# Orientación a Objetos I (Parte I)

## Objeto - Clases - Sobrecarga

### Competencias

- 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.

### Introducción

En este capítulo veremos un concepto clave en la programación y el porqué es uno de los más utilizados en cualquier lenguaje. Su nombre es “Programación Orientada a Objetos” o “P.O.O” por sus iniciales.

Entenderemos el porqué y cómo se llegó a este paradigma y porqué es tan popular en diversos ámbitos del desarrollo. Esto nos permitirá observar el comportamiento de los objetos en Java y para esto utilizaremos nuevamente el IDE de Eclipse. Aplicaremos lo aprendido en unidades anteriores para consolidar y construir nuestras primeras clases con las buenas prácticas de Java.

## Los objetos

Cuando hablamos de objetos es necesario entender que nos estamos refiriendo a todo tipo de objetos que podamos ver incluso en el mundo real, los cuales tienen atributos y comportamientos.

El ejemplo más usado es el de un automóvil. Cuando hablamos de un Automóvil, podemos ver atributos o propiedades tales como color, marca, modelo y todas esas características del objeto que lo hacen mantener un estado único que lo diferencia de otros objetos. Además de las propiedades, mencionamos que los objetos tienen un comportamiento, con esto nos referimos a lo que el objeto puede hacer, como Avanzar, Retroceder, Encenderse, entre otras.

Si analizamos estas funcionalidades sin mucho detalle, comprendemos que 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.

Esto mismo pasa en la Programación Orientada a Objetos, ya que 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.

# Las clases

## Instancia de una clase

Cuando tenemos un auto en especifico es porque ya pasamos por una fase de construcción del objeto, ya podemos decir que el auto tiene color, marca y atributos definidos, en programación a esto se le llama estado o instancia de una clase.

Podemos decir entonces que una 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.

## Estructura de una clase

Dentro del mundo tenemos muchos tipos de autos que tienen las mismas funcionalidades (Avanzar, Retroceder, Frenar, etc), pero con diferentes características, es decir, podemos tener autos de diferentes marcas, modelos, colores.

Entonces, podríamos decir que las clases son plantillas en las que nos basamos para crear objetos y que para crearlos debemos conocer los siguientes conceptos:

- **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:

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

Imagen 1. Palabras reservadas de JAVA.
Fuente: Desafío Latam

- **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**: Dentro de una clase se definen atributos de tipo clase, esto quiere decir que cualquier método u operación en la clase puede utilizarlos.

- **Atributo local:** Estas variables se pueden crear dentro de los métodos y solo 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 existe 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

En Java existe un operador llamado `this`, el cual nos 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.

Los constructores nos permiten crear instancias de nuestras clases en ejecución, estos siempre llevan parámetros de entradas ya que cada instancia es diferente a otra, es decir, cada clase en ejecución tiene los mismos atributos pero con distintos valores. Además los constructores:

- Llevan el mismo nombre de la clase y en sus paréntesis se declaran los parámetros de entradas.
- El operador `this` hace referencia a las variables de clases y se le asigna lo que viene como parámetro.

Un ejemplo de constructor es el que se puede ver a continuación:

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

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;
    }
```

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.

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;
   }
```

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;
   }
```

## 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
```

**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;
}
```

**Paso 3**: Posterior a la última declaración de variable que es color, generamos 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;
      }
}
```

**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. 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;
      }
      public double getAltura() {
            return altura;
      }

      public void setAltura(double altura) {
            this.altura = altura;
      }

      public double getAncho() {
            return ancho;
      }

      public void setAncho(double ancho) {
            this.ancho = ancho;
      }

      public String getTipoMaterial() {
            return tipoMaterial;
      }

      public void setTipoMaterial(String tipoMaterial) {
            this.tipoMaterial = tipoMaterial;
      }

}
```

## Ejercicio Propuesto (1): Creación de clase Animal

1.- Crear una clase Animal con las siguientes características:

- nombre: String
- raza: String
- color: String

2.- Crear el constructor con todos los parámetros.
3.- Generar los “getters and setters” para cada Atributo.

# Colaboración y Herencia

## Competencias

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

## Introducción

A continuación conoceremos los conceptos de Herencia y Colaboración que son los pilares fundamentales de la POO (Programación Orientada a Objetos). Estos conceptos ya los hemos estudiado en capítulos anteriores, sin embargo, ahora conoceremos la codificación detrás, realizando clases robustas donde la colaboración será de forma directa e indirecta. Profundizaremos en la sobrecarga de método utilizando herencia y sus palabras reservadas en codificación Java.

## Colaboración - Declaración de objetos dentro de otro Objeto

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, a esto se le llama atributos de tipo Objetos locales.

A continuación, se mostrará estos dos tipos de declaración:

```
public class Persona {

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

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;

    }
```

Ahora tenemos colaboración directa entre la clase `Persona` y `Lapiz`, cabe destacar que no utilizamos la palabra `new` para crear un `Lapiz`, Java interpreta que el objeto Lapiz, que viene por parámetro, ya se instanció en algún lugar de la **Java Virtual Machine** por ende ese `new` viene en el parámetro de entrada.

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;
    }
```

- Aquí tenemos un método el cual recibe y retorna un Objeto `Cuaderno`.
- El Objeto `Cuaderno` nace y muere en ese método.
- El parámetro `cuadernito` no es atributo de la clase y forma parte de atributos locales o específicos.

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 - super

No es nada más que crear clases que heredan atributos y métodos desde otra. A las clases que heredan elementos desde 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.

Esta palabra va al lado del nombre de clase y antes de los corchetes de apertura de clase:

```
public class Programador extends Persona {

    private  String lenguaje;
}
```

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;

    }
```

Los atributos `nombre`, `rut`, `altura` y `lapiz` son de la clase padre `Persona` y solo el atributo `lenguaje` es de su hija. Sin embargo, Java lo interpreta como una sola clase `Programador`. De esta forma la clase hija puede utilizar métodos u operaciones de la clase padre.

## 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.

```
public class Botella {

      private String tipoBotella;
      private String marca;

      public Botella(String tipoBotella, String marca) {
            super();
            this.tipoBotella = tipoBotella;
            this.marca = marca;
      }

      public String getTipoBotella() {
```

```
            return tipoBotella;
      }

      public void setTipoBotella(String tipoBotella) {
            this.tipoBotella = tipoBotella;
      }

      public String getMarca() {
            return marca;
      }

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

}
```

**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`.

```
public class Cerveza extends Botella {
      private int precio;

      public Cerveza(String tipoBotella, String marca, int precio) {
            super(tipoBotella, marca);
            this.precio = precio;
      }

      public int getPrecio() {
            return precio;
      }

      public void setPrecio(int precio) {
            this.precio = precio;
      }
}
```

**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`.

```
public class Botilleria {

	private Cerveza cerveza;
	private String nombre;

	public Botilleria(Cerveza cerveza, String nombre) {
		super();
		this.cerveza = cerveza;
		this.nombre = nombre;
	}

	public Cerveza getCerveza() {
		return cerveza;
	}

	public void setCerveza(Cerveza cerveza) {
		this.cerveza = cerveza;
	}

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
}
```

## Ejercicio Propuesto (2): Colaboración - Herencia

Para este ejercicio utilizaremos el ejercicio propuesto (1) de los animales.

1.- Agregar las siguientes clases con sus respectivos atributos:

a) Ave es un Animal:
   - color: String

b) Oso es un Animal:
   - tipo: String

c) Zoológico:
   - ave : Ave
   - oso: Oso
   - díasAbierto: int
   - dirección: String
   - Función llamada: agregarDías(), la cual retorna los días abiertos.

2.- Agregar nuevos días a "días abiertos", para esto se necesita validar que la cantidad de días abiertos no supere los 7 y sea mayor a cero.

3.- Para cada clase crear:
- Constructor con parámetros.
- Getters and setters.
- Realizar las relaciones necesarias.

## Resumen

Como hemos visto en esta unidad, 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.

Solución ejercicio propuesto (1)

```
public class Animal {
	private String nombre;
	private String raza;
	private String color;

	public Animal(String nombre, String raza, String color) {
		super();
		this.nombre = nombre;
		this.raza = raza;
		this.color = color;
	}

	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getRaza() {
		return raza;
	}
	public void setRaza(String raza) {
		this.raza = raza;
	}
	public String getColor() {
		return color;
	}
	public void setColor(String color){
	     this.color = color;
	}
}
```

## Solución ejercicio propuesto (2)

Clase Ave:

```
public class Ave extends Animal {

	private String tipo;

	public Ave(String nombre, String raza, String tipo) {
		super(nombre, raza);
		this.tipo = tipo;
	}

	public String getTipo() {
		return tipo;
	}

	public void setTipo(String tipo) {
		this.tipo = tipo;
	}
```

Clase Oso:

```
public class Oso extends Animal {

	private String tipo;

	public Oso(String nombre, String raza, String tipo) {
		super(nombre, raza);
		this.tipo = tipo;
	}

	public String getTipo() {
		return tipo;
	}

	public void setTipo(String tipo) {
		this.tipo = tipo;
	}
}
```

Clase Zoológico:

```
public class Zoologico {

	private Ave ave;
	private Oso oso;
	private int diasAbierto;
	private String direccion;
	public Zoologico(Ave ave, Oso oso, int diasAbierto, String
direccion) {
		super();
		this.ave = ave;
		this.oso = oso;
		this.diasAbierto = diasAbierto;
		this.direccion = direccion;
	}
	public Ave getAve() {
		return ave;
	}
	public void setAve(Ave ave) {
		this.ave = ave;
	}
	public Oso getOso() {
		return oso;
	}
	public void setOso(Oso oso) {
		this.oso = oso;
	}
	public int getDiasAbierto() {
		return diasAbierto;
	}
	public void setDiasAbierto(int diasAberto) {
		this.diasAbierto = diasAberto;
	}
	public String getDireccion() {
		return direccion;
	}
	public void setDireccion(String direccion) {
		this.direccion = direccion;
	}

	public int agregarDias(int dias) {

		if(getDiasAbierto()>=0 && getDiasAbierto()<=7 ) {
			this.diasAbierto = dias;
```

```
            }

            return this.diasAbierto;
      }

}
```