Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería en Informática

Proyecto Semestral de Laboratorio: Dobble en Java

Profesor Encargado: Miguel Trufa
Alumno: Nicolás Rojas
Rut: 20.058.348-5

1. Introducción

En el siguiente informe se trata la problemática de crear un juego de cartas llamado Dobble dentro del paradigma orientado a objetos utilizando el lenguaje de programación Java. Este proyecto busca aumentar el conocimiento sobre los alcances que tienen los paradigmas de la programación, si facilitan o dificultan la problemática, si las herramientas a utilizar permiten una mejor implementación y algunas diferencias en la implementación de código con respecto a las versiones de paradigmas anteriores como el funcional y el lógico.
Se comienza con una descripción del problema, en el que se da a conocer las reglas del juego Dobble, cómo se juega y las características básicas para desarrollar en la problemática. Continua con una breve descripción del paradigma, en la que se introducen los conceptos más importantes de la programación orientada a objetos en Java. Se analiza el problema en el que se muestran todos los requerimientos necesarios para implementar una solución junto con diagramas UML. Se describe y se presenta un modelo de solución para crear partidas de Dobble con usuarios por turnos y contra una CPU. Se presenta la estructura del proyecto junto con su implementación bajo el lenguaje de programación solicitado. Se agregan las instrucciones de uso para manejar el programa y finalmente comprobar junto con los ejemplos entregados para concluir el uso potencial del paradigma de programación dentro de una problemática como esta.

2. Desarrollo

2.1 Descripción del problema:

Dobble es un juego de cartas con un número determinado de símbolos, las cuales entre si solo pueden tener un único elemento en común. Este juego tiene una variedad de minijuegos, pero para este proyecto solo se contempla una versión modificada del minijuego “Torre Infernal”. Para jugarlo se deben colocar 2 cartas al azar sobre la mesa a la vista de todos los jugadores. Los jugadores deben reconocer el símbolo igual que hay entre las 2 cartas. El primer jugador que las reconozca robará 2 nuevas cartas aleatorias y ganará 2 puntos. Finalmente se acaba la partida cuando los jugadores lo deciden, y el ganador será el que tenga la máxima cantidad de puntos.
La complejidad de la creación de Dobble es el algoritmo para la generación de cartas con un único elemento en común, ya que esto implica tener conocimiento en propiedades matemáticas como los planos proyectivos finitos, plano de fano, geometría, entre otras cosas. Para esto se utiliza la documentación de un algoritmo en JavaScript proporcionado por Micky Dore[1], el cual explica a través de listas, matrices e iteraciones la forma de generar cartas según una cantidad de elementos dados como símbolos en una variable.

2.2 Descripción del Paradigma:

El paradigma de programación utilizado en este proyecto es el orientado a objetos, el cual pertenece a la familia de los paradigmas imperativos. Está basado en el concepto de “objetos”, y los programas son diseñados con el fin de crear objetos en memoria, y que estos interactúen entre sí. Los objetos pueden contener atributos y métodos. El lenguaje de programación que se utiliza dentro de este proyecto es Java, desarrollado por James Gosling en el año 1991.
Dentro de Java se pueden crear Clases, que corresponden a la especificación de los atributos y comportamientos para un cierto tipo de
objeto. Estas sirven de plantilla para la creación de objetos. Además, se puede decir que las clases son la implementación de un TDA.

Algunos conceptos claves de este paradigma son:

- Objeto: unidad básica que representa un objeto de la vida real.
- Interfaz: plantilla para la construcción de clases.
- Clase: plantilla para la creación de objetos.
- Atributo: características individuales que diferencian un objeto de otro.
- Método: operación que puede realizar un objeto.
- Overriding: redefine un método de una clase.

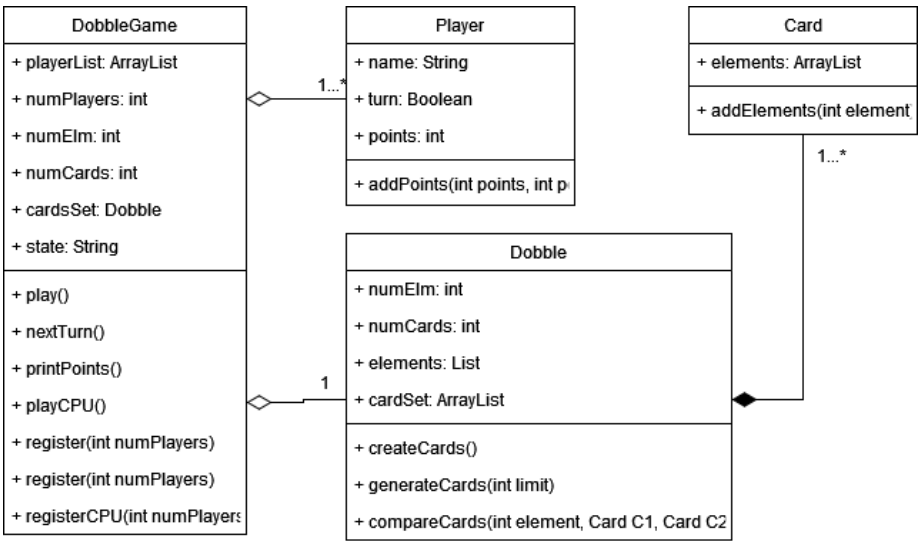
[1] Dore, M. (2021, 30 diciembre). The Dobble Algorithm - Micky Dore. https://mickydore.medium.com/the-dobble-algorithm-b9c9018afc52

2.3 Análisis del Problema:

El programa deseado tiene los siguientes requerimientos funcionales:

1) TDAs: Teniendo en cuenta que Dobble es un juego de cartas, hay que pensar en una estructura que nos permita almacenar los valores de las cartas, es decir, los elementos que se utilizarán como símbolos, y el mazo final. Además del mazo de cartas, se necesita una mesa en la cual dejar el mazo, en la cual estarán los jugadores, sus puntuaciones, el modo juego, etc.
2) Constructor de juego: Algo que define a Dobble es su mazo con cartas con un solo elemento en común, por lo que se debe crear un método que sea capaz de construir un mazo de acuerdo a una cantidad determinada de elementos. Éste también debe poder agregar el número de jugadores, sus puntuaciones, el estado del juego y las cartas que hay sobre la mesa.
3) Register: Este método deberá registrar una cantidad determinada de jugadores, los cuales serán capaces de jugar dentro de la partida.
4) Play: Los jugadores registrados pueden realizar jugadas dentro de la partida. Este método debe ser capaz de dar 2 cartas para cada jugador y determinar si el elemento que escribió el jugador por consola es el que hay en ambas cartas. En caso de que el jugador haya estado en lo correcto, se le sumarán 2 puntos. Luego pasará el turno al siguiente jugador.
5) ToString: A través de la sobreescritura las clases podrán imprimir en pantalla la información de los objetos creados.
6) Equals: A través de la sobreescritura las clases podrán reconocer si un objeto es igual que otro objeto.

Al utilizar el paradigma orientado a objetos, podemos considerar objetos a representaciones de la vida real. Los objetos principales para este problema deben ser las cartas, el mazo de cartas, los jugadores y el tablero, lo que en un diagrama UML queda representado de la siguiente manera:



2.4 Diseño de la solución:

Para el uso de TDAs en este trabajo se han usado interfaces para dar un molde a las clases que se utilizarán para el manejo de los datos que debe tener el tablero.
Las interfaces son:

1) Carta: contiene un método para añadir elementos a las cartas
2) Cartas: contiene métodos para crear un conjunto de cartas, limitarlo y comprar cartas del mazo con la respuesta de un jugador.
3) Jugador: contiene un método para añadir puntos a un jugador.
4) Mesa: Contiene métodos para registrar jugadores en multijugador, registrar un único jugador para el modo VS CPU, jugar en multijugador, jugar contra la CPU, mostrar en pantalla la puntuación, cambiar de turno de un jugador a otro.

Las clases creadas que utilizan estas interfaces son:

1) Card usa Carta:
   Atributos:
   - elements = Lista de elementos que contiene la carta. Estos elementos pueden ser más o menos dependiendo de lo que ingrese el usuario.

   Métodos:
   - addElements = Método que añade elementos a una carta. Su entrada es un elemento del tipo entero.

2) Dobble usa Cartas:
   Atributos:
   - numElm = número de elementos que tiene una carta.
   - numCards = número de cartas que tiene un mazo.
   - elements = lista de elementos que pueden ser usados para la creación de una carta.
   - cardSet = lista de cartas, la que forma el mazo.

   Métodos:
   - createCards() = método que crea cartas utilizando el algoritmo de Micky Dore.
   - generateCards(int limit) = método que crea cartas y limita según lo que pida el usuario.
   - compareCards(int element, Card C1, Card C2) = método que compara los elementos en común que tienen ambas cartas y la respuesta del jugador.

3) Player usa Jugador:
   Atributos:
   - name = nombre del jugador.
   - turn = dice si es el turno del jugador
   - points = puntos del jugador

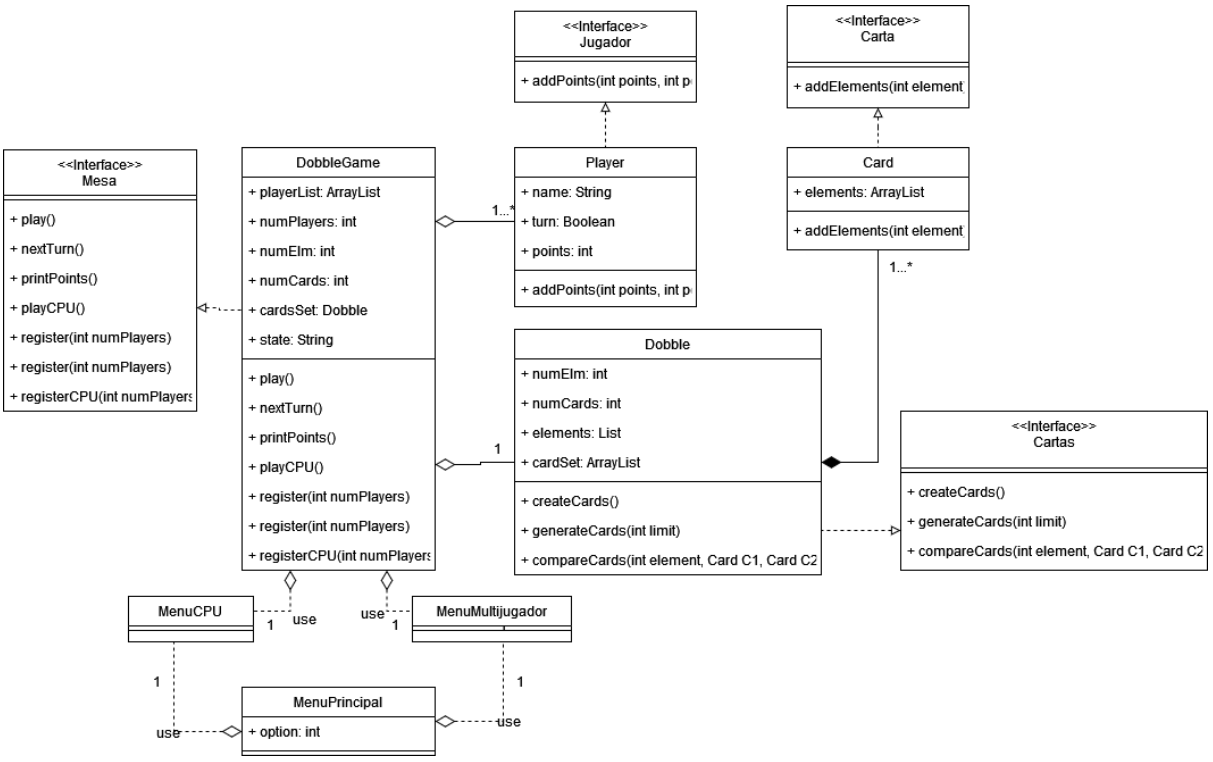
Métodos:
- addPoints(int points, int pointsToAdd) = método que agrega puntaje a un jugador.

4) DobbleGame usa Mesa:
   Atributos:
   - playerList = lista de jugadores.
   - numPlayers = número de jugadores a registrar.
   - numElm = número de elementos por carta.
   - numCards = número de cartas en el mazo.
   - cardsSet = mazo de cartas.
   - state = indica si la partida está en juego.

   Métodos:
   - play() = método que toma 2 cartas aleatorias y pregunta un elemento al jugador para luego añadir puntaje si gana.
   - nextTurn() = que termina el turno del jugador y actualiza el estado del siguiente jugador.
   - printPoints() = método que imprime en pantalla los puntajes de los jugadores registrados.
   - playCPU() = que toma 2 cartas aleatorias y pregunta un elemento al jugador para añadir puntaje si gana, y luego la CPU juega automáticamente.
   - register(int numPlayers) = método que tiene como entrada el número de jugadores. Esto permite crear jugadores que luego serán almacenados en la lista.
   - registerCPU() = método que registra un jugador y una CPU.

Además, para una mejor experiencia se han creado 3 clases relacionadas a menús:
1) MenuPrincipal = Este menú despliega las opciones para jugar en multijugador, contra la CPU o para cerrar el programa.
2) MenuMultijugador = Este menú registra jugadores y luego despliega las opciones para jugar, mostrar el puntaje o acabar la partida.
3) MenuCPU = Este menú registra un solo jugador y luego despliega las opciones para jugar, mostrar el puntaje o acabar la partida.

Con lo implementado tenemos un nuevo diagrama UML:



En el diagrama podemos ver como el MenuPrincipal puede variar entre MenuCPU y MenuMultijugador. La clase DobbleGame puede usar cualquiera de ambos menús para proceder con el juego. Las clases DobbleGame, Dobble, Card y Player utilizando las interfaces Mesa, Cartas, Carta y Jugador. Pueden haber 1 o muchas cartas en Dobble y solo un mazo Dobble en DobbleGame. Así mismo pueden haber 1 o muchos jugadores en el modo multijugador, o un jugador y una CPU en el caso de elegir el ModoCPU.

2.5 Aspectos de implementación:

Para la implementación de este proyecto usamos Java con OpenJDK 11 y el IDE IntelliJ IDEA. Usamos el algoritmo de Micky Dore como referencia para crear nuestra versión en Java mediante ciclos como for y condicionales. Una vez el mazo de cartas está creado, se aplica un método para randomizar las cartas que le aparecerán a los jugadores. Las cartas se “barajan” cada vez que un jugador va a sacar 2 cartas.
Debido a que las limitaciones del paradigma no permite que haya varios jugadores simultáneamente, se ha establecido que el juego sea a través de turnos, por lo que los jugadores deberán escribir las instrucciones como lo va solicitando en la consola. En cuanto a la generación de cartas, la lista de elementos es de 1 a 100, por lo que si se generan muchas cartas podría salirse del rango de esta lista. Otro supuesto para este proyecto es que el juego utilizará el mismo mazo hasta que los jugadores decidan finalizar el juego, es decir, que cuando se colocan las 2 cartas sobre la mesa, estas vuelven al mazo y pueden volver a aparecer en la mesa.

Todos los archivos .java se encuentran en la carpeta “src\main\java\com\onicalls”.
Estos archivos son:

1) Card.java = contiene a la clase Card con sus respectivos atributos y métodos.
2) Carta.java = contiene la interfaz Carta.
3) Cartas.java = contiene la interfaz Cartas.
4) Dobble.java = contiene a la clase Dobble con sus respectivos atributos y métodos.
5) DobbleGame.java = contiene a la clase DobbleGamecon sus respectivos atributos y métodos.
6) Jugador.java = contiene la interfaz Jugador.
7) labDobbleJava_200583485_NicolasRojas.java = contiene la clase principal donde se ejecuta el método para llamar a MenuPrincial.
8) MenuCPU.java = contiene el menú para jugar contra la CPU.
9) MenuMultijugador.java = contiene el menú para jugar contra varios jugadores.
10) MenuPrincipal.java = contiene el menú que indica qué modo de juego jugar.
11) Mesa.java = contiene la interfaz Mesa.
12) Player.java = contiene a la clase Player con sus respectivos atributos y métodos.

2.6 Instrucciones de uso:

Para el correcto uso de este programa debe seguir cuidadosamente las siguientes reglas:

1) Lo primero es revisar que los archivos mencionados en el punto 2.5 estén agregados correctamente dentro de la carpeta de su ordenador.
2) Puede hacer uso de un IDE como InteliJ IDEA para abrir el proyecto y ejecutarlo desde el botón RUN en la clase labDobbleJava_200583485_NicolasRojas.
3) Para ejecutar el código desde la consola de Windows, primero se deben compilar todas clases, comenzando por aquellas que no dependen de otras utilizando el comando javac. Y luego, para ejecutar se utiliza el mismo comando, seguido de la clase principal.
4) Al ejecutar el programa le pedirá al usuario que ingrese un número del 1 al 3, donde 1 es para jugar contra otros jugadores, 2 para jugar contra el CPU y 3 para terminar.
5) En caso de haber elegido 1, el programa preguntará por el número de jugadores, el cual es un número entero mayor a 2. Inmediatamente el programa le pedirá que ingrese los nombres de los jugadores. Finalmente preguntará la cantidad de elementos que debe tener una carta (intente no usar números mayores a 10) y la cantidad máxima de cartas a generar (si no sabe escriba 0).
6) El programa le pedirá al usuario que ingrese un número del 1 al 3, donde 1 es para jugar, 2 para ver los puntajes y 3 para terminar el juego. Si escribe 1, el programa indicará de quién es el turno, pedirá un elemento, indicará si ganó o perdió y luego pasará el turno al siguiente jugador. Si elige 2 mostrará el puntaje de todos los jugadores registrados. Para finalizar el juego debe presionar 3.

2.7 Resultados:

En este proyecto se pudo implementar todo lo necesario para que el juego funcione, siempre y cuando se sigan las instrucciones correctamente. La siguiente tabla de autoevaluación funciona de la siguiente manera:
a) 0: No realizado
b) 0.25: Funciona a lo más el 25% de las veces.
c) 0.5: Funciona el 50% de las veces.
d) 0.75: Funciona el 75% de las veces.
e) 1: Funciona el 100% de las veces.

Método	Puntaje
Constructor	1
register	1
play	1
toString	1
equals	1
vs CPU	1

3. Conclusión

Se han podido cumplir todos los objetivos que requería el proyecto. El uso de objetos como representación de la realidad junto con los atributos y métodos ha permitido un mejor desarrollo del proyecto con diferencia a los paradigmas anteriores como el funcional o el lógico.

En este proyecto se pudo aprender el potencial del paradigma orientado a objetos, la resolución de problemas, el uso de objetos, clases, interfaces, atributos y métodos. Además de la importancia del acoplamiento y cohesión entre las clases, pues un buen diseño de solución requiere una alta cohesión y un bajo grado de acoplamiento. Por ello, se debe realizar un buen análisis del problema, para así llevar a cabo el diseño de la solución.

4. Referencias

1) Dore, M. (2021, 30 diciembre). *The Dobble Algorithm - Micky Dore*. https://mickydore.medium.com/the-dobble-algorithm-b9c9018afc52

2) *Qué es la programacion orientada a objetos*. (2021, September 18). Desarrollo Web. https://desarrolloweb.com/articulos/499.php

3) *Canelo, M. M. (2022, April 18). ¿Qué es la Programación Orientada a Objetos? Profile Software Services. https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/*

4) *Dobble Kids cómo jugar*. (2019, 18 agosto). YouTube. https://www.youtube.com/watch?v=7XTXe7FqAM8

5) *How to make the card game Dobble (and the Maths behind it!)*. (2021, 8 abril). YouTube. https://www.youtube.com/watch?v=oyqD1Sg5M4Q