Clase grabada

Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

English Always | alkemy

Clase grabada

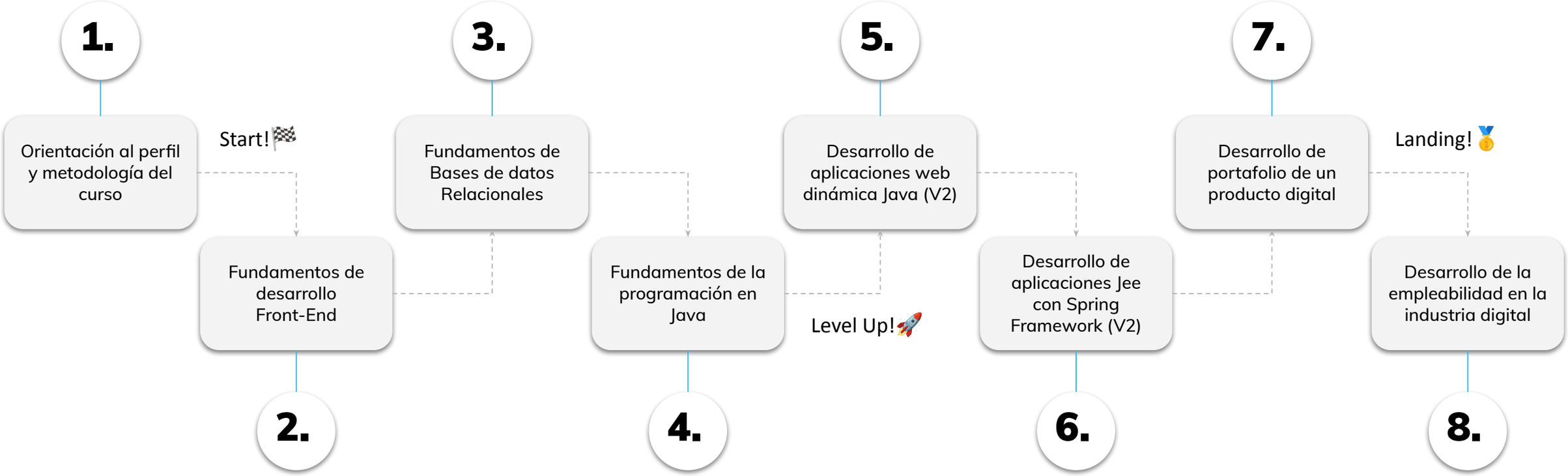
Polimorfismo y principios básicos de diseño

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

English Always | alkemy

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



English Always | alkemy

REPASO CLASE ANTERIOR

En la clase anterior trabajamos 📚:

✔ *Conceptos de Cohesión y Acoplamiento*

English Always | alkemy

LEARNING PATHWAY

4.6 Start!🏁

Polimorfismo y principios básicos de diseño

Principio de Responsabilidad Única

Responsabilidad Única

Sólo una responsabilidad

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- *Conocer los principios de diseño SOLID*
- *Comprender el principio de Responsabilidad Única*

Rompehielo🥶

Siendo responsables: 🙌

En la imagen ¿podemos ver varios objetos, o sólo un objeto con muchas posibilidades?

Respondan en el chat o levantando la mano: ✍️

1. ¿Qué objetos observan en la imagen?
2. ¿Cuál es la desventaja de tener tantas herramientas en un solo elemento?
3. ¿Sería mejor tener cada herramienta por separado en vez de combinadas?

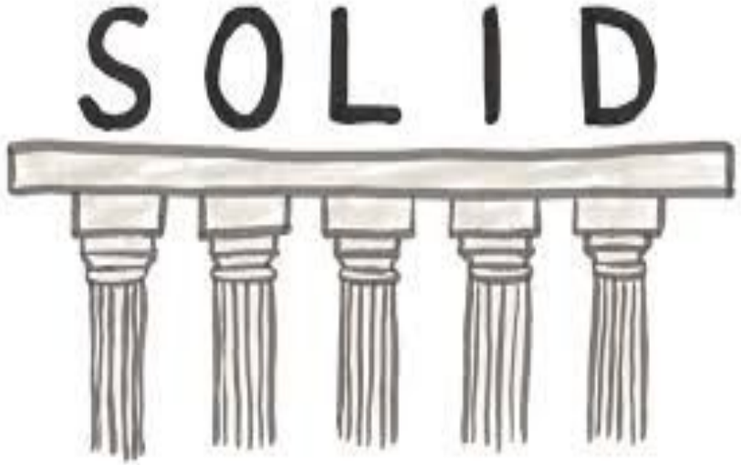
¿Cómo imaginan una clase de Java que cumpla tantas funciones? ¿Será práctica?

Principios SOLID

Principios SOLID

¿Qué son y para qué se utilizan?:

Los principios **SOLID** son 5 principios básicos de la programación orientada a objetos. Estos fueron desarrollados por Robert C. Martin (Uncle Bob) en el año 2000. Los principios SOLID **tienen como objetivo eliminar las malas prácticas en el diseño y desarrollo de software**. La aplicación de estos principios **ayudan al desarrollador a escribir un código mantenible, escalable y robusto**.



Principios SOLID

Los principios SOLID son:

- **Single Responsibility Principle (**Principio de responsabilidad única).
- **Open/Closed Principle (**Principio de abierto/cerrado).
- **Liskov Substitution Principle** (Principio de sustitución de Liskov).
- **Interface Segregation Principle** (Principio de segregación de interfaces).
- **Dependency Inversion Principle** (Principio de inversión de dependencia).

Principios SOLID

El término SOLID es un acrónimo de los 5 principios.

En esta clase veremos el primero de ellos.

Principio de Responsabilidad Única

Responsabilidad Única

Este principio establece que **cada clase debe tener una única responsabilidad** dentro de nuestro software, y esta responsabilidad debe estar **definida y ser concreta**.

Todos los métodos deben estar alineados con la finalidad de la clase.

“Una clase debe tener solo una razón para cambiar” – Uncle Bob

Si nos encontramos con una **clase que dispone de métodos que realiza tareas con distintas finalidades se debería refactorizar el código** y crear clases nuevas que se corresponda con los objetivos del método.

Responsabilidad Única

Gracias a este principio nuestras clases tendrán un **bajo acoplamiento** y a la hora de realizar cambios menos clases se verán afectadas.

Algunos síntomas que nos pueden hacer sospechar (bad smells) que **no se está cumpliendo este principio** serían:

- Una clase es demasiado larga, demasiadas líneas de código.
- Cada vez que hay que introducir una modificación o una nueva funcionalidad, es necesario tocar en muchos sitios.
- Hay demasiadas relaciones entre clases.
- Mezcla funcionalidades de distintas capas de la arquitectura.

Evaluación Integradora ✨

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

Trabajo Integrador del Módulo 💪🏽

Iremos completándolo progresivamente clase a clase.

English Always | alkemy

LIVE CODING

Ejemplo en vivo

¡Poniendo en práctica!

Evaluar si la clase CuentaBancaria sigue el principio de responsabilidad única, centrándose únicamente en la gestión del saldo y las transacciones

1. *Refactorizar según el principio de responsabilidad única*

Veamos un ejemplo parecido:👉

Tiempo: 30 minutos

LIVE CODING

Ejemplo en vivo

```
public class CuentaBancaria{

 private List<Coin> coins;

 public void addCoin(Coin coin) {
 }

 public void sendPayment(double amount) {
 }

 public double getBalance() {
 }

 public void refreshPrices() {
  // actualiza precios de mercado de las coins
 }

 public void showPortfolio() {
  // muestra balance total y por coin
 }

 public double calculateTaxes() {
  // calcula impuestos sobre ganancias
 }

}
```

Ejercicio N° 1

Sólo una responsabilidad

Sólo una responsabilidad

Manos a la obra: 🙌

Tenemos una aplicación cuya función es calcular la suma del área de dos polígonos y mostrar el resultado al usuario. En un principio podríamos tener clases para definir los objetos Cuadrado y su atributo int lado; y Circulo con atributo int radio. que extienden de la clase padre Poligono.

Los métodos a crear, además de los get y set, serán imprimirPoligono(), multiplicarArea() y sumarArea().

Crea el programa de manera tal que cada clase tenga una única responsabilidad.

Si consideras necesario crear más clases (lo será), hazlo!

Tiempo🕐: *30 minutos*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Comprender la importancia de los principios SOLID para el desarrollo de software*
- ✓ *Reconocer la implementación del principio de Responsabilidad Única*

English Always | alkemy

#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: página 9*
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 👋🏻

English Always | alkemy

Momento:

Time-out!

⏰5 min.