# 2 Assignment Tasks

## 2.1 Task 1 : Implementation Description. (2 points )

### • How flexible is your implementation, e.g., how you add or remove in future new drink types?

With both the abstract factory pattern for different semantic element types according to different

customer types it is quite easy to add or remove these and thusly this design is quite flexible.

It deserves noting that with five distinct customer types as per the requirements the number

of corresponding semantic element classes grows very quickly, 3 for every 1 new customer type.

With the given requirements, the elements package is approaching 20 classes. Further, the email

package is approaching 15 classes, mostly different customer type classes. Increased code means

more code to maintain, however, the flexibility is good as previously mentioned.

Further, the decorator pattern, shown in the utilities package, is also quite flexible with

regards to adding or removing functionalities following this pattern. Therein lies the beauty

of the pattern.

Both the abstract factory pattern and the decorator pattern are demonstrating high cohesion

and low coupling, resulting in high flexibility. Removing a class or adding a class in either

class is a simple as that, no adjustments to code outside of the class being added or removed

are necessary.

• How is the simplicity and understandability of your implementation?

Even though this implementation demonstrates the layering of three patterns (Singleton EmailGenerationSystem,

the abstract factory for semantic elements according to customer types, and the decorator pattern

for ad hoc functionality to the emails such as spell and grammar checks and encryption)

the implementation remains simple and intuitive to understand given a basic understanding

of the requirements provided by the assignment. Although there are surely other ways to satisfy

the requirements, it seems that this layering of these three designs may be the most

intuitive approach. If different semantic elements are needed per customer types it makes sense

to start there, the abstract factory allows for high cohesion and low coupling to achieve this.

The decorator pattern allows for high cohesion and low coupling to achieve adding

functionalities at will to emails and a Singleton is perhaps the only way to enforce a

central, singular generation system. Although the parts are complex and the overall design

is layered, the implementation seems to remain simple and understandable consequently.

• How you avoided duplicated code?

By adhering as strictly as possible to the patterns described above and factoring out

any necessary tangential processes such as verifying an email address provided

for a sender or receiver is in the proper format is factored out into it's own function.

Even in the Main class, factoring out the log format into its own function. Otherwise

there is very little repetition. Although it deserves mentioning that the assembleEmail()

method of the utilities package classes (adhering to the decorator pattern), although

are similar are stand in's for essentially different algorithms. This goes for the check()

and encrypt() functions in the respective Checker and Encryptor classes as well. These

are currently placeholders for these respective algorithms since this assignment was

about the design and not these algorithms which in of themselves are studies in

their own right - we're simply showing here where these belong in the design and consequently

the illusion of duplicated code may come across when in fact this is not an accurate assessment. Otherwise, by adhering to the respective patterns there is almost no duplicated code necessary.

## 2.2 Task 2 - UML Class Diagram. (5 points)

Complete UML Class Diagram [link](https://lucid.app/lucidchart/6da3b896-06f6-42c1-8c34-fd1574ed10e2/edit?page=0_0&invitationId=inv_2c5297b9-e2d1-4a56-ad34-a287ffa5051d)

. Scribe: Abstract Factory Pattern Class Diagram

Diagram, schematic

Description automatically generated

. Scribe: Decorator Pattern UML Class Diagram

Diagram

Description automatically generated

. Scribe: Singleton Pattern UML Class Diagram

Diagram

Description automatically generated

# Task 3 - Implement your solution in Java (13 points )

<https://github.com/metcs/met-cs665-assignment-3-MichaelKramerGuitar>