ENUMERADOS EN JAVA

Ejercicio 1 Este código nos puede dar un error. Se propone que compruebes y expliques el error y encuentres una solución basada en la excepción que se produce.

```
public class PruebaValueOf {

    enum Animal {
        PERRO, GATO
    };

    public static void main(String[] args) {

        Animal animal = Animal.valueOf(args[0]);

        switch (animal) {
         case PERRO:
            System.out.println("El perro ladra.");
            break;
         case GATO:
            System.out.println("El gato maulla.");
            break;
         default:
            System.out.println("No es un tipo aceptado.");
         }
     }
}
```

Ejercicio 2 sobre una clase Enum. Los futbolistas están caracterizados por una demarcación a la hora de jugar un partido de fútbol, por tanto las demarcaciones en las que puede jugar un futbolista son finitas y por tanto se pueden enumerar en: Portero, Defensa, Centrocampista y Delantero. Con esta especificación podemos crearnos la siguiente clase "Enum" llamada "Demarcación":

```
public enum Demarcacion
{
    PORTERO, DEFENSA, CENTROCAMPISTA, DELANTERO
}
```

Por convenio los nombres de los enumerados se escriben en mayúsculas.

Es muy importante entender que un "***Enum***" en java ***es realmente una clase*** (cuyos objetos solo pueden ser los definidos en esta clase: PORTERO,…, DELANTERO) ***que hereda*** de la clase "***Enum (java.lang.Enum)***" y por tanto los enumerados tienen una serie de métodos heredados de esa clase padre. A continuación, vamos a mostrar algunos de los métodos más utilizados de los enumerados:

```
public enum Demarcacion{PORTERO, DEFENSA, CENTROCAMPISTA, DELANTERO}

// Instancia de un enum de la clase Demarcación.
Demarcacion delantero = Demarcacion.DELANTERO;

// Devuelve un String con el nombre del enumerado (DELANTERO).
delantero.name();
```

```
// Devuelve un String con el nombre del enumerado (DELANTERO).
delantero.toString();

// Devuelve un entero con la posición del enum según está declarada
(3).
delantero.ordinal();

// Compara el enum que invoca el método con el enum parámetro, según
el orden en el que están declarados los enum.
delantero.compareTo(Enum otro);

// Devuelve un array que contiene todos los enum.
Demarcacion.values();


// Devuelve el enumerado que corresponde con la cadena pasada por
// parámetro
System.out.println(Demarcacion.valueOf("DELANTERO"));
```

Ejemplo para probar:

```
Demarcacion delantero = Demarcacion.DELANTERO;
Demarcacion defensa = Demarcacion.DEFENSA;

// Devuelve un String con el nombre de la constante
System.out.println("delantero.name()= " + delantero.name());
System.out.println("defensa.toString()= " + defensa.toString());

// Devuelve un entero con la posición de la constante según está
declarada.
System.out.println("delantero.ordinal()= " + delantero.ordinal());

// Compara el enum con el parámetro según el orden en el que están //
declaradas las constantes.
System.out.println("delantero.compareTo(portero)= " +
delantero.compareTo(defensa));
System.out.println("delantero.compareTo(delantero)= " +
delantero.compareTo(delantero));

// Recorre todas las constantes de la enumeración
for(Demarcacion d: Demarcacion.values()){
        System.out.println(d.toString()+" - ");
}
```

Tenemos como salida los siguientes resultados:

```
delantero.name()= DELANTERO
defensa.toString()= DEFENSA
delantero.ordinal()= 3
delantero.compareTo(defensa)= 2
delantero.compareTo(delantero)= 0
PORTERO - DEFENSA - CENTROCAMPISTA - DELANTERO
```

Como ya se ha dicho, un **enum** es una clase especial que **limita la creación de objetos a los especificados en su clase** (por eso, su constructor es privado, como se ve en el siguiente fragmento de código); pero estos objetos pueden tener atributos como cualquier otra clase. En la siguiente declaración de la clase, vemos un ejemplo en el que

definimos un enumerado "Equipo" que va a tener dos atributos; el nombre y el puesto en el que quedaron en la liga del año 2009/2010.

```
public enum Equipo
{
        BARÇA("FC Barcelona",1), REAL_MADRID("Real Madrid",2),
        SEVILLA("Sevilla FC",4), VILLAREAL("Villareal",7);

        private String nombreClub;
        private int puestoLiga;

        private Equipo (String nombreClub, int puestoLiga){
                this.nombreClub = nombreClub;
                this.puestoLiga = puestoLiga;
        }

        public String getNombreClub() {
                return nombreClub;
        }

        public int getPuestoLiga() {
                return puestoLiga;
        }
}
```

Como se ve BARCA, REAL_MADRID, etc. son el nombre del enumerado (u objetos de la clase Equipo) que tendrán como atributos el "nombreClub" y "puestoLiga". Como se ve en la clase ***definimos un constructor que es privado*** (es decir que solo es visible dentro de la clase Equipo) y solo definimos los métodos "get". Para trabajar con los atributos de estos enumerados se hace de la misma manera que con cualquier otro objeto; se instancia un objeto y se accede a los atributos con los métodos get. En el siguiente fragmento de código vamos a ver cómo trabajar con enumerados que tienen atributos:

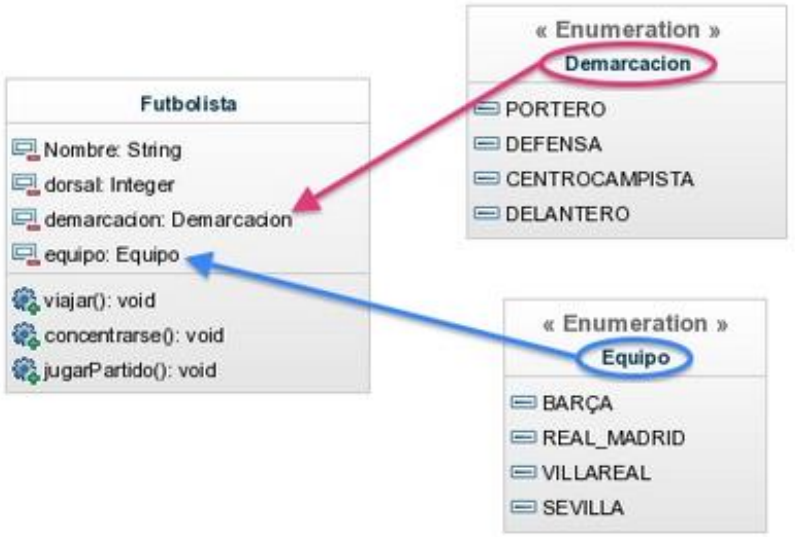
```
// Instanciamos el enumerado
Equipo villareal = Equipo.VILLAREAL;

// Devuelve un String con el nombre de la constante
System.out.println("villareal.name()= "+villareal.name());

// Devuelve el contenido de los atributos
System.out.println("villareal.getNombreClub()="
+villareal.getNombreClub());
System.out.println("villareal.getPuestoLiga()="
+villareal.getPuestoLiga());
```

Como salida de este fragmento de código tenemos lo siguiente:

```
villareal.name()= VILLAREAL
villareal.getNombreClub()= Villareal
villareal.getPuestoLiga()= 7
```

Es muy importante tener claro que los enumerados no son Strings (aunque pueden serlo), sino que son objetos de una clase que solo son instanciables desde la clase que se implementa y que no se puede crear un objeto de esa clase desde cualquier otro lado que no sea dentro de esa clase. Es muy común (sobre todo cuando se está aprendiendo qué son los enumerados) que se interprete que un enumerado es una lista finita de Strings y en realidad es una lista finita de **objetos** de una determinada clase con sus atributos, constructor y métodos getter aunque estos sean privados.

A continuación, vamos a poner un sencillo ejemplo en el que vamos a mezclar los dos enumerados anteriores (Demarcación y Equipo). En este ejemplo, vamos a crear unos objetos de la clase **Futbolista**, que representarán a los jugadores de la selección española de fútbol que ganaron el mundial de fútbol de Sudáfrica en el año 2010. Esta clase va a caracterizar a los futbolistas por su nombre, su dorsal, la demarcación en la que juegan y el club de fútbol al que pertenecen; tal y como vemos en el siguiente diagrama de clases.



Como vemos, los atributos de demarcación y equipo son de la clase **Demarcacion** y **Equipo** respectivamente y son los enumerados vistos anteriormente; por tanto un futbolista solo podrá pertenecer a uno de los cuatro equipos que forman el enumerado "Equipo" y podrá jugar en alguna de las cuatro demarcaciones que forman el enumerado "Demarcación". A continuación, se muestra la implementación de la clase "Futbolista":

```
package Main;

public class Futbolista {

        private int dorsal;
        private String Nombre;
        private Demarcacion demarcacion;
        private Equipo equipo;

        public Futbolista() {
        }
```

```
        public Futbolista(String nombre, int dorsal,
        Demarcacion demarcacion, Equipo equipo) {
                this.dorsal = dorsal;
                Nombre = nombre;
                this.demarcacion = demarcacion;
                this.equipo = equipo;
        }

        // Metodos getter y setter

        ..............................


        @Override
        public String toString() {
                return this.dorsal + " - " + this.Nombre + " - "
                                + this.demarcacion.name()
                    + " - " + this.equipo.getNombreClub();
        }
}
```

Dada esta clase podemos crearnos ya objetos de la clase futbolista, como mostramos a continuación:

```
Futbolista casillas = new Futbolista("Casillas", 1,
Demarcacion.PORTERO, Equipo.REAL_MADRID);

Futbolista capdevila = new Futbolista("Capdevila", 11,
Demarcacion.DEFENSA, Equipo.VILLAREAL);

Futbolista iniesta = new Futbolista("Iniesta", 6,
Demarcacion.CENTROCAMPISTA, Equipo.BARCA);

Futbolista navas = new Futbolista("Navas", 22, Demarcacion.DELANTERO,
Equipo.SEVILLA);
```

Como vemos, la demarcación y el equipo al que pertenecen solo pueden ser los declarados en la clase enumerado. Si llamamos al método "toString()" declarado en la clase futbolista, podemos imprimir por pantalla los datos de los futbolistas:

```
System.out.println(casillas);
System.out.println(capdevila);
System.out.println(iniesta);
System.out.println(navas);
```

Y dado el siguiente método "toString()"

```
@Override
public String toString() {
        return this.dorsal + " - " + this.Nombre + " - "
                        + this.demarcacion.name() + " - " +
                        this.equipo.getNombreClub();
}
```

Tenemos como salida lo siguiente:

```
1 - Casillas - PORTERO - Real Madrid
11 - Capdevila - DEFENSA - Villareal
6 - Iniesta - CENTROCAMPISTA - FC Barcelona
22 - Navas - DELANTERO - Sevilla FC
```

En **resumen**: Un enumerado está formado por objetos definidos en la misma clase con constructor privado y si tiene atributos, estos solo tienen que tener métodos "getter" para obtener el valor del atributo.