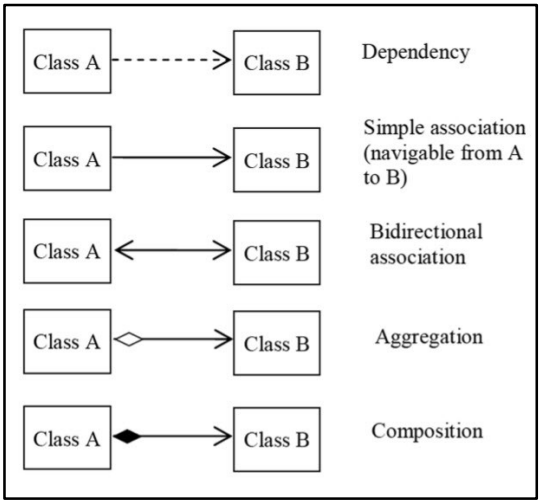
روابط بین کلاس ها

نمودارهای کلاس علاوه بر نشان دادن کلاس های جداگانه، روابط بین کلاس ها را نیز نشان می دهند که در ادامه به معرفی انواع آن خواهیم پرداخت.



وابستگی *(Dependency)*

وابستگی کلی ترین رابطه بین کلاس هاست و اینگونه تعریف می شود:

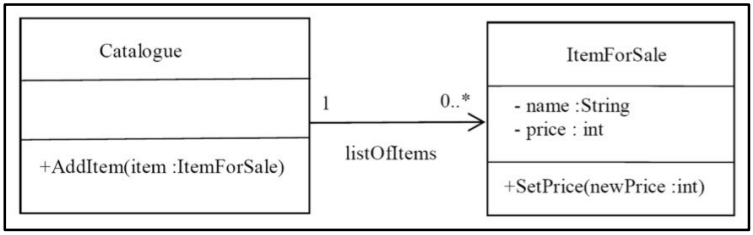
- کلاس **A** به نوعی از امکانات تعریف شده توسط کلاس **B** استفاده می کند
- تغییرات در کلاس **B** ممکن است بر کلاس **A** تأثیر بگذارد

معمولا این ارتباط در حالتی تعریف می شود که کلاس **A** دارای متدی است که یک شی بصورت پارامتری از کلاس **B** به آن ارسال می شود، یا از یک متغیر محلی آن کلاس استفاده می کند، یا متدهای "استاتیک" را در کلاس **B** فرا می خواند.

ارتباط ساده *(Simple Association)*

در این نوع از رابطه کلاس **A** از اشیاء کلاس **B** استفاده می کند ، یعنی یک نمونه از کلاس **A** می تواند به نمونه های کلاس **B** که با آن مرتبط است دسترسی پیدا کند. البته عکس این رابطه درست نیست بدین معنی که نمونه های کلاس **B** از کلاس **A** اطلاعی ندارد.

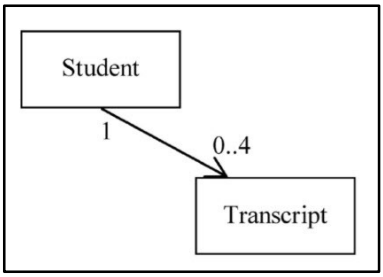
یک ارتباط ساده معمولاً با یک متغیر نمونه در کلاس **A** از نوع کلاس هدف **B** تعریف می شود.



مثال : همانطور که در شکل دیده می شود نمونه ای از کلاس کاتالوگ نیاز به دسترسی به صفر یا بیشتر از اجناس فروش **ItemForSale** دارد تا بتواند موارد را از یک کاتالوگ اضافه یا حذف کند. اما یک **ItemForSale** نیازی به دسترسی به یک کاتالوگ برای تعیین قیمت آن یا انجام روش های مرتبط دیگر ندارد.

ارتباط دوطرفه *(Bidirectional Association)*

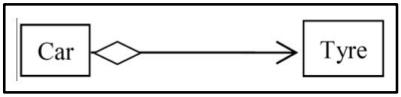
ارتباط دوطرفه زمانی است که کلاس های **A** و **B** دارای یک ارتباط دو طرفه بوده و هرکدام از خصوصیات یا متدهای دیگری استفاده می کنند.



مثال : یک مثال از ارتباط دو طرفه ممکن است بین "مدرک تحصیلی" و "دانشجو" باشد. یعنی در سیستم آموزشی بخواهیم بدانیم هر دانشجو باتوجه به رشته و مقطع باید چه مدرکی بگیرد و در مورد خود مدرک باید اطلاعات مربوط به دانشجو در آن ثبت شود.

رابطه تجمع *(aggregation)*

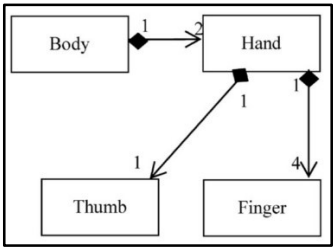
این رابطه وضعیتی را نشان می دهد که در آن اشیاء کلاس **B** به کلاس **A** تعلق دارند، اما نمونه های کلاس دوم استقلال وجودی دارند یعنی می توانند بدون وجود کلاس **A** نیز وجود داشته باشند.



شکل بالا ارتباطی بین دو کلاس خودرو و چرخ یا لاستیک آن را نمایش می دهد ، همانطور که می دانیم چرخ متعلق به کلاس خودرو بوده ولی بدون خودرو هم میتواند وجود داشته باشد (مثلا وسایل یدکی خودرو در فروشگاه).

رابطه ترکیب *(Composition)*

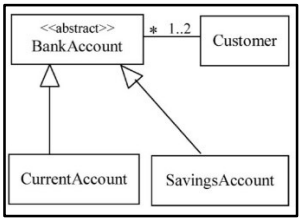
رابطه ترکیب شبیه به تجمیع است، اما دلالت بر یک رابطه تعلق بسیار قوی تر دارد، یعنی شی (های) از کلاس **B** بخشی از یک شی کلاس **A** است و در این مورد، اشیاء کلاس **B** بخشی جدایی ناپذیر از کلاس **A** بوده و اشیاء کلاس **B** هرگز به جز به عنوان بخشی از کلاس **A** وجود ندارند، یعنی "طول عمر" یکسانی دارند.



بعنوان ساده ترین مثال برای ترکیب میتوان ارتباط بین بدن و دست را در نظر گرفت ، همانطور که می دانید دست بخشی از بدن انسان بوده ولی در حالت طبیعی بدون بدن انسان نمی تواند وجود خارجی داشته باشد.

ارث بری *(Inheritance)*

وراثت یکی از اصول مهم برنامه نویسی شئ گراست که با استفاده از آن امکان استفاده مجدد از کدهای موجود فراهم می شود. بر اساس این اصل یک کلاس می تواند رفتار یا صفاتی را از کلاس دیگر به ارث ببرد



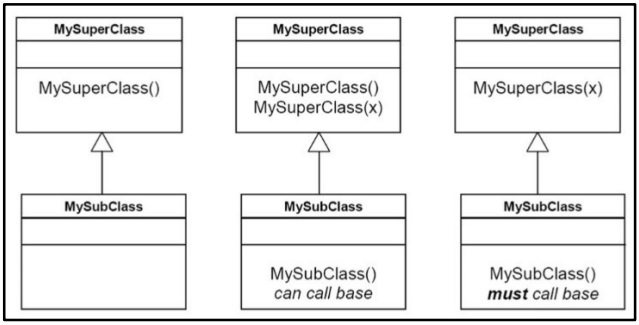
بعنوان مثال میتوان ارتباط بین حساب بانکی و حساب پس انداز را در نظر گرفت ، همانطور که می دانیم حساب پس انداز نوعی از حساب بانکی می باشد که مثلا خصوصیتی بنام نرخ بهره به آن اضافه شده است.

ارتباط بین کلاس دانشجو و کلاس دانشجوی کارشناسی ارشد را نیز میتوان برای وراثت در نظر گرفت ، همانطور که می دانیم دانشجوی کارشناسی ارشد هم دانشجوست (یعنی ثبت نام کرده، در کلاس درسی حاضر شده و امتحان می دهد) اما متدی بنام دفاع از پایانامه دارد که دانشجوی معمولی آن را انجام نمی دهد.

مثال : کلاس دانشجوی کارشناسی ارشد می تواند متدهایی مانند ثبت نام ، انتخاب واحد را از کلاس دانشجو ارث بری نماید.

متدهای سازنده در ارث بری

همانطور که اشاره شد سازنده بهنگام نمونه سازی از کلاس اگر بصورت عمومی تعریف شود بصورت خودکار فرآخوانی می شود اما بهنگام ارث بری از کلاس ها باید نکات زیر را رعایت کنیم



بعنوان نکته اول باید بدانیم که سازنده کلاس والد را می توانیم از کلاس فرزند فراخوانی کنیم (هر زبان برنامه نویسی برای این کار دستورات مخصوص به خود را دارد) اما بعنوان نکته دوم باید دانست که در حالتی که این سازنده پارامتر دار باشد حتما باید توسط کلاس فرزند فراخوانی و مقدار دهی شود.