# Programación orientada a objetos

## Objeto, clases y sobrecarga

{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_

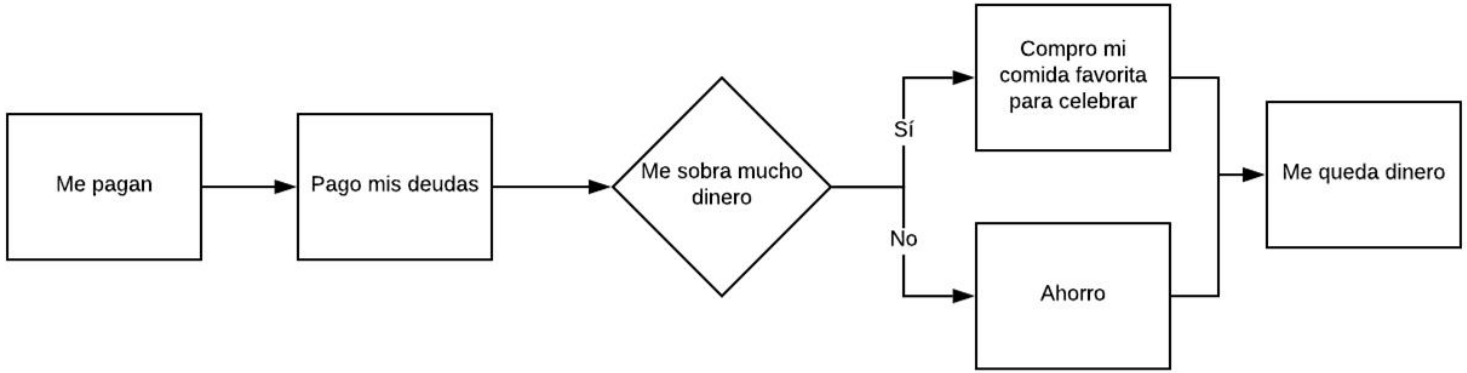
# ¿Qué aprenderás en esta sesión?

- *Comprender el paradigma de la Programación Orientada a Objetos (POO).*
- *Comprender la estructura e instancia de una clase y sus atributos para aplicar la sobrecarga de métodos.*

# ¿Qué entendemos por POO?

# Una pincelada de historia

- Programación Lineal / Tipo espagueti
- Programación Estructurada
- Programación Orientada a Objetos



# /* Objetos y clases */

# Los objetos

Tienen atributos, comportamientos y estado.

**Ejemplo**: Automóvil

- Atributos o propiedades: color, marca, modelo y todas esas características del objeto que lo hacen mantener un estado único que lo diferencia de otros objetos.
- Comportamiento: lo que el objeto puede hacer, como Avanzar, Retroceder, Encenderse, entre otras.

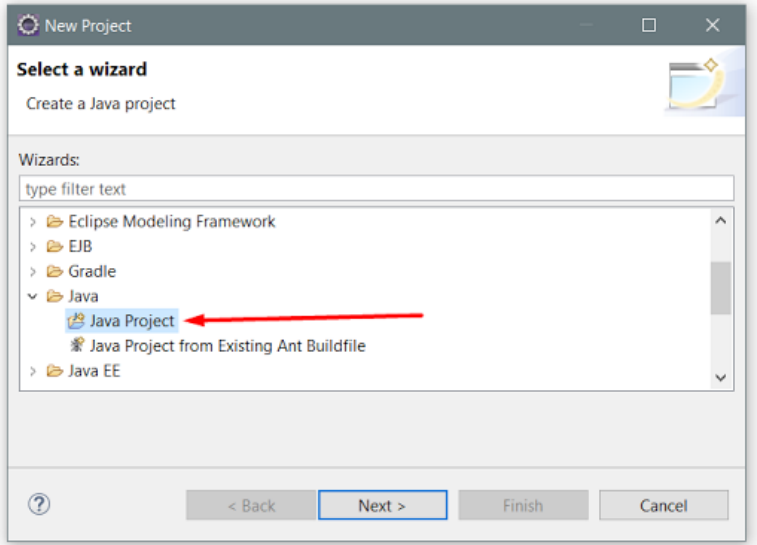
*Cada funcionalidad contempla una o más piezas diferentes que interactúan entre sí para lograr el objetivo, por ende, tenemos un sistema completo dentro del auto, compuesto por piezas modulares.*

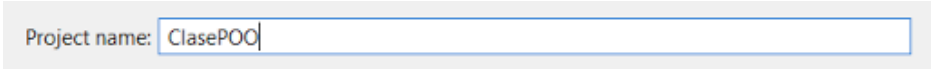
# Los objetos

En la Programación Orientada a Objetos tenemos muchas piezas y todas ellas con características que cumplen funciones por sí solas o trabajando en conjunto con otras piezas. A estas piezas les llamamos objetos (que en Java se les conoce como “Plain Old Java Object” o también conocidos como “POJO”).

Las características de estos objetos las almacenamos en variables y las funciones las llamamos métodos; los métodos, no son nada más que porciones de código dentro de un objeto.

# Creando el proyecto

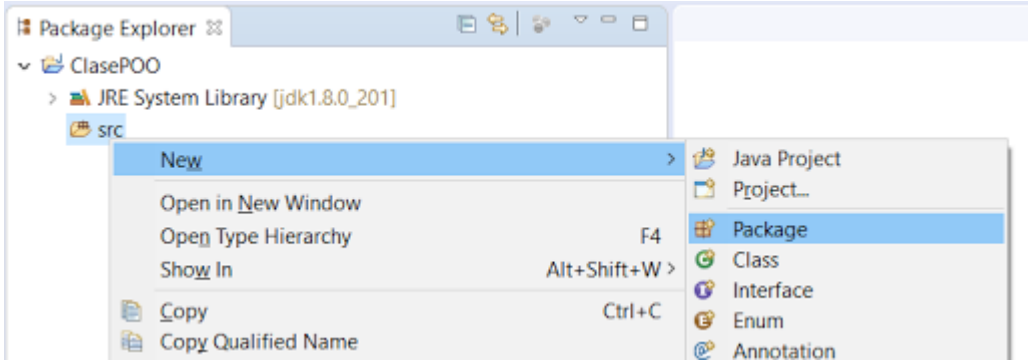


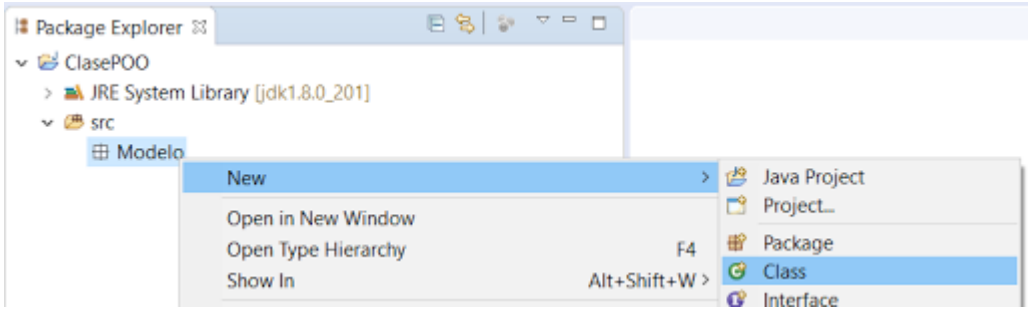

{desafío} latam_

# Creando el paquete

El paquete se debe llamar Modelo:
New > Package

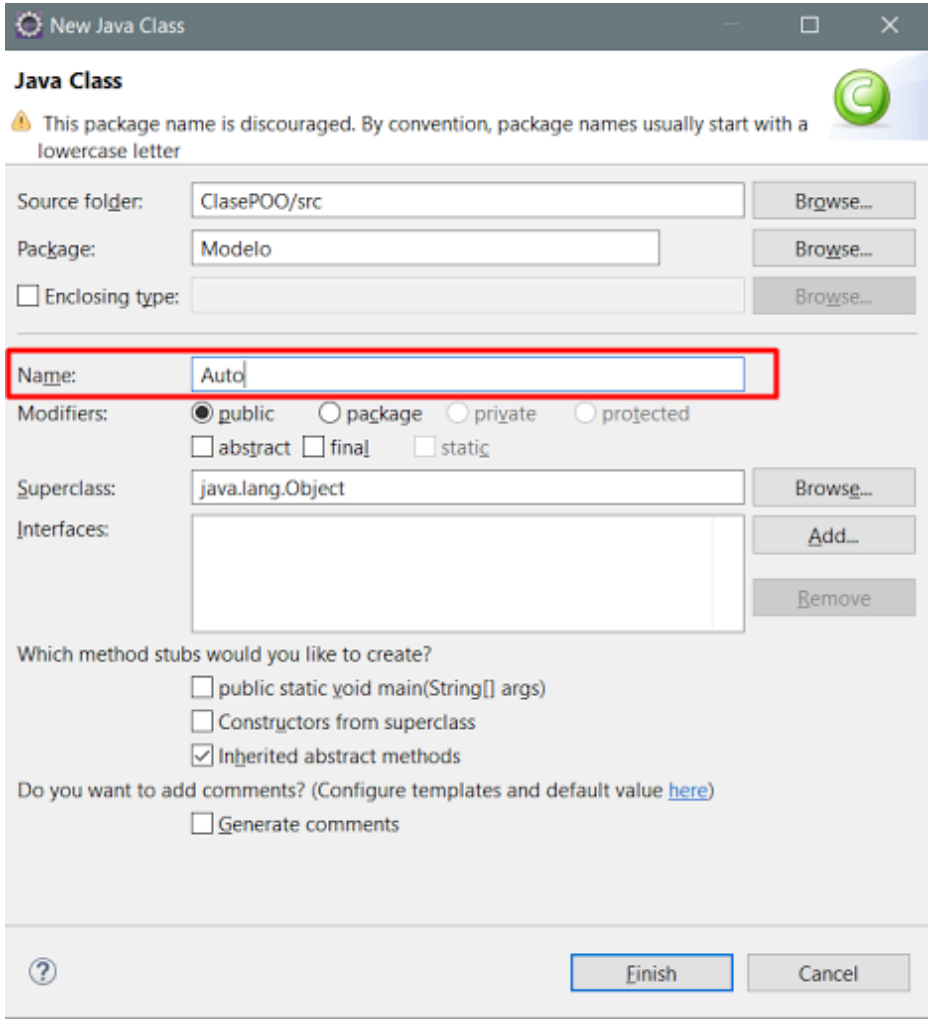



Para crear la clase vamos a la opción:
New > Class




{desafío} latam_

# Creando la clase




{desafío} latam_

# Agregando atributos

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

# Las clases

## *Instancia*

Es una forma de representar una clase "dándole vida" en forma de objeto. Se dice que para “instanciar” una clase, debemos ocupar el operador y la palabra reservada new. Con esta palabra podemos crear una nueva instancia de la clase.

```
Cliente cliente = new Cliente();
```

Cuando se realiza una instancia con el operador new generamos copias de nuestras clases para realizar su ejecución y tratamientos de nuevos valores, pero sin afectar el objeto principal.

# Los métodos

- Porciones de código que realizan una o más operaciones dentro del programa.

- En java, deben estar escritos dentro de una clase, tener un nombre único dentro de esa clase y deben tener la siguiente forma:

```
public void nombreDelMetodo(){
}
```

# Los métodos

## *Creación*

```
public class Auto{
...
        public void encenderMotor(){
                motorEncendido =
true;
        }
}
```

Método **encenderMotor** deja el **motorEncendido** en true.

# Los métodos

## *Creación*

```
public class Auto{
...
          public void aumentarVelocidad(int velocidad){
                    velocidadActual = velocidadActual +
velocidad;
          }
}
```

El método **aumentarVelocidad** recibe una **velocidad** y la agrega a la **velocidadActual**

# Los métodos

## *Creación*

```
public class Auto{
...
          public void frenar(){
                    while(velocidadActual > 0){
                              velocidadActual = velocidadActual-1;
                    };
          }

          public void apagarMotor(){
                    motorEncendido = false;
                    velocidadActual = 0;
          }
}
```

# Las clases

## *Estructura*

Son plantillas en las que nos basamos para crear objetos y para crearlos debemos conocer los siguientes conceptos:

- Palabras reservadas de Java
- Modificador de acceso
- class
- Nombre Clase
- Atributo de instancia
- Atributo local
- Operaciones o Métodos

# Palabras reservadas de Java

Existen palabras únicas del lenguaje que no se pueden utilizar como variables, nombre de métodos o de clases.

A continuación, veremos una tabla con algunas de las más utilizadas en Java:

| | | | | |
|---|---|---|---|---|
| abstract | continue | for | new | switch |
| assert | default | goto | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | thows |
| case | enum | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

- Modificador de acceso

Determina la accesibilidad que tendrán otras clases sobre la clase.
Normalmente, las clases utilizan el término public.

- class

Palabra reservada del lenguaje que determina la definición de una clase.

- Nombre Clase

Identificador que representa el nombre de la clase.

- Atributo de instancia

Cualquier método u operación en la clase puede utilizarlos.

- Atributo local

Se pueden crear dentro de los métodos y serán visibles desde dentro y no fuera de la clase.

- Operaciones o Métodos

Son todas las operaciones especiales de una clase que no se declaran como atributos de la clase. Estos métodos nos permiten que la clase realice acciones propias y existen de 1 a N métodos dentro de una clase. Estos métodos deben ser coherentes con la clase.

Ejemplo de una clase llamada Persona:

```
public class Persona {
private String nombre;
private String rut;
private double altura;
}
```

# /* Constructores y sobrecarga de métodos */

{desafío}
latam_

# Constructores

**Constructores**:

- Permiten crear instancias de nuestras clases en ejecución, incluyendo parámetros de entradas para que cada instancia sea diferente a otra, es decir, cada clase en ejecución tiene los mismos atributos pero con distintos valores.
- Llevan el mismo nombre de la clase y en sus paréntesis se declaran los parámetros de entradas.

**Operador this**: permite hacer referencia a variables o métodos de la clase, este operador solo puede ser ocupado de esta forma, de lo contrario lanzará error de compilación.

# Valores por defecto

| Tipo de dato | Valor |
| --- | --- |
| int | 0 |
| boolean | false |
| String | null |
| Clases | null |

# Constructores

## *Ejemplo*

```
public Persona(String nuevoNombre, String nuevoRut, double nuevaAltura) {
     this.nombre = nuevoNombre;
     this.rut = nuevoRut;
     this.altura = nuevaAltura;
}
```

# Constructores

## *Ejemplo*

Con esto podemos crear una instancia de un objeto dentro o fuera de la clase, como se muestra a continuación con la clase Persona, donde los parámetros que le colocamos al interior del paréntesis son aquellos que declaramos en el constructor previo.

```
public Persona instanciaClase(){
     Persona personita = new Persona ("Juan","1-9",12.2);
     return personita;
}
```

# Constructores

## *Ejemplo*

En este método tenemos dos casos particulares:

- Una variable de tipo local llamada personita que es de tipo Persona.
- Una instancia de clase Persona ocupando el operador new con nuevos valores provenientes del ejemplo constructor.

# Constructores

## *Ejemplo*

También podemos tener dos o más constructores según vayamos necesitando.
Por ejemplo, si necesitamos un constructor que solo reciba el nombre y la altura, pero sin el Rut, tendríamos lo siguiente:

```
public Persona(String nuevoNombre, double nuevaAltura) {
     this.nombre = nuevoNombre;
     this.altura = nuevaAltura;
}
```

# Constructores

## *Ejemplo*

Este mismo constructor lo podemos llamar dentro del primer constructor, esto para realizar sobrecarga de métodos, es decir, generamos un ahorro de lógica y optimizamos recursos en Java.

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

# Modifiers

## *Modificadores de acceso*

Permiten acceder al valor de un atributo (getter = obtención)
Permiten colocar el valor de un atributo (setter = colocar)

*"Permiten definir una propiedad para un atributo determinado que puede iniciarse tanto dentro como fuera de una clase"*

# Modifiers

| Desde | `public` | `default` | `private` |
|---|---|---|---|
| La misma clase | Si | Si | Si |
| Otra clase, mismo paquete | Si | Si | No |
| Clase de otro paquete | Si | No | No |

# Modifiers

## *Getter*

Obtiene el valor de una variable, por lo que si accedemos a este método desde otra función o clase, podremos ver el valor guardado en un objeto.

La función getter es típicamente pequeña y simple, ya que retorna el dato guardado en un objeto.

Por ejemplo, podemos ver a continuación cómo se inicia una variable con getter.

```
public class RazaPerro{
     private String raza;
     public String getRaza(){
          return raza;
     }
}
```

# Modifiers

## *Setter*

Por su parte, el método setter permite cambiar el valor guardado de una variable de un objeto, es decir, el método permite actualizar el valor de una variable.

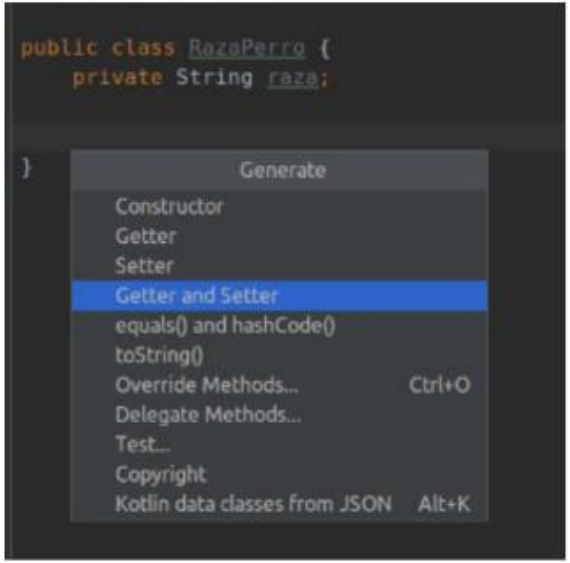
A continuación veremos un ejemplo de cómo sería iniciar y “setear” esa variable.

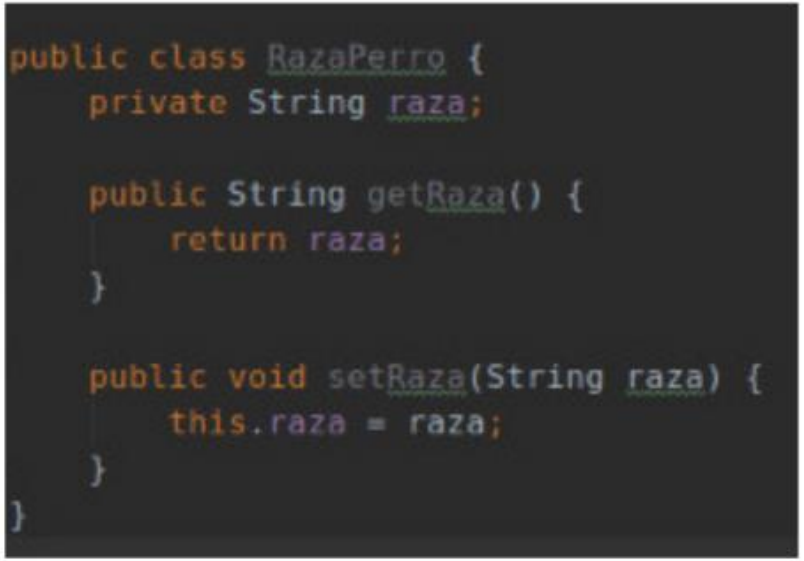
```
public class RazaPerro{
     private String raza;
     public void setRaza(String raza){
          this.raza = raza;
     }
}
```

# Modifiers

## *Getter y Setter*

Para generar ambos métodos, podemos inicializar las variables en la clase, hacer clic secundario al interior de la clase y seleccionar la opción “getters and setters” para que genere automáticamente los modificadores de acceso en esa clase con esos parámetros.




{desafío} latam_

# Ejercicio guiado

# Auto

## *PASO 1*

Crear una clase llamada “Auto” dentro de un package cualquiera de un nuevo proyecto.

```
Click secundario sobre el package -> new -> class
```

# Auto

## *PASO 2*

Declaramos las siguientes variables o características en la clase Auto.

- altura: double
- ancho: double
- tipo material: String
- color: String

```
public class Auto {
      private double altura;
      private double ancho;
      private String tipoMaterial;
      private String color;
}
```

# Auto

## *PASO 3*

Generar el constructor para los parámetros de altura y tipo material. Para esto, podemos hacer clic secundario sobre la clase auto y buscamos la opción “Source” -> “Generate Constructor using fields” y seleccionamos las variables.

Nos debería quedar algo como esto:

```
public class Auto {
      private double altura ;
      private double ancho ;
      private String tipoMaterial;
      private String color;
      public Auto (double altura, String tipoMaterial) {
            this.altura = altura;
            this.tipoMaterial = tipoMaterial;
      }
}
```

# Auto

## *PASO 4*

Crear los “getter and setter” haciendo clic secundario sobre la clase Auto nuevamente y buscamos la opción “Source” -> “Generate Getters and Setters” y seleccionamos las variables.

# Auto

¡Compartamos el resultado!

# ¿Para qué nos sirve un modificador de acceso en la declaración de una Clase?

{desafío} latam_

# ¿En qué situación aplicamos la palabra reservada “`this`”?

{desafío} latam_

# Próxima 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.*

{desafío}
latam_

Academia de
talentos digitales