Object-Oriented Design and Methodology: Final

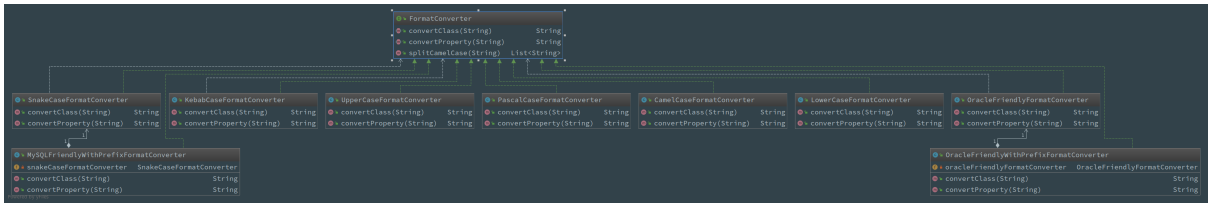
Pitipat Chairoj (Fronk)

ID: 5980759

pitipat.chi@student.mahidol.edu

Link to Github repository: https://github.com/pitipat1998/ooc-final

Before we delve into the explanation of the design for this assignment, I want to mention a few dependencies I use to implement this project. Firstly, dependencies for string conversion are apache commons lang3 and apache commons text in which apache commons lang3 provides a very handy tool (StringUtils) to convert strings into uppercase and lowercase easily, and additionally, it also provides method for splitting camelCase strings into a list of words. Furthermore, apache commons text has a tool (CaseUtils) for me to easily convert any strings into camelCase and PascalCase. Apart from string conversion dependencies, dependencies I use for rest api is spring boot starter web, with an addition of lombok boilerplate. For testing, both service layer and controller layer, I use spring boot starter test, apache httpcomponents and gson, for json conversion.



Now let's first have a look at how classes are related. In this final assignment, I created an interface called *FormatConverter* that every other converters must implement. The interface consists of 2 signatures, which are *convertClass* and *convertProperty*, for converters to implement for their own format. Moreover, the interface also provide an default but optional method *splitCamelCase* for any converters that implement this interface to use it to split the input name into a more easy-to-handle format.

Creation design patterns I use for this assignment are factory-method and singleton. I implements my own a some-kind of factory method for converters in the service layer. This factory simply does the following, it first reads the input format, then find an associative converter for the particular format to do the conversion before returning the result string. Another and most prevalent design pattern for this assignment is singleton. I did not do this

design pattern directly, but indirectly through the use of java beans – components, services and controllers. Java beans can be achieve by using spring annotations and the rest, spring will handle for us, including instantiation, naming and etc. Every converters in this assignment is a singleton because I only need one instance of each converter to done all the jobs. Creating a new instance of converter for each request seems to be a little bit wasted of resources. In addition, by having each converter as a singleton or a bean, I can do one really good trick provided by spring which is dependency injection. Dependency injection allows me to have an instance of a class I wanted to use other classes by automatically instantiate the desired class into the classes that depend on it before starting to use them, and spring magically do this internally through *@Autowired* annotation. This helps reduce a lot of line of codes.

Now that I have covered some design patterns for the main functional parts, next things I want to dive into is the architecture of this assignment. As I already mentioned some of the word, which are service and controller layers, this as a whole can be called as multi-layered architecture. In this architecture, we layered logic and entities into different level of abstractions, and each layer can call only its lower layer. For example, Controller layers can only call Service layers, and Service layer can only calls Repository layers. With this scheme, it provides an encapsulation of data and logic in lower layers from the higher layer which consequently results in cleaner codes, easy to manipulate, and easy to understand when we work in a big project. Nonetheless, in this assignment I only use Controller layer, the highest layer that is responsible for serving request and response to users, and Service layer, a layer that contains most of logic to process request received from controller layer, due to no requirements of database. Moreover, I use DTO (Data Transfer Object), another design pattern, to gather information to pass around between Controller layer and Service layer.