

Curso de Java a distancia

Clase 23: Definición de constantes en Java mediante la palabra clave final

Hasta ahora hemos visto que cuando definimos un atributo su valor puede ser cambiado en cualquier momento. Hay situaciones que ciertos datos a almacenar se los debe inicializar y nunca más van a cambiar, en esos casos debemos emplear constantes.

Si sabemos el valor que debe almacenar la constante podemos definirla y asignarle el valor

Problema

Implementar una clase llamada Circulo. Definir una constante donde se debe almacenar el valor de PI (relación entre la longitud de una circunferencia y su diámetro). Además definir otro atributo donde almacenar el radio del círculo.

Al constructor debe llegar el radio y otro método debe retornar el perímetro.

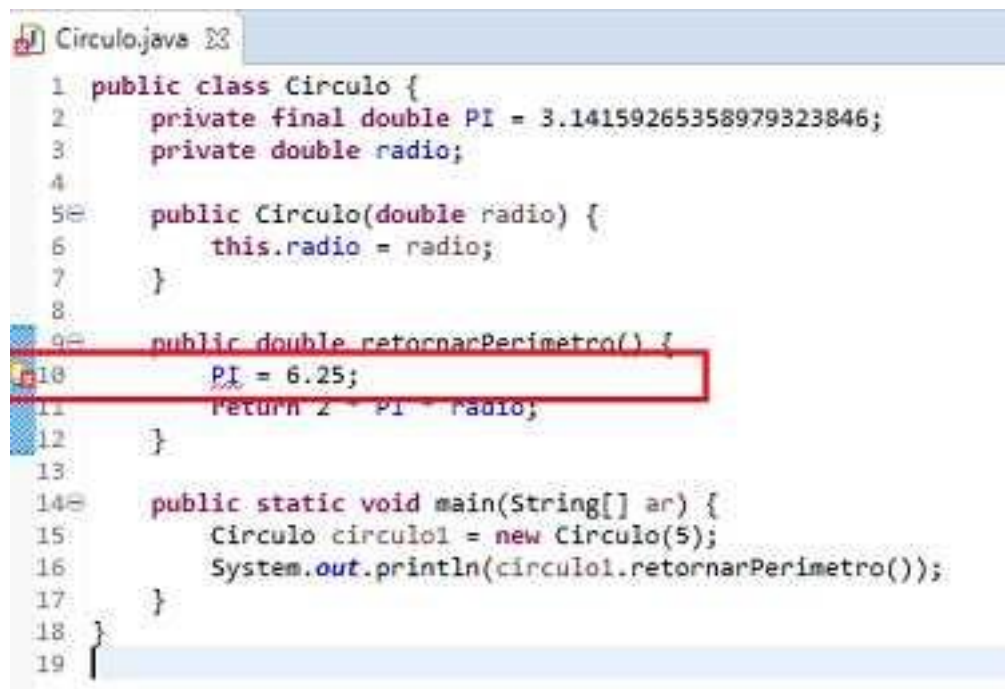
Programa:

```
public class Circulo {  
    private final double PI = 3.14159265358979323846;  
    private double radio;  
  
    public Circulo(double radio) {  
        this.radio = radio;  
    }  
  
    public double retornarPerimetro() {  
        return 2 * PI * radio;  
    }  
  
    public static void main(String[] ar) {  
        Circulo circulo1 = new Circulo(5);  
        System.out.println(circulo1.retornarPerimetro());  
    }  
}
```

Como vemos para definir una constante le antecedemos al tipo de dato la palabra clave 'final':

```
private final double PI = 3.14159265358979323846;
```

Una constante no puede variar su valor durante la ejecución del programa. Si intentamos asignarle otro valor se genera un error de compilación:



```
1 public class Circulo {
2     private final double PI = 3.14159265358979323846;
3     private double radio;
4
5     public Circulo(double radio) {
6         this.radio = radio;
7     }
8
9     public double retornarPerimetro() {
10        PI = 6.25;
11        return 2 * PI * radio;
12    }
13
14    public static void main(String[] ar) {
15        Circulo circulo1 = new Circulo(5);
16        System.out.println(circulo1.retornarPerimetro());
17    }
18 }
19 }
```

Una constante puede ser: private, public o protected (según nuestra necesidad)

Si no sabemos que valor se cargará en la constante hasta que procedamos a ejecutar el programa, la inicialización se deberá hacer obligatoriamente en el constructor de la clase. Veamos un ejemplo para ver su sintaxis.

Problema

Implementar una clase llamada CajaDeAhorro. Se debe almacenar el número de documento del titular y el monto depositado. Una vez que se carga el documento no permitir su cambio. Plantear dos constructores uno que llegue el documento del titular y el monto a depositar, y un segundo constructor que solo llegue el documento

Programa:

```
public class CajaDeAhorro {
    private final String documento;
    private float monto;

    public CajaDeAhorro(String documento, float monto) {
        this.documento = documento;
        this.monto = monto;
    }

    public CajaDeAhorro(String documento) {
        this.documento = documento;
        this.monto = 0;
    }

    public void imprimir() {
        System.out.println("Documento:" + documento + " Saldo:" + monto);
    }

    public static void main(String[] args) {
        CajaDeAhorro caja1 = new CajaDeAhorro("21222333", 1000);
    }
}
```

```

        CajaDeAhorro caja2 = new CajaDeAhorro("36454444");
        caja1.imprimir();
        caja2.imprimir();
    }
}

```

Como vemos definimos la constante llamada 'documento' pero no asignamos valor:

```
private final String documento;
```

Esto nos obliga que en el constructor o los constructores se deba inicializar en forma obligatoria la constante 'documento':

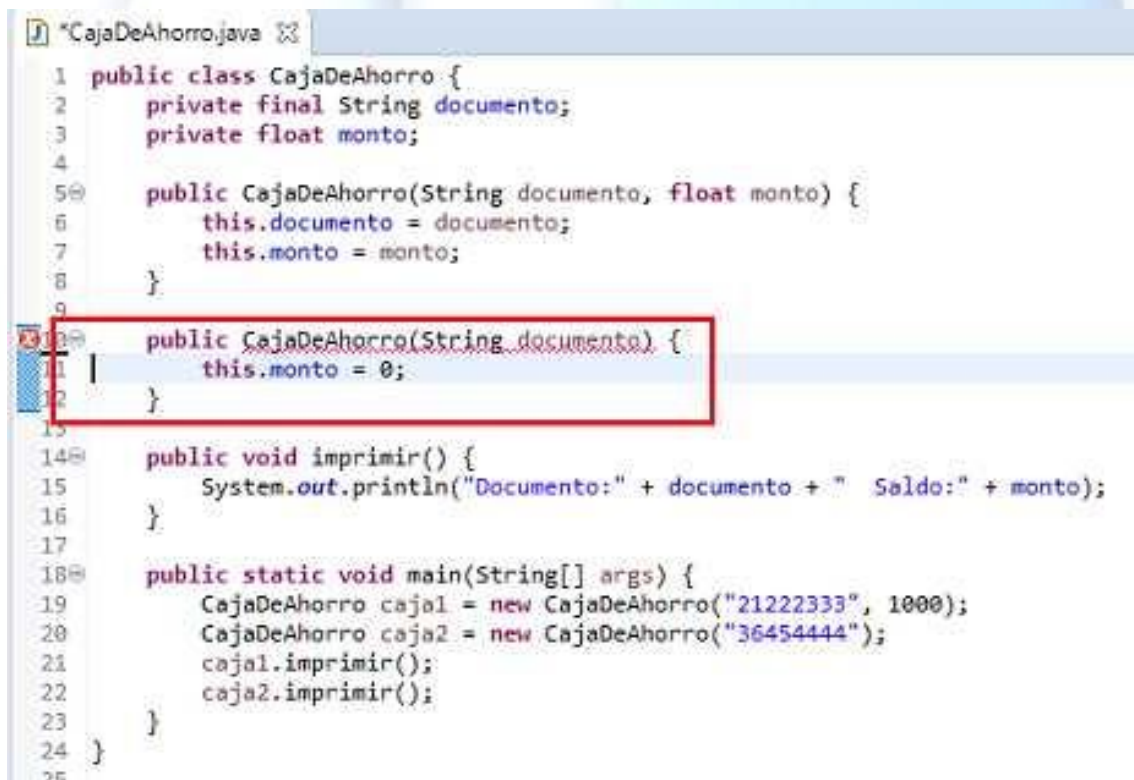
```

public CajaDeAhorro(String documento, float monto) {
    this.documento = documento;
    this.monto = monto;
}

public CajaDeAhorro(String documento) {
    this.documento = documento;
    this.monto = 0;
}

```

En el caso que nos olvidemos de inicializar la constante en el constructor el compilador de Java nos lo informará:



```

1  public class CajaDeAhorro {
2      private final String documento;
3      private float monto;
4
5      public CajaDeAhorro(String documento, float monto) {
6          this.documento = documento;
7          this.monto = monto;
8      }
9
10     public CajaDeAhorro(String documento) {
11         this.monto = 0;
12     }
13
14     public void imprimir() {
15         System.out.println("Documento:" + documento + " Saldo:" + monto);
16     }
17
18     public static void main(String[] args) {
19         CajaDeAhorro caja1 = new CajaDeAhorro("21222333", 1000);
20         CajaDeAhorro caja2 = new CajaDeAhorro("36454444");
21         caja1.imprimir();
22         caja2.imprimir();
23     }
24 }

```

Si bien nuestro programa funcionará perfectamente si no definimos a 'documento' como constante, el uso de esta característica del lenguaje nos puede evitar errores que en programas grandes pueden ser difíciles de encontrar.

Recordemos bien que no podemos asignar otro valor a una constante fuera del momento de declaración o del constructor, luego se genera un error sintáctico si codificamos:

```
public void imprimir() {
```

```
documento = "12345678"; //Error en tiempo de compilación
System.out.println("Documento:" + documento + " Saldo:" + monto);
}
```

Definición de constantes estáticas

Podemos definir una constante con el modificador 'static', recordemos que con este modificador hace que se reserve espacio para la constante en forma independiente a que se creen o no instancias de dicha clase.

Si definimos una constante con el modificador 'static' estamos obligados a inicializar su valor cuando lo declaramos y no podemos hacerlo en el constructor.

Problema

Plantear una clase Persona con los atributos nombre y edad. Implementar un método que retorne si es mayor de edad (almacenar en una constante estática el valor 18 que representa la mayoría de edad)

Programa:

```
public class Persona {
    private static final int MAYOREDADEAD = 18;
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public void imprimir() {
        System.out.println(nombre + " " + edad);
    }

    public String retornarNombre() {
        return nombre;
    }

    public boolean esMayor() {
        if (edad >= MAYOREDADEAD)
            return true;
        else
            return false;
    }

    public static void main(String[] args) {
        Persona persona1 = new Persona("Juan", 23);
        if (persona1.esMayor())
            System.out.println("Es mayor de edad " + persona1.retornarNombre());
        else
            System.out.println("No es mayor de edad " + persona1.retornarNombre());
    }
}
```

No importa la cantidad de objetos de la clase Persona que se creen, solo se reserva un espacio para la constante entera llamada 'MAYOREDADEAD':

```
private static final int MAYOREDADEAD = 18;
```

Depende si queremos acceder a la constante desde fuera de la clase debemos utilizar el modificador 'public' en lugar de private.

Constante local a un método.

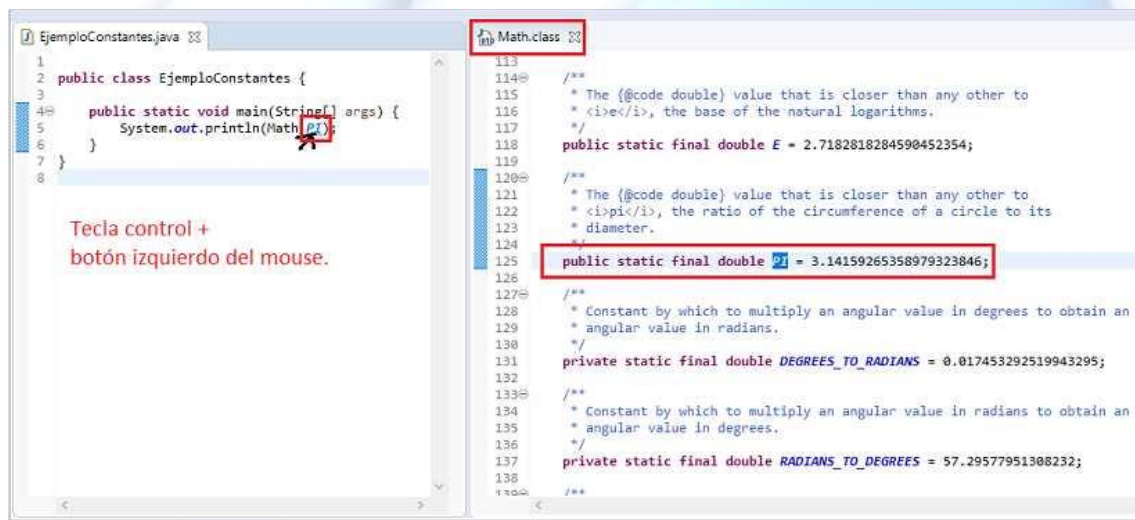
En el caso que definamos una constante dentro de un método, la misma se debe inicializar en el mismo momento que se la declara, o antes que se la consulte:

```
public class EjemploConstantes {  
  
    public static void main(String[] args) {  
        final double PI = 3.14159265358979323846;  
        final double radio;  
        radio=20;  
        double perimetro = 2 * PI * radio;  
        System.out.println(String.format("El perímetro de un círculo de radio %s es %s", radio,  
perimetro));  
    }  
}
```

Ejemplos de métodos estáticos en las clases estándares de Java

Existen clases en los paquetes que suministra Java que definen constantes.

Si necesitamos acceder al código fuente donde se define la constante debemos presionar la tecla 'Control' y presionar el botón izquierdo del mouse sobre la constante, luego se abre el archivo fuente con la declaración:



- La clase Math define 2 constantes públicas (es decir que las podemos acceder desde afuera de la clase):

```
public static final double E = 2.7182818284590452354;  
public static final double PI = 3.14159265358979323846;
```

Hay otras constantes pero al ser privadas no las podemos acceder desde fuera de la clase y no tiene sentido ni conocerlas (tiene una utilidad solo dentro de la clase Math):

```
private static final double DEGREES_TO_RADIANS = 0.017453292519943295;  
private static final double RADIANS_TO_DEGREES = 57.29577951308232;  
private static final long negativeZeroFloatBits = Float.floatToRawIntBits(-0.0f);  
private static final long negativeZeroDoubleBits = Double.doubleToRawLongBits(-0.0d);
```

- La clase Integer define 4 constantes públicas:

```
public static final int MIN_VALUE = 0x80000000;  
public static final int MAX_VALUE = 0x7fffffff;
```



```
public static final int SIZE = 32;
public static final int BYTES = SIZE / Byte.SIZE;
```

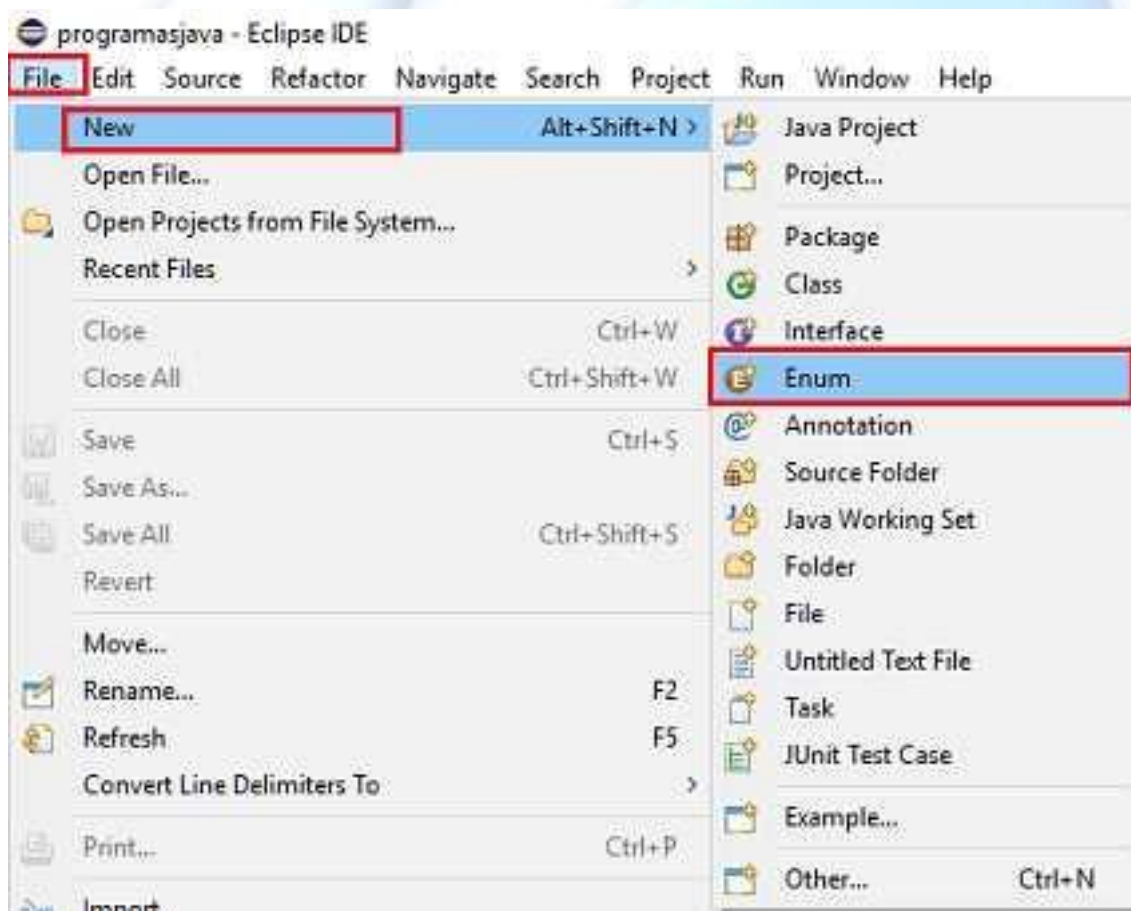
Podemos imprimir el contenido de estas cuatro constantes con el siguiente código:

```
public static void main(String[] args) {
    System.out.println(Integer.MAX_VALUE); // Mayor valor entero: 2147483647
    System.out.println(Integer.MIN_VALUE); // Menor valor entero: -2147483648
    System.out.println(Integer.SIZE);      // Número de bits que requiere un entero: 32
    System.out.println(Integer.BYTES);     // Número de bytes que requiere un entero: 4
}
```

Declaración de tipos enum

Básicamente en un conjunto de constantes que se las asocia a un tipo 'enum'. En lugar de class utilizamos la palabra clave 'enum' y entre llaves definimos las constantes.

Si queremos crear un tipo enum global debemos utilizar la opción de Eclipse:



Recordemos primero crear un proyecto y luego crear el enum:

```
public enum Operaciones {
    SUMA, RESTA, MULTIPLICACION, DIVISION
}
```

Por convención la lista de valores se escribe en mayúscula y se los separa por una coma.

Luego podemos definir una variable de tipo Operaciones:

```
public class PruebaEnum {  
    public static void main(String[] args) {  
        int valor1 = 10;  
        int valor2 = 20;  
        Operaciones operacion;  
        operacion=Operaciones.RESTA;  
        if (operacion == Operaciones.RESTA) {  
            int resta = valor1 - valor2;  
            System.out.println(resta);  
        }  
        operacion=Operaciones.SUMA;  
        if (operacion == Operaciones.SUMA) {  
            int suma = valor1 + valor2;  
            System.out.println(suma);  
        }  
    }  
}
```

Se crea una variable del tipo Operaciones sin la necesidad del operador new:

```
Operaciones operacion;
```

Luego podemos almacenar cualquiera de los valores enumerados:

```
operacion=Operaciones.RESTA;
```

Podemos utilizar el operador == para verificar si la variable almacena un determinado valor del enum:

```
if (operacion == Operaciones.RESTA) {
```

Podemos cambiar en cualquier momento el valor de la variable 'operacion':

```
operacion=Operaciones.SUMA;
```

En programas grandes el objetivo de los enum es hacer más legible un programa.

Declaración de enum dentro de una clase.

El lenguaje Java permite declarar tipos enum dentro de una clase y agregarles el modificador de acceso public o private.

Problema

Crear un proyecto y dentro del mismo crear dos clases. La primer clase se debe llamar 'Carta', declarar un enum que represente los cuatro palos del mazo y dos atributos uno del tipo de dato enum y el número de carta. Por otro lado declarar una clase llamada 'Mazo' que contenga un arreglo de 8 elementos de tipo 'Carta'. Extraer al azar una carta, mostrarla y según el tipo de palo mostrar la cantidad de puntos que gana.

Programa:

```
public class Carta {  
    public enum Palo {  
        TREBOL, DIAMANTE, CORAZON, PICA  
    };  
  
    private int numero;  
    private Palo palo;
```

```

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

    public void imprimir() {
        System.out.println(numero + " - " + palo.toString().toLowerCase());
    }

    public Palo retornarPalo() {
        return palo;
    }
}

public class Mazo {
    private Carta[] cartas;

    Mazo() {
        cartas = new Carta[8];
        cartas[0] = new Carta(1, Carta.Palo.TREBOL);
        cartas[1] = new Carta(2, Carta.Palo.TREBOL);
        cartas[2] = new Carta(1, Carta.Palo.DIAMANTE);
        cartas[3] = new Carta(2, Carta.Palo.DIAMANTE);
        cartas[4] = new Carta(1, Carta.Palo.PICA);
        cartas[5] = new Carta(2, Carta.Palo.PICA);
        cartas[6] = new Carta(1, Carta.Palo.CORAZON);
        cartas[7] = new Carta(2, Carta.Palo.CORAZON);
    }

    public void imprimir() {
        System.out.println("Listado completo del mazo de cartas");
        for (Carta carta : cartas)
            carta.imprimir();
    }

    public void sacarUnaCartas() {
        System.out.println("Una carta elegida al azar");
        Carta carta = cartas[(int) (Math.random() * 8)];
        carta.imprimir();
        switch (carta.retornarPalo()) {
            case CORAZON:
                System.out.println("Gana 4 puntos");
                break;
            case DIAMANTE:
                System.out.println("Gana 3 puntos");
                break;
            case PICA:
                System.out.println("Gana 2 puntos");
                break;
            case TREBOL:
                System.out.println("Gana 1 puntos");
                break;
        }
    }
}

public static void main(String[] ar) {
    Mazo mazo = new Mazo();
    mazo.imprimir();
    mazo.sacarUnaCartas();
}

```



```
}  
}
```

La clase carta declara con el modificador public el enum 'Palo', así podemos accederlo desde otras clases:

```
public class Carta {  
    public enum Palo {  
        TREBOL, DIAMANTE, CORAZON, PICA  
    };  
}
```

La clase Carta define un atributo del tipo 'Palo':

```
private Palo palo;
```

En la clase Mazo cuando creamos cada carta pasamos al constructor una de los valores enumerados. Como el tipo de dato enumerado se encuentra en la clase Carta debemos primero disponer el nombre de la clase, seguidamente el tipo enum y finalmente la constante del tipo enum:

```
cartas = new Carta[8];  
cartas[0] = new Carta(1, Carta.Palo.TREBOL);  
cartas[1] = new Carta(2, Carta.Palo.TREBOL);  
cartas[2] = new Carta(1, Carta.Palo.DIAMANTE);  
...
```

Dijimos que uno de los objetivos del tipo enum es hacer nuestro programa más legible, luego para mostrar la cantidad de puntos que le corresponde según el tipo de palo podemos emplear una estructura switch:

```
switch (carta.retornarPalo()) {  
    case CORAZON:  
        System.out.println("Gana 4 puntos");  
        break;  
    case DIAMANTE:  
        System.out.println("Gana 3 puntos");  
        break;  
    case PICA:  
        System.out.println("Gana 2 puntos");  
        break;  
    case TREBOL:  
        System.out.println("Gana 1 puntos");  
        break;  
}
```

Como vemos en cada case se compara directamente por cada uno de los valores del enum sin necesidad de anteceder el nombre del enum.

Si utilizamos if es necesario indicar nombre de clase, enum y valor:

```
if (carta.retornarPalo()==Carta.Palo.CORAZON)  
    System.out.println("Gana 4 puntos");
```

Declaración de enum con atributos y métodos

En algunas situaciones podemos crear tipos enum que además de las constantes tengan asociados otros valores cada constante mediante atributos.

Cuando se declara un enum podemos definir constructores y otros métodos a la misma.

Problema

Declarar un tipo enum llamado 'Mes', asociar para cada constante el número de mes que corresponde.

Programa:

```
public enum Mes {
    ENERO(1), FEBRERO(2), MARZO(3), ABRIL(4), MAYO(5), JUNIO(6), JULIO(7),
    AGOSTO(8), SEPTIEMBRE(9), OCTUBRE(10),
    NOVIEMBRE(11), DICIEMBRE(12);
    private final int numero;

    Mes(int numero) {
        this.numero = numero;
    }

    public int retornarNumero() {
        return numero;
    }
}

public class PruebaEnum {
    public static void main(String[] ar) {
        Mes mes1 = Mes.AGOSTO;
        System.out.print(mes1 + " " + mes1.retornarNumero()); // AGOSTO 8
    }
}
```

Este ejemplo nos muestra que un enum es un tipo especial de clase que tiene una enumeración de constantes, pero puede tener asociados atributos y métodos.

Como el constructor tiene un parámetro de tipo int luego cada valor enumerado pasa al constructor un valor int:

ENERO(1)

El valor se almacena en el atributo 'numero':

```
private final int numero;
```

Cuando definimos una variable de tipo 'Mes' almacenamos alguno de sus valores enumerados:

```
Mes mes1 = Mes.AGOSTO;
```

Luego podemos recuperar tanto el valor del tipo enumerado como el entero asociado a dicho valor enumerado:

```
System.out.print(mes1 + " " + mes1.retornarNumero()); // AGOSTO 8
```

Acceder a todos los valores enumerados.

Cuando se declara un enum heredamos un método llamado values que nos retorna un arreglo con todos los valores enumerados de la declaración.

Si queremos mostrar todos los nombres de meses y el número de mes que corresponde podemos modificar el proyecto anterior:

```
public class PruebaEnum {

    public static void main(String[] ar) {
        for (Mes mes : Mes.values())
```

```

        System.out.println(mes + " " + mes.retornarNumero());
    }
}

```

Tenemos un resultado similar a:

Recordemos que la otra forma de recorrer un arreglo en Java es a través de subíndices, la modificación del programa anterior queda:

```

public class PruebaEnum {

    public static void main(String[] ar) {
        Mes[] meses = Mes.values();
        for (int f = 0; f < meses.length; f++)
            System.out.println(meses[f] + " " + meses[f].retornarNumero());
    }

}

```

Como vemos es mucho más legible recorrer un arreglo completo para consultarlo con la primer variante de 'for'.

Tipos de datos primitivos y clases de envoltura en Java

Hemos estado utilizando en todos los conceptos anteriores la mayoría de datos primitivos que provee Java, ahora los enumeraremos a todos.

Recordemos que los tipos de datos primitivos son aquellos que almacenan directamente el valor, a diferencia de los tipos de datos referencia que almacenan la dirección de memoria donde se almacena el dato (los objetos son tipo de datos referencia)

Los tipos de datos primitivos los podemos agrupar en:

Tipos enteros

Según el valor entero máximo a almacenar podemos elegir entre:

- byte: -128 a 127
- short: -32768 a 32767
- int: -2147483648 a 2147483647
- long: -9223372036854775808 a 9223372036854775807

Imaginemos que tenemos que definir un array de 10000 valores enteros. La elección del tipo entero va a ser importante, teniendo en cuenta que un byte ocupa un 1 byte, un short 2 bytes, un int ocupa 4 bytes y un long ocupa 8 bytes.

Si vamos a almacenar en el array edades por ejemplo, lo más conveniente será utilizar el tipo byte y no long.

Tipos reales

- float: [Formato en coma flotante de simple precisión](#)
- double: [Formato en coma flotante de doble precisión](#)

Tipo lógico

- boolean: puede almacenar solo alguno de dos valores (true/false)

Tipo caracter

- char: puede almacenar un único carácter Unicode de 16 bits

Para cada uno de los tipos de datos primitivos existen una clase de envoltura asociada:

Tipo primitivo Clase de envoltura	
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Problema

Imprimir los valores máximos y mínimos que pueden almacenar cada tipo de dato primitivo numéricos. Emplear las clases de envoltura para recuperar dichos valores.

Programa:

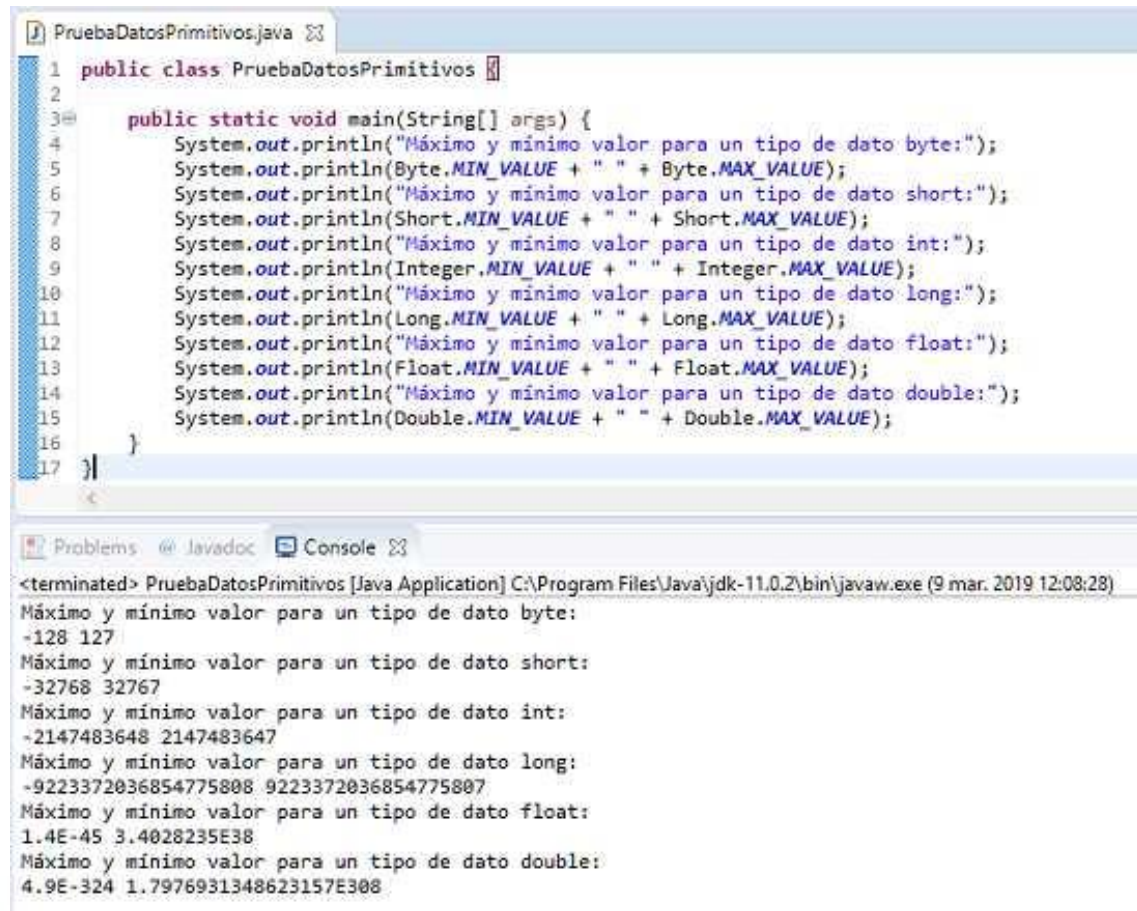
```
public class PruebaDatosPrimitivos {  
  
    public static void main(String[] args) {  
        System.out.println("Máximo y mínimo valor para un tipo de dato byte:");  
        System.out.println(Byte.MIN_VALUE + " " + Byte.MAX_VALUE);  
        System.out.println("Máximo y mínimo valor para un tipo de dato short:");  
        System.out.println(Short.MIN_VALUE + " " + Short.MAX_VALUE);  
        System.out.println("Máximo y mínimo valor para un tipo de dato int:");  
        System.out.println(Integer.MIN_VALUE + " " + Integer.MAX_VALUE);  
        System.out.println("Máximo y mínimo valor para un tipo de dato long:");  
        System.out.println(Long.MIN_VALUE + " " + Long.MAX_VALUE);  
        System.out.println("Máximo y mínimo valor para un tipo de dato float:");  
        System.out.println(Float.MIN_VALUE + " " + Float.MAX_VALUE);  
        System.out.println("Máximo y mínimo valor para un tipo de dato double:");  
    }  
}
```

```

        System.out.println(Double.MIN_VALUE + " " + Double.MAX_VALUE);
    }
}

```

Tenemos como resultado:



The screenshot shows an IDE with a file named 'PruebaDatosPrimitivos.java'. The code defines a public class 'PruebaDatosPrimitivos' with a 'main' method. The 'main' method prints the minimum and maximum values for various primitive data types: byte, short, int, long, float, and double. The output in the console window below the code is as follows:

```

<terminated> PruebaDatosPrimitivos [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (9 mar. 2019 12:08:28)
Máximo y mínimo valor para un tipo de dato byte:
-128 127
Máximo y mínimo valor para un tipo de dato short:
-32768 32767
Máximo y mínimo valor para un tipo de dato int:
-2147483648 2147483647
Máximo y mínimo valor para un tipo de dato long:
-9223372036854775808 9223372036854775807
Máximo y mínimo valor para un tipo de dato float:
1.4E-45 3.4028235E38
Máximo y mínimo valor para un tipo de dato double:
4.9E-324 1.7976931348623157E308

```

Recordemos de conceptos anteriores que hemos utilizado una serie de métodos estáticos de las clases de envoltura como por ejemplo para la conversión de datos:

```

String cadena="30";
int monedas=Integer.parseInt(cadena);

```

Cuando veamos más adelante clases genéricas estaremos obligados a utilizar las clases de envoltorio en lugar de los tipos de datos primitivos.

Caracter de subrayado para mejorar la legibilidad del código.

En Java podemos utilizar el caracter '_' para separar bloques de números cuando inicializamos variables enteras y reales:

```

public static void main(String[] args) {
    int premio=1_000_000;
    System.out.println("Premio mayor:"+premio);
}

```

Estamos definiendo una variable entera con el valor 1 millón, como podemos ver es más legible que escribir:

```
int premio=1000000;
```

No hay diferencias luego de compilar el programa cuando empleamos el caracter '_'.

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar