل از اينكه اين كار انجام شود ايا مقدماتي لازم است يا نه؟ اگر لازم باشد انها را مشخص كنيم و وضعيت سيستم را در پايان اين كارمان مشخص كنيم.

4. با يك دايره توپر ان را نشان داده و شروع مي كنيم و activityها و يا action هايي كه رخ مي دهد را به شكل مان اضاف م يكنيم.اگر مجموعه اي از عمليات پيچيده است ان مجموعه را با يك activity diagramواضح تر مي كنيم.

5. خطوط انتقال را وصل مي كنيم.ابتدا خطوط ساده و بعد خطوط شرطي و بعد خطوط موازي را مي كشيم.

6. اگر اشيايي وجود دارد كه با ارسال ان ها در مراحلمختلف ما مي توانيم خوانايي برنامه را بالا ببريم ان اشيا را به سيستم اضافه مي كنيم.

Modeling a Workflow



براي مدل كردن Operation ها:

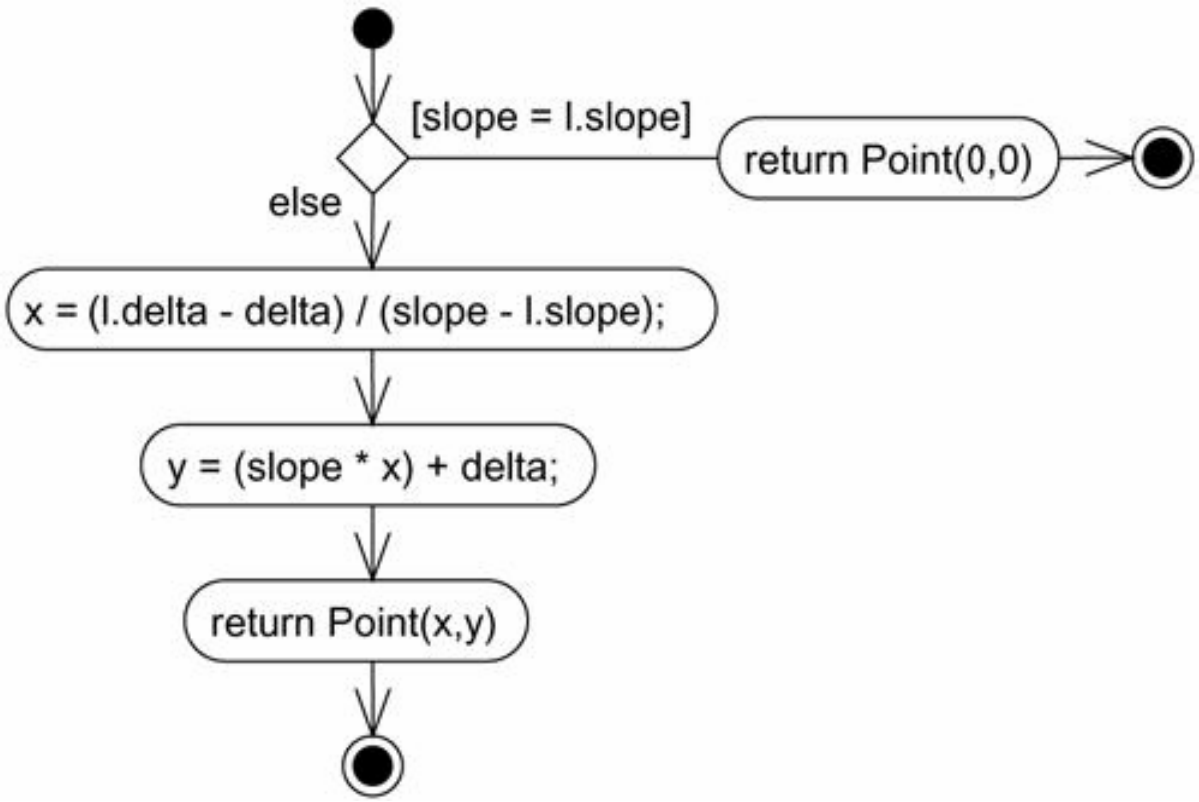
①خود operation را تعريف مي كنيم.

②پيش شرط ها و وضعيت هاي نهايي و نيز وضعيتي كه برايش اتفاق مي افتد را مشخص مي كنيم.

③با شك دايره توپر شروع مي كنيم و در ابتدا عملياتي كه در ان Opeartion انجام مي شود را مشخص مي كنيم.در اين مرحله شرط ها و حلقه هاي تكرار را اضاف مي كنيم.

④نهايتا اگر operation مربوط به تگ Object يا Active object است بايد از ان استفاده كنيم.

Modeling an Operation



فصل دوازدهم

رویداد ها و سیگنال ها

Events and Signals

رویداد ها : آن چیزهای که اتفاق می افتند به آنها رویداد گویند و هر کدام از این رویداد ، رویداد یا جریانی هستند که می توان از نظر زمان ومکان مشخص کرد .

☑ در State machine ما از رویداد ها استفاده می کنیم .

☑ این رویدادهایی که در سیستم مدل می کنیم محرک های هستند که باعث انتقال در سیستم می شوند در واقع یک شی می تواند توسط این رویدادها از یک حالت به حالت دیگر برود.

چیزهای که یک رویداد شامل آنها می تواند باشد :

۱- سیگنال
۲- فراخوانی call
۳- گذشت زمان passing a time
۴- تغییر در state

انواع رويداد ها عبارتند از :

1. رويدادهايي كه بين Actor و سيستم مي باشد كه آن ها را External مي ناميم.
2. رويدادهايي كه بين اجزاي دروني سيستم مي باشد كه آن ها را Internal مي ناميم.

سيگنال: نوع خاصي از رويداد است كه به صورت غير همزمان كار مي كند.يعني يك محرك را مي فرستد و منتظر جواب نمي ماند و كار خود را ادامه مي دهد. يك سيگنال يك شي نامدار است كه در سيستم توسط يك شي به صورت غيرهمزمان رها مي شود و يك شي ديگر ان را دريافت مي كند.

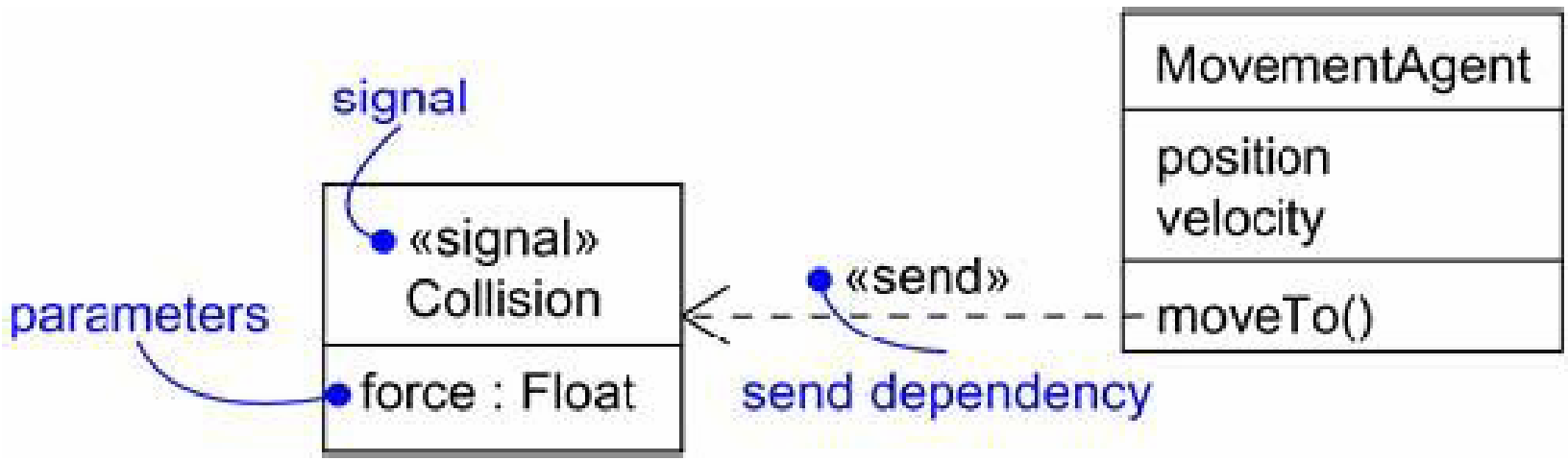
☑ براي نمايش دادن سيگنال از رابطه dependency استفاده مي كنيم.

مثال : فرستادن Exception ها

☑ خود سيگنال داراى Attribute و operation ميباشد .

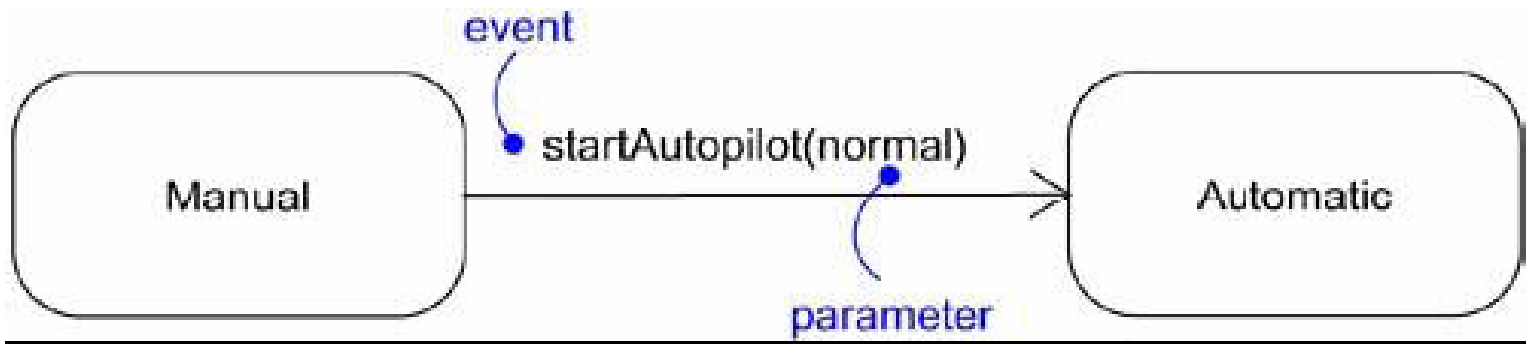
☑ برای نمایش دادن سیگنال از رابطه وابستگی و **Stereotype « send »** استفاده می کنیم .

مثال :



فرا خواني (call) : يعني call event با صدا زدن يك متد و تمام شدن ان متد از يك وضعيت به وضعيت ديگري مي رود همچنين نوعی رویداد بوده که حالت همزمانی دارد .

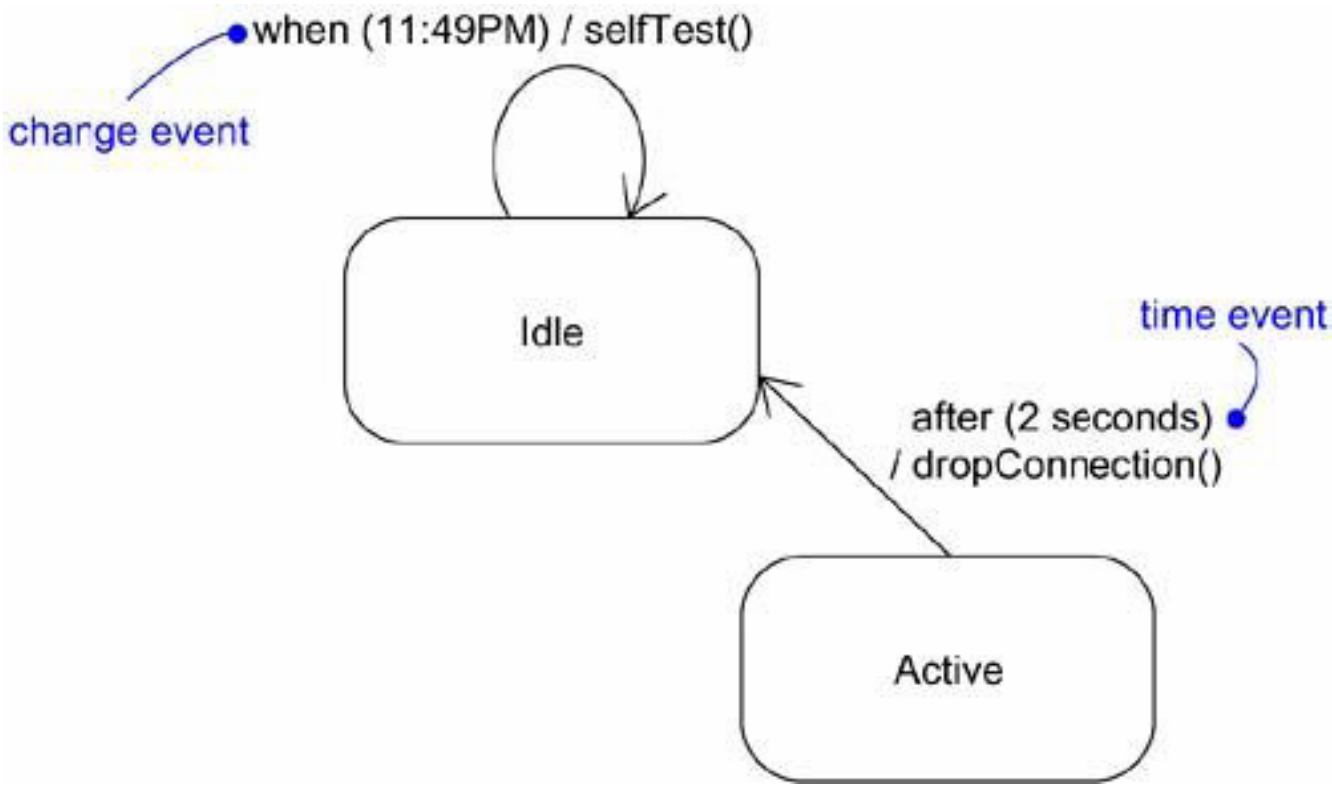
☑مانند مثال زیر که در بازی با استفاده از یک دکمه ماشین از حالت دستی به اتوماتیک می بریم .



گذشت زمان Time and Change Events :

با كلمه after مي باشد . روي هر انتقال دو اطلاع مهم مي توانيم داشته باشيم يكي زمان و ديگري اتفاقي كه مي افتد .

مثال : با فرض كانكت بودن مودم :



☑ روی یک انتقال ما دو اطلاع مهم داریم که با اسلش (/) جدا می شود .

۱- زمان مورد که باید بگذرد .
۲- Action و کارهای که بعد از این زمان باید انجام دهد .

تغییر وضعیت : با کلمه when مي اید و بعد از اسلش قرار مي گیرد.

☑ روی یک انتقال ما دو اطلاع مانند فوق در تغییر زمان داریم :
۱- ابتدا پس از when یک شرط و سپس بعد از اسلش (/)
۲- کار های که باید انجام شود .

✓ **نكته :** در در سيگنال و فراخواني دو object داريم ولي در تغيير زمان و تغييروضعيت ممكن است يك object داشته باشيم .

تفاوت سيگنال و فراخوانی (تفاوت Signal & Call)

در يك فراخواني وقتيمتدي را صدا مي زنيم صدا زننده متوقف مي شود تا عملا كار متد صدا زننده شده تمام شود كه به اين كار سنكرون مي گوييم .

در سيگنال براي Run كردن يك thread متد Start ان صدا زده مي شود . يعني به طورغير همزمان است . پس تفاوت Call و Signal در اين است كه Call همزمان يا سنكرون است ولي Signal غير همزمان مي باشد .

مانند : تلفن و پست

براي مدل كردن Exception ها تنها دو كار را انجام مي دهيم:

❶براي هر كلاس و اينترفيس تمام متد هاي ان exception هارا در مي اوريم.

❷Exceptionها را به ترتيب از super به sub مرتب مي كنيم .

❸برای هر Operation باید مشخص کنیم چه استثنایی بوجود می آید .

دليل مدل كردن رويداد ها:

در state machine رويدادهايي كه مدل مي كنيم يك محرك هايي هستند كه موجب تغيير وضعيت مي شوند در ماشين وضعيت.

فصل سیزدهم

ماشین وضعیت

State machine

ماشين وضعيت چيست؟

يك ماشين وضعيت مدل كننده چرخه زندگي يك شي مي باشد.اين شي مي تواند نمونه اي از يك كلاس يك use case يا حتي كل سيستم باشد .

مثال : یک Thread یک نمونه ماشین وضعیت است .

✓ **نکته :** با رخ دادن یک Event ما می توانیم در ماشین وضعیت حرکت کنیم

مثال : سیستم تنظیم دما

✓ **نکته :** Activity دیاگرام نوع خاصی از stateMachine است که فقط یک رویداد دارد .

تفاوت بین State machine و Activity diagram ؟

در state machine چيزي كه براي ما مهم است حالت event در event ي است كه وجود دارد.انجام فعاليت ها و عملياتي كه رخ مي دهد به چه صورت است .اما در Activity diagram نوع خاصي از state machine كه تنها event ي كه در دروني است وجود دارد پايان يك عمليات است.

تعریف StateMachine :

یك state machine یك رفتار است كه مشخص كننده دنباله اي از حالت هاست كه یك شي درزندگي خودش خواهد رفت در واكنش به یك سري رویداد ها به اضافه پاسخي كه به ان رویداد ها خواهد داد.

تعریف State :

یک حالت یا وضعیت یا حالتی است که یک شی در حیات خود خواهد داشت و این حالت ها مطابق با یکسری شرط ها و معیارها خواهد بود یا اینکه انجام یک عملیات خاص خواهد بود یا انتظار برای یک رویداد خاص خواهد بود .

☑ يك state حالتي است كه يك شي در حيات خودش خواهد داشت كه :

❶ اين حالت ها مطابق با يك سري شرط ها و معيار ها خواهد بود.
❷ يا اينكه انجام يك عمليات خاصي خواهد بود.
❸ يا اينكه انتظار براي يك رويداد خاص خواهد بود.

✓ **نكته :** يك Object براي يك مقدار محدودي در يك state قرار مي گيرد.يعني يك Objectزماني كه در يك state به سر مي برد بايد محدود باشد.

✓ **نكته :** يک شی در يک state بايد در زمان محدود بسر ببرد .

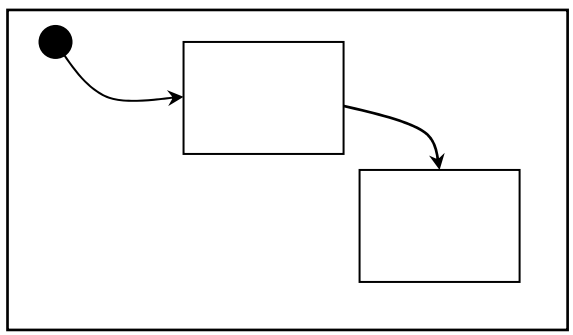
☑ **قسمت های State :**

❶ نام : name

❷ یک Action ورود / یک Action خروج (Entry and Exit Actions)

❸ Internal Transition آن انتقال های که می تواند انجام شود درون State بدون تغییر State

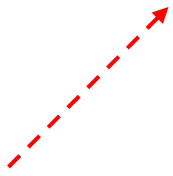
❹ sub state (زیر وضعیت) : یک وضعیت می تواند یک زیر وضعیت داشته باشد مانند :



❺ Deferred events : ليست رويداد هايي است كه ان state ان ها را پس مي دهد.

Transitionيا انتقال : (خطوط ارتباطی State ها)

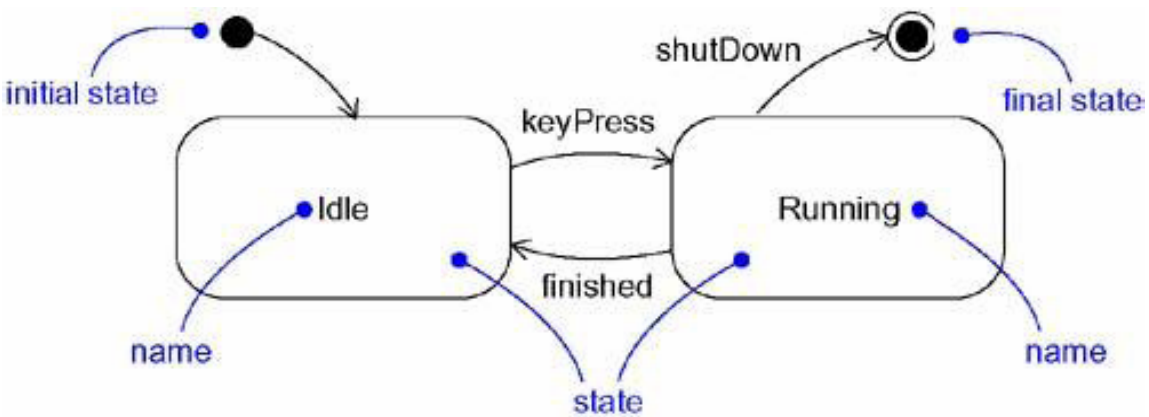
يك رابطه اي است ميان دو حالت يا وضعيت كه مشخص مي كند كه شي در وضعيت اول با رخ دادن يك رويداد با انجام يك پاسخ وارد وضعيت دوم مي شود .



Transition

Internal Transition : انتقال داخلی : وضعيت آن تغيير نمی كند ولی وضعيت

مثال :



☑ براي نام گذاری State ها :

۱- اسمي كه براي يك state انتخاب مي شود يا در لغت نامه سيستم بايد تعريف شده باشد يا اينكه خودمان ان را به لغت نامه اضافه كنيم .
۲- اسم گذاري در اين قسمت مانند اسم گذاري در Class ها مي باشد.يعني حروف اول را بايد با حروف بزرگ نوشته شود.

☑در يك transition يا انتقال مي تواند پنج جز قرار بگيرد كه ان ها عبارتند از :

❶ state شروع كننده : شروع اش از كجاست.
❷ event يا ان رويدادي كه موجب اين انتقال مي شود.
❸ state ي كه به ان منتقل مي شويم.
❹پاسخ و عملي كه همزمان به ان انتقال انجام مي شود كه غير قابل تجزيه مي باشد.
❺شرط محافظ: شرطي كه روي رويداد قرار مي گيرد.يعني انتقال وقتي اتفاق مي افتد كه شرط رويداد بر قرار باشد.

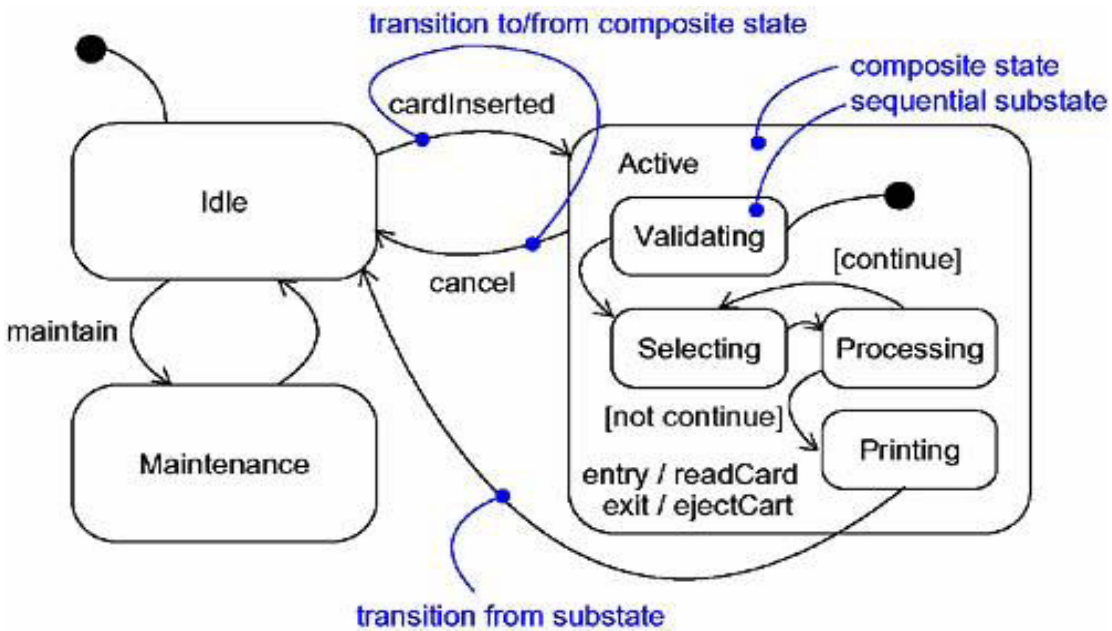
✓ فرق self transition با internel transition ؟

در self transition دو عمليات ورود و خروج state انجام مي شود اما در انتقال دروني ان دو عمليات مربوط به ورود و خروج از يك state انجام نمي شود.

✓ **نكته :** يك sub state زماني اتفاق مي افتد كه يك سيستم دريك وضعيت باشد حال ان كه در همان وضعيت مي تواند يك state ديگري را انجام دهد .

Sub State زير وضعيت : در يک سيستم ممکن است يک State زير حالت های مختلفی در همان حالت به خود بگيرد .

مثال : يک زير وضعيت



دو نوع زیر وضعیت داریم :

sequntial ترتیبی

concurrent موازی

زیر حالت ترتیبی sequntial نقطه مقابل زیر حالت موازی concurrent است .

✓ **نکته :** در زیر حالتی که در آن زیر حالت و وضعیت آن موازی بود باید همه زیر حالت در حالت نهایی ◉ و اتمام قرار گیرند تا از این حالت خارج شویم .

History States (سابقه ی حالت) :

فرضا ممکن است در انجام کاری در انجام یک حالت فرضا نسخه ی پشتیبان گرفتن ما سه مرحله یا حالت داریم ممکن است قسمتی از کار انجام داده باشیم ولی ناگهان برق برود و دیگر از صفحه شروع نمی کنیم بلکه از H کارهای که انجام شده

☑ **زير وضعيت موازی يا همان Concurrent sub stateمانند عمليات Fork& join در Java می باشد :**

۱- در يك state كه از چند Sub state به شكل موازي تشكيل شده زماني از ان وضعيت خارج مي شود كه همهsub state موازي خاتمه يافته باشند و اگر يكي از ان ها تمام نشده باشد ان state خارج نمي شود .

۲- در مكانيزم Sequence به ترتيب پس از اجراي همه State كه به صورت مرتب است
از ان state خارج مي شويم .

۳- در مكانيزم Backing up كه با استفاده از History مي باشد.اين history مشخص مي كند كدام sub state بايد انجام شود.يعني بر حسب اينكه قبلا كدام sub state انجام شده مشخص مي كند به كدام sub state برود .

☑ مراحل مدل سازی يك State machine :

1. بايد مشخص كنيم كه يك كلاس يا use case يا سيستم.اگركلاس است بايد به جمع اوري اطلاعات راجع به كلاس ها وكلاس هاي مجاور اطلاعات جمع كنيم.در مورد سيستم بزرگ بايد سيستم را از ديد يك رفتار خاص مدل كنيم .

2. Initial state و Final state را رسم كنيم .

3. رويدادهايي را مشخص كنيم كه Object بايد پاسخ دهد.اگراز قبل مشخص شده باشد اين رويداد ها را مي توان در Interface كه صاحب ان Object مي باشد مشخص كنيم . اگر مشخص نشده باشد بايد به دنبال تكامل ان شي با ديگر اشيا گشت .

4. از state اول شروع به رسم كردن وضعيت هاي سطح بالايي كه شي مي تواند داشته باشد مي كنيم و همزمان ان ها را مشخص مي كنيم . در مرحله بعد انتقال را رسم مي كنيم البته همرام با رويداد اولي كه موجب اين انتقال مي شود.

5. پاسخ را بايد در كنار ان بنويسيم .

6. در صورت لزوم هر كدام از state ها كه لازم است با اضافه كردن sub state توسعه مي دهيم.

7. در اين مرحله بايد مدل را چك كنيم كه ايا مدل هاي نوشته شده با Interface تطابق دارن يا نه؟

8. با يك ابزار يا يك ليست بين قسمت هاي مختلف حركت كنيد ببينيم ايا رفتارهاي سيستم درست است يا نه؟ ايا نقاط كور وجود دارد يا نه؟

فصل چهاردهم

State chart diagram

از اين خصيصه براي دو كار استفاده مي شود:

❶مدل كردن سيستم هاي عكس العملي

❷مهندسي مستقيم و معكوس

State chart diagram : محلي است كه state machine در ان قرار دارد.

✓**نكته:** براي مدل كردن state machine بهترين روش استفاده از روش Switch case است كه بر روي نام state ها كار مي كند .

FE & RE

Forward and Reverse Engineering

Forward Engeneer

برای مهندسی پیشرو ما ابتدا یک متغیر از State ها در نظر می گیریم و استفاده switch case .

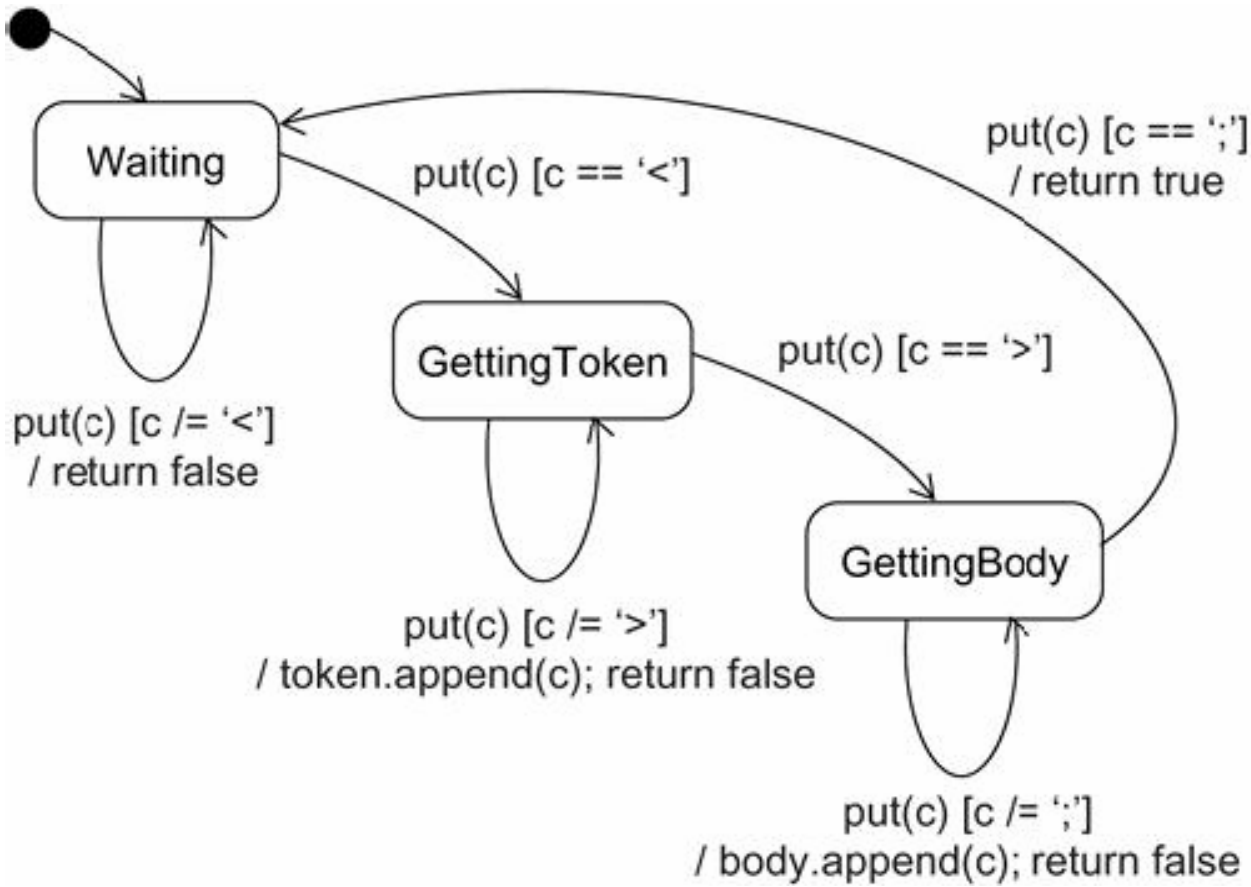
Reverse Engeneer

كردن از نظر تئوري قابل انجام است ولي معمولا نتيجه اي كه مي دهد نتيجه اي قابل توجه و خوب نيست.

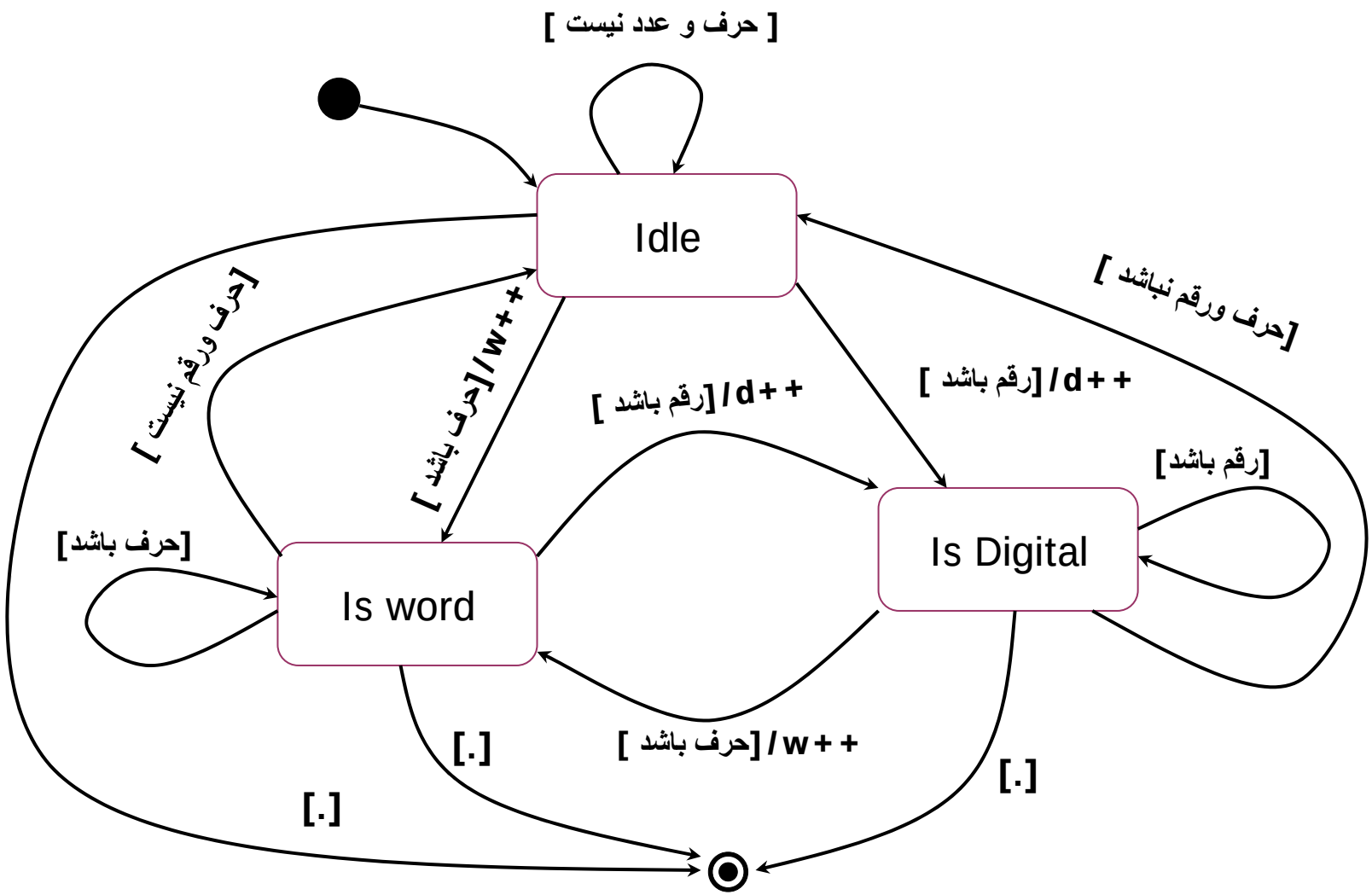
✓نکته : در State chart بیشتر از FE استفاده می شود .

مثال : می خواهیم برنامه ای بنویسیم که یک عبارت به این شکل است یا خیر ؟

Message : ‘<‘ String ’>’ String ‘ ; ’



مثال : می خواهیم State machin این برای اینکه تعداد اعداد و حروف بشمارد ؟



کارهاي Engineer در state chart diagram عبارتند از:

1. براي مدل كردن مسايل مفهومي
2. براي مدل كردن كلاس و در كد و در Engineer كردن
3. بايد حالتي نزديك به sequence باشد و ان را ضميمه