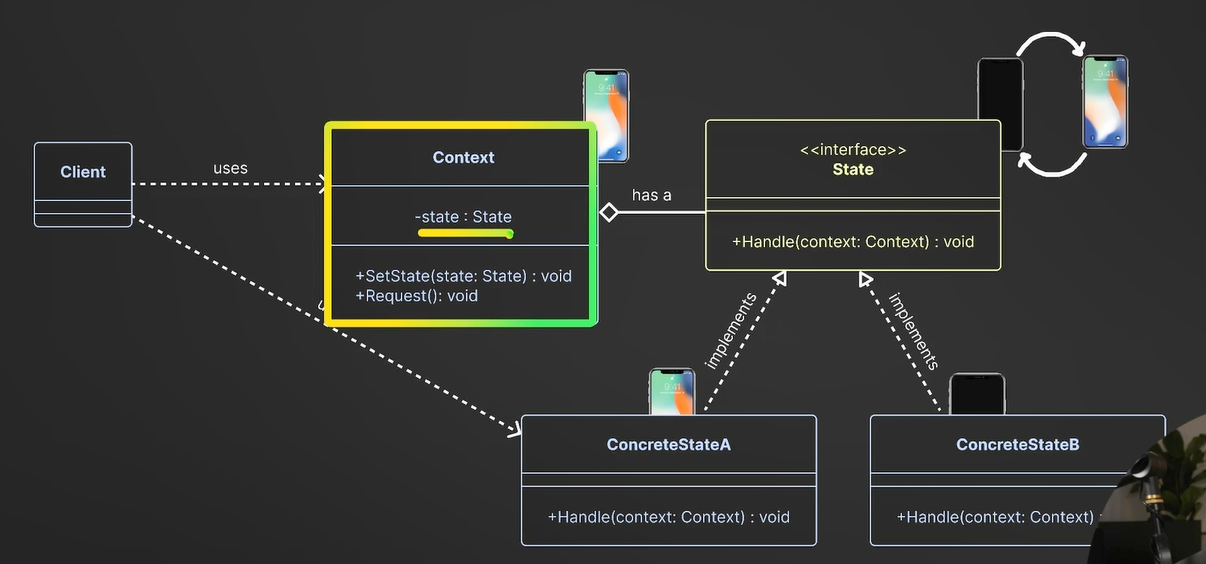
1. Singleton؟
   1. تعریف:
   2. این پترن thread-safe نیست به دلایل زیر:
      1. Race condition: ممکنه چندتا thread به صورت همزمان درخواست ایجاد شئ بدن که هنوز برای اولین بار ایجاد نشده.
      2. Partial initialization: ممکنه در حین پروسه ایجاد instance برای یه thread، یه thread دیگه ای بیاد و درخواست این شئ رو بکنه که در این صورت هر دو thread دارن عمل new رو انجام میدن.  
         برای رفع این مشکل، از یه lock و lock object بعد از چک کردن null بودن instance، استفاده میشه.
2. Factory method؟
3. Abstract factory؟
4. Builder؟
5. Strategy؟
6. Observer؟
7. State؟
   1. تعریف کلی این پترن میگه که یه آبجکت زمانی که وضعیت درونی اش تغییر میکنه میتونه رفتارشو تغییر بده. مثلا دکمه پاور یه گوشی رو در نظر بگیریم؛ اگه گوشی در وضعیت روشن باشه این دکمه کار خاموش/لاک کردن رو انجام میده و اگه خاموش باشه، کار روشن کردن رو.  
      دقت کنیم که در هر حال، اینترفیس ما که همون دکمه هه باشه یکسانه؛ چیزی که داره تغییر میکنه رفتار گوشی هست (که در این حالت داره وضعیت رو هم تغییر میده)
   2. یه نمونه از class diagram این پترن رو داشته باشیم:  
      
   3. این پترن شباهت های زیادی به Strategy داره. در واقع انگار عین همدیگه ان اما تفاوتی هم دارن. در state، متد داخل اینترفیس یه شئ از نوع context رو داره میگیره. در حالی که داخل strategy اینطور نیست. از طرفی، در state، هر ConcreteState میتونه از state های دیگه باخبر باشه در حالی که در strategy اینطوری نیست و هر ConcreteStrategy داره به صورت مستقل عمل می کنه.  
      در واقع، strategy داره میگه یه عمل با چه روش ها و مدل هایی میتونه انجام بشه اما state میگه چطور میتونم از یه وضعیت به یه وضعیت دیگه برم و در هر وضعیت چه کار هایی میتونم انجام بدم.  
      توی strategy، بیشتر داره external state رو هندل می کنه (مثلا کاربر میخواد یه کاری رو به شکل های مختلف انجام بده) اما state سمت internal state هست (یه درخواست در چه مرحله ای از flow مورد نظر هست).
   4. مزایا:
      1. حفظ single responsibility: هر state، خودش داره کار های مربوط به خودشو هندل می کنه. کس دیگه ای کارای اونو براش انجام نمیده.
      2. جلوگیری از کثیف شدن کد و استفاده بیش از اندازه از condition ها
      3. جلوگیری از branching (یک حالت داخل خودش یه حالت دیگه صدا کنه و اونم دوباره به همین شکل و تمام این کار ها داخل یه scope اتفاق بیفته در صورتی که راحت میشه هر کدوم رو جداگونه توی کلاس خودش نوشت)
      4. به مرور زمان، سمت کلاینت تغییری ایجاد نمیشه. تنها جایی که تغییر میتونه بکنه ConcreteState ها هستن که یا خودشون تغییر می کنن یا state های جدید اضافه میشه اما سمت کلاینت دست نخورده میمونه و حتی اینترفیسمون هم احتمال پایینی داره که تغییر کنه.
   5. معایب:
      1. تعریف تعداد زیادی کلاس (برای هر State یه کلاس تعریف میشه)
      2. در سناریو های ساده، استفاده از این پترن پیچیدگی کد رو افزایش میده.
      3. ممکنه جابجایی بین state ها پیچیده باشه و سختی هایی ایجاد کنه (که البته من مخالفم چون وقتی تمام کار های یه state خاص داره توی یه جا انجام میشه، کار ما راحتتره و اگه اشتباهی هم پیش میاد یا چیزی جا می مونه دیگه این تقصیر برنامه نویسه نه پترن. در واقع یا بیزنس مشکل داره یا رعایت نشده یا نکته فنی ای رو برنامه نویس از قلم انداخته)
      4. با بزرگتر شدن پروژه هم ممکنه پیچیدگی بالا بره؛ مثلا یه سری validation به state ها اضافه بشه که قبلا نیاز نبوده.
      5. تست کردنش میتونه سخت باشه چون سناریو های زیادی رو ممکنه نیاز باشه بررسی کنیم.