UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

INFORME DE LABORATORIO 4:
SIMULACIÓN DE DOBBLE EN JAVA

Nombre: Bastián Escribano
Profesor: Gonzalo Martínez
Asignatura: Paradigmas de Programación

11 de julio 2022

Tabla de Contenidos

1. INTRODUCCIÓN

Un paradigma se entiende como un conjunto de intuiciones, de parecer y de modos de emprender la realidad. Los paradigmas están en todas partes y en el mundo de la programación no es la excepción. Existen quizá miles de lenguajes de programación, los cuales probablemente tengan una forma diferente de llamar a sus métodos (o funciones), donde cambie hasta inclusive su sintaxis, es por esto por lo que, al dominar un lenguaje de programación, al cambiar a otro lenguaje, cuesta y es casi volver a aprender desde cero. Sin embargo, existen unos cuantos paradigmas de programación los cuales, para ser más exactos, declaran la gramática en la cual se está programando, esto significará que sea más fácil adaptarse a un nuevo lenguaje de programación. En este informe se introducirá, se planteará un problema y su respectiva solución a dos paradigmas de programación, el cual es el paradigma orientado a objetos de programación y además, el paradigmas orientado a eventos. Específicamente, será solucionado en el lenguaje de programación **Java**.

1.1 DESCRIPCIÓN DEL PROBLEMA

Se pide desarrollar una simulación de Dobble, El juego **Dobble** y en particular la generación de las piezas se rige por propiedades matemáticas dentro de los cuales destacan los números primos. Para implementar esto, se deben tener algunos elementos en cuenta:

Dobble: En base a parámetros que se ingresen, se debe generar un mazo de cartas para que sea un juego válido. El juego Dobble es un juego matemático y matemáticamente tiene que estar bien construido para que tenga sentido.

DobbleGame: Se necesitan jugadores, modos de juegos y un mazo para empezar un juego. **Métodos:** Son las métodos que modifican los objetos, jugadores, y diferentes modos de juego.

Todo debe ser implementado utilizando el Paradigma Orientado a Objetos, a través del lenguaje de programación Java.

DESCRIPCIÓN DE PARADIGMA LOS PARADIGMAS

El Paradigma Orientado a Objetos se basa en la definición de objetos, es decir, abstracciones de entidades que pueden tener atributos y métodos, con el fin de interactuar entre si (intercambio de información). Existen objetos de distintas clases, donde sus características y comportamientos pueden variar. Los comportamientos pueden leer o escribir las características del mismo objeto, por lo que pueden permitir interacciones entre sí. Uno de los lenguajes más populares en el ámbito de la programación orientada a objetos es Java, ya que es un lenguaje que trabaja fuertemente con los conceptos de clases, métodos y atributos.

En el Paradigma Orientado a Objetos se tienen algunos elementos bastante importantes a la hora de construir un código utilizando este paradigma, como, por ejemplo:

Clases: Son la definición de las características de un objeto las cuales tienen atributos y métodos. Una clase corresponde a una implementación de un TDA.

Objetos: Son instancias de una clase, es decir, representaciones activas de estas.
Atributos: Corresponde a lo que compone a una clase, como son distintos tipos de datos. También pueden ser otras clases.

Métodos: Son los comportamientos de los objetos. En palabras simples, son las "funciones" que componen a las clases. Los métodos expresan comportamientos que pueden realizar un objeto sobre si mismo o sobre otros objetos.

La **programación dirigida por eventos**, es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.

Interfaz Gráfica: Con la evolución de los lenguajes orientados a eventos, la interacción del software con el usuario ha mejorado enormemente permitiendo la aparición de interfaces que, aparte de ser la vía de comunicación del programa con el usuario, son la propia apariencia del mismo. Estas interfaces, también llamadas GUI (Graphical User Interface), han sido la herramienta imprescindible para acercar la informática a los usuarios, permitiendo en muchos casos, a principiantes utilizar de manera intuitiva y sin necesidad de grandes conocimientos, el software que ha colaborado a mejorar la productividad en muchas tareas.

Uno de los periféricos que ha cobrado mayor importancia tras la aparición de los programas orientados a eventos ha sido el ratón, gracias también en parte a la aparición de los sistemas operativos modernos con sus interfaces gráficas. Estas suelen dirigir directamente al controlador interior que va entrelazado al algoritmo.

Eventos: En contraposición al modelo clásico, la programación orientada a eventos permite interactuar con el usuario en cualquier momento de la ejecución. Esto se consigue debido a que los programas creados bajo esta arquitectura se componen por un bucle exterior permanente encargado de recoger los eventos, y distintos procesos que se encargan de tratarlos. Habitualmente, este bucle externo permanece oculto al programador que simplemente se encarga de tratar los eventos, aunque en algunos entornos de desarrollo (IDE) será necesaria su construcción.

Para crear un objeto, se debe hacer uso de un constructor, el cual es un tipo especial de método. Permite indicar los valores iniciales de sus atributos y tiene la misión de reservar la memoria necesaria para poder albergar todos los datos del objeto.

Algo también importante mencionar es que se tienen distintos tipos de diagramas que permiten organizar de mejor forma las ideas sobre las clases, atributos y métodos en un código donde se utilizan varias clases. El principal diagrama UML es el diagrama de clase,

el cual muestra las relaciones que existen entre las distintas clases que componen un código.

1.2 OBJETIVOS

Como objetivos del proyecto se tiene aprender sobre el paradigma orientado a eventos y la programación orientada a objetos, para así obtener la habilidad de programar de otra forma distinta a la que se tiene costumbre actualmente. Otro objetivo es programar correctamente en Java y aprender a utilizar las herramientas de este lenguaje para completar el proyecto de laboratorio y así poder tener una base para los futuros laboratorios y otros proyectos que en un futuro se desarrollen con la **Programación Orientada a Objetos y la Programación Orientada. Eventos.**

2. DESARROLLO

2.1 ANALISIS DEL PROBLEMA

Matemáticamente, es posible generar un mazo con cartas que contengan una cierta cantidad de símbolos y que se repita por lo menos 1 símbolos en cada carta diferente del mazo, a esto lo llamaremos **el factor de repetición**. Esto es posible a una conexión matemática que se basa en números primos y en potencia de números primos. Este factor es el más importante y su nombre es **orden**, por lo tanto, el orden debe ser necesariamente un número primo o una potencia de un número primo para que se cumpla que exista al menos una repetición por carta.

Por ejemplo, el número 2 es un número r primo, por lo tanto el orden puede ser 2. Ahora, a través de un cálculo matemático es posible determinar la cantidad máxima de cartas que pueden ser generadas en base a ese orden y la cantidad de símbolos que debe tener el mazo para que se cumpla el factor de repetición.

$$cardsSet(getOrden) = getOrden^2 + getOrden + 1$$

El cardsSet será la cantidad de cartas asociadas al orden y también la cantidad de símbolos que deberá tener el cardsSet. Por lo tanto, nuestro mazo estará compuesto de

$$cardsSet(2)=2^2 +2+1=7$$

La cantidad de símbolos por carta está determinada por

$$cantSímbolos(getOrden) = getOrden + 1$$

Sin embargo, es posible determinar la cantidad de cartas y símbolos totales a través de otra manera matemática

$$cardsSet(getSímbolosPorCarta) = getSímbolosPorCarta^2 - getSímbolosPorCarta + 1$$

De ambas formas será posible encontrar la cantidad total de cartas y símbolos

$$cardsSet(3)=3^2 -3+1=7$$

Como se habla de una cantidad máxima de símbolos necesariamente por carta, no se debe descuidar el factor de repetición para que exista la conexión entre carta y carta.
En la figura 1. Es posible observar tras el ejemplo inicial el patrón de repetición que podría tener el mazo con cartas que contengan 3 símbolos **diferentes**.

	#1	#2	#3	#4	#5	#6	#7
A	1	1	1	0	0	0	0
B	1	0	0	1	1	0	0
C	1	0	0	0	0	1	1
D	0	1	0	1	0	1	0
E	0	1	0	0	1	0	1
F	0	0	1	1	0	0	1
G	0	0	1	0	1	1	0

Figura 1: cardsSet con 3 símbolos por carta (orden 2).

Además de la generación del mazo, se debe generar el juego, para esto se deberán más funcionalidades dentro de la generación del mazo y del propio juego.

- **Dobble:** Corresponde a un conjunto de asociaciones, donde un usuario genera un mazo de juego a partir de una serie de parámetros a definir. Posteriormente el cardsSet será útil para un juego.

- **DobbleGame:** Corresponden a aquellos que interactúan con el juego, registrándose en esta para luego poder realizar las operaciones disponibles. Por ejemplo, iniciar un juego.

Para poder implementar a los elementos mencionados anteriormente en Java, se pueden representar de la siguiente manera:

Dobble()
Card(ArrayList<Integer> listElements)
Player(String name, int score, boolean turnStatus)
DobbleGame(ArrayList gameArea, ArrayList gameDeck, ArrayList<Player> gamePlayers, int gameNumPlayers, String gameMode)

Cada **TDA** (Tipo de Dato Abstracto) está adentro de un archivo único, por lo tanto solo bastará con revisar a cada Class.java

2.2 DISEÑO DE LA SOLUCIÓN

Para diseñar la solución, no solo es necesario utilizar los tipos de datos nativos de java, sino que también se deben crear tipos de datos específicos, a través de lo que se conoce como TDA, utilizando la siguiente estructura: Representación, constructores, selectores, modificadores y otros métodos asociados al TDA.

Los TDAs creados son utilizados para la construcción de las operaciones de otros TDAs y para las operaciones obligatorias y opcionales. Los TDAs más importantes son:

2.2.1.1 Clase Dobble

- Representación: Cada vez que se haga ingreso de las variables a los métodos respectivamente, se hará entrega de un mazo de cartas listo para iniciar un juego. Este será construido a través de los elementos que reciba la función constructora del cardsSet.
- Constructor: Dado una lista de elementos, el número de símbolos por carta, la cantidad máxima de cartas y una función que genera un número aleatorio, se crea una lista que contiene una lista con símbolos correspondientes a un set de cartas válido.
- Selectores, modificadores y otros métodos: *ver tabla 1 en anexos, página 12.*

2.2.1.2 Clase DobbleGame

- Representación: Para iniciar un juego se necesita la cantidad de jugadores, un mazo, el modo de juego y un número aleatorio que podría servir en un caso particular.
- Constructor: Dado una lista que contiene el número de jugadores, un mazo, un modo de juego específico y una seed que devuelva un número aleatorio al llamarla. Se devolverá un arreglo específico como todas las características mencionadas.

2.2.2 OPERACIONES OBLIGATORIAS

constructor de juegos: Permite crear e inicializar un juego Dobble, indicando cantidad de jugadores, modo de juego, set de cartas y otros datos que sean necesarios para tales efectos.

register: Se implementa un método a través de la clase DobbleGame que por medio de agregación, permite la modificación y la obtención de un arreglo de Player

play: Permite a los jugadores realizar jugadas por turno, los cuales son controlados internamente por el programa y expresados a través de interacciones por consola.
toString: Se implemente un método que permite producir representaciones basadas en Strings de los distintos objetos que componen el sistema. Se asume que se sobreescribirá un método toString por cada clase.

equals: Permite determinar si un objeto tiene el mismo estado que otro objeto.

2.3 ASPECTOS DE IMPLEMENTACIÓN

2.3.1 COMPILADOR

Para este proyecto es necesario utilizar algún IDE de Java, como, por ejemplo, Netbeans o InteliJ IDEA. Para este proyecto específicamente, se utilizó InteliJ IDEA en su versión 2021.3.1 (Ultimate Edition). En el caso de JDK, se utilizó la versión 11.0.13.8 para compilar el programa. Se utilizó solo funcionalidades pertenecientes a la librería estándar de Java y el programa fue probado y compilado en el sistema operativo macOS 13.

2.3.2 ESTRUCTURA DEL CODIGO

Como se utilizó el patrón MVC, el código fuente tiene tres carpetas: controller, view y model, además del main. La carpeta model contiene a todas las clases correspondientes a los “TDAs”. En la carpeta controller está la clase donde se desarrollan las funcionalidades obligatorias y opcionales. En la carpeta view está la clase del menú interactivo, donde se desarrolló todo el código para la ejecución de este.

2.4 INSTRUCCIONES DE USO

2.4.1 EJEMPLOS DE USO

Lo primero que se debe verificar es si tenemos JDK / JRE instalado en su versión 11. Luego, se debe verificar que se tenga la carpeta SRC proyecto.

En el caso de INTELIJ se debe abrir la carpeta del proyecto y ejecutar el programa. Cuando se ejecute exitosamente se deberá seleccionar una opción del menú. El menú es un menú sin fin hasta que el usuario desee terminar la ejecución del programa.

2.4.2 RESULTADOS ESPERADOS

Se espera crear una simulación de Dobble correctamente, que funcione sin errores ni ambigüedades y que conserve los fundamentos del paradigma orientado a objetos.

Junto con lo anterior, se espera trabajar correctamente con clases y métodos, ya que son fundamentales para el desarrollo del proyecto. Por último, se espera que cada método haga su procedimiento correctamente, que las clases tengan representaciones correctas y que al ejecutar el programa se tenga un menú completamente funcional que permita utilizar las distintas funcionalidades implementadas en el editor.

2.4.3 POSIBLES ERRORES

Posibles errores con las entradas y los tipos de datos deseados, es probable que el programa se caiga en el particular caso donde se ingrese algo no válido.

2.5 RESULTADOS Y AUTOEVALUACIÓN

2.5.1 RESULTADOS OBTENIDOS

Los resultados obtenidos fueron los esperados, ya que se logró crear todas las funcionalidades obligatorias y la mitad de las opcionales. El programa es completamente funcional, incluyendo en casos de contraejemplos donde las entradas colocadas son erróneas o donde se decide ingresar otro número que no corresponde a ninguna opción en el menú. Por lo tanto, se logró crear una simulación de Dobble correctamente.

2.5.2 AUTOEVALUACIÓN

La Autoevaluación se realiza de la siguiente forma: 0: No realizado – 0.25: Funciona 25% de las veces – 0.5: Funciona 50% de las veces 0.75: Funciona 75% de las veces – 1: Funciona 100% de las veces.

Para ver la tabla de Autoevaluación, ver la Tabla N°1 en anexos, página 13.

3. CONCLUSIÓN

Tras realizar y finalizar el proyecto, se puede concluir que se cumplió el principal objetivo de este proyecto e informe era introducir y comprender el paradigma orientado a eventos y aplicar el concepto de programación multiparadigma, el cual se ha cumplido correctamente.

El paradigma orientado a objetos es complejo para quien no haya salido del paradigma imperativo, sobre todo en un principio, entender la idea de que absolutamente todo es un método o un es complicado. Sin embargo, durante de este proyecto se ha presentado un entendimiento exponencial del paradigma. Si bien no se logró completar los requerimientos en su totalidad, se completó la gran mayoría y el método que no se completó quedó con gran avance lo cual significa que la idea está y que podría implementarse.

4. BIBLIOGRAFÍA Y REFERENCIAS

1. Gacitúa, D. (2021 - 2022). “Proyecto Semestral de Laboratorio”. Paradigmas de Programación. Enunciado de Proyecto Online. Recuperado de: https://docs.google.com/document/d/1pORdJwp5WJ7V3DIAQik9B58llpdxpurQRVTKPZHOpS8/edit
2. Multiples Autores. (2007). “Object-Oriented Analysis & Design”. Libro online. Recuperado de: https://drive.google.com/file/d/1f9yqn9FTNtPGMX0Yc11PtpyS7KIv322X/view?usp=sharing
3. Gacitúa, D. (2021 - 2022). “5 - P. Orientado a Objetos”. Paradigmas de Programación. Material de clases Online. Recuperado de: https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=20
4. Chacon, S. y Straub, B. (2020). “Pro Git – Todo lo que necesitas saber sobre Git’’. Libro Online. Recuperado de : https://drive.google.com/file/d/1mHJsfvGCYcIhdrmK- lBI6a1WS-U1AAPi/view

ANEXOS Tabla N°1: Autoevaluación de los requerimientos funcionales

Requerimientos Funcionales	Evaluación
DobbleGame	1
Register	1
Play	0.75
toString	1
Card	1
Equals	1
User vs User	1

Figura N°1: Diagrama de Análisis

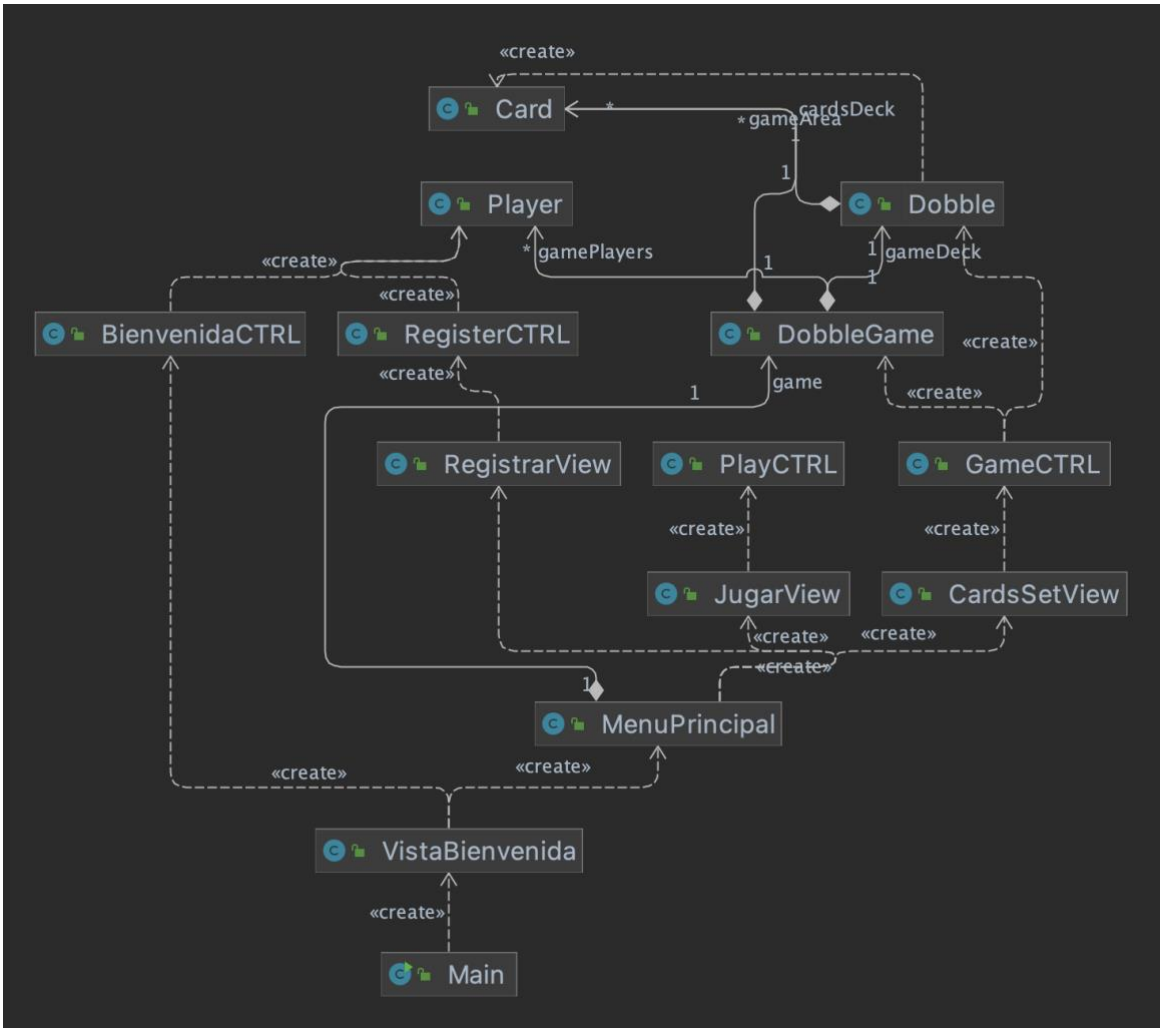

Figura N°2: Diagrama de Diseño

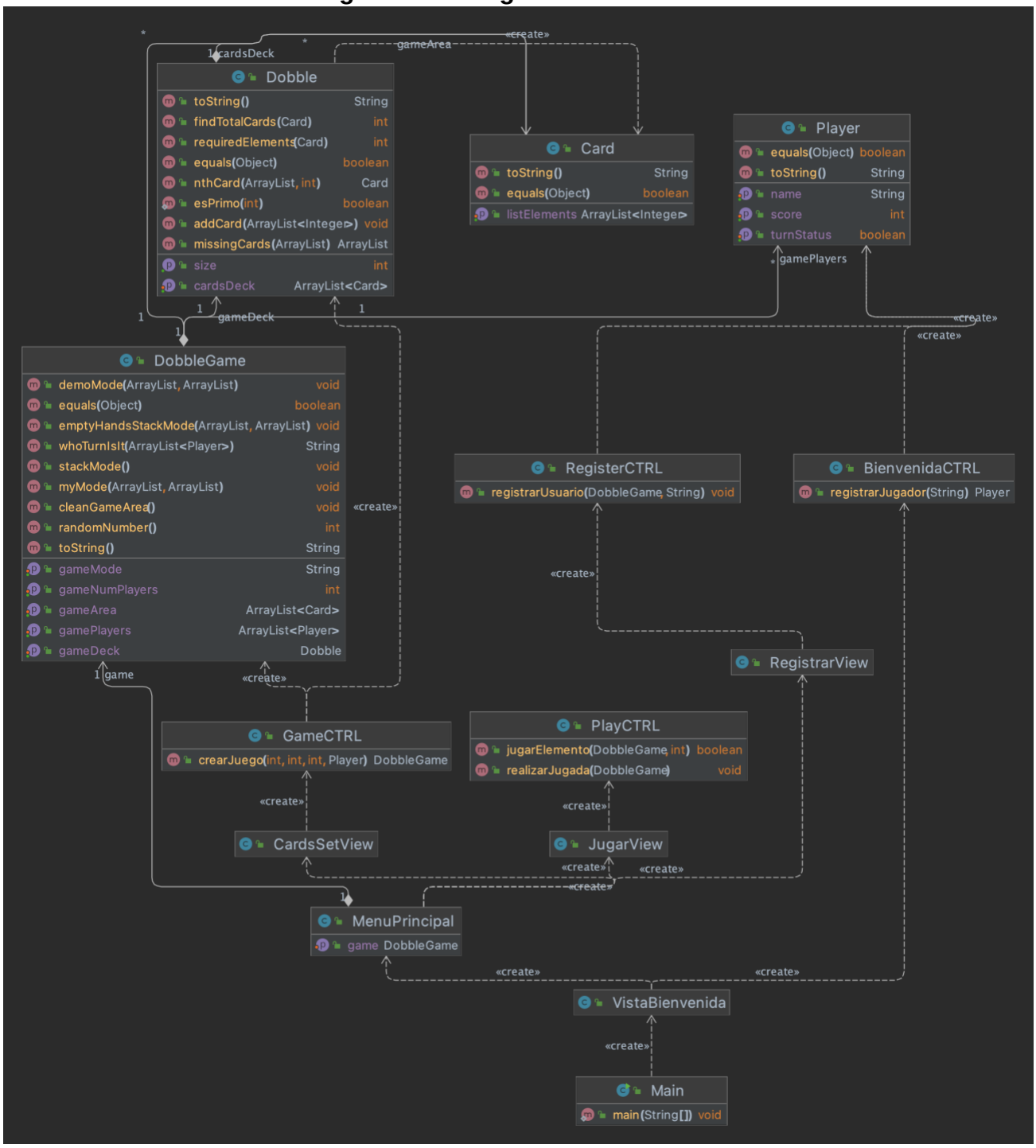

Figura N°3: Ejemplo de Ejecución

Figura N°5: Ejemplo de Ejecución

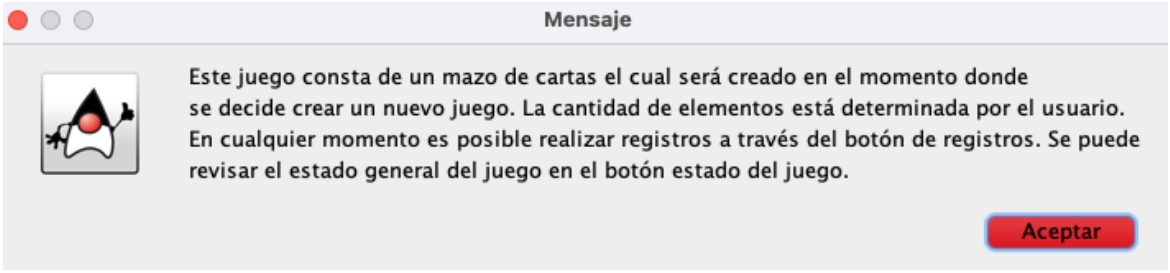

Figura N°4: Ejemplo de Ejecución