Definición de la clase enum

La clase enum es un tipo especial en Java que en cierta medida se puede usar como una clase y admite ciertas posibilidades especiales. Nos permiten definir listas de elementos, los cuales se identifican por su posición.

Además de este comportamiento básico, las enumeraciones de Java pueden definir métodos y atributos, lo que las hace muy flexibles a la hora de definir, por ejemplo, las distintas opciones que pueden aparecer en un menú.

Ejemplos de explicación

El ejemplo siguiente muestra cómo definir las distintas opciones de un menú para un cajero automático:

```
public enum opcionesMenu {

    SALIR("Salir de la agenda"),
    AGREGAR_CONTACTO("Agregar contacto"),
    MOSTRAR_CONTACTO("Mostrar contactos"),
    BORRAR_CONTACTO("Borrar contactos"),
    MODIFICAR_CONTACTO("Modificar contactos");

```

Cada elemento de la enumeración posee un orden:

- 0. Salir
- 1. Agregar contacto
- 2. Mostrar contacto
- 3. Borrar contacto
- 4. Modificar contacto

El orden lo podemos utilizar para filtrar un elemento de la enumeración en un switch:

```
opcionesMenu opcionMenu= opcionesMenu.getOpcion(opcion);

switch(opcionMenu) {
case SALIR:
    System.out.println("saliendo");
    break;
case AGREGAR_CONTACTO:
    System.out.println("Diga nombre");
    nombre=sc.nextLine();

    System.out.println("Diga número de teléfono");
    tfno=sc.nextLine();

    a.agregarContacto(new Contacto(nombre), tfno);
    break;

case MOSTRAR_CONTACTO:
    a.mostrarContactos();
    break;
```

Ahora puedo cambiar el orden de los elementos dentro de la enumeración o puedo añadir nuevos elementos en cualquier posición dentro de la enumeración y el código anterior seguirá funcionando.

La clase Enum

Al definir una enumeración se está extendiendo a la clase java.lang.Enum. Por ello, tenemos disponible todos los métodos definidos en ella, los cuales son:

- clone(): Lanza CloneNotSupportedException lo que garantiza que las enumeraciones nunca se clonen. Es raro que se use ya que la clase Enum sirve precisamente para no tener que repetir ningun elemento que esté en esta clase
- compareTo(): Compara esta enumeración con el objeto especificado para el pedido.
- values(): Devuelve un array con todos los elementos de la enumeración.

```
opcionesMenu [] om = opcionesMenu.values();
    for (int i = 0; i < om.length; i++) {
        System.out.println(om[i]);
    }
```

- ordinal(): Devuelve el orden del elemento dentro de la enumeración.No se usa salvo en estructuras de datos sofisticadas basadas en enumeraciones como EnumSet y EnumMap.
- name(): Devuelve el nombre del elemento. Es más recomendable usar toString() ya que devuelve un nombre más fácil de usar, aunque se usa en casos específicos en los que se quiere obtener el nombre más exacto.

El efecto colateral de extender implícitamente un Enum es que una enumeración no se puede extender a ninguna clase.

Ejemplos de utilización de la clase Enum

Aparte de los métodos implementados en la clase podemos definir atributos y métodos propios:

```
3  public enum opcionesMenu {
4
5      SALIR("Salir de la agenda"),
6      AGREGAR_CONTACTO("Agregar contacto"),
7      MOSTRAR_CONTACTO("Mostrar contactos"),
8      BORRAR_CONTACTO("Borrar contactos"),
9      MODIFICAR_CONTACTO("Modificar contactos");
0
1
2      private String descripcion;
3
4      private opcionesMenu(String descripcion) {
5          this.descripcion=descripcion;
6      }
7
8      public String getDescripcion() {
9          return descripcion;
0      }
1
```

Cada vez que se crea un elemento de la enumeración se le pasa un String en el constructor que se almacena en el atributo String descripcion:

```
//Ejemplo con el atributo descripcion
opcionesMenu om = opcionesMenu.AGREGAR_CONTACTO;
System.out.println(om.getDescripcion());
```

Esto mostrará “Agregar contacto”.

Podemos añadir más métodos a nuestra enumeración para hacer las cosas más fáciles:

```
public static opcionesMenu getOpcion(int posicion) {
    return values()[posicion];
}

public static String getMenu() {
    StringBuilder sb=new StringBuilder();

    for (opcionesMenu opcion : opcionesMenu.values())
        sb.append(opcion.ordinal());
        sb.append(".- ");
        sb.append(opcion.getDescripcion());
        sb.append("\n");
    }
    return sb.toString();
    }
```

Así podemos mostrar nuestro menú con nuestras opciones, y pedirle al usuario que elija una:

```
36          do {
37              System.out.println(opcionesMenu.getMenu());
38              System.out.println("Elije una opción:");
39              aux=sc.nextLine();
40              opcion=Integer.parseInt(aux);
41
42              opcionesMenu opcionMenu= opcionesMenu.getOpcion(opcion);
43
44              switch(opcionMenu) {
45              case SALIR:
46                  System.out.println("saliendo");
47                  break;
48              case AGREGAR_CONTACTO:
49                  System.out.println("Diga nombre");
50                  nombre=sc.nextLine();
51
52                  System.out.println("Diga número de teléfono");
53                  tfno=sc.nextLine();
54
55                  a.agregarContacto(new Contacto(nombre), tfno);
56                  break;
57
58              case MOSTRAR_CONTACTO:
59                  a.mostrarContactos();
60                  break;
```

```
Console
Main (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (21 feb 2024 17:19:07) [pid: 38712]
0.- Salir de la agenda
1.- Agregar contacto
2.- Mostrar contactos
3.- Borrar contactos
4.- Modificar contactos

Elije una opción:
```

Interfaces en enumeraciones

Una enumeración no puede extender a ninguna clase, pero puede implementar todas las interfaces que necesite:

```
2
3  public enum opcionesMenu implements DarDescripcion{
4
5      SALIR("Salir de la agenda"),
6      AGREGAR_CONTACTO("Agregar contacto"),
7      MOSTRAR_CONTACTO("Mostrar contactos"),
8      BORRAR_CONTACTO("Borrar contactos"),
9      MODIFICAR_CONTACTO("Modificar contactos");
10
11
```

Cuya interfaz es:

```
3 public interface DarDescripcion {
4     String getDescripcion();
5 }
6
```

Conclusiones

- Las enumeraciones nos sirven para definir elementos con un orden entre ellos bajo un tipo común.
- Para definir una enumeración utilizamos la palabra reservada enum.
- Una enumeración implícitamente se extiende a la clase java.lang.Enum, luego explícitamente no puede extenderse a ninguna otra clase.
- Una enumeración puede definir atributos y métodos, lo que las hace bastante ricas.
- Una enumeración puede implementar todas las interfaces que necesite.

Como material complementario adjunto un archivo con el proyecto hecho para la explicación de esta clase.