John Strobel Strob105

Design Document

Between the three ways of creating the various simulation entities, instantiating them as is in the provided code is probably the worst, objectively. From our point of view, relegating everything to a constructor (that can be easily modified without having to modify the points of instantiation) fits our purposes perfectly. This is true regardless of which version of instantiation you choose to employ. This leaves three instantiation strategies for the proposed factory: having a single class with many different construction methods, an abstract factory class in which one instantiates subclasses to do the construction, and finally a single class that 'has' another class called FactoryBehavior.

Of the three strategies, creating subclasses the least efficient and controllable. The whole purpose of this style of design is to abstract away the creation process from the user-programmer (aka. future me) and ensure that modification will not break any existing code. I believe that if you have a class for each type of object that you are trying to instantiate, the problem of having to hunt through code in order to fix each point of instantiation will not be solved. The only difference is that you will be changing the factory instead of a class definition, which is little consolation. There is an upside to this, though: if you want to change the factory for a single object, you can easily avoid change to the other types of object factories.

Having a single class that 'has' a behavior class that determines how the factory operates is actually quite similar to creating factory subclasses. Instead of a single creating entity, you must create a factory and factory behavior for each type of entity that you want to create. The cons are much the same as above, but there is a pro: because the behaviors would be loosely coupled with the factory main class, it should be easy to modify the behavior parent or children classes.

Finally, the best option is using a single factory class and writing a method for each different desired construction type. The advantages to this system are, primarily, that you don't have to deal with *another* complex class structure on top of the one that you are trying to manage

by creating factories. In this case, each type of object is represented by a single function, and the factory can be stored as a global object. The downsides to this would be that, in large projects, you would likely end up instantiating multiple versions of this factory anyways, somewhat offsetting the benefit. Additionally, with a large library of objects, having a single factory with 80 functions would likely be overwhelming for developers. However, in a project like the one given where there are perhaps 10 different types of objects, a single factory is the optimal solution. This is the solution I opted to implement, using a unique method for each object that the factory should instantiate.