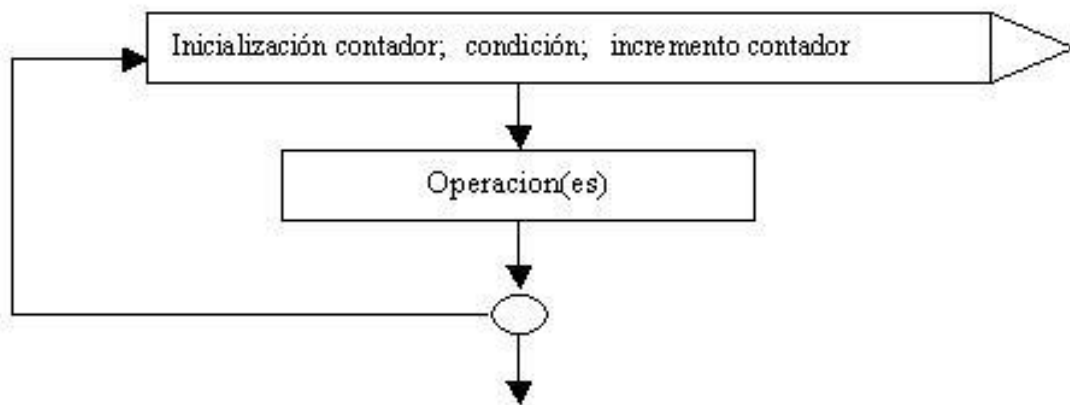


Curso de Java a distancia

Clase 22: Estructura repetitiva 'for' adaptada para recorrer colecciones

En los primeros conceptos de este curso vimos como trabaja la estructura repetitiva for.



En su forma más típica y básica, esta estructura requiere una variable entera que cumple la función de un CONTADOR de vueltas. En la sección indicada como "inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial. En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de un falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa)

Cuando tenemos que iterar por los elementos de una colección (hasta ahora hemos visto los arreglos) accedemos a los mismos por medio de un subíndice que normalmente corresponde al contador de un for:

```
public class SueldoEmpleados {  
  
    public static void main(String[] args) {  
        int []sueldos= {2000, 6000, 7000, 4300};  
        for(int f=0;f<sueldos.length;f++)  
            System.out.println("Sueldo: "+sueldos[f]);  
        int gastos=0;  
        for(int f=0;f<sueldos.length;f++)  
            gastos+=sueldos[f];  
        System.out.println("Gasto total en sueldos de la empresa: "+gastos);  
    }  
}
```

```
}
```

Definimos un vector llamado sueldos y lo inicializamos con 4 elementos:

```
int []sueldos= {2000, 6000, 7000, 4300};
```

Como hemos visto en conceptos anteriores para mostrar los valores almacenados y sumarlos debemos acceder a cada elemento por medio de un subíndice:

```
for(int f=0;f<sueldos.length;f++)  
    System.out.println("Sueldo: "+sueldos[f]);  
int gastos=0;  
for(int f=0;f<sueldos.length;f++)  
    gastos+=sueldos[f];
```

Java introduce una variante de la estructura repetitiva for que nos permite iterar por cada elemento del arreglo con la siguiente sintaxis:

```
public class SueldoEmpleados2 {  
  
    public static void main(String[] args) {  
        int[] sueldos = { 2000, 6000, 7000, 4300 };  
        for (int monto : sueldos)  
            System.out.println("Sueldo: " + monto);  
        int gastos = 0;  
        for (int monto : sueldos)  
            gastos += monto;  
        System.out.println("Gasto total en sueldos de la empresa: " + gastos);  
    }  
}
```

Si bien se utiliza la misma palabra clave 'for', su contenido es muy distinto (tiene un solo argumento):

```
for (int monto : sueldos)  
    System.out.println("Sueldo: " + monto);
```

Previo a los dos puntos definimos una variable del mismo tipo de datos que los elementos del arreglo 'int monto' y luego de los dos puntos indicamos el nombre del arreglo a recorrer.

La variable 'monto' almacena en cada iteración del for un sueldo del arreglo 'sueldos'.

Es decir que la primera vuelta del for la variable 'monto' tiene almacenado el número '2000', la segunda iteración tiene el valor '6000' y así sucesivamente.

En ningún momento accedemos a los elementos del arreglo por medio de un subíndice.

Esta estructura repetitiva no reemplaza a la otra estructura for debido que hay muchos casos donde no podemos resolverla con ésta, por ejemplo no podemos modificar los elementos del arreglo dentro del for, ni saber el índice de la componente accedida.

La sintaxis es mucho más legible que el for clásico y su uso es adecuado cuando debemos consultar todos los elementos de una colección.

Problema

Se desea almacenar los sueldos de operarios. Cuando se ejecuta el programa se debe pedir la cantidad de sueldos a ingresar. Luego crear un arreglo con dicho tamaño. Imprimir todos los sueldos ingresados y mostrar el mayor de ellos.

Programa:

```
import java.util.Scanner;
```

```
public class SueldoEmpleados3 {  
    private Scanner teclado;  
    private int[] sueldos;  
  
    public SueldoEmpleados3() {  
        teclado = new Scanner(System.in);  
        System.out.print("Cuantos sueldos cargará:");  
        int cant;  
        cant = teclado.nextInt();  
        sueldos = new int[cant];  
        for (int f = 0; f < sueldos.length; f++) {  
            System.out.print("Ingrese sueldo:");  
            sueldos[f] = teclado.nextInt();  
        }  
    }  
  
    public void imprimir() {  
        for (int sueldo : sueldos)  
            System.out.println(sueldo);  
    }  
  
    public void sueldoMayor() {  
        int mayor = sueldos[0];  
        for (int sueldo : sueldos)  
            if (sueldo > mayor)  
                mayor = sueldo;  
        System.out.println("El sueldo mayor que paga la empresa es " + mayor);  
    }  
  
    public static void main(String[] ar) {  
        SueldoEmpleados3 se = new SueldoEmpleados3();  
        se.imprimir();  
        se.sueldoMayor();  
    }  
}
```

Como vemos la carga del vector requiere un 'for' clásico para poder modificar los elementos del arreglo accediéndolos mediante su subíndice:

```
for (int f = 0; f < sueldos.length; f++) {  
    System.out.print("Ingrese sueldo:");  
    sueldos[f] = teclado.nextInt();  
}
```

En cambio para imprimir todos los elementos del arreglo y recuperar el mayor elementos podemos utilizar la nueva sintaxis de for:

```
public void imprimir() {  
    for (int sueldo : sueldos)  
        System.out.println(sueldo);  
}  
  
public void sueldoMayor() {  
    int mayor = sueldos[0];  
    for (int sueldo : sueldos)  
        if (sueldo > mayor)  
            mayor = sueldo;  
    System.out.println("El sueldo mayor que paga la empresa es " + mayor);  
}
```

Recorrer una matriz

También podemos utilizar la nueva sintaxis para recorrer una arreglo de dos dimensiones.

Problema

Crear una matriz de $n * m$ filas (cargar n y m por teclado) Imprimir la matriz completa utilizando la estructura for que recorre colecciones.

Programa:

```
import java.util.Scanner;

public class PruebaMatriz1 {
    private Scanner teclado;
    private int[][] mat;

    public PruebaMatriz1() {
        teclado = new Scanner(System.in);
        System.out.print("Cuántas fila tiene la matriz:");
        int filas = teclado.nextInt();
        System.out.print("Cuántas columnas tiene la matriz:");
        int columnas = teclado.nextInt();
        mat = new int[filas][columnas];
        for (int f = 0; f < mat.length; f++) {
            for (int c = 0; c < mat[f].length; c++) {
                System.out.print("Ingrese componente:");
                mat[f][c] = teclado.nextInt();
            }
        }
    }

    public void imprimir() {
        for (int[] fila : mat) {
            for (int elemento : fila)
                System.out.print(elemento + " ");
            System.out.println();
        }
    }

    public static void main(String[] ar) {
        PruebaMatriz1 pm = new PruebaMatriz1();
        pm.imprimir();
    }
}
```

La carga de la matriz no se puede hacer con la nueva sintaxis del for, recordar que solo nos sirve para recorrer la colección.

La impresión de la matriz utilizamos dos for anidados con la nueva sintaxis:

```
public void imprimir() {
    for (int[] fila : mat) {
        for (int elemento : fila)
            System.out.print(elemento + " ");
        System.out.println();
    }
}
```

La variable 'fila' es un vector que almacena en cada iteración una fila de la matriz 'mat':

```
for (int[] fila : mat) {
```

Luego el for interno itera el vector 'fila' y almacena en la variable 'elemento' cada uno de los valores enteros del vector 'fila':

```
        for (int elemento : fila)
            System.out.print(elemento + " ");
        System.out.println();
    }
```

Recorrer un arreglo con elementos de tipo objeto.

Problema

Crear un proyecto y dentro del mismo crear dos clases. La primer clase se debe llamar 'Carta' y definir los atributos palo y numero. Por otro lado declarar una clase llamada 'Mazo' que contenga un arreglo de 6 elementos de tipo 'Carta'.

Imprimir todas las cartas. Imprimir una carta al azar entre las 6 cartas.

Programa:

```
public class Carta {
    private int numero;
    private String palo;

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

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

public class Mazo {
    private Carta []cartas;

    Mazo() {
        cartas=new Carta[6];
        cartas[0]= new Carta(1,"Trebol");
        cartas[1]= new Carta(2,"Trebol");
        cartas[2]= new Carta(3,"Trebol");
        cartas[3]= new Carta(4,"Trebol");
        cartas[4]= new Carta(5,"Trebol");
        cartas[5]= new Carta(6,"Trebol");
    }

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

    public void imprimirUnaAlAzar() {
        System.out.println("Una carta elegida al azar");
        cartas[(int)(Math.random()*6)].imprimir();
    }

    public static void main(String[] ar) {
```

```

        Mazo mazo=new Mazo();
        mazo.imprimir();
        mazo.imprimirUnaAlAzar();
    }
}

```

La ejecución del proyecto genera una salida similar a:

```

Mazo.java
1 public class Mazo {
2     private Carta []cartas;
3
4     Mazo() {
5         cartas=new Carta[6];
6         cartas[0]= new Carta(1,"Trebol");
7         cartas[1]= new Carta(2,"Trebol");
8         cartas[2]= new Carta(3,"Trebol");
9         cartas[3]= new Carta(4,"Trebol");
10        cartas[4]= new Carta(5,"Trebol");
11        cartas[5]= new Carta(6,"Trebol");
12    }
13
14    public void imprimir() {
15        System.out.println("Listado completo del mazo de cartas");
16        for(Carta carta: cartas)
17            carta.imprimir();
18    }
19
20    public void imprimirUnaAlAzar() {
21        System.out.println("Una carta elegida al azar");
22        cartas[(int)(Math.random()*6)].imprimir();
23    }
24
25
26    public static void main(String[] ar) {
27        Mazo mazo=new Mazo();
28        mazo.imprimir();
29        mazo.imprimirUnaAlAzar();
30    }
31
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

Carta.java
1 public class Carta {
2     private int numero;
3     private String palo;
4
5     Carta(int numero, String palo) {
6         this.numero = numero;
7         this.palo = palo;
8     }
9
10    public void imprimir() {
11        System.out.println(numero + " - " + palo);
12    }
13
14 }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

Problems  javadoc  Console
<terminated> Mazo [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (6 mar. 2019 19:55:57)
Listado completo del mazo de cartas
1 - Trebol
2 - Trebol
3 - Trebol
4 - Trebol
5 - Trebol
6 - Trebol
Una carta elegida al azar
4 - Trebol

```

En el método imprimir de la clase 'Mazo' hemos utilizado el for alternativo para recorrer el arreglo:

```

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

```

En cada iteración del for la variable 'carta' almacena un elemento del arreglo 'cartas'.

Atributos estáticos de una clase

En los primeros conceptos de POO (Programación Orientada a Objetos) dijimos que los atributos que se definen en una clase reservan espacio en forma independiente para cada instancia de la misma.

A diferencia de los anteriores los atributos estáticos tienen un comportamiento muy distinto a los atributos vistos hasta el momento. Un atributo estático reserva espacio para el mismo indistintamente que definamos un objeto de dicha clase. En caso de crear varios objetos de dicha clase todas las instancias acceden al mismo atributo estático.

Para declarar un atributo de tipo estático agregamos el modificador 'static' en su definición:

```
public class Matematica {  
    public static float PI = 3.1416f;  
}  
  
public class Prueba {  
    public static void main(String[] ar) {  
        System.out.println(Matematica.PI);  
    }  
}
```

La clase Matematica define un atributo estático:

```
public static float PI = 3.1416f;
```

Luego podemos acceder a dicho atributo directamente a través del nombre de la clase sin tener que crear un objeto de la clase 'Matematica':

```
System.out.println(Matematica.PI);
```

Problema

Definir un atributo estático que almacene la cantidad de objetos creados de dicha clase.

Programa:

```
public class Persona {  
    private String nombre;  
    private int edad;  
    public static int cantidad;  
  
    public Persona(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
        cantidad++;  
    }  
  
    public void imprimir() {  
        System.out.println(nombre + " " + edad);  
    }  
}  
  
public class PruebaPersona {  
  
    public static void main(String[] args) {  
        System.out.println("Valor del atributo estático cantidad:" + Persona.cantidad);  
        Persona per1 = new Persona("Juan", 30);  
        per1.imprimir();  
        System.out.println("Valor del atributo estático cantidad:" + Persona.cantidad);  
        Persona per2 = new Persona("Ana", 20);  
        per2.imprimir();  
        System.out.println("Valor del atributo estático cantidad:" + Persona.cantidad);  
        Persona per3 = new Persona("Luis", 10);
```

```

        per3.imprimir();
        System.out.println("Valor del atributo estático cantidad:" + Persona.cantidad);
    }
}

```

Un atributo estático de tipo entero se inicializa en cero cuando lo definimos (lo definimos de tipo public para poder acceder a su valor desde afuera de la clase):

```
public static int cantidad;
```

Luego podemos consultarlo inclusive antes de crear un objeto de dicha clase:

```
System.out.println("Valor del atributo estático cantidad:" + Persona.cantidad);
```

Cada vez que creamos un objeto de la clase Persona:

```
Persona per1 = new Persona("Juan", 30);
```

Se incrementa en uno el atributo estático 'cantidad':

```

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

```

No importa cuantos objeto de la clase Persona se creen luego existe un solo atributo cantidad:

```

Persona per1 = new Persona("Juan", 30);
Persona per2 = new Persona("Ana", 20);
Persona per3 = new Persona("Luis", 10);

```

Creamos 3 objetos de la clase Persona y los tres comparten el mismo atributo 'cantidad'.

Un atributo estático se lo puede acceder tanto a través del nombre de la clase como a través del nombre de un objeto:

```

Persona per1 = new Persona("Juan", 30);
System.out.println("Valor del atributo estático cantidad:" + Persona.cantidad); // 1
System.out.println("Valor del atributo estático cantidad:" + per1.cantidad); // 1

```

Métodos estáticos de una clase

Así como una clase puede tener atributos estáticos, Java también permite definir métodos estáticos que se crean independientemente a la definición de objetos. Un método estático puede llamarse sin tener que crear un objeto de dicha clase.

Igual que los atributos estáticos, un método estático tiene ciertas restricciones:

- No puede acceder a los atributos de la clase (salvo que sean estáticos)
- No puede utilizar el operador this, ya que este método se puede llamar sin tener que crear un objeto de la clase.
- Puede llamar a otro método siempre y cuando sea estático.
- Un método estático es lo más parecido a lo que son las funciones en los lenguajes estructurados (con la diferencia que se encuentra encapsulado en una clase)

Si recordamos cada vez que creamos un programa en Java debemos especificar el método main:


```
public static void main(String[] args)
```

El método main es estático para que la máquina virtual de Java pueda llamarlo directamente sin tener que crear un objeto de la clase que lo contiene.

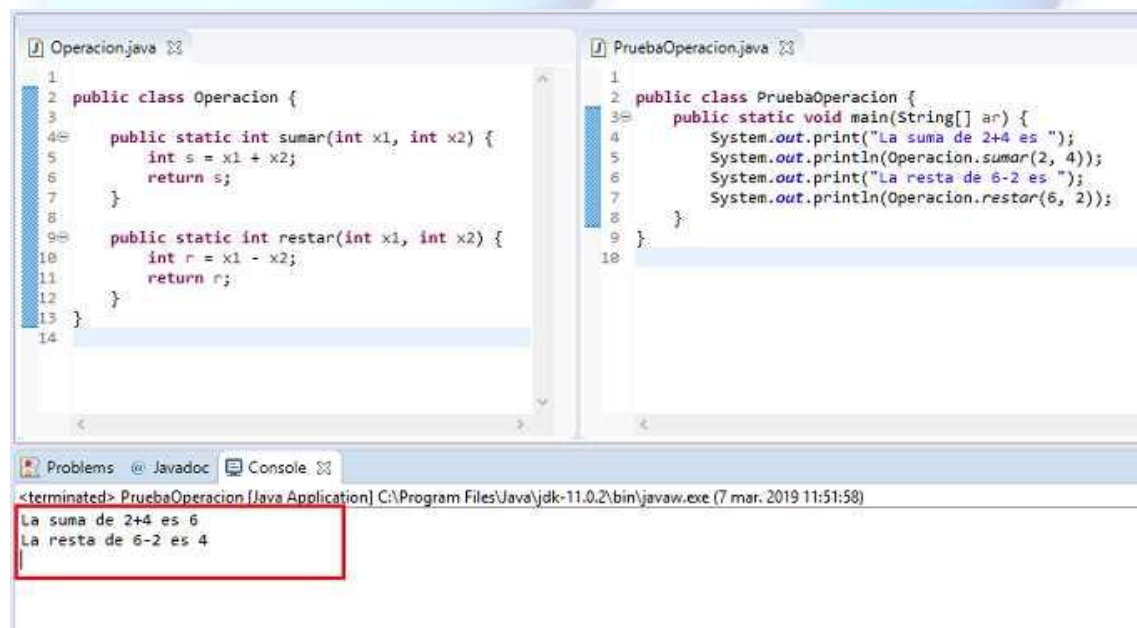
Problema

Implementar una clase llamada Operacion. Definir dos métodos estáticos que permitan sumar y restar dos valores enteros.

Programa:

```
public class Operacion {  
  
    public static int sumar(int x1, int x2) {  
        int s = x1 + x2;  
        return s;  
    }  
  
    public static int restar(int x1, int x2) {  
        int r = x1 - x2;  
        return r;  
    }  
}  
  
public class PruebaOperacion {  
    public static void main(String[] ar) {  
        System.out.print("La suma de 2+4 es ");  
        System.out.println(Operacion.sumar(2, 4));  
        System.out.print("La resta de 6-2 es ");  
        System.out.println(Operacion.restar(6, 2));  
    }  
}
```

La ejecución del proyecto genera una salida:



The screenshot shows an IDE with two open files: `Operacion.java` and `PruebaOperacion.java`. The `Operacion.java` file contains two static methods: `sumar` and `restar`. The `PruebaOperacion.java` file contains a `main` method that calls these two methods. Below the code editor, the console output is displayed, showing the results of the program execution: "La suma de 2+4 es 6" and "La resta de 6-2 es 4".

```
Operacion.java  
1  
2 public class Operacion {  
3  
4     public static int sumar(int x1, int x2) {  
5         int s = x1 + x2;  
6         return s;  
7     }  
8  
9     public static int restar(int x1, int x2) {  
10        int r = x1 - x2;  
11        return r;  
12    }  
13 }  
14  
  
PruebaOperacion.java  
1  
2 public class PruebaOperacion {  
3     public static void main(String[] ar) {  
4         System.out.print("La suma de 2+4 es ");  
5         System.out.println(Operacion.sumar(2, 4));  
6         System.out.print("La resta de 6-2 es ");  
7         System.out.println(Operacion.restar(6, 2));  
8     }  
9 }  
10  
  
Problems @ Javadoc Console  
<terminated> PruebaOperacion [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (7 mar. 2019 11:51:58)  
La suma de 2+4 es 6  
La resta de 6-2 es 4
```

Agregamos la palabra clave 'static' antes de indicar el valor que retorna el método:

```
public static int sumar(int x1, int x2) {  
    int s = x1 + x2;  
    return s;  
}
```

Luego cuando llamamos al método estático lo hacemos antecediendo el nombre de la clase:

```
System.out.println(Operacion.sumar(2, 4));
```

Es importante notar que no se crean objetos de la clase Operacion para poder llamar a los métodos sumar y restar.

No es obligatorio que una clase defina todos sus métodos estáticos, veremos con un ejemplo como podemos tener métodos estáticos y no estáticos.

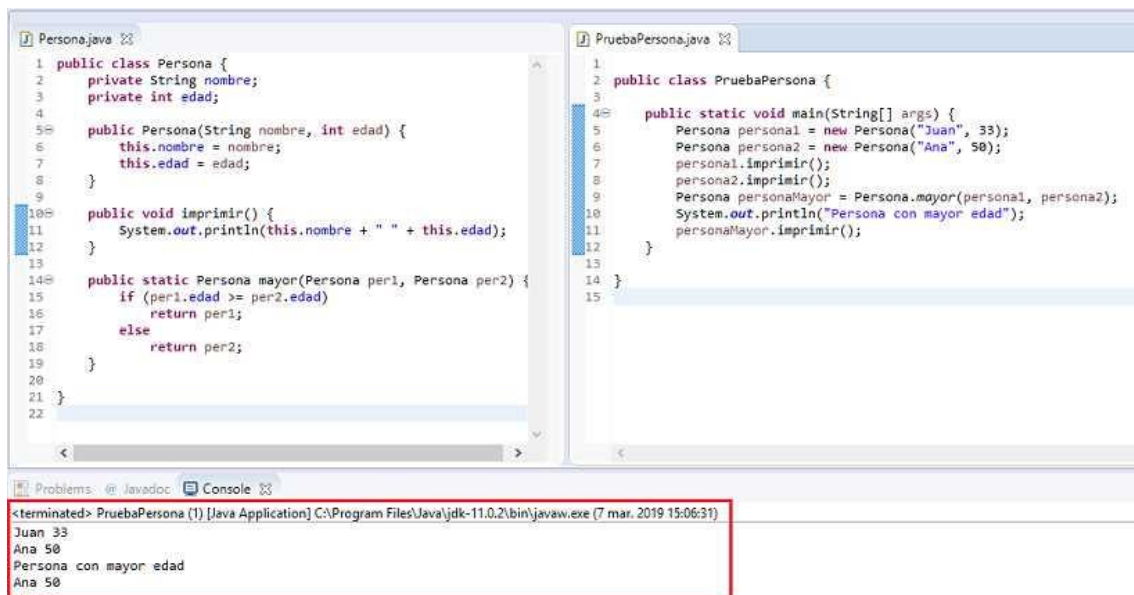
Problema

Declarar una clase Persona con los atributos nombre y edad. Definir un método estático que reciba como parámetros dos objetos de la clase Persona y me retorne la que tiene una edad mayor, si son iguales retorne cualquiera de las dos.

Programa:

```
public class Persona {  
    private String nombre;  
    private int edad;  
  
    public Persona(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
  
    public void imprimir() {  
        System.out.println(this.nombre + " " + this.edad);  
    }  
  
    public static Persona mayor(Persona per1, Persona per2) {  
        if (per1.edad >= per2.edad)  
            return per1;  
        else  
            return per2;  
    }  
}  
  
public class PruebaPersona {  
  
    public static void main(String[] args) {  
        Persona persona1 = new Persona("Juan", 33);  
        Persona persona2 = new Persona("Ana", 50);  
        persona1.imprimir();  
        persona2.imprimir();  
        Persona personaMayor = Persona.mayor(persona1, persona2);  
        System.out.println("Persona con mayor edad");  
        personaMayor.imprimir();  
    }  
}
```

La ejecución del proyecto genera una salida:



```
1 public class Persona {
2     private String nombre;
3     private int edad;
4
5     public Persona(String nombre, int edad) {
6         this.nombre = nombre;
7         this.edad = edad;
8     }
9
10    public void imprimir() {
11        System.out.println(this.nombre + " " + this.edad);
12    }
13
14    public static Persona mayor(Persona per1, Persona per2) {
15        if (per1.edad >= per2.edad)
16            return per1;
17        else
18            return per2;
19    }
20 }
21
22
```

```
1
2 public class PruebaPersona {
3
4     public static void main(String[] args) {
5         Persona persona1 = new Persona("Juan", 33);
6         Persona persona2 = new Persona("Ana", 50);
7         persona1.imprimir();
8         persona2.imprimir();
9         Persona personaMayor = Persona.mayor(persona1, persona2);
10        System.out.println("Persona con mayor edad");
11        personaMayor.imprimir();
12    }
13
14 }
15
```

Problems: 0 Javadoc Console

<terminated> PruebaPersona (1) [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (7 mar. 2019 15:06:31)

Juan 33
Ana 50
Persona con mayor edad
Ana 50

El método 'mayor' no puede acceder a los atributos edad y nombre de la clase Persona, pero si puede acceder al atributo 'edad' de cada uno de los parámetros:

```
public static Persona mayor(Persona per1, Persona per2) {
    if (per1.edad >= per2.edad)
        return per1;
    else
        return per2;
}
```

El método 'mayor' retorna la referencia de alguno de los dos objetos que llegan como parámetro.

Para llamar al método mayor lo hacemos a través de la clase Persona y le pasamos la referencia de dos instancias de la clase Persona llamados 'persona1' y 'persona2' que previamente creamos:

```
Persona personaMayor = Persona.mayor(persona1, persona2);
System.out.println("Persona con mayor edad");
personaMayor.imprimir();
```

No es la única forma que podemos resolver este problema, si no queremos utilizar un método estático en la clase Persona luego la solución es:

Programa:

```
public class Persona {
    private String nombre;
    private int edad;

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

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

    public Persona mayor(Persona per) {
```

```

        if (this.edad >= per.edad)
            return this;
        else
            return per;
    }
}

public class PruebaPersona {

    public static void main(String[] args) {
        Persona persona1 = new Persona("Juan", 33);
        Persona persona2 = new Persona("Ana", 50);
        persona1.imprimir();
        persona2.imprimir();
        Persona personaMayor = persona1.mayor(persona2);
        System.out.println("Persona con mayor edad");
        personaMayor.imprimir();
    }
}

```

Ahora el método 'mayor' no es estático y recibe como parámetro un objeto de la clase Persona y lo compara con el atributo 'edad' de la instancia de la clase respectiva:

```

public Persona mayor(Persona per) {
    if (this.edad >= per.edad)
        return this;
    else
        return per;
}

```

Para llamar al método 'mayor' debemos hacerlo obligatoriamente a partir de una instancia de la clase 'Persona':

```

Persona personaMayor = persona1.mayor(persona2);
System.out.println("Persona con mayor edad");
personaMayor.imprimir();

```

Ejemplos de métodos estáticos en las clases estándares de Java

Existen una gran cantidad de clases en los paquetes que suministra Java que contienen métodos estáticos, pasemos a recordar muchos que vimos en conceptos anteriores y otros nuevos.

- La clase Math tiene una gran cantidad de métodos estáticos:

```

sin, cos, tan, asin, acos, atan, toRadians, toDegrees, exp, log, log10, sqrt, cbrt,
IEEEremainder, ceil, floor, rint, atan2, pow, round, random, addExact, subtractExact,
multiplyExact, incrementExact, decrementExact, negateExact, toIntExact, multiplyFull,
multiplyHigh, floorDiv, floorMod, abs, max, min, fma, ulp, signum, sinh, cosh, tanh,
hypot, expm1, log1p, copySign, getExponent, nextAfter, nextUp, nextDown, scalb,
powerOfTwoD, powerOfTwoF

```

En Eclipse cuando escribimos el nombre del método estático aparece una 's' en el ícono, además del nombre, parámetros y descripción del mismo:

El objetivo de mostrar los métodos estáticos de las clases más comunes del API de Java es hacer notar que son ampliamente utilizados.



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