

Tipos Enumerativos

Tipos Enumerativos

JAVA soporta 2 categorías de tipos de datos de propósito específico:

Una categoría de clases: **Tipos Enumerativos**

Una categoría de interfaces: **Anotaciones**

Tipos Enumerativos

- Los **tipos enumerativos** se incorporan a la plataforma JAVA partir de JAVA 5.0. Constituyen una **categoría especial de clases**.
- Un tipo enumerativo es un tipo de datos que tiene asociado un **conjunto de valores finito y acotado**.
- La palabra clave **enum** se usa para definir un tipo enumerativo:

package labo;

El cuerpo del tipo enum es una lista separada por comas de los valores posibles

public enum Estados {CONECTANDO, LEYENDO, LISTO, ERROR ;}

- Los **valores** son **constantes públicas de clase (public static final)**
- A los valores de un tipo enumerativo se llama **valores enumerados** y también **constantes enum**.
- A una variable de tipo **Estados** se le puede asignar uno de los 4 valores definidos o **null**. Dichos valores se referencian de la siguiente manera: **Estados.CONECTANDO, Estados.LEYENDO, Estados.LISTO, Estados.ERROR**.
- **El tipo enumerativo es una clase y sus valores son instancias de dicha clase.** Garantiza **seguridad de tipos**. Es una diferencia fundamental con usar constantes de tipo primitivo. El compilador puede chequear si a un método se le pasa un objeto de tipo **Estado**.
- Por **convención**, los valores de los tipos enumerativos se escriben en **mayúsculas** como cualquier otra constante de clase.

Características de los Tipos Enum

Cuando se crea un tipo enumerado el compilador crea una clase que es subclase de `java.lang.Enum`. No es posible extender la clase `Enum` para crear un tipo enumerativo propio. La única manera de crear un tipo enumerativo es usando la palabra clave `enum`.

`public abstract class Enum<E> extends Enum<E> extends Object implements Comparable<E>, Serializable`

Constructors		
Modifier	Constructor	Description
protected	<code>Enum(String name, int ordinal)</code>	Sole constructor.

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type		Method	
protected final Object		<code>clone()</code>	
final int		<code>compareTo(E o)</code>	
final Optional<Enum.EnumDesc<E>>		<code>describeConstable()</code>	
final boolean		<code>equals(Object other)</code>	
protected final void		<code>finalize()</code>	
final Class<E>		<code>getDeclaringClass()</code>	
final int		<code>hashCode()</code>	
final String		<code>name()</code>	
final int		<code>ordinal()</code>	
String		<code>toString()</code>	
static <T extends Enum<T>>		<code>valueOf(Class<T> enumClass, String name)</code>	

Los tipos enumerativos no tienen constructores públicos. Las únicas instancias son las declaradas por el tipo enum.

- Los tipos enumerativos implementan la interface `java.lang.Comparable` y `java.io.Serializable`.
- El método `compareTo()` establece un orden entre los valores enumerados de acuerdo al orden en que aparecen en la declaración del `enum`. Es **final**.
- Es seguro comparar valores enumerativos usando el operador `==` en lugar de el método `equals()` dado que el conjunto de valores posible es limitado. El método `equals()` internamente usa el operador `==` y además es **final**.
- El método `name()` devuelve un String con el nombre de la constante enum. Es **final**.
- El método `ordinal()` devuelve un entero con la posición del enum según está declarado. Es **final**.
- El método `toString()` puede sobreescribirse. Por defecto retorna el nombre de la instancia del enumerativo.

^T No es posible extender un tipo enumerativo, son implícitamente final. El compilador define *final* a la clase que lo soporta.

Usos de Tipos Enumerativos

Tipos Enum y sentencia Switch

```
Estados misEstados=Estados.LEYENDO;
```

```
switch(misEstados) {  
  case CONECTANDO: {  
    System.out.println(misEstados);  
    break;  
  }  
  case LEYENDO: {  
    System.out.println(misEstados);  
    break;  
  }  
  case LISTO: {  
    System.out.println(misEstados);  
    break;  
  }  
  case ERROR:  
    throw new IOException("error");  
}
```

La sentencia **switch** soporta **tipos enumerativos**.

Si el tipo de la declaración de la expresión **switch** es un tipo enumerativo, las etiquetas de los **case** deben ser todas **instancias sin calificación** de dicho tipo.

Es ilegal usar **null** como valor de una etiqueta **case**.

Si NO se incluyen todos los valores posibles del tipo enumerativo en las etiquetas de los **case** o la etiqueta **default**, el compilador emite una advertencia.

Usos de Tipos Enumerativos

EnumMap y EnumSet

La clase `java.util.EnumMap` es una implementación especializada de un **Map** que requiere como clave un tipo **Enumerativo** y la clase `java.util.EnumSet` es una implementación de **Set** adaptada a valores de tipo **Enumerativo**. Ambas estructuras de datos están optimizadas para tipos Enumerativos.

```
package labo;
import java.util.EnumMap;
public class TestEnumHash {
    public static void main(String[] args) {
```

```
        EnumMap<Estados,String> mensajes = new EnumMap<>();
        mensajes.put(Estados.CONECTANDO, "Conectando...");
        mensajes.put(Estados.LEYENDO, "Leyendo...");
        mensajes.put(Estados.LISTO, "Listo!!...");
        mensajes.put(Estados.ERROR, "Falla en la descarga....");
```

```
        Estados miEstado= getEstado();
        String unMensaje = mensajes.get(miEstado);
        System.out.print(unMensaje);
```

```
    }
    public static Estados getEstado(){
        return Estados. CONECTANDO;
    }
}
```



Crea un Map vacío cuyas claves son del tipo enum Estados

Tipos Enumerativos Enriquecidos

Los tipos enumerativos pueden incluir métodos y propiedades.

```
package labo;  
public enum Prefijo {  
    MM("m", .001),  
    CM("c", .01),  
    DM("d", .1),  
    DAM("D", 10.0),  
    HM("h", 100.0),  
    KM("k", 1000.0);
```

Las instancias se declaran al principio

El constructor y los métodos se declaran igual que en las clases

Cada constante u objeto Prefijo se declara con valores para la **abreviatura** y para el factor **multiplicador**

Cuando se declaran **propiedades** y **métodos**, la lista de constantes enumerativas termina en ;

```
    private String abrev;  
    private double multiplicador;
```

Propiedades de los Prefijos: abreviatura y factor multiplicador

```
    Prefijo(String abrev, double multiplicador) {  
        this.abrev = abrev;  
        this.multiplicador = multiplicador;  
    }
```

Se debe proveer de un **constructor**.

Los **valores declarados para las propiedades** se pasan al constructor cuando se crean las constantes.

El **constructor** de un tipo enumerativo se define con acceso privado o privado del paquete.

El compilador crea automáticamente las instancias.

NO puede ser invocado.

```
    public String abrev() { return abrev; }  
    public double multiplicador() { return multiplicador; }
```

Métodos que permiten **recuperar la abreviatura** y el **factor multiplicador** de cada Prefijo



Tipos Enumerativos Enriquecidos

```
package labo;
```

```
public class TestPrefijo {
```

```
public static void main(String[] args) {
```

```
    double longTablaM= Double.parseDouble(args[0]);
```

```
    for (Prefijo p : Prefijo.values() )
```

```
        System.out.println("La longitud de la tabla en "+ p+ " "+longTablaM*p.multiplicador());
```

```
    }
```

```
}
```

values() es un **método de Clase** que inserta el compilador y que permite **recuperar en un arreglo todos los valores del enumerativo** en el orden en que fueron declarados.

java TestPrefijo 15

La longitud de la tabla en MM 0.015

La longitud de la tabla en CM 0.15

La longitud de la tabla en DM 1.5

La longitud de la tabla en DAM 150.0

La longitud de la tabla en HM 1500.0

La longitud de la tabla en KM 15000.0

Tipos Enumerativos Enriquecidos

Sobreescritura del método **toString()** de una enumeración:

```
package labo;
```

```
public enum Señales {  
    VERDE, ROJO, AMARILLO;
```

```
    public String toString() {  
        String id = name();    Recupera el nombre de la instancia  
        String minuscula = id.substring(1).toLowerCase();  
        return id.charAt(0) + minuscula;  
    }  
}
```

Verde
Rojo
Amarillo

```
package labo;  
import static java.lang.System.out;
```

```
public class PruebaSeñales {  
    public static void main (String args[]){  
        for (Señales s: Señales.values())  
            out.println(s);  
    }  
}
```

Tipos Enumerativos Enriquecidos

Es posible asociar **comportamiento diferente** con cada constante enumerativa.

package enumerativos;

public enum Operation {

PLUS, MINUS, TIMES, DIVIDE;

public double apply(double x, double y){

switch (this){

case PLUS: return x+y;

case MINUS: return x-y;

case TIMES: return x*y;

case DIVIDE: return x/y;

}

throw new AssertionError("Operación desconocida: " + this);

}

}

Este código funciona bien, pero...

- **NO compilará** si no le ponemos la sentencia **throw** porque el final del método es técnicamente alcanzable a pesar que nunca se alcanzará (están los **cases** de todos los enumerativos).
- El **código es frágil**: si se agrega una nueva constante enumerativa y nos olvidamos de agregar el **case** correspondiente en el switch, el enumerativo compilará pero dará un error en ejecución cuando intenta aplicar la nueva operación.

El enumerativo representa las **operaciones** de una calculadora básica de cuatro funciones aritméticas.
¿Cómo implementamos la operación aritmética representada por cada constante?

Tipos Enumerativos Enriquecidos

Es posible asociar **comportamiento diferente** con cada constante enumerativa.

```
package enumerativos;
```

```
public enum Operation {
```

```
    PLUS("+") {double apply(double x, double y) {return x + y;} },
```

```
    MINUS("-") {double apply(double x, double y) {return x - y;} },
```

```
    TIMES("*") {double apply(double x, double y) {return x * y;} },
```

```
    DIVIDE("/") {double apply(double x, double y) {return x / y;} };
```

```
    private final String symbol;
```

```
    Operation(String symbol) {
```

```
        this.symbol = symbol;
```

```
    }
```

```
    public String toString() {
```

```
        return symbol;
```

```
    }
```

```
    abstract double apply(double x, double y);
```

Esta es una mejor solución para **asociar comportamiento específico con cada constante enumerativa**:

- Declarar un **método abstracto** en el tipo enumerativo
- **Sobreescribirlo** con un **método concreto** en cada constante.

Son **implementaciones de métodos de constantes específicas**.

```
} El código es robusto: si nos olvidamos de agregar el método apply() cuando agregamos una nueva constante, el compilador lo recordará, los métodos abstractos en un tipo enum deben sobreescribirse con cada constante.
```

Tipos Enumerativos Enriquecidos

Combinar implementaciones de métodos (apply() y toString()) y las constantes enum

```
package enumerativos;

public class TestEnum{
    public static void main(String[] args) {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);
        for (Operation op : Operation.values())
            System.out.printf("%f %s %f = %f%n",x, op, y, op.apply(x, y));
    }
}
```

¿Cuál es la salida de ejecutar TestEnum 12 5 ?

12,000000 + 5,000000 = 17,000000

12,000000 - 5,000000 = 7,000000

12,000000 * 5,000000 = 60,000000

12,000000 / 5,000000 = 2,400000

Tipos Enumerativos Enriquecidos

Los tipos Enumerativos **NO pueden extenderse** pero si pueden **implementar interfaces**.

```
package labo;
```

```
public interface Abreivable {  
    String abrev();  
}
```

Se define la interface **Abreivable** para cualquier objeto que pueda abreviarse

Cualquier método que acepte como parámetro un objeto **Abreivable** aceptará una instancia de **UnidadesTemperatura**.

```
package labo;
```

```
public enum UnidadesTemperatura implements Abreivable {
```

```
    GRADOCELSIUS("°C"),  
    GRADOFARENHEIT("°F"),  
    GRADOREAUMUR("°R"),  
    GRADONEWTON("°N");  
    private String abrev;  
    UnidadesTemperatura(String abrev){  
        this.abrev=abrev;  
    }  
    public String abrev() {  
        return abrev;  
    }  
}
```

```
public static void imprimirAbreviatura(Abreivable s){  
    out.println(s.abrev());  
}  
  
public static void main(String args[]){  
    for (UnidadesTemperatura u :  
        UnidadesTemperatura.values() )  
        imprimirAbreviatura(u);  
    }  
}
```