Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

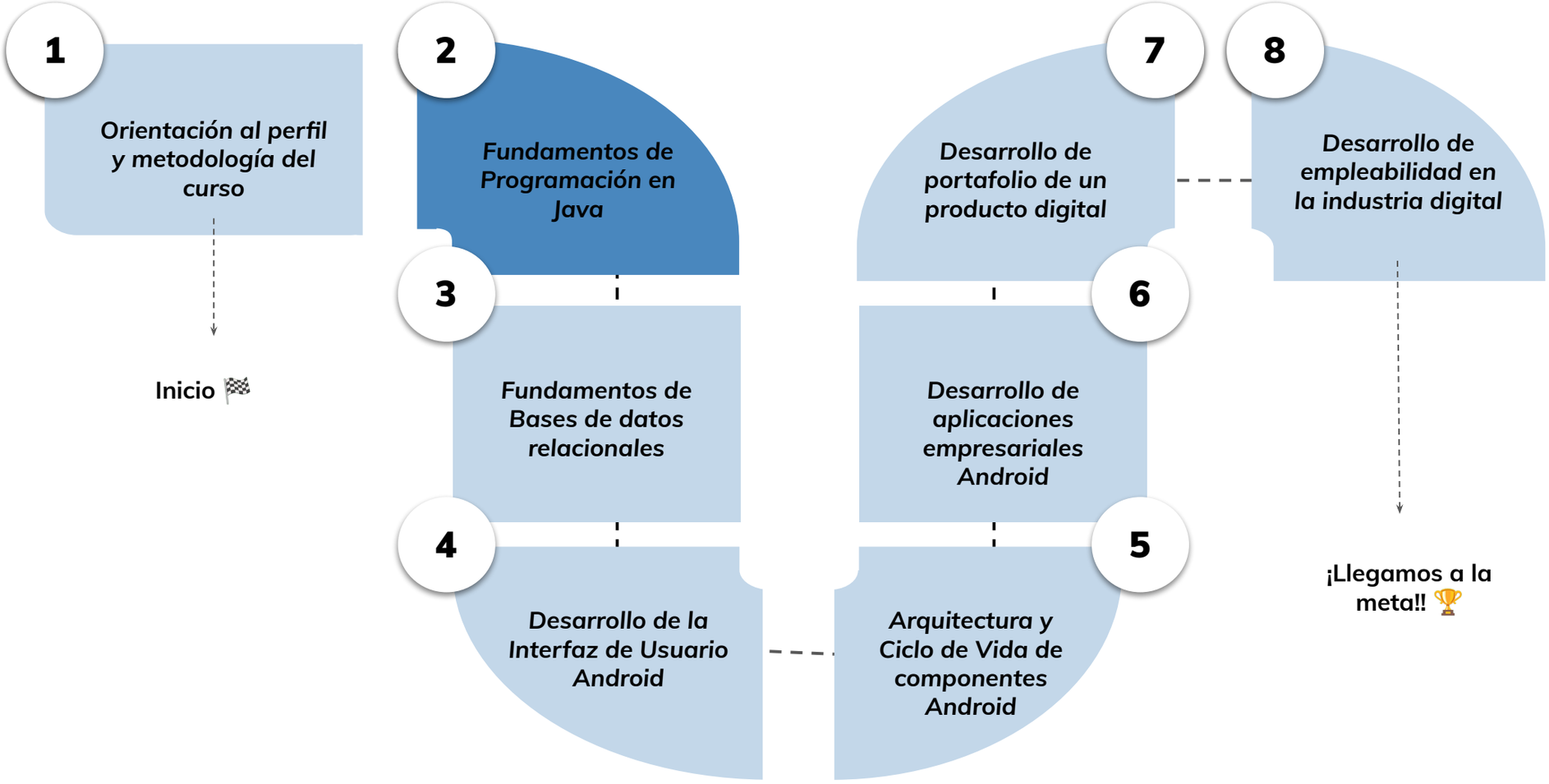
Clase grabada

Polimorfismo y principios básicos de diseño

AE6.1: Utilizar principios básicos de diseño orientado a objetos para la implementación de una pieza de software acorde al lenguaje java para resolver un problema de baja complejidad.

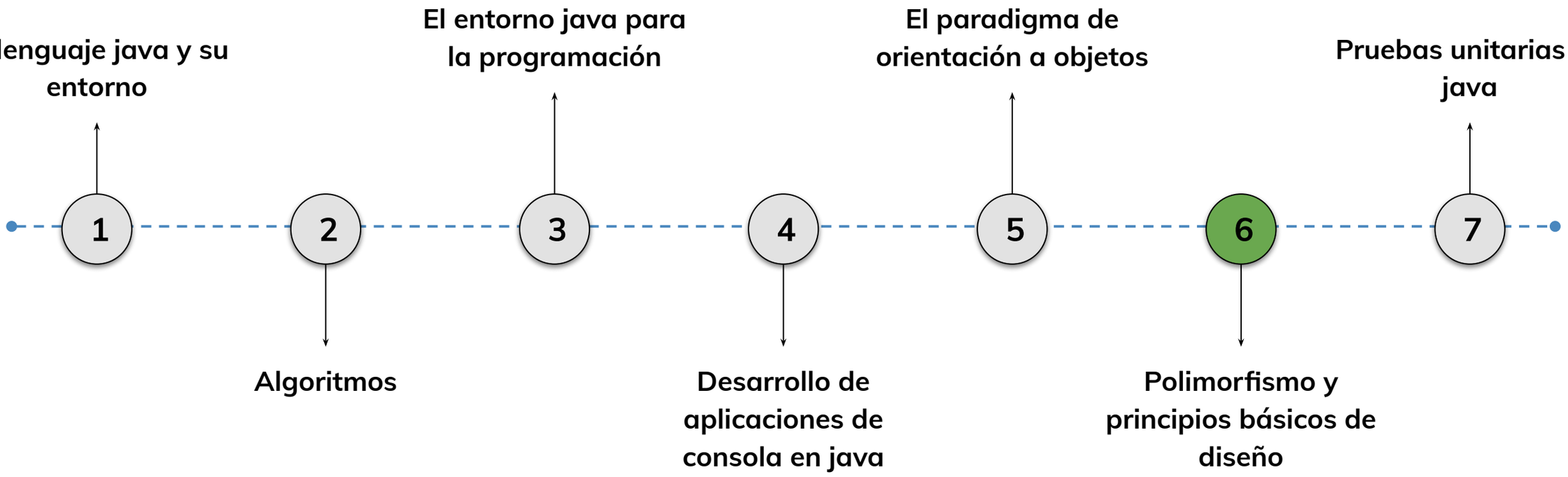
Hoja de ruta

*¿Cuáles **módulos** conforman el programa?*



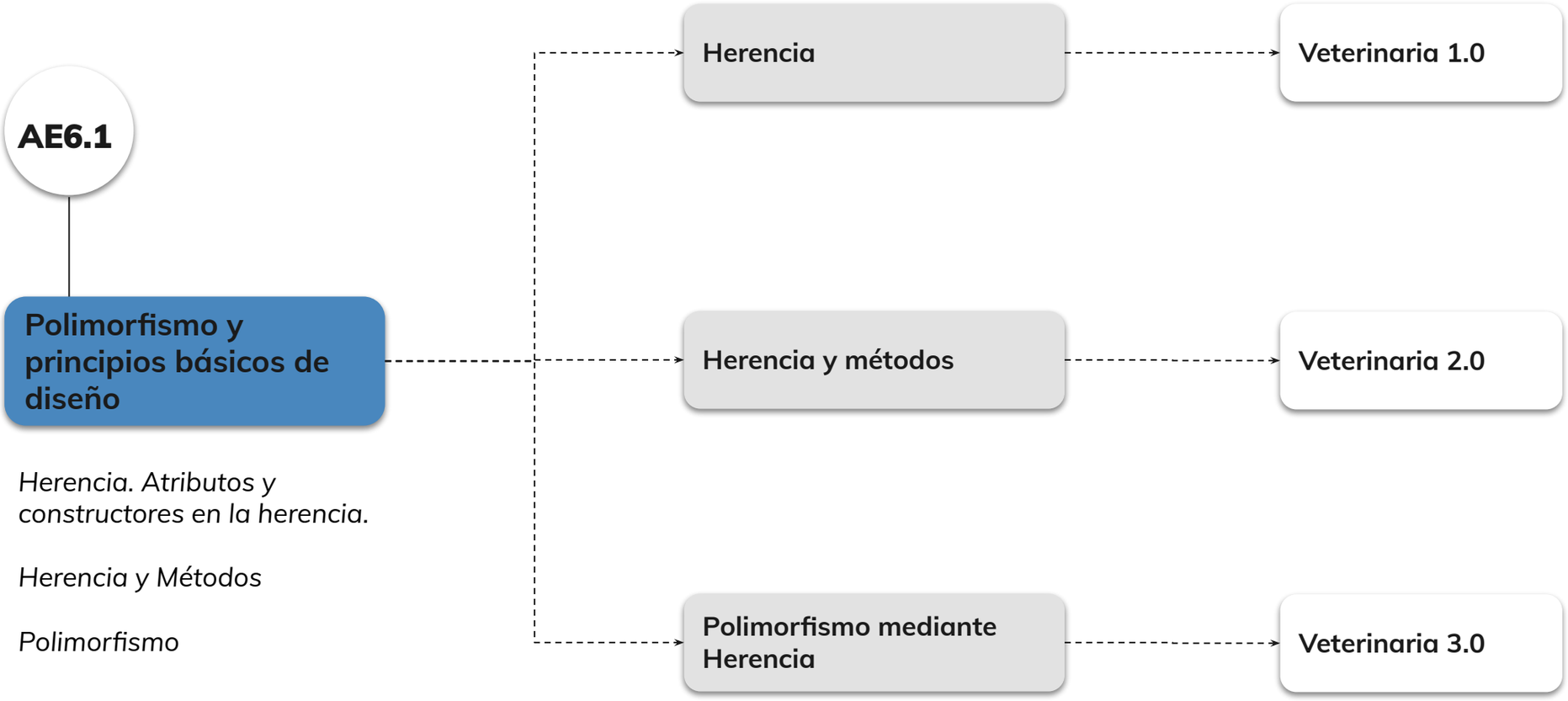
Roadmap de lecciones

*¿Cuáles **lecciones** estaremos estudiando en este módulo?*



Learning Path

¿Cuáles temas trabajaremos hoy?



alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Comprender el concepto de herencia y su implementación en programación.
- Comprender la implementación de la herencia de métodos en la programación orientada a objetos.
- Comprender el concepto de polimorfismo en programación orientada a objetos.
- Aprender a implementar el polimorfismo mediante herencia.

< > 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?

alkemy

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.

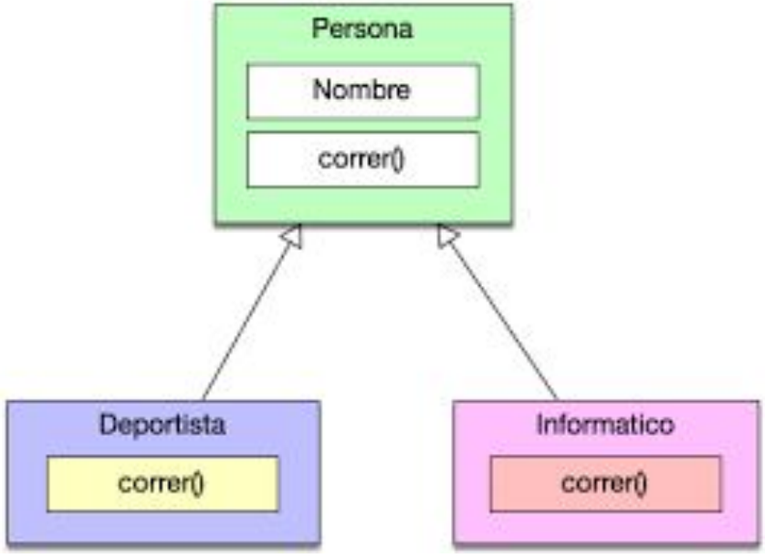
En esta relación, la frase “Un objeto es un-tipo-de una superclase” debe tener sentido, por ejemplo: un perro es un tipo de animal, o también, una heladera es un tipo de electrodoméstico.

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.

```java
public class Vehiculo {

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

}

public class Coche extends Vehiculo{

    private String color;

}
```

LIVE CODING

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

LIVE CODING

https://gist.github.com/Mariocanedo/26ec1018900abc0c4688af07f2faecae

Tiempo: 20 Minutos

alkemy

Herencia y constructores

Herencia: Constructores

Los constructores no se heredan de manera directa. Todos los constructores definidos en una superclase pueden ser usados 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;

    }
```

LIVE CODING

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*

Momento:

Time-out!

⏰5 -10 min.

alkemy

Herencia y métodos

Herencia y métodos

Repasemos algunas características de la Herencia:

- Una subclase hereda de la superclase sus componentes (atributos y métodos).
- Los constructores no se heredan.
- Una subclase puede acceder a los miembros públicos y protegidos de la superclase como si fuesen miembros propios.
- Una subclase puede añadir a los miembros heredados, sus propios atributos y métodos (extender la funcionalidad de la clase).
- También puede modificar los métodos heredados.
- Una subclase puede, a su vez, ser una superclase, dando lugar a una jerarquía de clases.

Herencia y métodos

¿Cómo se comportan los métodos en la Herencia?: Sobreescritura

Los métodos heredados pueden ser redefinidos en las clases hijas. Este mecanismo se denomina sobreescritura. La sobreescritura permite a las clases hijas utilizar un método definido en la superclase.

Una subclase sobreescribe un método de su superclase cuando define un método con las mismas características (nombre, número y tipo de argumentos) que el método de la superclase. Las subclases emplean la sobreescritura de métodos la mayoría de las veces para agregar o modificar la funcionalidad del método heredado de la clase padre.

Herencia y métodos

Sobreescritura: @Override

La sobreescritura permite que las clases hijas sumen sus métodos en torno al funcionamiento, y esto se logra escribiendo la anotación @Override arriba del método que queremos sobreescribir. El método debe llamarse igual en la subclase como en la superclase.

```
@Override  //indica que se modifica un método heredado
  public void leer(){
}
```

Cuando en una subclase se redefine un método de una superclase, se oculta el método de la clase padre y todas las sobrecargas del mismo en la clase padre. Por eso para ejecutar el método leer() se debe escribir super.leer();

LIVE CODING

¡Probando los métodos!:

Vamos a trabajar sobre la clase **Cuenta** *con los métodos* ***depositar()*** *y* ***retirar()****.*

Luego, crearemos las ***subclases*** *CuentaPesoCL y CuentaUSD que implementarán estos métodos realizando el cambio de moneda correspondiente.*

Vamos a mostrar por pantalla los resultados para verificar.

Tiempo: 30 Minutos

alkemy

Ejercicio N° 2

Veterinaria 2.0

Veterinaria 2.0

Contexto: 🙌

Seguiremos trabajando sobre las clases Animal y Perro creadas en la clase anterior!

Consigna: ✍️

1- Declara los siguientes métodos en la superclase Animal: comer(), dormir() y emitirSonido().

2- La subclase Perro tendrá el siguiente comportamiento:
comer(): llama al método comer() de la superclase Animal
dormir(): llama al método dormir() de la superclase
emitirSonido(): return "Guau";

3- Declara otra subclase Gato que también hereda de Animal e implementa:

comer(): llama al método comer() de la superclase Animal

dormir(): llama al método dormir() de la superclase

emitirSonido(): return "Miau";

Veterinaria 2.0

En el método main:

Finalmente, crea objetos de tipo Perro y Gato y llama a sus métodos para verificar que el llamado a los métodos y su sobreescritura estén bien implementados!

Tiempo🕛: *25 minutos*

Polimorfismo

Polimorfismo

¿Qué es el polimorfismo?:

En programación orientada a objetos, polimorfismo es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación. Dicho de otro modo el objeto como entidad puede contener valores de diferentes tipos durante la ejecución del programa.

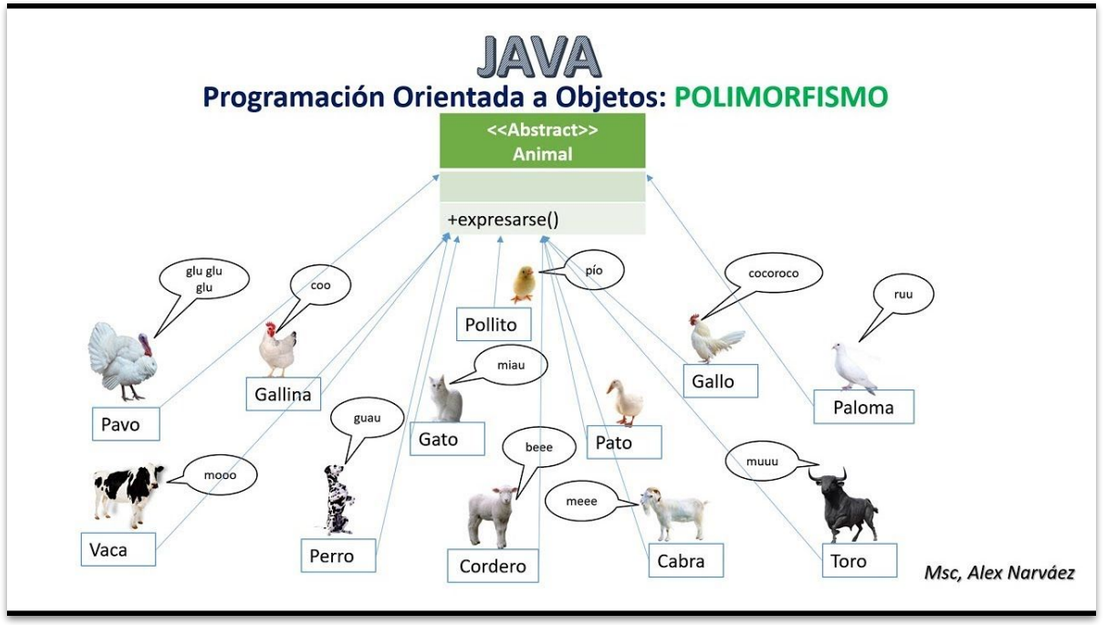
Basicamente, el polimorfismo en Java alude al modo en que se pueden crear y utilizar dos o más métodos con el mismo nombre para ejecutar funciones diferentes.

Polimorfismo

Veamos un ejemplo:

Imagina que tienes una mascota. Puede ser un perro, un gato o un pájaro. Cada uno de ellos tiene sus propias características y comportamientos específicos, como ladrar, maullar o volar. Ahora, piensa en cómo podrías representar todas estas mascotas en un programa de Java.

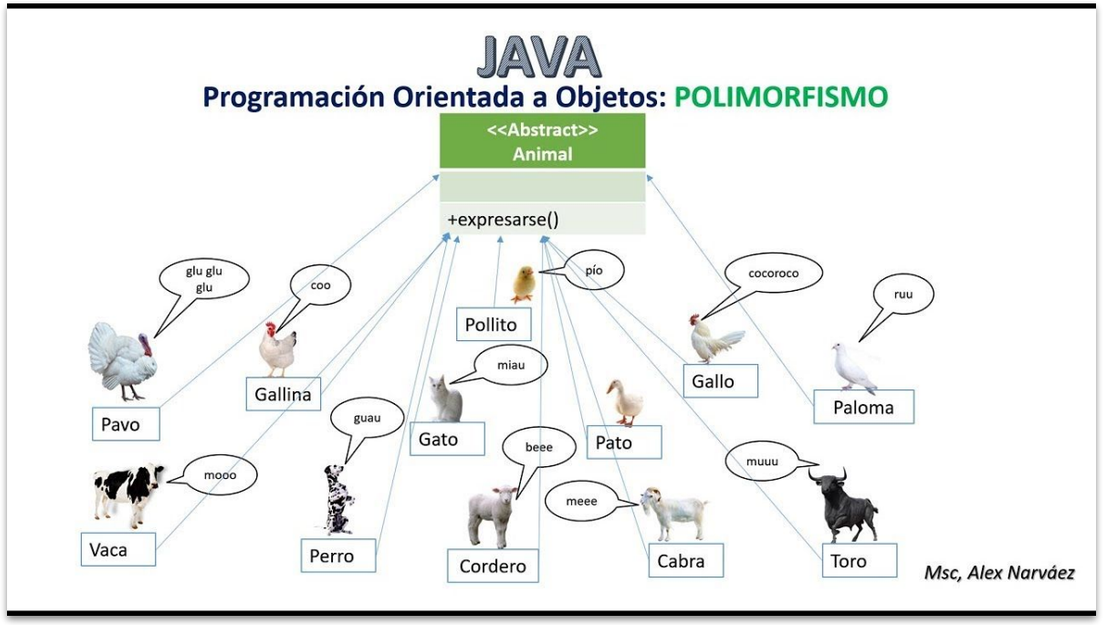
En lugar de crear una clase separada para cada tipo de mascota, podemos utilizar el polimorfismo para tratar a todas las mascotas de manera general. Esto significa que podemos tener una clase genérica llamada "Mascota" y luego crear subclases como "Perro", "Gato" y "Pájaro" que heredan de la clase "Mascota".



Polimorfismo

¿Pero, es igual a la clase Animal de los ejercicios?:

En realidad en este caso, por ejemplo, la magia del polimorfismo ocurre cuando creamos un arreglo o una lista de mascotas.
Podemos agregar instancias de diferentes subclases en la misma lista y tratarlas como si fueran del tipo genérico "Mascota".
Cuando llamamos al método "hacerSonido()" en una mascota particular, Java se encarga de invocar el comportamiento correspondiente según el tipo de mascota real. Por ejemplo, si la mascota es un perro, se ladrará; si es un gato, se maullará; y así con todas las mascotas que agregues.



LIVE CODING

Apliquemos lo aprendido en el proyecto de Billetera Virtual:

1- Declararemos una superclase FormaDePago con el método void realizarPago():

2- Luego se crean subclases TarjetaDeCrédito y Moneda que heredan de FormaDePago. La particularidad del pago con tarjeta es poder elegir cantidad de cuotas, mientras que Moneda puedes elegir el tipo de moneda.

3- En la clase Billetera se debe declarar una referencia de tipo FormaDePago: FormaDePago metodoPago;

4- De esta manera podremos asignar el método de pago según se necesite:

```
metodoPago = new TarjetaDeCredito();
metodoPago = new Moneda();
```

Y utilizar polimorfismo al llamar: metodoPago.realizarPago();

Tiempo: 30 Minutos

LIVE CODING

Polimorfismo con Interfaces:

1- Definir la interfaz

2- Implementar clases concretas

3- Uso polimórfico en el cliente

Explicación breve:

- *La variable de tipo Notificador puede hacer referencia a cualquier clase que implemente la interfaz.*
- *El cliente no necesita saber si se envía un correo o un SMS: depende de la abstracción, no de la implementación concreta.*

Tiempo: 30 Minutos

Ejercicio N° 3

Veterinaria 3.0

Veterinaria 2.0

Contexto: 🙌

Vamos a manipular la clase Animal que contenga un método hacerRuido() que devuelva un saludo “Hola”.

Luego, las clases Perro y Gato (que extienden de Animal) deben sobreescribir el método hacerRuido() con el ruido que corresponda a cada uno.

Consigna: ✍️

En el main vamos a crear un ArrayList de animales y los siguientes animales:

- Animal a = new Animal();
- Animal b = new Perro();
- Animal c = new Gato();

Agregar a la lista a cada uno y luego, con un for each, recorrer la lista llamando al método hacerRuido() de cada ítem.

Tiempo🕐: *30 minutos*

Resumen

¿Qué logramos en esta clase?

- ✓ *Comprender el uso del polimorfismo en herencia*
- ✓ *Reconocer la correcta implementación del polimorfismo en programación orientada a objetos*
- ✓ *Comprender la utilidad de las relaciones de herencia en programación orientada a objetos*
- ✓ *Reconocer la importancia de la sobreescritura en los métodos heredados*
- ✓ *Comprender la implementación de la herencia en métodos*
- ✓ *Comprender la utilidad de las relaciones de herencia en programación orientada a objetos*

¿Alguna consulta?

alkemy

¡Ponte a prueba!

Momento de ejercitación

Te invitamos a aprovechar esta última sección del espacio sincrónico para realizar de manera individual las ***actividades disponibles en la plataforma.*** *Estas propuestas son clave para afianzar lo trabajado y* ***forman parte obligatoria del recorrido de aprendizaje.***

💪 Análisis de caso ———— 💪 Selección Múltiple

💪 Comprensión lectora

Si al resolverlas surge alguna duda, compartela o tráela al próximo encuentro sincrónico.

alkemy

< ¡Muchas gracias! >