



CICLO 2

[FORMACIÓN POR CICLOS]

Fundamentos de la Programación **ORIENTADA A OBJETOS EN JAVA**



Ingeni@
Soluciones TIC



UNIVERSIDAD
DE ANTIOQUIA
Facultad de Ingeniería

En este punto ya conocemos de manera superficial el significado de los diferentes conceptos que rodean la programación orientada a objetos (POO). A continuación, vamos a hacer una introducción a cómo se implementa este paradigma en el lenguaje de programación Java, dejando para un estudio posterior los conceptos más avanzados.

Java es un lenguaje multiparadigma, es decir, usándolo podemos resolver un mismo problema con diferentes acercamientos. Sin embargo, el paradigma dominante en el lenguaje es la programación orientada a objetos que, como sabemos, consiste en organizar los programas como colecciones de objetos que colaboran entre sí. Es por esto que, en Java, cada línea de código que escribamos hace parte de una clase. Un ejemplo de esto es el programa más sencillo que se puede hacer en un lenguaje de programación, en este caso el "¡Hola, Mundo!", que se muestra a continuación.

```
package co.edu.udea.udea_ruta2_ciclo2;

public class ClasePrincipal {
    public static void main (String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}
```

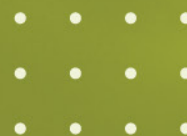
Como se puede observar, para este sencillo programa se define una clase llamada `ClasePrincipal`. Una clase, generalmente, se debe guardar en un archivo fuente de Java del mismo nombre (en este caso `ClasePrincipal.java`). Y además, la primera línea del archivo de la clase debe indicar el paquete en el cual se sitúa esta, en este caso el paquete `co.edu.udea.udea_ruta2_ciclo2`.

La clase anterior es un tipo de clase especial. En una aplicación de consola (es decir, que se ejecuta en la línea de comandos o consola), es necesario que exista una clase principal, aunque no es necesario que se llame así. La clase principal será aquella que cuente con un método público, estático, de tipo `void`, llamado `main`, y que reciba como parámetros un array de `Strings`. En resumen, este:

```
public static void main (String[] args) {
    //Instrucciones...
}
```

Ver:

https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/main/java/co/edu/udea/udea_ruta2_ciclo2/ClasePrincipal.java



Esto quiere decir, que al ejecutar el programa que estamos escribiendo, lo primero que se ejecutará son las instrucciones que se encuentren dentro del método `main` de la clase principal. Y desde ahí, podremos interactuar con otros objetos o llamar otros métodos. Excepcionalmente, cuando un programa tiene varias clases principales, es necesario seleccionar cuál es la clase principal a utilizar; la forma de hacerlo depende de cada IDE.

Elementos de la Programación Orientada a Objetos en Java

Clases

En este punto cabe señalar que una aplicación desarrollada en Java no se limita a la clase principal. Como hemos señalado antes, una aplicación orientada a objetos se compone, como su nombre lo indica, de diferentes objetos interactuando y colaborando entre sí. Para esto, es necesario que existan las clases. Como sabemos, una **clase** sirve de plantilla para la creación posterior de objetos que puedan ser utilizados en la aplicación. Para esto, se inicia por declarar la clase usando la palabra clave `class`, y agregando de manera opcional antes el modificador de acceso `public`. El cuerpo de la clase irá entre llaves, y esta deberá ir en un archivo con su mismo nombre y la extensión `.java`. También, se deberá especificar en la primera línea del archivo, el paquete en el que se sitúa la clase. A continuación, vemos cómo sería esto para la clase *Bicicleta*.

```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class Bicicleta {

}
```


Y a continuación, vamos cómo sería esto para una clase *Estudiante*.

```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class Estudiante {

}
```





Por convención y buenas prácticas, en Java las clases se nombran empezando con mayúscula y con el resto de las letras en minúscula; y si el nombre de la clase incluye varias palabras, estas deberán ir juntas, empezando cada palabra con mayúscula y el resto de letras en minúscula.

Atributos

En una lección anterior, exploramos las variables y las usamos como variables locales. Y es que las variables en Java se pueden usar de tres formas: como variables locales (dentro de un método), como parámetros (para pasar datos a un método al llamarlo), y como atributos. Los **atributos** son la forma que tenemos de implementar las propiedades que compartirán todos los objetos de dicha clase, independientemente de si el valor de cada atributo entre objetos es el mismo o no.

En Java, los atributos se declaran usando un modificador de acceso (`public`, `protected`, `private` o ninguno), el tipo (que puede ser un tipo de datos primitivo, el tipo `String` u otra clase), y el nombre, seguidos de punto y coma. Para el caso de la clase *Bicicleta* declararemos algunos atributos.

```
package co.edu.udea.udea_ruta2_ciclo2.poo;
```

```
public class Bicicleta {  
    private String marca;  
    private String color;  
    private double velocidad;  
}
```

Y de igual manera, lo haremos con la clase *Estudiante*.

```
package co.edu.udea.udea_ruta2_ciclo2.poo;
```

```
public class Estudiante {  
    private String nombres;  
    private String apellidos;  
    private int edad;  
    private double promedio;  
}
```



En este punto vale la pena hacer un alto en el camino y hablar de los modificadores de acceso. El modificador de acceso que se le pone a atributos, métodos y constructores, determina la **visibilidad** del atributo, método o constructor. Es decir, desde dónde es posible obtener o cambiar el valor del atributo, o bien llamar al método. Esto puede variar dependiendo de lo que queramos y de prácticas como el encapsulamiento (que veremos posteriormente). Los modificadores de acceso permiten definir, por ejemplo, que un atributo sólo se pueda modificar o acceder desde dentro de la clase donde se define, o bien, que un método se pueda llamar desde cualquier parte. En la siguiente tabla se listan los modificadores de acceso en Java, y qué nivel de visibilidad permite cada uno:

Visibilidad	Niveles de Acceso				
	Misma clase	Mismo paquete	Subclase en el mismo paquete	Subclase fuera del paquete	Cualquier otra clase por fuera del paquete
public	Si	Si	Si	Si	Si
protected	Si	Si	Si	Si	No
[ninguno]	Si	Si	Si	No	No
private	Si	No	No	No	No

En los ejemplos anteriores, todos los atributos se declararon como `private`, eso quiere decir que sólo serán accesibles directamente desde la misma clase. Además, los atributos, tal como los demás tipos de variables, se deben nombrar usando la notación *camel-case*. Es decir, la primera palabra totalmente en minúsculas, y si el nombre de la variable incluye varias palabras, estas deberán ir juntas, empezando con una letra mayúscula a partir de la segunda palabra (por ejemplo, `volumen`, `colorCubierta`, `platoDelanteroActual`).





Métodos

Una vez se definen los atributos de una clase, tenemos los métodos. Los **métodos** es la forma en la que implementamos en Java las operaciones que definen el comportamiento de los objetos de la clase. Comúnmente, los objetos interactúan entre sí (o nosotros interactuamos con los objetos) llamando sus métodos. Un método se define, en principio, escribiendo:

- Un modificador de acceso (siguiendo las mismas reglas de los modificadores de acceso de los atributos).
- El tipo de datos que devuelve, que puede ser un tipo de dato primitivo, el tipo String, otra clase o void, si no va a devolver un valor.
- El nombre del método, para el cual se usa también la notación *camel-case*. Además, se recomienda que los nombres de los métodos inicien con un verbo. Después de todo, un método es una acción.
- Entre paréntesis, cero, uno o varios parámetros, de ser necesarios, separados por coma (,). Los parámetros se definen con su tipo de datos y su nombre.
- Y finalmente, el cuerpo del método. Es decir, lo que este realizará, encerrado entre llaves ({}).

En el ejemplo a continuación, agregamos a la clase *Bicicleta* algunos métodos.



```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class Bicicleta {
    private String marca;
    private String color;
    private double velocidad;

    public String getMarca() {
        return marca;
    }

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

    public String getColor() {
        return color;
    }


    public void setColor(String color) {
        this.color = color;
    }

    public double getVelocidad() {
        return velocidad;
    }

    public void pedalear(double aceleracion) {
        this.velocidad = this.velocidad + aceleracion;
    }

    public void frenar() {
        if (this.velocidad > 0) {
            this.velocidad--;
        }
    }
}
```





En el ejemplo anterior podemos notar varias cosas:

- Existen dos métodos tipo `void`. Estos métodos no devolverán ningún valor y se limitan a aumentar o disminuir la velocidad de la bicicleta.
- Aunque los atributos son privados, y no se pueden acceder desde objetos ajenos a esta clase directamente, los métodos son públicos, permitiéndonos modificar o acceder el valor de los atributos, si bien de forma indirecta.
- Se recomienda que los métodos empiecen con un verbo. En este caso, la mayor parte de ellos empiezan con uno de dos verbos (que no son obligatorios):
 - o `get`: A los métodos cuyo nombre inicia con este verbo se les suele denominar *getters*, y nos permiten obtener el valor de un atributo del objeto.
 - o `set`: A los métodos cuyo nombre inicia con este verbo se les suele denominar *setters*, y nos permiten modificar el valor de un atributo del objeto.
- En los métodos que queremos devolver un valor (y siempre que tenga un tipo de retorno deberá ser así), dicho valor deberá ser retornado usando la palabra clave `return`, seguida de lo que vamos a devolver y punto y coma. La palabra clave `return` también indica que en esa instrucción finaliza la ejecución del método.
- En los métodos encontramos la palabra clave `this`. Esta palabra clave es similar a la palabra clave `self` en Python, y permite referirse al objeto actual. Es decir que, si desde un método se especifica `this`, seguido de un atributo o un método de dicha clase, se está refiriendo al atributo o método de dicho nombre del objeto actual.

En el siguiente ejemplo, agregamos a la clase *Estudiante* algunos métodos.




```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class Estudiante {
    private String nombres;
    private String apellidos;
    private int edad;
    private double promedio;

    public String getNombres() {
        return nombres;
    }

    public void setNombres(String nombres) {
        this.nombres = nombres;
    }

    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }


    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public double getPromedio() {
        return promedio;
    }

    public void setPromedio(double promedio) {
        this.promedio = promedio;
    }
}
```





Como hemos podido observar, en la clase *Estudiante* se siguieron muchas de las convenciones y reglas seguidas en la clase *Bicicleta*.

Algunos de los métodos de las dos clases anteriores cuentan con parámetros. Existen algunos aspectos adicionales a tener en cuenta:

- Los parámetros también podrán ser también, además de variables u objetos, *arrays*.
- Un parámetro no podrá tener el mismo nombre de otro parámetro del mismo método, ni de una variable local dentro del método.

Constructores

Otro elemento importante de una clase es su constructor o constructores. Los **constructores** se definen para dar opciones a quien cree un objeto de la clase, para que los atributos tengan valores iniciales, es decir, para que se inicialicen. Al momento de escribir una clase, podemos definir uno o varios constructores (como mínimo un constructor sin parámetros), y luego, a la hora de crear (instanciar) un objeto, quien lo haga llamara a uno de dichos constructores. Un constructor se define de manera similar a un método, con:

- El modificador de acceso `public`.
- El mismo nombre de la clase.
- La lista de cero, uno o varios parámetros encerrados entre paréntesis.
- El cuerpo del constructor encerrado entre llaves `{ }`.

Existe la posibilidad de escribir una clase sin constructores. En ese caso, el compilador de Java asigna uno por defecto. Sin embargo, en la mayoría de los casos esto no es recomendable. En el siguiente ejemplo, creamos dos constructores para la clase *Bicicleta*.



```
package co.edu.udea.udea_ruta2_ciclo2.poo;
```

```
public class Bicicleta {  
    private String marca;  
    private String color;  
    private double velocidad;
```

```
    public Bicicleta() {  
        this.marca = "GW";  
        this.color = "gris";  
        this.velocidad = 0;  
    }
```

```
    public Bicicleta(String marca, String color, double  
velocidadInicial) {  
        this.marca = marca;  
        this.color = color;  
        velocidad = velocidadInicial;  
    }
```

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

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

```
    public String getColor() {  
        return color;  
    }
```

```
    public void setColor(String color) {  
        this.color = color;  
    }
```

```
    public double getVelocidad() {  
        return velocidad;  
    }
```

```
    public void pedalear(double aceleracion) {  
        this.velocidad = this.velocidad + aceleracion;  
    }
```

```
    public void frenar() {  
        if (this.velocidad > 0) {  
            this.velocidad--;  
        }  
    }  
}
```

Y en el siguiente ejemplo, hacemos lo propio con la clase *Estudiante*. Notemos que se acostumbra situar los constructores entre los atributos y los métodos. Notemos, también, que cuando un parámetro de un método o de un constructor, y un atributo de la clase tienen el mismo nombre, es posible distinguirlos con la palabra clave `this`. Siempre podremos referirnos al atributo usando la palabra clave `this`, seguida de un punto y del nombre del atributo.

```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class Estudiante {
    private String nombres;
    private String apellidos;
    private int edad;
    private double promedio;

    public Estudiante() {
        this.nombres = "";
        this.apellidos = "";
        this.edad = 0;
        this.promedio = 0.0;
    }

    public Estudiante(String nombres, String apellidos, int
edad, double promedio) {
        this.nombres = nombres;
        this.apellidos = apellidos;
        this.edad = edad;
        this.promedio = promedio;
    }

    public String getNombres() {
        return nombres;
    }

    public void setNombres(String nombres) {
        this.nombres = nombres;
    }

    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
}
```



```
public int getEdad() {  
    return edad;  
}  
  
public void setEdad(int edad) {  
    this.edad = edad;  
}  
  
public double getPromedio() {  
    return promedio;  
}  
  
public void setPromedio(double promedio) {  
    this.promedio = promedio;  
}  
}
```

Sobrecarga de métodos

Si observamos bien los dos ejemplos anteriores, podremos notar que para cada clase definimos dos constructores, obviamente con el mismo nombre. Del mismo modo, podríamos definir dos métodos con el mismo nombre, siempre que estos se diferencien en el número y/o el tipo de sus parámetros.

A esto se le denomina, sobrecarga de métodos. A continuación, vemos un ejemplo de lo que podríamos hacer, de ser necesario, con el método `frenar` de la clase *Bicicleta*.



```
package co.edu.udea.udea_ruta2_ciclo2.poo;

public class Bicicleta {

    //...

    public void frenar() {
        if (this.velocidad > 0) {
            this.velocidad--;
        }
    }


    public void frenar(double aceleracionNegativa) {
        if (this.velocidad > aceleracionNegativa) {
            this.velocidad = this.velocidad -
            aceleracionNegativa;
        }
    }

    public void frenar(String intensidad) {
        if (intensidad.equals("fuerte")) {
            if (this.velocidad > 10) {
                this.velocidad = this.velocidad - 10;
            }
        } else {
            if (this.velocidad > 0) {
                this.velocidad--;
            }
        }
    }
}
```

Paquetes

Para terminar con los conceptos básicos de la programación orientada a objetos en Java, no podemos olvidarnos de los paquetes. Hemos observado que para cada clase es necesario especificar en qué paquete está situada.





Un **paquete** es algo similar a una carpeta, donde se pueden agrupar clases, interfaces y otros tipos relacionados. Guiados por los nombres de los paquetes, otros desarrolladores pueden encontrar fácilmente clases relacionadas dentro de una aplicación. El uso de paquetes tiene, además de la ventaja de poder agrupar clases, permitir que dos clases tengan el mismo nombre siempre que se encuentren en paquetes diferentes. Además, los paquetes permiten también permitir y restringir el acceso a clases, métodos y atributos desde otras clases, vía los modificadores de acceso (como vimos en una tabla más arriba).

Para especificar que una clase se encuentra en un paquete, se usa la instrucción `package`, siempre en la primera línea del archivo donde se escribe la clase, tal como lo vemos en los ejemplos de arriba. Además, el archivo `.java` de la clase deberá estar dentro de una carpeta que refleje el nombre del paquete, y esta podrá estar a su vez dentro de otras carpetas que cumplan la misma característica. Esta última tarea la facilitan enormemente la mayor parte de los IDEs.

Finalmente, es necesario tener en cuenta que para usar una clase de un paquete diferente al de la clase actual, será necesario *importarla*, mediante la instrucción `import`, tal como lo veremos posteriormente.


Creación y uso de objetos a partir de clases existentes

Una vez se han creado las clases, podremos crear **objetos** a partir de estas. Si en una clase definimos los atributos que puede tener un objeto y los métodos mediante los cuales podremos interactuar con él. Al crear el objeto estamos pasando de esa plantilla abstracta a algo más concreto e individual. Los objetos tendrán entonces un comportamiento (mediante los métodos previamente definidos) y un estado (que comprende los valores de sus atributos en un momento dado).

Empecemos entonces con la creación de objetos. La creación de un objeto comprende tres momentos:

1. Declaración
2. Instanciación
3. Inicialización





Para **declarar** un objeto, lo hacemos de la misma manera en la que se declara una variable, esto es, se escribe el nombre de la clase, luego el nombre del objeto y luego punto y coma. Las convenciones para nombrar objetos son las mismas recomendadas para nombrar variables. Abajo declaramos dos objetos, uno de la clase *Bicicleta*, llamado `miBicicleta`, y otro de la clase *Estudiante*, llamado `inscrito`.

```
Bicicleta miBicicleta;  
Estudiante inscrito;
```

Se puede declarar un objeto, y dejar el siguiente momento, el de la instanciación, para después. Sin embargo, no podremos acceder a sus métodos ni a sus atributos, ya que es en el momento de la **instanciación** cuando se crea realmente el objeto. A continuación, se instancian los dos objetos declarados previamente.

```
miBicicleta = new Bicicleta();  
inscrito = new Estudiante("Pepito", "Pérez", 17, 3.5);
```

Para la instanciación, se usa la palabra clave `new`, llamando después uno de los constructores existentes en la clase declarada, y terminando con punto y coma. Si nos fijamos bien, en las dos líneas anteriores estamos realmente llamando los constructores que definimos junto a las clases *Bicicleta* y *Estudiante* en ejemplos anteriores. Al llamar dichos constructores, estos se ejecutan, pasando al momento de la **inicialización**. También es posible hacer los dos pasos anteriores, la declaración y la instanciación, al mismo tiempo.

```
Bicicleta miBicicleta = new Bicicleta();  
Estudiante inscrito = new Estudiante("Pepito", "Pérez", 17, 3.5);
```

Una vez hemos instanciado los objetos, es posible llamar a sus métodos públicos y acceder a sus atributos públicos de manera directa. A los atributos y métodos de un objeto se le llaman de forma genérica **miembros**. Para acceder a un miembro de un objeto, basta con escribir el nombre del objeto, seguido de punto, luego seguido del nombre del miembro, y finalmente pasando los valores que sea necesario al método (si cuenta con parámetros). Un atributo, o algo devuelto por un método, se podrán imprimir, sumar, concatenar o lo que sea que permita su tipo de datos. Todo esto se puede apreciar en el ejemplo completo a continuación².

²Los ejemplos anteriores se pueden ver acá: https://github.com/leonjara-millo/udea_ruta2_ciclo2/tree/main/main/java/co/edu/udea/udea_ruta2_ciclo2/poo




```

package co.edu.udea.udea_ruta2_ciclo2.poo;

public class PruebaObjetos {

    public static void main(String[] args) {
        Bicicleta miBicicleta = new Bicicleta();
        Estudiante inscrito = new Estudiante("Pepito", "Pérez",
17, 3.5);

        System.out.println(miBicicleta.getMarca());
        miBicicleta.setMarca("Specialized");
        System.out.println("La marca de mi bicicleta es " +
miBicicleta.getMarca());
        System.out.println("Velocidad inicial: " + miBicicleta.
getVelocidad());
        miBicicleta.pedalear(5);
        miBicicleta.pedalear(2);
        System.out.println("Velocidad actual: " + miBicicleta.
getVelocidad());
        miBicicleta.frenar(6);
        System.out.println("Velocidad actual: " + miBicicleta.
getVelocidad());
        System.out.println("Me llamo " + inscrito.getNombres()
+ " " + inscrito.getApellidos());
        System.out.println(inscrito.getEdad());
        System.out.println("Promedio: " + inscrito.
getPromedio());
        inscrito.setPromedio(4.7);
        System.out.println("Nuevo promedio: " + inscrito.
getPromedio());
        System.out.println("Operación loca... velocidad
más promedio: " + (miBicicleta.getVelocidad() + inscrito.
getPromedio()));
    }
}

```

Lo anterior, una vez se ejecuta, imprime:

```

GW
La marca de mi bicicleta es Specialized
Velocidad inicial: 0.0
Velocidad actual: 7.0
Velocidad actual: 1.0
Me llamo Pepito Pérez
17
Promedio: 3.5
Nuevo promedio: 4.7
Operación loca... velocidad más promedio: 5.7

```