Clase grabada

Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

English Always | alkemy

Clase grabada

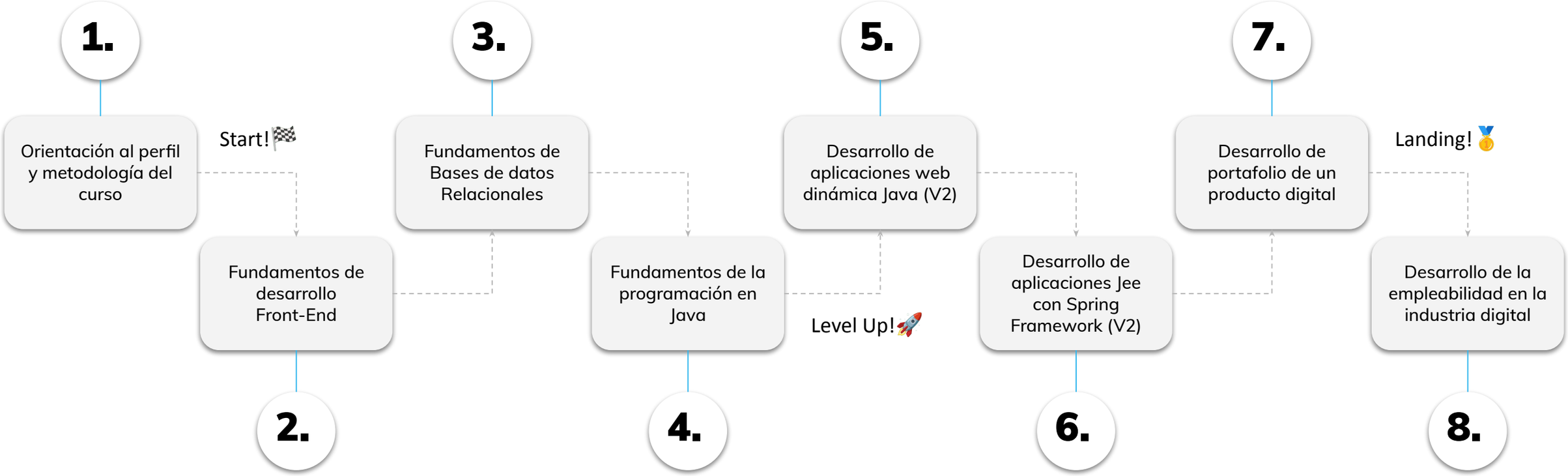
Polimorfismo y principios básicos de diseño

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 📚:

- ✓ *Diagramas de Clase*

English Always | alkemy

LEARNING PATHWAY

4.6 Start! 🏁

Polimorfismo y principios básicos de diseño

Herencia.
Atributos y constructores en la herencia.

Herencia

Veterinaria 1.0

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- ***Comprender el concepto de herencia y su implementación en programación***

Rompehielo🥶

¿Qué representa la imagen?: 🙌

Respondan levantando la mano o en el chat!

- ¿Cómo crees que se puede representar un árbol genealógico en programación?
- ¿Qué tipo de relación definirías?

Herencia

Herencia

¿Qué es y para qué se utiliza?

La herencia es una de las características fundamentales de la Programación Orientada a Objetos.
Mediante la herencia podemos **definir una clase a partir de otra ya existente**.

La clase nueva se llama clase **hija** o **subclase** y la clase existente se llama clase **padre** o **superclase**.

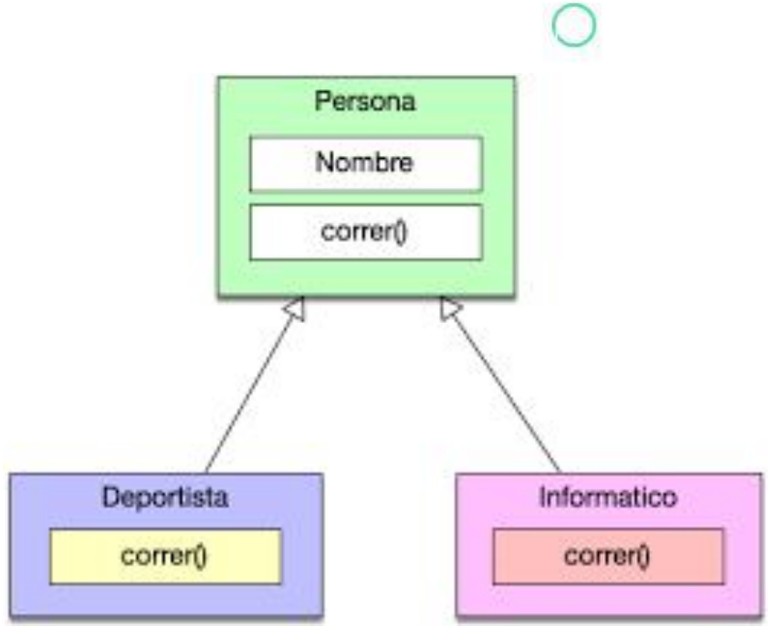
En esta relación, la frase “Un objeto **es un-tipo-de** una superclase” debe tener sentido, por ejemplo: un perro **es un tipo** de animal, o también, una heladera **es un tipo** de electrodoméstico.

Herencia

La herencia apoya el concepto de "**reutilización**", es decir, cuando deseamos crear una nueva clase y ya existe una clase que incluye parte del código que queremos, podemos reutilizar los campos y métodos de la clase existente.

La manera de usar herencia es a través de la palabra **extends**.

```
public class SubClase extends SuperClase {
      // Contenido de la clase ;
}
```



Herencia y atributos

Herencia: Atributos

En la **superclase** los atributos están creados con el modificador de acceso **protected**. Esto es porque el modificador de acceso protected permite que las subclases puedan acceder a los atributos de la superclase **sin la necesidad de getters y setters**.

Todos los atributos de la superclase se heredan a la clase hija.

Una subclase puede acceder a los miembros **públicos** y **protegidos** de la clase padre como si fuesen miembros propios.

```
public class Vehiculo {

    protected int id;
    protected String tipo;
    protected String marca;

}

public class Coche extends Vehiculo{

    private String color;

}
```

LIVE CODING

Ejemplo en vivo

Practicando con Herencia:

Veamos un ejemplo simple de dos clases relacionadas a través de la herencia.

1. *Crear un programa donde se ejecute la declaración de una superclase Persona con atributos nombre y rut. Luego, declarar una clase hija Empleado con el atributo “cargo”. También declarar una clase hija Cliente con atributo “tipo” (titular o adjunto).*

Tiempo: 20 minutos

Herencia y constructores

Herencia: Constructores

Los constructores no se heredan de manera directa. Todos los constructores definidos en una superclase **pueden ser usado**s desde constructores de las subclases **a través de la palabra reservada super**.

La palabra clave **super** es la que permite elegir qué constructor quiero usar.

Si la superclase tiene definido el constructor vacío y no colocamos una llamada explícita super, se llamará el constructor vacío de la superclase.

```
public class Coche extends Vehiculo{

    private String color;

    public Coche(String tipo, String marca, int id, String color) {
        super(tipo, marca, id);

        this.color = color;

    }
```

Herencia: Constructores

La palabra clave **super** sirve para hacer referencia o llamar a los atributos, métodos y constructores de la superclase en las clases hijas.

super.atributoClasePadre;
super.metodoClasePadre;

```
public class Coche extends Vehiculo{

    private String color;

    public Coche(String tipo, String marca, int id, String color) {
        super(tipo, marca, id);

        this.color = color;

    }
```

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

Practicando con Herencia:

Veamos un ejemplo simple de dos clases relacionadas a través de la herencia.

1. *Dada la clase Cuenta, crear una subclase CuentaCorriente que herede el constructor usando super(). Crear un objeto CuentaCorriente y verificar el constructor.*

Tiempo: 20 minutos

Ejercicio N° 1

Veterinaria 1.0

Veterinaria 1.0

Consigna ✍️:

Dada una clase Animal con atributos nombre y peso, crear una subclase Perro que herede los atributos y además incluya la raza.

Luego, crear un objeto Perro utilizando el constructor super() e imprimir todos sus valores por pantalla.

Tiempo🕐: *20 minutos*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ ***Comprender la utilidad de las relaciones de herencia en programación orientada a objetos.***

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: páginas 1 - 3*
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.