Programación orientada a objetos

Colaboración y herencia

{desafío} latam_

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

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Aplicar las técnicas de POO para crear clases que contengan otra clase conociendo técnicas de colaboración.*
- *Aplicar la sobrecarga de métodos con Herencia para entender la jerarquía de clases.*

¿Qué es herencia y colaboración?

{desafío} latam_

/* Colaboración */

Colaboración

Declaración de Objetos dentro de otro Objeto

- Objetos locales

Podemos declarar un atributo de tipo Objeto a nivel de clases y para esto necesitamos conocer el objeto y su relación con la clase, también podemos utilizar instancias de otros objetos solo en operaciones particulares de la clase.

```
public class Persona {
     private String nombre;
     private String rut;
     private double altura;
     // Asociación de dependencia - colaboración de objetos
     private Lapiz lapiz;
```

Colaboración

Declaración de Objetos dentro de otro Objeto

Se crea un atributo de tipo Objeto llamado Lapiz y se agrega al constructor con parámetros de la clase Persona.

```
public Persona(String nuevoNombre, String nuevoRut, double nuevaAltura,Lapiz nuevoLapiz) {
     // A esto se le llama sobrecarga de métodos
     this(nuevoNombre,nuevaAltura);
     this.rut = nuevoRut;
     this.lapiz = nuevoLapiz;
}
```

Colaboración

Declaración de Objetos dentro de otro Objeto

Dentro de nuestra clase Persona podemos tener métodos u operaciones que necesiten un objeto solo para un evento en especial, aquí también tenemos colaboración, pero no a nivel de clases, sino a nivel de método, donde el objeto nace y muere dentro del método.

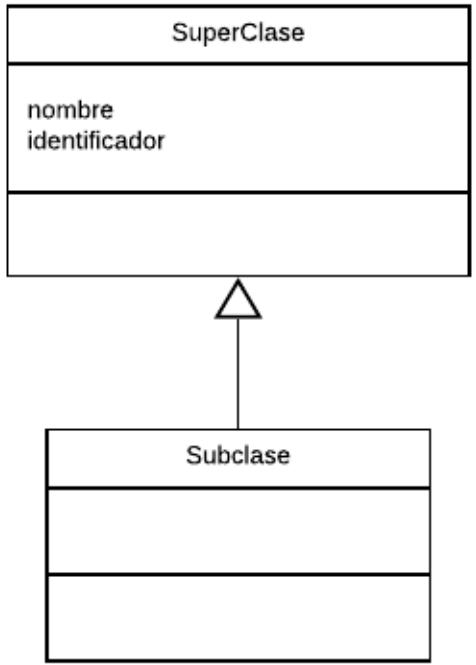
```
public Cuaderno reciboCuaderno(Cuaderno cuadernito){
     return cuadernito;
}
```

Podemos ver que el objeto Cuaderno tiene sus propios atributos, pero a su vez se utiliza en clase Persona, no siendo parte de esta como tal, sino solo en su método.

/* Herencia */

Herencia

- Heredar atributos y métodos desde una superclase hacia una o más subclases.
- Permite reutilizar atributos y métodos.
- Sirve para categorizar las clases de un programa.



Herencia - super

Crear clases que heredan atributos y métodos de otra.

A las clases que heredan elementos de otras se les llama **subclase**, y a las que heredan sus elementos a otras se les llama **superclase**.

Para realizar la herencia en Java necesitamos utilizar la palabra reservada ***extends*** esta palabra le indica a Java que se está realizando herencia de dos objetos.

```
public class Programador extends Persona {
     private String lenguaje;
}
```

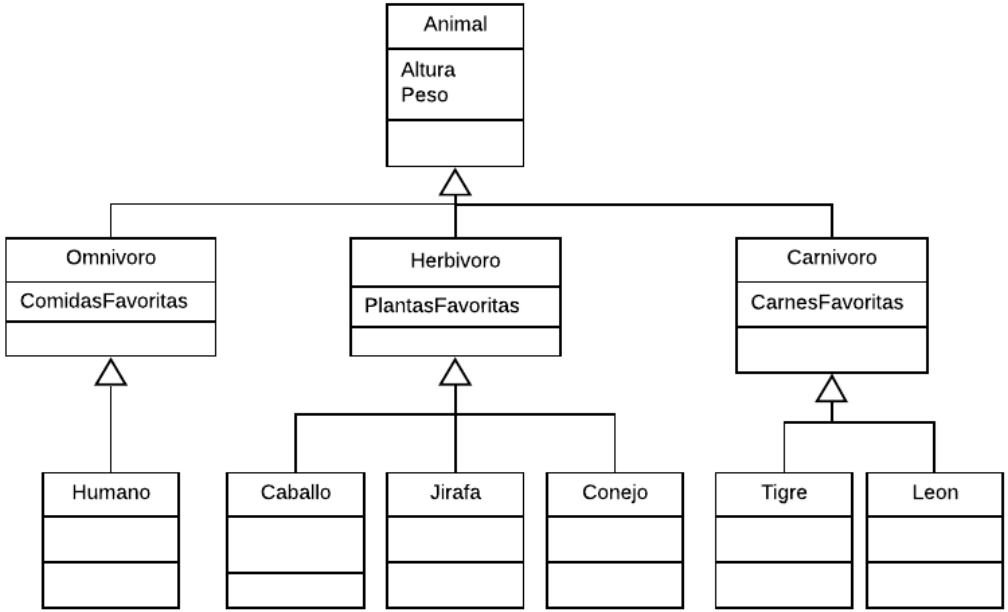
Herencia - super

El uso de la palabra reservada **super** se utiliza para la sobrecarga de métodos con herencia, ya que Java interpreta este super como la llamada al constructor padre.

Siguiendo el ejemplo de la Clase Persona llamaremos siempre a la firma del constructor padre y después se agregan los atributos propios de la clase hija.

```
public Programador(String nombre, String rut, double altura,Lapiz lapiz, String lenguaje)
{
     super(nombre,rut,altura,lapiz);
     this.lenguaje= lenguaje;
}
```

Ejemplo


Animal
Altura
Peso
Omnivoro
ComidasFavoritas
Herbivoro
PlantasFavoritas
Carnivoro
CarnesFavoritas
Humano
Caballo
Jirafa
Conejo
Tigre
Leon

Ejercicio guiado

Botillería

- Paso 1

Creamos las siguientes clases para el proyecto: Botella, Cerveza y Botillería.

- Paso 2

En la clase Botella crearemos los siguientes atributos:

- tipo Botella: String
- marca: String

Para luego generar su constructor y “getters and setters” correspondientes.

Botillería

- Paso 3

En la clase Cerveza crearemos el siguiente atributo: `precio: int`
Para luego generar su constructor y “getter and setter” correspondiente. Sin embargo, Cerveza es una subclase de Botella. Por lo tanto, usaremos la palabra reservada extends para aplicar herencia desde la subclase Cerveza hacia la superclase Botella.

- Paso 4

En la clase Botilleria crearemos los siguientes atributos:

- cerveza: Cerveza
- nombre: String

Para luego generar el constructor y “getters and setters” correspondientes, pero esta vez usaremos una clase Cerveza que viene extendida desde Botella.

Botillería

¡Compartamos el resultado!

¿Cómo quedó el código resultante?

La orientación a objetos tiene énfasis en la construcción y desarrollo de ciertas variables para manejar distintas partes del programa. Esto con la idea de que el código sea más estable y se ejecute sin mayores problemas.

/* Encapsulamiento */

Encapsulamiento

Los métodos públicos son accesadores y mutadores para atributos, más conocidos como Getter (Accesador) y Setter (Mutador).

Esto sirve para prevenir errores inesperados por una mala intervención de los atributos de una instancia del modelo del programa.

Encapsulamiento

Ventajas

- **Integridad**

Los datos se crean y modifican a través de accesadores.

- **Reglas de negocio**

Los accesadores pueden tener reglas de negocio y validaciones antes de modificar los datos.

- **Estabilidad**

Podría haber errores que hagan que se termine la ejecución si una variable recibe un dato que no soporta.

Ejemplo

Regla de negocio

Si tuviésemos que desarrollar una aplicación con nuestra clase Auto, pero el cliente nos dice que si uno de los autos apaga su motor mientras el auto está en movimiento, podría romperse una pieza del sistema del Auto, por ende, el cliente quiere prevenir que sus Autos se descompongan y que no se pueda apagar el motor hasta que el auto esté detenido.

Ejemplo

Regla de negocio

Usando atributos sin encapsulamiento

Viendo un ejemplo ERRÓNEO: al cambiar de private a public, se puede acceder a los atributos de la clase sin un accesador.

```
public class Auto{
     public String marca;
          public String modelo;
     public String color;
          public int velocidadActual;
     public boolean motorEncendido;
}
```

Ejemplo

Regla de negocio

Usando atributos sin encapsulamiento

Se agrega prueba este código en el método main y se puede ver que no hay errores de acceso a los atributos como antes. Esto no aplica reglas de negocio al momento de cambiar los datos de los atributos de auto y tenemos la regla de que los autos no pueden tener el motor apagado (motorEncendido = false) cuando la velocidad sea mayor a 0 (y es 50). Conclusión: Cliente molesto, compañeros de trabajo molestos, jefe molesto, etc. Por no aplicar la regla de negocio.

```
Auto instancia = new Auto();
System.out.println("Auto creado");
instancia.motorEncendido = true;
System.out.println("Motor Encendido: "+instancia.motorEncendido);
instancia.velocidadActual = 50;
System.out.println("Velocidad: "+instancia.velocidadActual);
instancia.motorEncendido = false;
System.out.println("Motor Encendido: "+instancia.motorEncendido);
```

Ejemplo

Regla de negocio

Usando atributos sin encapsulamiento

- Ahora se agrega un if para comprobar la velocidad antes de apagar el motor.
- Pero se va a tener que escribir este if cada vez que se necesite apagar el motor.
- Resultado: Compañeros de trabajo molestos, Jefe molesto. Cliente feliz y mucha pérdida de tiempo codificando los if.

```
Auto instancia = new Auto();
System.out.println("Auto creado");
instancia.motorEncendido = true;
System.out.println("Motor Encendido: "+instancia.motorEncendido);
instancia.velocidadActual = 50;
System.out.println("Velocidad: "+instancia.velocidadActual);
if(instancia.velocidadActual == 0){
          instancia.motorEncendido = false;
}
System.out.println("Motor Encendido: "+instancia.motorEncendido);
```

Ejemplo:

"Aplicando encapsulamiento"

Creando Accesadores y Mutadores

Vamos entonces a volver a cambiar los atributos a private y agregar getter, setters y métodos que permitan modificar los datos validando que se cumplan las reglas de negocio.

- Accesadores: devuelven valor de un atributo
- Mutadores Cambian valor de un atributo

Creando Accesadores y Mutadores

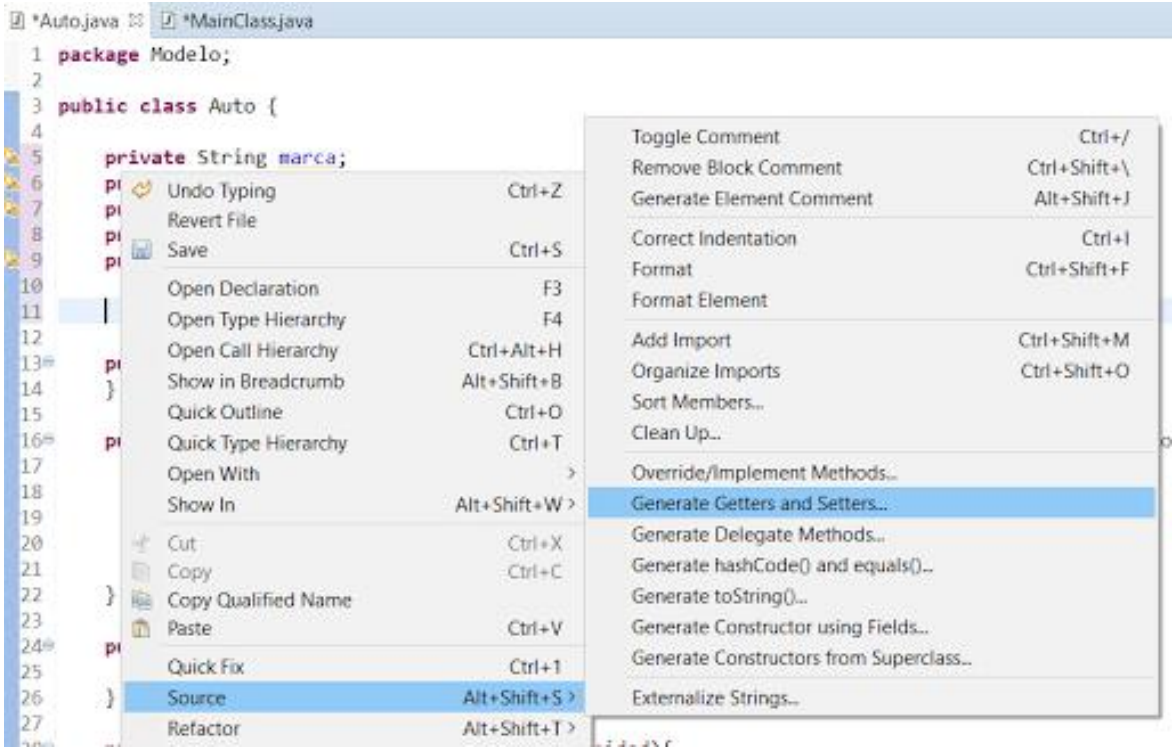
```
private String marca;
private String modelo;
private String color;
private int velocidadActual;
private boolean motorEncendido;

public void setMarca(String marca){
          this.marca = marca;
}

public String getMarca(){
          return this.marca;
}
```

Forma rápida de crear el código en Eclipse

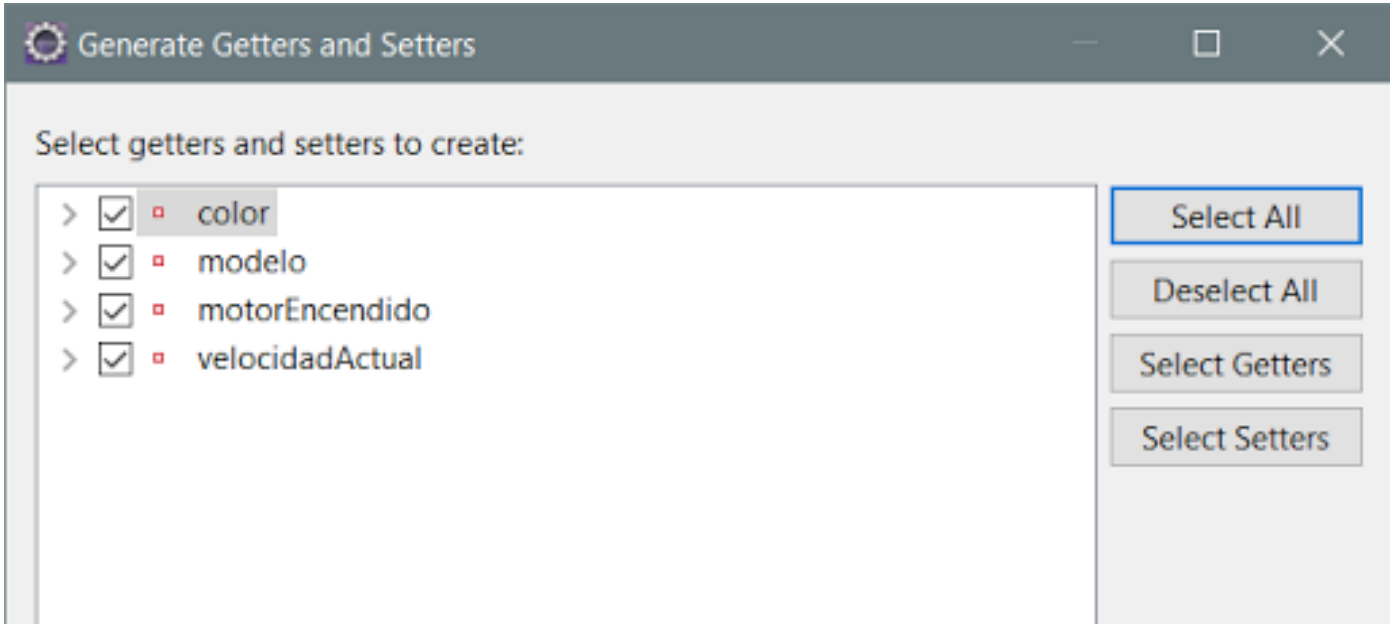
Alt + Shift + S → Clic derecho



{desafío} latam_

Forma rápida de crear el código en Eclipse

Seleccionar los necesarios. y dar clic en finish



{desafío} latam_

Corrigiendo el Main

Aparecen errores al cambiar los modificadores de acceso, se corrigen usando los getter y setter.

```
//Creamos la instancia
Auto instancia = new Auto();

//Encendemos el motor y aceleramos hasta 100
instancia.setMotorEncendido(true);
instancia.setVelocidadActual(100);

//Vamos a imprimir el estado de nuestra instancia:
System.out.println("Auto: ");
System.out.println("Velocidad: "+instancia.getVelocidadActual());
System.out.println("Motor Encendido: "+instancia.getMotorEncendido());
```

Usando el método apagarMotor()

```
public void apagarMotor(){
            if(this.velocidadActual == 0){
                        this.setMotorEncendido(false);
}
public static void main(String[] args) {
            //Creamos la instancia
            Auto instancia = new Auto();
            //Encendemos el motor y aceleramos hasta 100
            instancia.setMotorEncendido(true);
            instancia.setVelocidadActual(100);
            //Intentamos apagar el motor
            instancia.apagarMotor();
            //Vamos a imprimir el estado de nuestra instancia:
            System.out.println("Auto: ");
            System.out.println("Velocidad: "+instancia.getVelocidadActual());
            System.out.println("Motor Encendido: "+instancia.getMotorEncendido());
}
```

Usando el método frenar()

```
public static void main(String[] args) {
	//Creamos la instancia
	Auto instancia = new Auto();
	System.out.println("Auto creado");
	//Encendemos el motor y aceleramos hasta 100
	instancia.encenderMotor();
	System.out.println("Motor Encendido: "+instancia.getMotorEncendido());
	instancia.aumentarVelocidad(100);
	System.out.println("Velocidad: "+instancia.getVelocidadActual());
	//Frenamos hasta detenernos
	instancia.frenar();
	//Intentamos apagar el motor
	instancia.apagarMotor();
	//Vamos a imprimir el estado de nuestra instancia:
	System.out.println("Auto: ");
	System.out.println("Velocidad: "+instancia.getVelocidadActual());
	System.out.println("Motor Encendido: "+instancia.getMotorEncendido());
}
```

Modificador de acceso

protected

Desde	protected
La misma clase	Si
Una subclase que hereda de esta	Si
Una subclase que no hereda de esta	No
Otra clase, mismo paquete	Si
Clase de otro paquete	No

¿Dónde utilizamos la palabra SUPER?

¿Cuál de las siguientes sintaxis nos sirve para declarar un atributo de tipo objeto Mesa en otra Clase?

Próxima sesión...

- *Desafío guiado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales