Tipos Enumerativos

Tipos Enumerativos

La versión 5 de la plataforma Java incorpora 2 nuevos tipos referenciales:

Una nueva categoría de clases: **Tipos Enumerativos**

Una nueva categoría de interfaces: **Anotaciones**

Tipos Enumerativos

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

```
package clase3;
public enum Estados {CONECTANDO, LEYENDO, LISTO, ERROR ;}
```

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

- ✓ Los valores son constantes públicas de clase (public static final) y se hace referencia a ellas de la siguiente manera: **Estados.CONECTANDO**, **Estados.LEYENDO**. A una variable de tipo **Estados** se le puede asignar uno de los 4 valores definidos o **null**. Los valores de un tipo enumerado se llaman **valores enumerados** y también **constantes enum**.
- ✓ 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 **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 (incorporada en Java 5.0). No es posible extender la clase Enum para crear un tipo enumerativo propio. La única manera de definir un tipo enumerativo es usando la palabra clave enum.

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

Constructor	
protected	**Enum(String name, int ordinal)** Es inaccesible para los programadores.

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

Métodos	
protected Object	**clone()**
int	**compareTo(E o)**
boolean	**equals(Object other)**
Class<E>	**getDeclaringClass()**
int	**hashCode()**
String	**name()**
int	**ordinal()**
String	**toString()**
static <T extends Enum<T>> T	**valueOf(Class<T> enumType, String name)**

- Los tipos enumerativos implementan la interface **java.lang.Comparable** y **java.io.Serializable**. El método **compareTo()** establece un orden entre los valores enumerados considerando el 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()** ya que el conjunto de valores posible es limitado.
- El método **equals()** internamente usa el operador **= =** y además es **final**, por lo tanto no puede sobreescribirse.
- El método **toString()** puede sobreescribirse. Por defecto retorna el nombre de la instancia del enumerativo

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

```
package clase5;
public enum ServerEstados extends
Estados{
.............
}
```

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 ");
}
```

En las versiones anteriores a JAVA 5.0, la sentencia **switch** sólo se podía usar con valores **int**, **short**, **char** y **byte**. Al tener un conjunto finito de valores, los tipos enumerados son ideales para usar con la sentencia **switch**.
A partir de Java 5.0 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

Java 5.0 incorpora la clase **java.util.EnumMap** que es una implementación especializada de un **Map** que requiere como clave un tipo **Enumerativo** y la clase **java.util.EnumSet** que requiere valores de tipo **Enumerativo**. Ambas estructuras de datos están optimizadas para tipos Enumerativos.

EnumMap permite asociar un valor con cada instancia del tipo Enumerativo.

```
package clase3;
import java.util.EnumMap;
public class TestEnumHash {
        public static void main(String[] args) {
                EnumMap<Estados,String> mensajes =   new EnumMap<Estados,String>(Estados.class);
                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 enumerativo vacío con el tipo de claves dada por el tipo Estados

Tipos Enumerativos Avanzados

Los tipos enumerativos son poderosos, pueden incluir métodos y atributos.

```
package clase3;
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 se declara con valores para la abreviatura y para el factor multiplicador

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

Propiedades de los Prefijos

Se debe proveer de un constructor.
Los valores declarados, se pasan al constructor cuando se crean las constantes

El constructor de un tipo enumerativo tiene acceso privado o privado del paquete. Automáticamente crea las instancias y NO puede ser invocado.

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

Tipos Enumerativos

Avanzados (Continuación)

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

```
package clase3;

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());
  }

}
```

```
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 Avanzados(Continuación)

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

```
package clase3;

public enum Señales {
 VERDE, ROJO, AMARILLO;

 public String toString() {
  String id = name();
  String minuscula = id.substring(1).toLowerCase();
  return id.charAt(0) + minuscula;
 }
}
```

Recupera el nombre de la instancia

```
package clase3;
import static java.lang.System.out;

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

Verde
Rojo
Amarillo

Tipos Enumerativos Avanzados (Continuación)

ASOCIAR COMPORTAMIENTO 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 desconocias: " + 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.

Tipos Enumerativos Avanzados (Continuación)

ASOCIAR COMPORTAMIENTO 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 manera de asociar comportamiento específico con cada constante enumerativa:
Declarar un método abstracto en el tipo enumerativo y sobreescribirlo con un método concreto en cada constante.
Estos métodos son implementaciones de métodos de constantes específicas.

El código es más 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

Avanzados (Continuación)

COMBINAR IMPLEMENTACIONES DE MÉTODOS DE CONSTANTE ESPECÍFICA CON DATOS DE CONSTANTE ESPECÍFICA (appy() y toString())

```
package enumerativos;

public class TestStaticAnidadas {
    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 TestStaticAnidadas 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 Avanzados (Continuación)

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

```
package clase3;

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

Se define la interface Abreviable para cualquier objeto que pueda abreviarse

```
package clase3;

public enum UnidadesTemperatura implements Abreviable {
        GRADOCELSIUS("°C"),
        GRADOFARENHEIT("°F"),
        GRADOREAUMUR("°R"),
        GRADONEWTON("°N");

private String abrev;

UnidadesTemperatura(String abrev){
   this.abrev=abrev;
}

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

Cualquier método que acepte como parámetro un objeto **Abreviable** aceptará una instancia de **UnidadesTemperatura**, que es una enumerativo.

```
package clase3;

public class PruebaUnidadesTemperatura {

   public static void imprimirAbreviatura(Abreviable s){
       System.out.println(s.abrev());
   }

   public static void main(String args[]){

      for (UnidadesTemperatura u : UnidadesTemperatura.values())
          imprimirAbreviatura(u);
   }
}
```