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

# Paradigmas de Programación

# Proyecto semestral de laboratorio

Versión 1.0 - actualizada al 19/11/2023

## Laboratorio 3 (Paradigma Orientado a Objetos - Lenguaje Java)

(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 orientado a objetos usando el lenguaje de programación Java en la resolución de un problema acotado.

**Resultado esperado:** Programa para realizar una simulación de un chatbot.

**Profesor responsable:** **Edmundo Leiva** y **Gonzalo Martinez**, etiquetar a edmundo.leiva@usach.cl y 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.

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 clases y métodos implementen comprobación de tipo, esto es opcional.**

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

- ☐ **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales solicitados.

- ☐ **(obligatorio) Lenguaje y herramientas de trabajo:** En el presente laboratorio, se debe realizar una implementación basada en **Java.**
- ☐ **En caso de usar Java:**
  La implementación de este laboratorio debe ser en el lenguaje de programación Java, utilizando 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**. Además, puede utilizar alguno de los siguientes IDEs (a su elección) - la versión indicada son las versiones mínimas a utilizar, puede utilizar una versión mayor - :
  **El uso de IDE es obligatorio para este laboratorio. No basta con usar un editor de texto y extensiones asociadas. Pueden ocupar versiones modernas de estos IDEs:**
    a. Apache NetBeans 12 (link)
    b. Eclipse IDE 2020-09 (link)
    c. IntelliJ IDEA Community 2020.3 (link)

  Para el proceso de compilación, se puede elegir entre una de las siguientes herramientas:
    1. Un *script de compilación manual* en Batch (Windows) o Bash (GNU/Linux, macOS o Windows Subsystem for Linux (WSL)) que utilice el comando `javac` para compilar sus clases. El entregable debe ser una carpeta con el código fuente y el script de compilación (no debe incluir archivos binarios ni archivos .class)
    2. 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 de su elección, 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 3eros, tales como, Lombok u otros**.

  Independientemente de la opción escogida para compilar el proyecto, su Informe debe especificar las instrucciones de compilación, la herramienta de compilación usada (script manual o 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 script o sin gradle, su laboratorio no será evaluado**.

Si implementan una interfaz gráfica, dicha implementación debe utilizar **AWT** y/o **Swing** como biblioteca de entorno gráfico. Estas bibliotecas son parte del *core* de Java, por lo que no se debe agregar dependencias externas al proyecto.

**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.**

- ☐ **(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).

- ☐ **(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.**

- ☐ **(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*.

- ☐ **(1 pto) (obligatorio) Documentación JavaDoc:** Se debe documentar el código indicando una breve descripción de las clases creadas, sus atributos, métodos públicos y relaciones. Procure utilizar comentarios tipo Javadoc (link) para esto.

- ☐ **(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.

- ☐ **(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.

- ☐ **(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.**

**(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.** **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.**

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

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**.

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. System: Representa el sistema con sus atributos y comportamientos.
b. Chatbot: Representa el chatbot con sus atributos y comportamientos.
c. Flow: Representa el flujo con sus atributos y comportamientos.
d. Option: Representa cada option con sus atributos y comportamientos.
e. User: Representa a cada usuario y su comportamiento asociado.

| | |
|---|---|
| **Prerrequisitos para evaluación** | Ninguno |
| **Requisitos de implementación** | - Como ***mínimo*** su diseño debe contemplar los siguientes TDAs, **System**, **Chatbot**, **Flow**, **Option, User.**<br><br>- Exprese los TDAs necesarios a través de **interfaces y/o herencia** que luego deben ser **implementadas mediante clases**.<br><br>- Especificar representación de manera clara para cada clase implementada (en el informe y en el código a través de comentarios). Luego implementar **constructores, getters, setters** y otros métodos según lo requerido en los requisitos a continuación.<br><br>- Sólo debe implementar los getters y setters que efectivamente se utilizan en su código. **No puede existir código declarado que no se utilice en el resto de clases**. |

**RF 2. (0.5 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 *system*, *chatbot, option, entre otros. Para ser más específico, el menú debe exponer los requisitos para que los revisores puedan evaluarlos. Si hay un RF de constructor, debe ser expuesto por el menú para evaluarlo. 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 documentadas en su Informe. La implementación de este menú debe manejarse de manera independiente del resto de las clases que conforman el problema.

**Un ejemplo del menú se representa a continuación (usted puede ofrecer más opciones, desplegar otros submenús, etc)**:

### Sistema de Chatbots - Inicio ###
1. Login de Usuario

2. Registro de Usuario

INTRODUZCA SU OPCIÓN:

### Sistema de Chatbots - Registro ###

1. Registrar usuario normal

2. Registrar usuario administrador

INTRODUZCA SU OPCIÓN:

### Sistema de Chatbots - Registro Usuario Administrador ###

INTRODUZCA NOMBRE DEL USUARIO ADMINISTRADOR:

### Sistema de Chatbots - Login ###

INTRODUZCA NOMBRE DE USUARIO:

> El usuario ingresa el nombre de usuario y presiona ENTER. Luego, el sistema expone el menú principal en donde puede comenzar a operar con el sistema. Dependiendo del tipo de usuario (normal o administrador) distintas opciones aparecerán en pantalla.
Por ejemplo se ingresa “Usuario01” el cuál es un tipo de usuario administrador. Este puede agregar o modificar un chatbot.

### Sistema de Chatbots - Usuario Administrador ###

Bienvenido Usuario01 usted es administrador.

1. Crear un Chatbot

2. Modificar un Chatbot

3. Ejecutar un Chatbot

4. Visualizar todos los chatbots existentes en el sistema

5. Visualizar todos los chatbots con sus flujos y opciones creadas

6. Ejecutar una simulación del sistema de chatbot

6. Salir

INTRODUZCA SU OPCIÓN: _

> Luego el usuario ingresa su selección y presiona ENTER, el programa continúa con el siguiente menú/vista/option/flow/chatbot

| | |
|---|---|
| **Prerrequisitos para evaluación** | - Clases y estructuras que forman el programa |
| **Requisitos de implementación** | - Implementar sistema de menú basado en terminal/consola<br><br>- **El menú puede variar según la cantidad de requisitos funcionales implementados durante la realización del laboratorio**.<br><br>**- Dado que no hay un script de prueba, todos los requerimientos deben ser expuestos a través de alguna interacción con el menú.**<br><br>- La aplicación debe permitir crear un system, chatbots, flows, junto con operar sobre ellas y visualizar su estado. |

**No obstante, puede implementar una vista gráfica (GUI) usando Swing (Java). Este es un requerimiento opcional. La interacción es mediante el menú o GUI, o es uno o es el otro, no ambos. Si realiza una interfaz gráfica (GUI) no debe realizar menú. Caso contrario, si realiza menú no debe realizar GUI.**
**No hay bonificación extra por realizar GUI o menú vía terminal, ambos reciben el mismo puntaje.**

**Antes de realizar alguna pregunta referente al menú y algún requisito, lea el enunciado completo.**

**Las siguientes funcionalidades mínimas deben estar implementadas en su solución. Estas funcionalidades pueden ser utilizadas mediante uno o más métodos dentro de las clases correspondientes.**

**EN ESTE LABORATORIO NO HAY UN SCRIPT DE PRUEBA, TODO SE EJECUTARÁ A TRAVÉS DEL MENÚ QUE USTED IMPLEMENTE.**

**Independiente de que los requerimientos y su evaluación se plantean de forma lineal, es decir, primero TDA Option Constructor, luego TDA Flow Constructor y así consecutivamente. El orden de las interacciones que ustedes implementan en el menú es decisión de ustedes.**

**IMPORTANTE: Al momento de la ejecución de su entrega, el sistema ya debe tener un sistema guardado con chatbots, flujos, options y usuarios.**

1. **(0.1 pts) TDA Option - constructor**. Método constructor de una opción para flujo de un chatbot. Cada opción se enlaza a un chatbot y flujo especificados por sus respectivos códigos.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 1 |
| **Requisitos de implementación** | El código de un *option* es único.<br><br>Su unicidad se verifica al momento de agregarlo a un flow a fin de evitar duplicidad. |
| **Parámetros de entrada** | code (Int) X message (String) X ChatbotCodeLink (Int) X InitialFlowCodeLink (Int) X Keyword (lista de 0 o más palabras claves) |

2. **(0.1 pts) TDA Flow - constructor**. Método constructor de un flujo de un chatbot.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 2 |
| **Requisitos de implementación** | Los flujos quedan identificados por un ID único. Su unicidad se verifica al momento de agregarlo a un chatbot a fin de evitar duplicidad.<br><br>Se debe verificar que las opciones añadidas no se repitan en base al id de éstos. |
| **Parámetros de entrada** | id (int) X name-msg (String) X Option (Lista de 0 o más opciones) |

3. **(0.1 pts) TDA Flow - modificador - flowAddOption**. Método modificador para añadir opciones a un flujo.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 3 |
| **Requisitos de implementación** | El método también verifica que las opciones añadidas no se repitan en base al id de éstos.<br><br>Es irrelevante cómo lo agrega, puede ser al principio, final, o en cierta posición, no es relevante para este laboratorio. |
| **Parámetro de entrada** | option |

4. **(0.1 pts) TDA chatbot - constructor**. Método constructor de un chatbot.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 4 |

| | |
|---|---|
| **Requisitos de implementación** | Los *chatbots* quedan identificados por un ID único. Su unicidad se verifica al momento de agregarlo a un sistema a fin de evitar duplicidad.<br><br>El método también verifica que los flujos añadidos no se repitan en base al id de éstos. |
| **Parámetro de entrada** | chatbotID (int) X name (String) X welcomeMessage (String) X startFlowId(int) X flows (Lista de 0 o más flujos) |

5. **(0.1 pts) TDA chatbot - modificador - chatbotAddFlow**. Método modificador para añadir flujos a un chatbot.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 5 |
| **Requisitos de implementación** | El método también verifica que los flujos añadidos no se repitan en base al id de éstos.<br><br>Es irrelevante cómo lo agrega, puede ser al principio, final, o en cierta posición, no es relevante para este laboratorio. |
| **Parámetro de entrada** | flow |

6. **(0.1 pts) TDA Usuario - constructor.** Método constructor de usuarios.

A diferencia de entregas pasadas, ahora el sistema tiene dos tipos de usuario, administrador y usuario común.

El usuario común puede:

- **Login/Logout**
- Interactuar con el chatbot (**system-talk**)
- Consultar por la síntesis de un chatbot que le pertenece (**system-synthesis**)
- Simular el diálogo entre dos chatbots del sistema (**system-simulate**)

El usuario administrador tiene las mismas capacidades que el usuario común pero además puede ejecutar:

- Crear/Agregar Chatbots al sistema (**chatbot constructor, addChatbotToSystem**)
- Crear/Agregar flujos (Flows) a un Chatbot (**flow constructor, addFlowToChatbot**)

- Crear/Agregar opciones (Options) a un flujo (Flow) (**option constructor, addOptionToFlow**)
- Todas las operaciones para agregar o modificar algún TDA, es parte del administrador. Mientras que el usuario común sólo puede realizar operaciones de lectura, ejecutar un chatbot, visualizar chatbots, pero no alterar el chatbot, flujo u opciones.

Para diferenciar las interacciones del usuario común con el usuario administrador, puede hacerlo mediante el menú. Esto es, luego del login, dependiendo del usuario, se abre el menú administrador o el menú de usuario común.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 5 |
| **Requisitos de implementación** | El método también verifica que los usuarios añadidos no se repitan en base al nombre (username) de estos, por ejemplo, no pueden existir dos Usuario1.<br><br>Existen dos tipos de usuario en el sistema. Usuario administrador y usuario normal. La diferencia entre ambos es que el usuario administrador puede acceder a funcionalidades de crear/modificar los TDAs.<br><br>Para la implementación de los tipos de usuario, puede realizarlo mediante interface o alguna forma de herencia. |
| **Parámetro de entrada** | username (String) |

7. **(0.1 pts) TDA system - constructor**. Método constructor de un sistema de chatbots. Deja registro de la fecha de creación.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 6 |
| **Requisitos de implementación** | El sistema además de contener los distintos chatbots, también contiene el chatHistory de cada usuario que interactúa con el sistema. Sobre el chatHistory, éste |

| | |
|---|---|
| | corresponde al registro completo del diálogo entre usuario y cada uno de los chatbots con los que interactúa.<br><br>El historial se mantiene para cada usuario y debe tener el String formateado de cada mensaje del usuario y chatbot (para luego ser visualizado en la consola/terminal a través del uso del menú), fecha, hora y emisor (usuario o sistema). |
| **Parámetros de entrada** | name (string) X InitialChatbotCodeLink (Int) X chatbots (Lista de 0 o más chatbots) |
| | |

8. **(0.1 pts) TDA system - modificador - systemAddChatbot**. Método modificador para añadir chatbots a un sistema.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 7 |
| **Requisitos de implementación** | Debe verificar que el chatbot no exista en el sistema a partir del id de éste.<br><br>Es irrelevante cómo lo agrega, puede ser al principio, final, o en cierta posición, no es relevante para este laboratorio. |
| **Parámetro de entrada** | chatbot |

9. **(0.1 pts) TDA system - modificador - systemAddUser**. Método modificador para añadir usuarios a un sistema. Piénselo como un registro (Register)

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 8 |
| **Requisitos de implementación** | Debe verificar que el usuario no exista en el sistema a partir del id de éste, que está dado por su nombre de usuario (String).<br><br>Es irrelevante cómo lo agrega, puede ser al principio, final, o en cierta posición, no es relevante para este laboratorio. |
| **Dominio** | user |

10. **(0.1 pts) TDA system - login - systemLogin**. Método que permite iniciar una sesión en el sistema.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 9 |
| **Requisitos de implementación** | Solo pueden iniciar sesión usuarios registrados<br><br>No se puede iniciar sesión si ya existe una sesión iniciada por otro usuario. |
| **Dominio** | user (string) |

11. **(0.1 pts) TDA system - logout - systemLogout**. Método que permite cerrar una sesión abierta.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 10 |
| **Requisitos de implementación** | Sin requisitos especiales |

| | |
|---|---|
| **Parámetros de entrada** | no hay |

12. **(1.5 pto) system-talk.** Método que permite interactuar con un chatbot.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 11 |
| **Requisitos de implementación** | Solo se puede conversar si se ha iniciado una sesión con un usuario previamente registrado. |
| **Parámetros de entrada** | message (string)<br>;message puede ser la opción o palabra clave |

13. **(1.5 pto) system-synthesis**. Método que ofrece una síntesis del chatbot para un usuario particular a partir de *chatHistory* contenido dentro del sistema

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 12 |
| **Dominio** | usuario (String) |

14. **(0,5 pto) system-simulate**. Permite simular un diálogo entre dos chatbots del sistema.

| | |
|---|---|
| **Prerrequisitos para evaluación (req. funcionales)** | 13 |
| **Requisitos de implementación** | El registro de las interacciones queda en chatHistory.<br>La semilla (seed) se establece para generar números aleatorios. Debe ocupar clases propias de Java para este aspecto, más abajo se muestra un ejemplo de Random.<br>El número máximo de interacciones es una cota superior. Si no se pueden generar más interacciones, la simulación termina. |

| | |
|---|---|
| **Dominio** | maxInteractions (int) X seed (int) X system |

Para la generación de números aleatorios puede utilizar la clase Random de java.util. Esto es sólo una sugerencia, si encuentra una forma de generar números aleatorios usando sólo métodos internos de java, pueda usarla.

Un ejemplo del uso de Random con seeds:

```
import java.util.Random;

public class RandomExample {
    public static void main(String[] args) {
        // Crear una instancia de Random con un seed específico
        long seed = 1234;
        Random random = new Random(seed);

        // Generar algunos números aleatorios
        int randomNumber1 = random.nextInt();
        int randomNumber2 = random.nextInt();

        // Imprimir los números generados
        System.out.println("Primer número aleatorio: " + randomNumber1);
        System.out.println("Segundo número aleatorio: " + randomNumber2);
    }
}
```

**Ejemplo de Output para synthesis**

1695486245 - user312321: Hola

1695486245 - Inicial: Flujo Principal Chatbot 1

Bienvenido

¿Qué te gustaría hacer?

1) Viajar

2) Estudiar

1695486245 - user312321: 1

1695486245 - Agencia Viajes: Flujo 1 Chatbot1

¿Dónde te Gustaría ir?

1) New York, USA

2) París, Francia

3) Torres del Paine, Chile

4) Volver

1695486245 - user312321: 4

1695486245 - Inicial: Flujo Principal Chatbot 1

Bienvenido

¿Qué te gustaría hacer?

1) Viajar

2) Estudiar

1695486245 - user312321: 1

1695486245 - Agencia Viajes: Flujo 1 Chatbot1

¿Dónde te Gustaría ir?

1) New York, USA

2) París, Francia

3) Torres del Paine, Chile

4) Volver

1695486245 - user312321: 1

1695486245 - Agencia Viajes: Flujo 2 Chatbot1

¿Qué atractivos te gustaría visitar?

1) Central Park

2) Museos

3) Ningun otro atractivo

4) Cambiar destino

1695486245 - user312321: 3

1695486245 - Agencia Viajes: Flujo 3 Chatbot1

¿Vas solo o acompañado?

1) Solo

2) En pareja

3) En familia

4) Agregar más atractivos

5) En realidad quiero otro destino

# Leer esto 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 jerarquia, 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 (#)