Implementation Descriptions

"E-Mail Generation" Application will auto generate based on specific customer types.

(KINDLY REFER JAVA DOCS for more detail)

Design Pattern

Used ***Factory Method pattern*** to design as we want system to generate right email automatically as per customer type instead explicitly calling the methods.

Assumptions

Types of Customers : Business, Returning, Frequent, New and VIP

Company/Client Name : Email Advertisement

Packages

- **cs665** -> "Creator" (wrt factory pattern) or main package
- **emailtypes** -> "Products" (wrt factory pattern) or contains classes dedicated to each possible email types

General Working

Client through Main function calls email factory with recipient types which then creates the class of that specific recipient type.

Classes/Interfaces

main package :

- **EmailGeneratorFactory** : Interface declaring the factory method. It got structure of calling recipient specific email.
- **EmailAdvertismentClientFactory** : Implements and overrides methods of "EmailGeneratorFactory". As per the recipient type received it will create the email type object and send email.

emailtypes package :

- **TemplateEmail** : It is an abstract class which will be extended by other types of email classes. It defines a general structure of email which is then modified as per requirement by the classes. Sub classes are changing email subject, body and message to be printed once email delivered.
- **BusinessEmail** : For recipient type business.
- **FrequentEmail** : For recipient type frequent.
- **NewEmail** : For recipient type new.
- **ReturningEmail** : For recipient returning.
- **VipEmail** : For recipient vip.
- **InvalidEmail** : Handles all incorrect invalid requests made.

Field and Methods

EmailGeneratorFactory

createEmail(String)

It take type as argument and create the object of that type of email. Client only calls this function all the complexity of object creation and different email format is handled without client knowing the implementation.

createEmail(String, String)

Used to add new recipient to list. It take type and recipient name as argument and create the object of that type of email. Client only calls this function all the complexity of object creation and different email format is handled without client knowing the implementation.

TemplateEmail

subject (String) : Email subject

body (String) : Email body

recipient (List<String>) : Email recipients

Getters and Setter Methods

getSubject()

setSubject(String)

getBody()

setBody(String)

getRecipient()

setRecipient(List<String>)

Other Methods

addRecipient(String) : Adds new recipient to the list and sends them mail.

sendEmail(String) : Sends the email to the new recipient added using addRecipient method. If some things are to be done before sending email to recipients, those could be implemented in this method. It provides the flexibility to extend application. It's an overloaded method.

sendEmail() : Sends the email to recipient. If some things are to be done before sending email to recipients, those could be implemented in this method. It provides the flexibility to extend application.

BusinessEmail, FrequentEmail, NewEmail, ReturningEmail, VipEmail, InvalidEmail

<?>EmailSubject : Subject value for the recipient type which is over rided or assigned to "subject" inherited. In future if subject content needs to be changed then either a new variable can be created or its value can be updated thus not touching the main code.

<?>EmailBody : Works similar to subject variable. It contains the email body.

sendEmail() : Sends the email to recipient. If some things are to be done before sending email to recipients, those could be implemented in this method. It provides the flexibility to extend application. This is override from parent class.

addRecipient(String) : Adds new recipient to the mailing list of type.

sendEmail(String) : Overloaded function, sends email to new recipient.

constructor : It sets subject and body value and sends email.

Goals of Software Design

Aim of this application is to deliver the recipient specific application just by knowing their type and it is suffice that requirement. There are different classes for each purpose and each identity like creating mail, for business recipient, for returning recipient, etc so it's a modular code with more cohesion and less coupling implementing hiding.

If later types needs to be increased developer just need to add more class in email types or if way the mail is sent needs to be changed one can update sendEmail method. Thus its flexible and reusable code.

To increase code under stability classes, variables and methods name symbolises operation they perform.

2.2 Task 2 - UML Class Diagram.