Clase grabada

Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

English Always | alkemy

Clase grabada

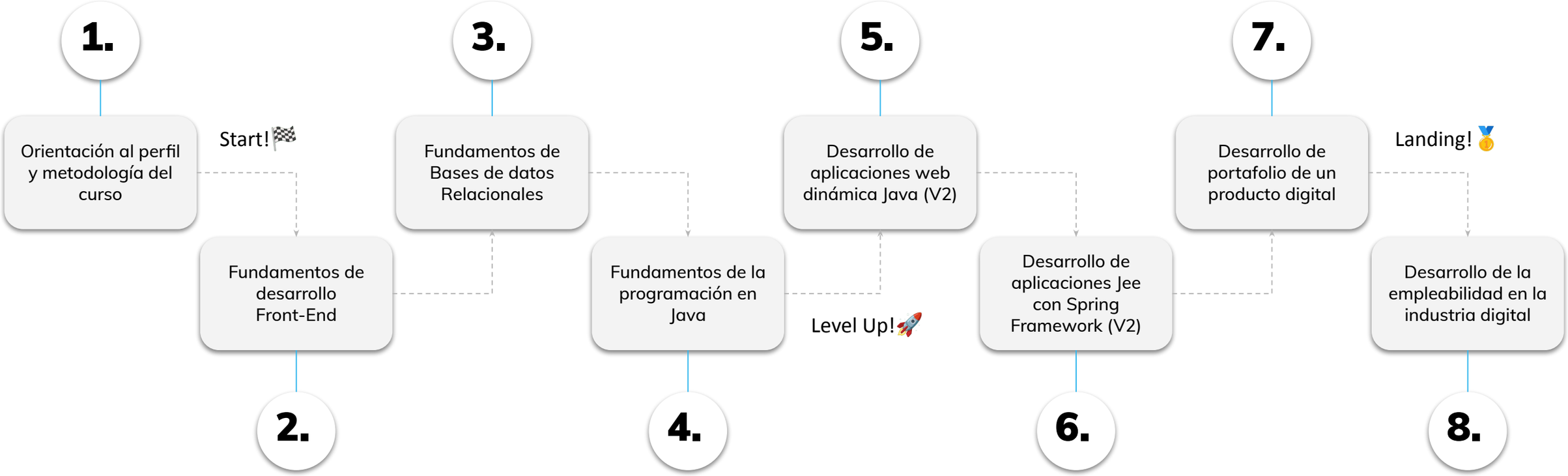
Polimorfismo y principios básicos de diseño

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

English Always | alkemy

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



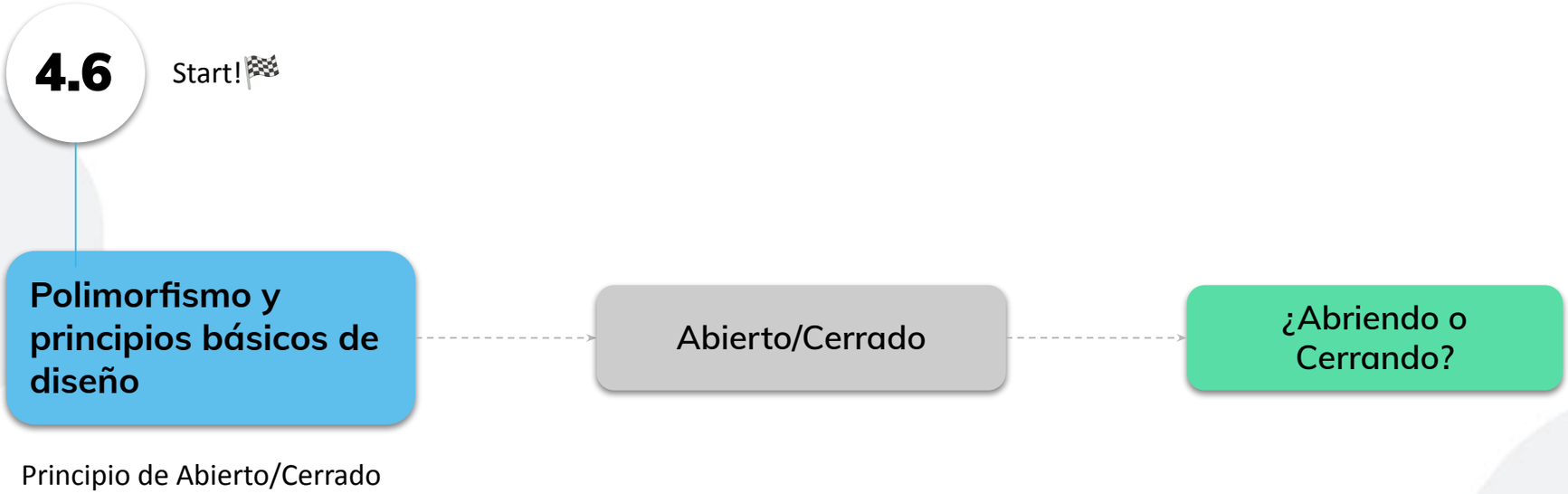
English Always | alkemy

REPASO CLASE ANTERIOR

En la clase anterior trabajamos 📚:

- ✓ *Principios de diseño SOLID*
- ✓ *Principio de Responsabilidad Única*

LEARNING PATHWAY



English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- ***Comprender el principio de Abierto/Cerrado***

Rompehielo🥶

Ampliando: 🙌

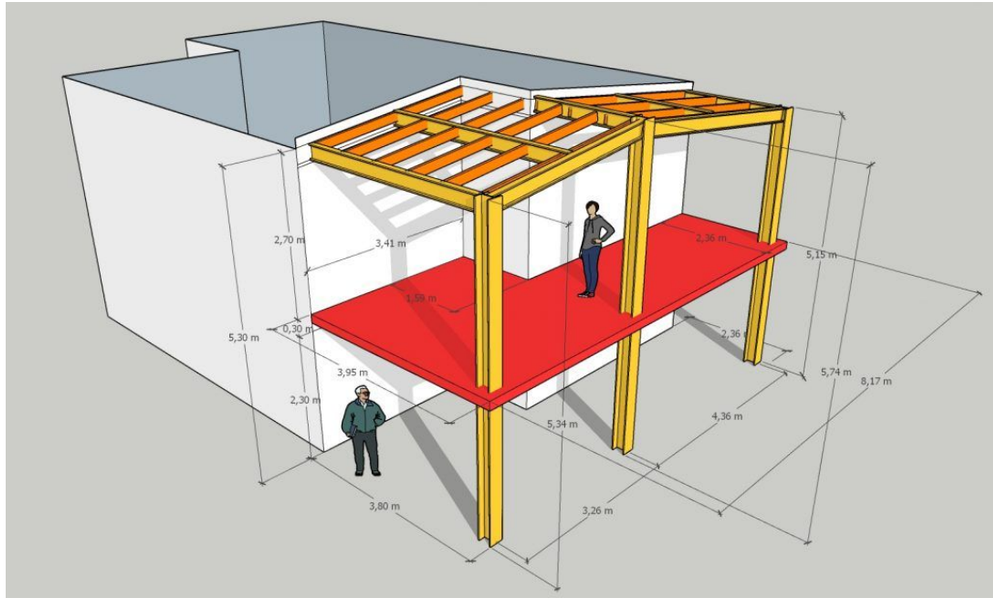
En la imagen podemos identificar una ampliación de una casa: un posible garage abajo y una sala a definir arriba.

Respondan en el chat o levantando la mano: ✍️

1. En lugar de construir una casa nueva más grande, ¿qué se hizo en este caso?
2. ¿Cuáles son las ventajas de ampliar vs reconstruir la casa?

Podemos decir que ésta opción es más económica y rápida, porque se mantiene la estructura original.

¿Pueden pensar en otros ejemplos de sistemas que se expanden agregando nuevos componentes en lugar de modificando los existentes?



Principio de Abierto/Cerrado

Abierto/Cerrado

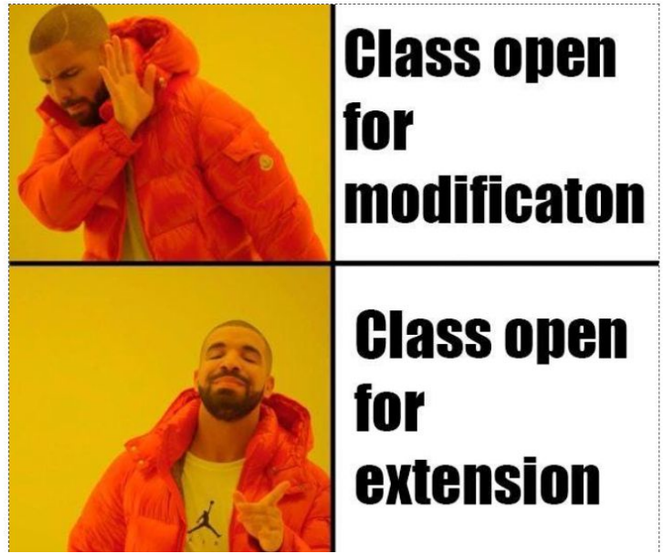
Este principio establece que una entidad de software (**clase, módulo, función**, etc) **debe quedar abierta para su extensión, pero cerrada para su modificación**.

Con **abierta para su extensión**, nos quiere decir que una entidad de software debe tener la **capacidad de adaptarse a los cambios** y nuevas necesidades de una aplicación, pero con la segunda parte de "**cerrada para su modificación**" nos da a entender que la adaptabilidad de la entidad no debe darse como resultado de la modificación del core de dicha entidad si no **como resultado de un diseño que facilite la extensión sin modificaciones**.

Abierto/Cerrado

Las ventajas que nos ofrece diseñar el código aplicando este principio es un **software más fácil de mantener**:

- Minimiza los cambios en la base de código de la aplicación y amplia funcionalidades sin modificar partes básicas de la aplicación probadas.
- Evita generar nuevos errores en funcionalidades que ya estaban desarrolladas, probadas y funcionando correctamente.
- A la hora de implementar test unitarios nos ayudará a no tener que modificar dichos tests cada vez que se mejore y amplíe el código.
- Mejora la fiabilidad de nuestra base de código



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!

Imaginemos que tenemos una wallet básica con la siguiente estructura:

```
public class Wallet {
 public void payWithCreditCard(double amount){
   //pago con tarjeta de crédito
  }
 public void payWithDebitCard(double amount){
   //pago con tarjeta de débito
  }
 public void payWithPayPal(double amount){
   //pago con PayPal
  }
}
```

LIVE CODING

Ejemplo en vivo

Ahora, si el cliente nos informa que se necesita agregar un nuevo método de pago, como criptomonedas, deberíamos modificar la clase Wallet agregando un nuevo método. Esto es algo que deberíamos evitar (el modificar la clase).
Una buena alternativa para respetar el principio de Abierto/Cerrado sería crear una interfaz PaymentMethod:

```
public interface PaymentMethod {
  void pay(double amount);
}
```

De esta manera, se podrían crear nuevas clases de pago implementando la interfaz PaymentMethod sin alterar Wallet.

Tiempo: 30 minutos

Ejercicio N° 1

¿Abriendo o Cerrando?

¿Abriendo o Cerrando?

Manos a la obra: 🙌

Teniendo en cuenta el ejercicio anterior donde aplicamos el principio de Responsabilidad Unica con la clase padre Poligono, ahora es necesario refactorizar ese código para incluir en los cálculos un Triangulo.

Los calculos (multiplicarArea() y sumarArea()) deberían estar en una clase llamada Cálculo. Ésta clase no se puede modificar para respetar el principio de Abierto/Cerrado.

En este ejercicio, debes modificar el diseño de tal manera que permite añadir funcionalidades, en este caso concreto nuevos tipos de polígonos, sin modificar las clases y métodos existentes.

Tiempo🕐: *30 minutos*

¿Abriendo o Cerrando?

Paso a paso: ⚙️ ¡Una ayuda lógica!

Creando el método área en la clase padre Poligono, los objetos que extienden de ella, Cuadrado y Circulo, implementaran la lógica del cálculo del área, que será diferente para cada polígono.

De este modo la clase Calculo queda mucho más limpia y no será necesario modificarla cada vez que añadamos un nuevo tipo de objeto Poligono o queramos modificar la lógica del cálculo del área en alguna implementación concreta.

Tiempo🕐: *30 minutos*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ ***Reconocer la implementación del principio de Abierto/Cerrado***

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.