

# PROYECTO FINAL

*Programación II.*



Camila Cortés Pérez // Natalia Contreras Aliaga.

Geoffrey Hecht  
Ivonne Flores

## ÍNDICE

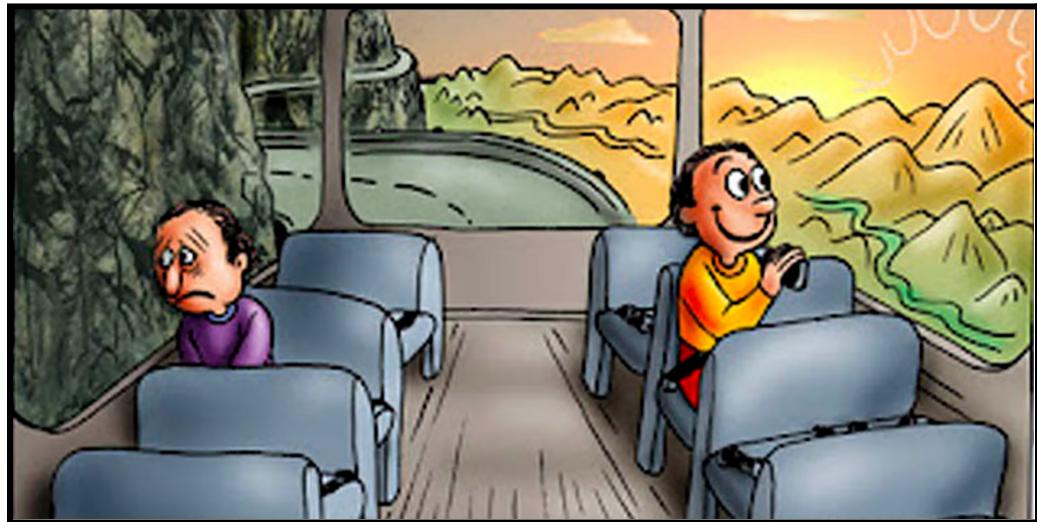
|   |    |
|---|----|
| Introducción.....                                       | 2. |
| Enunciado General (Sistema de Reserva de Asientos)..... | 3. |
| Diagramas.....  | 4. |
| Patrones Usados.....                                    | 5. |
| Interfaz.....   | 6. |
| Decisiones Técnicas.....                                | 7. |
| Problemas Encontrados.....                              | 8. |
| Conclusión.....   | 9. |

## INTRODUCCIÓN

La programación orientada a objetos es un paradigma de programación que parte del concepto de "objetos" como base, los cuales contienen información en forma de campos (a veces también referidos como atributos o propiedades) y código en forma de métodos.

Esta metodología fue estudiada a fondo durante el curso de Programación II, que ahora es implementada para nuestro proyecto a presentar "sistema de reserva de asientos de autobús".

A través de lo aprendido, implementando los conocimientos de las tareas anteriormente hechas, se crea un código que soluciona lo pedido. Además se presenta el proceso, las decisiones técnicas y las problemáticas.



## SISTEMA DE RESERVA DE ASIENTOS DE AUTOBÚS

La descripción del trabajo consiste en:

“El sistema de reserva de asientos de autobús permite al personal de una empresa de autobús elegir y reservar asientos de forma conveniente para su cliente. Los usuarios pueden visualizar una representación gráfica de los asientos disponibles en el autobús y seleccionar los que deseen ocupar. El sistema muestra información detallada sobre cada asiento, como su ubicación, número y categoría (por ejemplo, semi cama, Salón Cama).

Una vez que los usuarios seleccionan los asientos deseados, el sistema verifica la disponibilidad y permite confirmar la reserva mostrando el precio a pagar. En caso de que algún asiento ya esté reservado por otro pasajero, se informa al usuario para que pueda elegir otro asiento disponible. El personal confirma el pago (no gestionado por el sistema) lo que reserva los asientos.

El sistema debe gestionar varios tipos de autobuses (por ejemplo, con diferente número de plazas, o de 1 o 2 pisos...) y debe mostrar un menú que permita seleccionar el autobús en función de su horario y recorrido (se supone que estos datos están disponibles con los autobuses vacíos cuando se lanza el software)."

La primera versión del trabajo comentada por nosotras contemplaba comprar pasajes ida y vuelta, pero después de consultarla lo hemos reducido. También se había hablado de la primera interfaz contener los horarios, pero al final decidimos que era mejor que tuviese una propia.

Esta primera entrega contenía inicialmente 3 ventanas (inicio con selección de origen, destino y día, selección de horarios y selección de asientos), además se había contemplado que los detalles de la compra se mostrasen en la interfaz de las selecciones de asientos. Después de empezar a avanzar, se decidió mostrarlo aparte ya que así se podría simular una ventana de confirmación de compra.

Finalmente, después de mucho avanzar, nos quedamos con 5 ventanas; inicio, selección de horario, selección de asientos (dos diferentes, para semi cama y salón cama) y finalmente la confirmación de la reserva.:

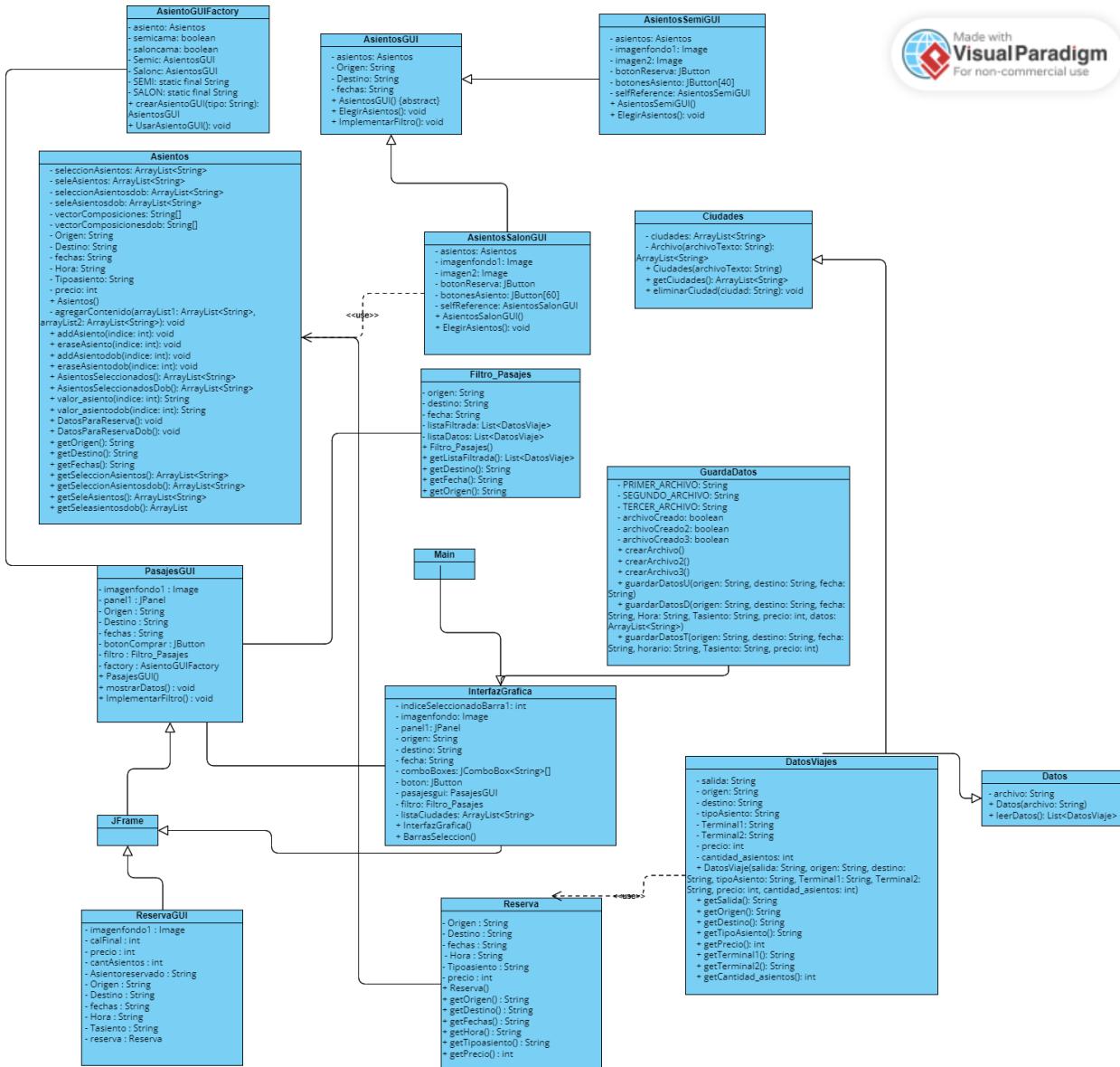
Lo más importante aquí es el uso de botones. Para hacer los asientos se decidió hacer todos los botones con un vector. Este crea una serie de botones de asiento en una interfaz gráfica. El bucle for se ejecuta dos veces: una vez para los índices del 0 al 19 y otra vez para los índices del 20 al 39. Cada iteración del bucle crea un nuevo botón de asiento, establece su posición en la ventana y le asigna un ícono.

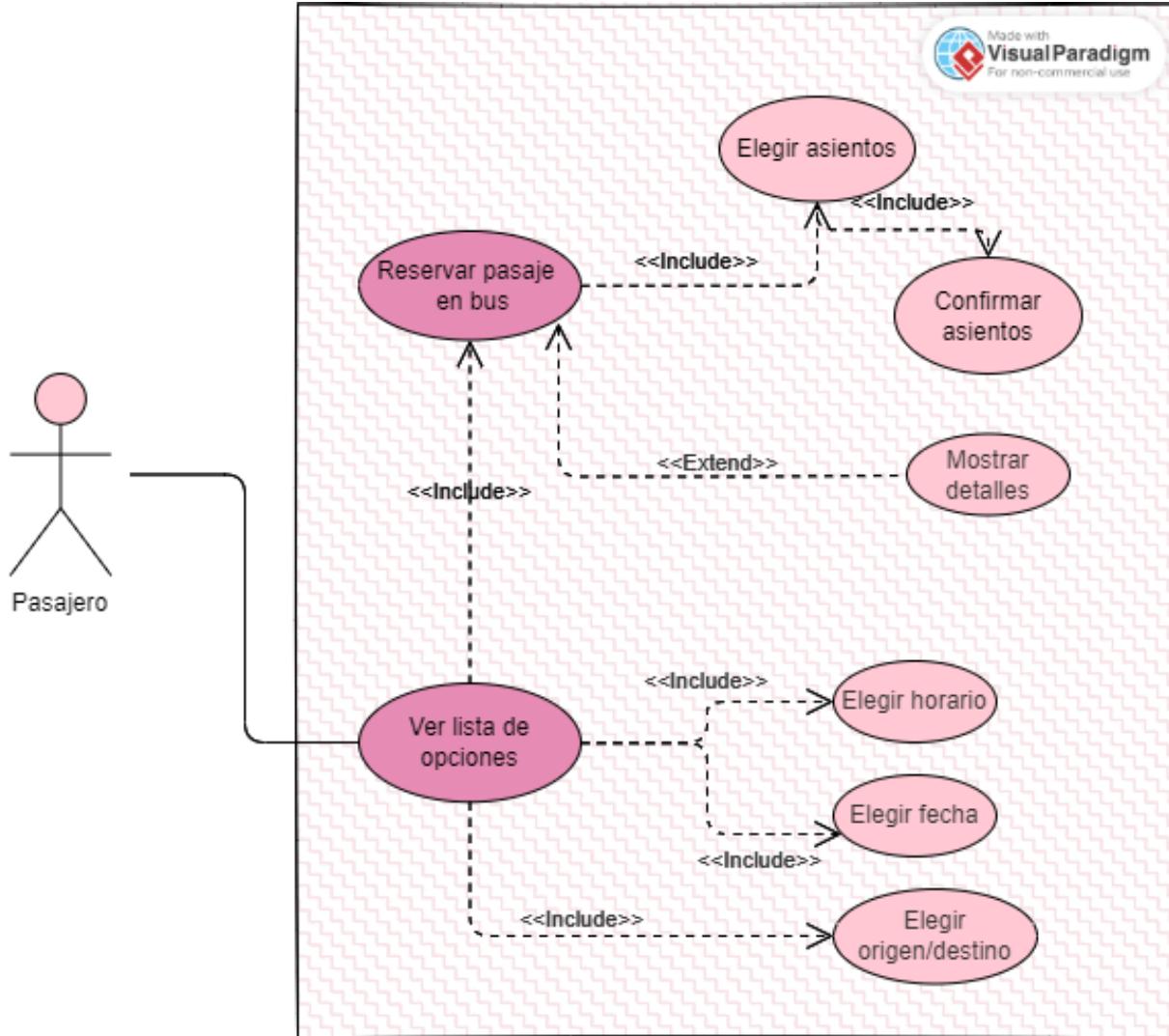
La funcionalidad del código se basa en sus clases:

- **GuardaDatos**: esta clase proporciona una manera de crear y escribir datos relacionados con pasajes y asientos en tres archivos de texto diferentes. Se utilizan métodos específicos para guardar datos en cada archivo, y se puede utilizar el método `crearArchivo()` una vez para crear el archivo inicialmente o vaciar su contenido si ya existe. Es importante tener en cuenta que la clase no proporciona métodos para leer los datos de los archivos, ya que su enfoque está en la escritura de datos.
- **InterfazGrafica**: esta interfaz gráfica permite a los usuarios seleccionar el origen, destino y fecha de un viaje en barras desplegables y luego realizar una búsqueda para mostrar los resultados en una nueva ventana `PasajesGUI`. El código está diseñado para interactuar con otros componentes del programa relacionados con el almacenamiento y procesamiento de datos.
- **Asientos**: esta clase proporciona funcionalidades para manejar la selección y reserva de asientos de autobús para los viajes, tanto para asientos de tipo "Semi cama" como "Salon cama". Tiene los métodos para agregar y eliminar asientos seleccionados, para obtener los valores de los asientos según su numeración y para obtener los datos de la reserva. Además, guarda los datos de la reserva en archivos de texto utilizando la clase `GuardaDatos`.
- **AsientosGUI**: proporciona la estructura básica para implementar una interfaz gráfica de selección de asientos para la aplicación. Las subclases que hereden de esta clase implementan el método abstracto `ElegirAsientos()` para definir la lógica y la apariencia específica para que el usuario pueda seleccionar asientos para la reserva.
- **AsientosGUIFactory**: proporciona una forma de crear y gestionar las interfaces de selección de asientos para los tipos de asiento "Semi cama" y "Salon cama". Dependiendo del tipo de asiento seleccionado, se crea la instancia adecuada y se muestra la interfaz correspondiente para que el usuario pueda elegir los asientos para la reserva.
- **AsientosSalonGUI**: proporciona una interfaz gráfica para la selección de asientos específica para el tipo "Salon cama". Permite al usuario seleccionar y reservar asientos, y luego realizar la reserva mostrando los detalles de la misma en una nueva ventana. Si todos los asientos disponibles para el tipo "Salon cama" ya han sido reservados, muestra un mensaje de advertencia y redirige al usuario a seleccionar otro viaje.
- **AsientosSemiGUI**: proporciona una interfaz gráfica para la selección de asientos específica para el tipo "Semi cama". Permite al usuario seleccionar y reservar asientos, y luego realizar la reserva mostrando los detalles de la misma en una nueva ventana. Si todos los asientos disponibles para el tipo "Semi cama" ya han sido reservados, muestra un mensaje de advertencia y redirige al usuario a seleccionar otro viaje.
- **PasajesGUI**: proporciona una interfaz gráfica para mostrar los detalles de los viajes filtrados y permitir al usuario comprar pasajes para el viaje seleccionado. También muestra los detalles del filtro seleccionado previamente. Además, utiliza `AsientoGUIFactory` para crear y utilizar la interfaz de selección de asientos correspondiente según el tipo de asiento elegido.
- **Reserva**: se utiliza para obtener los detalles de la última reserva almacenada en "DatosPasaje.txt". Una vez creada una instancia de esta clase, se pueden llamar a sus métodos para obtener los valores del origen, destino, fecha, hora, tipo de asiento y precio de la última reserva.

- ReservaGUI: es responsable de mostrar los detalles de una reserva realizada en una interfaz gráfica y ofrece opciones al usuario para realizar una nueva compra o salir de la aplicación. La información de la reserva se obtiene a través de la clase Reserva.
- Ciudades: es una utilidad para leer nombres de ciudades desde un archivo de texto, almacenarlos en una lista y proporcionar métodos para acceder a la lista de ciudades y eliminar ciudades específicas de la lista.
- Datos: define dos clases: DatosViaje y Datos, que trabajan en conjunto para leer datos desde un archivo de texto y almacenarlos en una lista de objetos DatosViaje. Permite leer datos de un archivo de texto que contiene información de viajes y almacenar esos datos en una lista de objetos DatosViaje, que representan los detalles de cada viaje.
- Filtro\_Pasajes: se encarga de leer los datos de los filtros seleccionados desde un archivo, leer los datos de los viajes disponibles desde otro archivo y luego realizar un filtrado de datos para obtener una lista de objetos DatosViaje que cumplen con los criterios de origen y destino seleccionados en el filtro.

## DIAGRAMAS





## PATRONES USADOS

En la creación del código, se utilizó el patrón Factory Method, que es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.

El patrón de diseño Factory (fábrica) se utiliza en el código para encapsular la creación de objetos y proporcionar una interfaz para crear instancias de una clase concreta, en función de los parámetros proporcionados. En este caso, la clase AsientoGUIFactory actúa como la fábrica para crear objetos de las clases AsientosSemiGUI y AsientosSalonGUI, ambas implementando la interfaz AsientosGUI.

Veamos cómo se utiliza el patrón de diseño Factory en el código:

- **Creación de instancias mediante el método crearAsientoGUI:** El método crearAsientoGUI recibe un parámetro tipo que indica el tipo de asiento que se desea crear. Si el tipo es "Semi cama", se crea una instancia de la clase AsientosSemiGUI, y si el tipo es "Salon cama", se crea una instancia de la clase AsientosSalonGUI. Si el tipo no es válido, el método retorna null.
- **Utilización de la interfaz de asientos mediante el método UsarAsientoGUI:** El método UsarAsientoGUI se utiliza para mostrar y utilizar la interfaz de asientos según el tipo de asiento seleccionado. Si el tipo es "Semi cama", se muestra la interfaz de asientos para elegir asientos de tipo "Semi cama" y se aplica un filtro para mostrar los datos relacionados. Si el usuario elige menos de 40 asientos, la interfaz se muestra y permanece visible. Si elige 40 asientos o más, la interfaz se oculta y se cierra.

De manera similar, si el tipo es "Salon cama", se muestra la interfaz de asientos para elegir asientos de tipo "Salon cama" y se aplica un filtro para mostrar los datos. Si el usuario elige menos de 60 asientos, la interfaz se muestra y permanece visible. Si elige 60 asientos o más, la interfaz se oculta y se cierra.

El patrón Factory en este caso permite encapsular la creación de instancias de las clases AsientosSemiGUI y AsientosSalonGUI, lo que facilita el mantenimiento y la extensibilidad del código. Además, proporciona una forma clara y coherente de crear y utilizar objetos de las interfaces AsientosGUI, ocultando los detalles de implementación de las clases concretas. Esto ayuda a mejorar la modularidad y facilita futuras actualizaciones o cambios en la lógica de creación de objetos de la interfaz.

## INTERFAZ



Mi compra

# Autobuses C&N

Desde
Hasta
Día

Concepción
Santiago
30-06-2023

Pasajes
→
→
→

|                    |                             |   |
|--------------------|-----------------------------|---|
| Origen: Concepción | Salida: 12:00               | <a href="#" style="border: 1px solid #ccc; padding: 2px 10px; border-radius: 5px; color: inherit; text-decoration: none;">Comprar</a> |
| Destino: Santiago  | Tipo de Asiento: Semi cama  |   |
| <br>               |                             |   |
| Origen: Concepción | Salida: 10:00               | <a href="#" style="border: 1px solid #ccc; padding: 2px 10px; border-radius: 5px; color: inherit; text-decoration: none;">Comprar</a> |
| Destino: Santiago  | Tipo de Asiento: Salon cama |   |

Asientos

**Autobuses C&N**

Asientos

Desde
Hasta
Día

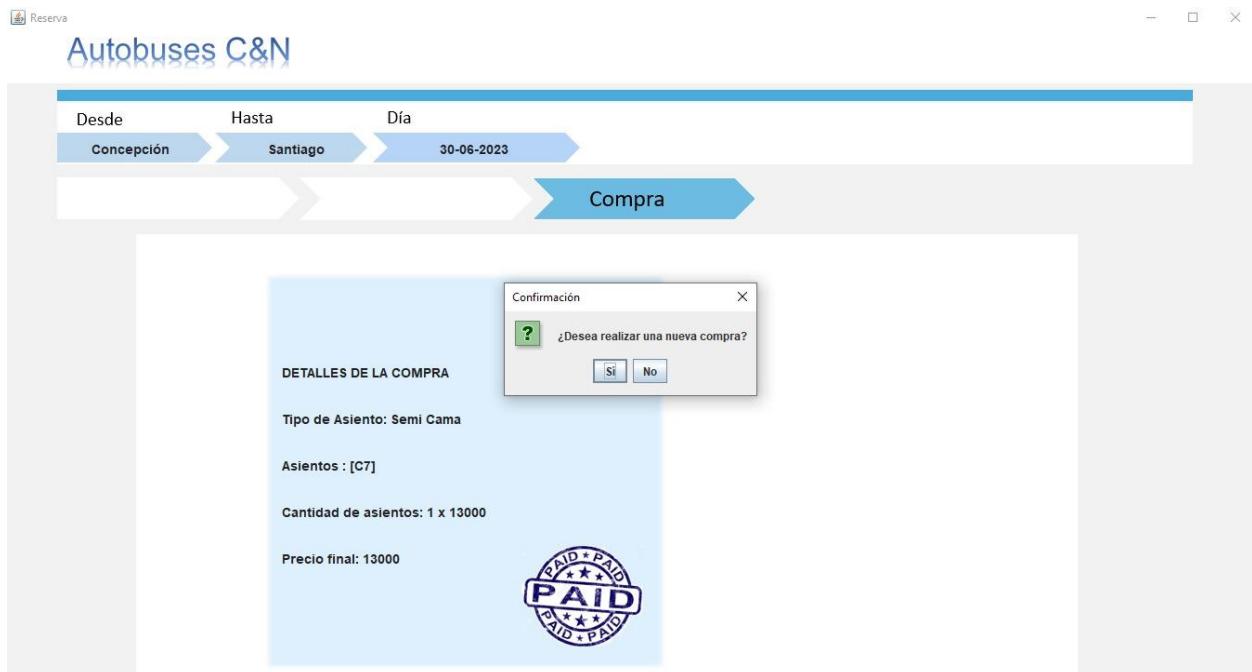
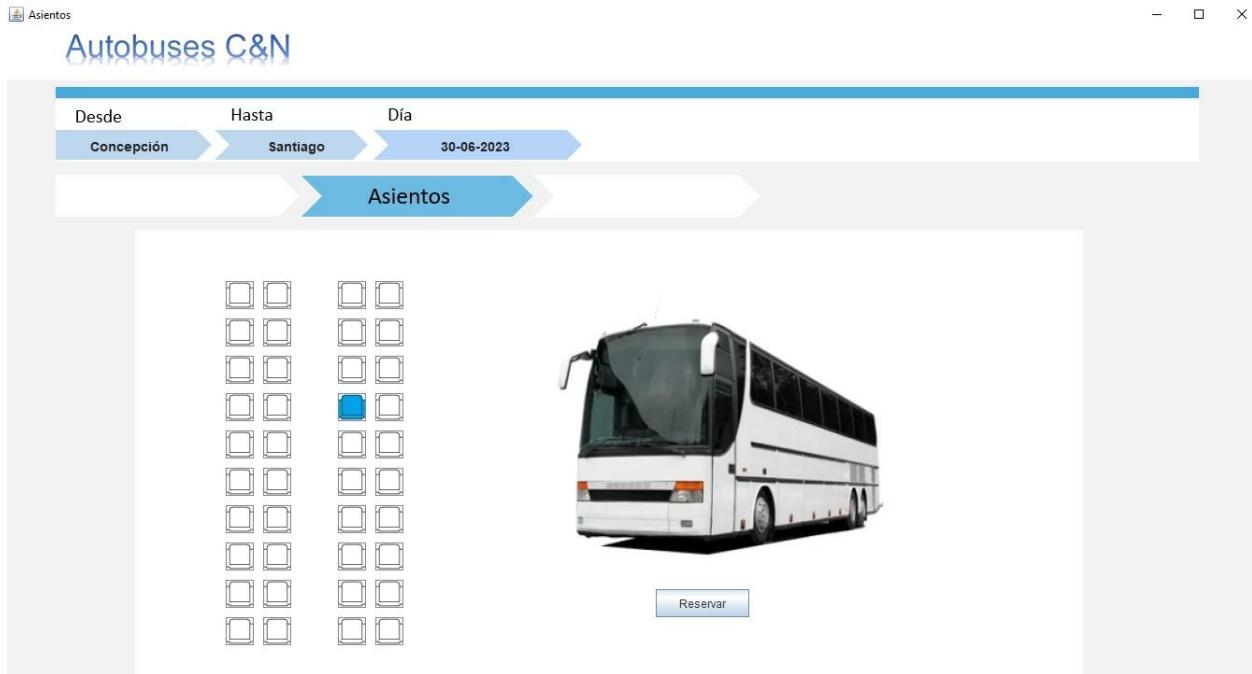
Concepción
Santiago
30-06-2023

Asientos

Piso 2
Piso 1



Reservar



## DECISIONES TÉCNICAS

Al momento de crear el código, pasamos por diferentes procesos que nos llevaron a tomar diferentes decisiones:

1. VENTANAS: Al principio se había creado una sola ventana que se creaba desde Pasaje hasta Asiento, que representaba la selección del horario que dirigía a la selección de los asientos. Para poder distinguir entre Salón Cama y Semi Cama se decidió mejor separarlo en dos clases (Asientos y AsientosDob respectivamente), lo que permitía hacer dos redirecciones y además crear libremente los dos tipos de buses.
2. BOTONES: En un principio se había intentado hacer individualmente cada botón (que representan los asientos). Debido a complicaciones cuando se intentaba guardar las interfaces, además para optimizar y agilizar el proceso, se decidió crear un vector que creará todos los botones.
3. IMAGENES: Cuando se llegó al momento de darle un sentido más gráfico a la interfaz, en un momento se intentó crear colores y formas dentro del mismo código, pero fue mucho más fácil cuando se decidió que el fondo y los botones se crearan a partir de imágenes. Por ejemplo, cuando un asiento se reserva, lo que permite que cambie de color para mostrar que está reservado es simplemente el cambio del ícono del JButton.
4. GETTERS: Necesitábamos guardar la información de la primera interfaz y que llegase a la última, en Reserva. Para no tener que crear más punteros, se decidió que todos las clases tendrán getters, así, en la siguiente clase se puede utilizar y además pasar a la siguiente clase.

En general, se mantuvieron las primeras ideas compartidas en el grupo, ya que se discutían y después se ejecutaban.

## PROBLEMAS ENCONTRADOS

Durante el proceso de creación tuvimos que lidiar con diferentes problemas:

1. **JLABELS Y ARRAYLIST:** Cuando se intentó crear el JLabel para mostrar los detalles de la compra en Reserva, hubieron diferentes problemas. Primero, el ArrayList que contiene el número de asiento seleccionado aunque estaba siendo recibido correctamente desde las clases AsientosDob y Asientos, no estaba siendo impreso correctamente en la ventana. Para esto primero verificamos que el ArrayList estuviese funcionando de forma correcta; luego, vimos que como habían dos clases, había que en verdad recibir dos ArrayList (que, aunque suene obvio, en el momento no nos habíamos dado cuenta que no se había hecho). Luego de corregir ese problema, se pudo empezar a imprimir correctamente los ArrayLists.
2. **ASIENTOS:** Para crear y reservar los asientos, obviamente nuestra primera pregunta fue cómo los podemos seleccionar. Aunque ya se había decidido hacerlo con botones, la pregunta era cómo cumplir con lo demás. El primer paso fue crearlos, pero necesitábamos poder tener un botón que al seleccionarlo pudiese reservar y si se selecciona de nuevo lo quite. Así nos dimos cuenta que nuestra respuesta era más fácil de lo que pensábamos, era un switch. A partir de esto, las ideas empezaron a surgir y surgir, hasta incluso poder darle un identificativo a cada asiento.

Aunque surgieron más problemas dentro del trabajo, eran solucionados con sólo realizar una consulta. Las dos mencionadas fueron aquellas que más impactaron en la creación del código, pero al mismo tiempo fueron base en el avance y permitieron que surgieran más soluciones a incógnitas que aún no nacían.

## CONCLUSIÓN

Como conclusión, la creación de este proyecto de “sistema de reserva de asientos de autobús” permitió al grupo poner a prueba sus conocimientos aprendidos durante el curso de Programación II. Además, se tomó como vital el trabajo en equipo, que será vital en el futuro.

También, el buen uso de github, la buena comunicación y el uso de recursos permitió que no surgiesen mayores problemas al momento de crear el código. Se tomó en conocimientos los patrones de diseños, además de aprender a fondo el uso de interfaces gráficas, los UML y el uso de github y intellij.

