Game Programming Patterns – Sequencing – Update Method

"Simulate a collection of independent objects by telling each to process one frame of behavior at a time."

* we want a skeleton to move one step each frame in a patrol.
  ↳ so we rely on the outer game loop to trigger an update. Now we need to track direction w/a variable.

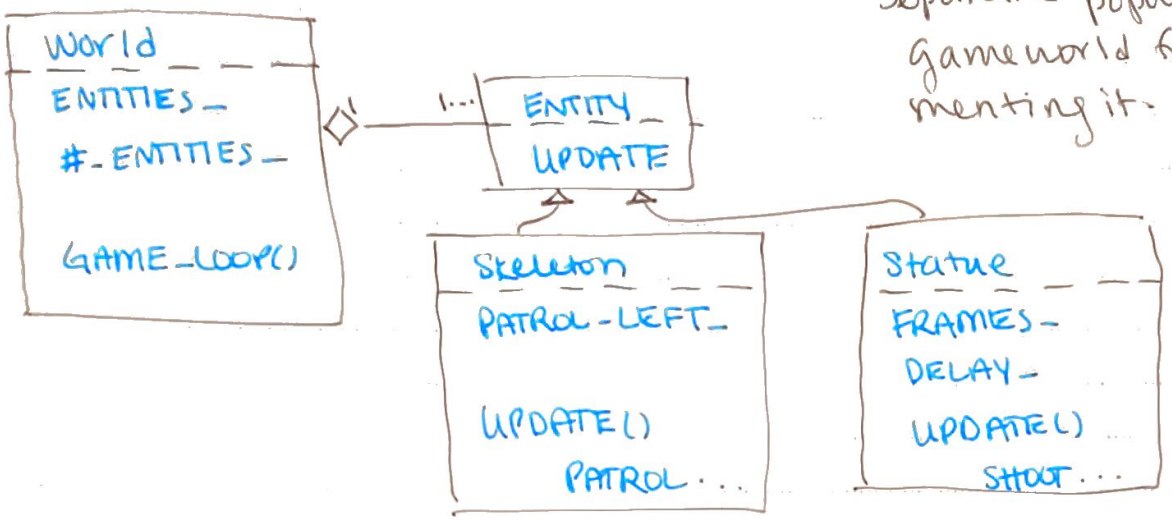
"Anytime "mushed" accurately describes your architecture, you likely have a problem"

So... each entity in the game should encapsulate its own behavior. Use an abstract update() method + the game loop has a collection of objs that can be updated

- Helpful to use if you need to update behavior or animation every frame for a collection of objects, the objs. are mostly independent + need to be run simultaneously / simulated over time
- Typically splitting behavior over frames makes code ↑ complex. If you have a lang that supports lightweight concurrency (generators, coroutines, fibers) + the obj. code can pause, you may be able to be more imperative
- You need to store state so the obj. can resume (State pattern)
- Objs. aren't truly concurrent, so order matters (or you could use double buffer pattern), but this is helpful for networked games and if two entities have a conflict.
- Be careful modifying the object list while updating, so you can cache the collection at the beginning of the loop + only update what was present at the beginning of the loop
- Also be care. when removing. you could update backwards so you only remove objs that have already been updated or mark "dead" objs + remove after iterating (+ skip anything "dead")

"Favor 'object composition' over 'class inheritance'"

"It's now much easier to add new entities into the game world because each one brings along everything it needs to take care of itself."



Separate populating the game world from implementing it.

* The Update() method may need to consider variable time steps. ↳ take in the elapsed time.
- What class does the Update() method live on?
  - ↳ Entity: makes it difficult to reuse behaviors
  - ↳ Component: can decouple parts of a single entity. (Rendering, physics, etc)
  - ↳ Delegate: (State or Type Obj patterns can do this) share behavior across a bunch of entities of the same "kind" You may have update() on main class but it forwards to a delegated object
- How to handle dormant objects?
  - ↳ single collection w/ inactive objs: waste time, CPU cycles + possibly blow up your cache
  - ↳ sep. " only active objs: extra mem., helpful when speed is tighter than memory. the second collection could just be inactive vs all objs to mitigate mem. usage.
    - ↳ ↑ inactive objs regularly, this can be helpful
    - keep collections in sync