BIOS

OBJECT-ORIENTED PROGRAMMING

ENUMS

- Un ***Enum*** *en java es una ¨clase¨ especial que representa un grupo de constantes.*
- ***Enum*** es la abreviatura de "*enumeraciones*", que significa *"enumerado específicamente"*.
- Para crear un *enum* usamos la palabra ***enum*** en lugar de class o interface.

public ***enum*** **Dia {**
LUNES,
MARTES,
MIERCOLES,
JUEVES,
VIERNES,
SABADO,
DOMINGO **}**

ENUMS

- Internamente, cada ***enum*** en Java se implementa como una clase.
- Cada constante en la clase de enumeración es un ***objeto*** de tipo ***enum***.
- Los ***enum*** se pueden pasar como argumento a las sentencias ***switch***.
- Los objetos de ***enum*** son estáticos y finales por defecto.
- El método ***main***() se puede crear dentro de una clase de ***enum***. Por lo tanto, los enum se pueden invocar directamente: **Dia.LUNES**

public ***enum*** **Dia {**
LUNES,
MARTES,
MIERCOLES,
JUEVES,
VIERNES,
SABADO,
DOMINGO **}**

ALGUNAS PARTICULARIDADES

- Podemos tener enums definidos dentro de clases
- Pueden declarar múltiples variables.
- Pueden incluir métodos y atributos.
- No pueden extenderse ni tampoco heredan otras clases, porque ya extienden de ***java.lang.Enum***
- Si pueden implementar Interfaces.

```
class HelloWorld {
    enum Day {
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
    }
    public static void main(String[] args) {
        for(Day day : Day.values())
        System.out.println(day);
    }
}
```

✅ USAR ENUMS CUANDO:

Cuando tenemos valores que sepa que no van a cambiar, como meses, días, colores, baraja de cartas, tipos, etc.

✅ CASOS DE USO COMUNES DE ENUMS:

- **Menús y opciones fijas:** Para representar elementos de un menú donde las opciones son fijas y conocidas.
- **Estados de un proceso:** Como en un flujo de trabajo donde los estados posibles están predefinidos.
- **Días de la semana, meses del año:** Para manejar estos valores de manera segura y legible.

ENUM VS CLASS

Aspecto	Clases	Enums
Propósito	Representar cualquier tipo de entidad, concepto o abstracción.	Representar un **conjunto fijo y bien definido de constantes.**
Instanciación	Se crean objetos con `new` o constructores estáticos.	**No se pueden crear nuevas instancias** fuera de las que declara el enum; cada constante es única y se crea implícitamente en tiempo de carga.
Constantes internas	Puedes declarar `static final` tú mismo, pero no es obligatorio.	Cada constante del enum es implícitamente `public static final`, inmutable y no puede ser sobrescrita.
Herencia	Puede extender **una** clase (excepto `final`) y cualquier número de interfaces.	**No puede extender** otra clase (ya hereda de `java.lang.Enum`), **pero sí puede implementar interfaces.**
Atributos y métodos	Puede tener cualquier campo y método (instancia o estático).	También puede definir campos, constructores y métodos (incluso abstractos que cada constante implementa).
Uso en `switch`	Se puede usar si defines constantes `static final`, pero no es común.	Se usa habitualmente: `switch(enumValue) { case VALOR1: … }`.

ENUM METHODS

Método	Tipo	¿Para qué sirve?	Detalles clave
`static T[] values()`	Generado por el compilador en cada enum	Devuelve un **array** con todas las constantes en el orden declarado.	Ideal para recorrer o mostrar todas las opciones disponibles.
`String name()`	Instancia	Retorna el **identificador literal** de la constante tal cual aparece en el código fuente.	Inmutable.
`int ordinal()`	Instancia	Devuelve la **posición (índice)** de la constante, empezando en 0.	Cambia si reordenás las constantes; no lo uses para persistir.
`String toString()`	Instancia	Por defecto delega a `name()`.	Se puede **sobre-escribir** para mostrar textos más legibles.

CONSTRUCTORES: ENUMS

Ya definimos que cada constante que creemos en un ***enum*** sera una instancia/objeto de ese enum, creada automaticamente por el compilador y no mediante el uso de ***new***.

Pero si podremos crear nuestros constructores para los enums que definamos y el compilador los usará para crear nuestras constantes como objetos.

```
CALCULATE(description:"Compute accrued leave days"),
TABLE(description:"Show monthly accrual table"),
EXIT(description:"Exit program");

private final String description;

MenuOption(String description) {
 this.description = description;
}
```

¿PARA QUÉ SIRVE ENTONCES EL CONSTRUCTOR?

1 - Inicializar campos: Cada constante llama implícitamente al constructor en el momento de cargarse la clase. Así puedes asociarle datos como la descripción del ejemplo.

2- Garantizar inmutabilidad y seguridad de tipos: Como el constructor es siempre private (o sin modificador), nadie fuera del enum puede construir valores nuevos ni alterar los existentes.

3- Añadir lógica de validación: Puedes comprobar precondiciones o derivar valores internos dentro del constructor, igual que en una clase normal.

4- Sobrecarga de constructores: Al igual que una clase, un enum puede tener varios constructores privados para crear constantes con distinta configuración.

```
MenuOption(String description) {
 this.description = description;
}
```

LOOP - ENUMS

Los ***enum*** tienen un método ***values()*** que devuelve un array de todas las constantes. Este método es útil para recorrer las constantes de un enum en bucle ***for***.

```java
package enums;
public enum MenuOption {
    CALCULATE(description:"Compute accrued leave days"),
    TABLE(description:"Show monthly accrual table"),
    EXIT(description:"Exit program");

    private final String description;

    MenuOption(String description) {
     this.description = description;
    }
```

```java
/* Imprime el menú recorriendo el enum con un for-each */
public static void printMenu() {
    System.out.println(x:"\n=== Leave Days Calculator ===");
    for (MenuOption opt : MenuOption.values()) {
        System.out.printf(format:"%d) %s%n", opt.ordinal() + 1, opt.getDescription());
    }
    System.out.print("Choose [1-" + MenuOption.values().length + "]: ");
}
```

```java
public enum Month {
    JANUARY(index:1), FEBRUARY(index:2), MARCH(index:3), APRIL(index:4), MAY(index:5), JUNE(index:6),
    JULY(index:7), AUGUST(index:8), SEPTEMBER(index:9), OCTOBER(index:10), NOVEMBER(index:11), DECEMBER(index:12);

    private static final double DAYS_PER_MONTH = 1.6;
    private final int index;                        // 1-12

    Month(int index) { this.index = index; }

    /** Días acumulables hasta este mes, capados a 20 días */
    public double accruedDays() {
        return Math.min(index * DAYS_PER_MONTH, b:20);
    }
}
```

MONTH ENUM

```java
case TABLE -> {
    System.out.println(x:"\nMonth\tAccrued Days");
    for (Month m : Month.values()) {            // for-each sobre Month
        System.out.printf(format:"%-9s %.1f%n", m.name(), m.accruedDays());
    }
}
```

For each para el enum de month: por cada month llama al metodo definido en el enum: accured days()

```
=== Leave Days Calculator ===
1) Compute accrued leave days
2) Show monthly accrual table
3) Exit program
Choose [1-3]: 2

Month    Accrued Days
JANUARY   1.6
FEBRUARY  3.2
MARCH     4.8
APRIL     6.4
MAY       8.0
JUNE      9.6
JULY      11.2
AUGUST    12.8
SEPTEMBER 14.4
OCTOBER   16.0
NOVEMBER  17.6
DECEMBER  19.2
```

CONSOLE PRINT

SWITCH ENUMS

Los enums se utilizan a menudo en switch para comprobar valores correspondientes:

switch(enumValue) { **case VALOR1**: ...
case VALOR2:...}

```java
package enums;
public enum MenuOption {
    CALCULATE(description:"Compute accrued leave days"),
    TABLE(description:"Show monthly accrual table"),
    EXIT(description:"Exit program");

    private final String description;

    MenuOption(String description) {
     this.description = description;
    }
```

```java
choice = MenuOption.fromOrdinal(Integer.parseInt(sc.nextLine()));
```

```java
/* switch sobre el enum del menú */
switch (choice) {
    case CALCULATE -> {
        System.out.print(s:"Months worked (0-12): ");
        int months = Integer.parseInt(sc.nextLine());
        if (months < 0 || months > 12) {
            System.out.println(x:"Months must be between 0 and 12.");
            break;
        }
        double days = Math.min(months * 1.6, b:20);
        System.out.printf(format:"Accrued leave: %.1f days%n", days);
    }
    case TABLE -> {
        System.out.println(x:"\nMonth\tAccrued Days");
        for (Month m : Month.values()) {            // for-each sobre Month
            System.out.printf(format:"%-9s %.1f%n", m.name(), m.accruedDays());
        }
    }
    case EXIT -> System.out.println(x:"Goodbye!");
}
```

COMPLETED EXAMPLE: ENUMS

```java
package enums;
public enum MenuOption {
    CALCULATE(description:"Compute accrued leave days"),
    TABLE(description:"Show monthly accrual table"),
    EXIT(description:"Exit program");

    private final String description;

    MenuOption(String description) {
     this.description = description;
    }

    public String getDescription() { return description; }

    /* Imprime el menú recorriendo el enum con un for-each */
    public static void printMenu() {
        System.out.println(x:"\n=== Leave Days Calculator ===");
        for (MenuOption opt : MenuOption.values()) {
            System.out.printf(format:"%d) %s%n", opt.ordinal() + 1, opt.getDescription());
        }
        System.out.print("Choose [1-" + MenuOption.values().length + "]: ");
    }

    /* Convierte la opción numérica en la constante del enum */
    public static MenuOption fromOrdinal(int choice) {
        if (choice < 1 || choice > values().length) throw new IllegalArgumentException();
        return values()[choice - 1];
    }
}
```

```java
package enums;
public enum Month {
    JANUARY(index:1), FEBRUARY(index:2), MARCH(index:3), APRIL(index:4), MAY(index:5), JUNE(index:6),
    JULY(index:7), AUGUST(index:8), SEPTEMBER(index:9), OCTOBER(index:10), NOVEMBER(index:11), DECEMBER(index:12);

    private static final double DAYS_PER_MONTH = 1.6;
    private final int index;                        // 1-12

    Month(int index) { this.index = index; }

    /** Días acumulables hasta este mes, capados a 20 días */
    public double accruedDays() {
        return Math.min(index * DAYS_PER_MONTH, b:20);
    }
}
```

COMPLETED EXAMPLE: MAIN APP

```java
public class LeaveCalculator {
    Run | Debug
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        MenuOption choice;

        do {
            MenuOption.printMenu();                // for-each sobre MenuOption
            try {
                choice = MenuOption.fromOrdinal(Integer.parseInt(sc.nextLine()));
            } catch (Exception e) {
                System.out.println(x:"Invalid selection.");
                choice = null;  // Reset choice to null for next iteration
                continue;
            }

            /* switch sobre el enum del menú */
            switch (choice) {
                case CALCULATE -> {
                    System.out.print(s:"Months worked (0-12): ");
                    int months = Integer.parseInt(sc.nextLine());
                    if (months < 0 || months > 12) {
                        System.out.println(x:"Months must be between 0 and 12.");
                        break;
                    }
                    double days = Math.min(months * 1.6, b:20);
                    System.out.printf(format:"Accrued leave: %.1f days%n", days);
                }
                case TABLE -> {
                    System.out.println(x:"\nMonth\tAccrued Days");
                    for (Month m : Month.values()) {            // for-each sobre Month
                        System.out.printf(format:"%-9s %.1f%n", m.name(), m.accruedDays());
                    }
                }
                case EXIT -> System.out.println(x:"Goodbye!");
            }

        } while (choice != MenuOption.EXIT);
        sc.close();
    }
}
```