

Curso de Java Standard



Ing. Octavio Robleto



octavio.robleto@gmail.com



<https://octaviorobleto.com>



Introducción

Muchas veces nos encontramos con la necesidad de definir un tipo de dato en un objeto que debería estar delimitado por una lista de constantes.

Por ejemplo al asignarle un color a un Automóvil en la mayoría de los casos asignaremos como atributo del **Auto** un **String**:

```
public abstract class Auto implements MantenimientoMecanico, Archivo {  
    private String color;  
    private String marca;  
    private Patente patente;  
    public static String concesionaria = "Autos EducacionIT";  
    private Integer puestos;
```

Nos podemos encontrar con el problema que coloquen un color que no este definido, aceptado o simplemente se comentan errores al momento de asignarlo.

```
autoFamiliar1.setColor("Asul");  
autoFamiliar1.setColor(Constants.R0J0);
```

Enumerados

Este lo podemos lograr con los objetos Enum en Java, aunque se puede definir de forma simple que los Enumerados son una lista de constantes, el concepto es un poco mas complejo y completo.

Es una clase especial que se define de forma particular; en vez de la palabra reservada **class** se debe colocar la palabra reservada **enum**, esto le indica al Java que esta clase hereda de **Enum** del paquete `java.Lang` y como en todas las clases en Java se heredan los atributos y métodos de la clase heredada.

Un objeto de este tipo son **TYPE-SAFE** (de tipo seguro), lo que significa que un enumerado tiene su propio **Namespace** (espacio de nombres), solo puede contener los valores definidos por la lista. Por lo tanto, una enumeración le brinda una manera de definir con precisión un nuevo tipo de dato que tiene un número fijo de valores válidos.

La ventaja es que al indicar que el atributo a asignar es de tipo enumerado el compilador de Java evaluará el valor asignado y solo permitirá de ese tipo.

```
public enum Colores {  
    NEGRO, AZUL, MARRON,  
    GRIS, VERDE, NARANJA,  
    ROSA, PURPURA, ROJO,  
    BLANCO, AMARILLO;  
}
```



Como estándar de nomenclatura para los enumerados se recomienda el **UPPER CASE** como para todo lo que sea constante.

```
autoFamiliar1.setColor(Colores.AZUL);
```

Métodos

Tipo	Método	Descripción
String	toString()	Devuelve una breve descripción del objeto enum.
String	name()	Devuelve el nombre del enumerado.
int	ordinal()	Devuelve la posición del enum según este declarado comenzando desde cero.
Enum[]	values()	Devuelve un arreglo con todos los elementos que se encuentren en el enum.
Enum	valueOf():	Devuelve la constante enum del valor de cadena especificado si existe, de lo contrario mostrara un error.

```
Colores color = Colores.valueOf("NEGRO");  
System.out.println(color.name());  
System.out.println(color.ordinal());  
System.out.println(color.toString());  
System.out.println(Arrays.toString(Colores.values()));
```

Constructor

Aunque ya aclaramos que los **enum** son una clase, debemos indicar que no podemos instanciarla:

~~Colores color = new Colores();~~

Cada elemento del **enum** es un objeto, lo que quiere decir que cada una de esas constantes que añadimos al enumerado al final se esta instanciando de forma implícita dentro de la clase **enum**, si le colocamos los paréntesis del constructor por defecto no presentara ningún error.

Lo que esta ocurriendo es que los enumerados tienen los constructores con el acceso privado, por eso no se pueden instanciar desde otra clase.

```
public enum Colores {  
    NEGRO(), AZUL(), MARRON(),  
    GRIS(), VERDE(), NARANJA(),  
    ROSA(), PURPURA(), ROJO(),  
    BLANCO(), AMARILLO();  
  
    private Colores() {  
  
    }  
}
```



Atributos en los enum

Si quisiéramos podemos darles atributos a nuestros objetos enumerados, pero solo se deben asignar al instanciarlos dentro del **enum**, por lo que deberemos crear un constructor que soporte dichos atributos y crear los métodos **getters** para poder obtenerlos.

Solo puede existir un constructor en el enumerado y cada objeto del **enum** debe poseer siempre la misma cantidad de atributos.

```
Colores color = Colores.valueOf("NEGRO");
System.out.println(color.name());
System.out.println(color.ordinal());
System.out.println(color.toString());
System.out.println(Arrays.toString(Colores.values()));

System.out.println(color.getHEX_CODIGO());
System.out.println(color.getRGB_CODIGO());
```


```
public enum Colores {
    NEGRO("000000", "0,0,0"), AZUL("0000FF", "0,0,255"),
    MARRON("4E3B31", "78,59,49"), GRIS("808080", "128,128,128"),
    VERDE("008000", "0,128,0"), NARANJA("FF6600", "255,102,0"),
    ROSA("FFC0CB", "255,192,203"), PURPURA("800080", "128,0,128"),
    ROJO("FF0000", "255,0,0"), BLANCO("FFFFFF", "255,255,255"),
    AMARILLO("FFFF00", "255,255,0");

    private String HEX_CODIGO;
    private String RGB_CODIGO;

    private Colores(String HEX_CODIGO, String RGB_CODIGO) {
        this.HEX_CODIGO = HEX_CODIGO;
        this.RGB_CODIGO = RGB_CODIGO;
    }

    public String getHEX_CODIGO() {
        return HEX_CODIGO;
    }

    public String getRGB_CODIGO() {
        return RGB_CODIGO;
    }
}
```



Enumerados y los switch

Además de sus múltiples ventajas los enumerados pueden utilizarse literalmente en los **switch** lo que hace que este tipo de clases nos de mas opciones y así evitar menos errores en los casos necesarios.

```
Colores color = Colores.valueOf("NEGRO");
switch (color) {
case ROJO:
case AMARILLO:
case AZUL:
    System.out.println("Es un color primario tradicional RYB");
    break;
case NEGRO:
case MARRON:
case GRIS:
case VERDE:
case NARANJA:
case ROSA:
case PURPURA:
case BLANCO:
    System.out.println("No es un color primario tradicional RYB");
    break;
default:
    break;
}
```