Apunte: Enums en Java

¿Por qué Enums?

En formularios reales (registro, nivel educativo, etc.) dejar un campo libre como String genera ambigüedades: “Secundario”, “secundaria”, “segundo”… todo significa lo mismo pero está escrito distinto. Solución: usar Enumeraciones (Enums) para estandarizar los valores posibles y evitar errores de entrada/comparación. Además, el código “te sugiere” las opciones válidas y mejora la calidad y legibilidad.

1) Enums básicos en Java

1.1 Definición

```
public enum NivelEstudio {
    SIN_ESTUDIOS, PRIMARIO, SECUNDARIO, TERCIARIO, UNIVERSITARIO,
POSGRADO;
}
```

• Convención: MAYÚSCULAS y '_' para separar palabras.
• Son “clases especiales” cuyas instancias (constantes) están fijadas dentro del enum.

1.2 Reemplazar String por Enum en el modelo (caso Persona/Empresa)

Problema original: Persona tenía nivelEstudio como String → múltiples variantes y errores.
Refactor: cambiar el tipo del atributo a NivelEstudio.

```
public class Persona {
    private String nombre;
    private String apellido;
    private NivelEstudio nivelEstudio;

    public Persona(String nombre, String apellido, NivelEstudio
nivelEstudio) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.nivelEstudio = nivelEstudio;
    }

    public NivelEstudio getNivelEstudio() { return nivelEstudio; }
    // getters/setters restantes…
}
// Uso desde main
Persona p = new Persona("Pepe", "Fulano", NivelEstudio.SECUNDARIO);
```

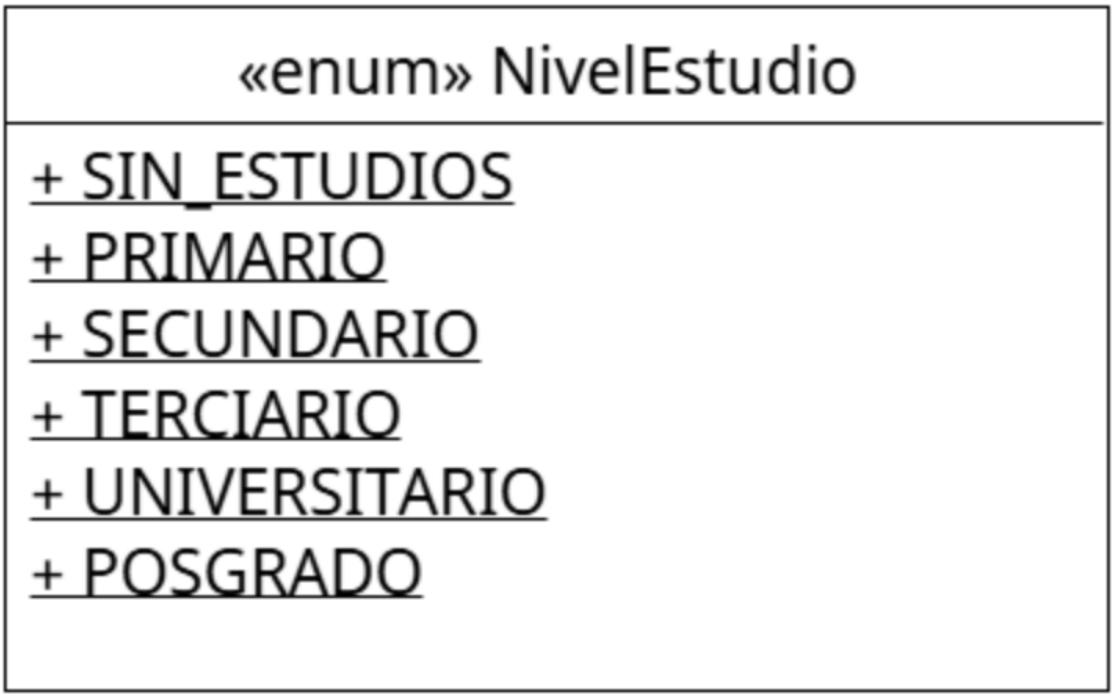
1.3 Comparación de Enums

Los enumerados se comparan por referencia; == funciona igual que equals()

```
if (persona.getNivelEstudio() == NivelEstudio.SECUNDARIO) {
    // coincide
}
```

2) Enums y UML

Se representan con el estereotipo <<enumeration>>. Los literales se listan uno debajo del otro; en UML suelen aparecer subrayados (estáticos) y públicos (+).



Relaciones:

• Si Persona tiene un atributo nivelEstudio: NivelEstudio → asociación (atributo).

• Si Empresa usa NivelEstudio en un método (mostrarPorNivel) pero no lo guarda como atributo → dependencia/uso (flecha punteada).

3) Métodos útiles de los Enums

3.1 Métodos de instancia

```
NivelEstudio n = NivelEstudio.PRIMARIO;
System.out.println(n.ordinal()); // 1 si SIN_ESTUDIOS es 0
System.out.println(n.name());    // "PRIMARIO"
System.out.println(n.toString());// "PRIMARIO" (salvo override)
```

3.2 Métodos de clase (estáticos)

```
NivelEstudio[] niveles = NivelEstudio.values();
for (int i = 0; i < niveles.length; i++) {
    System.out.println((i+1) + ") " + niveles[i]);
```

```
}

NivelEstudio n1 = NivelEstudio.valueOf("PRIMARIO");  // OK
NivelEstudio n2 = NivelEstudio.valueOf("Primario");  // ERROR:
IllegalArgumentException
```

Patrón menú: mostrar values() numerado (1..N), leer un entero y mapear con values()[op-1].

4) Enums con atributos y métodos propios

```
public enum Continente {
    AMERICA(1_002_000_000L, 42_320_000.0),
    EUROPA(741_000_000L, 10_180_000.0),
    AFRICA(1_216_000_000L, 30_370_000.0),
    ASIA(4_463_000_000L, 44_580_000.0),
    OCEANIA(40_000_000L, 8_525_989.0),
    ANTARTIDA(1_106L, 14_000_000.0);

    private final long habitantes;
    private final double superficieKm2;

    private Continente(long habitantes, double superficieKm2) {
        this.habitantes = habitantes;
        this.superficieKm2 = superficieKm2;
    }

    public long getHabitantes() { return habitantes; }
    public double getSuperficieKm2() { return superficieKm2; }

    public double densidadPoblacional() {
        return habitantes / superficieKm2; // personas por km²
    }

    @Override
    public String toString() {
        return name() + " (hab=" + habitantes + ", sup=" +
superficieKm2 + " km²)";
    }
}
```

- El constructor de un enum siempre es private.
- Podés calcular propiedades derivadas (p. ej. densidadPoblacional()).
- Podés iterar con values() y mostrar el estado (aprovechando un toString() sobreescrito).

5) Errores frecuentes y buenas prácticas

• Usar String para dominios cerrados → usar Enum.
• valueOf(String) es sensible a mayúsculas y requiere coincidencia exacta.
• Comparar Enums con ==.
• Modelar en UML el <<enumeration>> y distinguir asociación (atributo) de dependencia (uso).
• Si el enum tiene estado (atributos), inicializar cada literal; constructor private.