



Recibe una cálida:

# ¡Bienvenida!

---

Te estábamos esperando 😊 

# ➤ Polimorfismo y principios básicos de diseño

---

**Plan formativo:** Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

# HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



# REPASO CLASE ANTERIOR

En la clase anterior trabajamos :

✓ *Polimorfismo mediante interfaces*



# LEARNING PATHWAY

4.6

Start! 🏁

**Polimorfismo y  
principios básicos de  
diseño**

Principios de diseño orientado a  
objetos. y Mantenibilidad y  
Reutilizabilidad

Mantenibilidad y  
Reutilizabilidad

Manteniendo

# OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



***Reconocer la importancia de los principios de diseño orientado a objetos***



***Comprender los conceptos de mantenibilidad y reutilizabilidad***





# Rompehielo 🧊

**Respondan en el chat o levantando la mano: 🙋**

Imaginen que deben construir una casa.

1. ¿Usarían los mismos ladrillos, puertas y ventanas en todas las habitaciones o preferirían componentes modulares que permitan cambiar un dormitorio sin afectar la cocina?
2. ¿Por qué?
3. ¿Cómo aplicarían esta lógica en programación?



# ➤ Principios de Diseño Orientado a Objetos






# Principios de Diseño Orientado a Objetos



## ¿Qué son y para qué se utilizan?:

Un principio de diseño en Programación Orientada a Objetos (POO) es una regla o **guía general** que se utiliza para construir software de calidad y bien estructurado. Estos principios nos brindan directrices **para organizar y diseñar nuestras clases y objetos de manera efectiva**, promoviendo un **código limpio, mantenible y fácil de entender**.



Al seguir estos principios, podemos obtener múltiples **beneficios** en términos de **organización** del código, **mantenibilidad**, **reutilización** de código, **flexibilidad** y **calidad** del software en general.



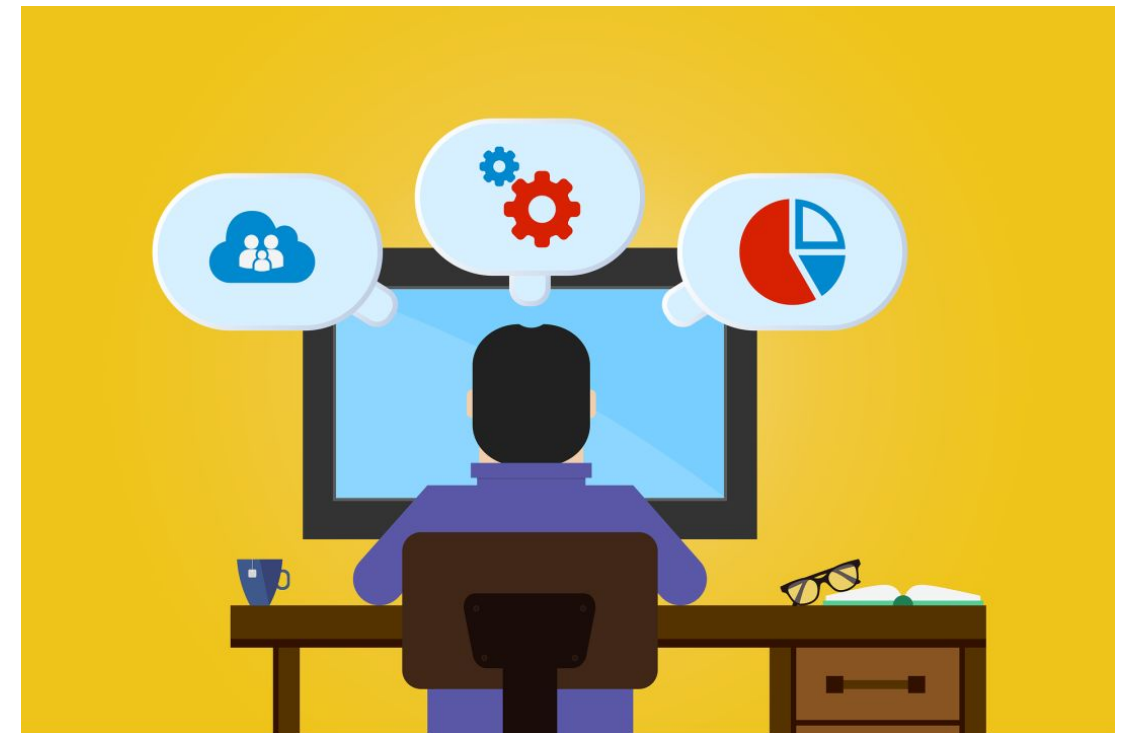
# ➤ Mantenibilidad



# Mantenibilidad

Se refiere a la **capacidad** de un software **para ser modificado, corregido, mejorado o adaptado de manera eficiente y efectiva** a lo largo del tiempo.

- Código bien organizado, poco acoplado, con responsabilidades claras.
- Agregar features o arreglar bugs de forma rápida y sencilla.
- Depende de tener un diseño limpio, módulos independientes y una arquitectura escalable.
- Equipos pequeños pueden mantener grandes sistemas si están bien diseñados.
- Reducir duplicación y extraer abstracciones aumenta la mantenibilidad.





# Mantenibilidad

La mantenibilidad es importante porque los sistemas de software evolucionan constantemente. Se requieren actualizaciones, correcciones de errores y mejoras en función de los cambios en los requisitos y las necesidades de los usuarios.

Esto facilita la comprensión del sistema, la identificación rápida de problemas y la implementación de cambios sin introducir nuevos errores.



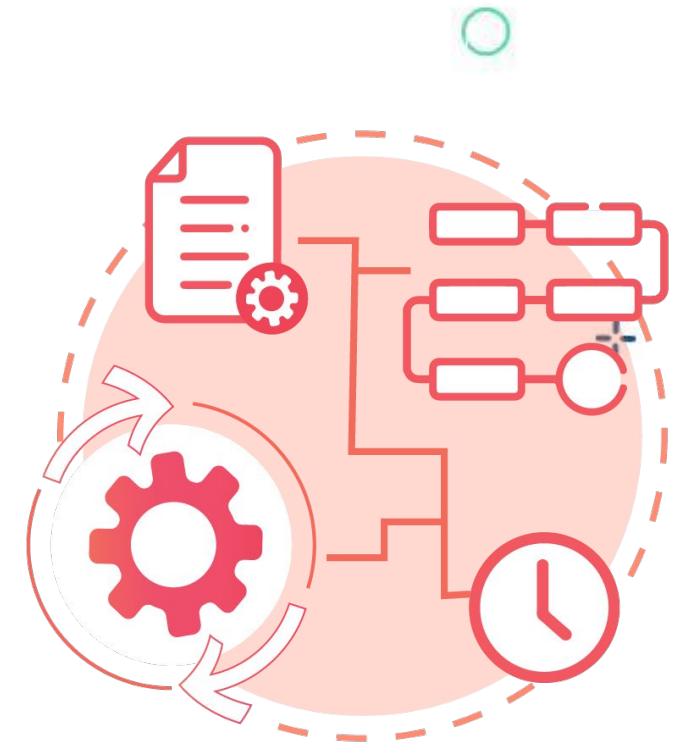
# › Reutilizabilidad



# Reutilizabilidad

Se refiere a la capacidad de **utilizar componentes, módulos o partes de software existentes** en diferentes contextos o aplicaciones **sin tener que reescribirlos** desde cero.

- En lugar de reinventar la rueda, se crean piezas reutilizables.
- La POO permite crear clases y librerías paramétricas y genéricas.
- Las interfaces y la herencia facilitan reutilizar comportamientos.
- Se reduce redundancia y duplicación de esfuerzos.
- Menor cantidad de código implica menor costo de mantenimiento.
- Aumenta la productividad y simplicidad del software.





# Reutilizabilidad



La reutilizabilidad se logra mediante el diseño e implementación de componentes y bibliotecas que son independientes, modulares y fácilmente integrables en diferentes sistemas. Esto implica utilizar técnicas como la encapsulación, la abstracción y la creación de interfaces claras y flexibles.



# Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro...

## Trabajo Integrador del Módulo 💪

Iremos completándolo progresivamente clase a clase.





# LIVE CODING

Ejemplo en vivo

## Reutilizando:

*Vamos a tomar una clase específica y generalizarla para hacerla reusable y fácil de mantener. En este caso, cambiaremos de Cuenta a CuentaBancaria. Esto nos permitirá generalizar aún más el alcance de la Clase. También vamos a refactorizar código repetido de ser necesario.*

 **Tiempo: 30 minutos**





# **Ejercicio N° 1**

# Manteniendo



# Manteniendo

## Contexto: 🙌

Vamos a analizar el siguiente código (pueden copiarlo y pegarlo en el IDE) dónde debemos mejorarlo para que sea entendible, mantenible y reutilizable.

## Consigna: 📝

1. Implementar una mejor encapsulación de los atributos.
2. Agregar comentarios y documentación para explicar cómo funciona el código.
3. Hacer que el código sea más flexible y modular, permitiendo la gestión de más detalles de los empleados.

**Tiempo** 🕒: 40 minutos



# Manteniendo

## Clase de administración de empleados

```
public class EmployeeManagement {  
    private ArrayList<Employee> employees;  
    public EmployeeManagement() {  
        employees = new ArrayList<>();  
    }  
    public void addEmployee(String name, int age) {  
        Employee employee = new Employee(name,  
age);  
        employees.add(employee);  
    }  
}
```

## Clase main

```
public static void main(String[] args) {  
    EmployeeManagement employeeManagement  
= new EmployeeManagement();  
    employeeManagement.addEmployee("John",  
30);  
    employeeManagement.addEmployee("Alice",  
25);  
    employeeManagement.removeEmployee(1);  
    employeeManagement.printEmployeeList();  
}  
}
```

○

# ¿Alguna consulta?

+



# RESUMEN

¿Qué logramos en esta clase?

- ✓ **Comprender la importancia de los conceptos de reutilizabilidad y mantenibilidad**



# #WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 🙌 🙌 🙌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
  - a. Material 1 (Foro)
  - b. *Lectura Módulo 4, Lección 6: páginas 7 - 8*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

# ¡Muchas Gracias!

Nos vemos en la próxima clase 🙌





Momento: ✚

# Time-out!

🕒 5 min.

