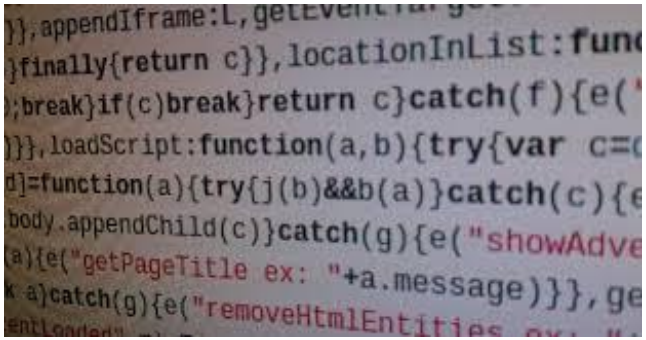


UT 11

UTILIZACIÓN AVANZADA DE CLASES. Segunda Parte

IES JUAN DE LA CIERVA
DPTO. INFORMÁTICA

CONTENIDOS DE LA PRESENTACIÓN

Objetivos

En esta unidad, se pretende que el alumno sepa manejar las relaciones entre clases, la herencia, polimorfismos, siendo capaz de crear sus propias clases en función de la necesidad de cada momento.

Contenidos

- Herencia.
- Superclases y subclases.
- Constructores y herencia.
- Clases y métodos: abstractos y finales.
- Polimorfismos.

Polimorfismo y Enlace Dinámico

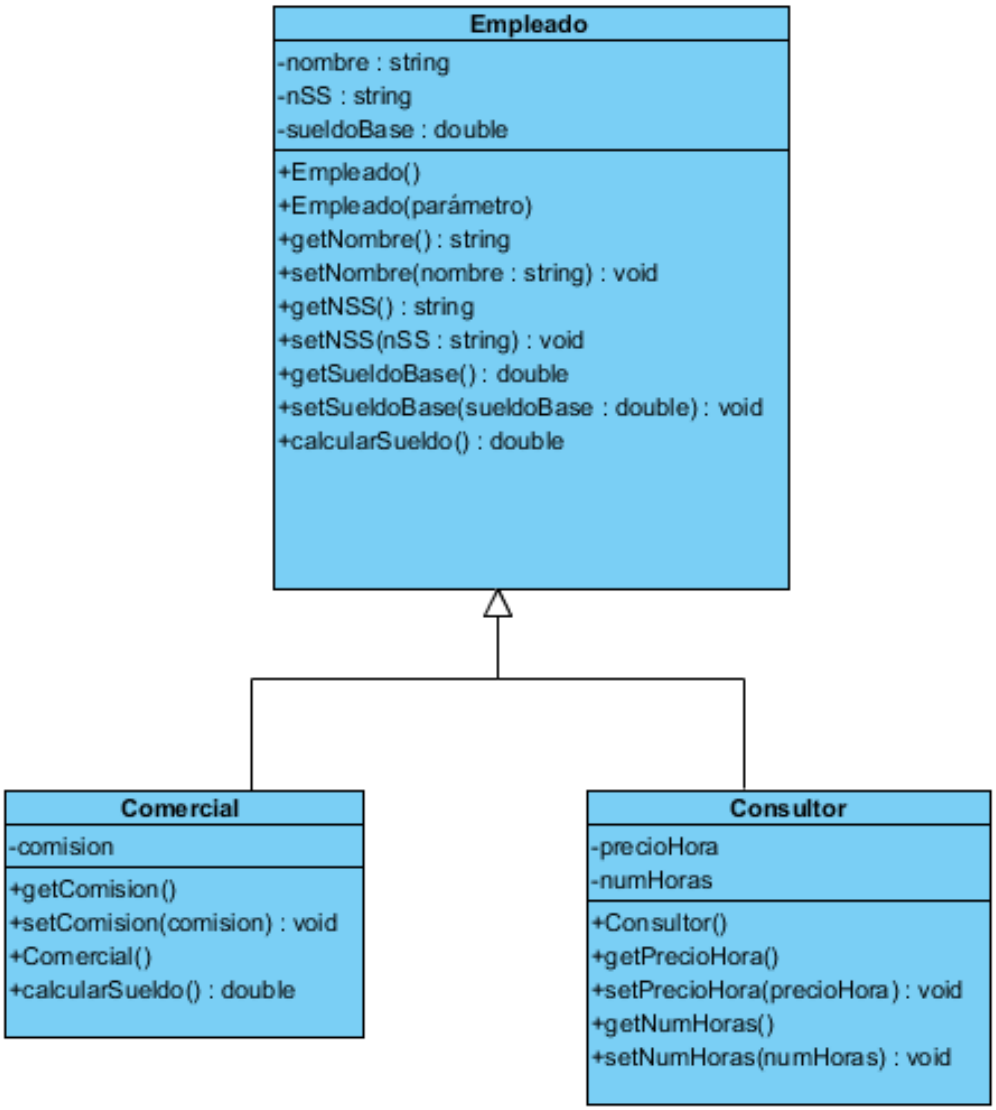
POLIMORFISMO y ENLACE DINÁMICO

- [RAE 2001]: Cualidad de lo que tiene o puede tener distintas formas
- El polimorfismo en POO se da por el uso de la herencia
- El polimorfismo es la propiedad de los objetos, que les permite adoptar diferentes formas en tiempo de ejecución, así como que un objeto de una clase derivada sea utilizado como un objeto de la clase base.
- Debido a esta propiedad y al principio de sustitución, se puede utilizar una objeto de la subclase siempre que el programa espere un objeto de la superclase.

POLIMORFISMO y ENLACE DINÁMICO

- Se produce por distintas implementaciones de los métodos definidos en la clase padre (**sobreescribir):**
  - Distinta implementación entre clase hija y padre
  - Distinta implementación entre clases hija
- Una misma llamada ejecuta distintas sentencias dependiendo de la clase a la que pertenezca el objeto.
- El código a ejecutar se determina en tiempo de ejecución => **Enlace dinámico**

POLIMORFISMO y ENLACE DINÁMICO



Ejemplo3_Polimormismo

POLIMORFISMO: Ejercicio

Práctica 11_4:

- Probar el funcionamiento del Polimorfismo en las clases de la Práctica 11_2.

- Añade a las clases Aire, Agua, Nitrógeno un método al que llamarás descripción(), diferente para cada una de las clases.

- Programa un vector de objetos Congelados de tamaño 5 al que llamarás vectorCongelados[ ] y en el que cargarás los 5 objetos creados en la práctica 11_2 (2 congelados por aire, 2 por agua, 1 por nitrógeno).

Modificador Final

Modificador final

- Para una clase, **`final`** significa que la clase no puede extenderse
- Para un método, **`final`** significa que no puede redefinirse en una clase derivada
- Para un dato miembro, **`final`** significa también que no puede ser redefinido en una clase derivada, como para los métodos, pero además significa que su valor no puede ser cambiado en ningún sitio; es decir el modificador **`final`** sirve también para definir valores constantes.

Métodos y clases abstractas

Métodos y clases abstractas

Imaginemos un escenario en el que no tenga sentido crear cierto tipo de objetos de una clase o se quiere evitar su creación, pero la clase es útil para generalizar miembros de otras clases que heredan de ella.

En este caso nos interesará declarar esa clase como **Clase Abstracta**.

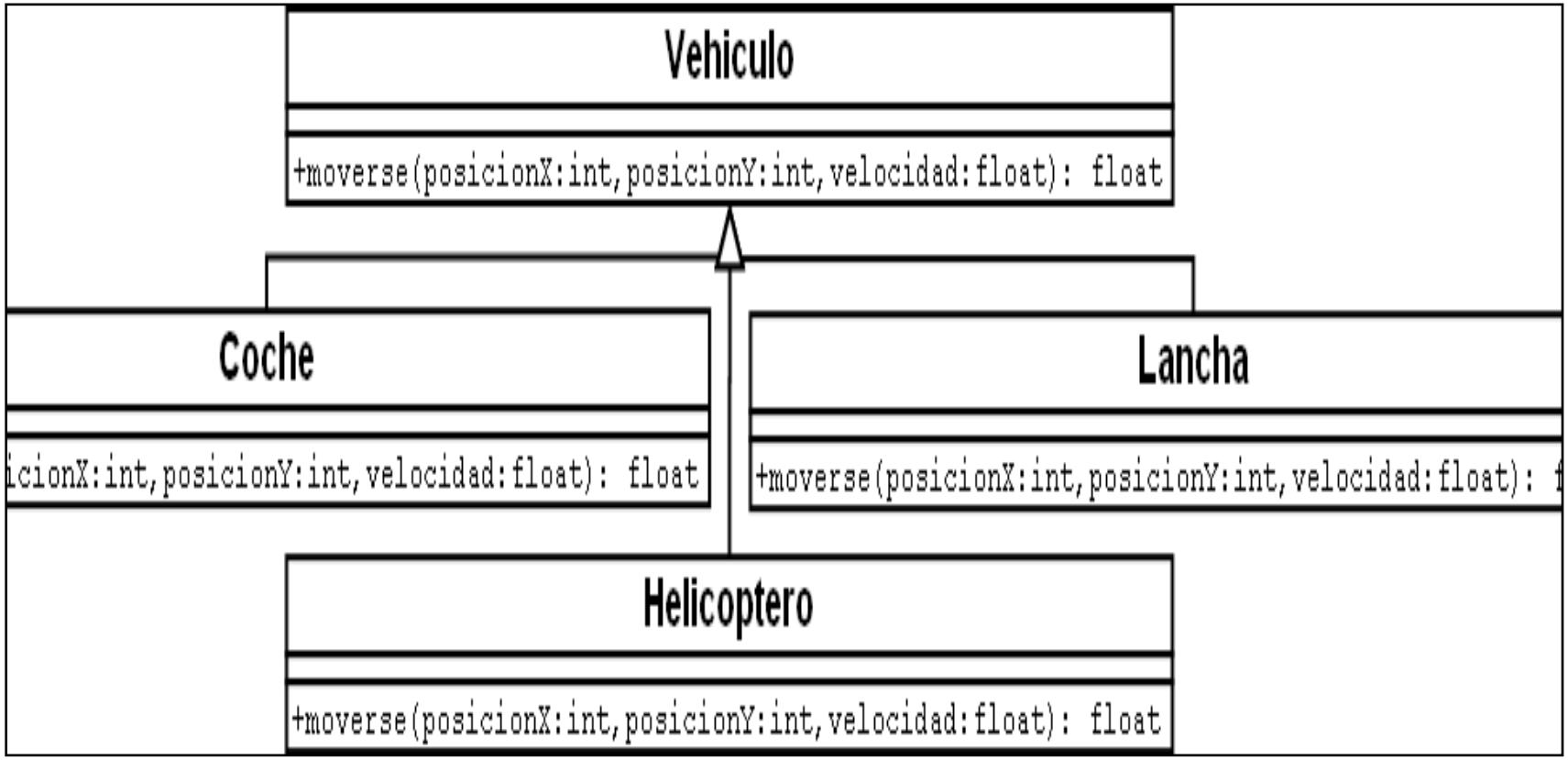
Métodos y clases abstractas

- Una clase abstracta según hemos dicho ***no puede instanciarse***, por lo que ***no tiene sentido que tenga constructores***.
- Una clase abstracta ***sí puede tener herencia***.
- Una clase abstracta ***puede tener atributos y métodos***.
- Los ***métodos*** de una clase abstracta pueden ***ser abstractos o normales***. Método abstracto es aquel que no tiene cuerpo.

Métodos y clases abstractas

- Para que una clase se declare como abstracta tenemos que usar la palabra clave **abstract**
- Las clases que heredan de una clase abstract tienen la obligación de implementar los métodos abstractos de la misma.
- Si la clases hija no queremos que implemente este método, la misma también debería ser abstract.
- Un método abstract no puede ser static.

Métodos y clases abstractas



Métodos y clases abstractas

Coche
#potencia: int #litros: int #diesel: boolean #gasolina: boolean
+moverse(posicionX:int,posicionY:int,velocidad:float): float

Helicoptero
#potencia: int #litros: int #diesel: boolean #gasolina: boolean #inclinacion: float
+moverse(posicionX:int,posicionY:int,velocidad:float): float

Lancha
#potenciaCadaMotor: int #litros: int #diesel: boolean #gasolina: boolean #numMotoresMax: int #numMotores: int
+moverse(posicionX:int,posicionY:int,velocidad:float): float

Métodos y clases abstractas

Si creamos una superclase abstracta vehículo con un método abstracto moverse():

- No podremos crear objetos Vehículo
- Deberemos sobreescribir el/los métodos abstractos de Vehículo en las subclases.

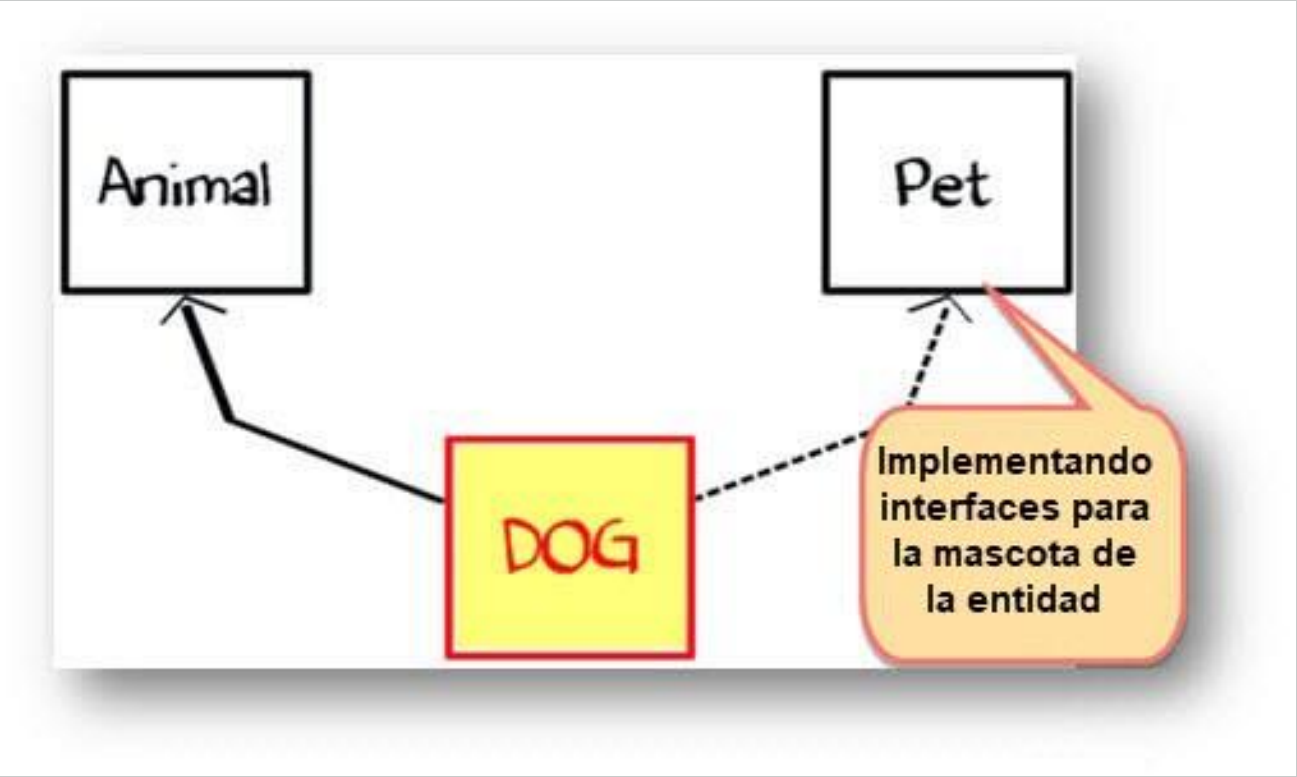
Métodos y clases abstractas: Ejemplo

```
public class Vehiculo{
    /* ... */

    Vehiculo(){
        /* ... */
    }

    /*...*/
    public abstract float moverse(int posX, int posY, float velocidad);

    public float mueveADestino(Destino destino){
        int i;
        Ruta ruta= new Ruta(destino);
        float recorrido=0;

        for(i=0;i<ruta.Length();i++){
            moverse(ruta[i].posicionX,ruta[i].posicionY);
        }

        return recorrido;
    }
}
```

Métodos y clases abstractas: Ejercicio

A partir de la jerarquía de clases programada en la práctica 11_4 de la Empresa Alimentaria, vamos a practicar el uso de una clase abstracta.

1. Elimina (o comenta) los métodos descripcion() de la clase Congelado y de sus subclases.
2. Crea un nuevo método abstracto descripcion() en la clase Congelado.
3. Resuelve los problemas que esto te genera en el programa en ese momento y haz una lista con la explicación de cada uno y cómo lo has solucionado.


Animal
Pet
DOG
Implementando interfaces para la mascota de la entidad

Interfaces

Interfaces

- Definición informal:

  Es una clase abstracta sin variables de instancia en donde todos sus métodos son abstractos (deprecated a partir de la versión 8 de Java)

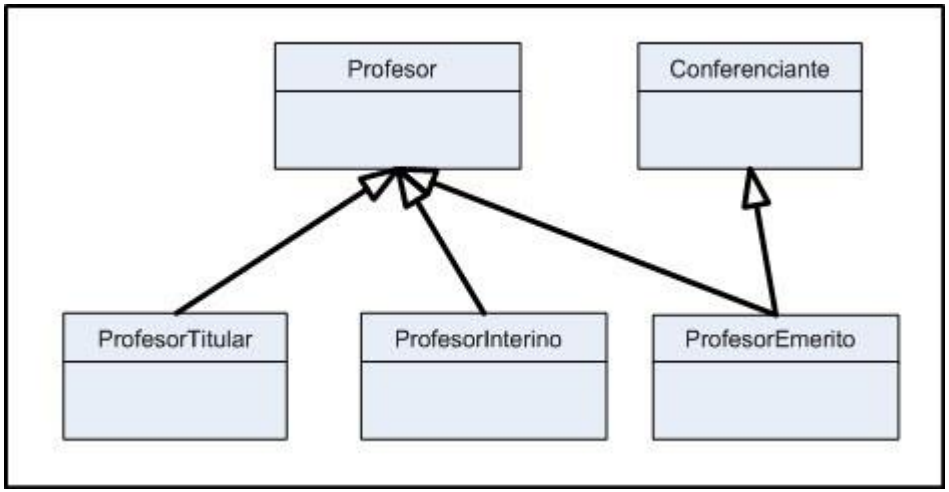
- Definición formal:

  Es un contrato que establece una clase en el cual esta clase asegura que implementará un conjunto de métodos

Interfaces

- Para que una clase use un interfaz debe implementar todos sus métodos e indicar que está usando(implementando) un interfaz.

(*)Una clase puede implementar varias interfaces.



Interfaces

- Contrato de compromiso.
- Conjunto de operaciones que una clase se compromete a implementar.
- La interfaz marca qué métodos, con su firma
- Desde Java 8, pueden incluir también métodos con cuerpo (abstractos, estáticos y por defecto).
- También puede incluir constantes.

Interfaces

- Utilidad de los interfaces: “simular” herencia múltiple.

- Nota: los interfaces solo tienen sentido en lenguajes de programación que solo permiten usar la herencia simple

Interfaces

DEFINICIÓN DE INTERFACES

- **interface**
- Mismas normas de acceso que una clase.
- Mismas reglas de nombres que una clase.
- También existe herencia de interfaces (**extends**). En este caso, sí que puede ser múltiple.

```
public interface GroupedInterface extends Interface1, Interface2  {
     // constant declarations
     // base of natural logarithms
     double E = 2.718282;
     // method signatures
     void doSomething (int i, double x);
     int doSomethingElse(String s);
}
```

Interfaces

```
public interface VehiculoCarrera {
    public float aceleracionDe0a100();
    public float usarNitro();
    public void usarMarchaEconomica();
    public void frenadaEmergencia();
}
```

Interfaces

```
    public float aceleracionDe0a100(){
        /* ...*/
    }
    public float usarNitro(){
        /* ...*/
    }
    public void usarMarchaEconomica(){
        /* ... */
    }
    public void frenadaEmergencia(){
        /* ... */
    }
}
```

Interfaces: Implementación

- **implements**
- Una clase puede implementar más de una interfaz

```
1  package Ejemplo5_Interfaces;
2
3  public class Rectangulo implements FiguraGeometrica {
4      public int ancho;
5      public int alto;
6
7      @Override
8      public boolean isLargerThan(FiguraGeometrica fg) {
9          Rectangulo rc = (Rectangulo)fg;
10         if(this.getArea()>rc.getArea()) return true;
11         return false;
12     }
13
```

Interfaces: como tipos

- Una interfaz puede ser el tipo de dato a usar para crear una instancia de un objeto.
- La clase del objeto a crear debe implementar dicha interfaz.
- Muy útil, sobre todo, para recibir argumentos en métodos.

```
FiguraGeometrica rect1 = new Rectangulo(10,20);
Rectangulo rect2 = new Rectangulo(15,10);
```

Interfaces: métodos por defecto

- Novedad en Java 8
- **default**.
- Un método puede tener una implementación por defecto, descrita en la interfaz.

```
default public void metodoPorDefecto() {
    System.out.println("Se ejecuta el método por defecto");
}
```

Enumerados

Enumerados (enum)

- ***Un enumerado (o Enum) es una clase "especial"*** (tanto en Java como en otros lenguajes) ***que limitan la creación de objetos a los especificados explícitamente en la lista que concretamos en la implementación de la clase.***

Enumerados (enum)

- ¿Cómo se declara una clase de tipo enum?

Sintaxis:

```
[Público/Privado] enum nombreTipoEnumerado {
        ELEMENTO1, ELEMENTO2, ELEMENTO3, …,
            ELEMENTOn
};
```

Ejemplo:

```
enum Talla {PEQUEÑA,MEDIANA,GRANDE};
```

Enumerados (enum)

- Dentro de las llaves se declaran las variables (objetos) de que consta el tipo enumerado.
- Por convención, sus nombres se escriben en letras mayúsculas para recordarnos que son valores fijos (que en cierto modo las podemos ver como constantes).
- Una vez declarado el tipo enumerado, todavía no existen variables hasta que no las creemos explícitamente, de la misma manera que ocurre con cualquier tipo Java.

Enumerados (enum)

¿Cómo se da valor a un objeto de una clase de tipo enum?

Para crear una variable de tipo enumerado lo haremos con una declaración simple que recuerda a la creación de una variable tipo primitivo:

<nombreTipoEnumerado> <miNombreElegido>;

< miNombreElegido >.<valor válido>

O

<nombreTipoEnumerado> <miNombreElegido>=<valor válido>

Ejemplo:

Talla t ;
Talla.PEQUEÑA;

(*) las clases de tipo enum son Static, podemos utilizar el nombre de la clase al usar los objetos.

Enumerados (enum)

Ejemplo:

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

```
Demarcacion puesto;
 puesto = Demarcacion.DELANTERO;
```

O

```
Demarcacion puesto = Demarcacion.DELANTERO;
```

Enumerados (enum)

- Métodos que pueden utilizar los objetos de clases de tipo enum, entre otros.

  - <objeto>.toString()
  - <objeto>. ordinal()
  - <objeto>. compareTo()

Enumerados (enum)

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 de la constante (DELANTERO)
delantero.name();

// Devuelve un String con el nombre de la constante (DELANTERO)
delantero.toString();

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

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

Enumerados (enum)

Ejemplo 1:

```
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());
    System.out.println(delantero.compareTo(defensa));
```

Enumerados (enum)

Ejemplo 2:

```
public class TestEnum {
    enum TipoDeMadera { ROBLE, CAOBA, NOGAL, CEREZO, BOJ };

    public static void main (String[ ] Args) {
        TipoDeMadera maderaUsuario;
        maderaUsuario = TipoDeMadera.ROBLE;
        System.out.println ("La madera elegida por el usuario es " +
              maderaUsuario.toString().toLowerCase() );
        System.out.println ("¿Es la madera elegida por el usuario caoba?
Resultado: " +        (maderaUsuario==TipoDeMadera.CAOBA) );
        System.out.println ("¿Es la madera elegida por el usuario roble? Resultado:
" +          (maderaUsuario==TipoDeMadera.ROBLE) );
   }
```

(*) Importante la declaración está fuera del main () e incluso fuera de la clase

Enumerados (enum)

Ejemplo 2:

El resultado de la ejecución será similar a este:

La madera elegida por el usuario es roble
¿Es la madera elegida por el usuario caoba? Resultado: false
¿Es la madera elegida por el usuario roble? Resultado: true