Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática
Paradigmas de Programación

Paradigmas de Programación

Proyecto semestral de laboratorio

Versión 1 - actualizada al 17/11/2024

Laboratorio 3 (Paradigma Orientado a Objetos - Lenguaje Java)

Versión 0.9 20/10/24 : Versión preliminar

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

Enunciado General: Procure consultar los aspectos generales del proyecto de laboratorio en el documento general.

Fecha de Entrega: Ver calendario clase a clase donde se señala el hito

Objetivo del laboratorio: Aplicar conceptos del paradigma de programación funcional usando el lenguaje de programación Scheme en la resolución de un problema acotado.

Resultado esperado: Juego Conecta4

Profesor responsable: @ Gonzalo Martinez (gonzalo.martinez@usach.cl)

Recomendaciones: El laboratorio está diseñado como un conjunto de ejercicios a abordar bajo cada paradigma. En este sentido, el desarrollo del laboratorio constituye un espacio para practicar y prepararse además para la evaluación de cátedra del correspondiente paradigma. Por tanto, se recomienda incorporar en sus hábitos de estudio/trabajo el desarrollo de las funcionalidades de forma diaria. En total el laboratorio 1 lista N funcionalidades a cubrir. Para alcanzar la nota de aprobación, se deben cubrir las M primeras y para la nota máxima se pueden cubrir Q más. Procure destinar tiempo para analizar y hacer una propuesta de diseño para el laboratorio completo antes de proceder a la implementación de la solución. No es necesario que sus predicados implementen comprobación de tipo, esto es opcional.

Requerimientos No Funcionales (RNF)

Algunos son ineludibles/obligatorios, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

Requerimientos No Funcionales. Algunos son ineludibles/obligatorios, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

1. **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales solicitados.
2. **(obligatorio) Lenguaje y herramientas de trabajo:** En el presente laboratorio, se debe realizar una implementación basada en **Java.**
3. La implementación de este laboratorio debe usar un SDK preciso, concretamente OpenJDK versión 11 (link). **Si bien existen versiones mayores de OpenJDK, tales como la versión 16 a 21, el laboratorio será ejecutado con la versión 11**.
4. Se debe utilizar el IDE IntelliJ IDEA Community Edition 2024.1 (link). **Se subraya que el uso de IDE es obligatorio para este laboratorio. No basta con usar un editor de texto y extensiones asociadas.**
5. Para el proceso de compilación, se debe integrar a su proyecto la herramienta *Gradle* (link a guía para crear proyecto Java usando Gradle). El entregable final debe ser un proyecto creado con el IDE integrado con Gradle (y Gradle Wrapper) e incluyendo solo código fuente y archivos de configuración (no debe incluir archivos binarios ni archivos .class). **No se pueden utilizar librerías de terceros, tales como, Lombok u otros** excepto lo indique explícitamente el enunciado.

   ADVERTENCIA: su Informe debe especificar las instrucciones de compilación, la herramienta de compilación usada (Gradle) y el ambiente de desarrollo donde se ejecutó el entregable (Sistema Operativo y versión exacta de JDK).

   En caso de no poder compilar y ejecutar su proyecto con las instrucciones que usted indique, el laboratorio no será revisado. **Asimismo, si entrega su laboratorio sin Gradle, su laboratorio no será evaluado**.

En general debe solo usar bibliotecas que son parte del *core* de Java (lo nativo de Java lo puede encontrar en este enlace) , por lo que no se debe agregar dependencias externas al proyecto, con excepción de las indicadas por sus profesores que redactaron este enunciado.

Independientemente si es una aplicación por consola o por interfaz gráfica, sólo debe realizar un entregable. Asimismo, su aplicación debe ser Java. Solo debe realizar un entregable con un lenguaje de programación.

6. **(obligatorio) Interacciones con el programa:** Todas las interacciones con el programa deben ser mediante **consola/terminal o una interfaz gráfica (GUI)**. Puede recurrir al uso de System.in y System.out (en Java). Para sus

funcionalidades, solo se permite el uso de la biblioteca estándar de Java (sin tener dependencias externas, como las disponibles en formato JAR).

7. **(obligatorio) Uso del paradigma:** Su solución debe demostrar la aplicación del paradigma orientado a objetos. No basta con que su solución esté implementada en Java. Su diseño y correspondiente implementación debe seguir los lineamientos del paradigma Orientado a Objetos. **Si usted entrega todo contenido en un solo archivo Main y sin ocupar clases, se reprobara inmediatamente.**

8. **(obligatorio) Prerrequisitos:** Para cada funcionalidad se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada. Ej: Para evaluar la funcionalidad *add*, debe estar implementada la funcionalidad *authentication*.

9. **(1 pto) (obligatorio) Documentación JavaDoc:** Se debe documentar el código indicando una breve descripción de las clases e interfaces creadas, sus atributos, métodos y relaciones. Utilice comentarios como lo presenta Javadoc (link). Este comando de java SDK genera un archivo HTTP que permite de manera muy fácil saber - mediante un browser - cómo usted organizó la documentación de su código.

10. **(1 pto) Organización del código:** Se debe cuidar la organización del código (orden y claridad). Procure que su diseño de clases no viole los principios de bajo acoplamiento y alta cohesión que muestra la adopción de parte del/la estudiante de las buenas prácticas de programación.

11. **(1.5 pto) Diagrama de análisis:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML a nivel de análisis que describa las entidades y relaciones del problema abordado. Este diagrama se debe crear ANTES del proceso de desarrollo del software (antes de la etapa de codificación o programación).

12. **(1.5 pto) Diagrama de diseño:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML tras la implementación de la solución, este diagrama debe ser coherente con la implementación en código de su solución incluyendo todas las clases de su código. **Este diagrama se debe crear después del desarrollo de la solución.**

13. **(1 pto) Uso de git:** Historial de trabajo en GitHub tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 2 semanas**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 ptos. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.25 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

Requerimientos Funcionales (RF)

La nota correspondiente al apartado de RF comienza en 1.0 y por cada RF correcto dicho puntaje escrito en el enunciado se suma a la nota base.

Para que el requerimiento sea evaluado, DEBE cumplir con el prerrequisito de evaluación y requisito de implementación. En caso contrario la función no será evaluada.

Requerimientos Funcionales. Para que el requerimiento sea evaluado, DEBE cumplir con el prerrequisito de evaluación y requisito de implementación. En caso contrario la función no será evaluada. El total de requerimientos permiten alcanzar una nota mayor que 7.0, por lo que procura realizar las funciones que consideres necesarias para alcanzar un 7.0. Si realizas todas las funciones y obtienes el puntaje máximo, la nota asignada será igualmente un 7.0. El puntaje de desborde se descarta.

1. **RF01 Clases y estructuras que forman el programa (1 pts)**.

   Especificar e implementar abstracciones apropiadas para el problema. **Recomendamos leer el enunciado completo y ver el material complementario presentado al comienzo de este enunciado a fin de que analice el problema y determine el o los TDAs y representaciones apropiadas para la implementación de cada uno**. Luego, planifique bien su enfoque de solución de manera que los TDAs y representaciones escogidos sean aplicables a ambos tipos de funciones.

   Para la implementación debe regirse por la estructura de especificación e implementación de TDA vista en clases: Representación, Constructores, Funciones/Métodos de Pertenencia, Selectores, Modificadores y Otros Métodos. Procurar hacer un uso adecuado de esta estructura a fin de no afectar la eficiencia de sus Métodos. En el resto de los Métodos se debe hacer un uso adecuado de la implementación del TDA (ej: usar selectores, modificadores, constructores, según sea el caso. No basta con implementar un TDA y luego NO hacer uso del mismo). **Solo implementar las clases y métodos estrictamente necesarios dentro de esta estructura**.

   Si alguna clase contiene métodos que no son utilizados se procede a descontar puntaje relativo a este RF.

   Como parte del diseño orientado a objetos de su solución, **considere como mínimo modelar las siguientes entidades (y sus respectivas relaciones)** dentro de su programa:

   a. **Game**: Juego
   b. **Board:** Tablero
   c. **Player:** Jugador

Prerrequisitos para evaluación	Ninguno
Requisitos de implementación	- Como ***mínimo*** su diseño debe contemplar los siguientes TDAs: **Game, Board, Player. Usted puede crear más clases y TDAs si lo considera necesario. El diseño de software e implementación del programa queda a su disposición.** - Exprese los TDAs necesarios a través de **interfaces y/o herencia** que luego deben ser **implementadas mediante clases**. - Especificar representación de manera clara para cada clase implementada (en el informe y en el código a través de comentarios en formato JavaDoc). Luego implementar **constructores, getters, setters, modificadores** y otros métodos según lo requerido en los requisitos a continuación. - Sólo debe implementar los getters y setters y métodos en general que efectivamente se utilizan en su código. **No debe existir código declarado, pero que no se utilice en el resto de las clases declaradas**.

2. **RF02. (1 pts) Menú interactivo por terminal*:** su programa debe incluir un menú por terminal/consola que permita la interacción del usuario con la solución, implementando las entidades y funcionalidades que permitan crear *las clases especificadas en el RF1. Para ser más específico, el menú debe exponer los requisitos para que los revisores puedan evaluarlos. Si el RF no es expuesto mediante el menú interactivo, no será revisado. Si hay un requerimiento de agregar algo, debe ser expuesto por el menú de alguna forma.* El sistema debe proveer retroalimentación al usuario de las acciones realizadas por él (si la acción pudo concretarse con éxito o no).

Las instrucciones de funcionamiento de su solución deben estar explicadas y documentadas en su Informe. **IMPORTANTE**: La implementación de este menú debe manejarse de manera independiente del resto de las clases que conforman la solución al problema establecido.

- **IMPORTANTE: EN ESTA ENTREGA NO EXISTE UN SCRIPT DE EJECUCIÓN. TODAS LAS INTERACCIONES SE REALIZARÁN MEDIANTE ESTE MENÚ. AL FINAL DEL DOCUMENTO SE ENCUENTRA UN EJEMPLO DE MENÚ. EL DISEÑO E IMPLEMENTACIÓN DEL MENÚ QUEDA A DISPOSICIÓN DEL ESTUDIANTE.**

3. **RF03. (0.1 pts) TDA Player - constructor**. Método que permite crear un jugador.

Prerrequisitos para evaluación (req. funcionales)	1-2
Parámetros de entrada	id (int) X name (string) X color (string) X wins (int) X losses (int) X draws (int) X remaining-pieces (int) donde: ● wins: número de veces que ha ganado ● losses: número de veces que ha ganado ● draws: número de veces que ha estado en empate ● remaining-pieces: cantidad de fichas actuales ○ La regla del juego es que se comienza con 21 fichas, pero ustedes pueden crear juegos más cortos con una cantidad de fichas mayor a 4 y menor a 21. ○ La cantidad de fichas por jugador no afecta el desarrollo del laboratorio ni las jugadas a realizar.

4. **RF04. (0.1 pts) TDA Piece - constructor**. Método que crea una ficha de Conecta4.

Prerrequisitos para evaluación (req. funcionales)	3
Parámetros de entrada	color (string)

5. **RF05. (0.1 pts) TDA Board - constructor**. Método para crear un tablero de Conecta4.

Prerrequisitos para evaluación (req. funcionales)	4
Parámetros de entrada	No recibe parámetros de entrada

6. **RF06. (0.1 pts) TDA Board - otros - sePuedeJugar**. Método que permite verificar si se puede realizar más jugadas en el tablero.

Prerrequisitos para evaluación (req. funcionales)	5
Requisitos de implementación	Verifica si en el tablero se puede realizar una jugada. Para realizar una jugada en el tablero debe existir al menos una posición disponible y fichas disponibles.
Parámetros de entrada	No recibe parámetros de entrada

7. **RF07. (0.7 pts) TDA Board - modificador - jugar ficha**. Jugar una ficha en el tablero

Prerrequisitos para evaluación (req. funcionales)	6
Requisitos de implementación	El método debe colocar la ficha en la posición más baja disponible de la columna seleccionada.
Parámetros de entrada	Column (int) X Piece (piece)

8. **RF08. (0.2 pts) TDA Board - otros - verificar victoria vertical**. Método que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma vertical.

Prerrequisitos para evaluación (req. funcionales)	7
Requisitos de implementación	El método debe verificar si hay 4 fichas consecutivas del mismo color en cualquier columna. El retorno del método debe indicar quien es el ganador si es que existe alguno.

9. RF09 (0.1 pts) TDA Board - otros - verificar victoria horizontal. Método que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma horizontal.

Prerrequisitos para evaluación (req. funcionales)	8
Requisitos de implementación	El método debe verificar si hay 4 fichas consecutivas del mismo color en cualquier fila. El retorno del método debe indicar quien es el ganador si es que existe alguno.

10. RF10. (0.3 pts) TDA Board - otros - verificar victoria diagonal. Predicado que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con la regla de conectar 4 fichas de forma diagonal.

Prerrequisitos para evaluación (req. funcionales)	9
Requisitos de implementación	El método debe verificar si hay 4 fichas consecutivas del mismo color en cualquier diagonal (ascendente o descendente). El retorno del método debe indicar quien es el ganador si es que existe alguno.

11. RF11. (0.1 pts) TDA Board - otros - entregarGanador. Predicado que permite verificar el estado actual del tablero y entregar el posible ganador que cumple con cualquiera de las reglas previamente definidas.

Prerrequisitos para evaluación (req. funcionales)	10
Requisitos de implementación	Utilizar los métodos de verificación vertical, horizontal y diagonal para determinar si hay un ganador.

12. RF12. (0.1 pts) TDA Game - constructor. Método que permite crear una nueva partida.

Prerrequisitos para evaluación (req. funcionales)	11

13. RF13. (0.1 pts) TDA Game - otros - history. Método que genera un historial de movimientos de la partida.

Prerrequisitos para evaluación (req. funcionales)	12

14. RF14. (0.1 pts) TDA Game - otros - esEmpate. Método que verifica si el estado actual del juego es empate.

Prerrequisitos para evaluación (req. funcionales)	13
Requisitos de implementación	Verificar si el tablero se encuentra completo o si ambos jugadores se han quedado sin fichas. Tablero completo significa que todas las posiciones del mismo se encuentran ocupadas.
Retorno	*boolean (True si es empate, False si no)*

15. RF15. (0.1 pts) TDA Player - otros - actualizarEstadisticas. Predicado que actualiza las estadísticas del jugador, ya sea victoria, derrotas o empates.

Prerrequisitos para evaluación (req. funcionales)	14
Requisitos de implementación	Actualizar las estadísticas del jugador (victorias, derrotas o empates) después de finalizar un juego.

16. RF16. (0.1 pts) TDA Game - selector - getCurrentPlayer. Método que obtiene el jugador cuyo turno está en curso.

Prerrequisitos para evaluación (req. funcionales)	15
Requisitos de implementación	Obtener el jugador cuyo turno está en curso.

17. RF17. (0.1 pts) TDA Game - selector - boardGetState. Método que entrega por pantalla el estado actual del tablero en el juego.

Prerrequisitos para evaluación (req. funcionales)	16
Requisitos de implementación	Obtiene e imprime una representación del estado actual del tablero.

18. RF18. (0.1 pts) TDA Game - modificador - endGame. Método que finaliza el juego actualizando las estadísticas de los jugadores según el resultado y no permitiendo futuras jugadas.

Prerrequisitos para evaluación (req. funcionales)	17
Requisitos de implementación	Finalizar el juego, actualizando las estadísticas de los jugadores según el resultado.

19. RF19. (2 pts) TDA Game - modificador - realizarMovimiento. Método que permite a un jugador realizar un movimiento.

Prerrequisitos para evaluación (req. funcionales)	18

Requisitos de implementación	Este método debe realizar lo siguiente: - Verificar que sea el turno del jugador correcto. - Actualizar el estado del juego después de realizar un movimiento: ● Colocar la pieza en el tablero. ● Cambiar el turno al otro jugador. ● Disminuir la cantidad de fichas del jugador que realizó el movimiento. ● Actualizar el historial de movimientos. - Verificar si hay un ganador o si el juego ha terminado en empate después del movimiento. - Si el juego ha terminado, actualizar las estadísticas de ambos jugadores.
Parámetros de entrada	player (player) X column (int)

EJEMPLO DE MENÚ Y FLUJO DE EJECUCIÓN

ESTO ES SÓLO UN EJEMPLO, EL DESARROLLO DEL MENÚ QUEDA A LIBRE DISPOSICIÓN

Conecta4 - Menú Principal

Bienvenido al juego Conecta4

Seleccione una opción:

1. Crear nuevo juego

2. Visualizar estado actual

3. Realizar jugada

4. Ver estadísticas generales

5. Salir del juego

Ingrese su opción: 1

Crear Nuevo Juego

--- Configuración Jugador 1 ---

Ingrese nombre del jugador 1: Juan

Color asignado: Rojo

--- Configuración Jugador 2 ---

Ingrese nombre del jugador 2: María

Color asignado: Amarillo

--- Configuración del Juego ---

Ingrese cantidad de fichas por jugador (4-21): 10

¡Juego creado exitosamente!

Presione enter para volver al menú principal...

```
### Realizar Jugada ###
Turno de: Juan (Rojo)
Fichas restantes: 10

Seleccione columna (1-7): 4

Movimiento realizado:
| - - - - - - - |
| - - - - - - - |
| - - - - - - - |
| - - - - - - - |
| - - - - - - - |
| - - - R - - - |

### Realizar Jugada ###
Turno de: María (Amarillo)
Fichas restantes: 10

Seleccione columna (1-7): 3

Movimiento realizado:
| - - - - - - - |
| - - - - - - - |
| - - - - - - - |
| - - - - - - - |
| - - - - - - - |
| - - A R - - - |
```

Realizar Jugada

Turno de: Juan (Rojo)

Fichas restantes: 9

Seleccione columna (1-7): 4

Movimiento realizado:

| - - - - - - - |

| - - - - - - - |

| - - - - - - - |

| - - - - - - - |

| - - - R - - - |

| - - A R - - - |

Realizar Jugada

Turno de: María (Amarillo)

Fichas restantes: 9

Seleccione columna (1-7): 3

Movimiento realizado:

| - - - - - - - |

| - - - - - - - |

| - - - - - - - |

| - - - - - - - |

| - - A R - - - |

| - - A R - - - |

```
### Realizar Jugada ###
Turno de: Juan (Rojo)
Fichas restantes: 8

Seleccione columna (1-7): 4

Movimiento realizado:
| - - - - - - - |
| - - - - - - - |
| - - - - - - - |
| - - - R - - - |
| - - A R - - - |
| - - A R - - - |

### Realizar Jugada ###
Turno de: María (Amarillo)
Fichas restantes: 8

Seleccione columna (1-7): 3

Movimiento realizado:
| - - - - - - - |
| - - - - - - - |
| - - - - - - - |
| - - A R - - - |
| - - A R - - - |
| - - A R - - - |
```

Realizar Jugada

Turno de: Juan (Rojo)

Fichas restantes: 7

Seleccione columna (1-7): 4

¡VICTORIA! Juan (Rojo) ha ganado con 4 en línea vertical

Estado final del tablero:

| - - - - - - - |

| - - - - - - - |

| - - - R - - - |

| - - A R - - - |

| - - A R - - - |

| - - A R - - - |

Estadísticas Actualizadas

Juan (Rojo):

- Victorias: 1

- Derrotas: 0

- Empates: 0

María (Amarillo):

- Victorias: 0

- Derrotas: 1

- Empates: 0

```
Fichas restantes:
- Juan (Rojo): 6
- María (Amarillo): 7

¿Desea jugar otra partida? (S/N): N

### Conecta4 - Menú Principal ###
1. Crear nuevo juego
2. Visualizar estado actual
3. Realizar jugada
4. Ver estadísticas generales
5. Salir del juego

Ingrese su opción: 5

¿Está seguro que desea salir? (S/N): S
¡Gracias por jugar Conecta4!
```

Leer si tienen problemas de compilación y ejecución con Gradle

La entrega se ejecutará vía terminal/consola usando gradle. Para esto los revisores ocupan los siguientes comandos. Deben verificar si su entrega funciona con los siguientes comandos que permiten compilar y ejecutar. No se revisará con el IDE ni tampoco se modificará

Linux/Unix:

1. Compilación:
   ./gradlew build
2. Ejecución:
   ./gradlew run

Windows:

1. Compilación:
   gradlew.bat build
2. Ejecución:
   gradlew.bat run

Si la clase Scanner de Java, o el programa en sí con el uso del menú interactivo arroja un error, posiblemente es porque falta una configuración en el archivo de configuración de Gradle (build.gradle). Para esto usen de referencia la siguiente configuración:

build.gradle en Groovy (Si realizaron el archivo de configuración build.gradle en otro lenguaje, tal como Kotlin, debe buscar el equivalente de la siguiente configuración).

```
# IMPORTANTE: SI SU PROGRAMA FALLA EN TERMINAL/CONSOLA NECESITA AGREGAR ESTO
plugins {
   id 'application'
}


repositories {
   mavenCentral()
}


# IMPORTANTE: SI SU PROGRAMA FALLA EN TERMINAL/CONSOLA NECESITA AGREGAR ESTO
run {
   standardInput = System.in
}
```

```
# Acá debe ir la ruta en donde se encuentra el main, org es la carpeta de mayor jerarquía, luego la carpeta example y luego la clase Main. OJO: NO ES QUE SU PROGRAMA TENGA QUE TENER LA CARPETA org, O LA CARPETA example, ES SOLO UN EJEMPLO.
application {
   mainClass = 'org.example.Main'
}
```

Al momento de usar de referencia la configuración anterior, eliminen los comentarios (#)

// mvn:

mvn run

// gradle

gradle run