una visión simple de las enumeraciones java

No tienen que ver con los arrays, se estudian en este bloque porque serán utilizadas en ejemplos posteriores.

Las enumeraciones son un recurso Java que no es imprescindible pero que muchos programadores las usan y por tanto para entender los ejemplos de la web que se consultan constantemente debemos conocer un mínimo este recurso. Se usan enumeraciones en prácticamente todos los lenguajes de programación modernos.

Las enumeraciones en java tiene un uso sencillo y un uso avanzado. Para el uso avanzado hay que entender más profundamente lo que es una enumeración y es tener claros los conceptos de composición, herencia y otros. En este boletín nos dedicamos a la visión simple.

Para empezar, los términos enumeración y tipo enumerado son equivalentes

¿Qué es una enumeración?(visión simple)
Un tipo especial de clase con las siguientes características sintácticas básicas:

- En lugar de la palabra class se utiliza enum
- Entre las llaves se "lista" o "enumera" un conjunto de constantes separadas por comas.
- una variable de tipo enumeración puede coger valores de las constantes de la enumeración

```
enum Color{   //enum en lugar de class
   RED, GREEN, BLUE;
}

public class Unidad4{
   public static void main(String[] args){
      Color c1 = Color.RED; //c1 es una variable que toma el valor RED
      System.out.println(c1);
      System.out.println(Color.RED);
   }
}
```

En realidad c1 no es más que una variable referencia a un objeto "constante" creado por el compilador Java.
Observa como el println responde con RED, es decir con el nombre de la constante

las enumeraciones se pueden definir fuera de una clase o bien anidada dentro de otra clases

```
public class Unidad4 {

   enum Color {
      RED, GREEN, BLUE;
   }

   public static void main(String[] args) {
      Color c1 = Color.RED;
      System.out.println(c1);
   }
}
```

Más adelante entenderemos como funciona la anidación de clases.

otro ejemplo:

```
enum Color {
```

```
    ROJO, VERDE, AMARILLO;
  }
class Coche{
  String modelo;
  Color color;

  public Coche(String modelo, Color color) {
    this.modelo = modelo;
    this.color = color;
  }

}
public class Unidad4 {
  public static void main(String[] args) {
    Coche coche1= new Coche("Seat Paliza",Color.AMARILLO);
    System.out.println(coche1.modelo+" "+ coche1.color);
  }
}
```

Ejemplo: enumeraciones y sentencia IF

```
enum Color {
    ROJO, VERDE, AMARILLO;
  }
class Coche{
  String modelo;
  Color color;

  public Coche(String modelo, Color color) {
    this.modelo = modelo;
    this.color = color;
  }

}
public class Unidad4 {
  public static void main(String[] args) {
    Coche coche1= new Coche("Seat Paliza",Color.AMARILLO);
    if(coche1.color==Color.ROJO){
      System.out.println("que feo");
    }else if(coche1.color==Color.VERDE){
      System.out.println("no está mal");
    }else{
      System.out.println("que bonito");
    }

  }
}
```

Ejemplo: enumeraciones y sentencia switch

observa la diferencia sintáctica respecto al if: en el case ponemos solo la constante sin precederla del nombre de la enumeración (ponemos ROJO no Color.ROJO)

```
enum Color {
  ROJO, VERDE, AMARILLO;
}

class Coche {

  String modelo;
  Color color;

  public Coche(String modelo, Color color) {
    this.modelo = modelo;
    this.color = color;
  }

}

public class Unidad4 {

  public static void main(String[] args) {
    Coche coche1 = new Coche("Seat Paliza", Color.AMARILLO);
    switch (coche1.color) {
      case ROJO:
        System.out.println("que feo");
```

```
            break;
         case VERDE:
            System.out.println(" no está mal");
            break;
         default:
            System.out.println("que bonito");
      }

   }
}
```

por supuesto podemos usar en el switch el operador ->

```
switch (coche1.color) {
         case ROJO -> System.out.println("que feo");
         case VERDE-> System.out.println(" no está mal");
         default-> System.out.println("que bonito");
}
```

Ejemplo: iterar sobre los valores de una enumeración

Utilizando el for mejorado y lo que devuelve el método predefinido values()

```
enum Color {
   ROJO, VERDE, AMARILLO;
}

public class Unidad4 {

   public static void main(String[] args) {
      System.out.println("colores disponibles");

      for (Color c : Color.values()) {
         System.out.println(c);
      }

   }
}
```

EJERCICIO U3_B12_E1:

El siguiente código crea una baraja utilizando el String palo. Reforma el código para que haga lo mismo pero utilizando una enumeración Palo.

```
class Carta {

   private String palo;
   private int numero;

   public Carta(String palo, int numero) {
      this.palo = palo;
      this.numero = numero;
   }

   public String toString() {
      return "(" + palo + ", " + numero + ')';
   }

}


class Baraja {

   //baraja española de 40 cartas. No hay 8 y 9.
   final int NUM_CARTAS = 40;
   Carta[] cartas = new Carta[NUM_CARTAS];


   public Baraja() {
      //crea una baraja ordenada por palos y números
      String[] palos = {"Bastos", "Copas", "Oros", "Espadas"};
   int ultimaCarta=0;
      for (int i = 0; i < 4; i++) {
         for (int j = 0; j < 12; j++) {
```

```
            if(j==7 || j==8){
          continue;
        }
      cartas[ultimaCarta] = new Carta(palos[i], j+1);
      ultimaCarta++;
          }
        }
    }

 }


public class Unidad4{
    public static void main(String[] args) {
        Baraja baraja= new Baraja();
        for(Carta c:baraja.cartas){
            System.out.println(c);
        }
    }
}
```