Conclusiones sobre los proyectos

Grupo 1

Andres Felipe Charry

Ana María Hernández

Julian Mondragón

Diseño y Programación Orientada a Objetos

Ingeniería de Sistemas y Computación

Universidad de los Andes

Bogotá D.C.

Año 2023

Conclusiones sobres los proyectos

A continuación se presentan una serie de análisis, reflexiones y aspectos que como grupo consideramos importantes a resaltar frente a la realización de los proyectos 1, 2 y 3 en la materia de Diseño y Programación Orientada a Objetos. Todo ello surge de las observaciones planteadas al completar cada requerimiento o actividad solicitada, así como desde la percepción que adoptamos una vez finalizadas las entregas.

1. ¿Qué cosas salieron bien y qué cosas salieron mal?

a) Cosas que salieron bien

- El manejo de las vistas según el rol que desempeña el usuario: al inicio del proyecto 1 nos tomamos varios días pensando de qué forma era posible distribuir las responsabilidades y presentarlas a quien manejara el programa según su cargo: cliente, administrador local, administrador general o empleado. Esta separación fue crucial ya que guió el desarrollo del proyecto, pues no solo nos obligó a definir límites sobre qué puede o no realizar una persona que usa el sistema, también nos ayudó a entender que las aplicaciones pueden ser ejecutadas por distintos tipos de usuario según lo que se indique, lo cual fue un aspecto novedoso porque los proyectos de materias anteriores solo abarcaban un punto de vista. De esta manera fue mucho más sencillo realizar las funciones teniendo en cuenta que buscábamos asegurar que las operaciones de cada tipo de usuario fueran permitidas de forma lógica y eficiente.

- **Conexión entre lógica e interfaz:** la conexión entre la lógica y la interfaz gráfica salió naturalmente. Esto se debe a que cuando realizamos el primer proyecto, todo lo hicimos pensando en que no necesariamente las aplicaciones son realizadas siempre en consolas, a veces implican componentes gráficos. De esta manera, procuramos hacer el intercambio de datos entre la lógica y aquello que se le presenta al usuario de modo que fuese mantenible a través del tiempo por si se requería otro tipo de presentación. En resumen, la clave estuvo en separar adecuadamente la recepción de datos, el código con la solución concreta de los requerimientos y la visualización de las respuestas. Aprovechamos este recurso para la segunda entrega, e incluso pensamos que hace que el proyecto se pueda extender con mayor fluidez a futuro.

- Herencia en el registro de usuarios: creamos la clase UsuarioGenerico porque desde el principio identificamos distintos atributos propios de cualquier usuario, tal y como el login, la contraseña, el nombre y rol. Con esa superclase en mente generamos que otras clases como Cliente, AdminLocal, AdminGeneral y Empleado se extendieran de UsuarioGenerico, para luego especializarse individualmente en la recolecta de datos relevantes del tipo de usuario según la información solicitada para los requerimientos. Esto fue beneficioso porque promueve la organización del código, la reutilización, la eficiencia en el desarrollo y si deseamos ejercer algún cambio en los atributos o métodos comunes se aplicaría automáticamente a todas las subclases.

b) Cosas que salieron mal

- Clases con demasiadas líneas de código: al inicio del proyecto pensamos en realizar la carga de datos, las estructuras de datos y el manejo de la lógica de la aplicación en la clase Empresa. Más adelante notamos que esa clase estaba empezando a llenarse de distintas responsabilidades en lugar de enfocarse en una sola, como se menciona en el principio Single Responsibility (principios SOLID). Esto no fue para nada conveniente, puesto que nos perdíamos al leer nuestro propio código para modificarlo, e incluso en caso de aplicar cambios complicaba todo el manejo del sistema. Es por ello que eventualmente tuvimos que crear la clase Inventario, en la cual se distribuyó la carga de datos que tenía Empresa y también se guardaron los objetos en estructuras de datos. Todo ello no solo nos costó tiempo y esfuerzo, sino que nos hizo aprender la importancia de repartir adecuadamente los roles y responsabilidades de una clase. Ahora preferimos tener muchas clases con alta cohesión y bajo acoplamiento a pocas clases recargadas de código.

- Creación de una clase innecesaria: al crear las clases referentes a los vehículos existía VehículoAlquilado y VehículoGeneral. Con el tiempo nos dimos cuenta que VehículoGeneral, a pesar de tener los atributos de un vehículo completo, no iba a tener métodos porque en ningún momento se trabaja con un vehículo sin estado. En otras palabras, los carros están o no alquilados, nunca son generales o sin estado. Así, tuvimos que corregir el error y crear VehículoNoAlquilado. El no prever esta situación casi genera que no pudiéramos mandar a tiempo el proyecto uno, porque nos tardamos horas corrigiendo. Afortunadamente logramos mandar la entrega completa; sin embargo, nos tomó mucho trabajo bajo presión. Sabemos que no está mal reevaluar la arquitectura de clases, pero tendremos mayor cuidado en el momento del diseño y haremos planificaciones cuidadosas para evitar estos incidentes durante el desarrollo.

- Primeros pasos en el proyecto: al comenzar el desarrollo entendíamos el concepto de clase pero nunca lo habíamos aplicado. Es por ello que creamos HashMap para guardar los atributos de las cosas, donde las llaves eran el nombre de los atributos, como “nombreCliente” o “edad” y los valores eran las asignaciones de dichas características. Luego nos dimos cuenta que era mala idea porque complicaba la manera de trabajar en todo el proyecto. Justo al investigar como hacer el código de clases y generar instancias de ellas vimos que la programación orientada a objetos es una herramienta poderosa, porque se vale de la reutilización para producir programas mejor estructurados y da una imagen más clara de cómo manejar proyectos complejos al poderlos pensar en objetos casi como si fueran del mundo real.

2. ¿Qué decisiones resultaron acertadas y qué decisiones fueron problemáticas?

a) Decisiones acertadas

- **Casteo a String de datos tipo LocalDate:** durante el desarrollo de la lógica siempre tuvimos en consideración que existía la posibilidad de ir más allá de la consola, y empezar a realizar nuestras propias interfaces gráficas. Esto hizo que, aún sin conocer el tema —pues estábamos entregando el proyecto 1— intentáramos pensar en qué tipos de datos serían

útiles al momento de conectar el código fuente con el código de la GUI. Así, tomamos una decisión: castear datos de tipo LocalDate y LocalTime a String, por si eventualmente lo llegábamos a necesitar para realizar avisos, etiquetas o simplemente para imprimir de manera más cómoda. Esto nos ayudó más adelante en la realización del proyecto 2, porque varias veces necesitamos que la interfaz mostrara distintos JComboBox o JButton de Strings (no funcionaba con LocalDate), e incluso en el proyecto 3 al generar archivos pdf con las facturas, los cuales debido a la librería, solo permiten mostrar datos de tipo String.

- Crear la clase inventario: esta clase facilitó absolutamente todo, porque al separar las responsabilidades de la Empresa (controller) con las de Inventario (donde se leyeron los archivos y cargaron en estructuras de datos), la gestión de datos fue un proceso exitoso.

- Dibujar la interfaz gráfica de la forma más realista posible: tener una imagen de referencia sobre cómo deseábamos que quedara el resultado final de la interfaz fue una idea excelente. Adicionalmente procuramos hacer un dibujo lo más similar posible a lo que sería una interfaz hecha con Swing, entendiendo en qué formato se iban a mostrar las opciones a los diferentes usuarios (JComboBox, JList, JButton, JCalendar). Gracias a esto, cuando empezamos a utilizar el framework fue mucho más sencillo identificar los componentes que necesitábamos implementar, y la forma en que debían ser ordenados en los Layout que se ofrecen. Ahorramos tiempo porque ya conocíamos los colores de los elementos, el tipo de fuente, la estructuración y solo restaba comenzar a trabajar en el desarrollo.

- Distribuir el trabajo de interfaz gráfica y lógica en equipo: trabajar en equipo pero asignándole diferentes requerimientos fue importante. En diversas ocasiones nos vimos enfrentados a tener que integrar los componentes que cada uno de nosotros realizó: lógica, interfaz, errores, consola. Este proceso se mantuvo eficaz y fructífero todo el tiempo, dado a la comunicación sólida y constante entre nosotros. Tanto así que podíamos consultar dudas y darnos consejos sobre estrategias para realizar las partes del trabajo, lo cual hizo que cada proyecto saliera como esperábamos. De igual manera, al sustentar cada uno de nosotros estaba informado sobre cómo se hizo el proyecto gracias a ello.

- Sistema de permisos y roles: decidir tener una manejador de sesiones que definiera las acciones de los tipos de usuarios facilitó la implementación de los cambios exigidos en el proyecto tres. Definitivamente esta decisión fue elemental para el mantenimiento a largo plazo y la simplificación de los cambios, sin perder la eficiencia que tenía nuestro programa.

- Crear diferentes paquetes para las clases: ordenar el programa en paquetes como Interfaces, InterfazGrafica, Programa, Usuarios y Aplicacion fue clave para no perdernos en el desarrollo de una aplicación tan extensa y compleja como esta. También pudimos clasificar en qué parte del proyecto realizaríamos los cambios del proyecto 2 y el proyecto 3.

b) Decisiones problemáticas

- Falta de identificación de casos de herencia: durante el desarrollo del código y en la finalización de cada proyecto nos dimos cuenta que siempre hubo momentos en los cuales era posible emplear herencia para ahorrar tiempo y ordenar mejor las clases. A modo de

ilustración, durante la construcción de la interfaz gráfica se repitió mucho el uso de colores, estructuras y elementos. Ello se podía solucionar con clases de elementos comunes como paneles de solicitar datos o ubicaciones de botones y etiquetas, haciendo un trabajo más limpio y menos repetitivo, facilitando las actualizaciones, como es en el caso del proyecto tres.

- Falta de validaciones rigurosas en la entrada de datos en proyectos 1 y 2: si se ingresaba mal un dato, hacía que todo parara. A pesar de no conocer cómo hacer manejo de errores o diseño de pruebas en ese instante, estábamos en capacidad de comprobar si una placa tiene tres letras seguidas de tres números. Igualmente nos hizo falta un ciclo, como un while, que ayudara a que el menú no se interrumpiera cada vez que el usuario realizara una acción, mejorando la experiencia de uso del sistema.

- Mala distribución del tiempo: tomamos la decisión de dedicar poco tiempo a los proyectos cuando quedaban semanas para entregarlos, por lo cual en la semana de entrega estábamos corriendo con los requerimientos. Esto sumado a los fallos de diseño, las reevaluaciones y los cambios implementados hicieron que en repetidas situaciones tuviéramos que trabajar con rapidez, o tener dudas sobre el enunciado del proyecto y no poderlas solucionar por falta de tiempo. En próximos proyectos nos gustaría empezar con anticipación y calma para dar trabajos de mayor calidad.

- Mejor uso de las clases para cargar y leer datos de archivos: aún si las clases Empresa e Inventario fueron útiles e hicieron que el código estuviera ordenado, sentimos que con una tercera y cuarta clase las cosas hubieran sido más sencillas. Así, Empresa haría de controller, Inventario de information holder al tener las estructuras de datos con objetos; otra clase, como Reader hubiese sido la encargada de la lectura de archivos, y la clase Writer hubiese sido la encargada de la persistencia de los archivos de texto. Tal vez Reader y Writer se hubiesen podido fusionar para crear una tercera clase únicamente. En resumen, con clases auxiliares, el manejo de archivos se nos hubiera hecho más ameno y mantenible.

¿Qué tipo de problemas tuvieron durante el desarrollo de los proyectos y a qué se debieron?

En este punto mencionaremos algunas dificultades presentadas dado al desconocimiento de tecnologías o las dificultades de diseñar en un entorno incierto.

- No conocíamos la manera indicada de importar una librería lo cual nos hizo demorar casi dos días completos en entender de qué forma se podía usar Apache PDFBox sin que se siguiera indicando un error en la importación. De hecho, también fue confuso elegir una librería porque es la primera vez que usábamos una en Java. Este problema fue netamente por la falta de información, por lo cual estuvimos por horas mirando tutoriales, buscando páginas web con instrucciones sobre cómo importar librerías en el entorno de desarrollo Eclipse y leyendo la documentación existente.

- En el análisis del dominio no identificamos que los estados de un vehículo se podían manejar como un enumerate, lo cual reduce la posibilidad de cometer errores y es eficiente en la ejecución. Esto lo terminamos implementando gracias a la recomendación de nuestros

monitores, porque no conocíamos qué era un enumerate, ni la manera indicada de implementarlo.

- Otro problema por desconocimiento de la tecnología es que no sabíamos debuggear en Eclipse, de hecho nos sigue pareciendo un poco complicado. Sentimos que esto nos hubiese podido ayudar a identificar dónde está el fallo de los errores que hemos cometido en el desarrollo, solucionándolos con mayor rapidez.

- Un conflicto generado por no entender completamente el dominio del problema es que ignoramos la posibilidad de que un empleado pudiese reservar un carro. De hecho, leyendo la guía hay muchas cosas que no están incluidas y que se exigen en la rúbrica. Aprendimos que es importante ver ambas para entender completamente el contexto en el cual estamos solucionando un problema. Estas situaciones eventualmente podrían llevarnos a malentendidos si no las manejamos a tiempo.

- Una situación problemática fue empezar a pensar hasta qué punto se cometen errores en un programa y cuáles son razonables de evitar, dado a que nunca antes nos habíamos enfocado en un proceso como ese. No obstante, entendemos la importancia de este proceso y los recursos de la materia fueron un buen punto de partida para iniciar.

Conclusiones y recomendaciones finales

Esta última sección va dirigida a personas que deseen leer recomendaciones para este tipo de proyecto, adicional a algunas conclusiones que nos llevamos de los trabajos.

- Es fundamental entender el componente teórico del curso para poder poner en práctica los conceptos durante el desarrollo de los proyectos. Otras buenas ideas son extender los conocimientos en librerías útiles para Java y patrones de diseño, porque entender y conocerlos probablemente nos hubiera ayudado varias veces, aún teniendo en cuenta sus ventajas y desventajas.

- Entender el dominio del problema, los actores y los requerimientos tanto funcionales como no funcionales. En este curso aprendimos que el sistema es empleado por diferentes tipos de usuario, con distintos roles, fines y necesidades que se deben satisfacer.

- Es fundamental comenzar a trabajar en código que no solo sea útil y cumpla sus objetivos, sino que también sea extensible, mantenible y esté organizado.

- Anticiparse a errores comunes cometidos por el usuario es tan importante como cumplir con los requerimientos solicitados en el programa, porque brinda una mejor experiencia a quien usa la aplicación y nos hace entender que no siempre el caso esperado es el que sucede en la realidad.

- Hay muchísimas maneras de presentar la información al usuario, no solo la consola que solíamos manejar anteriormente. Una interfaz gráfica también es posible, y hay distintos frameworks que resultan de provecho para realizarla. Sería positivo aprender más sobre cuáles están disponibles y sus diferentes usos en los programas.

- Diseñar pruebas e implementarlas hace que contemplemos los fallos de nuestro programa, con el fin de poder corregirlos y que cada vez estemos desarrollando un software de mayor calidad.

- Por último, recomendamos la práctica contínua de los temas aprendidos en el curso, tanto para quienes deseen empezar a programar en java como para nosotros como estudiantes, ya que ayudan a hacer código de la forma más clara, correcta y ordenada posible, pensando en futuros cambios aplicados a programas complejos.