Módulo profesional Programación

UD 9.3 – Tipos Enumerados

Tipos enumerados

Supongamos que estamos escribiendo el juego de **Piedra-Papel-Tijera.**

Para representar los tres gestos de la mano podríamos usar tres enteros arbitrarios (por ejemplo, 0, 1, 2; o 88, 128, 168), tres cadenas ("Tijera", "Papel", "Piedra"), o tres caracteres ('s', 'p ',' t ')

¿Verdad?

Tipos enumerados

Un mejor enfoque es definir nuestra propia **lista de elementos permitidos** en una construcción llamada **enumeración** introducida en JDK 1.5. La sintaxis es la siguiente:

```
enum Nombre {
   ELEMENTO1, ELEMENTO2, ...;
}
```

Por ejemplo,

```
enum Gesto {
    PIEDRA, PAPEL, TIJERAS
}
```

Un enumerado es una clase especial, que proporciona una implementación segura de tipos de datos constantes (type-safe) en tu programa.

Tipos enumerados

En otras palabras, podemos declarar una variable del tipo **Gesto**, que toma valores de Gesto.TIJERAS, Gesto.PAPEL o Gesto.PIEDRA, pero NADA MÁS.

Por ejemplo,

```
Gesto playerMove;                //Declaro variables de tipo Gesto
Gesto computerMove;
playerMove = Gesto.TIJERAS;      //Asigno valores a las variables
computerMove = Gesto.PAPEL;
//playerMove = 0;                //Error de compilación
```

```
compilation error: incompatible types: int cannot be converted to Gesto
```

Tipos enumerados

Ejemplo, el palo de una carta solo puede ser trébol, diamante, corazón o pica. En otras palabras, tiene un **conjunto limitado de valores.** Antes de la introducción del tipo de enumeración en JDK 1.5, normalmente tenemos que usar una variable **int** para mantener estos valores. Por ejemplo,

```
class Palo {
        public static final int TREBOL 0;
        public static final int DIAMANTE 1;
        public static final int CORAZON 2;
        public static final int PICA 3;
        ...
}

class Carta {
    int palo; // Palo.TREBOL, Palo.DIAMANTE, Palo.CORAZON, Palo.PICA
    ...
}
```

Tipos enumerados

Los **inconvenientes** con constantes son:

- **No es type-safe.** Puede asignar cualquier valor int (por ejemplo, 88) a la variable int.
- **Fragilidad**: nuevas constantes romperían los códigos existentes.
- **Los valores impresos no son informativos:** el valor impreso de 0, 1, 2 y 3 no es muy significativo.

Tipos enumerados

JDK 1.5 introduce un nuevo tipo: enumeración junto con la palabra clave **enum**.

Por ejemplo, podríamos definir:

```
enum Palo {TREBOL , DIAMANTE , CORAZON , PICA};
```

- Se puede utilizar una enumeración para definir un conjunto de **constantes de enumeración**. Las constantes son implícitamente finales estáticas, por lo que no pueden ser modificadas.
- Puedes referirse a estas constantes como cualquier constante estática, por ejemplo, Palo.TREBOL, Palo.DIAMANTE, etc.
- La enumeración es type-safe. Tiene su propio espacio de nombres y funciona con la sentencia switch-case (igual que int y char).

Actividad

1. Copia, analiza y ejecuta el

Siguiente código

```
import java.util.*;

enum Suit { SPADE, DIAMOND, CLUB, HEART }
enum Rank { ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING }

class Card {  // A card
   private Suit suit;
   private Rank rank;

   Card(Suit suit, Rank rank) {   // constructor
      this.suit = suit;
      this.rank = rank;
   }

   Rank getRank() { return rank; }
   Suit getSuit() { return suit; }
   public String toString() { return "This card is " + rank + " of " + suit; }
}

class CardDeck {  // A deck of card
   List<Card> deck;
   CardDeck() {   // constructor
      deck = new ArrayList<Card>();
      for (Suit suit : Suit.values()) {
         for (Rank rank : Rank.values()) {
            deck.add(new Card(suit, rank));
         }
      }
   }
   public void print() {
      for (Card card : deck) System.out.println(card);   // print all cards
   }
   public void shuffle() {
      Collections.shuffle(deck);  // use java.util.Collections' static method to shuffle the List
   }
}

public class CardTest {
   public static void main(String[] args) {
      CardDeck deck = new CardDeck();
      deck.print();
      deck.shuffle();
      deck.print();
   }
}
```

Tipos enumerados

```
enum Palo {TREBOL , DIAMANTE , CORAZON , PICA};
```

El compilador traduce los enumerados a una clase final

```
final class Palo extends java.lang.Enum {
    ...
}
```

Todas las enumeraciones extienden implícitamente la clase **java.lang.Enum**. Como una clase solo puede heredar de un padre en Java, una enumeración no puede heredar nada más.


```
enum Palo {TREBOL , DIAMANTE , CORAZON , PICA};
```


Tipos enumerados

El compilador también crea una **instancia de la clase** para cada una de los elementos definidos dentro de la enumeración. El java.lang.Enum tiene estos métodos:

```
public final String name();  // Returns the name of this enum constant, exactly as declared in its enum declaration.
                             // You could also override the toString() to provide a more user-friendly description.
public String toString();    // Returns the name of this enum constant, as contained in the declaration.
                             // This method may be overridden.
public final int ordinal();  // Returns the ordinal of this enumeration constant.
```

Tipos enumerados

Definir tipos enumerado tiene ventaja en **colecciones de datos constantes,** tales como estados de un sistema, días de la semana, meses del año, etc.

Un tipo enumerado **restringe los posibles valores que puede tomar una variable**. Esto ayuda a reducir los errores en el código.

De manera similar la definición de una clase, se puede definir un tipo **enum**.

Tipos enumerados

Sintaxis

```
[modificadores] enum nombre {
    ELEMENTO 1,
    ELEMENTO 2,
    ELEMENTO 3,
    ...,
    ELEMENTO n
};
```

Dentro de las llaves se declaran las **variables** 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 podemos ver como constantes).

Una vez declarado el tipo enumerado, todavía no existen variables de este tipo hasta que no las creemos explícitamente, de la misma manera que ocurre con cualquier tipo Java.

Tipos enumerados

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

nombreEnumerado **nombreVariable**;

Tipos enumerados

Declaración de tipo enum y métodos más utilizados

```
public enum Demarcacion{PORTERO, DEFENSA, CENTROCAMPISTA, DELANTERO}
Demarcacion delantero = Demarcacion.DELANTERO;    // Instancia de un enum de la clase Demarcación
delantero.name();    // Devuelve un String con el nombre de la constante (DELANTERO)
delantero.toString();    // Devuelve un String con el nombre de la constante (DELANTERO)
delantero.ordinal();    // Devuelve un entero con la posición del enum según está declarada (3).
delantero.compareTo(Enum otro);    // Compara el enum con el parámetro según el orden en el que están declarados lo enum
Demarcacion.values();    // Devuelve un array que contiene todos los enum
```

https://docs.oracle.com/javase/8/docs/api/java/lang/Enum.html

```
enum Palo {TREBOL , DIAMANTE , CORAZON , PICA};
```

Tipos enumerados

Esta forma de creación de variables de tipo enumerado se justifica porque los tipos enumerados en principio no tienen constructores (más adelante veremos que sí los pueden tener).

Los **valores de un tipo enumerado son objetos propiamente dichos**, ¿de qué tipo? Del tipo enumerado cuyo nombre se haya indicado.

Ten en cuenta, porque es una confusión habitual, que los tipos enumerados no son enteros, ni cadenas (aunque a veces podamos hacer que se comporten de forma similar a como lo haría un entero o una cadena).

Cada elemento de un enumerado es un objeto único disponible para su uso.

Tipos enumerados

La identificación de cada objeto del tipo se hace con la sintaxis del punto, es decir, nos referimos a un elemento concreto como

nombreEnumerado.ELEMENTO1.

Esta sintaxis nos quiere recordar lo que sería un campo de una clase pero no es así: en este caso es un **objeto de un tipo enumerado.**

Cada objeto enum siempre es implícitamente **public static final.** Entonces, como es **static**, podemos acceder utilizando el nombre del ***enum***. Y, como es **final**, no podemos crear enumeraciones "*hijas*".

Tipos enumerados

Un tipo enumerado puede ser declarado dentro o fuera de una clase, pero no dentro de un método.

Por tanto no podemos declarar un enum dentro de un método main (programa principal); si lo hacemos nos saltará el error de compilación "**enum types must not be local**" (los tipos enumerados no pueden ser locales a un método).

Por tanto el tipo enumerado lo declararemos o bien antes del public class… o bien después del public class… pero fuera de cualquier método o constructor.

El tipo **enumerado** se puede pasar como un argumento para ***switch.***

Estruc
enume

```
public enum Demarcacion{PORTERO, DEFENSA, CENTROCAMPISTA, DELANTERO}
Demarcacion delantero = Demarcacion.DELANTERO;    // Instancia de un enum de la clase Demarcación
delantero.name();    // Devuelve un String con el nombre de la constante (DELANTERO)
delantero.toString();    // Devuelve un String con el nombre de la constante (DELANTERO)
delantero.ordinal();    // Devuelve un entero con la posición del enum según está declarada (3).
delantero.compareTo(Enum otro);    // Compara el enum con el parámetro según el orden en el que están declarados lo enum
Demarcacion.values();    // Devuelve un array que contiene todos los enum
```

Dado el siguiente fragmento de código:

```
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 la salida:

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

Tipos enumerados

Un **enum** es una clase especial que **limita la creación de objetos a los especificados en su clase**, pero estos objetos pueden tener **atributos** como cualquier otra clase.

En la siguiente declaración de la clase, vemos un ejemplo en la 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;
	}

}
```

Tipos enumerados

Como se ve BARÇA, 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".

El acceso de un constructor es package-private o private. No se puede invocar a un constructor de un enum.

Para trabajar con los atributos de estos enumerados se hace de la misma manera que con cualquier otro objeto; se accede a los atributos con los métodos get.

```
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;
	}

}
```

Tipos enumerados

En el siguiente fragmento de código vemos como 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
```

Tipos enumerados

Es muy importante que tengáis 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 esta aprendiendo que 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.**

Hay dos restricciones que se aplican a las enumeraciones. Primero, una enumeración no puede heredar otra clase. En segundo lugar, una enumeración no puede ser una superclase.

Ejemplo

```
enum TrafficSignal{
    RED("stop"), GREEN("start"), ORANGE("slow down");

    private String action;

    public String getAction(){
        return this.action;
    }
    private TrafficSignal(String action){
        this.action = action;
    }
}
```

```
public class EnumValueOfExample {

    public static void main(String args[]) {

      //valueOf method returns Enum instance with name matching to String passed to valueOf
      TrafficSignal signal = TrafficSignal.valueOf("RED");
      System.out.println("name : " + signal.name() + " action : " + signal.getAction());

      //Another Enum valueOf exampel
      signal = TrafficSignal.valueOf("GREEN");
      System.out.println("name : " + signal.name() + " action : " + signal.getAction());


      //valueOf will throw IllegalArgumentException if we pass invalid String
      signal = TrafficSignal.valueOf("Green");
    }

}
```

Actividad

Copia, analiza y ejecuta el siguiente código.

```
La velocidad típica para un avión es: 600 millas por hora.

Todas las velocidades de transporte:
COCHE: velocidad típica es 60 millas por hora.
CAMION: velocidad típica es 50 millas por hora.
AVION: velocidad típica es 600 millas por hora.
TREN: velocidad típica es 70 millas por hora.
BARCO: velocidad típica es 20 millas por hora.
```

```
//Uso de un constructor, una variable de instancia y un método.
enum Transporte{
    COCHE(60), CAMION(50), AVION(600), TREN(70), BARCO(20);
    private int velocidad; //velocidad típica de cada transporte

    //Añadir un onstructor
    Transporte(int s){velocidad=s;}
    //Añadir un método
    int getVelocidad(){return velocidad;}
}

class Enumerados {
    public static void main(String[] args) {
        Transporte tp;
       //Mostrar la velocidad de un avión
        System.out.println("La velocidad típica para un avión es: "+
        Transporte.AVION.getVelocidad()+ " millas por hora.\n");

        //Mostrar todas las velocidades y transportes
        System.out.println("Todas las velocidades de transporte: ");
        for (Transporte t:Transporte.values())
            System.out.println(t+ ": velocidad típica es "+t.getVelocidad()+" millas por hora.");
    }
}
```

Actividad: phone model

1. Crea un tipo enum PhoneModel, que contenga los siguientes elementos: IPHONE, HUAWEI, PIXEL, SAMSUNG, LG.
2. Crea un campo llamado precio (tipo int). Escribe un método getter para este campo.
3. Crea un constructor PhoneModel(int price) de un solo argumento que pueda utilizarse para crear las constantes enum. Los precios de los cinco modelos son: 9999, 8888, 6666, 9399 y 5588.
4. Crea una clase de prueba que recomiende posibles teléfonos para un usuario en función de su presupuesto.

Three sample runs:

```
Your budget: 4000
You do have sufficient money
```

```
Your budget: 8888
HUAWEI      price: 8888
PIXEL       price: 6666
LG          price: 5588
```

```
Your budget: 10000
IPHONE      price: 9999
HUAWEI      price: 8888
PIXEL       price: 6666
SAMSUNG     price: 9399
LG          price: 5588
```

Actividad: día de la semana

1. Crea un enum público DiaSemana con constantes para LUNES, MARTES,... hasta DOMINGO.
2. El enum debe tener un método de instancia boolean esDiaSemana() y un método de instancia boolean esFinSemana().El método esFinSemana() debe devolver lo contrario de esDiaSemana().
3. Escribe un programa que demuestre cómo se podría utilizar este enum, que tenga un método que tome un día de la semana como argumento e imprima un mensaje dependiendo de si el día de la semana es fin de semana o no.
4. Sugerimos que el método main haga un bucle sobre todos los valores del enum Semana y los envíe como argumento al método.

Sugerencia:

- cada enum en Java tiene un método estático values(), que devuelve un array de los valores en el enum, por lo que puede utilizar un bucle for-each para esto.
- cada enum tiene una implementación toString() que devuelve el nombre de la constante tal y como se declaró en el enum, por ejemplo "VIERNES".

Actividad

5. Utiliza tu clase DiaSemana para investigar si los enums implementan Comparable.
6. Declara un DiaSemana sabado = DiaSemana.SABADO. Usa un bucle sobre DiaSemana.values() con **dia** como variable de bucle e imprime cada valor y si es menor, igual o mayor que sabado, usando la llamada dia.compareTo(sabado)

Recuerda que el método compareTo() devuelve un int tal que un valor negativo significa que dia es menor que sábado, cero significa que dia se considera igual a sabado (cuando se comparan por orden) un valor positivo significa que dia es mayor que sábado.

¿Cuándo debería utilizar enum?

¿Cuándo deberías usar enumeraciones?

Cada vez que necesites un **conjunto finito de constantes, cuyos valores se conocen en tiempo de compilación.**

Esto incluye los tipos enumerados de forma natural (como los **días de la semana** y los **palos en una baraja de cartas**), así como otros conjuntos en los que conoce todos los valores posibles en el momento de la compilación, como las **opciones en un menú**, etc.

No es necesario que el conjunto de constantes en un tipo de enumeración permanezca fijo para siempre. En la mayoría de las situaciones, se puede agregar nuevas constantes a una enumeración sin romper los códigos existentes.

Recursos

- Curso youtube : Java desde 0
- Libro Java 9. Manual imprescindible. F. Javier Moldes Teo. Editorial Anaya
- App SoloLearn: Aprende a Programar. Curso Java