Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación.

IC2101 – Programación Orientada a Objetos.

Documentación del proyecto 0.

Profesor: Mauricio Avilés Cisneros.

Estudiantes:

Tamara Nicole Rodríguez Luna – 2021077818.

Binjie Liang Liang - 2023064642

Armando Garcia Paniagua - 2020065209

01 de mayo del 2024.

Introducción.

Este proyecto se desarrolla en el IS 2024 en Programación Orientada a Objetos para poner en práctica los conocimientos adquiridos durante el curso, permitiendo con buenas prácticas de programación según el propósito, requerimientos y características específicas del proyecto la creación de un sistema en óptimas condiciones.

El funcionamiento del proyecto se basa en un sistema con interfaz gráfica para administrar una agenda de un negocio de barbería donde únicamente existe un usuario que es el barbero, por ende, se diseñó con un menú adecuado a las diferentes acciones que puede realizar; entre las opciones se encuentran administrar los servicios ofrecidos, administrar la lista de espera de clientes, establecer el horario de atención, administrar las citas y los clientes de la barbería, cada una de las funcionalidades fue programada con distintas clases, contenedores y tipos de datos, la mayoría vistos en clase y otros fruto de la investigación tanto grupal como individual de los miembros del equipo según las necesidades del proyecto.

La administración de servicios involucra la creación, lectura, actualización y eliminación de diferentes servicios que quisiera ofrecer la barbería como teñido de pelo, recorte de pelo, extensiones, decolorado, recorte de barba y demás. Esto le permitirá a la barbería llevar un mejor registro y orden de sus servicios. Además de permitir reservar citas para recibir un servicio en específico

Por otro lado, la administración de la lista de espera de clientes permite al barbero llevar un registro ordenado de clientes interesados en sus servicios cuando ya tiene la agenda llena, pero si se libera un espacio dentro de su agenda puede pedir un cliente de la lista de espera para poder ofrecerle el espacio que se liberó.

El babero podrá establecer su horario de atención por cada día de la semana en la aplicación, así se permite tener un horario flexible y poder modificarlo por conveniencia del negocio.

También se desarrolló la administración de citas para llevar una agenda digital organizada para saber cuáles espacios tiene disponible y a quién atenderá en cuál hora, asegurando que no habrá conflictos en las reservas.

Por último, está la administración de clientes para que el barbero pueda tener un registro de los datos de cada uno de ellos y poder contactarlos para confirmar o cancelar las reservas que tengan en el negocio.

Presentación y análisis del problema.

Presentación del Problema

Se desea crear un programa que le permita al barbero llevar un registro de citas para su barbería donde pueda llevar registro de los servicios que ofrece, sus clientes y las citas.

Esto incluye subproblemas como: Crear un registro de clientes donde se puedan crear, consultar, actualizar y eliminar sin afectar el buen funcionamiento del sistema; crear un registro de citas que le permita saber al barbero su disponibilidad, permita reservar espacios y liberarlos; enviar notificaciones de confirmación a los clientes para asegurar que el espacio estará ocupado; crear un registro de servicios que ofrece la barbería a los clientes donde se puedan crear, consultar, actualizar y eliminar sin afectar el buen funcionamiento del sistema; crear una lista de espera donde se puedan almacenar los clientes que desean una cita, cuando la agenda está llena para que el barbero pueda contactarlos cuando se libere un espacio; crear un horario de atención que pueda ser diferente cada día de la semana.

Para la atención de todos los subproblemas es necesario la creación de clases para un mejor manejo de los datos de clientes, servicios y citas. Además de una clase controladora que reciba instrucciones desde las vistas y sea capaz de manejar las diferentes clases para crear las instancias necesarias para la correcta representación de los datos deseados.

Además, para llevar un mejor registro de los diferentes datos de la barbería se utilizan ArrayLists o Maps como colecciones. De este modo, se garantiza una mejor organización y por ende un mejor acceso a las diferentes instancias según sea necesario. Por ejemplo: Los horarios, los clientes y las citas se manejan por medio de un Map donde tienen por llave un String que los identifica y dan acceso a la instancia deseada. Mientras que la cola de espera y los servicios funcionan usando ArrayList y cuando se necesita alguna de las instancias se hace un recorrido o se consulta por medio del índice.

Con respecto a la organización del código se dividió en paquetes, uno para toda la lógica del programa, otro para una vista general de la aplicación, luego uno para la vista de citas, después uno para la vista de clientes y por último uno para la vista de servicios.

Las vistas están hechas a base de la biblioteca Java Swing entonces el barbero podrá tener un sistema estético y fácil de usar por medio de la interfaz gráfica que le comunican a la controladora lo que desea hacer el barbero para que pueda reflejarlo en la lógica programada en el sistema.

Para poder darle al sistema una persistencia en sus datos también se hizo un guardado y carga de la información del sistema utilizando el patrón Singleton para mantener la consistencia durante la ejecución de la aplicación.

Metodología

Para empezar, se realizó el diagrama de clases y se consultó varias veces con el profesor para asegurar que el diagrama esté en buena forma para poder usar de apoyo durante el resto del desarrollo del proyecto.

Luego se creó una clase de clientes para almacenar los datos individuales de cada cliente con las validaciones para el correo electrónico y el número de teléfono utilizando expresiones regulares, después se creó la controladora con un Map de clientes que usa como llave el email del cliente y guarda al cliente como valor, además se creó un ArrayList de clientes para que funcione como la cola de espera y guarde los clientes que deseen citas fuera de la agenda.

Ya con las clases cliente y controladora, así como la implementación de la cola de espera creadas se procedió a crear las clases Horario y Servicio para poder retener los datos pertinentes en esos objetos y crear las funciones correspondientes para la administración de las instancias a cada una de ellas además de la función de envío de email para la confirmación de la cita. También en la controladora se creó un Map de horarios con un String como llave que posee el nombre del día de la semana y el horario de ese día como valor, además en la controladora se creó un ArrayList para que guarde los servicios de la barbería.

Después se creó la clase cita junto con un enum para el estado de la cita del que la clase cita depende, se inició con la relación de una lista de citas en la clase cliente, pero al final se dejó solo a únicamente una cita por cliente y los métodos de crear, modificar, consultar, eliminar y confirmar en la controladora para la clase cita. Además, en la clase controladora se agregó un Map que usa el email del usuario como llave para acceder al valor que es la cita correspondiente.

Por último, de la lógica se implementó la biblioteca Serializable para poder guardar y cargar los datos junto con el patrón Singleton para asegurar la consistencia de los mismos.

En este punto el enfoque se fijó en las vistas de cada problema a abordar, primero se creó una general donde el barbero puede acceder a todo lo demás, luego una vista de servicios, después la de clientes, seguido un panel para el barbero administrador, después una vista para la lista de espera, por último; la de administración de citas.

Análisis Critico

Analizando de manera honesta y crítica nuestro proyecto de asignación de citas, hubo varias funcionalidades que no se pudieron agregar al proyecto final de este. Esas incluyen las siguientes:

- La eliminación automática de citas que ya expiraron.
- La tabla de citas no despliega los datos necesarios.
- Al crear una cita, no se verifica si ese cliente ya estaba agendado.

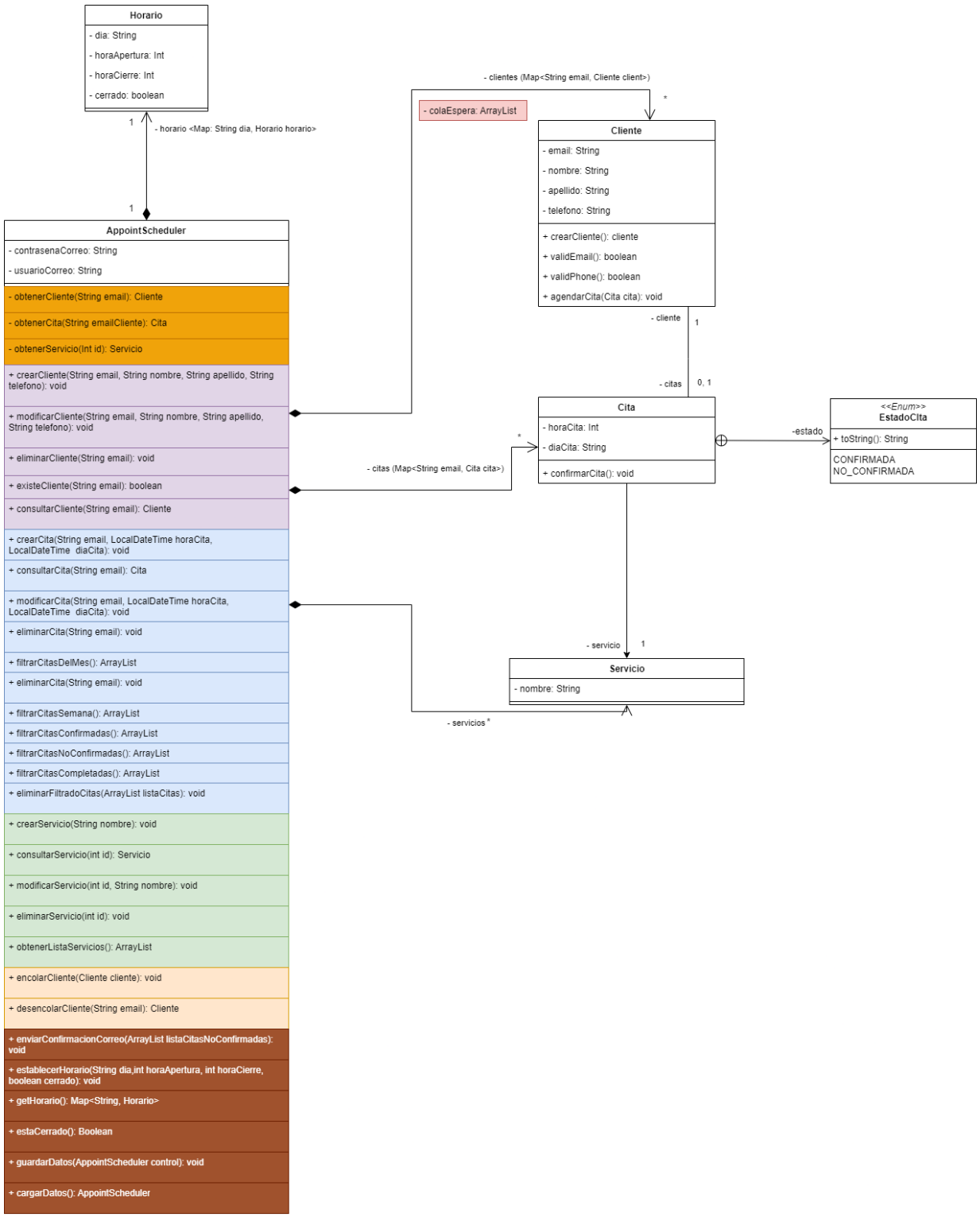
Estas funcionalidades se plantearon como "funcionalidades extra", pero no se dieron por la misma razón, que se implementarían como algo extra en el programa. Además, tenemos en cuenta que nuestro sistema, a pesar de estar completo y cumplir con la gran mayoría, sino que todas de sus funcionalidades pedidas por el profesor, pudimos haber mejorado más la eficiencia de este. Lamentablemente, no se logró debido a complicaciones como errores, bugs, entre otras razones. Si bien logramos desarrollar un sistema funcional, es importante reconocer las áreas de mejora y las limitaciones que enfrentamos durante el proceso de desarrollo.

Un área y mejora es la eliminación automática de citas expiradas requería un sistema de seguimiento y programación de tareas que no pudimos integrar de manera efectiva en nuestro código.

Otra área de mejora significativa es la interfaz de usuario y la visualización de datos. La tabla de citas no mostraba toda la información necesaria de manera clara y organizada, lo que dificultaba la gestión y el seguimiento de las citas. Aunque intentamos optimizar la presentación de los datos, nos enfrentamos a desafíos técnicos y limitaciones de tiempo que nos impidieron lograr un diseño más intuitivo y funcional.

Además, la falta de verificación de citas duplicadas para un mismo cliente fue una omisión importante. Esta característica habría mejorado la integridad de los datos y evitado conflictos y confusiones en la programación de citas. Sin embargo, debido a la complejidad de la lógica requerida y la priorización de otras tareas, no pudimos implementar esta función de manera adecuada.

Diagrama de clases detallado.



Conclusiones.

- Se concluye que JavaMail no es funcional sin incluir la dependencia de SunMail al proyecto.
- Se concluye que tener un documento con los requerimientos del programa, al igual que un diagrama de clases ayuda a agilizar en un gran porcentaje la culminación del proyecto.
- Se concluye que para los proyectos Maven, si se desea incluir librerías, no podemos simplemente descargar el JAR e incluirlo en la carpeta de librerías como pasa en proyectos Ant. Para Maven no es necesario descargar el JAR, pero si es necesario entrar al repositorio de Maven y buscar la dependencia que necesitamos, de esta forma incluirlo en el POM de nuestro proyecto.
- Se concluye que, si creas un archivo JAR en una versión de Java más actualizado, y alguien en otra computadora con una versión de Java menos actualizada, este no podrá ejecutar el JAR.
- Al ser un proyecto pequeño fue buena opción utilizar ArrayList, pero se puede volver ineficiente con una gran cantidad de datos.
- Para poder consultar más rápidamente datos que no tienen un identificador fue una buena opción utilizar Map.
- Java Swing es una buena biblioteca para tener una interfaz gráfica rápidamente, pero es muy común que desacomode los diferentes componentes de la ventana y haya que volver a acomodarlos.
- Utilizar expresiones regulares es más sencillo para validar un correo que tratar de hacerlo con funciones de procesamiento de texto.
- Usar el JCalendar como forma de obtener la fecha (día/mes/año) para asignárselas a la cita de los clientes.
- Usar el objeto JSpinner modificado para que se pueda obtener la hora asignada a la cita del cliente.

Recomendaciones

- Si se desea enviar correos por medio de JavaMail, es recomendable que se busque otros proveedores de servicio SMTP, de esta forma se evita hacerle cambios a tu cuenta de Google y además crear una contraseña para una aplicación de terceros. En nuestro caso, encontramos el servicio de MailerSend, que en su plan gratuito incluye el envió de 3000 emails al mes, lo cual es perfecto si solo se trata de implementar una aplicación pequeña.
- Si desea crear archivos ejecutables JAR, lo más recomendable es que incluya el plugin de Apache Maven Assembler en el proyecto, el cual con solo tocar el botón de build, ya te crea un archivo ejecutable en la carpeta target.
- Para que la interfaz de usuario se vea con un estilo Flat, es decir, un diseño limpio y fácil de ver, se recomienda el uso de la dependencia FlatLaf, el cual en su página oficial incluye toda la documentación de cómo usarla.
- Al hacer proyectos grupales, es importante que todos tengan las mismas versiones de cada dependencia que se necesita el proyecto. Por ejemplo, misma versión de editores de texto, mismas versiones de compiladores, mismas versiones de Java, Python, entre otros lenguajes. Esto para evitar problemas o errores en las que a uno se le hará complicado resolver, ya que hay veces que errores en la compatibilidad de versiones no indica explícitamente el error.
- Si se sabe que se va a manejar una gran cantidad de datos y se va a estar modificando el orden es mejor evitar el uso de todas las clases que heredan de List como lo es el ArrayList.
- El Map es una excelente opción para poder guardar y recuperar eficientemente diferentes instancias que puedan tener un identificador único.
- Aunque Java Swing es inmediato y tiene bastante documentación se recomienda investigar sobre otras alternativas para crear la interfaz gráfica para consumir menos tiempo en ello.
- Se recomienda investigar más a fondo sobre las expresiones regulares para tener un mejor entendimiento del tema, puesto que aparenta tener más usos que pueden ayudar en otros casos.

Referencias Bibliográficas.

Baeldung, & Baeldung. (2024, 8 enero). Sending Emails with Java | Baeldung. Baeldung. https://www.baeldung.com/java-email#:~:text=To%20format%20and%20style%20our,implement%20the%20tag

Brooks, R. (2020, 31 enero). The Power of Regular Expressions. Hall. https://www.hallme.com/blog/the-power-of-regular-expressions/#:~:text=The%20plus%20symbol%20(%20%2B%20)%3A,a%20character%20appears%200%20times

Casey, J. (2011, 7 febrero). Apache Maven Assembly Plugin – usage. https://maven.apache.org/plugins/maven-assembly-plugin/usage.html

com.sun.mail.smtp (JavaMail API documentation). (s. f.). https://javaee.github.io/javamail/docs/api/com/sun/mail/smtp/package-summary.html

Email sending service. (s. f.). MailerSend. https://www.mailersend.com/

Enviar correos con la API de correo. (s. f.). Google Cloud. https://cloud.google.com/appengine/docs/legacy/standard/java/mail/sending-mail-with-mail-api?hl=es-419

FlatLaf - Flat Look and Feel | FormDev. (s. f.). https://www.formdev.com/flatlaf/

JavaMail. (s. f.). https://javaee.github.io/javamail/

javax.mail.AuthenticationFailedException: 535-5.7.8 Username and Password not accepted. (s. f.). Stack Overflow En Español. https://es.stackoverflow.com/a/500544

John Rayworth. (2022, 18 abril). Netbeans Maven Project to JAR file [Vídeo]. YouTube. https://www.youtube.com/watch?v=P-4OH4wrQNs

Maven Repository. (s. f.). Maven Repository. https://mvnrepository.com/

Mkyong. (2019, 10 abril). JavaMail API - Sending email via Gmail SMTP example - Mkyong.com. Mkyong.com. https://mkyong.com/java/javamail-api-sending-email-via-gmail-smtp-example/

Porter, B., Laugstol, T., & Marbaise, K. H. (s. f.). Maven – Introduction to the Dependency Mechanism. https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

Working with Manifest Files: The Basics (The Java™ Tutorials > Deployment > Packaging Programs in JAR Files). (s. f.). https://docs.oracle.com/javase/tutorial/deployment/jar/manifestindex.html

Jenzri Nizar. (2013, 18 mayo). Java JSPinner Swing time [Vídeo]. YouTube. https://www.youtube.com/watch?v=Uh77miF-YMY

DevGuardian Code. (2022, 1 marzo). Setear fecha de JCalendar a componente Text Field en Java [Vídeo]. YouTube. https://www.youtube.com/watch?v=yc3sWITm4v8