Universidad Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería
Informatica

Paradigmas de Programación
Informe Nº 3: "Connect 4"
«Simplificado en Java»

Nombre:
Alfonso Palacios Vergara

Profesor:

Gonzalo Martinez Ramirez

Fecha de entrega:

26 de Diciembre de 2024

Índice

1. Introducción

Connect 4 es un popular juego de estrategia y habilidad creado en 1974, el cual consiste en la confrontación entre dos jugadores utilizando un tablero vertical de 6 filas por 7 columnas. El objetivo es ser el primero en alinear cuatro fichas del mismo color, ya sea horizontal, vertical o diagonalmente.
Es este contexto en el cual se encuentra el contenido del presente informe, el cual tiene como objetivo presentar y detallar una implementación en lenguaje Java del juego Connect 4 desarrollado dentro de la idea del paradigma orientado a objetos, el cual se basa en el uso de objetos y clases para su funcionamiento.
El presente informe está dividido en 3 secciones: Introducción, Desarrollo y Conclusión. En la sección de Introducción se realizará la descripción del problema, en la cual se describirá el contexto del problema a resolver, y la descripción del paradigma, en la cual se caracterizará el paradigma utilizado en la realización de la solución. Por su parte, en la sección de Desarrollo se analizará el problema presentado y los requerimientos funcionales que presentó, se explicará el diseño de la solución realizada con respecto a estos mismos requerimientos, se detallarán aspectos de la implementación realizada, se presentarán las instrucciones de uso del programa y, por último, se expondrán los resultados de la experiencia. Finalmente, en la sección de Conclusión se procederá a sintetizar y concluir respecto al paradigma y el problema resuelto.

1.1. Descripción del problema

El problema abordado en este trabajo consiste en diseñar e implementar un juego de Connect 4 (o Conecta 4) de manera digital de tal modo que esté representado (por lo menos) cada juego con sus jugadores (cada uno con su nombre, color de fichas, estadísticas y fichas restantes), su tablero y su historial de juegos.
En el juego se debe permitir crear y jugar los movimientos correspondientes por cada juego, realizar la verificación de victoria, derrota o empate en el tablero y actualizar las estadísticas e información de cada jugador luego de cada movimiento.

1.2. Descripción del paradigma

El paradigma en el cual la experiencia está sustentada es el **paradigma orientado a objetos**. El paradigma orientado a objetos se basa en abstracciones de elementos del mundo real, llamadas **objetos**. Los objetos son entidades que contienen datos o propiedades (atributos) y comportamientos, en la forma de operaciones que se pueden realizar sobre y/o con esos datos (métodos). A su vez, un objeto pertenece a una **clase**: una plantilla que define la estructura de los objetos, estableciendo qué atributos y métodos tendrán. Así, un objeto corresponde a una instancia de una clase, que puede diferir respecto a otros objetos de la misma clase en sus atributos, pero jamás en sus métodos [1].
Otro aspecto fundamental del paradigma orientado a objetos es aquel de la **herencia**. La herencia permite que una clase derive de otra, heredando sus atributos y comportamientos [1]. Este principio favorece la reutilización del código y establece jerarquías de clases, donde las clases más generales pueden ser especializadas en clases más específicas [3].

A su vez, las clases poseen la capacidad de responder a diferentes mensajes de manera diferente. Esta característica se llama **polimorfismo**: el concepto de que un mismo nombre pueda agrupar a diferentes clases, relacionadas entre sí por una superclase. Todo objeto denotado por este nombre, por ende, es capaz de responder a un conjunto de operaciones comunes, lo que permite que objetos de diferentes clases sean tratados como objetos de una clase común. Por último, el paradigma orientado a objetos hace uso del concepto del **encapsulamiento**. El encapsulamiento es el concepto que hace que los detalles internos de los objetos estén ocultos (protegidos), permitiendo que solo se acceda a través de una interfaz bien definida. Este principio ayuda a reducir la complejidad y facilita la reutilización del código [3].

2. Desarrollo

2.1. Análisis del problema

El problema, como se mencionó anteriormente, reside en realizar un programa que pueda emular el juego Connect 4. Con esto en mente, son presentados 17 requerimientos funcionales (ver Cuadro 2) que, en conjunto, permiten la correcta implementación de un juego de Connect 4. Cada uno de ellos se centra en realizar una función específica, necesaria para el correcto funcionamiento del programa.
Los RFs RF01, RF02, RF03 y RF10 corresponden a métodos que deben permitir crear un jugador, una pieza, un tablero y un juego, respectivamente.
El RF14 requiere permitir la selección del jugador cuyo turno está en curso, y el RF15 exige la selección del estado actual del tablero.
El método de RF05 consiste en modificar un tablero para jugar una ficha, RF16 especifica que se debe implementar un método que actualice el juego y las estadísticas de sus jugadores cuando el mismo termine, y RF17 se refiere a un método que permita realizar un movimiento en el juego, modificando el tablero y los jugadores según corresponda.
Por último, aquellos RFs RF06, RF07 y RF08 requieren la implementación de métodos que verifiquen a un ganador vertical, horizontal y diagonal (en ese orden) en un tablero, mientras que RF09 requiere la verificación general de un ganador en un tablero. Por otra parte, RF11 pide el historial de movimientos de un juego, RF12 exige la verificación de si hay un empate en el juego, y RF13 solicita la actualización de las estadísticas de un jugador en un juego.

2.2. Diseño de la solución

Dados los requisitos funcionales que se deben cubrir, lo primero que se debe realizar es identificar los TDA (Tipos de Datos Abstractos) necesarios. Se identificaron 3 elementos fundamentales para todo juego de Connect 4, expuestos en el Cuadro 1. La estructura sigue una dependencia jerárquica: cada juego debe poseer 1 tablero y 2 jugadores.
Existen 4 tipos de métodos para los TDA: constructor, que permite crear un tipo de dato abstracto teniendo en consideración los requisitos y tipos de datos que se requieren; modificadores, que realizan modificaciones a la cantidad o el contenido de un tipo de dato abstracto; selectores, que buscan y retornan información específica dentro de diferentes tipos de datos abstractos; y otros, que se encargan de realizar operaciones que no encajan en alguna de las

categorías anteriormente mencionadas.
El enfoque principal de la solución es realizar métodos que permitan gestionar los datos previamente representados como listas en laboratorios pasados, como atributos de los TDAs correspondientes. Se optó por la representación estática del tablero como un arreglo estático, debido a que las dimensiones del tablero de un juego de Connect 4 corresponden a valores estáticos. Se separaron las estructuras generales en función de lo mostrado en el Cuadro 1 en sus propias clases abstractas cuyos métodos abstractos fueron implementados posteriormente en clases no abstractas, agregando dentro sus métodos constructores, modificadores y de pertenencia. Tener las estructuras separadas de esta forma permite obtener una organización modular del código, permitiendo un alto grado de cohesión y un grado aceptable de acoplamiento.
Para el TDA Player (jugador), se optó por un diseño que permita 7 atributos fundamentales: el ID, nombre, color de la pieza, cantidad de victorias, derrotas, empates y piezas restantes del jugador.
Por otra parte, para el TDA Board (tablero) se optó por diseñar el tablero a través de un arreglo estático, como se mencionó anteriormente. También se decidió agregar 2 atributos más: la cantidad de filas y la cantidad de columnas del tablero, en pos de la simplificación de la implementación.
Finalmente, para el TDA Game (juego) se decidió diseñar al tipo de dato abstracto con 2 TDA Players, un TDA Board, un entero que sea el id del color de la pieza cuyo jugador sea el turno actual, y una lista dinámica que represente el historial de movimientos hechos.

2.3. Aspectos de implementación

Teniendo en cuenta lo anterior, las implementaciones de los RF mencionados en el Cuadro 2 se encuentran en las clases que heredan de cada TDA.
Se utilizó el SDK de Java OpenJDK v11 para la realización del código, utilizando a su vez el IDE IntelliJ IDEA Community Edition 2024.1 y el wrapper Gradle v8.5. A su vez, se utilizó un notebook HP Pavillon x360 con Windows 10 para la totalidad del proceso de programación.

2.4. Instrucciones de uso

Para utilizar los archivos se deben tener todos dentro del mismo directorio, ya que se requieren entre ellos (en específico, el archivo 'main' requiere de todos los archivos de clases de TDA, que a su vez requieren de las clases abstractas de TDA). Esto implica que ciertos métodos que están en un archivo son utilizados en otro (en especial de TDA a las clases que los implementan).
A la hora de ejecutar el programa, hay 2 opciones: se puede ejecutar en el software de IntelliJ, o en el símbolo del sistema (intérprete de comandos de Windows o el sistema operativo que se utilice). Para ejecutarlo en IntelliJ, siga el path:

- *Lab3_RUT_AlfonsoPalaciosVergara\codigoFuente_RUT_PalaciosVergara\main\java\org.palaciosalfonso.lab3\Main.java*

Para abrir el archivo Main del programa, y presione el botón triangular (presumiblemente verde) de arriba a la derecha para ejecutar el programa. Para ejecutar en la terminal, abra la terminal en el directorio base del programa (Lab3_RUT_AlfonsoPalaciosVergara) y utilice el comando '*gradlew.bat build*' seguido del comando '*gradlew.bat run*'.
Toda interacción del usuario con el programa es a través del menú creado al ejecutar el archivo Main. Una vez ejecutado el programa, introduzca por la terminal (ya sea de IntelliJ o la terminal del sistema) la opción que desea ejecutar según el programa indique, o introduzca el texto que el programa requiere dependiendo del contexto.

2.5. Resultados y autoevaluación

En general, se pudo implementar cada RF de manera efectiva, complementándose de buena manera con los TDAs y permitiendo representar fielmente un juego de Connect 4. Se realizaron pruebas en todos los casos borde, tales como introducir piezas en una columna hasta que ya no pueda soportar más piezas, intentar tener 2 o más formas en las cuales un jugador gana para verificar si el programa se rompía o introducir números fuera del rango permitido u otros tipos de variables (entre otras pruebas), sólo retornando algo no esperado cuando se introducía al método algo que no correspondiera al tipo de variable esperado, lo cual es aceptable, siendo que cada método se especializa en un tipo de operación y nunca serán llamadas con algo indebido a través del menú con el usuario. El resultado de la autoevaluación tiene puntaje máximo en todos los ítems, puesto que el grado de alcance de todos los RF se considera óptimo.

3. Conclusiones

Los alcances de la implementación se consideran aceptables para una buena representación digital en Java de Connect 4. Así mismo, se da por concluido el objetivo del informe.
El paradigma posee el beneficio de la capacidad para la abstracción fiel con respecto a entidades del mundo real, ocultando la complejidad y enfocándose solo en las funcionalidades necesarias. A su vez, posee la seguridad brindada por el encapsulamiento intrínseco del paradigma; la reutilización de código a través de la herencia; la capacidad de que los objetos de diferentes clases puedan ser tratados de manera uniforme, lo que simplifica la escritura de código y mejora la flexibilidad debido al polimorfismo; y la modularidad brindada por el uso de objetos y clases, lo que hace que el código sea más fácil de entender, desarrollar y mantener, ya que los componentes del sistema están bien definidos y encapsulados.
Sin embargo, aunque el paradigma orientado a objetos favorece la modularidad y la organización, también puede llevar a una mayor complejidad en el diseño del sistema. La creación de múltiples clases y relaciones entre ellas puede ser difícil de manejar, especialmente en sistemas grandes y complejos. Además, la planificación y la jerarquía de clases deben ser cuidadosamente diseñadas desde el principio para evitar problemas futuros. A su vez, el enfoque de diseño basado en objetos puede llevar a una sobreingeniería del sistema, creando una cantidad excesiva de clases y objetos para representar incluso las tareas más simples. Este tipo de diseño puede resultar innecesariamente complicado y generar un código más difícil de mantener. Todas estas características suponen limitaciones del paradigma en sí, y

de todos las implementaciones del mismo.
Todas estas características afectaron de gran manera a la implementación de la solución, puesto que la capacidad de abstracción significó que realizar los TDA fuera mucho más sencillo, mientras que, por otra parte, se tendió a realizar sobreingeniería en diferentes puntos del desarrollo del programa, por nombrar algunos ejemplos.
El paradigma orientado a objetos posee un enfoque diferente al paradigma lógico, como se puede apreciar comparando el uno con el otro. El paradigma orientado a objetos (POO) se basa en la utilización de objetos, instancias de una clase, que contienen atributos y métodos que definen su comportamiento, mientras que el paradigma lógico (PL) utiliza hechos y reglas lógicas que definen relaciones entre entidades. El POO es de enfoque imperativo, mientras que el PL es declarativo. Bajo el POO el programador es responsable y tiene completo control con respecto a la ejecución del programa, mientras que bajo el PL se está a la merced de un motor de inferencia. A su vez, el POO en general es más eficiente con respecto al rendimiento, debido a que no posee el bactracking y la inferencia innata del PL. En conclusión, el POO es ideal para sistemas complejos que requieren modularidad, reutilización de código y la interacción entre entidades con comportamientos definidos, mientras que el PL es más adecuado para aplicaciones que requieren inferencia automática, razonamiento y manipulación de relaciones lógicas complejas, como sistemas expertos, bases de datos lógicas y problemas de inteligencia artificial. Sin embargo, su uso está más limitado en aplicaciones generales debido a la falta de control sobre el proceso de ejecución y las posibles ineficiencias en términos de rendimiento.

Referencias

[1] Booch, G. (1991). *Object-oriented analysis and design with applications. Addison-Wesley.* (Accessed: 2024-12-26)

[2] Gabriel, R. P. (1996). *Patterns of software. Oxford University Press.* (Accessed: 2024-12-26)

[3] Liskov, B. (1987). *A behavioral notion of subtyping.* ACM SIGPLAN Notices. (Accessed: 2024-12-26)

4. Anexos

Elemento	**Descripción**
Game	Juego de Connect 4 compuesto por 2 jugadores, un tablero y un historial de juego
Board	Tablero de juego de Connect 4 compuesto por filas y columnas, con 42 espacios totales
Player	Jugador de un juego de Connect 4 compuesto por su id, nombre, el color de sus piezas, cantidad de victorias, derrotas y empates, y piezas restantes

Cuadro 1: Elementos principales Connect 4

RF	Descripción
1	TDA Player - constructor. método que permite crear un jugador.
2	TDA Piece - constructor. método que crea una ficha de Conecta4.
3	TDA Board - constructor. método que permite crear un tablero de Conecta4.
4	TDA Board - otros - sePuedeJugar?. método que permite verificar si se puede realizar más jugadas en el tablero.
5	TDA Board - modificador - jugar ficha. método que permite jugar una ficha en el tablero
6	TDA Board - otros - verificar victoria vertical. método que permite verificar el estado actual del tablero y entregar el posible ganador vertical.
7	TDA Board - otros - verificar victoria horizontal. método que permite verificar el estado actual del tablero y entregar el posible ganador horizontal.
8	TDA Board - otros - verificar victoria diagonal. método que permite verificar el estado actual del tablero y entregar el posible ganador diagonal.
9	TDA Board - otros - entregarGanador. método que permite verificar el estado actual del tablero y entregar el posible ganador.
10	TDA Game - constructor. método que permite crear una nueva partida
11	TDA Game - otros - history. método que genera un historial de movimientos de la partida.
12	TDA Game - otros - esEmpate?. método que verifica si el estado actual del juego es empate
13	TDA Player - otros - actualizarEstadisticas. método que actualiza las estadísticas del jugador, ya sea victoria, derrotas o empates.
14	TDA Game - selector - getCurrentPlayer. método que obtiene el jugador cuyo turno está en curso.
15	TDA Game - selector - boardGetState. método que entrega el estado actual del tablero en el juego.
16	TDA Game - modificador - endGame. método finaliza el juego actualizando las estadísticas de los jugadores según el resultado.
17	TDA Game - modificador - realizarMovimiento. Función que realiza un movimiento.

Cuadro 2: Requerimientos funcionales