DESARROLLO DE APLICACIONES WEB

Anexo I - Unidad 8

E n u m e r a c i o n e s

1r DAW
IES La Mola de Novelda
Departament d'informàtica

Professor: Joan Carbonell Picó

Índice

Anexo Unidad 8: Enumeraciones

1.- ENUMERADOS

1.1.-INTRODUCCIÓN

Versiones anteriores a JDK 5 carecían de una característica que muchos programadores consideraban necesaria: las enumeraciones. Éstas están presentes en la mayoría de los lenguajes común mente utilizados. A partir del JDK5, la enumeraciones fueron agregadas al lenguaje Java.

En su forma simple, una enumeración es una lista de constantes. En lenguajes como C++, las enumeraciones simplemente son listas de constantes de tipo entero. En Java, una enumeración define un tipo (una clase), esto expande enormemente el concepto de enumeración, ya que puede tener constructores, métodos y variables de instancia.

1.2.-FUNDAMENTOS

Una enumeración se crea utilizando la palabra clave ***enum***. Por ejemplo, ésta es una enumeración simple que lista algunas categorías de manzanas.

```
//Una enumeración de categorías de manzanas
enum Manzana {
        Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
```

Los identificadores Jonathan, GoldenDel y el resto, son llamados constantes de enumeración. Cada uno está implícitamente declarado como un miembro de tipo ***public***, ***static*** y ***final*** de la clase Manzana. Además, su tipo es el tipo de la enumeración en la cual fueron declarados, en este caso es Manzana.

Aunque las enumeraciones definen a una clase tipo, no se instancia un ***enum*** usando ***new***. En lugar de eso, se declara y usa una variable enumeración tal como se hace con los tipos primitivos. Ejemplos:

```
Manzana ap; // ap es una variable del tipo enumerado Manzana.
ap = Manzana.RedDel; // Asigna a la variable ap el valor RedDel.
if (ap == Manzana.GoldenDel) //Compara el valor de la variable ap con la constante GoldenDel
System.out.println(ap); //Imprime el valor de la variable ap
System.out.println(Manzana.Winesap); //Imprime la constante Winesap
```

Un valor de enumeración también puede ser utilizado para controlar una sentencia **switch**. Ejemplo:

```
//Usa una enumeración para controlar una sentencia switch
switch (ap) {
        case Jonathan; //Utilizamos los nombres de las constantes
        // ...
        case Winesap;
        // ...
...
```

1.3.-MÉTODOS VALUES( ) Y VALUESOF( )

Todas las enumeraciones automáticamente contienen dos métodos predefinidos: ***values( )*** y ***valueOf( )***. La sintaxis de estos métodos es la siguiente (*enum-type* es el tipo de enumeración.):

- public static enum-type[ ] values( ) → Devuelve un *array* que contiene una lista de constantes enumeradas.

- public static enum-type valueOf(String str) → Devuelve la constante enumerada cuyo valor corresponde a la cadena pasada en el parámetro *str*.

```
// Uso de los métodos predefinidos para las enumeraciones
// Una enumeración de tipos de manzana.
enum Manzana {
        Jonathan, GoldenDel, RedDel, Winesap, Cortland
}
class EnumDemo2 {
        public static void main(String args[]){
                Manzana ap;
                System.out.println("Estas son todas las constantes de tipo Manzana:");
                // usando el método values()
                Manzana allapples[] = Manzana.values();
                for(Manzana a : allapples)
                        System.out.println(a) ;
                System.out.println();
                // usando el método valueOf ()
                Scanner sc = new Scanner(System.in);
                System.out.print("Tipo de manzana: ");
                String tipoManzana = sc.nextLine(); //Introducimos Winesap
                ap = Manzana.valueOf (tipoManzana) ;
                System.out.println("ap contiene " + ap);
        }
}
```

La salida que produce el ejemplo es:

```
Estas son todas las constantes de tipo Manzana:
Jonathan
GoldenDel
RedDel
Winesap
Cortland

Tipo de manzana: Winesap
ap contiene Winesap
```

Podemos ver que el programa utiliza un ciclo al estilo ***for-each*** el cual itera a través del *array* de constantes obtenidas cuando se llama al método ***values()***. Para ilustrar esto, se creó la variable ***allapples*** y se le asignó una referencia a un *array* con los valores de la enumeración. Sin embargo, este paso no es necesario porque el ***for*** podría haber sido escrito como se muestra a continuación, eliminando la necesidad de la variable **allapples**:

```
for (Manzana a: Manzana.values())
    System.out.println(a);
```

2.- ENUMERACIONES COMO CLASE

Una enumeración de Java es un tipo de clase. Aunque no se instancia un ***enum*** utilizando ***new***, éstos tienen casi las mismas capacidades que las clases. Por ejemplo, se pueden tener constructores, agregar variables de instancia y métodos, e incluso implementar interfaces.

Cada constante de la enumeración es un objeto de su propio tipo enumerado. Así, cuando se define un constructor para un ***enum***, el constructor es llamado cuando cada constante de enumeración es creada. También, cada constante de enumeración tiene su propia copia de cualquier variable de instancia definida para la enumeración. Ejemplo:

```
enum Manzana {
  Jonathan(10), GoldenDel(9), RedDel(12), Winesap(15), Cortland(8);
  private int price; // precio de cada Manzana
  // constructor
  Manzana (int p) { price = p; }
  int getPrice () { return price; }
}

public class Enumeraciones {
  public static void main(String[] args) {
    Manzana ap;
    // mostrar el precio de Winesap
```

```
    System.out.println("Winesap cuesta " + Manzana.Winesap.getPrice() + " centavos.\n");
    // mostrar todos los tipos de manzana y su precio.
    System.out.println( "Todas las manzanas y sus precios: ");
    for(Manzana a : Manzana.values())
      System.out.println (a+ " cuesta " + a.getPrice() + " centavos.");
  }
}
```

La salida que produce el ejemplo es:

```
Winesap cuesta 15 centavos.

Todas las manzanas y sus precios:
Jonathan cuesta 10 centavos.
GoldenDel cuesta 9 centavos.
RedDel cuesta 12 centavos.
Winesap cuesta 15 centavos.
Cortland cuesta 8 centavos.
```

Al ejemplo podemos ver tres cosas:

- La variable de instancia precio, la cual es utilizada para almacenar el precio de cada tipo de manzana.
- El constructor Manzana, al cual se pasa el precio de cada manzana.
- El método getPrice( ), el cual regresa el valor del precio.

Cuando se declara la variable **ap** en *main( )*, el constructor Manzana es llamado una vez para cada constante especificada. Los argumentos para el constructor son especificados dentro del paréntesis al lado de cada constante: Jonathan(10), GoldenDel(9), RedDel(12), Winesap(15), Cortland(8);.

Estos valores son pasados al parámetro **p** de Manzana( ), el cual asigna el valor a la variable precio. El constructor es llamado una vez para cada constante. Dado que cada constante en la enumeración tiene su propia copia de la variable precio, es posible obtener el precio de un tipo específico de manzana llamando al método getPrice( ): Manzana.Winesap.getPrice()

El precio para cada una de la variedades es obtenido utilizando un ciclo a través de la enumeración con un ciclo ***for***. Debido a que hay una copia de precio para cada constante en la enumeración, el valor asociado con una constante es independiente del valor asociado con otra.

Una enumeración puede tener dos o más constructores sobrecargados, tal como las otras clases lo pueden hacer. Ejemplo:

```
// Uso de constructores en enumeraciones
enum Manzana {
        Jonathan(l0), GoldenDel (9), RedDel, Winesap (15), Cortland (8) ;
        private int price; // precio de cada manzana
        // constructor
        Manzana (int p) { price = p; }
        // constructor sobrecargado
        Manzana () { price = -1; }
        int getPrice () { return price; }
}
```

En este caso, para RedDel no se proporcionan argumentos. Esto significa que el constructor por omisión es llamado, y la variable precio para RedDel tendrá el valor –1.

En las enumeraciones existen dos restricciones: no puede heredar de otra clase y no puede ser una superclase. Por lo demás, una enumeración actúa de igual forma que cualquier otro tipo de clase. La clave es recordar que cada constante en la enumeración es un objeto de la clase en la cual está definida.

3.- HERENCIA DE LA CLASE ENUM

Todas las enumeraciones automáticamente heredan de: java.lang.Enum. Esta clase define varios métodos que están disponibles para el uso de todas las enumeraciones (en la API de Java los podemos consultar). Veamos tres de estos métodos:

- **public final int ordinal()** → Obtiene la posición de una constante en la enumeración, es decir, su valor ordinal. En la enumeración Manzana, Jonathan tiene un valor ordinal cero, GoldenDel tiene un valor ordinal 1, RedDel tiene un valor ordinal 2, y así sucesivamente.

- **final int compareTo(tipoEnum e)** → Compara los valores ordinales de dos constantes de la misma enumeración. Donde ***tipoEnum*** es el tipo de la enumeración, y ***e*** es la constante a comparar con la constante que invoca al método. La constante que invoca y la constante **e** deben ser del mismo tipo de enumeración. Si la constante que invoca tiene un valor ordinal menor que ***e***, entonces **compareTo( )** devuelve un valor negativo. Si los dos valores ordinales son iguales, entonces se devuelve cero. Si la constante que invoca tiene un valor mayor que ***e***, entonces se devuelve un valor positivo.

- **public final boolean equals(Object other)** → Compara la igualdad de una constante de enumeración con cualquier otro objeto. Esos dos objetos serán iguales sólo si ambos hacen referencia a la misma constante, dentro de la misma enumeración. El simple hecho de tener

valores ordinales en común no causará que **equals( )** regrese el valor de verdad si las dos constantes son de diferentes enumeraciones.

Ejemplo de los tres métodos:

```
enum Manzana {
  Jonathan(10), GoldenDel(9), RedDel(12), Winesap(15), Cortland(8);
  private int price; // precio de cada Manzana
  // constructor
  Manzana (int p) { price = p; }
  int getPrice () { return price; }
}

public class Enumeraciones {
  public static void main(String[] args) {
    Manzana ap, ap2, ap3;
  // Obtener todos los valores ordinales utilizando el método ordinal().
    System.out.println("Estas son todas las constantes manzana" + " y sus valores ordinales: ");
    for(Manzana a : Manzana.values())
      System.out.println(a+ " " + a.ordinal());
    ap = Manzana.RedDel;
    ap2 = Manzana.GoldenDel;
    ap3 = Manzana.RedDel;
    // uso de los métodos compareTo() y equals()
    if(ap.compareTo(ap2) < 0)
      System.out.println(ap + " va antes de " + ap2);
    if(ap.compareTo(ap2) > 0)
      System.out.println(ap2 + " va antes de " + ap);
    if(ap.compareTo(ap3) == 0)
      System.out.println(ap + " es igual a " + ap3);
    System.out.println() ;
    if(ap.equals(ap2))
      System.out.println("¡Error!") ;
    if(ap.equals(ap3))
      System.out.println(ap + " es igual a " + ap3);
    if (ap == ap3)
      System.out.println(ap + " == " + ap3);
  }
}
```

La salida que produce el ejemplo es:

Aquestes són totes les constants poma i els seus valors ordinals:
Jonathan 0
GoldenDel 1
RedDel 2
Winesap 3
Cortland 4
GoldenDel va abans de RedDel
RedDel es igual a RedDel

RedDel es igual a RedDel
RedDel = = RedDel