

**Asignatura:** Programación Orientada a objetos

**Carrera:** ingeniería en informática

**Docente:** MARIA JACINTA MARTINEZ CASTILLO

**Nombre del tema:** METODOS

**Nombres:** Kevin Alan Ortiz Flores  
Ángel de Jesús Contreras Herrera

**Numero De control:** 21010207  
21010180

**Grupo:** 2a3B

**Fecha de entrega:** 17 /Abril/ 2023

## **Introducción**

En esta tercera practica dejamos de realizar constructores sin parámetros ya que estos ejercicios tenían diferentes indicaciones, empezamos aprendiendo a poner private a los set y cuál era la diferencia de tenerlos en público, ya que al tenerlos en private solo se pueden llamar en la misma clase lo cual para poder mostrarlos en otra clase debemos crear un método en publico que llame al set, realizar operaciones que se necesiten y así obtener los valores del set privado mediante

un método público y con un toString() mostrar los datos de los parámetros del constructor e implementar los métodos públicos como privados que hemos realizado.

Veremos los atributos constantes, estos se llaman así porque una vez que se llenen, durante todo el programa tendrá ese valor y nunca más se podrá modificar, la forma de hacer un atributo constante es mediante un "final" es por ejemplo: `private final String nombre;` Esto se hace por que hay atributos que tienen datos fijos como el nombre, todas las personas tenemos un nombre, pero nunca se nos cambia el nombre cada día o a cada hora, por eso realizamos un final para tener un atributo constante.

Realizaremos métodos sobrecargados estos son como los constructores sobrecargados, con la diferencia que ahora son métodos, tienen el mismo nombre, pero diferentes parámetros la forma correcta de hacer un método sobrecargado es: `public void NombreDelMetodo (parametros);`

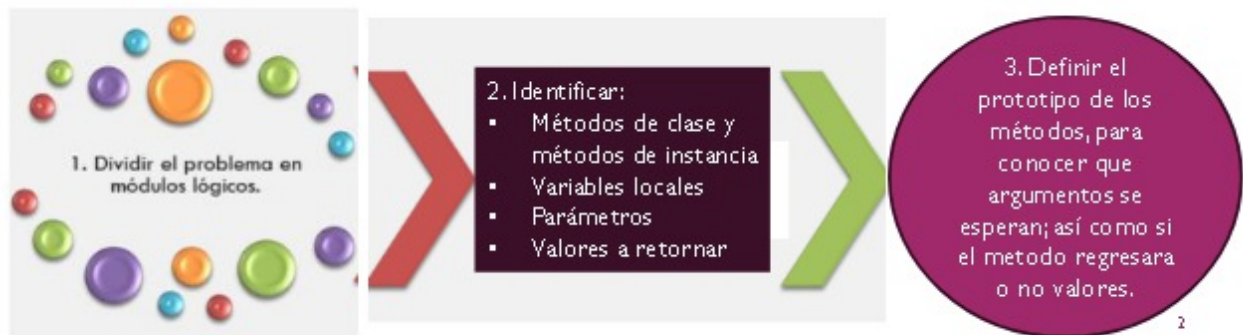
Haremos constructores sobrecargados pero con la particular diferencia de que usaremos un "this" para relacionar o mejor dicho obtener los parámetros de otros constructores sobrecargados anteriormente hechos, con esto podremos usar un único constructor donde llame todos los parámetros y mostrarlos, ejemplo: `public NombreDelConstructor (parámetro){`

`this( parámetros del constructor sobrecargado anterior );`

Por último, veremos las variables estáticas, las ocuparemos para los contadores que utilizaremos, esta variable se puede modificar desde cualquier lugar de la clase e incluso en otras clases, esta variable le pertenece a la clase, no al objeto la forma de hacer una variable estática es: `private static int contador=0;`

### Competencia específica:

Comprende y aplica los diferentes tipos de métodos, tomando en cuenta el ámbito y tiempo de vida de los datos durante la ejecución de un programa.



## Marco teórico:

### 3.1 Definición de un método.

Un método es una abstracción de una operación que puede hacer o realizarse con un objeto. Una clase puede declarar cualquier número de métodos que lleven a cabo operaciones de lo más variado con los objetos. En esta sección los métodos se clasifican en dos grupos: los métodos *de instancia* y los métodos *de clase*.

### 3.2 Estructura de un método.

Un método en Java consta de los siguientes componentes:

1. Modificadores de acceso: indican quién puede acceder al método (public, private, protected o sin modificador).
2. Tipo de retorno: el tipo de datos que devuelve el método o "void" si no devuelve ningún valor.
3. Nombre del método: el nombre que se le da al método.
4. Parámetros: si el método recibe algún dato como entrada, se especifica el tipo y nombre de los parámetros entre paréntesis. Si no recibe parámetros, se deja vacío.
5. Cuerpo: el código que se ejecuta cuando se llama al método, delimitado por llaves {}.

Un ejemplo de la estructura básica de un método en Java sería:

```
public int sumar(int num1, int num2) {  
    int resultado = num1 + num2;  
}
```

```
    return resultado;  
}
```

Este método tiene como modificador de acceso "public", un tipo de retorno "int", un nombre de método "sumar", y dos parámetros de tipo "int" llamados "num1" y "num2". El cuerpo del método realiza una operación de suma y devuelve el resultado.

### 3.3 Valor de retorno.

Los métodos con valor de retorno son módulos de programa que pueden recibir datos de entrada por medio de variables locales (parámetros) y posteriormente retorna un resultado al punto donde es llamado. Este tipo de métodos se utiliza para operar cualquier tipo de proceso que produzca un resultado. Los métodos con valor de retorno se clasifican en dos tipos:

- Métodos con valor de retorno sin parámetros.
- Métodos con valor de retorno con parámetros.

#### 1. Métodos con valor de retorno sin parámetros

Este tipo de métodos no reciben datos de entrada; pero de igual forma retornan un resultado al punto donde son llamados.

Su definición es de la siguiente manera:

```
tipo nombreMetodo(){  
    Declaracion de variables locales  
    Cuerpo del método  
    return valor;}  

```

Donde:

- valor: Es el valor que es retornado por el método.
- tipo: Es el tipo del valor de retorno.

Invocación (llamado):

```
variable = nombreMetodo();
```

Donde:

- variable: Es la variable que recibe el valor retornado por el método.

#### 2. Métodos con valor de retorno con parámetros

Este tipo de métodos reciben datos de entrada por medio de parámetros y retornan un resultado al punto de su llamado. Su definición es de la siguiente manera:

```
tipo nombreMetodo(tipo1 p, tipo2 q, tipo3 r, ...){  
    Declaración de variables locales  
    Cuerpo del método  
    return valor;  
}
```

```
}
```

Donde:

tipo: Es el tipo de valor de retorno. p, q, r, ... : son los nombres de los parámetros.

tipo1, tipo2, tipo3, ... : son los tipos de datos de los parámetros. valor: es el valor de retorno.

Invocación (llamado):

```
variable = nombreMetodo(v1, v2, v3, ...);
```

Donde:

variable: Es el nombre del método llamado.

v1, v2, v3, ... : Son los valores dados a los parámetros.

### 3.4 Declaración de un método.

Un método consta de un encabezado y un cuerpo. Para declarar el encabezado de un método, basta con escribir el tipo que retorna, seguido del nombre del método y entre paréntesis la lista de parámetros.

Ejemplos:

- void imprimir(); este método no retorna nada y no tiene parámetros.
- int sumar(int a, int b); este método recibe dos parámetros de tipo int y retorna un int.

El cuerpo de un método, está dado por una secuencia de instrucciones separados por comas, dentro de una llave que abre y otra que cierra { }. La secuencia puede ser vacía { }.

#### 3.4.1 De clase

En principio, los métodos de clase no operan sobre las variables de instancia de los objetos. Los métodos de clase pueden trabajar con las variables de clase pero no pueden acceder a las variables de instancia declaradas dentro de la clase, a no ser que se crea una nueva instancia y se acceda a las variables de instancia a través del nuevo objeto. Los métodos de clase también pueden ser llamados precediendolos con el identificador de la clase, sin necesidad de utilizar el de una instancia.

```
IdClase.idMetodo(parametros); // Llamada típica a un método de clase
```

La palabra `static` determina la declaración de un método de clase. Por defecto, si no se indica la palabra `static`, el método declarado se considera un método de instancia.

### 3.4.2 De Instancia

Un método de instancia en Java es un método que pertenece a una instancia particular de una clase. Esto significa que sólo puede ser invocado en una instancia de la clase, no en la clase misma. Un método de instancia se define dentro de la clase junto a los demás métodos, pero no tiene la palabra clave `'static'` en su definición.

La sintaxis para declarar un método de instancia es la misma que para declarar cualquier otro método, con la excepción de que no lleva la palabra clave `'static'`. Ejemplo:

```
public class MiClase {  
    //atributos de la clase  
  
    public void miMetodo(String parametro1, int parametro2){  
  
        //cuerpo del método  
    }  
  
    //otros métodos de instancia y/o clase  
}
```

En este ejemplo, `'miMetodo'` es un método de instancia de la clase `'MiClase'`, ya que no tiene la palabra clave `'static'`. Esto significa que sólo puede ser invocado desde una instancia de `'MiClase'`.

### 3.5 Ámbito y tiempo de vida de variables.

- El ámbito de una variable u objeto es el espacio del programa en el que esa variable existe. Por ello, se habla de “ámbito de vida”
- De forma general (hay excepciones que veremos más adelante), la vida de una variable comienza con su declaración y termina en el bloque en el que fue declarada (los bloques de código se delimitan por llaves: `{}`). Por ejemplo, ¿cuál es el ámbito de la variable `'radio'` y del vector `'args'`?

## Los principales tipos de ámbitos son:

- **Ámbito de objeto.** Los atributos de un objeto (que no son static) viven en el espacio de vida del objeto y son accesibles por cualquier método del objeto (siempre que el método no sea static). Por ello, a veces se llaman variables de objeto o variables de instancia

**Ámbito de método.** Variables y objetos declarados en un método. Su ámbito de vida se ciñe al método en el que fueron declaradas, por ello a veces se llaman variables de método o función

- **Ámbito de clase.** Las variables static viven con independencia de que hayamos hecho instancias de la clase. Podemos acceder a ellas (si son públicas) usando el nombre de la clase y viven desde que se declara la clase, por ello se llaman variables de clase

## 3.6 Argumentos y paso de parámetros

Las variables en la lista de parámetros se separan con comas. Los parámetros de la lista en la especificación del método son llamados parámetros formales. Cuando un método es llamado, estos parámetros formales son reemplazados por los parámetros actuales. Los parámetros actuales deben ser equivalentes en tipo, orden y número a los parámetros formales. Cuando es invocado un método con un parámetro de tipo primitivo, tal como "int", el valor del parámetro actual es pasado al método. El valor actual de la variable fuera del método no es afectado, independientemente de los cambios hechos al parámetro formal dentro del método.

## 3.7 Puntero this.

La variable de referencia 'this' apunta al objeto actual en el programa Java . Por lo tanto, si desea acceder a cualquier miembro o función del objeto actual, puede hacerlo utilizando la referencia 'this'. La referencia 'esto' generalmente se denomina 'este puntero', ya que apunta al objeto actual. El 'este puntero' es útil cuando hay algún nombre para los atributos y parámetros de la clase. Cuando surge una situación de este tipo, el 'este puntero' elimina la confusión, ya que podemos acceder a los parámetros mediante el puntero 'este'.

## ¿Cuándo usar 'esto' en Java?

En Java, el término 'esto' tiene los siguientes usos:

La referencia 'esto' se utiliza para acceder a la variable de instancia de clase. Incluso puede pasar 'esto' como argumento en la llamada al método.

'This' también se puede utilizar para invocar implícitamente el método de clase actual.

- Si desea devolver el objeto actual del método, utilice 'esto'.
- Si desea invocar el constructor de la clase actual, se puede usar 'this'.
- El constructor también puede tener 'this' como argumento.

La palabra clave 'esto' es una referencia al objeto actual en el programa Java. Se puede utilizar para evitar confusiones derivadas de los mismos nombres para variables de clase (variables de instancia) y parámetros de método.

Puede usar el puntero 'this' de muchas formas, como acceder a variables de instancia, pasar argumentos al método o constructor, devolver el objeto, etc. El puntero 'this' es una palabra clave importante en Java y es una función útil para acceder al objeto actual y sus miembros y funciones.

### 3.8 Sobrecarga de métodos.

Tanto en Java como en C++ es posible definir varios métodos para la misma clase con el mismo nombre, pero con diferencias en: En la lista de argumentos:

- El número
- El tipo de los argumentos
- o ambos

Esto se conoce como sobrecarga de métodos.

Si se tratan de crear dos métodos con la misma interfaz y diferentes tipos de retorno, la clase no se compila.

- Los constructores, como cualquier otro método, pueden ser sobrecargados:

```
class Circulo {  
    private double radio;  
  
    private double PI = 3.1416; /***** Constructores sobrecargados ***/  
    public Circulo() { }  
    public Circulo( double nuevoRadio ) {  
        setRadio( nuevoRadio ); }  
    public Circulo( Circulo circulo ) {  
        setRadio( circulo.getRadio() );
```



### 3.9 Constructores y destructores

Un constructor es un método especial de una clase, este método es llamado automáticamente al crear un objeto de esa clase. La función del constructor es iniciar el objeto.

- Un constructor se identifica porque tiene el mismo nombre que la clase a la que pertenece.
- Un constructor no puede retornar ningún valor.
- Toda clase de objetos contiene al menos un constructor, aun cuando no se haya definido ninguno se crea uno por defecto al momento de crear un objeto.
- Un constructor por defecto se puede declarar o puede ser omitido al crear una clase.
- Se puede crear uno, dos o más constructores según se necesite.

Existen 3 diferentes tipos de constructores en java.

- 1.- Constructor por defecto.
- 2.- Constructor de copia.
- 3.- Constructor común (Con argumentos).

#### **Constructor de copia.**

El constructor de copia se utiliza para inicializar a un objeto con otro objeto de la misma clase. Ejemplo

```
public class Persona {  
    private String nombre;  
    private String apellidoPaterno;  
    private String apellidoMaterno;  
    //Constructor por defecto  
    public Persona(){  
    }  
    //Constructor copia  
    public Persona(Persona persona){
```

```
        this.nombre=persona.nombre;
        this. apellidoPaterno=persona. ApellidoPaterno;
        this. apellidoMaterno=persona. ApellidoMaterno;
    }
}
```

### Constructor común.

El constructor común es aquel que recibe parámetros para asignarles los valores iniciales a un objeto, se crea de la siguiente manera.

```
public class Persona {
    private String nombre;
    private String apellidoPaterno;
    private String apellidoMaterno;
    //Constructor común
    public Persona(String nom, String app, String apm){
        this.nombre=nom;
        this. apellidoPaterno=app;
        this. apellidoMaterno=apm;
    }
}
```

### **Destruyores:**

Un destructor es un método opuesto a un constructor, este método en lugar de crear un objeto lo destruye liberando la memoria de nuestra computadora para que pueda ser utilizada por alguna otra variable u objeto.

En java no existen los destructores, esto es gracias al recolector de basura de la máquina virtual de java. Como su nombre lo dice, el recolector de basura recolecta todas las variables u objetos que no se estén utilizando y que no haya

ninguna referencia a ellos por una clase en ejecución, liberando así automáticamente la memoria de nuestra computadora.

Aunque Java maneja de manera automática el recolector de basura, el usuario también puede decir en qué momento Java pase el recolector de basura con la instrucción.

Notas:

- Una clase solamente puede tener un destructor.
- Los destructores no pueden heredarse o sobrecargarse.
- Los destructores no pueden invocarse, sino que son invocados automáticamente.
- Un destructor no acepta modificadores ni parámetros.

`System.gc();`

Aunque la instrucción anterior poco o casi nunca se utiliza es importante saber que se puede llamar en cualquier momento.

### **Material y Equipo:**

✓ Computadora Windows 8.1 pro con 4 GB de RAM y procesador Intel Pentium CPU J2900 2.41 GHz

✓ JDK 15.0.2 – NetBeans IDE 8.2 y Eclipse IDE 2021-09

✓ Videos del canal [Programación ATS](#)

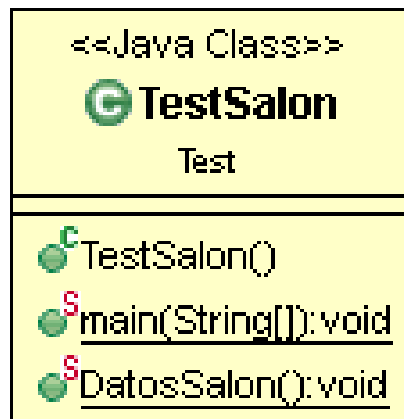
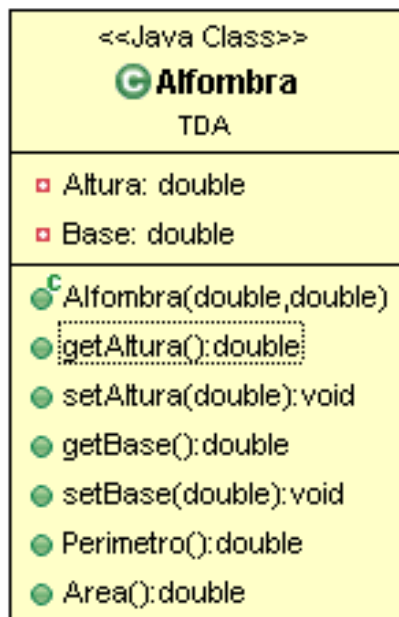
### **Desarrollo de la practica:**

1. Entrar a nuestro IDE favorito para crear nuevos proyectos.
2. Crearemos un proyecto llamado Tema3POO
3. Con este proyecto haremos 3 tipos de paquetes llamados: Tools, TDA y Test
4. En el paquete Tools crearemos una clase donde ira la plantilla de entradas y salidas que tendrá como nombre EntradaSalida

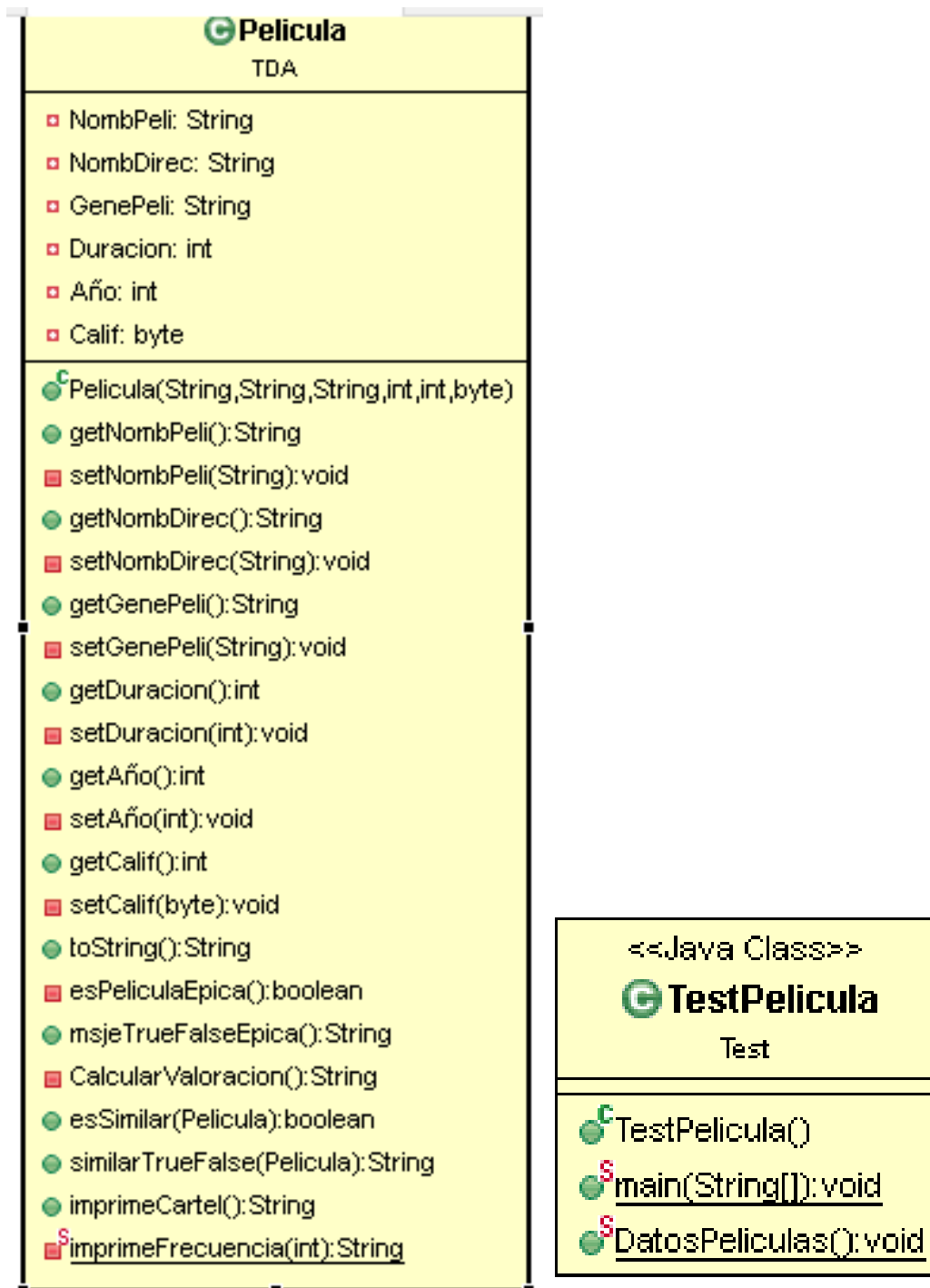
5. En el paquete TDA tendrá todas las clases que obtendrá métodos de cada ejercicio, así como también importaremos la clase EntradaSalida del paquete Tools
6. En el paquete Test, es donde ira mayormente la clase principal o main() para ejecutar el programa y se deberá importar el paquete TDA con su respectiva clase así como también el paquete Tools
7. Leer correctamente las indicaciones de los problemas para realizar los códigos, ya que cada problema especifica que debe tener la clase ya sea constructores sobrecargados, métodos sobrecargados, variables estáticas, ocupar un final en atributos, entre otras cosas que puedan indicar.
8. Hacer una corrida de escritorio para verificar el funcionamiento del código así como también revisar los diagramas de clase para saber si esta completo el código

### **Diagramas de clase:**

#### **Ejercicio 1:**



## Ejercicio 2:














### Ejercicio 3:

&lt;&lt;Java Class&gt;&gt;

 **ConversorMetro**

TDA

 metro: double <sup>F</sup>METROS\_CM: int <sup>F</sup>METROS\_MILIM: double <sup>F</sup>METROS\_PULGADAS: double <sup>F</sup>METROS\_PIES: double <sup>F</sup>METROS\_YARDAS: double <sup>C</sup>ConversorMetro(double) setMetro(double): void convMetrosCemtímetros(): double convMetrosMilímetros(): double convMetrosPulgadas(): double convMetrosPies(): double convMetrosYardas(): double conversionMetros(): void

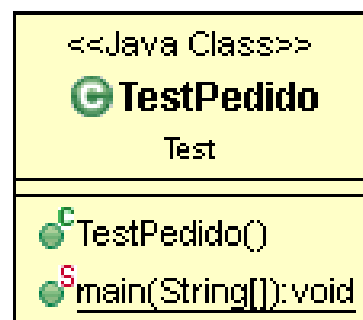
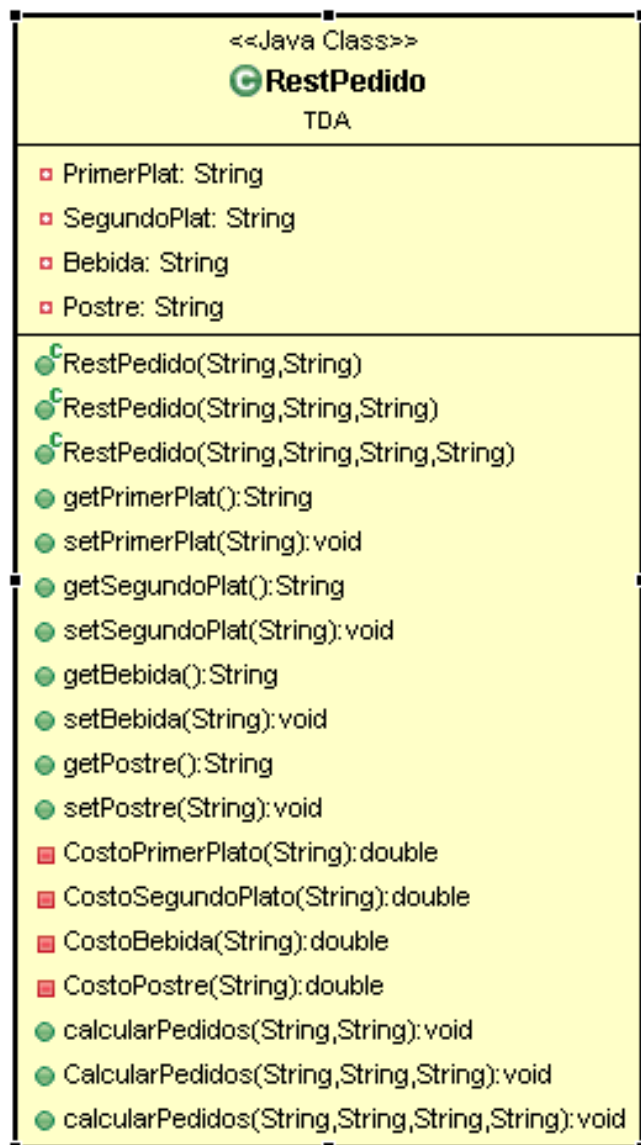
&lt;&lt;Java Class&gt;&gt;

 **TestConvertor**

Test

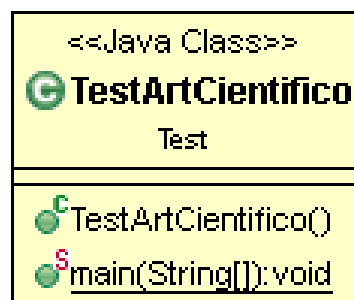
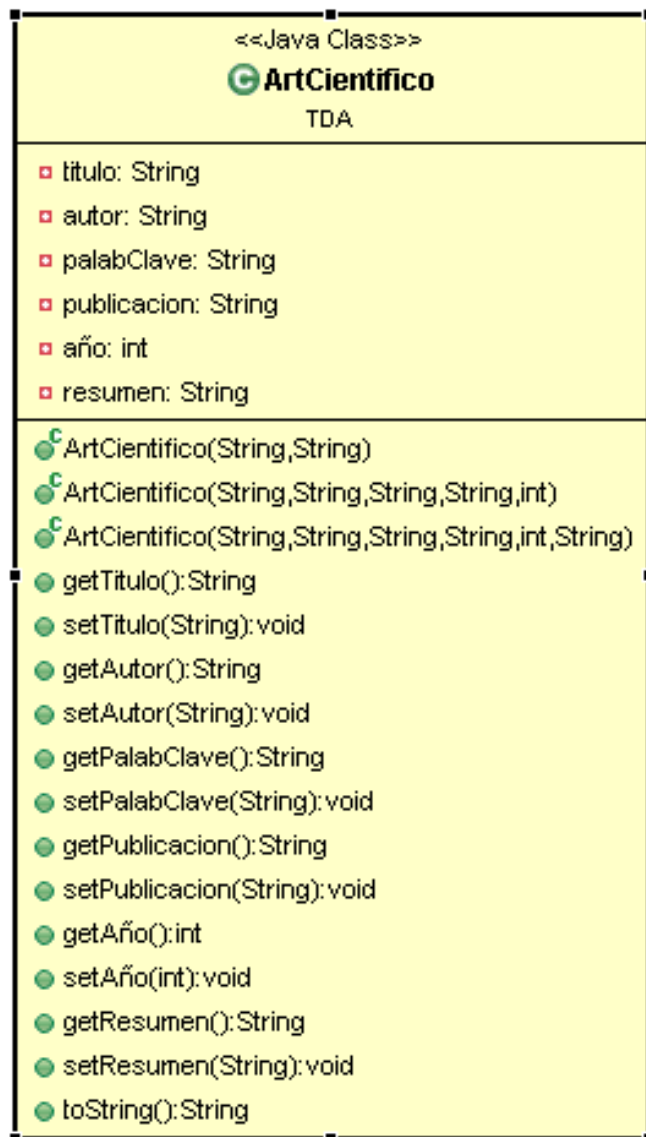
 <sup>C</sup>TestConvertor() <sup>S</sup>main(String[]): void

### Ejercicio 4:

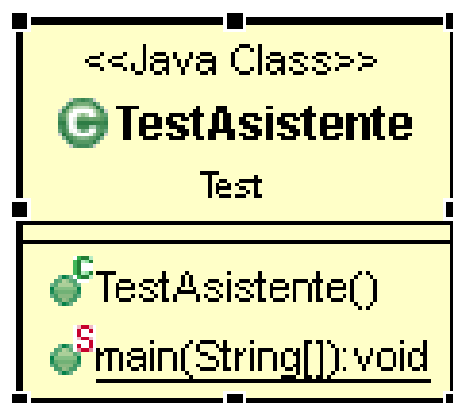
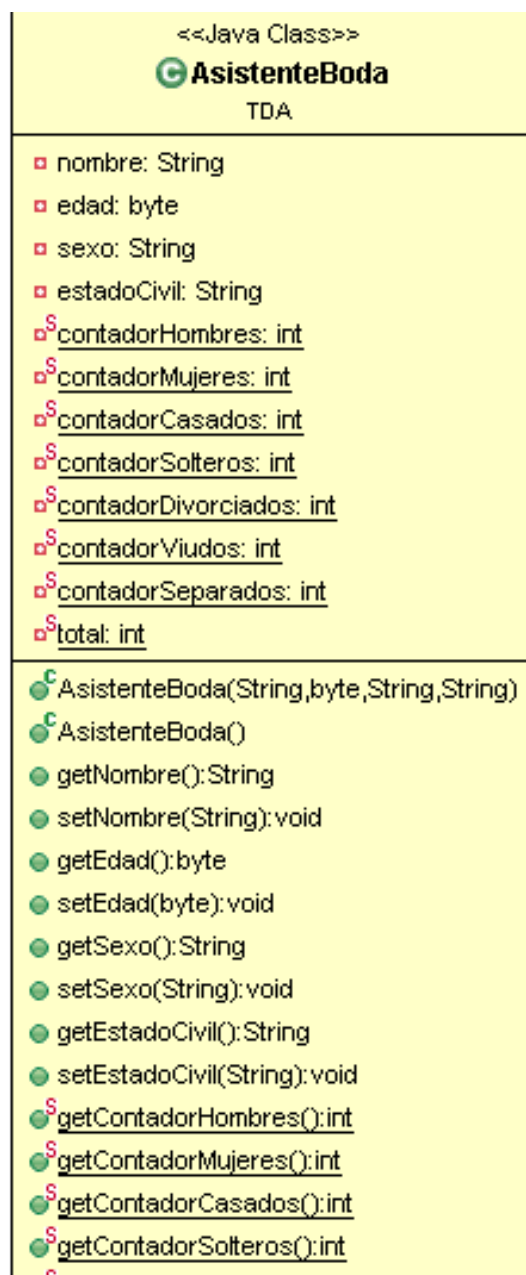




## Ejercicio 5:



## Ejercicio 6:



```
S getContadorDivorciados():int  
S getContadorViudos():int  
S getContadorSeparados():int  
S getTotal():int  
contadorSexo():void  
contadorEdoCivil():void  
imprimirDatos():String  
toString():String
```

### 1) Diseñar un método de clase y probar en una unidad ejecutable.

Se tiene un salón en forma rectangular, de 7 metros de largo por 6.5 metros de ancho. Además, se cuenta con dos alfombras una de 3.8 de largo por 4.6 de ancho y 4.5 de largo por 2.3 de ancho, también rectangulares, que se colocarán sobre el piso de dicho salón. Se desea saber que parte del piso quedará cubierta y qué parte no, para ayudar a decidir si se compra o no otras alfombras.

PD: Le recuerdo que ya tiene una clase que puede reutilizar para dar solución al ejercicio.

#### Código:

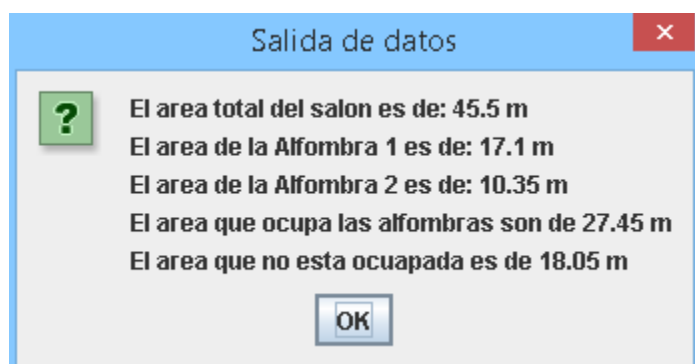
##### //Paquete TDA

```
package TDA;  
public class Alfombra {  
    private double Altura;  
    private double Base;  
    public Alfombra(double Altura, double Base) {  
        this.Altura=Altura;  
        this.Base=Base; }  
    public double getAltura() {  
        return Altura; }  
    public void setAltura(double altura) {  
        Altura = altura; }  
    public double getBase() {  
        return Base; }  
    public void setBase(double base) {  
        Base = base; }  
    public double Perimetro () {  
        return (2*(Altura+Base)); }  
    public double Area() {  
        return (Altura*Base); } }
```

## //Paquete Test

```
package Test;
import EntradaSalida.Tools;
import TDA.Alfombra;
import TDA.Salon;
public class TestSalon {
    public static void main(String[] args) {
        DatosSalon();
    }
    public static void DatosSalon() {
        Salon Sal=new Salon((int) 7, (double)6.5);
        Alfombra Alf1=new Alfombra((double)3.8,(double)4.5);
        Alfombra Alf2=new Alfombra((double)4.5,(double)2.3);
        double AreaAlf = (Alf1.Area()+Alf2.Area());
        double AreaSal=Sal.Area()-AreaAlf;
        Tools.imprimeSalida("El area total del salon es de: "+Sal.Area()
        +" m\n"+"El area de la Alfombra 1 es de: "+(float)Alf1.Area()+" m\n"+
        "El area de la Alfombra 2 es de: "+(float)Alf2.Area()+" m\n"+
        "El area que ocupa las alfombras son de "+(float)AreaAlf+" m\n"+"El area
que no esta ocuapada es de "+
        (float)AreaSal+" m");
    }
}
```

## Resultados:



En este código haremos un programa que muestre al usuario el área que cubren dos alfombras a un salón de clase para que sepan si comprar o no las alfombras, para realizar este código usaremos las fórmulas de un rectángulo para saber el área, ya que las alfombras y el salón tienen forma de rectángulo,

sumando las dos áreas de las alfombras y restándolo con el área del salón, sabremos si cubren o no el área completo del salón para comprar las alfombras.

## 2)Clase Película

Realizar un programa en Java que defina una clase Película con los siguientes atributos privados:

- Nombre: es el nombre de la película y es de tipo String.
- director: representa el nombre del director de la película y es de tipo String.
- Género: es el género de la película, con los siguientes valores: ACCIÓN, COMEDIA,

### DRAMA y SUSPENSO.

- Duración: duración de la película en minutos, con los siguientes valores :60, 115,120, 150,180,191, 210,240, y 270.
- Año: representa el año de realización de la película.
- Calificación: es la valoración de la película por parte de sus usuarios y es de tipo byte, con los siguientes valores 0.10.

✓ Se debe definir un constructor público para la clase, que recibe como parámetros los valores de todos sus atributos. También se deben definir los siguientes métodos:

✓ Métodos get y set para cada atributo y con los derechos de acceso privados para los set y públicos para los get.

✓ Un método que muestre en pantalla todos los valores de los atributos.

✓ Un método privado boolean esPelículaEpica(), el cual devuelve un valor verdadero si la duración de la película es mayor o igual a tres horas, en caso contrario devuelve falso.

✓ Un método privado String calcularValoración(), el cual según la tabla 1 devuelve una

TABLA VALORACION DE LA PELICULA

CALIFICACION	VALORACION	CALIFICACION	VALORACION
0-2	Muy mala	7-8	Buena
2-5	Mala	8-10	Excelente
5-7	Regular		

✓ El método privado boolean `esSimilar()` compara la película actual con otra película que se le pasa como parámetro. Si ambas películas son del mismo género y tienen la misma valoración, devuelve verdadero; en caso contrario, devuelve falso.

✓ Definir un método privado denominado `imprimir Cartel` que muestra en pantalla los datos de la película en el siguiente formato:

----- Título -----

\*\*\* (valoración)

Género, año

Director

La valoración se debe convertir a una cantidad determinada de asteriscos (\*), recuerde que ya tiene un metodo usado en frecuencia de vocales. En una clase que contenga una unidad ejecutable prueba lo siguiente: Diseñe un metodo de clase crear 2 instancias tipo película; determinar si son películas épicas; calcular su respectiva valoración y determinar si son similares e imprimir ambos objetos. Mostrar en una ventana emergente los resultados de la ejecución del método `main`

Objeto	Nombre	Director	Genero	Duración	año	Calificación	Épica	Similares
pe11	Gandhi	Richard Attenborough	Drama	191	1982	8		
pe12	Thor	Kenneth Branagh	Acción	115	2011	7		

**Código:**

**//Paquete TDA**

**package** TDA;

**public class** Pelicula {

**private** String NombPeli,NombDirec,GenePeli;

```
private int Duracion,Año;
private byte Calif;
public Pelicula (String NombPeli,String NombDirec,String GenePeli,int
Duracion,int Año,byte Calif) {
    this.NombPeli=NombPeli;
    this.NombDirec=NombDirec;
    this.GenePeli=GenePeli;
    this.Duracion=Duracion;
    this.Año=Año;
    this.Calif=Calif;
}
public String getNombPeli() {return NombPeli;}
private void setNombPeli(String NombPeli) {this.NombPeli = NombPeli;}
public String getNombDirec() {return NombDirec;}
private void setNombDirec(String NombDirec)
{this.NombDirec=NombDirec;}
public String getGenePeli() {return GenePeli;}
private void setGenePeli(String GenePeli) {this.GenePeli = GenePeli;}
public int getDuracion() {return Duracion;}
private void setDuracion(int Duracion) {this.Duracion = Duracion;}
public int getAño() {return Año;}
private void setAño(int Año) {this.Año = Año;}
public int getCalif() {return Calif;}
private void setCalif(byte Calif) {this.Calif = Calif;}
public String toString() {
    return "Pelicula: "+NombPeli+"\nDirector: "+NombDirec+"\nGenero:
"+GenePeli+"\nDuracion: "+Duracion+" Min\nAño de Lanzamiento:
"+Año+"\nCalificacion: "+
Calif+"/10"+" \nLa pelicula es epica: "+esPeliculaEpica()+"\nValoracion:
"+CalcularValoracion()+"\n"+ "\n"+imprimeCartel()+"\n";}
private boolean esPeliculaEpica() {
    return (Duracion<180);}
public String msjeTrueFalseEpica() {
    return (esPeliculaEpica())?"Pelicula epica":"No fue pelicula epica";}
private String CalcularValoracion() {
    String Val=" ";
    switch ((Calif<=2)?1:(Calif>2 && Calif<=5)?2:(Calif>5 && Calif<=7)?3:
(Calif>7 && Calif<=8)?4:(Calif>8 && Calif<=10)?5:6) {
        case 1: Val="Muy mala";break;
        case 2: Val="Mala";break;
        case 3: Val="Regular";break;
        case 4: Val="Buena";break;
        case 5: Val="Exelente";break;}
    return Val;}
public boolean esSimilar(Pelicula Peli2) {
    return (Peli2.getGenePeli().equals(this.getGenePeli())) &&
Peli2.CalcularValoracion().equals(CalcularValoracion());}
```

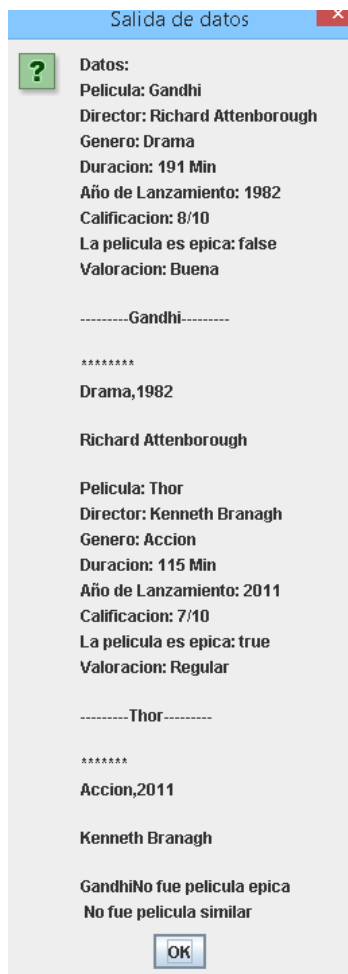
```
public String similarTrueFalse(Pelicula peli2) {  
    return esSimilar(peli2)? " Pelicula similar": " No fue pelicula similar";}  
public String imprimeCartel() {  
    return "-----"+NombPeli+"-----"+"\\n"+"\\n"+imprimeFrecuencia(Calif)  
+"\\n"+GenePeli+", "+Año+"\\n"+"\\n"+NombDirec;}  
private static String imprimeFrecuencia (int j) {  
    String cad="";  
    for (int i=1;i<=j; i++ ) {  
        cad+="*";  
    }  
    return cad;}}
```

## //Paquete Test

```
package Test;  
import EntradaSalida.Tools;  
import TDA.Pelicula;  
public class TestPelicula {  
    public static void main(String[] args) {  
        DatosPeliculas(); }  
    public static void DatosPeliculas() {  
        Pelicula Peli1=new Pelicula("Gandhi", "Richard  
Attenborough", "Drama", 191, 1982, (byte)8);  
        Pelicula Peli2=new Pelicula("Thor", "Kenneth  
Branagh", "Accion", 115, 2011, (byte)7);  
        String msje= Peli1.toString();  
        msje+="\\n"+Peli2.toString();  
        msje+="\\n"+Peli1.getNombPeli()+Peli1.msjeTrueFalseEpica();  
        msje+="\\n"+Peli1.similarTrueFalse(Peli2);  
        Tools.imprimeSalida("Datos: \\n"+msje);}}
```

## Resultados:





En este código realizaremos un programa que muestre el nombre del director, nombre de la película, genero, duración, año de lanzamiento, calificación, si fue épica y la valoración, para ello realizamos diferentes métodos para saber su calificación, su valoración y si fue épica o no y por último un método para realiza un cartel solicitado donde se mostrara los datos de la película, para mostrar todos los datos utilizaremos un toString() para mostrar en un sola pantalla emergente.

### 3)clase ConversorMetros

Realizar un programa en Java que permita realizar las siguientes conversiones de unidades de longitud, con el siguiente atributo privado y constantes:

- Atributo llamado metros (de tipo double) para definir la cantidad de metros a convertir a diferentes unidades de longitud

Constantes:

- METROS\_CM=100
- METROS\_MILIM = 1000
- METROS\_PULGADAS = 39.37
- METROS\_PIES = 3.28
- METROS\_YARDAS = 1.09361

La clase tiene un constructor para inicializar la cantidad de metros a convertir y un conjunto de métodos private para realizar diferentes conversiones, cada uno consu valor de retorno correspondiente.

- Metros a centímetros.
- Metros a milímetros.
- Metros a pulgadas.
- Metros a pies.
- Metros a yardas

En una clase que contenga una unidad ejecutable presentar un menú que permita crear instancias y seleccionar el tipo de conversión a realizar.

## Código:

### //Paquete TDA

```
package TDA;
import EntradaSalida.Tools;
public class ConversorMetro {
    double metro;
    final int METROS_CM=100;
    final double METROS_MILIM=1000;
    final double METROS_PULGADAS=39.37;
    final double METROS_PIES=3.28;
    final double METROS_YARDAS=1.09361;
    public ConversorMetro(double metro) {
        this.metro = metro;}
    public void setMetro(double metro) {
        this.metro = metro;}
    private double convMetrosCemtímetros () {
        return (METROS_CM*metro);}
    private double convMetrosMilímetros () {
        return (METROS_MILIM*metro);}
    private double convMetrosPulgadas () {
        return (METROS_PULGADAS*metro);}
    private double convMetrosPies () {
        return (METROS_PIES*metro);}
    private double convMetrosYardas () {
        return (METROS_YARDAS*metro);}
    public void conversionMetros() {
        String opc=" ";
        do {
```

```
opc=Tools.boton("METROS_CM,METROS_MILIMETROS,METROS_PULGADAS,METROS_PIE  
S,METROS_YARDAS,SALIR");  
    switch (opc) {  
        case "METROS_CM": Tools.imprimeSalida("Metros a centimetros:  
"+convMetrosCemtímetros()); break;  
        case "METROS_MILIMETROS": Tools.imprimeSalida("Metros a  
milimetros: "+convMetrosMilimetros()); break;  
        case "METROS_PULGADAS": Tools.imprimeSalida("Metros a pulgadas:  
"+convMetrosPulgadas()); break;  
        case "METROS_PIES": Tools.imprimeSalida("Metros a pies:  
"+convMetrosPies()); break;  
        case "METROS_YARDAS": Tools.imprimeSalida("Metros a yardas:  
"+convMetrosYardas()); break;}  
    }while (!opc.equals("SALIR"));
```

## //Paquete Test

```
package Test;  
import EntradaSalida.Tools;  
import TDA.ConversorMetro;  
public class TestConvertor {  
    public static void main(String[] args) {  
        ConversorMetro obj=new ConversorMetro(Tools.leerByte("Cantidad de metros: "));  
        obj.conversionMetros();}
```

## Resultados:

Dato de entrada

Cantidad de metros:

10

OK Cancel

Unidades

? Selecciona una Unidad

METROS\_CM METROS\_MILIMETROS METROS\_PULGADAS METROS\_PIES METROS\_YARDAS SALIR

Salida de datos

? Metros a centímetros: 1000.0

OK

Salida de datos

? Metros a milímetros: 10000.0

OK

Salida de datos

? Metros a pulgadas: 393.7

OK

Salida de datos

? Metros a pies: 32.8

OK

Salida de datos

? Metros a yardas: 10.9361

OK

En este código realizaremos un programa que convierta los metros a diferentes unidades de medida, como centímetros, milímetros, pulgadas, pies y yardas, para realizar este código usaremos “final” en las variables ya que son medidas únicas y que nunca se cambiarán durante todo el código, y ocuparemos estas en unos métodos que realizaremos para hacer la conversión, que es una simple multiplicación de metros ingresado con el valor de cada unidad de medida y así obtendremos el resultado para mostrar al usuario su conversión

#### 4)clase Pedido

Realizar un programa en Java que permita calcular el pedido que realiza un cliente en un restaurante.

Los pedidos de un restaurante están conformados por las siguientes partes:

- Un primer plato (“Crema de espinacas”, “Ensalada de verduras”, “Crema de brócoli”, “Caldo tlalpeño”, “Sopa mixteca”).
- Un segundo plato (“Filete de Pescado”, “Milanesa de Pollo”, “Bistec a la mexicana”, “Pollo en escabeche”, “Carne asada”, “Lomo relleno”, “Pollo a la plancha”).
- Una bebida (“Coca cola”, “Pepsi”, “Naranjada”, “Limonada”, “Agua de sabor”).
- Un postre (“Pastel helado”, “Helado”, “Fresas con crema”, “Plátanos fritos”, “Flan casero”, “Gelatina”).

Cada uno de dichas partes tiene un nombre y un precio. Se requiere definir métodos sobrecargados para calcular el valor del pedido dependiendo si el cliente solicita:

- ✓ Un primer plato y una bebida.
- ✓ Un primer plato, un segundo plato y una bebida.
- ✓ Un primer plato, un segundo plato, una bebida y un postre.

En una clase que contenga una unidad ejecutable presentar los costos del pedido solicitado (imprimir en una ventana emergente).

**código:**

**//Paquete TDA**

```
package TDA;
import EntradaSalida.Tools;
public class RestPedido {
    private String PrimerPlat;
    private String SegundoPlat;
    private String Bebida;
    private String Postre;
    public RestPedido(String primerPlat, String bebida) {
        this.PrimerPlat = primerPlat;
        this.Bebida = bebida;}
    public RestPedido(String primerPlat, String segundoPlat, String bebida) {
        this.PrimerPlat = primerPlat;
        this.SegundoPlat = segundoPlat;
        this.Bebida = bebida;}
    public RestPedido(String primerPlat, String segundoPlat, String bebida,
String postre) {
        this.PrimerPlat = primerPlat;
        this.SegundoPlat = segundoPlat;
        this.Bebida = bebida;
        this.Postre = postre;}
```

```
public String getPrimerPlat() {
    return PrimerPlat;}
public void setPrimerPlat(String primerPlat) {
    PrimerPlat = primerPlat;}
public String getSegundoPlat() {
    return SegundoPlat;}
public void setSegundoPlat(String segundoPlat) {
    SegundoPlat = segundoPlat;}
public String getBebida() {
    return Bebida;}
public void setBebida(String bebida) {
    Bebida = bebida;}
public String getPostre() {
    return Postre;}
public void setPostre(String postre) {
    Postre = postre;}
private double CostoPrimerPlato(String primerPlat) {
    double total = 0;
    switch (primerPlat) {
        case "Crema de espinacas": total = 75.0; break;
        case "Ensalada de verduras": total = 95.0; break;
        case "Crema de brócoli": total = 70.0; break;
        case "Caldo tlalpeño": total = 123.90; break;
        case "Sopa mixteca": total = 99.90; break; }
    return total;}
private double CostoSegundoPlato(String segundoPlat) {
    double total=0;
    switch (segundoPlat) {
        case "Filete de Pescado": total = 80.0; break;
        case "Milanesa de Pollo": total = 70.0; break;
        case "Bistec a la mexicana": total = 85.0; break;
        case "Pollo en escabeche": total = 75.0; break;
        case "Carne asada": total = 90.0; break;
        case "Lomo relleno": total = 95.0; break;
        case "Pollo a la plancha": total = 80.0; break; }
    return total;}
private double CostoBebida(String bebida) {
    double total=0;
    switch (bebida) {
        case "Coca Cola":
        case "Pepsi": total += 20.0; break;
        case "Naranja": case "Limonada":
        case "Agua de sabor": total += 15.0; break;}
    return total;}
private double CostoPostre(String postre) {
    double total=0;
```

```
switch (postre) {
case "Pastel helado": total += 40.0; break;
case "Helado": total += 30.0; break;
case "Fresas con crema": total += 35.0; break;
case "Plátanos fritos": total += 40.0; break;
case "Flan casero": total += 25.0; break;
case "Gelatina": total += 20.0; break; default: break; }
return total; }

public void calcularPedidos (String primerPlat, String bebida) {
double total =CostoPrimerPlato(primerPlat)+CostoBebida(bebida);
Tools.imprimeSalida("Total a pagar"+total); }

public void CalcularPedidos (String primerPlat, String segundoPlat,
String bebida) {
double total =CostoPrimerPlato(primerPlat)
+CostoSegundoPlato(segundoPlat)+CostoBebida(bebida);
Tools.imprimeSalida("Total a pagar"+total); }

public void calcularPedidos(String primerPlat, String segundoPlat, String
bebida, String postre) {
double total =CostoPrimerPlato(primerPlat)
+CostoSegundoPlato(segundoPlat)+CostoBebida(bebida)+CostoPostre(postre);
Tools.imprimeSalida("Total a pagar"+total);}}
```

### //Paquete Test

```
package Test;
import TDA.RestPedido;
public class TestPedido {
public static void main (String[] args) {
RestPedido obj = new RestPedido(null, null, null, null);
obj.calcularPedidos("Sopa mixteca", "Pepsi");
obj.CalcularPedidos("Ensalada de verduras", "Filete de Pescado",
"Naranja");
obj.calcularPedidos("Ensalada de verduras", "Lomo relleno", "Agua
de sabor", "Helado");}
```

### Primer pedido:

Three Java Swing windows are shown side-by-side:

- Menu de comida (left):** Title bar "Menu de comida". Contains a green question mark icon, a label "Seleccione su primer plato:", a dropdown menu showing "Crema de espinacas", and "OK" and "Cancel" buttons.
- Menu de comida (middle):** Title bar "Menu de comida". Contains a green question mark icon, a label "Seleccione su bebida:", a dropdown menu showing "Pepsi", and "OK" and "Cancel" buttons.
- Salida de datos (right):** Title bar "Salida de datos". Contains a green question mark icon, a label "Total a pagar95.0", and an "OK" button.

### Segundo pedido:

Menu de comida

Seleccione su primer plato:  
Caldo tlalpeño

OK Cancel

Menu de comida

Seleccione su segundo plato:  
Lomo relleno

OK Cancel

Menu de comida

Seleccione su bebida:  
Agua de sabor

OK Cancel

Salida de datos

Total a pagar 233.9

OK

### Tercer Pedido:

Menu de comida

Seleccione su primer plato:  
Crema de espinacas

OK Cancel

Menu de comida

Seleccione su segundo plato:  
Pollo a la plancha

OK Cancel

Menu de comida

Seleccione su bebida:  
Limonada

OK Cancel

Menu de comida

Seleccione su postre:  
Pastel helado

OK Cancel

Salida de datos

Total a pagar 210.0

OK

En este código haremos un programa que muestre el precio total del pedido de un cliente, pero existen 3 tipos de pedidos donde haremos una sobrecarga de métodos para tener el mismo método, pero con diferentes parámetros ya que las indicaciones dicen que existen 3 tipos de pedidos y dependiendo de que escoja el cliente se deberá mostrar los precios acordes al pedido, para eso crearemos un método llamado RestPedido con diferentes parámetros para crear una sobrecarga y 4 métodos para obtener los precios de plato uno, plato dos, bebida y postre y con un toString hacer la suma de todo y mostrarlo al usuario.

### 5) clase Artículo Científico

Realizar un programa en Java que permita modelar un artículo científico. Los artículos científicos contienen los siguientes metadatos: nombre del artículo, autor, palabras claves, nombre de la publicación, año y resumen. Se deben definir tres constructores sobrecargados:

- ✓ El primero inicializa un artículo científico con solo su título y autor.
- ✓ El segundo constructor, un artículo científico con su nombre, autor, palabras claves, nombre de la publicación y año. Debe invocar al primer constructor.



✓ El tercer constructor, un artículo científico con su nombre, autor, palabras claves, nombre de la publicación, año y resumen. Debe invocar al segundo constructor.

✓ Se requiere un método que imprima los atributos de un artículo en pantalla.

Realizar una unidad ejecutable que utilice el tercer constructor para instanciar un artículo científico e imprima los valores de sus atributos en pantalla. Recuerde que: a veces es necesario inicializar un objeto de diferentes formas. Para ello, se realiza la sobrecarga de constructores. Los constructores sobrecargados deben tener diferente número o tipo de parámetros. Cada constructor realiza una tarea diferente. Se puede utilizar la palabra reservada `this` para llamar a un constructor desde otro. La llamada `this` debe ser la primera línea de dicho constructor. Ejemplo:

```
public ArtículoCientífico(String título, String autor, String[] palabrasClaves, String
publicación, int año) {
    this(título, autor); // Invoca al constructor sobrecargado
    this.palabrasClaves = palabrasClaves;
    this.publicación = publicación;
    this.año = año;
}
```

**código:**

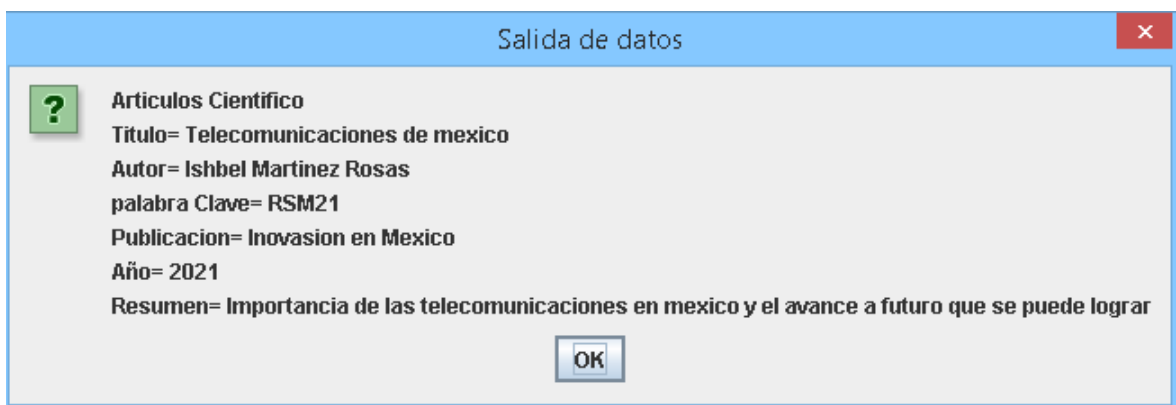
### //Paquete TDA

```
package TDA;
public class ArtCientifico {
    private String titulo;
    private String autor;
    private String palabClave;
    private String publicacion;
    private int año;
    private String resumen;
    public ArtCientifico(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
    }
    public ArtCientifico(String titulo, String autor,
String palabClave, String publicacion, int año) {
        this(titulo, autor);
        this.palabClave=palabClave;
        this.publicacion=publicacion;
        this.año=año;}
    public ArtCientifico(String titulo, String autor,
String palabClave, String publicacion, int año, String
resumen) {
```

```
        this(titulo,autor,palabClave,publicacion,año);
        this.resumen=resumen;}
public String getTitulo() {
    return titulo;}
public void setTitulo(String titulo) {
    this.titulo = titulo;}
public String getAutor() {
    return autor;}
public void setAutor(String autor) {
    this.autor = autor;}
public String getPalabClave() {
    return palabClave;}
public void setPalabClave(String palabClave) {
    this.palabClave = palabClave;}
public String getPublicacion() {
    return publicacion;}
public void setPublicacion(String publicacion) {
    this.publicacion = publicacion;}
public int getAño() {
    return año;}
public void setAño(int año) {
    this.año = año;}
public String getResumen() {
    return resumen;}
public void setResumen(String resumen) {
    this.resumen = resumen;}
public String toString() {
    return "Articulos Cientifico"+"\\nTitulo= " +
    titulo + "\\nAutor= " + autor + "\\npalabra Clave= " +
    palabClave + "\\nPublicacion= "
        +publicacion+"\\nAño= " +año+"\\nResumen=
    " + resumen;}}
//Paquete Test
package Test;
import EntradaSalida.Tools;
import TDA.ArtCientifico;
public class TestArtCientifico {
    public static void main(String[] args) {
        ArtCientifico obj = new ArtCientifico (""," ",
        " ", " ",0, " ");
        obj.setTitulo(Tools.leerString("Ingrese el nombre
        del articulo"));
        obj.setAutor(Tools.leerString("Ingrese el nombre
        del autor"));
```

```
obj.setPalabClave(Tools.leerString("Ingrese la  
palabra clave"));  
obj.setPublicacion(Tools.leerString("Ingrese nombre  
de la publicacion"));  
obj.setAño(Tools.leerInt("Ingrese el año"));  
obj.setResumen(Tools.leerString("Ingrese el resumen  
del articulo"));  
Tools.imprimeSalida(obj.toString());
```

## Resultados:



En este código realizamos un programa que nos muestre el título del artículo, nombre del autor, palabra clave, publicación, año y un pequeño resumen, pero para realizar esto la indicación nos decía que usáramos constructores sobrecargados, lo cual cada constructor tenía diferentes parámetros y para unir los parámetros usamos un `this` (parámetros del anterior constructor sobrecargado) e invocamos el último constructor sobrecargado hecho.

## 6) Clase AsistenteBoda (Control de registro de los asistentes)

Implemente una aplicación que permita controlar los asistentes que son mayores de edad y que asistirán a la boda; para ello deberá de ingresar el nombre, la edad, sexo (femenino, masculino), estado civil (solter@, casad@, viud@, separad@ y divorciad@). Deberá de capturar los datos y mostrar las estadísticas usando variables estáticas (solo los contadores) e imprimir en una sola ventana emergente las frecuencias y el valor obtenido.

- ✓ Total de hombres
- ✓ Total de mujeres
- ✓ Total de casados

- ✓ Total de solteros
  - ✓ Total de viudos
  - ✓ Total de separados
  - ✓ Total de divorciados.
  - ✓ Imprimir los datos de la persona que tiene mayor edad y los datos de la persona que tiene menor edad.
  - ✓ Imprimir todos los datos capturados usando una ventana emergente.
- Probar en una unidad ejecutable que permita capturar los datos de tipo AsistenteBoda, y mostrar las frecuencias obtenidas, los datos de la persona que tiene mayor edad y los datos de la persona que tiene menor edad, y todos los datos capturados.

### código:

#### //Paquete TDA

```
package TDA;
public class AsistenteBoda {
private String nombre;
private byte edad;
private String sexo;
private String estadoCivil;
private static int contadorHombres=0;
private static int contadorMujeres=0;
private static int contadorCasados=0;
private static int contadorSolteros=0;
private static int contadorDivorciados=0;
private static int contadorViudos=0;
private static int contadorSeparados=0;
private static int total=0;
public AsistenteBoda(String nombre, byte edad, String sexo,String
estadoCivil){
    this.nombre = nombre;
    this.edad = edad;
    this.sexo = sexo;
    this.estadoCivil=estadoCivil;
    total++; }
public AsistenteBoda() { }
public String getNombre() {
    return nombre; }
public void setNombre(String nombre) {
    this.nombre = nombre; }
public byte getEdad() {
    return edad; }
```

```
public void setEdad(byte edad) {
    this.edad = edad; }
public String getSexo() {
    return sexo;}
public void setSexo(String sexo) {
    this.sexo = sexo; }
public String getEstadoCivil() {
    return estadoCivil; }
public void setEstadoCivil(String estadoCivil) {
    this.estadoCivil = estadoCivil; }
public static int getContadorHombres() {
    return contadorHombres; }
public static int getContadorMujeres() {
    return contadorMujeres; }
public static int getContadorCasados() {
    return contadorCasados; }
public static int getContadorSolteros() {
    return contadorSolteros;}
public static int getContadorDivorciados() {
    return contadorDivorciados;}
public static int getContadorViudos() {
    return contadorViudos;}
public static int getContadorSeparados() {
    return contadorSeparados;}
public static int getTotal() {
    return total; }
public void contadorSexo(){
    if(sexo.equalsIgnoreCase("Hombre")){
        contadorHombres++; }
    else if (sexo.equalsIgnoreCase("Mujer")){
        contadorMujeres++; } }
public void contadorEdoCivil(){
    if(estadoCivil.equalsIgnoreCase("Solter@") ||
estadoCivil.equalsIgnoreCase("soltera")){
        contadorSolteros++; }
    else if(estadoCivil.equalsIgnoreCase("Casad@") ||
estadoCivil.equalsIgnoreCase("casada")){
        contadorCasados++;}
    else if(estadoCivil.equalsIgnoreCase("Divorciad@") ||
estadoCivil.equalsIgnoreCase("divorciada")){
        contadorDivorciados++;}
    else if(estadoCivil.equalsIgnoreCase("Viud@") ||
estadoCivil.equalsIgnoreCase("viuda")){
        contadorViudos++;}
    else if(estadoCivil.equalsIgnoreCase("Separad@") ||
estadoCivil.equalsIgnoreCase("viuda")){
```

```
        contadorSeparados++;} }  
    public String imprimirDatos(){  
    return ("Total de hombres: "+contadorHombres+"\nTotal de mujeres"  
+contadorMujeres+"\nTotal de casados: "+contadorCasados+"\nTotal de  
solteros: "+contadorSolteros+"\nTotal de "  
+"viudos: "+contadorViudos+"\nTotal de separados:  
"+contadorSeparados+"\nTotal de divorciados"+contadorDivorciados); }  
    @Override  
    public String toString() {  
        return " [nombre=" + nombre + ", edad=" + edad + ", sexo=" + sexo  
+ ", estadoCivil=" + estadoCivil "]" ;}}}
```

## //Paquete Test

```
package TDA;  
public class AsistenteBoda {  
    private String nombre;  
    private byte edad;  
    private String sexo;  
    private String estadoCivil;  
    private static int contadorHombres=0;  
    private static int contadorMujeres=0;  
    private static int contadorCasados=0;  
    private static int contadorSolteros=0;  
    private static int contadorDivorciados=0;  
    private static int contadorViudos=0;  
    private static int contadorSeparados=0;  
    private static int total=0;  
    public AsistenteBoda(String nombre, byte edad, String sexo,String  
estadoCivil) {  
        this.nombre = nombre;  
        this.edad = edad;  
        this.sexo = sexo;  
        this.estadoCivil=estadoCivil;  
        total++; }  
    public AsistenteBoda() { }  
    public String getNombre() {  
        return nombre; }  
    public void setNombre(String nombre) {  
        this.nombre = nombre; }  
    public byte getEdad() {  
        return edad; }  
    public void setEdad(byte edad) {  
        this.edad = edad; }  
    public String getSexo() {  
        return sexo;}  
    public void setSexo(String sexo) {  
        this.sexo = sexo; }  
    public String getEstadoCivil() {  
        return estadoCivil; }  
    public void setEstadoCivil(String estadoCivil) {  
        this.estadoCivil = estadoCivil; }
```

```
public static int getContadorHombres() {
    return contadorHombres; }
public static int getContadorMujeres() {
    return contadorMujeres; }
public static int getContadorCasados() {
    return contadorCasados; }
public static int getContadorSolteros() {
    return contadorSolteros; }
public static int getContadorDivorciados() {
    return contadorDivorciados; }
public static int getContadorViudos() {
    return contadorViudos; }
public static int getContadorSeparados() {
    return contadorSeparados; }
public static int getTotal() {
    return total; }
public void contadorSexo() {
    if(sexo.equalsIgnoreCase("Hombre")){
        contadorHombres++; }
    else if (sexo.equalsIgnoreCase("Mujer")){
        contadorMujeres++; } }
public void contadorEdoCivil() {
    if(estadoCivil.equalsIgnoreCase("Solter@") ||
estadoCivil.equalsIgnoreCase("soltera")){
        contadorSolteros++; }
    else if(estadoCivil.equalsIgnoreCase("Casad@") ||
estadoCivil.equalsIgnoreCase("casada")){
        contadorCasados++; }
    else if(estadoCivil.equalsIgnoreCase("Divorciad@") ||
estadoCivil.equalsIgnoreCase("divorciada")){
        contadorDivorciados++; }
    else if(estadoCivil.equalsIgnoreCase("Viud@") ||
estadoCivil.equalsIgnoreCase("viuda")){
        contadorViudos++; }
    else if(estadoCivil.equalsIgnoreCase("Separad@") ||
estadoCivil.equalsIgnoreCase("viuda")){
        contadorSeparados++; } }
public String imprimirDatos() {
return ("Total de hombres: "+contadorHombres+"\nTotal de mujeres"
+contadorMujeres+"\nTotal de casados: "+contadorCasados+"\nTotal de
solteros: "+contadorSolteros+"\nTotal de "
+"viudos: "+contadorViudos+"\nTotal de separados:
"+contadorSeparados+"\nTotal de divorciados"+contadorDivorciados); }
@Override
public String toString() {
    return " [nombre=" + nombre + ", edad=" + edad + ", sexo=" +
sexo + ", estadoCivil=" + estadoCivil
+ "]" ; } }
```

## Conclusiones:

En conclusión aprendimos cosas nuevas que nos ayudan y facilitan el uso y funcionamiento de un código, aprendimos el objetivo de los métodos y constructores sobrecargados, son parecidos pero tienen diferente funcionalidad. Los métodos sobrecargados tienen el mismo nombre, pero diferentes parámetros, esto nos ayuda a evitar crear diferentes métodos en cambio los constructores sobrecargados, podemos crear varios y con un "this" podemos obtener los parámetros del anterior constructor sobrecargado, logramos aprender cuando usar atributos constantes usando un "final" antes de declarar el tipo de atributo, siempre se usará cuando el valor del atributo sea fijo y nunca cambie, como los nombres o el valor de cada unidad de medida, también supimos cuando ocupar variables estáticas en este proyecto lo ocupamos para nuestros contadores de personas y un dato particular de estas variables es que se pueden modificar en la misma o diferentes clases.



## Bibliografía:

- *Métodos en Java.* (s. f.).  
<http://puntocomnoesunlenguaje.blogspot.com/2012/04/metodos.html>
- Garro, A. (s. f.). *Tipos de métodos | Java.*  
<https://www.arkaitzgarro.com/java/capitulo-14.html#:~:text=Un%20m%C3%A9todo%20es%20una%20abstracci%C3%B3n,y%20los%20m%C3%A9todos%20de%20clase.>
- Pérez, D. A. D. (s. f.). *Métodos con valor de retorno.*  
<http://inprojava.blogspot.com/2009/10/metodos-con-valor-de-retorno.html>
- Rodríguez, A. (s. f.). *Concepto de métodos de clase o static y métodos de instancia. Diferencias. Método main de Java. (CU00683B).*  
[https://www.aprenderaprogramar.com/index.php?option=com\\_content&view=article&id=650:concepto-de-metodos-de-clase-o-static-y-metodos-de-instancia-diferencias-metodo-main-de-java-cu00683b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188#:~:text=Un%20m%C3%A9todo%20de%20instancia%20es,invocado%20sin%20existir%20una%20instancia.](https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=650:concepto-de-metodos-de-clase-o-static-y-metodos-de-instancia-diferencias-metodo-main-de-java-cu00683b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188#:~:text=Un%20m%C3%A9todo%20de%20instancia%20es,invocado%20sin%20existir%20una%20instancia.)