Simulación de un sistema de comunicación celular

Eliuth Balderas Neri

Instituto Tecnológico de Monterrey

Campus Querétaro

Programación orientada a objetos

Profesor:

Pedro Oscar Pérez Murueta

Fecha de entrega: 18 de junio del 2022

Índice de contenido

Planteamiento del problema

En el presente proyecto se busca hacer una simulación de un sistema de comunicación celular. Dicho sistema está compuesto por 3 clases base: Customer, Bill y Operator, la cual cuenta con dos clases derivadas: InternetOperator y VoxOperator. El sistema le permite a los clientes hablar y enviar mensajes a terceras personas o incluso conectarse a internet. Cada uno de los servicios tiene un costo en específico. Así pues, al finalizar el uso de los servicios, se crea una factura para cada cliente, en donde se le de el total,

Diagrama de clases UML

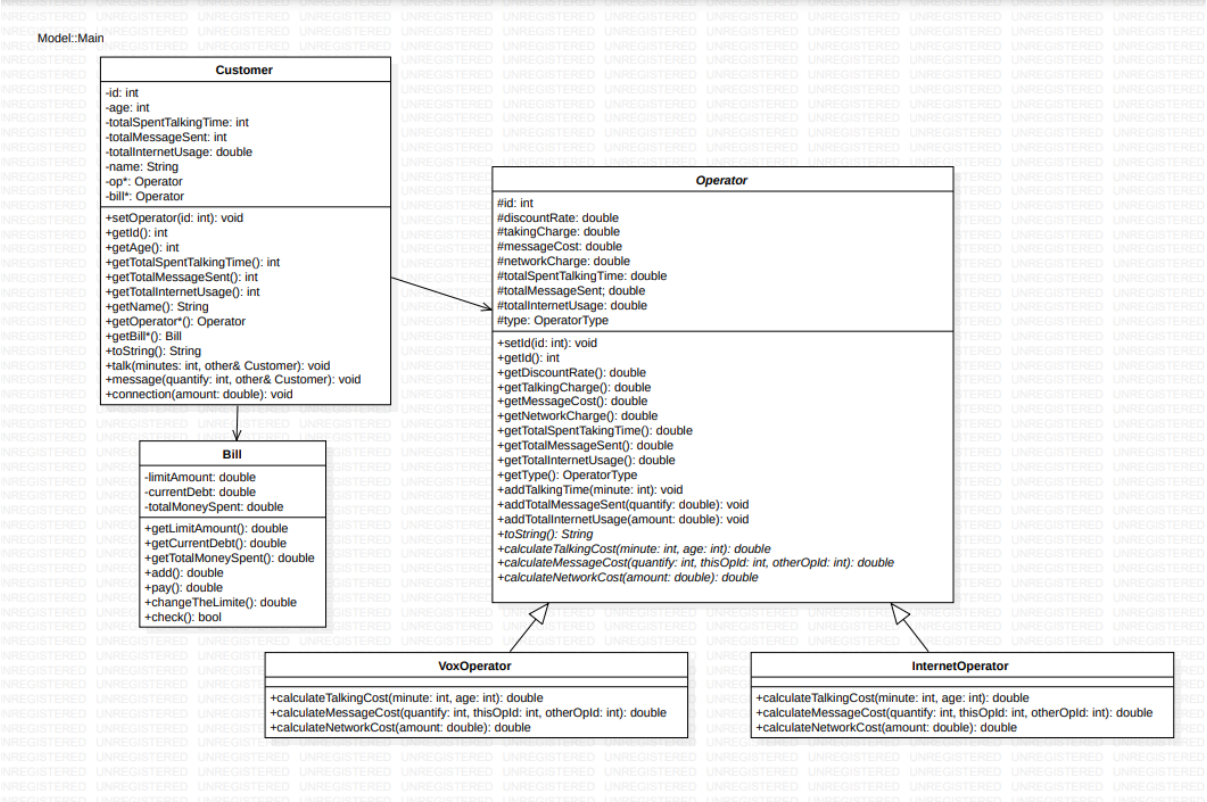
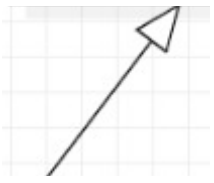


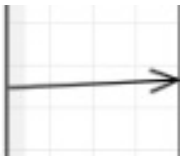
Figura 1. Muestra la implementación del proyecto mediante un diagrama UML

Como se mencionaba con anterioridad, este proyecto cuenta con 3 clases base: Customer, Bill y Operator, de la cual se derivan: InternetOperator y VoxOperator, esto se puede ver en la figura 1.

Por ende, en el diagrama se representan las clases derivadas de Operator, InternetOperator y VoxOperator con la siguiente flecha.



Por otro lado, dentro del programa se utiliza asociación dirigida, representada con la siguiente flecha:



La asociación dirigida se da entre Customer y Bill, así como entre Customer y Operator, ya que se relacionan las clases entre sí, es decir, una clase la relacionamos con otra mediante argumentos como el siguiente:

```
Operator* operator;
Bill* bill;
```

Dicho argumento se encuentra en la Clase Customer y es una manera de asociar una clase con otra.

Argumentación de las partes

Las clases se pueden ver identificadas dentro del diagrama, así como en la implementación del código, en donde se muestran las 5 clases diferentes. Como se explicaba en la parte anterior. Por otro lado, en cuanto al concepto de herencia, este se aplica cuando las clases de VoxOperator e InternetOperator reciben atributos y métodos de Operator, ya que para la aplicación de los métodos, necesitan de los atributos y métodos de Operator. Eto, nos permite reutilizar el código sin la necesidad de repetir el código. En cuanto a modificadores de acceso, se utilizaron público y private. Este último fue utilizado para todos los métodos y public fue utilizado para los atributos, para que de esta manera las demás clases pudieran acceder a ellos. Por otro lado, la sobrecarga de métodos se da principalmente en las clases VoxOperator e InternetOperator puesto a que en ambas clases se tiene que hacer la misma función, sin embargo, reciben distintos parámetros y la forma de hacerlo es distinto. Lo mismo pasa con el polimorfismo aplicado dentro de los métodos, ya que responden de distinta manera, debido a que se les envían parámetros diferentes. Por otra parte, en cuanto a las clases abstractas se implementó una clase abstracta, es decir, Operator, ya que esta era necesaria para que sus clases derivadas, mencionadas con anterioridad, pudieran implementar, de manera forzosa, los métodos que se definían en la clase Operator. Finalmente, la sobrecarga de operadores se implementó al comparar distintos objetos, en este caso, cuál de los clientes tenía mayor cantidad de internet usados, entre otros aspectos.

Identificación de casos que harían que el proyecto deje de funcionar

Probablemente, el problema más evidente es cuando uno de los archivos no estén dentro de la carpeta en donde el proyecto se encuentra, ya que si no es de esta manera, el programa no podrá funcionar adecuadamente, si bien mandará un error, en realidad no devolverá los datos requeridos. Por otro lado, si los datos no se encuentran en el formato adecuado, el programa tampoco será capaz de leerlos y por ende, tampoco arrojará los datos

esperados. Finalmente, se deben detectar aquellos casos que no se hayan validado dentro de las funciones, ya que no caerán en ningún caso, y así no se tomará la decisión adecuada dentro del programa lo cuál puede hacer que este no funcione como debería de funcionar.

Conclusión personal

Dentro del proyecto, se puede observar cómo es que diversas funciones necesitan de distintas validaciones que no permiten que el programa continúe su curso normal, ya que mandaría un error. No obstante, el pensar en todas las excepciones y programarlas resulta un tanto complicado y laborioso. Por ende, la implementación de excepciones es una buena forma de indicarle al usuario los datos que se están buscando para que pueda ingresarlos de la manera correcta y que el programa pueda funcionar adecuadamente. Este tipo de clases son una buena práctica para programas futuros.

Referencias:

GitHub - Manchas2k4/tc1030-project-com. GitHub. (2022). Retrieved 9 June 2022, from https://github.com/Manchas2k4/tc1030-project-com.git.

GitHub - Manchas2k4/tc1030-project-com. GitHub. (2022). Retrieved 9 June 2022, from https://github.com/Manchas2k4/tc1030/tree/master/project-market