**ENUMERATED TYPES**

* Un tipo dato ***enumerated*** consiste en un conjunto de valores predefinidos.
* Podemos usar el tipo de dato para crear variables que pueden contener únicamente los valores que pertenecen al tipo de dato ***enumerated***
* Algunas veces es de gran ayuda crear nuestros propios tipos de datos que tengan un conjunto específico de valores legales.
* Por ejemplo, supongamos que queremos crear un tipo de dato llamado ***Dia*** y los valores legales en este tipo de dato fueran los nombres de los días de la semana (Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo)
* Cuando creemos una variable del tipo ***Dia***, podremos almacenar únicamente los nombres de los días de la semana en esa variable.
* Cualquier otro valor sería ilegal.
* En java, dicho tipo es conocido como ***tipo de dato enumerado*** *(enumerated data type)*
* Se utiliza la palabra ***enum*** para crear nuestro propio tipo de dato y especificamos los valores que pertenecerán a ese tipo. Aquí un ejemplo:

|  |
| --- |
|  |

* El ejemplo crea un tipo de dato enumerado llamado **Dia**.
* Los identificadores Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo son conocidos como constantes enum.
* Estas constantes representan los valores que perteneces al tipo de dato **Dia**.
* Las constantes enum no están encerradas entre comillas, por lo tanto, no son Strings.
* Las constantes enum deben ser identificadores o nombres legales.
* Cuando creamos los nombres para las constantes enum, no es requerido que sea escrita toda la palabra en mayúscula.
* Pudimos haber escrito las constantes enum como lunes, Martes, miércoles, Jueves, etc.
* Sin embargo, ya que representan valores constantes, el estándar es escribirlas en mayúsculas.
* Una vez que hemos creado nuestro tipo de dato enumerado en nuestro programa, podemos declarar variables de ese tipo, por ejemplo:

|  |
| --- |
|  |

* Ya que **diaTrabajo** es una variable de tipo **Dia**, los únicos valores legales que podemos asignarle son las constantes enum:
  + Dia.LUNES
  + Dia.MARTES
  + Dia.MIERCOLES
  + Dia.JUEVES
  + Dia.VIERNES
  + Dia.SABADO
  + Dia.DOMINGO

Si intentamos asignar cualquier otro valor, un error de compilación ocurrirá.

* Notamos que la forma de asignar es Dia.MIERCOLES en lugar de MIERCOLES. La mayoría de las veces se debe escribir el nombre completo perteneciente a una constante enum.

**ENUMERATED TYPES ARE SPECIALIZED CLASSES**

* Cuando escribimos la declaración de un tipo de dato enumerado, estamos creando un tipo de clase.
* Además, las constantes enum que agregamos entre las llaves son objetos de esa clase.
* En el ejemplo anterior, **Dia** es una clase, y las constantes enum **Dia.LUNES, Dia.MARTES…** etc son instancias de la clase **Dia**
* Cuando asignamos **Dia.MIERCOLES** a la variable **diaTrabajo**, estamos asignando la dirección del objeto **Dia.MIERCOLES** a la variable **diaTrabajo**.

|  |
| --- |
|  |

* Las constantes enum, los cuales son objetos, vienen equipados con algunos métodos. Uno de ellos es el método toString, el cual retorna el nombre en forma de String de la constante enum.
* El siguiente ejemplo muestra el String MIERCOLES.
* El método toString es llamado implícitamente cuando un objeto es pasado a System.out.println.

|  |
| --- |
|  |

* Las constantes enum tienen un método llamado **ordinal.** Dicho método retorna un valor entero el cual representa el valor ordinal de la constante.
* El valor ordinal de la constante es la posición de la declaración enum. La primera constante tiene la posición 0.

|  |
| --- |
|  |

* A continuación, un ejemplo:

|  |  |
| --- | --- |
|  |  |

* Los últimos métodos de los tipos de datos enumerados a discutir son el **equals** y **compareTo**.
* El método **equals** acepta un objeto en su argumento y retorna **true** si ese objeto es igual a la constante enum que hace la llamada.
* Por ejemplo, el siguiente código es verdadero:

|  |
| --- |
|  |

* El método **compareTo** está diseñado para comparar constantes enum del mismo tipo. Acepta un objeto como argumento y retorna uno de los posibles siguientes valores:

1. Un **entero negativo** si el valor ordinal de la constante enum que hace la llamada es menor que el valor ordinal de la constante enum que se pasa como argumento.

|  |
| --- |
|  |

1. **Cero** si la constante enum que hace la llamada es igual a la constante enum que se pasa como argumento.

|  |
| --- |
|  |

1. Un **entero positivo** si el valor ordinal de la constante enum es mayor que el valor ordinal de la constante enum que es pasada como argumento.

|  |
| --- |
|  |

* En el siguiente ejemplo se muestra que el viernes es mayor que lunes.

|  |
| --- |
|  |

* A continuación, se muestra un ejemplo completo:

|  |  |
| --- | --- |
|  |  |
|  | |
|  | |
|  | |
| **Salida:** | |
| Carro{tipo=PORSCHE, color=AZUL, precio=1500000.0} | |

SWITCHING ON ENUMERATED TYPES

* Java permite utilizar constantes enum en la sentencia switch. Veamos el siguiente ejemplo:

|  |
| --- |
|  |

* La sentencia switch tiene la sección default, aunque tiene una sentencia para cada constante enum correspondiente al tipo de carro. Esto manejara situaciones en la que más constantes enum sean agregadas después a la enumeración **CarroTipo.** Este tipo de planeación es un ejemplo de “programación defensiva” (“Defensive programming”)

Fuente:

2016 – Starting out with Java – Tonny Gaddis (Tema 8.9 página 533)