TIPOS ENUMERADOS

Los tipos enumerados sirven para restringir el contenido de una variable a una serie de valores predefinidos. Esto suele ayudar a reducir los errores en nuestro código.

A partir de Java SE 5.0 se incluyó una modalidad de tipos enumerados que mantiene la seguridad de los tipos. En la práctica viene a ser como si definiéramos nuestros propios tipos de variables.

En Java, los tipos enumerados se pueden definir fuera o dentro de una clase. Otra ventaja que traen los tipos enum de Java es que al ser una “especie de clase” podemos añadirles métodos, variables de instancia, constructores, etc… lo que los hace muy potentes.

En las versiones anteriores a la versión 1.5 o 5 de Java no existían los tipos de datos enum con lo que debíamos usar constantes de la siguiente forma:

```
final String COLOR_ROJO = "rojo";
final String COLOR_VERDE = "verde";
final String COLOR_AZUL = "azul";
```

A partir de la versión 5 de Java se incorporaron al lenguaje los tipos de datos enumerados con el objetivo de mejorar varios aspectos sobre el uso de las constantes. Básicamente, un enum en Java es un conjunto fijo y relacionado de constantes y deberían usarse siempre que se necesite representar un conjunto de constantes con esas características.

Los enums se definen de la siguiente forma:

```
public enum Dia {
   LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO
}
public enum Color {
   ROJO("FF0000"), VERDE("00FF00"),  AZUL("0000FF");
   private String rgb;
   Color(String rgb) {
           this.rgb = rgb;
   }
   public String getRgb() {
           return rgb;
   }
}
```

No se deben poner métodos set porque si una variable enum utiliza el método set, el valor también se cambia en el enum, por lo tanto, no es conveniente ya que lo normal es que los atributos que se definan con el enum sean constantes.

Como son constantes, por las convenciones del lenguaje se escriben en mayúscula. Pero los enum son algo más que constantes, la clase del tipo del enum puede definir métodos y otras propiedades, como el método getRgb del ejemplo anterior o en el siguiente caso que se usa el método values() y que se añade automáticamente a todos los enums.

```
public class PruebaValues {
  public static void main(String[] args) {
          for (Dia d : Dia.values()) {
              System.out.printf("El día de la semana: %s%n", d);
          }

          for (Color c : Color.values()) {
              System.out.printf("El color %s tiene como RGB: %s%n", c, c.getRgb());
          }
  }
}
```

Algunas cosas que hacen de los enum muy interesantes es que al definir las constantes de este modo obtenemos «type safety». Si decimos que un método de una clase recibe un enum, el compilador comprobará en tiempo de compilación que cuando lo usamos le pasemos realmente un

valor de ese enum, cosa que con constantes definidas como int o String el compilador solo comprueba que pasemos un int o String y por tanto podremos pasar cualquier valor aunque no sea una de las constantes esperadas por el método. Otra de las ventajas que ya se ve en ejemplo anterior es que los enums pueden tener comportamiento a través de los métodos que defina. También, específicamente diseñadas para los enums existen las clases EnumSet y EnumMap que trabajan con enums de forma más eficiente que con Set y Map. Además, los enums pueden ser usados en expresiones switch cosa que con constantes de tipo String solo podemos hacer a partir de la versión 7 de Java.

Para comparar valores de enums podemos hacerlo con el operador == o con el método equals. Usar el operador == tiene la ventaja de evitar un posible NullPointerException. El compilador comprueba que se estén comparando dos valores del mismo enum, es decir, comprueba la compatibilidad de tipos en tiempo de compilación.

```
Dia dia = null;
if (dia == Dia.LUNES);      // se ejecuta
if (dia.equals(Dia.LUNES)); // lanza NullPointerException

if (Dia.LUNES==Color.ROJO) // no compila, incompatibilidad de tipos
```

Estos son motivos suficientes para usar enums en vez de constantes y == en vez de equals para los enums a partir de la versión 5 de Java.

Debido a que los tipos enumerados son como una clase de tipo especial en Java, hay muchas cosas que se pueden realizar dentro de un *enum*, además de declarar constantes, un tipo enumerado puede contener constructores, métodos y variables. No se puede invocar al constructor directamente, éste se invoca una vez que se crea el tipo enumerado y es definido por los argumentos utilizados para crearlo. Se puede definir más de un argumento en un constructor de un tipo enumerado.

Ejemplo:

```
public enum Vaso {
        JARRA(500,20.7f), TUBO(250,16.8f), TERCIO(333,12.3f), CAÑA(200,8.5f);
        private int cc;
        private float altura;
        Vaso(int cc,float altura) {
                this.cc=cc;
                this.altura=altura;
        }
        public int getCc() {
                return cc;
        }
        public float getAltura() {
                return altura;
        }
}
public class BebidaCerveza {
        enum MarcaCerveza { AMBAR, GUINNESS, HEINEKEN }
        private Vaso vaso;
        private MarcaCerveza marca;
        BebidaCerveza(MarcaCerveza marca, Vaso vaso) {
                this.marca=marca;
                this.vaso=vaso;
        }
        public void servir() {
                System.out.printf("Sirviendo %d cc. de cerveza %s en una %s de %.2f cm de altura",
                vaso.getCc(),marca.toString().toLowerCase(),vaso.toString().toLowerCase(),vaso.getAltura());
        }
}
public class PruebaEnum {
        public static void main(String[] args) {
                BebidaCerveza birra = new BebidaCerveza(BebidaCerveza.MarcaCerveza.AMBAR, Vaso.JARRA);
                birra.servir();
        }
}
```