Los *Enum* (abreviatura de “enumeraciones”) en Java son una característica poderosa que permite definir un conjunto fijo de constantes relacionadas bajo un solo tipo. Gracias a los Enum podemos llegar incluso a ahorrarnos unas cuantas tablas a nivel de base de datos.

```
/**
 * @author <a href="mailto:jordyan1996@gmail.com">Jordy Andrs Rodrguez</a>
 * @project gesdos-model
 * @class EstadoEnum
 * @description
 * @date 02/08/2017
 */
public enum EstadoEnum {

	ACTIVO("A"), INACTIVO("I"), CERRADO("C"), PROVISIONAL("P");

	private String value;

	/**
	 * @author <a href="mailto:jordyan1996@gmail.com">Jordy Andrs Rodrguez</a>
	 * @date 02/08/2017
	 * @param value
	 */
	private EstadoEnum(String value) {
		this.value = value;
	}

	/**
	 * @author <a href="mailto:jordyan1996@gmail.com">Jordy Andrs Rodrguez</a>
	 * @date 02/08/2017
	 * @return the value
	 */
	public String getValue() {
		return value;
	}

}
```

¿Qué es un Enum en Java?

Un *Enum* en Java es un tipo especial de clase que representa un grupo de constantes (objetos inmutables). Por ejemplo, si tienes un conjunto de estados para un semáforo (ROJO, AMARILLO, VERDE), en lugar de usar variables independientes o cadenas de texto, puedes agruparlos en un *Enum*.

```
public enum Semaforo {
    ROJO, AMARILLO, VERDE;
}
```

Algo interesante a recalcar que una vez compilados los enum se transforman en clases y sus miembros se convierten en constantes estáticas.

```
public final class Semaforo extends java.lang.Enum<Semaforo> {

    // Definición de las constantes del enum como instancias estáticas
y finales
    public static final Semaforo ROJO = new Semaforo("ROJO", 0);
    public static final Semaforo AMARILLO = new Semaforo("AMARILLO",
1);
    public static final Semaforo VERDE = new Semaforo("VERDE", 2);

    // Constructor privado para evitar la creación de otras instancias
    private Semaforo(String name, int ordinal) {
        super(name, ordinal);
    }

    // Método values() generado automáticamente para devolver todas
las constantes del enum
    public static Semaforo[] values() {
        return new Semaforo[]{ROJO, AMARILLO, VERDE};
    }

    // Método valueOf() generado automáticamente para obtener una
constante del enum por su nombre
    public static Semaforo valueOf(String name) {
        return (Semaforo) Enum.valueOf(Semaforo.class, name);
```

```
    }
}
```

Podríamos decir entonces que los enum nos son también una abreviación de código, ya que podemos lograr algo similar con una clase convencional pero el proceso es mucho más largo.

Cosas que me gustan de los Enum en Java (características clave)

• **Tipo Seguro**: Los *Enum* son tipos seguros, lo que significa que no pueden contener valores fuera de los definidos. Esto ayuda a prevenir errores comunes relacionados con el uso de valores no válidos.

• **Compatibilidad con el Switch**: Los *Enum* pueden ser utilizados en declaraciones switch, lo que facilita el manejo de las diferentes constantes de manera estructurada.

• **Inmutabilidad**: Los valores de un *Enum* son inmutables, lo que garantiza que no puedan ser cambiados una vez definidos.

• **Comportamiento Complejo**: Los *Enum* pueden tener métodos, constructores, e incluso implementar interfaces, lo que los hace muy flexibles para manejar casos más complejos.

Ejemplo real usando Enum en Java

Cuando diseñamos un sistema es común tener una columna de estado sobre los registros de una tabla en base de datos, estos valores son perfectamente compatibles con lo que nos ofrece un Enum, veamos.

```
public enum EstadoEnum {
  ACTIVO, INACTIVO;
}
```

Este Enum nos permite almacenar el estado de un registro cualquiera para hacer comparaciones, podría ser algo así:

```
if (obj.getEstado().equals(EstadoEnum.ACTIVO.name())) {}
```

Eso no es todo...

El código anterior aunque válido se queda corto en la realidad de los Enum, básicamente porque generalmente no se almacena el valor “ACTIVO” es más común representar el estado a través de una letra, para este caso sería algo como “A” para ACTIVO e “I” para INACTIVO.

A nivel de código crear un Enum con esa característica puede ser raro, así que hay una alternativa mejor, hablamos de crear un constructor para el Enum:

```
public enum EstadoEnum {

	ACTIVO("A"), INACTIVO("I");

	private String value;

	private EstadoEnum(String value) {
		this.value = value;
	}

	public String getValue() {
		return value;
	}

}
```

Este Enum es más complejo que el anterior, pero ahora sí podemos almacenar los valores tal y como están en base de datos, la comparación para este caso es un tanto diferente:

```
if (obj.getEstado().equals(EstadoEnum.ACTIVO.getValue())) {}
```

En algunos casos los Enum nos pueden ayudar a explicar el código, por ejemplo en casos donde pocas letras no basten para dar contexto de lo que se está almacenando o comparando.