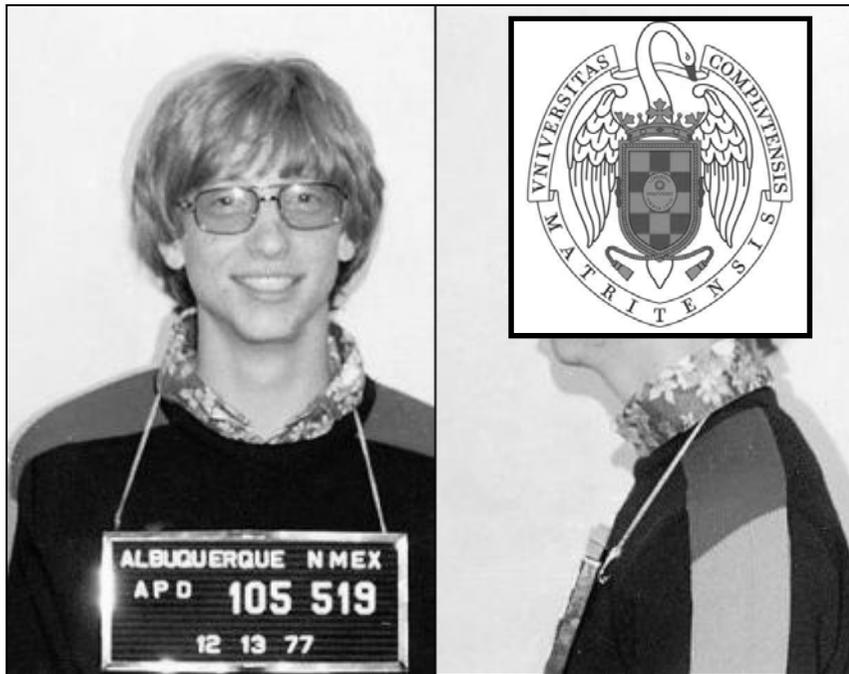


LPS: Enumerados



Federico Peinado
www.federicopeinado.es

Depto. de Ingeniería del Software e
Inteligencia Artificial
disia.fdi.ucm.es

Facultad de Informática
www.fdi.ucm.es

Universidad Complutense de Madrid
www.ucm.es

Enumerados

- ◎ Tipos definidos por el usuario (*Enum Types*) con una lista invariable de nombres propios como valores
 - Disponibles desde Java 1.5, y más potentes que en otros lenguajes
 - Se compilan como clases “predefinidas implícitamente” (extienden `java.lang.Enum` y tienen métodos predefinidos - que veremos en el Javadoc- , aunque se pueden añadir más)
 - Sus valores son atributos estáticos, objetos que se construyen la primera vez que se utilizan) y que, curiosamente, son del propio tipo enumerado y se les puede añadir atributos y métodos propios
- ◎ Referencias sobre enumerados
 - Java Tutorials
 - <http://java.sun.com/docs/books/tutorial/java/javaOO/enum.html>
 - Thinking in Java (4th Edition)
 - Enumerated Types (páginas 725 – 759)

Ejemplo

- ◉ Antes de Java 1.5 solían usarse constantes enteras para simular enumerados

```
class Semaforo {  
    public static final int ROJO = 0;  
    public static final int AMBAR = 1;  
    public static final int VERDE = 2;  
}
```

- ◉ La notación actual es mucho más elegante, segura y comprensible

- Los valores se suelen seguir escribiendo en mayúsculas

```
enum Semaforo {ROJO, AMBAR, VERDE};
```

- ◉ El compilador traduce los enumerados a una clase final

```
final class Semaforo extends java.lang.Enum {  
    public static final Semaforo ROJO;  
    public static final Semaforo AMBAR;  
    public static final Semaforo VERDE;  
  
    public static final Semaforo[] values();  
    public static final Semaforo valueOf(java.lang.String);  
  
    static {};  
}
```



Métodos interesantes

◎ **int compareTo (E o)**

- Los enumerados son comparables (interfaz **Comparable**)

◎ **boolean equals (E o)**

- Sólo en los enum **equals** es equivalente a **==**

◎ **String name ()**

- Devuelve el nombre del valor (como **toString**)

◎ **int ordinal ()**

- Devuelve posición del valor en la lista de valores

◎ **static E valueOf (String s)**

- Devuelve el valor *E* correspondiente al nombre *s*

Uso de enumerados

- ◉ Normalmente todos los atributos estáticos en Java –como pasa con los enumerados- requieren que su nombre sea cualificado (indicando la clase a la que pertenecen)
 - Esto puede evitarse usando la cláusula `static import`
- ◉ Los enumerados resulta útiles para las instrucciones **switch**
 - Fíjate que justo en los **case** no hace falta cualificarlos

```
Semaforo color = ...;
switch(color) {
    case ROJO:
        color = Semaforo.VERDE;
        break;
    case VERDE:
        color = Semaforo.AMBAR;
        break;
    case AMBAR:
        color = Semaforo.ROJO;
        break;
}
```

Añadir código al enumerado

- ◉ Al heredar ya de `java.lang.Enum`, los enumerados no se pueden extender
 - Aunque sí se pueden sobrescribir los métodos que traen por defecto

- ◉ Así se añade código a un tipo enumerado:

```
public enum Die6fSymbol {
    ONE, TWO, THREE, FOUR, FIVE, SIX;

    int getValue() {
        return ordinal()+1;
    };
}
```

```
public enum OzWitch() {
    WEST("Miss Gulch"), EAST("Wicked Witch"),
    NORTH("Glinda"), SOUTH("Doh");

    private String description;
    private OzWitch(String desc) { description = desc;};
    public String getDesc() { return description;};
}
```

Interfaces y enumerados

- ◉ Para definir “subcategorías” de un enumerado no se puede hacer con herencia, pero sí agrupando varios enumerados bajo una interfaz
 - De esta forma distintos tipos de enumerados se comportan como un solo tipo
 - Podemos incluso *forzar* a que se implementen ciertos métodos comunes

```
public interface Food {  
    enum MainCourse implements Food {  
        LASAGNE, BURRITO, LENTILS;  
    }  
    enum Dessert implements Food {  
        TIRAMISU, GELATO, FRUIT;  
    }  
    ...  
}
```

```
Food food = MainCourse.LASAGNE;  
food = Dessert.TIRAMISU;
```

Añadir código a los valores

- Se pueden definir métodos abstractos del enumerado que son *sobreescritos e implementados* después por cada uno de los valores

- Así se añade código a los valores de un enumerado:

```
public enum Die6fSymbol implements DieSymbol {
    ONE {
        public String toString(){
            String s= "One's the Loneliest number";
            return s;
        }
    },
    TWO {
        public String toString(){
            String s= "Two's Company";
            return s;
        }
    },
    ...
};
```

Ejemplos



Ejemplo *Piedra, Papel o Tijeras*

```
public enum Resultado {GANA, PIERDE, EMPATA};
public enum JuegoPPT {
    PIEDRA {
        public Resultado competir(JuegoPPT oponente) {
            return evaluar(TIJERAS, oponente);
        }
    },
    PAPEL {
        public Resultado competir(JuegoPPT oponente) {
            return evaluar(PIEDRA, oponente);
        }
    },
    TIJERAS {
        public Resultado competir(JuegoPPT oponente) {
            return evaluar(PAPEL, oponente);
        }
    };
    public abstract Resultado competir(JuegoPPT oponente);
    public Resultado evaluar (JuegoPPT perdedor, JuegoPPT oponente) {
        if (oponente == this)
            return Resultado.EMPATA;
        else if (oponente == perdedor)
            return Resultado.GANA;
        else return Resultado.PIERDE;
    }
}
```

Ejemplo *Semáforo aleatorio*

```
import java.util.*;

public class GeneradorEnums {
    private static Random rand = new Random(47);

    public static <T extends Enum<T>> T random(Class<T> ec) {
        return random(ec.getEnumConstants());
    }
    public static <T> T random (T[] values) {
        return values[rand.nextInt(values.length)];
    }
}

enum Semaforo {ROJO, AMBAR, VERDE};

for (int i = 0; i<10; i++)
    System.out.print(GeneradorEnums.random(Semaforo.class) + ", ");
```

- ⦿ Ejemplo de salida:
AMBAR, VERDE, ROJO, ROJO, VERDE, AMBAR, VERDE, ROJO, AMBAR, AMBAR,

Críticas, dudas, sugerencias...



Federico Peinado
www.federicopeinado.es