

Guía de aprendizaje

**Diseño y modelado de Algoritmos**

Exp 1 – Semana 2

Fundamentos de Programación (PRY2201)

**Facilitador disciplinar:** Josué Oteiza

**Asesor par:** Luis Videla

# Índice

[Introducción a la semana 3](#_Toc151637167)

[Resultado de aprendizaje 4](#_Toc151637168)

[Conceptos relevantes 4](#_Toc151637169)

[Preguntas activadoras 5](#_Toc151637170)

[Actividad 5](#_Toc151637171)

[Diseño y modelado de Algoritmos 6](#_Toc151637172)

[¿Cómo diseñar mi algoritmo? 6](#_Toc151637173)

[Diseño y modelado de algoritmos estructurados en Java 10](#_Toc151637175)

[Ejemplo de programación Estructurada en Java 15](#_Toc151637177)

[Video 21](#_Toc151637178)

[Lectura complementaria 21](#_Toc151637179)

[Cierre de la semana 22](#_Toc151637180)

[Referencias 23](#_Toc151637181)

[Lecturas de la semana 23](#_Toc151637182)

[Apuntes 24](#_Toc151637183)

# Introducción a la semana

Te damos la bienvenida a la semana 2 de la asignatura Fundamentos de Programación. En esta semana, titulada "Diseño y modelado de algoritmos", profundizaremos en el diseño de algoritmos utilizando el lenguaje de programación Java.

En otras palabras, continuaremos construyendo sobre los conocimientos adquiridos en la semana 1, donde aprendiste los conceptos básicos de Java y cómo escribir programas simples. En esta ocasión, nos centraremos en la importancia de diseñar algoritmos eficientes y estructurados para resolver problemas más complejos.

A lo largo de esta guía de aprendizaje, exploraremos los siguientes temas:

**Conceptos de diseño de algoritmos:** aprenderás a abordar problemas de programación desde un enfoque estructurado.

**Estructuras de control:** conocerás las estructuras de control condicionales (if, else) y los bucles (for, while) en Java. Estas estructuras son fundamentales para controlar el flujo de ejecución de un programa.

**Modelado de algoritmos:** practicarás cómo modelar algoritmos a través de pseudocódigo. Esto te ayudará a visualizar y planificar tus programas antes de escribir código.

# Resultado de aprendizaje

### El estudiante será capaz de:

**RA1.** Utiliza estrategias de abstracción para la construcción de algoritmo, aplicando pseudocódigo con el objetivo de dar solución a problemáticas planteadas.

### Indicador de logro:

**IL1.** Identifica estrategias de abstracción y tipos de algoritmos aplicables en la construcción de algoritmos.

**IL2.** Aplica pseudocódigo en la implementación de algoritmos, brindando una solución a las problemáticas planteadas.

# Conceptos relevantes

| Algoritmos estructurados | Estructuras de control | Pseudocódigo |
| --- | --- | --- |
| Resolución de problemas | Sintaxis de Java | Eficiencia de algoritmos |

# Preguntas activadoras

* ¿Por qué es importante el diseño de algoritmos estructurados en programación y cómo puede mejorar la eficiencia de un programa?
* ¿Cuál es la diferencia entre una estructura condicional (if-else) y un bucle (for o while) en Java?
* ¿Por qué es importante considerar la eficiencia de un algoritmo? ¿Qué factores influyen en la eficiencia de un programa en Java?

# Actividad

**Descripción de la actividad**

En esta segunda semana realizarás una actividad formativa individual con encargo, llamada " Diseñando y modelando Algoritmos ". Aquí, deberás seleccionar un tema y a través de un análisis, determinar la mejor técnica de abstracción y tipo de algoritmo para proponer una solución en pseudocódigo y crear un algoritmo en Java.

# Diseño y modelado de Algoritmos

## ¿Cómo diseñar mi algoritmo?

Si recordamos lo que es un algoritmo, este es un conjunto prescrito de instrucciones o reglas que permite llevar a cabo una actividad mediante pasos sucesivos. Este conjunto ordenado de operaciones sistemáticas permite hacer un cálculo y hallar la solución de un tipo de problemas. Es decir que, cuando diseñamos un algoritmo y escribimos un código, no solo buscamos que funcione, sino que también sea claro y eficiente. Un algoritmo bien diseñado es como un mapa que guía a una aplicación a través de una serie de instrucciones lógicas, permitiendo que realice tareas específicas de manera efectiva. Para lograr esto, es esencial comprender y aplicar los conceptos de diseño de algoritmos que vimos anteriormente.

La claridad en el diseño se refiere a la capacidad de un algoritmo para ser comprendido fácilmente por otros programadores o programadoras. Al seguir una estructura lógica y nombrar las variables y funciones de manera descriptiva, facilitamos la lectura y el mantenimiento del código.

Su modularidad implica dividir el código en partes más pequeñas y manejables, lo que facilita la depuración y la colaboración en proyectos de programación. La eficiencia, por su parte, se relaciona con la optimización del algoritmo para que realice su tarea en el menor tiempo posible, utilizando la menor cantidad de recursos. Diseñar algoritmos eficientes es crucial en aplicaciones que requieren un alto rendimiento.

Para comenzar a diseñar un algoritmo, necesitas saber qué tipos de algoritmos podemos utilizar, dependiendo de los problemas o tareas específicas a desarrollar. Estos son:

**1. Algoritmos de búsqueda**: se utilizan para encontrar un elemento o un valor específico en una colección de datos, como una lista, un arreglo o una estructura de datos.

Ejemplos: una búsqueda lineal, búsqueda binaria, búsqueda en árboles binarios, búsqueda en grafos, etc.

# Video

Profundicemos a través del siguiente ejemplo sobre los algoritmos de búsqueda:

<https://videosduoc.duoc.cl/media/t/1_1it0z3cg>

**2. Algoritmos de ordenamiento:** se utilizan para organizar elementos en una colección de datos en un orden específico, como orden ascendente o descendente.

Ejemplos: Algoritmo de burbuja, algoritmo de selección, algoritmo de inserción, algoritmo de merge sort, algoritmo de quicksort.

**3. Algoritmos de grafos:** se utilizan para resolver problemas relacionados con estructuras de datos de grafos, como encontrar caminos más cortos, determinar la conectividad o encontrar ciclos.

Ejemplos: Búsqueda en anchura (BFS), búsqueda en profundidad (DFS), algoritmo de Dijkstra, algoritmo de Kruskal, algoritmo de Bellman-Ford.

**4. Algoritmos de optimización:** se utilizan para encontrar la mejor solución entre un conjunto de posibles soluciones, generalmente maximizando o minimizando una función objetivo.

Ejemplos: Algoritmo de búsqueda local, algoritmo de programación lineal, algoritmo genético, algoritmo de colonia de hormigas.

**5. Algoritmos de recursión:** se basan en la idea de que una función se llama a sí misma para resolver un problema en pasos más pequeños, generalmente dividiendo el problema en subproblemas.

Ejemplos: cálculo del factorial, búsqueda en árboles binarios de búsqueda, algoritmo de Torres de Hanoi.

**6. Algoritmos de aprendizaje automático:** son un subconjunto de algoritmos de optimización que se utilizan para entrenar modelos matemáticos a partir de datos, con el objetivo de realizar tareas como clasificación, regresión, clustering o detección de patrones.

Ejemplos: regresión lineal, máquinas de soporte vectorial (SVM), redes neuronales, algoritmos de agrupamiento k-means, algoritmos de bosques aleatorios, algoritmo de gradiente descendente.

Ahora que conoces los tipos de algoritmos, ¿recuerdas las estructuras de control que vimos la semana anterior?

**Figura 1**

*Estructuras de control*

Entre las estructuras de control que revisamos, conociste las estructuras condicionales (if, else) y los bucles (for, while) en Java; fundamentales para controlar el flujo de ejecución de un programa. Si hablamos del cometido de las estructuras condicionales como "if" y "else”, estas permiten que el programa tome decisiones en función de ciertas condiciones.

*Nota*. Ejemplos de estructuras de control. Cidecame UAEH (s.f.). *Estructuras de control.* <http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro4/2_estructuras_de_control.html>

**Figura 2**

*Estructura de control condicional compuesta*

Por ejemplo, un programa puede decidir si ejecutar una sección de código o no, según si se cumple una condición específica. Esto es vital para crear programas flexibles y adaptables.

*Nota*. Ejemplo de estructura de control compuesta if-else. Portal Académico (s.f.). *Estructura de control condicional compuesta (if-else).* Portal académico. <https://portalacademico.cch.unam.mx/cibernetica2/estructuras-condicionales/condicional-compuesta>

Mientras tanto, los bucles, como "for" y "while," son útiles para repetir una secuencia de acciones múltiples veces, lo que ahorra tiempo y esfuerzo al evitar la repetición manual del código.

Básicamente, las estructuras de control son el conjunto de reglas que permiten controlar el flujo de ejecución de las instrucciones de un algoritmo o de un programa. Estas y otros elementos son lo que nos permitirán ir diseñando nuestros algoritmos en Java de forma estructurada. Comencemos entonces, a diseñar un algoritmo estructurado.

## Diseño y modelado de algoritmos estructurados en Java

Diseñar y modelar algoritmos en Java, implica seguir un proceso estructurado para resolver problemas de programación de manera eficiente y lógica. Es precisamente en este proceso, que aparece un elemento cuyo desempeño es crucial:

El pseudocódigo es una forma de modelado de un algoritmo o programa que utiliza un lenguaje de descripción simple y cercano al lenguaje humano para expresar la lógica del algoritmo paso a paso. Es una herramienta útil en la fase de diseño de algoritmos, ya que permite planificar y estructurar la lógica antes de la implementación en un lenguaje de programación específico.

En esta instancia, se hace relevante mencionar las estrategias de abstracción (vistas en la semana anterior), puesto que se aplican durante el diseño de un algoritmo, lo cual puede ocurrir tanto antes como después de la etapa de pseudocódigo.

# Video

En el siguiente video, reforzaremos lo que aprendiste en la semana 1, y profundizaremos en la utilización de estas estrategias a través de un ejemplo:

<https://videosduoc.duoc.cl/media/t/1_v4909et3>

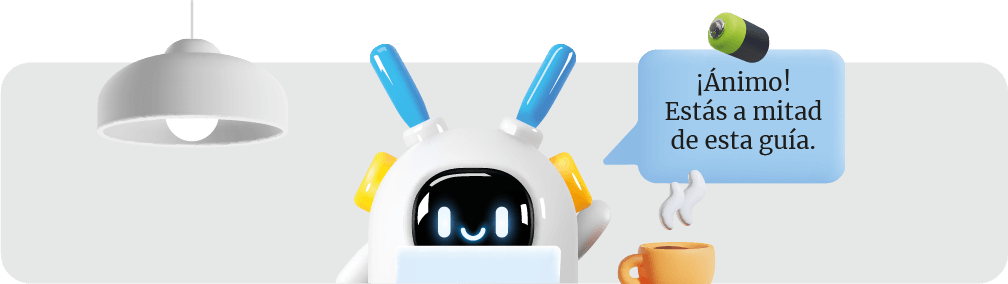
Ahora, como primera etapa, previo a comenzar a diseñar tu algoritmo, debes **entender la problemática planteada.** En otras palabras, tienes que asegurarte de comprender completamente lo que estás tratando de resolver, identificando los requisitos y restricciones del problema.

La siguiente etapa implica **definir el objetivo o la tarea que debe realizar el algoritmo.** Por ejemplo, definir las entradas (datos de entrada) y salidas (resultados esperados). Ya teniendo esto especificado, deberás convertir pseudocódigo a Java, y para ello, debes seguir los siguientes pasos:

1. **Identificar estructuras de control:** analiza las estructuras de control en tu pseudocódigo, como bucles (for, while) y condicionales (if, else), y traduce estas estructuras a las correspondientes en Java.
2. **Declarar variables:** convierte las declaraciones de variables en pseudocódigo a declaraciones de variables en Java.
3. **Sintaxis Java:** ajusta la sintaxis según las reglas de Java. Esto incluye el uso correcto de las llaves ‘{}’, el punto y coma ‘;’, y la correcta indentación del código.

Importante

Se llama **indentación** de código al hecho de utilizar sangrado (mover ligeramente hacia la derecha las líneas de código), facilitando así la lectura e indicando visualmente si nos encontramos en el interior de una estructura concreta, en el programa principal, etc.

Observa a través del siguiente ejemplo, cómo se elabora pseudocódigo en Java, utilizando los pasos recién mencionados:

**Figura3**

*Pseudocódigo*

Nota. Ejemplo de pseudocódigo. PSeInt. (s.f.). *Suma de dos números enteros.* PSeInt.[PSeInt (sourceforge.net)](https://pseint.sourceforge.net/index.php?page=ejemplos.php&cual=Suma&mode=flexible)

Este pseudocódigo describe un algoritmo paso a paso:

1. Lee dos números desde el usuario
2. Calcula la suma de los dos números
3. Muestra el resultado

Aquí aplicamos el **Paso 1: identificar estructuras de control**

**Figura 4**

*Implementación de pseudocódigo en Java*

*Nota*. Ejemplo de suma de números enteros en Java. CódigoRoot. (2023). *Suma de dos números introducidos por teclado en Java.* <https://codigoroot.net/blog/suma-de-dos-numeros-introducidos-por-el-usuario-en-java/>

En este ejemplo de Java, la estructura de control de entrada (leer desde el usuario) se implementa utilizando la clase *‘Scanner’.* No hay bucles ni condicionales en este pseudocódigo, pero si los hubiera, se traducirían directamente a las estructuras de control correspondientes en Java (por ejemplo, ‘*if’* para condicionales y *‘for’* o *‘while’* para bucles).

**Paso 2: declarar variables en Java**

En el pseudocódigo, se menciona la variable *‘Suma’*. En Java, necesitamos declarar explícitamente las variables antes de usarlas:

**Figura 5**

*Variable ‘suma’*

*Nota.* Ejemplo de suma de números enteros en Java. CódigoRoot. (2023). *Suma de dos números introducidos por teclado en Java.* <https://codigoroot.net/blog/suma-de-dos-numeros-introducidos-por-el-usuario-en-java/>

**Paso 3: ajustar la sintaxis en Java**

La sintaxis en Java debe seguir las reglas del lenguaje. En este caso, las llaves ‘{}’ delimitan el alcance de las declaraciones y se utilizan correctamente en el método ‘main’, mientras que los puntos y comas ‘;’ se utilizan al final de las declaraciones en Java.

Como ves, el pseudocódigo ayuda a garantizar que el algoritmo esté bien diseñado antes de pasar a la fase de implementación en un lenguaje de programación específico (en tu caso, Java).

Ahora, observemos un ejemplo de programación estructurada:

## Ejemplo de programación estructurada en Java

Una estudiante de programación desea crear un programa que le permita calcular sus notas ponderadas. El problema para ella es que algunos ramos tienen 3 notas, otros 4 y otros 5, y no está segura de cómo hacerlo.

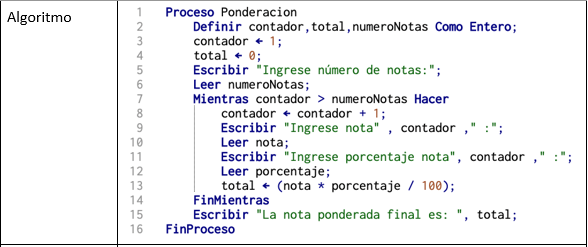
Si analizamos el caso, la estudiante ya realizó las etapas de:

1. Entender la problemática planteada: ramos con diferente cantidad de notas, y
2. Definir la tarea que debe realizar el algoritmo: calcular las notas ponderadas.

Sin embargo, a medida que ha realizado algunos intentos, obtiene errores. A continuación, se muestra lo que lleva:

**Figura 6**

*Ejemplo de algoritmo*



*Nota.* Ejemplo de algoritmo estructurado en pseudocódigo. *PSeInt. (s.f.).* PSeInt. <https://pseint.sourceforge.net/>

**¿Qué crees que está haciendo mal? ¿Cómo evitar los errores?**

En su caso, no cumplió con el paso correspondiente a **identificar las variables,** puesto que, de haberlo hecho, hubiese aplicado las siguientes:

* ‘notasPorRamo’: arreglo bidimensional para almacenar las notas de cada ramo.
* ‘pesosPorRamo’: arreglo para almacenar la cantidad de notas por ramo.
* ‘sumaNotasPonderadas’: variable para acumular la suma de notas ponderadas.
* ‘sumaPesos’: variable para acumular la suma de pesos.

Una vez aplicadas estas variables, llevamos a pseudocódigo, obteniendo la siguiente solución a la problemática planteada:

**Figura 7**

*Ejemplo de algoritmo en Java*

*Nota.* Ejercicio de programación estructurada en Java. The Apache Software Foundation. (2023). *Apache NetBeans (19).* [Software]. Apache NetBeans. <https://netbeans.apache.org/front/main/>

El algoritmo resuelto anteriormente, corresponde a una calculadora de promedio ponderado para estudiantes que desean realizar este cálculo en función de las notas y los créditos de varios cursos. El código parece estar bien estructurado y es fácil de entender, lo que a primera vista nos funciona para el objetivo de la estudiante, pero si queremos hacer que el código sea más robusto, podemos aplicar las siguientes mejoras:

1. **Comentarios descriptivos:** el código tiene comentarios que explican lo que hace, lo cual es bueno para facilitar la comprensión. Sin embargo, podrías agregar algunos comentarios más descriptivos para explicar la lógica detrás de algunas secciones del código, lo que ayudaría a otras personas (y a ti mismo/a) a entenderlo mejor en el futuro.

**Figura 8**

*Comentarios descriptivos*

*Nota.*Ejemplo de comentarios descriptivos en secciones de código. The Apache Software Foundation. (2023). *Apache NetBeans (19).* [Software]. Apache NetBeans. <https://netbeans.apache.org/front/main/>

1. **Manejo de errores:** actualmente, el código no maneja posibles errores. Sería útil agregar manejo de errores para garantizar que los usuarios ingresen datos válidos (por ejemplo, números positivos para las notas y los créditos). Esto evitaría que el programa falle si el usuario ingresa datos incorrectos. En el siguiente ejemplo, se implementó manejo de errores utilizando bloques ‘try-catch’ para capturar posibles excepciones, como ‘InputMismatchException’:

**Figura 9**

*Manejo de errores con bloques ‘try-catch’*

*Nota.*Ejemplo de manejo de errores utilizando bloques ‘try-catch’. The Apache Software Foundation. (2023). *Apache NetBeans (19).* [Software]. Apache NetBeans. <https://netbeans.apache.org/front/main/>

1. **Validación de entrada:** asegúrate de agregar validaciones para asegurar que las entradas del usuario, como el número de notas y el número de créditos, sean válidos y no negativos, como en la imagen:

**Figura 10**

*Validación de entrada*

*Nota.*Ejemplo que muestra validación de entrada en el número de notas. The Apache Software Foundation. (2023). *Apache NetBeans (19).* [Software]. Apache NetBeans. <https://netbeans.apache.org/front/main/>

1. **División por cero:** puede ocurrir una división por cero si el usuario ingresa un número de créditos de 0 para alguna nota. Puedes añadir una validación para evitar esto:

**Figura 11**

*Validación de entrada*

*Nota.*Ejemplo que muestra validación para evitar la división por cero. The Apache Software Foundation. (2023). *Apache NetBeans (19).* [Software]. Apache NetBeans. <https://netbeans.apache.org/front/main/>

1. **Cierre del Scanner:** el código cierra el ‘Scanner’ al final, lo cual es una buena práctica, pero si se lanzara una excepción antes de cerrarlo, el programa se bloquearía sin cerrar correctamente el ‘Scanner’. Sería más seguro usar un bloque `try-with-resources` para asegurarse de que el ‘Scanner’ se cierre apropiadamente, incluso en caso de excepciones:

**Figura 12**

*Cierre de ‘Scanner’*

*Nota*. Utilización de un bloque ‘try-with-resources’ para garantizar que el ‘Scanner’ se cierre de manera segura. The Apache Software Foundation. (2023). *Apache NetBeans (19).* [Software]. Apache NetBeans. <https://netbeans.apache.org/front/main/>

En general, el código hace un buen trabajo al calcular el promedio ponderado de los cursos, pero incorporar estas sugerencias puede mejorar su robustez y usabilidad.

# Video

En el siguiente video podrás ver más en detalle cómo realizar una programación estructurada en Java utilizando estructuras de control:

<https://www.youtube.com/watch?v=Fw2MNOZ9-rs&list=PLJqxoTMFSPhNuWwic72Y-KGG8zoegvhyL>

# Lectura complementaria

Sierra K., Bates B., Gee B. (2022). Java: A Learner’s Guide ava to Real-World Programming. O´Reilly. <https://www.pdfiles.net/storage/Books/headfirst/Head_First_Java_A_Brain-Friendly_Guide.pdf>

# Cierre de la semana

El diseño y modelado de algoritmos es una parte fundamental de la informática y la ciencia de la computación. Para abordar esta tarea de manera efectiva, es necesario utilizar estrategias de abstracción para simplificar los problemas, explorar diferentes tipos de algoritmos para encontrar la mejor solución, expresar la lógica en pseudocódigo para una comprensión clara y, finalmente, implementar los algoritmos en un lenguaje de programación como Java.

La capacidad de diseñar y modelar algoritmos eficientes es esencial para resolver problemas en una amplia variedad de campos, desde la optimización de rutas de entrega hasta la búsqueda de información en grandes bases de datos. A medida que avanzamos en la era de la información, la importancia de estas habilidades solo seguirá creciendo, y el dominio de estas estrategias y herramientas es esencial para cualquier profesional en el campo de la informática.

# Referencias

Robledano, A. (2019). *Qué es pseudocódigo.* <https://openwebinars.net/blog/que-es-pseudocodigo/>

Samm Systems. (2020). *Programación Estructurada P1 - Java (Estructuras de Control).* [Video]. <https://youtu.be/Fw2MNOZ9-rs?si=x2V7plPNhGwYRkDO>

# Lecturas de la semana

* Capítulo 4: Making the most variables and their values

Burd, B. (2022). Java for Dummies. New Jersey: John Wiley & Sons.<https://webezproxy.duoc.cl/login?url=http://biblioteca.duoc.cl/bdigital/elibros/a50163-Java_fordummies/68/> Páginas 57 a 95

* Capítulo 1: Introducción a Java

Vegas Gertrudix, J. M. (2022). Java 17: Fundamentos prácticos de programación. Bogotá: Ediciones de la U.<https://webezproxy.duoc.cl/login?url=http://biblioteca.duoc.cl/bdigital/elibros/a50229-Java_17/36/> Páginas 40 a 57

# Apuntes

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Reservados todos los derechos Fundación Instituto Profesional Duoc UC. No se permite copiar, reproducir, reeditar, descargar, publicar, emitir, difundir, de forma total o parcial la presente obra, ni su incorporación a un sistema informático, ni su transmisión en cualquier forma o por cualquier medio (electrónico, mecánico, fotocopia, grabación u otros) sin autorización previa y por escrito de Fundación Instituto Profesional Duoc UC La infracción de dichos derechos puede constituir un delito contra la propiedad intelectual.