به نام خدای رنگین کمان

مبحث شی گرایی (Object oriented programing) از قسمت هایی هست که از دنیای واقعی وارد دنیای برنامه نویسی شده و بسیار پرکاربرد و ضروری می باشد. همچنین برای درک بهتر فریم ورک جنگو خوب است که مفاهیم شی گرایی را به خوبی متوجه شده باشیم .

در این توضیحات از منابع سایت سبزلرن استفاده شده است.

مفاهیم زیر بسیار مکرر هستند :

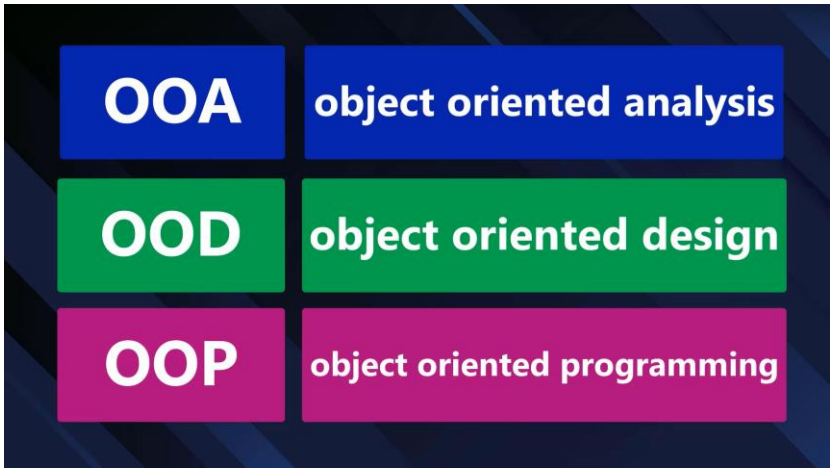
کلاس (Class) : طرح کلی که از روی آن ما شی هایمان را میسازیم.

شی (Object) : این یک نمونه (Instance) از کلاس هست که در اصل همان شی ما میباشد.

رفتار (Method) : رفتارهایی که ما برای شی خودمان باید در کلاس تعریف کنیم.

خصوصیات (Attributes) : ویژگی هایی برای شی مان که میتواند باعث تفاوت در شی هایمان شود و در کلاس تعریف می کنیم.

در کنار اسم oop اسامی دیگری نیز معمولا شنیده می شوند مثل ooa , ood



OOA : در اصل آنالیز شی گرا و مرحله اول هست که ما بررسی می کنیم چه رفتار ها ویژگی هایی و...لازم هست.

پاسخ به سوال چیکار باید بکنیم؟

OOD : در مرحله ی طراحی ما اسم گذاری برای شی ها ، تعریف رفتار ها و ارتباط بین اشیا و پاسخ به سوال چطور این کار را انجام دهیم ؟

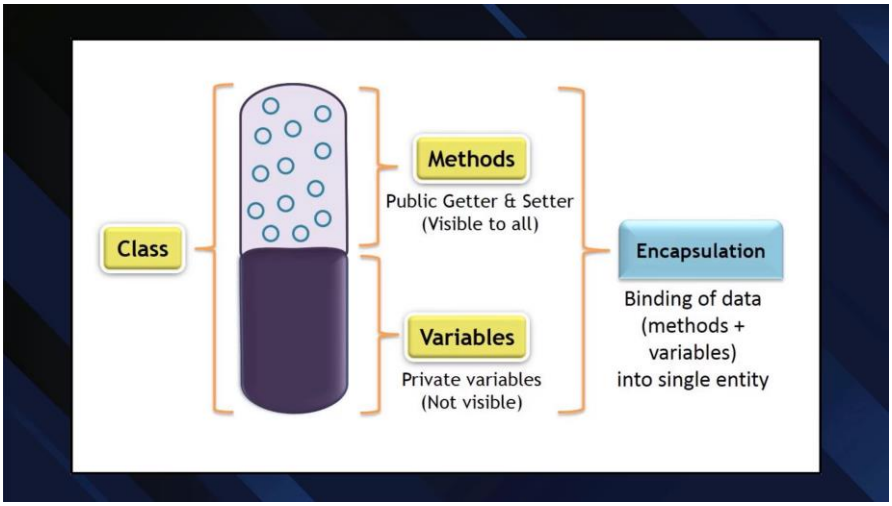
در مرحله ی آخر هم پیاده سازی می کنیم، اما امروز از این روش استفاده نمی کنند و یک بخش کوچک را در نظر می گیریم و اونا تا پیاده سازی می بریم و بعدش مرحله ی کوچک بعدی.

شی گرایی یک پارادایم در اصل هست، مثل تابع.

خوده پایتون بر اساس شی گرایی ساخته شده است. و همه چیز در پایتون در اصل یک شی هستش.

مفهوم کپسوله سازی (encapsulation): مثلا توی یک کپسول قرص انواع اقسام ویتامین هارو توی یک کپسول قرار داده اند.یعنی دسته بندی رفتار ها و خصوصیات را کناره هم جمع کرده تا یک کپسوله سازی کرده باشیم.

مفهوم پنهان سازی (data hiding): بعد از کپسوله سازی ما باید یک سری داده هارو از کاربر مخفی(اجازه دسترسی و تغییرات را ندهیم) کنیم. مثلا بتونه موجودی رو ببینه ولی نتونه موجودی خودش رو تغییر بده.



معمولا توی کپسوله سازی سعی بر این است که پیچیدگی رو کم کنیم ولی توی پنهان سازی سعی بر این مدیریت امنیت است.

زبان هایی مثل جاوا و سی پلاس پلاس یکسری کلمات خاص برای تعریف میزان این امنیت در کلاس ها دارند(private, public, protected)

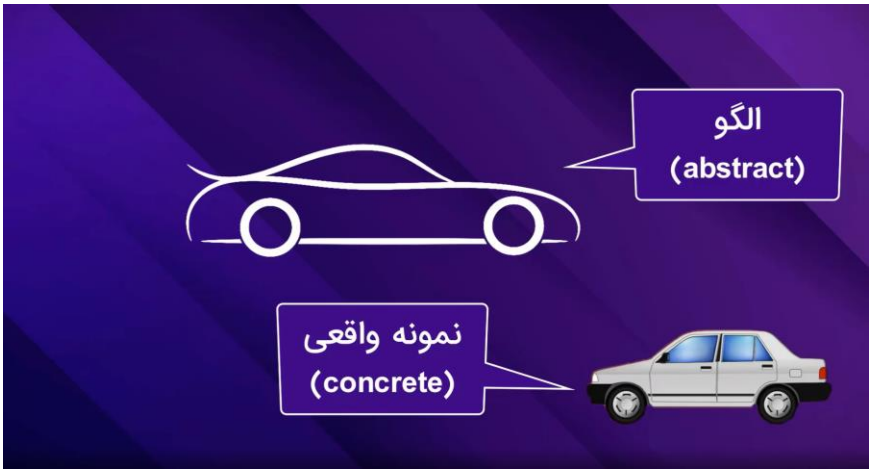
مفهوم روابط (interface): برای استفاده از اون شئ ساخته و با آن رابطه دارند که ما بتوانیم راحت تر اون شئ رو کنترل کنیم. مثل یک کنترل برای مدیریت تلوزیون. تقریبا مشخصات و توانایی اون شی رو تعریف میکنه.

در رابط نویسی توی دنیای برنامه نویسی ما به این سوال پاسخ میدیم که چه چیزی نیاز من هستش؟ و اینکه چجوری اینکار انجام میگیره برامون مهم نیست و فقط میخایم نتیجه بده. قابلیت هارو تعریف میکنی توی اینترفیس ولی پیاده سازی اون جداست و جای دیگست.

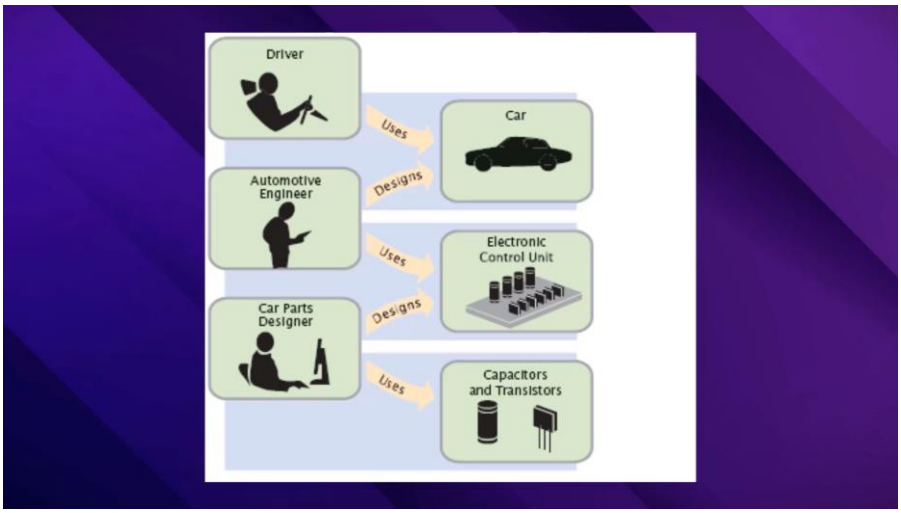
انتزاع (abstraction): وقتی که کلمه خودرو رو میشنوی فقط یک خودروی خاص مدنظر نیست.

میوه خوردم.... چه میوه ای ؟؟؟

یعنی یک ویژگی کلی و انتزاعی رو میفهمی و به صورت دقیق نمیدونی. پس انتزاع یعنی توجه به کلیات ، بدون توجه زیاد به جزئیات. پس ما مفهوم ماشین(abstract class) رو داریم ولی نمونه پیاده سازی آن پراید(concrete class) هست. و ما خوده ماشین رو نداریم بلکه انواع و نمونه هایی از آن هست.

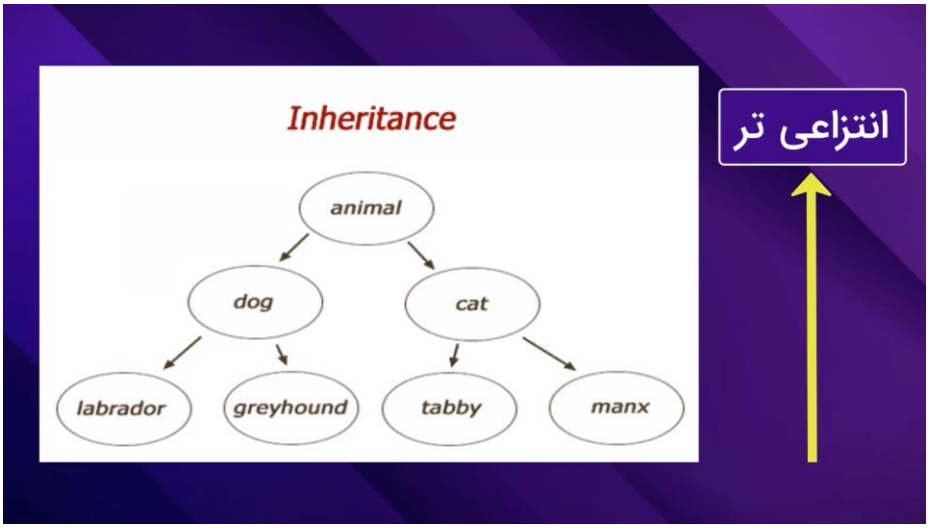


انتزاع میتونن سطح بندی بشن مثلا سطح یک مهندس خودرو بیشتره تا یک مصرف کننده خودرو.



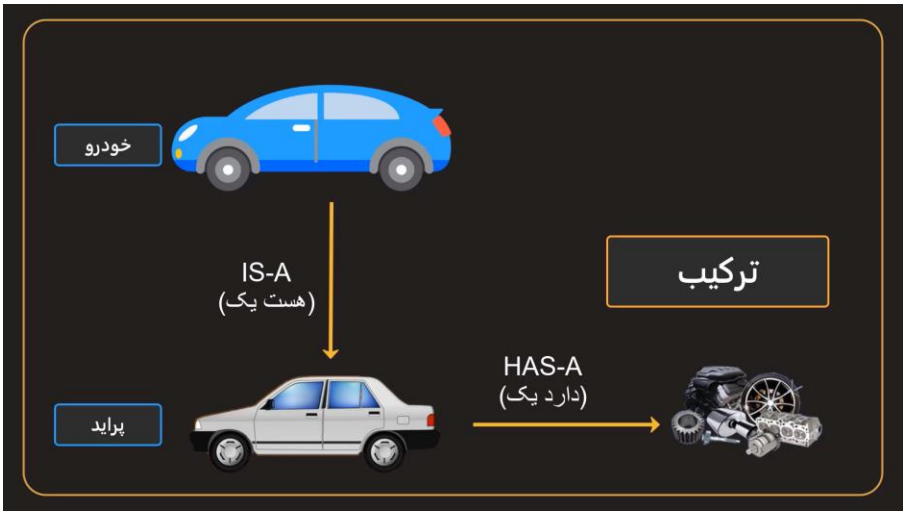
هرچه جزئیات کمتر باشه یعنی سطح انتزاع ما بالاتره.

برای سطح هر انتزاع هم می توانیم یک کلاس تعریف کنیم مثلا کلاس انتقال موجودی که میتونه از یه سری کلاس های دیگه ارث بری کنه و حتی یک سری کلاس ها ازش ارث بری کنند و میتونند به صورت جزئی تر بررسی بشوند.



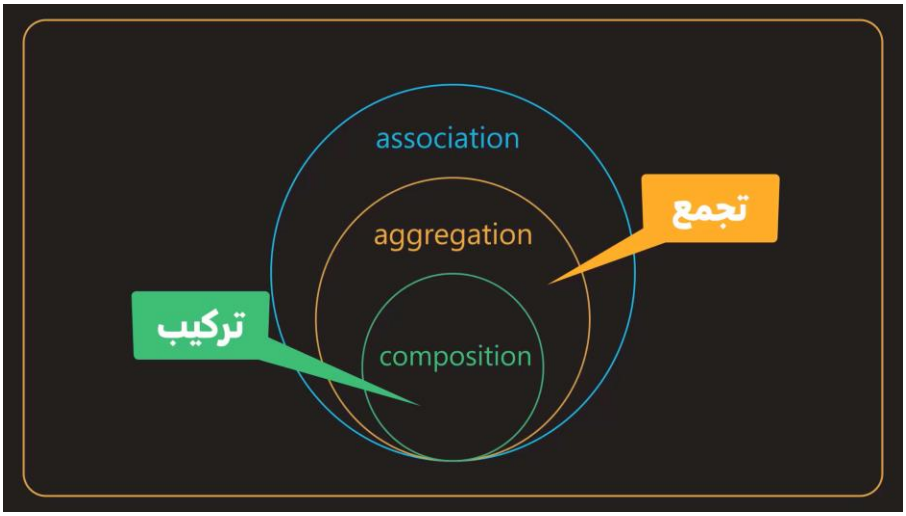
ترکیب(composition): با چه تکنیک هایی میشه انتزاع رو پیاده سازی کرد؟ یکیشون ترکیب و یکیشون ارث بری هستش.

ترکیب یعنی جمع آوری چندین شئ باهم ، برای ایجاد یک شئ جدید. برای زمانی استفاده می شود که یک شئ بخشی از یک شی دیگر باشد.



در این شکل پراید از خودرو یک ارث بری کرده و از آن تشکیل شده است ولی قطعات درون موتور در اصل ترکیبی از سایر قطعات هستند که از خودرو ارث بری نکردند و رابطه ترکیبی دارند .مثل میللنگ سوپاپ و.....

سه نوع رابطه داریم :



در association رابطه ضعیفی باهم دارند ولی حتی اگر یکیشون از بین هم برود زیاد تاثیری روی اون یکی نداره. مثل رابطه ی اکانت ها با هم توی اینستاگرام، یا رابطه استاد دانشگاه با دانشجو. یعنی در این رابطه معمولا شی اول جزئی از شئ دوم نیست.

در رابطه aggregation میتونیم بگیم اشیاء کوچکتر اون شئ اصلی رو میسازند برای بررسی دو تا شرط رو باید چک کنیم :

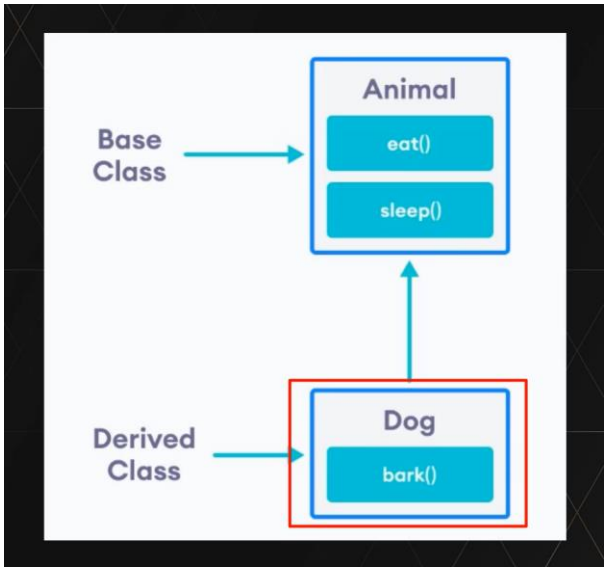
1. آیا این دو شئ مستقل از هم می توانند باشند و وجود دارند یا خیر؟
2. آیا ایجاد شدن یا نابود شدن شئ ها باعث دلالت ایجاد یا نابود شدن شئ دیگر میشود؟

مثل اشیا باتری و دوربین و شارژر در موبایل های قدیمی که مثلا شما میتوانی شارژر رو به بقیه گوشی ها هم بزنی یا باتری اون موبایل رو در بیاری و توی یک گوشی دیگه استفاده کنی. پس به جواب سوال یک میشه بله چون باتری جدا تولید میشه و سوال دو هم خیر چون میتونیم باتری رو عوض کنیم.

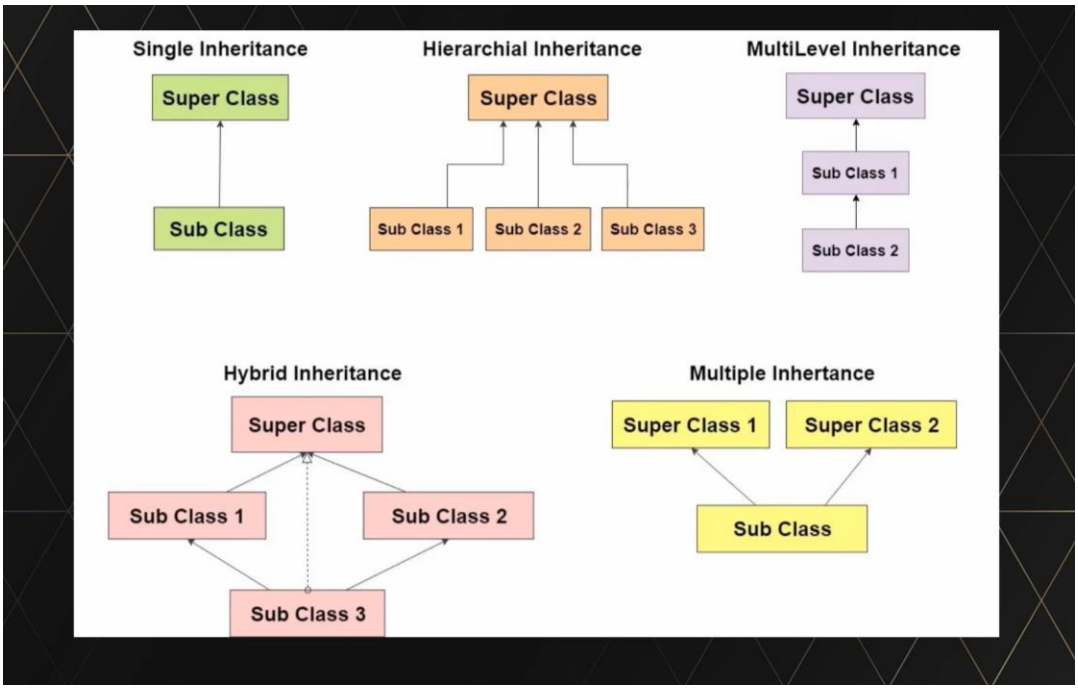
در رابطه compositions که وابستگی بسیار زیاده و به آن رابطه ی مرگ می گویند. یعنی مستقل از هم نیستند و فایده ای ندارند و نابود شدن هر کدوم دومی رو هم نابود میکنه. مثل برگه ی امتحانی و سوالاتش که اگه برگه گم بشه سوالات هم گم میشه. یا مدرسه رو بترکونی کلاساش هم نابود میشه.

بسته به دیدگاه شما میتونه این نوع های ترکیبات متفاوت باشه.

ارث بری (inheritance) : یعنی یک سری رفتار ها از کلاس پدر به کلاس فرزند منتقل میشه هرچند میتونه رفتار های خودش رو هم داشته باشه.



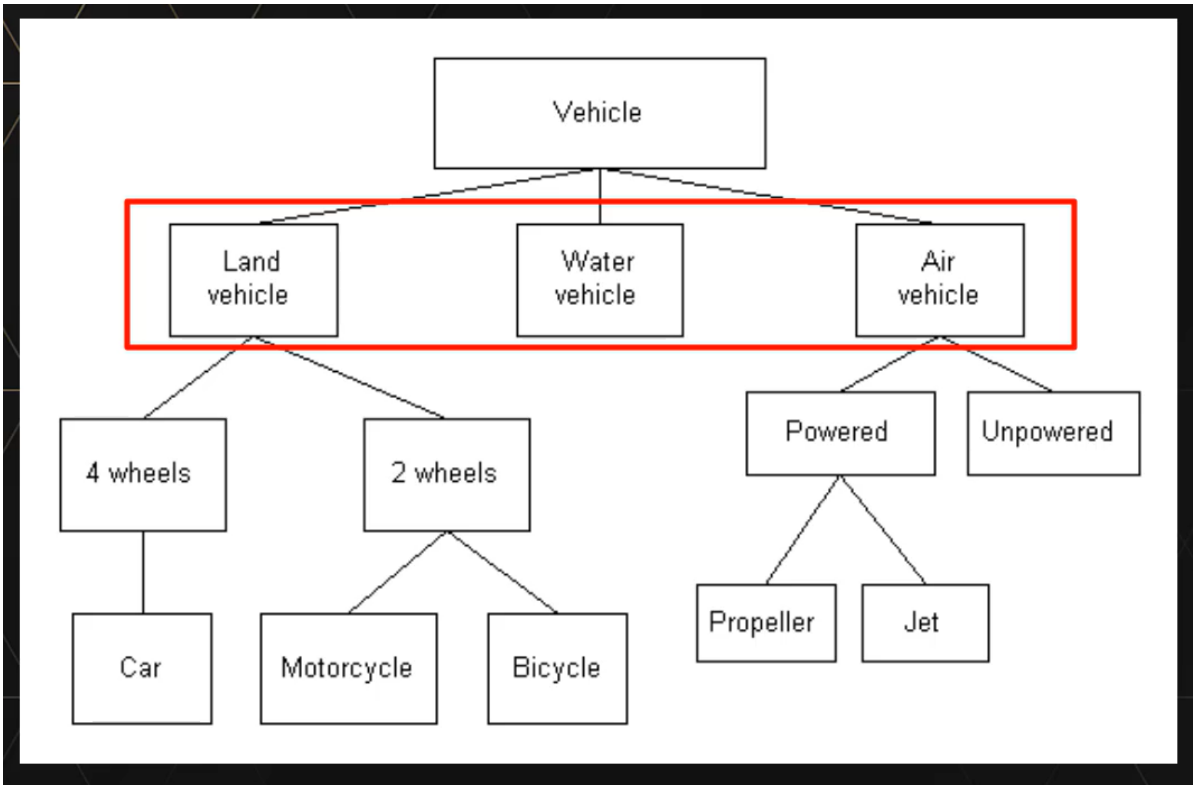
حالت های مختلف ارث بری که 5 نوعه و پایتون همرو ساپورت میکنه:



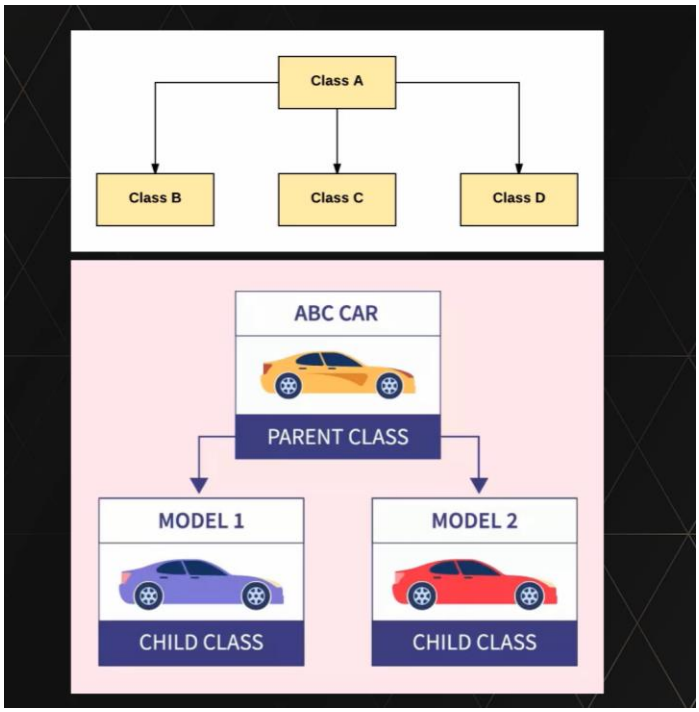
برای سینگل میشه سگ رو از کلاس حیوان مثال زد.

برای وراثت چند گانه میشه فرزند که از پدر مادر ارث بری میکنه.

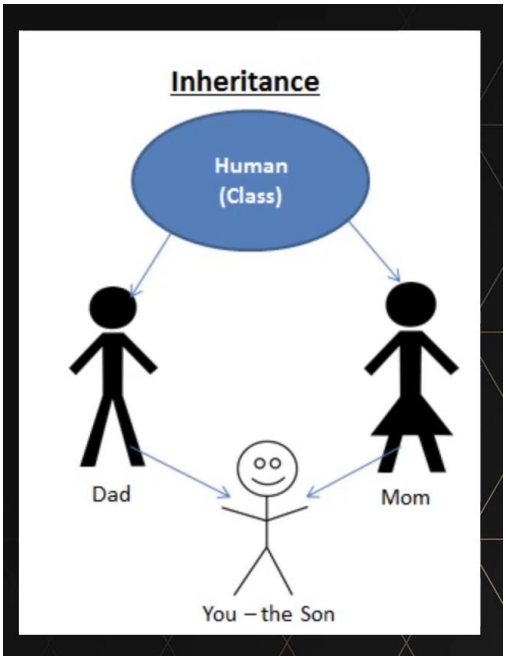
برای وراثت چند سطحی میشه وسیله نقلیه رو مثال بزنیم که آبی و هوایی و زمینی رو ارث بری کنیم و توی زمینی میشه چند چرخ و 4 چرخ و .... اررث بری کرد. اینجا میتونیم از هست – یک استفاده کنیم(is-a)



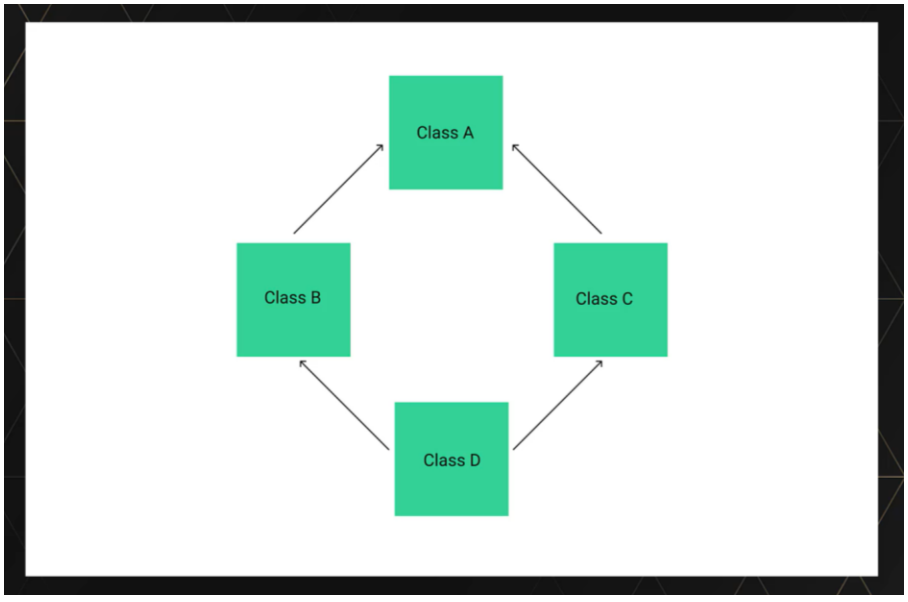
برای وراثت سلسله مراتبی میشه دوتا ماشین رو مثال زد که از خودرو ارث بری میکنند.



توی وراثت ترکیبی میشه به نوعی حالت های بالا ترکیب کنند که به این صورت باشد. که خیلی هم سخته و کمتر استفاده میشوند. مثلا زبان های جاوا و سی شارپ از این وراثت تا این لحظه پشتیبانی نمیکنند ولی پایتون میکنه .😊



مشکل الماس مشکلیه که در آن یک شئ از دو تا ابر کلاس دیگر ارث بری میکنه که این دوتا ابرکلاس از یک کلاس دیگر ارث بری میکنند.



MRO(method resolution order) : در مورد ترتیب جست و جو های متود ها و اتربیوت های من هست یا اول کدوم کلاس رو رو بررسی میکنه که ارث بری کرده یا نه.

در پایتون به صورت از پایین به بالا و از چپ به راست هست. مثلا در شکل بالا اول کلاس دی بعد کلاس بی و بعد کلاس سی و در اخر کلاس ای.

همه ی کلاس ها در پایتون از کلاس object ارث بری میکنند.

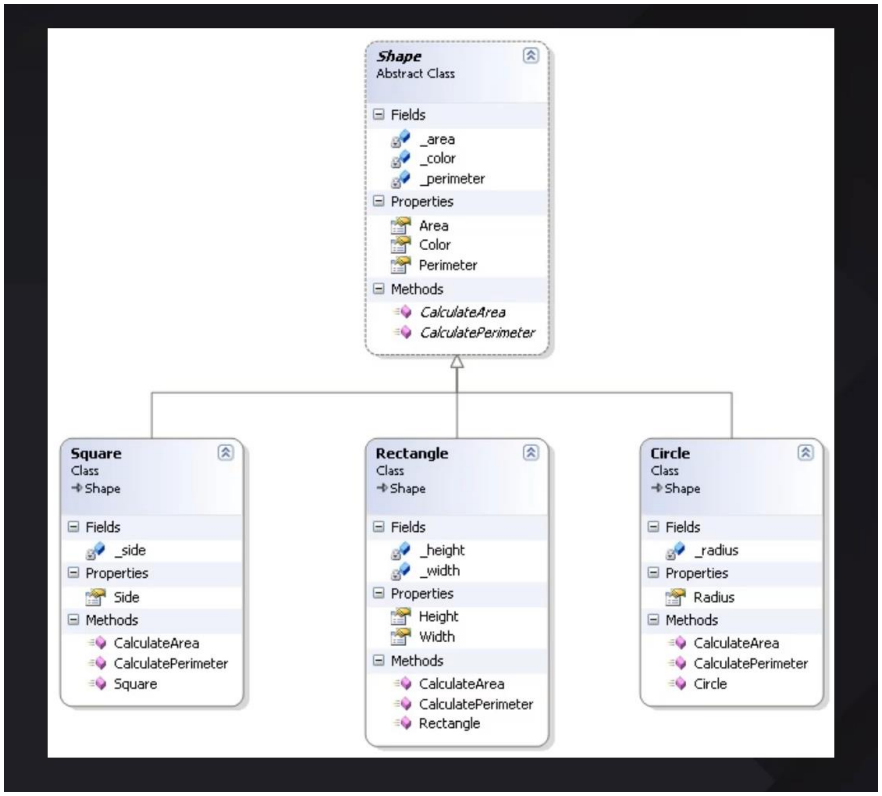
چند ریختی (polymorphism) : داره به چند شکلی بودن اشیا اشاره داره مثل فیلم ترانسفورمرز.

یا مثلا یک مرد میتونه یک جایی همسر باشه یک جایی فرزند باشه یک جایی پدر باشه و.....

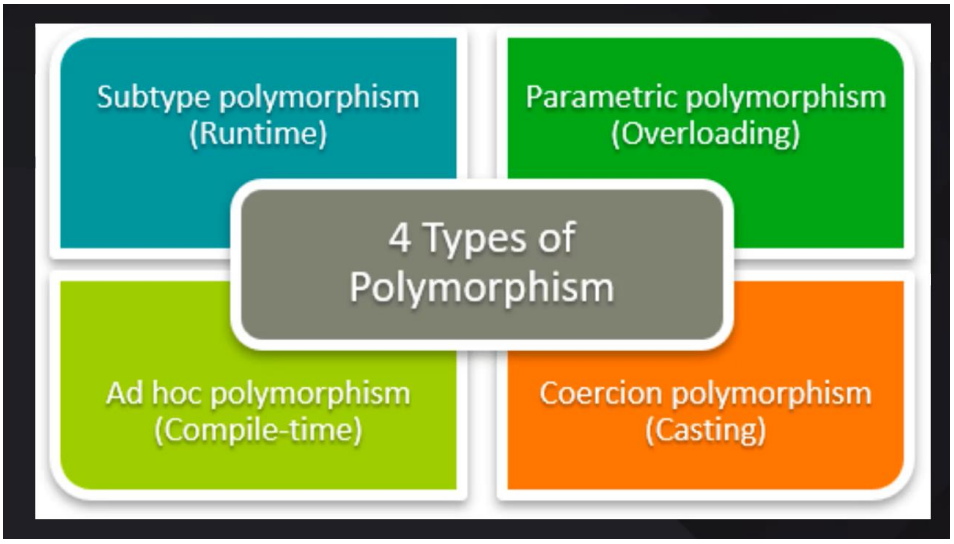
مثلا تابع ()move توی هر شی حییون میتونه حالت های مختلفی داشته باشه مثلا ماهی شنا میکنه یا گربه چهارپایی میره (میتونه حالت دیفالت رو هم داشته باشه یا اینکه باید حتما پیاده سازی بشه که یعنی دیفالتی وجود نداره.)

یا مثلا متود حقوق یا تسک برای همه کلاس کارمند تعریف شده و بسته به مقام هر کارمند این رفتار ها متفاوت میشوند.

مثلا در مثال زیر ما محاسبه محیط و مساحت را داریم.



انواع چند ریختی داریم و تقریبا هر زبان یک نوع پیاده سازی خاص خودش رو داره اما به صورت کلی:



در حالت ران تایم (ساب تایپ) : در 70 درصد مواقع منظور از چند ریختی همین نوع هستش. در این حالت ما بیشتر overwrite رو داریم که همون تابع ()move رو ارث بری کردیم ولی دوباره شخصی سازیش هم میکنیم.

در حالت اورلودینگ (پارامتریک) : بیشتر در زبان سی پلاس پلاس هست و به این صورته که یک تابع میتونه روی چندین نوع کلاس مختلف اعمال میشه مثلا جمع کردن که باید بتونه انواع داده هارو جمع بکنه.

در حالت کامپایل تایم(موردی) : چند تا تابع متفاوت دارم که اسماشون یکیه ولی ورودی های متفاوت دارند که این مورد به دلیل عدم اجازه شباهت اسم توابع در پایتون به راحتی قابل پیاده سازی نیست.

در حالت اخر(casting) یک نوع به نوع دیگه تبدیل خواهد شد.

مثلا تابع ()len برای چندین نوع داده کار میکنه و در اصل چند ریختی شده است.

بر اساس رفتار شی اگه قضاوت کنیم بهش duck typing می گویند که یعنی خصوصیات و رفتار شکل مورد توجه هست.