**ENUMERADOS EN JAVA**

Tabla de Contenido:

**1. Enum en Java**

[**2. Cómo funcionan los Enum**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#2_Como_funcionan_los_Enum)

[**3. Declaración de enum en Java**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#3_Declaracion_de_enum_en_Java)

[**4. Ejemplo con enum en Java**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#4_Ejemplo_con_enum_en_Java)

[**4.1. Código de Ejemplo**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#41_Codigo_de_Ejemplo)

[**5. Puntos importantes de enum**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#5_Puntos_importantes_de_enum)

[**6. Métodos values(), ordinal() y valueOf()**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#6_Metodos_values_ordinal_y_valueOf)

[**7. enum: Constructores, métodos, variables de instancia**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#7_enum_Constructores_metodos_variables_de_instancia)

[**7.1. Explicación del Ejemplo**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#71_Explicacion_del_Ejemplo)

[**8. Enum y Herencia**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#8_Enum_y_Herencia)

[**8.1. Uso de ordinal( ) y compareTo()**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#81_Uso_de_ordinal_y_compareTo)

[**9. enum y Métodos**](https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#9_enum_y_Metodos)

## 1. Enum en Java

¿Qué es un ENUM? En su forma más simple, **una enumeración es una lista de constantes con nombre que definen un nuevo tipo de datos**. Un objeto de un tipo de enumeración solo puede contener los valores definidos por la lista. Por lo tanto, una enumeración proporciona una manera de definir con precisión un nuevo tipo de datos que tiene **un número fijo de valores válidos.**

Por ejemplo, los 4 palos en un mazo de cartas pueden ser 4 enumeradores llamados espadas, bastos, oros y copas, que pertenecen a un tipo enumerado llamado Baraja. Otros ejemplos incluyen tipos de enumerados naturales (como los planetas, días de la semana, meses del año, colores, direcciones, etc.).

Desde una perspectiva de programación, las enumeraciones son útiles siempre que necesite definir un **conjunto de valores que represente una colección de elementos.** Por ejemplo, se puede usar una enumeración para representar un conjunto de códigos de estado, como *éxito*, *espera*, *error* y *reintentos*, que indican el progreso de alguna acción.

En el pasado, dichos valores se definían como **variables finales**, pero las enumeraciones ofrecen un enfoque más estructurado. Todos los enum implícitamente extienden de la clase java.lang.Enum.

 public abstract class Enum<E extends Enum<E>>

      extends Object implements Comparable<E>, Serializable

## 2. Cómo funcionan los Enum

**¿Qué es un ENUM en Java?** Los enum en Java se usan cuando conocemos todos los valores posibles en tiempo de compilación, como las opciones en un menú, los modos de redondeo, los indicadores de línea de comando, etc.

En Java (desde 1.5), las enumeraciones se representan utilizando el tipo de datos **enum**. Las enumeraciones Java son más potentes que las enumeraciones C/C++. En Java, también podemos agregarle atributos/variables, métodos y constructores. El objetivo principal de enum es definir nuestros propios tipos de datos (**tipos de datos enumerados**).

## 3. Declaración de enum en Java

La declaración de Enum puede hacerse fuera de**una clase**, o dentro de una clase (class), pero NO dentro de un método.

**EJEMPLO**

// Un simple ejemplo donde se declara enum

// fuera de cualquier clase (Nota la palabra enum en lugar

// de la palabra class)

enum Color

{

ROJO, VERDE, AZUL;

}

public class Test

{

// El método

public static void main(String[] args)

{

Color c1 = Color.ROJO;

System.out.println(c1);

}

}

**Salida:**

**ROJO**

Una enumeración se crea usando la palabra clave **enum**.

La primera línea dentro de enum debe ser una lista de constantes y luego otras cosas como métodos, variables y constructores.

De acuerdo con las convenciones de nomenclatura de Java, se recomienda que nombremos las constante con **mayúsculas**.

## 4. Ejemplo con enum en Java

**EJEMPLO**

Enumeración simple que enumera varias formas de transporte:

//Una enumeración de transporte

enum Transporte{

COCHE, CAMIÓN, AVION, TREN, BARCO;

}

Los identificadores *COCHE*, *CAMION*, etc. se denominan **constantes de enumeración**.

Cada uno se declara **implícitamente** como un miembro público (**public**) y estático (**static**) de *Transporte*. Además, el tipo de las constantes de enumeración es el tipo de enumeración en el que se declaran las constantes, que es *Transporte* en este caso. Por lo tanto, en el lenguaje de Java, estas constantes se llaman **auto-tipado**.

Una vez que hayas definido una enumeración, puedes crear una variable de ese tipo. Sin embargo, aunque las enumeraciones definen un tipo de clase, no crea una instancia de una enumeración usando *new*. En cambio, declaras y utilizas una variable de enumeración de la misma manera que haces uno de los tipos primitivos. Por ejemplo, esto declara *tp* como una variable del tipo de enumeración *Transporte*:

Transporte tp;

Como tp es de tipo Transporte, los únicos valores que se le pueden asignar son los definidos por la enumeración. Por ejemplo, esto asigna tp el valor AVION:

tp = Transporte.AVION;

Se pueden comparar dos constantes de enumeración utilizando el operador relacional **==**. Por ejemplo, esta declaración compara el valor en tp con la constante TREN:

If (tp == Transporte.TREN) // ...

Un valor de enumeración también se puede usar para controlar una sentencia [**switch**](https://javadesdecero.es/basico/sentencia-switch-java-ejemplo/). Por supuesto, todas las declaraciones de **case** deben usar constantes de la misma enumeración que la utilizada por la expresión de switch. Por ejemplo, este *switch* es perfectamente válido:

//Uso de enum para controlar una sentencia switch

switch(tp){

case COCHE:

//

case CAMION:

//

}

Observa que en las sentencias case, los nombres de las constantes de enumeración se usan sin estar calificados por el nombre de tipo de enumeración. Es decir, se utiliza CAMION, no Transporte.CAMION. Esto se debe a que el tipo de enumeración en la expresión de switch ya ha especificado implícitamente el tipo de enumeración de las constantes de case.

No es necesario calificar las constantes en las declaraciones de **case** con su nombre de tipo enum. De hecho, intentar hacerlo provocará un error de compilación.

### 4.1. Código de Ejemplo

Cuando se muestra una constante de enumeración, como en una instrucción println(), se genera su nombre. Por ejemplo, dada esta declaración:

System.out.println(Transporte.BARCO);

Se muestra el nombre BARCO.

El siguiente programa reúne todas las piezas y muestra la enumeración Transporte:

enum Transporte{

COCHE, CAMION, AVION, TREN, BARCO;

}

class Enumerados {

public static void main(String[] args) {

Transporte tp;

tp=Transporte.AVION;

System.out.println("Valor de tp: "+tp);

System.out.println();

tp=Transporte.TREN;

//Comparación de 2 valores enum

if (tp==Transporte.TREN)

System.out.println("tp tiene el valor de TREN\n");

//enum para controlar sentencia switch

switch(tp){

case COCHE:

System.out.println("Un auto lleva personas.");

break;

case CAMION:

System.out.println("Un camión lleva carga.");

break;

case AVION:

System.out.println("Un avión vuela.");

break;

case TREN:

System.out.println("Un tren corre sobre rieles.");

break;

case BARCO:

System.out.println("Un barco navega en el agua.");

break;

}

}

}

Salida:

Valor de tp: AVION

tp tiene el valor de TREN

Un tren corre sobre rieles.

Las constantes en Transporte las pondremos en mayúsculas. (Por lo tanto, se usa COCHE, no coche.) Sin embargo, **no es obligatorio el uso de mayúsculas**. Debido a que las enumeraciones a menudo reemplazan las variables finales, que tradicionalmente se han usado en mayúsculas, se ha llegado al acuerdo de que las **constantes de enumeración las escribamos en mayúsculas**.

## 5. Puntos importantes de enum

Esta explicación puede ser muy útil para que entendáis bien los enumerados:

* Cada enum es implementado **internamente** mediante el uso de *class*.

Internamente enum Color que definimos antes se convierte en:

class Color

{

public static final Color ROJO = new Color();

public static final Color AZUL = new Color();

public static final Color VERDE = new Color();

}

* Cada constante enum representa un objeto de tipo enum.
* El tipo enum se puede pasar como un argumento para *switch*.

Vemos un ejemplo más:

// Un programa Java para demostrar el trabajo de enum

// en case de switch (Archivo Test.Java)

import java.util.Scanner;

// Una clase enum

enum Dia

{

LUNES, MARTES, MIERCOLES, JUEVES,

VIERNES, SABADO, DOMINGO;

}

// Controlador de clase que contiene un objeto de "Dia" y

// main().

public class Test

{

Dia dia;

// Constructor

public Test(Dia dia)

{

this.dia = dia;

}

// Imprime una línea sobre el DIA usando switch

public void diaEs()

{

switch (dia)

{

case LUNES:

System.out.println("Los lunes son feos.");

break;

case VIERNES:

System.out.println("Los viernes son mejores.");

break;

case SABADO:

case DOMINGO:

System.out.println("Los fines de semana son mejores.");

break;

default:

System.out.println("Los días entre semana son regulares.");

break;

}

}

// Metodo

public static void main(String[] args)

{

String str = "LUNES";

Test t1 = new Test(Dia.valueOf(str));

t1.diaEs();

}

}

Salida:

Los lunes son feos.

* Cada constante enum siempre es implícitamente public static final. Entonces, como es static, podemos acceder utilizando el nombre del *enum*. Y como es final, no podemos crear enumeraciones “*hijas*“.
* Podemos declarar el método main() dentro de enum. Por lo tanto, podemos invocar enum directamente desde el *Símbolo del sistema*.

Por ejemplo:

// Un programa Java para demostrar que podemos tener

// main() dentro de enum

enum Color

{

ROJO, VERDE, AZUL;

// Método

public static void main(String[] args)

{

Color c1 = Color.ROJO;

System.out.println(c1);

}

}

El hecho de que enum define una clase permite que la enumeración de Java tenga poderes que las enumeraciones en otros lenguajes no tienen. Por ejemplo, puede **darle constructores**, **agregar variables** y **métodos de instancia**, e incluso **implementar interfaces**.

Aunque los **ejemplos anteriores** muestran la mecánica de crear y usar una enumeración, **no muestran todas sus capacidades**. A diferencia de la forma en que se implementan las enumeraciones en algunos otros lenguajes, *Java implementa enumeraciones como tipos de clases*. Aunque no creemos una instancia de una enumeración usando new, actúa de forma muy similar a otras clases.

## 6. Métodos values(), ordinal() y valueOf()

Todas las enumeraciones tienen automáticamente dos métodos predefinidos: **values()** y **valueOf()**. Sus formas generales se muestran aquí:

public static tipo-enum[ ] values( )

public static tipo-enum valueOf(String str)

* Estos métodos están presentes dentro de **java.lang.Enum**.
* El método **values​​()** se puede usar para devolver todos los valores presentes dentro de enum.
* El **orden** es importante en las enumeraciones. Al usar el método **ordinal()**, se puede encontrar cada índice de la constante enum, al igual que el índice de matriz.
* El método **valueOf()** devuelve la constante enum del valor de cadena especificado, si existe.

**values()**, se trata de un método especial que el compilador agrega automáticamente cuando se crea un enum. Este método devuelve un array conteniendo todos los valores del enumerado en el orden en que son declarados y se usa comúnmente en combinación con el ciclo for-each para iterar sobre los valores de un tipo enum.

Ejemplo de código. Compílalo y comprueba el resultado de ejecución (hay que definir el tipo enumerado en una clase independiente previamente):

// Programa Java para demostrar el funcionamiento de values(),

// ordinal() y valueOf()

enum Color

{

ROJO, VERDE, AZUL;

}

public class Test

{

public static void main(String[] args)

{

// Llamando a values()

Color arr[] = Color.values();

// enum con bucle

for (Color col : arr)

{

// Llamando a ordinal() para encontrar el índice

// de color.

System.out.println(col + " en el índice "

+ col.ordinal());

}

// Usando valueOf(). Devuelve un objeto de

// Color con la constante dada.

// La segunda línea comentada causa la excepción

// IllegalArgumentException

System.out.println(Color.valueOf("ROJO"));

// System.out.println(Color.valueOf("BLANCO"));

}

}

Salida:

ROJO en el índice 0

VERDE en el índice 1

AZUL en el índice 2

ROJO

## 7. enum: Constructores, métodos, variables de instancia

Es importante comprender que cada constante de enumeración es un objeto de su tipo de enumeración (como si fuera un objeto de una clase). Por lo tanto, **una enumeración puede definir constructores, agregar métodos y tener variables de instancia**.

La primera vez que llamamos a una instancia del enum se **inicializan todas** las instancias del mismo, se ejecutan los constructores para cada instancia respetando el orden definido. **Cada constante** de enumeración puede llamar a **cualquier método** definido por la enumeración. Cada constante de enumeración tiene su propia copia de cualquier variable de instancia definida por la enumeración.

* enum puede contener constructor y se ejecuta por separado para cada constante enum en el momento de la carga de la clase enum.
* No podemos crear objetos enum explícitamente y, por lo tanto, no podemos invocar el constructor enum directamente.

**Ejemplo**:

enum Color{

// Enums

ROJO("Rojo", 3), AZUL("Azul", 5);

// Atributos

private String nombreColor;

private int numColor;

public int contador=0;

// Constructor

Color(String nom, int col) {

this.nombreColor = nom;

this.numColor = col;

this.contador ++;

System.out.println(this.nombreColor + " - " +

this.numColor + " - " + this.contador);

// Rojo - 3 - 1

// Azul - 5 - 1

}

}

public class C1{

public static void main(String[] args){

//System.out.println("Inicializando: " + Color.ROJO);

// Se inicializan todas las instancias la primera vez que se

//usa el enum.

Color c1;

c1 = Color.AZUL;

System.out.println("Inicializando: " + c1);

Color c2;

c2 = Color.ROJO;

System.out.println("Inicializando: " + c2);

}

}

La siguiente versión de Transporte ilustra el uso de un constructor, una variable de instancia y un método. Da a cada tipo de transporte una velocidad típica:

//Uso de un constructor, una variable de instancia y un método.

enum Transporte{

COCHE(60), CAMION(50), AVION(600), TREN(70), BARCO(20);

private int velocidad; //velocidad típica de cada transporte

//Añadir un constructor

Transporte(int s){velocidad=s;}

//Añadir un método

int getVelocidad(){return velocidad;}

}

class Enumerados {

public static void main(String[] args) {

Transporte tp;

//Mostrar la velocidad de un avión

System.out.println("La velocidad típica para un avión es: "+

Transporte.AVION.getVelocidad()+ " millas por hora.\n");

//Mostrar todas las velocidades y transportes

System.out.println("Todas las velocidades de transporte: ");

for (Transporte t:Transporte.values())

System.out.println(t + ": velocidad típica es "

+ t.getVelocidad() + " millas por hora.");

}

}

Salida:

La velocidad típica para un avión es: 600 millas por hora.

Todas las velocidades de transporte:

COCHE: velocidad típica es 60 millas por hora.

CAMION: velocidad típica es 50 millas por hora.

AVION: velocidad típica es 600 millas por hora.

TREN: velocidad típica es 70 millas por hora.

BARCO: velocidad típica es 20 millas por hora.

### 7.1. Explicación del Ejemplo

Esta versión de Transporte agrega tres cosas:

La **primera** es la variable de instancia **velocidad**, que se usa para mantener la velocidad de cada tipo de transporte.

La **segunda** es el constructor **Transporte**, que pasa la velocidad de un transporte.

La tercera es el método **getVelocidad()**, que devuelve el valor de velocidad. Solo admite geters, que es lo que tiene sentido.

|  |
| --- |
| Cuando una variable de tipo *Transporte* (variable **tp** en el ejemplo)se declara en ***main() y se inicializa a un valor de la enumeración***, el constructor de Transporte se llama una vez para cada constante que se especifica. Observa cómo se especifican los argumentos para el constructor, poniéndolos entre paréntesis, después de cada constante, como se muestra aquí:  **COCHE(60), CAMION(50), AVION(600), TREN(70), BARCO(20);** |

Estos valores se pasan al parámetro de *Transporte()*, que luego asigna este valor a la *velocidad*. Hay algo más que notar sobre la lista de constantes de enumeración: **termina con un punto y coma**. Es decir, la última constante, *BARCO*, va seguida de un punto y coma. Cuando una enumeración contiene otros miembros, la lista de enumeración debe terminar en punto y coma.

Como cada **constante de enumeración tiene su propia copia de velocidad**, puede obtener la velocidad de un tipo de transporte especificado llamando a *getVelocidad()*. Por ejemplo, en *main()* la velocidad de un avión se obtiene mediante la siguiente llamada:

**Transporte.AVION.getVelocidad()**

La velocidad de cada transporte se obtiene al recorrer la enumeración mediante un [bucle for](https://javadesdecero.es/intermedio/bucle-for-each/). Como hay una copia de velocidad para cada constante de enumeración, el valor asociado con una constante es separado y distinto del valor asociado con otra constante. Este es un concepto poderoso, que está disponible solo cuando las enumeraciones se implementan como clases, como lo hace Java.

Aunque el ejemplo anterior contiene solo un constructor, una enumeración puede ofrecer dos o más formas de sobrecargas, al igual que cualquier otra clase.

## 8. Enum y Herencia

Hay dos **restricciones** que se aplican a las enumeraciones.

* Una enumeración no puede heredar de otra clase (ya hereda implícitamente de java.util.Enum).
* Una enumeración no puede ser una superclase (no se puede heredar de ella).

Esto significa que una enumeración no se puede extender. De lo contrario, enum actúa como cualquier otro tipo de clase. **La clave es recordar que cada una de las constantes de enumeración es un objeto de la clase en la que está definida**.

* Aunque no puede heredar de una superclase al declarar una enumeración, todas las enumeraciones heredan automáticamente una: **java.lang.Enum**. Esta clase define varios métodos que están disponibles para el uso de todas las enumeraciones.
* Muy a menudo, no necesitará usar estos métodos, pero hay dos que puede emplear ocasionalmente: **ordinal()** y **compareTo()**.
* El método **toString()** se reemplaza en la clase java.lang.Enum, que devuelve el nombre de la constante enum.
* enum puede implementar muchas interfaces.

### 8.1. Uso de ordinal( ) y compareTo()

El método ordinal() se muestra aquí:

**final int ordinal()**

Devuelve el valor ordinal de la constante invocadora. Los valores ordinales comienzan en cero. Por lo tanto, en la enumeración Transporte, COCHE tiene un valor ordinal de cero, CAMION tiene un valor ordinal de 1, AVION tiene un valor ordinal de 2, y así sucesivamente.

Se puede comparar el valor ordinal de dos constantes de la misma enumeración utilizando el método compareTo(). Tiene esta forma general:

**final int compareTo(tipo-enum e)**

Aquí, *tipo-enum* es el tipo de enumeración y *e* es la constante que se compara con la constante de invocación. Recuerda, tanto la constante de invocación como *e* deben ser de la misma enumeración.

* Si la constante de invocación tiene un valor ordinal menor que *e,* entonces *compareTo()* devuelve un valor **negativo**.
* Si los dos valores ordinales son iguales, se devuelve **cero**.
* Si la constante de invocación tiene un valor ordinal mayor que *e*, se devuelve un valor **positivo**.

El siguiente programa muestra **ordinal()** y **compareTo()**:

package enumTransportes4;

//Demostración de ordinal() y compareTo()

enum Transporte {

COCHE, CAMION, AVION, TREN, BARCO;

}

class Enumerados {

public static void main(String[] args) {

Transporte tp1, tp2, tp3;

// Obtiene todos los valores ordinales usando ordinal().

System.out.println("Aquí están todas las constantes de Transporte

y sus valores ordinales: ");

// Acuerdate de que Transporte.values() devuelve un array con las

// constantes de la enumeración

for (Transporte t : Transporte.values())

System.out.println(t + " " + t.ordinal());

tp1 = Transporte.AVION;

tp2 = Transporte.TREN;

tp3 = Transporte.AVION;

System.out.println();

// Uso de CompareTo()

if (tp1.compareTo(tp2) < 0)

System.out.println(tp1 + " esta antes que " + tp2);

else if (tp1.compareTo(tp2) > 0)

System.out.println(tp1 + " esta despues que " + tp2);

else

System.out.println(tp1 + " es igual que " + tp3);

tp1 = Transporte.CAMION;

tp2 = Transporte.COCHE;

tp3 = Transporte.CAMION;

// Uso de CompareTo()

if (tp1.compareTo(tp2) < 0)

System.out.println(tp1 + " esta antes que " + tp2);

if (tp1.compareTo(tp2) > 0)

System.out.println(tp1 + " esta despues que " + tp2);

if (tp1.compareTo(tp3) == 0)

System.out.println(tp1 + " es igual que " + tp3);

}

}

Salida:

Aquí están todas las constantes de Transporte y sus valores ordinales:

COCHE 0

CAMION 1

AVION 2

TREN 3

BARCO 4

AVION está antes que TREN

AVION es igual que AVION

**9. enum y Métodos**

* enum puede contener métodos concretos, es decir, que no tengan ningún método abstracto (**abstract**).

Por ejemplo (fijaos bien en la ejecución):

// Programa Java para demostrar que los enum pueden tener

// Constructores y métodos concretos.

// Una enumeración (Note la palabra enum en lugar de class)

enum Color

{

ROJO, VERDE, AZUL;

// enum constructor llamado por separado para cada constante

private Color()

{

System.out.println("Constructor llamado para : " + this);

}

// Solo métodos concretos (no abstractos) permitidos

public void colorInfo()

{

System.out.println(this + “ desde colorInfo()”);

}

}

public class Test

{

// Metodo

public static void main(String[] args)

{

Color c1 = Color.ROJO;

// Míralo en el Proyecto, está separado y se ve más claro

System.out.println(c1);

c1.colorInfo();

}

}

Salida:

Constructor llamado para : ROJO

Constructor llamado para : VERDE

Constructor llamado para : AZUL

ROJO

ROJO desde colorInfo()

Sacado (pero modificado) de:

<https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/#6_Metodos_values_ordinal_y_valueOf>