Game Programming Patterns - Revisited - Flyweight

* ex. rendering a giant, lush forest likely many of the details will be similar b/w trees
  - pull out common details into its own class (intrinsic state)
    instanced rendering allow you to tell the GPU to render one tree model for many instances of the tree
  - ② list of instances + their parameters (extrinsic state)

"Flyweight comes into play when you have objs that need to be more lightweight, generally because you have too many of them." * Flyweight is about efficiency

* this pattern becomes interesting when there isn't a well defined identity for the shared object.

"Sharing objects to save memory should be an optimization that doesn't affect the visible behavior of the app. Because of this, Flyweight objs are almost always immutable"

+ But since the Terrain class, once instantiated, only contains intrinsic state information the grid[x][y] can be full of pointers to instantiations of the Terrain tile types so the obj can be referenced directly.

"if you find yourself creating an enum and doing lots of switches on it, consider this pattern instead"

* the golden rule of optimization is to profile first
* can create the instances on demand, see factory method
* need to keep track of created instances, see object pool
* when using the State pattern you may be able to reuse the same state instance in 2+ state machines at the same time w/o problems