Clase 4-enums, genéricos

miércoles, 10 de septiembre de 2025 14:58

ENUMS
Tipo de datos con un conjunto de valores finito y acotado
El cuerpo es una liasta separa por comas de los valores posibles
El tipo enum es una clase, sus valores sonb instancias de la clase.
Cuando creamos un enum, crea una clase subclase de java.lang.Enum. NO Se puede extender esa clase por disenio
El equals y == funcionan igual en enums.

- 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.

Values(): devuelve un arreglo de todos los valores posibles

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

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 ;
Propiedades de los Prefijos: abreviatura y factor 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.
Métodos que permiten **recuperar la abreviatura** y el **factor multiplicador** de cada Prefijo

Declaro un constructor para mi enum (que no puedo usar) y digo cuales son mis objetos, pero es como si fuera un objeto.
NO es un primitivo, es un objeto. Una cantidad controlada de objetos.
Ejemplo de values y de llamar a los metodos definidos arriba

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

GENERICOS
Tienen como parmetros tipos de datos.
Para dar chequeo de tipos en compilación en colecciones.
No pueden declararse genéricos los tipos enumerativos, las clases anónimas y subclases de excepciones.

```
public class LinkedList <E>  extends AbstractSequentialList <E> implements List<E>, Queue<E>, Cloneable, Serializable
```

LinkedList es un tipo Genérico
E es un parámetro formal que denota un tipo de dato
Los elementos que se almacenan en la lista encadenada son del tipo desconocido E

```
Collection <String> col=new LinkedList<String>();
List <String> list= new ArrayList<>();  // a partir de JAVA 7
Collection <? extends Number> col=new LinkedList<Integer>();
```

TIPO PARAMETRIZADO

En el último, le digo que no me importa que venga siempre que extienda number.

No son tipos concretos, no pueden usarse en la sentencia new.

```
static int cantElemEnComun(Set<?> s1, Set<?> s2) {
    int result = 0;
    for (Object o1 : s1)
        if (s2.contains(o1)) result++;
    return result;
}
```

Tipos parametrizados comodines sin cota

El **?** indica un conjunto de "algún tipo desconocido".

Es el **conjunto parametrizado más general**, capaz de contener cualquier tipo de elemento. No interesa cuál es el parámetro del tipo real.

Puede ser Set<String>, Set<A>, Set<Long>, etc.

No poner nada es menos seguro. Idk por que, estaba en otra :)

`List<? extends Number> l;` La **familia de todos los tipos de listas** cuyos elementos son **subtipos de Number.**

`Comparable<? super  String>  s;` La **familia de todas las instanciaciones de la interface Comparable** para tipos que son **supertipos de String**.

Ejemplos

Suma de los números de una lista de números:

```
public static double sum(List<Number> list){
double sum = 0;
for(Number n : list){
 sum += n.doubleValue();
}
return sum;
}
```

Los **tipos parametrizados concretos son invariantes** por lo tanto List<Double> y List<Integer> no están relacionados con List<Number>, no tienen relación de subtipo ni supertipo y el método sum() no se puede usar.

Los **tipos parametrizados con comodines acotados** ofrecen mayor **flexibilidad** que los tipos invariantes.

```
public static double sum(List<? extends Number> list){
double sum = 0;
for(Number n : list){
sum += n.doubleValue();
}
return sum;
}
```

```
public static void main(String[] args) {
List<Integer> ints = new ArrayList<>();
ints.add(3); ints.add(5); ints.add(10);
double sum = sum(ints);
System.out.println("Suma de ints="+sum);
}
```

En el primero, cualquier cosa
En el segundo, como que le seteo el tipo
Creo, no estaba prestando atención.

Dos chequeos distintos

Los **tipos parametrizados** forman una jerarquía de tipos basada en el tipo base NO en el tipo de los argumentos. Los **tipos parametrizados son invariantes**.

```
public interface List<E> extends Collection<E> {}
public class ArrayList<E> extends AbstractList<E>  implements List<E> {}
List<Integer> listita = new ArrayList<>();
List<Integer> l= listita;
Collection<Integer> c = listita;
ArrayList<Number> n = listita;
List<Object> o = listita;
List li= listita;
```

¿Cuáles asignaciones cumplen las reglas de subtipo?

El de number está mal porque es de tipo integer, no number.
Si fuera integer estaría bien
El de abajo si.
Falla por el tipo de adentro.

Un ArrayList<Integer> es un List<Integer>, un Collection<Integer> y un List, pero NO es un ArrayList<Number> ni un List<Object> .

List<Integer> no es un subtipo de List<Number>

```
List<Integer> li = new ArrayList<>();
li.add(123);
List<Number> lo = li;
Number nro = lo.get(0);
lo.add(3.14);
Integer i = li.get(1);
```

No compila List<Number> lo=li; NO ES POSIBLE CONVERTIR DE List<Integer> a List<Number>

Si asumimos que compila:

- podríamos recuperar elementos de la lista como Number en vez de como Integer: Number nro = lo.get(0);
- podríamos agregar un objeto Double: lo.add(3.14);
- la línea li.get(1); daría error de *casting*, porque no puedo *castear* un Double a un Integer

Revisar
No podría compilar porque si le metiera algo double por ejemplo después si lo quiero sacar estaría diciendo que mi int tiene un double y no tiene sentido.