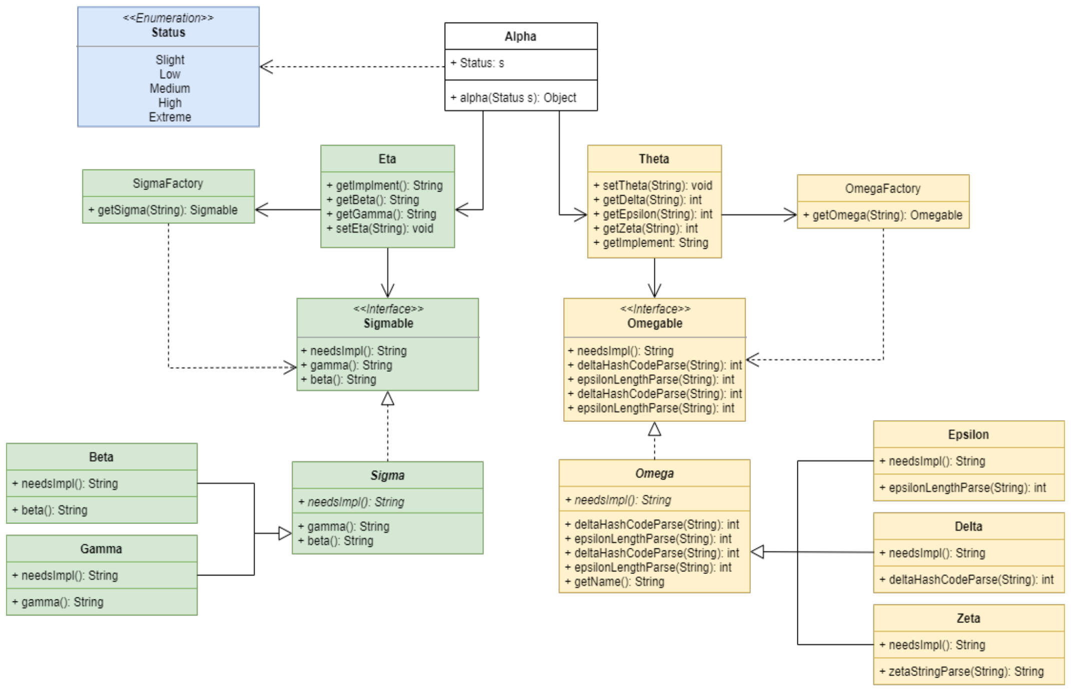
OOP Assessment 1

Conor Rabbitte G00365218

November 8, 2020



Assuming Delta, Epsilon, and Zeta (Yellow) are in a group of their own, Gamma and Beta (Green) are in another, and Status and Alpha don't belong to a group.

I extracted a new abstract class Omega from which Delta, Epsilon and Zeta extend. I then extracted an interface Omegable from Omega which encapsulates Omega and all its extending classes, leaving the interface as the only point of entry for all things extending Omega. Next, I used the factory creational design pattern with a singleton to create a class OmegaFactory, which has a dependency on Omegable for its ***getOmega(String): Omegable*** method. This is used to create new classes of type Omegable (e.g. Delta/Epsilon/Zeta). I decoupled the usage of an object from its creation by creating the Theta class as a dependency injection. Theta holds private references of OmegaFactory and

Omegable and through public methods allows for the creation and manipulation of the object created.

I repeated the steps above for Gamma and Beta by extracting an abstract class Sigma and then extracting an interface Sigmable. I created the SigmaFactory which has a dependency on Simgable. Finally, I created Eta as a dependency injection which holds private references on SigmaFactory and Sigmable which can be accessed through public methods.

Using these design patterns allowed me to half the dependencies on Alpha form six to three. This allows ease of extensibility and reusability through abstraction, interfaces, and factories. Alpha now contains two private references to Theta and Eta. Therefore in its alpha method it decides what class to create and manipulate, based on the Status parameter passed.

I believe this helps promote high cohesion and loose coupling. The user will only have to create an instance of Alpha and print out a line with the alpha method passing Status as the parameter.