Project Sample: Designing an Email Generator

This paper is a summarization of the project I worked on. In this project, I designed an email generator that will print out the header, main context and footer for each email. This program is designed to be customizable depending on the customer type (Business, VIP, etc.). The documentation will explain the design decisions followed by how the program should be run in the command line. Lastly, I attached the program folder for your reference.

# Assignment 3 - Factory Pattern Implementation in Email Generator.

Results Documentation

Task 1: Implementation Description

How the application is implemented is based on the following Design Principles:

1. **Flexibility:** The Factory Method implemented in this design allows for flexibility. The factory method "separates product construction code" from the "product that actually uses this" (Refactoring Guru, n.d.). What this means is that subclasses can be responsible for "creating the instance of the class" at runtime (Javatpoint, n.d.). For example, if the email factory class calls the "generateEmail" method, only the customer type has to be specified and the subclass will be responsible for creating the object.

2. **Understandability:** The classes and methods are named clearly. For example, the "EMailFactory" class indicates that it is a "Factory" class. In addition, the "generateEmail" method clearly indicates an email will be generated based on the Customer Type parameter.

3. **Duplicate code** is avoided by using inheritance relationships. For example, the Customer parent class allows for subclasses such as "VIP" to inherit it's attributes and methods. For example, if the "getCustomerType" method is called, the program is written to accurately get the right customer type attribute. Inside the VIP subclass, the "getCustomerType" method is pre-written to set the attribute to "VIP" so that when the method is called, it will accurately retrieve it.

4. **Reusability**: The implementation allows for reusability through the factory method. The "generateEmail" method can not only "create new objects" but "reuse exisitng objects as well" (Refactoring Guru, n.d.).

5. Software Design Pattern: The Factory Pattern was selected. The reason for this was that the company needs to generate customizable emails at the "right time" during run time. Predicting which object (or email) to generate can be difficult. Per Professor Teymourian's lecture and the module 3 notes, a method would be more suited for this rather than a constructor. The Factory pattern will allow for this. In addition, this would allow for more reusability of code.

First, an interface called "EMailGenerator" was created and uses the method "GenerateEmail" based on the Customer type input (see assumptions below). Once the method is called, the EMailFactory will be used to "get a Customer Object"(javatpoint, n.d.). The type of Customer passed as a parameter will be used to obtain the exact type of object needed.

Second, the Customer class was made abstract. This is to adhere to the requirement to have an inheritance relationship (as stated in "Recordiing for week 3, Group 4 Friday 7/23 Live Office"). The subclasses will be able to inherit the attributes and methods from that abstract class.

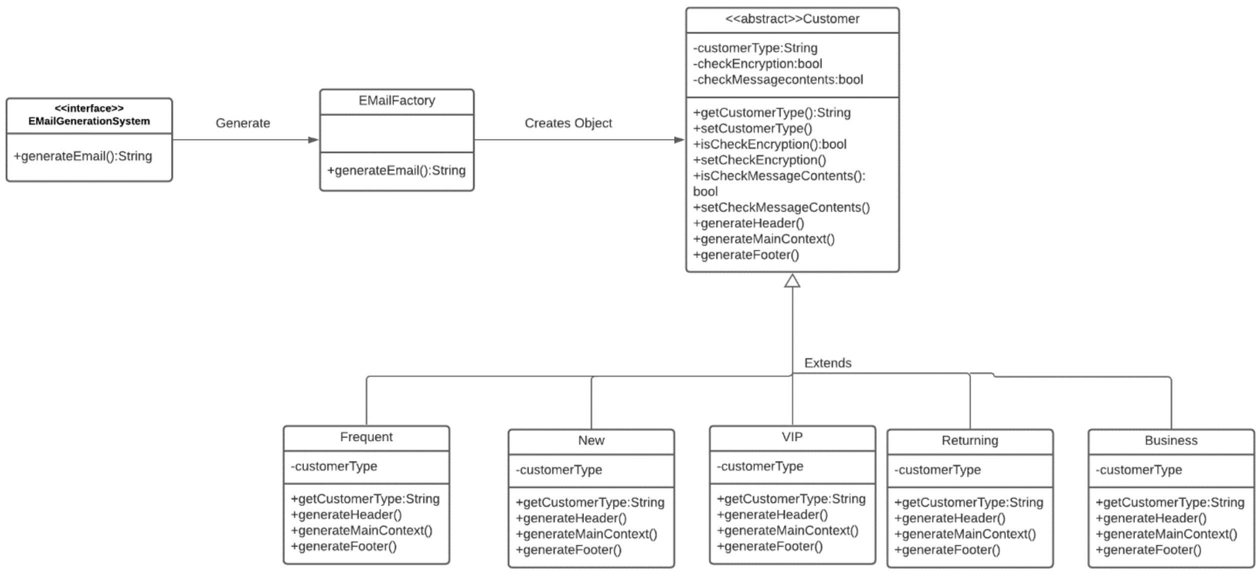
Third, there are two boolean variables (checkEncryption and checkMessageContents in Customer class) to verify the existence of these features in certain classes. These were written specifically for the test case. The reason it is a bool is because the Encryption Messages and check message contents are in the EmailFactory class. Per the requirements stated, these messages can only be added after the email is generated. Given the design is an email factory, retrieving it would require changing the factory class. Thus, to ensure the factory class remains the same and comply with the test case, this boolean check was created in the customer class. If there is a detection of certain emails being generated (such as for business or VIP), the boolean will be changed from "false" to "true". Otherwise, it will remain false.

One trade off noted in this project, the Email Factory class code is longer. This is because the "generateHeader", "generateMainContext" and "generateFooter" methods were called per object. The intent was to display the email being generated. Originally (see my github repository history), my Customer class had a code where the constructor called those methods. However, per Professor Kia's lecture on July 22,2021, that is not a favorable programming practice. As a result, I complied and removed those methods and had them generated in the factory class instead.

Task 2: UML Class Diagram

The UML Class Diagram was designed based on the requirements and Software Design Principles.

Picture of Diagram



Task 3 -Implement your solution in Java

These are the following results for Task 3:

How to compile the project

1. Use Apache Maven to compile and run this project.

2. Install Apache Maven (https://maven.apache.org/) on your system.

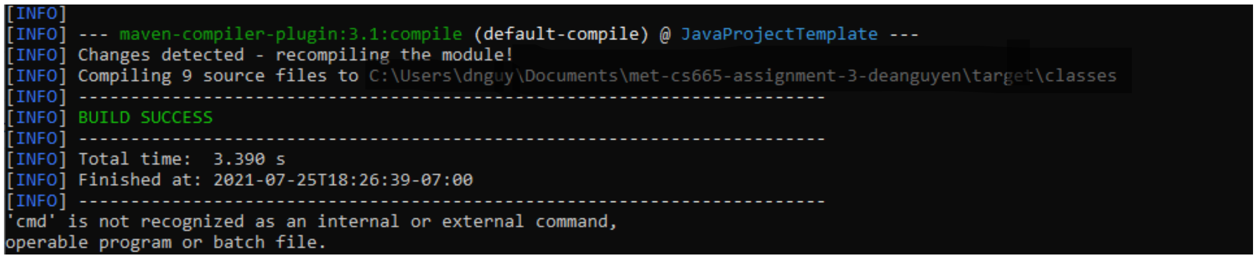
3. Download the zip file from blackboard and extract the file into a directory.

4. Type "cd" into the exact folder the file was extracted from.

5. Type "cd met-cs665-assignment-3-deanguyen-master".

6. Type on the command line: mvn clean compile

The outcome of the successful build should state "BUILD SUCCESS" as shown below.



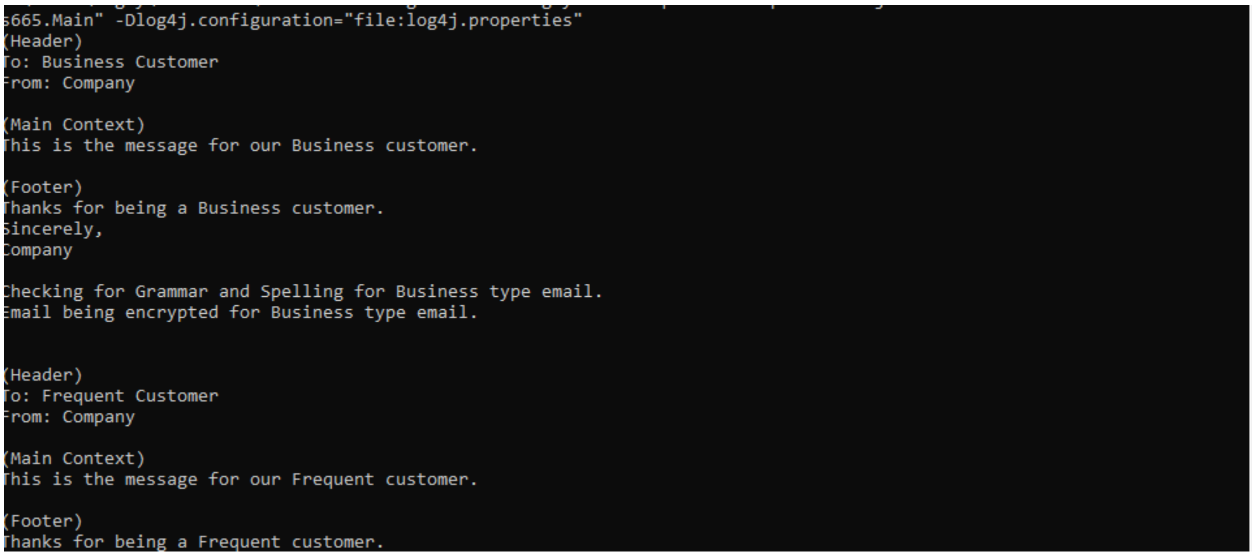
```
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ JavaProjectTemplate ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 9 source files to C:\Users\dnguy\Documents\met-cs665-assignment-3-deanguyen\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.390 s
[INFO] Finished at: 2021-07-25T18:26:39-07:00
[INFO] ------------------------------------------------------------------------
'cmd' is not recognized as an internal or external command,
operable program or batch file.
```

How to run

1. Use the following code below:

mvn -q clean compile exec:java -Dexec.executable="edu.bu.met.cs665.Main" -Dlog4j.configuration="file:log4j.properties"

The expected output is running the sample code in the main program. The outcome of this should appear as shown below:



```
s665.Main" -Dlog4j.configuration="file:log4j.properties"
(Header)
o: Business Customer
rom: Company

(Main Context)
his is the message for our Business customer.

(Footer)
hanks for being a Business customer.
incerely,
ompany

hecking for Grammar and Spelling for Business type email.
mail being encrypted for Business type email.


(Header)
o: Frequent Customer
rom: Company

(Main Context)
his is the message for our Frequent customer.

(Footer)
hanks for being a Frequent customer.
```

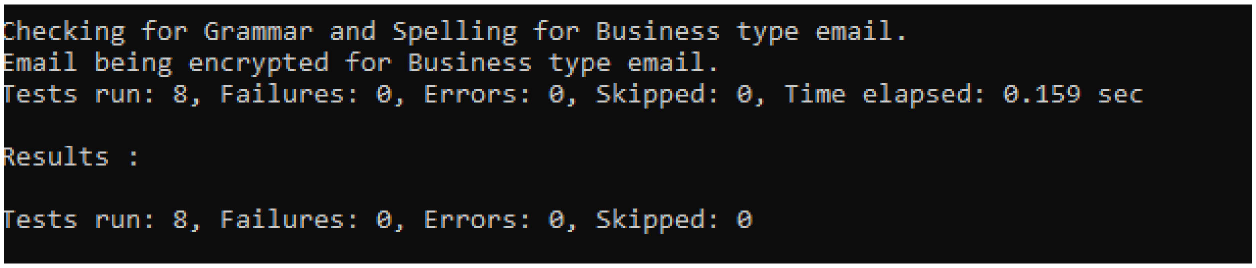
It is expected that the entire email each customer type (starting out with business) will be displayed for the user to see.

Run all the unit test classes (8 unit tests expected)

Use the following code: mvn clean compile test checkstyle:check spotbugs:check

The intent of the TestEmailGenerator.java class is to do the following:

1. Verify the email will be generated with 1 instance.
2. Check that objects for all the customer subclasses are created.
3. Verify the encryption and messages are created.



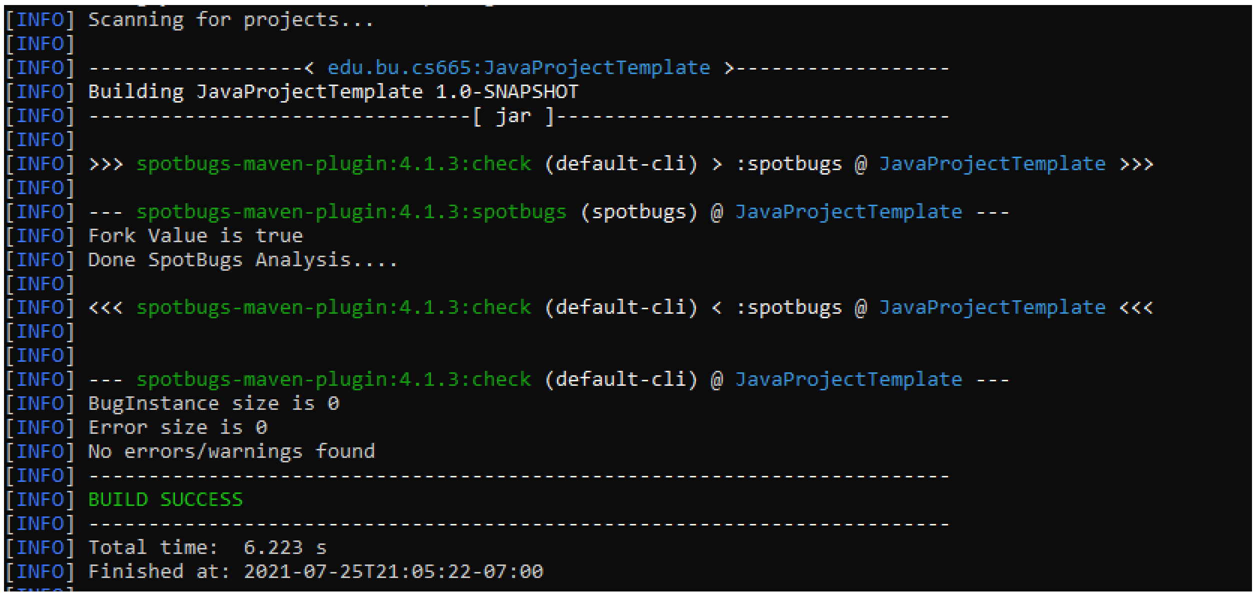
```
hecking for Grammar and Spelling for Business type email.
mail being encrypted for Business type email.
ests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.159 sec

esults :

ests run: 8, Failures: 0, Errors: 0, Skipped: 0
```

Find bugs in your project

To find bugs in the project, use the command mvn clean compile spotbugs:check. The expected output should appear as shown below:



```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< edu.bu.cs665:JavaProjectTemplate >------------------
[INFO] Building JavaProjectTemplate 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spotbugs-maven-plugin:4.1.3:check (default-cli) > :spotbugs @ JavaProjectTemplate >>>
[INFO]
[INFO] --- spotbugs-maven-plugin:4.1.3:spotbugs (spotbugs) @ JavaProjectTemplate ---
[INFO] Fork Value is true
[INFO] Done SpotBugs Analysis....
[INFO]
[INFO] <<< spotbugs-maven-plugin:4.1.3:check (default-cli) < :spotbugs @ JavaProjectTemplate <<<
[INFO]
[INFO]
[INFO] --- spotbugs-maven-plugin:4.1.3:check (default-cli) @ JavaProjectTemplate ---
[INFO] BugInstance size is 0
[INFO] Error size is 0
[INFO] No errors/warnings found
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  6.223 s
[INFO] Finished at: 2021-07-25T21:05:22-07:00
```

Referenced Links

https://refactoring.guru/design-patterns/factory-method

https://www.javatpoint.com/factory-method-design-pattern

https://learn.bu.edu/bbcswebdav/pid-9107740-dt-content-rid-56186992_1/courses/21sum2metcs665so2/course/module3/allpages.htm

https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

https://bostonu.zoom.us/rec/share/GplrWmvlinD6KdRXovI0JezUDjctqRhTbGed1VD5suRWbI4Fs9Qt0IGerYmzGvyu.mnH0D01e8gQE4kUC

https://bostonu.zoom.us/rec/share/-ueICjHwATBB9L66u4nL754iTdyFtwH5f7DpOQRkeNSuGVFgfx9lk9P_Ww2NCYy4.l__lNkEw7i4gvQHJ