Proyecto Final

Aplicaciones Reales Orientadas a Objetos:

Casino *The Paradise*

Equipo 2

Calderón Jiménez, David
Hernández Olvera, Humberto Ignacio
Ordiales Caballero, Iñaky

5 de febrero del 2021.

Objetivos

Objetivos generales del proyecto

- Que el alumno ponga en práctica todos los conceptos vistos a lo largo del curso, en el desarrollo de una aplicación real.
- Que el alumno fortalezca sus habilidades la programación orientada a objetos.
- Que el alumno fortalezca sus habilidades de trabajo en equipo.

Objetivos particulares opción Casino

Entre las posibles opciones para la elaboración de este proyecto final se ha optado por desarrollar el proyecto del ***Casino***, por lo que el objetivo propio de éste es elaborar un programa en donde se puedan apostar fichas en diferentes juegos de azar. Y como parte del desarrollo se deberá cumplir con los requerimientos establecidos por el cliente y definidos en el documento llamado Casino.

Objetivos personales

Nuestro objetivo personal para este último proyecto de la materia de Programación Orientada a Objetos es proporcionar al usuario la oportunidad de pasar un tiempo agradable disfrutando de los juegos tradicionales de un casino. Esto desde la comodidad de su hogar, pero sin dejar de sentir las emociones de las apuestas y la adrenalina de poder perderlo o ganarlo todo.

Desde un punto de vista más técnico y formativo. Nuestro objetivo para el proyecto es demostrar los conocimientos obtenidos durante el semestre y mostrarlos aplicados en un programa lo más real y cercano al mercado posible. Por esto se decidió hacerlo con interfaces gráficas para dar la imagen y sensación de un desarrollo profesional de una aplicación completa.

Introducción

A lo largo de todo el semestre de la materia de Programación Orientada a Objetos revisamos el paradigma orientado a objetos. En el estudiamos las propiedades básicas (*Modularidad y Abstracción*), las propiedades esenciales (*Encapsulación, Herencia y Polimorfismo*) y las propiedades de diseño (*Acoplamiento y Cohesión*). En cada tema se vieron los conceptos e ideas importantes tanto del paradigma como del lenguaje Java. Pero además del paradigma se vieron otro temas igualmente importantes. Se revisó la programación de hilos (paralelismo), el uso de excepciones, el manejo de archivo, el lenguaje UML, y los patrones de diseño.

En este proyecto final se unirán todos los temas vistos y mencionados anteriormente para crear una aplicación real que demuestre todos los conocimientos obtenidos en la materia. Nosotros decidimos escoger el desarrollar la aplicación para un casino, el cual debido a la situación mundial de la pandemia Covid-19 ha visto un decremento de sus clientes, razón por la cual decidió encargar el desarrollo de la aplicación de su casino.

Figura 1: Interfaz Inicial

Nosotros decidimos escoger esta opción porque pensamos que era la más interesante y divertida, pero porque también consideramos que era la que podríamos desarrollar de una mejor manera. Algo que desde un inicio consideramos esencial para el desarrollo fue el usar interfaces gráficas para la interacción con el usuario, en lugar de programar todo para que se desplegara en consola. Esto nos permitió darle toda una temática más real al proyecto desde crear un nombre (*Casino The Paradise*), imagen y logo para el casino, hasta que el usuario se pudiera desplazar con el mouse y haciendo click en los botones en lugar de ingresar opciones numéricas a la consola. El resultado final del programa fue una aplicación ejecutable que realmente cumple con el nombre del proyecto: ***Aplicaciones Reales Orientadas a Objetos.***

Análisis

En esta sección se realiza un análisis general del programa y de cómo se aplicaron los conceptos del curso en la solución realizada. Pero antes de pasar a hablar directamente de los conceptos del curso en la solución, mencionaremos lo que hace el programa. El programa que decidimos llamar ***Casino The Paradise*** es un simulador de un casino virtual, donde se pueden registrar usuarios y apostar en algunos de sus juegos de casino favoritos de forma virtual. El usuario además de acceder a los juegos de apuesta puede ver su información, comprar fichas y actualizar su rango de membresía que tiene diferentes beneficios. Por el lado de los técnicos y administradores del casino, el programa le permite conocer las estadísticas generales del casino, así como editar jugadores (cambiar datos o borrarlos). De esta manera el programa funciona como una aplicación de un casino a la cual sus usuarios pueden ingresar en diferentes instantes y disfrutarla.

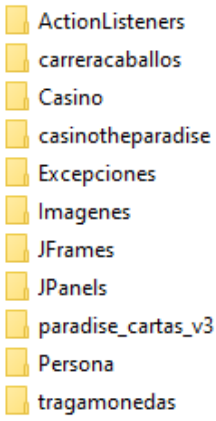
Ahora bien, para el desarrollo de un programa ya medianamente completo y real como éste, se usaron todos los conceptos revisados en el curso bajo el paradigma orientado a objetos. El desarrollo del proyecto nos permitió afianzar los conceptos propios del paradigma, ya que tuvimos que abstraer los diferentes elementos y pasar los mensajes entre el programa usando los objetos. Si bien en muchos momentos nos vimos tentados por crear instancias y métodos estáticas, decidimos evitarlo en la medida de los posible para no irnos hacia el paradigma estructurado. A continuación se describe la forma en que usamos los conceptos en la solución del programa del casino.

Colecciones

Las colecciones en el paradigma orientado a objetos nos permiten almacenar objetos de un mismo tipo en una sola instancia de tipo colección. Esto le da más orden y sencillez a la estructura del programa cuando se trabajan con varias instancias de una misma clase. Además de que nos ayudan a crear ciertas funcionalidades específicas. En el lenguaje de programación Java, se nos proporciona con diferentes clases que son las colecciones disponibles. En el caso de éste programa en particular, usamos las colecciones para almacenar las cartas de ambos juegos poker y blackjack en varios ArrayList. Esto en varias ocasiones para simular el mazo del dealer, las cartas en la mano del jugador y las cartas en la mesa. Decidimos utilizar ArrayList porque nos proporcionaba la forma más ordenada y sencilla para el manejo de cartas, aunque también pudimos utilizar una LinkedList u otra colección. Otra parte del programa donde no usamos una clase colección explícitamente, pero sí la lógica de se estructura fue en el manejo y creación de usuarios. Ya que simulando un mapa hash donde no se pueden repetir claves, así funcionaba nuestro registro de archivos donde no se podían repetir nombres de usuario. Además de igual forma se buscaba en todo el "mapa.[el] archivo de la clave deseada y si existía la regresaba. Aunque finalmente no se usó la clase hashmap porque no queríamos tener cargados en memoria a todos los usuarios o sus claves.

Uso de paquetes

No sólo en la programación orientada a objetos se implementa el uso de paquetes, pero en éste paradigma cobra todavía más relevancia debido a la característica esencial de abstraer los conceptos y elementos usados. El uso de paquetes trata de dividir en carpetas (paquetes) de manera lógica todas las clases utilizadas en un programa. De esta forma se le da más orden y limpieza, además de fomentar la reutilización de paquetes y código. Todas las clases de Java están organizadas en paquetes lógicos. En nuestro programa también decidimos llevar esto a cabo de la siguiente manera:



Uso de paquetes.

A pesar de que pueden parecer demasiadas carpetas, hay que recordar que el uso de paquetes se trata de organizar de manera lógica las clases utilizada y no sólo de compactarlas o esconderlas. Es por eso que en muchas ocasiones como nuestras clases no tenían la suficiente relación lógica entre ellas, se decidió dejarlas en paquetes separados y no juntarlas. En resumen se creo un paquete para cada uno de los cuatro juegos, para los JPanels usados, para los JFrames usados, para las imagenes de las interfaces, para las personas (clientes, administradores), para el casino, para las excepciones y para el ActionListener. De esta forma se intentó darle un orden al programa para ser código más entendible.

Modificadores de acceso

Los modificadores de acceso son una parte esencial del encapsulamiento del paradigma. Esto trata de evitar que la información sea visible y accesible desde todo el programa, lo que la vuelve muy vulnerable. Por esto el usar los modificadores de acceso (public, friendly, protected y private en lenguaje Java) es algo esencial en cualquier programa. En este programa del casino, no somos la excepción, por lo que a lo largo de todas las clases estuvimos trabajando con los modificadores de acceso para restringir la visibilidad y modificación de los datos. Algo importante de recordar es que los modificadores de acceso tienen que ver con la herencia y los paquetes para delimitar su alcance. Se utilizan los modificadores *private* para evitar que se modifiquen los valores desde otro lado del proyecto, con algunas pequeñas excepciones, y *public* para tener acceso a estos métodos o variables desde afuera de sus paquetes. En el caso de todas las interfaces JFrames y JPanels sus atributos eran privados para que nadie pudiera cambiarlos desde fuera del programa, a diferencia de la clase Deteccion que heredaba de ActionListener y que necesitábamos que diera acceso a sus atributos a todas las demás clases para controlar el flujo del programa como se explica más adelante en la sección de Desarrollo. Y como estos hay muchos más ejemplos del uso de los modificadores de acceso en las clases de Persona y de los juegos. Lo último que comentar del tema es que una manera de trabajar con modificadores de acceso restrictivos es mediante los métodos de acceso getters y setters. Al ponerlos en public, no importa si el atributo es private, a través de ellos se puede modificar.

Herencia y Polimorfismo

La herencia y el polimorfismo es otra de las bases del paradigma orientado a objetos. La herencia es el poder derivar clases de una ya existente, las clases derivadas conservaran sus atributos y métodos además de que se les podrán agregar nuevos atributos y métodos o sobre escribir los ya existentes. Por otro lado el polimorfismo consiste en que una instancia puede comportarse de diferentes formas usando los métodos de las diferentes clases que heredan de la clase padre. En nuestro programa se utilizaron los dos conceptos para la construcción de la solución. La Herencia fue la que más usamos, ya que todos las interfaces gráficas que utilizamos heredan de las clases correspondientes de los componentes de java swing. Las que más usamos fueron JPanels para todos los paneles mostrados y JFrames para las diferentes ventanas/cuadros que se utilizaron. Las clases que elaboramos heredaban de las de Java Swing para obtener los comportamientos gráficos y los métodos que facilitan estos, sin embargo nosotros les agregamos más atributos y métodos para hacer que funcionaran como necesitábamos. Otro ejemplo del uso de herencia es en las clases Cliente y Administrador que heredan de la clase persona, ya que ambas tienen los mismos atributos de una persona al utilizar la herencia no tenemos que reescribir todo esto de nuevo.
Para el caso del polimorfismo éste fue menos utilizado, pero forma una parte esencial del funcionamiento gráfico del programa. Para alternar entre los diferentes paneles que se mostraban en la ventana actual, se agregó el método cambiarPanel(JPanel panel) a nuestras clases JFrames. Gracias al polimorfismo, la función recibía una instancia de cualquier clase que heredara de JPanel y funcionaba correctamente. Con esto nos ahorramos sobrecargar el método para los más de 15 clases hijo de JPanel que se usaron en el desarrollo.

Clases Abstractas e Interfaces

Las clases abstractas no son propiamente una característica del paradigma, pero sí vienen con las características esenciales. Las interfaces a diferencia de la herencia se implementan, y lo que se busca con ellas es darle tanto implementaciones específicas como forzar a que se sobre escriban ciertos métodos en la clase que las implemente. Lo que se logra con ellas es una uniformidad entre todas las clases que implementen una interfaz, teniendo siempre algunos métodos específicos definidos en la interfaz al igual que algunos valores.
Las clases abstractas son clases que se declaran desde un inicio como abstractas y pueden contener tanto métodos abstractos como no abstractos. Este tipo de clases no pueden ser instanciadas, pero sí pueden servir como clases padres de otras. De esta forma se usa como una clase padre, pero de la cual no se pueden crear objetos, tal ve porque sean conceptos que sólo viven en la abstracción y no como objetos reales u otras razones.

A lo largo de nuestro proyecto usamos un par de interfaces para lograr una funcionalidad y forma uniforme entre nuestros componentes gráficos. Todas las clases que creamos y heredan de JFrame, también implementan la interfaz CuadroStandard que les proporciona unos atributos predefinidos y los fuerza a definir los métodos funcionales como cambiarPanel() esconder() o desplegar(). Por su parte los JPanels también implementan una interfaz para agregarles las medidas específicas y comunes en todas las interfaces, así como forzarlas a definir métodos importantes. Además se implementó repetidamente la interfaz Serializable para el uso de archivo objeto. En nuestro caso casi no usamos las clases abstractas, porque la mayoría de nuestro programa era sobre cosas concretas y no ideas o conceptos.

Uso de Excepciones

A lo largo del programa se utilizaron las excepciones tanto verificadas como no verificadas para robustecer el programa. Las excepciones son fallos manejables que se sabe pueden llegar a surgir durante la ejecución del programa, pero que de ser tratadas se evita el cierre o finalización abrupta de la aplicación. Como vimos a lo largo del semestre las excepciones verificadas son registradas en compilación y deben ser tratadas desde un inicio para que el programa compile correctamente. En nuestro caso al estar trabajando con archivos para crear una especie de "base de datos"de los usuarios, se tuvieron que tratar con las excepciones verificadas propias del manejo de archivos. Estas son las de la clase IO, además de la excepción de la lectura de archivos de objetos ClassNotFound. Todas estas se manejaron directamente en un bloque try catch donde botaban.
Las excepciones no verificadas son las que el compilador no detecta y dependen del programador para su prevención o eventual manejo. Nosotros evitamos muchas de éstas al restringir la forma en que los usuarios ingresaban la información, pero también creamos nuestras excepciones propias para el manejo de la información que ingresaban. En los métodos setters se checaban ciertas condiciones (que el nombre tuviera un largo coherente, no estuviera repetidos, ningun dato estuviera vacio, etc.) y de no cumplirse se lanzaba una excepción, esta no era manejada dentro del setter sino que se propagaba y ya en el panel de registro se manejaba mediante un bloque try catch. De esta forma podíamos indicarle al usuario de su error y controlarlo para que siguiera el programa. A continuación se muestran nuestras excepciones propias:

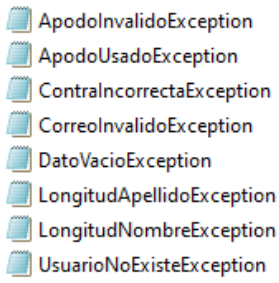


Figura 2: Excepciones propias del programa.

Manejo de Archivos

Finalmente, como ya se ha mencionado en múltiples ocasiones del análisis (y se hablará más en el desarrollo). El manejo de archivos es una parte muy importante de nuestro programa y de la gran mayoría de programas en general. Los archivos te permiten almacenar información que no necesite se creada en cada ejecución, sino que se crea en una y se guarda y se puede volver a acceder en otra. Esto nos da muchas más posibilidades para nuestros desarrollos. En nuestro caso el manejo de archivos es esencial para almacenar y llevar un registro sobre los usuarios, administradores y hasta sobre los datos propios del casino.

Nuestra forma de implementar el manejo de archivos fue a través de una clase auxiliar llamada ManejoArchivos la cual tenía los métodos necesarios para buscar, leer y escribir en un archivo de tipo objetos para nuestros usuarios y administradores. Lo que nos permitió este manejo de archivos fue el llevar el registro de jugadores y administradores, de tal forma que no se tenían que crear o ingresar sus datos con cada ejecución del programa, sino que lo hacían la primera vez y ya las siguientes podían ingresar con todo y sus datos guardados.

Con esto concluye el análisis de la implementación de los conceptos revisados en el curso y su uso en la solución de nuestro programa. En la siguiente sección se hablará y revisará detalladamente la implementación de dicha solución.

Desarrollo

En esta parte del reporte se hablará sobre el funcionamiento del programa, toda la parte lógica del flujo y diseño de la solución creada para la aplicación del casino. Lo primero que se decidió hacer para la solución fue establecer qué funcionalidades tendría el programa. Estas en buena parte fueron dadas por los requerimientos del cliente, pero además hubo que agregar detalles y hacer elecciones propias. La primera elección fue qué juegos de azar se incluirían, en un inicio se pensó en seis posibilidades: Çaballos", "BlackJack", "Poker", "Tragamonedas", "Boxeoz Ruleta". Sin embargo por cuestiones de tiempo finalmente se decidió eliminar boxeo y ruleta, dejándolos como futuras áreas de desarrollo del proyecto. Ya con los cuatro juegos necesarios decididos se pasó a diseñar el flujo del programa, es decir cómo el usuario navegaría por la aplicación y accedería a todas sus opciones. Recordando que desde un inicio se planeó el uso de interfaces gráficas, el diseño del flujo debía tomar en cuenta esto. Después de algunos borradores finalmente se llegó a este primer diseño del programa:

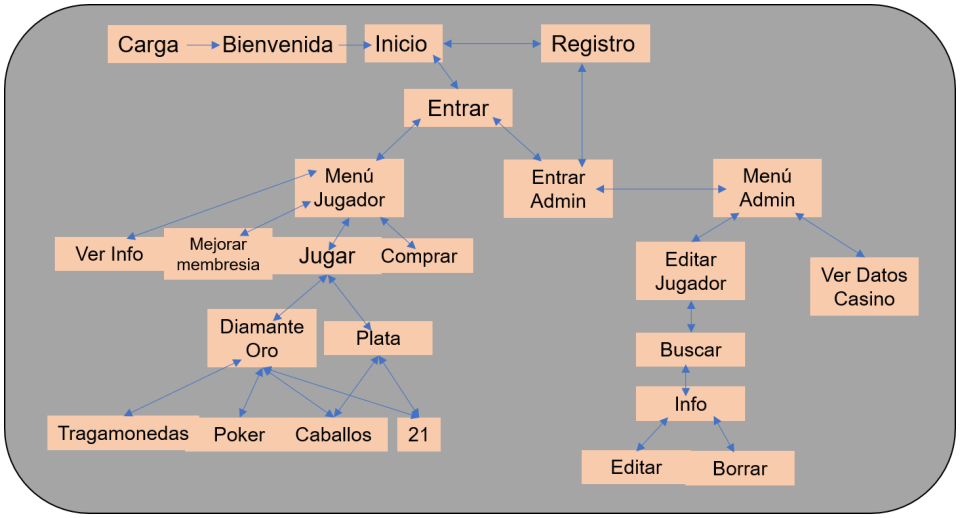


Diagrama 1. Flujo lógico del programa.

En este diagrama podemos ver la idea original del programa, el cual realmente no cambió mucho a la hora de implementarlo. La idea era que al iniciar el programa se mostrara una pantalla de carga, luego una de bienvenida y te dejara en la pantalla de inicio del casino. En esta pantalla de inicio se podría elegir si registrarse como usuario nuevo o iniciar sesión. Si se decidía iniciar sesión se le pedirían su nombre de usuario y contraseña, pero se tendría la opción adicional de entrar como administrador o técnico del programa. El usuario normal tiene las opciones de jugar, mejorar la membresía, comprar fichas y ver sus datos (nombre, membresía, fichas, etc...). Por el otro lado el administrador tendría las opciones de editar un jugador (cambiar datos o borrarlo) y ver las estadísticas/datos del casino.

Para empezar a desarrollar el programa se decidió dividir su elaboración en varias partes o módulos. Principalmente cada uno de los juegos sería un módulo diferente y el resto del programa sería otro módulo grande (con módulos más chicos para el registro de usuarios y todo el manejo de archivos). El desarrollo completo del programa se realizó en lenguaje Java y en el IDE NetBeans. En éste pudimos aprovechar el conjunto de componentes propios del lenguaje que crean los GUIs (Graphic User Interface) en NetBeans. La separación en módulos del proyecto que es de lo que hablábamos anteriormente se vió reflejado en esta parte gráfica en diferentes estructuras JFrame. Es decir cada juego tiene su propio JFrame y el resto del programa tiene un sólo JFrame. Para mostrar las diferentes secciones del programa lo que se usaba era cambiar el panel ,el componente JPanel, que se mostraba en el JFrame. En el momento en que se llegaba a un juego se escondía el JFrame del flujo del programa y se mostraba el del juego. Cuando se acababa el del juego se volvía a mostrar el principal y se desechaba el del juego.

Explicando un poco más de cómo se llevo a cabo el flujo del programa, la idea principal es encadenar una serie de estructuras switch-case dentro de ciclos do-while, de manera similar a como se ha hecho en proyectos anteriores sólo que en lugar de imprimir el menú en consola, se cambia la interfaz gráfica. Para esto se inicializan todos los JPanels y JFrames en el método main() y se sigue la siguiente estructura:

```
do{
    mostrar panel actual.
    obtener boton presionado del panel actual.
    switch(boton presionado){
        case 1: //Boton 1
            do{
                mostrar panel nuevo
                obtener boton panel nuevo
                switch(boton presionado)
                ...
            }while(opcion_x);
            break;
        case 2: //Boton 2
            do{
                ...
            }while(opcion_y);
            break;
        case 3: //Boton 3
            do{
                ...
            }while(opcion_z);
            break;
    }
}while(opcion_1 != 0);
```

Diseño lógico del programa.

En la figura anterior se aprecia en una especie de pseudocódigo cómo se implemento el flujo del programa. Al momento de pasarlo del diseño a la implementación real nos encontramos con diversas dificultades y problemas. Para todos los JPanels y JFrames se hizo que implementaran una interfaz específica que entre otras contenía las medidas de la interfaz y la ruta al ícono. Gracias a esto se logró tener un formato uniforme para toda la ejecución del programa. Con todo esto el primer desafía al que nos enfrentamos fue hacer que se pudiera cambiar entre paneles en un mismo frame y que se pudiera cambiar entre los frames mostrados. La solución a esto fue por una parte agregarle un método cambiarPanel(JPanel) a la interfaz de los JFrames y por el otro agregarle un método para esconderlo (poner su visibilidad falsa), uno para mostrarlo (poner su visibilidad verdadera) y uno para cerrarlo por completo. Ya con esto resuelto llegamos al mayor problema y el más importante de lo que respecta al flujo de las interfaces, cómo lograr pasar información sobre los botones que se aprietan en el panel mostrado al método main() para poder saber qué quiere hacer el usuario y qué es lo que debemos mostrarle en pantalla. Tras investigar mucho sobre las posiblidades y ver tutoriales de java swing, llegamos a la siguiente solución creativa: *Primero se creo una clase que heredara de ActionListener y se le agregaron dos atributos: booleano botón=false y entero opcion=0. Se les agregó la mimsa instancia de esta clase a todos los botones de todas las interfaces que utilizamos. Lo que hace esta clase es que cada vez que se registre una acción en el componente de java swing que esté .*"es*cuchando"va a llamar a uno de sus métodos. Aprovechando lo que hicimos fue que el método al que se llamaba cada vez que se apretara un botón cambiara el atributo booleano de la clase a true. Y dentro del cuerpo del método de acción del botón se modificaba el valor del atributo opción. Lo segundo que tuvimos que hacer fue crear un método en la clase donde se encuentra el main, el cual estuviera en un ciclo while checando cuando el atributo booleano botón de la clase ActionListener tomaba un valor true y en ese momento se rompía el ciclo y regresaba el valor del atributo entero opción del ActionListener.* Así en el cuerpo del main se llamaba a este método para obtener el número de opción del valor seleccionado el cuál se usaría para definir el switch-case, pero no se podía obtener el valor hasta que se presionara el botón o sucediera otra accion que modificara el atributo booleano botón del ActionListener. Con esto basicamente resolvimos todo el flujo de las interfaces y opciones del programa y sólo nos quedó preocuparnos por el contenido más funcional.

Lo siguiente con los que se trabajó fue con el manejo de los archivos del programa. Se necesitaban tratar con tres archivos de tipo objeto de tres clases diferentes. La clase Cliente que hereda de Persona y representa a los usuarios del casino, las instancias de esta clase tienen los datos de la persona entre los cuales se encuentra sus fichas, el apodo de usuario y la contraseña con la que entrarán a su sesión en el programa. La clase Administrador que hereda de Persona y representa a los técnicos o administradores del casino, ellos tienen igual datos personales y contraseña para ingresar. Y la última clase, la clase Casino que contendrá la información estadística del Casino. Cuánto se ha ganado o perdido y en qué juegos, además del número de usuarios registrados. Estas tres clases eran los archivos de tipo objeto que manejaríamos. El diseño de la base de datos se planteó con tres carpetas para guardar los archivos respectivos de cada clase. Cada archivo representaría a un sólo usuario, administrador o casino. Esto para que cuando se tuviera que cargar al programa no se necesitara abrir un archivo y buscar entre todos los objetos que contenga, sino que

únicamente abriera el del objeto necesario. Asimismo se planeó que el nombre del archivo fuera el apodo de usuario del cliente, el nombre del administrador o el nombre del casino. De esta forma a la hora de ingresar o cualquier otra recuperación de la información se pidiera el apodo o nombre al usuario y mediantes el método exists() de la clase File se comprobara si existe su registro. En caso afirmativo se carga el objeto al programa y se le pedirá al usuario la contraseña correspondiente que debe coincidir con la del objeto guardado en el archivo.
Para todo este manejo se creó una clase auxiliar llamada ManejoArchivos la cual cuenta con tres métodos para cada una de las clases mencionadas anteriormente: el método comprobarExistencia() regresa valor booleano que indica si existe el archivo con el nombre pasado, el método guardarUsuario() que guarda en un archivo el objeto del usuario (cliente, admin o casino) y el método cargarUsuario() que lee de un archivo un objeto y lo regresa como valor de retorno. En la clase principal se creó una instancia de la clase ManejoArchivos al igual que en algunos paneles y a través de ella se hizo el manejo de los objetos que se debían guardar y cargar durante la ejecución.

Lo siguiente que tocó resolver fue el registro de nuevos usuarios y administradores. La siguiente imagen nos muestra las interfaces de registro:

Interfaces de registro.

Para esto se le pidió al usuario el valor de la mayoría de los atributos del cliente o administrador a registrar, los que no se pedían era porque derivaban de los anteriores. Como se puede ver, la primera medida que usamos para robustecer el programa fue el no permitirle al usuario introducir libremente todos los valores, sino que se añadieron deslizadores para los enteros y cajas de opciones para los géneros y membresías. De esta forma en esos campos que eran fácilmente de error, forzamos a introducir un valor correcto. Lo siguiente que se hizo fue el añadir excepciones propias para los demás valores de tipo String que escribía directamente sobre los espacios de textos. Algunas de estas fueron el largo del nombre y el apellido, que el correo forzosamente debía incluir un @ y la más importante, limitamos los caracteres del apodo que se podían ingresar. Esto último debido a que su archivo de registro usaría este valor para nombrarse, por lo que se evito cualquier símbolo que Windows no aceptara en sus nombres de archivos y además el carácter ñ que nos causó muchos problemas. Si se picaba el botón aceptar con algún dato vacío o erróneo se botarán las excepciones que evitarán el flujo del programa y mostraran etiquetas y mensajes ocultos con información de los datos erróneos. Si ya se tienen todos los datos correctos, el programa creará el objeto y lo guardará en un archivo en automático.

Con todos los elementos anteriores lo siguiente fue añadirles las funcionalidades, a parte de los juegos, al usuario y al administrador. Por un lado el usuario puede comprar fichas, ver sus datos y actualizar su membresía. Para estos tres se agregaron botones en el menú de usuario. Al ver datos se cambiaba la interfaz a una donde esta impresa la información del usuario, al comprar fichas se pasaba a una interfaz donde el usuario puede elegir entre comprar 200, 500, ó 1000 fichas. Al seleccionar cualquiera, esta cantidad se abona al usuario cliente en su atributo fichas e idealmente se le notifica. Y el botón para actualizar la membresía manda un mensaje pop up a la pantalla donde se informa de la actualización exitosa si tiene las suficientes fichas, o el error en caso contrario indicando que no las tiene y el precio de la actualización.
En el caso de las acciones del administrador se puso más interesante ya que el administrador puede ver la información del casino, pero también editar a un jugador (cambiar datos o inclusive borrarlo). Para el administrador después de entrar con su contraseña se le muestran sus opciones, en caso de ver datos casinos se le lleva a una interfaz con los datos del casino impresos. Pero si elige editar jugador, pasa a una interfaz donde se le pide ingresar el apodo del jugador que quiere editar y éste con ayuda de los métodos probarexistencia() y cargarUsuario() de la clase ManejoArchivos comprueba que exista y en caso efectivo lo carga directamente sin pedir la contraseña, en caso negativo manda una

excepción y un mensaje de error. A la hora de borrar al jugador, se borra el archivo y se pierde la referencia del objeto cargado en la ejecución. En caso de la edición se muestra una pantalla parecida al registro de usuario que cuida las mismas restricciones, pero que en caso de obtener datos correctos sobre escribe el archivo del usuario.

De esta forma lo último que faltaba de la implementación para el programa eran los juegos. De manera sencilla a continuación se explica su solución respectiva implementada:

Carrera de Caballos

La carrera de caballos es el juego que teníamos que implementar si o si, al momento de desarrollarlo nos dimos cuenta que esto se debía básicamente a que se tenían que utilizar hilos.

Figura 3: Interfaz gráfica para el juego de Carrera de Caballos.

El desarrollo de esta parte del programa fue difícil, pues el manejo de interfaces e hilos al mismo tiempo fueron un dolor de cabeza. Al inicio lo mas fácil fue decidir como seria el programa principal pues solo generaría números random y esto haría que unos caballos fueran mas rápidos que otros poco a poco esto se fue complicando al implementar interfaces gráficas. Básicamente la solución era tomar las coordenadas de los caballos en el eje Y y asignarles la posición de inicio en el eje X y sumarle a este eje, después de esto reimprimir y listo generamos el efecto del avance de cada caballo.

Al final el programa consto de 2 clases la primera es el JFrame de la carrera el cual contiene todos los elementos gráficos y la segunda clase era la de carrera la cual se instanciaba 5 veces una por cada caballo y esta constaba de todas las operaciones lógicas para designar que caballo era el ganador esto pasaba cuando un caballo chocaba con una barra que añadimos.

Para terminar se añadió el sistema de apuestas el cual nos mostraba el crédito y la apuesta en un JLabel que se reimprimía cada que había un cambio, para apostar añadimos 3 botones los cuales sumaban la cantidad de cada botón a nuestra apuesta y estos mismos contienen las sentencias lógicas para no apostar mas de lo que tenemos de crédito o mas de lo que nuestra membresia nos permite.

Tragamonedas

El tragamonedas fue uno de los programas mas sencillos pues la implementación de las apuestas fue la misma que en los caballos.

Figura 4: Interfaz gráfica para el juego de Tragamonedas.

La gran dificultad de este programa fue el lograr que las imágenes parecieran que rodaban en los paneles, esto me resulto demasiado complicado pues tenia que imprimir diferentes imágenes en un solo panel. La resolución se logro cambiando nuestra forma de ver las cosas e investigando un poco, gracias a esto llegamos a la conclusión de que era mas fácil colocar todas las imágenes en una solo y recorrerla imagen a imagen dándole el efecto de giro.

Este programa consta de 4 clases la primera es la que contiene al JFrame con todos los botones la cual contiene la mayor parte de las funciones relacionadas a las apuestas, las siguientes son Figuras y Casillas las cuales son las encargadas de mostrar las imágenes en los paneles y hacerlas girar y por ultimo tenemos la clase verificador la cual es la encargada de decir si ganamos o no y añadir lo ganado o perdido al crédito del jugador.

Con esto podemos conocer la función general del programa que básicamente se basa en que agregaremos la apuesta deseada e iniciaremos el juego con el botón de inicio, con esto activaremos el juego pasando todos los valores al verificador el cual al ver que se repite el valor de las casillas 3 veces nos dice que hemos ganado gracias a que cada valor corresponde a una imagen. Por ultimo añadirá las fichas ganados o perdidas a nuestra cuenta y podremos seguir jugando hasta que queramos parar.

Poker

El juego de Poker que se decidió implementar, fue la variante *Texas Hold'Em*, ya que es la que consideramos que era la más adecuada para nuestro casino.

Figura 5: Interfaz gráfica para el juego de Poker.

Cuando se selecciona jugar Poker, se tiene un botón que dice Ïnicia", al presionarlo, este inicia el juego, en donde se reparten dos cartas al usuario y dos cartas al oponente. El usuario siempre tendrá la apuesta ciega pequeña al iniciar, y el oponente la apuesta ciega grande. El oponente siempre imitará la acción que realice el usuario, esto es, si el jugador "pasa", entonces el oponente también "pasará", si el usuario apuesta cierta cantidad de fichas, el oponente apostará la misma cantidad de fichas.

Ya sea que el usuario apuesta, o pase, se mostrará la primera ronda de cartas, para después, volver a la ronda de apuestas. Esto se repite para la segunda ronda de carta-apuestas y la tercera ronda de carta-apuestas. Una vez que se hayan mostrado las cinco cartas, y el usuario apueste o pase, se activará el botón Resultado", con lo que se muestran las cartas del oponente y se manda en una ventana emergente, el resultado, ya sea que el jugador gane o pierda. Presionando el botón .ºk", en esta ventana emergente, regresará al usuario al menú de los juegos.

El funcionamiento interno del programa es como procede:
Cuando se inicia el juego, se crea una baraja estándar de 52 cartas y se barajea, se crea al jugador (usuario) y un oponente. Se reparten dos cartas a cada jugador (usuario y oponente), en el usuario solo conoce sus cartas y no sabe cuáles son las cartas de su oponente (tal cual como si se hiciera en un caso IRL), se hacen las apuestas ciegas y se procede a la ronda de apuestas. En la ronda donde se sacan las cartas del *river*, también se hace la simulación de sacar una carta extra que irá a la pila de cartas no usadas, con el propósito de mantener el realismo y la emoción del juego como si se estuviera realmente sentado ante una mesa de póker.

El oponente siempre realizará la misma acción que el usuario, esto significa que si el usuario alza la apuesta, no hay posibilidad de que el oponente vuelva a alzar la apuesta. Por lo tanto, cada ronda de apuestas se simplifica a una acción para cada jugador.

Al finalizar todas las rondas de sacar cartas para el river y de apuestas, se procede a realizar el conteo de quien ganó. Esto se realiza, obteniendo los valores de las cartas y sus palos, juntando en un ArrayList la mano de un jugador (del usuario o del oponente) y todo el river, posteriormente, probando cada posible caso de éxito, desde el menos probable (Escalera Real) hasta el más probable (Carta Alta), y regresando una puntuación de acuerdo al caso en el que cayó la mano que se está revisando.

Una vez obtenidos las puntuaciones de ambas manos, se procede a determinar quien es el ganador, simplemente revisando quien tiene la puntuación más alta.

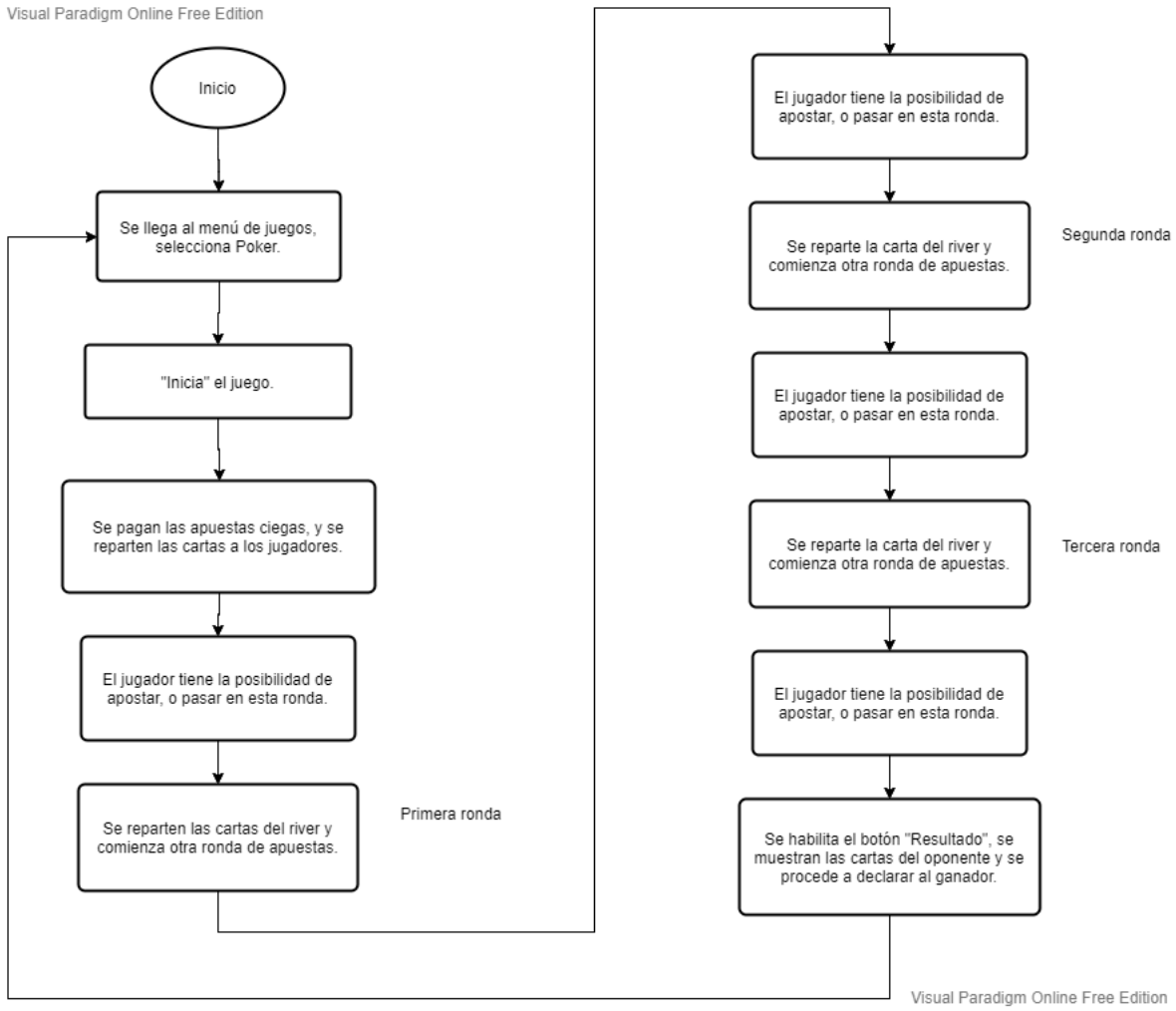


Figura 6: Diagrama de flujo del funcionamiento para el juego de Poker.

Blackjack

El juego de Blackjack o 21, es uno de los juegos recomendados, y decidimos implementarlo en nuestro casino.

Figura 7: Interfaz gráfica para el juego de BlackJack.

Cuando se selecciona el juego de BlackJack, se tiene el botón Ïnicia", al presionarlo, se sirven dos cartas al usuario y dos cartas al dealer, con las cartas del usuario siendo mostradas, mientras que las del dealer, sólo se muestra una carta, de acuerdo a las reglas del BlackJack.

Una vez hecha esta acción, se le da la opción al usuario de pedir carta, quedarse con sus cartas o apostar. Mientras la suma de la mano de jugador sea menor o igual a 21, el usuario tiene estos botones disponibles, pero si el usuario se pasa de 21, o decide quedarse con sus cartas, se procederá al turno del dealer, mostrando sus cartas actuales y si lo necesita, sacará cartas de la baraja, mientras, el usuario no tendrá disponibles los botones previamente mencionados.

Terminando el turno del dealer, aparece disponible el botón "Termina", con lo que se abre una ventana emergente, en donde se le dice al usuario si ganó o perdió. Y al presionar el botón .ºk", se regresa al menú de juegos.

Para poder determinar si el jugador es ganador o no, simplemente se revisa si la suma de los valores de las cartas que están en posesión del usuario, es igual o menor a 21 siempre y cuando, la suma de las cartas del dealer es menor a la del usuario o no se pasa de 21. No interesa de qué palo sean, sin embargo, en el caso de que se tengan Ases, se debe hacer una excepción, el valor de un As es de 11, pero si la suma pasa de 21, el As ahora pasa a tener un valor de 1.

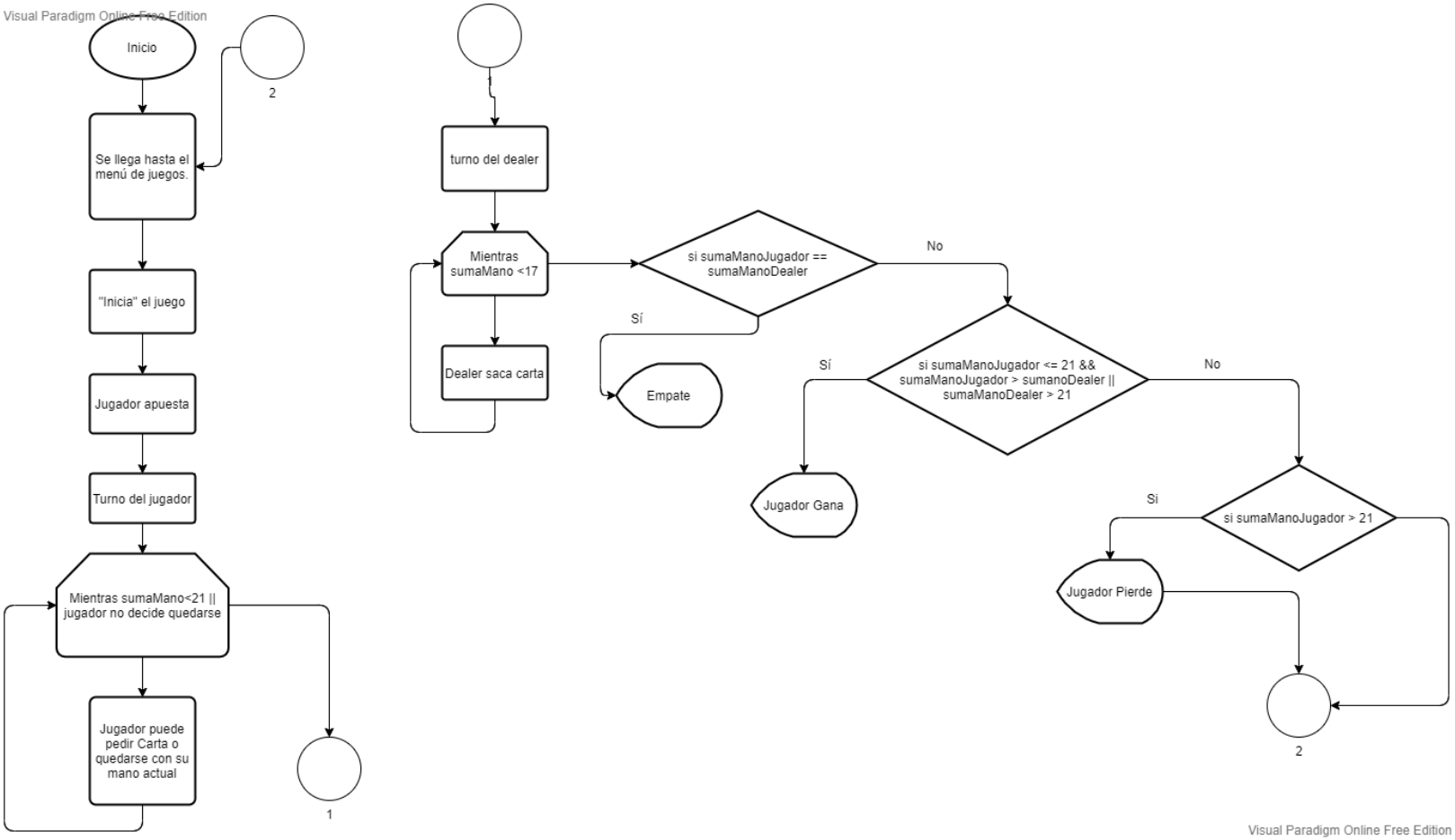


Figura 8: Diagrama de flujo del funcionamiento para el juego de BlackJack.

*Elementos de Puntuación Extra

En un principio durante las primera juntas que tuvimos, se habló de implementar los tres rubros de puntuación extra. Ciertamente el programa tenía lugar para usarlos, y la posibilidad de obtener mejor calificación era llamativa. Sin embargo, en estas primeras reuniones éramos muy ambiciosos y pensábamos más en las posibilidades que en las limitaciones. Por lo que a mitades del desarrollo completo del programa, nos dimos cuenta que no habíamos usado patrones de diseño y a esas alturas el agregarlos se sentía forzado y poco beneficioso (funcionalmente hablando, formativamente sí lo hubiera sido). Debido a esto únicamente se implementó el maneo de hilos y las interfaces gráficas como se mencionan a continuación. (De hecho se usó el método clone() en cierto punto del patrón de diseño Prototype, pero consideramos es muy poco uso como para contabilizarlo.)

Interfaces Gráficas

Todo la interacción de nuestro programa con el usuario se lleva a cabo mediante las interfaces gráficas. En total tenemos más de 20 interfaces diferentes. Se utilizaron JFrames y JPanels para poder lograr este cometido, principalmente mediante el uso del IDE NetBeans, ya que éste nos proporciona un entorno en donde poder diseñar la interfaz gráfica más fácilmente, incluyendo todas las líneas de código necesarias, para la creación de un elemento dentro del Frame o del Panel, así como para cuando se agrega un evento a un elemento.

El objetivo de usar interfaces gráficas para la interacción con el usuario fue hacer un programa más *amigable* para todo público y que asemejara un desarrollo más profesional de un simulador de casino. A lo largo de las secciones de análisis y desarrollo de este reporte ya se ha hablado mucho acerca de cómo se implementaron en el programa. Desde el diseño se planeó una interacción gráfica y a lo largo de toda la codificación se tuvo en cuenta. Para cada una de las interfaces se tuvo que crear un diseño que fuera acorde con la temática del casino, además se cuidaron los detalles como el ícono de las ventanas del casino (que es un dado azul) y el nombre de las ventanas (*Casino The Paradise*). En nuestra opinión estos detalles diferencian a los programas que se ven más profesionales y los que se ven más trabajos escolares. A pesar de que sabemos que se podrían mejorar ciertos aspectos de la interfaz gráfica, todo la sensación y la imagen de esta se ve bien cuidada y de calidad.

Por esto nosotros consideramos que nos merecemos el punto extra de las interfaces gráficas, ya que sí representaron un esfuerzo grande y cambian totalmente el programa y la experiencia del usuario.

Manejo de Hilos

El segundo elemento de puntuación extra que implementamos fue el manejo de hilos. Esto ya era prácticamente un requerimiento del programa para el juego de carrera de caballos. Por lo que el usarlos nos vino bien en ambos aspectos. Pero antes de hablar del manejo de los hilos en los juegos, vamos a mencionar el uso de hilos en las interfaces gráficas. Y esto debido a que cada instancia de una clase que herede de JFrame en realidad es un nuevo hilo de ejecución. Gracias a esto a manera general del programa se cuenta con el hilo del método main y otros 5 hilos de los diferentes JFrames utilizados, además de los que se usan dentro de los juegos. Los JFrames implementan la interfaz de Java *Runnable*, mediante la cual se crean las interfaces gráficas en un hilo separado al principal. Y mediante el método *java.awt.EventQueue.invokeLater*, se procede a ejecutar el *Runnable* creado para la interfaz gráfica. Todo esto se hace de forma hasta cierto punto automática, pero el entenderlo es importante para poder trabajar adecuadamente con ellos.
Además se usan los hilos en los juegos:

- Carrera de caballos

Hablando de este juego fue el único que esta estructurado completamente para manejarse con hilos debido a que la clase mas importante es la de carrera que en si es instanciada 5 veces para generar 5 hilos los cuales corresponden 1 a cada caballo, de esta manera cada caballo avanzara a su paso y cada hilo contiene la sentencia para detener el programa la cual es que si ya gano un caballo todos los demás se detendrán, de esta manera si gana un caballo se acaban todos los hilos.

Gracias a que usamos los hilos de programación de manera adecuada y útil para el programa, nosotros creemos que se cubre perfectamente este rubro de puntuación extra y que nos merecemos el punto extra por el manejo de hilos. A lo largo del programa se usaron los hilos semi automáticamente de las interfaces (se tenía que usar dispose() para matarlos) y en los juegos los usamos para lograr una concurrencia entre los diferentes elementos y objetos que participaban.

Conclusiones

David Calderón Jiménez

El desarrollo de un proyecto tan grande fue muy interesante de trabajar, ya que la planeacion de todo el proyecto fue mas coordinada que en proyectos anteriores, ya que este programa no esta diseñado para armarse por partes y después unir cada una de ellas y esto lo pudimos notar con el numero de reuniones que tuvimos pues fueron mucho mas que las que fueron en otros proyectos. Puedo decir que el desarrollo de este proyecto fue muy difícil pues el objetivo de este proyecto fue implementar todo lo visto en este semestre y todavía nosotros tomamos el reto de implementar interfaces gráficas lo que aumento aun mas su dificultad pero al ser final de semestre y ya estar mas libres pudimos tomarnos el tiempo de investigar lo suficiente para poder llegar a un programa que como equipo nos dejo muy satisfechos.

Humberto Ignacio Hernández Olvera

La parte lógica del programa, por ejemplo, para algunos de los juegos, se me hace fácil de entender, a tal grado que primero se tenía el juego en funcionamiento por consola, pero cuando se hizo la migración hacia la interfaz gráfica, me quedé totalmente perplejo, porque no entendía completamente que era lo que estaba pasando, seguía pensando que el mismo código iba a funcionar dentro de la interfaz, hasta que con la ayuda de mis compitas del equipo, logré entender que lo adecuado era, transformar la lógica del programa a funciones, con el propósito de ejecutar estas funciones a través de las acciones de los botones que se tuvieran en la interfaz.

A pesar de que fueron tiempos complicados con este proyecto, considero que se llevó a cabo una buena realización del mismo, logramos cubrir todos los aspectos obligatorios y aprendimos en el proceso, ahora ya tenemos un buen avance en la curva de aprendizaje para interfaces gráficas en Java, lo único que nos resta es no abandonarlo y seguir practicando, para que podamos utilizar en el futuro, y quizás, migrar a otras interfaces gráficas, ya sea aquí mismo con Java (JavaFX) o en otro lenguaje de programación.

Definitivamente este proyecto nos ayudó a aterrizar todos los conceptos que aprendimos a lo largo del semestre, el uso de paquetes, modularidad y encapsulación creo son los temas que más se utilizaron en el trabajo, pero no por eso significa que no se utilizaron los otros temas, como herencia, interfaces, incluso enums, que nos sirvieron para llevar a buen termino nuestro proyecto.

Iñaky Ordiales Caballero

En el inicio de la práctica se plantearon diversos objetivos, los del proyecto final, los de la opción del casino y los personales. Una vez que ya hemos acabado toda la elaboración del programa podemos mirar hacia atrás y ver cómo y hasta que punto se cumplieron éstos. Aunque la verdad es un tanto diferente a esto ya que no es que se hubieran leído los objetivos al principio y después hayamos empezado un programa sin considerarlos. En todo el desarrollo se tuvieron presentes tanto los objetivos del proyecto como los requerimientos del programa.

Con respecto a los objetivos del proyecto que eran los principales, estos se cumplieron satisfactoriamente. Al ser el proyecto más grande que al menos yo he realizado hasta la fecha (aunque estoy consiente que sigue sin ser muy grande) sí tuve que poner a prueba tanto mis habilidades de programación del paradigma orientado a objetos como el conocimiento de los conceptos vistos en el curso. Como el programa era más complejo y completo a lo que estamos acostumbrados a hacer en las prácticas, no te podías restringir a trabajar con un par de conceptos del curso, ya que estos se iban entrelazando en todas las partes del programa y necesitaban coordinar para lograr el funcionamiento esperado. Además se dividió el trabajo en los miembros del equipo y mediante estarnos apoyando, todos pudimos entregar nuestra parte y juntarle en el proyecto final. El objetivo del casino se cumple con las funcionalidades y juegos del programa.

Finalmente como conclusión de todo el proyecto y con respecto a los objetivos personales que nos planteamos en un inicio, creo que estos se cumplieron muy bien. Yo sí sentí que mostré la gran mayoría de mis conocimientos de la materia, además de que los puse a prueba. Una vez viendo el producto final yo quedo muy satisfecho con él. La parte gráfica se ve bastante bien y funciona de manera compresible para el usuario. Creo que es el primer programa que se podría mostrar y dejar a un público general para usar, esto ya que no necesitan de ingresar datos en consola lo que es desconcertante y poco atractivo para la mayoría. Quedo satisfecho y feliz con ***Casino The Paradise***.

Fuentes de información

[1] Morgan Eckenroth y col. *Big Java Poker*. URL: https://github.com/yashbhutwala/java-poker-gui-ai/blob/master/src/controller/CardScorer.java (visitado 05-02-2020).

[2] JavaAtPoint. *Java Swing Tutorial - javatpoint*. URL: https://www.javatpoint.com/java-swing (visitado 05-02-2020).

[3] Oracle. *Java™ Platform, Standard Edition 15 API Specification*. URL: https://docs.oracle.com/en/java/javase/15/docs/api/index.html (visitado 05-02-2020).

[4] Oracle. *Trail: Creating a GUI With JFC/Swing*. URL: https://docs.oracle.com/javase/tutorial/uiswing/.

[5] Wikipedia. *Swing (Java)*. URL: https://en.wikipedia.org/wiki/Swing_(Java) (visitado 05-02-2020).

Casino The Paradise