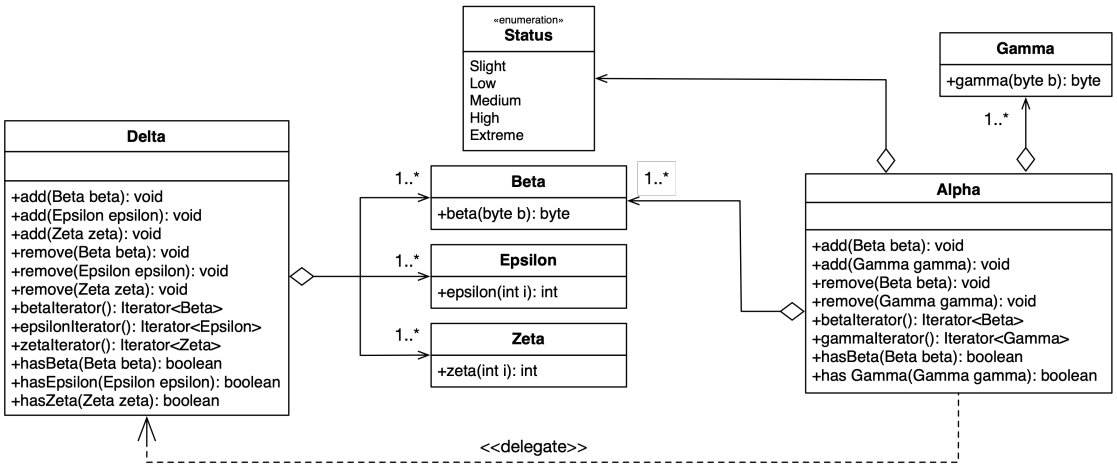
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

Advanced Object-Oriented Design Principles & Patterns (2020)
ASSESSMENT II

***Note*: This assessment constitutes 25% of the total marks for this module.**

The following UML diagram shows a **poorly designed** suite of classes in an application:



You are required to provide a ***redesign and refactoring*** of the classes above that eliminates redundancy and duplication, provides maximum reusability, extensibility and satisfies the following additional requirements:

- A Client class should be able to **iterate over all instances** of Beta, Epsilon, Zeta and Gamma in the system and **update their state in a uniform** way. The Client class should also create instances of the main objects in your design and call their key methods. Ideally, the class Client should not need to know about any of the add/remove or iterator methods in Delta or Alpha.
- Instances of the class Beta or any of its subtypes should never be stored by Alpha, but the class Delta should exist as a singleton. Possible subtypes of Beta are limited to the classes Eta, Theta, Iota and Kappa, all of which implement an additional method that takes in a byte and returns a byte. In addition, a range of other restrictions could potentially be applied to control access to Delta from Beta.

You are free to change any of the classes in any way you wish, including renaming, changing method signatures and deriving new types. Any new classes or class names should be taken from the set *{Eta, Theta, Iota, Kappa, Lambda, Omicron, Sigma, Omega}*. You must implement your design as a set of Java classes and ***document your rationale in no more than 300 words*** in a README file and a UML class diagram. You should also document your rationale for the design of each class in *JavaDoc* comments after the *package* statement.

Note that there is no single "correct" answer to this assessment – there are many possible solutions, all with their advantages and drawbacks. Any design patterns that may apply should already exist in the problem. State any **assumptions or known issues** relating to your design in comments at the top of your classes or in the README file.

INSTRUCTIONS FOR SUBMITTING YOUR WORK

Please **read the following carefully** and pay particular attention to the files that you are required to submit and those that you should not include with your work:

- ***The sumbission must be uploaded to Moodle by midnight on Sunday 20th December 2020***.
- A **set of stubs** for the classes in the UML diagram is available on Moodle.
- Use the module name **gmit.software** and the package name **ie.gmit.sw**.
- Instrument all methods with some minimal functionality or with *System.out.println()* statements. Note that the **focus of this assignment is on design, not on robustness**. Do not waste your time implementing features and functionality that are not needed.
- The project must be submitted as a Zip archive ***(not a 7z, rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "*Upload Assessment II - (25%)*" link on Moodle. Only the contents of the submitted Zip will be considered. ***Do not add comments to the Moodle assignment upload form.***
- The name of the Zip archive should be <*id*>.zip where <*id*> is your GMIT student number.
- The Zip archive should have the structure shown below. Do NOT submit the assignment as an Eclipse project.

Marks	Component	Category
40	**src**	A directory that contains the packaged source code for your work.
30	**README.pdf**	A **PDF** file detailing the main features of your application in **no more than 300 words**. All **features and their design rationale** must be fully documented. You should consider also including the UML class diagram in this document to help explain your design clearly.
10	**design.png**	A **UML class diagram** of your API design. The UML diagram should only show the relationships between the key classes in your design. Do not show private methods or attributes in your class diagram. You can create high quality UML diagrams online at **www.draw.io**.
10	**docs**	A directory containing the **JavaDocs** for your application. You can generate JavaDocs with the following command from inside the "*src*" folder of the Eclipse project: javadoc -d [path to javadoc destination directory] ie.gmit.sw Make sure that you read the **JavaDoc** tutorial provided on Moodle and comment your source code correctly using the **JavaDoc** standard.